9.2 Scheduling algorithms

Sunday, April 26, 2020 2:57 AM

Kortsiktig tidsstyringskriterier

Hovedmålet med kortsiktig tidsstyring er å tildele prosessortid på en måte som optimaliserer en eller flere aspekter av systemets oppførsel. Det er et sett med kriterer som er etablert for ulike situasjoner som krever tidsstyring. Disse kriteriene kan igjen kategoriseres inn i to dimensjoner: bruker-orienterte og system-orienterte kriterier.

Bruker-orienterte kriterier relaterer til oppførselen til systemet sett fra den individuelle bruker eller prosess. Et eksempel på dette er responstiden i et interaktivt system.

System-orienterte kriterier fokuserer på effektive og effektiv bruk av prosessoren. Et eksempel på dette er gjennomstrømming (throughput), som er raten prosesser fullføres. Dette er åpenbart et mål for systemytelse og noe vi ønsker å optimere.

I tillegg har vi ytelse-relaterte kriterier som er kvantitative og kan lett måles, eksempelvis responstid og gjennomstrømming. Andre kriterer kalles *Other*, og innbefatter målinger som ikke er lette å måle eller analysere – eksempelvis forutsigbarhet.

Tabellen under viser de viktigste kriteriene og deres sammenheng. Merk at de er gjensidige avhengige, så det er umulig å optimalisere alle på én gang.

###	Bruker-orientert	System-orientert
Ytelse-orientert	**Gjennomløpstid (Turnaround time):** Tiden fra innsending av en prosess til den er ferdig. **Responstid:** Tiden det tar fra innsending av en forespørsel inntil responsen mottas. **Tidsfrister:** Når tidsfrister for gjennomføring av prosesser settes, så maksimerer tidsstyreren andelen tidsfrister som imøtekommes.	**Gjennomstrømming:** Tidsstyreren burde forsøke å maksimere antall prosesser som fullføres per tidsenhet. Dette måler hvor mye arbeid som blir gjort. **Prosessorutnyttelse:** Dette er andelen tid hvor prosessoren er opptatt. Dette er viktig i et delt system.
Ikke ytelse-orientert *(Other)*	**Forutsigbarhet:** En jobb burde kunne utføres med samme tid og kostnad uavhengig av systemets last. Stor variasjon I responstid og gjenomløpstid er forvirrerende for brukerne.	**Rettferdighet:** Ved fravær av brukerveiledning skal alle prosesser behandles likt og ingen skal ende opp med starvation. **Prosessprioritet:** Tildeler ulik grad av prioritet hos ulike prosesser som tidsstyrereren må overholde. **Ressursbalanse:** Tidsstyreren skal sørge for at alle ressurser er i bruk.

Bruken av prioriteter

I mange systemer er det vanlig å ilegge hver prosess en grad av prioritet. Utfordringen som oppstår med dette er at prosesser med lavest prioritet risikerer starvation. For å løse dette kan for eksempel prioriteten til prosesser endres underveis. Dette vil gi noe overhead da OS-et må oftere sjekke tilstanden og ventetiden til ulike prosesser.

Alternative tidsstyringsmetoder

Vi ser nærmere på en del ulike tidsstyringsmetoder med deres fordeler og ulemper i denne seksjonen.

Seleksjonsfunksjonen bestemmer hvilken prosess blant prosessene som er klare (i Ready-tilstand) som skal kjøre neste gang. Funksjonen kan være basert på prioritet, ressurskrav, eller andre kjørings-karakteristikker for prosessen. I dette tilfellet ser man på:

> *w* = tid brukt i systemet så langt
> *e* = tid brukt under utførelse så langt
> *s* = total tjenesetid som kreves av prosessen (inkludert *e*) - generelt må dette bestemmes av brukeren.

Valgmodus spesifiserer ved hvilket tidspunkt seleksjonsfunksjonen skal utføres. Det varierer mellom to kategorier:

> **Ikke avbrytbar (non-preemptive):** Prosessen som kjører fortsetter inntil den terminerer eller blokkerer seg selv for I/O- eller en annen OS-tjeneste.
> **Avbrytbar (preemtive):** Den kjørende prosessen kan avbrytes og flyttes til Ready-tilstanden av OS-et. Dette kan gjøres til fordel for en annen prosess når et avbrudd skjer fordi en ny prosess er I Ready-tilstanden, eller ved klokke-avrbudd. Dette krever mer overhead, men kan gi bedre tjeneste til den totale menden prosesser, ettersom ingen prosess får monopol på prosessoren.

