Informasjon om kandidat/gruppe

Kurs: IKT102

Øvingsoppgave: Prosess Øving 2 for opsys

Dato: 30/09/2019

Kandidat/gruppe:

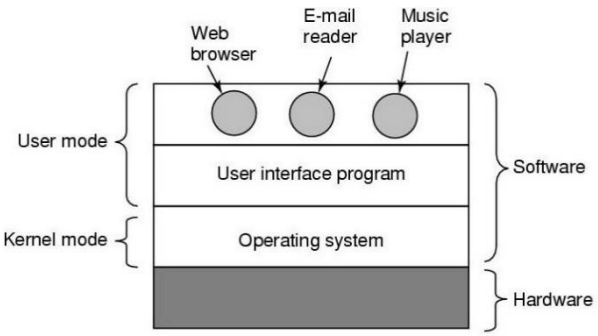
Oppgave 1

Svar 1.1

Hva er et operativsystem og hva er hovedfunksjonen i et operativsystem?

[2]Da er vansklig å sei kva eit operativsystem er utan å seie at det er eit software som kjører i Kernal mode og sjølv da er ikkje alltid sant. Deler av problemet er at operativsystemet utfører to essensielle urelaterte funksjonar: tilby applikasjonsprogrammerere eit reint sett av ressursar istaden for hardware sine rotet og administrera disse hardware ressursane[2].

[1]Eit operativsystem sin jobb er å gi brukerprogram eit betre, enklare, finare, model, av datamaskinen og å håndtering administrering av ressursa.



Figur 1.1

Her ser ein enkel oversikt over hovuddelen.

Me ser hardware på bunnen. hardware består av «chips», disks, tastatur, skjerm, og andre liknande fysiske object.

Over hardware liger software.

Mesteparten av datamaskina har 2 driftsmodusa: Kernal mode og User mode.

Operativsystemet, da mest fundamentale delen av software, blir kjørt i Kernal mode. I Kernal har komplett tilgang til all hardware og kan kjøra alle instruksjonane som maskina taklar å kjøra. Resten av software ligg i User mode, der kun ein undergruppa av maskinen sine instruksjona er tilgjengelig. Spesielt dei instruksjonane som påvirke kontrollen av maskina eller gjer I/O er forbudt til User mode program.

Ein viktig forskjell mellom OS og normal(User mode) software er at viss ein brukar ikkje liker eit spesifikt email system så kan ein skaffa seg eit anna system eller laga eit. Ein kan ikkje skriva sitt eigen klokke avbryter behandler, som er ein del av OS og er beskytta av hardware mot forsøk av brukaren til å modifisera da. Denna distinksjonen er av og til litt uskarp in ein del av systemer(som kanskje ikkje har kernal mode) eller «Interpreted» systemer(som java-basert systemer som også bruker «interpretation», ikkje hardware, til å separera komponenta). I tillegg er da mange systemer da er programmer som kjører i user mode, men hjelper OS eller utfører privilegert

funksjonar. F.eks. da er ofta program som tillater brukar og skifta passord. Da er ikkje ein del av OS og kjører ikkje i kernel mode, men bærer tydelig ut sensitive funskjonar og må være beskytta på ein spesiell måte. [1]

[3] Arkitekturen(bus strukturen, I/O, instruksjonssett og minne organisator) til meste parten av datamaskina i maskin-språket er primitivt og pinlig å programmera, spesielt for input/output. for å gjera detta poenge meir konkret, ta i betraktning moderne SATA harddiska som er brukt i meste parten av datamaskina. Grensesnittet har blitt revidert mange ganger og er meir komplisert enn da var. Eit software programmet ''disk driver'' har med hardware og gir eit grensesnitt til å lesa og skriva disk blokker, utan å gå inn i detaljer.

OS inneheld mange drivere for I/O, men til og med detta nivået er for lavt for meste parten av applikasjonar. Av denna grunnen gir OS alle enda eit lag med abstraksjon for å bruka diska: filer. Ved hjelp av denna abraksjonen, program kan laga, skriva og lesa filer, utan å måte takle med det rotete detaljane av korleis eit hardware faktisk fungerer.

