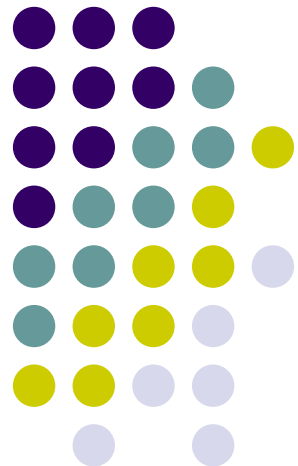
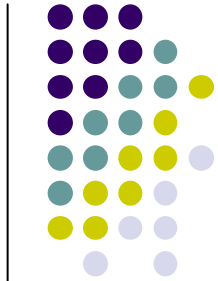
Penjadwalan CPU

Badrus Zaman

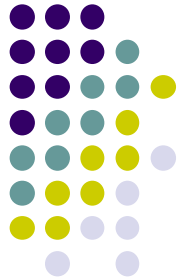


Penjadwalan CPU

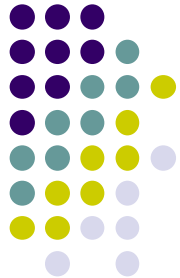


- Konsep Dasar dan Definisi
- Kriteria Penjadualan
- Algoritma Penjadualan

Konsep Dasar Penjadwalan



- SO modern umumnya merupakan sistem multitasking.
- Tujuan Utama : untuk mempunyai proses berjalan secara bersamaan, untuk memaksimalkan kinerja dari CPU.
- Pemanfaatan CPU maksimum diperoleh dengan multiprogramming
- CPU-I/O Burst Cycle- Pelaksanaan proses terdiri dari suatu siklus tunggu I/O dan eksekusi CPU



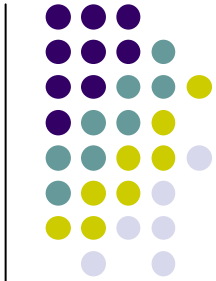
- Proses dapat juga dibagi atas 2 macam :

a. I/O-bound process – menghabiskan waktu lebih banyak untuk mengerjakan I/O daripada di CPU

b. CPU-bound process – jarang melakukan permintaan I/O, menggunakan lebih banyak waktunya di CPU

Sistem dengan kinerja yang terbaik akan memiliki kombinasi proses CPU bound dan I/O bound yang seimbang.

Definisi Penjadwalan Proses



- Definisi
  - Kumpulan kebijaksanaan dan mekanisme sistem operasi yang mengatur urutan dan jangka waktu eksekusi proses-proses yang aktif.
- Tugas
  - Memilih proses, menentukan kapan serta berapa lama proses tersebut boleh menggunakan processor

Komponen Penjadwalan Proses

1. **Antrian Penjadwalan (Scheduler Queue)**
   - Ready queue
   - I/O queue
   - Job queue
2. **Penjadwal (Scheduler)**
   - **Short-Term Scheduler**
   - **Medium-Term Scheduler**
   - **Long-Term Scheduler**

Komponen Penjadwalan Proses

2. Penjadwal (Scheduler)

suatu program dengan algoritma tertentu yang menyeleksi proses yang akan dieksekusi. Jenis scheduler :

a. **penjadwal jangka pendek (short-term scheduler)**

menyeleksi proses yang berada di antrian ready.

Komponen Penjadwalan Proses

b. **Penjadwalan jangka menengah (Medium-term Scheduler)**

- Jika ruang memori utama tidak cukup menampung proses, SO akan melakukan ***swapping*** yaitu memindahkan image process ke memori sekunder seperti disk.
- Umumnya yang dikorbankan adalah proses yang berstatus blocked atau menunggu event.
- Jika event sudah selesai, image process harus dikembalikan ke memori utama.
- Medium-term scheduler bertugas menyeleksi proses yang akan di swapping (swap-out) dan yang akan dikembalikan ke memori utama (swap-in).

