5.8 Summary

Friday, April 17, 2020 4:10 PM

De grunnleggende temaene for moderne operativsystemer er multiprogrammering, multiprosessering, og distribuert prosessering. Sentralt i alle disse temaene er konseptet om samtidighet (concurrency). Når flere prosesser executer smatidig vil det oppstå utfordringer med konflikthåndtering og samarbeid.

De ulike måtene prosesser samhandler på må ta hensyn til gjensidig utelukkelse og vranglåser (og i noen tilfeller starvation).

Gjensidig utelukkelse brukes til å håndtere konflikter som konkurranse om ressurser og synkronisering mellom prosesser så de kan samarbeide. Det finnes tre grunnleggende tilnærminger for å støtte gjensidig utelukkelse i systemer:

1. **Spesielle maskin-instruksjoner:** Denne metoden reduserer overhead, men er fortsatt ineffektiv på grunn av at den bruker busy-waiting.
2. **Semaforer:** Her er gjensidig utelukkelse implementert via OS-et. Semaforer er hensiktsmessig for signalisering mellom prosesser og kan enkelt brukes til å håndheve gjensidig utelukkelse.
3. **Meldingsutveksling:** Her er gjensidig utelukkelse implementert via OS-et. Hensiktsmessig for håndheving av gjensidig utelukkelse og kan også tilby effektive måter for interprosess-kommunikasjon.

Utfordringene med synkronisering kan løses med semaforer, meldingsutveksling og monitorer. De har dog sine fordeler og ulemper i ulike settinger.

Semaforer er generelle og kan brukes til et bredt spekter av problemer, men de har ikke direkte tilrettelegging for gjensidig utelukkelse, så dette ansvaret ligger hos utvikler. De er lavnivå som betyr at de sparer plass.

Monitorer følger en objektorientert struktur og er lettere å verifisere. I tillegg sikrer de automatisk gjensidig utelukkelse, så passer dermed bra til problemer hvor dette er vesentlig.

Meldingsutveksling kan brukes ved både delt og ikke delt lager, som for eksempel ved distribuerte systemer. Hensiktsmessig når tidsstyring er vesentlig.

Fordeler med monitorer vs. semaforer

En fordel med monitorer istedenfor semaforer er at synkroniseringsfunksjonene er begrenset til monitoren. Dermed er det lettere å verifisere at synkroniseringen er gjort riktig, og det er i tillegg lettere å oppdage feil (hva slags feil). I tillegg håndhever monitorer gjensidig utelukkelse, mens for semaforer ligger dette ansvaret hos programmerer. Fordelen med semaforer er dog at de tillater flere tråder å aksessere sin kritiske seksjon om gangen, mens monitorer er én tråd om gangen. Semaforer brukes derfor i tilfeller for det eksisterer enkle gjensidige utelukkelser og hastighet er viktigere.

Kort fortalt

Monitor: kontrollerer kun én tråd om gangen som kan execute i monitoren. (her trengs det en lås for å execute den enkle tråden).
Semafor: en lås (counter) som beskytter en delt ressurs. (her trengs det en lås for å aksessere den gitte ressursen).

EGNE BEMERKNINGER

Det kan være litt vanskelig å skille samtidighet og parallellitet ved første øyekast. Samtidighet handler om å *håndtere* flere ting på en gang. Parallellitet handler om å *gjøre* flere ting på en gang.

Samtidighet
Er når flere oppgaver starter, kjører og fullføres i overlappende tidsperioder, og i ingen spesifikk rekkefølge. Samtidighet er mulig hvis vi snakker om minst to oppgaver eller flere.

Selv om det virker som om oppgaver kjører samtidig, så stemmer ikke det nødvendigvis. De utnytter *time-slicing* hvor hver oppgave kjører deler av deres oppgaver og så går inn i vente-modus. Når oppgave 1 går i vente-modus, så tildeles en prosessor til oppgave 2 for å fullføre dens oppgave. Operativsystemer basert på prioritet av oppgaver tildeler dermed prosessorer og andre ressurser på tur til alle oppgaver og gir dem en sjanse til å fullføre. For sluttbrukeren virker det som om alle oppgavene kjøres i parallell. Dette kalles samtidighet.

Parallellitet
Er når flere oppgaver ELLER flere deler av en unik oppgave faktisk kjører på samme tidspunkt, e.g. multikjerne-prosessorer.

Parallellitet krever ikke at minst to oppgaver eksisterer. Her kjører faktisk deler av oppgavene eller flere oppgaver fysisk samtidig, altså på samme tidspunkt. Dette kan gjøres ved en multi-kjerne infrastruktur hos prosessoren; der en kjerne tildeles en oppgave eller deloppgave.

Parallellitet krever hardware med flere prosessringsenheter. I et uniprosessor-system kan du oppnå samtidighet, men IKKE parallellitet.