Les 3: Procesplanning

Pareto principle — States that for many phenomena
80% of consequences stem from 20% of the causes.

best3-1

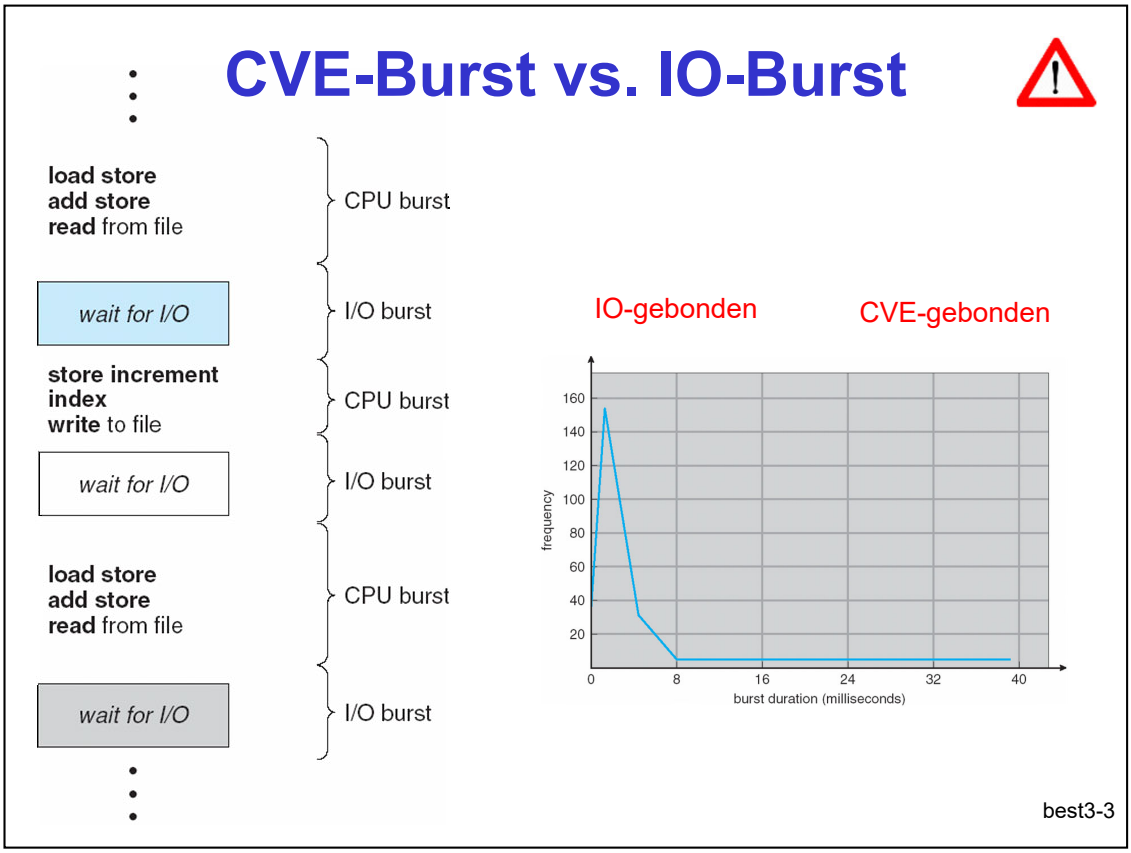
Gemultiprogrammeerde systemen zijn gebaseerd op een planningsalgoritme dat de beschikbare rekentijd van een processor verdeelt over de processen die klaar staan om uitgevoerd te worden. Multiprogrammering laat toe om, indien een proces moet wachten, snel om te schakelen naar een ander proces (dat zich op dat ogenblik reeds in het geheugen bevindt). Op deze manier worden de processorcycli die anders tijdens het wachten verloren zouden gaan toch nog gerecupereerd en wordt het CVE-gebruik gemaximaliseerd.

Ofschoon we over een procesplanner (process scheduler) spreken, is het in feite een draadplanner. In besturingssystemen met draden zijn het immers de draden die gepland worden, niet de processen (deze hebben geen activiteiten, maar dienen enkel maar als container van de systeemmiddelen).

Verder is het belangrijk te weten dat de planningsalgoritmen die in deze les behandeld worden in principe bruikbaar zijn op alle planningsniveaus (korte, middellange en lange termijn), zij het met andere input en met andere parameters.

Overzicht

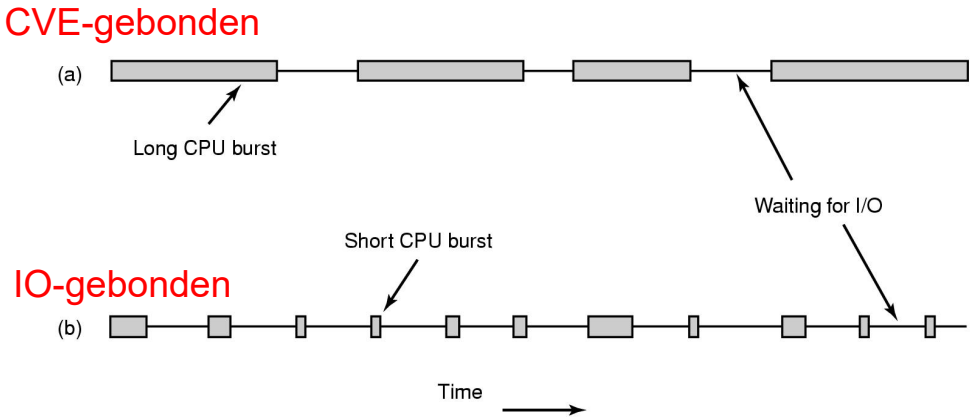
- Wat is procesplanning?
- Planningsalgoritmen
  - Monoprocessorsystemen
  - Multiprocessorsystemen
- Concrete gevallen



Tijdens zijn uitvoering zal een proces (of een draad) afwisselen tussen CVE-activiteit en IO-activiteit. Men noemt de tijd die verstrijkt tijdens de CVE-activiteit of de IO-activiteit een burst. De IO-burst bestaat uit wachttijd op een randapparaat en kan noch door de processor, noch door de planner beïnvloed worden. De CVE-bursts geven aan hoelang er dient gerekend te worden tussen twee IO-operaties. Bij interactieve processen zal de hoeveelheid rekenwerk tussen twee IO-operaties doorgaans zeer kort zijn. Bij rekenprocessen (b.v. de inversie van een grote matrix) kan de CVE-burst bijzonder lang worden.

In de dia staat een histogram van de CVE-burst. Hieruit blijkt dat de meeste CVE-bursts kleiner zijn dan 8 ms, maar dat er occasioneel ook heel lange bursts voorkomen. Bij het ontwerpen van een planningsalgoritme zullen we er dus moeten voor zorgen dat de talrijke kleine bursts efficiënt afgehandeld worden, maar dat dit de processen met extreem lange bursts niet te veel benadeelt.

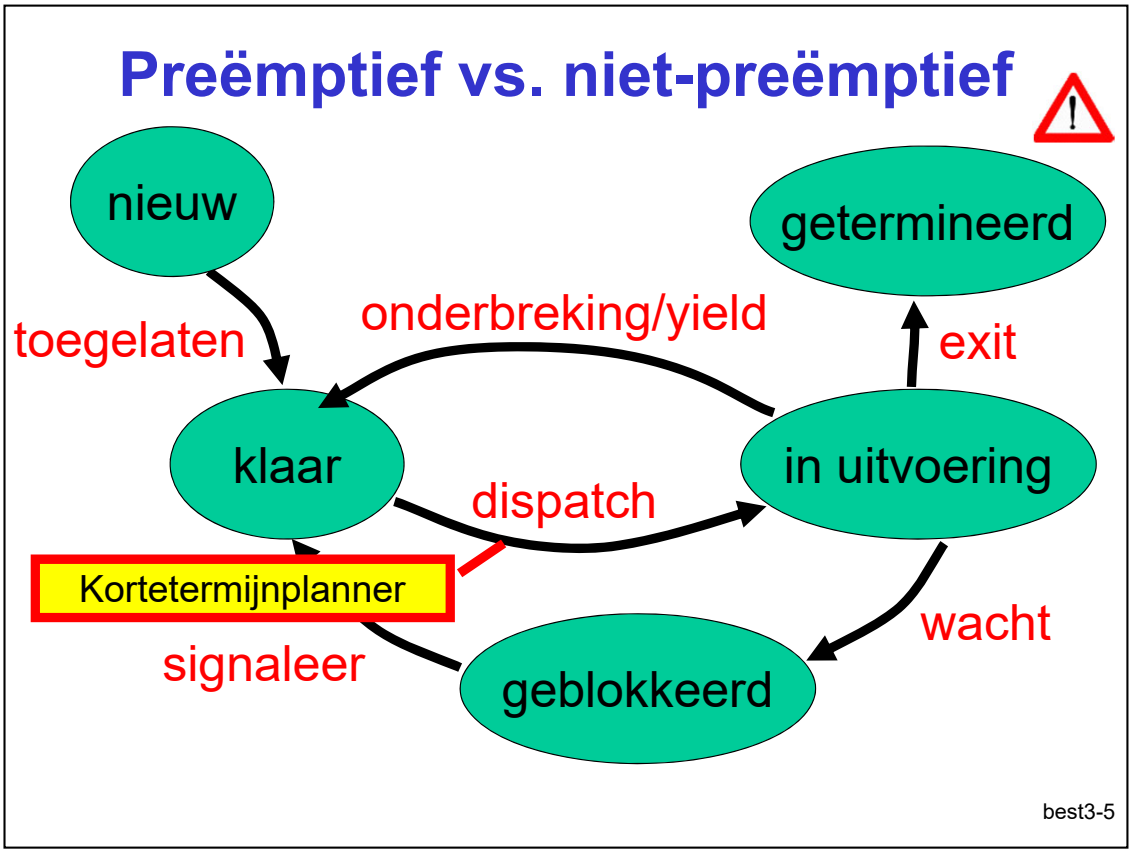
IO-gebonden vs. CVE-gebonden processen



Moderne applicaties zijn doorgaans sterk io-gebonden

best3-4

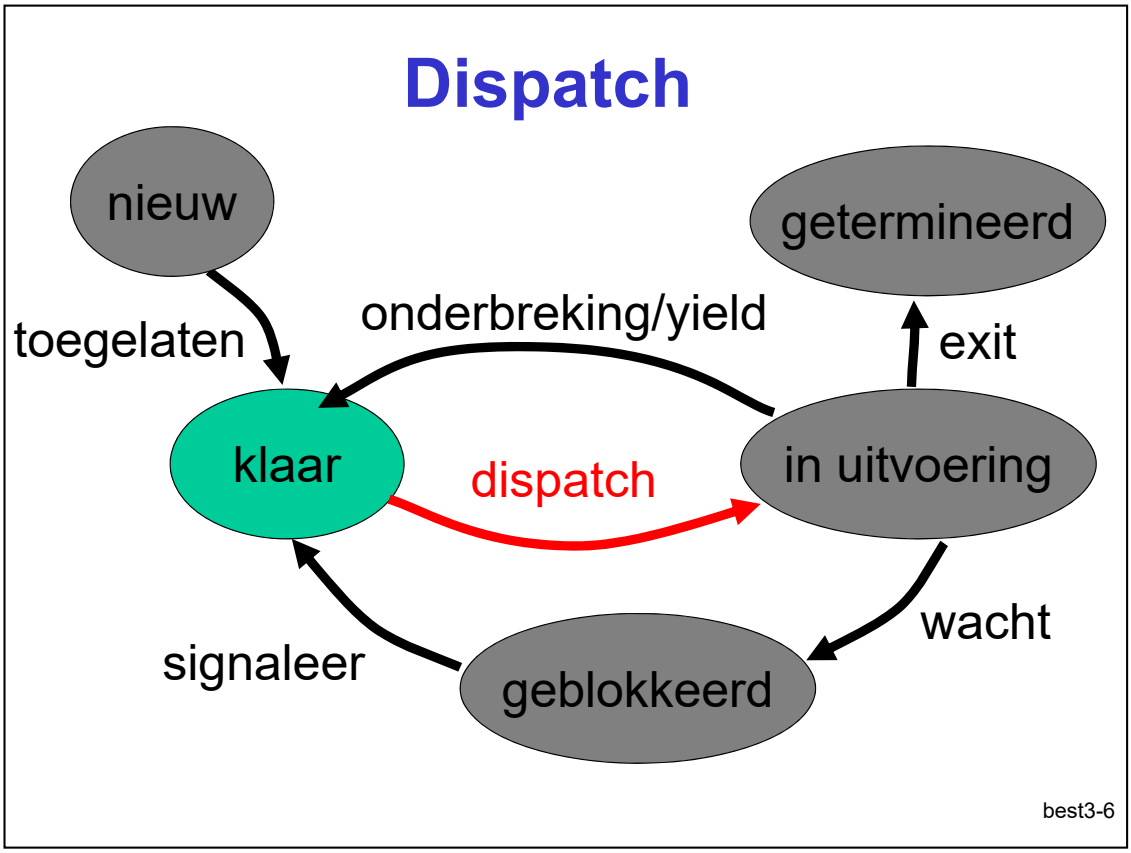
Op basis van de grafiek uit de vorige dia kunnen processen opgedeeld worden in twee grote categorieën: processen met relatieve lange CVE-bursts die CVE-gebonden (CPU-bound) genoemd worden, en processen met relatief korte CVE-bursts die IO-gebonden (IO-bound) genoemd worden. Dit is een belangrijk kenmerk van een proces. Ideaal zal een verzameling van processen een aantal CVE-gebonden en een aantal IO-gebonden processen bevatten. De lange wachttijden van de IO-gebonden processen kunnen dan opgevuld worden met de lange CVE-bursts van de CVE-gebonden processen. Een onevenwicht tussen de beide categorieën zal leiden tot een niet-optimaal gebruik van het computersysteem (ofwel zal de processor ondergebruikt worden, ofwel zal het IO-subsysteem ondergebruikt worden).



