Eksamen Opsys 2019

**Kva er eit Operativsystem og kva er hovud funksjonanen i eit Operativsystem?**

Eit Operativsystem er hovudprogrammet som styrer hardware og software. Operativsystemet har kontroll over filer, minne, prosesser og enheter. Operativsystemet er som ein mellommann mellom hardware og software. OS gir også eit elegant og fint interface som er enkelt og bruke sånn at brukarane slepper å sjå og bruke dei tunge kodane som går i bakgrunnen.

I moderne OS har ein to modusa, Kernel- og Bruker-Modus.

I Kernel modus har ein tilgang til alle OSane sinne resursa.

I Bruker modus må dei kjørande programma ta eit ‘’systemkall’’ der prosessane sender ein forespørsel for å få tilgang til resursane og tid på CPUen.

**kva er ein extended machine?**

OS holder styr på komponentar og gir applikasjonsprogram tilgang til desse komponentane. Hardware i datamaskinar er kompliserte og er unødvendig og læra for dei fleste og OS skjuler desse kodane som er vansklige å jobba med, og gir oss eit enklare interface å jobba med og gir meir oversiktlig data.

**Kva er recourse managment?**

OS sinne oppgave er å halda styr på datamaskinens prosessar for å unngå kaos. Viss eit program bruker ein resurs på maskina så må eit program som vil ha tilgang til same resurs sende ein forespørsel og vente på sin tur. Ei av hovud oppgavene til OS er og halde styr på kva program som har tilgang og bruker kva resurs til ein kvar tid, innvilga forespørsla, å ta hensyn til bruken og redusera mulige konflika.

**Kva er Monolithic Structure av OS?**

Heila OS jobber i Kernel mode i da Monolithic system. Detta aukar størrelsen på kjernen samt OSet. Kjernen gir forskjellige tjenester som f.eks. minne administrasjon, file administrasjon, prosess scheduling, osv. ved bruk av function call. detta gjer at kjøring av OS er raskt fordi tjenestane er implinitert under da same adresse område.

**Kva er Microkernel?**

Microkernel er da nær minimum beløp av software som kan gi mekanismane som trengt for og implementera eit OS. Bruker tjenesten og kernel tjenesten er implementert i forskjellige adresse områder. Bruker tjeneste er under bruker adresse område og kernel tjenestene er under kernel adresse område, derfor reduserer det kernel og størelsen på OSet.

**Kva er prosesser?**

Ein prosess er eit datamaskins program under kjøring av ein eller fleire tråder. Da inneheld programmets koder og aktiviteta. Spørs på OSet, ein prosess kan vær laga av fleire tråder som kjører instruksjonane i samtidg

Ein prosess er eit program under eksekvering og bygger på prinsippa om resursgruppering (minneområda, bruk av ressursa, filer, osv.) og eksekvering (den delen der prosessen faktisk blir kjørt på CPUen). Moderne datamaskina har fleire prosessa i sitt system for å auka effektiviteta. Alle prosessa har minst ein tråd, som er den delen av prosessen som kjører på CPUen. For å kallas ein prosess og bli kjørt på CPUen må nokre ting væra på plass, det som definerer prosessen, nemlig PCB(process control block). Denna tabellen inneheld informasjon om prosessen som når den blei oppretta, prioritert, child- og parent-prosesser, kor mykje tid den har fått på CPUen, alarms og signals, samt peikara til minneområde og stack osv. Assosiert med kvar prosess er dens adresse område ei lista av minne områder fra 0 til maksimum, som prosessen kan lesa og skriva.

For å implementera prosess modellen så har OS ein matrise av strukturer som er kalt process table med ein inngang per prosess. desse inngangane inneheld viktig informasjon om prosessens tilstand, inkludert program counter, stack pointer, memory allocation, tilstand til dens åpne filer, dens regnskap- og planleggingsinformasjon, og alt anna om prosessen som må lagra når prosessen bytter frå, running til ready eller blocked så kan da starta opp igjen som om da aldri stoppa.

**Kva er multiprogrammering?**

Multiprogrammering er eit system som støtter fleire prosessora i ein datamaskin. Da gir ein illusjon som om at datamaskinen multitaske, men sanninga er litt annerledes. Ein prosessor kan KUN gjera ein oppgava omgongen, men desse oppgavene blir utført så utrolig raskt alt fra eit par millisekund til hundredelsmillisekund noko som gir eit utrykk om parallellitet. For mennesker så er parallelle aktiviteta vansklig og sjå. Derfor er OS designa opp over åra sekvensielle prosessa som gjer parallellisme enklar å ha med og gjera.

**Kva er ein Thread?**

Ein Thread eller tråd på norsk er ein komponent i ein prosessor. Antalle tråder i ein prosessor bestemmer kor mykje ting den kan gjera, ein tråd kan kun gjera ein ting, men da er mulig å ha fleire tråder i ein prosessor noko som gjer at den kan gjera fleire ting. Ulikt prosessora som har ulike resursa når ein har fleire så må tråder dela resursen dei får. Tråder og prosessora deler same resursa , men ein prosessor inneheld fleire resurser enn tråder. Problemet med tråder er at dei er ikkje beskytta.

