Buku Ajar
SISTEM OPERASI

Uwais Inspirasi Indonesia

Perpustakaan Nasional RI

Ronal Watrianthos dan Iwan Purnama -Uwais Inspirasi Indonesia-Agustus 2018-vi + 79
Judul: Buku Ajar Sistem Operasi
ISBN: 978-602-5891-24-3

Penulis: Ronal Watrianthos dan Iwan Purnama
Editor: Sudi Suryadi
Design Cover: Uwais Inspirasi Indonesia
Cetakan Pertama, Agustus 2018

Diterbitkan Oleh:
Uwais Inspirasi Indonesia
Ds. Sidoarjo, Kec. Pulung, Kab. Ponorogo
Email: Penerbituwais@gmail.com
Telp: 0352-571 892
WA: 0895-2366-1093/0823-3033-5859

Buku Ajar
SISTEM OPERASI

Ronal Watrianthos

Iwan Purnama

Uwais Inspirasi Indonesia

KATA PENGANTAR

Bismillahirrahmaanirrahiim,

Puji dan syukur penulis panjatkan ke hadirat Allah SWT karena berkat Rahmat dan Hidayah-Nya, serta segala kemudahan yang telah diberikan, penulis dapat menyelesaikan pembuatan Buku Ajar dengan judul "Sistem Operasi".

Adapun tujuan pembuatan Buku Ajar Sistem Operasi ini adalah membantu para mahasiswa untuk mencari referensi yang tepat dalam mempelajari mata kuliah Sistem Operasi. Buku ajar ini terdiri dari 9 Bab yang disusun dengan bahasa yang mudah dipahami oleh para mahasiswa. Setiap babnya akan dijelaskan secara terperinci sebagai penunjang materi yang diajarkan di kelas.

Penulis menyadari bahwa buku ini masih banyak kekurangan, karena itu penulis mengharapkan masukan yang positif agar di kemudian hari dapat memperbaiki kekurangan buku ini.

Akhir kata hanya ucapan terima kasih dan doa yang bisa penulis sampaikan kepada semua pihak yang telah membantu dalam penyusunan buku ini. Semoga buku ini dapat memberikan manfaat bagi para pembacanya.

Rantauprapat, Juli 201

Tim Penulis

DAFTAR ISI

BAGIAN I

I. PENGENALAN SISTEM OPERASI

Gambar 1.1 Sistem Operasi

Sistem operasi merupakan sebuah penghubung antara pengguna dari komputer dengan perangkat keras komputer. Sebelum ada sistem operasi, orang hanya mengunakan komputer dengan menggunakan sinyal analog dan sinyal digital.

Seiring dengan berkembangnya pengetahuan dan teknologi, pada saat ini terdapat berbagai sistem operasi dengan keunggulan masing-masing. Untuk lebih memahami sistem operasi maka sebaiknya perlu diketahui terlebih dahulu beberapa konsep dasar mengenai sistem operasi itu sendiri.

A. Definisi Sistem Operasi

Sistem komputer pada dasarnya terdiri dari empat komponen utama, yaitu perangkat keras, program aplikasi, system operasi, dan para pengguna. Sistem operasi berfungsi untuk mengatur dan mengawasi penggunaan perangkat keras oleh berbagai program aplikasi serta para pengguna.

Sistem operasi berfungsi ibarat pemerintah dalam suatu negara, dalam arti membuat kondisi komputer agar dapat

menjalankan program secara benar. Untuk menghindari konflik yang terjadi pada saat pengguna menggunakan sumber daya yang sama, sistem operasi mengatur pengguna mana yang dapat mengakses suatu sumber daya.

Sistem operasi juga sering disebut *resource allocator*. Satu lagi fungsi penting sistem operasi adalah sebagai program pengendali yang bertujuan untuk menghindari kekeliruan (error) dari penggunaan komputer yang tidak perlu.

B. Tujuan Mempelajari Sistem Operasi

Mempelajari Sistem Operasi bertujuan agar nantinya mengenal dan mampu merancang sendiri serta dapat memodifikasi sistem operasi yang telah ada sesuai dengan kebutuhan kita. Juga agar bisa memilih alternatif sistem operasi, memaksimalkan penggunaan sistem operasi, sehingga konsep dan teknik sistem operasi dapat diterapkan pada aplikasi-aplikasi lain.

C. Sasaran Sistem Operasi

Sistem operasi mempunyai tiga sasaran utama yaitu:

1. **Kenyamanan**, yaitu membuat penggunaan komputer menjadi lebih nyaman.
2. **Efisien**, yaitu agar penggunaan sumber daya dalam sistem komputer dapat digunakan secara efisien.
3. **Mampu berevolusi**, yaitu dalam membangun sistem operasi dimungkinkan untuk lebih mudah dalam pengembangan, pengujian, serta penggunaan sistem-sistem yang baru.

D. Sejarah Sistem Operasi

Menurut Tanenbaum, sistem operasi mengalami perkembangan yang sangat pesat, yang dapat dibagi ke dalam empat generasi:

I. Generasi Pertama (1945-1955)

Generasi pertama merupakan awal perkembangan sistem komputasi elektronik sebagai pengganti sistem komputasi mekanik, hal itu disebabkan kecepatan manusia untuk menghitung terbatas dan manusia sangat mudah untuk membuat kecerobohan, kekeliruan bahkan kesalahan.

Pada generasi ini belum ada sistem operasi, maka sistem komputer diberi instruksi yang harus dikerjakan secara langsung.

II. Generasi Kedua (1955-1965)

Generasi kedua memperkenalkan *Batch Processing System*, yaitu *Job* yang dikerjakan dalam satu rangkaian, lalu dieksekusi secara berurutan. Pada generasi ini sistem komputer belum dilengkapi sistem operasi, tetapi beberapa fungsi sistem operasi telah ada, contohnya fungsi sistem operasi ialah FMS dan IBSYS.

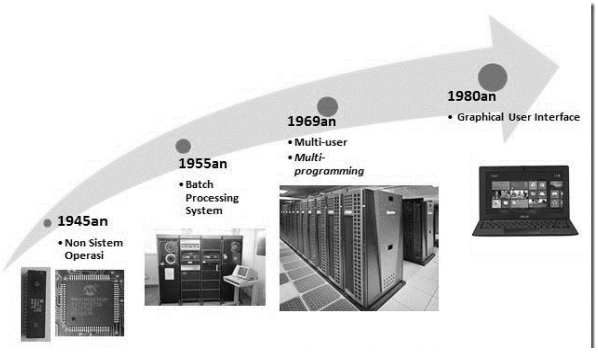
III. Generasi Ketiga (1965-1980)

Pada generasi ini perkembangan sistem operasi dikembangkan untuk melayani banyak pemakai sekaligus, di mana para pemakai interaktif berkomunikasi lewat terminal secara online ke komputer, maka sistem operasi menjadi multi-user (digunakan banyak pengguna sekaligus) dan multi-programming (melayani banyak program sekaligus).

IV. Generasi Keempat (Pasca 1980an)

Di generasi keempat, sistem operasi sudah dipergunakan untuk jaringan komputer dimana pemakai menyadari keberadaan komputer-komputer yang saling terhubung satu sama lainnya.

Pada masa ini para pengguna juga telah dinyamankan dengan *Graphical User Interface*(GUI) yaitu antar-muka komputer yang berbasis grafis yang sangat nyaman, juga dimulai era komputasi tersebar dimana komputasi-komputasi tidak lagi berpusat di satu titik tetapi dipecah dibanyak komputer sehingga tercapai kinerja yang lebih baik.



Gambar 1.2 Sejarah Sistem Operasi

E. Layanan Sistem Operasi

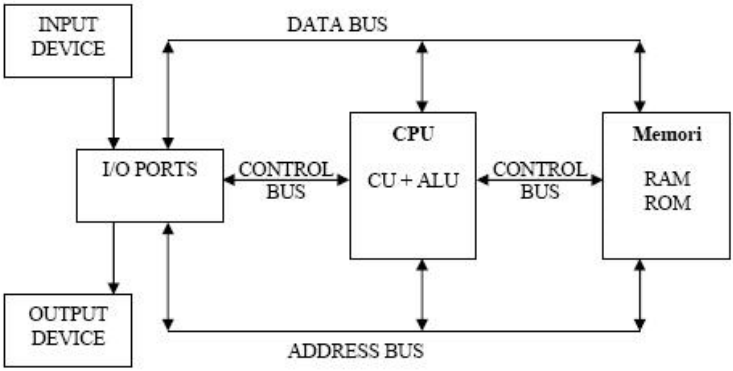
Sebuah sistem operasi yang baik menurut Tanenbaum, harus memiliki layanan sebagai seperti:

1. **Pembuatan program** yaitu sistem operasi menyediakan fasilitas dan layanan untuk membantu para pemrogram untuk menulis program.
2. **Eksekusi Program** yang berarti Instruksi-instruksi dan data-data harus dimuat ke memori utama, perangkat-parangkat masukan/keluaran dan berkas harus diinisialisasi, serta sumber-daya yang ada harus disiapkan, semua itu harus ditangani oleh sistem operasi
3. **Pengaksesan I/O Device**, artinya Sistem Operasi harus mengambil alih sejumlah instruksi yang rumit dan sinyal kendali agar pemrogram dapat berfikir sederhana dan perangkat pun dapat beroperasi.

4. **Pengaksesan terkendali terhadap berkas**, yang artinya disediakannya mekanisme proteksi terhadap berkas untuk mengendalikan pengaksesan terhadap berkas.
5. **Pengaksesan sistem**, artinya pada pengaksesan digunakan bersama (*shared system*).
6. **Fungsi pengaksesan**, yaitu menyediakan proteksi terhadap sejumlah sumber-daya dan data dari pemakai serta menyelesaikan konflik-konflik dalam perebutan sumber-daya.
7. **Deteksi dan pemberian tanggapan pada kesalahan**, yaitu jika muncul permasalahan muncul pada sistem komputer maka sistem operasi harus memberikan tanggapan yang menjelaskan kesalahan yang terjadi serta dampaknya terhadap aplikasi yang sedang berjalan.
8. **Akunting**, yang artinya sistem operasi harus bisa mengumpulkan data statistik penggunaan beragam sumber-daya dan memonitor parameter kinerjanya.

F. STRUKTUR KOMPUTER

Struktur komputer didefinisikan sebagai cara-cara dari setiap komponen yang saling terkait. Struktur sebuah komputer secara sederhana, dapat digambarkan dalam diagram blok pada gambar dibawah.



Gambar 1.3 Struktur Komputer

Struktur sebuah sistem komputer dapat dibagi menjadi:

1. **Sistem Operasi Komputer**

   Dewasa ini sistem komputer multiguna terdiri dari CPU (Central Processing Unit), serta sejumlah *device controller* yang dihubungkan melalui bus yang menyediakan akses ke memori. Setiap *device controller* bertugas mengatur perangkat yang tertentu (contohnya disk drive, audio device, dan video display).
   CPU dan *device controller* dapat dijalankan secara bersamaan, namun demikian diperlukan mekanisme sinkronisasi untuk mengatur akses ke memori.

   Pada saat pertama kali dijalankan atau pada saat *boot*, terdapat sebuah program awal yang mesti dijalankan. Program awal ini disebut program *bootstrap*. Program ini berisi semua aspek dari sistem komputer, mulai dari register CPU, device controller, sampai isi memori.

   Interupsi merupakan bagian penting dari sistem arsitektur komputer. Setiap sistem komputer memiliki mekanisme yang berbeda. Interupsi bisa terjadi apabila perangkat keras (hardware) atau perangkat lunak (software) minta "dilayani" oleh prosesor. Apabila terjadi interupsi maka prosesor menghentikan proses yang sedang dikerjakannya, kemudian beralih mengerjakan *service routine* untuk melayani interupsi tersebut. Setelah selesai mengerjakan *service routine* maka prosesor kembali melanjutkan proses yang tertunda.

2. **Struktur I/O**

   Bagian ini akan membahas struktur I/O, interupsi I/O, dan DMA, serta perbedaan dalam penanganan interupsi.

a. Interupsi I/O

Untuk memulai operasi I/O, CPU me-load register yang bersesuaian ke *device controller*. Sebaliknya device controller memeriksa isi register untuk kemudian menentukan operasi apa yang harus dilakukan.

Pada saat operasi I/O dijalankan ada dua kemungkinan, yaitu *synchronous I/O* dan *asynchronous I/O*. Pada synchronous I/O, kendali dikembalikan ke proses pengguna setelah proses I/O selesai dikerjakan.

Sedangkan pada asynchronous I/O, kendali dikembalikan ke proses pengguna tanpa menunggu proses I/O selesai. Sehingga proses I/O dan proses pengguna dapat dijalankan secara bersamaan.

b. Struktur DMA

Direct Memory Access (DMA) suatu metoda penanganan I/O dimana device controller langsung berhubungan dengan memori tanpa campur tangan CPU.

Setelah men-set buffers, pointers, dan counters untuk perangkat I/O, device controller mentransfer blok data langsung ke penyimpanan tanpa campur tangan CPU. DMA digunakan untuk perangkat I/O dengan kecepatan tinggi.

Hanya terdapat satu interupsi setiap blok, berbeda dengan perangkat yang mempunyai kecepatan rendah dimana interupsi terjadi untuk setiap byte (word).

3. Struktur Penyimpanan

Program komputer harus berada di memori utama (biasanya RAM) untuk dapat dijalankan. Memori utama adalah satu-satunya tempat penyimpanan yang dapat diakses secara langsung oleh prosesor.

Idealnya program dan data secara keseluruhan dapat disimpan dalam memori utama secara permanen. Namun demikian hal ini tidak mungkin karena ukuran memori utama relatif kecil untuk dapat menyimpan data dan program secara keseluruhan.

Memori utama juga bersifat *volatile*, sehingga tidak bisa menyimpan secara permanen dan apabila komputer dimatikan maka data yang tersimpan di memori utama akan hilang.

a. Memori Utama

Hanya memori utama dan register merupakan tempat penyimpanan yang dapat diakses secara langsung oleh prosesor. Oleh karena itu instruksi dan data yang akan dieksekusi harus disimpan di memori utama atau register.

Untuk mempermudah akses perangkat I/O ke memori, pada arsitektur komputer menyediakan fasilitas pemetaan memori ke I/O. Dalam hal ini sejumlah alamat di memori dipetakan dengan device register. Membaca dan menulis pada alamat memori ini menyebabkan data ditransfer dari dan ke device register. Metode ini cocok untuk perangkat dengan waktu respon yang cepat seperti video controller.