Denna abstraksjonen er nøkkelen til å administrera all denna kompleksiteten. Goode abstaksjonar omgjer nesten umulige oppgaver om til 2 håndterbare. Den første er å definera og implimintera. Den andre bruker desse abstraksjonane til å løysa problema for hand.

Prosessor, minne, disk og andre einheter er veldig kompliserte og presenterer eit vansklig, pinlig og inkonsekvent grensesnitt for brukara som må skriva eit software til å bruka dei. Nokre gonger er det på grunn av behov for bakover kompatibilitet med eldre hardware. Ofte så forstår ikkje hardware designeren kor mykje trøbbel dei forårsaker for software. Ein av dei store oppgavane til OS er å gjemma hardware og presentera program med fine, reine, elegante, konsistente, abstraksjona til å virka i stadenfor.[3]

[4] konseptet av eit OS som først og fremst gir abstraksjonar til applikasjons program er eit topp-ned visning. Eit alternativ er bunn-opp visning hold at OS er der til å administrere alle delane til eit komplekst system. Moderne datamaskina består av prosessor, minne, tidtaker, disk, osv. I bunn-opp, visning, jobben til OS er å gi ein ryddig og kontrollert tildeling av prosessoren, mine og I/O blant dei diverse programma som vil ha dei.

Moderne OS tillater mange program til å vær i minne og bli kjørt samtidig. Sjå for deg at 3 program prøver å skriva ut samtidig til den same printeren. Dei første linjene er kanskje fra det første programmet, dei neste er fra det andre programmet, så nokre linjer fra det tredje programmet, osv. resultatet ville blitt eit kaos. OS kan gi ein orden i da potensiella kaoset med å buffra alle «output» som skal til printeren frå disken. Når eit program er ferdig så kan OS då kopiera «output» fra disken file der da har blitt lagra for printeren, medan det første programmet kan fortsetta å generera meir «output» uvitande til at «output» faktisk ikkje går til printeren(enda).

Når ein datamaskin har fleire brukara blir behovet for administrering og beskytta minnet, I/O einheter og andre resursa er større sidan brukarane kan ellers forstyrra med kvarandre. I tillegg er brukara ofte nødt til å ikkje bare dela hardware, men også informasjon også.

Ein kortoppsummering, detta synet av OS holder da at primære oppgave er å halda auga med kva program som køyrer som bruker kva resurs, til å gi resursforespørsmål, til å redegjøra for bruk og til å mekla konflikt forespørsel fra forskjellige program og brukara.

Resurs administrering inkludere multiplexing(deling) resursa i 2 forskjellige veier. Når ein resurs er «time» multiplexed, forskjellige program eller brukara tar tur for å bruka da. Først eit av programma får bruka resursa så eit anna, osv. F.eks. når ein kun har 1 CPU og fleire program som vil bruka den, så allokerer OS CPUen til eit program så etter det har kjørt ei stund så får eit anna program tilgang til å bruka CPUen så når da er ferdig får eit anna program tilgang og tilslutt får da første programmet tilgang igjen. Bestemma korleis resurs er «time» multiplexed «kven som får gå og kor lenge dei får gå» er ei av oppgavene til OS.

Den andre typen av multiplexing er «space» multiplexing. I staden for at ''kundane'' tar tur, kvar og ein av dei får dela av resursa. F.eks. hovud minne er normalt delt opp blant fleire kjørande program, så kvar og ein kan vær ''bosatt'' samtidig. Forutsatt da er nok minne til å multiprogrammera, da er meir effektivt og da held fleire program i minnet samtidig, enn å gi ein av dei alt av da, spesielt viss da kun trenger ein liten del av da totala. Sefølgelig reiser detta spørsmålet om rettferdighet, beskyttelse og så vidare, og da er opp til OS å løysa detta. Ein anne resurs som er «space» multipexed er disk. I mange system ein enkel disk kan halda flier fra fleire brukara samtidig. Allokering disk plass og halda auga med kven som bruker kva disk block er typisk OS oppgava.[4]

Oppgave 2

Svar 2.1

Forklar grundig hva en prosess er og hva en prosess gjør i et operativsystem?