Tråder er eit uavhengig sett med verdier for prosess registeret for ein enkel core. Siden detta inkluderer program counteren, så vil da kontrollera kva som skal kjørast i kva orden. Da inkluderer også stack pointer, som har bedre punkt til eit unikt area av minne for kvar tråd ellers vil dei begynnar og forstyra kvarandre.

**Kva er prosess administrator?**

Prosess administrator eller process manager er den delen som gir resurser til prosessorane.

**Kva er eit register?**

Eit register er eit lite sett av data som holder plasser som er ein del av prosessoren.

**Kva er program counter?**

Program counter indikerer kor datamaskinen er i sekvensen.

**Kva er program status word?**

Program status word er ein samling av informasjon av statusen til programma.

**Kva er Stack pointer?**

Stack pointer er eit lite register som lagrar adressene til dei siste programforespørsla i ein stack.

**Kva er ein process state?**

Ein process state indikerer kva state prosessen er i.

**Kva er Scheduling parameters?**

scheduling parameters er rammene som settes for planleggeren.

**Kva er parant process?**

er namnet til ein prosess som lagar ein anna prosess.

**kva er signal?**

Signal er ein påminnelse til ein prosess at ein heldelse har skjedd.

**Kva er Childrens CPU time?**

Er tida prosessen som er laga av parant process får på CPUen.

**Kva er Time of next alarm?**

Er tida til neste alarm.

**Kva gjer memory managment?**

Memory managment håndterer rammen.

**Kva er file managment?**

Da er eit program som bruker eit grensesnitt for å håndtere filer.

**Kva er Root directory?**

Da er da nederste nivået i filsystemet og alt kjem får root.

**Kva er Working directory?**

Da er der du er i filsystemet.

**Kva er file directory?**

Da er ein beskrivelse av ei fil.

* User Id er da brukeren har tilgang til
* Group Id er da gruppa har tilgang til.

**Kva er Kernel level Thread?**

Kernel level thread er treigare og ineffektiv. Kernel må halda styr på prosesser og tråder. Den treng ein full thread control block for kvar tråd. I user level tråd er det andre problem.

**Kva er Context Switch?**

Er når ein prosess bytter state mellom ready, blocked og running. Detta gjer at prosessorane kan multitaska. For å skilla mellom tråder og prosesser kan me kalla den Thread Switch og process Switch. Thread switch bytter mellom tråder og process switch mellom prosesser.

**Kva er Atomic Operation?**

Da er ein prosess som kan lesa og skriva til minnet samtidig på samme buss operasjon. Detta gjer at andre prosessa og enheter ikkje kan bruka samme minne.

**Kva er Busy waiting?**

Er når OS gjentatte ganger leiter etter at eit kriterie skal bli møtt. Da vil seie at den venter på at ein heldelse skal skje.

**Kva er Spinlock?**

Da er ein type lås som bruker Busy waiting.

**Kva er Semafor?**

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

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

**Kva er Mutex?**

Når semaforens evne til å tella ikkje lenger er bruk for, så blir ein enklar 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 trådpakker som er fullstendig implementert i bruker område.

**Kva er Deadlock?**

Deadlock er når to eller fleire prosessa prøver og få tilgang til den samme resursen, men ingen kommer noko vei. Da skjer når kvar av prosessane har tilgang til kvar sin bestemt del av resursen, men trenger tilgang til resten av resursen til den andre parten for å gjennomføra prosessen, dermed kommer dei ingen vei og prosessen blir blokkert for alltid.

**Kva er dei 4 betingelsene for ein Deadlock?**

* Kvar resurs er enten tilordna til ein prosess eller er tilgjengelig for bruk
* Prosessen som allereie holder på resursar som den tidligare fekk tilgang til, kan også spørra om nye resursa
* Resursa som prosessen tidligare har fått tilgang til, kan ikkje tas vekk frå ein prosess. prosessen kan kun gi frå seg resursen for at andre skal kunne ta resursen i bruk.
* Da må vær ei lista med 2 eller fleire prosessa som venter på ein resurs som allereie er i bruk av den neste i rekka

Når alle desse betingelsane er sanne så kan ein resurs Deadlock skje, men andre detaljer er også viktige. F.eks. om ein resurs er laget for å kunne bli brukt av fleire prosesser samtidig.

**Kva er Race Condition?**

Da er ein uønska situasjon som skjer når ein einhet eller system prøver og utføra 2 eller fleire operasjona samtidig. 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 er ferdig. Så begynner ein ny prosess og oppdaterer denne verdien. Så når den originale prosessen då starter opp igjen der den blei interrupta så vil den sletta den andre prosessens verdi som då gjer at den blir sittande å venta på ein output som aldri kjem.

**Kva må ein gjere for å unngå Deadlock og Race Condition?**

Det er 4 betingelsa som må bli møtt.

