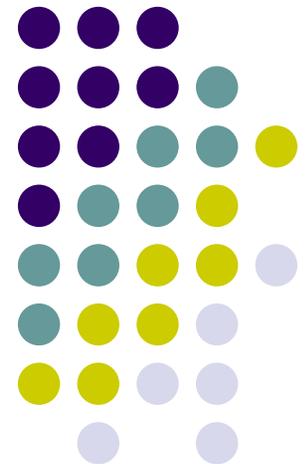


# 6

## Penjadualan CPU

---





# Penjadualan CPU

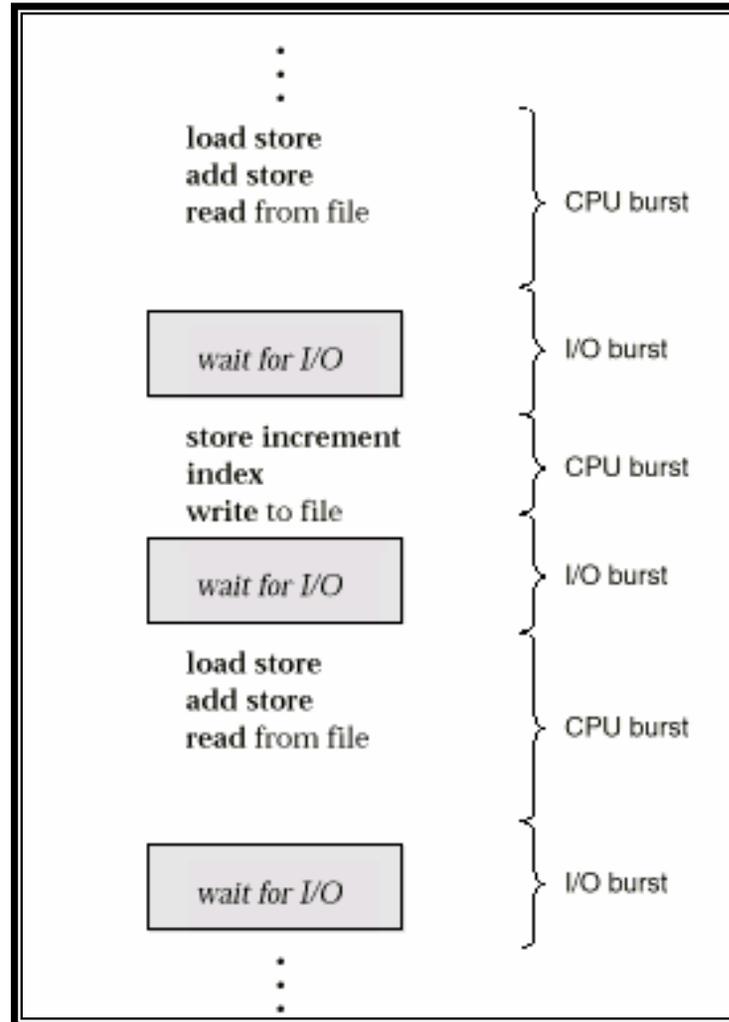
- Konsep Dasar
- Kriteria Penjadualan
- Algoritma Penjadualan
- Penjadualan Multiple-Processor
- Penjadualan Real-Time
- Evaluasi Algorithm



