UNIVERSITETET I BERGEN

KANDIDAT

307

PRØVE

INF113 0 Innføring i operativsystem

Emnekode	INF113
Vurderingsform	Skriftlig eksamen
Starttid	22.11.2024 08:00
Sluttid	22.11.2024 11:00
Sensurfrist	--
PDF opprettet	04.01.2025 15:01

Seksjon 1

Oppgave	Status	Poeng	Oppgavetype
i			Informasjon eller ressurser

Kort

Oppgave	Status	Poeng	Oppgavetype
1	Besvart	2/2	Tekstfelt
2	Riktig	2/2	Fyll inn tall
3	Riktig	2/2	Paring
4	Besvart	0/3	Tekstfelt
5	Delvis riktig	2.5/4	Sammensatt
6	Riktig	5/5	Nedtrekk

Middels

Oppgave	Status	Poeng	Oppgavetype
7	Besvart	2/3	Tekstfelt
8	Besvart	3/3	Tekstfelt
9	Besvart	3/3	Tekstfelt
10	Besvart	3/4	Tekstfelt
11	Besvart	2/3	Tekstfelt

Lang

Oppgave	Status	Poeng	Oppgavetype
12	Besvart	9/12	Langsvar

13	Besvart	7/12	Langsvar
14	Besvart	9/12	Langsvar

Poeng fra delmappe 1, 2, 3

Oppgave	Status	Poeng	Oppgavetype
15	Ubesvart	30/30	Muntlig

1 Hva er forskjellen mellom et program og en prosess?
Skriv ditt svar her (2-3 setninger)

Et program ligger på disk, men kjører ikke. En prosess er et programm som kjører.

Maks poeng: 2

2 En enkel OS bruker **base&bounds** til virtuell minne. Bounds inneholder slutt-adressen til minne-regionen. Alle tall nedover har basis 10.

Prosess A har base = 1200. bounds = 1400.

Hva er den fysiske adressen til følgende virtuelle adresser?
Skriv **-1** dersom det er en ugyldig adresse.

virtuell adresse	fysisk adresse
12	1212
14	1214
120	1320
1400	-1

Maks poeng: 2

3 Finn de begrepene som passer sammen:

	sector	block	page	page frame
virtual memory	○	○	● ✓	○
physical RAM	○	○	○	● ✓
hard disk	● ✓	○	○	○
file system	○	● ✓	○	○

Maks poeng: 2

4 (a) Hva er systemprogrammet **fsck**, og hva brukes den til?
Skriv ditt svar her (1-2 setninger)

Brukes til å utføre Filopperasjoner, scanne filer.

(b) Det har blitt sjelden nå at man trenger **fsck** som sysadmin. Hvorfor det?
Skriv ditt svar her (1-2 setninger)

For fsck trenger bare read permissions, og det har man som oftest som vanlig bruker.

Maks poeng: 3

5 Et **RAID-lagringssystem** slår sammen flere fysiske harddisker til én stor lagringsplass.

(a)
Når vi slår sammen 4 harddisker, med 1TB størrelse hver, i en RAID-4 (parity), hvor mye lagringsplass har vi faktisk tilgjengelig?

3 TB

(b)
Du skal sammenligne denne RAID-4-løsningen med en løsning som bruker bare én stor disk av tilsvarende størrelse. Velg de setningene som beskriver situasjonen med en **random** I/O workload:

Velg ett eller flere alternativer

- [x] RAID-4 har verre I/O read latency
- [ ] RAID-4 har verre I/O read throughput
- [x] RAID-4 har verre I/O write throughput
- [ ] RAID-4 har verre I/O write latency

(c)
Etter en diskfeil måtte vi erstatte disk 3 av 4 i et RAID-4-opplegg. Hva er innholdet til den manglende datablokken?

Disk 1	DIsk 2	Disk 3	Disk 4
0110	1001	1101	0010

Maks poeng: 4

6 Fem korte svar

(a)
Når et system har **swap space** tilgjengelig, så kan prosessene bruke

mer ✓ (like mye, mer, mindre) minne enn uten swap.

