# Sistem Operasi
## (Penjadwalan CPU)

Oleh

Ir. I Gede Made Karma, MT



## PENJADWALAN CPU

- Konsep Dasar
- Kriteria Penjadwalan
- Algoritma Penjadwalan
- Penjadwalan *Multiple-Processor*
- Penjadwalan *Real-Time*
- Evaluasi Algoritma

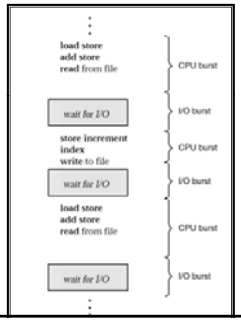


## Konsep Dasar

- Utilisasi CPU maksimum diperoleh dengan *multiprogramming*
- CPU–I/O Burst Cycle – eksekusi proses meliputi sebuah siklus eksekusi CPU dan menunggu I/O
- Distribusi CPU secara penuh

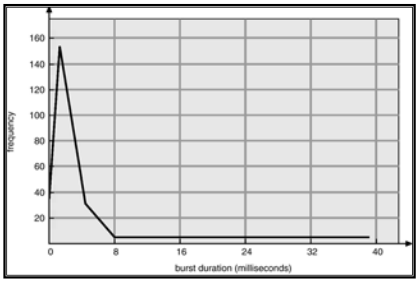


## Pertukaran Urutan Pada CPU Dan I/O Burts






## Histogram Waktu CPU-Burst






## Penjadwal CPU

- Memilih di antara proses-proses dalam memori yang siap dieksekusi, dan mengalokasikan CPU ke salah satu di antara proses tersebut
- Penjadwalan CPU:
  1. Memindah dari status ***running*** ke ***wait***
  2. Memindah dari status ***running*** ke ***ready***
  3. Memindah status dari ***waiting*** ke ***ready***
  4. ***Terminate***
- Penjadwalan no. 1 dan 4 adalah *nonpreemptive*
- Penjadwalan yang lain adalah *preemptive*

## Dispatcher

- Modul *dispatcher* memberikan kendali pada CPU untuk proses terseleksi dengan *short-term scheduler*; meliputi:
  - mengubah context
  - mengubah ke mode *user*
  - Meloncatkan ke lokasi yang tepat dalam program *user* untuk mengulang kembali program
- *Dispatch latency* – waktu yang diperlukan oleh *dispatcher* untuk menghentikan sebuah proses dan mulai menjalankan proses yang lainnya

## Kriteria Penjadwalan

- CPU *utilization*– menjaga kesibukan CPU semaksimal mungkin
- *Throughput* – banyaknya proses lengkap yang dieksekusi per satuan waktu
- *Turnaround time* – lama waktu yang diperlukan untuk mengeksekusi sebuah proses
- *Waiting time* – lamanya waktu menunggu pada sebuah proses yang berada dalam antrian **ready**
- *Response time* – lamanya waktu yang diperlukan semenjak sebuah *request* dikirim hingga respon diberikan pertama kali, **bukan** *output* (untuk lingkungan *time-sharing*)

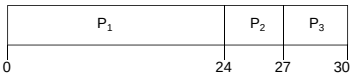
## Kriteria Optimal

- CPU *utilization* maksimum
- *Throughput* maksimum
- *Turnaround time* minimum
- *Waiting time* minimal
- *Response time* minimal

## Pernjadwalan First-Come, First-Served (FCFS)

| Proses | Burst Time |
|---|---|
| $P_1$ | 24 |
| $P_2$ | 3 |
| $P_3$ | 3 |

- Andaikan proses-proses tersebut lewat dalam: $P_1$ , $P_2$ , $P_3$ *Gantt Chart* jadwal adalah:

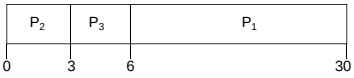



- Waiting time for $P_1$ = 0; $P_2$ = 24; $P_3$ = 27
- Average waiting time: (0 + 24 + 27)/3 = 17

## FCFS Scheduling (Cont.)

Suppose that the processes arrive in the order $P_2$ , $P_3$ , $P_1$ .