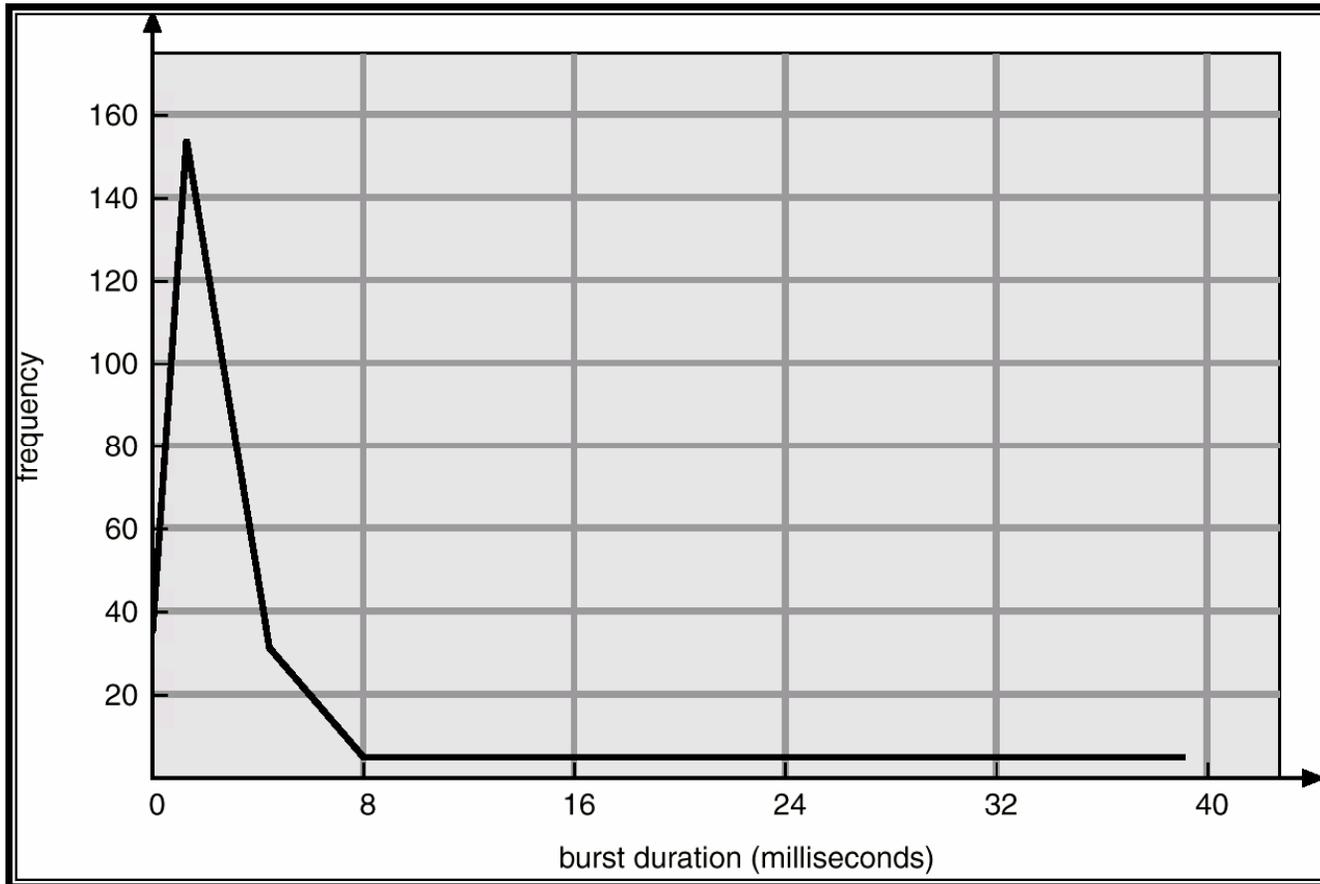
# Konsep Dasar

- Memaksimalkan kinerja CPU melalui multiprogramming
- CPU–I/O Burst Cycle – Eksekusi proses terdiri dari siklus eksekusi CPU dan I/O wait.
- Pendistribusian CPU burst

# Penggantian Rangkaian Urutan CPU dan I/O Burst



# Histogram CPU-burst Times



# Penjadual CPU



- Algoritma scheduling:
  - **Memilih** dari proses-proses yang berada di memori (ready to execute) dan memberikan jatah CPU ke salah **satu** proses tersebut.
- 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.
  - Penjadualan 1 dan 4 termasuk nonpreemptive
  - Penjadualan lainnya termasuk preemptive



# Jenis Penjadualan

- Preemptive: OS dapat mengambil (secara interrupt, preempt) CPU dari satu proses setiap saat.
- Non-preemptive: setiap proses secara sukarela (berkala) memberikan CPU ke OS.
- Contoh:
  - Penjadualan untuk switch dari running ke wait atau terminate: *non-preemptive*.
  - Penjadualan proses dari running ke ready: *pre-emptive*.
    - *Prasyarat untuk OS real-time system.*