(b)
En typisk arbeidsflyt som et **shell**-program bruker til å kjøre andre programmer kalles for

fork - exec - wait ✓ (exec - fork - wait, fork - exec - wait, wait - fork - exec, exec - wait - fork, wait - exec - fork, fork - wait - exec)

(c)

context switch er navnet til prosedyren der man bytter fra en prosess til en annen ✓ (fra en prosess til en annen, fra virtuell til fysisk minne, fra RAID til ZFS, fra minne til swap)

(d)

C-kode ✓ (C-kode, Maskinkode, Assembly-kode) kan flyttes til en annen maskinarkitektur uten endringer.

(e)
Scheduling-policyen der man går videre til neste steg med jevnt avstand kalles for

Round Robin ✓ (First-In-First-Out, Shortest-Job-First, Least-Used-First, Round Robin)

Maks poeng: 5

7 Ada kjører dette C-programmet:

```
#include <stdio.h>
int main() {
  printf("Hello!\n");
  printf("My name is Ada.\n");
  printf("\nGood bye\n");
}
```

Hun kjører programmet under **strace**, og ser at de tre kall til printf() utløste bare **én write-systemkall**:

```
write(1, "Hello!\nMy name is Ada.\n\nGood bye"..., 33) = 33
```

Hva er fordelen med å slå sammen flere write-kall i én? Hva kalles denne proseduren?

Skriv ditt svar her (3-5 setninger)

Fordelen et at OSen ikke må avbryte/vente på prosessen for å gjøre IO hver gang det skal writes noe. Denne proseduren kalles buffering, hvor programmet lagret write-systemkallene i en buffer for å skrive det ut når prosessen ikke aktivt kjører på CPUen. Dette gir bedre preformance.

Maks poeng: 3

8 Data som er lagret på en disk kan undergå "bit flips" der en tilfeldig bit bytter fra 0 til 1 eller omvendt.

Hva er faren med dette? Hvordan kan en filsystem sikre seg mot dette?

Skriv ditt svar her (3-5 setninger)

Faren er at dataen, bitmapps, inodes, osv kan endre seg som fører til at filsystemet blir feil, uten at vi vil det. En måte man kan sikre seg mot dette er å bruke checksums for å finne ut om en bit har endret seg. Hvis du da bruker et RAID system med mirroring, eller parity (med checksum) for å finne ut hvilke bit som er feil og rette opp.

Maks poeng: 3

9 Hva er den store ulempen med **Segmentation** - metoden til adress-translation? Hvordan løses dette i **Paging?**

Skriv ditt svar her (3-5 setninger)

Segmentation gjør at det fysike minnet blir del opp i med data spredd litt over alt. Det kan føre til at du ikke får allokert en mengde minne, selv om du fysikt har plass for det, siden du ikke har sammenhengende minne. Paging fikser dette med å dele opp alt i like store deler (ofte 4kb) og lar en MMU slotte inn pagene hvor det er ledig i minnet, og holder kontroll på adressene i et page table. Dette fikser segmentation med hvis pagene er for store kan det lede til internal segmentation der delen mellom Stack og Heap står tomt.

Maks poeng: 3

10 En **atomisk** transaksjon er en transaksjon som enten fullføres i sin helhet, eller har ingen effekt.

Forklar kort hvordan en **log-basert filsystem** kan sikre at **I/O writes** til filsystemet skjer atomisk.

Skriv ditt svar her (3-5 setninger)

I et logbasert filsystem så overskriver du ikke dataen du erstatter, du bare forsetter å skrive den nye daten på "enden" av filsystemet og oppdaterer Inoden for å peke på de nye data blokkene. Hvis noe skjer med systemet og det blir avbrutt i en write, så kan du gjennopprette dataen siden du ikke overskrivde den. Treffer du enden av filsystemet så går du fra starten og overskiver den eldste ubrukte daten.