Ein prosess er eit program under eksekvering og bygger på prinsippa om ressursgruppering (minneområde, 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 øke effektiviteta. Alle prosesser har minst ein tråd, som er den delen av prosessen som kjøre på CPUen. For å kunne 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 innehelde informasjon om prosessen som når den blei oppretta, prioritert, child- og parent-prosesser, hvor mykje tid den har fått på CPUen, alarms og signals, samt peikara til minneområda og stack, osv [10] Assosiert med kvar prosess er dens ''address space'' ei lista av ''memory locations'' fra 0 til eit maksimum, som prosessen kan lesa og skriva.

Adresse plassen innheld kjørbare program, programmas data og dens ''stack''. I tillegg assosiert med kvar prosess er eit sett med resursa, vanligvis inkludert register, ei lista av åpne filer, lista av relaterte prosessa og all anna informasjon som trengs for å kjøra eit program.[10]

Ein god start for prosessa er og tenka på multiprogrammerings system.

[5]Sjå for deg ein bruker PC som starter opp, mange prosessar starter opp, mange som ikkje er kjent til brukeren. F.eks. ein prosess starter opp for email, ein anna kjører antivirus program og sjekker om det er komme nye oppdateringa. I tillegg kan da henda brukeren printer ut noko, samtidig som brukeren surfer på nettet. Alle desse aktivitetane må bli administrert og eit multiprogrammerings system som støtter multiple prosessora kommer veldig til hjelp her.

I multiprogrammering system så bytter CPUen fra prosess til prosess veldig fort. Kjører kvar for ti millisekund til hundremillisekund, men strengt tatt så kjører CPUen kun 1 program om gangen, men på 1 sekund så kan da jobba på fleire program, noko som gir inntrykk av parallellitet. Folk snaker om ‘’pseudoparallelism’’ in denna konteksten, i kontrast med den sanne hardware parallelliteten til multiprosessor systemet(som har 2 eller fleire CPUa som deler same fysiske minne). Halda auga med fleire parallelle aktiviteta er vansklig for folk og gjera. Derfor er OS designa opp over åra sekvensiell prosessa som gjer parallellismen enkler å ha med å gjera.[5]

[9] for og 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 alloctarion’’, tillstanden til dens åpne filler, dens regnskap- og planleggingsinformasjon, og alt anna om prosessen som må bli lagra når prosessen bytter fra ‘’running’’ til ‘’ready’’ eller ‘’blocked’’ så da kan bli starta opp igjen som om da aldri stoppa.[9]

Oppgave 3

Svar 3.1

Forklar grundig hva en tråd er og hva en tråd gjør i et operativsystem?

[13] Ein tråd er dei minste sekvensielle av eit programmert instruksjon som kan bli administrert av ein planlegger, som er typisk ein del av OS. Implementeringen av ein tråd og ein prosess varierer mellom operativsystema, men i fleste tilfeller er ein tråd ein komponent av ein prosessor.[13]

[11] I tradisjonell OS har kvar prosess eit adresseområde og har ein enkel tråd som kontrollerer. I mange situasjoner er da ønskelig å ha fleire tåder som kontrollerer i same adresseområde som kjører ‘’quasi-parallel’’ som om dei er separerte prosessa(utenom dei som har delt adresseområde).[11]

[12]Hovud grunnen til å ha tråder er at i mange applikasjoner, så er det mange aktiviteter som skjer om gangen. Med å bryta ned disse applikasjonane in i fleire sekvensielle tråder som kjører ‘’quasi-parallel’’, så blir programmets model enkelt.