- The Gantt chart for the schedule is:




- *Waiting time* untuk $P_1$ = 6; $P_2$ = 0 ; $P_3$ = 3
- Rata-rata *waiting time*: (6 + 0 + 3)/3 = 3
- Lebih baik dari kasus sebelumnya
- *Convoy effect* proses pendek di samping proses panjang

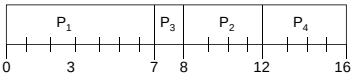
## Penjadwalan Shortest-Job-First (SJF)

- Dikaitkan dengan lamanya masing-masing proses di *CPU* selanjutnya. Lamanya waktu proses digunakan sebagai dasar penjadwalan untuk memperoleh waktu terpendek
- 2 skema:
  - *nonpreemptive* – CPU akan diberikan sekali lagi kepada proses yang tidak dapat di-*preempt* hingga selesai sepenuhnya dilaksanakan CPU
  - *preemptive* – jika ada proses baru yang melewati CPU yang panjangnya kurang dari waktu proses yang sedang berjalan, maka di-preempt. Skema ini disebut *Shortest-Remaining-Time-First* (SRTF)
- SJF adalah optimal – memberikan rata-rata waktu tunggu minimum untuk sejumlah proses yang diberikan

## Contoh Non-Preemptive SJF

| Proses | 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)

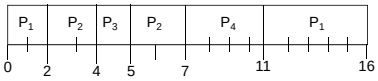



- Rata-rata waiting time = (0 + 6 + 3 + 7)/4 - 4

## Contoh Preemptive SJF

| Proses | 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)




- Rata-rata waiting time = (9 + 1 + 0 +2)/4 = 3

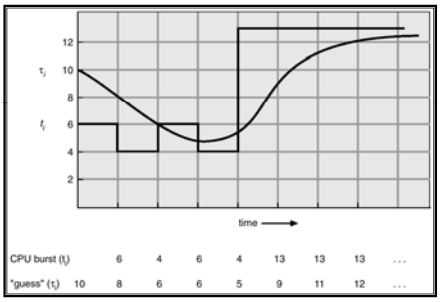
## Menentukan Panjang CPU Burst Selanjutnya

- Hanya dapat diperkirakan
- Dapat dikerjakan dengan menggunakan panjang CPU bursts sebelumnya, menggunakan rata-rata *exponential*

1. $t_n$ = actual lenght of $n^{th}$ CPU burst
2. $\tau_{n+1}$ = predicted value for the next CPU burst
3. $\alpha, 0 \le \alpha \le 1$
4. Define :

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

## Prediksi Panjang CPU Burst Selanjutnya




## Contoh Rata-rata Exponential

- $\alpha = 0$
  - $\tau_{n+1} = \tau_n$
  - Nilai sebelumnya tidak dihitung
- $\alpha = 1$
  - $\tau_{n+1} = t_n$
  - Hanya *CPU burst* aktual terakhir yang dihitung
- Jika formula dikembangkan, diperoleh:

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

- Jika $\alpha$ dan $(1-\alpha) <= 1$, masing-masing *term* selanjutnya lebih kecil dari pada sebelumnya

## Penjadwalan Dengan Prioritas

- Nomor prioritas *(integer)* dikaitkan dengan masing-masing proses
- CPU dialokasikan ke proses dengan prioritas tertinggi (nilai *integer* terkecil ≡ prioritas tertinggi)
  - *Preemptive; prioritas lebih tinggi langsung menggantikan*
  - *Nonpreemptive; menunggu sampai waktu habis.*
- SJF adalah penjadwalan dengan prioritas dimana prioritas diprediksikan pada berdasarkan *CPU burst time* selanjutnya
- Masalah ≡ ***starvation*** – proses dengan prioritas rendah bisa jadi tidak pernah dieksekusi
- Solusi ≡ ***aging*** – sebagai waktu *progress* yang menambah prioritas proses

