# **Oppgave 1 ­ Prosesser**

a) Forklar hva en prosess er og hvordan prosesser gjør det mulige å kjøre flere

programmer samtidig på en datamaskin.

b) Kjøring av flere prosesser samtidig fører til hyppige kontekstskifter? Hva er

et kontekstskifte? Beskriv i detalj hva som skjer i et kontekstskifte?

c) En prosess skal underveis i kjøringen utføre en bestemt oppgave. Dette kan

gjøres på to forskjellige måter: i) Ved å gjøre et vanlig funksjonskall eller ii) ved

å starte en ny trådprosess som gjør det aktuelle funksjonskallet. Forklar

forskjellen i disse to måtene å gjøre det på.

**Maks poeng: 15**

# **Oppgave 2 ­ Kommunikasjon mellom prosesser**

a) Hvorfor er det viktig med kommunikasjon mellom prosesser?

For at en skal kunne lage effektive programsystemer må prosesser kunne kommunisere med hverandre. F.eks. utveksle data eller synkroniseres slik at en prosess ikke kjører uten at de nødvendige operasjonene er utført av en annen prosess.

b) Kommunikasjon mellom prosesser har blitt enklere ved bruk av trådprosesser. Hvorfor?

c) Ved bruk av semaforer er det viktig at kallene wait og signal kommer i en

bestemt rekkefølge. La oss anta en binær semafor mutex som benyttes til

gjensidig utelukkelse slik som vist nedenfor. Mutex har startverdien 1. I dette

eksemplet er det en feil. Hva er feil, og hva blir konsekvensene? Hva er rett

programkode i dette tilfellet?

wait(mutex);

adgang til felles dataområde

wait(mutex);

I dette tilfelle vil prosessen lese at mutex-verdien er 1, som betyr at prosessen kan fortsette i det felles dataområdet. Deretter endrer prosessen mutex-verdien til 0 for å sikre at ingen andre prosesser hopper inn i dataområdet. Etter det vil den lese av at verdien er 0, og vil stille seg i semaforkø. Dette gjør at ingen andre heller får tilgang til dataområdet siden verdien aldri blir endret tilbake til 1. Rett programkode ville i dette tilfelle vært :

Wait(mutex);

Adgang til felles dataområdet.

Signal(mutex);

Her får prosessen sagt fra seg det felles dataområdet slik at andre prosesser slipper til.

d) Ta utgangspunkt i operasjonene wait() og signal() og forklar kort hvordan vi

kan sette opp en mekanisme for gjensidig utelukkelse og en mekanisme for å

synkronisering.

Om vi tar utgangspunkt i et eksempel der vi har to prosessorer, en konsument (henter data) og en produsent (genererer data). Disse to prosessene skriver og henter fra en buffer. Når en prosess er å henter/skriver til bufferen er det viktig at ikke den andre gjør det samtidig. For å oppnå gjensidig utelukkelse bruker vi en binær semafor mutex. For å passe på at ikke konsumenten henter data fra en tom buffer eller at en produsent ikke produserer mer data til en full buffer må vi bruke synkronisering. Vi har dermed to tellende semaforer: opptatt og ledig. Disse viser antall ledige og opptatte plasser i bufferen.

For å illustrere dette med pseudo-kode vil det se slik ut:

Prosess Konsument

Begin

Repeat

Wait(opptatt);

Wait(mutex);

Hent data;

Signal(mutex);

Signal(ledig);

Konsumer data;

Forever

End

Prosess produsent

Begin

Repeat

Produser data;

Wait(ledig);

Wait(mutex);

Legg inn data;

Signal(mutex);

Signal(opptatt);

Forever

End.

e) Semafor­operasjonene wait() og signal() må være atomiske for å fungere

korrekt og for å sikre gjensidig utelukkelse. Hva betyr det at en operasjon er

atomisk og hvordan sikres det at operasjonene wait og signal er atomiske?

At en operasjon er atomisk betyr at en instruksjon kan skje uten avbrudd, altså at ingen andre prosesser får adgang til dataen før instruksjonen er ferdig. Wait og signal vil begge disable interrupts noe som gjør dem til atomiske operasjoner. Vanligvis ville vi ikke ha brukt disable interrupts siden ingen andre prosesser får kjøre på CPU-en da avbrudd ikke vil virke siden avbruddssystemet er avslått. Om det er feil i programkoden inne i den kritiske regionen vil det medføre at avbruddssystemet ikke vil bli slått på igjen. Men siden semaforer er garantert å fullføre instruksjonene vil ikke dette være et problem, det foregår dessuten i meget kontrollerte former.

