8.2 Operating system software

Thursday, April 23, 2020 6:34 PM

Når man implementerer software-delen av minnehåndteringen er OS-et avhengig av tre fundamentale valg:

1. Hvorvidt man ønsker å bruke virtuelle minne teknikker
2. Hvorvidt bruken av paging eller segmentering (eller begge)
3. Hvordan algoritmene for ulike aspekter av minnehåndterinen implementeres

De to første valgene er avhengig av hardware-plattformen som er tilgjengelig. UNIX støtter for eksempel ikke et virtuelt minne ettersom prosessorene ikke støtter paging eller segmentering. Hverken av disse teknikkene er praktiske uten hardware-støtte for adresse-oversettelse og andre grunnleggende funskjoner. De fleste operativsystemer i dag benytter segmentering med paging, så utfordringene i software-implementasjonen omfatter i størst grad paging. Dette innbefatter hovedsakelig å redusere antall page-feil som oppstår, ettersom de forårsaker betydelig software-overhead.

Operativsystemets retningslinjer for virtuelt minne

Oppgaven med minnehåndtering i et paging-miljø er utrolig komplekst, så det finnes ikke noen fasit-svar for løsninger som er best i alle tilfeller. Ytelsen til et sett med retningslinjer avhenger av:

- Hovedminnets størrelse
- Den relative hastigheten til hoved- og sekundærminnet
- Størrelsen og antaller prosesser som konkurrerer om samme ressurser
- Execution-oppførselen til individuelle programmer.

Disiplinene vi ser nærmere på er:

- Fetch policy/Innhentingsprinsipper
- Plasseringsprinsipper
- Erstatningsprinsipper
- Håndtering av prosessene som er i minnet
- Rensingprinsipper
- Innlastingskontroll

Fetch policy / Innhenting

Dette omhandler hvilken page som skal hentes inn i hovedminnet. De to vanligste alternativene kalles: prepaging og etterspørsel-paging.

- **Etterspørsel-paging:** En page hentes inn i hovedminnet kun når referansen som kalles er en lokasjon i vedkommende page. I starten av kjøringen til en prosess vil det oppstå en del page-feil, men ettersom flere og flere pages blir lastet inn så synker antall page-feil på grunn av lokalitetsprinsippet til det når et veldig lavt nivå.
- **Prepaging:** Her hentes det inn andre pages enn den som er etterspurt av page-feil. Prepaging utnytter at sekundærminner gjerne har søketid og rotasjonsforsinkelser. Hvis en prosess sine pages er sammenhengende lagret i sekundærminnet, så er det mer effektivt å hente inn flere sammenhengende pages om gangen istedenfor å hente én om gangen over en lengre periode. Denne metoden er naturligvis ineffektiv dersom pages som lastes inn aldri blir referert. Prepaging kan settes til verk enten når en prosess først starter opp, eller hver gang det oppstår en page-feil. Den siste måten er å foretrekke da den er usynlig til programmerer (altså flytter vi ansvaret fra programmerer).

  Merk at prepaging ikke er det samme som bytting (swapping). Når en prosess byttes ut av hovedminnet og blir satt i hvilemodus, så vil alle dens tilhørende pages flyttes ut. Når prosessen får fortsette vil alle pages som tidligere var i hovedminnet bli returnert til hovedminnet.

Placement policy / Plasseringsprinsipper

Plasseseringsbestemmelsen er hvor i minnet en del av prosessen skal ligge. I et system med kun segmentering omfatter disse de samme metodene som var nevnt for dynamisk partisjonering; best-fit, first-fit og lignende. For systemer med paging eller segmentering/paging så er det ikke noe problem hvor pages befinner seg.

Replacement policy / Erstatningsprinsipper (Utbytting)

Utbytting omfatter hvilken page i hovedminnet som skal byttes ut med den nye pagen som skal lastes inn. Hver metode har som mål å velge den prosessen som har minst sannsynlighet til å bli referert i nærmeste fremtid. Likevel kan noen frames være låst, og kan dermed ikke byttes ut (e.g. frames som holder kjernen av OS-et).