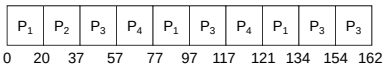
## Round Robin (RR)

- Masing-masing proses memperoleh sebagian kecil waktu CPU (*time quantum*), biasanya 10-100 *millisecond*. Setelah waktu tersebut terlewati, proses dikosongkan kembali dan ditambahkan ke akhir antrian ***ready***
- Jika ***n*** adalah proses dalam antrian ***ready*** dan ***q*** adalah *time quantum*, kemudian setiap proses memperoleh ***1/n*** waktu CPU yaitu sepotong dari ***q*** satuan waktu. Tidak ada proses menunggu yang lebih dari ***(n-1)q*** satuan waktu
- Kinerja:
  - ***q*** besar ⇒ **FIFO**
  - ***q*** kecil ⇒ ***q*** harus memiliki respek *context switch* cukup besar, jika tidak waktu tambahan akan sangat tinggi

## Contoh RR Dengan Time Quantum = 20

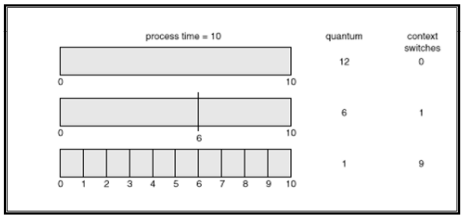
| Process | Burst Time |
|---|---|
| $P_1$ | 53 |
| $P_2$ | 17 |
| $P_3$ | 68 |
| $P_4$ | 24 |

- Gantt chart:

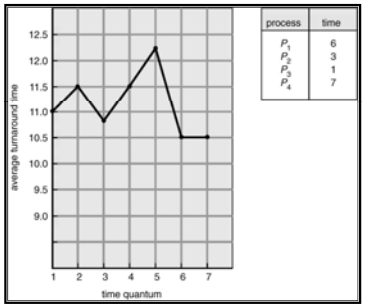



- Khas, rata-rata *turnaround* > SJF, tetapi memiliki *response* lebih baik

## Time Quantum Dan Context Switch Time



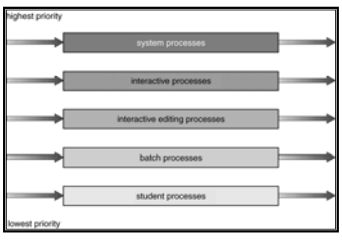

## Variasi Turnaround Time Dengan Time Quantum




## Multilevel Queue

- Antrian ***ready*** dibagi ke dalam beberapa antrian terpisah:
  → *foreground (interactive)*
  → *background (batch)*
- Masing-masing antrian memiliki algoritma penjadwalan sendiri
  → foreground – RR
  → background – FCFS
- Penjadwalan harus dikerjakan di antara antrian
  - *Fixed priority scheduling*; (misal, melayani seluruh *foreground* kemudian *background*). Sangat mungkin pada *starvation*
  - *Time slice* – masing-masing antrian memperoleh sejumlah bagian waktu tertentu pada CPU yang dapat dijadwalkan di antara proses; misal, 80% untuk *foreground* dalam RR
  - 20% untuk *background* dalam FCFS

## Penjadwalan Multilevel Queue



## Multilevel Feedback Queue

- Sebuah proses dapat berpindah di antara sejumlah variasi antrian; *aging* dapat diimplementasikan untuk hal ini.
- Penjadwal *multilevel-feedback-queue* didefinisikan oleh parameter berikut:
  - Jumlah antrian
  - Algoritma penjadwalan untuk masing-masing antrian
  - Metode yang digunakan untuk menentukan kapan untuk *upgrade* sebuah proses
  - Metode yang digunakan untuk menentukan menurunkan sebuah proses
  - Metode yang digunakan untuk menentukan antrian mana yang akan masuk yang memerlukan layanan