Register yang terdapat dalam prosesor dapat diakses dalam waktu 1 clock cycle. Hal ini

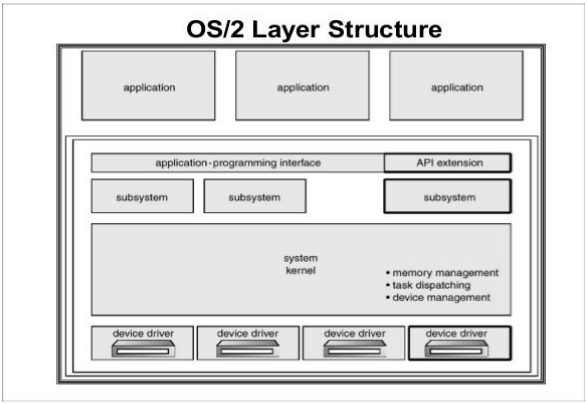
menyebabkan register merupakan media penyimpanan dengan akses paling cepat bandingkan dengan memori utama yang membutuhkan waktu relatif lama. Untuk mengatasi perbedaan kecepatan, dibuatlah suatu penyangga (buffer) penyimpanan yang disebut cache.

b. Magnetic Disk

Magnetic Disk berperan sebagai secondary storage pada sistem komputer modern. Magnetic Disk disusun dari piringan-piringan seperti CD. Kedua permukaan piringan diselimuti oleh bahan-bahan magnetik. Permukaan dari piringan dibagi-bagi menjadi track yang memutar, yang kemudian dibagi lagi menjadi beberapa sektor.

BAGIAN II

II. STRUKTUR SISTEM OPERASI



Gambar 2.1 Struktur Sistem Operasi

Sistem komputer modern yang semakin kompleks dan rumit memerlukan sistem operasi yang dirancang dengan sangat hati-hati agar dapat berfungsi secara optimum dan mudah untuk dimodifikasi. Pada kenyataannya tidak semua sistem operasi mempunyai struktur yang sama. Namun menurut Avi Silberschatz, Peter Galvin, dan Greg Gagne, umumnya sebuah sistem operasi modern mempunyai komponen sebagai berikut:

A. Manajemen Proses

Proses adalah keadaan ketika sebuah program sedang dieksekusi. Sebuah proses membutuhkan beberapa sumber daya untuk menyelesaikan tugasnya. Sumber daya tersebut dapat berupa CPU time, memori, berkas-berkas, dan perangkat-perangkat I/O.

Sistem operasi bertanggung jawab atas aktivitas-aktivitas yang berkaitan dengan managemen proses, seperti:

1. Pembuatan dan penghapusan proses pengguna dan sistem proses.
2. Menunda atau melanjutkan proses.
3. Menyediakan mekanisme untuk proses sinkronisasi.
4. Menyediakan mekanisme untuk proses komunikasi.
5. Menyediakan mekanisme untuk penanganan deadlock.

B. Manajemen Memori Utama

Memori utama atau lebih dikenal sebagai memori adalah sebuah *array* yang besar dari word atau byte, yang ukurannya mencapai ratusan, ribuan, atau bahkan jutaan. Setiap word atau byte mempunyai alamat tersendiri.

Memori Utama berfungsi sebagai tempat penyimpanan yang akses datanya digunakan oleh CPU atau perangkat I/O. Memori utama termasuk tempat penyimpanan data yang sementara (volatile), artinya data dapat hilang begitu sistem dimatikan.

Sistem operasi bertanggung jawab atas aktivitas-aktivitas yang berkaitan dengan managemen memori seperti:

1. Menjaga *track* dari memori yang sedang digunakan dan siapa yang menggunakannya.
2. Memilih program yang akan di-load ke memori.
3. Mengalokasikan dan meng-dealokasikan ruang memori sesuai kebutuhan.

C. Manajemen Secondary Storage

Data yang disimpan dalam memori utama bersifat sementara dan jumlahnya sangat kecil. Oleh karena itu, untuk meyimpan keseluruhan data dan program komputer dibutuhkan *secondary-storage* yang bersifat permanen dan mampu menampung banyak data.

Contoh dari *secondary-storage* adalah harddisk, disket, dll. Sistem operasi bertanggung-jawab atas aktivitas-aktivitas yang berkaitan dengan disk-management seperti: free-space management, alokasi penyimpanan, penjadualan disk.

D. Manajemen Sistem I/O

Sering disebut *device manager*. Menyediakan "device driver" yang umum sehingga operasi I/O dapat seragam (membuka, membaca, menulis, menutup).

Contoh: pengguna menggunakan operasi yang sama untuk membaca berkas pada hard-disk, CD-ROM, dan floppy disk. Komponen Sistem Operasi untuk sistem I/O:

1. *Buffer*: menampung sementara data dari/ ke perangkat I/O.
2. *Spooling*: melakukan penjadualan pemakaian I/O sistem supaya lebih efisien (antrian dsb.).
3. Menyediakan driver untuk dapat melakukan operasi "rinci" untuk perangkat keras I/O tertentu.

E. Manajemen Berkas

Berkas adalah kumpulan informasi yang berhubungan sesuai dengan tujuan pembuat berkas tersebut. Berkas dapat mempunyai struktur yang bersifat hirarkis (direktori, volume, dll.). Sistem operasi bertanggung-jawab:

1. Pembuatan dan penghapusan berkas.
2. Pembuatan dan penghapusan direktori.
3. Mendukung manipulasi berkas dan direktori.
4. Memetakan berkas ke secondary storage.
5. Mem-backup berkas ke media penyimpanan yang permanen (*non-volatile*).

F. Sistem Proteksi

Proteksi mengacu pada mekanisme untuk mengontrol akses yang dilakukan oleh program, prosesor, atau

pengguna ke sistem sumber daya. Mekanisme proteksi harus:

1. Bisa membedakan antara penggunaan yang sudah diberi izin dan yang belum.
2. Bisa menentukan pengontrolan yang akan dilakukan terhadap sumber daya.
3. Bisa memberikan *punishment* terhadap kesalahan yang terjadi.

G. Jaringan

Sistem terdistribusi adalah sekumpulan prosesor yang tidak berbagi memori atau *clock*. Tiap prosesor mempunyai memori sendiri. Prosesor-prosesor tersebut terhubung melalui jaringan komunikasi. Sistem terdistribusi menyediakan akses pengguna ke bermacam sumber-daya sistem. Akses tersebut dapat menyebabkan:

1. Meningkatnya proses komputasi.
2. Meningkatnya ketersediaan data.
3. Meningkatnya kehandalan.

H. Command Interpreter System

Sistem Operasi menunggu instruksi dari pengguna (*command driven*). Program yang membaca instruksi dan mengartikan *control statements* umumnya disebut: *control-card interpreter*, *command-line interpreter*, dan *UNIX shell*.

Command-Interpreter System sangat bervariasi dari satu sistem operasi ke sistem operasi yang lain dan disesuaikan dengan tujuan dan teknologi I/O devices yang ada. Contohnya: CLI, Windows, Pen-based (touch), dan lain-lain.

I. Layanan Sistem Operasi

Sistem operasi memberikan layanan sebagai berikut:

1. Eksekusi program, yaitu kemampuan sistem untuk "load" program ke memori dan menjalankan program.
2. Operasi I/O, yaitu sistem operasi harus menyediakan mekanisme untuk melakukan operasi I/O atas nama pengguna.
3. Sistem manipulasi, berkas yaitu kemampuan program untuk operasi pada berkas (membaca, menulis, membuat, and menghapus berkas).
4. Komunikasi, yaitu pertukaran data/informasi antar dua atau lebih proses yang berada pada satu komputer (atau lebih).
5. Deteksi error yaitu menjaga kestabilan sistem dengan mendeteksi "error" pada perangkat keras mau pun operasi.

Dalam efesiensi penggunaan sistem, sistem operasi memberikan pelayanan:

1. *Resource allocator*, yaitu SO mengalokasikan sumber daya ke beberapa pengguna atau job yang jalan pada saat yang bersamaan.
2. Proteksi yaitu SO menjamin akses ke sistem sumber daya dikendalikan (pengguna dikontrol aksesnya ke sistem).
3. *Accounting* yaitu SO merekam kegiatan pengguna dan jatah pemakaian sumber daya (berdasarkan prinsip keadilan atau kebijaksanaan).

J. System Calls

System call menyediakan *interface* antara program (program pengguna yang berjalan) dan bagian OS. System call menjadi jembatan antara proses dan sistem operasi. System call ditulis dalam bahasa assembly atau bahasa tingkat tinggi yang dapat mengendalikan mesin (Bahasa C).

Contoh: UNIX menyediakan system call: read, write => operasi I/O untuk berkas. Sering pengguna program harus memberikan data (parameter) ke OS yang akan dipanggil.
Contoh padaUNIX: *read(buffer, max_size, file_id);*

K. Mesin Virtual

Mesin Virtual/Virtual Machine (VM) adalah sebuah mesin yang mempunyai dasar logika yang menggunakan pendekatan lapisan-lapisan (*layers*) dari sistem komputer. Sehingga sistem komputer dengan tersendiri dibangun atas lapisan-lapisan tersebut, dengan urutan lapisannya mulai dari lapisan terendah sampai lapisan teratas adalah sebagai berikut:

1. Perangkat keras (semua bagian fisik komputer)
2. *Kernel* (program untuk mengontrol disk dan sistem file, *multi-tasking*, *load-balancing*, *networking* dan *security*)
3. Sistem program (program yang membantu general user)

Kernel yang berada pada lapisan kedua ini, menggunakan instruksi perangkat keras untuk menciptakan seperangkat *system call* yang dapat digunakan oleh komponen-komponen pada level sistem program.

Sistem program kemudian dapat menggunakan *system call* dan perangkat keras lainnya seolah-olah pada level yang sama. Meskipun sistem program berada di level tertinggi, namun program aplikasi bisa melihat segala sesuatu pada tingkatan di bawahnya seakan-akan mereka adalah bagian dari mesin. Pendekatan dengan lapisan-lapisan inilah yang kemudian menjadi kesimpulan logis pada konsep Virtual Machine.

L. Perancangan Sistem dan Implementasi

Target untuk pengguna sebuah sistem operasi harus nyaman digunakan, mudah dipelajari, dapat diandalkan, aman dan cepat. Sedangkan target untuk sistem, sebuah sistem operasi harus gampang dirancang, diimplementasi, dan dipelihara, sebagaimana fleksibel, error, dan efisien.

Sebuah Sistem Operasi mempunyai mekanisme dan kebijaksanaan:

1. Mekanisme menjelaskan bagaimana melakukan sesuatu kebijaksanaan memutuskan apa yang akan dilakukan. Pemisahan kebijaksanaan dari mekanisme merupakan hal yang sangat penting, ini mengizinkan fleksibilitas yang tinggi bila kebijaksanaan akan diubah nanti.
2. Kebijaksanaan memutuskan apa yang akan dilakukan. Pemisahan kebijaksanaan dari mekanisme merupakan hal yang sangat penting, ini mengizinkan fleksibilitas yang tinggi bila kebijaksanaan akan diubah nanti.

Implementasi Sistem biasanya menggunakan bahasa *assembly*, sedangkan sistem operasi sekarang dapat ditulis dengan menggunakan bahasa tingkat tinggi. Kode yang ditulis dalam bahasa tingkat tinggi dapat dibuat dengan cepat, lebih ringkas, lebih mudah dimengerti dan di-*debug*. Sistem operasi lebih mudah dipindahkan ke perangkat keras yang lain bila ditulis dengan bahasa tingkat tinggi.

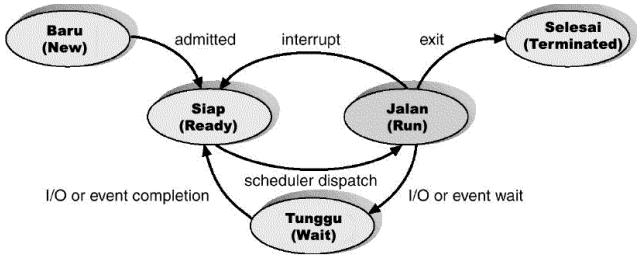
M. System Generation

Sistem operasi dirancang untuk dapat dijalankan diberbagai jenis mesin yang mana setiap sistemnya harus dikonfigurasi untuk tiap komputer. Program SYSGEN mendapatkan informasi mengenai konfigurasi khusus dari sistem perangkat keras, seperti:

1. *Booting*, yaitu memulai komputer dengan me-*load* kernel.
2. *Bootstrap program*, yaitu kode yang disimpan di code ROM yang dapat menempatkan kernel, memasukkannya kedalam memori, dan memulai eksekusinya.

BAGIAN III

III. Proses



Gambar 3.1 Status Proses

Satu selingan pada diskusi kita mengenai sistem operasi yaitu bahwa ada sebuah pertanyaan mengenai apa untuk menyebut semua aktivitas CPU. *Sistem batch* mengeksekusi *jobs*, sebagaimana suatu sistem *time-shared* telah menggunakan program pengguna, atau tugas-tugas/ pekerjaan-pekerjaan.

Pada sistem tunggal, seperti Microsoft Windows dan Macintosh OS, seorang pengguna mampu untuk menjalankan beberapa program pada saat yang sama: sebuah Word Processor, Web Browser, dan paket e-mail.

Bahkan jika pengguna dapat melakukan hanya satu program pada satu waktu, sistem operasi perlu untuk mendukung aktivitas program internalnya sendiri, seperti managemen memori. Dalam banyak hal, seluruh aktivitas ini adalah serupa, maka kita menyebut seluruh program itu proses-proses (processes).

A. Definisi Proses

Secara informal, proses adalah program dalam eksekusi. Suatu proses adalah lebih dari kode program, dimana kadang kala dikenal sebagai bagian tulisan. Proses juga termasuk aktivitas yang sedang terjadi, sebagaimana

digambarkan oleh nilai pada program counter dan isi dari daftar prosesor/*processor's register*. Suatu proses umumnya juga termasuk *process stack*, yang berisikan *data temporer* (seperti parameter metoda, address yang kembali, dan variabel lokal) dan sebuah data section, yang berisikan variabel global.

Program itu sendiri bukanlah sebuah proses, suatu program adalah satu entitas pasif seperti isi dari sebuah berkas yang disimpan didalam disket, sebagaimana sebuah proses dalam suatu entitas aktif dengan sebuah program counter yang mengkhususkan pada instruksi selanjutnya untuk dijalankan dan seperangkat sumber daya/*resource* yang berkenaan dengannya.

Walau dua proses dapat dihubungkan dengan program yang sama, program tersebut dianggap dua urutan eksekusi yang berbeda. Sebagai contoh, beberapa pengguna dapat menjalankan copy yang berbeda pada mail program, atau pengguna yang sama dapat meminta banyak copy dari program editor.

Tiap-tiap proses ini adakah proses yang berbeda dan walau bagian tulisan-*text* adalah sama, *data section* bervariasi. Dalam SO, juga umum untuk memiliki proses yang menghasilkan banyak proses begitu ia bekerja.

