2.3 Major achievements

Wednesday, April 1, 2020 11:39 PM

Operativsystemer er en av de mest komplekse delene i software noensinne utviklet. Dette reflekterer utfordringen med å imøtekomme vanskelighetene som bekvemmelighet, effektivitet og mulighet til å utvikle seg. Det har vært fire store teoretiske fremskritt i utviklen av operativsystemer:

1. Prosesser
2. Minnehåndtering
3. Informasjonsbeskyttelse og sikkerhet
4. Tidsstyring- og ressurs-håndtering

1. Prosessen

Sentralt i designet av operativsystemer er konseptet av *prosesser*. Denne betegnelsen er mer generell enn en *jobb*. Problemene som oppstod med de tidligere nevnte batch-systemene er at det var vanskelig å diagnotisere hvilke errors som oppstod; om feilene lå i hardware eller software for eksempel. Det er som regel fire hovedgrunner til slike errorer:

1. **Uriktig synkronisering:** Det er ofte en rutine må suspenderes i påvente av en annen handling i systemet. Uriktig design av signaliseringsmekanismene kan resultere i at signalene blir borte eller at duplikate signaler blir motatt.
2. **Feilet gjensidig utelukkelse (failed mutex):** Det skjer ofte at en bruker eller et program vil forsøke å ta i bruk en delt ressurs samtidig. Gjensidig utelukkelse handler jo om at kun en rutine skal tillates å kjøre om gangen. Denne implementasjonen av mutex er svært vanskelig å verifisere som korrekt under alle mulige sekvenser av handlinger.
3. **Ubestemte programoperasjoner:** Resultatet av et program bør normalt kun avhenge av input i programmet, og ikke aktiviteter fra andre programmer i det delte systemet. Likevel, når programmer deler minne og deres execution så kan det oppstå at de skriver over samme del i minne. Dette kan være problematisk. Altså kan rekkefølgen i planleggingen av ulike programmer påvirke resultatet av et gitt program.
4. **Vranglås (deadlocks):** Det er mulig for to eller flere programmer å bli påventende hverandre hvis begge trenger ressurser som hver av de sitter på. En slik vranglås kan avhenge av timing av ressurstildeling og -slipp.

For å håndtere disse problemene trenger vi en systematisk måte å moniterere og kontrollere de ulike programmene som blir utført av prosessoren. Konseptet av en *prosess* gir et fundament.
Vi kan anse en prosess som bestående av tre komponenter:

1. Et executable (kjørbart) program
2. Den assosierte dataen som programmet krever (variabler, arbeidsrom, buffers osv.)
3. Programmets execution-kontekst

Execution-kontekst, eller *prosesstilstand* er essensielt. Dette er den interne data som lar OS-et overvåke og kontrollere prosessen. Denne interne informasjonen er adskilt fra prosessen, ettersom OS-et har informasjon som ikke er tillatt for prosessoren. Konteksten inneholder all informasjon OS-et trenger for å håndtere prosessen, og all informasjonen prosessoren trenger for å kunne utføre prosessen ordentlig. Kontektsen inkluderer blant annet innholdet om prosessorregistre som PC (program counter) og data-registre. Den inneholder også informasjon til bruk for OS-et, som prioriteten på prosessene og om en prosess eventuelt venter for gjennomføring av en I/O-handling.

Prosessen kan dermed realiseres som en datastruktur. En prosess kan enten execute eller påvente execution. Hele tilstanden til prosessen er til enhver tid tilgjengelig i konteksten. Nye funksjoner kan legges til i OS-et ved å inkludere den nye og nødvendige informasjonen som støtter funksjonen i konteksten. Denne prosess-strukturen løser mange problemer med multiprogrammering og ressursdeling.

Ekstra smakebit: En enkel prosess med sine tilhørende ressurser kan kan deles opp i flere *samtidige* tråder som kan samarbeide om å execute og utføre arbeidet av en prosess. Dette introduserer en ny måte for parallell aktivitet håndert av hardware og software.

2. Minnehåndtering

OS-et har fem prinsipper av lagringshåndtering:

1. **Prosessisolasjon:** OS-et må forhindre uavhengige prosesser fra å forstyrre hverandres minne, både data og instruksjoner.
2. **Automatisk allokering og håndtering:** Programmer bør allokeres dynamisk på tvers av minnehierarkiet som påkrevd. Allokeringen bør være transparent til programmereren. Dermed slipper programmereren alle bekymringer relatert til minnebegrensninger og OS-et kan oppnå effektivitet ved å allokere minne til jobber etter etterspørsel.

3. **Støtte av modulærprogrammering:** Programmerere bør kunne definere programmoduler, og å kunne dynamisk lage, ødelegge og endre størrelsene på modulene.
4. **Beskyttelse og aksesskontroll:** Deling av minne, på et hvilket som helst nivå i minnehierarkiet, kan forårsake at et program adresserer minne til et annet program. Dette kan være ønskelig når det er behov for deling av gitte applikasjoner. Ved andre tilfeller kan det true integriteten til programmer og til og med OS-et selv. OS-et må altså tillate deler av minnet å være aksesserbart på ulike måter mellom ulike brukere.
5. **Langtidslagring:** Mange applikasjonprogrammer krever lagring av informasjon i lengre perioder etter computeren er blitt skrudd av.

