Løsningsforslag eksamen høst 2022 Operativsystemer

Datamaskinarkitektur

1)

1 **Logiske porter**

Hvilke input til en OR-port vil gi output lik 1?

(Ett eller flere riktige svar. Trekk for gale svar)

Velg ett eller flere alternativer

- ☐ true
- ☐ 00
- ☐ 0
- ☐ false
- ☑ 11 ✓
- ☐ 1
- ☑ 10 ✓
- ☑ 01 ✓

Riktig. 10 av 10 poeng. Prøv igjen

2)

2 **Logiske porter**

Skriv inn sannhetstabellen for en AND-port med input A og B:

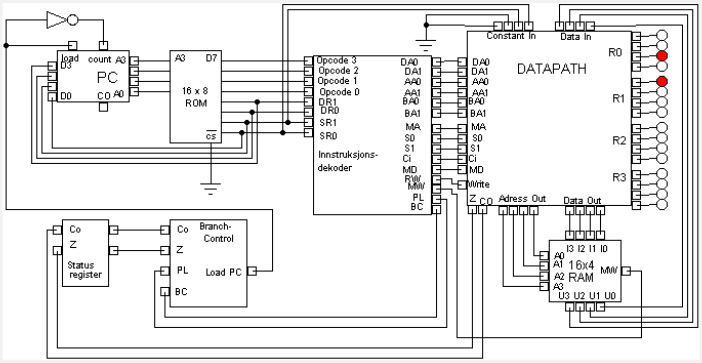
A	B	UT
0	0	0 ✓
0	1	0 ✓
1	0	0 ✓
1	1	1 ✓

Riktig. 10 av 10 poeng. Prøv igjen

3)

3 **Datamaskinarkitektur**

Hvilken del av denne CPU'en sørger for at det er mulig å hoppe i koden, slik at man kan lage if-tester og løkker i programmet?



Velg ett alternativ

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

Riktig. 10 av 10 poeng. Prøv igjen

Linux i praksis

4)

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

5)

```
kan@os:~/mangefiler$ grep rett *
fil133:rett fil
kan@os:~/mangefiler$ cat fil133
kZMKa
rett fil
```

6)

```
kan@os:~/mangefiler$ sudo find / -name sys.txt
/usr/lib/kernel/sys.txt
kan@os:~/mangefiler$ cat /usr/lib/kernel/sys.txt
IOaqD
```

7)

```
chmod 640 chmod.txt
```

8)

```
kan@os:~/tusenfiler$ find . -type f -newermt "11 Feb 2013 00:00" -a ! -newermt "11 Feb 2013 23:59" -ls
  6428149      4 -rw-rw-r--   1 kan      kan             6 Feb 11  2013 ./fil634.txt
kan@os:~/tusenfiler$ cat fil634.txt
DAsNB
```

Alternativer:

```
kan@os:~/tusenfiler$ ls -l | grep 2013
-rw-rw-r-- 1 kan kan 6 Aug 26  2013 fil10.txt
-rw-rw-r-- 1 kan kan 6 Jul  8  2013 fil229.txt
-rw-rw-r-- 1 kan kan 6 May 22  2013 fil515.txt
-rw-rw-r-- 1 kan kan 6 Oct 11  2013 fil517.txt
-rw-rw-r-- 1 kan kan 6 Dec  4  2013 fil559.txt
-rw-rw-r-- 1 kan kan 6 Feb 11  2013 fil634.txt
-rw-rw-r-- 1 kan kan 6 Nov 18  2013 fil653.txt
-rw-rw-r-- 1 kan kan 6 Dec 23  2013 fil705.txt
-rw-rw-r-- 1 kan kan 6 Mar 30  2013 fil766.txt
-rw-rw-r-- 1 kan kan 6 Mar 19  2013 fil807.txt
-rw-rw-r-- 1 kan kan 6 May 31  2013 fil885.txt
-rw-rw-r-- 1 kan kan 6 Jul 17  2013 fil888.txt
kan@os:~/tusenfiler$ ls -lt | grep 2013
-rw-rw-r-- 1 kan kan 6 Dec 23  2013 fil705.txt
-rw-rw-r-- 1 kan kan 6 Dec  4  2013 fil559.txt
-rw-rw-r-- 1 kan kan 6 Nov 18  2013 fil653.txt
-rw-rw-r-- 1 kan kan 6 Oct 11  2013 fil517.txt
-rw-rw-r-- 1 kan kan 6 Aug 26  2013 fil10.txt
-rw-rw-r-- 1 kan kan 6 Jul 17  2013 fil888.txt
-rw-rw-r-- 1 kan kan 6 Jul  8  2013 fil229.txt
-rw-rw-r-- 1 kan kan 6 May 31  2013 fil885.txt
-rw-rw-r-- 1 kan kan 6 May 22  2013 fil515.txt
-rw-rw-r-- 1 kan kan 6 Mar 30  2013 fil766.txt
-rw-rw-r-- 1 kan kan 6 Mar 19  2013 fil807.txt
-rw-rw-r-- 1 kan kan 6 Feb 11  2013 fil634.txt
```