Komponen Penjadwalan Proses

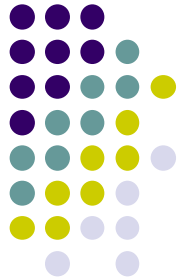
c. Dispatcher

suatu program SO yang berfungsi melakukan pengalihan eksekusi dari proses yang running ke proses yang dipilih oleh “short-term scheduler”.

bertugas memindahkan isi register prosesor ke PCB proses yang dihentikan.

Dispatch latency – terdapat waktu yang terbuang (CPU idle) dimana dispatcher menghentikan satu proses dan menjalankan proses lain.

Kriteria Penjadwalan Proses



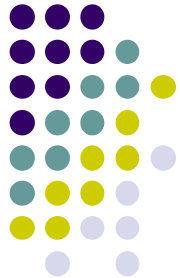
Dalam melakukan penjadwalan proses, SO mempertimbangkan sejumlah faktor:

1. Keadilan (fairness)

   Memastikan bahwa setiap proses mendapat giliran yg adil, tetapi tidak selalu berarti jatah waktu yang sama.

   perlu dipastikan tidak terjadi proses yang tidak terlayani dalam jangka waktu yang lama.

Kriteria Penjadwalan Proses



2. Efisiensi (Processor Utilization)

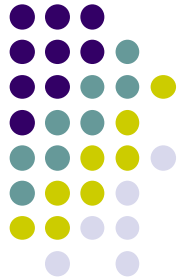
   Penggunaan waktu CPU (CPU Time) seoptimal mungkin → processor terpakai terus menerus selama masih ada antrian ready.

3. Waktu tanggapan (Response time)

   Mempercepat (secepat dan sependek mungkin) waktu tanggap dengan pemakai secara interaktif.

   response time adalah waktu antara pengguna memberikan input dengan SO memberikan output atau umpan balik ke pengguna.

Kriteria Penjadwalan Proses



4. Waiting Time

   harus seminim mungkin.

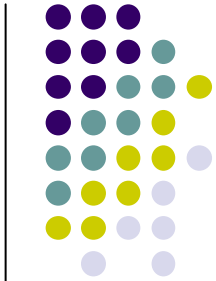
   merupakan durasi waktu yang dihabiskan suatu proses dalam antrian ready selama siklus hidupnya.

5. Turn Around Time

   harus seminim mungkin.

   merupakan durasi waktu dari suatu proses aktif dalam sistem sampai dengan selesai.

Kriteria Penjadwalan Proses



6. Throughput

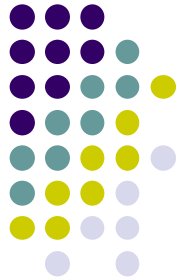
   rata-rata proses yang dapat diselesaikan per satuan waktu.

   nilai throughput harus tinggi.

Kriteria Penjadwalan yang Optimal

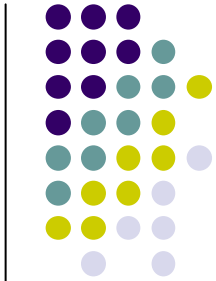
- Memaksimumkan utilisasi CPU
- Memaksimumkan throughput
- Meminimukan turnaround time
- Meminimumkan waiting time
- Meminimumkan response time

Strategi Dasar Penjadwalan



- Non-preemptive (Run to completion)
  - Algoritma Penjadualan dimana proses-proses yg sedang running tidak bisa dihentikan sementara, dan harus running terus sampai selesai
  - Strategi in bisa membahayakan sistem / proses lain
    - bila terjadi crash, maka SO tidak berfungsi
  - Umumnya digunakan pada sistem sekuential

Strategi Dasar Penjadwalan



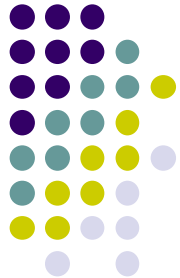
- Preemptive
  - Algoritma penjadualan yg memungkinkan beberapa proses yg sedang running, bisa dihentikan sementara.
  - Cocok untuk SO yang menerapkan multitasking, real time dan time sharing.

Pemicu terjadinya penjadwalan

- Kapan keputusan untuk algoritma dilakukan:
  - Saat suatu proses:

    1. Switch dari status running ke waiting.
    2. Switch dari status running ke ready.
    3. Switch dari status waiting ke ready.
    4. Terminates.

