MATA KULIAH SISTEM OPERASI
KODE MATA KULIAH CCS210

DISUSUN OLEH
NIZIRWAN ANWAR & TEAM

FAKULTAS ILMU KOMPUTER
UNIVERSITAS ESA UNGGUL
JAKARTA
2018

MATERI

"SCHEDULING"
(PENJADUALAN)

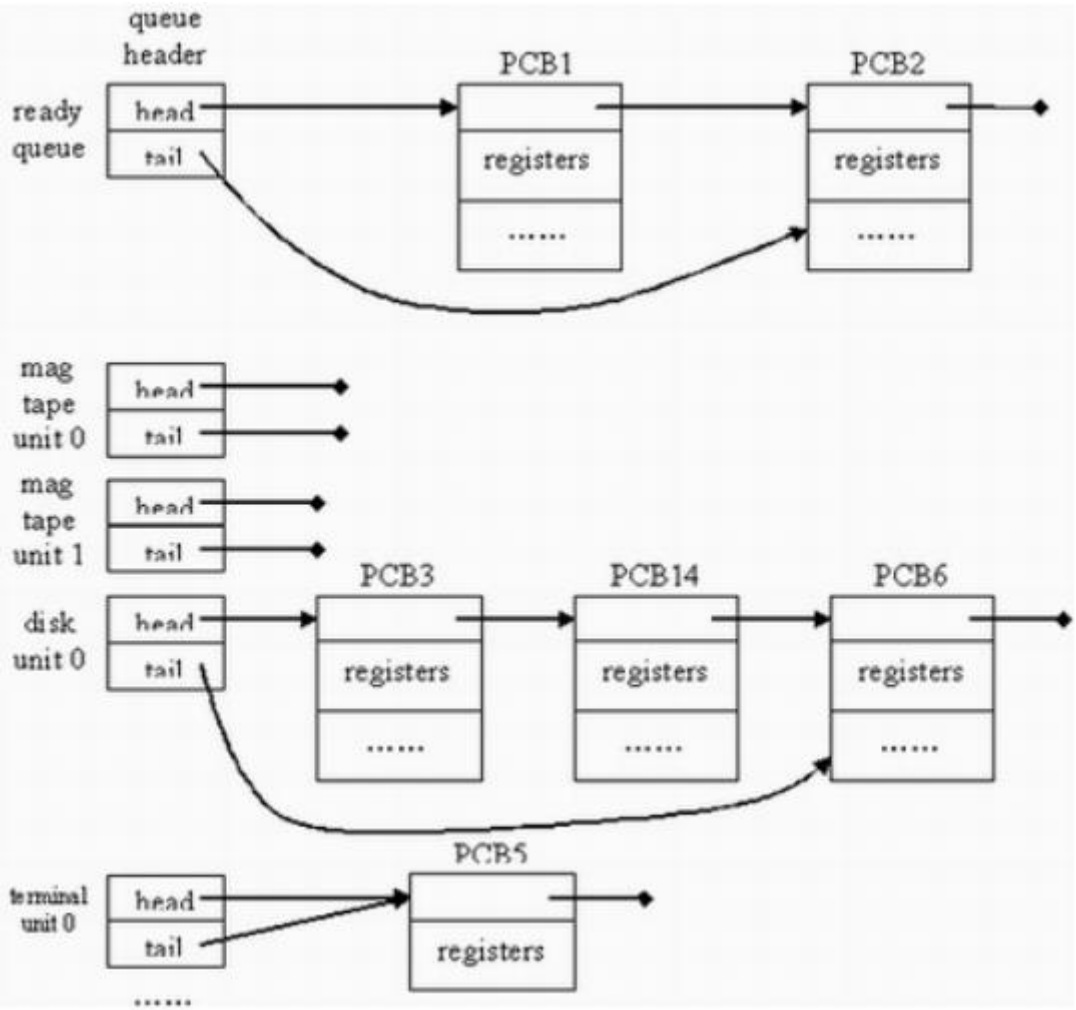
4.1 Konsep Penjadualan

Multiprogramming bertujuan untuk memaksimalkan penggunaan CPU dengan cara mengatur alokasi waktu yang digunakan oleh CPU, sehingga proses berjalan sepanjang waktu dan memperkecil waktu *idle*. Untuk sistem yang bersifat prosesor tunggal (*uniprosesor*), hanya ada satu proses yang dapat berjalan setiap waktunya. Jika proses yang ada lebih dari satu, maka proses yang lain harus menunggu sampai CPU bebas dan siap untuk dijadualkan kembali.

4.2 "Queue Scheduling"

Ketika sebuah proses memasuki sistem, proses itu diletakkan di dalam ***job queue***. Pada antrian ini terdapat seluruh proses yang berada dalam sistem. Sedangkan proses yang berada pada memori utama, siap dan menunggu untuk mengeksekusi disimpan dalam sebuah daftar yang bernama ***ready queue***. Antrian ini biasanya disimpan sebagai *linked list*. *Header* dari *ready queue* berisi *pointer* untuk PCB pertama dan PCB terakhir pada list. Setiap PCB memiliki *pointer field* yang menunjuk kepada PCB untuk proses selanjutnya dalam *ready queue*.

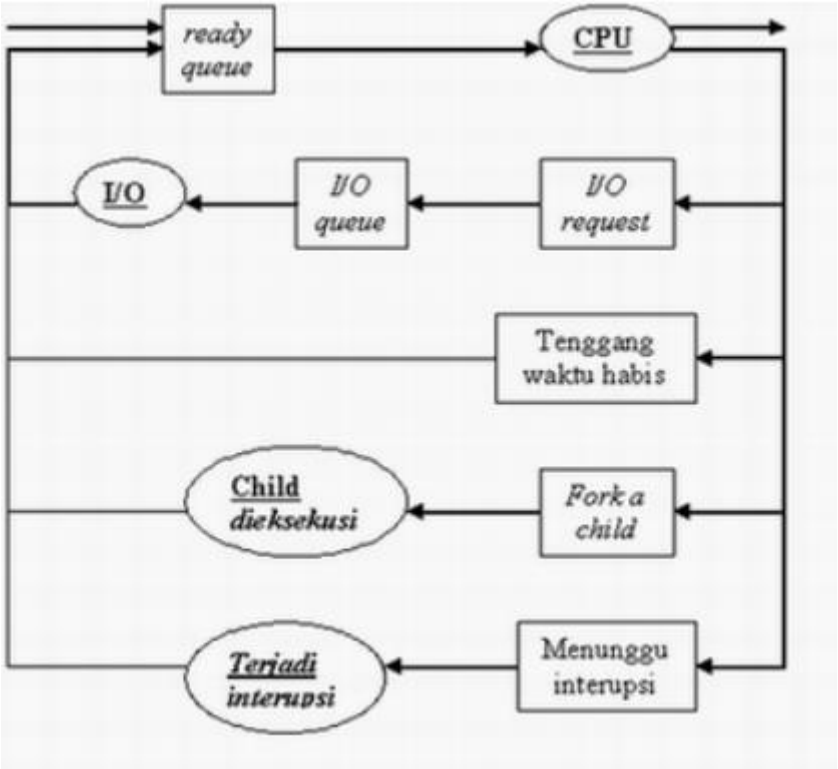
Sistem operasi juga memiliki antrian lain. Ketika sebuah proses dialokasikan ke CPU, proses tersebut berjalan sebentar lalu berhenti, di-interupsi, atau menunggu suatu hal tertentu, seperti selesainya suatu permintaan M/K. Dalam permintaan M/K, dapat saja yang diminta itu adalah *tape drive*, atau peralatan yang di-*share* secara bersama-sama, seperti disk. Karena ada banyak proses dalam sistem, disk dapat saja sibuk dengan permintaan M/K dari proses lainnya. Untuk itu proses tersebut mungkin harus menunggu disk tersebut. Daftar dari proses-proses yang menunggu peralatan M/K tertentu disebut dengan ***device queue***. Tiap peralatan memiliki *device queue*-nya masing-masing.



