Informasjon om kandidat/gruppe

Kurs: IKT102 Operativsystem

Øvingsoppgave: Øving 4 IPC, files and I/O

Dato: 22 November 2019

Kandidat/gruppe: Pål Tømte Karlsen og Ola Grytting

Oppgave 1

Hva er DEADLOCK? Forklar litt rundt DEADLOCK

DEADLOCK er ein situasjon som skjer når la oss sei prosess A og prosess B spør om å få tilgang til f.eks. printer. A spør om tilgang til printeren og får det. Så spør B om tilgang og får tilgang. Så spør A igjen men spørsmålet er satt på pause fram til B er ferdig. Dessverre når B er ferdig så slepper den ikkje taket på printeren og spør om tilgang til ein skanner. Siden B ikkje slepper taket på printeren så sitter A og B i ein Block situasjon som vil bli sånn for alltid. [1]

Kort sakt DEADLOCK skjer når 2 eller fleire medlemmer i ei gruppe venter på eit annet medlem, inkludert seg sjølv, til å gjer nåke, som f.eks. senda ein melding eller meir normalt frigjøra ein lås. [2]

Oppgave 2

Hva er betingelsene for DEADLOCK

I eit OS så er skjer DEADLOCK når ein prosess eller ein tråd går inn i ein ''waiting state'' fordi ein forespurt system resursen er hold av ein annen prosess som ''venter'' som igjen ''venter'' på på ein annen resurs som er hold av ein prosess som ''venter''. Viss ein prosess ikkje er i stand til å endra tilstanden sin på ubestemt tid fordi resursen som blir spørt om er okkupert av ein annen ''ventande'' prosess. [2]

Oppgave 3

Hva er race condition. Hvorfor kan det skje at noen prosesser blir blandet inn i “race conditions” (interrupts/ context switch)?

Race condition er ein uønska situasjon som skjer når ein einhet eller system prøver å utføra 2 eller

fleire operasjona samtidig [3].

Prosessa blir blanda inn i race condition når f.eks. ein verdi blir lest inn av ein prosess så kommer det ein interrupt og CPUen seier at prosessen ferdig. Så begynner ein ny prosess og oppdaterer den originale verdien. Så når den andre prosessen er ferdig så begynner den første prosessen der den blei interrupta og fortsetta å utføra sin prosess. den vil så sletta den andre prosessen sinn verdi som igjen gjer at den blir sittande fast og venter på ein output som aldri kjem

Oppgave 4

Hvorfor er ”race condition” og ”deadlock” så vanskelig å finne og gjøre noe med? Gjør det noe at vi har slike problemer i et operativsystem?

Race condition er så vansklig å finna fordi da er så tilfeldig. Ein tråd kan fullføra sin kjørings tid før tida si på prosessen er ferdig og då kan ein annen tråd kjøra. Viss detta skjer så vil ikkje problemet oppstå. Tåderutførelse er ikkje planlagt på forhånd derfor kan ein ikkje kontrollera tid og utførelsesrekkefølge. Også, tråder kan bli kjørt forskjellig fra normalt til debuging mode. Du kan også sjå at viss ein kjører kvar tråd i serie så oppstår ikkje feilen. Denna tilfeldigheten gjør at denna feilen er mykje vanskligare og finna å gjera noko med.

Race condition er ekstremt problematisk når da kommer til banker for da kan oppstå at ein mottar penger som ein kanskje ikkje sku hatt. La oss sei ein har 2 tråder

Sum = 100

Value1 = 50

Value2 = 15

Tråd 1:

Sum = Sum + value1

Tråd 2:

Sum = Sum – value2

Først kjører tråd 1 den skriver inn summen og plusser på value1, men før ikkje summert dei sammen. Så kjører tråd 2 og den får kjørt heilt ut og får 85, men så får tråd 1 fullført sin aksjon og

summerer og får 150 derfor når tråd 1 er ferdig får ein 150 istedenfor 85.

DEADLOCK skjer ikkje alltid. Viss ein tråd får tillatelse til og utføra og låsa begge resursene ein trenger før prosessen stopper tråden, så kan tråden fullføra operasjonen sin og låsa opp begge resursane sine. Etter at tråden har låst opp resursane sine så kan ein annen tråd kjøra sin prosess som forventa.

