n

# **Oppgave 1. Operativsystemer: Diverse. Vekt: 30%**

**a) Sidedelte systemer (paging) for minneadministrasjon løste flere problemer. Hvilke problemer var det?**

Med sidedelte systemer deles prosessens virtuelle adresserom inn i like store deler kalt sider («pages»). Internminnet er også delt inn i like store enheter kalt «frames». En «page» som hentes fra disk tar plass i en «frame» i internminnet. Dette gjør at man reduserer fragmentering til et minimum siden «page» og «frame» størrelsen er lik vil man ved behov hente inn deler av prosessen inn i internminnet som passer eksakt inn i en «frame» i internminnet. Det er dermed ikke nødvendig å lete etter en passende plass i minnet.

Den eneste formen for fragmentering som finnes er intern fragmentering, som skyldes at den siste «page» som prosessen allokerer vil i gjennomsnittet være halvfull og dermed går en halv «page» til spille

**b) Påstand: To programmer innenfor et virtuelt minnesystem kan ha samme virtuelle (logiske) adresse. Diskuter holdbarheten til denne påstanden.**

Med et virtuelt minnesystem har vært program sitt eget kart. Dette kartet inneholder en oversikt over virtuelle adresser, altså adressene programmene bruker, og deres tilsvarende fysiske adresser, altså det hardware bruker for å snakke med RAM-en. Siden hvert program tror den har alt minnet for seg selv vil det oppstå situasjoner der flere programmer kan ha referanser til samme minneadresse. Dette er mulig på grunn av at de har sitt eget kart der den virtuelle adressen peker på ulike fysiske adresseres.

**c) Hva brukes et baseregister til i forbindelse med minneadministrasjon? En sidetabell kan sees på som en tabell med mange base-registre. Forklar.**

*Et baseregister sier noe om hvor en prosess starter i det fysiske minnet.* En sidetabell er en datastruktur som inneholder to kolonner, «page» nummer og «frame» nummer. «Page» nummer oppbevarer navnet på de forskjellige del-prosessene mens «frame» nummeret forteller oss start adressen til del-prosessen i det fysiske minnet. På denne måten kan man se på sidetabellen som en liste med start adresser til forskjellige del-prosesser i det fysiske minnet. Siden et baseregister forteller noe om hvor en prosess starter i det fysiske minnet kan man se på sidetabellen som en tabell med base-registre.

*Siden en prosess kan både ha ting lagret i primær minnet (CPU) og sekunder minnet (HDD) trenger vi å vite hvilke deler som er lagret hvor. Siden det virtuelle minnet og det fysiske minnet er delt opp i blokker med en bestemt størrelse må ofte prosesser som ikke får plass innenfor en slik blokk deles opp. Oversikten over delene prosessen er blitt delt opp i finner vi i sidetabellen,*

**d) Skriv et kort C-program (pseudo-kode) for Linux som oppretter 2 barneprosesser. Den første barneprosessen skal skifte ut koden sin med koden til ls-kommandoen, den andre prosessen skal skifte ut koden med koden til ps-kommandoen.**

**e) I operativsystemer opererer en med kernel modus og user modus. Forklar hva dette er og hva hensikten med slike moduser.**

***Kernel modus:*** I kernel modus har den kjørende koden ubegrenset tilgang til den underliggene hardwaren. Den kan kjøre alle CPU-instruksjoner og referere til alle adressene i minnet. Kernel modus er reservert for de mest pålitelige funksjonene i et operativsystem, og et krasj i kernel modus ville vært katastrofalt, de vil stoppe maskinen.

***User modus:*** I user modus har den kjørende koden ikke direkte tilgang til hardwaren eller alle minne referanser. Kode som kjøres i user modus må delegere til systemets API for å få tilgang til hardware eller minnet. På grunn av beskyttelsen av isolasjonen til user modus vil krasj være overkommelig.

Grunnen til at vi separerer disse modusene er for økt sikkerhet siden ikke alle prosesser har fri tilgang til CPU-en og minnet. Siden nesten all kode på maskinen utføres i user modus vil det ikke være kritisk om vi hadde fått et krasj, dette hadde ikke vært tilfelle om alt ble kjørt i kernel modus.

**f) En datamaskin bruker 22-bits virtuelle adresser og 16-bits fysiske adresser. Sidestørrelsen er 8K. Hvor mange innslag trengs i sidetabellen?**

Vi regner om størrelsen på de virtuelle adressene.

