3.2 Process states

Sunday, April 5, 2020 5:32 PM

Som diskutert tidligere; for at et program skal kunne executes, så opprettes det en prosess eller en oppgave for det programmet. Fra prosessorens standpunkt executer den instruksjoner fra sitt repertoar i en sekvens diktert av de endrede verdiene i programteller(PC)-registeret. Over tid kan programtelleren referere til kode i forskjellige programmer, hvor dens execution involverer en sekvens av instruksjoner for det andre programmet.

Vi kan karakterisere oppførselen til en individuell prosess ved å liste sekvensen av instruksjoner som executes for den prosessen. En slik liste kalles *sporingen (trace)* av prosessen. Vi kan forklare oppførselen av prosessoren ved å se på hvordan sporene til ulike prosesser er innflettet. Det er et *dispatcher* program som bytter prosessoren fra en prosess til en annen.

To-tilstander prosessmodell

Operativsystemets prinsipielle ansvar er å kontrollere execution av prosesser; dette inkluderer avgjørelser om det innflettende mønsteret for execution og tildeling av ressurser til prosessene. Det første steget i å designe et operativsystem som skal kontrollere prosesser er å beskrive oppførselen som vi ønsker at prosessene utøve. Vi kan konstruere den enkleste modellen ved å observere at ved ethvert tidspunkt, så blir en prosess enten executed av en prosessor, eller ikke. Dermed har prosessene to tilstander; enten *kjørende* eller *ikke-kjørende*. Når OS-et oppretter en ny prosess, så oppretter den også en prosesskontrollblokk for prosessen og vedlegger prosessen i systemet i en *ikke-kjørende* tilstand. Prosessen eksisterer og anerkjennes av OS-et, og venter dermed på en mulighet til å execute. Fra tid til annen vil den nåværende prosessen som kjører bli avbrutt, og da vil *dispatcher*-delen av OS-et selektere en annen prosess å kjøre. Den tidligere prosessen flyttes da fra *kjørende* tilstand til *ikke-kjørende* tilstand, og omvendt for den andre prosessen.

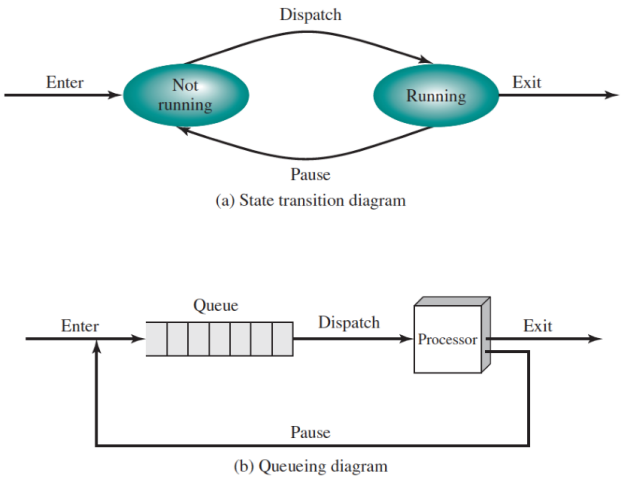

Figure 3.5 Two-State Process Model

Som vi kan lese fra figuren må OS-et altså holde styr på hvilke prosesser som er i kjørende eller ikke-kjørende tilstand. Dette implementeres ved at hver prosess har en tilhørende informasjonsdel, som inneholder hvilken tilstand den er i og lokasjon i minnet; dette er prosesskontrollblokken. Prosesser som ikke kjører legges til i en kø, hvor hver inngang i køen har en peker til prosesskontrollblokken for hver prosess. Dispatcheren er ansvarlig for å hente ut prosesser som skal executes.