# Dispatcher



- Modul Dispatcher: mengatur dan memberikan kontrol CPU kepada proses yang dipilih oleh “short-term scheduler”:
  - switching context
  - switching ke user mode
  - Melompat ke lokasi yang lebih tepat dari user program untuk memulai kembali program
- *Dispatch latency* – terdapat waktu yang terbuang (CPU idle) dimana dispatcher menghentikan satu proses dan menjalankan proses lain.
  - Save (proses lama) dan restore (proses baru).



# Kriteria Penjadualan

- Utilisasi CPU: menjadikan CPU terus menerus sibuk (menggunakan CPU semaksimal mungkin).
- Throughput: maksimalkan jumlah proses yang selesai dijalankan (per satuan waktu).
- Turn around time: minimalkan waktu selesai eksekusi suatu proses (sejak di submit sampai selesai).
- Waiting time: minimalkan waktu tunggu proses (jumlah waktu yang dihabiskan menunggu di ready queue).
- Response time: minimalkan waktu response dari sistim terhadap user (interaktif, time-sharing system), sehingga interaksi dapat berlangsung dengan cepat.

# Kriteria Penjadualan yang Optimal



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



# Algoritma Penjadualan

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

# First-Come, First-Served (FCFS)



- Algoritma:
  - Proses yang request CPU pertama kali akan mendapatkan jatah CPU.
  - Sederhana – algoritma maupun struktur data: menggunakan FIFO queue (ready queue).
- FIFO: Non preemptive
  - Timbul masalah “waiting time” terlalu lama jika didahului oleh proses yang waktu selesainya lama.
    - Tidak cocok untuk time-sharing systems.
    - Digunakan pada OS dengan orientasi batch job.

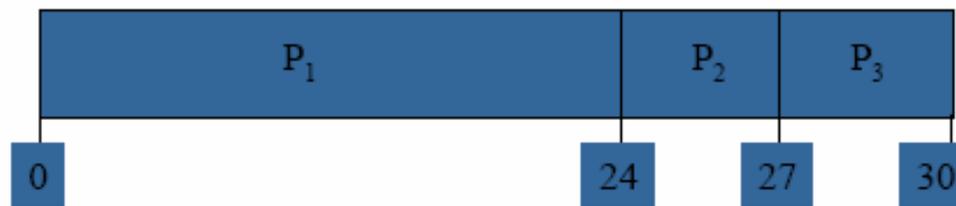


# FCFS (Cont.)

- Example: 

<u>Process</u>	<u>Burst Time</u>
$P_1$	24
$P_2$	3
$P_3$	3

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



- Waiting
- Average waiting time:  $(0 + 24 + 27)/3 = 17$



# FCFS (Cont.)

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



- Waiting time untuk  $P1 = 6$ ;  $P2 = 0$ ;  $P3 = 3$
- Average waiting time:  $(6 + 0 + 3)/3 = 3$ 
  - Lebih baik dari kasus sebelumnya
- *Convoy effect* proses yang pendek diikuti proses yang panjang

# Shortest-Job-First (SJR)



- Penggabungan setiap proses merupakan panjang dari burst CPU berikutnya. Panjang tersebut digunakan untuk penjadualan proses pada waktu terpendek
- Terdapat 2 skema :
  - *nonpreemptive* – CPU hanya satu kali diberikan pada suatu proses, maka proses tersebut tetap akan memakai CPU hingga proses tersebut melepaskannya
  - *preemptive* – jika suatu proses tiba dengan panjang CPU burst lebih kecil dari waktu yang tersisa pada eksekusi proses yang sedang berlangsung, maka dijalankan preemptive. Skema ini dikenal dengan Shortest-Remaining-Time-First (SRTF).
- SJF akan optimal, keteika rata-rata waktu tunggu minimum untuk set proses yang diberikan

