Høgskulen på Vestlandet

Institutt for datateknologi, elektroteknologi og realfag

Eksamen i:	DAT103 – Datamaskiner og operativsystemer
Eksamensdato:	10 januar 2022
Tid for eksamen:	9:00 – 13:00
Hjelpemidler:	Både skrevne og trykte hjelpemidler, samt nettressurser. Ingen kommunikasjon med andre personer (Dvs., oppgavene skal løses individuelt. Samarbeid med andre anses som fusk.)
Målform:	Norsk bokmål

Eksamensoppgavene består av 7 sider, der denne forsiden ikke er inkludert.
Du kan svare på enten engelsk eller norsk.

!!! VELDIG VIKTIG !!! Les dette før du fortsetter.

- Under eksamen kan du kontakte oss via Zoom, mellom 9:30 – 13:00:
  Meeting ID: 658 1074 1912
  Passcode: h21-dat103
  Du må vente i et virtuelt venterom til vi slipper deg inn i møterommet for å stille spørsmål.
- Oppgavene 3, 4.2 og 6.1 er **spesifikke** for kandidatnummeret ditt på denne DAT103-eksamenen.
- Disse oppgavene vil bli rettet i henhold til kandidatnummeret ditt.
- **Halvparten av poengene** vil bli trukket dersom du ikke svarer på disse oppgavene i henhold til ditt kandidatnummer.
- I Oppgavene 3, 4.2 og 6.1 bruker vi
  Even for å indikere spørsmål og verdier for **like kandidatnummer**
  Odd for å indikere spørsmål og verdier for **ulike kandidatnummer**
- Hvis du er i tvil bør du ta kontakt med oss.

Lykke til!
Dag og Violet

Oppgave 1 – Flervalgsoppgave (28%)

For hver av flervalgsoppgavene nedenfor, velg ikke mer enn et av alternativene

1.1 Hvilken av de følgende er **riktig**?

(a) Programtelleren inneholder instruksjonen prosessoren utfører akkurat nå
(b) Programtelleren inneholder adressen for instruksjonen prosessoren utfører akkurat nå
(c) Programtelleren er et register
(d) Alle de ovennevnte
(e) Ingen av de ovennevnte

1.2 Hvilken av det følgende er **feil** om "pipelining"?

(a) "Pipelining" tillater utførelse av flere instruksjoner på samme tid på en kjerne
(b) Stadiene for en instruksjonssyklus i "pipeline" krever ulik tid for prosessering
(c) Av og til er det ikke den instruksjonen som er hentet på forhånd (prefetched) som er den neste som blir utført
(d) Alle de ovennevnte
(e) Ingen av de ovennevnte

1.3 En sidefeil (page fault) hender når ...

(a) En forespurt side (page) er i "backing store"
(b) En forespurt side er funnet i translation look-aside buffer (TLB)
(c) En forespurt side er i minnet
(d) Alle de ovennevnte
(e) Ingen av de ovennevnte

1.4 Ta utgangspunkt i en prosessor som trenger 5 ns for å få tilgang til cache-minne og 200 ns for å få tilgang til hovedminnet. Anta at cache-treffrate (cache hit ratio) er 95%, hva er da gjennomsnittlig tilgangstid for prosessoren?

(a) 5 ns
(b) 10 ns
(c) 15 ns
(d) 20 ns
(e) 100 ns
(f) Ingen av de ovennevnte

1.5 Ta utganspunkt i instruksjonen `mov edx [hello]`. Hvor mange operander har denne instruksjonen?

(a) 3
(b) 2
(c) 1
(d) 0
(e) Ingen av de ovennevnte

1.6 Ta utgangspunkt i en instruksjon med to operander O_1, O_2. Hvis O_1 bruker *PC-relativ* adresseringsmodus, O_2 bruker *indirekt* adresseringsmodus, hvor mange minnetilganger trengs i alt for å hente O_1 og O_2?