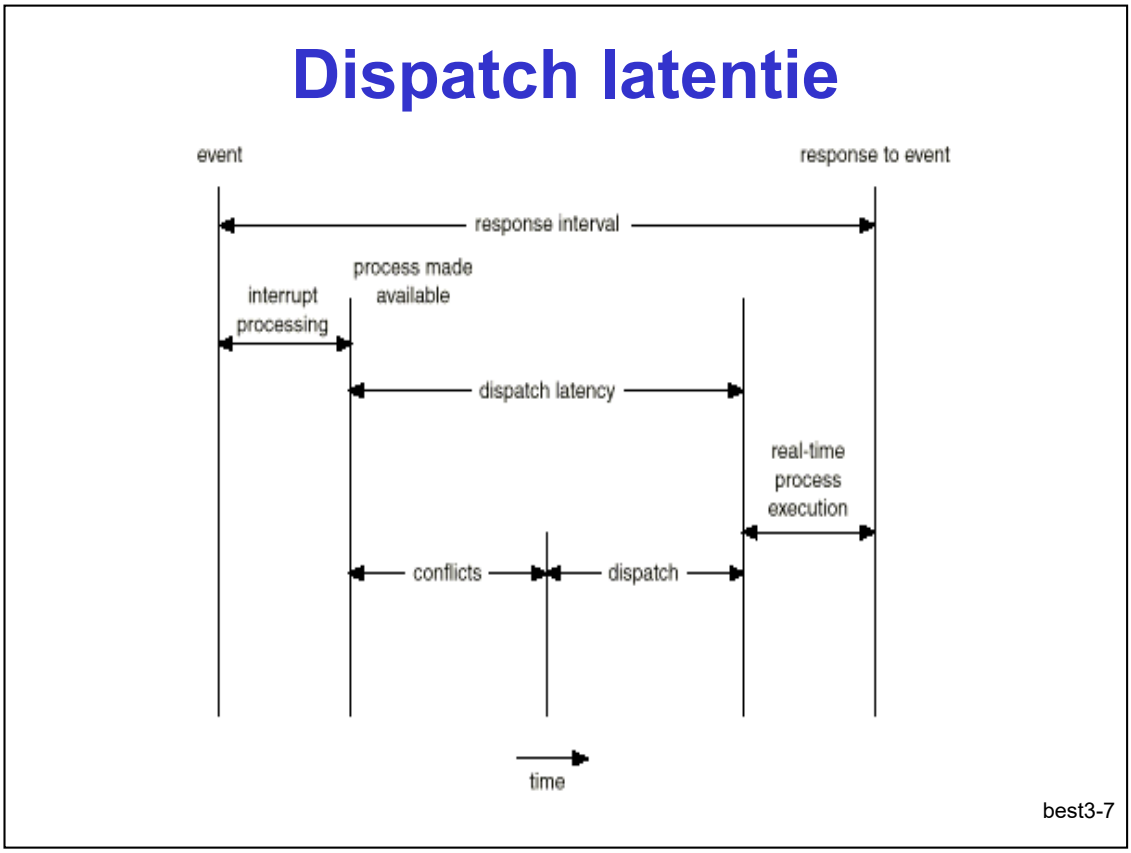
De kortetermijnplanner (ook procesplanner of CVE-planner genoemd) selecteert een geschikt proces uit de klaarlijst. De kortetermijnplanner komt tussen bij de volgende vier overgangen:

1. Van ‘in uitvoering’ naar ‘geblokkeerd’
2. Van ‘in uitvoering’ naar ‘klaar’
3. Van ‘geblokkeerd’ naar ‘klaar’
4. Van ‘in uitvoering’ naar ‘getermineerd’

Indien de planner enkel opgeroepen wordt in het geval 1, 4 en het geval ‘yield’ van 2 dan spreekt men van niet-preëmptieve planning (soms ook coöperatieve planning genoemd). Indien de planner opgeroepen wordt in de vier gevallen, dan spreekt men over preëmptieve planning. Om een preëmptieve planningsstrategie te kunnen ondersteunen moet men kunnen beschikken over de hardware om processen te onderbreken (timers e.d.m.). Bovendien is de kern van een preëmptief besturingssysteem ook een heel stuk complexer omdat processen (en dus in principe ook het besturingssysteem zelf) op gelijk welk ogenblik kunnen onderbroken worden. Gegevensstructuren waarvan verschillende onderdelen atomair moeten aangepast worden om consistent te blijven zullen dus extra synchronisatie vereisen. In sommige besturingssystemen wacht men daarom bij het optreden van een onderbreking tijdens de uitvoering van een systeemoproep totdat deze systeemoproep afgelopen is alvorens de onderbreking af te handelen. Dit maakt de kernel eenvoudiger, maar introduceert voor sommige toepassingen (in ware tijd) een onaanvaardbare vertraging.



Het is de taak van de dispatcher om ervoor te zorgen dat het gekozen proces op de processor belandt. Dit houdt in dat de context veranderd wordt, dat er omgeschakeld wordt van gebruikersmode (de procesplanner wordt door de kernel uitgevoerd en loopt dus in systeemmode). Tenslotte wordt er gesprongen naar het begin van het proces.



De dispatch latentie is de tijd nodig om een proces te stoppen en een ander in te laden.

Een proces wordt gestopt door het optreden van een timeronderbreking. De eerste vertraging die optreedt is die van de afhandeling van de onderbreking. Vervolgens wordt er nagegaan of het onderbroken proces zich wel in een onderbreekbare toestand bevindt (zo zullen sommige besturingssystemen niet toelaten dat een proces onderbroken wordt tijdens de uitvoering van een systeemoproep). Mogelijks zal het proces nog wat tijd nodig hebben om dergelijke conflicten op te lossen. Uiteindelijk vindt de eigenlijke dispatch of het inladen van het nieuwe proces plaats.

In de praktijk duurt een contextwisseling tussen de 1 en 3 µs op een Pentium processor.

Planningscriteria

- **CVE-gebruik** (CPU-Utilization) – de belasting van de processor, uitgedrukt in procent.
- **Doorvoer** (throughput) – # aantal processen dat afgewerkt wordt per tijdseenheid
- **Doorlooptijd** (turnaround time) – de totale tijd tussen opstarten en termineren van een proces
- **Wachttijd** (waiting time) – de tijd dat een proces in de klaarlijst doorbrengt
- **Antwoordtijd** (response time) – de tijd die verloopt tussen een aanvraag en het begin van antwoord (niet het einde)

best3-8

Afhankelijk van de omgeving waarin een computersysteem zal gebruikt worden, kan men verschillende metrieken beschouwen om de prestatie van een planner te evalueren.

CVE-gebruik: Het CVE-gebruik kan variëren tussen 0% en 100%. De CVE wordt niet gebruikt indien er geen enkel proces kan gevonden worden dat klaar staat om uitgevoerd te worden. In dat geval zal de CVE in de planner in een wachtlus terechtkomen (ofwel het idle proces uitvoeren) totdat er een proces naar de toestand `klaar' overgaat.

Doorvoer: Dit is het aantal processen dat per tijdseenheid afgewerkt wordt. Voor kleine processen kan dit 10 per seconde zijn, voor grote processen een paar per uur of per dag. Deze maat wordt vooral gebruikt in omgevingen (bijvoorbeeld banken) waar de belasting van een systeem bestaat uit een aantal processen die dag na dag opnieuw uitgevoerd moeten worden.

Doorlooptijd: Dit is een criterium dat slaat op de uitvoering van één enkel proces. Het is de tijd die verloopt tussen het aanbieden van het proces en het termineren ervan. Dit omvat dus de tijd die verstrijkt terwijl het proces zich in het systeem bevindt, d.i. de tijd dat het proces uitgevoerd wordt, de tijd dat het proces aan het wachten is op IO, en de tijd dat een proces aan het wachten is op de CVE.

Wachttijd: De planningstrategie heeft geen invloed op de tijd die een proces moet kunnen beschikken over de processor, of moet wachten op IO. De strategie heeft vooral een invloed op de tijd dat een proces dat klaar is om uitgevoerd te worden moet wachten op de CVE. De wachttijd is de totale tijd dat een proces aan het wachten is in de toestand `klaar'.

Antwoordtijd: Daar waar de doorlooptijd slaat op de tijd nodig om een totaal proces uit te voeren en dus ook in belangrijke mate beïnvloed wordt door de snelheid van de randapparaten, slaat de antwoordtijd op de tijd die verstrijkt tussen het aanleggen van een stimulus aan het systeem, en het begin van reactie erop. De antwoordtijd is de tijd waarmee een interactieve gebruiker het meest geconfronteerd wordt.

Algemene optimalisatiecriteria

- Max CVE-gebruik
- Max doorvoer
- Min doorlooptijd
- Min wachttijd
- Min antwoordtijd

- Fairness: gelijke behandeling van gelijkaardige processen
- Max IO-gebruik

best3-9

Het doel van een planningstrategie zal steeds zijn om het CVE-gebruik en de doorvoer zo hoog mogelijk te maken, en de doorlooptijd, wachttijd en antwoordtijd zo klein mogelijk. Hierbij zijn echter wel een aantal randbedenkingen te maken.

Het heeft geen zin om de antwoordtijden in een interactief programma kleiner proberen te maken dan wat door een menselijke gebruiker waargenomen kan worden. Het verschil tussen 5 en 10 ms is hier niet belangrijk.

Soms is het belangrijker om de maxima te minimaliseren in plaats van de gemiddelde waarden. De modale gebruiker zal minder geïrriteerd geraken met uniforme antwoordtijden die 30% trager zijn dan met een gemiddelde vertraging van 10% met af en toe uitschieters van 1000%. Anders gezegd, het kan belangrijker zijn om de variantie op de antwoordtijden klein te houden, eerder dan de antwoordtijden zelf. Voorspelbaarheid in deze blijkt een belangrijke troef te zijn.

Het optimaliseren van één bepaald criterium heeft niet noodzakelijk ook een positief effect op de andere criteria. Zo zal het minimaliseren van de wachttijd van 1 proces bereikt worden door absolute prioriteit te geven aan dit proces, en zelfs tijdens het uitvoeren van een IO-operatie te blijven wachten om meteen na het ontvangen van een signaal de uitvoering te kunnen verderzetten. De wachttijd van het betrokken proces zal 0 zijn, maar de wachttijd van de andere processen kan zeer nadelig beïnvloed worden.

Specifieke optimalisatiecriteria

Batchsystemen

- Doorvoer maximaliseren
- Doorlooptijd minimaliseren
- CVE-gebruik maximaliseren

Interactieve systemen

- Antwoordtijd minimaliseren
- Fairness garanderen

Ware-tijdsystemen

- Deadlines respecteren
- Voorspelbaarheid

best3-10

Afhankelijk van het soort van toepassingsdomein kunnen sommige optimalisatiecriteria zwaarder wegen dan andere.

Overzicht

- Wat is procesplanning?
- Planningsalgoritmen
  - Monoprocessorsystemen
  - Multiprocessorsystemen
- Concrete gevallen

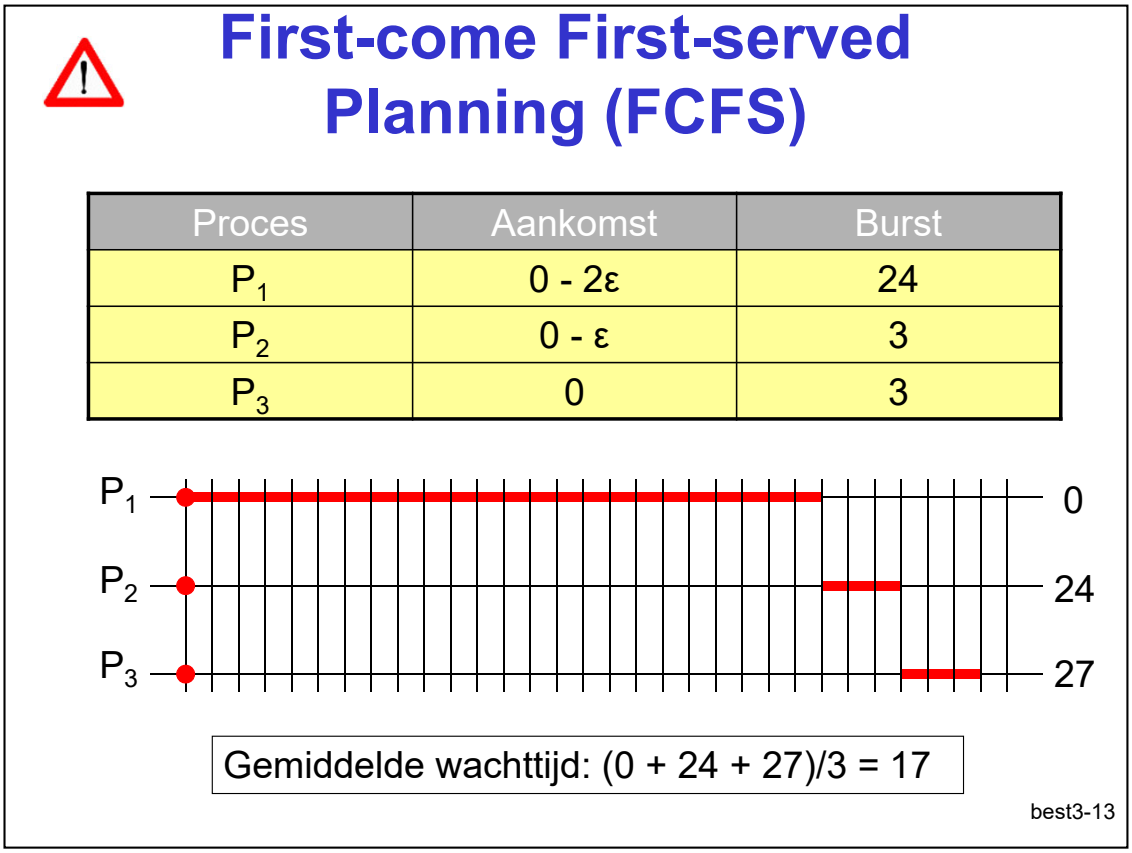
best3-11

Planningsalgoritmen

1. First-come First-served (FCFS)
2. Shortest Job First (SJF)
3. Shortest Remaining Time First (SRTF)
4. Prioriteit
5. Round Robin
6. Highest Response Ratio Next
7. Gewaarborgde planning
8. Fair Share
9. Loterijplanning
10. Multilevel Queue
11. Multilevel Feedback Queue
12. Earliest Deadline First
13. Rate Monotonic Scheduling

best3-12

Dit is de (vrij lange) lijst van planningsalgoritmen die vervolgens besproken zullen worden.



