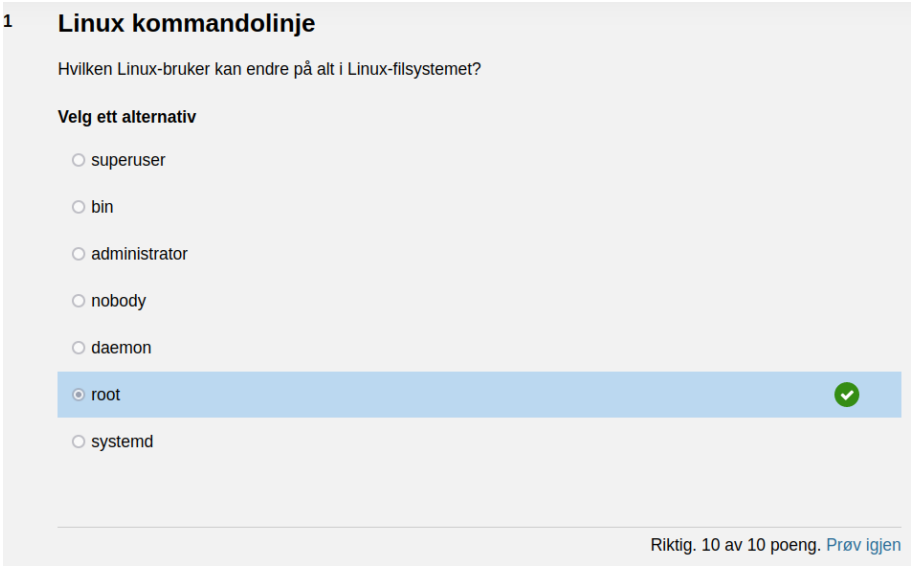
Løsningsforslag eksamen 2023 Operativsystemer

Linux kommandolinje

1)



2)

3)

HA Men den lokale variabelen $minvar vil jo ikke bli arvet av det nye shellet?

Du har rett, og jeg beklager for misforståelsen. Den lokale variabelen `**minvar**` blir ikke arvet av det nye bash-shellmiljøet. Det skyldes at lokale variabler er begrenset til det nåværende skopet eller underprosessen der de ble opprettet.

Når du starter et nytt bash-shell med kommandoen `**bash**`, vil det nye shellmiljøet ikke ha kjennskap til den lokale variabelen `**minvar**` som ble definert i det tidligere shellmiljøet. Derfor, når du skriver `**echo $minvar**` i det nye shellmiljøet, vil det ikke være noen definert verdi for `**minvar**`, og kommandoen vil gi tom utdata.

Men den globale variabelen `**DINVAR**`, som ble eksportert ved hjelp av `**export**`-kommandoen, vil bli arvet av det nye shellmiljøet. Derfor vil `**echo $DINVAR**` gi utdataen "HALLO" i det nye shellmiljøet.

Tusen takk for korrigeringen, og jeg beklager igjen for forvirringen.

Linux i praksis

4)

```
kan@os:~$ cat file1
g2k1L
```

5) Man kan lete manuelt eller bruke en kommando som:

```
kan@os:~/tusenmapper$ find . -type f
./emulex/xway/unlock/enavtusen.kode
kan@os:~/tusenmapper$ cat ./emulex/xway/unlock/enavtusen.kode
AegJL
```

6)

```
root@os:~# service docker start
 * Starting Docker: docker       [ OK ]
 root@os:~# docker ps -a
 CONTAINER ID   IMAGE          COMMAND                CREATED      STATUS                     PORTS     NAMES
 dccf270611ef   ubuntu:20.04   ``tail -f /dev/null''   5 days ago Exited (137) 5 days ago             Xhhgo
root@os:~#
```

Xhhgo

7)

```
root@os:~# docker start dcc
dcc
```

```
Tre alternativer:
```

```
root@os:~# docker exec dcc cat /root/xfile2
9W3qX

root@os:~# docker exec -it dcc bash
root@dccf270611ef:/# cat /root/xfile2
9W3qX

root@os:~# docker cp dcc:/root/xfile2 .
Successfully copied 2.05kB to /root/.
root@os:~# cat xfile2
9W3qX
```

8)

```
kan@os:~/run$ for prog in *
> do
> ./$prog | grep R3
> done
W7R3M
kan@os:~/run$
```