Algoritma Penjadwalan



- First-come, first-served (FCFS)
- Shortest-Job-First (SJF)
- Shortest-Remaining-Time-First (SRTF)
- Priority
- Round-Robin (RR)
- Multilevel Queue
- Multilevel Feedback Queue

First-Come, First-Served (FCFS)

- Setiap proses diberi jadwal eksekusi berdasarkan urutan waktu kedatangan.
- FIFO jarang digunakan secara tersendiri tetapi dikombinasikan dengan algoritma lain karena:
  - Timbul masalah “waiting time” terlalu lama jika didahului oleh proses yang waktu selesainya lama.
  - Job yang pendek harus menunggu job yang panjang
  - Job yang penting harus menunggu job yang kurang penting.

First-Come, First-Served (FCFS)

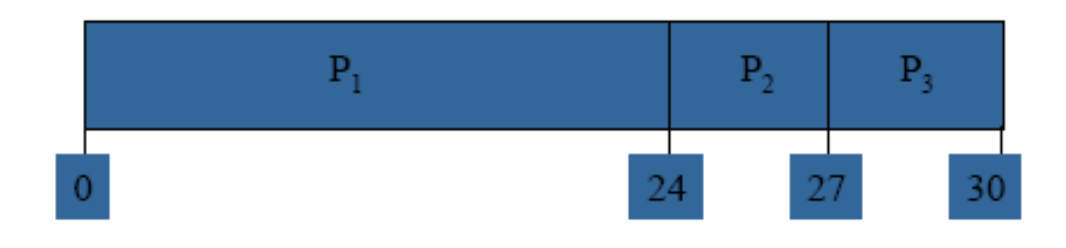
- First in First Out
- Algoritma penjadwalan non-preemptive
- FIFO cocok untuk sistem batch → sangat jarang berinteraksi dengan user.
- Sangat buruk untuk sistem interaktif atau real time karena cenderung memberikan response time yang buruk.

FCFS (Cont.)

- Example:

Process	Burst Time
P_1	24ms
P_2	3ms
P_3	3ms

- Asumsi ketiga proses masuk bersamaan, yaitu detik ke 0
- Diketahui proses yang tiba adalah P_1, P_2, P_3. Gantt chart-nya adalah :



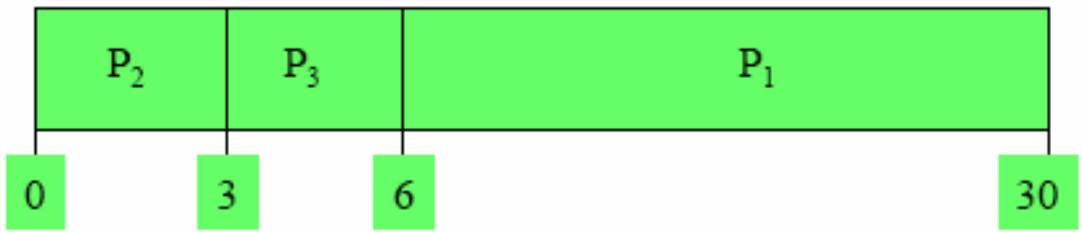
- Secara umum rumus untuk menghitung waiting time adalah

$$t_{waiting} = t_{end} - t_{start} - t_{burst\ time}$$

- Waiting time untuk *P1* = 0ms; *P2* = 24ms; *P3* = 27ms
- Average waiting time: (0 + 24 + 27)/3 = 17ms

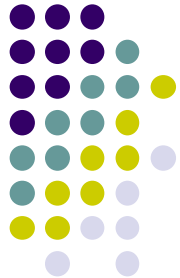
FCFS (Cont.)