De meest eenvoudige planner is de first-come first-served (FCFS) planner. Deze voert de processen uit in de volgorde waarin ze aankomen in de klaarlijst. In dit voorbeeld gaan we ervan uit dat er op tijdstip 0 drie processen zijn. De volgorde wordt vastgelegd door de aankomsttijd die we hier aangeven, ε vroeger te leggen. Het eerste proces is kennelijk CVE-gebonden, de twee andere IO-gebonden.

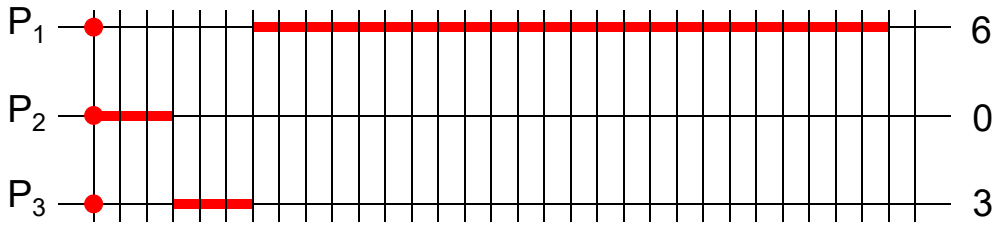
In het planningsschema (Gantt-grafiek) worden de aankomsttijden aangegeven door middel van een bolletje voor het corresponderende proces (elk proces heeft zijn eigen tijdlijn).

Aangezien FCFS niet-preëmptief is, zullen de bursts uitvoeren totdat het proces hetzij blokkeert hetzij termineert. Deze planner is zeer eenvoudig implementeerbaar en rechtvaardig (processen kunnen elkaar niet inhalen).

In wat volgt zullen we de kwaliteit van een planner uitdrukken aan de hand van de metriek gemiddelde wachttijd. De wachttijd is de tijd die de processen in de klaarlijst doorbrengen. De wachttijd staat per proces rechts van de tijdlijn aangegeven. De gemiddelde wachttijd in dit voorbeeld is 17 eenheden (dat kunnen ms zijn, of tijdskwanta of wat dan ook).

First-come First-served Planning (FCFS)

Proces	Aankomst	Burst
P_1	0	24
P_2	0 - 2ε	3
P_3	0 - ε	3



Gemiddelde wachttijd: (6 + 0 + 3)/3 = 3 << 17

best3-14

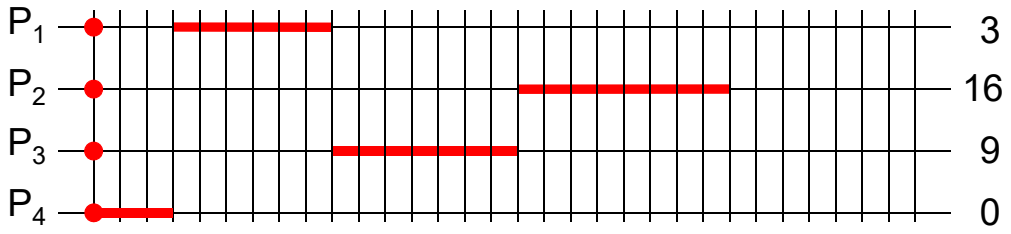
Problemen met deze planner zijn wel dat hij geen rekening houdt met de aard van het proces (IO-gebonden of CVE-gebonden), en dat hij zeer gevoelig is aan de lengte van de bursts. Door de volgorde van de processen enigszins te wijzigen kan men de gemiddelde wachttijd aanzienlijk reduceren.

De regel is dat de wachttijd kleiner wordt naarmate men erin slaagt om de korte bursts eerst te plannen. Op deze manier vermijdt men het zgn. konvooi-effect waarbij tal van processen met een kleine burst moeten wachten op een proces met een lange burst.

Deze gevoeligheid van FCFS aan de burstlengte maakt dit planningsalgoritme onbruikbaar voor ware-tijdssystemen, en voor interactieve systemen. Niemand zou aanvaarden dat een proces met een burst van 10 min voorgaat op de interactieve processen.

Shortest job first planning (SJF)

Proces	Aankomst	Burst
P_1	0	6
P_2	0	8
P_3	0	7
P_4	0	3



Gemiddelde wachttijd: (3 + 16 + 9 + 0)/4 = 7

best3-15

De conclusie van de vorige dia suggereert meteen een kandidaat voor een optimaal planningsalgoritme, namelijk shortest job first (SJF). Door die bursts met de kortste burstlengte eerst te plannen (indien gelijk FCFS), zorgt men er automatisch voor dat de wachttijd van de andere processen ook zo kort mogelijk is (want de burstlengte van het geplande proces is automatisch een deel van de wachttijd van de resterende processen).

Jammer genoeg is SJF een niet-causaal algoritme. Probleem is dat men de burstlengte nodig heeft nog voor deze uitgevoerd werd, en dat het onmogelijk is om de precieze burstlengte te kennen zonder de burst effectief uit te voeren. In de praktijk zal men dan ook moeten werken met een schatting van de burstlengte.

Op het allerhoogste niveau — in de jobplanner, dit is de planner die beslist wanneer en welke job er aan het systeem aangeboden wordt — kan men de opgegeven vermoedelijke uitvoeringstijd van een proces gebruiken. Deze vermoedelijke uitvoeringstijd moet dan wel door de gebruikers opgegeven worden, en wel liefst zo correct mogelijk. Enerzijds heeft de gebruiker er baat bij om de uitvoeringstijd zo klein mogelijk te schatten opdat de job zo snel mogelijk zou geselecteerd worden voor uitvoering (SJF), en anderzijds moet de opgegeven tijd wel volstaan om de job uit te voeren. Zoniet wordt hij halverwege afgebroken en gaan de berekende resultaten verloren. SJF wordt voor jobplanning veel gebruikt.

Op het niveau van de procesplanning kan men niet zomaar aan de gebruikers vragen om een schatting te maken van de CVE-bursts. Daar zal men zijn toevlucht zoeken tot een voorspelling op basis van een extrapolatie van de reeds uitgevoerde bursts. Hierbij gaan we ervan uit de het gedrag van het programma niet zoveel wijzigt in de tijd.

Burstschatting

$$\tau_{n+1} = \alpha\, t_n + (1 - \alpha)\, \tau_n$$

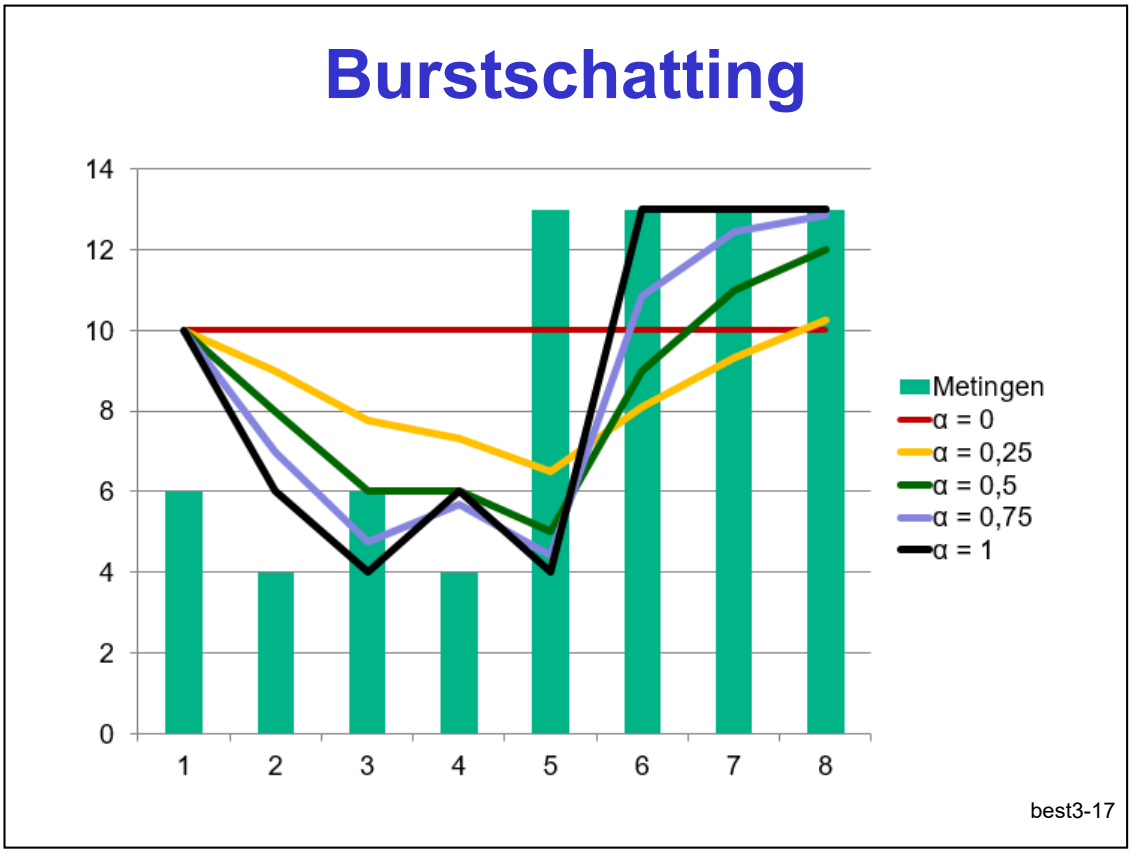
τ_n : voorspelling burst n
t_n : werkelijke lengte van burst n
α : wegingsfactor

Exponentieel gemiddelde:

$$\tau_{n+1} = \alpha\, t_n + (1 - \alpha)\, \alpha\, t_{n-1} + \ldots$$
$$+ (1 - \alpha)^j\, \alpha\, t_{n-j} + \ldots$$
$$+ (1 - \alpha)^{n+1} \tau_0$$

best3-16

In de bovenstaande formule is de schatting van de volgende burst een gewogen gemiddelde van de vorige burst en van de schatting van de vorige burst. Indien men in die formule alle burstschattingen vervangt door hun formule, dan krijgt men de formule onderaan de dia. Dan wordt meteen duidelijk waarom men de schatting van de burst het exponentieel gemiddelde noemt van alle voorgaande bursts.

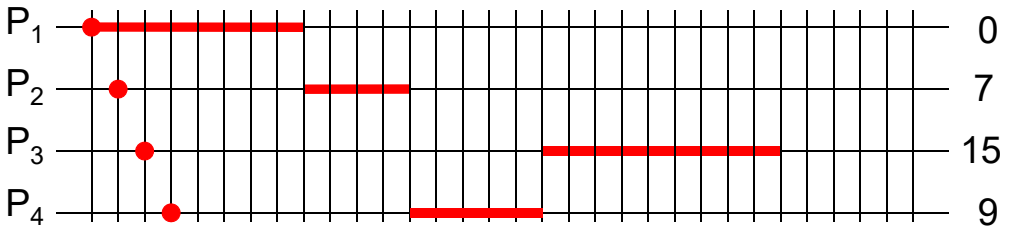


Met de factor α kan men de snelheid instellen waarmee het exponentieel gemiddelde de werkelijke burstevolutie zal volgen. Indien men α = 0 kiest is men blind voor wat er zopas gebeurd is, en kiest men dus voor de vaste τ_0, dit is de schatting waarmee men het systeem initieel opstart. Indien men α = 1 kiest, dan negeert men de totale voorgeschiedenis en kijkt men enkel naar de laatste burst.

De tussenliggende waarden laten toe om de schatter gevoeliger of minder gevoelig te maken voor de recente evolutie van de bursts.

Shortest job first planning

Proces	Aankomst	Burst
P_1	0	8
P_2	1	4
P_3	2	9
P_4	3	5



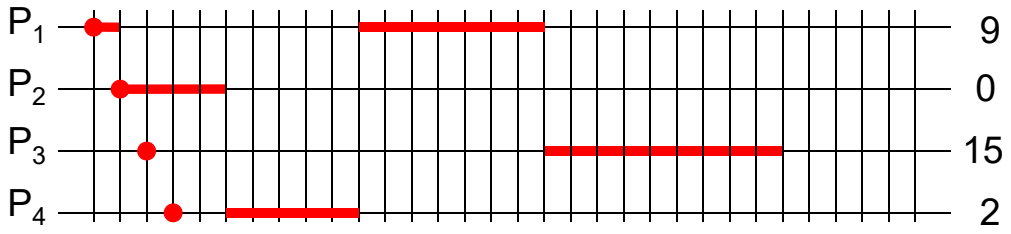
Gemiddelde wachttijd: (0 + 7+15+9)/4 = 7,75

best3-18

Het (niet-preëmptieve) SJF algoritme is optimaal voor een gegeven verzameling van processen. Indien er zich echter continu processen aanbieden (wat meestal het geval is), dan kan het voorkomen dat er zich een proces met een kortere burst aanbiedt, dan het proces dat aan het uitvoeren is. In die gevallen zal SJF niet noodzakelijk de kortste gemiddelde wachttijd opleveren. De oplossing bestaat erin om een preëmptieve variant van het SJF algoritme te implementeren. De impliceert dat een lopend proces kan onderbroken worden om de processor aan een net aangekomen proces met een kortere burstlengte te geven. Dit algoritme wordt ook *shortest remaining time first* algoritme genoemd en wordt op de volgende dia geïllustreerd.