Scripting

9)

```
kan@os:~$ cat find.sh
#! /bin/bash

dir=$1
if [ ! -d "$dir" ]
then
   echo \"$dir\" er ikke en mappe!
   exit
fi

for prog in $dir/*
do
   res=$(./$prog | grep R3)
   if [ "$res" ]
   then
      echo "$prog" gir strengen $res
   fi
done

kan@os:~$ ./find.sh run
run/run314 gir strengen W7R3M
kan@os:~$ ./find.sh run/run500
"run/run500" er ikke en mappe!
kan@os:~$ ./find.sh
"" er ikke en mappe!
```

Følgende følger oppgaveteksten i litt større grad (men begge løsninger er ok):

```
for i in {1..500}
do
   res=$($dir/run$i | grep R3)
```

```
   if [ "$res" ]
   then
      echo "$dir/run$i" gir strengen $res
   fi
done
```

10)

```
kan@os:~/tusenfiler$ ./fredag.sh | wc -l
145
```

11)

```
(Get-ChildItem | Where-Object {$_.LastWriteTime.dayOfWeek -eq "Friday"}).Length
```

Alternativt:

```
foreach ($fil in Get-ChildItem){
    if ($fil.LastWriteTime.dayOfWeek -eq "Friday"){
       $sum++;
    }
}
$sum
```

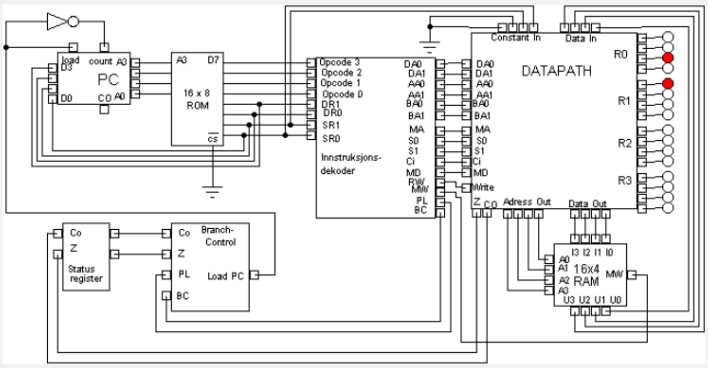
Datamaskinarkitektur

12)

12

Datamaskinarkitektur

Hvilken del av denne CPU'en inneholder ALU'en hvor innstruksjonene utføres?



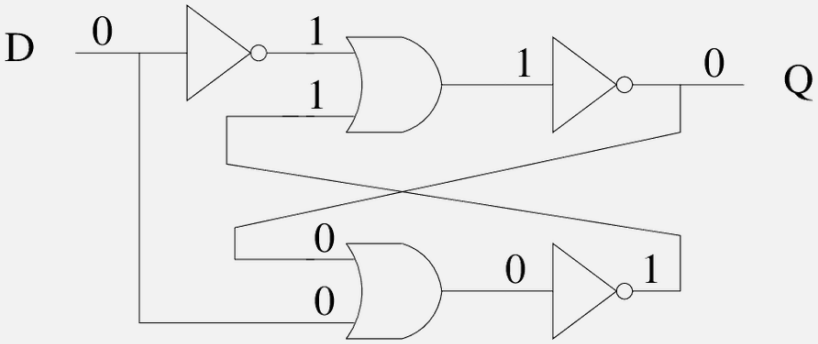
Velg ett alternativ

- PC
- RAM
- DATAPATH ✓
- Ingen av delene
- ROM
- Instruksjonsdekoder
- Branch-control

Riktig. 10 av 10 poeng. Prøv igjen

13)

