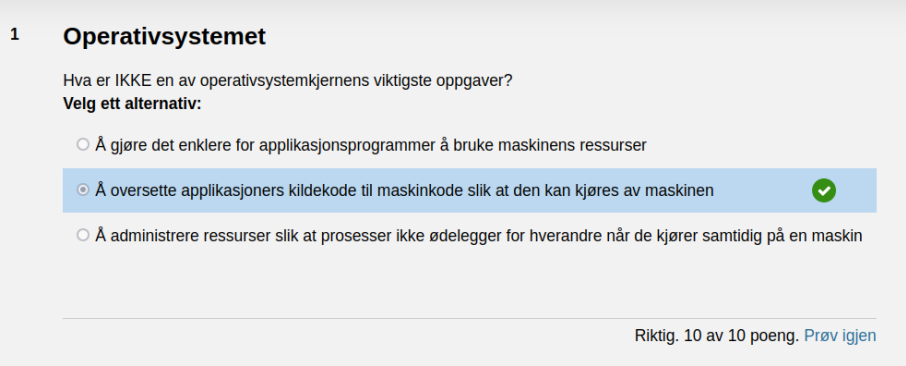
Løsningsforslag eksamen 2023 Operativsystemer

Operativsystemet og prosesser

1)



2)

l 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.

3)

En trap er forårsaket av prosessen selv og skjer på bestemete steder i koden. Hvis samme program kjører om og om igjen vil en trap skje på nøyaktig samme sted hver gang. Et interrupt er derimot forårsaket av en ytre hendelse og vil kunne komme på alle mulige tidspunkter. Timer-interrupts kommer riktignok regelmessig, men tidspunktet bestemmes ikke av koden som kjørers men av en timer.

4)

Et systemkall gir en prosess som kjører i user mode muligheten til å utføre operativsystem-funksjoner inne i OS-kjernen, inkludert maskininstruksjoner som det bare er mulig å utføre når prosessoren er i kernel mode. En slik prosess bruker systemkall til å utføre OS-kjerne tjenester.

5)

Svaret for 4 CPUer er galt, fordi scheduleren fordeler CPU-tid jevnt mellom alle prosessene som kjøres. De kjøres ikke i to grupper med 4 og to prosesser som ChatGPT mener. Alle 6 kjører samtidig, men to vil da tii en hver tid vente. Og generelt er fremgangsmåten feil, prosessene kjører samtidig, ikke sekvensielt. Men for de andre tilfellene blir svaret likevel riktig.

Prosessene scheduleres til å bytte på å bruke de tilgjengelige CPUene slik at alle får like mye CPU-tid. Totalt behøves 6 CPU * 10 miutter = 60 CPU*minutter. Med for eksempel 4 CPUer tilgjengelig tar det da 60 CPU*minutter/4 CPU = 15 minutter å få alle prosessene ferdig.

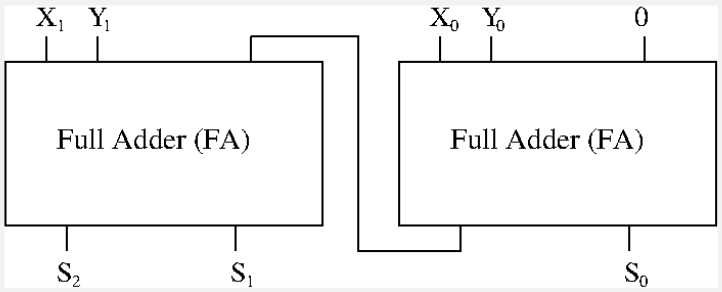
CPUer	beregning	minutter
1	60/1	60
2	60/2	30
4	60/4	15
6	60/6	10
8	60/6	10

Hvis det ikke er flere CPUer enn prosesser, kan man altså dele antall CPU-minutter som må utføres på antall CPUer. Når det er flere CPUer enn prosesser, blir ikke disse utnyttet. Med 8 CPUer vil bare 6 av CPUene brukes og det vil totalt ta 10 minutter å bli ferdig.