Maks poeng: 4

11 I en **inode** finnes det blant annet noen tidsstempler (created/modified/accessed). I **ext2** er tidsstemplene lagret som en 32-bit int, som teller sekunder siden 1970-01-01. Dette valget innebærer et problem som vil ramme flere og flere systemer som fortsatt bruker 32-bit int til tidspunkter. Beskriv kort dette problemet.

Skriv ditt svar her (3-5 setninger)

Problemet er litt som det 2000 års problemet, hvor alle trodde at datamaskiner ikke ville håntere tusenårs skiftet. Hvor når en 32-bit int ikke lenger har plass til datoene da max størrelsen blir 2^32 lagret som sekunder siden 1970-01-01 og om ikke noe gjøres så vil systemer over hele verden kræsje. Dette blir nok fikset i en OS update.

Maks poeng: 3

12 Forklar hvordan en **multi-level feedback queue** (MLFQ) håndterer følgende situasjoner i scheduling av prosesser:

a) Hvordan bestemmes **rekkefølgen** prosessene blir kjørt i?
b) Hvordan blir en **ny prosess** håndtert?
c) Når kan en prosess **flytte opp eller ned** i prioriteten?
d) Hvordan påvirker MLFQ-scheduling **turnaround** og **responstid**?

Skriv ditt svar her

a) I en multilevel feedback queue så har man flere køer, med forskjellige prioriteter, og som har en viss tid de lar hver prosess kjøre. Rekkefølgen blir at den køen med høyest prioritet blir kjørt helt ferdig før den går videre til å kjøre prosesser i køen med nest høyest prioritet, osv. Hver kø har også et eget scheduling scheme som bestemmer hvilke prosess internt som skal kjøre.

b) En ny prosess ender alltid opp i den køen med høyest prioritet først, om ikke noe annet et programmert inn. Det er også forskjell om OSen avbryter den kjørende prosessen, fra en lavere prioritet kø, og bytter til den nye eller om den lar den kjøre ferdig sin tid før den bytter til den nye prosessen i den høyeste prioriterte køen.

c) Når en prosess har brukt opp kjøretiden sin i en kø uten å bli ferdig, så blir den flyttet ett hakk ned til en kø med lavere prioritet, men ofte med mer kjøretid.

d) En MLFQ gir god turnaround til da alle de raske prosessene blir ferdig i den første køen og de som trenger mer tid blir flyttet nedover og utført til slutt. Responstid fra prosess til prosess blir at korte prosesser får veldig god responstid, mens lange prosesser vil føles tregere da de må vente til de andre køene med høyere prioritet er ferdig.

For å ta et eksempel så kan vi se for oss at vi har en MLFQ med 3 køer 1, 2 og 3. Kø 1 har høyest prioritet og lar prosesser kjøre i 1ms, kø 2 har nest høyest prioritet og lar prosesser kjøre i 3ms og kø 3 har lavest prioritet og lar prosesser kjøre til de er ferdig (går utifra at det også er limited direct execution innebygd).

Vi får nå 5 prosesser a,b,c,d,e,f inn og alle går i kø1. Kø1 kjører a i 1ms, og flytter den til kø 2, så b i 1ms og flytter til kø2. Prosess c,d,e,f trengte bare 0.5ms og er ferdig. I kø2 så kjører a i 3ms før den blir flyttet til kø3 og b i 3ms før den blir flyttet til kø3. I kø3 så har vi Round Robbin så den kjører hver prosess i 10ms og roterer prosess a og b innad i seg selv til de er ferdig.

Systemet vårt vil følses responsivt da ofte prosesser som med UI ofte trenger kort CPU tid, men prosesser som krever lang tid må kjøre i "bakgrunnen" da ingenting annet skjer.

Ord: 417

Maks poeng: 12

13 Vi har en virtuell adresserom med **7 bits**, tilsvarende 128 virtuelle adresser **fra 0x00 til 0x7f**.

En **multi-level page table** brukes til oversettelsen:
De første 2 bits av en virtuell adresse blir brukt til en page directory,
de neste 2 til en page table. De siste 3 bits er offset.