De generelle algoritmene for utbytting

- **Optimal**
- **Last recently used (LRU)**
- **Least frequently used (LFU)**
- **First-in-First-out (FIFO)**
- **Clock**

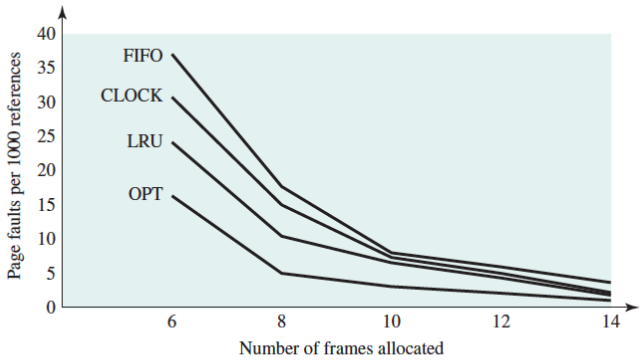


Figure 8.16 Comparison of Fixed-Allocation, Local Page Replacement Algorithms

Optimal (Kan faktisk ikke implementeres)

Denne metoden er mer et teoretisk konsept som sier at selekteringen bør være den pagen som i lengst fremtid vil bli referert, altså resulterer dette i minst page-feil. Dessverre er dette umulig å implementere da OS-et må vite alle hendelser i fremtiden. Derfor brukes dette mer som en sammenligning fra praksis til teori.

LRU (Minst nylig referert)

Denne metoden bytter ut den pagen i hovedminnet som ikke har blitt referert over lengst tidsperiode. I henhold til lokalitetsprinsippet bør dette være den pagen som med størst sannsynlighet ikke kommer til å bli referert i nærmeste fremtid. Denne er nesten like bra som optimal, men problemet er at den er vanskelig å implementere. Resultatet er ofte mye overhead for å lagre tiden for hver referanse, eller stack med referanser. Krever altså en tidsstempel for hver page tilhørende en prosess.

LFU (Minst ofte referert)

Krever en referanseteller for hver page tillhørende en prosess.

FIFO (Først inn - Først ut)

Krever sirkulær peker for hver prosess. FIFO behandler page frames tildelt en prosess som et sirkulært buffer, og pages blir fjernet i *round-robin* stil. Logikken er at pagen som har vært lengst I minnet, er den som burde byttes ut. Dette stemmer ofte ikke.

Clock

Klokkemetoder er et forsøk på å implementere LRU uten å så mye overhead. Algoritmene blir referert til som klokker da frames-ene kan visualiseres som en sirkel med pekeren i midten.

- **U-Clock:** Hver frame i primærlageret er tilordnet et bit (U-bit for Used-bit) som speiler bruken av tilhørende page. Når en page hentes inn til en slik frame, settes tilhørende U-bit til 1. Ubit (re)settes også til 1 hver gang tilhørende page refereres (leses/skrives). Når en ny page må hentes inn fra sekundærlageret og det ikke er noen ledig frame i primærlageret, søkes det sirkulært - fra siste frame som fikk hentet inn en side, etter en frame hvor Ubit er satt til 0. I denne søkeprosessen settes alle U-bit som er 1 til 0 før en fortsetter søket. Til slutt vil en således treffe en frame hvor U-bit står i 0 – selv om noen pager hvis U-bit ble satt til 0 i søkeprosessen kan ha rukket å få satt sitt U-bit til 1 igjen gjennom ny lesing/skriving, og denne velges til innhenting av den nye pagen.

  Dette ligner på FIFO, bortsett fra at en frame med U = 1 vil passeres.

- **UM-Clock:** Klokkealgoritmen kan gjøres kraftigere ved å øke antall bits i bruk. I alle prosessorer som støtter paging er det en M-bit (modifiy-bit) assosiert til hver page i hovedminnet, og dermed også hver frame i hovedminnet. Denne bit-en er forklart tidligere, og er for å sjekke om man må oppdatere pagen som ligger i sekundærminnet. Dette kan utnyttes av klokke-algoritmen. Hver frame faller inn I en av fire kategorier:

Ikke aksessert nylig, ikke modifisert	(U=0; M=0)

Aksessert nylig, ikke modifisert	(U=1; M=0)
Ikke aksessert nylig, modifisert	(U=0; M=1)
Aksessert nylig, modifisert	(U=1; M=1)

Med denne klassifiseringen fungerer UM-clock slik:

1. Starter ved posisjonen til pekeren og scanner frame-bufferet. Under denne scanningen røres ikke U-biten. Den første framen med (U = 0, M = 0) velges for utbytting.
2. Hvis steg 1 feiler, så scannes det igjen for å se etter (U = 0, M = 1). Dersom det finnes velges denne framen for utbytting. Under denne scanningen settes alle til U = 0 for hver forbipassering.
3. Hvis steg 2 feiler, skal pekeren ha returnert til startposisjonen og alle frames skal være 0. Gjenta steg 1, og hvis nødvendig, gjenta steg 2.