13 **Konstruksjon av D-vippe**



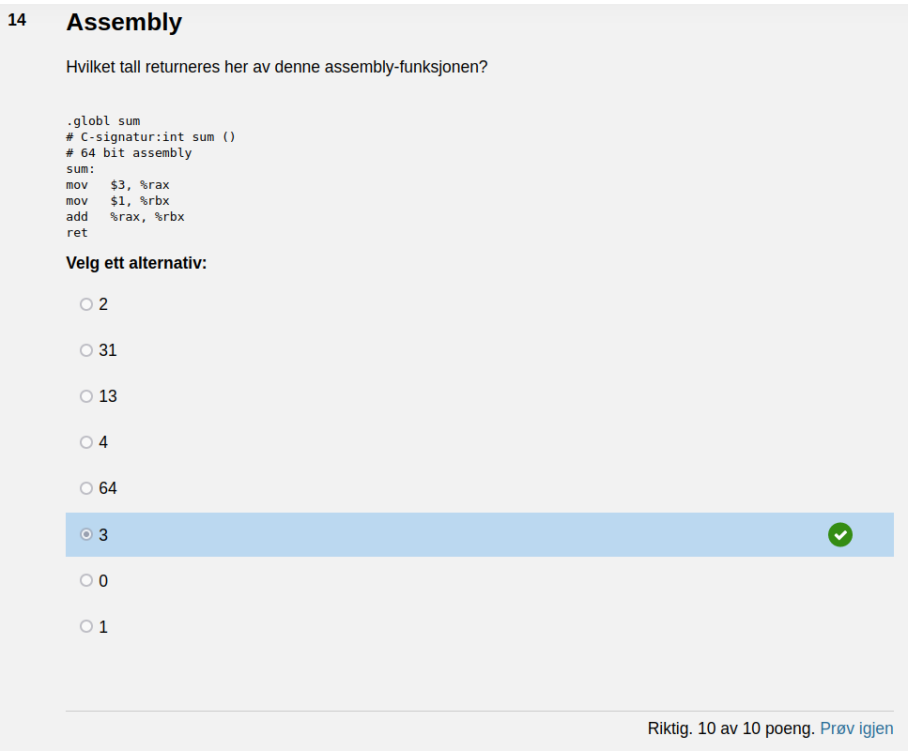
Figuren viser et forsøk på å lage en D-vippe som skal kunne brukes til å lagre ett bit i et register. Hva er problemet som gjør at denne konstruksjonen må utvides noe for å få en velfungerende D-vippe?

Velg ett alternativ:

- Om man sender inn D = 1 vil det også resultere i Q = 0
- Kretsen kortslutter når ledningene krysser hverandre
- Verdien til D kan ikke endres
- Verdien til Q kan ikke endres
- To enere inn i en OR-port gir kortslutning
- Man kan ikke velge å beholde verdien til Q selvom D endres ✓
- Siden det er tre NOT-porter vil D = 0 gi Q = 1
- Q vil hele tiden flippe mellom 0 og 1 på grunn av at signalet sendes bakover i kretsen

Riktig. 10 av 10 poeng. Prøv igjen

14)



15) Grunnen til dette er relatert til pipelining og branch predicition. Hver instruksjon blir delt opp i flere deler og kjøres delvis i parallell. Men en if-test er en forgrening (branch) og dermed kan man ikke vite hva som er neste instruksjon. Derfor gjetter CPUen basert på erfaring hvilken branch som vil bli fulgt og legger tilsvarende instruksjoner inn i pipe'en. Med et usortert array vil det hele tiden variere hvilken branch som følges og dermed vil gjettingene ofte være feil. Men når arrayet er sortert vil i første halvdel av programmet samme forgreining følges og i andre halvdel den andre forgreiningen følges. Dermed vil branch-prediction nesten alltid lykkes og programmet gpr nesten tre ganger så fort som når den feiler omtrent halvparten av gangene med et usortert array. (Om man fjerner if-testen helt fra programmet, går det omtrent like for med og uten sortering)

Serialisering av tråder

16)

```
kan@os:~$ cd java/
kan@os:~/java$ javac SynchThread.java
kan@os:~/java$ java SynchThread
Starter to threads!
Thread nr. 2, med prioritet 5 starter
Thread nr. 1, med prioritet 5 starter
Trhead nr. 2 ferdig. Saldo: 14437422
Trhead nr. 1 ferdig. Saldo: 15509217
Endelig total saldo: 14437422
kan@os:~/java$ java SynchThread
Starter to threads!
Thread nr. 2, med prioritet 5 starter
Thread nr. 1, med prioritet 5 starter
Trhead nr. 1 ferdig. Saldo: 14754622
Trhead nr. 2 ferdig. Saldo: 3198248
Endelig total saldo: 3198248
kan@os:~/java$ java SynchThread
Starter to threads!
```

```
Thread nr. 1, med prioritet 5 starter
Thread nr. 2, med prioritet 5 starter
Trhead nr. 2 ferdig. Saldo: -20912923
Trhead nr. 1 ferdig. Saldo: -6264403
Endelig total saldo: -6264403
kan@os:~/java$
```

