4.1 Processes and threads

Wednesday, April 8, 2020 3:19 AM

Konseptet om en prosess har blitt presentert så langt til å ha to (uavhengige) hovedkarakteristikker:

1. **Resursseierskap:** En prosess inkluderer et virtuelt adresserom til å holde prosessbildet. Fra tid til annen kan en prosess bli tildelt kontroll eller eierskap av ressurser, som hovedminnet, I/O-kanaler, I/O-enheter og filer. OS-et utfører en beskyttelsesfunksjon for å unngå uønsket fortyrrelser mellom prosesser og ressursene de interagerer med. Kalles prosess eller *task* på engelsk.

2. **Planlegging/execution:** Execution av en prosess følger execution-sti (trace) gjennom en eller flere programmer. En slik execution kan innflettes med andre prosesser. Dermed har en prosess en tilstand og en dispatcher-prioritet, og er selve entiteten som planlegges og dispatches av OS-et. Kalles *lightweight process* på engelsk.

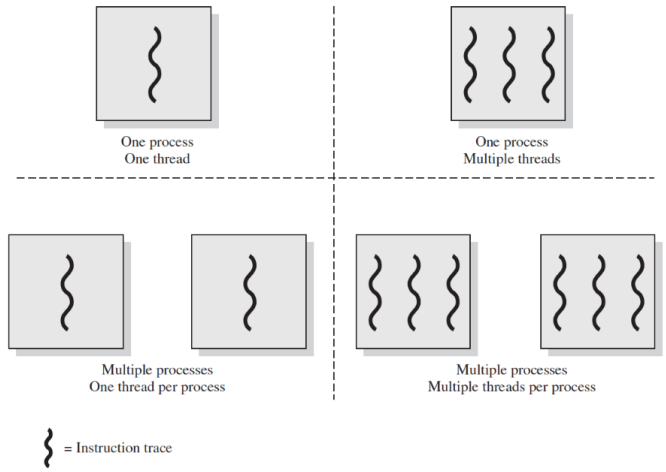

Figure 4.1 Threads and Processes

Multitråding/flertråding

Multitråding referer til egenskapen et OS tilbyr til å støtte flere, samtidige stier av execution i en enkel prosess. Den tradisjonelle tilnærmingen med en enkel tråd av execution per prosess kalles en *single-threaded* tilnærming. I et multitrådet miljø er en prosess definert om en enhet av ressurstildeling og en enhet av beskyttelse; med følgende assosiasjoner:

- Et virtuelt adresserom som holder prosessbildet
- Beskyttet aksess til prosessorer, andre prosesser, filer og I/O-ressurser.

I en prosess kan det være en eller flere tråder, der hver tråd har:

- En tråd-execution-tilstand (Running, Ready osv.)
- En lagret trådkontekst når den ikke kjører. En måte å se på en tråd; er som en uavhengig programteller som opererer i en prosess
- En execution-stack
- Noen per-tråd statisk lagring for lokale variabler
- Aksess til minnet og ressurser av dens prosess, delt med alle tråder tilkoblet den prosessen

For en enkel-trådet prosessmodell inkluderer representasjonen av en prosess dens prosesskontrollblokk og bruker-adresserommet, samt bruker- og kjerne-stacks for å håndtere kall/returnerings-oppførsel av execution for en prosess. Når prosessen kjører kontrollerer den prosessor-registrene. Inneholdet i disse registrene lagres når den ikke kjører.

I et multitrådet miljø, så er det fortsatt en enkel prosesskontrollblokk og brukers adresserom, men nå er det adskilte stacker for hver tråd, samt adskilte kontrollblokker for hver tråd som inneholder registerverdier, prioritetsnivå, og annen tråd-relatert tilstandsinfromasjon. Altså deler alle trådene i en prosess tilstanden og ressurser av den prosessen. De ligger i samme adresserom og har aksess til samme data. Når en tråd endrer et element av data i minnet, så må andre tråder kunne se resultatet hvis og når de aksesserer det elementet. Hvis en tråd åpner en fil med lese-privileger, så må andre tråder i den samme prosessen også kunne lese fra den filen.

Figuren under viser forskjellen mellom tråder og prosesser fra prosesshåndterers perspektiv.