Datamaskinarkitektur

6)

6 **Full Adder**



Anta for denne kretsen at $X_1 = 1, X_0 = 0, Y_1 = 1, Y_0 = 1$
Hva blir da resultatet $S_2S_1S_0$?
Velg ett alternativ

- 010
- 011
- 101 ✓
- 000
- 001
- 110
- 100
- 111

Riktig. 10 av 10 poeng. Prøv igjen

7)

7 **Datamaskinarkitektur, CPU**

Hvilken av følgende enheter er ikke en del av CPUen?
Velg ett alternativ:

- Datapath
- Registere
- RAM ✓
- ALU
- Branch-Control
- Instruksjonsdekoder
- Program Counter

Riktig. 10 av 10 poeng. Prøv igjen

8)

8 **Simulerings-CPU**

For å kunne kjøre koden må CPU'en ha maskinkode. Bruk tabellen nedenfor og oversett linje 5-7 i assemblykoden vist under til maskinkode.

Instruksjonssett

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

```
5 ADD R0 <- R0 + R1
6 CMP R0 R2
7 JNE 4 (Jump Not Equal 4, hopp til linje 4 hvis R0 != R2)
```

Skriv inn binærkoden for line 5-7 med nuller og enere her:

5	0 ✓	1 ✓	0 ✓	0 ✓	0 ✓	0 ✓	0 ✓	0 ✓	0 ✓	1 ✓
6	1 ✓	1 ✓	0 ✓	0 ✓	0 ✓	0 ✓	0 ✓	1 ✓	0 ✓
7	1 ✓	1 ✓	1 ✓	1 ✓	0 ✓	1 ✓	0 ✓	0 ✓

Riktig. 10 av 10 poeng. Prøv igjen

Linux kommandolinje

9)

9

Linux kommandolinje

Hvilken mappe referer / til i et bash-shell?

Velg ett alternativ

- Roten av filsystemet ✓
- Mappen du står i
- Hjemme-mappen din
- Mappen over den du står i
- Hjemme-mappen til root
- Mappen under den du står i
- En skjult mappe

Riktig. 10 av 10 poeng. Prøv igjen

10)

10

Kommandoen find

Hva gjør følgende find-kommando?

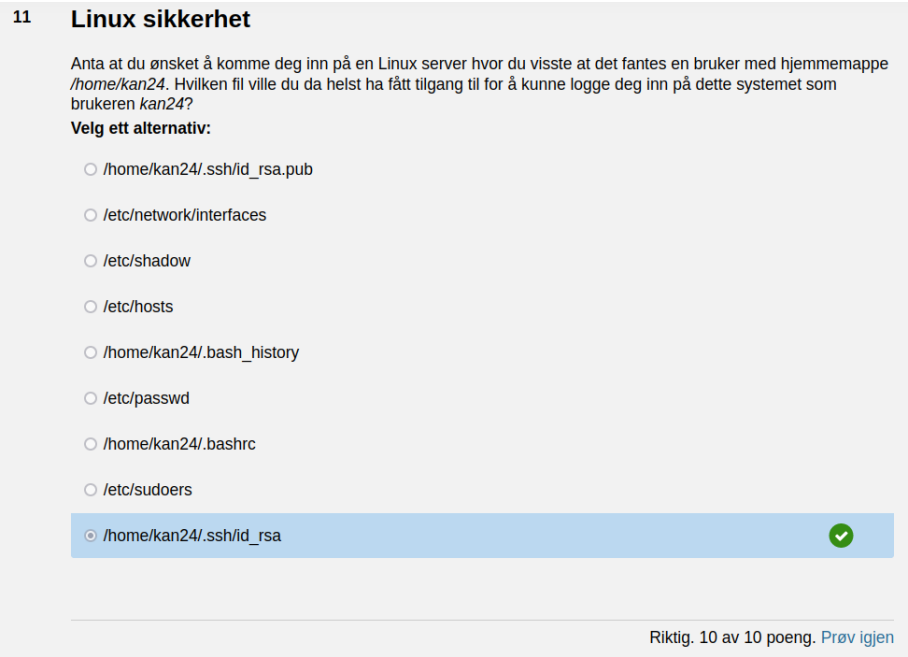
```
kan24@os:~$ find mem -executable -type f
mem/mem4
mem/mem1
mem/mem2
mem/mem3
kan24@os:~$
```

Velg ett alternativ:

- Finner alle filer på hele filsystemet.
- Finner alle kjørbare filer i mappen mem og i alle dens undermapper. ✓
- Finner alle mapper i mappen mem.
- Finner alle filer i mappen mem og i alle dens undermapper.
- Finner alle mapper i mappen mem og i alle dens undermapper.
- Finner alle mapper på hele filsystemet.
- Finner alle filer i mappen mem.
- Finner alle mapper som kan aksesseres i mappen mem og i alle dens undermapper.