Shortest remaining time first planning (SRTF)

Proces	Aankomst	Burst
P_1	0	8
P_2	1	4
P_3	2	9
P_4	3	5

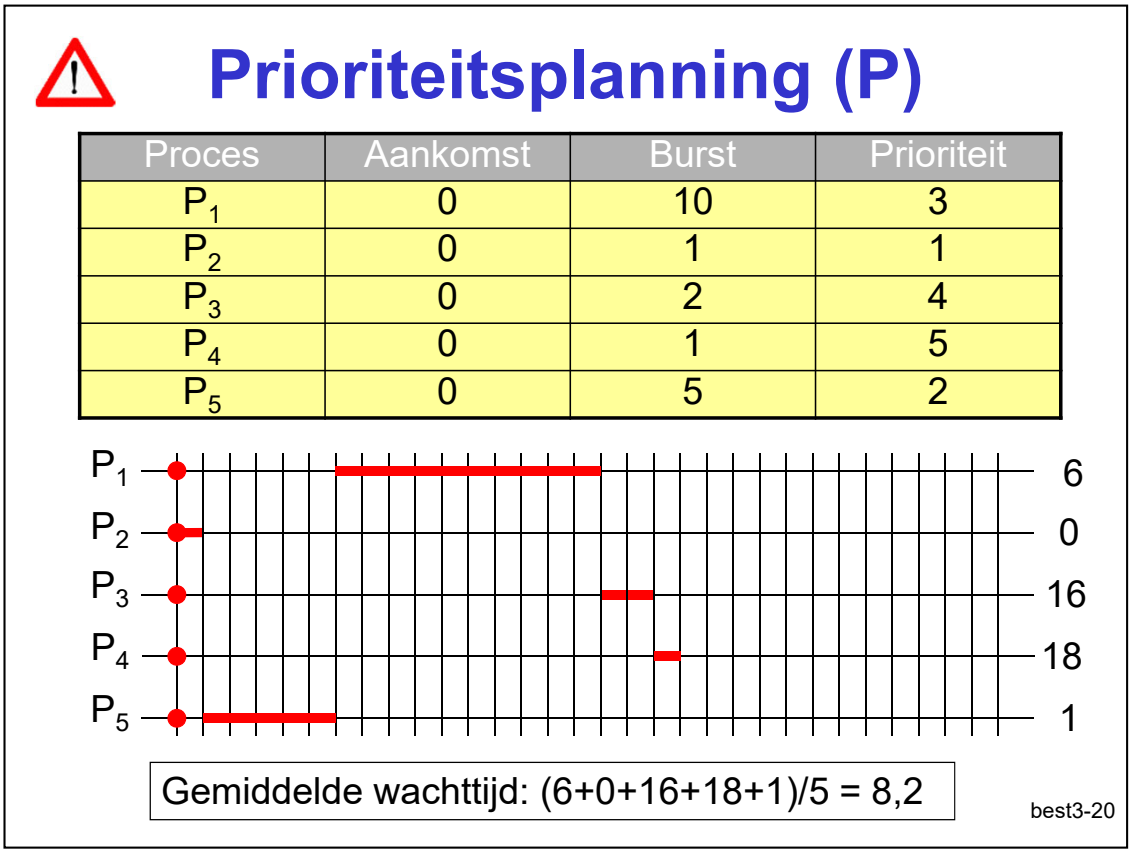


Gemiddelde wachttijd: (9 + 0 + 15 + 2)/4 = 6,5

best3-19

De gemiddeld wachttijd kan inderdaad gereduceerd worden door het proces P_1 te onderbreken ten voordele van P_2.

Nadeel van alle algoritmen die louter vertrekken van de burstlengte is dat CVE-gebonden processen (met een lange burstlengte) gediscrimineerd worden. Meer zelfs, bij overbelaste machines zullen zij gewoon niet meer aan de bak komen omdat er telkens voldoende processen met korte burst zijn die voorrang krijgen.



In dit algoritme wordt de volgorde niet bepaald door een schatting van de volgende burst, maar wel door een prioriteit. Hoe kleiner de waarde, des te hoger de prioriteit van het proces. Processen met gelijke prioriteit worden FCFS gepland. SJF is eigenlijk prioriteitsplanning met de prioriteit = burstlengte. Prioriteiten worden uitgedrukt door getallen (b.v. 0..7, 0..4095). Hoe kleiner het getal, des te hoger beschouwen we de prioriteit. Dit is evenwel niet in alle systemen de gebruikte conventie.

Men spreekt over verschillende soorten prioriteiten: externe prioriteiten en interne prioriteiten. Externe prioriteiten zijn prioriteiten die opgedrongen worden door de buitenwereld zoals de aard van de gebruiker (student, personeel), de urgentie van de taak, enz. Interne prioriteiten komen vanuit het proces zelf zoals het aantal systeemmiddelen dat door dit proces gebruikt wordt, het al dan niet CVE-gebonden of IO-gebonden zijn, deadlines, enz. Het besturingssysteem zal in veel gevallen processen een prioriteitsverhoging toepassen op die processen die actief interageren met de gebruiker (b.v. als het bijhorende venster de focus heeft).

Prioriteitsplanning kan preëmptief of niet-preëmptief zijn. Preëmptief betekent dan dat processen kunnen onderbroken worden door nieuwe processen met een hogere prioriteit. De meeste besturingssystemen hebben een prioriteitsplanner als basisplanner. Voor ware-tijdssystemen moet deze noodzakelijk preëmptief zijn om voorspelbare reactietijden te kunnen garanderen. Omwille van de goed eigenschappen zullen de meeste andere systemen ook een preëmptiever prioriteitsplanner bevatten (ofschoon complexer om te implementeren).

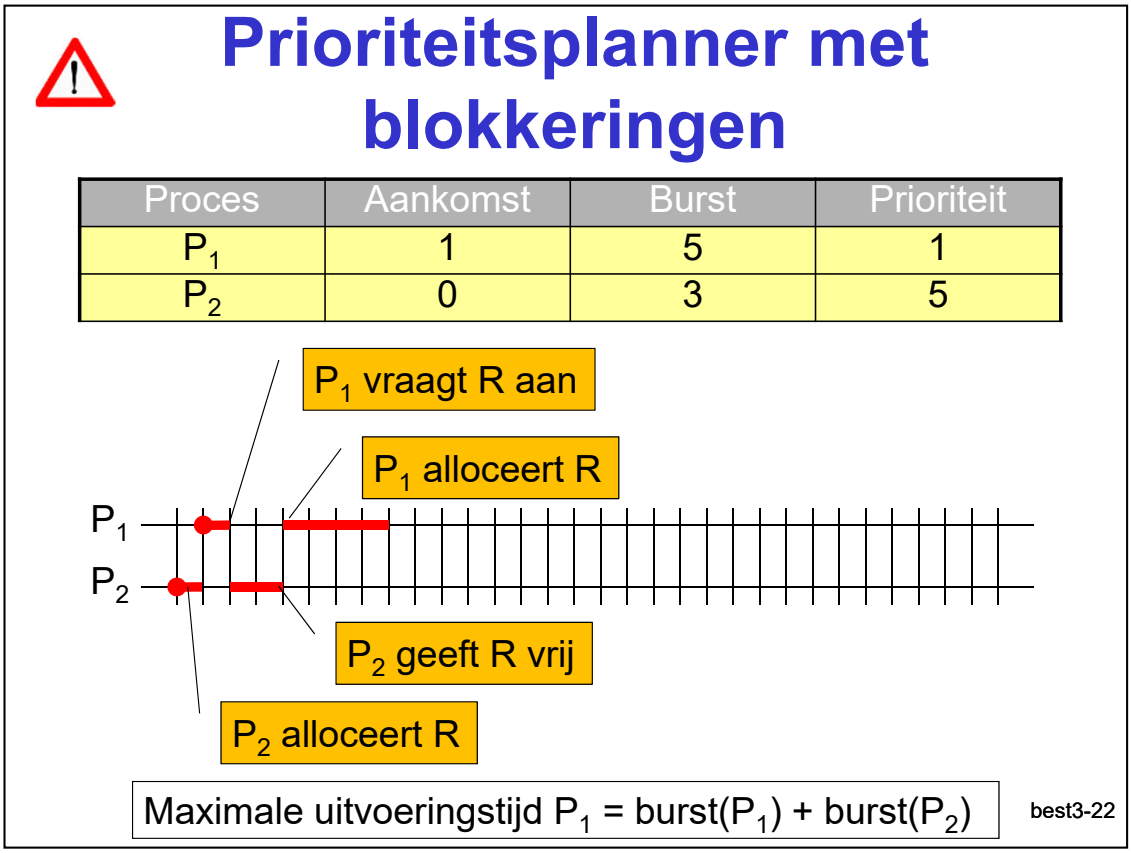
Prioriteitsplanning

- Processen met lage prioriteit kunnen zeer lang in een systeem blijven hangen; oplossing: veroudering = langzame verhoging van de prioriteit (aging).
- Bij preëmptie kan een proces met hoge prioriteit een proces met lage prioriteit onderbreken. Indien dat proces tijdelijk eigenaar is van vereiste systeemmiddelen kan de prioriteit van het proces tijdelijk verhoogd moeten worden om de systeemmiddelen vrij te geven.

best3-21

Een mogelijk gevaar bij prioriteitsplanning is dat processen met een zeer lage prioriteit zeer lang zullen moeten wachten om uitgevoerd te worden. Het proces met de laagste prioriteit zal in principe moeten wachten totdat alle andere processen uitgevoerd zijn. Bij zwaarbelaste machines kan dit ogenblik blijvend uitgesteld worden. Een elegante oplossing voor dit probleem is het gebruik van veroudering van processen. Hierbij wordt de prioriteit van het proces verhoogd in functie van zijn leeftijd (bijvoorbeeld eens om de 15 min). Na verloop van tijd zal de prioriteit zodanig opgeklommen zijn dat het proces toch zal uitgevoerd worden.

Bij prioriteitsplanning met preëmptie kan er zich een anomalie voordoen. Zo kan het gebeuren dat een proces met lage prioriteit onderbroken wordt ten voordele van een proces met hoge prioriteit, maar dat het proces met lage prioriteit op dat ogenblik essentiële systeemmiddelen in gebruik had. Dan zal de prioriteit van dat proces met lage prioriteit tijdelijk verhoogd moeten worden totdat die systeemmiddelen terug vrijgegeven zijn. Pas dan kan het nieuwe proces de fakkel overnemen. Dit mechanisme wordt prioriteitsovererving genoemd.

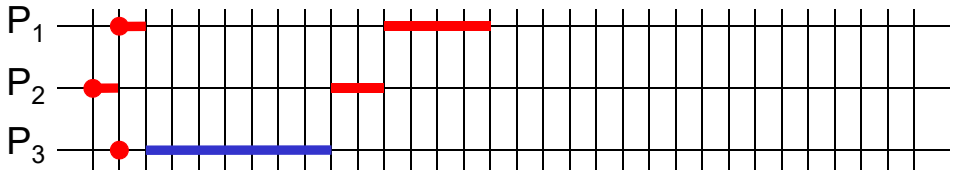


Prioriteiten bepalen echter niet alleen de volgorde van de uitvoering van de processen. De processen zullen nu en dan systeemmiddelen aanvragen die op dat ogenblik in gebruik kunnen zijn door een ander proces. In dat geval zal er eerst moeten gewacht worden tot de systeemmiddelen vrijkomen. In de bovenstaande figuur wensen P_1 en P_2 beide het systeemmiddel R te gebruiken. P_1 zal in dit geval het proces P_2 (met lagere prioriteit) moeten laten voorgaan. Dit gebeurt automatisch omdat P_1 blokkeert op het alloceren van het systeemmiddel en P_2 daardoor automatisch het proces met de hoogste prioriteit in de klaarlijst wordt. Was P_1 op ogenblik 0 aangekomen, dan had het zonder onderbreking zijn volledige burst kunnen afwerken.

In het bovenstaande geval is er echter nog steeds een bovengrens te berekenen op de maximale uitvoeringstijd. Deze kan nooit langer zijn dan de burstlengte van het proces P_1 vermeerderd met de burstlengte van alle processen die mogelijks het systeemmiddel in bezit kunnen hebben.

Prioriteitsinversie

Proces	Aankomst	Burst	Prioriteit
P_1	1	5	1
P_2	0	3	5
P_3	1	7	3



Maximale uitvoeringstijd P_1 = burst(P_1) + burst(P_2) + ??

Oplossing: geef P_2 op t=2 tijdelijk de prioriteit van P_1

best3-23

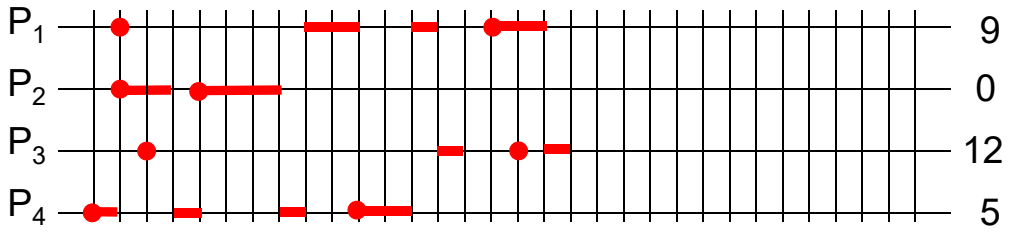
Als er zich nog meer processen aandienen in het systeem, dan kan de situatie echter snel verslechteren. In bovenstaand voorbeeld dient proces P_3 zich aan samen met proces P_1. Als P_1 blokkeert, dan blijkt P_2 niet langer het enige proces in de klaarlijst te zijn, en bovendien niet langer het proces met de hoogste prioriteit. In dit geval zal P_1 dus aanzienlijk langer moeten wachten op het verwerven van het systeemmiddel. Wat echter bijzonder erg is, is dat het moet wachten op een proces met een lagere prioriteit (3) dat verder niets te maken heeft met het systeemmiddel dat aangevraagd wordt. Deze situatie wordt prioriteitsinversie genoemd. De prioriteitsplanner beslist hier om een proces met prioriteit 3 voorrang te geven op de uitvoering van een proces met prioriteit 1.

De meest elegante manier om dit tegen te gaan is om proces P_2 tijdelijk dezelfde prioriteit te geven als het proces dat het systeemmiddel aanvraagt. Op deze manier wordt vermeden dat processen met lagere prioriteit profiteren van de situatie.

Prioriteitsinversie was de oorzaak van het plotse falen van de Mars Pathfinder in 1997. Gelukkig is prioriteitsinversie een softwareprobleem dat vanop afstand kon opgelost worden.

Prioriteitsplanning

Proces	Aankomst	Burst	IO	Burst	Prio
P_1	1 - ε	3	2-1ε	2	3
P_2	1	2	1-2ε	3	1
P_3	2	1	2-3ε	1	3
P_4	0	3	2-4ε	2	2



Gemiddelde wachttijd: (9+0+12+5)/4 = 26/4=7,5

best3-24

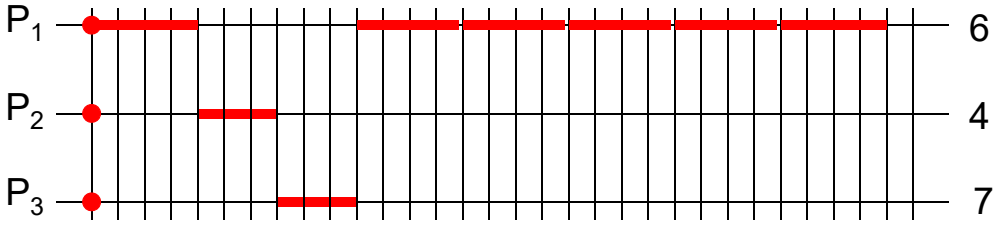
Dit is een voorbeeld van een planning van 2 cpu-bursts per proces. De gebruikte planner is een preëmptieve prioriteitsplanner, FCFS bij gelijke prioriteit.