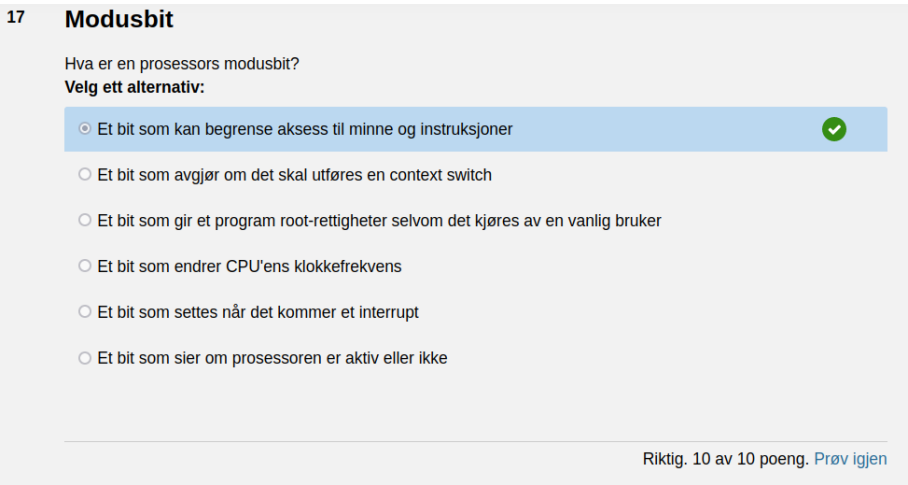
Programmet starter to tråder som begge endrer verdien på den felles variabelen saldo. Den ene tråden minker den med 1 50 millioner ganger og den andre øker den med to like mange ganger. Hvis trådene ikke ødelegger for hverandre burde altså resultatet bli 50 millioner. Men siden serialiseringen ikke virker som den skal vil de to trådene overskrive hverandres resultater når de skriver en rekke ganger til samme felles variabel i RAM uten å synkronisere skrivingen. Feilen er at objektet lock ikke er definert som static. Dermed vil hver tråd låse med synchronized(lock) kun sin egen variabel lock og dermed blir det ingen synkronisering. Trådene må ha en felles lock-variabel og det oppnås ved å deklarere den som static. I den opprinnelige koden har hver tråd sin egen monitor-lås og den vil ikke holde den andre tråden ute fra kritisk avsnitt. Etter kompilering med `public static Object lock` blir kjøringen:

```
kan@os:~/java$ java SynchThread
Starter to threads!
Thread nr. 1, med prioritet 5 starter
Thread nr. 2, med prioritet 5 starter
Trhead nr. 1 ferdig. Saldo: 82351407
Trhead nr. 2 ferdig. Saldo: 50000000
Endelig total saldo: 50000000
```

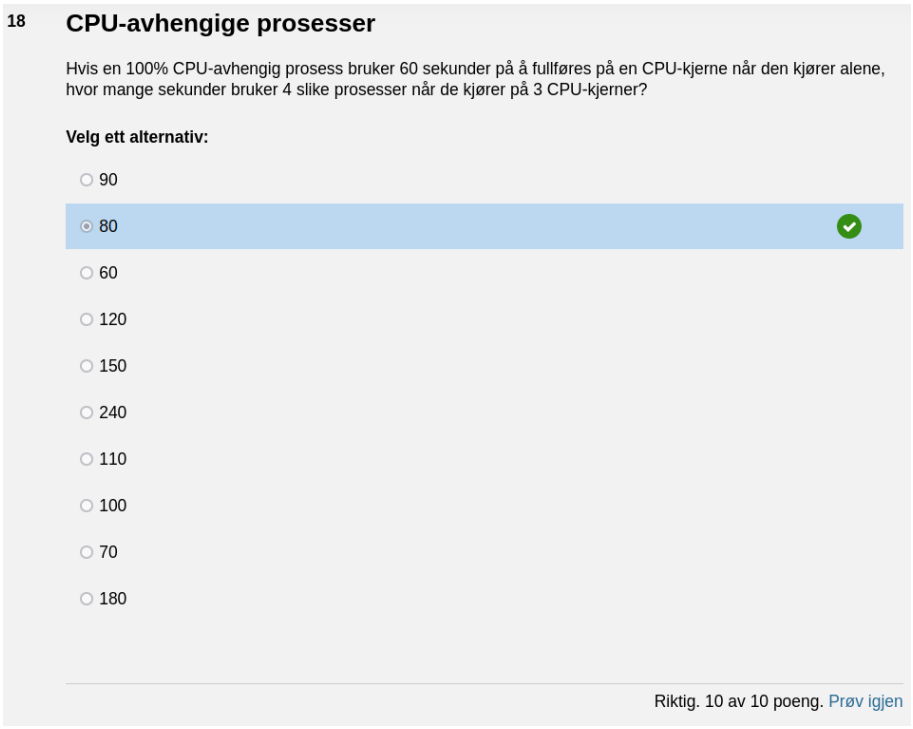
og resultatet blir alltid 50 000 000.