- Diketahui proses yang tiba adalah P_2, P_3, P_1. Gant chart-nya adalah :



- Waiting time untuk *P1* = 6ms*; P2* = 0ms*; P3* = 3ms
- Average waiting time: (6 + 0 + 3)/3 = 3ms
  - Lebih baik dari kasus sebelumnya
- *Convoy effect* proses yang pendek diikuti proses yang panjang
- Ide dari algoritma penjadwalan SJF (Shortest Job First)

SJF (Shortest Job First)



Pada penjadwalan SJF, proses yang memiliki CPU burst paling kecil dilayani terlebih dahulu. Terdapat dua skema :

1. Non preemptive, bila CPU diberikan pada proses, maka tidak bisa ditunda sampai CPU burst selesai.
2. Preemptive, jika proses baru datang dengan panjang CPU burst lebih pendek dari sisa waktu proses yang saat itu sedang dieksekusi, proses ini ditunda dan diganti dengan proses baru. Skema ini disebut dengan Shortest-Remaining-Time-First (SRTF)

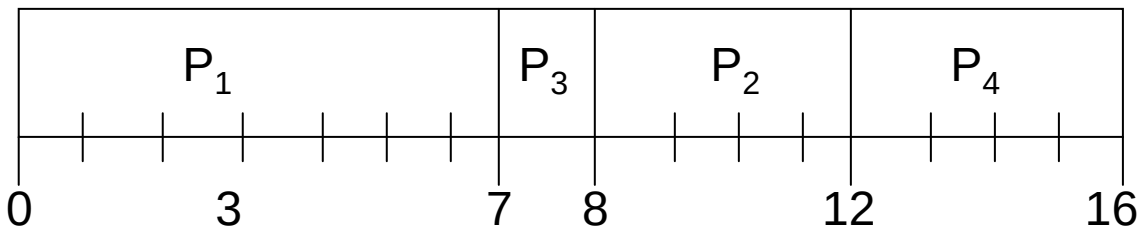
Ada beberapa kekurangan dari algoritma ini yaitu:

- Susahnya untuk memprediksi burst time proses yang akan dieksekusi selanjutnya. Diasumsikan waktu running (burst time) sudah diketahui.
- 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.

Contoh Non-Preemptive SJF

Process	Arrival Time	Burst Time
P_1	0.0ms	7ms
P_2	2.0ms	4ms
P_3	4.0ms	1ms
P_4	5.0ms	4ms

- SJF (non-preemptive)



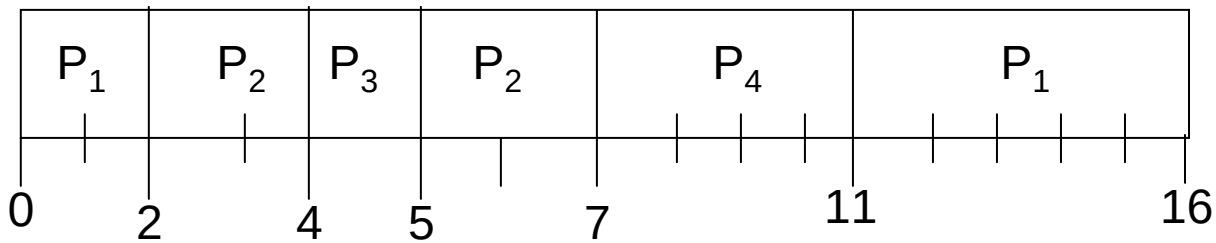
- Waiting time P1=0, P2=6, P3=3ms, P4=7ms
- Average waiting time = (0 + 6 + 3 + 7)/4 = 4ms

Contoh Preemptive SJF

(Shortest-Remaining-Time-First (SRTF)

Process	Arrival Time	Burst Time
P_1	0.0	7
P_2	2.0	4
P_3	4.0	1
P_4	5.0	4

- SJF (preemptive)

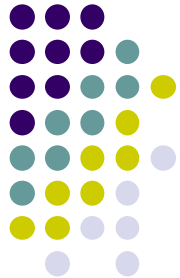


- Average waiting time = (9 + 1 + 0 +2)/4 = 3ms

Shortest-Remaining-Time-First (SRTF)

- Meskipun algoritma ini optimal, namun pada kenyataannya sulit untuk diimplementasikan karena sulit untuk mengetahui panjang *CPU burst* berikutnya.

