

Sistem Operasi

Bahan Kuliah IKI-20230

**Gabungan Kelompok Kerja 21–28 Semester Genap
2002/2003 dan 41–49 Semester Ganjil 2003/2004 Mata
Kuliah Sistem Operasi**

Sistem Operasi: Bahan Kuliah IKI-20230

oleh Gabungan Kelompok Kerja 21–28 Semester Genap 2002/2003 dan 41–49 Semester Ganjil 2003/2004 Mata Kuliah Sistem Operasi

\$Revision: 1.2 \$ Edisi

Diterbitkan 8 Desember 2003

Hak Cipta © 2003 oleh Gabungan Kelompok Kerja 21–28 Semester Genap 2002/2003 dan 41–49 Semester Ganjil 2003/2004 Mata Kuliah Sistem Operasi.

Silakan menyalin, mengedarkan, dan/ atau, memodifikasi bagian dari dokumen – \$Revision: 1.2 \$ – yang dikarang oleh Gabungan Kelompok Kerja 21–28 Semester Genap 2002/2003 dan 41–49 Semester Ganjil 2003/2004 Mata Kuliah Sistem Operasi, sesuai dengan ketentuan "*GNU Free Documentation License* versi 1.1" atau versi selanjutnya dari FSF (*Free Software Foundation*); tanpa bagian "*Invariant*", tanpa teks "*Front-Cover*", dan tanpa teks "*Back-Cover*". Lampiran A ini> berisi salinan lengkap dari lisensi tersebut. Ketentuan ini **TIDAK** berlaku untuk bagian dan/ atau kutipan yang bukan dikarang oleh Gabungan Kelompok Kerja 21–28 Semester Genap 2002/2003 dan 41–49 Semester Ganjil 2003/2004 Mata Kuliah Sistem Operasi.

Catatan Revisi

Revisi 1.8 08-12-2003 Revised by: Kelompok 49

Versi rilis beta buku OS

Revisi 1.7 17-11-2003 Revised by: Kelompok 49

Versi rilis alfa buku OS

Revisi 1.5 17-11-2003 Revised by: Kelompok 49

Penggabungan pertama seluruh pekerjaan kelompok 41 sampai kelompok 48. Masih ada beberapa gambar yang belum lengkap. Rujukan

Revisi 1.4 08-11-2003 Revised by: Kelompok 49

Pengubahan template versi 1.3 dengan template yang baru yang akan digunakan dalam versi 1.4-2.0

Revisi 1.3.0.5 12-11-2003 Revised by: RMS46

Revisi ini diedit oleh Rahmat M. Samik-Ibrahim: dipilah sesuai dengan sub-pokok bahasan yang ada.

Revisi 1.3 30-09-2003 Revised by: RMS46

Revisi ini diedit oleh Rahmat M. Samik-Ibrahim: melanjutkan perbaikan tata letak dan pengindeksan.

Revisi 1.2 17-09-2003 Revised by: RMS46

Revisi ini diedit oleh Rahmat M. Samik-Ibrahim: melanjutkan perbaikan.

Revisi 1.1 01-09-2003 Revised by: RMS46

Revisi ini diedit oleh Rahmat M. Samik-Ibrahim: melakukan perbaikan struktur SGML, tanpa terlalu banyak mengubah isi buku.

Revisi 1.0 27-05-2003 Revised by: RMS46

Kompilasi ulang, serta melakukan sedikit perapihan.

Revisi 0.21.4 05-05-2003 Revised by: Kelompok 21

Perapihan berkas dan penambahan entity.

Revisi 0.21.3 29-04-2003 Revised by: Kelompok 21

Perubahan dengan menyempurnakan nama file.

Revisi 0.21.2 24-04-2003 Revised by: Kelompok 21

Merubah Kata Pengantar.

Revisi 0.21.1 21-04-2003 Revised by: Kelompok 21

Menambahkan Daftar Pustaka dan Index.

Revisi 0.21.0 26-03-2003 Revised by: Kelompok 21

Memulai membuat tugas kelompok kuliah Sistem Operasi.

Persembahan

Buku ini dipersembahkan *dari* Gabungan Kelompok Kerja 21–28 Semester Genap 2002/2003 dan 41–49 Semester Ganjil 2003/2004 Mata Kuliah Sistem Operasi, *oleh* Gabungan Kelompok Kerja 21–28 Semester Genap 2002/2003 dan 41–49 Semester Ganjil 2003/2004 Mata Kuliah Sistem Operasi, *untuk* siapa saja yang ingin mempelajari Sistem Operasi. Tim penyusun buku ini ialah sebagai berikut:

Kelompok 21 (Koordinator)

Dhani Yuliarso, Fernan, Hanny Faristin, Melanie Tedja, Paramanandana D.M., Widya Yuwanda.

Kelompok 22

Budiono Wibowo, Agus Setiawan, Baya U.H.S., Budi A. Azis Dede Junaedi, Heriyanto, Muhammad Rusdi.

Kelompok 23

Indra Agung, Ali Khumaidi, Arifullah, Baihaki A.S., Christian K.F. Daeli, Eries Nugroho, Eko Seno P., Habrar, Haris Sahlan.

Kelompok 24

Adzan Wahyu Jatmiko, Agung Pratomo, Dedy Kurniawan, Samiaji Adisasmito, Zidni Agni.

Kelompok 25

Nasrullah, Amy S. Indrasari, Ihsan Wahyu, Inge Evita Putri, Muhammad Faizal Ardhi, Muhammad Zaki Rahman, N. Rifka N. Liputo, Nelly, Nur Indah, R. Ayu P., Sita A.R.