Et OS imøtekommer typisk disse kravene med et virtuelt minne og et filsystem. Filsystemet implementerer langtidslagring hvor informasjonen lagres i navngitte objekter kalt filer. Fil-konseptet er et bekvemmelig konsept for programmerer, og en hensiktsmessig enhet av aksess og sikkerhet for OS-et.

Virtuelt minne

Er en fasilitet som tillater programmer å adressere minnet fra et logisk standpunkt, uten å måtte bekymre seg for størrelsen av hovedminnet som er fysisk tilgjengelig. Det virtuelle minnet har som hensikt å imøtekomme kravene når flere brukeres jobber ligger i hovedminnet samtidig. Når en prosess utføres er det noen av prosessens *pages* som ligger i hovedminnet, hvis det refereres til en page som ikke er i hovedminnet så blir dette avdekket av minnehåndterings-hardwaren som sammen med OS-et sørger for at den manglende pagen blir lastet inn fra sekundærminne (e.g. disk). Et slikt skjema kalles virtuelt minne.

Pages

Ettersom prosesser varierer i størrelse er det vanskelig å pakke dem kompakt hvis prosessoren bytter mellom flere prosesser. Dermed ble paging systemer introdusert, som tillater prosesser å bli kompromittert til et gitt antall fastsatte størrelser av blokker, kalt pages. Hver page i en prosess kan lokaliseres hvor som helst i hovedminne. Paging systemet tilbyr en dymanisk overføring mellom virtulle adresser i programmet og en *real address,* eller en fysisk adresse i hovedminnet.

Virtuell adresse

Et program refererer til et ord med en virtuell adresse som består av page-nummer og en offset i selve pagen.

Prosess-isolasjon kan oppnås ved å gi hver prosess en unik og uoverlappende del av hovedminnet. Delt minne kan oppnås ved overlappende deler av to virtuelle minneplasser. Filer håndteres i langtidslagringen. Filer og deler av filer kan kopieres til det virtuelle minnet.
OS-designeren må utvikle en adresse-oversetter som genererer lite overhead, og en lagringsallokering som minimerer trafikken mellom nivåene i minnet.

3. Informasjonssikkerhet

Ettersom det er blitt flere tid-delene systemer og computernettverk, har beskyttelse av informasjon blitt enda viktigere. Generelt er det problemer med å kontrollere aksess til computersystemer og informasjonen lagret i dem som vi er mest bekymret for.

De største sikkerhetsproblemene for operativsystemer kan kategoriseres til:

1. **Tilgjengelighet:** Gjelder beskyttelse av systemet mot avbrudd.
2. **Konfidensialitet:** Forsikrer at brukere ikke kan lese data de ikke er autorisert til å lese.
3. **Dataintegritet:** Beskyttelse av data fra uautoriserte modifikasjoner.
4. **Autentisering:** Omhandler verifisering og identifisering av brukere, samt validitet av meldinger og data.

4. Planleggings- (også kalt tidsstyring) og ressurshåndtering

Et hovedansvar til OS-et er å håndtere ressursene den har tilgjengelig (e.g. plass i hovedminnet, I/O-enheter, prosessorer) og å planlegge deres bruk til ulike aktive prosesser. Enhver ressurstildeling og tidsstyringsdisiplin må vurdere følgende tre faktorer:

1. **Rettferdighet:** Vi ønsker at alle prosesser som konkurrerer om bruken av en gitt ressurs har lik og rettferdig aksess til den ressursen. Spesielt for jobber av samme kalsse, altså de med lignende behov.
2. **Forskjellig respons:** På den andre siden må kanskje OS-et diskriminere mellom ulike klasser av jobber av ulike tjenestebehov. OS-et bør forsøke at tildeling og planleggingsvalg møter den totale mengden av krav. Disse valgene bør også være dynamiske.
3. **Effektivitet:** OS-et bør forsøke å maksimere gjennomstrømming (throughput), minimerere responstid, og for tid-deling tillate så mange brukere som mulig. Disse kriteriene motstrider hverandre, så å finne den rette balansen for en gitt situasjon er et pågående problem i forskningen på OS.

OS-et håndterer et antall køer, hvorpå hver kø har en liste med prosesser som venter på en ressurs. Det er en kortsiktig-kø som består av prosesser i hovedminnet som er klare med en gang en prosessor blir gjort tilgjengelig. Enhver av disse prosessene kan bruke neste prosessor. Det er opp til den kortsiktige tidsstyreren å velge hvilken prosess som får prosessoren som er klar. En vanlig strategi er å gi hver prosess i køen et slags tidsstempel; denne teknikken kalles *round-robin,* som igjen implementerer en sirkulær-kø (som en sentinelnode). En annen strategi er å innføre prioritet.

En langsiktig-kø er en liste med nye jobber som venter på prosessoren. OS-et legger jobben til i systemet ved å overføre en prosess fra langsiktig-køen til kortsiktig-køen. Dette gjør at at en del av hovedminnet må allokeres den tillagte prosessen, som igjen gjør at OS-et må være påpasselig med minnet og prosesstider for å ikke sende for mange prosesser til systemet.