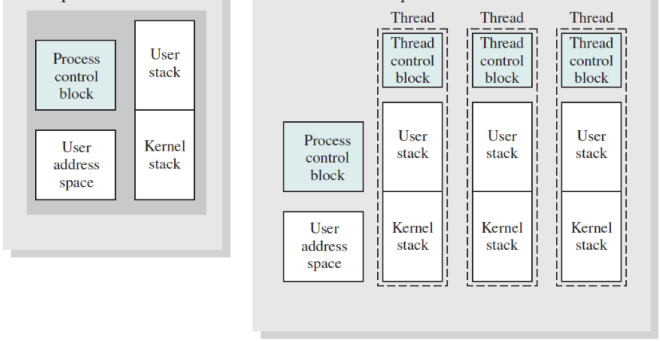

Figure 4.2 **Single-Threaded and Multithreaded Process Models**

Fordelene med tråding kommer fra ytelseimplikasjoner:

- Det tar langt mindre tid å opprette en ny tråd i en eksisterende prosess en å lage en helt ny prosess.
- Det tar mindre tid å terminere en tråd enn en prosess.
- Det tar mindre tid å skifte mellom to tråder i samme prosess enn et prosess-skifte.
- Tråder forbedrer effektiviteten i kommunikasjonen mellom forskjellige executing programmer. I de fleste OS-er så krever kommunikasjon mellom uavhengige prosesser innblanding av kjernen (kernel) for å tilby beskyttelse og mekanismer for kommunikasjon. Men fordi tråder i samme prosess deler minne og filer, kan de kommunisere effektivt uten å involvere kjernen (kernel).

Altså er det langt mer effektivt å bruke en samling av tråder istedenfor adskilte prosesser dersom det er en applikasjon eller funksjon som skal implementeres som et sett med relaterte enheter av execution. Tråder er også effektivt på enkelte prosessorer for å forenkle strukturen av et program som logisk utfører flere forskjellige funksjoner. Fire eksempler på bruk av tråder i enkel-bruker multiprosessorsystemer inkluderer:

1. **Forgrunns- og bakgrunnsarbeid:** Tråder kan bli delegert arbeid på en systematisk måte; som at en tråd viser menyer og leser av brukerinput, mens en annen tråd executer bruker-kommandoer og oppdaterer menyen.
2. **Asynkront prosessering:** Asynkrone elementer i et program kan implementeres som tråder.
3. **Hastighet til execution:** En multitrådet prosess kan beregne en batch av data mens den leser neste batch fra en enhet. I et multiprosessorsystem kan flere tråder fra samme prosess execute samtidig. Dermed kan en annen tråd execute mens en annen kanskje er blokkert av en I/O-operasjon.
4. **Modulær programstruktur:** Programmer som involverer flere ulike aktiviterer eller ulike kilder og destinasjoner for input og output kan være enklere å designe og implementere ved bruk av tråder.

I et OS som støtter tråder, så er planlegging og dispatching gjort på et tråd-basis, dermed er mesteparten av tilstandsinformasjon som omhandler execution håndtert i tråd-nivå datastrukturer. Dersom OS-et må håndtere på prosess-nivå, som for eksempel ved bytting (swapping), så må i så fall alle tilhørende tråder suspenderes samtidig de også. Det samme gjelder ved terminering av en prosess.

Trådfunksjonalitet

I likhet med prosesser har tråder også executiontilstander som kan synkroniseres med hverandre.

Tråd-tilstander

Som for prosesser er de viktigiste tilstandene for tråder Running, Ready og Blocked. Det gir ikke mening at tråder er kan være i suspendert tilstand da de er et konsept forbundet med prosesser. Det er fire grunnleggende tråd-operasjoner assosiert med endringer i tråd-tilstander:

1. **Opprettelse:** Det er vanlig at når en prosess opprettes, så opprettes det også en tråd for den prosessen. Deretter kan en tråd i prosessen også opprette andre tråder i samme prosess, som da tilbyr en intruksjonspeker og argumenter for den nye tråden. Den nye tråden får sitt eget registerkontekt og stack-rom, og plasseres i Ready-køen.
2. **Blokkering:** Når en tråd må vente på en hendelse vil den blokkeres og dermed lagre brukerregistre, programteller og stack-pekere. Prosessoren kan da se på execution av andre tråder som er i Ready i samme prosess eller en annen prosess.
3. **Opphevelse av blokkering:** Når hendelsen som gjorde tråden blokkert skjer, så flyttes tråden til Ready.
4. **Ferdig:** Når en tråd fullfører, så blir dens registerkontekst og tilhørende stacker frigjort.

Et problem som kan oppstå er om en blokkert tråd resulterer i at hele prosessen blir blokkert og altså ikke kan kjøres. Dette diskuteres videre i bruker-nivå tråder og kjerne-nivå tråder. I en uniprosessor kan miltiprogrammering muliggjøre innfletting av flere tråder i flere prosesser. Execution kan da sendes fra en tråd til en annen når enten den nåværende kjørende tråden er blokkert, eller på grunn av en time slice.

Trådsynkronisering

Alle trådene i en prosess deler samme adresserom og andre ressurser (e.g. åpne filer). Enhver endring utført av en tråd på en ressurs påvirker miljøet til de andre trådene i samme prosess. Det er nødvendig å synkronisere aktivitetene av ulike tråder slik at ikke forstyrrer hverandre eller ødelegger datastrukturene. Det er teknikker for å håndtere slik synkronisering (Se kap 5 og 6).