Kelompok 26

Rakhmad Azhari, Adhe Aries, Adityo Pratomo, Aldiantoro Nugroho, Framadhan A., Pelangi, Satrio Baskoro Y.

Kelompok 27

Teuku Amir F.K., Alex Hendra Nilam, Anggraini W., Ardini Ridhatillah, R. Ferdy Ferdian, Ripta Ramelan, Suluh Legowo, Zulkiffi.

Kelompok 28

Christiono H, Arief Purnama L.K., Arman Rahmanto, Fajar, Muhammad Ichsan, Rama P. Tardan, Unedo Sanro Simon.

Kelompok 41

Ahmad Furqan S K., Aristo, Obeth M S.

Kelompok 42

Puspita K S, Retno Amelia, Susi R, Sutia H.

Kelompok 43

Agus Setiawan, Adhita Amanda, Afaf M, Alisa Dewayanti, Andung J Wicaksono, Dian Wulandari L, Gunawan, Jefri Abdullah, M Gantino, Prita I.

Kelompok 44

Arnold W, Antonius H, Irene, Theresia B, Ilham W K, Imelda T, Dessy N, Alex C.

Kelompok 45

Bima Satria T, Adrian Dwitomo, Alfa Rega M, Bobby, Diah Astuti W, Dian Kartika P, Pratiwi W, S Budianti S, Satria Graha, Siti Mawaddah, Vita Amanda.

Kelompok 46

Josef, Arief Aziz, Bimo Widhi Nugroho, Chrysta C P, Dian Maya L, Monica Lestari P, Muhammad Alaydrus, Syntia Wijaya Dharma, Wilmar Y Igenesjz, Yenni R

Kelompok 47

Bayu Putera, Enrico, Ferry Haris, Franky, Hadyan Andika, Ryan Loanda, Satriadi, Setiawan A, Siti P Wulandari, Tommy Khoerniawan, Wadiyono Valens, William Hutama.

Kelompok 48

Amir Murtako, Dwi Astuti A, M Abdushshomad E, Mauldy Laya, Novarina Azli, Raja Komkom S.

Kelompok 49 (Koordinator)

Fajran Iman Rusadi, Carroline D Puspa.

Daftar Isi

Kata Pengantar	i
1. Konsep Dasar Perangkat Komputer	1
2. Konsep Dasar Sistem Operasi	2
3. Proses dan Penjadwalan.....	3
4. Sinkronisasi dan <i>Deadlock</i>	4
5. Manajemen Memori	5
5.1. Swapping.....	5
6. Sistem Berkas.....	6
6.1. Sistem Berkas.....	6
6.1.1. Konsep Berkas	6
6.1.2. Atribut berkas	6
6.1.3. Tipe berkas	7
6.1.4. Operasi Berkas.....	7
6.1.5. Struktur Berkas	8
6.1.6. Metode Akses	9
6.2. Struktur Direktori.....	9
6.2.1. Operasi Direktori	9
6.2.2. Beberapa Struktur Direktori	10
6.2.2.1. Direktori Satu Tingkat (<i>Single Level Directory</i>).....	10
6.2.2.2. Direktori Dua Tingkat (<i>Two Level Directory</i>)	11
6.2.2.3. Direktori dengan Struktur Tree (<i>Tree- Structured Directory</i>)	11
6.2.2.4. Direktori dengan Struktur Graf Asiklik (<i>Acyclic-Structured Directory</i>).....	12
6.2.2.5. Direktori dengan Struktur Graf Umum.....	12
6.3. Konsep Mounting, Sharing, dan Proteksi.....	13
6.3.1. <i>Mounting</i>	13
6.3.1.1. <i>Mounting Overview</i>	14
6.3.1.2. Memahami <i>Mount Point</i>	14
6.3.1.3. <i>Mounting</i> Sistem Berkas, Direktori, dan Berkas	15
6.3.2. <i>Sharing</i>	16
6.3.2.1. <i>Multiple User</i>	16
6.3.2.2. <i>Remote File System</i>	16
6.3.2.3. <i>Cient-Server Model</i>	17
6.3.3. Proteksi	17
6.3.3.1. Tipe Akses.....	17
6.3.3.2. Kontrol Akses	18
6.3.3.3. Pendekatan Pengamanan Lainnya.....	19
6.4. Implementasi Sistem Berkas	20
6.4.1. Struktur Sistem Berkas	20
6.4.2. Implementasi Sistem Berkas	23
6.4.2.1. Gambaran.....	24
6.4.2.2. Partisi dan <i>Mounting</i>	25
6.4.2.3. Sistem Berkas Virtual	25
6.4.3. Implementasi Direktori.....	27
6.4.3.1. Algoritma	27