Tidsstyringsalgoritmer / Scheduling algorithms

Turnaround Time – TAT
TAT er tiden T_r (r for *resident*) som et element bruker i systemet (ventetid + tjenestetid). Et nyttig mål kan være normalisert gjennomløpstid (turnaround time), som er forholdet mellom turnover tid og tjenestetid. Det indikerer den relative forsinkelsen som prosessen opplever.

First-come-first-served – FCFS
Denne algoritmen har verken avbrudd eller prioritering. Aktive prosesser kjøres til de er fullført eller til de blokkers av forespørsel på I/O eller annen OS-tjenester. Denne metoden gjør det bedre ved prosessor-avgrenset prosesser enn I/O-avgrensede. FCFS medører ueffektiv bruk av prosessor og I/O-enheter, og er som regel ikke bra alene i et uniprosessor-system. Kombinert med *prioritering*, kan den ha en viss nyte. Med en kø for hver prioritetsgrad, kan en FCFS-algoritme benyttes på hver kø.

Round Robin / Time slicing – RR
RR fordeler tiden på de ulike prosessene i periodiske intervaller. Ved avbrudd byttes prosessen ut i samme ordning som FCFS, bare at tidsfordelingen er mer rettferdig. Utfordringen med RR er å velge størrelsen på tidssegmentene. Håndteringen av avbruddene og tidsstyring samt dispatching inkluderer også noe overhead. Derfor burde veldig små segmenter unngås. Det er vanlig at segmentet settes lik tiden det tar for en typisk interaksjon eller prosessfunksjon, hvis ikke vil de fleste prosesser kreve to segmenter. En ulempe med RR er at den behandler prosessor-avgrensede og I/O-avgrensede prosesser likt, selv om prosesser som er I/O-avgrenset som regel er kortere. Dette gir dårlig ytelse for I/O-avgrensede prosesser og økt variasjon i responstid. Dette kan løses noe ved bruk av Virtuell Round Robin, som har en egen kø for I/O-prosesser som prioriteres over prosessor-avgrensede prosesser.

Shortest Process Next – SPN
SPN-algoritmen har ikke avbrudd, men prioriterer prosessen som forventes å ha kortest prosesseringstid til å utføres først. Overordnet prestasjon øker, særlig i form av responstid. Dog vil responstidens variasjon øke, som gjør at forutsigbarheten synker. Dessuten kan det være vanskelig å vite i forkant prosesseringstid for hver prosess. En metode for å bestemme prosesseringstid kan være å bruke gjennomsnitt eller eksponentielt gjennomsnitt. En slik ordning favoriserer nylige prosesser (som dermed

også oftest er mer korrekt). SPN-algoritmen kan medføre starvation for større prosesser, så lenge det er mange små prosesser.

Shortest Remaining Time – SRT
SRT-algoritmen er en avbruddsversjon av SPN. Tidsstyreren favoriserer prosesser med minst gjenværende prosesseringstid, inkludert nye prosesser. Dette skiller seg fra favoriseringen av lange prosesser i FCFS. SRT må dermed også ha et estimat på prosesseringstid – hvilket igjen kan resultere i starvation. I motsetning til RR er det ingen ekstra avbrudd, hvilket reduserer overhead. Likevel må tjenestetiden bli registrert, som øker overhead. SRT skal gi bedre gjennomløps-ytelse enn SPN fordi korte jobber får umiddelbar prioritet.

Highest Response Ratio Next – HRRN
Generelt kan vi ikke vite hva tjenestetiden er på forhånd, men vi kan approksimere den. Enten basert på historie, eller input fra konfigurasjonshåndtereren. Når den nåværende prosessen fullføres eller blokkeres, velges den neste prosessen med høyest verdi av respons-forholdstallet *R*. Dette tar hensyn til alderen til prosessen. Selv om små prosesser favoriseres (større tjenestetid ville gitt større forholdstall), vil en eldrende prosess som ikke velges få økt *R* for å forhindre starvation. Likningen under gir verdien til forholdstallet *R*.