B. Keadaan Proses

Sebagaimana proses bekerja, maka proses tersebut merubah *state* (keadaan statis/ asal). Status dari sebuah proses didefinisikan dalam bagian oleh aktivitas yang ada dari proses tersebut. Tiap proses mungkin adalah satu dari keadaan berikut ini:

1. *New*: Proses sedang dikerjakan/ dibuat.
2. *Running*: Instruksi sedang dikerjakan.

3. *Waiting*: Proses sedang menunggu sejumlah kejadian untuk terjadi (seperti sebuah penyelesaian I/O atau penerimaan sebuah tanda/ signal).
4. *Ready*: Proses sedang menunggu untuk ditugaskan pada sebuah prosesor.
5. *Terminated*: Proses telah selsesai melaksanakan tugasnya/ mengeksekusi.

Nama-nama tersebut adalah arbitrer/berdasar opini, istilah tersebut bervariasi disepanjang sistem operasi. Namun, sistem operasi tertentu juga lebih baik menggambarkan keadaan/status proses. Penting untuk menyadari bahwa hanya satu proses dapat berjalan pada prosesor mana pun pada waktu kapan pun. Namun, banyak proses yang dapat *ready* atau *waiting*. Keadaan diagram yang berkaitan dangan keadaan tersebut dijelaskan pada Gambar 3.1.

C. Proces *Control Block*

Tiap proses digambarkan dalam sistem operasi oleh sebuah *Process Control Block* (PCB), juga disebut sebuah *control block*. Sebuah PCB ditunjukkan dalam Gambar 3.2. PCB berisikan banyak bagian dari informasi yang berhubungan dengan sebuah proses yang spesifik, seperti:

1. Keadaan proses seperti *new, ready, running, waiting, halted*, dan juga banyak lagi.
2. *Program counter* mengindikasikan address dari perintah selanjutnya untuk dijalankan untuk proses ini.
3. *CPU Register*, dalam PCB register bervariasi dalam jumlah dan jenis, tergantung pada rancangan komputer. Register tersebut termasuk accumulator, index register, stack pointer, general-puposes register, ditambah code information pada kondisi apa pun.

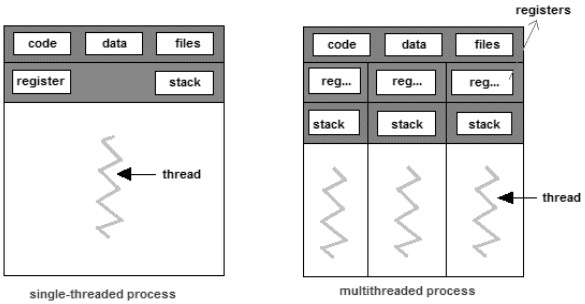
Bersamaan dengan program counter, dalam PCB keadaan/status informasi harus disimpan ketika gangguan terjadi, untuk memungkinkan proses tersebut berjalan/bekerja dengan benar setelahnya (lihat Gambar 3.2).

4. *Informasi Manajemen Memori*, dalam PCB informasi ini termasuk dalam suatu informasi sebagai nilai dari dasar dan batas register, tabel page/halaman, atau tabel segmen tergantung pada sistem memori yang digunakan oleh sistem operasi.
5. Informasi pencatatan, dalam PCB informasi ini termasuk jumlah dari CPU dan waktu riil yang digunakan, batas waktu, jumlah akun, jumlah job atau proses, dan banyak lagi.
6. Informasi status I/O, dalam PCB informas inii termasuk daftar dari perangkat I/O yang di gunakan pada proses ini, suatu daftar berkas-berkas yang sedang terbuka, dan banyak lagi.

Gambar 3.2 PCB

BAGIAN IV

IV. Thread



Gambar 4.1 Thread

Thread merupakan unit dasar dari penggunaan CPU, yang terdiri dari Thread_ID, program counter, register set, dan stack. Sebuah thread berbagi code section, data section, dan sumber daya sistem operasi dengan Thread lain yang dimiliki oleh proses yang sama. Thread juga sering disebut *lightweight process*.

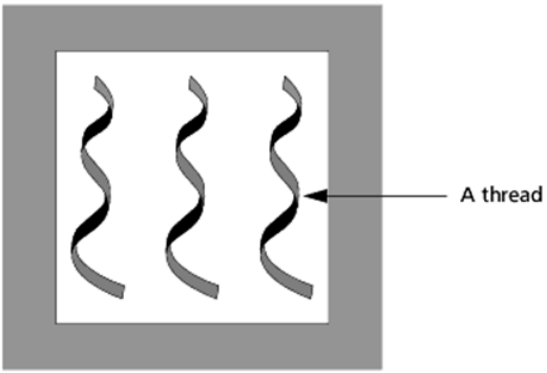
Sebuah proses tradisional atau *heavyweight process* mempunyai thread tunggal yang berfungsi sebagai pengendali. Perbedaan antara proses dengan thread tunggal dengan proses dengan thread yang banyak (Multi thread) adalah proses dengan thread yang banyak dapat mengerjakan lebih dari satu tugas pada satu satuan waktu.

Banyak sistem operasi modern telah memiliki konsep yang dikembangkan agar memungkinkan sebuah proses untuk memiliki eksekusi *multithreads*, agar dapat secara terus menerus mengetik dan menjalankan pemeriksaan ejaan didalam proses yang sama, maka sistem operasi tersebut memungkinkan proses untuk menjalankan lebih dari satu tugas pada satu waktu.

A. Definisi Thread

Secara informal, proses adalah program yang sedang dieksekusi. Ada dua jenis proses, proses berat (*heavyweight*) atau biasa dikenal dengan proses tradisional, dan proses ringan atau kadang disebut *thread*.

Thread saling berbagi bagian program, bagian data, dan sumber daya sistem operasi dengan thread lain yang mengacu pada proses yang sama. Thread terdiri atas ID thread, program counter, himpunan register, dan stack sehingga dengan banyaknya kontrol thread, proses dapat melakukan lebih dari satu pekerjaan pada waktu yang sama.



Gambar 4.2 A Thread

Sebuah thread dalam sistem operasi memberikan keuntungan sebagai berikut:

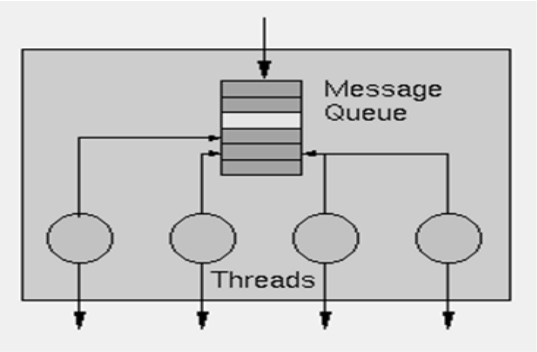
1. Tanggap, artinya ketika *Multithreading* mengizinkan program untuk berjalan terus walaupun pada bagian program tersebut di block atau sedang dalam keadaan menjalankan operasi yang lama/ panjang.
   Sebagai contoh, multithread web browser dapat mengizinkan pengguna berinteraksi dengan suatu

thread ketika suatu gambar sedang diload oleh thread yang lain.

2. Pembagian sumber daya, artinya *thread* mampu membagi memori dan sumber daya dari proses. Keuntungan dari pembagian kode adalah aplikasi mempunyai perbedaan aktifitas *thread* dengan alokasi memori yang sama.
3. Ekonomis, artinya ketika Sistem Operasi mengalokasikan memori dan sumber daya untuk membuat proses adalah sangat mahal. Alternatifnya, thread mampu membagi sumber daya dari proses sehingga lebih ekonomis untuk membuat proses.
4. Pemberdayaan arsitektur multiprosesor, maksudnya adalah dengan *multithreading* dapat ditingkatkan dengan menggunakan arsitektur multiprosesor, di mana setiap thread dapat jalan secara parallel pada prosesor yang berbeda.

   Pada arsitektur prosesor tunggal, CPU biasanya berpindah-pindah antara setiap thread dengan cepat, sehingga terdapat ilusi paralelisme, tetapi pada kenyataannya hanya satu thread yang berjalan di setiap waktu.

B. Model Multithreading



Gambar 4.3 Multithreading

Implementasi *Multithreading* dalam sistem operasi terdiri dari:

1. Model *Many to One*
   Model *many to one* ini memetakan beberapa tingkatan pengguna thread hanya kesatu buah kernel thread. Managemen proses thread dilakukan oleh (di ruang) pengguna, sehingga menjadi efisien, tetapi apabila sebuah thread melakukan sebuah pemblokingan terhadap sistem pemanggilan, maka seluruh proses akan berhenti (*blocked*).

   Kelemahan dari model ini adalah *multihreads* tidak dapat berjalan atau bekerja secara paralel di dalam multiprosesor dikarenakan hanya satu thread saja yang bisa mengakses kernel dalam suatu waktu.

2. Model *One to One*
   Model *one to one* memetakan setiap thread pengguna ke dalam satu kernel thread. Hal ini membuat model *one-to-one* lebih sinkron daripada model *many-to-one* dengan mengizinkan thread lain untuk berjalan ketika suatu thread membuat pemblokingan terhadap sistem pemanggilan. Hal ini juga mengizinkan multiple thread untuk berjalan secara parallel dalam multiprosesor.

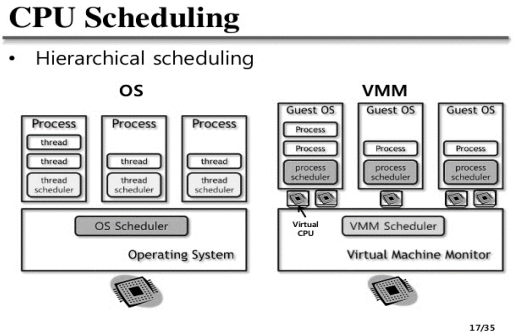
   Kelemahan model ini adalah dalam pembuatan thread pengguna dibutuhkan pembuatan korespondensi thread pengguna karena dalam proses pembuatan kernel thread dapat mempengaruhi kinerja dari aplikasi maka kebanyakan dari implementasi model ini membatasi jumlah thread yang didukung oleh sistem. Model *one-to-one* diimplementasikan oleh Windows NT dan OS/2.

3. Model *Many to Many*
   Beberapa tingkatan thread pengguna dapat menggunakan jumlah kernel thread yang lebih kecil atau sama dengan jumlah thread pengguna. Jumlah dari kernel thread dapat dispesifikasikan untuk beberapa aplikasi dan beberapa mesin (suatu aplikasi dapat dialokasikan lebih dari beberapa kernel thread dalam multiprosesor daripada dalam uniprosesor) dimana model many-to-one mengizinkan pengembang untuk membuat thread pengguna sebanyak mungkin, konkurensi tidak dapat tercapai karena hanya satu thread yang dapat dijadualkan oleh kernel dalam satu waktu.

   Model one-to-one mempunyai konkurensi yang lebih tinggi, tetapi pengembang harus hati-hati untuk tidak membuat terlalu banyak thread tanpa aplikasi dan dalam kasus tertentu mungkin jumlah thread yang dapat dibuat dibatasi.

BAGIAN V

V. Penjadwalan CPU



Gambar 5.1 Urutan Kedatangan Proses

Penjadwalan CPU adalah basis dari multi programming sistem operasi dengan cara men-switch CPU di antara proses. Akibatnya sistem operasi bisa membuat komputer produktif.

A. Konsep Dasar

Tujuan dari multi programming adalah agar proses bisa berjalan secara bersamaan dengan tujuan untuk memaksimalkan kinerja dari CPU. Pada sistem uniprosesor, tidak pernah ada proses yang berjalan lebih dari satu. Bila ada proses yang lebih dari satu maka yang lain harus mengantri sampai CPU bebas.

Ide dari multi porgamming sangat sederhana. Ketika sebuah proses dieksekusi yang lain harus menunggu sampai selesai. Di sistem komputer yang sederhana CPU akan banyak dalam posisi *idle* sehingga semua waktu akan terbuang. Dengan multiprogamming sistem operasi bisa menggunakan waktu secara produktif. Beberapa proses disimpan di dalam memori dalam satu waktu. Ketika proses harus menuggu, sistem operasi mengambil CPU untuk

memproses proses tersebut dan meninggalkan proses yang sedang dieksekusi.

Penjadwalan adalah fungsi dasar dari suatu sistem operasi. Hampir semua sumber komputer dijadwal sebelum digunakan. CPU salah satu sumber dari komputer yang penting yang menjadi sentral dari sentral penjadwalan di sistem operasi.

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 *penjadwalan short term*. Penjadwal memilih dari sekian proses yang ada di memori yang sudah siap dieksekusi, den mengalokasikan CPU untuk mengeksekusinya

Penjadwalan CPU mungkin akan dijalankan ketika terjadi proses:

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

B. Kriteria Penjadwal

Algoritma penjadwalan CPU yang berbeda mempunyai property yang berbeda. Dalam memilih algoritma yang digunakan untuk situasi tertentu, harus dipikirkan properti yang berbeda untuk algoritma yang berbeda. Banyak kriteria yang dianjurkan untuk membandingkan penjadwalan CPU algoritma. Kritria yang biasanya digunakan dalam memilih adalah:

1. *CPU utilization*, artinya 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 sampai 90 persen.

2. *Throughput* artinya, jika CPU mengeksekusi suatu proses, berarti CPU telah melaksanakan sebuah kerja. Salah satu ukuran kerja adalah banyak proses yang diselesaikan per unit waktu, yang kemudan disebut sebagai *throughput*. Untuk proses yang lama mungkin 1 proses per jam, untuk proses yang sebentar mungkin 10 proses perdetik.
3. *Turnaround time* yaitu, interval dari waktu yang diizinkan dengan waktu yang dibutuhkan untuk menyelesaikan sebuah proses. *Turnaround time* merupakan jumlah periode untuk menunggu untuk bisa ke memori, menunggu di *ready queue*, eksekusi di CPU, dan melakukan I/O.
4. *Waiting time*, artinya dalam penjadwalan CPU algoritma penjadwal CPU tidak mempengaruhi waktu untuk melaksanakan proses tersebut atau I/O, hanya mempengaruhi jumlah waktu yang dibutuhkan proses di antrian ready, sehingga 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 bisa memproduksi output diawal, dan bisa meneruskan hasil yang baru sementara hasil yang sebelumnya telah diberikan ke user. Ukuran yang lain adalah waktu dari pengiriman 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.

Sehingga kriteria penjadwalan CPU adalah dengan memaksimalkan *CPU utilization* dan *throughput*, dan *minimalkan turnaround time*, *waiting time*, dan *response time.*

C. Algoritma Penjadwalan

Penjadwalan CPU berurusan dengan permasalahan memutuskan proses mana yang akan dilaksanakan, oleh karena itu banyak bermacam algoritma penjadwalan, seperti berikut:

1. *Algoritma First Come First Served (FCFS)*

   Ini merupakan algoritma yang paling sederhana, dengan skema proses yang meminta CPU mendapat prioritas. Implementasi dari FCFS mudah diatasi dengan FIFO queue.