Tijdens de IO-burst bevindt het proces zich in de geblokkeerde lijst. Op het einde van de IO-burst wordt het toegevoegd aan de klaarlijst, in volgorde van aankomst.

Round Robin planning (RR)

Proces	Aankomst	Burst
P_1	0 - 2ε	24
P_2	0 - ε	3
P_3	0	3

Kwantum = 4 eenheden



Gemiddelde wachttijd: (6+4+7)/3 = 5,66

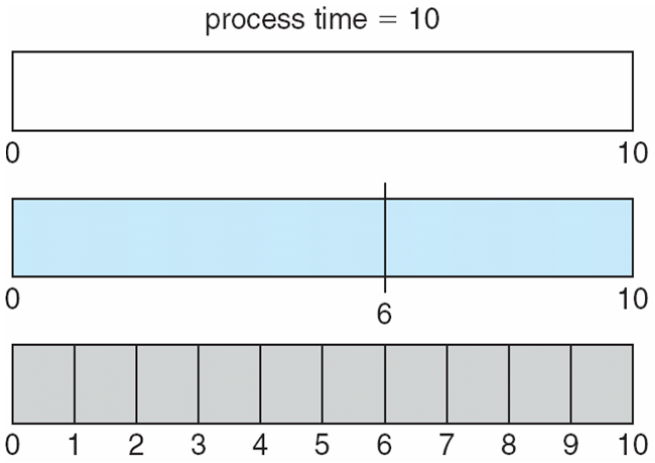
best3-25

Dit algoritme werd speciaal ontwikkeld voor time sharing systemen. Het is een uitbreiding van het FCFS algoritme. Een CVE-burst die langer is dan een opgegeven tijdskwantum zal na het verstrijken van dit kwantum onderbroken worden en terug achteraan de wachtrij van processen toegevoegd worden. Indien het proces moet wachten op IO, zal het vrijwillig afstand doen van de processor. Bij synchronisatiemechanismen die gebaseerd zijn op actieve synchronisatie zal dit natuurlijk niet het geval zijn, en zal het volledige tijdskwantum opgebruikt worden.

Tijdskwanta (time slices) variëren van 10 tot 100 ms op hedendaagse systemen. De processen uit de klaarlijst krijgen elk om beurt een tijdskwantum. De gemiddelde wachttijd van processen die met RR gepland worden kan vrij lang zijn. Uit de dia volgt dat voor de processen met een tijdskwantum van 4 eenheden de gemiddelde wachttijd 5.66 zal zijn (vergelijk dit met de 17 en 3 van FCFS).

De doorlooptijd van een proces zal afhangen van de belasting van het totale systeem. Indien er n processen klaar zijn voor uitvoering zal elk proces gemiddeld 1/n van de totale CVE-tijd ter beschikking krijgen.

Tijdskwantum vs. Contextwisseling



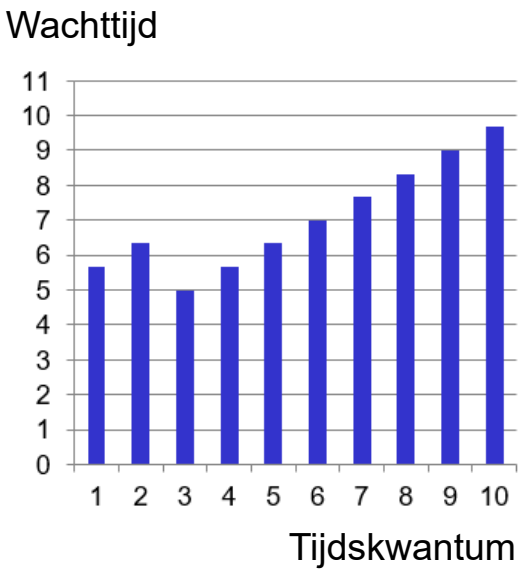
Vuistregel: 80% van de bursts binnen 1 kwantum afgewerkt

best3-26

De efficiëntie van RR is afhankelijk van de grootte van het tijdskwantum. Met een oneindig groot tijdskwantum bekomt men FCFS, met een oneindig klein tijdskwantum verkrijgt men processor sharing (dit betekent dat de processen dan 1/n van de rekenkracht krijgen). Het probleem is dat met kleiner wordende tijdskwanta het aantal contextwisselingen ook toeneemt (een contextwisseling kan tot 10 µs tijd vergen). Deze contextwisselingen veroorzaken een constante belasting van het systeem (hun aantal per tijdseenheid is min of meer vast voor een gegeven tijdskwantum). In de praktijk probeert men ervoor te zorgen dat ongeveer 80% van de CVE-bursts in één tijdskwantum kunnen afgewerkt worden. Dit wil zeggen dat in de praktijk slechts 20% van alle bursts werkelijk onderbroken zullen worden. Een andere vuistregel die men hanteert is dat de grootte van een tijdskwantum minstens tienmaal de tijdsduur van een contextwisseling moet zijn. Verder moet het tijdskwantum klein genoeg zijn om alle actieve processen ongeveer viermaal per seconde aan de beurt te laten komen.

Voor RR is het zo dat indien er n processen zijn in de klaarlijst, en het tijdskwantum q tijdseenheden is, elk proces 1/n van de CVE-tijd krijgt in stukken van maximaal q tijdseenheden. Een proces zal hierbij nooit langer dan (n-1)q tijdseenheden moeten wachten.

Wachttijd ifv van het tijdskwantum



Proces	Burst
P_1	24
P_2	3
P_3	3

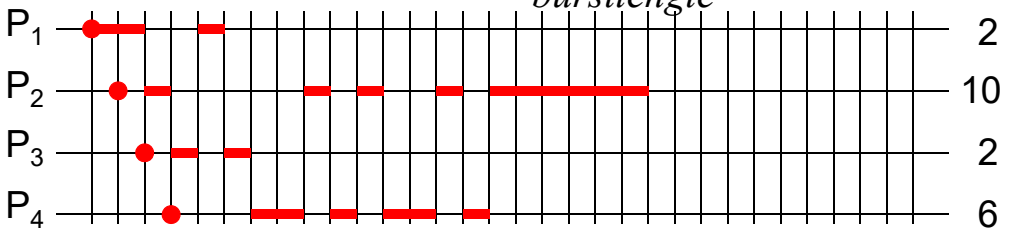
best3-27

Als je het voorbeeld van RR uitwerkt met tijdskwanta variërend van 1 tot 10 eenheden, dan merk je dat de kleinste gemiddelde wachttijd optreedt bij een tijdskwantum van 3 eenheden (wachttijd van 5 eenheden), en dat deze korter is dan in het geval van een tijdskwantum van 4 eenheden (wachttijd van 5,66 eenheden, zie voorbeeld).

Highest response ratio next planning (HRRN)

Proces	Aankomst	Burst
P_1	0	3
P_2	1	10
P_3	2	2
P_4	3	6

$$response\_ratio = \frac{wachttijd + burstlengte}{burstlengte}$$



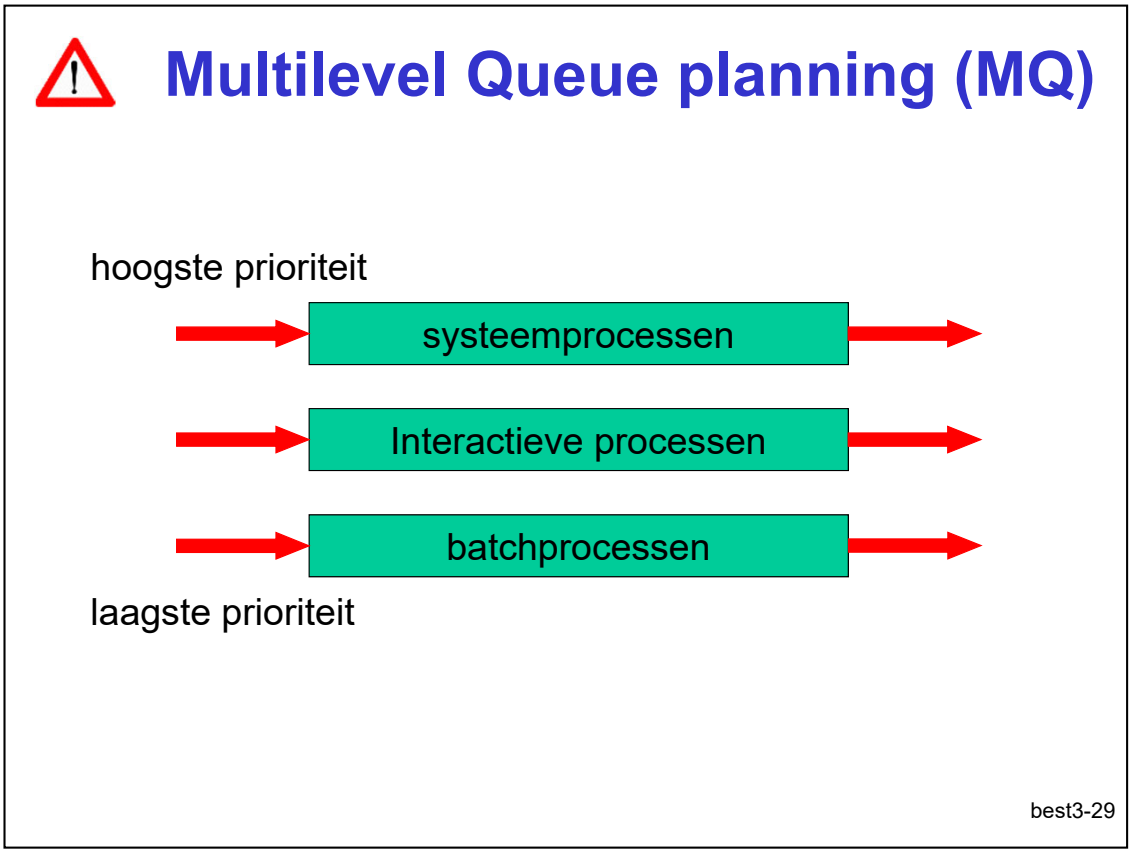
Gemiddelde wachttijd: (2+10+2+6)/4 = 5

best3-28

Bij een gemengde werklast kan men de relatieve vertraging in rekening brengen. Deze wordt uitgedrukt als de response_ratio die een maat is voor de wachttijd van het proces i.f.v. zijn burstlengte. Een response_ratio van 2 wijst op het feit dat het proces reeds even lang gewacht heeft als de lengte van zijn burst. Een proces met een lange burst zal m.a.w. heel wat meer wachttijd kunnen accumuleren alvorens een response ratio van 2 te bereiken dan een proces met een kleine burst.

Nieuw aankomende processen hebben een response_ratio van 1 (wachttijd = 0), en zullen niet geselecteerd worden indien er nog processen met een hogere response_ratio in het systeem aanwezig zijn. Hun response_ratio neemt per tijdskwantum echter snel toe (hoe kleiner de burst hoe steiler de klim). Van zodra het proces de response_ratio van de andere processen bereikt zal het ook zijn deel van de rekenkracht kunnen opeisen. Dit algoritme bevoordeligt de processen met een kleine burstlengte zonder daarbij de processen met een lange burstlengte compleet lam te leggen. Nadeel van dit algoritme is wel dat indien de response_ratio van de processen in het systeem hoog opgelopen is, nieuwe processen geruime tijd zullen moeten wachten om aan de bak te komen (totdat hun response_ratio even hoog opgelopen is)..

HRRN kan zowel preëmptief als niet-preëmptief gebruikt worden.

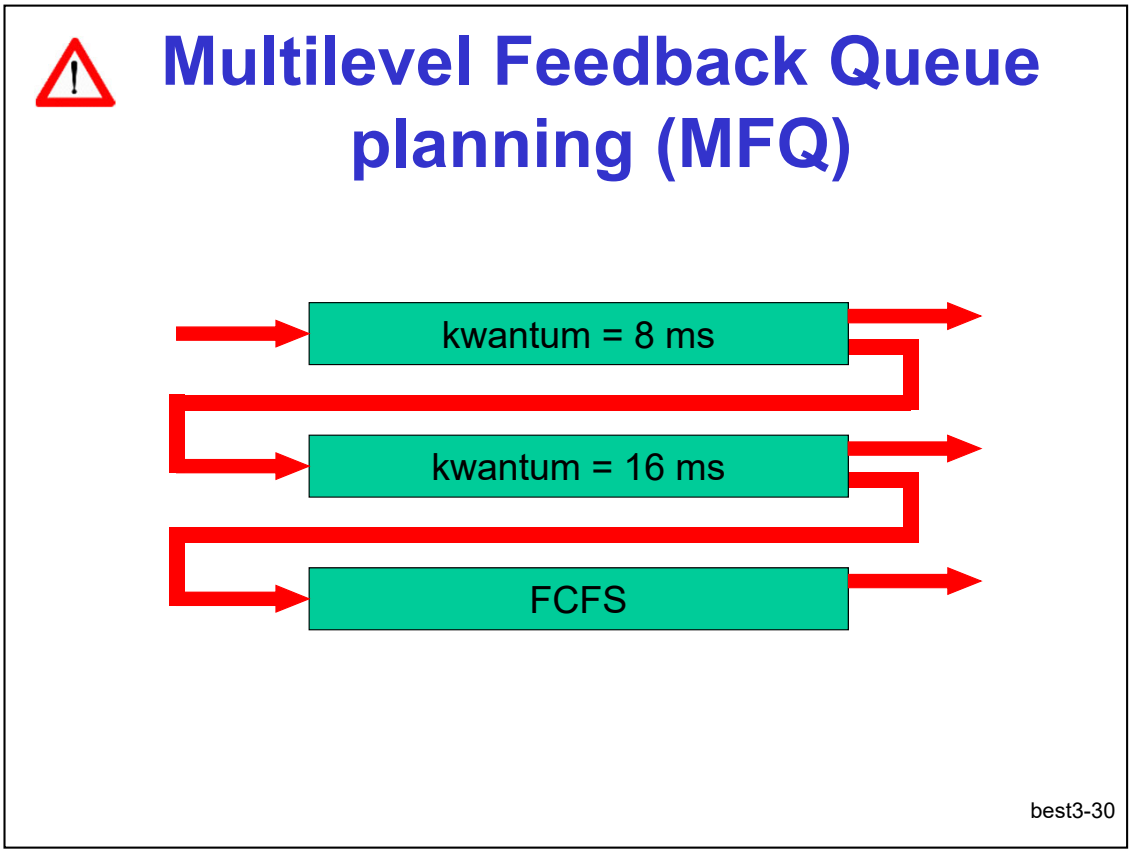


Om de planning van processen beter te kunnen afstemmen op de noden van die processen kan men de processen in de klaarlijst opdelen in categorieën waarbij de processen van een bepaalde categorie ondergebracht wordt in een afzonderlijke wachtrij. Elk van deze wachtrijen kan beschikken over een eigen planningsalgoritme.