Denna feilen er lett og oppdaga når kodane er side om side, men praktisk så er kodane separerte og kanskje i forskjellige moduler eller områder. På grunn av detta så er da ekstremt vansklig å finna fordi fra same kode så kan korrekt og ukorrekt utførelse skje.

DEADLOCK er eit problem fordi da skaper uendelig venting og usikkerhet, 2 eller fleire system som venter på kvarandre som skaper uendelig venting, detta fører til mangel på ytelse og gjennomstrømming. [4]

Oppgave 5

Hva er en semafor og hvordan virker den? Bruk et eksempel for å forklare. Hva er en ”Mutex”?

Semafor er ein variabel eller abstrakt data type som blir brukt til å kontrollera tilgong til vanlige resursa av fleire prosessa i eit samtid system som eit multitasking operativ system.

Da virker med at variabelen blir brukt til å løysa kritiske seksjon problem og å oppnå prosess synkronisering i eit multi prosesserings miljø. [5]

Ein har ein variabel A og ein boolean B. A er kun tilgjengelig viss B er sann. Så B er ein semafor for A.

Viss ein tenke at signal(B) rett før ein togstasjon(A). i detta tilfelle, viss signalet er grønt, så kan toget kjøra inn i stasjonen. Viss signalet er noko anna enn grønt så er ikkje stasjonen tilgjengelig. [6]

Når semaforens evne til og tella ikkje lenger er bruk for, så blir ein enklare versjon av semaforen som hetter Mutex av og til brukt. Mutex er kun brukande til å administrera ‘’mutual exclusion’’ til nokre delte resursa eller deler av kode. Dei er enkle og effektive å implementera, noko som gjer at

dei er spesielt nyttige i trådepakker som er fullstendig implementerte i bruker område. [7]

Oppgave 6

Hva er Busy waiting og hvorfor er dette uønsket i operativsystemer. Forklar en løsning som bruker ”busy waiting”.

Busy waiting er ein ‘’loop’’ der prosessen heila tida sjekker om ein viss betingelse er sann som f.eks. tastatur input eller ein resurs er ledig.

Busy waiting resulterer i sløsing med ressurser, da holder CPUen opptatt konstant med å heila tida sjekka loopen. Program som aktivt ‘’venter’’ på noko skal skje, kan skapa system feil pågrunn av ‘’race condition’’. [8]

Ein har fleire løysningar som bruker busy waiting

- Dekker’s algorithm
- Lamport’s bakery algorithm
- Szymanski’s algorithm
- Peterson’s algorithm
- Osv. [9]

Peterson’s Solution bruker busy waiting. Da er eit samtids program algoritme som bruker mutual exclusion som tillatter 2 eller fleire prosessa til å dela eingangs resursa uten konflikt. Denne løysinga blir brukt til og løysa race condition [10]

Oppgave 7

Forklar grundig hvordan DMA(Direct memory Access) virker og hva som er fordelene/ulempene

med det?