6.4.3.1.1. <i>Linear List</i>	27
6.4.3.1.2. <i>Hash Table</i>	28
6.4.3.2. Direktori pada CP/M.....	28
6.4.3.3. Direktori pada MS-DOS	29
6.4.3.4. Direktori pada UNIX	29
6.5. <i>Filesystem Hierarchy Standard</i>	30
6.5.1. Pendahuluan	30
6.5.2. Sistem Berkas	30
6.5.3. Sistem Berkas <i>Root</i>	31
6.5.3.1. Tujuan	31
6.5.3.2. Persyaratan.....	31
6.5.3.3. Pilihan Spesifik	32
6.5.3.4. <i>/bin</i> : Perintah <i>binary</i> dasar (untuk digunakan oleh semua pengguna).....	32
6.5.3.5. <i>/boot</i> : Berkas statik untuk me- <i>load boot</i>	32
6.5.3.6. <i>/dev</i> : Berkas <i>device</i>	32
6.5.3.7. <i>/etc</i> : Konfigurasi sistem <i>host-specific</i>	32
6.5.3.8. <i>/home</i> : Direktori home user	33
6.5.3.9. <i>/lib</i> : <i>Shared libraries</i> dasar dan modul <i>kernel</i>	33
6.5.3.10. <i>/lib<qual></i> : Format alternatif dari <i>shared libraries</i> dasar.....	33
6.5.3.11. <i>/media</i> : Mount point media removable	33
6.5.3.12. <i>/mnt</i> : <i>Mount point</i> untuk sistem berkas yang di- <i>mount</i> secara temporer.....	33
6.5.3.13. <i>/opt</i> : Aplikasi tambahan untuk paket perangkat lunak	33
6.5.3.14. <i>/root</i> : Direktori <i>home</i> untuk <i>user root</i>	33
6.5.3.15. <i>/sbin</i> : Binary sistem	33
6.5.3.16. <i>/srv</i> : Data untuk servis yang disediakan oleh sistem	34
6.5.3.17. <i>/tmp</i> : Berkas-berkas temporer.....	34
6.5.4. Hirarki <i>/usr</i>	34
6.5.4.1. Tujuan	34
6.5.4.2. Persyaratan.....	34
6.5.4.3. Pilihan spesifik.....	34
6.5.4.4. <i>/usr/X11R6</i> : Sistem X Window, Versi 11 Release 6.....	35
6.5.4.5. <i>/usr/bin</i> : Sebagian perintah pengguna.....	35
6.5.4.6. <i>/usr/include</i> : Direktori untuk <i>include-files</i> standar.....	35
6.5.4.7. <i>/usr/lib</i> : <i>Libraries</i> untuk pemrograman dan <i>package</i>	35
6.5.4.8. <i>/usr/lib<qual></i> : Format <i>libraries</i> alternatif.....	36
6.5.4.9. <i>/usr/local/share</i>	36
6.5.4.10. <i>/usr/sbin</i> : Sistem <i>binary</i> standar yang non-vital.....	36
6.5.4.11. <i>/usr/share</i> : Data arsitektur independen.....	36
6.5.4.12. <i>/usr/src</i> : Kode <i>source</i>	36
6.5.5. Hirarki <i>/var</i>	36
6.5.5.1. Tujuan	36
6.5.5.2. Persyaratan.....	37
6.5.5.3. Pilihan Spesifik	37
6.5.5.4. <i>/var/account</i> : <i>Log accounting</i> proses.....	37
6.5.5.5. <i>/var/cache</i> : Aplikasi data <i>cache</i>	38
6.5.5.6. <i>/var/crash</i> : <i>System crash dumps</i>	38
6.5.5.7. <i>/var/games</i> : Data variabel <i>game</i>	38
6.5.5.8. <i>/var/lib</i> : Informasi status variabel	38

6.5.5.9. /var/lock: <i>Lock</i> berkas	38
6.5.5.10. /var/log: Berkas dan direktori <i>log</i>	39
6.5.5.11. /var/mail: Berkas <i>user mailbox</i>	39
6.5.5.12. /var/opt: Data variabel untuk /opt	39
6.5.5.13. /var/run: Data variabel <i>run-time</i>	39
6.5.5.14. /var/spool: Aplikasi data <i>spool</i>	39
6.5.5.15. /var/tmp: Berkas temporer yang diletakkan di dalam <i>reboot</i> sistem	39
6.5.5.16. /var/yp: Berkas database Network Information Service (NIS)	39
6.6. Konsep Alokasi Blok Sistem Berkas	40
6.6.1. Metode Alokasi	40
6.6.1.1. <i>Contiguous Allocation</i>	40
6.6.1.2. <i>Linked Allocation</i>	42
6.6.1.3. <i>Indexed Allocation</i>	44
6.6.1.4. Kinerja Sistem Berkas.....	46
6.6.2. Manajemen Ruang Kosong	47
6.6.2.1. <i>Bit Vector</i>	47
6.6.2.2. <i>Linked List</i>	47
6.6.2.3. <i>Grouping</i>	49
6.6.2.4. <i>Counting</i>	49
6.6.3. Efisiensi dan Kinerja.....	49
6.6.3.1. Efisiensi.....	49
6.6.3.2. Kinerja.....	50
6.6.4. <i>Recovery</i>	52
6.6.4.1. Pengecekan Rutin.....	52
6.6.4.2. <i>Backup dan Restore</i>	53
6.6.5. <i>Log-Structured File System</i>	54
6.7. Rangkuman	54
6.8. Latihan	55
6.9. Rujukan	57
6.9.1. Rujukan Buku:.....	57
6.9.2. Rujukan Internet:	57
7. I/O.....	58
8. Studi Kasus: GNU/Linux	59
Daftar Pustaka	60
A. GNU Free Documentation License	61
A.1. PREAMBLE	61
A.2. APPLICABILITY AND DEFINITIONS	61
A.3. VERBATIM COPYING.....	62
A.4. COPYING IN QUANTITY	62
A.5. MODIFICATIONS.....	63
A.6. COMBINING DOCUMENTS.....	64
A.7. COLLECTIONS OF DOCUMENTS	64
A.8. AGGREGATION WITH INDEPENDENT WORKS.....	65
A.9. TRANSLATION	65
A.10. TERMINATION.....	65
A.11. FUTURE REVISIONS OF THIS LICENSE.....	65
A.12. How to use this License for your documents	66

Indeks.....67

Daftar Tabel

6-1. Direktori/link yang dibutuhkan dalam / .	31
6-2. Direktori/link yang dibutuhkan dalam / .	32
6-3. Direktori/link yang dibutuhkan dalam /usr.	34
6-4. Direktori/link yang merupakan pilihan dalam /usr.	34
6-5. Direktori/link yang dibutuhkan dalam /var.	37
6-6. Direktori/link yang dibutuhkan di dalam /var	37