Opprettelse og terminering av prosesser

Prosessopprettelse

Når en prosess skal legges til de som allerede håndteres, så bygger OS-et datastrukturene som brukes til å håndtere prosessen, og tildeler adresserom i hovedminnet til prosessen. Det er fire hendelser som resulterer i at en prosess blir opprettet:

1. **Ny batchjobb:** OS-et besitter en batchjobb-kontrollstrøm. Når OS-et er forberedt på nytt arbeid, så leser den neste sekvens av jobbkontrollkommandoer.
2. **Interaktiv pålogging:** En bruker i terminalen logger på systemet.
3. **Opprettet av OS til å utfylle en tjeneste:** OS-et kan opprette en prosess til å utføre en funksjon på vegne av et brukerprogram, uten at brukeren trenger å vente.
4. **Opprettet av eksisterende prosess:** For grunner av modularitet eller til å utnytte parallellitet, så kan et brukerprogram diktere opprettelsen av flere prosesser. Når en prosess oppretter en annen kalles den *parentprocess*, mens den som ble opprettet blir kalt *childprocess*.

Prosessterminering

Ethvert computersystem må ha en måte å indikere at prosessoren har fullført prosessen eller oppgaven. Grunner til prosessterminering:

- **Normal fullføring:** Prosessen executer et OS-tjenestekall som indikerer at den har fullført kjøringen.
- **Tidsbegrensning oversteget:** Når en prosess har kjørt lengre enn den spesifiserte tidsbegrensningen. Dette kan måles i *total elapsed time, tidsbruk på executing*, eller for brukerprogrammer hvis brukeren har brukt lang tid på å sende inn et nytt input.
- **Utilgjengelig minne:** Prosessen krever mer minne enn systemet kan tilby.
- **Grensebrudd:** Prosess prøver å aksessere en minnelokasjon den ikke har tillatelse til å aksessere.
- **Beskyttelsesfeil: P**rosessen forsøker å bruke en ressurs den ikke har adgang til, som for eksempel en fil den ikke har aksess til. Eller at den prøver å bruke ressursen på feil måte, som å skrive til en read-only fil.
- **Aritmetisk feil:** Prosessen prøver på en ugyldig aritmetisk operasjon, som å dele på null, eller prøver å lagre numre som er større enn hardware aksepterer.
- **I/O-feil:** En error kan oppstå under input eller output, som at den ikke finner filen, eller feil i å skrive eller lese etter et spesifisert antall forsøk, eller ugyldige operasjoner.
- **Ugyldig instruksjon:** Prosessen forsøker å execute en ikke-eksisterende instruksjon.
- **Privilegert instruksjon:** Prosessen forsøker å bruke en instruksjon reservert for operativsystemet.
- **Datamisbruk:** En del av data som er av feil type eller ikke initialisert.
- **Operator- eller OS-innblanding:** Av en eller annen grunn så har operatoren eller operativsystemet terminert prosessen; kan være eksistensen av en vranglås.
- **Forelderterminering:** Når en *parent process* terminering kan det hende at OS-et terminerer alle tilhørende *child processes*.
- **Forelderforespørsel:** En *parent process* har typisk autoritet til å terminere sine egne *child processes.*

Fem-tilstander prosessmodell

En mer effektiv køhåndteringen enn to-tilstanders prosessmodellen krever ta dispatcher skanner prosessene i køen. Det er nemlig slik at noen prosesser som er i *ikke-kjørende* tilstand fortsatt ikke er klare til å kjøres. Dermed er naturlig å innføre to nye tilstander for prosesser som er ikke-kjørende: *blokkert* og *klar*. De fem nye tilstandene i diagrammet blir da:

1. **Kjørende (Running):** Prosessen som for øyeblikket kjøres.
2. **Klar (Ready):** En prosess som er forberedt på å execute når muligheten byr seg.
3. **Blokkert/ventende (Blocked):** En prosess som ikke kan execute inntil en viss hendelse skjer, som for eksempel fullføringen av en I/O-operasjon.
4. **Nylig opprettet (New):** En prosess som akkurat er blitt opprettet men som ennå ikke er blitt lagt til i executable prosesser av OS-et. En ny prosess er typisk ikke lastet inn i hovedminnet, selv om dens prosesskontrollblokk har blitt opprettet.
5. **Exit:** En prosess som har fjernet fra executable prosesser av OS-et, enten fordi den er stanset eller fordi den ble aborted av en eller annen grunn.