Penjadwalan Prioritas



- Algoritma:
  - Setiap proses akan mempunyai prioritas (bilangan integer).
  - CPU diberikan ke proses dengan prioritas tertinggi (smallest integer = highest priority).
  - Preemptive: proses dapat di interupsi jika terdapat prioritas lebih tinggi yang memerlukan CPU.
  - Nonpreemptive: proses dengan prioritas tinggi akan mengganti pada saat pemakain time-slice habis.
  - Jika beberapa proses memiliki prioritas yang sama, maka akan digunakan algoritma FCFS

Penjadwalan Prioritas

- Problem = Starvation
  - Proses dengan prioritas terendah mungkin tidak akan pernah dieksekusi
  - Solution = Aging
    - Prioritas akan naik jika proses makin lama menunggu waktu jatah CPU.

- 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).

Starvation adalah kondisi dimana proses yang kekurangan resource tidak akan pernah mendapat resource yang dibutuhkan sehingga mengalami starvation (kelaparan),biasanya terjadi setelah deadlock.

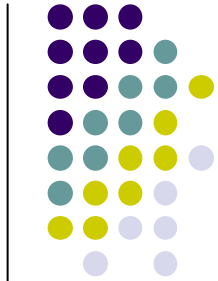
Penjadwalan Prioritas

Process	Burst Time	Priority
P1	10	3
P2	1	1
P3	2	3
P4	1	4
P5	5	2

- gantt chart :

Waktu tunggu untuk P_1 adalah 6, P_2 adalah 0, P_3 adalah 16, P_4 adalah 18 dan P_5 adalah 1 sehingga rata-rata waktu tunggu adalah $(6 + 0 + 16 + 18 + 1)/5 = 8.2$ milidetik.

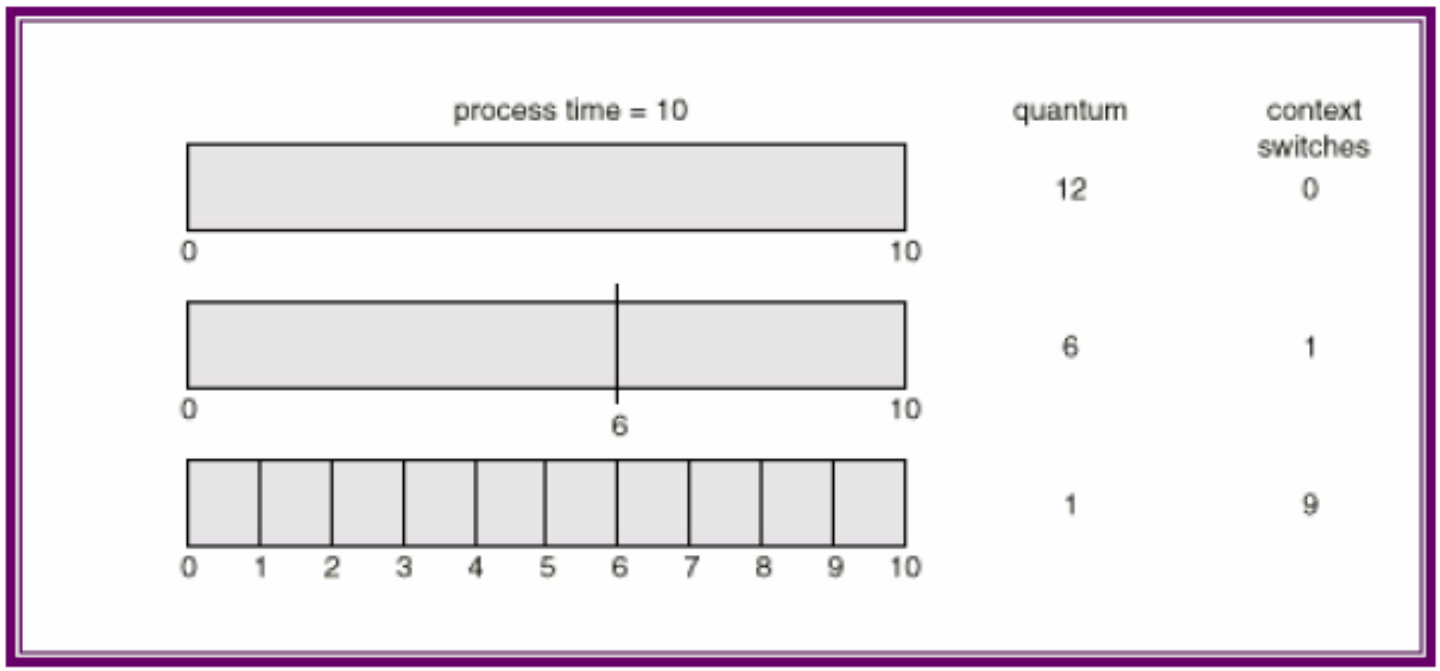
RR (Round Robin)