222 = 4 Mbyte

Deretter deler vi dette på sidestørrelsen for å finne antall innslag som trengs i sidetabellen.

4Mbyte/8Kbyte = 512

Det trengs 512 innslag trengs i sidetabellen.

# **Oppgave 2. Operativsystemer: Prosesser og kommunikasjon mellom prosesser. Vekt: 30%**

**a) Gi en definisjon av følgende begreper:**

**i. Program**

En sekvens av instruksjoner en datamaskin kan tolke og utføre.

**ii. Prosess**

En prosess er en samling av ressurser som er nødvendige for å utføre en oppgave som er beskrevet med programkode på en datamaskin.

**iii. Tråd**

Utførende enhet som går gjennom programkoden og utfører maskininstruksjoner

Føler det er naturlig å nevne forskjeller mellom et program og en prosess siden definisjonene er såpass like. Et program er en statisk enhet der koden ikke forandrer seg når programmet utføres, mens en prosess er en dynamisk enhet som endres hele tiden under kjøringen av programmet.

**b) Forklar hvordan semaforer virker.**

En semafor er en variabel som brukes for å styre tilganger til ressurser. Dette gjøres ved hjelp av to operasjoner, wait og signal.

**Wait:** denne operasjonen sjekker verdien på variabelen semafor, og gjør følgende ut i fra hvilken verdi semafor har:

* Semafor-verdi > 0 betyr at prosessen kan fortsette.
* Semafor- verdi = 0 betyr at prosessen må flyttes til ventekø.

**Signal:** denne operasjonen øker verdien til semaforen. Hvis en eller flere prosesser venter på semaforen, vil en av disse bli valgt og kan dermed fullføre sin wait-operasjon og får dermed adgang til felles dataområde.

**c) Anta at prosessene P0 og P1 deler en felles variabel V2, prosessene P1 og P2 deler variabelen V0 og prosessene P2 og P3 deler variabelen V1**

**Vis hvordan disse prosessene kan bruke enableInterrupt og disableInterrupt (dvs slå av og på avbruddssystemet) for å styre adgangen til V0, V1 og V2 slik at kritisk region-problemer ikke oppstår.**

**d) Vis også hvordan prosessene kan bruke semaforer for å styre adgangen til V0, V1 og V2 slik at kritisk region-problemer ikke oppstår.**

**e) To prosesser P1 og P2 er laget slik at P2 skriver ut data som er produsert av P1. Lag et enkelt program (skjelett, pseudo-kode) for P1 og P2 for å illustrere hvordan de synkroniserer hverandre.**

**f) I utgangspunktet ønsker vi ikke å bruke enableInterrupt og disableInterrupt for å sikre gjensidig utelukkelse. Hvorfor? Men i semaforer bruker vi det likevel. Forklar hvorfor det er akseptabelt å bruke enableInterrupt og disableInterrupt i semaforer.**

Grunnen til at det er akseptabelt å bruke enableInterrupt og disableinterrupt i semaforer er fordi når slår av avbruddsystemet tar det ikke lang tid før det er skrudd på igjen og det vil ikke ha noen innvirkning på responstid eller andre ting som har med avbrudd å gjøre. Dette foregår under meget kontrollerte former, der en prosess ufører f.eks. en wait-operasjon vil den slå av avbruddssystemet, deretter uføre noen operasjoner på semaforvariabelen, og tilslutt slå på avbruddssystemet igjen. Siden prosesser som utfører wait- eller signal-operasjon er garantert til å fullføres er det ingen risiko for at prosesser kan bli hengende inni disse operasjonen. Derfor vil en prosess alltid slå på avbruddsystemet igjen, og det er ingen risiko for “død” maskin som følge av avslått avbruddsystem. Mens om vi bruker enableInterrupt og disableInterrupt på andre måter står vi i fare for at dette skjer, og det er derfor ikke ønskelig å bruke.

# **Oppgave 3. Datateknikk: Moderne prosessorarkitektur Vekt: 40%**

**a) En instruksjon sørger for at følgende 5 hendelser ( i., ii.,…,v. ) skjer:**

***i. Dekod instruksjonen.***

***ii. Hent operand fra minnet.***

***iii. Hent instruksjonen fra minnet og legg den i instruksjonsregisteret.***

***iv. Lagre resultatet i minnet.***

***v. Utfør instruksjonen.***

**Ordne disse ( i. - v. ) i riktig rekkefølge.**

iii. Hent instruksjonen fra minnet og legg den i instruksjonsregisteret.

i. Dekod instruksjonen.

ii. Hent operand fra minnet.

v. Utfør instruksjonen.

iv. Lagre resultatet i minnet.

