# PENJADWALAN PROSES

## Pendahuluan

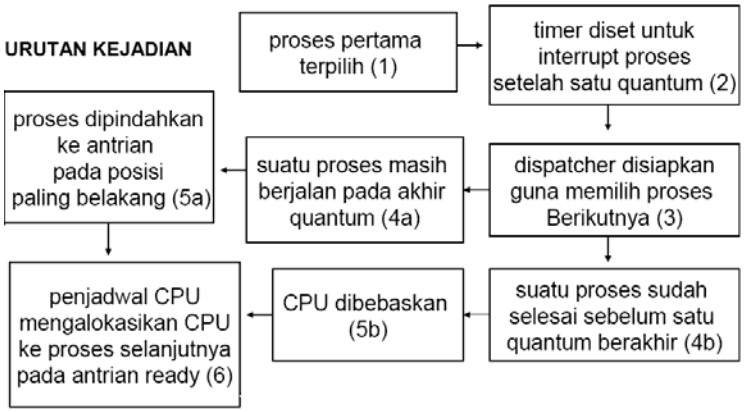
Penjadwalan berkaitan dengan permasalahan memutuskan proses mana yang akan dilaksanakan dalam suatu sistem. Proses yang belum mendapat jatah alokasi dari CPU akan mengantri di ready *queue*. Algoritma penjadwalan berfungsi untuk menentukan proses manakah yang ada di ready queue yang akan dieksekusi oleh CPU.

Kriteria untuk mengukur dan optimasi kinerje penjadwalan :

- **Adil (*fairness*),** Adalah proses-proses yang diperlakukan sama, yaitu mendapat jatah waktu pemroses yang sama dan tak ada proses yang tak kebagian layanan pemroses sehingga mengalami kekurangan waktu.
- **Efisiensi (*eficiency*),** Efisiensi atau utilisasi pemroses dihitung dengan perbandingan (rasio) waktu sibuk pemroses.
- **Waktu tanggap (response time),** Untuk Sistem interaktif didefinisikan sebagai waktu yang dihabiskan dari saat karakter terakhir dari perintah dimasukkan atau transaksi sampai hasil pertama muncul di layar. Waktu tanggap ini disebut terminal *response time*. Untuk sistem waktu nyata Didefinisikan sebagai waktu dari saat kejadian (internal atau eksternal) sampai instruksi pertama rutin layanan yang dimaksud dieksekusi, disebut *event*.
- ***Turn around time,*** Adalah waktu yang dihabiskan dari saat program atau job mulai masuk ke sistem sampai proses diselesaikan sistem. Waktu yang dimaksud adalah waktu yang dihabiskan di dalam sistem, diekspresikan sebagai penjumlah waktu eksekusi (waktu pelayanan job) dan waktu menunggu, yaitu : *Turn arround time = Burst time + Wait time*.
- ***Throughput***, Adalah jumlah kerja yang dapat diselesaikan dalam satu unit waktu. Cara untuk mengekspresikan throughput adalah dengan jumlah job pemakai yang dapat dieksekusi dalam satu unit/interval waktu.

## Round Robin

Algoritma ini menggilir proses yang ada di antrian. Proses akan mendapat jatah sebesar time quantum. Jika time quantum-nya habis atau proses sudah selesai, CPU akan dialokasikan ke proses berikutnya. Tentu proses ini cukup adil karena tak ada proses yang diprioritaskan, semua proses mendapat jatah waktu yang sama dari CPU yaitu (1/n), dan tak akan menunggu lebih lama dari (n-1)q dengan q adalah lama 1 quantum.




Algoritma ini sepenuhnya bergantung besarnya time quantum. Jika terlalu besar, algoritma ini akan sama saja dengan algoritma *first come first served*. Jika terlalu kecil, akan semakin banyak peralihan proses sehingga banyak waktu terbuang.

Permasalahan utama pada *Round Robin* adalah menentukan besarnya *time quantum*. Jika *time quantum* yang ditentukan terlalu kecil, maka sebagian besar proses tidak akan selesai dalam 1 *quantum*. Hal ini tidak baik karena akan terjadi banyak switch, padahal CPU memerlukan waktu untuk beralih dari suatu proses ke proses lain (disebut dengan *context switches time*). Sebaliknya, jika *time quantum* terlalu besar, algoritma *Round Robin* akan berjalan seperti algoritma *first come first served*. *Time quantum* yang ideal adalah jika 80% dari total proses memiliki *CPU burst time* yang lebih kecil dari 1 time quantum.