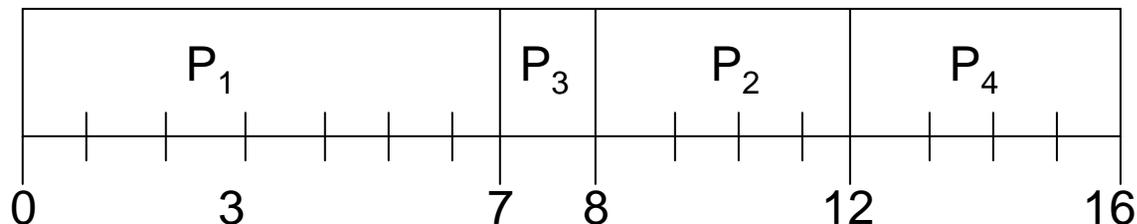


# Contoh Non-Preemptive SJF

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 (non-preemptive)



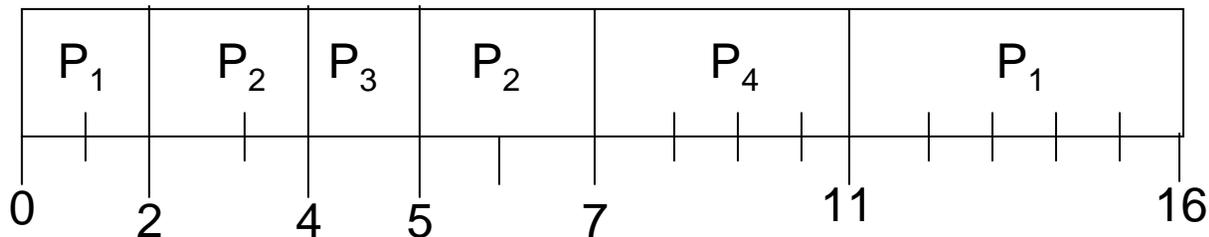
- Average waiting time =  $(0 + 6 + 3 + 7)/4 - 4$



# Contoh Preemptive SJF

<u>Process</u>	<u>Arrival</u>	<u>TimeBurst</u>	<u>Time</u>
$P_1$	0.0	7	
$P_2$	2.0	4	
$P_3$	4.0	1	
$P_4$	5.0	4	

- SJF (preemptive)



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



# Penjadualan Prioritas

- Algoritma:
  - Setiap proses akan mempunyai prioritas (bilangan integer).
  - CPU diberikan ke proses dengan prioritas tertinggi (smallest integer <sup>0</sup> 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.
  - SJF adalah contoh priority scheduling dimana prioritas ditentukan oleh waktu pemakaian CPU berikutnya.
- Problem = Starvation
  - Proses dengan prioritas terendah mungkin tidak akan pernah dieksekusi
  - Solution = Aging
    - Prioritas akan naik jika proses makin lama menunggu waktu jatah CPU.



# Round Robin (RR)

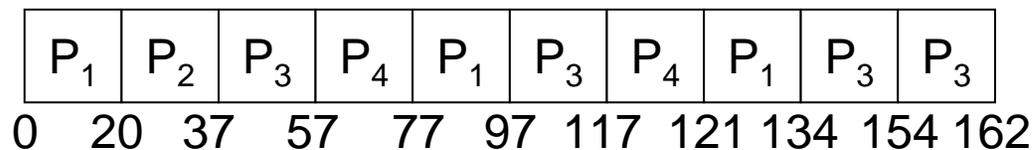
- Setiap proses mendapat jatah waktu CPU (time slice/quantum) tertentu misalkan 10 atau 100 milidetik.
  - Setelah waktu tersebut maka proses akan di-preempt dan dipindahkan ke ready queue.
  - Adil dan sederhana.
- Jika terdapat  $n$  proses di “ready queue” dan waktu quantum  $q$  (milidetik), maka:
  - Maka setiap proses akan mendapatkan  $1/n$  dari waktu CPU.
  - Proses tidak akan menunggu lebih lama dari:  $(n-1) q$  time units.
- Performance
  - $q$  besar  $\Rightarrow$  FIFO
  - $q$  kecil  $\Rightarrow q$  harus lebih besar dengan mengacu pada context switch, jika tidak overhead akan terlalu besar