9)

```
kan@os:~$ sudo su
root@os:/home/kan# service docker start
 * Starting Docker: docker       [ OK ]
root@os:/home/kan# docker image ls
REPOSITORY   TAG       IMAGE ID       CREATED       SIZE
ubuntu       20.04     b7bab04fd9aa   2 weeks ago   72.8MB
```

ID=b7bab04fd9aa

10)

```
root@os:/home/kan# docker container ps -a
CONTAINER ID   IMAGE          COMMAND               CREATED      STATUS                      PORTS     NAMES
9d6343d4e020   ubuntu:20.04   "tail -f /dev/null"   3 days ago   Exited (137) 2 days ago             elated_meitne
root@os:/home/kan# docker start 9d
```

```
9d
root@os:/home/kan# docker exec -it 9d bash
root@9d6343d4e020:/# cd
root@9d6343d4e020:~# cat xfile2
cAYbl
```

alternativt:

```
root@os:~# docker start 9d
9d
root@os:~# docker cp 9d:/root/xfile2 .
root@os:~# cat xfile2
cAYbl
```

11) To typiske kjøringer:

```
kan@os:~/lock$ ./a.out
Starter; svar verdi: 0
Starter; svar verdi: 0
Avslutter; svar verdi: 7237388
Avslutter; svar verdi: 10395395
Main avslutter; svar verdi: 10395395
kan@os:~/lock$ ./a.out
Starter; svar verdi: 0
Starter; svar verdi: 0
Avslutter; svar verdi: 10072023
Avslutter; svar verdi: 10403016
Main avslutter; svar verdi: 10403016
kan@os:~/lock$
```

I thread.c lager main tråder som kjører uavhengig av hverandre og der begge med funksjonen inc() oppdaterer en felles variabel med navn svar ti millioner ganger. Deretter venter main på at begge trådene skal bli ferdig og skriver ut den totale summen. Her øker begge tråder verdien til variabelen like mange ganger og dermed vet vi at verdien må bli det dobbelte av det hver tråd øker med hvis det ikke inntreffer en race condition; altså 20 millioner. Når programmet kjøres blir resultatet forskjellig hver gang og bare i overkant av 10 millioner. Det betyr at race conditions intreffer hele tiden og at trådene overskriver hverandres resultater. Siden det er to tilgjengelige CPUer vil de kjøre på hver sin CPU og siden det ikke er noen koordinering CPUene imellom om å vente på hverandre når en variabel skal hentes fra RAM, vil de fortløpende overskrive hverandres resultater. (at svaret er rett over 10 millioner er forenlig med det som skjer hvis begge henter ut verdien 1 omtrent samtidig og øker den til 2 og så skriver tilbake omtrent samtidig, da vil den totale økningen være 1, mens den burde vært 2)

Opsjonen -O omtales i manualsiden for gcc med "With -O, the compiler tries to reduce code size and execution time". Dette betyr at kompilatoren lager kjørbar kode som er minst mulig og kjører raskest mulig.

12) Når programmet kjøres med `taskset -c 0` vil begge trådene tvinges til å kjøre på samme CPU. Når man kompilerer med opsjonen -O vil kompilatoren lage mest mulig effektiv kode og C-kodelinjen `svar++;` vil derfor oversettes til en enkelt maskininstruksjon. Dermed vil økningen av verdien til svar aldri kunne avbrytes av en context switch siden denne instruksjonen alltid vil fullføres før den andre tråden tar over.