**Contoh 1:**

- *Arrival time* setiap proses sama
- *Time quantum* = 10 ms.

| ***Process*** | ***Arrival Time*** | ***Burst Time*** |
|---|---|---|
| P1 | 0 | 40 |
| P2 | 0 | 50 |
| P3 | 0 | 10 |
| P4 | 0 | 20 |

| **Proses** | **Waktu (ms)** | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | **10** | **20** | **30** | **40** | **50** | **60** | **70** | **80** | **90** | **100** | **110** | **120** | | | |
| P1 | | | | | | | | | | | | | | | |
| P2 | | | | | | | | | | | | | | | |
| P3 | | | | | | | | | | | | | | | |
| P4 | | | | | | | | | | | | | | | |

Waiting Time :

- P1 = 30 ms + 20 ms + 10 ms = 70 ms
- P2 = 10 ms + 30 ms + 20 ms + 10 ms = 70 ms
- P3 = 20 ms
- P4=30 ms + 20 ms = 50 ms
- Total *Waiting Time* : 210 ms
- Rata-rata *Waiting Time* : 210 ms/5 = 42 ms

**Contoh 2:**

- *Arrival time* masing-masing proses berbeda-beda
- *Time Quantum* = 10 ms.

| ***Process*** | ***Arrival Time*** | ***Burst Time*** |
|---|---|---|
| P1 | 0 | 40 |
| P2 | 20 | 50 |
| P3 | 50 | 10 |
| P4 | 70 | 20 |

| **Proses** | **Waktu (ms)** | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | **10** | **20** | **30** | **40** | **50** | **60** | **70** | **80** | **90** | **100** | **110** | **120** | | | | |
| P1 | | | | | | | | | | | | | | | | |
| P2 | | | | | | | | | | | | | | | | |
| P3 | | | | | | | | | | | | | | | | |
| P4 | | | | | | | | | | | | | | | | |

*Waiting Time :*

- P1 = 20 ms + 20 ms = 30 ms
- P2 = 10 ms + 20 ms +10 ms + 10 ms = 50 ms
- P3 = 0 ms
- P4=10 ms + 10 ms = 20 ms
- Total *Waiting Time* : 30 ms + 50 ms + 0 ms + 20 ms
- Rata-rata *Waiting Time* : 100 ms/4 = 25 ms

***Priority Scheduling***

Priority Scheduling merupakan algoritma penjadwalan yang mendahulukan proses yang memiliki prioritas tertinggi. Setiap proses memiliki prioritasnya masing-masing.

Prioritas suatu proses dapat ditentukan melalui beberapa karakteristik antara lain:

- Time limit.
- Memory requirement.
- Akses file.
- Perbandingan antara burst M/K dengan CPU burst.
- Tingkat kepentingan proses.

Priority scheduling juga dapat dijalankan secara preemptive maupun non-preemptive. Pada preemptive, jika ada suatu proses yang baru datang memiliki prioritas yang lebih tinggi daripada proses yang sedang dijalankan, maka proses yang sedang berjalan tersebut dihentikan, lalu CPU dialihkan untuk proses yang baru datang tersebut. Sementara itu, pada non-preemptive, proses yang baru datang tidak dapat menganggu proses yang sedang berjalan, tetapi hanya diletakkan di depan queue.

Kelemahan pada priority scheduling adalah dapat terjadinya *indefinite blocking* (*starvation*). Suatu proses dengan prioritas yang rendah memiliki kemungkinan untuk tidak dieksekusi jika terdapat proses lain yang memiliki prioritas lebih tinggi darinya.
Solusi dari permasalahan ini adalah aging, yaitu meningkatkan prioritas dari setiap proses yang menunggu dalam queue secara bertahap.

Contoh: Setiap 10 menit, prioritas dari masing-masing proses yang menunggu dalam queue dinaikkan satu tingkat. Maka, suatu proses yang memiliki prioritas 127, setidaknya dalam 21 jam 20 menit, proses tersebut akan memiliki prioritas 0, yaitu prioritas yang tertinggi (semakin kecil angka menunjukkan bahwa prioritasnya semakin tinggi).

