**Oppgave 1. Generelt om opsys og om prosesser (vekt: 25%)**

**a)** *Vi sier at operativsystemet har to viktige roller. Den ene er å være ressursadministrator. Den andre er å være et abstraksjonslag slik at programvare kan kjøre på maskinen uten å måtte kjenne til maskinvaredetaljer. Forklar hva dette abstraksjonslaget er for noe.*

**Dette abstraksjonslaget eksisterer som en barriere mellom maskinvaren og applikasjoner. Dette lage kalles for kjernen (kernel), og gjør at ikke applikasjoner kan få tilgang til maskinvaren uten tillatelse, noe som beskytter maskinvaren fra fiendtlige programmer. Dette funker ved at operativsystemet har noe vi kaller «dual-mode» som består av kjernemodus og brukermodus. Vanligvis vil maskinen være i brukermodus, som har begrenset tilgang, og om den trenger tilgang til f.eks. maskinvare må den utføre et systemkall, dette vil sette den over i kjernemodus og sjekke om prosessen som kjøres har de riktige rettighetene til å få tilgang til maskinvaren, om den har det så får den tildelt ressursen om ikke oppstår det en feilmelding og prosessen blir kastet ut.**

**b)** *I en enkel modell av prosessene kan en prosess være i 3 tilstander. Hvilke tilstander er det?*

**Klar – En prosess står i denne tilstanden når den kun mangler en ressurs,**

**denne ressursen er CPU-en. Prosesser i denne tilstanden danner en kø som blir kalt for CPU-køen.**

**Kjørende – Prosesser som kjører på CPU-en er i en kjørende tilstand.**

**Blokkert – Om en prosess ikke har tilgang på ressursene den trenger f.eks.**

**I/O-utsyr blir den plassert i en blokkert tilstand til den får tilgang til ressursen. Prosessene i denne tilstanden danner en kø som kalles for ressurs-kø.**

**c)** *Det er 4 mulige overganger mellom disse tilstandene nevnt i oppgaven ovenfor. Hvilke er det og hva er det som fører fra en tilstand til en annen? d) En prosess kan splitte kjøringen opp i flere tråder. Forklar hvorfor det å operere med flere tråder innenfor en prosess er hensiktsmessig.*

**Klar 🡪 Kjørende: En prosess går fra klar til kjørende når den blir tatt ut av CPU-køen og plassert på CPU-en.**

**Kjørende 🡪 Klar: Om en prosess overskrider tidskvarten sin blir den plassert tilbake i klar tilstand (tilbake i CPU-køen). Grunnen til at operativsystemet gjør dette er slik at vi får simulert at flere prosesser kjører samtidig selv om de kun kjører en om gangen på CPU-en. Dette kalles for kvasiparallell.**

**Kjørende 🡪 Blokkert: Om en prosess trenger tilgang til en ressurs mens den kjører blir den plassert i en blokkert tilstand. Dette er for å spare tid, slik at andre prosesser som kun trenger tilgang til CPU-en kan fortsette å kjøre.**

**Blokkert 🡪 Klar: Når en prosess får tilgang til ressursen den har ventet på kan den stilles tilbake i CPU-køen og går derfor tilbake til en klar tilstand. Alle prosesser som er blitt blokkert må gjennom klar tilstand før de får kjøre på CPU-en igjen.**

**Oppgave 2. Minneadministrasjon (vekt: 25%)**

***a)*** *Et viktig begrep i minneadministrasjon er adressebinding. I dagens operativsystemer skjer adressebindingen under kjøring. Slik har det ikke alltid vært. - Hva er adressebinding og til hvilke tidspunkt skjer det? - Forklar også hvordan adressebinding under kjøring foregår.*

**Adressebinding skjer når en virtuell adresse peker på et adresseområde i fysisk minnet som er utenfor base+bound til rammen hos forrige binding. Når dette ikke skjer automatisk vil man møte et avbrudd. I automatisk gjennomføring gjøres enten ved å finne rammen til den virtuelle adressen sin side i TLB(En hurtigbuffer for adressebinding), eller om adressen ikke nylig har blitt brukt må man finne rammen til adressen i segmentabellen. Dette tar da noe lengre tid. Når man har funnet rammen peker dette til et sted i fysisk minne hvor man så kan lese det man trenger.**