Det andre argumentet for å ha tråder er sidan dei er lettare ein prosessa og dei er enklare(raskare) og laga og øydelegga enn prosess. I mange system laging av tråder går fra 10-100 ganger raskere enn å laga ein prosess. når mengden av tråder som trengs forandrer seg konstant så er denne egenskapen god å ha.

Ein tredje argument er eit ytelseargument. Tråder gir ingen ytelse når alle er knytta til CPUen, men når da er betydlig databehandlig og I/O så ved gi trådane tillatelse til og overlappe disse

aktivitetane så blir applikasjonane raskare.[12]

Oppgave 4

Svar 4.1

Forklar kort hva de forskjellige emnene i tabellen gjør/hva de er for noe?

Prosess administrering:

Er ein integral del av moderne OS. OS må allokera resursa til prosessen, tillatta prosessen å dela og utveksla informasjon, beskytta resursane for kvar prosess fra andre prosessa og aktivera synkronisering mellom prosessane. For å møta dissa krava så må OS oppretthalda ein data struktur for kvar prosess, som beskriver tilstanden og resurs eigerskap av prosessen og som tillatter OS til og utøva kontroll over kvar prosess.[14]

Minne administrering:

Dela av OS som administrere minne hierarkiet er kalt minne administrering. Dens jobb er å effektivt administrera minne: halda eit auga med kva del av minnet er i bruk, allokera minnet til prosessoren når dei trengs og deallokera da når dei er ferdig.[6]

Fil administrering:

Er eit dataprogram som gir eit brukergrensesnitt til å administrera filer og folders. Det vanligaste operasjonen som er gjort på filer eller folders er laging, åpning, gi nytt namn, flytta eller kopiera, sletta og søking av filer, i tillegg modifisera fillers egenskap og fil tillattelse. Folders og filer blir vist i eit hierarki tre basert på deirast katalogstruktur.[15]

Oppgave 5

Svar 5.1

Er det noen problemer med å implementere tråder i bruker modus? Kjerne modus? Diskuter dette.

Da er to hovud plassa til å implementera tråda: bruker modus og kjerne modus. Valget er litt kontroversielt og ein hybrid implementering er og mullig.

[17]Når me tar heile tråd pakken i bruker modus så veit kjernen ingenting om dei. Kjernen trur den administrerer enkle single-threads prosessa. Den åpenbare fordelen med detta er at ein bruker-nivå tråd pakke kan bli implementert i eit OS som ikkje støtter tråder.

Eit problem med bruker-nivå tråder er viss ein tråd starter å kjøra, så kan ingen andre tråder i den prosessen kjøre utenom viss den første tråden frivillig gir opp CPUen.

Eit anna problem og kanskje det største argumentet mot buker-nivå tråder er da at program

generelt vil ha tråder presist i applikasjonen der tråden blokkerer ofte. Desse trådene foretar kontinuerlig systemkall. Når ei fella har oppstått i kjernen for å gjennomføra systemkall, så er da knapp noko jobb for kjernen å skifta tråder viss den gamle er blokkert, og å ha kjernen stå for denna eliminere for kontinuerlig gjør velgte systemkall som sjekker om systemkall er sikre. For applikasjonar som er essensielt kun CPU-bound og sjeldent blokkert så kan ein spør seg kva poenget med tråder er.[17]

[18]No la oss sjå litt på viss kjernen veit om og administrerer trådene. Da er ingen tråd tabell i kvar prosess. I staden for så har kjernen ein tråd tabell som holder auga med alle trådane i systemet.

Medan kjernen løyser mange problem så blir ikkje alle løyst. F.eks. kva skjer viss ein multithreaded prosess splitter? Har den nye prosessen lika mange tråder som den gamle hadde eller har den bare 1? i mange tilfeller så avhenger det beste tilfelle av kva prosessen ska gjer etterpå. Viss den skal få exec til å starta eit nytt program så er sannsynligvis 1 tråd da korrekta svaret, men viss den skal fortsetta å kjøra, så er reprodusering alle trådane best.