- Merupakan penjadwalan preemptive.
- Setiap proses dianggap penting dan mendapat jatah waktu CPU (time slice/quantum) tertentu misalkan 10 atau 100 milidetik.
  - Setelah waktu tersebut maka proses akan di-preempt dan dipindahkan ke ready queue.
  - Adil dan sederhana.

RR (Round Robin)

Algoritma Round-Robin ini di satu sisi memiliki keuntungan, yaitu adanya keseragaman waktu. Namun di sisi lain, algoritma ini akan terlalu sering melakukan switching seperti yang terlihat pada Gambar 4-4. Semakin besar quantum-timenya maka switching yang terjadi akan semakin sedikit.

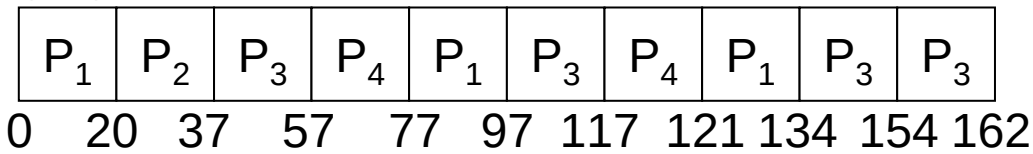


Gambar 4-4 : Menunjukkan waktu kuantum yang lebih kecil meningkatkan context switch

Contoh RR (Q= 20)

Process	Burst Time
P_1	53ms
P_2	17ms
P_3	68ms
P_4	24ms

- Gantt Chart

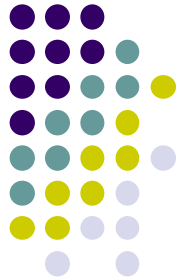


- Waiting time P1= 134 – 0 – 53 = 81ms
  P2= 37 – 0 – 17 = 20ms
  P3= 162 – 0 – 68 = 94ms
  P4= 121 – 0 – 24 = 97ms

