TDAT 1003 DTK og OS

Eksamen 2016

Oppgave 1.

1. **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.**

Abstraksjonslaget i operativsystemet skal gjøre slik at programmene skal kompilere og kjøre på akkurat samme måte uansett maskinvare. Hvis man kun har 4gb med RAM, så skal et program kunne programmeres og kjøres på lik måte som hvis man kun hadde 100mb RAM. Det er også viktig å merke seg at med dårligere maskinvare kan brukeren oppleve programmer som ikke responsive og trege, så dette gjelder til en viss grad. Utviklerne må ta litt hensyn til maskinvaren som er tilgjengelig for brukerne.

Programmene som kjøres vil ha en illusjon av at de har så mye ressurser som de kan tenke seg, til tross for at dette ikke er slik i virkeligheten. Denne illusjonen gjør at alle programmer uansett type prosessor, antall gb RAM eller annet lignende komponenter som man finner i en pc, kan kjøre.

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

En prosess kan være i tre ulike tilstander: klar, ikke klar og ferdig. Også på engelsk: ready, unready, executed/finished.

Klar vil være tilstanden til en prosess hvis den er klar til å kjøre, da vil den havna bakerst i listen over programmer som venter på å kjøre. Ikke klar vil være tilstanden til en prosess hvis den f.eks. venter på IO, altså den er ikke klar enda til å kjøre. Ferdig vil være tilstanden til en prosess hvis den er ferdig å kjøre, og ikke har mer som skal kjøres, eller f.eks. venter på IO.

1. **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?**

**«Prosess exception»/unntak:** Hvis det blir kastet et unntak, f.eks. hvis man deler på 0, så vil **«exception handler»**: stoppe prosessen slik at ingen feil oppstår. Prosessen får da tilstanden ferdig/ «executed». Eksempel på dette er hvis man deler på 0.

**«Interrupt»:** Hvis det oppstår en forstyrrelse fra OS eller IO under kjøring av en prosess, f.eks. for å forhindre at programmer kjører i det uendelige eller sjekke for input fra bruker, vil prosessen bli stoppet etter at instruksjonen er ferdig.

**«Schedueler»**: OS kan bestemme at en annen tråd eller prosess skal få kjøre vil den stoppe kjøringen og bytte til en annen.

1. **En prosess kan splitte kjøringen opp i flere tråder. Forklar hvorfor det å operere med flere tråder innenfor en prosess er hensiktsmessig.**

Tråder er «sub-prosesser» av en prosess. Disse trådene deler mye av ressursene til prosessen, blant annet stacken og heapen er felles for alle trådene i en prosess. Fordelen med tråder er at man kan mye mer effektivt og enklere bytte mellom flere oppgaver på samme tid. Dette gjør at man får en illusjon av parallell kjøring av flere oppgaver, også kalt «thread abstraction».

Oppgave 2.

1. **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?**

Adressebinding er når en virtuell minneadresse blir konvertert til en fysisk adresse i minnet. Denne konverteringen foregår hver gang en virtuell adresse blir brukt av prosessoren. Prosessoren ser kun denne virtuelle adressen og sender ut en forespørsel på minnebussen. Deretter finnes det elektronikk som konverterer den virtuelle adressen og finner den fysiske minneadressen. Legg merke til at prosessoren ikke bryr seg om disse fysiske adressene, men kun de virtuelle adressene.

Denne adressekonverteringen skjer hver gang prosessoren skal hente data eller instruksjoner fra en adresse. Prosessoren sender ut en signal til TLB (Translation Lookaside Buffer) som vil sjekke de ulike cachene hvis det finnes flere, hvis ikke vil den se i primærminnet. Hvis den ikke finnes i primærminnet, vil IO moduler sjekke andre lagre.

* + **Forklar også hvordan adressebinding under kjøring foregår**.

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

Prosessorer opererer med virtuelle adresser til det virtuelle minnet til prosessene. Alle prosessene vil også ha et område i det fysiske minnet, men dette håndterer «kernelen».

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

Hvis man tenker og går ut ifra at disse programmene er i to forskjellige virtuelle minnesystemer vil ikke dette være noe problem. Når prosessoren skal hente data eller instruksjoner i minnet, brukes den virtuelle adressen. Denne adressen gjøres da om til den fysiske adressen i minnet.

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

Prosessorer opererer med virtuelle adresser til det virtuelle minnet til prosessene.

Oppgave 3.

1. **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?**

For at hver tast skal ha sin unike kode/adresse, må vi ha minst 104 adresser. Den første mulige bit som går opp i dette kravet er 7 bits. 27 = 129 adresser. Hvis man går ned på 6 bits, 26 = 64, får man ikke nok adresser.

* + **Hvor mange flere taster kunne tastaturet ha hatt uten at vi trenger å bruke flere bits?**

129 – 104 = 25. Tastaturet kunne ha hatt 25 flere taster hvis vi hadde trengt det. NB! 0 er også en adresse, derfor 129, ikke 128.