Eit anna problem er signal. Husk at signal er sendt til prosessen og ikkje til trådane, i da minsta i dei klassiske modelane. Når eit signal kommer inn, kva tråd skal ta seg av da? Tråder kunne registrer deiras interesse i visse signal, så når eit signal kommer in så ville da bli gitt til tråden som seier da vil ha da. Men kva skjer når 2 eller meir tråder registrerer seg for da sama signalet? Detta er kun 2 av problema tråder introduserer og da er mange andre.[18]

Oppgave 6

Svar 6.1

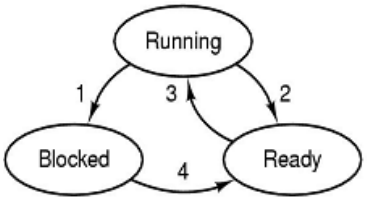
Kontekstbytte, hva er det? Forklar?

Kontekst bytte er ein prosess av lagra tilstanden av ein prosess eller ein tråd, så da kan bli gjennopptatt og kjøring forsetter fra same plassen seinare. Detta tillater fleire prosessa til å dela ein CPU og er ein essensiel del av multitasking OS.[8]

Oppgave 7

Svar 7.1

Se på de 3 tilstandene en prosess/tråd kan være i figur 1. Forklar figuren(figuren på tavlen med wait queue, run queue and CPU er tilsvarende figur med forskjellige navn)



1. Process blocks for input
2. Scheduler picks another process
3. Scheduler picks this process
4. Input becomes available

ein "ready" prosess har blitt lasta inn I hovud minnet og venter på å bli kjørt på CPUen.

Ein prosess beveger seg inn I ‘’running’’ når da er valgt til å bli kjørt. Prosessens instruksjon er kjørt av ein av CPUane i systemet.

Ein prosess beveger seg inn I ‘’blocked’’ når da ikkje kan fortsetta vidare utan ein ekstern forandring i tilstanden og ein heldelse som skjer. F.eks. ein prosess kan blokkera ein anrop til I/O einheter some in printer, viss printeren ikkje er ledig.[7]

Oppgave 8

Svar 8.1

Hva er forskjellen på in “compute bound” prosess og en ”I/O bound” prosess? Hvordan vil dette påvirke prosess planleggingen

Eit program er CPU-bound viss da ville gått raskare viss CPUen var raskare f.eks. da bruker mesteparten av sin tid med å bare bruka CPUen(kalkulering). Eit program som regner nye tall for pi vil vanligvis vær CPU-bound, da kun regner tall.

Eit program er I/O-bound viss da ville gått raskare viss I/O delsystem var raskare. Nøyaktig kva I/O system som er meint kan variera. Typisk er da assosiert med disk, men kan selfølgelig vær netverk eller kommunikasjon. Eit program som ser gjennom store menger med filer for noko data kan være I/O-bound, sidan flaskehalsen er då lesing av data fra disken. [16]

Vist me tenker me har ein CPU-bound prosess og mange I/O-bound prosessa. Når prosessen starter så kan da følgande senario kan skje. CPU-bound prosess vil få og ha CPUen. Mens detta skjer, så vil alle dei andre prosessane gjer ferdig deira I/O og vil bevega dei inn i klar køen, og venta på CPUen. Medan prosessen venter i køen, så vil I/O einhetane vær inaktive. Eventuelt så vil CPU-bound prosessen gjer ferdig sin CPU utbrudd og bevega seg in i I/O einheta. Alle I/O-bound prosessa, som er har små CPU utbrudd, vil kjøra rask og bevega seg tilbake i I/O køen. På detta punktet sitter CPUen inaktiv. CPU-bound prosessen vil så bevega seg tilbake i klar køen og bli allokert i CPUen. Igjen så vil alle I/O prosessa enda opp med å venta i klar køen fram til CPU-bound prosessen er ferdig. Da er ein konvoi effekt medan alle dei andre prosessane venter på den eine store prosessen til å bli ferdig og komma seg ut av CPUen. Denna effekten resultere i lavare CPU og andre einhet utnyttelse som kan mulig viss den mindre prosessen hadde fått lov til å gått først.[19]

Referanser

[1] Andrew S Tanenbaum og Herbert Bos, “1 Introduksjon” i Modern Operating System, 4. utgave. Nederland: Pearson, 2015, kap. 1, sider. 1–3.