**Contoh**

- *Arrival time* bersamaan
- *Non Preemptive*

| ***Process*** | ***Arrival Time*** | ***Burst Time*** | ***Priority*** |
|---|---|---|---|
| P1 | 0 | 10 | 1 |
| P2 | 0 | 10 | 2 |
| P3 | 0 | 5 | 1 |
| P4 | 0 | 7 | 0 |
| P5 | 0 | 8 | 0 |

| Priority | Procces | Waktu (ms) | | | | | |
|---|---|---|---|---|---|---|---|
| | | 7 | | 15 | 25 | 30 | 40 |
| 0 | P4 =7 | 7 | | | | | |
| | P5 = 8 | | 8 | | | | |
| 1 | P1 = 10 | | | 10 | | | |
| | P3 = 5 | | | | 5 | | |
| 2 | P2 =10 | | | | | 10 | |

**Contoh**

- *Arrival time* berbeda-beda
- *Non Preemptive*

| ***Process*** | ***Arrival Time*** | ***Burst Time*** | ***Priority*** |
|---|---|---|---|
| P1 | 0 | 10 | 1 |
| P2 | 5 | 10 | 2 |
| P3 | 10 | 5 | 1 |
| P4 | 15 | 7 | 0 |
| P5 | 15 | 8 | 0 |

| Priority | Procces | Waktu (ms) | | | | | | |
|---|---|---|---|---|---|---|---|---|
| | | 5 | 10 | 15 | 22 | 30 | 40 | |
| 0 | P4 =7 | | | | 7 | | | |
| | P5 = 8 | | | | | 8 | | |
| 1 | P1 = 10 | 10 | | | | | | |
| | P3 = 5 | | | 5 | | | | |
| 2 | P2 =10 | | | | | | 10 | |

***FCFS (First Come First Served)***

Algoritma ini merupakan algoritma penjadwalan yang paling sederhana yang digunakan CPU. Dengan menggunakan algoritma ini setiap proses yang berada pada status ready dimasukkan kedalam *FIFO queue* atau antrian dengan prinsip first in first out, sesuai dengan waktu kedatangannya. Proses yang tiba terlebih dahulu yang akan dieksekusi.

**Contoh:**

| ***Process*** | ***Arrival Time*** | ***Burst Time*** |
|---|---|---|
| P1 | 0 | 24 |
| P2 | 0 | 3 |
| P3 | 0 | 3 |

Hitunglah waiting time rata-rata dan *turnaround time* (*burst time* + *waiting time*) dari ketiga proses tersebut dengan menggunakan algoritma FCFS.

Urutan kedatangan adalah P1, P2 , P3; gantt chart untuk urutan ini adalah:

| Proses | Waktu (ms) | | | |
|---|---|---|---|---|
| | 24 | 27 | 30 | |
| P1 | | | | |
| P2 | | | | |
| P1 | | | | |

*Waiting time*

- P1=0 ms
- P2=24 ms
- P3=37 ms
- Rata-rata = (0 ms + 24 ms +27 ms )/3 = 17ms.

*Turnaround time*:

- P1 = 24 ms
- P2 =27 ms (dihitung dari awal kedatangan P2 hingga selesai dieksekusi),
- P3 = 30 ms.
- Rata-rata = (24 ms + 27 ms +30 ms)/3 = 27 ms.

**Kelemahan dari algoritma ini:**

- Waiting time rata-ratanya cukup lama.
- Terjadinya *convoy effect*, yaitu proses-proses menunggu lama untuk menunggu 1 proses besar yang sedang dieksekusi oleh CPU. Algoritma ini juga menerapkan konsep non-preemptive, yaitu setiap proses yang sedang dieksekusi oleh CPU tidak dapat di-interrupt oleh proses yang lain.

***SJF (Shortest Job First)***

Pada algoritma ini setiap proses yang ada di ready queue akan dieksekusi berdasarkan burst time terkecil. Hal ini mengakibatkan *waiting time* yang pendek untuk setiap proses dan karena hal tersebut maka waiting time rata-ratanya juga menjadi pendek, sehingga dapat dikatakan bahwa algoritma ini adalah algoritma yang optimal.