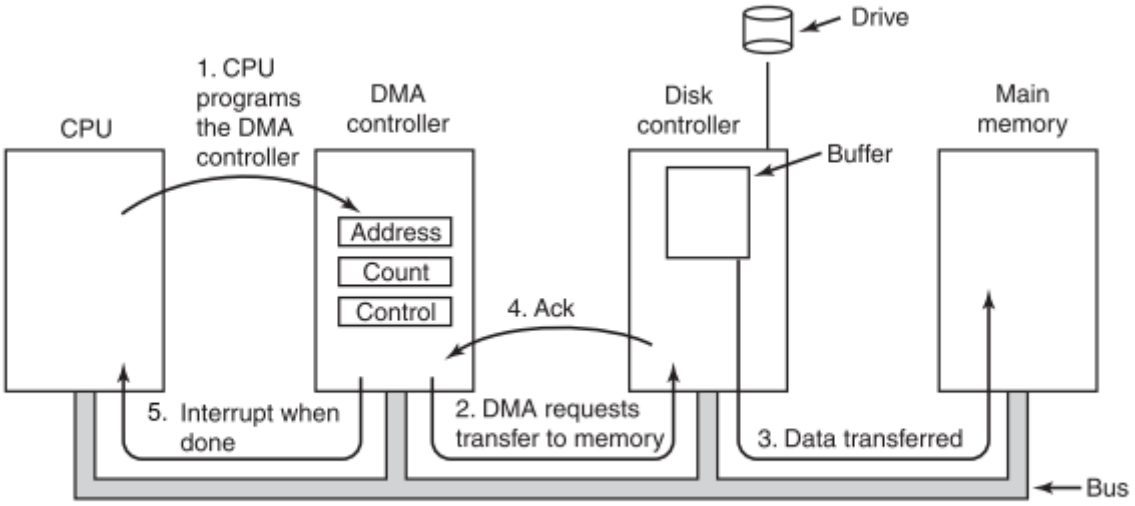


Figure 5-4. Operation of a DMA transfer.

DMA fungerer med at da først så programmerer CPUen DMA ved å setta registeret sånn at da veit kva som skal kor(1). Da gir også ut kommandoa til disken og forteller den til å lesa data fra disken inn i interne bufferen. Når gyldig data er i disk kontrollerens buffer så kan DMAen begynna. DMAen starter med å senda ''read'' forespørsel over til bussen til disken(2). Denna ''read'' forespørselen ser ut som alle andre ''read'' forespørsla og disken veit ikkje/bryr seg ikkje om da kom fra CPUen eller DMAen. Typisk så skriver minne adressa til buss adressas linje så når disken henter da nesta ordet fra sin interne buffer så veit da kor da skal skrivas(3). Når skrivingo er ferdig så sender disken ein bekreftelse signal til DMAen(4). DMAen øker så minne adressen som skal brukas og minker byte tellingo. Viss byte tellingo er større enn 0 så går ein gjennom 2 til 4 igjen heilt til tellingo når 0.

Når da skjer så avbryte DMAen CPUen for a la den veta at sendinga er gjort. Når operativ systemet starter opp, så trenger det ikkje å kopiera disk blokko til minne fordi da er der alt. [11]

Fordelen med DMA er at da har høg overføringshastighet, mindre CPU syklusa pr overføring.

Ulempen er at DMA overføring behøver ein DMA kontroller til å bære ut operasjonane, derfor er da dyrare system.

Oppgave 8

Forklar grundig hvordan Interrupts virker og hva som er fordelene/ulempene med det?

Når ein I/O einhet er ferdig med jobben sin så sender den eit signal med bussen linja så den har fått tildelt(viss interupt er aktivert på OSet). Detta signalet blir så oppdaga av interrupt kontroll chipen, som då bestemmer kva som skal gjerast. Viss ingen andre interrupt venter så begynner interrupt

kontrolleren med eingang å behandle interrupten. Derimot viss ein annen interrupt er i gang eller viss ein annen einhet sender ein liknande forespørsel med høgare prioritet så blir einheten med lavare prioritet ignorert foreløpig. I detta tilfelle så fortsetter einheten å senda eit signal heilt til CPUen tar hand om det.

For å handtera interrupts så putter kontrolleren eit nummer på andresse linja for å spesifisera kva einhet som vil ha oppmerksomhet og sender eit signal for å interrupta CPUen. Detta signalet forteller CPUen å stoppa kva enn den gjer og starta med noko anna. Nummeret på adresse linja blir brukt som ei index inn i eit ''table'' som hetter ''interrupt vector'' til å henta eit nytt program teller. Denne program telleren pekker til starten på tilsvarande interrupt service procedure. Traps og interrupt bruker same mekansime fra detta punktet fram over, ofte deller same interrupt vector. Kort tid etter det starter å kjøre så erkjenner interrupt service procedure interrupten ved å skriva inn ein viss verdi inn i interrupt kontrollanes I/O port. Denna erkjenninga forteller kontrolleren at den er ledig til å senda nye interrupt. Ved å ha CPUen utsetta denna erkjenninga fram til den er klar til å ta imot neste interrupt, race condition blir involvert sånn at fleire interrupts ikkje skjer samtidig. Eldre maskiner hadde ikkje ein sentral interrupt kontroller så kvar einhets kontroller sendte inn sinn egen interrupt. [12]

Fordelen med interrupt er at da reduserer ventetida. Uten interrupt så måtte CPUen ha spørt I/O enhetene om da har hendt noko sida sist og da bruker veldig mykje CPU tid og er ekstremt sløsing med resursa, men ulempen er at da kan skapa ein situasjon som heter livelock.