Når programmet derimot ikke kompileres med opsjonen -O, vil C-kodenlinjen `svar++;` oversettes til tre linjer maskinkode, slik man kan se i neste oppgave. Dermed vil vanligvis dette skje:

```
kan@os:~/lock$ taskset -c 0 ./a.out
```

```
Starter; svar verdi: 0
Starter; svar verdi: 2023319
Avslutter; svar verdi: 19379144
Avslutter; svar verdi: 20000000
Main avslutter; svar verdi: 20000000
```

men ca en av fem ganger skjer noe lignende dette:

```
kan@os:~/lock$ taskset -c 0 ./a.out
Starter; svar verdi: 0
Starter; svar verdi: 2275142
Avslutter; svar verdi: 19399210
Avslutter; svar verdi: 16998306
Main avslutter; svar verdi: 16998306
```

Når dette programmet kjøres med `taskset -c 0` vil begge trådene tvinges til å kjøre på samme CPU.

Dermed vil de stort sett kjøres hver for seg og resultatet blir 20 millioner. Men en sjelden gang kommer det en context switch rett etter at en tråd har økt et rigister lokalt med en og før tråden har rukket å skrive til RAM og da vil den andre trådens resultater bli overskrevet når en context switch går tilbake til den første tråden igjen. Dette må også bety at den ene linjen `svar++` i programmet utgjør flere linjer maskinkode, det vil si at verdien av svar først lastes ned i et register, så økes og lastes opp til RAM igjen (se neste oppgave).

13) Når man kompilerer en.c på denne måten får man som resultat en fil en.s som viser assembly-kode som tilsvarer den maskinkoden som lages når en.c blir kompilert til maskinkode. Hvis man kompilerer uten opsjonen -O og ser på koden inne i `enlinje:` som inneholder den kompilerte assemblykoden for C-instruksjonen `svar++`, kan man se følgende tre linjer:

```
movl svar(%rip), %eax
addl $1, %eax
movl %eax, svar(%rip)
```

Disse linjene viser at svar først blir lastet inn fra RAM til registeret %eax, dette registeret økes med en og resultatet lastes ut i RAM igjen. Hvis det skjer en context switch til den andre tråden etter den første eller den andre av disse instruksjonene, vil tredje linje overskrive alle addisjonene som den andre tråden gjør i mellomtiden. Og dette forklarer resultatene fra forrige oppgave.

Når man derimot kompilerer med opsjonen -O vil man se av koden i en.s at `svar++` oversettes til en enkelt linje:

```
        addl    $1, svar(%rip)
```

Dermed vil en context switch aldri kunne ødelegge for beregningene når trådene kjøres på samme CPU med taskset og resultatet blir alltid det samme.

Prosesser

14)

14 **Threads og cores**

På en Linux server med hyperthreading får du følgende output fra en kommando:

```
$ grep "" /sys/devices/system/cpu/cpu*/topology/thread_siblings_list
/sys/devices/system/cpu/cpu0/topology/thread_siblings_list:0,4
/sys/devices/system/cpu/cpu1/topology/thread_siblings_list:1,5
/sys/devices/system/cpu/cpu2/topology/thread_siblings_list:2,6
/sys/devices/system/cpu/cpu3/topology/thread_siblings_list:3,7
/sys/devices/system/cpu/cpu4/topology/thread_siblings_list:0,4
/sys/devices/system/cpu/cpu5/topology/thread_siblings_list:1,5
/sys/devices/system/cpu/cpu6/topology/thread_siblings_list:2,6
/sys/devices/system/cpu/cpu7/topology/thread_siblings_list:3,7
```

Utifra dette resultatet kan man konkludere med at serveren har

4 ✓ cores og

2 ✓ thread(s) per core.

Fyll inn riktig tall i boksene over.

Riktig. 10 av 10 poeng. Prøv igjen

15)

15 **En CPU-avhengige prosess**

En 100% CPU-avhengig prosess kjører alene og direkte på en CPU på en Linux-server uten virtualisering. Du ser på statistikk fra /proc/stat for CPUen den kjører på. I hvilken modus er da neste alle ticks som rapporteres?