**b)** *På en 64-bits CPU har prosesser et gigantisk adresserom. Likevel kan prosessene kjøre på en maskin med relativt lite minne. Forklar.*

**Grunnen til at man kan ha et såpass gigantisk adresserom er fordi disse adressene er virtuelle og ikke enda bunnet til en fysisk ramme. Denne bindingen skjer ikke før adressen det faktisk gjelder tas i bruk av en blokk. Man får da illusjonen av en evig mengde adresse selv om maskinvaren har begrensninger.**

**c)** *To programmer innenfor et virtuelt minnesystem kan ha samme virtuelle adresse. Diskuter holdbarheten til denne påstanden.*

**To forskjellige programmer kan ha samme virtuelle adresse, da disse kan bindes til forskjellige områder i fysisk minnet etter at segment-tabellen har blitt tatt i bruk. Påstanden er derfor veldig holdbar. De virtuelle adressene utnytter noe som kalles address layout randomization, hvor de virtuelle adressene blir vilkårlig satt i virtuelt minne noe som kan forårsake at et denne typen ting oppstår. Det er viktig å påpeke at dette ikke går for primærminnet.**

**d)** *Hva menes med prosessens adresseområde? Kan prosessens adresseområde være større enn fysisk tilgjengelig minne? Forklar.*

**En prosess har et adresseområde som beskriver antall adresser det er mulig å holde innad en prosess sin heap og stack. En adresse peker da til en blokk i fysisk minne. En prosess kan ikke ha et større adresseområde enn det minnet den har fått tildelt tillater. Alle prosesser får tildelt en begrenset mengde ressurser (Som kan finnes i PCB) og prosessen kan derfor ikke peke til flere rammer i fysisk minne enn rammene den har fått tildelt av minnet.**

**Oppgave 3. Tallsystemer (vekt: 15%)**

***a)*** *Et vanlig tastatur har 104 taster. Når du trykker en tast sendes det et bitmønster til datamaskinen. Dette bitmønsteret identifiserer tasten som ble trykket, og må derfor være unikt for hver enkelt tast. Svar på følgende to spørsmål: - Hvor mange bits må bitmønsteret minst være for at hver tast skal ha sitt unike bitmønster? - Hvor mange flere taster kunne tastaturet ha hatt uten at vi trenger å bruke flere bits?*

**For at hver tast skal ha sitt unike bitmønster må vi ha 7 bits i bitmønsteret. Dette gir oss 128 forskjellige kombinasjoner (27 = 128). Siden vi kun bruker 104 av disse 128 kombinasjonene kan vi ha 24 ekstra taster uten at vi må bruke flere bits.**

**b)** *Skriv følgende bitmønster på heksadesimal og desimal form: 1001011101 Vis fremgangsmåten nøye, og forklar hva du gjør.*

**For å finne bitmønsteret på heksadesimal form deler vi opp bitmøsntere etter hvert fjerde tall: (00)10 0101 1101. Videre bruker vi tabellen for å oversette de tre forskjellige kombinasjonene med 0ere og 1ere og får: 25D.**

**Vi bruker heksadesimalformen for å regne om til desimalform:**

**2 \* 16^2 + 5 \* 16^1 + D \* 16^0 = 2 \* 256 + 5 \* 16 + 13 = 512 + 80 + 13 = 605.**

**Oppgave 4. Prinsippet om lokalitet (vekt: 15%)**

*Man skulle kanskje tro at når CPU aksesserer primærminnet, så hadde alle minnelokasjoner den samme sannsynligheten for å bli brukt til enhver tid. I praksis viser det seg imidlertid at dersom man vet hvilken lokasjon som ble aksessert forrige gang, så kan man med stor sannsynlighet forutsi (omtrent) hvor neste aksess blir.*

***a)*** *Hvor ligger sannsynligvis neste aksess?*

**Neste aksess ligger mest sannsynligvis sekvensielt etter forrige aksess, eller like i nærheten.**

**b)** *Denne egenskapen ved programmene har gitt opphav til et viktig prinsipp i datateknikk, nemlig prinsippet om lokalitet. Hva sier prinsippet om lokalitet?*

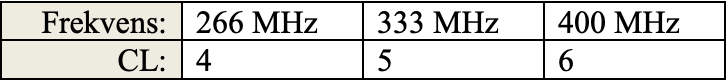
**Prinsippet om lokalitet sier at dersom en minnelokasjon er benyttet en gang er det svært sannsynlig at den eller en lokasjon i nærheten snart blir benyttet en gang til.**