(a) 4
(b) 3
(c) 2
(d) 1
(e) 0
(f) Ingen av de ovennevnte

1.7 La ℓ være en lås, og `x` en delt `int` variabel initialisert til `0`. Ta utgangspunkt i to prosesser P_1 og P_2 som samtidig kjører de følgende kodesekvensene:

```
1 // Run by P1 //
2 x--;
```

```
3 // Run by P2 //
4 acquire(ℓ);
5 x--;
6 release(ℓ);
```

Hvilken av de følgende er **feil**?

(a) Verdien av `x` er -1 når P_1 og P_2 er ferdige
(b) Verdien av `x` er -2 når P_1 og P_2 er ferdige
(c) Det er en "race condition"
(d) Alle de ovennevnte
(e) Ingen av de ovennevnte

1.8 Hvilken av de følgende er en funksjon for en "dispatcher"?

(a) Å kontekst-bytte prosesser
(b) Å gi kontroll over CPUen til prosessen valgt ut av korttids-planleggeren
(c) Å hoppe til riktig posisjon i bruker-programmet for å starte opp igjen det programmet
(d) Alle de ovannevnte
(e) Ingen av de ovannevnte

1.9 Hvilken av de følgande er **feil**?

(a) Sideinndeling (paging) tillater minnet allokert til en prosess å være usammanhengende
(b) Sideinndeling løser problemet med intern fragmentering
(c) Sideinndeling løser problemet med ekstern fragmentering
(d) Sideinndeling unngår problemet med thrashing
(e) Alle de ovennevnte
(f) Ingen av de ovennevnte

1.10 Hvilken av de følgende planleggingsalgoritmene har et utsultingsproblem?

(a) Korteste jobb først
(b) "Multilevel Queue Scheduling"
(c) Korteste-tid-igjen-først
(d) Alle de ovennevnte
(e) Ingen av de ovennevnte

1.11 Hvilken av de følgande er **riktig**?

(a) PCBer blir egentlig ikke brukt av operativsystemet, men er bare vedlikeholdt for å kunne tilby brukeren statistikker om prosessene
(b) PCBer er datastrukturene operativsystemet bruker for å implementere multi-programmering, kontekst-bytte, og også prosess-"forks"
(c) En PCB inneholder ikke en programteller
(d) En PCB inneholder en datamaskins tjenernavn
(e) Alle de ovennevnte
(f) Ingen av de ovennevnte

1.12 Hvilken av de følgende er **feil**?

(a) `fork()`-systemkallet er brukt for å opprette en ny prosess på en UNIX datamaskin
(b) En ny barne-prosess som blir opprettet ved bruke `fork()`-systemkallet er i utgangspunktet identisk med foreldre-prosessen
(c) En prosess kan bruke retur-verdien fra `fork()`-systemkallet for å avgjøre om den er foreldre- eller barneprosessen
(d) `exec()`-systemkallet kan brukes til å bytte ut programinnholdet for en barneprosess
(e) Alle de ovennevnte
(f) Ingen av de ovennevnte

1.13 Hvilken av de følgende er **riktig**?

(a) Å opprette en ny prosess krever nesten ingen ressurser, og det er derfor ingen grunn til å ikke opprette prosesser
(b) En prosess og en tråd er bare forskjellige navn på nøyaktig det samme
(c) En tråd blir også kallt en lett prosess
(d) Det er bare mulig å opprette tråder på et multi-CPU/kjerne-system
(e) Alle de ovennevnte
(f) Ingen av de ovennevnte

1.14 Hvilken av de følgende er **feil**?

(a) Et systemkall blir brukt av et brukerprogram for å be operativsystemet om å gjøre en handling for seg
(b) Et systemkall kan gjøre en handling som brukerprogrammet ikke har tilgang til å gjøre selv
(c) Som regel blir systemkall brukt ved hjelp av høy-nivå systembibliotek
(d) Et systemkall blir også kallt et programvareavbrudd (software interrupt)
(e) Ingen av de ovennevnte

Oppgave 2 – Diverse (13%)