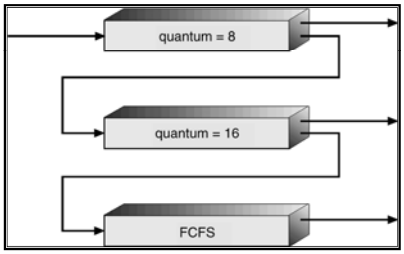


## Contoh Multilevel Feedback Queue

- Tigas antrian:
  - $Q_0$ – time quantum 8 *millisecond*
  - $Q_1$ – time quantum 16 *millisecond*
  - $Q_2$ – FCFS
- Penjadwalan
  - Sebuah *job* baru masuk antrian $\boldsymbol{Q_0}$ yang dilayani secara FCFS. Ketika diterima CPU, *job* menerima 8 *millisecond*. Jika tidak selesai dalam 8 *millisecond*, *job* dipindah ke antrian $\boldsymbol{Q_1}$
  - Pada $\boldsymbol{Q_1}$ *job* dilayani kembali dengan FCFS dan menerima tambahan 16 *millisecond*. Jika masih belum selesai, akan dikosongkan dan dipindahkan ke antrian $\boldsymbol{Q_2}$



## Multilevel Feedback Queues






## Penjadwalan Multiple-Processor

- Penjadwalan CPU lebih komplek daripada multiple CPU yang tersedia
- *Homogeneous processor* yang tidak lebih dari sebuah *multiprocessor*
- *Load sharing*
- *Asymmetric multiprocessing* – hanya 1 *processor* yang mengakses struktur data sistem, mengurangi kebutuhan untuk *data sharing*

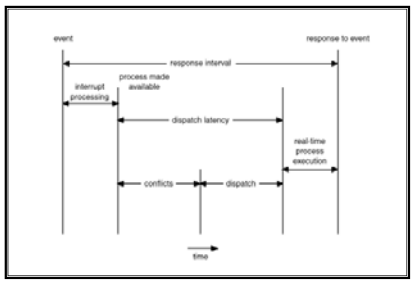


## Penjadwalan Real-Time

- Sistem *hard real-time* system – dibutuhkan untuk menyelesaikan tugas-tugas kritis dengan jumlah waktu terbatas
- Komputasi *soft real-time* – dibutuhkan oleh proses-proses kritis yang menerima prioritas lebih dari lainnya



## Dispatch Latency

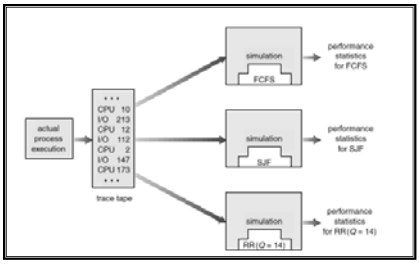


## Evaluasi Algoritma

- *Deterministic modeling* – menggunakan beban pekerjaan yang dibuka sebelumnya dan mendefinisikan kinerja masing-masing kinerja untuk menyelesaikan beban pekerjaan
- Model-model anrian
- Implementasi

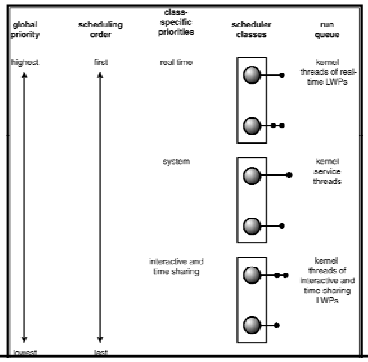


## Evaluasi Penjadwal CPU dengan Simulasi






## Penjadwalan Solaris 2






## Prioritas-Prioritas Windows 2000

| | 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 |