Velg ett alternativ:

- ○ irq
- ○ system
- ○ guest
- ○ softirq
- ◉ user ✓
- ○ idle
- ○ steal
- ○ nice
- ○ guest_nice
- ○ iowait

Riktig. 10 av 10 poeng. Prøv igjen

Internminne og disk

16)

16 **Lagringsenheter**

En byte skal kopieres til et register i CPU. Hvilken av følgende enheter går det raskest å kopiere fra?

Velg ett alternativ

- SSD disk
- L3 cache
- RAM
- L2 cache
- L1 cache ✓
- HDD disk

Riktig. 10 av 10 poeng. Prøv igjen

17)

17 **RAID 3**

Anta at du har et RAID 3 oppsett med 4 disker og en paritets-disk. Alle dataene på disk 3 har gått tapt etter en disk-crash. Bruk informasjonen fra paritets-disken til å gjennopprette det som var på disk 3 og skriv inn 0 eller 1 for hver rad på disk 3.

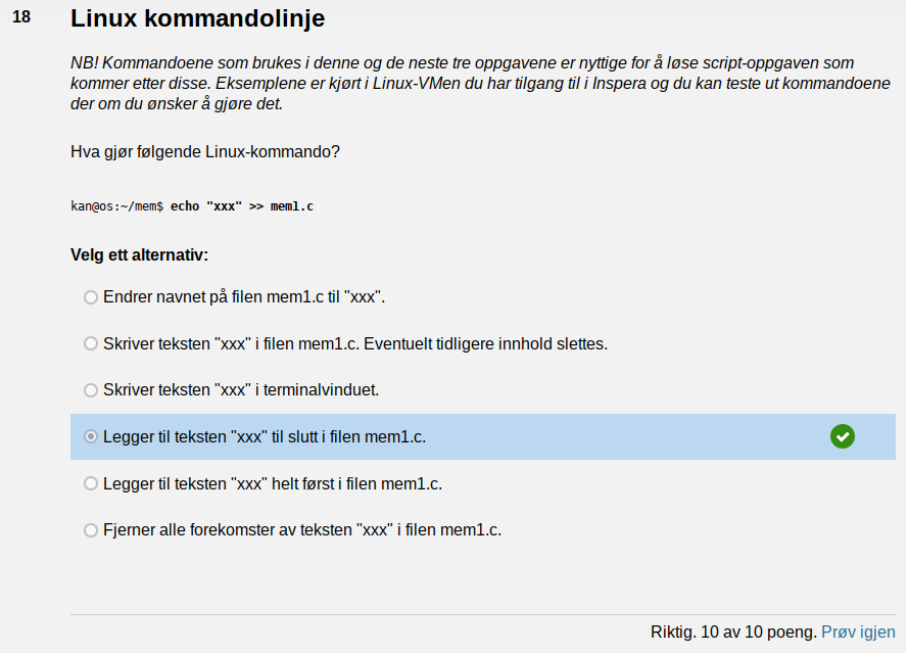
RAID 3

disk 1	disk 2	disk 3	disk 4	paritets-disk
0	0	1 ✓	1	0
0	1	1 ✓	1	1
1	0	0 ✓	0	1
0	0	0 ✓	1	1
1	1	1 ✓	0	1
0	1	0 ✓	1	0