2.1 La ℓ_1 og ℓ_2 være låser, og `x` en delt `int` variabel initialisert til `1`. Ta utgangspunkt i tre prosesser P_1, P_2 og P_3 som samtidig kjører de følgende kodesekvensene:

```
1  // Kjørt av P1 //
2  acquire(ℓ1);
3  x++;
4  release(ℓ1);
```

```
5  // Kjørt av P2 //
6  acquire(ℓ2);
7  x--;
8  release(ℓ2);
```

```
9  // Kjørt av P3 //
10 acquire(ℓ1);
11 x++;
12 release(ℓ1);
```

List opp alle mulige verdier som `x` kunne ha når alle tre prosessene er ferdige.

2.2 Når en applikasjon har behov for å utføre flere oppgaver samtidig (i parallell), så kan dette gjøres enten ved å starte flere prosesser eller tråder. Hva er forskjellene mellom prosesser og tråder? Gi eksempel på "typisk" bruk for begge.

2.3 Maskivarekomponenter som CPU, minne, I/O-kontroller, etc. kan kommunisere enten via en delt buss eller via en punkt-til-punkt–forbindelse. Hva er forskjellene mellom disse to typene forbindelser, og hva er deres relative fordeler/ulemper for hver? Punkt-til-punkt–forbindelser blir stadig mer populære. Hvorfor er dette?

2.4 Hva er forskjellene mellom en CPU og en kjerne? Er det noen mulige forskjeller med tanke på ytelse om to kjerner er fysisk plasserte i den samme brikken, eller om de er plasser i to ulike brikker? Hvorfor og hvorfor ikke?

2.5 Tidlige datasystem hadde ikke mulighet til å lagre program (prosesseringsinstruksjoner) som "data". I steden var de "hard-kodet" med ledninger. Hvorfor er det viktig å være i stand til å lagre program som data? Hva er konsekvensene om dette ikke er mulig?

Oppgave 3 – Sideinndeling (10%)

Ta utgangspunkt i at 1 KB er 1024 byter, og anta at sidestørrelsen er

Even ***kandidatnummer*****:** 4 KB
Odd ***kandidatnummer*****:** 8 KB

3.1 Skriv ned, med desimaltall, *sidenummeret* og "*offset*" for adressen `32543`.

3.2 Ta utgangspunkt i et datasystem som bruker 64-biter logisk adresse og har fysisk minne som er delt inn i F rammer.

(a) Hva er det logiske adresserommet?
(b) Hva er størrelsen på hver ramme i det fysiske minnet?
(c) Om systemet har en *invertert* sidetabell, hvor mange plasser har tabellen?

Oppgave 4 – CPU-planlegging (16%)

4.1 Generelt vil en prosess i et system være i en av disse fem tilstandene: `ny`, `klar`, `kjørende`, `ventende`, `avsluttet`. En *tilstandstransisjon* for en prosess betyr at en prosess går fra en tilstand til en annen, f.eks., en tilstandstransisjon `klar` $\rightarrow$ `kjørende` betyr at en prosess går fra `klar`-tilstand til `kjørende`-tilstand.

Skriv ned tilstandstransisjon(ene) hvor planleggingsavgjørelse (scheduling decision) vil skje om operativsystemet bruker *avbrytbar* (preemptive) planleggingsalgoritme?

4.2 Ta utgangspunkt i følgende tabell for utbruddstid (burst time) for prosessene P_1, P_2, P_3 og P_4 som kommer på *tid* $= 0$:

Prosess	Utbruddstid
P_1	4
P_2	7
P_3	6
P_4	2

Anta at rekkefølgen de kommer er

Even ***kandidatnummer*****:** P_3, P_2, P_1, P_4

Odd ***kandidatnummer*****:** P_4, P_1, P_3, P_2

Om *Først komen først handtert* planleggingsalgoritme blir brukt, hva er den gjennomsnittlige behandlingstiden (turnaround time)? Bruk et Gantt-diagram for å rettferdiggjøre svaret ditt.