Riktig. 10 av 10 poeng. Prøv igjen

11)



Linux i praksis

12)

```
kan@os:~$ cat file1
p5QVZ
```

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

```
kan24@os:~$ find mapper24 -name fil.txt
mapper24/halt.target.wants/udev/usb/fil.txt
kan24@os:~$ cat mapper24/halt.target.wants/udev/usb/fil.txt
RnNFg
```

14)

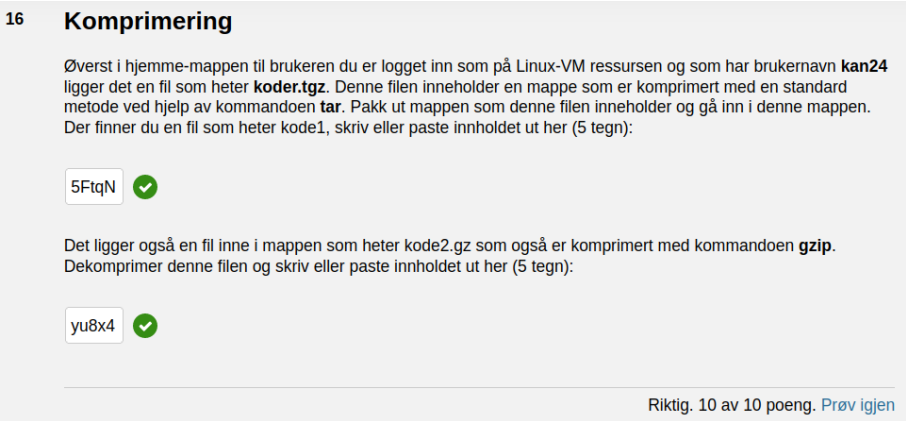
15)

```
kan24@os:~$ cat ./-x
W7R3M
```

Kan også sees med `jed -x`.

16)

```
kan24@os:~$ cat ./-x
W7R3M
kan24@os:~$ tar xfz koder.tgz
kan24@os:~$ cd koder
kan24@os:~/koder$ ls -l
total 8
-rw-r--r-- 1 kan24 kan24  6 May 19 12:49 kode1
-rw-r--r-- 1 kan24 kan24 32 May 19 12:49 kode2.gz
kan24@os:~/koder$ cat kode1
5FtqN
kan24@os:~/koder$ gunzip kode2.gz
kan24@os:~/koder$ ls -l
total 8
-rw-r--r-- 1 kan24 kan24 6 May 19 12:49 kode1
-rw-r--r-- 1 kan24 kan24 6 May 19 12:49 kode2
kan24@os:~/koder$ cat kode2
yu8x4
```



17)

```
kan24@os:~$ env | grep CODE
CODE=dh2if
kan24@os:~$ cat /etc/environment
PATH="/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin:/usr/games:/usr/local/games:/snap/bin"
CODE=dh2if
kan24@os:~$ grep CODE ~/.bashrc
export CODE=dh2if
kan24@os:~$ set | grep CODE
CODE=dh2if
```

Så egentlig 4 forskjellige måter.

18)

```
kan24@os:~$ for fil in $(find mapper24 -executable -type f)
> do
> ./$fil | grep EX
> done
yEXZB
```

18 **Kjørbare filer på Linux-VM**

Øverst i hjemme-mappen til brukeren du er logget inn som på Linux-VM og som har brukernavn **kan24** ligger det en mappe som heter **mapper24**. I denne mappen og nedover i dens undermapper ligger det blant annet 108 filer som heter **run1**, **run2**, ... , **run108**. Dette er program som kan kjøres og som alle gir en streng på 5 tegn når de kjøres. Ett av programmene gir når det kjøres en streng på 5 tegn som inneholder substrengen "EX" (som for eksempel "ahEXf"). *Legg merke til at denne strengen ikke finnes som en streng inne i programmet men er et resultat av å kjøre programmet.* Finn ut hvilken av de 108 strengene du får når du kjører run-programmene som inneholder "EX" og skriv eller kopier og paste ut denne tekststrengen her (5 tegn):

yEXZB

PS! Du blir i neste oppgave om bash-scripting bedt om å skrive et script som løser denne oppgaven. Dermed vil du for kunne få uttelling for de forsøkene du gjør for å løse denne oppgaven, selvom du ikke finner frem til den riktige strengen.