Gambar 1 Device antrian

Penjelasan Gambar 1

Penjadualan proses dapat direpresentasikan secara umum dalam bentuk diagram antrian, seperti yang ditunjukkan oleh Gambar 1. Setiap kotak segi empat menunjukkan sebuah antrian. Dua tipe antrian menunjukan antrian yang siap dan seperangkat *device queues*. Lingkaran menunjukkan sumber daya yang melayani antrian, dan tanda panah mengindikasikan alur dari proses-proses yang ada dalam sistem.



Gambar 2 Diagram antrian

Penjelasan Gambar 2;

Sebuah proses baru pertama-tama diletakkan dalam *ready queue*. Proses tersebut menunggu di dalam *ready* antrian sampai dia dipilih untuk eksekusi, atau dengan kata lain di-*dispatched*. Begitu proses tersebut dialokasikan ke CPU dan sedang berjalan, beberapa kemungkinan di bawah ini dapat terjadi:

a) Proses tersebut mengeluarkan permintaan M/K, lalu ditempatkan dalam sebuah antrian M/K.

b) Proses tersebut dapat membuat sub proses yang baru dan menunggu untuk di-terminasi.

c) Proses tersebut dapat dikeluarkan secara paksa dari CPU, sebagai hasil dari suatu interupsi, dan diletakkan kembali dalam *ready queue*.

Pada dua kemungkinan pertama (proses meminta M/K atau membuat sub proses baru), proses berganti keadaan dari *waiting state* menjadi *ready state*, lalu diletakkan kembali dalam *ready queue*. Proses akan meneruskan siklus ini sampai dia di terminasi, yaitu saat dimana proses tersebut dikeluarkan dari seluruh antrian yang ada dan memiliki PCB-nya sendiri dan seluruh sumber daya yang dia gunakan dialokasikan kembali.

4.3 "Scheduler"

Sebuah proses berpindah-pindah di antara berbagai penjadualan antrian seumur hidupnya. Sistem operasi harus memilih dan memproses antrian-antrian ini berdasarkan kategorinya dengan cara tertentu. Oleh karena itu, proses seleksi ini harus dilakukan oleh ***scheduler*** yang tepat. Dalam sistem *batch*, seringkali proses yang diserahkan lebih banyak daripada yang dapat dilaksanakan dengan segera..

Proses-proses ini disimpan pada suatu *mass-storage device* (disk), dimana proses tersebut disimpan untuk eksekusi di lain waktu. ***Long-term scheduler***, atau ***job scheduler***, memilih proses dari tempat ini dan mengisinya ke dalam memori.

Sedangkan ***short-term scheduler***, atau ***CPU scheduler***, hanya memilih proses yang sudah siap untuk melakukan eksekusi, dan mengalokasikan CPU untuk proses tersebut. Hal yang cukup jelas untuk membedakan kedua jenis *scheduler* ini adalah frekuensi dari eksekusinya. *Short-term scheduler* harus memilih proses baru untuk CPU sesering mungkin. Sebuah proses dapat mengeksekusi hanya dalam beberapa milidetik sebelum menunggu permintaan M/K. Seringkali, *short-term scheduler* mengeksekusi paling sedikit sekali setiap 100 milidetik. Karena durasi waktu yang pendek antara eksekusi-eksekusi tersebut, *short-term scheduler* seharusnya cepat.