4.3 Ta utgangspunkt den følgende tabellen for prosesseringstid og periode for to *periodiske* prosesser P_1 og P_2:

Proeess	Prosesseringstid	Periode
P_1	30	80
P_2	40	60

Fristen for fullførelse av hver prosess er begynnelsen av prosessen sin neste periode. For eksempel, fristen for fullførelse av P_1 er på tidene $80, 160, 240, \ldots$, mens fristen for fullførelse av P_2 er på tidene $60, 120, 180, 240, \ldots$. Anta at planlegging bare blir utført når en prosess kommer eller blir avsluttet.

(a) Anta at P_2 har høyere prioritet enn P_1. Bruk *Rate-monotonic Scheduling* for å planlegge de to prosessene. Ved *tid* $= 280$, kan de to prosessene tilfredsstille alle sine frister?

(b) Bruk *Tidligste tidsfrist først* for å planlegge de to prosessene. Ved *tid* $= 280$, kan de to prosessene tilfredsstille alle sine frister?

Tegn et *Gantt-diagram* for utførelsene av hver av planleggingsalgoritmene for å rettferdiggjøre svaret ditt. Tidslinjen for hvert av Gantt-diagrammene skal holde frem til *tid* $= 280$, eller til tidspunktet hvor en prosess ikke kan tilfredsstille sin frist, dvs., om en prosess ikke kan tilfredsstille sin frist ved *tid* $= 150$, da skal tidslinjen for Gantt-diagrammet være frem til 150.

Oppgave 5 – Minnetilgangstid (7%)

5.1 Ta utgangspunkt i en side-maskinvare (paging hardware) med et translation look-aside buffer (TLB). Anta at sidetabellen og *alle sidene* til en prosess-forespørsel er i det fysiske minnet. Anta at prosent-andelen av gangene som et sidenummer av interesse er funnet i TLB[1] er 85%, hvert TLB-oppslag tar 1 ns, og hver minnetilgang tar 200 ns. Hva er den effektive tilgangstiden (EAT)?

5.2 Ta utgangspunkt i en prosess som har sin side-tabell og **noen** av sine forespurte sider i fysisk minne. Anta at prosessen forsøker å få tilgang til en side hvor sidenummeret ikke er funnet i TLB, og en *sidefeil* (page fault) hender. Hvor mange "page transfers", dvs., å flytte sider inn i (eller ut av) "backing store", er nødvendig for å hente denne siden dersom

(a) det er ubrukte rammer hovedminnet?

(b) det er ikke ubrukte rammer, men alle rammene er rene?

[1] Dette blir også kallt "TLB hit ratio"

Oppgave 6 – Cache **(13%)**

6.1 Ta utgangspunkt i B_0, B_2, B_4, B_6 og B_8 er mappet til C_0, og B_1, B_3, B_5, B_7 og B_9 er mappet til C_1. Anta at de følgende er i cachen:

Even ***kandidatnummer*****:** B_3 og B_8
Odd ***kandidatnummer*****:** B_2 og B_9

hvor B_i refererer til den i-ende blokken i hovedminnet og C_j refererer til den j-ende cachelinjen. Hva er antallet cache-feil (cache misses) om B_5, B_2, B_8, B_9, B_8 blir aksessert i den viste rekkefølgen? Grunngi svaret ditt ved å identifisere hvilke aksesser som førte til cache-feil.

6.2 En enkelt-trådet applikasjon har behov for å prosessere en større mengde data. Dataene består av verdier som skal prosesseres i flere prosesserings-operasjoner (steg), på samme måte, og uavhengig av andre data-verdier. Dette betyr at prosesseringen kan bli utført på to måter (tilnærminger):

- Samme *operasjon* først: først utfør samme (enkle) operasjon på alle vardier, så gå videre til den neste (enkle) operasjonen.
- Samme *verdi* først: først utfør alle operasjoner på den samme (enkle) verdien, så gå vidare til den neste (enkle) verdien.

