8.1.1 Paging w/ virtual memory

Wednesday, April 22, 2020 5:51 PM

Fra kap 7 så vi på hvordan enkel paging blir utnyttet ved å dele inn hovedminnet i like store frames. Hver prosess deles inn i et antall pages som har like stor størrelse som frames. Denne inndelingen gjøres også i paging for systemer som bruker virtuelt minne, men forskjellen her er at ikke alle pages trenger å lastes inn i frames i hovedminnet under execution.

I enkel paging har hver prosess sin egen page-tabell, og dette er også nødvendig for paging med virtuelt minne. Forskjellen er at page-tabellen blir mer kompleks ved bruk av virtuelt minne. Ettersom alle pages tilhørende en prosess ikke nødvendigvis ligger i minnet må tabellen holde oversikt over hvilke pages som er i hovedminnet eller ikke – dette gjøres ved en bit P, som er lik 1 dersom pagen er i minnet. En annen bit M indikerer om innholdet i pagen har blitt endret sist den var lastet inn i hovedminnet. Hvis pagen ikke er blitt endret på trenger man ikke overskrive det som ligger i sekundørminnet når man bytter den ut av hovedminnet. Denne metoden gir som sagt ikke ekstern fragmentering, men noe intern fragmentering.

Page-tabell (struktur)

Hovedfunksjonen til page-tabellen er å konvertere mellom virtuell (logisk) adresse til fysisk adresse. Når en bestemt prosess kjører blir startadressen til prosessens page-tabell holdt på av et register. Page-nummeret til en virtuell adresse brukes til å indeksere tabellen og hente tilhørende frame-nummer. Sammen med offset utgjør dette den fysiske adressen. De fleste page-tabellene lagres i det virtuelle minnet for å spare plass i hovedminnet, og derfor gjøres paging også på tabellen i seg selv. En ulempe med denne tabellstrukturen er at størrelsen er proporsjonal med det virtuelle adresserommet.

Struktur

Virtuell adresse	Page-nummer, offset
Rad i page-tabellen	P- og M-flagg, kontrollbits, frame-nummer

TLB (Translation Lookaside Buffer)

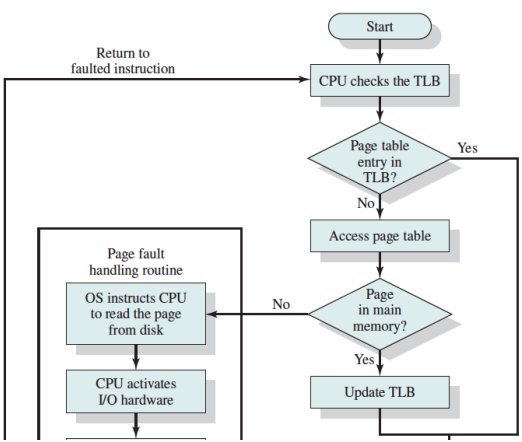
I prinsippet kan hver virtuelle minnereferanse gi opphav til to fysiske minneaksesser: én for å hente den passende page-tabellen, den andre til å hente data. For at dette ikke skal doble minneaksesstiden har de fleste virtuelle minne implementasjonene en TLB. Dette er en høyhastighets cache for data tilhørende en page-tabell som er blitt brukt nylig.

Ettersom TLB ikke inneholder alle radene i en page-tabell så fungerer ikke indeksering i TLB basert på page-nummer. Istedenfor består hver rad i TLB av page-nummer og hele page-tabell oppføringen. Prosessoren er utstyrt med hardware som lar den sjekke for match i TLB basert på flere page-numre om gangen. Denne teknikken kalles *assosiativ mapping*.