Daftar Gambar

6-1. Single Level Directory	10
6-2. Two Level Directory	11
6-3. Tree-Structured Directory	11
6-4. Acyclic-Structured Directory	12
6-5. General Graph Directory	12
6-6. Mount Point	14
6-7. <i>Disk Organization</i>	20
6-8. <i>Layered File System</i>	21
6-9. <i>Schematic View of Virtual File System</i>	26
6-10. <i>A UNIX directory entry</i>	29
6-11. <i>Contiguous allocation</i>	40
6-12. <i>Linked allocation</i>	42
6-13. <i>Indexed allocation</i>	44
6-14. Ruang kosong <i>linked list</i>	47
6-15.	50
6-16.	51
6-17. Macam-macam lokasi <i>disk-caching</i>	53

Kata Pengantar

Buku ini merupakan hasil karya Gabungan Kelompok Kerja 21–28 Semester Genap 2002/2003 dan 41–49 Semester Ganjil 2003/2004 Mata Kuliah Sistem Operasi Fakultas Ilmu Komputer Universitas Indonesia (Fasilkom UI). Kelompok Kerja 21-28 mengawali penulisan buku ini dan Kelompok Kerja 41-49 melakukan revisi dan perbaikan.

Penulisan buku ini bertujuan untuk mengatasi kelangkaan bahan kuliah berbahasa Indonesia, yang dapat dimanfaatkan sebagai rujukan oleh para peserta kuliah khususnya kuliah Sistem Operasi.

Sebagai pengantar Sistem Operasi, buku ini sengaja dirancang bagi siapa saja yang berminat untuk mengetahui apa itu sebenarnya sistem operasi. Penulis mengawali buku ini dengan pengenalan Konsep Dasar Perangkat Komputer yang ditulis dimuka pada Bab 1 sebagai awalan dalam mengenal komputer lebih jauh. Pada bab ini secara singkat dibahas mengenai keseluruhan komponen yang membangun komputer. Konsep Dasar Sistem Operasi ditulis dalam Bab 2 sebagai gambaran umum sistem operasi komputer. Penjelasan lebih rinci mengenai sistem operasi diberikan pada Bab 3 sampai Bab 7. Sebagai tambahan, GNU/Linux sebagai salah satu sistem operasi yang ada saat ini, diulas pada Bab 8.

Tiap-tiap bab berisi soal-soal latihan agar pembaca dapat mengulas kembali pembahasan pada bab tersebut dan mengevaluasi sejauh mana pengetahuan mengenai bab tersebut.

Gambar dipilih sedemikian rupa sehingga dapat memberikan ilustrasi yang membantu pembaca untuk lebih memahami pembahasan.

Kami menyadari bahwa ini masih banyak kekurangannya. Silakan menyampaikan kritik/ tanggapan/ usulan anda ke <writeme03 AT yahoogroups DOT com>.

Bab 1. Konsep Dasar Perangkat Komputer

Bab 2. Konsep Dasar Sistem Operasi

Bab 3. Proses dan Penjadwalan

Bab 4. Sinkronisasi dan *Deadlock*

Bab 5. Managemen Memori

5.1. Swapping

Bab 6. Sistem Berkas

6.1. Sistem Berkas

Semua aplikasi komputer butuh menyimpan dan mengambil informasi. Ketika sebuah proses sedang berjalan, proses tersebut menyimpan sejumlah informasi yang terbatas, dibatasi oleh ukuran alamat virtual. Untuk beberapa aplikasi, ukuran ini cukup, tapi untuk lainnya terlalu kecil.

Masalah berikutnya adalah apabila proses tersebut berhenti maka informasinya hilang. Padahal ada beberapa informasi yang penting dan harus bertahan beberapa waktu bahkan selamanya.

Adapun masalah ketiga yaitu sangatlah perlu terkadang untuk lebih dari satu proses mengakses informasi secara bersamaan. Untuk memecahkan masalah ini, informasi tersebut harus bisa berdiri sendiri tanpa tergantung dengan sebuah proses.

Pada akhirnya kita memiliki masalah-masalah yang cukup signifikan dan penting untuk dicari solusinya. Pertama kita harus bisa menyimpan informasi dengan ukuran yang besar. Kedua, informasi harus tetap ketika proses berhenti. Ketiga, informasi harus bisa diakses oleh lebih dari satu proses secara bersamaan. Solusi dari ketiga masalah diatas adalah sesuatu yang disebut *file* atau berkas.

Berkas adalah sebuah unit tempat menyimpan informasi. Berkas ini dapat diakses lebih dari satu proses, dapat dibaca, dan bahkan menulis yang baru. Informasi yang disimpan dalam berkas harus persisten, dalam artian tidak hilang sewaktu proses berhenti. Berkas-berkas ini diatur oleh sistem operasi, bagaimana strukturnya, namanya, aksesnya, penggunaannya, perlindungannya, dan implementasinya. Bagian dari sistem operasi yang mengatur masalah-masalah ini disebut sistem berkas.

Untuk kebanyakan pengguna, sistem berkas adalah aspek yang paling terlihat dari sebuah sistem operasi. Dia menyediakan mekanisme untuk penyimpanan *online* dan akses ke data dan program. Sistem berkas terbagi menjadi dua bagian yang jelas; koleksi berkas (masing-masing menyimpan data yang berkaitan) dan struktur direktori (mengatur dan menyediakan informasi mengenai semua berkas yang berada di sistem). Sekarang marilah kita memperdalam konsep dari berkas tersebut.