   Contoh:
   Misalnya urutan kedatangan proses adalah P1, P2, P3, adalah pada gambar 5.1.

Process	Burst Time
P_1	24
P_2	3
P_3	3

Gambar 5.2 Urutan Kedatangan Proses

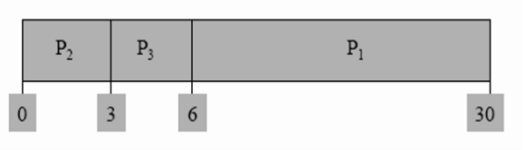
Gant chart- nya adalah :

Gambar 5.3Gant Chart Proses

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

Diketahui proses yang tiba adalah P2, P3, P1.

Gant chart-nya adalah :



Gambar 5.4 Urutan Kedatangan Proses

Waiting time untuk P1 = 6; P2 = 0; P3 = 3
Average waiting time: (6 + 0 + 3)/3 = 3

Penjadwalan FCFS algoritma adalah *nonpremptive*. Ketika CPU telah dialokasikan untuk sebuah proses, proses tetap menahan CPU sampai selesai.

FCFS algortima jelas merupakan masalah bagi sistem time-sharing, di mana sangat penting untuk user mendapatkan pembagian CPU pada regular interval. Itu akan menjadi bencana untuk megizinkan satu proses pada CPU untuk waktu yang tidak terbatas

2. *Algoritma Shortest Job Fist (SJF)*
   Salah satu algoritma yang lain adalah Shortest Job First. Algoritma ini berkaitan dengan waktu setiap proses. Ketika CPU bebas proses yang mempunyai waktu terpendek untuk menyelesaikannya mendapat prioritas.

   Seandainya dua proses atau lebih mempunyai waktu yang sama maka FCFS algoritma digunakan untuk menyelsaikan masalah tersebut. Ada dua skema dalam SJFS ini yaitu:

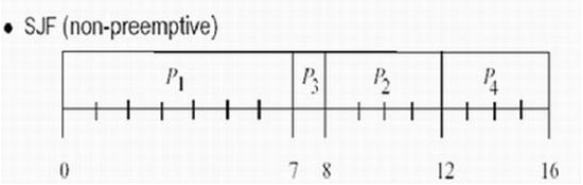
   a. *Nonpremptive*, ketika CPU memberikan kepada proses itu tidak bisa ditunda hingga selesai.

b. *Premptive*, bila sebuah proses datang dengan waktu prose lebih rendah dibandingkan dengan waktu proses yang sedang dieksekusi oleh CPU maka proses yang waktunya lebih rendah mendapatkan prioritas. Skema ini disebut juga Short - Remaining Time First (SRTF).

Contoh Non-Premptive:

Process	Arrival Time	Burst Time
P1	0.0	7
P2	2.0	4
P3	4.0	1
P4	5.0	4

SJF (Non Premptive):

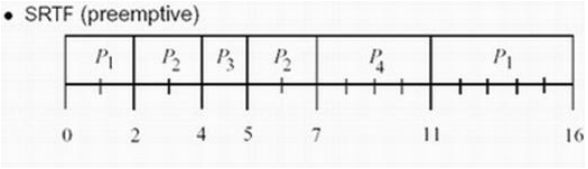


Average waiting time = (0 + 6 + 3 + 7)/4 – 4

Contoh Premptive SJF:

Process	Arrival Time	Burst Time
P1	0.0	7
P2	2.0	4
P3	4.0	1
P4	5.0	4

SJF (Premptive):



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

SJF algoritma mungkin adalah yang paling optimal, karena ia memberikan rata-rata minimum *waiting* untuk kumpulan dari proses yang mengantri dengan mengeksekusi waktu yang paling pendek baru yang paling lama, sehingga akibatnya rata-rata waktu menunggu jadi menurun.

Hal yang sulit dengan SJF algoritma adalah mengetahui waktu dari proses berikutnya. Untuk penjadwal *long term* (lama) di sistem *batch*, bisa menggunakan panjang batas waktu proses yang user sebutkan ketika dia mengirim pekerjaan. Oleh karena itu SJF sering digunakan di penjadwal long term.

Walau pun SJF optimal tetapi ia tidak bisa digunakan untuk penjadwal *CPU short term*. Tidak ada jalan untuk mengetahui panjang dari CPU burst berikutnya. Salah satu cara untuk mengimplementasikannya adalah dengan memprediksikan CPU burst berikutnya.

3. *Penjadwalan Prioritas*

Penjadwalan SJF (Shortest Job First) adalah kasus khusus untuk algoritma penjadwal prioritas. Prioritas dapat diasosiasikan masing-masing proses dan CPU dialokasikan untuk proses dengan prioritas tertinggi. Untuk proritas yang sama dilakukan dengan FCFS.

Ada pun algoritma penjadwal prioritas adalah setiap proses akan mempunyai prioritas (bilangan integer). Beberapa sistem menggunakan integer dengan urutan kecil untuk proses dengan prioritas rendah, dan sistem lain juga bisa menggunakan integer urutan kecil untuk proses dengan prioritas tinggi. Tetapi dalam teks ini diasumsikan bahwa integer

kecil merupakan prioritas tertinggi. CPU diberikan ke proses dengan prioritas tertinggi (integer kecil adalah prioritas tertinggi).

Dalam algoritma ini ada dua skema yaitu:

a. *Preemptive*: proses dapat di interupsi jika terdapat prioritas lebih tinggi yang memerlukan CPU.
b. *Nonpreemptive*: proses dengan prioritas tinggi akan mengganti pada saat pemakain time-slice habis.

SJF adalah contoh penjadwal prioritas dimana prioritas ditentukan oleh waktu pemakaian CPU berikutnya. Permasalahan yang muncul dalam penjadwalan prioritas adalah *indefinite blocking* atau *starvation*.

Kadang-kadang untuk kasus dengan prioritas rendah mungkin tidak pernah dieksekusi. Solusi untuk algoritma penjadual prioritas adalah *aging*. Prioritas akan naik jika proses makin lama menunggu waktu jatah CPU.

4. Penjadwalan Round Robin

Algoritma Round Robin (RR) dirancang untuk sistem *time sharing*. Algoritma ini mirip dengan penjadwal FCFS, namun *preemption* ditambahkan untuk switch antara proses. Antrian ready diperlakukan atau dianggap sebagai antrian sirkular. CPU meng-alokasikan masing-masing proses untuk interval waktu tertentu sampai satu time *slice/quantum*.

Berikut algoritma untuk penjadwal Round Robin:

a. Setiap proses mendapat jatah waktu CPU (time slice/quantum) tertentu, Time

slice/quantum umumnya antara 10 - 100 milidetik.

b. Setelah time slice/ quantum maka proses akan di-preempt dan dipindahkan ke antrian ready.
c. Proses ini berlangsung adil dan sederhana.

Jika terdapat 'n' proses di 'antrian ready' dan waktu quantum 'q' (milidetik), maka:

a. Setiap proses akan mendapatkan 1/n dari waktu CPU.
b. Proses tidak akan menunggu lebih lama dari: (n-1)q time units.

Kinerja dari algoritma ini tergantung dari ukuran time *quantum*. *Time Quantum* dengan ukuran yang besar maka akan sama dengan FCFS. *Time Quantum* dengan ukuran yang kecil maka time quantum harus diubah ukurannya lebih besar dengan respek pada alih konteks sebaliknya akan memerlukan ongkos yang besar.

D. Penjadwalan Multipe Processor

Multiprocessor membutuhkan penjadwalan yang lebih rumit karena mempunyai banyak kemungkinan yang dicoba tidak seperti pada processor tunggal. Pada penjadwalan multipresor penjadwalan menjadi lebih kompleks, Jika ada beberapa prosesor yang identik tersedia maka load sharing akan terjadi.

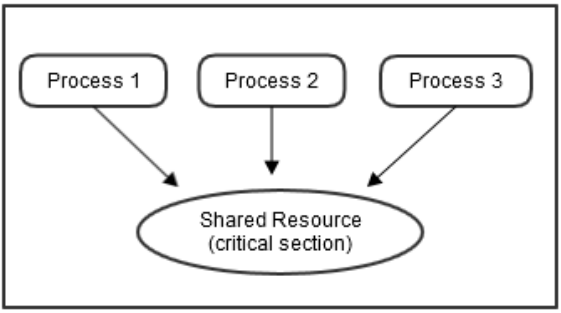
Dalam kasus ini, bagaimana pun, satu prosesor bisa menjadi idle dengan antrian yang kosong sedangkan yang lain sangat sibuk. Untuk mengantisipasi hal ini kita menggunakan *ready queue* yang biasa. Semua proses pergi ke satu *queue* dan dijadwalkan untuk prosesor yang bisa dipakai.

Dalam skema tersebut, salah satu penjadwalan akan digunakan. Salah satu cara menggunakan *symetric multiprocessing* (SMP) di mana setiap prosesor menjadwalkan diri sendiri. Setiap prosesor memeriksa *ready queue* dan memilih proses yang akan dieksekusi.

Beberapa sistem membawa struktur satu langkah kedepan, dengan membawa semua keputusan penjadwalan, I/O prosesing, dan aktivitas sistem yang lain ditangani oleh satu prosesor yang bertugas sebagai master prosesor. Prosesor yang lain mengeksekusi hanya user code yang disebut *asymmetric multiprosessing* jauh lebih mudah.

BAGIAN VI

VI. Sinkronisasi



Gambar 6.1 Sinkronisasi Sistem Operasi

Sinkronisasi mengacu pada salah satu dari dua proses yang berbeda tetapi saling berkaitan satu sama lainnya. Dua proses ini merupakan sinkronisasi data dan sinkronisasi proses. Inti dari permasalahan sinkronisasi ini nantinya akan mengacu pada sebuah gagasan yang di dalamnya terdapat banyak proses yang pada titik tertentu akan bertemu, bergabung dalam rangka untuk mencapai sebuah kesepakatan ataupun komitmen untuk suatu urutan tindakan tertentu.

Sinkronisasi ini mengacu pada gagasan yang menjaga beberapa salinan dari dataset yang ada dalam koherensi antara satu sama lain.

A. Definisi Sinkronisasi

Sinkronisasi merupakan suatu proses pengaturan jalannya beberapa proses pada waktu yang bersamaan untuk menyamakan waktu dan data supaya tidak terjadi *inconsitensi* (ketidak konsistenan) data akibat adanya akses data secara konkuren agar hasilnya bagus dan sesuai

dengan apa yang diharapkan. Disini sinkronisasi diperlukan agar data tersebut tetap konsisten.

Pengertian dari Sinkronisasi adalah akses bebarengan untuk berbagi dua bersama dapat mengakibatkan inkosistensi data. Pemeliharaan konsistensi data memerlukan mekanisme untuk memastikan eksekusi dari proses kerjasama.

Jadi sinkronisasi itu lebih jelasnya merupakan akses secara berbarengan untuk berbagi dua bersama dalam mengakibatkan inkonsistensi data. Pemeliharaan konsistensi data memerlukan mekanisme yang tepat untuk memastikan eksekusi dari proses kerjasama.

Sinkronisasi ini dibutuhkan ketika menemukan sebuah kasus yang menyebabkan ketidakkonsistenan data sehingga data menjadi tidak konkuren. Jadi pada proses sinkronisasi primitif biasanya yang digunakan adalah untuk mengimplementasikan sinkronisasi data dan proses ini di lakukan oleh fungsi lain di luar sistem utama dari sistem operasi.

Tujuan dari sinkronisasi itu sendiri ialah untuk menghindari terjadinya inkonsitensi data karena pengaksesan oleh beberapa proses yang berbeda serta untuk mengatur urutan jalannya proses-proses sehingga dapat berjalan dengan baik dan sesuai apa yang di harapkan. Adapun manfaat sikronisasi pada sistem operasi adalah:

1. Adanya akses-akses data yang sama yang di lakukan secara bersamaan bisa saja menyebabkan data menjadi tidak konsisten
2. Agar semua data yang ada tetap konsisten membutuhkan mekanisme-mekanisme agar bisa di pastikan proses eksekusi berjalan.

3. Adanya *Race Condition* yang merupakan kondisi dimana beberapa proses mengakses dan memanipulasi data secara bersamaan akan membuat nilai terkhirnya nanti bergantung dari proses mana yang duluan diakhiri.

B. Race Condition

Race Condition merupakan situasi dimana beberapa proses mengakses dan memanipulasi data secara bersamaan. Nilai terakhir dari data bergantung dari proses mana yang selesai terakhir. Untuk menghindari *Race Condition,* proses-proses secara bersamaan harus di-sinkronisasikan.

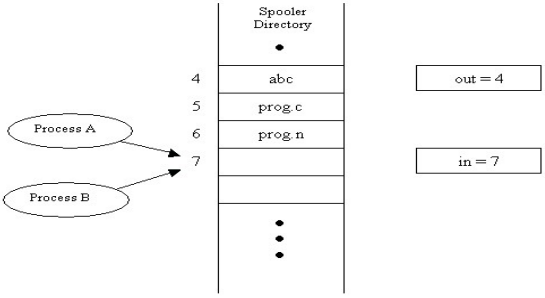
Dalam beberapa sistem operasi, proses-proses yang berjalan bersamaan mungkin untuk membagi beberapa penyimpanan umum, masing-masing dapat melakukan proses baca (*read*) dan proses tulis (*write*). Penyimpanan bersama (*shared storage*) mungkin berada di memori utama atau berupa sebuah berkas bersama, lokasi dari memori bersama tidak merubah kealamian dari komunikasi atau masalah yang muncul.

Untuk mengetahui bagaimana komunikasi antar proses bekerja, mari kita simak sebuah contoh sederhana, sebuah *print spooler*. Ketika sebuah proses ingin mencetak sebuah berkas, proses tersebut memasukkan nama berkas ke dalam sebuah spooler direktori yang khusus. Proses yang lain, printer daemon, secara periodik memeriksa untuk mengetahui jika ada banyak berkas yang akan dicetak, dan jika ada berkas yang sudah dicetak dihilangkan nama berkasnya dari direktori.

Bayangkan bahwa spooler direktori memiliki slot dengan jumlah yang sangat besar, diberi nomor 0, 1, 2, 3, 4,... masing-masing dapat memuat sebuah nama berkas.

Juga bayangkan bahwa ada dua variabel bersama, out, penunjuk berkas berikutnya untuk dicetak, dan in, menunjuk slot kosong di direktori. Dua variabel tersebut dapat menangani sebuah two-word berkas untuk semua proses. Dengan segera, slot 0, 1, 2, 3 kosong (berkas telah selesai dicetak), dan slot 4, 5, 6 sedang terisi (berisi nama dari berkas yang antre untuk dicetak).