Fra TLB til hovedminnet

Til slutt må mekanismene i det virtuelle minnet samhandle med hovedminnets cache. Først sjekker minnesystemet TLB for å se om den matchende page-tabell oppføringen ligger der. Dersom den befinner seg der kombineres frame-nummer med offset for å generere den reelle (fysiske) adressen. Hvis ikke må den hentes fra page-tabellen. Når den reelle adressen er generert sjekkes cache for å se om blokken med data vi ønsker å adressere er tilstede. Hvis ja, returner blokken til prosessoren. Hvis nei, hent blokken fra hovedminnet. Hvis den refererte datablokken kun er i sekundærminnet må pagen som inneholder adressen i den datablokken lastes inn i hovedminnet, og dens tilhørende datablokk lastes inn i cache. I tillegg må oppføringen i page-tabellen for den pagen oppdateres.

Figiruen under viser logikken:



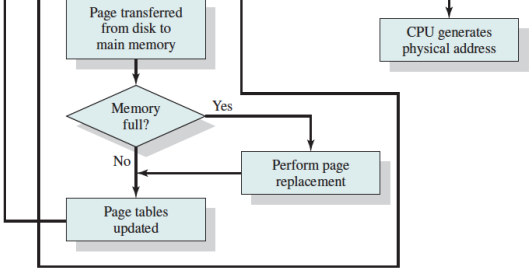


Figure 8.7 Operation of Paging and Translation Lookaside Buffer (TLB)

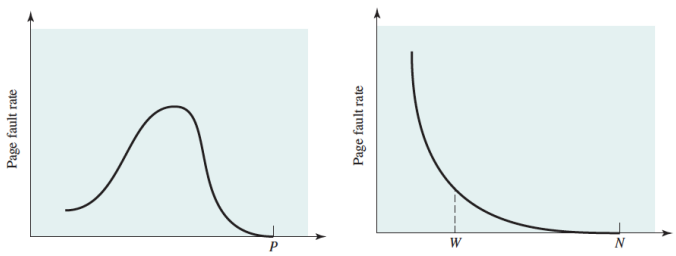
Page-størrelse

Det er flere vurderinger som må tas i hardware designet av hvor store pages skal være. Ved veldig små størrelser vil det være plass til mange flere pages tilhørende en prosess i hovedminnet om gangen. Etter litt kjøring vil alle pages i minnet inneholde noen deler av nylig brukte referanser, og dette kommer jo av lokalitetsprinsippet – dermed vil det være få page-feil.

Ved større størrelser på pages vil hver individuelle page inneholde lokasjoner lengre fra nylig brukte referanser. Altså mister vi effekten av lokalitetsprinsippet og antall page-feil økes. Etterhvert når page-størrelsen nærmer seg størrelsen til prosessen vil antall page-feil igjen synke da hele prosessen får plass i pagen. Figur *b* under viser hvordan feilene avtar ved fast page-størrelse ettersom antall pages i hovedminnet vokser.

Vurderingene ved valg av størrelse omfatter:

- **Intern fragmentering:** Ved mindre page-størrelser er det mindre intern fragmentering fordi man får utnyttet minnerommet.
- **Antall pages som kreves per prosess:** Ved små page-størrelser kreves det mange pages per prosess, hvilket betyr større page-tabeller. For store programmer må da deler av denne tabellen ligge i det virtuelle minnet for ikke å oppta for stor del av hovedminnet. Dette kan medføre doblede page-feil for en enkelt referanse (da vi dermed bruker to minner).
- **Sekundærminnets fysiske egenskaper:** Bedre anlagt til større page-størrelser for å transportere blokker av data mer effektivt.



P = size of entire process
W = working set size
N = total number of pages in process

Figure 8.10 Typical Paging Behavior of a Program

Disse figurene demonstrerer følgende:

1. Hvis page-størrelsen er stor nok rommer den en hel prosess/tråd i en enkelt page slik at page-feil ikke vil oppstå. Ulempen med dette er at en ikke vil ha plass til så mange prosesser/tråder i hovedminnet.
2. Hvis page-størrelsen er liten nok vil det sjeldent skje page-feil ettersom det rommes mange nok deler av prosessen/tråden i fokus som er tildelt hovedminnet-området. Utfordringen her er at en ikke har plass til så mange prosesser/tråder i hovedminnet grunnet enormt overhead til page-tabeller osv.