De følgende page frames er i bruk, spørsmålene finnes etter denne oversikten.

Page frame 0
Page directory

index	page frame
00	3
01	-/-
10	-/-
11	7

[...]

Page frame 3
Page table

index	page frame
00	-/-
01	8
10	12
11	-/-

[...]

Page frame 7
Page table

index	page frame
00	-/-
01	10
10	11
11	13

Page frame 8
Data Page

address	content
000	'H'
001	'e'
010	'l'
011	'l'
100	'o'
101	','
110	-space-
111	-space

Page frame 9
Data Page

address	content
000	'W'
001	'o'
010	'r'
011	'l'
100	'd'
101	'!'
110	-null-
111	-null

Page frame 10
Data Page

address	content
000	-null-
001	-null-
010	-null-
011	-null-
100	0x03
101	0x04
110	-null-
111	-null

Page frame 11
Data Page

address	content
000	0x02
001	0x07
010	-null-
011	-null-
100	0x05
101	0x0a
110	-null-
111	-null

Page frame 12
Data Page

address	content
000	'A'
001	'l'
010	'i'
011	'c'
100	'e'
101	'!'
110	-null-
111	-null

Page frame 13
Data Page

address	content
000	0x03
001	0x04

Address	Content
000	0x03
001	0x0a
010	0x05
011	0x0a
100	0x03
101	0x05
110	0x02
111	0x07

[...]

Spørsmål

(a) Hvor stor er én page i bytes?

(b) Hvor mange pages trengs til prosessens hele virtuelle adresserom?

(c) Utifra situasjonen over, kjører vi disse C-programlinjene (en int er 4 byte, lagret little-endian)

```
int * a = 0x74;
printf("%d\n", *a);
```

Hva er det som skrives ut? Inkluder stegene

(d) Utifra situasjonen over, kjører vi disse C-programlinjene

```
char * greeting = 0x08;
printf("%s\n", greeting);
```

Hva er det som skrives ut? Inkluder stegene

Skriv ditt svar her

a) i en MLFQ er standaren 4kb, men her så ser vi at vi har 8 plasser i en page frame og at vi har plass til en char i hver content så blir det 8 * SizeOf(char) i bytes. 1 byte er 4 bits.

b) Siden det er OSen/MMU som holder kontrol på Page Directory og Page Table så kan vi se på page directory at prosessen bruker talbes 8 - 13 så 6 pages er nok for programmet. da vi ser at stacken her ligger nederst og peker opp på heapen.

c) Siden 1111111 base 2 = 128, og 0x7f - 0x74 = 11 så kan vi trekke fra (11) 1011 base 2 fra 1111111 for å få adressen vi trenger, da ender vi opp med 1110100, de

to første er directory, 11 = dir 7, de to neste er frame, 10 = frame 11, og 100 i framen blir 0x05. Siden vi da printer %d og *a så printer vi det som ligger i adressen a peker på i decimal. Printer "5"

d) 8 i binær med 7 bit i lengde blir 0001000. Så følger vi samme prosess, 00 = dir 3, 01 = frame 8, 000 = "H". Så når vi printer greeting så starter vi på "H" og siden vi bruker %s så fortsetter vi til vi kommer til en end char "\0" eller null. så vi printer "Hello, World!"

Ord: 235

Maks poeng: 12

14 Vi ser på et enkelt filsystem. Root-mappen i filsystemet (dvs. "/") tilsvarer inode 2.

Filsystemet på disken inneholder:

Bitmap for data

block	occupied?
1-5	0 0 0 0 1
6-10	0 1 1 0 1
11-15	1 1 0 0 0

Bitmap for inodes

inode	occupied?
1-5	1 1 1 1 1
6-10	1 0 0 0 0

Inodes

Nr.	Type	reference count	data blocks
2	dir	3	5
3	file	1	7, 8
4	file	1	10
5	dir	2	11
6	file	1	12

Data blocks:

Data block 5
2 . // 2 .. // 3 worlds.txt // 5 my_data // 4 alice.txt //

...