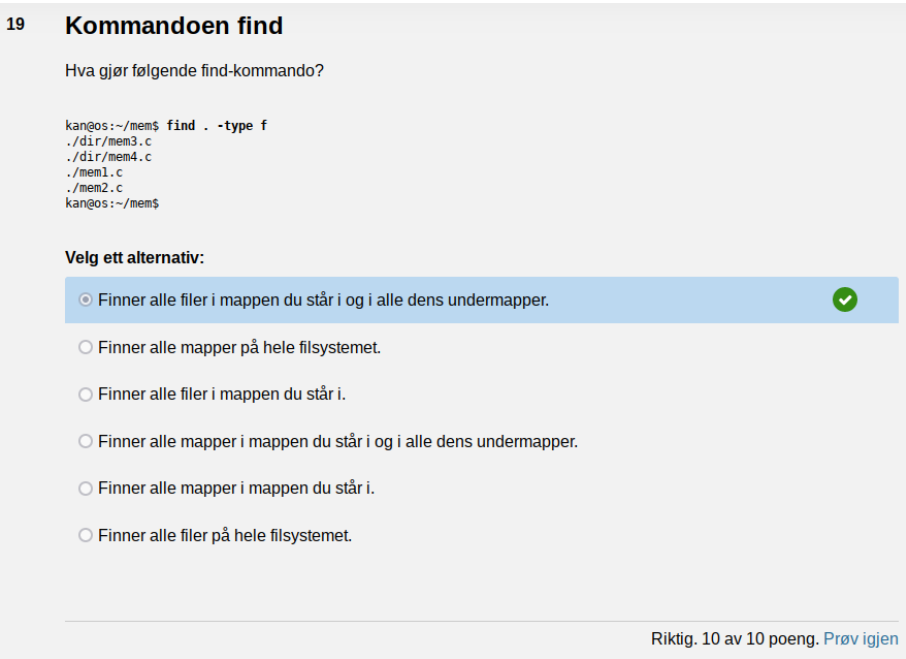
Riktig. 10 av 10 poeng. Prøv igjen

Bash

18)



19)



20)

1. md5sum renger ut en hash-streng for filen mem1.c og skriver ut denne strengen.
2. tekst-strengen 'xxx' legges til i filen mem1.c
3. md5sum renger på nytt ut en hash-streng for filen mem1.c og skriver ut denne strengen. Etter endringen av filen kan man se at hash-strengen nå er totalt forskjellig.

Ved å lagre hash-strengen for en fil kan man kontrollere om filen har blitt endret. Hvis man regner ut

hash-strengen på nytt og den har endret seg, vet man at filen også har endret seg.

21)

1. hash for mem1.c lages og skrives til sum.txt (eventuelt innhold overskrives)
2. hash for mem2.c lages og legges til filen sum.txt
3. strengen 'xxx' legges til mem1.c
4. -c opsjonen til md5sum gjør at den sjekker alle md5-summer som ligger i sum.txt og varsler om FAILED hvis den er endret og gir OK om den ikke er endret. Og skriver ut hvor mange av summene som ikke matchet til slutt.

22)

```
#! /bin/bash

if [ -f sum.txt ]
then
   md5sum -c sum.txt
else
   for file in $(find . -type f)
   do
      md5sum $file >> sum.txt
   done
fi
```

PowerShell

23) Kommandoene som finner full path til alle filer i en mappe og dens undermapper, er:

```
(Get-ChildItem -Recurse -File).FullName
```

(som også er det man må endre scriptet i siste oppgave til for at det skal virke korrekt)

24) I PowerShell blir strengen

```
Get-FileHash mem1.c -Algorithm MD5
1FD4C21162F21633E862B3AF2E0511A8
```

mens den i bash blir helt det samme

```
kan@os:~/win$ md5sum mem1.c
1fd4c21162f21633e862b3af2e0511a8  mem1.c
```

bortsett fra at bokstavene er skrevet med små bokstaver, så det er tydligvis samme hashing-algoritme.

Kommandoen som gir kun hash-strengen er:

```
(Get-FileHash mem1.c -Algorithm MD5).Hash
```

(som også brukes i andre del av scriptet i neste oppgave)

25) Problemet er at scriptet ikke finner mem3.c og mem4.c som ligger under dir-mappen (i /home/kan/win/dir). Dette er fordi scriptet bare skriver navnet på filene (mem3.c og mem4.c) og det går fint for filene som ligger øverst, men scriptet vil ikke finne filene under dir. Det kan fikses ved å endre .Name til .FullName i den siste else-løkken:

```
foreach ($file in (Get-ChildItem -Recurse -File).Name){ # Finner alle filer
```

må endres til

```
foreach ($file in (Get-ChildItem -Recurse -File).FullName){ # Finner alle filer
```

Dermed vill full path lagres i sum.txt og scriptet virker som det skal (siden det da vil stå /home/kan/win/dir/mem3.c i sum.txt og ikke bare mem3.c).