Sammenligning av algoritmene

- LRU & LFU er i utgangspunktet for ressurskrevende til å brukes for slik implisitt I/O (men kan brukes – gjerne i kombinasjon, for håndtering av eksplisitt I/O)

- U-CLOCK & UM-CLOCK er mindre ressurskrevende og passer godt for slik implisitt I/O

- UM-CLOCK er litt mer plasskrevende og tidskrevende enn U-CLOCK, men gir samtidig bedre resultater – og er således å foretrekke hvis en har råd til denne ekstra overheaden. UM-clock foretrekker pages som ikke er blitt modifisert til utbytting, hvilket resulterer i færre I/O-operasjoner.

- CLOCK/U-CLOCK har mindre overhead med brukbar ytelse, mens LRU har bedre ytelse med stor overhead. Således er LRU og lignende gode algoritmer i praksis uaktuelle her da kostnadene mht. plassbehov og eksekveringstid blir for store sammenlignet med ved bruk av CLOCK/U-CLOCK og lignende billige algoritmer.

Resident Set Management / Håndtering av prosessene som er i minnet

Paging med virtuelt minne gjør at det ikke er nødvendig å hente inn alle pages tilhørende en prosess når man forbereder execution. Derfor må OS-et ta et valg på hvor mange pages som skal hentes inn, det vil si hvor mye av hovedminnet som skal tildeles. Det er flere faktorer som spiller inn her:

- Jo mindre minne som tildeles en prosess, desto flere prosesser kan ligge i hovedminnet om gangen. Dette øker sannsynligheten for at OS-et vil finne minst en prosess som er i Ready-tilstand. Dette reduserer den tapte tiden på grunn av bytting.
- Dersom et relativt lite antall pages tilhørende en prosess ligger i hovedminnet, så vil det til tross for lokalitetsprinsippet være en høy page-feil-rate.
- Over en viss størrelse, så vil ikke ekstra tildeling av plass i hovedminnet til en prosess ha noen særlig effekt på page-feil-raten på grunn av lokalitetsprinsippet.

Metoder for tildelingen av minne

- **Bestemt tildeling (fixed-allocation):** En prosess får et bestemt antall frames i hovedminnet hvor den kan execute. Dette antallet bestemmes ved innlasting (opprettelse av prosessen) og blir bestemt ut fra hva slags type prosess det er, eller på intrukser av programmerer. Når det oppstår en page-feil i execution hos en prosess ved bestemt tildeling, så vil en av pagene tilhørende den prosessen måtte bli erstattet av den etterspurte pagen.
- **Variende tildeling (variable-allocation):** Her vil antallet frames en page tilhørende en prosess er inndelt i (page frames) variere underveis i livssyklusen til prosessen. Her kan tildelingen av page frames økes og minkes underveis basert på page-feil-ratene hos prosessene; en prosess med mange page-feil vil tildeles flere page frames i samsvar med lokalitetsprinsippet, mens en prosess med relativt få page-feil vil frigjøres page frames i hovedminnet i beste håp om at det ikke øker page-feil-raten i stor grad. Denne metoden virker best, men vanskeligheten med denne metoden er at den krever OS-et å inspiserere oppførselen til aktive prosesser underveis. Dette gir selvfølgelig mye overhead I software, og er avhengig av hardware-mekanismer hos prosessor.

Replacement scope / Utbyttingsomfang

Et lokalt utbyttingsomfang tilsier å kun velge blant pages i hovedminnet til prosessen som fikk en page-feil når det velges hvilken page som skal byttes ut.

Et globalt utbyttingsomfang tilser å vurdere alle ulåste pages i hovedminnet som kandidater for utbytting, uavhengig av hvilken prosess som eier pagen.

####	Lokal utbytting	Global utbytting

Bestemt tildeling	*OK* • Antall frames tildelt en prosess er bestemt • Pagen som skal byttes ut velges blant frames tildelt prosessen pagen tilhører. • ULEMPE: Kan oppstå thrashing ved for store tildelinger. Kan sinke systemet ved for små tildelinger, da det medfører mye page-feil.	Ikke mulig.
Varierende tildeling	*Avansert, men ofte bedre.* • Antallet frames tildelt en prosess kan variere gjennom levetiden for å opprettholde arbeidssettet til prosessen. • Pagen som skal byttes ut velges balnt frames tildelt prosessen pagen tilhører.	*OK* • Pagen som skal byttes ut velges blant alle tilgjengelige frames i hovedminnet; dette gjør at mengden av en prosess I hovedminnet vil variere. Kan forbedres med et page-buffer.