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

Heksadesimal: Tar 4 og 4 bits i gangen siden heksa = 16 = 24. Deretter konverterer jeg disse til de heksadesimale verdiene som følger, lavest til høyest (0-1610): 0 1 2 3 4 5 6 7 8 9 A B C D E F

1101 = 8 + 4 + 1 = 1310 = D16

0101 = 4 + 1 = 510 = 516

10 = 210  = 216

*1001011101 = 25D16*

1\*20+ 1\*22+ 1\*23+ 1\*24+ 1\*26+ 1\*29= 1 + 4 + 8 + 16 + 64 + 512 = *60510*

Oppgave 4.

**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.**

1. **Hvor ligger sannsynligvis neste aksess?**

PC registeret i CPU vil i de fleste tilfeller øke verdien med en, slik at neste instruksjon som hentes og kjøres, vil være den instruksjonen rett etter den som kjører nå. Altså instruksjonene kjøres som oftest sekvensielt, derfor ligger også instruksjonene sekvensielt i primærminnet. Unntaket i dette tilfellet er hvis en instruksjon spesifiserer et instruksjonshopp, som i for-looper. På grunnlag av dette kan man si at neste aksess vil mest sannsynlig være neste minneadresse i instruksjonsområdet, eller litt bak hvis programmet går i en loop. Generelt vil minnelokasjonene som brukes klumpe seg rundt de siste lokasjonene som ble brukt.

1. **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 hvis en minnelokasjon blir brukt en gang, er det høy sannsynlighet for at den eller lokasjonen ved siden av snart kommer til å bli brukt igjen. Man ser at minnelokasjonene som brukes av CPU gjerne klumper seg sammen i en gitt periode. Hele minnet brukes over lengre tid, men det er nyttig å kunne ha deler av minnet i cache ved å bruke prinsippet om lokalitet, siden CPU-en er såpass hurtig. Fordelen med å bruke dette prinsippet er at man kan med en viss sikkerhet vite at neste instruksjon eller data som skal brukes vil mest sannsynlig ligge i cache, noe som senker tiden CPU må vente på aksesser.

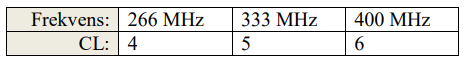
1. **Gjelder prinsippet både for instruksjoner og for data, eller gjelder det bare for en av dem? Begrunn svaret nøye, og gi eksempler på at det gjelder / ikke gjelder.**

Prinsippet gjelder både for instruksjoner og for data. Generelt i programmer brukes gjerne data flere ganger i instruksjoner. Et eksempel er while og for løkker i programmering, ved å bruke disse vil man bruke de samme data og instruksjonene gjentatte ganger. Denne typen kalles temporal lokalitet, romlig lokalitet gjelder for sekvensiell utføring.

Dette prinsippet gjelder derimot ikke ved hopp-instruksjoner eller funksjonskall, disse ligger et annet sted i minnet og derfor vil ikke man ikke kunne forutsi dette. Det er også derfor man sier at «det er mest sannsynlig» at neste aksess ligger i cache, fordi man egentlig ikke kan vite det helt sikkert. Funksjonskall nøstes sjelden dypt derfor vil ikke dette være et stort problem med at man «bommer» gjentatte ganger i cache.

Oppgave 5.

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:



1. **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.**

Forsinkelsen til primærminnet vil ikke endre uansett frekvens. Ved overføring har frekvens mye å si, men ved forsinkelse vil det være samme uansett. Man ser at dess høyere frekvens, dess flere pulser må man vente, altså CL. Hvis man bruker formelen 1/frekvens \* CL, vil man få samme resultat. Ventetiden i alle tre kolonene vil bli rundt 15 nanosekunder. Derfor kan man si at ventetiden til minnet ikke blir tregere for høyere frekvenser, men man må vente til starten av neste klokkepuls før man kan gjøre noe som helst, som vil gi variable ventetider. Disse ekstra ventetidene er minimale, og vil minske med økningen av frekvensen.

1. **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.**

Ved forsinkelse vil ikke høyere frekvens ha mye å si, som vi så i oppgave a. Derimot ved overføring vil man kun overføre data ved hver klokkepuls. Det betyr at hvis man har en lav klokkefrekvens så vil overføringen bruke lengre tid, siden man må vente på neste klokkepuls i lengre tid. Derfor er det ønskelig å øke frekvensen så mye som mulig uten at det oppstår praktiske problemer.

1. **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.**

DDR (Double Data rate) har muligheten til å overføre data hver gang signalet endrer seg fra lav til høy verdi. Altså DDR kan overføre to ganger per klokkepuls, ved starten og slutten av det «høye» signalet. Det fleste nye minne i generelle pc-er i dag av typen DDR.

* + **Viser dataoverføringene i figuren et eksempel på DDR? Begrunn svaret.**

Dataoverføringene i figuren er ikke er eksempel på DDR. Grunnen til dette er at man ser at hver dataoverføring (pilene nedover) skjer kun en gang hver klokkepuls. Det er kun ved en ny klokkepuls at overføringen foregår. Ved bruk av DDR ville man ha like mange overføringer men kortere bortover den horisontale aksen siden det ville gått fortere.