Algoritma ini dapat dibagi menjadi dua bagian yaitu :

- ***Preemptive*** . Jika ada proses yang sedang dieksekusi oleh CPU dan terdapat proses di ready queue dengan burst time yang lebih kecil daripada proses yang sedang dieksekusi tersebut, maka proses yang sedang dieksekusi oleh CPU akan digantikan oleh proses yang berada di ready queue tersebut. Preemptive SJF sering disebut juga *Shortest-Remaining- Time-First scheduling.*
- ***Non-preemptive*** . CPU tidak memperbolehkan proses yang ada di ready queue untuk menggeser proses yang sedang dieksekusi oleh CPU meskipun proses yang baru tersebut mempunyai burst time yang lebih kecil.

**Contoh 1.**

- *Arrival time* yang sama
- Non Preemptive

| ***Process*** | ***Arrival Time*** | ***Burst Time*** |
|---|---|---|
| P1 | 0 | 40 |
| P2 | 0 | 50 |
| P3 | 0 | 10 |
| P4 | 0 | 20 |

| Proses | Waktu (ms) | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | 10 | 20 | 30 | 40 | 50 | 60 | 70 | 80 | 90 | 100 | 110 | 120 | | | |
| P3 | | | | | | | | | | | | | | | |
| P4 | | | | | | | | | | | | | | | |
| P1 | | | | | | | | | | | | | | | |
| P2 | | | | | | | | | | | | | | | |

*Waiting Time :*

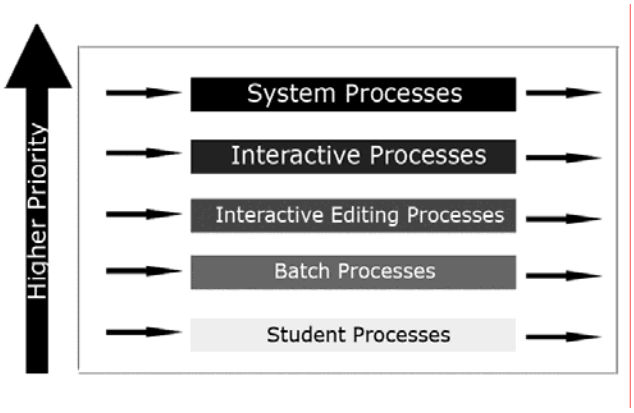
- P3 = 0 ms
- P4 = 10 ms
- P1 = 30 ms
- P2= 70 ms
- Total *Waiting Time* : 110 ms
- Rata-rata *Waiting Time* : 210 ms/5 = 42 ms

Ada beberapa kekurangan dari algoritma ini yaitu:

- Susahnya untuk memprediksi burst time proses yang akan dieksekusi selanjutnya.
- Proses yang mempunyai burst time yang besar akan memiliki waiting time yang besar pula karena yang dieksekusi terlebih dahulu adalah proses dengan burst time yang lebih kecil.

***Multilevel Queue***

Ide dasar dari algoritma ini berdasarkan pada sistem prioritas proses. Prinsipnya, jika setiap proses dapat dikelompokkan berdasarkan prioritasnya, maka akan didapati queue seperti pada gambar berikut:



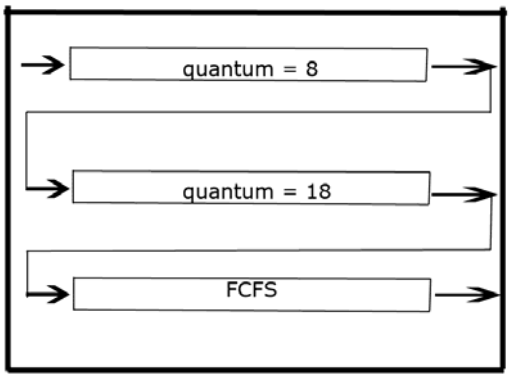
Dari gambar tersebut terlihat bahwa akan terjadi pengelompokan proses-proses berdasarkan prioritasnya. Kemudian muncul ide untuk menganggap kelompok-kelompok tersbut sebagai sebuah antrian-antrian kecil yang merupakan bagian dari antrian keseluruhan proses, yang sering disebut dengan algoritma multilevel queue.

Dalam hal ini, dapat dilihat bahwa seolah-olah algoritma dengan prioritas yang dasar adalah algoritma *multilevel queue* dimana setiap queue akan berjalan dengan algoritma FCFS yang memiliki banyak kelemahan. Oleh karena itu, dalam prakteknya, algoritma multilevel queue memungkinkan adanya penerapan algoritma internal dalam masing-masing sub-antriannya yang bisa memiliki algoritma internal yang berbeda untuk meningkatkan kinerjanya.