6.1.1. Konsep Berkas

Berkas adalah sebuah koleksi informasi berkaitan yang diberi nama dan disimpan di dalam *secondary storage*. Biasanya sebuah berkas merepresentasikan data atau program. Adapun jenis-jenis dari berkas:

- *Text file*: yaitu urutan dari karakter-karakter yang diatur menjadi barisan (dan mungkin halaman)
- *Source file*: yaitu urutan dari berbagai subroutine dan fungsi yang masing-masing kemudian diatur sebagai deklarasi-deklarasi diikuti oleh pernyataan-pernyataan yang dapat dieksekusi.
- *Object file*: yaitu urutan dari byte-byte yang diatur menjadi blok-blok yang dapat dipahami oleh penghubung system.
- *Executable file*: adalah kumpulan dari bagian-bagian kode yang dapat dibawa ke memori dan dijalankan oleh loader.

6.1.2. Atribut berkas

Selain nama dan data, sebuah berkas dikaitkan dengan informasi-informasi tertentu yang juga penting untuk dilihat pengguna, seperti kapan berkas itu dibuat, ukuran berkas, dan lain-lain. Kita akan sebut informasi-informasi ekstra ini atribut. Setiap sistem mempunyai sistem atribusi yang berbeda-beda, tapi pada dasarnya memiliki atribut-atribut dasar seperti berikut ini:

- Nama: nama berkas simbolik ini adalah informasi satu-satunya yang disimpan dalam format yang dapat dibaca oleh pengguna.
- *Identifier*: tanda unik ini yang biasanya merupakan sebuah angka, mengenali berkas didalam sebuah berkas system; tidak dapat dibaca oleh pengguna.
- Tipe: informasi ini diperlukan untuk sistem-sistem yang mendukung tipe berkas yang berbeda.
- Lokasi: informasi ini adalah sebuah penunjuk pada sebuah *device* dan pada lokasi berkas pada *device* tersebut.
- Ukuran: ukuran dari sebuah berkas (dalam bytes, words, atau blocks) dan mungkin ukuran maksimum dimasukkan dalam atribut ini juga.
- Proteksi: informasi yang menentukan siapa yang bisa melakukan *read*, *write*, *execute* dan lainnya.
- Waktu dan identifikasi pengguna: informasi ini dapat disimpan untuk pembuatan berkas, modifikasi terakhir, dan penggunaan terakhir. Data-data ini dapat berguna untuk proteksi, keamanan, dan *monitoring* penggunaan.

6.1.3. Tipe berkas

Salah satu atribut dari sebuah berkas yang cukup penting adalah tipe berkas. Saat kita mendesain sebuah sistem berkas, kita perlu mempertimbangkan bagaimana operating sistem akan mengenali berkas-berkas dengan tipe yang berbeda. Apabila sistem operasi bisa mengenali, maka menjalankan berkas tersebut bukan suatu masalah. Seperti contohnya, apabila kita hendak mengeprint bentuk *binary-object* dari sebuah program, yang didapat biasanya adalah sampah, namun hal ini dapat dihindari apabila sistem operasi telah diberitahu akan adanya tipe berkas tersebut.

Cara yang paling umum untuk mengimplementasikan tipe berkas tersebut adalah dengan memasukkan tipe berkas tersebut ke dalam nama berkas. Nama berkas dibagi menjadi dua bagian. Bagian pertama adalah nama dari berkas tersebut, dan yang kedua, atau biasa disebut *extension* adalah tipe dari berkas tersebut. Kedua nama ini biasanya dipisahkan dengan tanda '.', contoh: file.txt.

6.1.4. Operasi Berkas

Fungsi dari berkas adalah untuk menyimpan data dan mengizinkan kita membacanya. Dalam proses ini ada beberapa operasi yang dapat dilakukan berkas. Adapun operasi-operasi dasar yang dilakukan berkas, yaitu:

- Membuat Berkas (*Create*):

Kita perlu dua langkah untuk membuat suatu berkas. Pertama, kita harus temukan tempat didalam sistem berkas. Kedua, sebuah entri untuk berkas yang baru harus dibuat dalam direktori. Entri dalam direktori tersebut merekam nama dari berkas dan lokasinya dalam sistem berkas.

- Menulis Berkas (*Write*):

Untuk menulis sebuah berkas, kita membuat sebuah *system call* yang menyebutkan nama berkas dan informasi yang akan ditulis kedalam berkas.

- Membaca Berkas (*Read*):

Untuk membaca sebuah berkas menggunakan sebuah *system call* yang menyebut nama berkas yang dimana dalam blok memori berikutnya dari sebuah berkas harus diposisikan.

- Memposisikan Sebuah Berkas (*Reposition*):

Direktori dicari untuk entri yang sesuai dan *current-file-position* diberi sebuah nilai. Operasi ini di dalam berkas tidak perlu melibatkan I/O, selain itu juga diketahui sebagai *file seek*.

- Menghapus Berkas (*Delete*):

Untuk menghapus sebuah berkas kita mencari dalam direktori untuk nama berkas tersebut. Setelah ditemukan, kita melepaskan semua *file space* sehingga dapat digunakan kembali oleh berkas-berkas lainnya dan menghapus entry direktori.

- Menghapus Sebagian Isi Berkas (*Truncate*):

User mungkin mau menghapus isi dari sebuah berkas, tapi menyimpan atributnya. Daripada memaksa pengguna untuk menghapus berkas tersebut dan membuatnya kembali, fungsi ini tidak akan mengganti atribut, kecuali *file length* dan mendefinisikan ulang panjang berkas tersebut menjadi nol.