**c)** *Gjelder prinsippet både for instruksjoner og for data, eller gjelder det bare for en av dem? Begrunn svaret nøye, og gi eksempler på at det gjelder / ikke gjelder.*

**Prinsippet gjelder for både data og instruksjoner. Prinsippet gjelder for instruksjoner fordi de ofte har en sekvensiell utførelse eller siden vi bruker løkker, der kode gjentas flere ganger, i tillegg er mange programkonstruksjoner er iterative. Prinsippet gjelder for data siden mange datastrukturer er gjerne sammenhengende (eks arrays), noe som bør aksesseres sekvensielt. Vi har også tilfeller der prinsippet ikke gjelder, feks ved bruk av systemkall og hopp-instuksjoner, men dette skjer sjeldnere enn det nevnt ovenfor.**

**Oppgave 5. Moderne primærminne (vekt: 20%)**

*Moderne RAM-typer er synkrone. Det betyr at alle hendelser synkroniseres med hjelp at et klokkesignal (pulser med fast frekvens). Når vi skal lese fra minnet må vi alltid vente på at minnet skal gjøre informasjonen tilgjengelig. Denne forsinkelsen (engelsk: latency) oppgis i antall pulser, og fremgår av spesifikasjonene til minnet. I beste fall er forsinkelsen gitt av parameteren CL (CAS-latency). Tabellen nedenfor viser et eksempel på at et minne kan brukes med ulike frekvenser, hvor hver frekvens har en tilhørende forsinkelse:*

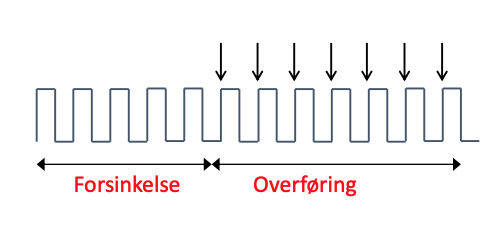


***a)*** *Du trenger ikke forklare hva CL er, men svar på følgende spørsmål: Vi ser at en og samme minnebrikke kan brukes på ulike frekvenser. Men forsinkelsen (latency) blir større ved høyere frekvenser. Betyr det at minnet blir tregere for høyere frekvenser? Begrunn svaret nøye.*

**Nei, dette betyr ikke at minnet er tregere ved høyere ved frekvens. Aksesstiden vil ikke endre seg uansett hvordan vi endrer frekvensen, men det som endrer seg er antall klokkepulser. Dette vil øke ved økt frekvens, noe som gjør at det tar kortere tid mellom hver klokkepuls. Dette har ikke noe effekt på aksesstiden, men har en effekt på dataoverføringen etter aksesstiden. CAS-latency (CL) forteller oss hvor mange klokkepulser vi må vente før dataoverføringen begynner.**

**b)** *Ta utgangspunkt i figuren nedenfor og forklar i hvilken av de to fasene (forsinkelse og overføring) vi har nytte av høyest mulig frekvens. Begrunn svaret ditt nøye.*

**Vi vil ha mest nytte av høyest mulig frekvens ved overføringsfasen. Ved høy frekvens vil tiden mellom klokkepulsene være kortere enn ved en lavere frekvens, noe som betyr at vi får overført data hurtigere. Om vi hadde økt frekvensen i forsinkelsensfasen hadde dette ikke hatt noe effekt på forsinkelsens tiden, siden denne er like uansett frekvens.**



**c)** *I figuren ovenfor viser de loddrette pilene tidspunkt der data overføres på databussen. Synkrone minneteknologier kan være såkalt DDR-teknologi. DDR betyr Double Data Rate. - Forklar hva vi mener med DDR. - Viser dataoverføringene i figuren et eksempel på DDR? Begrunn svaret.*

**DDR som står for double data rate er når vi får overført data på begynnelsen og på slutten av hver klokkepuls, noe som betyr vi får overført dobbelt så mye data med denne typen minneteknologi. Figuren ovenfor viser ikke et eksempel på bruk av DDR siden den kun viser en overføring på begynnelsen av hver klokkepuls, og ikke en overføring på begynnelsen og på slutten.**