Een mogelijk onderscheid dat tussen de processen kan gemaakt worden is: voorgrondprocessen (interactieve) en achtergrondprocessen (batch). De voorgrondprocessen kunnen RR gebruiken terwijl de achtergrondprocessen bij voorkeur FCFS zullen aanwenden, of een algoritme gebaseerd op prioriteiten. Men kan het aantal categorieën verder uitbreiden en verfijnen (systeemprocessen, interactieve processen, batch processen, studentenprocessen,…).

Voor de planning van de wachtrijen onderling heeft men ook verschillende mogelijkheden. Een eerste mogelijkheid betreft het gebruik van een prioriteitsalgoritme met preëmptie. Dit wil zeggen dat processen uit een wachtrij met een lagere prioriteit slechts aan bod kunnen komen nadat er geen processen uit een wachtrij met een hogere prioriteit meer ter beschikking zijn. Van zodra er zich een dergelijk proces aanbiedt, moeten alle processen van een wachtrij met lagere prioriteit wijken.

Een andere mogelijkheid is om de beschikbare CVE-tijd te verdelen onder de wachtrijen waarbij de voorgrondprocessen maximaal 80% van de tijd kunnen opnemen en de rest voor de achtergrondprocessen gereserveerd wordt. Indien een bepaalde categorie van processen niet voorhanden is worden hun tijdskwanta natuurlijk verdeeld over de andere wachtrijen.



De voorgaande planningalgoritmen werken geen van allen goed indien het aantal actieve processen groot wordt (100 of meer). Het is niet moeilijk om in te zien dat het tijdskwantum dan kleiner moet zijn dan 2.5 ms (responstijd voor een proces is dan 0,25s) hetgeen op de grens van het aanvaardbare ligt. Daarom werken een aantal systemen met tijdskwanta van variabele lengte.

Bij multilevel feedback queue planning worden de processen niet statisch ingedeeld bij een bepaalde wachtrij, maar kunnen zij migreren tussen wachtrijen. Het is de bedoeling om processen naar hun aard dynamisch in een bepaalde wachtrij onder te brengen. Onder bepaalde voorwaarden kan een proces dan van wachtrij veranderen.

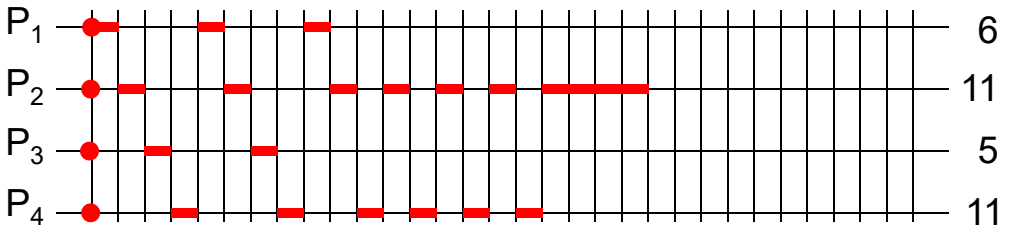
Een voorbeeld van een multilevel feedback queue systeem is een systeem bestaande uit 3 wachtrijen. Processen komen steeds binnen in wachtrij 1. Daar krijgen ze een tijdskwantum van 8 ms. Indien dit niet volstaat worden ze onderbroken en verwezen naar wachtrij 2 waar ze een tijdsquantum van 16 ms zullen ter beschikking krijgen. Indien ook dat niet volstaat dan gaan ze over naar de derde wachtrij waar ze FCFS afgewerkt worden. Een proces uit wachtrij 2 kan slechts uitgevoerd worden indien er zich geen processen in wachtrij 1 bevinden. Een proces in wachtrij 2 kan ook onderbroken worden door een nieuw proces in wachtrij 1. Processen die moeten wachten op IO komen na het aflopen van de IO-operatie opnieuw automatisch in wachtrij 1 terecht.

In het algemeen geval wordt een multilevel feedback queue gekarakteriseerd door de volgende parameters: (i) aantal wachtrijen, (ii) planningsalgoritme voor elk van de wachtrijen, en (iii) criteria om van wachtrij te veranderen (in beide richtingen).

Dit soort van algoritme geeft de voorkeur aan interactieve processen (waarvan de bursts niet groter zijn dan 8 ms). Hierdoor zullen de antwoordtijden van deze processen klein zijn. Interactieve processen die een tijdlang geen uitvoer genereren zullen op zwaar belaste systemen snel naar de laagste regionen van de multilevel feedback queue verhuizen.

Gewaarborgde planning (G)

Proces	Aankomst	Burst
P_1	0	3
P_2	0	10
P_3	0	2
P_4	0	6



Gemiddelde wachttijd: (6+11+5+11)/4 = 8,25

best3-31

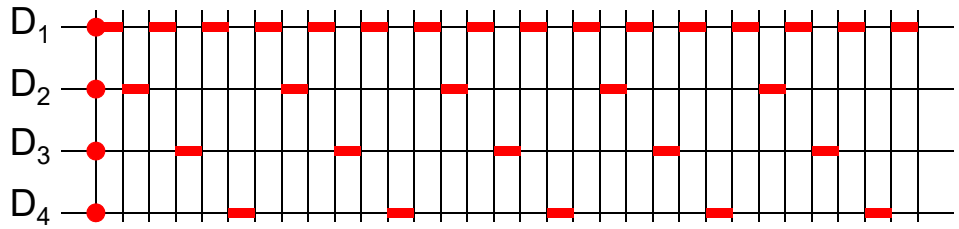
Om ervoor te zorgen dat alle processen vooruitgang maken, ook al hebben ze een zeer lange burstlengte, zou men kunnen overwegen om aan elk proces in het systeem een evenredig deel van de rekenkracht te geven (guaranteed scheduling). Concreet zou men b.v. aan 4 processen elk een kwart van de rekentijd kunnen toekennen. Het planningscriterium is dan de gebruikte CVE-tijd/de rechtmatige CVE-tijd.

Hoe kleiner de verhouding, des te hoger de prioriteit van het bijhorende proces. Een verhouding van 0,5 betekent dat het proces de helft van de rekentijd gekregen heeft waarop het principieel recht had. Bij de aankomst heeft een proces 0 eenheden uitgevoerd, en had het ook recht op 0 eenheden (Indien we 0/0 gelijk stellen aan 1, geven we geen absolute voorkeur aan beginnende processen. Indien we 0/0 gelijkstellen aan 0, geven we wel meteen de processor aan nieuwe processen). Nadat proces 1 één eenheid tijd gebruikt heeft, is de verhouding: 1/(0,25) = 4 omdat het proces in dit geval recht heeft op 25% van de rekentijd. Na 1 eenheid gewacht te hebben, wordt dit 1/(0.5) = 2, en dan 1/(0.75) = 1,33 enz.

Het proces met de kleinste verhouding blijft aan de beurt totdat zijn verhouding opgeklommen is tot de op één na kleinste verhouding. Als men gewaarborgde planning toepast op jobniveau, dan zullen IO-gebonden processen door dit planningsalgoritme bevoordeligd worden. Doordat ze regelmatig staan wachten op hun randapparaten zullen ze slechts in beperkte mate gebruik maken van de hen ter beschikking gestelde rekentijd, waardoor ze telkens wanneer ze actief worden over veel resterende rekenkracht zullen kunnen beschikken.

Fair-share planning (FS)

Draad	Aankomst	Proces
D_1	0	P_1
D_2	0	P_2
D_3	0	P_2
D_4	0	P_2



best3-32

Planners die systeemmiddelen verdelen op basis van draden, kunnen misleid worden door processen met zeer veel draden. Fair-share planning zorgt ervoor dat de rekenkracht van een processor niet gemonopoliseerd kan worden door processen met veel draden. Het zorgt ervoor dat de gebruikers of processen hun deel van de rekenkracht krijgen, die ze dan intern moeten verdelen over de verschillende processen of draden.

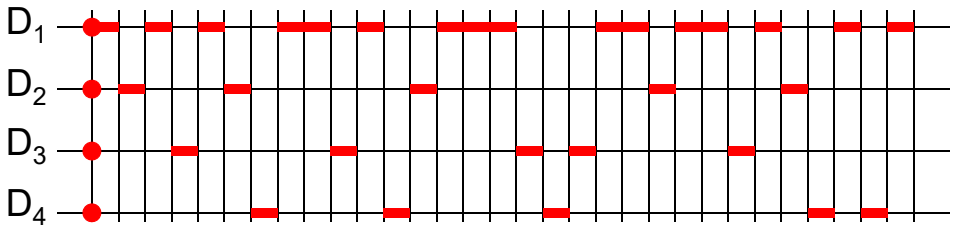
Hier is er een geval van een proces met 1 draad, en een proces met drie draden. Zoals uit het planningsschema blijkt zal proces 2 in totaal maar even veel rekentijd krijgen als proces 1. Het creëren van meerdere draden helpt in dit geval dus niet om een groter deel van de rekenkracht van de processor te krijgen.

Dit mechanisme kan gebruikt worden bij de implementatie van dienstverleningskwaliteit (QOS: Quality of Service). Op deze manier kan men er b.v. voor zorgen dat bepaalde processen steeds kunnen beschikken over een zeker deel van de rekenkracht van de processor. Op dezelfde manier kan men er ook voor zorgen dat ze niet meer dan dat deel gebruiken. Beide zijn belangrijk. Bij het verwerking van een videostroom in ware tijd kan het essentieel zijn dat b.v. 25% van de rekenkracht gealloceerd wordt voor deze taak, en dat andere taken hieraan niets kunnen veranderen (minimale garantie). Anderzijds kan het ook zo zijn dat indien op één bepaalde server b.v. de websites van 3 bedrijven 'gehost' worden, men garanties wil inbouwen dat een bepaalde website niet meer dan 40% van de totale systeemmiddelen voor zijn rekening kan nemen (tenzij men meer betaalt). Indien er meer trafiek ontstaat voor de betrokken website, zullen bij fair share planning enkel de aanvragen voor die website vertraagd worden. De andere websites zullen daar geen

hinder van ondervinden.

Lotterijplanning (L)

Draad	Aankomst	Lotjes
D_1	0	50
D_2	0	10
D_3	0	10
D_4	0	10



best3-33

Lotterijplanning is een manier om fair-share of gewaarborgde planning te implementeren. Bij de creatie van een proces worden er een aantal lotjes verdeeld aan de processen. De planner trekt willekeurig een lotje uit het aantal in omloop zijnde lotjes. Het proces of de draad dat met het getrokken lotje geassocieerd is, krijgt het volgende tijdskwantum.

Lotjes kunnen uitgewisseld worden tussen processen. Zo kan een proces dat staat te wachten beslissen om tijdelijk zijn lotjes over te dragen aan het proces waarop het staat te wachten om op die manier dat proces sneller vooruit te laten gaan. Dit is vergelijkbaar met het principe van de prioriteitsinversie.

Ware-tijdssystemen

- **Harde ware tijd**: te laat = geen antwoord = fout
  - B.v. controletoepassingen zoals een lasrobot
- **Zachte ware tijd**: te laat = vervelend maar niet zo erg
  - B.v. Video, toekennen van voldoend hoge prioriteit volstaat vaak

- Periodische processen & aperiodische processen
- Statische en dynamische planning: vaste of veranderlijke prioriteiten

best3-34

De voornaamste verschillen tussen planning in ware tijd en wat we tot nog toe besproken hebben is dat we bij de planning in ware tijd steeds moeten kunnen garanderen dat een antwoord binnen een vooraf bepaalde tijd geleverd zal worden. Hiervoor is het onder andere nodig dat

- de maximale uitvoeringstijden van alle systeemoproepen gekend zijn, en deze bij voorkeur onderbreekbaar zijn;
- men aan processen die moeten uitgevoerd worden in ware tijd de hoogste prioriteit kan toekennen zolang men dit nodig acht.

Alle elementen van een besturingssysteem die dit niet kunnen garanderen kunnen niet aangewend worden bij de opbouw van een harde ware-tijdsysteem.

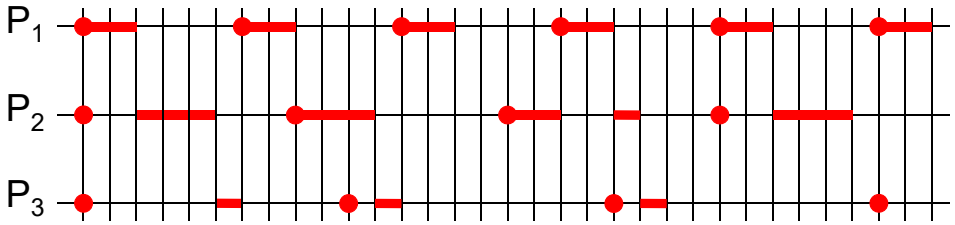
De grote moeilijkheid hierbij is het op voorhand kunnen bepalen van de (maximale) uitvoeringstijd (WCET of worst case execution time). Als men deze niet kan achterhalen (b.v. omwille van het voorkomen van componenten waarvan men de maximale uitvoeringstijd niet kan bepalen of waarvan de maximale uitvoeringstijd zodanig veel groter is dan wat men normaal gezien kan verwachten, dat deze onbruikbaar is in de praktijk), dan kan men geen ware-tijdsysteem bouwen. Vandaar dat men in harde ware-tijdsystemen vaak tevergeefs zal zoeken naar caches of virtueel geheugen. Deze zijn gewoon te onvoorspelbaar.

Rate Monotonic planning

Proces	Periode = P	Burst = C
P_1	6	2 (33%)
P_2	8	3 (37,5%)
P_3	10	1 (10%)

$$\sum_{i=1}^{m} \frac{C_i}{P_i} \leq 1$$

∑ = 80.5%



best3-35

Dit is een planningmethode die vaak gebruikt wordt bij het plannen van periodieke taken, dit zijn taken die met een bepaalde frequentie moeten herhaald worden (b.v. 10x per seconde of 100x per seconde). Dit soort van taken komt vaak voor bij signaalverwerkingstoepassingen waarbij men aan een gegeven tempo inkomende signalen (metingen, audio, video) moet verwerken.