Berawal dari *priority scheduling*, algoritma ini pun memiliki kelemahan yang sama dengan *priority scheduling*, yaitu sangat mungkin bahwa suatu proses pada queue dengan prioritas rendah bisa saja tidak mendapat jatah CPU. Untuk mengatasi hal tersebut, salah satu caranya adalah dengan memodifikasi algoritma ini dengan adanya jatah waktu maksimal untuk tiap antrian, sehingga jika suatu antrian memakan terlalu banyak waktu, maka prosesnya akan dihentikan dan digantikan oleh antrian dibawahnya, dan tentu saja batas waktu untuk tiap antrian bisa saja sangat berbeda tergantung pada prioritas masing-masing antrian.

***Multilevel Feedback Queue***

Algoritma ini mirip sekali dengan algoritma *multilevel queue*. Perbedaannya ialah algoritma ini mengizinkan proses untuk pindah antrian. Jika suatu proses menyita CPU terlalu lama, maka proses itu akan dipindahkan ke antrian yang lebih rendah. Hal ini menguntungkan proses interaksi karena proses ini hanya memakai waktu CPU yang sedikit. Demikian pula dengan proses yang menunggu terlalu lama. Proses ini akan dinaikkan tingkatannya. Biasanya prioritas tertinggi diberikan kepada proses dengan CPU burst terkecil, dengan begitu CPU akan terutilisasi penuh dan M/K dapat terus sibuk. Semakin rendah tingkatannya, panjang CPU burst proses juga semakin besar.




Algoritma ini didefinisikan melalui beberapa parameter, antara lain:

- Jumlah antrian.
- Algoritma penjadwalan tiap antrian.
- Kapan menaikkan proses ke antrian yang lebih tinggi.
- Kapan menurunkan proses ke antrian yang lebih rendah.
- Antrian mana yang akan dimasuki proses yang membutuhkan.

Dengan pendefinisian seperti tadi membuat algoritma ini sering dipakai, karena algoritma ini mudah dikonfigurasi ulang supaya cocok dengan sistem. Tapi untuk mengatahui mana penjadwal terbaik, kita harus mengetahui nilai parameter tersebut.

*Multilevel feedback queue* adalah salah satu algoritma yang berdasar pada algoritma *multilevel queue*. Perbedaan mendasar yang membedakan multilevel feedback queue dengan multilevel queue biasa adalah terletak pada adanya kemungkinan suatu proses berpindah dari satu antrian ke antrian lainnya, entah dengan prioritas yang lebih rendah ataupun lebih tinggi, misalnya pada contoh berikut.

- Semua proses yang baru datang akan diletakkan pada queue 0 ( quantum= 8 ms).
- Jika suatu proses tidak dapat diselesaikan dalam 8 ms, maka proses tersebut akan dihentikan dan dipindahkan ke queue 1 ( quantum= 16 ms).
- Queue 1 hanya akan dikerjakan jika tidak ada lagi proses di queue 0, dan jika suatu proses di queue 1 tidak selesai dalam 16 ms, maka proses tersebut akan dipindahkan ke queue 2.
- Queue 2 akan dikerjakan bila queue 0 dan 1 kosong, dan akan berjalan dengan algoritma FCFS.

Disini terlihat bahwa ada kemungkinan terjadinya perpindahan proses antar queue, dalam hal ini ditentukan oleh time quantum, namun dalam prakteknya penerapan algoritma multilevel feedback queue akan diterapkan dengan mendefinisikan terlebih dahulu parameter-parameternya, yaitu:

- Jumlah antrian.
- Algoritma internal tiap queue.
- Aturan sebuah proses naik ke antrian yang lebih tinggi.
- Aturan sebuah proses turun ke antrian yang lebih rendah.
- Antrian yang akan dimasuki tiap proses yang baru datang.

Contoh: Terdapat tiga antrian; Q1=10 ms, FCFS Q2=40 ms, FCFS Q3=FCFS proses yang masuk, masuk ke antrian Q1. Jika dalam 10 ms tidak selesai, maka proses tersebut dipindahkan ke Q2. Jika dalam 40 ms tidak selesai, maka dipindahkan lagi ke Q3. Berdasarkan hal-hal di atas maka algoritma ini dapat digunakan secara fleksibel dan diterapkan sesuai dengan kebutuhan sistem. Pada zaman sekarang ini algoritma multilevel feedback queue adalah salah satu yang paling banyak digunakan.