Lebih atau kurang secara besamaan, proses A dan B, mereka memutuskan untuk antre untuk sebuah berkas untuk dicetak. Situasi ini digambarkan seperti gambar 6.2 berikut:



Gambar 6.1 Race Condition

Dalam Murphy's Law kasus tesebut dapat terjadi. Proses A membaca in dan menyimpan nilai "7" di sebuah variabel lokal yang disebut *next_free_slot*. Sebuah clock interrupt terjadi dan CPU memutuskan bahwa proses A berjalan cukup lama, sehingga digantikan oleh proses B. Proses B juga membaca in, dan juga mengambil nilai 7, sehingga menyimpan nama berkas di slot nomor 7 dan memperbaharui nilai in menjadi 8. Maka proses mati dan melakukan hal lain.

Akhirnya proses A berjalan lagi, dimulai dari tempat di mana proses tersebut mati. Hal ini terlihat dalam *next_free_slot*, ditemukan nilai 7 di sana, dan menulis nama

berkas di slot nomor 7, menghapus nama berkas yang baru saja diletakkan oleh proses B. Kemudian proses A menghitung *next_free_slot* + 1, yang nilainya 8 dan memperbaharui nilai in menjadi 8.

Direktori *spooler* sekarang secara internal konsisten, sehingga printer daemon tidak akan memberitahukan apa pun yang terjadi, tetapi poses B tidak akan mengambil output apa pun. Situasi seperti ini, dimana dua atau lebih proses melakukan proses *reading* atau *writing* beberapa *shared data* dan hasilnya bergantung pada ketepatan berjalan disebut *race condition*.

C. Critical Section

Critical Section adalah sebuah segmen kode dimana sebuah proses memiliki sumber daya bersama yang diakses terdiri dari:

1. *Entry Section*, kode yang digunakan untuk masuk ke dalam critical section
2. *Critical Section*, kode dimana hanya ada satu proses yang dapat dieksekusi pada satu waktu
3. *Exit Section*, akhir dari *critical section*, mengizinkan proses lain
4. *Remainder Section*, merupakan kode istirahat setelah masuk ke *critical section*.

D. Semaphore

Semaphore adalah pendekatan yang diajukan oleh Djikstra, dengan prinsip bahwa dua proses atau lebih dapat bekerja sama dengan menggunakan penanda-penanda sederhana. Seperti proses dapat dipaksa berhenti pada suatu saat, sampai proses mendapatkan penanda tertentu itu.

Sembarang kebutuhan koordinasi kompleks dapat dipenuhi dengan struktur penanda yang cocok untuk

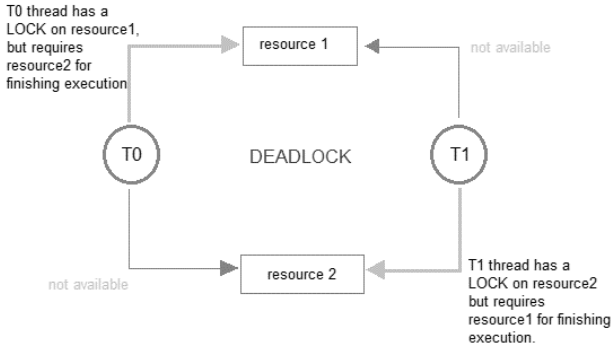
kebutuhan itu. Variabel khusus untuk penanda ini disebut *semaphore*.

Semaphore mempunyai dua sifat, yaitu:

1. Semaphore dapat diinisialisasi dengan nilai non-negatif.
2. Terdapat dua operasi terhadap semaphore, yaitu Down dan Up. Usulan asli yang disampaikanDjikstra adalah operasi P dan V.

BAGIAN VII

VII. Deadlock



Gambar 7.1 Deadlock Sistem Operasi

Misalkan pada suatu komputer terdapat dua buah program, sebuah tape drive dan sebuah printer. Program A mengontrol tape drive, sementara program B mengontrol printer. Setelah beberapa saat, program A meminta printer, tapi printer masih digunakan. Berikutnya, B meminta tape drive, sedangkan A masih mengontrol tape drive.

Dua program tersebut memegang kontrol terhadap sumber daya yang dibutuhkan oleh program yang lain. Tidak ada yang dapat melanjutkan proses masing-masing sampai program yang lain memberikan sumber dayanya, tetapi tidak ada yang mengalah. Kondisi inilah yang disebut *Deadlock* atau pada beberapa buku disebut *Deadly Embrace.*

A. Definisi *Deadlock*

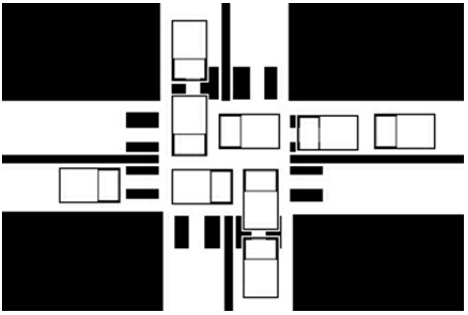
Deadlock yang mungkin dapat terjadi pada suatu proses disebabkan proses itu menunggu suatu kejadian tertentu yang tidak akan pernah terjadi. Dua atau lebih

proses dikatakan berada dalam kondisi *deadlock*, bila setiap proses yang ada menunggu suatu kejadian yang hanya dapat dilakukan oleh proses lain dalam himpunan tersebut.

Terdapat kaitan antara *overhead* dari mekanisme koreksi dan manfaat dari koreksi *deadlock* itu sendiri. Pada beberapa kasus, *overhead* atau ongkos yang harus dibayar untuk membuat sistem bebas *deadlock* menjadi hal yang terlalu mahal dibandingkan jika mengabaikannya. Sementara pada kasus lain, seperti pada *real-time process control*, mengizinkan *deadlock* akan membuat sistem menjadi kacau dan membuat sistem tersebut tidak berguna.

Contoh berikut ini terjadi pada sebuah persimpangan jalan. Beberapa hal yang dapat membuat *deadlock* pada suatu persimpangan, yaitu:

- Terdapat satu jalur pada jalan.
- Mobil digambarkan sebagai proses yang sedang menuju sumber daya.
- Untuk mengatasinya beberapa mobil harus preempt (mundur).
- Sangat memungkinkan untuk terjadinya starvation (kondisi proses tak akan mendapatkan sumber daya).



Gambar 7.1 Proses Terjadi Deadlock

B. Strategi Menghadapi *Deadlock*

Strategi untuk menghadapi deadlock dapat dibagi menjadi tiga pendekatan, yaitu:

1. Mengabaikan adanya *deadlock*.
2. Memastikan bahwa *deadlock* tidak akan pernah ada, baik dengan metode pencegahan, dengan mencegah empat kondisi *deadlock* agar tidak akan pernah terjadi. Metode Menghindari *deadlock*, yaitu mengizinkan empat kondisi *deadlock*, tetapi menghentikan setiap proses yang kemungkinan mencapai *deadlock*.
3. Membiarkan *deadlock* untuk terjadi, pendekatan ini membutuhkan dua metode yang saling mendukung, yaitu:
    a. Pendeteksian *deadlock*, yaitu untuk mengidentifikasi ketika deadlock terjadi.
    b. Pemulihan *deadlock*, yaitu untuk mengembalikan kembali sumber daya yang dibutuhkan pada proses yang memintanya.

C. Mencegah *Deadlock*

Metode ini merupakan metode yang paling sering digunakan. Metode Pencegahan dianggap sebagai solusi yang bersih dipandang dari sudut tercegahnya *deadlock*. Tetapi pencegahan akan mengakibatkan kinerja utilisasi sumber daya yang buruk.

Metode pencegahan menggunakan pendekatan dengan cara meniadakan empat syarat yang dapat menyebabkan *deadlock* terjadi pada saat eksekusi Coffman (1971).

Syarat pertama yang akan dapat ditiadakan adalah *Mutual Exclusion*, jika tidak ada sumber daya yang secara khusus diperuntukkan bagi suatu proses maka tidak akan pernah terjadi *deadlock*. Namun jika membiarkan ada dua

atau lebih proses mengakses sebuah sumber daya yang sama akan menyebabkan chaos.

Langkah yang digunakan adalah dengan *spooling* sumber daya, yaitu dengan mengantrikan job-job pada antrian dan akan dilayani satu-satu.

Beberapa masalah yang mungkin terjadi adalah:

1. Tidak semua dapat di-*spool*, tabel proses sendiri tidak mungkin untuk di-*spool*.
2. Kompetisi pada ruang disk untuk *spooling* sendiri dapat mengarah pada *deadlock*.

Hal inilah yang menyebabkan mengapa syarat pertama tidak dapat ditiadakan, jadi *mutual exclusion* benar-benar tidak dapat dihilangkan.

Cara kedua dengan meniadakan kondisi *hold* and *wait* terlihat lebih menjanjikan. Jika suatu proses yang sedang menggunakan sumber daya dapat dicegah agar tidak dapat menunggu sumber daya yang lain, maka *deadlock* dapat dicegah. Langkah yang digunakan adalah dengan membuat proses agar meminta sumber daya yang mereka butuhkan pada awal proses sehingga dapat dialokasikan sumber daya yang dibutuhkan. Namun jika terdapat sumber daya yang sedang terpakai maka proses tersebut tidak dapat memulai prosesnya.

Masalah yang mungkin terjadi:

1. Sulitnya mengetahui berapa sumber daya yang dibutuhkan pada awal proses.
2. Tidak optimalnya pengunaan sumber daya jika ada sumber daya yang digunakan hanya beberapa waktu dan tidak digunakan tapi tetap dimiliki oleh suatu proses yang telah memintanya dari awal.

Meniadakan syarat ketiga *non preemptive* ternyata tidak lebih menjanjikan dari meniadakan syarat kedua, karena dengan meniadakan syarat ketiga maka suatu proses dapat dihentikan ditengah jalan. Hal ini tidak dimungkinkan karena hasil dari suatu proses yang dihentikan menjadi tidak baik.

Cara terakhir adalah dengan meniadakan syarat keempat *circular wait*. Terdapat dua pendekatan, yaitu:

1. Mengatur agar setiap proses hanya dapat menggunakan sebuah sumber daya pada suatu waktu, jika menginginkan sumber daya lain maka sumber daya yang dimiliki harus dilepas.
2. Membuat penomoran pada proses-proses yang mengakses sumber daya. Suatu proses dimungkinkan untuk dapat meminta sumber daya kapan pun, tetapi permintaannya harus dibuat terurut.

Masalah yang mungkin terjadi dengan mengatur bahwa setiap proses hanya dapat memiliki satu proses adalah bahwa tidak semua proses hanya membutuhkan satu sumber daya, untuk suatu proses yang kompleks dibutuhkan banyak sumber daya pada saat yang bersamaan. Sedangkan dengan penomoran masalah yang dihadapi adalah tidak terdapatnya suatu penomoran yang dapat memuaskan semua pihak.

Secara ringkas pendekatan yang digunakan pada metode pencegahan deadlock dan masalah-masalah yang mungkin menghambatnya, terangkum dalam tabel di bawah ini.

Syarat	Langkah	Kelemahan
Mutual Exclusion	*Spooling* sumber daya	Dapat menyebabkan *chaos*
Hold and Wait	Meminta sumber daya di awal	Sulit memperkirakan di awal dan tidak optimal
No Pre-emptive	Mengambil sumber daya di tengah proses	Hasil proses tidak akan baik
Circular Wait	Penomoran permintaan sumber daya	Tidak ada penomoran yang emuaskan semua pihak

D. Menghindari Deadlock

Pendekatan metode ini adalah dengan hanya memberi kesempatan ke permintaan sumber daya yang tidak mungkin akan menyebabkan *deadlock*. Metode ini memeriksa dampak pemberian akses pada suatu proses, jika pemberian akses tidak mungkin menuju kepada *deadlock*, maka sumber daya akan diberikan pada proses yang meminta.

Jika tidak aman, proses yang meminta akan di-suspend sampai suatu waktu permintaannya aman untuk diberikan. Kondisi ini terjadi ketika setelah sumber daya yang sebelumnya dipegang oleh proses lain telah dilepaskan.

Kondisi aman yang dimaksudkan selanjutnya disebut sebagai *safe-state*, sedangkan keadaan yang tidak memungkinkan untuk diberikan sumber daya yang diminta disebut *unsafe-state*.

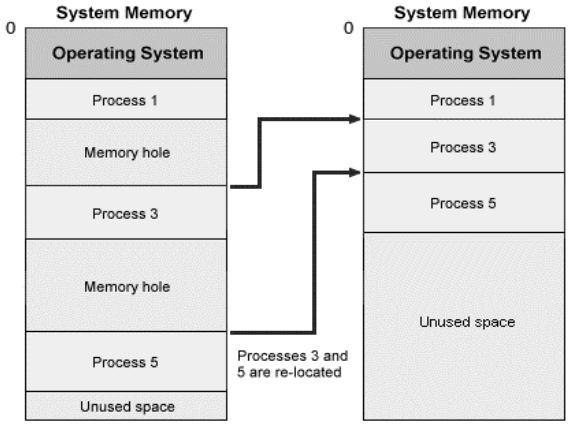
1. Kondisi Aman (*Safe state*)
   Suatu keadaan dapat dinyatakan sebagai *safe state* jika tidak terjadi *deadlock* dan terdapat cara untuk memenuhi semua permintaan sumber daya yang

ditunda tanpa menghasilkan *deadlock* dengan cara mengikuti urutan tertentu.

2. Kondisi Tak Aman (*Unsafe state*)
   Suatu state dinyatakan sebagai state tak selamat (*unsafe state*) jika tidak terdapat cara untuk memenuhi semua permintaaan yang saat ini ditunda dengan menjalankan proses-proses dengan suatu urutan.

BAGIAN VIII

VIII. Manajemen Memori



Gambar 8.1 Manajemen Memori

Memori merupakan inti dari sistem komputer modern. CPU mengambil instruksi dari memori sesuai yang ada pada program counter. Instruksi dapat berupa menempatkan/menyimpan dari/ke alamat di memori, penambahan, dan sebagainya. Dalam managemen memori ini, akan dibahas bagaimana urutan alamat memori yang dibuat oleh program yang berjalan.

A. Ruang Alamat Fisik dan Logik

Alamat yang dibuat CPU akan merujuk ke sebuah alamat logik. Sedangkan alamat yang dilihat oleh memori adalah alamat yang dimasukkan ke register di memori, merujuk pada alamat fisik pada pengikatan alamat, waktu compile dan waktu penempatan menghasilkan daerah dimana alamat logik dan alamat fisik sama.

Sedangkan pada waktu eksekusi menghasilkan alamat fisik dan logik yang berbeda. Kita biasanya menyebut

alamat logik dengan alamat virtual. Kumpulan alamat logik yang dibuat oleh program adalah ruang alamat logik. Kumpulan alamat fisik yang berkoresponddensi dengan alamat logik sibut ruang alamat fisik. Pemetaan dari virtual ke alamat fisik dilakukan oleh *Memory-Management* Unit (MMU), yang merupakan sebuah perangkat keras.