# Contoh RR (Q= 20)

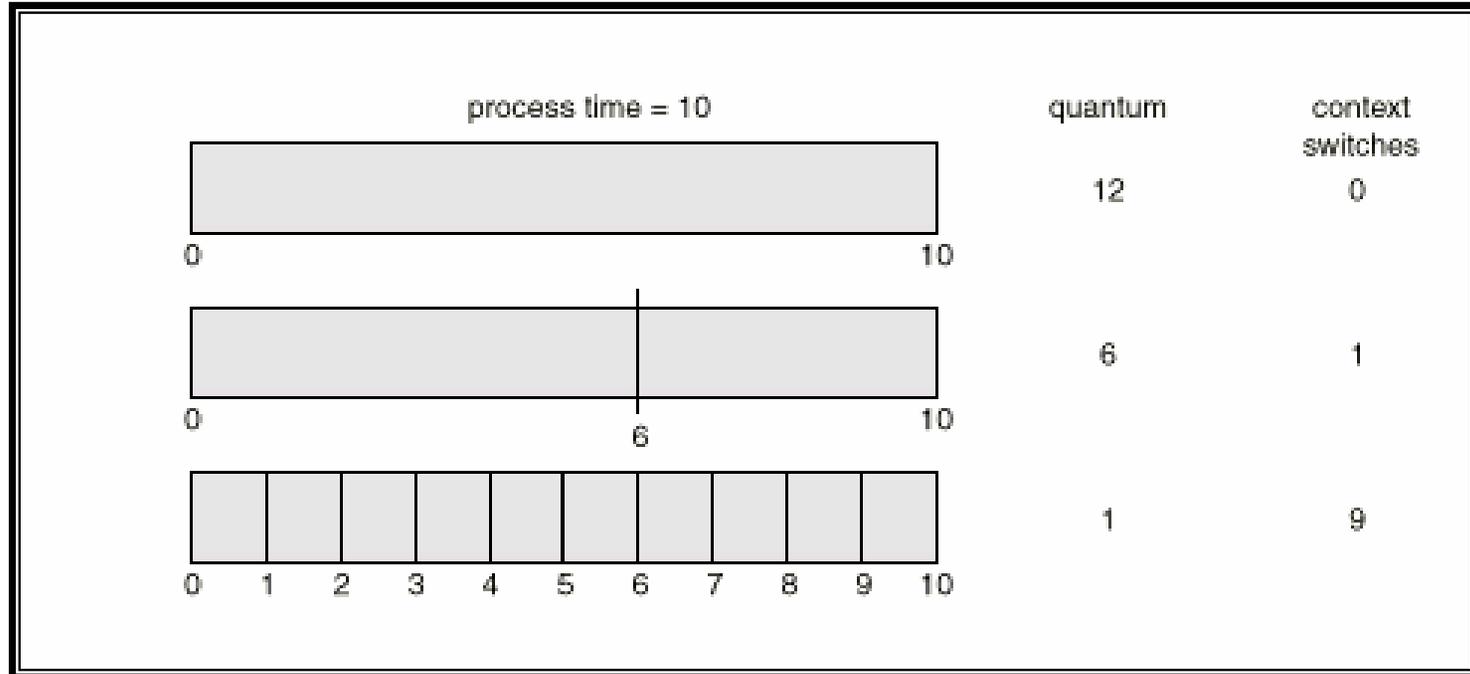
<u>Process</u>	<u>Burst Time</u>
$P_1$	53
$P_2$	17
$P_3$	68
$P_4$	24