Rata2 = (81+20+94+97)/4 = 73ms

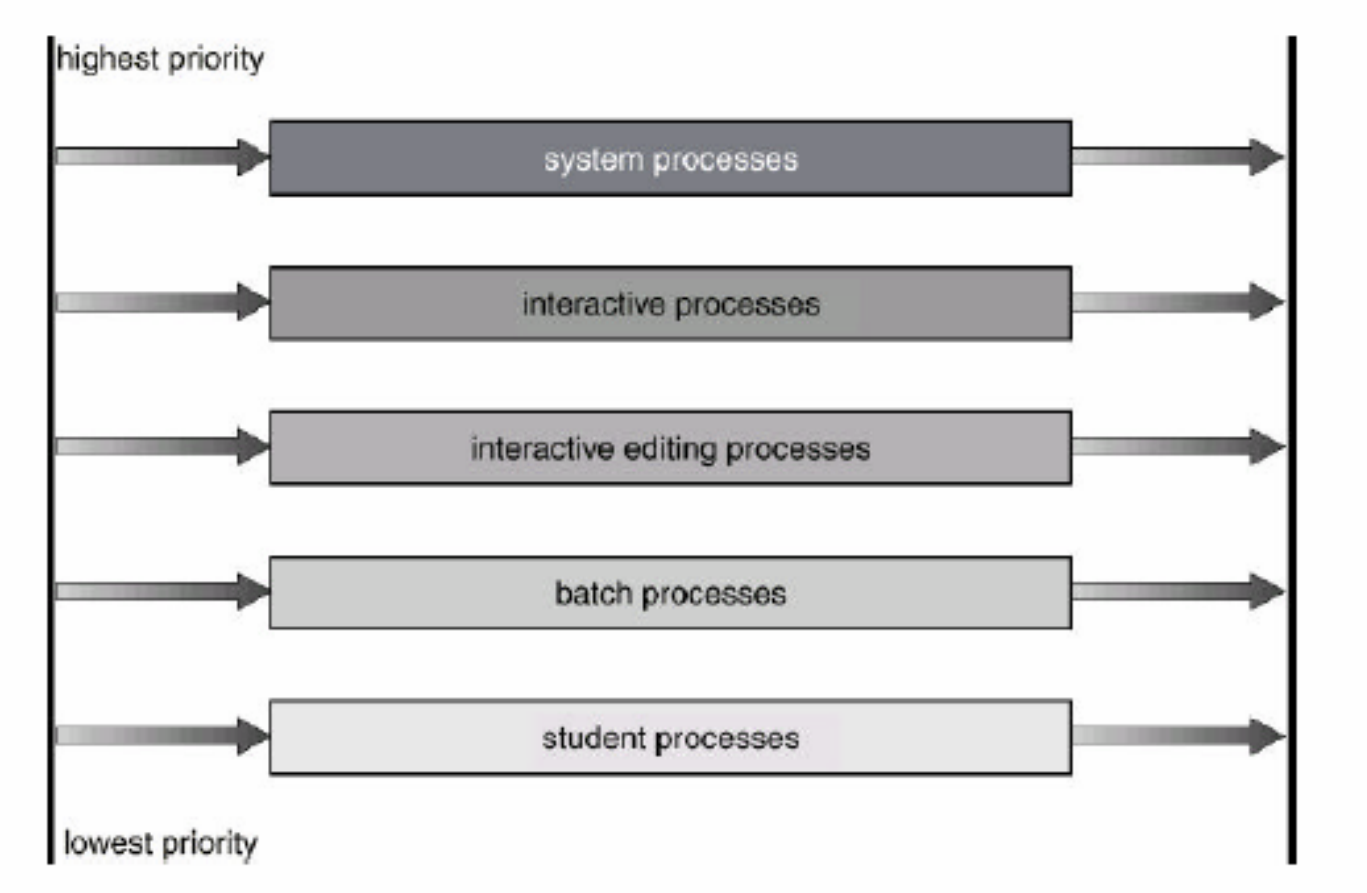
- Tipikal: lebih lama waktu rata-rata turnaround dibandingkan SJF, tapi mempunyai response terhadap user lebih cepat.

Multilevel Queue

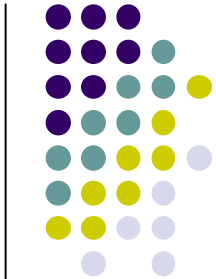


- Kategori proses sesuai dengan sifat proses:
  - Interaktif (response cepat)
  - Batch dll
- Partisi “ready queue” dalam beberapa tingkat (multilevel) sesuai dengan proses:
  - Setiap queue menggunakan algoritma schedule sendiri
  - Foreground proses (interaktif, high prioritiy): RR
  - Background proses (batch, low priority): FCFS
- Setiap queue mempunyai prioritas yang fixed.

Multilevel Queue



Multilevel Feedback Queue



- mirip sekali dengan algoritma *multilevel queue*
- Perbedaannya ialah algoritma ini mengizinkan proses untuk pindah antrian
- Algoritma ini didefinisikan melalui beberapa parameter, antara lain:

  a. Jumlah antrian.

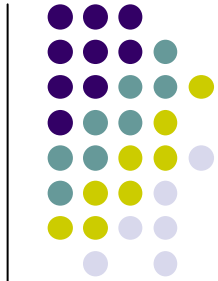
  b. Algoritma penjadwalan tiap antrian.

  c. Kapan menaikkan proses ke antrian yang lebih tinggi.

  d. Kapan menurunkan proses ke antrian yang lebih rendah.