Eenvoudig uitgelegd komt het erop neer dat men de periode van de taak als prioriteit hanteert. Hoe kleiner de periode (of des te hoger de frequentie), des te hoger de prioriteit is. Dit wil concreet zeggen dat men steeds de voorkeur geeft aan die taken die het vaakst per tijdseenheid moeten uitgevoerd worden.

De voorwaarden om RMS (Rate Monotonic Scheduling) te kunnen toepassen zijn:

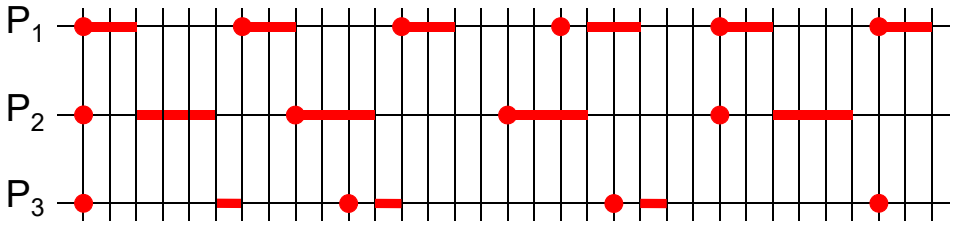
- $C_i < P_i$
- Alle processen zijn onafhankelijk
- Per proces, maar één C_i
- De aperiodische processen hebben geen deadlines
- Geen overhead voor preëmptie of contextwisseling

Prioriteit = rate = $1/P_i$. Processen met de hoogste frequentie hebben de hoogste prioriteit

Earliest Deadline First Planning

Proces	Periode	Burst
P_1	6	2 (33%)
P_2	8	3 (37,5%)
P_3	10	1 (10%)

$\sum = 80.5\%$



best3-36

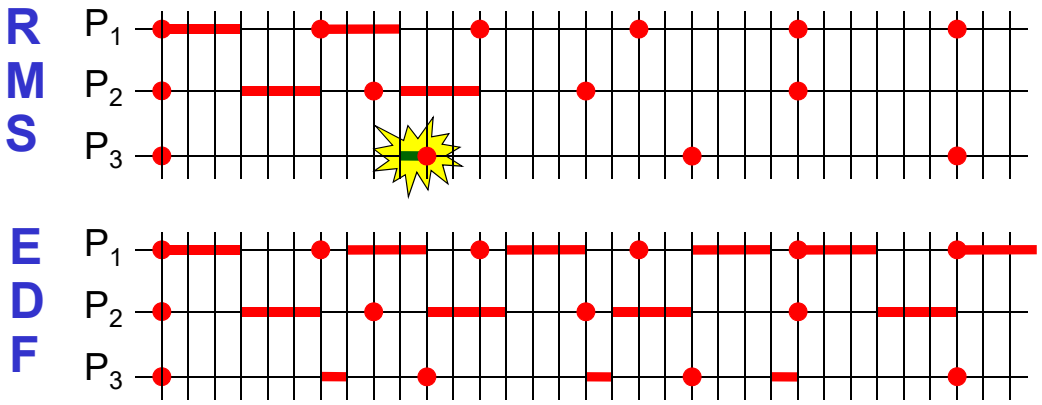
Bij earliest deadline first planning geeft men prioriteit aan het proces met de dichtste deadline. In dit voorbeeld is dit niet-preëmptief. Men zou echter ook een preëmptieve variant van het algoritme kunnen bedenken. Dit algoritme kan dan zeer snel inspelen op zeer dringende oproepen.

Vergelijking RMS - EDF

Proces	Periode = P	Burst = C
P_1	6	3 (50%)
P_2	8	3 (37,5%)
P_3	10	1 (10%)

$$\sum_{i=1}^{m} \frac{C_i}{P_i} \leq m(2^{1/m} - 1)$$

$\sum$ = 97.5%
> 69.3%



best3-37

Uit deze vergelijking volgt dat RMS kan falen bij zware belasting van het systeem (hier 97,5%), terwijl EDF er ook in dit geval nog in slaagt om een goede planning te vinden. Van RMS is geweten dat een planning pas gegarandeerd kan worden op voorwaarde dat de belasting van het systeem niet hoger ligt dan de formule aangegeven in het kadertje. Voor m (aantal processen) gaande naar oneindig resulteert dit in een belasting van 69,3%.

EDF is dus superieur aan RMS, maar heeft ook nadelen. EDF moet al zijn beslissingen dynamisch maken, en dit brengt overhead met zich mee. Bij RMS kan alles statisch gepland worden. Eenmaal de prioriteiten vastliggen volstaat een gewonen prioriteitsplanner om een bewijsbaar correcte planning te realiseren (op voorwaarde dat de belasting binnen de perken blijft). De overhead van een prioriteitsplanning is heel laag.

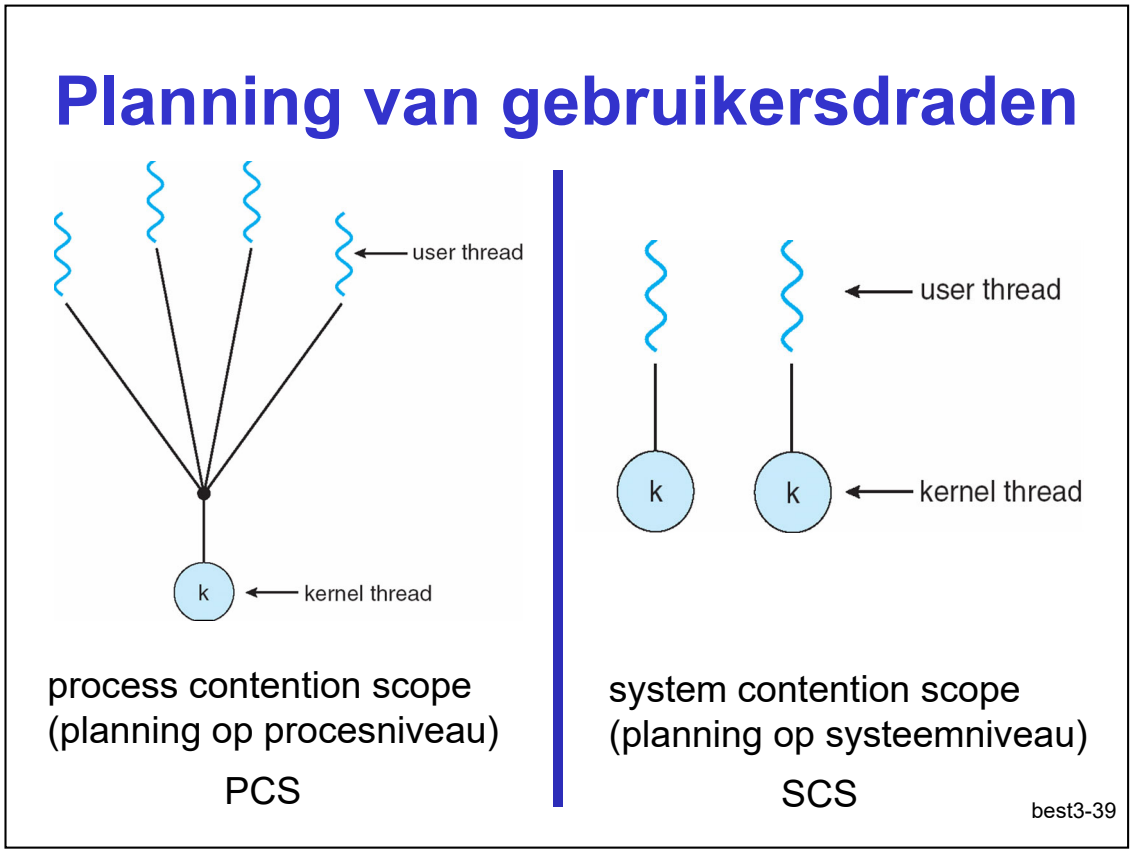
Toepassingsdomeinen

Planneralgoritme	Batch	Interactief	Real-time
First-come First-served (FCFS)	X		
Shortest Job First (SJF)	X	X	
Shortest Remaining Time First (SRTF)	X	X	
Prioriteit		X	X
Round Robin		X	
Gewaarborgde planning		X	
Highest Response Ratio Next		X	
Fair Share		X	
Loterijplanning		X	
Multilevel Queue	X	X	X
Multilevel Feedback Queue		X	
Earliest Deadline First			X
Rate Monotonic Scheduling			X

best3-38

In de bovenstaande tabel wordt aangegeven voor welke domeinen de gegeven planneralgoritmen van toepassing zijn. Zoals duidelijk blijkt uit te tabel zijn niet alle planneralgoritmen even goed toepasbaar/nuttig in de verschillende domeinen. Batch is een niet-interactieve werklast. Algoritmen die interactief werken mogelijk maken hebben daarbij niet veel nut.

Planneralgoritmen zijn een mooi voorbeeld van scheiding tussen mechanisme en politiek. De algoritmen liggen vast, maar de parameters zijn instelbaar (b.v. de prioriteiten). Een MLFQ planningsalgoritme is b.v. bijzonder interessant omdat het parameteriseerbaar is, en omdat dit ons toelaat om het precies af te stemmen op een bepaalde belasting. Het mechanisme is in dit geval de multilevel feedback queue, terwijl de politiek bepaald wordt door de parameters van het mechanisme (tijdskwanta, voorwaarden voor overgang, enz.). Een dergelijke scheiding is steeds te verkiezen boven systemen die de politiek vastkoppelen aan een mechanisme.



De gebruikersdraden voeren competitie voor een systeemdraad. Dit gebeurt op het procesniveau (PCS: process contention scope of lokale planning). De gebruikersplanner zal een gebruikersdraad voor een zekere tijd binden aan een systeemdraad. De procesplanner selecteert de systeemdraad volgens de regels gedicteerd door de procesplanner. De tijdschaal van de gebruikersdraden en de systeemdraden is daarbij onafhankelijk. Het is b.v. mogelijk dat een gebruikersdraad gedurende 20 ms gebonden wordt aan de systeemdraad, en dat de systeemdraad werkt met een tijdskwantum van 8 ms. De gebruikersdraad zal dan 2,25 tijdskwanta van de kerneldraad actief zijn, en dan vervangen worden door een andere gebruikersdraad. Omgekeerd kan het ook zijn dat een gebruikersproces maar gedurende 2 ms gebonden wordt aan de systeemdraad. Dan kunnen er tijdens 1 activatie van de systeemdraad 4 gebruikersdraden uitgevoerd worden. De systeemdraad is niet op de hoogte van het feit dat in de gebruikersruimte nog een tweede planner actief is.

De gebruikersdraden zullen doorgaans niet preëmptief zijn, maar spontaan hun binding met de systeemdraad afstaan. De contextwisseling van twee gebruikersdraden vergt niet veel overhead (aangezien er geen tussenkomst van de kernel vereist is). Het voordeel van een gebruikersplanner per proces te hebben is dat men ook gebruik kan maken van processpecifieke planningsalgoritmen. De procesplanner kan uiteraard geen rekening houden met processpecifieke toestanden.

Bij de planning van kerneldraden spreekt men van SCS of system contention scope (globale planning). Dit betekent dat er een systeemwijde competitie bestaat tussen alle

systeemdraden. Op het systeemniveau wordt meestal een preëmptief planningsalgoritme toegepast. In de Pthreads bibliotheek kan men kiezen tussen PCS en SCS bij de creatie van een draad. PCS zal echter niet ondersteund worden door Linux en MacOS.

Multiprocessorplanning

- Een globale planner (goed voor belastingsspreiding)
- Verschillende lokale planners (goed voor asymmetrische architecturen)
- Bendeplanning (gang scheduling)

best3-40

Voor multiprocessors heeft men de keuze tussen één planner per processor, één gemeenschappelijke planner voor alle processors, of een combinatie van beide. Het voordeel van een lokale planner is dat de individuele processors hun eigen planning zo goed mogelijk kunnen regelen. Bovendien is er het voordeel dat indien niet alle processors identiek zijn (bepaalde processen die enkel op een bepaalde processor kunnen uitgevoerd worden), men die processen rechtstreeks aan één processor kan binden (zgn. processor affinity). Anderzijds is het zo dat de belasting over de processors heen niet gemakkelijk kan verdeeld worden. Terwijl bepaalde processors overbelast zijn, kunnen andere onbelast zijn.

In de globale visie zijn er twee aanpakken mogelijk. Ofwel gaan de processors zelf op zoek naar werk in de wachtrijen (pull migration), waarbij dit natuurlijk goed gesynchroniseerd moet gebeuren, ofwel is er een meesterprocessor die het werk over de andere processors verdeelt (push migration). Deze processor kan dan waken over de geschikte belastingsverdeling over de processors, met inbegrip van het feit dat bepaalde processen gebonden zijn aan bepaalde processors.

In de visie waarbij er zowel een lokale als een globale planner aanwezig zijn, zal de lokale planner indien hij dit nodig acht de globale wachtrij consulteren om te zien of er nog voor hem geschikte processen ter beschikking zijn. De lokale planners kunnen eventueel ook door een meesterprocessor gedwongen worden om bepaalde processen in hun wachtrij op te nemen.

Bij het plannen van draden op een single core processor moet men ook rekening houden met het feit dat de draden van een particulier proces best na elkaar uitgevoerd worden, zoniet verliest men het voordeel van het hergebruik van de werkverzameling en de

beperkte contextwisseling. Op multiprocessors kan men ook aan bendeplanning (gang scheduling) gaan doen. Dit houdt in dat bij een contextwisseling alle processors van context wisselen en dat alle draden van een proces simultaan uitgevoerd worden. Alle draden van 1 proces worden dan in groep gepland. Dit kan voordelen hebben omdat de draden sneller vooruitkomen (zo kan men bv overwegen om actieve synchronisatie te gebruiken voor kritieke secties die maar heel kortstondig bezet zijn). Op die manier kunnen heel kortdurende toestandsveranderingen naar de geblokkeerde toestand vermeden worden.