* Fleire prosessar kan ikkje jobba samtidig i kritiske seksjona
* Man kan ikkje anta noko om hastigheten eller antall CPUer
* Ingen prosessa som jobber å utsida av sin kritiske seksjon kan blokkera andre prosessar
* Ingen prosessa skal måte venta evig for å få tilgang til ein kritisk seksjon.

Da er vansklig å finna fram til problemet. Fordi deadlock og race condition kan skje når som helst. Så sjølv om du tester systemmer/programmet tusenvis av ganger kan det hende at du aldri klarer å gjenskape problemet.

**Korleis fungerer DMA?**

Direct Memory Access fungerer med at da først programmerer CPUen DMAen med å setta registeret sånn at da veit kva som skal kor(1). Den gir også ut kommandoer til disken og forteller den til å lesa data frå disken inn i interne bufferen. Når gyldig data er i disk kontrollerens buffer så kan DMAen begynna. DMAen starter med å senda ‘’read’’ forespørsla over til bussen til disken(2). Denna ‘’read’’ forespørselen ser ut som alle andre ‘’read’’ forespørsla og disken veit/bryr seg ikkje om da kommer frå CPUen eller DMAen. Typisk så skriver minne adressen til buss adresselinja så når disken henter da nesta ordet frå sin interne buffer så veit da kor da skal skrivast(3). når skrivinga er ferdig så sender disken ein bekreftelse signal til DMAen(4). DMAen aukar så minne adressa som skal brukast og minker byte tillinga. Viss byte tellinga er størren enn 0 så går ein gjennom 2 til 4 igjenn heilt til tellinga når 0. Når da skjer så avbryter DMAen CPUen og forteller den at sendinga er gjort. Når OS starter opp, så treng det ikkje å kopiera frå disk blokka til minne for da er alt der.

Fordelen med DMA er at da har høgare overførings hastighet og mindre CPU syklusa pr overføring, men ulempen er at DMA overføringer krever ein DMA kontroller noko som gjer at systemet er dyrare.

**Korleis fungerer ein interrupt?**

Når ein I/O einhet er ferdig med jobben sin så sender den eit signal med buss linja så den har fått tildelt. 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 og behandlar interrupten. Derimot viss det er andre interrupt som er i gang eller viss ein anna einhet sender eit liknanade forespørsel med høgare prioritet så blir einheten med lavare prioritet ignorert foreløpig. I detta tilfellet så fortsetter interrupten med lav prioritet å senda signal heilt til CPUen tar hand om det.

Fordelen med interrupts er at da minker vente tida. Uten interrupt så måtte CPUen ha spørt I/O einhetene om da har hendt noko sida sist og da bruker veldig mykje CPU tid og er ekstremt stor sløsing med resursar, men ulempen er at da skapar ein situasjon som heter livelock

**Korleis er filsystemet implementert i ein harddisk?**

Disken blir delt opp i fleire sektora der sektor 0 av disken er kalla Master Boot Record(MBR) og blir brukt til å starta opp maskina. Slutten av MBR inneheld partisjons bord. Bordet gir start og slutt adressene av kvar partisjon og ein av partisjonanen er markert som aktiv. Når maskina blir skurd på, starter BIOS MBRen. Da førsta MBRen gjer er å finna den aktive delen av harddisken, den leser delen inn i den førte blokka som kallas BOOT block, og kjører den. Programmet i blokka lastas opp i OSet som er lagret i den aktive partisjonen. Alle partisjona startar med ein Boot block. Da filsystemet inneheld varierer frå system til system. Mange filsystem innehelder fleire deler som f.eks. superblock. Superblock inneheld vanligvis eit nummer til å identifisera fil system typen. Antall blokker i systemet og andre administrerande informasjon. Vidare frå dei andre blokkene kjem det ting som liste av pointers. Så kjem det kanskje i-nodes. Så root directory som inneheld toppen av fil system treet. Og til slutt kjem resten av disken som inneheld resten av filene.

**Kva er I-nodes?**

I-nodes er ein metode som held styr på kva blokker som tilhører filer, og kvar fil struktur kan kallas ein index node, som laga reit skjema over egenskapene og disk adressene til blokkene. Fordelene med denna oversikta over dei linkede filene er at i-nodene må kunna væra i minne når den tilsvarande filene er åpne. Viss kvar i-node tar i bruk x antall bytes og ein maksimum mengde y filer kan vær åpne samtidig, så vil det totala minnet som blir brukt kun være xy bytes og det er kun så mykje plass som må bli reservert på forhand.

Når ein lagrer informasjon på ein harddisk så lagrer man vanlig viss filene på ei rekke, men viss ein vil endra på ei tidligare fil så kan ein ikkje bare skrive over informasjonene vedsiden av fila. Så då lagrer ein den nye fila på slutten av lenka. Da er her I-nodes kjem inn. Me bruker I-nodes til å halda styr på filer som detta. Disken er delt opp inn i mange blokker og viss ei fil er større enn ei blokk så blir filene fordelt over fleire blokker og kvar blokk har ein I-node som fortell kva filer høyrer til kor. Kvar i-node er i samme størrelse og ligg i superblokka. I-noden lagrar data når filene blei lagra kven lagra filene osv. I-node