Keenam operasi diatas merupakan operasi-operasi dasar dari sebuah berkas yang nantinya bisa dikombinasikan untuk membentuk operasi-operasi baru lainnya. Contohnya apabila kita ingin menyalin sebuah berkas, maka kita memakai operasi *create* untuk membuat berkas baru, *read* untuk membaca berkas yang lama, dan *write* untuk menulisnya pada berkas yang baru.

6.1.5. Struktur Berkas

Berkas dapat distruktur dalam beberapa cara. Cara yang pertama adalah sebuah urutan *bytes* yang tidak terstruktur. Akibatnya sistem operasi tidak tahu atau peduli apa yang ada dalam berkas, yang dilihatnya hanya bytes. Ini menyediakan fleksibilitas yang maksimum. User dapat menaruh apapun yang mereka mau dalam berkas, dan sistem operasi tidak membantu, tapi tidak juga menghalangi.

Cara berikutnya, adalah dengan *record sequence*. Dalam model ini, sebuah berkas adalah sebuah urutan dari rekaman-rekaman yang telah ditentukan panjangnya, masing-masing dengan beberapa struktur

internal. Artinya adalah bahwa sebuah operasi *read* membalikan sebuah rekaman dan operasi *write* menimpa atau menambahkan suatu rekaman.

Struktur berkas yang ketiga, adalah menggunakan sebuah *tree*. Dalam struktur ini sebuah berkas terdiri dari sebuah *tree* dari rekaman-rekaman tidak perlu dalam panjang yang sama, tetapi masing-masing memiliki sebuah *field key* dalam posisi yang telah ditetapkan dalam rekaman tersebut. Tree ini disort dalam *field key* dan mengizinkan pencarian yang cepat untuk sebuah *key tertentu*.

6.1.6. Metode Akses

Berkas menyimpan informasi. Apabila sedang digunakan informasi ini harus diakses dan dibaca melalui memori komputer. Informasi dalam berkas bisa diakses dengan beberapa cara. Berikut adalah beberapa caranya :

- *Sequential Access*

Akses ini merupakan yang paling sederhana dan paling umum digunakan. Informasi di dalam berkas diproses secara berurutan. Sebagai contoh, editor dan kompilator biasanya mengakses berkas dengan cara ini.

- *Direct Access*

Metode berikutnya adalah *direct access* atau bisa disebut *relative access*. Sebuah berkas dibuat dari rekaman-rekaman logical yang panjangnya sudah ditentukan, yang mengizinkan program untuk membaca dan menulis rekaman secara cepat tanpa urutan tertentu.

6.2. Struktur Direktori

Beberapa sistem komputer menyimpan banyak sekali berkas-berkas dalam disk, sehingga diperlukan suatu struktur pengorganisasian data-data agar lebih mudah diatur.

6.2.1. Operasi Direktori

Silberschatz, Galvin dan Gagne mengkategorikan operasi-operasi terhadap direktori sebagai berikut:

- Mencari Berkas

Mencari lewat struktur direktori untuk dapat menemukan entri untuk suatu berkas tertentu. berkas-berkas dengan nama yang simbolik dan mirip, mengindikasikan adanya keterkaitan diantara berkas-berkas tersebut. Oleh karena itu, tentunya perlu suatu cara untuk menemukan semua berkas yang benar-benar memenuhi kriteria khusus yang diminta.

- Membuat berkas

berkas-berkas baru perlu untuk dibuat dan ditambahkan ke dalam direktori.

- Menghapus berkas

Saat suatu berkas tidak diperlukan lagi, berkas tsb perlu dihapus dari direktori.

- Menampilkan isi direktori

Menampilkan daftar berkas-berkas yang ada di direktori, dan semua isi direktori dari berkas-berkas dalam list tsb.

- Mengubah nama berkas

Nama berkas mencerminkan isi berkas terhadap pengguna. Oleh karena itu, nama berkas harus bisa diubah-ubah ketika isi dan kegunaannya sudah berubah atau tidak sesuai lagi. Mengubah nama berkas memungkinkan posisinya berpindah dalam struktur direktori.

- Akses Sistem berkas

Mengakses tiap direktori dan tiap berkas dalam struktur direktori. Sangatlah dianjurkan untuk menyimpan isi dan stuktur dari keseluruhan sistem berkas setiap jangka waktu tertentu. Menyimpan juga bisa berarti menyalin seluruh berkas ke *magnetic tape*. Teknik ini membuat suatu cadangan salinan dari berkas tersebut jika terjadi kegagalan sistem atau jika berkas itu tidak diperlukan lagi.

Sedangkan Tanenbaum juga menambahkan hal-hal berikut sebagai operasi yang dapat dilakukan terhadap direktori tersebut:

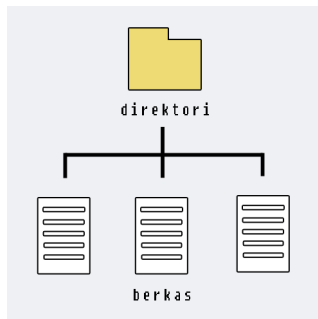
- Membuka direktori
- Menutup direktori
- Menambah direktori
- Mengubah nama direktori
- Menghubungkan berkas-berkas di direktori berbeda
- Menghapus hubungan berkas-berkas di direktori berbeda

6.2.2. Beberapa Struktur Direktori

6.2.2.1. Direktori Satu Tingkat (*Single Level Directory*)

Struktur Direktori ini merupakan struktur direktori yang paling sederhana. Semua berkas disimpan dalam direktori yang sama.