(Instruksjoner som kan fungere på vektorer av verdier er ikke tilgjengelig.) For å avgjøre hvilken tilnærming som er mest effektiv, har begge tilnærmingene blitt implementert og testet. Testene viser at den siste metoden er raskeste. Basert på dine kunnskapar om CPU cacher, forklar hvordan dette kan skje.

6.3 Underveis på testene nevnt ovenfor, blir det også lagt merke til en betydelig økelse i hastigheten på en CPU som har dobbelt så mye L1 cache som en annen ellers identisk CPU. Hva er mulige grunner for dette? Er det noe du kan forsøke for å også forbedre ytelsen for applikasjonen på CPUen med mindre L1 cache (uten å endre maskinvaren)?

6.4 Mellom CPUen og hovedminnet (RAM), er det som regel tre lag (L1, L2, L3) med cache minne. Dette minnet er minkende raskt og økende stort. Forklar grunnene for dette, og hvordan disse lagene samhandler. Hvorfor er det så "komplisert" — hvorfor ikke bare et enkelt lag med cache?

Oppgave 7 – Prosess-synkronisering (13%)

7.1 En mediaavspiller-applikasjon har to tråder. Den ene (nedlasteren) tråden vil laste ned media fra en nett-tjener, og plassere det på et lokalt buffer med 10 plasser. Den andre (dekoderen) tråden vil lese data fra dette bufferet og dekode det. Tabellen `buffer` er plassert på delt minne for å utveksle data, og en normal variabel `bufferStatus`, også i delt minne, blir brukt for å indikere statusen for dette bufferet. Det er initialisert til 0.

```
1  // Kode brukt av nedlasteren //
2  writePos=0;
3  while(true){
4    while(bufferStatus<10){
5      DownloadToBuffer(buffer[writePos]);
6      writePos++;
7      writePos=writePos%10;
8      bufferStatus++;
9    }
10 }
```

Listing 1

```
1  // Kode brukt av dekoderen //
2  readPos=0;
3  while(true){
4    while(bufferStatus>0){
5      DecodeFromBuffer(buffer[readPos]);
6      readPos++;
7      readPos=readPos%10;
8      bufferStatus--;
9    }
10 }
```

Listing 2

Det meste av tiden fungerer denne mediaavspilleren greit. Men, i skjeldne tilfeller krasjer den med en feilmelding som indikerer at en av trådene forsøker å få tilgang til feil plass i buffer-tabellen. Basert på dine kunnskaper om prosess-synkronisering, forklar hvordan dette kan skje.

7.2 Modifiser koden i Listings 1 og 2 ovenfor slik at den bruker semaforar for å synkronisere trådene. Løsningen skal tilfredsstille alle krav til "mutual exclusion", fremgang og "bounded waiting".

```
1  wait(int *semaphore){
2    while(*semaphore <= 0) {} ; // "busy waiting" //
3    *semaphore--;
4  }
5
6  signal(int *semaphore){
7    *semaphore++;
8  }
```

7.3 Den opprinnelige løsningen i Listings 1 og 2 bruker "busy waiting". Hva er ulempene med dette? Hvorfor er det av og til fortsatt den bedre løsningen? Forklar en fremgangsmåte for å unngå "busy waiting".

7.4 Ta utgangspunkt i følgende fremgangsmåte brukt av to prosesser for å synkronisere tilgang til delte data:

```
1  // Kode brukt av P1 //
2  while(p!=1) {} ; // "busy waiting" //
3  // kritisk seksjon //
4  p=2;
5  // ikke-kritisk seksjon //
```

```
6  // Kode brukt av P2 //
7  while(p!=2){} ; // "Busy waiting" //
8  // kritisk seksjon //
9  p=1;
10 // ikke-kritisk seksjon //
```

(a) Garanterer denne fremgangsmåten fremgang (progress) (ja/nei)?

(b) Garanterer denne fremgangsmåten gjensidig eksklusivitet (mutual exclusion) (ja/nei)?