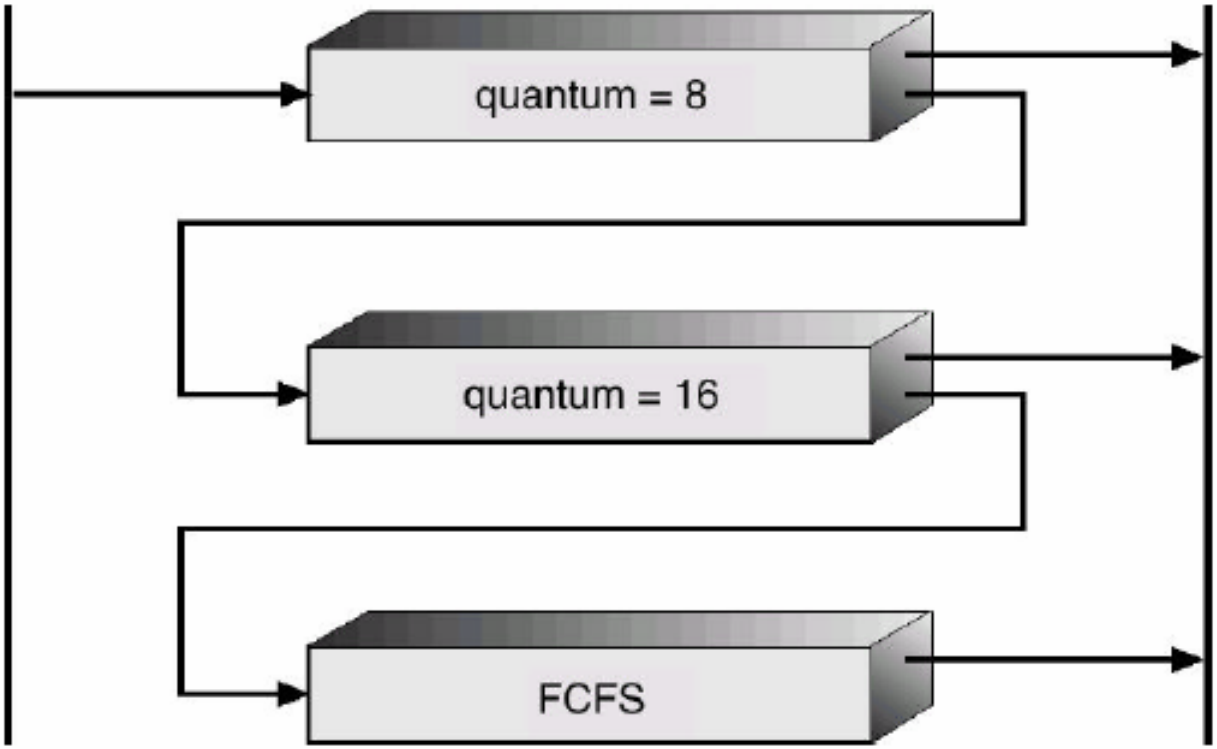
  e. Antrian mana yang akan dimasuki proses yang membutuhkan.

Multilevel Feedback Queue



- 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.

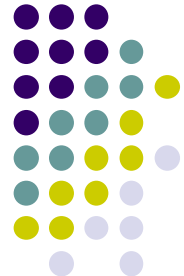
Multilevel Feedback Queue



Penjadwalan Multiple-Processor

- Seperti halnya pada prosesor tunggal, prosesor jamak juga membutuhkan penjadwalan. Namun pada prosesor jamak, penjadwalannya jauh lebih kompleks daripada prosesor tunggal karena pada prosesor jamak memungkinkan adanya *load sharing* antar prosesor yang menyebabkan penjadwalan menjadi lebih kompleks

Penjadwalan Multiple-Processor



- Penjadwalan *asymmetric multiprocessing* atau penjadwalan *master/slave* menangani semua keputusan penjadwalan, pemrosesan I/O, dan aktivitas sistem lainnya hanya dengan satu prosesor (*master*). Dan prosesor lainnya (*slave*) hanya mengeksekusi proses.
- SMP (*Symmetric multiprocessing*) adalah pendekatan kedua untuk penjadwalan prosesor jamak. Dimana setiap prosesor menjadwal dirinya sendiri (self scheduling).