Feedback – FB
En annen måte å favorisere korte jobber på er å straffe jobber som har kjørt lengre. FB baseres dermed på den nåværende execution-tiden så langt. TIdsstyringen gjøres ved tidsavbrudd og en dynamisk prioriteringsmekanisme. Når en prosess først inntrer systemet legges den i en kø. Etter første avbrudd og først når prosessen er i Ready-tilstand igjen legges den til i neste kø. For hvert avbrudd legges prosessen i en kø av lavere prioritet. I alle køene bortsett fra den av lavest prioritet, benyttes det en FCFS-mekanisme. Når prosessen når køen med lavest prioritet legges den tilbake i første kø slik at den kan fullføre execution. Køene behandles dermed ved RR. FB kan medføre starvation dersom mange mindre prosesser entrer systemet. For å kompensere for dette kan man variere tidsavbruddet for hver kø, e.g. at første kø har tidsavbrudd etter 1 tidsenhet, neste kø etter 2 tidsenheter og de påfølgende køene etter 2^n tidsavbrudd. En annen metode som kan forhindre starvation er å sette en tidsgrense for utførelse, der en prosess vil få høyest prioritet etter en gitt tid.

Sammenligning av ytelsen hos hver algoritme

Denne sammenligningen er generell, og det viktig å huske på at situasjonene vil variere, så en algoritme er aldri optimal for *alle* situasjoner.

Kø-analyse
Ved å favorisere kortere jobber ser man generelt at den gjennomsnittlige gjennomløptstiden synker. Forbedringen er størst ved bruk av avbrudd, men man kan notere at den overordnede ytelsen ikke påvirkes mye av dette. Ved å skille prioritetsklasser kan vi øke ytelsen ytterligere. Ved bruk av prioritet uten avbrudd er forbedringene betydelig, noe som forsterkes enda mer ved bruk av avbrudd. Prosesser som er lengre og av lavere prioritet vil få dårligere ytelse ved denne ordningen.

Simuleringsmodell

Med utgangspunkt i modellene over kan vi se nærmere på ytelsen til hver algoritme.
Figur 9.14 viser normalisert gjennomløpstid, mens Figur 9.15 viser gjennomsnittlig ventetid. For gjennomløpstidene ser vi at FCFS gjør det dårligst, hvorav en tredjedel av prosessene har en normalisert gjennomløpstid større enn 10 ganger tjenestetiden; hvorav disse er de korteste prosessene. På den andre siden er den absolutte ventetiden uniform, hvilket er forventningsrett fordi tidsstyring er uavhengig av tjenestetid. RR har en gjennomsnittlig gjennomløpstid på 5, bortsett fra de korteste prosessene som executer med mindre enn ett kvantum, og behandler dermed alle prosesser rettferdig. SRT gjør det bedre enn SPN bortsett fra de 7% lengste prosessene. Uten avbrudd har vi sett at FCFS favoriserer lange prosesser, mens SPN favoriserer korte. HRRN fungerer som et kompromiss mellom disse to, hvilket reflekteres i figurene. Til slutt ser vi at FB gjør det veldig bra for små prosesser.

Rettferdig-delt tidsstyring / Fair-share scheduling (FSS)

I et multibruker system kan individuelle bruker-applikasjoner eller jobber være organisert i flere prosesser (eller tråder). I dette tilfellet ligger ikke fokuset på hvordan en bestemt prosess utfører jobben, men heller på hvordan settet av prosesser som utgjør en applikasjon opptrer. Derfor kan det være nyttig å lage en ordning som gjør tidsstyringsbeslutninger på grunnlag av hele settet. Dette kalles *faire-share scheduling*. Hver bruker har en vekt som definerer andelen av ressursene (prosessoren) som dedikeres til brukeren. Systemet deler brukerne inn i et sett med rettferdig delte grupper, og tildeler andelen prosessor-ressurser til hver gruppe. Tidstyringen gjøres på grunnlag av prioritet. Prioriteten til en prosess synker i takt med at prosessen bruker prosessoren og ettersom gruppen pprosessen tilhører benytter prosessoren.