Varierende tildeling, lokalt omfang

Denne metoden innfører et begrep om arbeidssett som definerer mengden minne som en prosess krever i et gitt tidsintervall. Arbeidssettstørrelsen speiler omfanget av det settet med sider som til enhver tid aksesseres. Lokalitetsprinsippet tilsier at arbeidssettet og dermed også arbeidssettstørrelsen holder seg stabile over visse tidsrom. Med jevne mellomrom endrer dog arbeidssettet seg, og arbeidssettstørrelsen vil svinge tilsvarende opp og ned. I slike overgangssituasjoner vil sidene som aksesseres i praksis tilhøre «to arbeidssett», og den tilhørende «arbeidssettstørrelsen» vil da være større enn både før og etter.

Figuren under demonstrerer dette fenomenet:

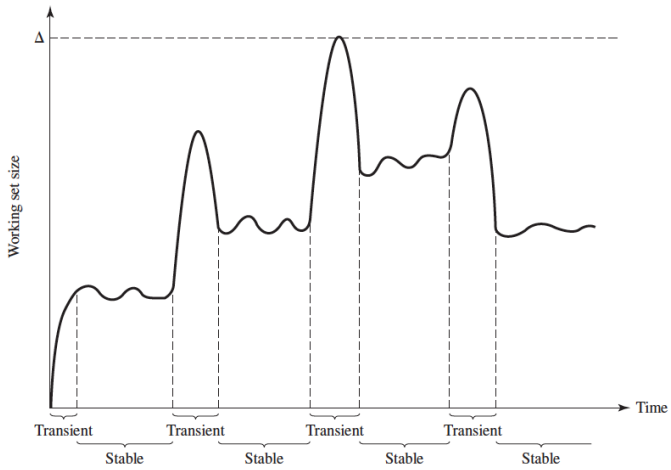


Figure 8.18 Typical Graph of Working Set Size [MAEK87]

Cleaning policy / Rensing

Rensing er det motsatte av fetching, og går da utpå å avgjøre når en modifisert page skal skrives ut til sekundærminnet. Det er to grunnleggende alternativer:

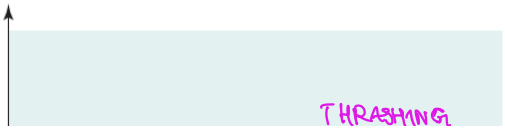
- **Etterspørsel-rensing:** En page skrives ut til sekundærminnet kun når den har blitt valgt til utbytting.
- **Precleaning / Pre-rensing:** Skriver hvilke pages som er modifisert før deres page frames trengs, slik at flere bages kan skrives ut samtidig.

Begge disse metodene har ulemper. En god løsning bruker page-buffering med modifiserte- og ikke-modifiserte lister.

Load Control / Innlastningskontroll

Avgjør hvor mange prosesser som skal ligge i hovedminnet, også kalt multiprogrammeringsnivået. Dette er veldig viktig for å unngå thrashing.

L = S criterion illustrerer sammenhengen mellom page-feil udner levetiden (L) og page byttingstid (S, swaptime).



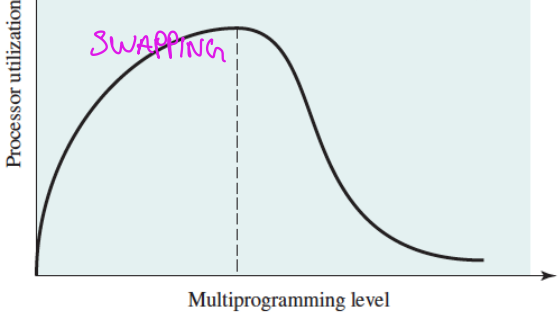


Figure 8.19 Multiprogramming Effects

Suspendering av prosesser

Dersom graden av multiprogrammering reduseres må en prosess suspenderes (byttes ut). Dette gjøres basert på *prioritet, høyest page-feil-rate, minst sett med prosess-deler, største prosess* eller *størst execution-vindu*.

For å få prosesser/tråder I hovedminnet gir dårlig CPU-utnyttelse da det ofte ikke er prosesser/tråder tilgjengelig for bruk av prosessorkraft. For mange prosesser vil også gi dårlig utnyttelse av CPU da det dermed brukes mye prosessorkraft på bytting inn og ut av hele prosesser/tråder. Ved å overvåke page-feil-frekvensen (PFF) til hver enkelt prosess/tråd kan man tilnærme et balansepunkt. Hver prosess/tråd må ha såpass mye data I hovedminnet at dens page-feil-frekvens holder seg under en viss øvre grense. Antall aktive prosesser/tråder vil da justeres til enhver tid basert på svingningene I hver enkelt prosess/tråd sin tilhørende page-feil-frekvens.