Jika memerlukan waktu 10 mili detik untuk menentukan suatu proses eksekusi selama 100 mili detik, maka 10/(100 + 10) = 9 persen dari CPU sedang digunakan (atau terbuang) hanya untuk pekerjaan penjadualan. *Long-term scheduler*, pada sisi lain, mengeksekusi jauh lebih jarang. Mungkin ada beberapa menit waktu yang dibutuhkan untuk pembuatan proses baru dalam sistem. *Long-term scheduler* mengontrol *degree of multiprogramming* (jumlah proses dalam memori). Jika *degree of multiprogramming* stabil, maka tingkat rata-rata penciptaan proses harus sama dengan tingkat rata rata proses meninggalkan sistem. Maka dari itu *long-term scheduler* mungkin dipanggil hanya ketika suatu proses meninggalkan sistem. Karena interval yang lebih panjang antara eksekusi, *long-term scheduler* dapat menggunakan waktu yang lebih lama untuk menentukan proses mana yang harus dipilih untuk dieksekusi. Sangat penting bagi *long-term scheduler* membuat seleksi yang hati-hati. Secara umum, proses dapat dibedakan atas dua macam, yaitu proses ***I/O bound*** dan proses ***CPU bound***. Proses *I/O bound* adalah proses yang lebih banyak menghabiskan waktunya untuk mengerjakan M/K dari pada melakukan komputasi. Proses *CPU-bound*, di sisi lain, jarang melakukan permintaan M/K, dan menggunakan lebih banyak waktunya untuk melakukan komputasi. Oleh karena itu, *long-term scheduler* harus memilih gabungan proses yang baik antara proses *I/O bound* dan *CPU bound*. Jika seluruh proses adalah *I/O bound*, *ready queue* akan hampir selalu kosong, dan *short-term scheduler* akan memiliki sedikit tugas.

Jika seluruh proses adalah *CPU bound*, *I/O waiting queue* akan hampir selalu kosong, peralatan akan tidak terpakai, dan sistem akan menjadi tidak seimbang. Sistem dengan kinerja yang terbaik akan memiliki kombinasi yang baik antara proses *CPU bound* dan *I/O bound*. Pada sebagian sistem, *long-term scheduler* dapat jadi tidak ada atau kerjanya sangat minim. Sebagai contoh, sistem *time-*

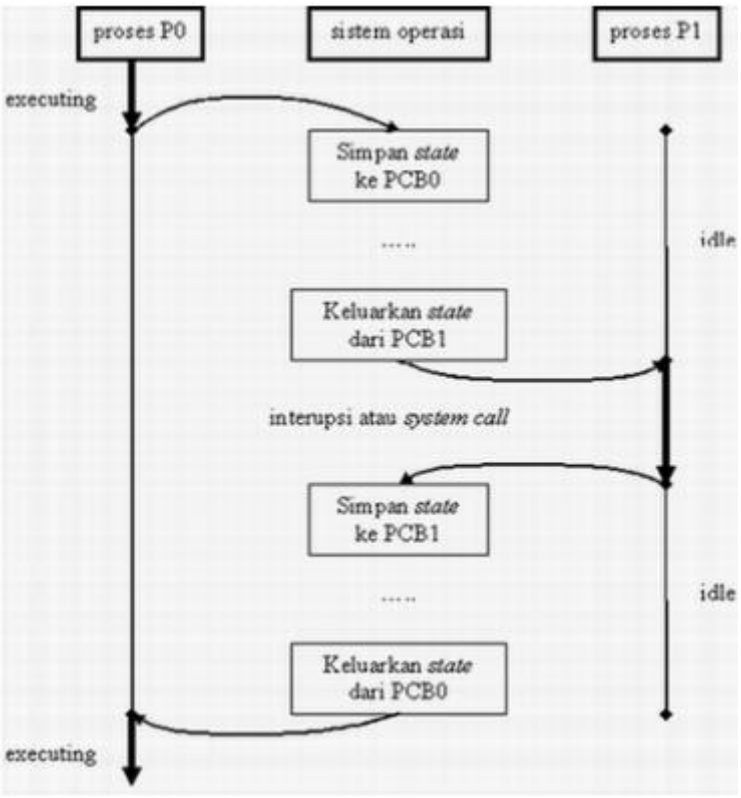
sharing seperti UNIX sering kali tidak memiliki *long-term scheduler*. Stabilitas sistem-sistem seperti ini bergantung pada keterbatasan fisik (seperti jumlah terminal yang ada) atau pada penyesuaian sendiri secara alamiah oleh manusia sebagai pengguna. Jika kinerja menurun pada tingkat yang tidak dapat diterima, sebagian pengguna akan berhenti. Sebagian sistem operasi, seperti sistem *time-sharing*, dapat memperkenalkan sebuah *scheduler* tambahan, yaitu ***medium-term scheduler***. *Scheduler* ini digambarkan pada Gambar 3. Ide utama atau kunci dari *scheduler* ini terkadang akan menguntungkan untuk memindahkan proses dari memori (dan dari pengisian aktif dari CPU), dan akibatnya *degree of multiprogramming* akan berkurang. Di kemudian waktu, proses dapat dibawa kembali dalam memori dan eksekusinya dapat dilanjutkan pada keadaan dimana proses itu dipindahkan tadi. Skema ini disebut *swapping*. Proses di-*swapped out* dan di-*swapped. in* oleh *scheduler* ini. *Swapping* mungkin diperlukan untuk meningkatkan mutu penggabungan proses, atau karena perubahan dalam kebutuhan memori yang mengakibatkan memori harus dibebaskan.