Ved å ha en nylig opprettet-tilstand sparer OS-et plass i hovedminnet, ettersom selve prosessen ikke trenger å ligge i hovedminnet; kun den nødvendige informasjonen for å kunne hente den inn. Terminering forflytter en prosess til exit-tilstand. På dette tidspunktet kan ikke lenger prosessen executes. Figuren under demonstrer overgangene mellom hver tilstand:

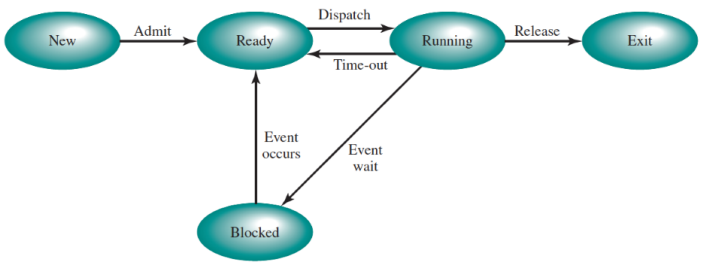

Figure 3.6 Five-State Process Model

Disse overgangene inkluderer:

- Null → New: En ny prosess er opprettet til å execute et program, med opphav i en av grunnene i prosessopprettelse.
- New → Ready: OS-et vil forflytte en prosess fra New til Ready når den er klar til å ta på seg en ny prosess. De fleste systemer setter en grense basert på antallet eksisterende prosesser, eller antallet virtuelt minne som er tilknyttet til eksisterende prosesser. Denne grensen forsikrer at det ikke er for mange prosesser som igjen kan senke ytelsen.
- Ready → Running: Når det er tid for å selektere en prosess å kjøre, så velger OS-et en av prosessene som er i Ready-tilstand. Se kap 9 & 10 om *Scheduling* (planlegging).
- Running → Exit: Skjer når den nåværende prosessen termineres av OS-et eller hvis prosessen indikerer at den har fullført kjøringen eller aborted.
- Running → Ready: Den vanligste grunnen til denne overgangen er at den kjørende prosessen har nådd det maksimalt tillate tidsbruket for en uavbrudd execution. Praktisk talt alle

multiprogrammerte operativsystemer har denne tidsbegrensningen.

- Running → Blocked: En prosess er satt til blokkert dersom den må vente. En forespørsel til OS-et er som regel i form av et systemtjeneste-kall. Det vil si et kall fra et kjørende program til en prosedyre som er en del av OS-koden.
- Blocked → Ready: En prosess i blokkert-tilstand som forflyttes til Ready når hendelsen den ventet på har blitt gjort; altså er den dermed klar til å executes.
- Ready → Exit: I noen systemer så kan en forelderprosess terminere en barnprosess når som helst. I tillegg hvis en forelderprosess termineres, så kan alle barnprosessene også eventuelt termineres.
- Blokkert → Exit: Dette gjelder som ovennevnte.

Alle disse tilstandene kan håndteres med to typer køer: en blokkert-kø og en klar-kø. For en blokkert-kø, som altså venter på andre hendelser, så kan det være effektivt å implementere flere køer; der hver kø er dedikert til de gitte hendelsene prosessene venter på. For eksempel kan man ha en egen blokkert-kø for prosesser som venter på I/O-operasjoner. Dette er hensiktsmessig da OS-et må scanne alle prosessene i en blokkert-kø for å avgjøre hvilke som skal prioriteres med hensyn til effektivitet. Dersom dispatcher dikteres av et prioritetsskjema, så kan det være hensiktsmessig med klar-køer basert på prioritetsnivåer.

Prosesser i hvilemodus (*suspended processes)*

De tre tilstandene Ready, Running, og Blocked er en systematisk å modellere oppførselen til prosessene med hensyn til OS-et. Hvis systemet ikke benytter seg av virtuelt minne må hver prosess som skal executes lastes fullstending inn i hovedminnet, og da kan det være nødvendig å ha flere tilstander til den ovennevnte modellen.

