Løsningsforslag Eksamen høst 2018 Operativsystemer

Oppgave 1 - Linux kommandolinje

a) `mv /tmp/fil.txt .`

b) Kun brukeren `haugerud`

c) `tail -n 5 index.html | grep name  | cut -d\" -f 4`

d) `sort -n -k 3 biler`

Oppgave 2 - Bash-scripting

a)

```
#! /bin/bash
if [ ! "$1" ]
then
    nr=3 # to mellomrom etter cpu
else
    nr=$(($1 + 2))
fi

for i in $(seq 1 30)
do
   idle1=$(grep cpu /proc/stat | head -n 1 | cut -d' '  -f $nr)
   sleep 1
   idle2=$(grep cpu /proc/stat | head -n 1 | cut -d' '  -f $nr)
   ((idle = $idle2 - $idle1))
   echo $idel
done
```

Det er totalt 100 jiffies pr sekund og 8 CPUer. Det betyr at fjerde kolonne, som er idle, får så godt som alle jiffies som igjen betyr at det er tilnærmet ingen CPU-bruk på serveren.

Oppgave 3 - Prosesser

a) Et multitasking operativsystem får det til å se ut som om mange prosesser kan kjøre samtidig ved å dele opp tiden i små biter og la hver prosess som kjører få en bit CPU-tid (størrelsesorden et hundredels sekund) av gangen i en Round Robin kø. En hardware timer sender jevnlig et interrupt-signal som gjør at OS kan ta over kontrollen over CPUen. Dermed unngås det at vanlige brukerprosesse tar over styringen selvom de kjører instruksjoner direkte på CPUen.

b) Linux OS lar de fem prosessene bytte på å kjøre på de fire CPUene slik at de i gjennomsnitt får kjøre like mye. Byttene skjer ganske ofte, slik at fordelingen blir rettferdig også over korte tidsrom som 5-10 sekunder. Kommandoen top vil vise omtrent 80% CPU-tid for hver av prosessene (men dette vil variere litt og ligge mellom 70 og 90%).

c) 5 minutter

d) 31

e) 4 byte = 32 bit, $2^{31} - 1$ Ca 2 milliarder. ($2^{10} = 1024$ og $2^{30} \sim 10^9$, helt presist er $2^{31} - 1 = 2147483647$)

f) Programmet regner ut summen av alle tallene fra 1 til 40000.

```
$ gcc sum.c
$ ./a.out
```

g) Sluttsvaret som regnes ut øker når variabelen N øker. For N = 50000 er svaret ca 1.2 milliarder og det øker til 1.8 milliarder for N = 60000. Dermed blir svaret en god del større enn 2 milliarder for N = 70000. Dermed er det ikke plass til å lagre sluttresultatet med 4 byte, beregningen feiler, gir bit-owerflow og et tilfeldig resultat. I dette tilfellet er svaret opplagt galt siden det er negativt.

h) Et enkelt kompilert C-program(som ikke benytter tråder) består av en serie instruksjoner som må utføres en for en. Et OS vet ikke hva instruksjonene utfører og vil ikke kunne fordele kode i blinde på de fire CPUene for å utnytte den ekstra prosessorkraften. Dermed må instruksjonene i programmet lastes inn på en enkelt CPU og kjøres der(hele prosessen kan flyttes til en annen CPU under kjøring, men kun en CPU av gangen kan kjøre koden).

i) For å kunne utnytte de fire CPUene må programmet endres slike at det bruker tråder og for å få til det må koden paralleliseres, slik at utregningene kan utføres samtidig. Oppgaven kan deles i fire like store deler der hver tråd regner ut sin del. Den første tråden kan regne ut summen fra 1 til 10000, den andre fra 10001 til 20000 og så videre. Til slutt legges svaret fra de fire trådene sammen. For et C-program kunne parallelliseringen gjøres med pthreads.

j) I dette tilfellet er det neste leddet rekken summen av de to foregående og regneoperasjonene må gjøres sekvensielt basert på de foregående resultatene. Dermed vil det ikke gi noen gevinst om man bruker flere uavhengige tråder, disse trådene måtte isåfall vente på hverandres resultater og beregningen ville tatt enda lenger tid enn for en enkelt tråd (matematisk finnes det formler for summen av N tall (N(N+1)/2) og for det N'te tallet i Fibonacci-rekken, men da vil man heller ikke utnytte flere CPUer).

Oppgave 4 - Internminne

a)

	10^3	10^6	2^10	2^20
1000	◉	○	○	○
1 000 000	○	◉	○	○
1024	○	○	◉	○
1 048 576	○	○	○	◉

b)

c) Registere er en del av CPUen og det kan ta ca ett nanosekund eller mindre å flytte data mellom dem. Fra cache (som er SRAM som registerne) kan det ta 1-5 nanosekunder (avhengig av om det er L1, L2 eller L3). Fra RAM kan det ta omtrent 10 nanosekunder eller mer. Diskaksess er svært mye langsommere og det kan ta ett millisekund eller mer (noe raskere med NVMe SSD).

d) MMU (Memory Management Unit) oversetter de logiske/virtuelle adressene som CPUen bruker til fysiske RAM-adresser før de sendes ut på databussen for å lese fra eller skrive til RAM. MMU må implementeres i hardware fordi oversettelsen må foregå ekstremt raskt. Ellers ville overhead bli alt for stort hvis CPUen selv skulle bruke klokkesykler til å utføre de nødvendige operasjonene. Da ville i praksis ta altfor lang tid siden det vanligvis er mange minneoppslag i vanlig kode.

e) Figuren viser prinsippet for hvordan en virtuell adresse blir oversatt til en fysisk adresse av MMU. En 16-bits virtuell adresse kommer inn og de fire første bit'ene tolkes som indeks i page-tabellen(eksempelet bruker en 4-KB page-størrelse). De fire første bit'en = 2 og peker derfor på page-entry nr 2. Denne inneholder 110 = 6 og den fysiske siden ligger derfor i frame nr 6. 0010 i den innkommende adressen blir erstattet med 110 og adressen 8196 endres da til den utgående fysiske adressen 24580. (Offsett er 100 = 4 og 8196 = 2*4096 + 4 = 8196. Utgående adresse blir da 6*4096 + 4 = 24580) 4096 er første byte i page 1 og adressen 6666 ligger i denne siden. Men page nr 1 peker på frame nr 1, så den fysiske adressen blir den samme, altså 6666.

f) Virtuell page nr 7 er tom og har ingen referanse til en fysisk frame. Det betyr at denne fysiske siden ikke finnes i RAM men ligger på disk. Det ville derfor inntreffe en page fault og siden måtte hentes inn fra disk før forespørselen kunne oppfylles.

Oppgave 5 - Powershell

a) Lister alle prosesser som kjører nå og som har blitt startet de siste 50 minuttene.

b)

```
$days = @()
for($day = 0; $day -lt 7; $day++){
    $days += 0;
    }

$files = ls
```

```
Foreach ($fil in ls -r C:\Users 2> $null){
    $day = [int]$fil.CreationTime.DayOfWeek
    $days[$day]++
}

for($day = 0; $day -lt 7; $day++){
    $antall = $days[$day];
    echo "Dag ${day}: $antall"
    }
```