Gambar 3 *Medium-term Scheduler*

4.4 "Context Switch"

Mengganti CPU ke proses lain memerlukan penyimpanan keadaan dari proses lama dan mengambil keadaan dari proses yang baru. Hal ini dikenal dengan sebutan context switch. Context switch sebuah proses direpresentasikan dalam PCB dari suatu proses; termasuk nilai dari CPU register, status proses (dapat dilihat pada Gambar 4) dan informasi managemen memori. Ketika context switch terjadi, kernel menyimpan data dari proses lama ke dalam PCB nya dan mengambil data dari proses baru yang telah terjadual untuk berjalan.

Waktu context switch adalah murni overhead, karena sistem melakukan pekerjaan yang tidak begitu berarti selama melakukan pengalihan. Kecepatannya bervariasi dari mesin ke mesin, bergantung pada kecepatan memori, jumlah register yang harus di-copy, dan ada tidaknya instruksi khusus (seperti instruksi tunggal untuk mengisi atau menyimpan seluruh register). Tingkat kecepatan umumnya berkisar antara 1 sampai 1000 mikro detik.



Gambar 4 Context Switch

Penjelasan Gambar 4;

Waktu *context switch* sangat bergantung pada dukungan perangkat keras. Sebagai contoh, prosesor seperti UltraSPARC menyediakan beberapa set register. Sebuah proses *context switch* hanya memasukkan perubahan *pointer* ke set register yang ada saat itu. Tentu saja, jika proses aktif yang ada lebih banyak daripada proses yang ada pada set register, sistem menggunakan bantuan untuk meng-copy data register dari dan ke memori, sebagaimana sebelumnya. Semakin kompleks suatu sistem operasi, semakin banyak pekerjaan yang harus dilakukan selama *context switch*. Sebagai contoh, ruang alamat dari proses yang ada saat itu harus dijaga sebagai ruang alamat untuk proses yang akan dikerjakan berikutnya.

Bagaimana ruang alamat dijaga, berapa banyak pekerjaan dibutuhkan untuk menjaganya, tergantung pada metode managemen memori dari sistem operasi.