Register utamanya disebut *relocation-register*. Nilai pada *relocation register* bertambah setiap alamat dibuat oleh proses pengguna, pada waktu yang sama alamat ini dikirim ke memori.

Program pengguna tidak dapat langsung mengakses memori. Ketika ada program yang menunjuk ke alamat memori, kemudian mengoperasikannya, dan menaruh lagi di memori, akan di lokasikan awal oleh MMU, karena program pengguna hanya bernterkasi dengan alamat logik. Konsep untuk memisahkan ruang alamat logik dan ruang alamat fisik, adalah inti dari managemen memori yang baik.

B. Penukaran (*Swap*)

Sebuah proses membutuhkan memori untuk dieksekusi. Sebuah proses dapat ditukar sementara keluar memori ke *backing store* (disk), dan kemudian dibawa masuk lagi ke memori untuk dieksekusi.

Sebagai contoh, asumsi multiprogramming, dengan penjadwalan algoritma CPU Round-Robin. Ketika kuantum habis, manager memori akan mulai menukar keluar proses yang selesai, dan memasukkan ke memori proses yang bebas. Sementara penjadwalan CPU akan mangalokasikan waktu untuk proses lain di memori. Ketika tiap proses menghabiskan waktu kuantumnya, proses akan ditukar dengan proses lain.

Idealnya memori manager, dapat menukar proses-proses cukup cepat, sehingga selalu ada proses di memori siap dieksekusi, ketika penjadwal CPU ingin menjadwal ulang CPU. Besar kuantum juga harus cukup besar, sehingga jumlah perhitungan yang dilakukan antar pertukaran masuk akal.

Variasi dari kebijakan *swapping* ini, digunakan untuk algoritma penjadwalan berdasarkan prioritas. Jika proses yang lebih tinggi tiba, dan minta dilayani, memori manager dapat menukar keluar proses dengan prioritas yang lebih rendah, sehingga dapat memasukkan dan mengeksekusi proses dengan prioritas yang lebih tinggi. Ketika proses dengan prioritas lebih tinggi selesai, proses dengan prioritas yang lebih rendah, dapat ditukar masuk kembali, dan melanjutkan. Macam-macam pertukaran ini kadang disebut *roll out*, dan *roll in*.

Normalnya, sebuah proses yang ditukar keluar, akan dimasukkan kembali ke tempat memori yang sama dengan yang digunakan sebelumnya. Batasan ini dibuat oleh metode pengikat alamat. Jika pengikatan dilakukan saat *assemble* atau *load time*, maka proses tidak bisa dipindahkan ke lokasi yang berbeda. Jika menggunakan pengikatan waktu eksekusi, maka akan mungkin menukar proses kedalam tempat memori yang berbeda. Karena alamat fisik dihitung selama proses eksekusi.

Pertukaran membutuhkan sebuah *backing store*. *Backing store* biasanya adalah sebuah disk yang cepat. Cukup besar untuk mengakomodasi semua kopi tampilan memori. Sistem memelihara *ready queue* terdiri dari semua proses yang mempunyai tampilan memori yang ada di backing store, atau di memori dan siap dijalankan.

Ketika penjadwal CPU memutuskan untuk meng-eksekusi sebuah proses, dia akan memanggil *dispatcher*, yang mengecek dan melihat apakah proses berikutnya ada di antrian memori. Jika proses tidak ada, dan tidak ada ruang memori yang kosong, *dispatcher* menukar keluar sebuah proses dan memaasukan proses yang diinginkan. Kemudian memasukkan ulang register dengan normal, dan mentransfer pengendali ke proses yang diinginkan.

Konteks waktu pergantian pada sistem *swapping* lumayan tinggi. Untuk efisiensi kegunaan CPU, kita ingin waktu eksekusi untuk tiap proses lebih lama dari waktu pertukaran. Karenanya digunakan CPU penjadualan round-robin, dimana kuantumnya harus lebih besar dari waktu pertukaran.

Perhatikan bahwa bagian terbesar dari waktu pertukaran, adalah waktu pengiriman. Total waktu pengiriman langsung didapat dari jumlah pertukaran memori. Proses dengan kebutuhan memori dinamis, akan membutuhkan *system call* (meminta dan melepaskan memori), untuk memberi tahu sistem operasi tentang perubahan kebutuhan memori.

Ada beberapa keterbatasan *swapping*. Jika kita ingin menukar sebuah proses kita harus yakin bahwa proses sepenuhnya diam. Konsentrasi lebih jauh jika ada penundaan I/O. Sebuah proses mungkin menunggu I/O, ketika kita ingin menukar proses itu untuk mengosongkan memori. Jika I/O secara asinkronus, mengakses memori dari I/O buffer, maka proses tidak bisa ditukar.

Misalkan I/O operation berada di antrian, karena device sedang sibuk, maka bila kita menukar keluar proses P1 dan memasukkan P2, mungkin saja operasi I/O akan

berusaha masuk ke memori yang sekarang milik P2. Dua solusi utama masalah ini adalah:

1. Jangan pernah menukar proses yang sedang menunggu I/O.
2. Untuk mengeksekusi operasi I/O hanya pada buffer sistem operasi.

Secara umum, ruang pertukaran dialokasikan sebagai potongan disk, terpisah dari sistem berkas, sehingga bisa digunakan secepat mungkin.

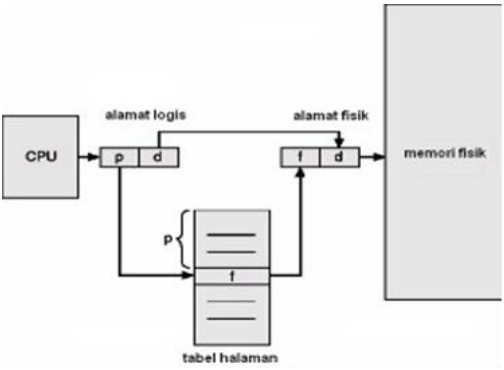
Belakangan pertukaran standar pertukaran digunakan di beberapa sistem. Ini membutuhkan terlalu banyak waktu untuk menukar dari pada untuk mengeksekusi untuk solusi managemen memori yang masuk akal. Modifikasi swapping digunakan di banyak versi di UNIX. Pertukaran awalnya tidak bisa, tapi akan mulai bila banyak proses yang jalan dan menggunakan batas jumlah memori.

C. Pemberian Page (Halaman)

Pemberian halaman mencegah masalah penting dari mengepaskan ukuran bongkahan memori yang bervariasi ke dalam penyimpanan cadangan, yang mana diderita oleh kebanyakan dari skema managemen memori sebelumnya. Ketika beberapa pecahan kode dari data yang tersisa di memori utama perlu untuk di tukar keluar, harus ditemukan ruang di penyimpanan cadangan.

Masalah pemecahan didiskusikan dengan kaitan bahwa teori utama juga lazim dengan penyimpanan cadangan, kecuali bahwa pengaksesanny lebih lambat, jadi kerapatan adalah tidak mungkin. Karena keuntungannya pada metode-metode sebelumnya, pemberian halaman dalam berbagai bentuk biasanya digunakan pada banyak sistem operasi.

Memori fisik yang dipecah menjadi blok-blok berukuran tetap disebut sebagai frame. Memori logis juga dipecah menjadi blok-blok dengan ukuran yang sama disebut sebagai halaman. Ketika proses akan dieksekusi, halamannya akan diisi ke dalam frames memori mana saja yang tersedia dari penyimpanan cadangan. Penyimpanan cadangan dibagi-bagi menjadi blok-blok berukuran tetap yang sama besarnya dengan frames di memori.



Gambar 8.2 Pemberian Halaman

Dukungan perangkat keras untuk pemberian halaman diilustrasikan pada gambar Gambar 8.2. Setiap alamat yang dihasilkan oleh CPU dibagi-bagi menjadi 2 bagian: sebuah nomor halaman (p) dan sebuah offset halaman (d). Nomor halaman digunakan sebagai indeks untuk tabel halaman. Tabel halaman mengandung basis alamat dari tiap-tiap halaman di memori fisik. Basis ini dikombinasikan dengan offset halaman untuk menentukan alamat memori fisik yang dikirim ke unit memori.

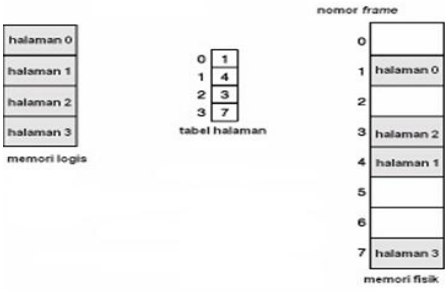
Ukuran halaman (seperti halnya ukuran frame) didefinisikan oleh perangkat keras. Khasnya ukuran dari sebuah halaman adalah pangkat 2 yang berkisar antara 512 byte dan 8192 byte per halamannya, tergantung dari arsitektur komputernya.

Penentuan pangkat 2 sebagai ukuran halaman akan memudahkan penterjemahan dari memori logis ke nomor halaman dan offset halaman. Jika ukuran ruang dari memori logis adalah 2 pangkat m, dan ukuran sebuah halaman adalah 2 pangkat n unit pengalamatan (byte atau word), maka pangkat tinggi m-n bit dari alamat logis menandakan offset dari halaman. Jadi, alamat logisnya adalah: *dimana p merupakan index ke tabel halaman dan d adalah pemindahan dalam halaman*.

Untuk konkritnya, walau kecil sekali, lihat pada Gambar 8.3. Menggunakan ukuran halaman4 byte dan memori fisik 32 byte (8 halaman), kami menunjukan bagaimana pandangan pengguna terhadap memori dapat dipetakan kedalam memori fisik. Alamat logis 0 adalah halaman 0, offset 0.

Pemberian index menjadi tabel halaman, kita dapati bahwa halaman 0 berada pada frame 5. Jadi, alamat logis 0 memetakan ke alamat fisik 20 (=(5x4)+0). Alamat logis 3 (page 0, offset 3) memetakan ke alamat fisik 23 (=(5x4)+3). Alamat logis 4 adalah halaman 1, offset; menurut tabel halaman, halaman 1 dipetakan ke frame 6. Jadi, alamat logis 4 memetakan ke alamat fisik 24 (=(6x4)+0). Alamat logis 13 memetakan ke alamat fisik 9.

Pembentukan pemberian halaman itu sendiri adalah suatu bentuk dari penampungan dinamis. Setiap alamat logis oleh perangkat keras untuk pemberian halaman dibatasi ke beberapa alamat fisik. Pembaca yang setia akan menyadari bahwa pemberian halaman sama halnya untuk menggunakan sebuah tabel dari basis register, satu untuk setiap frame di memori.



Gambar 8.3 Pemberian Halaman Pada Memori Fisik

Ketika kita menggunakan skema pemberian halaman, kita tidak memiliki pemecah-pemecahan dari luar dengan sembarang frame kosong dapat dialokasikan ke proses yang membutuhkan. Bagaimana pun juga kita mungkin mempunyai beberapa pemecahan di dalam. Mengingat bahwa *frame-frame* dialokasikan sebagai unit. Jika kebutuhan memori dari sebuah proses tidak menurun pada batas halaman, frame terakhir yang dialokasikan mungkin tidak sampai penuh.

Untuk contoh, jika halamannya 2048 byte, proses 72.766 byte akan membutuhkan 35 halaman tambah 1086 byte. Alokasinya menjadi 36 frame, menghasilkan fragmentasi internal dari 2048 - 1086 = 962 byte. Pada kasus terburuknya, proses akan membutuhkan n halaman tambah satu byte. Sehingga dialokasikan n + 1 frame, menghasilkan fragmentasi internal dari hampir semua frame.

Jika ukuran proses tidak bergantung dari ukuran halaman, kita mengharapkan fragmentasi internal hingga rata-rata setengah halaman per prosesnya. Pertimbangan ini memberi kesan bahwa ukuran halaman yang kecil sangat diperlukan sekali. Bagaimana pun juga, ada sedikit pemborosan dilibatkan dalam masukan tabel halaman, dan pemborosan ini dikurangi dengan ukuran halaman

meningkat. Juga disk I/O lebih efisien ketika jumlah data yang dipindahkan lebih besar. Umumnya, ukuran halaman bertambah seiring bertambahnya waktu seperti halnya proses, himpunan data, dan memori utama telah menjadi besar. Hari ini, halaman umumnya berukuran 2 atau 4 kilobyte.

Ketika proses tiba untuk dieksekusi, ukurannya yang diungkapkan di halaman itu diperiksa. Setiap pengguna membutuhkan satu frame. Jadi, jika proses membutuhkan n halaman, maka pasti ada n frame yang tersedia di memori. Jika ada n frame yang tersedia, maka mereka dialokasikan di proses ini. Halaman pertama dari proses diisi ke salah satu frame yang sudah teralokasi, dan nomor frame-nya diletakkan di tabel halaman untuk proses ini. Halaman berikutnya diisikan ke frame yang lain, dan nomor frame-nya diletakkan ke tabel halaman, dan begitu seterusnya (Gambar 8.3).

Aspek penting dari pemberian halaman adalah pemisahan yang jelas antara pandangan pengguna tentang memori dan fisik memori sesungguhnya. Program pengguna melihat memori sebagai satu ruang berdekatan yang tunggal, hanya mengandung satu program itu.

Faktanya, program pengguna terpencar-pencar didalam memori fisik, yang juga menyimpan program lain. Perbedaan antara pandangan pengguna terhadap memori dan fisik memori sesungguhnya disetarakan oleh perangkat keras penterjemah alamat. Alamat logis diterjemahkan ke alamat fisik.

Pemetaan ini tertutup bagi pengguna dan dikendalikan oleh sistem operasi. Perhatikan bahwa proses pengguna dalam definisi tidak dapat mengakses memori yang bukan haknya. Tidak ada pengalamatan memori di luar tabel

halamannya, dan tabelnya hanya melingkupi halaman yang proses itu miliki.

Karena sistem operasi mengatur memori fisik, maka harus waspada dari rincian alokasi memori fisik: frame mana yang dialokasikan, frame mana yang tersedia, berapa banyak total frame yang ada, dan masih banyak lagi. Informasi ini umumnya disimpan di struktur data yang disebut sebagai tabel frame. Tabel frame punya satu masukan untuk setiap fisik halaman frame, menandakan apakah yang terakhir teralokasi ataukah tidak, jika teralokasi maka kepada halaman mana dari proses mana.

Tambahan lagi sistem operasi harus waspada bahwa proses-proses pengguna beroperasi di ruang pengguna, dan semua logis alamat harus dipetakan untuk menghasilkan alamat fisik. Jika pengguna melakukan pemanggilan sistem (contohnya melakukan I/O) dan mendukung alamat sebagai parameter (contohnya penyangga), alamatnya harus dipetakan untuk menghasilkan alamat fisik yang benar. Sistem operasi mengatur salinan tabel halaman untuk tiap-tiap proses, seperti halnya ia mengatur salinan dari counter instruksi dan isi register.