Multicore planners

- In essentie kunnen deze planners verschillende draden simultaan in uitvoering plannen.
- De cores kunnen zelf SMT (Simultaneous MultiThreading) aanbieden waardoor twee draden door dezelfde core kunnen uitgevoerd worden en op nanoschaal door de processor gepland worden.

best3-41

Virtualisatie

- Een hypervisor gebruikt zelf ook een planner om de diverse BS van systeemmiddelen te voorzien.
- Die systeemmiddelen worden dan door de planners van de gastbesturingssystemen verder verdeeld over de diverse processen en draden.
- In gastbesturingssysteem worden de systeemdraden hierdoor eigenlijk gebruikersdraden…

best3-42

Overzicht

- Wat is procesplanning?
- Planningsalgoritmen
  - Monoprocessorsystemen
  - Multiprocessorsystemen
- Concrete gevallen

best3-43

Solaris

Prioriteit	Tijdskwantum	Prioriteit na aflopen tijdskwantum	Prioriteit na blokkering
0	200	0	50
5	200	0	50
10	160	0	51
15	160	5	51
20	120	10	52
25	120	15	52
30	80	20	53
35	80	25	54
40	40	30	55
45	40	35	56
50	40	40	58
55	40	45	58
59	20	49	59

best3-44

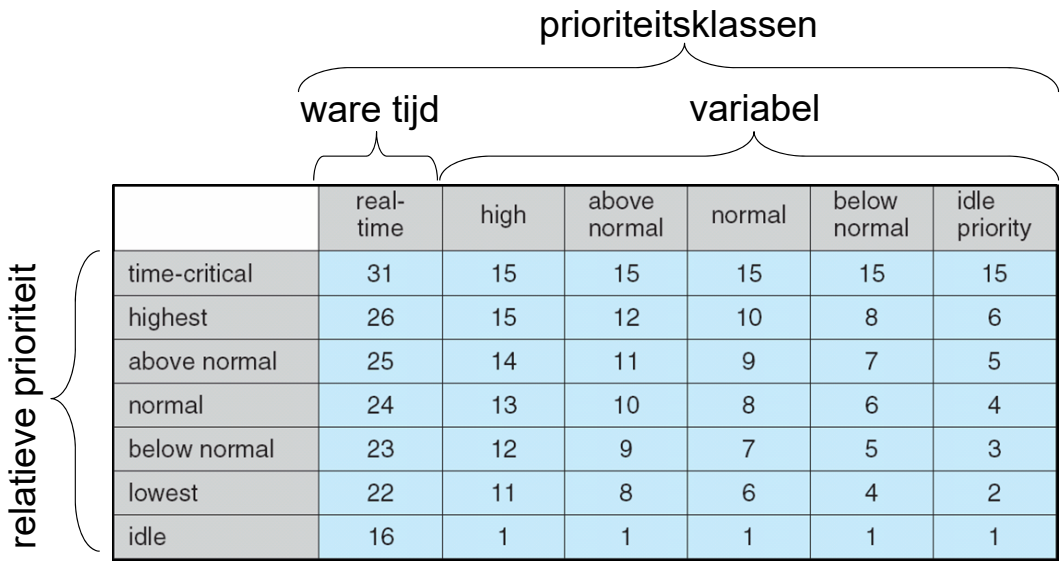
Bij Solaris varieert de lengte van het tijdskwantum in functie van de prioriteit. Hoe hoger de prioriteit, des te korter het tijdskwantum. Bovendien werkt Solaris met veranderlijke prioriteiten. Na het verstrijken van het tijdskwantum, wordt de prioriteit verlaagd zoals weergegeven in de tabel. Bij het vrijgeven van een blokkering, wordt de prioriteit verhoogd.

Naast de scheduling klassen realtime, system (b.v. de planner), timesharing en interactive (venstertoepassingen) zijn er ook nog de klassen fixed priority en fair share.

Bij fixed priority is de prioriteit onveranderlijk, en fair share probeert de toegewezen CVE-tijd gelijkmatig te verdelen over de diverse projecten (een project is een verzameling van processen, vergelijkbaar met een job in Windows).

Elke klasse heeft zijn eigen lijst van prioriteiten die finaal omgezet worden in een globale prioriteit die gebruikt wordt door de procesplanner (een preëmptieve prioriteitsplanner die processen met gelijke prioriteit RR uitvoert).

Windows



	real-time	high	above normal	normal	below normal	idle priority
time-critical	31	15	15	15	15	15
highest	26	15	12	10	8	6
above normal	25	14	11	9	7	5
normal	24	13	10	8	6	4
below normal	23	12	9	7	5	3
lowest	22	11	8	6	4	2
idle	16	1	1	1	1	1

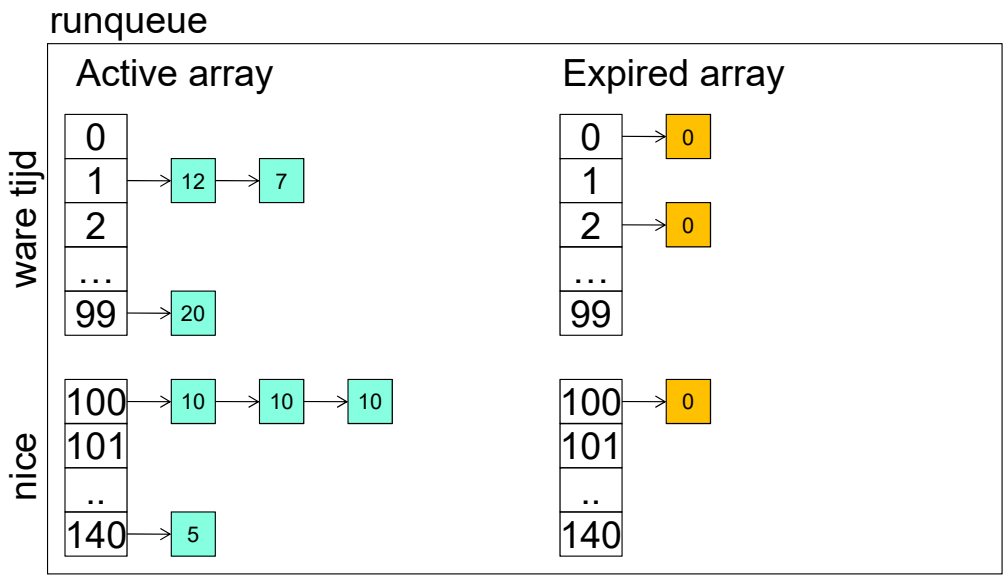
best3-45

Windows bezit slechts 32 prioriteitsniveaus die als volgt verdeeld zijn: 16-31: draden in ware tijd, 1-15: interactieve gebruikersprocessen. Prioriteit 0 wordt gebruikt voor het geheugenbeheer op zeer lage prioriteit. De procesplanner die hier dispatcher genoemd wordt overloopt de verschillende prioriteiten van hoog naar laag en selecteert de eerste draad die hij tegenkomt (preëmptieve prioriteitsplanner). Windows maakt een onderscheid tussen 6 klassen (real-time, high, above normal, normal, below normal, idle). Binnen deze klassen zijn de prioriteiten variabel, hetgeen uitgedrukt wordt door middel van een relatieve prioriteit (time-critical, highest, above normal, normal, below normal, lowest, idle). Bovenstaande tabel geeft dan de resulterende finale prioriteit weer.

Zoals gebruikelijk variëren de prioriteiten van de interactieve taken in functie van hun gedrag (voor de variabele klassen). Een proces start met de relatieve prioriteit normal (tenzij anders gespecificeerd), en daalt in prioriteit per afgelopen tijdskwantum (maar wel binnen dezelfde klasse). Indien het proces ontwaakt uit de geblokkeerde toestand, verhoogt de prioriteit (binnen dezelfde klasse). De verhoging is afhankelijk van het randapparaat: zo zal de verhoging na het wachten op een toetsenbord groter zijn dan de verhoging na het wachten op een harddiskonderbreking. Dit geeft voorrang aan de interactieve processen. Verder maakt Windows ook nog een onderscheid tussen processen die actief zijn op het scherm (die de focus bezitten), en processen die dat niet zijn.

De aanwezigheid van slechts 16 prioriteitsniveaus voor draden in ware tijd is één van de beperkingen op het inzetten van PC's met Windows als platform voor toepassingen in ware tijd.

Linux procesplanning



Preëmptief, prioriteitsplanner. De prioriteiten lopen van 0 tot 140 waarbij 0-99 realtime prioriteiten zijn en 100-140 de zgn. nice prioriteiten. Bij het begin van een plannercyclus krijgen alle taken in de runqueue hun tijdskwantum toegekend. In tegenstelling tot veel andere planners loopt het toegekende tijdskwantum op naarmate de prioriteit hoger is. Eenmaal alle taken hun tijdskwantum gekregen hebben, worden ze uitgevoerd, startend bij de taken met de hoogste prioriteit. Van zodra het tijdkwantum van een taak volledig uitgeput is, verhuist de taak naar de expired array (in de figuur staat het resterend tijdskwantum ingeschreven in het taakblokje). Als de active array geen taken meer bevat die in aanmerking komen voor uitvoering, start de hele plannercyclus opnieuw (de rol van active array en van expired array wordt dan omgekeerd). Door alle taken een tijdskwantum te geven, wordt vermeden dat taken met een lagere prioriteit verkommeren. Door taken met een hoge prioriteit een groter tijdskwantum te geven, krijgen ze meer systeemmiddelen, en bovendien zullen ze sneller geselecteerd worden door de kortetermijnplanner.

Realtime taken hebben doorgaans een vaste prioriteit. In het nice bereik (gebruikertaken) kan de prioriteit tot 5 eenheden verhoogd of verlaagd worden, in functie van de interactiviteit van de taak. Taken met lange io-tijden interageren vaak met personen en worden als interactief gemerkt. Interactieve taken krijgen een prioriteitsverhoging van 5. De taakprioriteit wordt herberekend bij de overgang van active array naar expired array. Bij elke nieuwe plannercyclus kunnen de prioriteiten veranderen.

Java

- Prioriteitsplanner (preëmptief of niet-preëmptief)
- 10 prioriteiten 1..10
  - MIN_PRIORITY = 1 (laagste prioriteit)
  - NORM_PRIORITY = 5
  - MAX_PRIORITY = 10 (hoogste prioriteit)
- Een draad erft de prioriteit van zijn ouder
- Veranderbaar met setPriority()
- Worden afgebeeld op de prioriteiten van de systeemdraden

best3-47

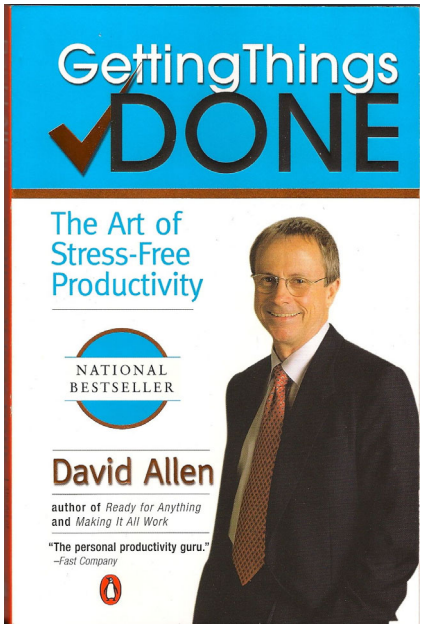
De Javaspecificatie is zeer vaag over planning – ongetwijfeld omdat men ervoor heeft proberen zorgen dat Java op zoveel mogelijk platformen kan uitgevoerd worden. De planner is een prioriteitsplanner, maar deze kan preëmptief of niet-preëmptief zijn. Bij een niet-preëmptieve planner zal men coöperatieve planning moeten toepassen waarbij de diverse draden vrijwillig afstand doen van de processor (via yield()). Java kent 10 prioriteitsniveaus en deze zijn statisch. Dit wil zeggen dat de planner de prioriteiten niet automatisch aanpast. De prioriteiten kunnen wel aangepast worden door de applicatie set (met setPriority()). De Javaprioriteiten worden afgebeeld op prioriteiten van de kerneldraden. Dit kan soms leiden tot vreemde situaties. In situaties waarbij er minstens 10 prioriteitsniveaus voor gebruikersprogramma's gereserveerd zijn, zullen alle Javaprioriteiten op individuele systeemprioriteiten afgebeeld worden. In situaties waar dit niet zo is (bv. Win 32) zullen sommige Javaprioriteiten op dezelfde systeemprioriteit afgebeeld worden. Zo zullen op het Win32 platform de Javaprioriteiten 1-2, 3-4, 6-7 en 8-9 samenvallen.

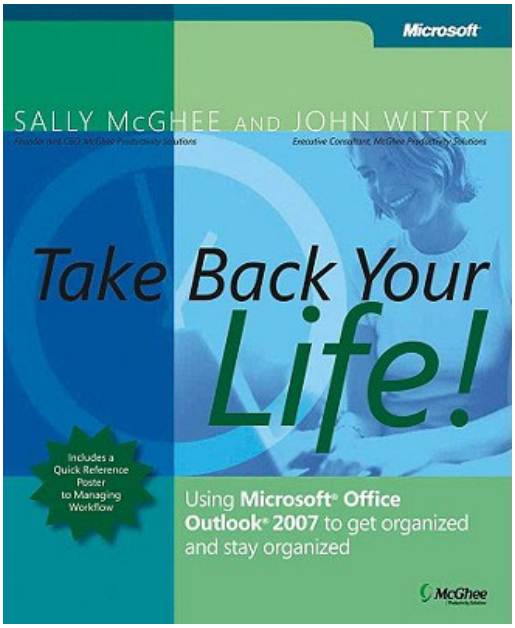
Mach

- 128 prioriteitsniveaus
- Ontvanger van een boodschap krijgt de hoogste prioriteit

best3-48

Mach heeft 128 prioriteitsniveaus en onderscheidt zich van de andere besturingssystemen doordat de ontvanger van een boodschap altijd meteen gepland wordt na het versturen van de boodschap, ongeacht zijn prioriteit. Dit zorgt ervoor dat het verzenden van boodschappen (een basismechanisme in Mach) versneld wordt.


Getting Things DONE
The Art of Stress-Free Productivity
NATIONAL BESTSELLER
David Allen
author of Ready for Anything and Making It All Work
"The personal productivity guru."
–Fast Company


Microsoft
SALLY McGHEE AND JOHN WITTRY
Take Back Your Life!
Includes a Quick Reference Poster to Managing Workflow
Using Microsoft® Office Outlook® 2007 to get organized and stay organized
McGhee

best3-49