Riktig. 10 av 10 poeng. Prøv igjen

Bash-scripting

19)

```
kan24@os:~$ cat find.sh
for fil in $(find mapper24 -executable -type f)
do
      res=$($fil | grep EX)
      if [ "$res" ]
      then
         echo "Filen heter ./$fil"
         echo "Den gir strengen $res"
      fi
done
kan24@os:~$ ./find.sh
Filen heter ./mapper24/qat_c62xvf/epx/run81
Den gir strengen yEXZB
```

Alternativt

```
kan24@os:~$ cat find.sh
for fil in $(find mapper24 -type f)
do
   if [ -x $fil ]
   then
      res=$($fil | grep EX)
      if [ "$res" ]
      then
         echo "Filen heter ./$fil"
         echo "Den gir strengen $res"
      fi
   fi
done
```

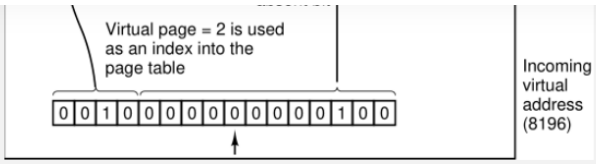
Om man eksplisitt leter i /home/kan24/mapper24 blir utskriften litt anderledes. Også ok om man begrenser søket til kun run-filer som i forrige oppgave.

Internminne

20)

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.

21)



Velg ett alternativ:

- 11
- 8204
- Gir page fault
- 24592
- 18
- 34088
- 31902
- 30128 ✓
- 24580
- 8200
- 8196
- 4096

Riktig. 10 av 10 poeng. Prøv igjen

22)

VIRT er den totale mengden virtuelt minne som er satt av for en prosess. Alt prosessen kan tenkes å bruke, men som ikke nødvendigvis ligger i RAM. I mem1.c deklareres det ikke noe ekstra RAM, men i mem2.c deklareres det et integer array med 1024*256 integer variabler. Hver av disse heltallsvariablene bruker 4 byte og dermed er den totale mengden ekstra virtuelt minne som deklareres i mem2.c 4*256*1024 byte = 1024*1024 byte = 1024 KByte (siden 1024 byte = 1 KByte). Dermed øker VIRT med 1024 siden top default rapporterer antall KByte i RAM-kolonnene.

23)

Programmet skriver ut at det skal lage et array med 'size' elementer. I neste linje allokeres det dynamisk et array med navn 'array' ved hjelp av et kall til malloc(). sizeof(int) er lik 4 siden en integer C-variabel er på

4 byte og dermed allokeres det rette antall byte for et array med 'size' elementer.

Verdien av VIRT øker fra 3668 for mem2 til 4828 for mem3:

```
kan24@os:~/mem$ echo '4828 - 3668 ' | bc
1160
```

Vi ser at VIRT har økt med 1024 + 136 byte som er størrelsen på arrayet pluss noen byte som brukes til å organisere minnebruken.

24)

Det som skjer i mem4.c er at arrayet med navn 'array' blir tatt i bruk og tilordnes verdier. RES er den delen av en prosess sitt virtuelle minne (VIRT) som er i bruk nå og fysisk ligger i RAM. Man ser at verdien av RES øker fra 952 for mem3 til 2444 for mem4:

```
kan24@os:~/mem$ echo '2444 - 952  ' | bc
1492
```

Verdien av RES har derfor økt med 1024 + 468, altså tas en vesentlig større del av RAM i bruk til administrasjon av minnehåndering enn de 1024 byte som selve arrayet utgjør. Verdien av VIRT for de to prosessene er akkurat den samme, 4828, for størrelsen på det virtuelle minne for en prosess endres ikke når minnet tas i bruk.

25)