- Gantt Chart



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

# Waktu Kuantum dan Waktu Context Switch



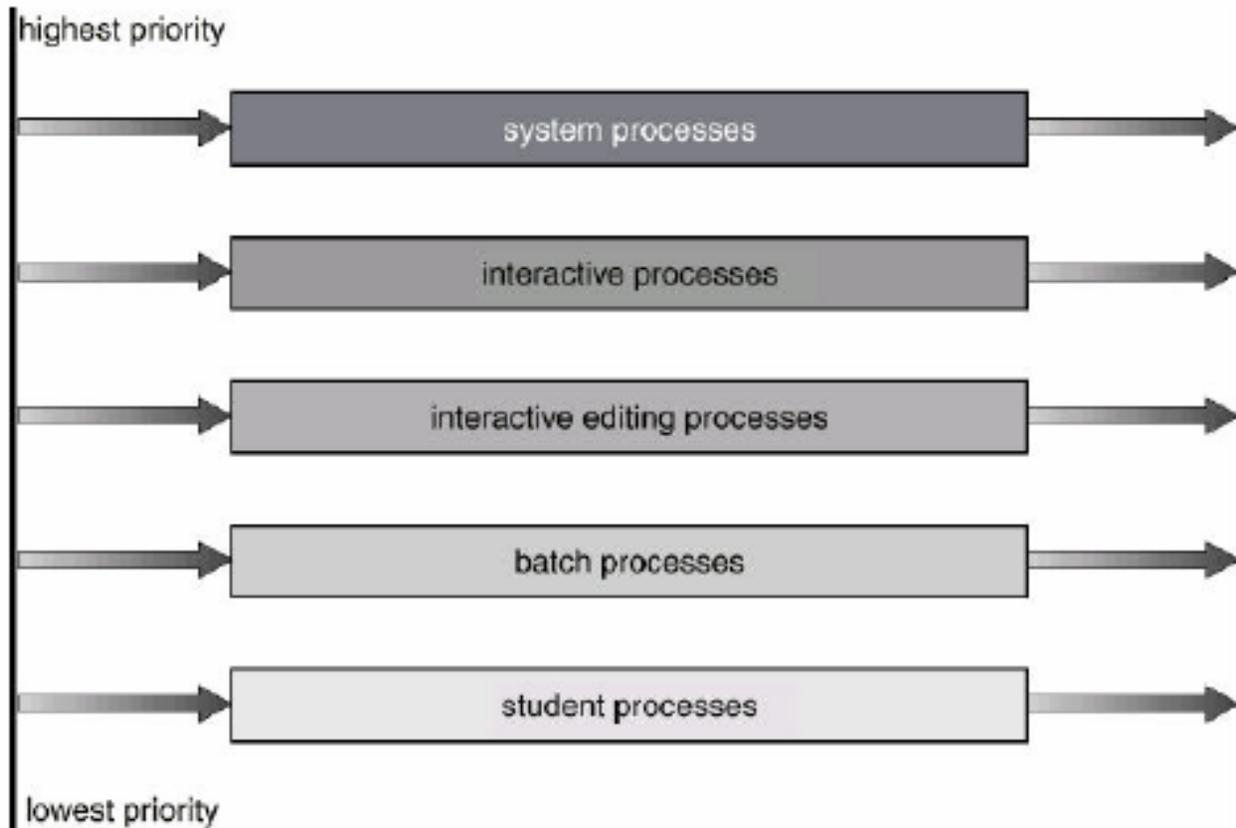


# Penjadualan Antrian Multitingkat

- 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 priority): RR
  - Background proses (batch, low priority): FCFS
- Setiap queue mempunyai prioritas yang fixed.