4.5 "Scheduling" CPU

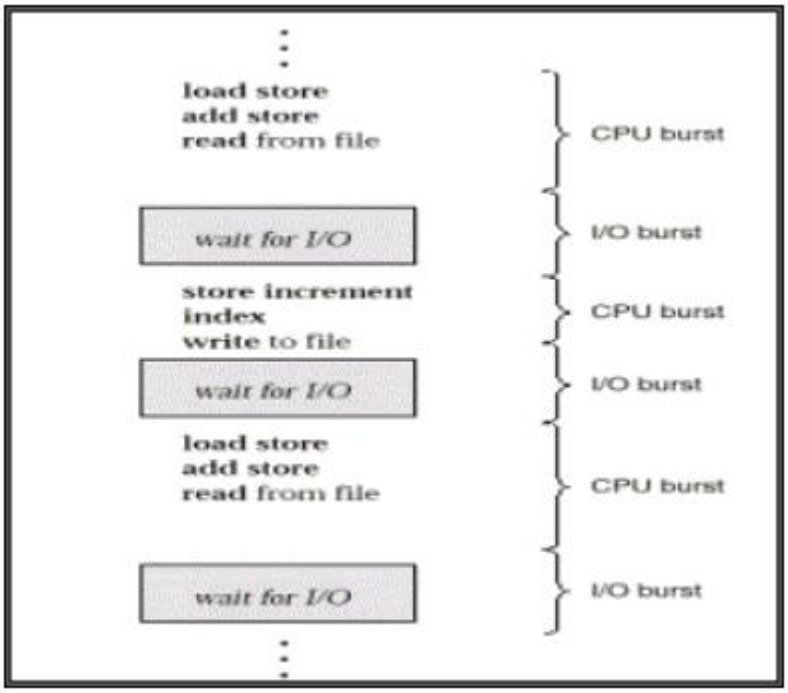
Penjadualan CPU adalah basis dari multi-programming sistem operasi. Dengan men-*switch* CPU diantara proses. Akibatnya sistem operasi dapat membuat komputer produktif, dengan konsep penjadualan dan beberapa algoritma penjadualan.

Tujuan dari multi-programming adalah untuk mempunyai proses berjalan secara bersamaan, untuk memaksimalkan kinerja dari CPU. Pada sistem prosesor tunggal, tidak pernah ada proses yang berjalan lebih dari satu. Bila ada proses yang lebih dari satu maka proses yang lain harus mengantri sampai CPU bebas proses.

Ide dari multi-progamming sangat sederhana. Ketika sebuah proses dieksekusi maka proses yang lain harus menunggu sampai proses pertama selesai. Pada sistem komputer yang sederhana CPU akan banyak dalam posisi idle. Sehingga waktu CPU ini sangat terbuang. Akan tetapi dengan multiprogamming, kita mencoba menggunakan waktu secara produktif. Beberapa proses di simpan di emori dalam satu waktu. Ketika suatu proses harus menuggu, Sistem operasi dapat saja akan menghentikan CPU dari suatu proses yang sedang diekseskusi dan memberikan sumberdaya kepada proses yang lainnya. Begitu seterusnya. Penjadualan adalah fungsi dasar dari suatu sistem operasi. Hampir semua sumber komputer dijadualkan sebelum digunakan. CPU salah satu sumber dari komputer yang penting yang menjadi sentral dari sentral penjadualan di sistem operasi.

4.5.1. Siklus *Burst* CPU-M/K

Keberhasilan dari penjadualan CPU tergantung dari beberapa properti prosesor. Pengeksekusian dari proses tersebut terdiri atas siklus CPU ekskusi dan M/K Wait. Proses hanya akan bolak-balik dari dua state ini. Pengeksekusian proses dimulai dengan CPU Burst, setelah itu diikuti oleh M/K burst, kemudian CPU Burst lagi lalu M/K Burst lagi begitu seterusnya dan dilakukan secara bergiliran. Dan, CPU Burst terakhir, akan berakhir dengan permintaan sistem untuk mengakhiri pengeksekusian daripada melalui M/K Burst lagi.