**Maks poeng: 15**

# **Oppgave 3 ­ Minneadministrasjon**

a) Baseregisteret er viktig innen hukommelsesadministrasjon. Forklar hva

base­registeret brukes til og hvorfor det er viktig innen

hukommelsesadministrasjon? Forklar også hvordan man kan betrakte en

sidetabell som en samling av flere baseregistre.x

b) Når det gjelder sideutskiftingsalgoritmer er det generelt mer ønskelig å skifte

ut en side det ikke er skrevet til enn en side det er skrevet til. Forklar dette. Si

også noe om hva som menes med at en side er skrevet til.

c) Et sidedelt system bruker virtuelle adresser på 20 bit. Sidestørrelsen er 8

Kbyte. Anta fysisk minne (RAM) på 512 Kbyte. (Litt hjelp: 2 opphøyd i 20 er lik

1024K). Virtuell adresse består av to felt, sidenummer og offset. Hvor mange

bit er det for hver av disse to feltene? Angi også hvor stort virtuelt minne er.

Begge svarene skal begrunnes.

d) Hvor mange innslag (linjer) er det i sidetabellen for systemet gitt i pkt c)

ovenfor?

220 = 1024 Kbyte

En virtuell adresse er på 1024 Kbyte

Vi vet at sidestørrelsen er 8Kbyte så for å finne hvor mange innslag det er i sidetabellen deler vi den virtuelle adressestørrelsen på sidestørrelsen:

1024 Kbyte / 8 Kbyte = 128

Dvs at vi har 128 innslag i sidetabellen gitt i oppgave c).

e) I sidedelte systemer opereres det med begrepet sidefeil. Hva er en

sidefeil?

En sidefeil er hver gang vi må hente data som ikke ligger i minnet på disk.

f) Trådprosesser har en viktig egenskap når det gjelder minnebruk. Hva er det?

Forklar.

**Maks poeng: 20**

# **Opppgave 4 ­ Parallellitet på instruksjonsnivå**

*Pipeline, superskalar prosessor og dynamisk utføring er eksempler på*

*mekanismer for parallellitet på instruksjonsnivå.*

a) Hva mener vi med parallellitet på instruksjonsnivå?

Med parallellitet på instruksjonsnivå mener vi at en og samme prosessor kan utføre flere instruksjoner parallelt.

b) Skisser hvordan de tre mekanismene pipeline, superskalar prosessor og

dynamisk utføring fungerer (5­10 linjer pr begrep)

**Pipeline:**

I en pipeline deles hentesyklusen opp i flere trinn slik at flere instruksjoner kan utføres parallelt. Dette øker effektiviteten, men det kan oppstå sitasjoner der instruksjoner som kjøres parallelt trenger samme ressurs, eller de er dataavhengige, slik at den ene instruksjonen er avhengig av resultatet til den andre instruksjonen. I slike tilfeller må CPU-en ta spesielle hensyn for rett utførelse. Dette er det vi kaller hasarder. Vi har tre ulike hasarder: strukturhasard, datahasard, kontrollhasard. En strukturhasard oppstår når flere instruksjoner vil ha samme ressurs, en datahasard er når en instruksjon er avhengig av resultatet fra en annen instruksjon for å fortsette og en kontrollhasard oppstår når en hoppinstruksjon gjør at neste instruksjon ikke ligger i neste sekvensielle minnelokasjon.

**Superskalar Prosessor:**

En superskalar prosessor består av flere parallelle pipelines. Den superskalar prosessorens grad forteller oss hvor mange parallelle pipelines den har. En superskalar prosessor av 3. grad vil f.eks. ha 3 parallelle pipelines. Man skulle tro at i en superskalar prosessor vil hasarder inntreffe oftere, men en kompilator som er tilpasset en superskalar prosessor vil bytte om på rekkefølgen til instruksjonene slik at instruksjoner som er dataavhengig ikke havner rett etter hverandre.