# Penjadualan Antrian Multitingkat

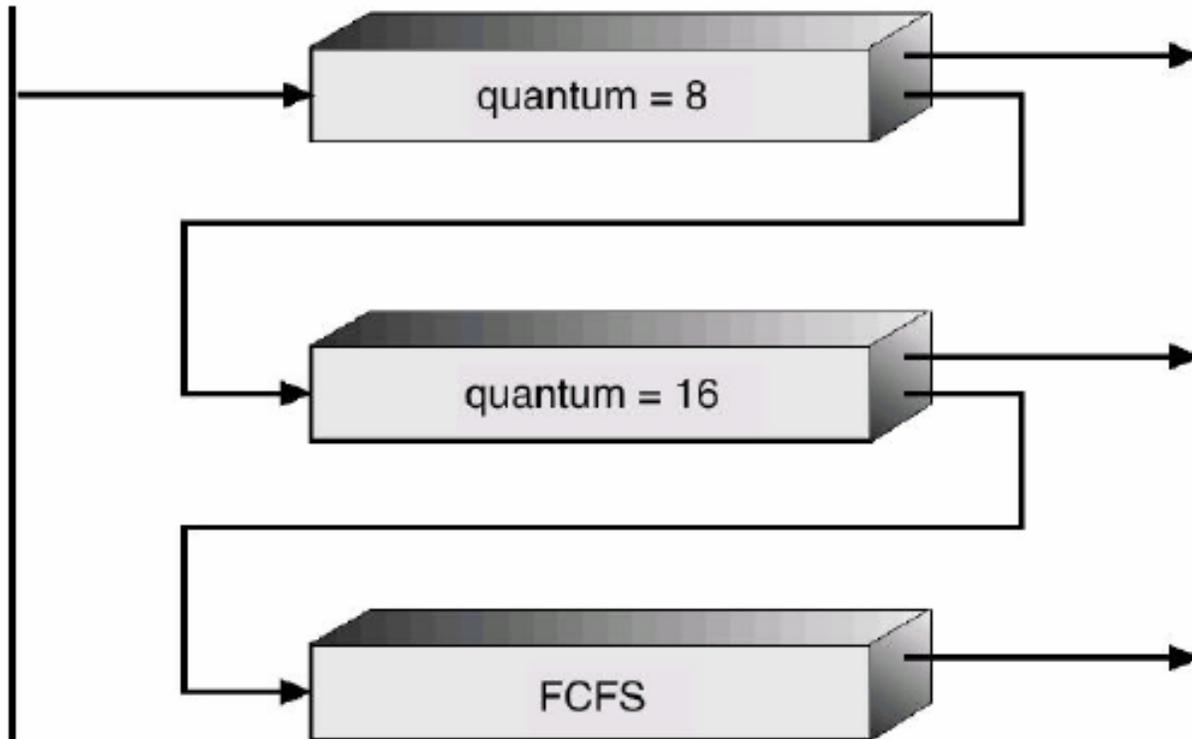


# Antrian Multitingkat Berbalikan



- Suatu proses dapat berpindah diantara beragam antrian;
- Perlu feedback untuk penentuan proses naik/turun prioritasnya (dinamis):
  - Aging dapat diimplementasikan sesuai dengan lama proses pada satu queue.
  - Suatu proses yang menggunakan CPU sampai habis (tanpa I/O wait) => CPU-bound (bukan proses interaktif) dapat dipindahk ke queue dengan prioritas lebih rendah

# Antrian Multitingkat Berbalikan





# Penjadualan Multiple-Processor

- Penjadualan CPU lebih kompleks ketika terdapat multiple Processor
- Processor yang homogen termasuk ke dalam multiprocessor
- *Homogeneous processors* within a multiprocessor.
- *Load sharing*
- *Asymmetric multiprocessing* – hanya ada satu processor yang dapat mengakses struktur sistem data, only one processor accesses the system data structures, sehingga meringankan kebutuhan sharing data

# Penjadualan Real-Time



- *Hard real-time systems*
  - Task kritis harus selesai dengan garansi waktu tertentu
  - OS akan melacak lamanya task tersebut dieksekusi (real time):
    - Mengetahui lama waktu system call, fungsi dan response dari hardware
    - Melakukan prediksi apakah task tersebut dapat dijalankan.
  - Mudah dilakukan untuk OS khusus pada peralatan/ pemakaian khusus (single task: control system)
  - Sulit untuk time-sharing sistim, virtual memory (faktor latency sebagian program aktif ada di disk).



# Penjadualan Real-Time

- *Soft real-time systems*
  - Membutuhkan penggunaan skema prioritas
  - Multimedia, highly interactive graphics
  - Prioritas tidak menurunkan over time
  - Dispancy latency yang rendah :
    - Penyisipan point preemsi sepanjang waktu system calls
    - Membuat keseluruhan kernel preemptable