10.2 Real-time scheduling

Tuesday, April 28, 2020 3:00 AM

Tidsstyring ved sanntid blir viktigere og viktigere med autonome systemer. Operativsystemets tidsstyring er kanskje den viktigste komponenten for et sanntidssystem. Vi kan definere et sanntidssystem ved å først definere sanntidsproblemer; de er ofte knyttet til en frist, og vi kategoriserer ofte oppgavene i enten *hard real-time tasks* og *soft real-time tasks*.

Hard real-time tasks er de oppgavene som *må* imøtekomme fristen, hvis ikke kan de utgjøre skade eller skape en fatal error i systemet.
Soft real-time tasks er de oppgavene som har en frist som er ønskelig å overholde, men ikke obligatorisk.

Oppgavene kan også deles inn aperiodiske og periodiske oppgaver. De *aperiodiske oppgavene* har en frist for når den må starte og(eller fullføres, mens en *periodisk oppgave* kan gjøres enten en gang per tidsenhet *n*, eller med *n* enhter seg i mellom (per def. periodisk).

Karakteristikker ved sanntidsoperativsystemer

Sanntidsoperativsystemer kan klassifisereres ut fra ulike krav i fem generelle områder:

1. Determinisme (forutsigbarhet), e.g. tids-avbruddshåndtering
2. Responsivitet, e.g. behovsbehandling
3. Brukerkontroll, e.g. prioritetsfastlegging
4. Pålitelighet, e.g. tjenestegarantering
5. Feiltilpasning, e.g. tjenesteprioritering

I et sanntidssystem ligger fokuset på å fullføre alle hard real-time tasks innen fristen og så mange soft real-time tasks (eller starte) innen fristen. Ofte kan ikke OS-et operere direkte med frister, men designes derfor isteden til å være så responsive som mulig. En metode for å løse dette er prioritet kombinert med en klokkebasert avbruddsmekanisme, eller enda bedre å benytte umiddelbare abrudd. I et slikt tilfelle vil OS-et reagere umiddelbart med mindre det er låst i en kritisk kode-seksjon.

Tidsstyringsalgoritmer for sanntid kan videre deles inn i fire klasser:

Statisk tabell-drevet
Gitt et sett med oppgaver og deres egenskaper beregnes tidsstyringen (tabellen) offline. Denne brukes for periodiske oppgavesett, og krever at hele tidsstyringen gjøres på nytt hvis oppgavesettet skulle sendre seg. Hvilke oppgaver som skal gjøres bestemmes ved kjøretid.

Statisk prioritet-drevet med avbrudd
Gitt et sett med oppgaver og deres egenskaper tildeles hver oppgave en fast prioritet. Prioritetene kombineres med en avbrudssmekanisme. Dette brukes også for periodiske oppgavesett. Primært benyttes tidligste tidsfrist først, dernest høyest prioritet først.

Dynamisk planleggingsbasert
Når det ankommer en ny oppgave vil det før execution starter forsøkes å lage en tidsstyringsplan som inneholder de gamle oppgavene og den nye oppgaven. Hvis den nye oppgaven kan legges til i tidsstyringsplanen uten at det forhindrer imøtekomming av tidsfrister, så vil tidsstyringen endres for å tilrettelegge for den nye oppgaven i tillegg. Gjennomførbarhet bestemmes altså dynamisk ved kjøretid istedenfor statisk på forhånd.

Dynamisk beste-forsøk
Gjør ingen gjennomføringhetsanalyse på forhånd. Systemet forsøker å imøtekomme alle tidsfrister og aborterer prosessene som ikke rekker fristene. Denne prioriterer tidligste tidsfrist først, og bruker deretter en FCFS-ordning.

Tidsstyring basert på tidsfrister

De fleste sanntidsoperativsystemer er designet med hensikt om å starte sanntidsoppgavene så raskt som mulig, og dermed vektlegger hurtig avbruddshåndtering og oppgave-dispatching. Likevel er ikke sanntidsapplikasjoner like avhengig av fart, men heller timingen av fullføring (eller starting) av opppgavene. Dermed fanges ikke dette aspektet med timing opp ved bruk av prioriteter.