Data block 7
No one would have believed in the last years of the nineteenth century that
this world was being watched keenly and closely by intelligences greater tha

Data block 8
n man's and yet as mortal as his own; that as men busied themselves about
their various concerns they were scrutinised and studied.

...

Data block 10
Alice was beginning to get very tired of sitting by her sister on the bank.

Data block 11
5 . // 2 .. // 6 bergen_rain.csv //

Data block 12
jan,feb,mar,apr,may,jun,jul,aug,sep,oct,nov,dec\n255,210,200,140,110,135,1
55,210,245,270,270,290\n

Oppgaver

OBS!
Hver deloppgave tar utgangspunktet i situasjonen ovenfor.
Effektene skal ikke kombineres!
OBS!

(a)
Ved et uhell inneholder "Bitmap for data" en feil. Istedenfor

6-10	0 1 1 0 1

finnes det

6-10	0 1 0 0 1

Er filsystemet inkonsistent? Hvilken fil er påvirket? Kan dette problemet oppdages automatisk? Hvordan ser dette ut fra brukerens side?

(b)
Når brukeren skrev ut "my_data/bergen_rain.csv" krasjet PCen, og blokk 11 ser sånn ut:

Data block 11
5 . // 2 ..

Er filsystemet inkonsistent? Kan dette problemet oppdages automatisk? Hvordan ser dette ut fra brukerens side?

(c)
Når man kjører en automatisk konsistens-sjekk, hvilke(t) spørsmål trenger svar fra brukeren for å fikse problemet (a)?

(d)
Når man kjører en automatisk konsistens-sjekk, hvilke(t) spørsmål trenger svar fra brukeren for å fikse problemet (b)?

Skriv ditt svar her

a) Ja, filsystemet er inkonsistent da systemet vil tro at blokk nr8 ikke er i bruk eller tom, selv om den er det. Filen som blir påvirket er filen "worlds.txt". Problemet kan oppdages automatisk da i Inoden til wordls.txt så står det at filen bruker block 7 og 8, siden dette ikke matcher bitmap for data så kan systemet se at det er en feil og rette opp i det. For brukeren så vil det se ut som at filsystemet

har mer plass en det egentlig har. Når det kommer til å lese filen så vil ikke brukeren se forskjell da det er Inoden for den filen som styrer lesing.

b) Ja, filsystemet er inkonsistent da bitmap for data og bitmap for inodes sier at det skal være data på block 6, men i når systemet går til Datablock 11 så er det ikke en fil på block 6 og da kræsjer systemet. Fra brukeren sin side så er det ut som et helt vanlig filsystem, hvor "bergen_rain.csv" finnes men når brukeren prøver å åpne filen så kommer det et system kræsj.

c) Siden ikke systemet kan vite om det er bitmappene som er feil eller inodene så trenger det hjelp fra brukeren. I spm a, så ville systemet trengt å vite om block 8 var del av filen eller ikke. Svarer brukeren ja så vil systemet oppdatere bitmap for data og justere laginsplass. svarer brukeren nei så blir det Inoden som blir oppdatert og brukeren vil ikke få opp det som står i block 8 når den leser av filen.

d) Siden ikke systemet kan vite hva block 11 refererer til, i Inoden står det at det skal være to referanser men det ligger bare en det. så trenger systemet hjelp fra brukeren til å finne filen som egentlig skal bli refferet til i block 11. Hvis brukeren viser til bergen_rain.csv så fikser systemet det med å legge til referansen. Om brukeren sier at det ikke skal være noe bergen_rain.csv (den er slettet) så oppdaterer systemet Inoden og bitmappene.

Ord: 340

Maks poeng: 12

15 Du trenger ikke gjøre noe her.

Vi bruker denne delen for å registrere poengsummen fra delmappene.

Maks poeng: 30