Gambar 5 Siklus *Burst*

4.5.2. Penjadualan CPU

Kapan pun CPU menjadi *idle*, sistem operasi harus memilih salah satu proses untuk masuk ke dalam antrian *ready* (siap) untuk dieksekusi. Pemilihan tersebut dilakukan oleh penjadual *short term*. Penjadualan memilih dari sekian proses yang ada di memori yang sudah siap dieksekusi, dengan mengalokasikan CPU untuk mengeksekusinya.

Penjadualan *Preemptive*

Penjadualan CPU mungkin akan dijalankan ketika proses:

1) Berubah dari running ke waiting state
2) Berubah dari running ke ready state
3) Berubah dari waiting ke ready
4) Terminates

Penjadualan dari no 1 sampai 4 non premptive sedangkan yang lain premptive. Dalam penjadualan nonpreemptive sekali CPU telah dialokasikan untuk sebuah proses, maka tidak dapat di ganggu, penjadualan model seperti ini digunakan oleh windows 3.X; windows 95 telah menggunakan penjadualan preemptive.

Penjadualan *Non-Preemptive*

Penjadualan non-preemptive terjadi ketika proses hanya:

1) berjalan dari running state sampai waiting state
2) dihentikan

ini berarti CPU menjaga proses sampai proses itu pindah ke waiting state ataupun dihentikan (proses tidak diinterupt) metode ini digunakan oleh Microsoft Windows 3.1 dan Macintosh. Ini adalah metode yang dapat digunakan untuk platforms hardware tertentu, karena tidak memerlukan perangkat keras khusus (misalnya timer yang digunakan untuk menginterupt pada metode penjadwalan preemptive)

Dispatcher

Komponen yang lain yang terlibat dalam penjadualan CPU adalan *dispatcher*. *Dispatcher* adalah modul yang memberikan kontrol CPU kepada proses yang fungsinya adalah:

1) *Switching context*
2) *Switching to user mode*
3) Lompat dari suatu bagian di progam user untuk mengulang progam.

4.6 Kriteria "Scheduling"

Algoritma penjadualan CPU yang berbeda mempunyai *property* yang berbeda. Dalam memilih algoritmayang digunakan untuk situasi tertentu, kita harus memikirkan properti yang berbeda untuk algoritma yang berbeda. Banyak kriteria yang dianjurkan untuk membandingkan penjadualan CPU algoritma. Kritria yang biasanya digunakan dalam memilih adalah: Kritria yang biasanya digunakan dalam memilih adalah:

1) *CPU utilization*: kita ingin menjaga CPU sesibuk mungkin. CPU utilization akan mempunyai range dari 0 ke 100 persen. Di sistem yang sebenarnya seharusnya ia mempunyai range dari 40 persen samapi 90 persen
2) *Throughput*: jika CPU sibuk mengeksekusi proses, jika begitu kerja telah ilaksanakan. Salah satu ukuran kerja adalah banyak proses yang diselesaikan per unit waktu, disebut througput. Untuk proses yang lama

mungkin satu proses per jam; untuk proses yang sebentar mungkin 10 proses perdetik.

3) *Turnaround time*: dari sudur pandang proses tertentu, kriteria yang penting adalah berapa lama untuk mengeksekusi proses tersebut. Interval dari waktu yang dijinkan dengan waktu yang dibutuhkan untuk menyelesaikan sebuah prose disebut *turn around time*. *Turn around time* adalah jumlah periode untuk menunggu untuk dapat ke memori, menunggu di ready queue, eksekusi di CPU, dan melakukan M/K

4) *Waiting time*: algoritma penjadualan CPU tidak mempengaruhi waktu untuk elaksanakan proses tersebut atau M/K; itu hanya mempengaruhi jumlah waktu yang dibutuhkan proses di antrian ready. Waiting time adalah jumlah periode menghabiskan di antrian ready.