Gambar 6-1. Single Level Directory

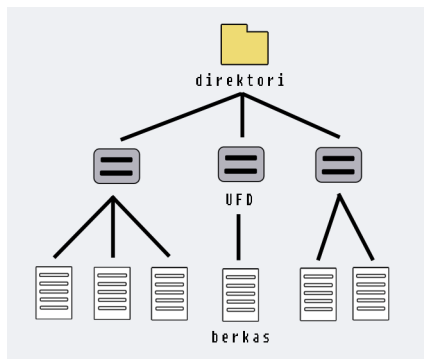


Direktori satu tingkat memiliki keterbatasan, yaitu bila berkas bertambah banyak atau bila sistem memiliki lebih dari satu pengguna. Hal ini disebabkan karena tiap berkas harus memiliki nama yang unik.

6.2.2.2. Direktori Dua Tingkat (*Two Level*) Direktori

Membuat direktori yang terpisah untuk tiap pengguna, yang disebut *User File Directory* (UFD). Ketika pengguna login, master directory berkas dipanggil. MFD memiliki indeks berdasarkan nama pengguna dan setiap entri menunjuk pada UFD pengguna tersebut. Maka, pengguna boleh memiliki nama berkas yang sama dengan berkas lain.

Gambar 6-2. Two Level Directory

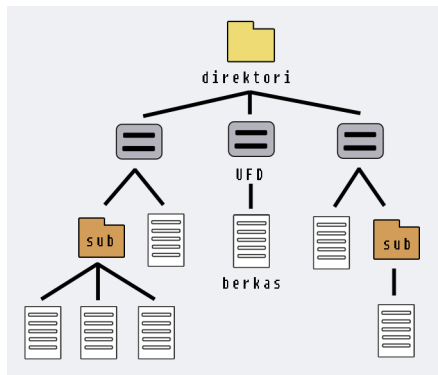


Meskipun begitu, struktur ini masih memiliki kerugian, terutama bila beberapa pengguna ingin mengerjakan tugas secara kerjasama dan ingin mengakses berkas dari salah satu pengguna lain. Beberapa sistem secara sederhana tidak mengizinkan berkas seorang pengguna diakses oleh pengguna lain.

6.2.2.3. Direktori dengan Struktur Tree (*Tree- Structured Directory*)

Dalam struktur ini, setiap pengguna dapat membuat subdirektori sendiri dan mengorganisasikan berkas-berkasnya. Dalam penggunaan normal, tiap pengguna memiliki apa yang disebut *current directory*. *Current directory* mengandung berkas-berkas yang baru-baru ini digunakan oleh pengguna.

Gambar 6-3. Tree-Structured Directory

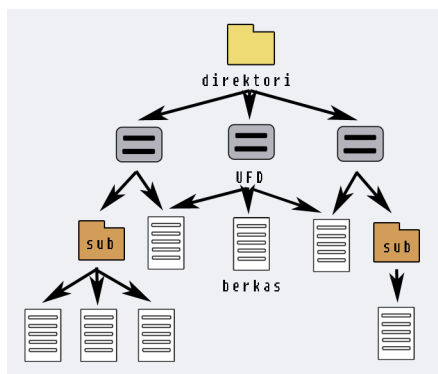


Terdapat 2 istilah, *path* relatif dan *path* mutlak. *Path* relatif adalah *path* yang dimulai dari *current directory*, sedangkan *path* mutlak adalah *path* yang dimulai dari *root directory*.

6.2.2.4. Direktori dengan Struktur Graf Asiklik (*Acyclic-Structured Directory*)

Direktori dengan struktur *tree* melarang pembagian berkas/direktori. Oleh karena itu, struktur graf asiklik memperbolehkan direktori untuk berbagi berkas atau subdirektori. Jika ada berkas yang ingin diakses oleh dua pengguna atau lebih, maka struktur ini menyediakan fasilitas *sharing*.

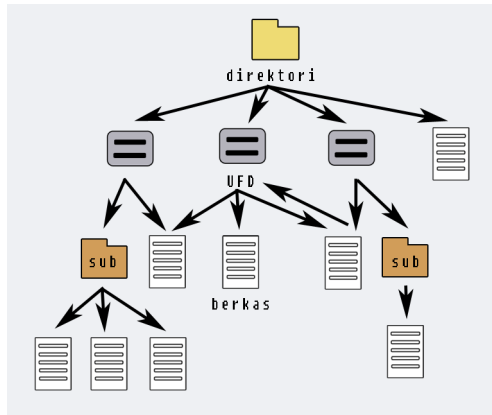
Gambar 6-4. Acyclic-Structured Directory



6.2.2.5. Direktori dengan Struktur Graf Umum

Masalah yang timbul dalam penggunaan struktur graf asiklik adalah meyakinkan apakah tidak ada siklus. Bila kita mulai dengan struktur direktori tingkat dua dan memperbolehkan pengguna untuk membuat subdirektori, maka kita akan mendapatkan struktur direktori *tree*. Sangatlah mudah untuk mempertahankan sifat pohon, akan tetapi, bila kita tambahkan sambungan pada direktori dengan struktur pohon, maka sifat pohon akan musnah dan menghasilkan struktur graf sederhana.

Gambar 6-5. General Graph Directory



Bila siklus diperbolehkan dalam direktori, tentunya kita tidak ingin mencari sebuah berkas 2 kali. Algoritma yang tidak baik akan menghasilkan *infinite loop* dan tidak pernah berakhir. Oleh karena itu diperlukan skema pengumpulan sampah (*garbage-collection scheme*).

Skema ini menyangkut memeriksa seluruh sistem berkas dengan menandai tiap berkas yang dapat diakses. Kemudian mengumpulkan apa pun yang tidak ditandai pada tempat yang kosong. Hal ini tentunya dapat menghabiskan banyak waktu.

