Løsningsforslag Eksamen vår 2018 Operativsystemer

Oppgave 1 - Linux kommandolinje (%)

a) `move: command not found`

b) `ssh os3.vlab.cs.hioa.no hostname`

c) `Output fra cat omdirigeres til /dev/null og forsvinner`

d) `sort -k 2 biler`

e) `grep name index.html | head -n 5 | cut -d\" -f 2`

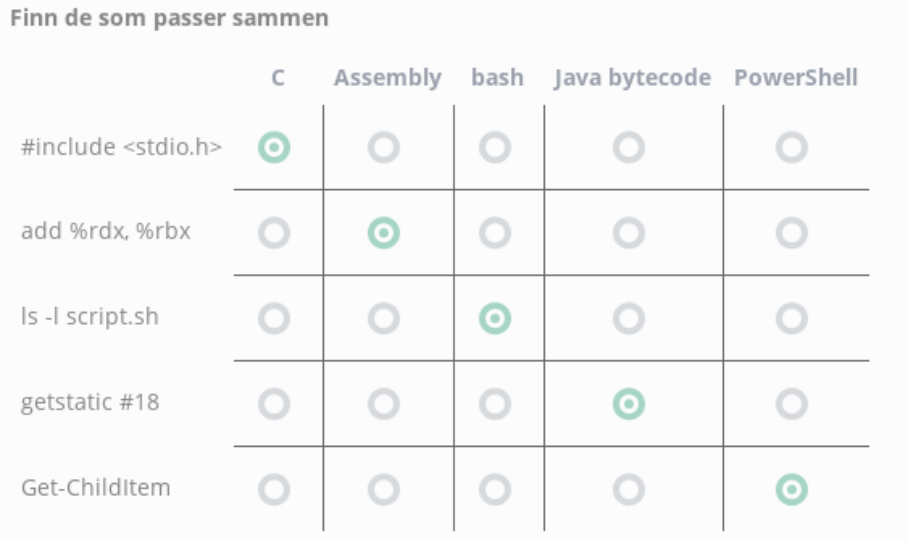
Oppgave 2 - Bash-scripting (%)

a)

```
#! /bin/bash

echo -n "dsh$ "
while read kommando
do
   if [ "$kommando" = "exit" ]
   then
       echo "Avslutter dsh"
       exit
   fi

   echo "#### $(hostname) ####"
   eval $kommando # bare $kommando er ok men virker
                  # egentlig ikke alltid f.eks echo $HOME
                  # (og eval ikke nevnt i pensum)

   for host in $(cat ~/hosts)
   do
      echo "#### $host ####"
      ssh $host $kommando
   done
   echo -n "dsh$ "
done
```

Oppgave 3 - C og Assembly (%)

a)



b) `Maskinkode`

c) `Assembly kode`

d)

```
$ gcc svar.c
$ ./a.out
Svar = 41
Kaller enlinje()...
Svar = 42
```

e)

```
$ gcc -c svar.c -o svar # Kompilerer main og lager maskinkode i fil "svar"
$ gcc -c en.c -o en     # Kompilerer funksjonen en og lager maskinkode i fil "en"
$ gcc svar en -o run    # Linker de to filene med maskinkode sammen til en
                        # kjørbar fil "run"
$ ./run                 # Kjører programmet
Svar = 41
Kaller enlinje()...
Svar = 42
```

Resultatet blir det samme fordi den eneste forskjellen er at main kaller en ekstern funksjon, men funksjonen gjør akkurat det samme som den interne i forrige oppgave.

f) Opsjonen -S ber gcc om å lage assembly-kode som tilsvarer den maskinkoden som ville bli laget med -c. Mye av koden er instruksjoner som definerer funksjoner og lagrer verdier i registere som vil bli endret ved kjøring. De tre sentrale linjene sørger for at variabelen `svar` sin verdi økes fra 41 til 42.

```
movl svar(%rip), %eax  # kopierer verdien av som ligger i RAM med addresse svar(%rip) til %eax
addl $1, %eax          # Adderer tallet 1 til registeret %eax
movl %eax, svar(%rip)  # Lagrer resultatet, 42, i RAM der hvor variablene svar ligger
```

g) Filen minimal.s er forenklet assembly-kode som gjør nøyaktig det samme som en.s og en, siden instruksjonen `incl` direkte øker det tallet som ligger i svar(%rip) med en. Dermed kan denne erstatte en.s og nøyaktig det samme vil skje om når man kjører programmet.