5) *Response time*: di sistem yang interaktif, *turnaround time* mungkin bukan waktu yang terbaik untuk kriteria. Sering sebuah proses dapat memproduksi *output* diawal, dan dapat meneruskan hasil yang baru sementara hasil yang sebelumnya telah diberikan ke user. Ukuran yang lain adalah waktu dari pengiriamn permintaan sampai respon yang pertama di berikan. Ini disebut response time, yaitu waktu untuk memulai memberikan respon, tetapi bukan waktu yang dipakai *output* untu respon tersebut

4.7 Algoritma “Scheduling”

Proses yang belum mendapat jatah alokasi dari CPU akan mengantri di ready queue. Di sini algoritma diperlukan untuk mengatur giliran proses-proses tersebut. Berikut ini adalah algoritmanya.

1) First-Come, First-Served

2) Shortest-Job First

3) Priority

4) Round-Robin

5) Multilevel Queue

6) *Multilevel Feedback Queue*

Catatan tambahan ;

Penjadualan CPU adalah pemilihan proses dari antrian ready untuk dapat dieksekusi. Algoritma yang digunakan dalam penjadulan CPU ada bermacam-macam. Diantaranya adalah *First Come First Serve* (FCFS), merupakan algoritma sederhana dimana proses yang datang duluan maka dia yang dieksekusi pertama

kalinya. Algoritma lainnya adalah *Shorthest Job First* (SJF), yaitu penjadulan CPU dimana proses yang paling pendek dieksekusi terlebih dahulu.

Kelemahan algoritma SJF adalah tidak dapat menghindari starvation. Untuk itu diciptakan algoritma Round Robin (RR). Penjadulan CPU dengan Round Robin adalah membagi proses berdasarkan waktu tertentu yaitu waktu quantum q. Setelah proses menjalankan eksekusi selama q satuan waktu maka akan digantikan oleh proses yang lain. **Permasalahannya** adalah bila waktu quantumnya besar sedang proses hanya membutuhkan waktu sedikit maka akan membuang waktu. Sedang bila waktu quantum kecil maka akan memakan waktu saat context-switch. Penjadualan FCFS adalah nonpreemptive yaitu tidak dapat diinterupsi sebelum proses dieksekusi seluruhnya. Penjadualan RR adalah preemtive yaitu dapat dieksekusi saat prosesnya masih dieksekusi. Sedangkan penjadualan SJF dapat berupa nonpreemtive dan preemtive.

Latihan Soal

Tema sub-materi status proses

a) Gambarkan sebuah model bagan status proses (process state diagram) dengan minimum lima (5) status.

b) Sebutkan serta terangkan semua nama status proses (process states) tersebut.

c) Sebutkan serta terangkan semua nama kejadian (event) yang menyebabkan perubahan status proses.

d) Terangkan perbedaan antara proses ”I/O Bound” dengan proses ”CPU Bound” berdasarkan bagan status proses tersebut.

e) Apa yang dimaksud dengan marshaling, jelaskan kegunaanya!

Tema sub-materi scheduling

1. Jelaskan perbedaan algoritma penjadualan berikut:

a) FCFS

b) Round Robin

c) Antrian Multilevel feedback

2. Tentukan perbedaan antara penjadualan preemptive dan nonpreemptive (cooperative). Nyatakan dan buatlah uraian mengapa nonpreemptive scheduling tidak dapat digunakan pada suatu komputer center. Di sistem komputer nonpreemptive, penjadualan yang lebih baik digunakan.

3. Gambarkan 4 diagram Chart yang mengilustrasikan eksekusi dari proses-proses tersebut menggunakan FCFS, SJF, prioritas nonpreemptive dan round robin.

Daftar Pustaka

Avi Silberschatz, Peter Galvin, Greg Gagne. Applied Operationg System Concepts 1st Ed. 2000. John Wiley & Sons, Inc.

William Stallings: Operating Systems -- Fourth Edition, Prentice Hall, 2001