**b) For hvert av trinnene i..v:**

**Angi om trinnet trengs for alle instruksjoner, eller om trinnet bare trengs for enkelte instruksjoner. Vær nøye med å begrunne dine påstander. Gi gjerne eksempler på instruksjoner som kan illustrere dine påstander.**

Trinnene ii. (Hent operand fra minnet) og iv. (Lagre resultatet i minnet) trengs ikke for alle instruksjoner. Om en instruksjon ikke trenger data er det ingen grunn for å hente noe fra minnet. Det samme gjelder om ikke instruksjonen lager noe ny data som må skrives til minnet. Eksempler på instruksjoner som trenger alle 5 trinnene vil være Read/Write instruksjoner.

**c) En prosessor bruker en 5-trinns pipeline med trinnene som er angitt i deloppgave a). Det skal utføres 1000 instruksjoner på denne prosessoren. Anta at hvert trinn utføres i løpet av en klokkesyklus. Hvor mange klokkesykluser tar dette under ideelle forhold? (Ideelle forhold vil si at vi får maksimal uttelling ved bruk av pipeline). Vis tydelig hvordan du finner svaret.**

Den første instruksjonen bruker 5 klokkesykluser. Deretter avsluttes de resterende

instruksjoner med en klokkepuls’ mellomrom. Formel blir P+n-1, der P er antall trinn i

pipen (5 her) og n er antall instruksjoner (1000 her).

5+1000-1= 1004

Det vil ta 1004 klokkesykluser å utføre 1000 instruksjoner under ideelle forhold.

**d) Vanligvis vil en prosessor bruke flere klokkesykluser enn det antall du beregnet i oppgave i forrige oppgave. Dette skyldes at det oppstår såkalte hasarder. Beskriv følgende hasarder og forklar hvordan de løses:**

***i. Strukturell hasard***

Flere instruksjoner trenger samme ressurs samtidig. For å unngå dette settes tomgangstrinn i den siste instruksjonen

***ii. Datahasard***

En instruksjon trenger resultatet fra en annen instruksjon som ikke har resultatet klart enda. Dette unngås også med å sette inn tomgangstrinn i den siste instruksjonen.

***iii. Kontrollhasard***

Hopp i programmet gjør at vi ikke vet hvilke instruksjoner som skal utføres før hoppinstruksjonen er utført. Dette unngås ved at CPU-en må gjette hvor det skal hoppes. Dette er enkelt for ubetingede hopp, men vanskelig for betingende hopp, og en konsekvens ved feilgjetting vil være en pipeflush.

**e) En av de følgende har ansvaret for å løse hasarder på en slik måte at programutføringen blir korrekt:**

***i. Brukeren***

***ii. Programmereren***

***iii. Kompilatoren***

***iv. Prosessoren***

**Hvem av disse (i. – iv.) har ansvaret? Vær nøye med å begrunne svaret ditt. Kommenter gjerne på om de andre også kan spille en rolle.**

Det er prosessorens ansvar for at hasarder blir løst på en slik måte at programutføringen blir korrekt. Men andre faktorer kan være med å hjelpe til å gjøre denne jobben lettere. For eksempel en god programmerer kan programmere med hensyn på hasarder som kan oppstå. En kompilator kan også være med på å løse datahasarder, der den stokker om på instruksjonsrekkefølgen slik at instruksjoner som er dataavhengige ikke kommer rett etter hverandre.

**f) En splittet cache er en cache som består av to del-cacher. Den ene del-cachen brukes bare til instruksjoner, og den andre brukes bare til data. Er pipeline en årsak til at det kan være fornuftig å bruke en slik splittet cache? Begrunn svaret nøye.**

Det er meget fornuftig å benytte en splittet cache med bruk av pipelines. Siden instruksjoner foregår parallelt vil vi kanskje ha en instruksjon som skal hentes mens en annen skal hente data. Uten en splittet cache ville vi fått en strukturell hasard siden bare en kan dra over bussen om gangen, men med en splittet cache får vi hentet instruksjoner og data samtidig.