Data som allokeres legges på heap. Slik ser maps for mem3 ut:

```
kan24@os:~/mem$ cat /proc/118/maps
5562067ad000-5562067ae000 r--p 00000000 00:31 38798404                   /var/lib/docker/overlay2/d107d82812dc20a15e4909114e94499ee981db3a63a5dd8
5562067ae000-5562067af000 r-xp 00001000 00:31 38798404                   /var/lib/docker/overlay2/d107d82812dc20a15e4909114e94499ee981db3a63a5dd8
5562067af000-5562067b0000 r--p 00002000 00:31 38798404                   /var/lib/docker/overlay2/d107d82812dc20a15e4909114e94499ee981db3a63a5dd8
5562067b0000-5562067b1000 r--p 00002000 00:31 38798404                   /var/lib/docker/overlay2/d107d82812dc20a15e4909114e94499ee981db3a63a5dd8
5562067b1000-5562067b2000 rw-p 00003000 00:31 38798404                   /var/lib/docker/overlay2/d107d82812dc20a15e4909114e94499ee981db3a63a5dd8
5562067b2000-5562068b2000 rw-p 00000000 00:00 0
556207547000-556207568000 rw-p 00000000 00:00 0                          [heap]
7fb91061d000-7fb910721000 rw-p 00000000 00:00 0
7fb910721000-7fb910749000 r--p 00000000 00:31 38681179                   /var/lib/docker/overlay2/d107d82812dc20a15e4909114e94499ee981db3a63a5dd8
7fb910749000-7fb9108de000 r-xp 00028000 00:31 38681179                   /var/lib/docker/overlay2/d107d82812dc20a15e4909114e94499ee981db3a63a5dd8
7fb9108de000-7fb910936000 r--p 001bd000 00:31 38681179                   /var/lib/docker/overlay2/d107d82812dc20a15e4909114e94499ee981db3a63a5dd8
7fb910936000-7fb910937000 ---p 00215000 00:31 38681179                   /var/lib/docker/overlay2/d107d82812dc20a15e4909114e94499ee981db3a63a5dd8
7fb910937000-7fb91093b000 r--p 00215000 00:31 38681179                   /var/lib/docker/overlay2/d107d82812dc20a15e4909114e94499ee981db3a63a5dd8
7fb91093b000-7fb91093d000 rw-p 00219000 00:31 38681179                   /var/lib/docker/overlay2/d107d82812dc20a15e4909114e94499ee981db3a63a5dd8
7fb91093d000-7fb91094a000 rw-p 00000000 00:00 0
7fb91094f000-7fb910951000 rw-p 00000000 00:00 0
7fb910951000-7fb910953000 r--p 00000000 00:31 38681013                   /var/lib/docker/overlay2/d107d82812dc20a15e4909114e94499ee981db3a63a5dd8
7fb910953000-7fb91097d000 r-xp 00002000 00:31 38681013                   /var/lib/docker/overlay2/d107d82812dc20a15e4909114e94499ee981db3a63a5dd8
7fb91097d000-7fb910988000 r--p 0002c000 00:31 38681013                   /var/lib/docker/overlay2/d107d82812dc20a15e4909114e94499ee981db3a63a5dd8
7fb910989000-7fb91098b000 r--p 00037000 00:31 38681013                   /var/lib/docker/overlay2/d107d82812dc20a15e4909114e94499ee981db3a63a5dd8
7fb91098b000-7fb91098d000 rw-p 00039000 00:31 38681013                   /var/lib/docker/overlay2/d107d82812dc20a15e4909114e94499ee981db3a63a5dd8
7ffd284a6000-7ffd284c7000 rw-p 00000000 00:00 0                          [stack]
7ffd28516000-7ffd2851a000 r--p 00000000 00:00 0                          [vvar]
7ffd2851a000-7ffd2851c000 r-xp 00000000 00:00 0                          [vdso]
ffffffffff600000-ffffffffff601000 --xp 00000000 00:00 0                  [vsyscall]
kan24@os:~/mem$
```

Tilsvarende utskrift for mem2 er nesten identisk, bortsett fra linjen hvor det står [heap] som viser at det for mem3 er lagt til et minnesegment på heap, det allokerte arrayet.

PowerShell

26)

26 **PowerShell-oneliner**

Hva gjør følgende PowerShell-kommando?

((Get-date) - (Get-Process dockerd).StartTime).TotalHours

Velg ett alternativ

- Gir antall timer alle docker containere på systemet har kjørt
- Gir antall timer dockerd-prosessen har kjørt ✓
- Gir antall CPU-timer dockerd-prosessen har brukt
- Starter opp dockerd-prosessen
- Viser hvilket tidspunkt dockerd-prosessen startet opp
- Lister alle docker-prosesser som kjører nå og som har blitt startet de siste 60 minuttene.

Riktig. 10 av 10 poeng. Prøv igjen

27)

27 **PowerShell vs Bash**

I PowerShell er det laget en rekke alias som gjør det enkelt for en som kjenner til Linux-kommandoer å bruke de samme kommandoene i PowerShell. Merk av hvilke Linux-kommandoer som tilsvarer hvilke PowerShell Cmdlets.

Finn de som passer sammen

	ls	mv	cp	rm	ps	pwd	kill	cat	cd
Remove-Item				✓					
Copy-Item			✓						
Get-Content								✓	
Get-ChildItem	✓								
Get-Process					✓				
Stop-Process							✓		
Set-Location									✓
Move-Item		✓							
Get-Location						✓			

Riktig. 10 av 10 poeng. Prøv igjen