Salinan ini digunakan untuk menterjemahkan alamat fisik ke alamat logis kapan pun sistem operasi ingin memetakan alamat logis ke alamat fisik secara manual. Ia juga digunakan oleh dispatcher CPU untuk mendefinisikan tabel halaman perangkat keras ketika proses dialokasikan ke CPU. Oleh karena itu pemberian halaman meningkatkan waktu alih konteks.

D. Segmentasi

Salah satu aspek penting dari managemen memori yang tidak dapat dihindari dari pemberian halaman adalah pemisahan cara pandang pengguna dengan tentang

bagaimana memori dipetakan dengan keadaan yang sebenarnya. Pada kenyataannya pemetaan tersebut memperbolehkan pemisahan antara memori logis dan memori fisik.

Segmentasi adalah sebuah bagian dari managemen memori yang mengatur pengalamatan dari memori yang terdiri dari segmen-segmen. *Logical address space* adalah kumpulan dari segmen-segmen yang mana tiap-tiap segmen mempunyai nama dan panjang. Alamat tersebut menunjukkan alamat dari segmen tersebut dan offset-nya didalam segmen-segmen tersebut.

Pengguna kemudian menentukan pengalamatan dari setiap segmen menjadi dua bentuk, nama segmen dan offset dari segmen tersebut (Hal ini berbeda dengan pemberian halaman, dimana pengguna hanya menentukan satu buah alamat, dimana pembagian alamat menjadi dua dilakukan oleh perangkat keras, semua ini tidak dapat dilihat oleh user).

Untuk kemudahan pengimplementasian, segmen-segmen diberi nomor dan direferensikan dengan menggunakan penomoran tersebut, daripada dengan menggunakan nama. Maka logical address space terdiri dari dua tipe yaitu nomor-segmen dan offset. Pada umumnya, program dari pengguna akan dikompilasi, dan kompilator tersebut akan membuat segmen-segmen tersebut secara otomatis.

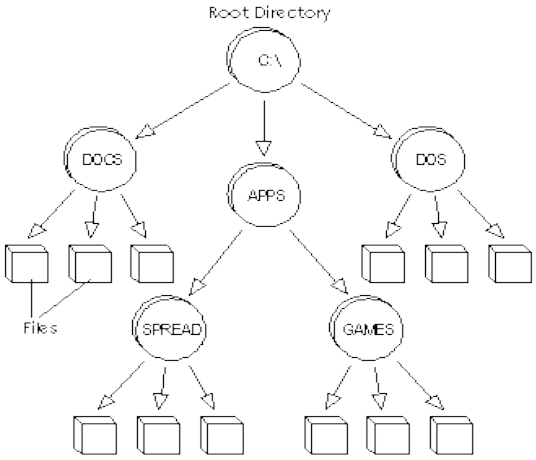
Jika mengambil contoh kompilator dari Pascal, maka kemungkinan kompilator tersebut akan membuat beberapa segmen yang terpisah untuk:

1. Variabel Global;
2. Prosedur dari pemanggilan *stack*, untuk menyimpan parameter dan pengembalian alamat;

3. Porsi dari kode untuk setiap prosedur atau fungsi; dan
4. Variabel lokal dari setiap prosedur dan fungsi.

BAGIAN IX

IX. Sistem Berkas



Gambar 9.1 Sistem Berkas

Sistem berkas merupakan mekanisme penyimpanan on-line serta untuk akses, baik data maupun program yang berada dalam Sistem Operasi. Terdapat dua bagian penting dalam sistem berkas, yaitu:

1. Kumpulan berkas, sebagai tempat penyimpanan data.
2. Struktur direktori, yang mengatur dan menyediakan informasi mengenai seluruh berkas dalam sistem.

Pada bab ini, akan dibahas tentang berbagai aspek dari berkas dan struktur, cara menangani proteksi berkas, cara mengalokasikan ruang pada disk, melacak lokasi data, serta meng-interface bagian-bagian lain dari sistem operasi ke penyimpanan sekunder.

A. Konsep Dasar

Seperti yang telah kita ketahui, komputer dapat menyimpan informasi ke beberapa media penyimpanan yang berbeda, seperti *magnetic disks*, *magnetic tapes*, dan *optical disks*. Agar komputer dapat digunakan dengan nyaman, sistem operasi menyediakan sistem penyimpanan dengan sistematika yang seragam.

Sistem Operasi mengabstraksi properti fisik dari media penyimpanannya dan mendefinisikan unit penyimpanan logis, yaitu berkas. Berkas dipetakan ke media fisik oleh sistem operasi. Media penyimpanan ini umumnya bersifat *non-volatile*, sehingga kandungan di dalamnya tidak akan hilang jika terjadi gagal listrik mau pun *system reboot*.

Berkas adalah kumpulan informasi berkait yang diberi nama dan direkam pada penyimpanan sekunder. Dari sudut pandang pengguna, berkas merupakan bagian terkecil dari penyimpanan logis, artinya data tidak dapat ditulis ke penyimpanan sekunder kecuali jika berada di dalam berkas.

Biasanya berkas merepresentasikan program (baik source mau pun bentuk objek) dan data. Data dari berkas dapat bersifat numerik, alfabetik, alfanumerik, atau pun biner. Format berkas juga bisa bebas, misalnya berkas teks, atau dapat juga diformat pasti. Secara umum, berkas adalah urutan bit, byte, baris, atau catatan yang didefinisikan oleh pembuat berkas dan pengguna.

Informasi dalam berkas ditentukan oleh pembuatnya. Ada banyak beragam jenis informasi yang dapat disimpan dalam berkas. Hal ini disebabkan oleh struktur tertentu yang dimiliki oleh berkas, sesuai dengan jenisnya masing-masing.

Contohnya:

- *Text file*; yaitu urutan karakter yang disusun ke dalam baris-baris.
- *Source file*; yaitu urutan subroutine dan fungsi, yang nantinya akan dideklarasikan.
- *Object file*; merupakan urutan byte yang diatur ke dalam blok-blok yang dikenali oleh linker dari sistem.
- *Executable file*; adalah rangkaian code section yang dapat dibawa loader ke dalam memori dan dieksekusi.

B. Atribut Pada Berkas

Berkas diberi nama, untuk kenyamanan bagi pengguna, dan untuk acuan bagi data yang terkandung di dalamnya. Nama berkas biasanya berupa string atau karakter.

Beberapa sistem membedakan penggunaan huruf besar dan kecil dalam penamaan sebuah berkas, sementara sistem yang lain menganggap kedua hal di atas sama. Ketika berkas diberi nama, maka berkas tersebut akan menjadi mandiri terhadap proses, pengguna, bahkan sistem yang membuatnya. Atribut berkas terdiri dari:

1. Nama; merupakan satu-satunya informasi yang tetap dalam bentuk yang bisa dibaca oleh manusia (*human-readable form*)
2. Type; dibutuhkan untuk sistem yang mendukung beberapa type berbeda
3. Lokasi; merupakan pointer ke device dan ke lokasi berkas pada device tersebut
4. Ukuran (*size*); yaitu ukuran berkas pada saat itu, baik dalam byte, huruf, atau pun blok
5. Proteksi; adalah informasi mengenai kontrol akses, misalnya siapa saja yang boleh membaca, menulis, dan mengeksekusi berkas
6. Waktu, tanggal dan identifikasi pengguna; informasi ini biasanya disimpan untuk:

a. pembuatan berkas,
b. modifikasi terakhir yang dilakukan pada berkas, dan
c. penggunaan terakhir berkas.

Data tersebut dapat berguna untuk proteksi, keamanan, dan monitoring penggunaan dari berkas. Informasi tentang seluruh berkas disimpan dalam struktur direktori yang terdapat pada penyimpanan sekunder.

Direktori, seperti berkas, harus bersifat *non-volatile*, sehingga keduanya harus disimpan pada sebuah device dan baru dibawa bagian per bagian ke memori pada saat dibutuhkan.

C. Operasi Pada Berkas

Sebuah berkas adalah jenis data abstrak. Untuk mendefinisikan berkas secara tepat, kita perlu melihat operasi yang dapat dilakukan pada berkas tersebut. Sistem operasi menyediakan *system calls* untuk membuat, membaca, menulis, mencari, menghapus, dan sebagainya.

Berikut dapat kita lihat apa yang harus dilakukan sistem operasi pada keenam operasi dasar pada berkas:

1. Membuat sebuah berkas.
   Ada dua cara dalam membuat berkas. Pertama, tempat baru di dalam sistem berkas harus di alokasikan untuk berkas yang akan dibuat. Kedua, sebuah direktori harus mempersiapkan tempat untuk berkas baru, kemudian direktori tersebut akan mencatat nama berkas dan lokasinya pada sistem berkas.

2. Menulis pada sebuah berkas.
   Untuk menulis pada berkas, kita menggunakan system call beserta nama berkas yang akan ditulisi

dan informasi apa yang akan ditulis pada berkas. Ketika diberi nama berkas, sistem mencari ke direktori untuk mendapatkan lokasi berkas. Sistem juga harus menyimpan penunjuk tulis pada berkas dimana penulisan berikut akan ditempatkan. Penunjuk tulis harus diperbaharui setiap terjadi penulisan pada berkas.

3. Membaca sebuah berkas.
   Untuk dapat membaca berkas, kita menggunakan system call beserta nama berkas dan di blok memori mana berkas berikutnya diletakkan. Sama seperti menulis, direktori mencari berkas yang akan dibaca, dan sistem menyimpan penunjuk baca pada berkas dimana pembacaan berikutnya akan terjadi. Ketika pembacaan dimulai, penunjuk baca harus diperbaharui. Sehingga secara umum, suatu berkas ketika sedang dibaca atau ditulis, kebanyakan sistem hanya mempunyai satu penunjuk, baca dan tulis menggunakan penunjuk yang sama, hal ini menghemat tempat dan mengurangi kompleksitas sistem.
4. Menempatkan kembali sebuah berkas.
   Direktori yang bertugas untuk mencari berkas yang bersesuaian, dan mengembalikan lokasi berkas pada saat itu. Menempatkan berkas tidak perlu melibatkan proses I/O. Operasi sering disebut pencarian berkas.
5. Menghapus sebuah berkas.
   Untuk menghapus berkas kita perlu mencari berkas tersebut di dalam direktori. Setelah ditemukan kita membebaskan tempat yang dipakai berkas tersebut (sehingga dapat digunakkan oleh berkas lain) dan menghapus tempatnya di direktori.
6. Memendekkan berkas.
   Ada suatu keadaan dimana pengguna menginginkan atribut dari berkas tetap sama tetapi ingin

menghapus isi dari berkas tersebut. Fungsi ini mengizinkan semua atribut tetap sama tetapi panjang berkas menjadi nol, hal ini lebih baik dari pada memaksa pengguna untuk menghapus berkas dan membuatnya lagi.

Enam operasi dasar ini sudah mencakup operasi minimum yang di butuhkan. Operasi umum lainnya adalah menyambung informasi baru diakhir suatu berkas, mengubah nama suatu berkas, dan lain-lain. Operasi dasar ini kemudian digabung untuk melakukan operasi lainnya.

Sebagai contoh misalnya kita menginginkan salinan dari suatu berkas, atau menyalin berkas ke peralatan I/O lainnya seperti printer, dengan cara membuat berkas lalu membaca dari berkas lama dan menulis ke berkas yang baru.

Hampir semua operasi pada berkas melibatkan pencarian berkas pada direktori. Untuk menghindari pencarian yang lama, kebanyakan sistem akan membuka berkas apabila berkas tersebut digunakan secara aktif. Sistem operasi akan menyimpan tabel kecil yang berisi informasi semua berkas yang dibuka yang disebut "tabel berkas terbuka". Ketika berkas sudah tidak digunakan lagi dan sudah ditutup oleh yang menggunakan, maka sistem operasi mengeluarkan berkas tersebut dari tabel berkas terbuka.

Beberapa sistem terkadang langsung membuka berkas ketika berkas tersebut digunakan dan otomatis menutup berkas tersebut jika program atau pemakainya dimatikan. Tetapi pada sistem lainnya terkadang mem-butuhkan pembukaan berkas secara tersurat dengan system call (open) sebelum berkas dapat digunakan.

Implementasi dari buka dan tutup berkas dalam lingkungan dengan banyak pernggguna seperti UNIX, lebih rumit. Dalam sistem seperti itu pengguna yang membuka berkas mungkin lebih dari satu dan pada waktu yang hampir bersamaan. Umumnya sistem operasi menggunakan tabel internal dua level. Ada tabel yang mendata proses mana saja yang membuka berkas tersebut, kemudian tabel tersebut menunjuk ke tabel yang lebih besar yang berisi informasi yang berdiri sendiri seperti lokasi berkas pada disk, tanggal akses dan ukuran berkas. Biasanya tabel tersebut juga memiliki data berapa banyak proses yang membuka berkas tersebut.

Jadi, pada dasarnya ada beberapa informasi yang terkait dengan pembukaan berkas yaitu:

1. Penunjuk Berkas.
   Pada sistem yang tidak mengikutkan batas berkas sebagai bagian dari system call baca dan tulis, sistem tersebut harus mengikuti posisi dimana terakhir proses baca dan tulis sebagai penunjuk. Penunjuk ini unik untuk setiap operasi pada berkas, maka dari itu harus disimpan terpisah dari atribut berkas yang ada pada disk.
2. Penghitung berkas yang terbuka.
   Setelah berkas ditutup, sistem harus mengosongkan kembali tabel berkas yang dibuka yang digunakan oleh berkas tadi atau tempat di tabel akan habis. Karena mungkin ada beberapa proses yang membuka berkas secara bersamaan dan sistem harus menunggu sampai berkas tersebut ditutup sebelum mengosongkan tempatnya di tabel. Penghitung ini mencatat banyaknya berkas yang telah dibuka dan ditutup, dan menjadi nol ketika yang terakhir membaca berkas menutup berkas tersebut barulah sistem dapat mengosongkan tempatnya di tabel.

3. Lokasi berkas pada disk.
   Kebanyakan operasi pada berkas memerlukan sistem untuk mengubah data yang ada pada berkas. Informasi mengenai lokasi berkas pada disk disimpan di memori agar menghindari banyak pembacaan pada disk untuk setiap operasi.

Beberapa sistem operasi menyediakan fasilitas untuk memetakan berkas ke dalam memori pada sistem memori virtual. Hal tersebut mengizinkan bagian dari berkas ditempatkan pada suatu alamat di memori virtual. Operasi baca dan tulis pada memori dengan alamat tersebut dianggap sebagai operasi baca dan tulis pada berkas yang ada di alamat tersebut.

Menutup berkas mengakibatkan semua data yang ada pada alamat memori tersebut dikembalikan ke disk dan dihilangkan dari memori virtual yang digunakan oleh proses.