Bytting (*swapping)*

En løsning på dette problemet er såkalt *swapping.* Dette involverer å flytte deler eller hele prosessen fra hovedminnet til disk. Når ingen av prosessene i hovedminnet er i Ready-tilstand, så swapper (les: bytter) OS-et en av de blokkerte prosessene ut til disk i en hvilemodus-kø (*suspended queue*). Dette er en kø som består av eksisterende prosesser som har blitt midlertidig fjernet fra hovedminnet, eller satt i hvilemodus. OS-et henter så en annen prosess fra hvilemodus-køen eller etterspør en ny prosess. Execution fortsetter da med den nylig ankommede prosessen.

Swapping er en I/O-operasjon, hvilket vi vet at i utgangspunktet er tregt. Likevel er I/O-operasjoner til disk genrelt den raskeste formen for I/O på et system, så swapping vil likevel forbedre ytelsen.
Når OS-et utfører et swap kan den velge mellom å hente inn en av prosessene i hvilemodus-køen, eller hente inn en ny en. Det er ikke sikkert det lønner seg å hente inn en suspendert prosess fra hvilemodus-køen dersom den er blokkert.

Løsningen er et design som tar hensyn til de to uavhengige konseptene: om en prosess venter på en hendelse (er blokkert), og om en prosess har blitt byttet ut av hovedminne (suspendert eller ikke). Denne tilstanden trenger fire tilstander:

1. Ready: Prosessen er i hovedminnet og tilgjengelig for execution.
2. Blocked: Prosessen er i hovedminnet og venter på en hendelse.
3. Blocked/Suspend: Prosessen er i sekundærminnet og venter på en hendelse.
4. Ready/Suspend: Prosessen er i sekundærminne men er nå tilgjengelig for execution hvis den lastes inn i hovedminnet.

Se s. 146 hvis du vil lese om alle tilstandsovergangene... gjesp

Sidenote: Ved bruk av et virtuelt minne trenger ikke hele prosesser å ligge i hovedminnet; det er tilstrekkelig med deler av prosessen. Men selv om et virtuelt minne kan fjerne behovet for swapping, så viser det seg at ytelsen til et virtuelt minnesystem kan kollapse dersom det er for mange aktive prosesser, der alle prosesser har deler i hovedminnet. Se kap 8.

Suspendert prosess

Vi kan generalisere konseptet av en *suspendert prosess*, altså en prosess i hvilemodus.

1. Prosessen er ikke umiddelbart tilgjengelig for execution.
2. Prosessen kan og kan ikke være avventende på en hendelse. Hvis den er blokkert, så er denne blokkerte betingelsen uavhengig av den suspenderte betingelsen, og dersom den blokkerte betingelsen skulle oppheve, så betyr ikke det at prosessen kan executes med en gang.
3. Prosessen ble definert i suspendert tilstand av en agent; enten seg selv, en forelderprosess, eller av OS-et for å forhindre dens execution.
4. Prosessen kan ikke flyttes til en annen tilstand med mindre en agent eksplisitt bestiller endringen.

Tabellen under viser noen av grunnene til suspendering av en prosess.

Swapping	OS-et må gi slipp på tilstrekkelig plass i hovedminnet for å hente inn en prosess som er klar til å executes.

Andre OS-grunner	OS-et kan suspendere en bakrunns- eller nytteprosess eller suspendere en prosess som den tror skaper et problem.
Interaktiv bruker-forespørsel	En bruker kan ønske å suspendere execution av et program for grunner som debugging eller i tilkoblingen til bruk av en ressurs.
Timing	En prosess kan executes periodisk og kan suspenderes mens den venter på neste tidsintervall.
Forelderprosess-forespørsel	En forelderprosess kan ønske å suspendere en execution av et avkom (child process) for å eksaminere og modifisere den suspenderte prosessen, eller for å koordinere aktiviteten mellom avkom.