Prosesser

17)



18)



19)

```
kan@os:~/regn$ time ./run1
Real:3.744
kan@os:~/regn$ time ./run2
Real:1.882
kan@os:~/regn$ cat run1
#! /bin/bash

./regn
./regn
kan@os:~/regn$ cat run2
#! /bin/bash

./regn&
./regn
```

I run1 kjøres først en instans av regn alene og så en ny instans av regn. I run2 legges regn-programmet i bakgrunnen slik at også det andre regn-programmet startes samtidig. Dermed vil de kunne kjøre på hver sin CPU-kjerne og det går omtrent dobbelt så fort som når en av gangen kjøres.

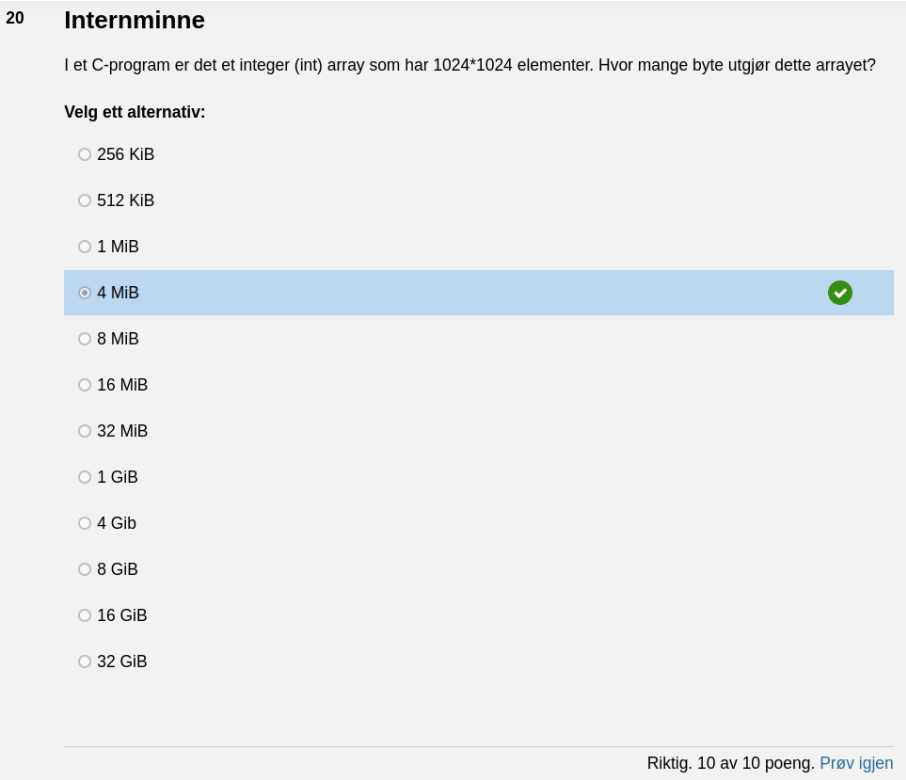
```
kan@os:~/regn$ time taskset -c 0 ./run1
Real:3.725
kan@os:~/regn$ time taskset -c 0 ./run2
Real:3.736
```

I begge disse tilfellene tvinges prosessene til å kjøre på samme CPU-kjerne (0) og dermed vil de to prosessene i run2 måtte dele på samme CPU-kjerne og det går omtrent like fort som å kjøre dem etterhverandre (den siste metoden kan gå litt saktere på grunn av flere context-switcher, men forskjellen er veldig liten og mindre enn den naturlige variasjonen i hvor lang tid programmet tar, så det er vanskelig å se med ett enkelt eksperiment).

Under eksamen i Inspera var det ofte så mange som kjørte disse prosessene samtidig at tiden de brukte kunne variere. Men man kunne likevel basere sin argumentasjon på hvor lang tid det i teksten ble fortalt programmene skulle bruke.

Internminne

20)



21)