D. Jenis Berkas

Pertimbangan utama dalam perancangan sistem berkas dan seluruh sistem operasi, apakah sistem operasi harus mengenali dan mendukung jenis berkas. Jika suatu sistem operasi mengenali jenis dari berkas, maka ia dapat mengoperasikan berkas tersebut.

Contoh apabila pengguna mencoba mencetak berkasyang merupakan kode biner dari program yang pasti akan menghasilkan sampah, hal ini dapat dicegah apabila sistem operasi sudah diberitahu bahwa berkas tersebut merupakan kode biner.

Teknik yang umum digunakan dalam implementasi jenis berkas adalah menambahkan jenis berkas dalam nama berkas. Nama dibagi dua, nama dan akhiran (ekstensi), biasanya dipisahkan dengan karakter titik.

Sistem menggunakan akhiran tersebut untuk mengindikasikan jenis berkas dan jenis operasi yang dapat dilakukan pada berkas tersebut. Sebagai contoh hanya berkas yang berakhiran .bat, .exe atau .com yang bisa dijalankan (eksekusi).

Program aplikasi juga menggunakan akhiran tersebut untuk mengenal berkas yang dapat dioperasikannya. Akhiran ini dapat ditimpa atau diganti jika diperbolehkan olehsistem operasi.

Beberapa sistem operasi menyertakan dukungan terhadap akhiran, tetapi beberapa menyerahkan kepada aplikasi untuk mengatur akhiran berkas yang digunakan, sehingga jenis dari berkas dapat menjadi petunjuk aplikasi apa yang dapat mengoperasikannya.

Sistem UNIX tidak dapat menyediakan dukungan untuk akhiran berkas karena menggunakan angka ajaib yang disimpan di depan berkas untuk mengenali jenis berkas. Tidak semua berkas memiliki angka ini, jadi sistem tidak bisa bergantung pada informasi ini.

Tetapi UNIX memperbolehkan akhiran berkas tetapi hal ini tidak dipaksakan atau tergantung sistem operasi, kebanyakan hanya untuk membantu pengguna mengenali jenis isi dari suatu berkas.

Tabel 9.1 Jenis Berkas

Jenisberkas	**Akhiran**	**Fungsi**
executable	exe, com, bat, bin	Program yang siap dijalankan
objek	obj, o	Bahasa mesin,kode terkompilasi

kodeasal(sourcecode)	c,cc,pas,java,asm,a	Kode asal dari berbagai bahasa
batch	bat, sh	Perintah pada shell
text	txt, doc	Data text, document
pengolahkata	wpd, tex, doc	Format jenis pengolah data
library	lib, a, DLL	Library untuk rutin program
print,gambar	ps, dvi, gif	Format ASCII atau biner untuk dicetak
archive	arc, zip, tar	Beberapa berkas yang dikumpulkan menjadi satu berkas. Terkadang dimampatkan untuk penyimpanan

E. Struktur Berkas

Kita juga dapat menggunakan jenis berkas untuk mengidentifikasi struktur dalam dari berkas. Berkas berupa source dan objek memiliki struktur yang cocok dengan harapan program yang membaca berkas tersebut. Suatu berkas harus memiliki struktur yang dikenali oleh sistem operasi.

Sebagai contoh, sistem operasi menginginkan suatu berkas yang dapat dieksekusi memiliki struktur tertentu agar dapat diketahui dimana berkas tersebut akan ditempatkan di memori dan di mana letak instruksi pertama berkas tersebut. Beberapa sistem operasi mengembangkan ide ini sehingga mendukung beberapa struktur berkas, dengan beberapa operasi khusus untuk memanipulasi berkas dengan struktur tersebut.

Kelemahan memiliki dukungan terhadap beberapa struktur berkas adalah: Ukuran dari sistem operasi dapat

menjadi besar, jika sistem operasi mendefinisikan lima struktur berkas yang berbeda maka ia perlu menampung kode untuk yang diperlukan untuk mendukung semuanya.

Setiap berkas harus dapat menerapkan salah satu struktur berkas tersebut. Masalah akan timbul ketika terdapat aplikasi yang membutuhkan struktur informasi yang tidak didukung oleh sistem operasi tersebut.

Beberapa sistem operasi menerapkan dan mendukung struktur berkas sedikit struktur berkas. Pendekatan ini digunakan pada MS-DOS dan UNIX. UNIX menganggap setiap berkas sebagai urutan 8-bit byte, tidak ada interpretasi sistem operasi terhadap dari bit-bit ini. Skema tersebut menawarkan fleksibilitas tinggi tetapi dukungan yang terbatas.

Setiap aplikasi harus menambahkan sendiri kode untuk menerjemahkan berkas masukan ke dalam struktur yang sesuai. Walau bagaimana pun juga sebuah sistem operasi harus memiliki minimal satu struktur berkas yaitu untuk berkas yang dapat dieksekusi sehingga sistem dapat memuat berkas dalam memori dan menjalankannya.

Sangat berguna bagi sistem operasi untuk mendukung struktur berkas yang sering digunakan karena akan menghemat pekerjaan pemrogram. Terlalu sedikit struktur berkas yang didukung akan mempersulit pembuatan program, terlalu banyak akan membuat sistem operasi terlalu besar dan pemrogram akan bingung.

F. Metode Akses

Ketika digunakan, informasi penyimpanan berkas harus dapat diakses dan dibaca ke dalam memori komputer. Beberapa sistem hanya menyediakan satu metode akses

untuk berkas. Pada sistem yang lain, contohnya IBM, terdapat banyak dukungan metode akses yang berbeda.

Masalah pada sistem tersebut adalah memilih yang mana yang tepat untuk digunakan pada satu aplikasi tertentu. Metode akses terbagi atas:

1. *Sequential Access* merupakan metode yang paling sederhana. Informasi yang disimpan dalam berkas diproses berdasarkan urutan. Operasi dasar pada suatu berkas adalah tulis dan baca. Operasi baca membaca berkas dan meningkatkan pointer berkas selama di jalur lokasi I/O. Operasi tulis menambahkan ke akhir berkas dan meningkatkan ke akhir berkas yang baru.
   Metode ini didasarkan pada tape model sebuah berkas, dan dapat bekerja pada kedua jenis device akses (urut mau pun acak).
2. *Direct Access* merupakan metode yang membiarkan program membaca dan menulis dengan cepat pada berkas yang dibuat dengan *fixed-length logical order* tanpa adanya urutan. Metode ini sangat berguna untuk mengakses informasi dalam jumlah besar. Biasanya database memerlukan hal seperti ini.
   Operasi berkas pada metode ini harus dimodifikasi untuk menambahkan nomor blok sebagai parameter. Pengguna menyediakan nomor blok ke sistem operasi biasanya sebagai nomor blok relatif, yaitu indeks relatif terhadap awal berkas.

   Penggunaan nomor blok relatif bagi sistem operasi adalah untuk memutuskan lokasi berkas diletakkan dan membantu mencegah pengguna dari pengaksesan suatu bagian sistem berkas yang bukan bagian pengguna tersebut.

G. Operasi Pada Direktori

Operasi-operasi yang dapat dilakukan pada direktori adalah:

1. Mencari berkas, kita dapat menemukan sebuah berkas didalam sebuah struktur direktori. Karena berkas-berkas memiliki nama simbolik dan nama yang sama dapat mengindikasikan keterkaitan antara setiap berkas-berkas tersebut, mungkin kita berkeinginan untuk dapat menemukan seluruh berkas yang nama-nama berkas membentuk pola khusus.
2. Membuat berkas, kita dapat membuat berkas baru dan menambahkan berkas tersebut kedalam direktori.
3. Menghapus berkas, apabila berkas sudah tidak diperlukan lagi, kita dapat menghapus berkas tersebut dari direktori.
4. Menampilkan isi direktori, kita dapat menampilkan seluruh berkas dalam direktori, dan kandungan isi direktori untuk setiap berkas dalam daftar tersebut.
5. Mengganti nama berkas, karena nama berkas merepresentasikan isi dari berkas kepada user, maka user dapat merubah nama berkas ketika isi atau penggunaan berkas berubah. Perubahan nama dapat merubah posisi berkas dalam direktori.
6. Melintasi sistem berkas, ini sangat berguna untuk mengakses direktori dan berkas didalam struktur direktori.

H. Tipe Akses Pada Berkas

Salah satu cara untuk melindungi berkas dalam komputer kita adalah dengan melakukan pembatasan akses pada berkas tersebut. Pembatasan akses yang dimaksudkan adalah kita, sebagai pemilik dari sebuah berkas, dapat menentukan operasi apa saja yang dapat dilakukan oleh pengguna lain terhadap berkas tersebut.

Pembatasan ini berupa sebuah *permission* ataupun *not permittedoperation*, tergantung pada kebutuhan pengguna lain terhadap berkas tersebut. Di bawah ini adalah beberapa operasi berkas yang dapat diatur aksesnya:

1. *Read*: Membaca dari berkas
2. *Write*: Menulis berkas
3. *Execute*: Meload berkas kedalam memori untuk dieksekusi.
4. *Append*: Menambahkan informasi kedalam berkas di akhir berkas.
5. *Delete*: Menghapus berkas.
6. *List*: Mendaftar properti dari sebuah berkas.
7. *Rename*: Mengganti nama sebuah berkas.
8. *Copy*: Menduplikasikan sebuah berkas.
9. *Edit*: Mengedit sebuah berkas.

Selain operasi-operasi berkas diatas, perlindungan terhadap berkas dapat dilakukan dengan mekanisme yang lain. Namun setiap mekanisme memiliki kelebihan dan kekurangan. Pemilihan mekanisme sangatlah tergantung pada kebutuhan dan spesifikasi sistem.

I. Organisasi Sistem Berkas Berkas

Sistem operasi menyediakan sistem berkas agar data mudah disimpan, diletakkan dan diambil kembali dengan mudah. Terdapat dua masalah desain dalam membangun suatu sistem berkas.

Masalah pertama adalah definisi dari sistem berkas. Hal ini mencakup definisi berkas dan atributnya, operasi ke berkas, dan struktur direktori dalam mengorganisasikan berkas-berkas.

Masalah kedua adalah membuat algoritma dan struktur data yang memetakan struktur logikal sistem berkas ke tempat penyimpanan sekunder. Pada dasarnya sistem

berkas tersusun atas beberapa tingkatan, yaitu (dari yang terendah):

- *I/O control*, terdiri atas *driver device* dan *interrupt handler*. Driver device adalah perantara komunikasi antara sistem operasi dengan perangkat keras.
- *Basic file system*, diperlukan untuk mengeluarkan perintah generik ke device driver untuk baca dan tulis pada suatu blok dalam disk.
- *File-organization module*, informasi tentang alamat logika dan alamat fisik dari berkas tersebut.
- Modul ini juga mengatur sisa disk dengan melacak alamat yang belum dialokasikan dan menyediakan alamat tersebut saat user ingin menulis berkas ke dalam disk.
- *Logical file system*, tingkat ini berisi informasi tentang simbol nama berkas, struktur dari direktori, dan proteksi dan sekuriti dari berkas tersebut.

Daftar Pustaka

[Coffmanea1997] E G Coffman, Jr., M J Elphick, dan A Shoshani, 1971, System Deadlocks, Computing
Surveys, Vol.3, No.2.

[Deitel1990] H M Deitek, 1990, Operating Systems, Massachusetts, Addison-Wesley, 2nd ed. [Hariyanto1997] B Hariyanto, 1997, Sistem Operasi, Informatika, Bandung.

[Havender1968] J W Havender, 1968, Avoiding Deadlock in Multitasking Systems, IBM Systems
Journal, Vol.7, No.2.

[Samik-Ibrahim2001] Rahmat Samik-Ibrahim, 2001, Ujian Mid Test 2001, Fakultas Ilmu Komputer, Universitas Indonesia.

[Silberschatz2000] Avi Silberschatz, Peter Galvin, dan Grag Gagne, 2000, Applied Operating Systems: First Edition, Edisi Pertama, John Wiley & Sons.

[Stallings2001] William Stallings, 2001, Operating Systems, Fourth Edition, Prentice Hall. [Tanenbaum1992] Andrew S Tanenbaum, 1992, Modern Operating Systems, Englewood Cliffs, New Jersey.

[Walsh2002] Norman Walsch dan Leonard Muellner, Bob Stayton, 1999, 2000, 2001, 2002, DocBook: The Definitive Guide, Version 2.0.7, O'Reilly.

Tentang Penulis

Ronal Watrianthos dilahirkan di Padang pada tanggal 12 Desember 1980. Anak pertama dari empat bersaudara ini menamatkan pendidikan dasar di SDN 13 Padang, SMPN 20 Padang, dan SMAN 6 Padang. Pendidikan tingginya diselesaikan tahun 2004 di Universitas Putra Indonesia 'YPTK' Padang Program Studi Sistem Komputer jenjang pendidikan Strata 1 (S1) dan mendapat gelar Sarjana Komputer (S.Kom). Pada tahun 2013 berhasil menyelesaikan pendidikan Strata II (S2) dan mendapat gelar Magister Komputer (M.Kom) di Universitas Putra Indonesia 'YPTK' Padang dengan kosentrasi jurusan Teknik Informatika.

Memiliki minat yang tinggi terhadap komputer khususnya bidang jaringan dan internet, perangkat keras, open source, serta sistem operasi. Mata kuliah yang diampu adalah Jaringan Komputer, Komunikasi Data, Organisasi Komputer, Sistem Operasi, dan Teknologi Open Source.

Saat ini menjadi dosen tetap di AMIK Labuhan Batu (Yayasan Universitas Labuhanbatu–YULB) Kabupaten Labuhanbatu Sumatera Utara pada Program Studi Manajemen Informatika. Penulis bisa dikontak melalui email: *mail.to.ronal@gmail.com*

Iwan Purnama lahir di Ajamu pada tanggal 12 Februari 1992. Setelah lulus Sekolah Menengah Atas di Madrasah Aliyah Swasta Al-Ikhlas Ajamu pada tahun 2010, Penulis melanjutkan pendidikan ke AMIK Stiekom Sumatera Utara konsentrasi Manajemen Informatika Komputer dan menyelesaikannnya pada tahun 2013. Pada tahun yang sama, Penulis melanjutkan kembali Pendidikan S1 di STMIK Triguna Dharma Medan dengan memperoleh gelar Sarjana Komputer Sistem Informasi Pada tahun 2014. Selanjutnya penulis melanjutkan pendidikan S2 Komputer di Universitas Putra Indonesia YPTK Padang Sumatera Barat dan memperoleh gelar Magister Sistem Informasi pada tahun 2016.

Penulis sekarang berprofesi sebagai Dosen Tetap AMIK Labuhanbatu, Sofware Developer, Instruktur Kursus Komputer di IbayKomputer.com sekaligus pemilik Ibaykomputer.com. Penulis dapat dihubungi melalui email : *iwanpurnama2014@gmail.com* atau *ibaykomputer@gmail.com*