Når ein CPU mottar ein interrupt så gir den kontroll til interrupt kontrolleren, i andre ord så har interrupta absolutt prioritet. Viss interrupta fortsetter og komma, så vil dei ha absolutt prioritet og vil skapa da som hetter livelock, der datamaskinen gjer meir og meir jobb, men gjennomstrømninga er lik null. [13]

Oppgave 9

Hva er prinsippene om at et operativsystem er enhetsuavhengig?

Funksjonene for et enhetsuavhengig operativsystem:

Må ha ett grensesnitt som er universell slik at den kan fungere med alle driverne som eksisterer i et

operativsystem.

Den trenger også en buffering.

En Error Reporting som sperrer en handling hvis en error/umulig handling oppstår i handlingen.

Allokering og løslate dedikerte enheter. Ettersom at en enhet for eksempel en printer kan brukes bare av en prosess på en gitt tid så må OS se igjennom forespørsler for bruk av en enhet og enten akseptere eller avvise disse forespørslene.

Gir en enhetsuavhengig blokkstørrelse. Ettersom at forskjellige disker kan ha forskjellige sektor størrelser så jobber det enhetsuavhengige operativsystemet/software, så den skjuler hvordan sektorene er bygget opp og viser heller en uniform/fast blokkstørrelse. [14]

Uniform interfacing for device drivers
Buffering
Error reporting
Allocating and releasing dedicate devices
Providing a device-independent block size

Bilde hentet fra [14].

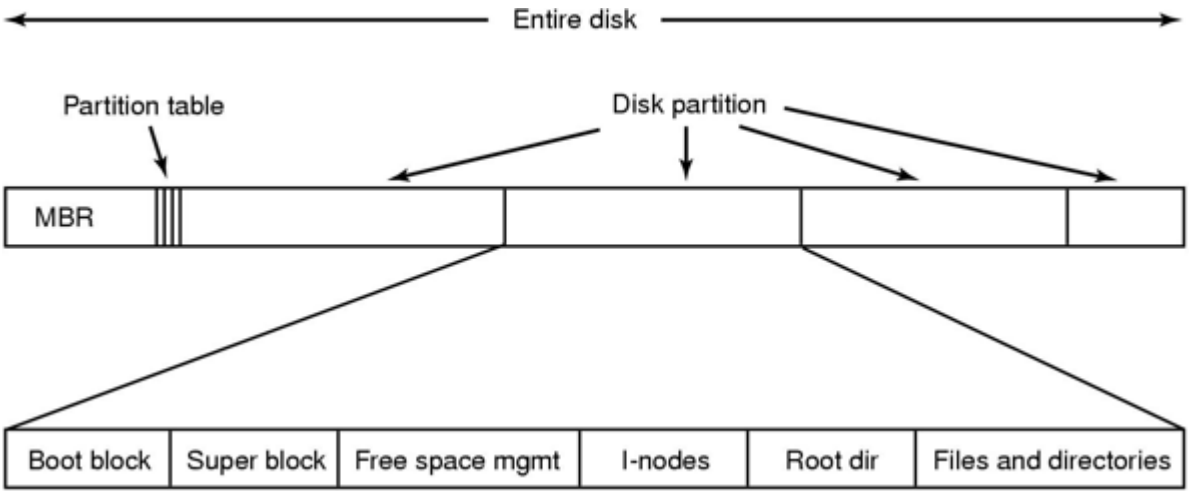
Oppgave 10

Hvordan er filsystemet implementert i en harddisk? Forklar litt om MBR, partisjoner, og hva som er i et filsystem.