Oppgave 4 - Synkronisering (%)

a) I main() startes det opp to tråder (pthreads) som vil scheduleres uavhengig av hverandre av kjernen og dermed kjøres samtidig (på hver sin CPU-core når systemet har flere enn en). Begge trådene kaller funksjonen inc() som går i en løkke og kaller den eksterne funksjonen enlinje() 10 millioner ganger. Kallene på pthread_join() gjør at main() venter til begge trådene er ferdige og skriver så ut verdien av den felles variabelen svar og avslutter. Hvis trådene ikke ødelegger for hverandre, burde sluttsvaret bli 20 millioner.)

b) De to trådene vil kjøre på hver sin CPU og uavhengig av hverandre hente ut verdien av svar og øke den med 10 millioner. Men de synkroniserer ikke med hverandre når verdier hentes og lagres. Hvis en CPU henter ut verdien av svar fra RAM, øker med en og legger tilbake, vil en annen CPU kunne hente samme verdi av svar, øke denne legge tilbake, og en av oppdateringen vil da ikke bli gjort. Dette skjer ofte og nesten halvparten av oppdateringene forsvinner(De to CPUene har forskjellig cache, men disse holdes synkronisert).

c) Ved hjelp av taskset tvinges nå begge trådene til å kjøre på samme CPU-core, core nr 0, og dermed vil de to trådene multitaske på samme CPU-core. Dermed er det bare en sjelden gang når det skjer en context switch etter at den ene tråden har lastet inn svar i et lokalt register (%ax som vi så i en.s koden), men før den har utført movl og lagt svaret tilbake. Da vil tråden som context switches inn lese samme verdi av svar og så øke og lagre dette en rekke ganger(typisk noen millioner ganger, kan det se ut som) til den opprinnelige tråden kommer tilbake. Den vil så fortsette med sin verdi og alle oppdateringene til tråd nr 2 forsvinner. Men vi ser at i noen kjøringer skjer ikke dette i det hele tatt og da blir resultatet 20 millioner, slik det burde bli (I dette tilfellet har også de to trådene samme cache).

d) Når programmet linkes med minimal.s vil økningen av variabelen svar utføres ved hjelp av en enkelt instruksjon incl. Dermed vil økningen av variabelen svar oppdateres i RAM uansett når en context switch måtte intreffe, for instruksjonen fullføres før context switchen utføres. Og da vil de to trådene ikke ødelegge for hverandre, så lenge de kjører på samme CPU-kjerne.

e) Igjen ser vi at oppdateringen av variabelen svar i RAM ikke blir synkronisert når to forskjellige CPUer henter og oppdaterer den samme variabelen og de overskriver hverandres økninger av variabelen. Dette til tross for at oppdateringen utføres av en enkelt instruksjon `incl svar(%rip)`. Likevel må verdien hentes inn til CPUen via minnebussen, økes med en og sendes tilbake igjen. Og det er ingen kontroll på om den andre CPUen henter inn verdien av variabelen svar samtidig. Begge vil da samtidig øke verdien med en og skrive tilbake til RAM. Igjen forsvinner nesten halvparten av oppdateringene.

f) I lockMinimal.s brukes X86 maskininstruksjonen lock før incl. Den vil for neste instruksjon som utføres låse minnebussen slik at instruksjoner på andre CPUer ikke samtidig kan hente eller lagre noe i RAM. Dette sikrer at instruksjonen etter lock som utføres på en variabel i minne får avsluttet hele sin operasjon uten at RAM endres. Dermed blir svaret hver eneste gang 20 millioner. (Og det tar lengre tid å kjøre programmet pga synkroniseringen)

g) Om en prosess P1 går inn og kjører KritiskAvsnitt() vil den ha satt lock til true. Om P2 ønsker å gå inn, må den da stå og vente med while(lock) til P1 er ferdig. Dette sikrer som oftest at to prosesser ikke er i kritisk avsnitt samtidig. Men hva om det skjer en Context Switch rett etter while(lock) når P1 kjører? Da rekker ikke P1 å sette lock til true og P2 kunne gå inn i kritisk avsnitt, switches ut og P1 kan gå inn i kritisk avsnitt samtidig! Altså er ikke denne metoden alltid korrekt.

Oppgave 5 - Powershell (%)

a) ls — Get-Member

b) `((Get-Date) - (ls fil.txt).CreationTime).TotalDays`

c)

```
$now = get-date
```

```
foreach ($fil in ls){
   $age += $now - $fil.CreationTime
   $Nfiler++
}

$aver = $age.TotalDays/$Nfiler

"Snitt-alder $aver for $Nfiler filer og mapper"
```