Kraftigere løsninger baseres på at man har ekstra informasjon om hver oppgave. Følgende informasjon om hver oppgave kan brukes til å lage nye løsninger i tidsstyringen av sanntidsoppgaver:

- **Tidspunkt ved Ready:** Tiden når en oppgave blir klar for execution. Dette kan både være aperiodiske og periodiske oppgaver.
- **Start-tidsfristen:** Dette er tidspunktet hvor oppgaven må starte.
- **Fullføring-tidsfristen:** Dette er tidspunktet hvor oppgaven må ha blitt fullført. Sanntidsapplikasjoner har som regel enten start-tidsfrister eller fullføring-tidsfrister.
- **Prosesseringstid:** Dette er tiden det tar for å execute oppgaven til den er fullført.

- **Ressurskrav:** Settet av ressurser (bortsett fra prosessoren) som oppgaven krever mens den executer.
- **Prioritet:** Måler den relative viktigheten av oppgaven.
- **Deloppgave-struktur:** En oppgave kan dekonstrueres til en obligatorisk deloppgave og en valgfri deloppgave. Den obligatoriske deloppgaven har da en hard (streng) tidsfrist.

RMS – Rate Monotonic Scheduling

RMS er en av de bedre metodene for å løse tidsstyring for periodiske oppgaver. RMS gir prioritet til oppgaver på grunnlag av perioden deres, hvorav oppgaver med kortere periode får høyere prioritet – samt blir betjent først. Oppgaver med lengre periode får lavere prioritet. For å garantere at alle tidsfrister skal møtes må summen av prosessorutnyttelsen (U = C/T) for hver oppgave holde ulikheten under; der 1 er den totale utnyttelsen til prosessoren. RMS har ofte utnyttelse til oppmot 90%, og garanterer at den når alle frister så lenge følgende uliket holder:

Der C = utførelsestiden, andel prosessortid som kreves for oppgaven, mens T = perioden til oppgaven.

Den påkrevde infromasjonen for å rangere prosesser og sjekke garanti-kravet er liten, men garanti-kravet i seg elv er vanskelig da det forutsetter en slakk på oppmot 30.7% = (1 – 0.693) i systemet (som er når antallet prosesser *n* går mot uendelig).

Det er også lettere å oppnå stabilitet med RMS. Når et system ikke kan møte alle tidsfrister på grunn av overload eller flyktige errorer, så må tidsfristene til essensielle oppgaver kunne garanteres gitt at settet med oppgaver kan tidsstyres. Med *tidligst-tidsfrist tidsstyring* skifter en periodisk oppgaves prioritet fra en periode til en annen – dette kan gjøre det vanskeligere å sikre at essensielle oppgaver imøtekommer tidsfristene sine.

Prioritetsinvertering

Invertering av prioritet oppstår når forholdene i et system tvinger en oppgave av høyere prioritet til å utføres etter en oppgave med lavere prioritet. Et eksempel er når en lavere prioritetsoppgave har låst en ressurs som en oppgave av høyere prioritet trenger. Problemet er mer kritisk ved *ubegrenset prioritetsinvertering* som er når varigheten til en prioritetsinvertering avhenger av både tiden det tar å håndtere den delte ressursen, samt de uforutsigbare handlingene til de urelaterte oppgavene. Det er to løsninger på dette problemet:

- **Prioritetsarv:** Gir oppgaver av lavere prioritet den samme prioriteten som en oppgave av høyere prioritet dersom den holder på ressursen som den andre venter på. Prioriteten økes så fort oppgaven av høyere prioritet er blokkert på ressursen, og senkes tilbake så for den frigir ressursen.

- **Prioritetstak:** Hver ressurs har en egen prioritet. Prioriteten til ressursen er høyere enn prioriteten til brukeren med høyest prioritet. Tidsstyreren tildeler denne prioriteten dynamisk til oppgaver som aksesserer ressursen. Så fort oppgaven er ferdig med ressursen settes den tilbake til normal.