Det finnes mange filsystemer, disse er lagret opp i disker. De aller fleste disker kan bli delt opp i partisjoner, disse partisjonene kan ha uavhengige filsystemer på hver partisjon som disken er delt opp i. MBR (Master Boot Record) blir brukt for å starte opp datamaskinen. MBR finnes i sektor 0 eller sektor () av disken. Det finnes et «partition table» i enden av MBR, denne tabellen gir start og ende adresser til hver partisjon. En partisjon i tabellen er markert som aktiv. Når datamaskinen blir skrudd på så leser BIOS inn og eksekverer MBR. MBR sin første oppgave er å lokalisere den aktive delen i partisjonen og lese inn den sin første blokk som heter «boot block» og eksekvere den. [15]

I ett filsystem finner vi MBR, Disk partisjoner, Bootblock, Superblock, Free Space mgmt, I-Nodes,

Root dir, Files and directories. Se figur nedenfor.



Figur hentet fra [16].

Oppgave 11

Forklar hvordan "I-nodes" virker.

En i-node lister opp attributter og diskadresser til en fil. Den inneholder for eksempel filrettighetene og modifikasjonstiden til filen [17] . Med en i-node kan en finne alle blokkene til en fil. Det er en stor fordel ved å bruke i-noder ovenfor «linked files» som bruker «in-memory table». Denne fordelen er at i-noder trenger kun å være i minne når filen den korresponderer med er åpen. Hver i-node har kun plass til X antall disk adresser, hvis en fil vokser seg større enn hvor mye i-noden har plass til så blir den siste adressen en peker som viser til enda en i-node som har flere adresser [16]. [18]

References

[1] A. S. T. &. H. Bos, "Modern Operating System," Pearson Education Limite, 2015, pp. 435-436.

[2] "Wikipedia," 16 11 2019. [Online]. Available: https://en.wikipedia.org/wiki/Deadlock. [Accessed 18 11 2019].

[3] "stackoverflow.com," 7 4 2015. [Online]. Available: https://stackoverflow.com/questions/34510/what-is-a-race-condition. [Accessed 18 11 2019].

[4] Microsoft, "Microsoft.com," 18 6 2012. [Online]. Available: https://support.microsoft.com/en-ca/help/317723/description-of-race-conditions-and-deadlocks. [Accessed 20 11 2019].

[5] "Wkipedia.org," 4 11 2019. [Online]. Available: https://en.wikipedia.org/wiki/Semaphore_(programming). [Accessed 20 11 2019].

[6] "Wikipedia," 4 11 2019. [Online]. Available: https://en.wikipedia.org/wiki/Semaphore_(programming)#Examples. [Accessed 20 11 2019].

[7] A. S. T. &. H. Bos, "Modern Operating System," Pearson Education Limited, 2015, p. 132.

[8] "Wikipedia.org," 16 9 2019. [Online]. Available: https://en.wikipedia.org/wiki/Busy_waiting. [Accessed 21 11 2019].

[9] "wikipedia.org," 12 11 2019. [Online]. Available: https://en.wikipedia.org/wiki/Mutual_exclusion. [Accessed 21 11 2019].

[10] "Wikipedia.org," 23 10 2019. [Online]. Available: https://en.wikipedia.org/wiki/Peterson%27s_algorithm. [Accessed 21 11 2019].

[11] "Modern Operating system," Pearson education limited, 2014, p. 344.

[12] A. S. T. &. H. Bos, "Modner Operating System," pp. 348-350.

[13] E. Kohler, "Hardvar.edu," 25 5 2005. [Online]. Available: http://www.read.seas.harvard.edu/~kohler/class/05s-osp/notes/notes12.html. [Accessed 22 11 2019].

[14] H. Bos and A. S. Tanenbaum, "Modern Operating Systems," Pearson Education, 2015, pp. 361-367.

[15] "Modern Operating System," Pearson Education, 2015, pp. 281-282.

[16] H. Øysæd, *08_DAT_103_Filsystemer*, 2017.

[17] Wikipedia, "inode," Wikipedia, 28 Desember 2015. [Online]. Available: https://no.wikipedia.org/wiki/Inode. [Accessed 22 November 2019].

[18] H. Bos and A. S. Tanenbaum, "Modern Operating System," Pearson Education, 2015, pp. 286-287.