[2] Andrew S Tanenbaum og Herbert Bos, “1.1 What is an Operating System” i Modern Operating System, 4. utgave. Nederland: Pearson, 2015, kap. 1.1 , sider. 3–4.

[3] Andrew S Tanenbaum og Herbert Bos, “1.1.1 The Operating system as an extended machine” i Modern Operating System, 4. utgave. Nederland: Pearson, 2015, kap. 1.1.1 , sider. 4–5.

[4] Andrew S Tanenbaum og Herbert Bos, “1.1.2 The Operating system as a Resource Manager” i Modern Operating System, 4. utgave. Nederland: Pearson, 2015, kap. 1.1.2 , sider. 5–6.

[5] Andrew S Tanenbaum og Herbert Bos, “2.1 Processes” i Modern Operating System, 4. utgave. Nederland: Pearson, 2015, kap. 2.1 , sider. 85–86.

[6] Andrew S Tanenbaum og Herbert Bos, “3 Memory management” i Modern Operating System, 4. utgave. Nederland: Pearson, 2015, kap. 3 , sider. 181.

[7] wikipedia ‘’process state’’ 03/10/2019 15:30.

https://en.wikipedia.org/wiki/Process_state#Primary_process_states

[8] wikipedia ‘’context switch’’ 03/10/2019 15:20.

https://en.wikipedia.org/wiki/Context_switch

[9] Andrew S Tanenbaum og Herbert Bos, “2.1.6 Implementation of processes” i Modern Operating System, 4. utgave. Nederland: Pearson, 2015, kap. 2.1.6 , sider. 94.

[10] Andrew S Tanenbaum og Herbert Bos, “1.5.1 Processes” i Modern Operating System, 4. utgave. Nederland: Pearson, 2015, kap. 1.5.1 , sider. 39.

[11] Andrew S Tanenbaum og Herbert Bos, “2.2Threads” i Modern Operating System, 4. utgave. Nederland: Pearson, 2015, kap. 2.2 , sider. 97.

[12] Andrew S Tanenbaum og Herbert Bos, “2.2.1 thread usage” i Modern Operating System, 4. utgave. Nederland: Pearson, 2015, kap. 2.2.1 , sider. 97-98.

[13] wikipedia ‘’Threads(Computing)’’ 04/10/2019 15:30

https://en.wikipedia.org/wiki/Thread_(computing)

[14] wikipedia ‘’Process management(computing)’’ 04/10/2019 15:40

https://en.wikipedia.org/wiki/Process_management_(computing)

[15] Wikipedia ‘’File management’’ 04/10/2019 15:50

https://en.wikipedia.org/wiki/File_manager

[16] stackoverflow ‘’ What do the terms "CPU bound" and "I/O bound" mean?’’ 04/10/2019 16:05
https://stackoverflow.com/questions/868568/what-do-the-terms-cpu-bound-and-i-o-bound-mean

[17] Andrew S Tanenbaum og Herbert Bos, “2.2.4 Implementing Threads in the User space” i Modern Operating System, 4. utgave. Nederland: Pearson, 2015, kap. 2.2.4 , sider. 109-111.

[18] Andrew S Tanenbaum og Herbert Bos, “2.2.5 Implementig Threds in the kernel” i Modern Operating System, 4. utgave. Nederland: Pearson, 2015, kap. 2.2.5 , sider. 111-112.

[19] wikipedia ‘’I/O bound’’ 04/10/2019 20:50

https://en.wikipedia.org/wiki/I/O_bound

Figur 1.1 http://4.bp.blogspot.com/-uZwvrT6_RVo/Vp3x0KN2nFI/AAAAAAAAAcI/O05TY2xr3QA/s1600/layer.jpg