**Dynamisk utføring:**

Med dynamisk utføring vil prosessoren utføre hopp ved best mulig flyt ved hjelp av algoritmer som gjetter hopp. Dersom en instruksjon må vente på primærminnet vil prosessoren fortsette påfølgende instruksjoner, og lagre evt. Data som blir generert i en buffer. Hentingen fra bufferen vil skje strengt sekvensielt. Ved hopp skiller vi mellom dynamisk forutsigelse og statisk forutsigelse. Med dynamisk forutsigelse vil det lages en automatisk mekanisme som oppbevarer tidligere hoppdata. På bakgrunn av tidligere hopp data vil prosessoren gjette hvilke hopp som skal tas. Med statisk forutsigelse vil prosessoren gjette det samme for hvert hopp.

c) Hvem har ansvaret for å sikre korrekt utføring og for å løse opp i konflikter

som skyldes parallellitet på instruksjonsnivå?

Prosessoren har hovedansvaret for dette, men flere ting kan gjøre denne jobben lettere for prosessoren. En programmerer kan f.eks. programmere slik at det blir enklere for prosessoren å gjette hvordan hopp som skjer oftest. En kompilator kan også hjelpe til slik jeg snakket om i avsnittet om superskalare prosessorer. Der bytter den rekkefølge på instruksjoner for å unngå hasarder.

**Maks poeng: 20**

# **Oppgave 5 ­ Pipeline og cache**

For å oppnå høy ytelse på en prosessor med pipeline­arkitektur benyttes

gjerne separat data­ og instruksjons­cache. Hvorfor er dette fornuftig? Gi

eksempler og tegn en figur for å underbygge svaret ditt.

Med en separat data- og instruksjons-cache kan du både hente instruksjoner og skrive/hente data samtidig. Dette kan du fordi du vil ha en buss til hver cache, noe som ikke hadde vært mulig om alt lå i en cache, for da ville alt gått over en buss.

**Maks poeng: 10**

# **Oppgave 6 ­ Datarepresentasjon**

a) 2­er komplement er den vanligste metoden til å representere heltall med

fortegn.

1. Hvordan ser man at et bitmønster representerer et negativt tall når man

bruker 2­er komplement?

Vet at fortegnet til tallet er oppgitt ved den mest signifikante biten til bitmønstre. Om den mest signifikante biten er 1 vil bitmønsteret representere et negativt tall. Om MSB er 0 vil bitmønsteret representere et positivt tall.

2. Hvis det samme bitmønsteret er gitt på heksadesimal form, hvordan ser

man da hvorvidt bitmønsteret representerer et negativt tall? (Vær nøye

med å begrunne svaret og angi eventuelle forutsetninger for at det skal

være rett)

Om det hadde vært snakk om et hexadesimal tall hadde vi sett om hexadesimalen som representerer MSB er fra 8-F, da vil tallet bli negativt. Om det er 0-7 vil tallet være positivt. Dette er fordi 8-F hexadesimal vil alltid ha 1 som MSB, og derfor vil det bli negativt, mens 0-7 vil alltid ha 0 så det vil bli positivt.

c) Anta at følgende to bit­mønstrene representerer 8­bits tall på 2­er

komplement form.

Bitmønster 1: 10010100

Bitmønster 2: 01011011

Finn følgende for hvert av de to bitmønstrene:

1. Representerer bitmønsteret et positivt eller negativt tall? Svaret skal

begrunnes.

Bitmønster 1 er negativt fordi MSB er 1, mens bitmønster 2 vil være positivt for MSB er 0.

2. Hvilken verdi representerer bitmønsteret (skriv verdien i titall­systemet)?

Vi har lært flere metoder for å gjøre dette. Du velger selv hvilken metode

du vil bruke, men vis tydelig hvordan du går frem.

Bitmønster 1:

Siden dette er negativt må vi regne om for å finne hvilken verdi bitmønsteret representerer. Dette gjøres ved å legge til en og speile tallet.

Tar vare på alle tallene slik dem er frem til første 1er men tallene etter snur vi:

10010100 = 01101100

Dette blir da:

(0\*27)+(1\*26)+(1\*25)+(0\*24)+(1\*23)+(1\*22)+(0\*21)+(0\*20)

= 0 + 64 + 32 + 0 + 8 + 4 + 0 + 0

=108

Husker at dette representerte et negativ tall så da får vi:

-108.

Bitmønster 2:

Siden dette er positivt representerer det heltallet, slik at vi kan oversette direkte fra binær til desimal:

0 1 0 1 1 0 1 1

= (0\*27)+(1\*26)+(0\*25)+(1\*24)+(1\*23)+(0\*22)+(1\*21)+(1\*20)

= 0 + 64 + 0 + 16 + 8 + 0 + 2 + 1

= 91.

**Maks poeng: 20**