6.3. Konsep Mounting, Sharing, dan Proteksi

6.3.1. Mounting

Mounting adalah proses mengkaitkan sebuah sistem berkas yang baru ditemukan pada sebuah piranti ke struktur direktori utama yang sedang dipakai. Piranti-piranti yang akan di-*mount* dapat berupa *cd-rom*, disket atau sebuah *zip-drive*. Tiap- tiap sistem berkas yang akan di-*mount* akan diberikan sebuah *mount point*, atau sebuah direktori dalam pohon direktori sistem Anda, yang sedang diakses.

Sistem berkas yang dideskripsikan di `/etc/fstab` (*fstab* adalah singkatan dari *filesystem tables*) biasanya akan di-*mount* saat komputer baru mulai dinyalakan, tapi bisa juga me-*mount* sistem berkas tambahan dengan menggunakan perintah:

```
mount [nama piranti]
```

atau bisa juga dengan menambahkan secara manual *mount point* ke berkas `/etc/fstab`. Daftar sistem berkas yang di-*mount* dapat dilihat kapan saja dengan menggunakan perintah *mount*. Karena *permission*nya hanya di-set *read-only* di berkas *fstab*, maka tidak perlu khawatir pengguna lain akan mencoba mengubah dan menulis *mount point* yang baru.

Seperti biasa saat ingin mengutak-atik berkas konfigurasi seperti mengubah isi berkas *fstab*, pastikan untuk membuat berkas cadangan untuk mencegah terjadinya kesalahan teknis yang bisa menyebabkan

suatu kekacauan. Kita bisa melakukannya dengan cara menyediakan sebuah disket atau *recovery-disk* dan mem-*back-up* berkas *fstab* tersebut sebelum membukanya di editor teks untuk diutak-atik.

Red Hat linux dan sistem operasi lainnya yang mirip dengan UNIX mengakses berkas dengan cara yang berbeda dari MS-DOS, Windows dan Macintosh. Di linux, segalanya disimpan di dalam sebuah lokasi yang bisa ditentukan dalam sebuah struktur data. Linux bahkan menyimpan perintah-perintah sebagai berkas. Seperti sistem operasi modern lainnya, Linux memiliki struktur *tree*, hirarki, dan organisasi direktori yang disebut sistem berkas.

Semua ruang kosong yang tersedia di *disk* diatur dalam sebuah pohon direktori tunggal. Dasar sistem ini adalah direktori *root* yang dinyatakan dengan sebuah garis miring ("/). Pada linux, isi sebuah sistem berkas dibuat nyata tersedia dengan menggabungkan sistem berkas ke dalam sebuah sistem direktori melalui sebuah proses yang disebut *mounting*.

Sistem berkas bisa di-*mount* maupun di-*umount* yang berarti sistem berkas tersebut bisa tersambung atau tidak dengan struktur pohon direktori. Perbedaannya adalah sistem berkas tersebut akan selalu di-*mount* ke direktori *root* ketika sistem sedang berjalan dan tidak dapat di-*mount*. Sistem berkas yang lain di-*mount* seperlunya, contohnya yang berisi *hard drive* berbeda dengan *floppy disk* atau CD-ROM.

6.3.1.1. *Mounting Overview*

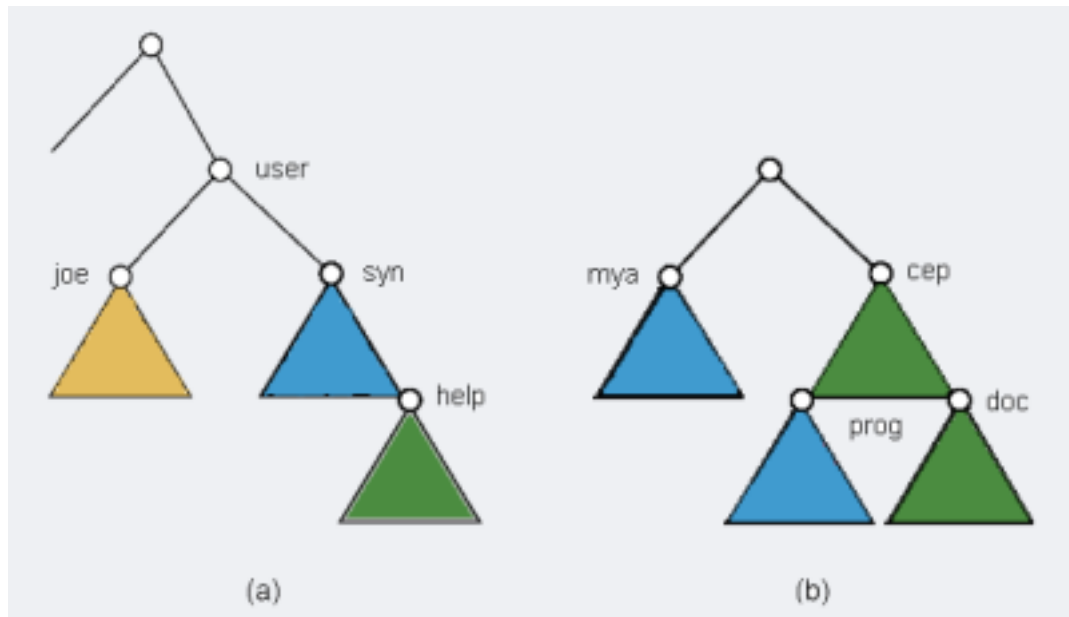
Mounting membuat sistem berkas, direktori, piranti dan berkas lainnya menjadi dapat digunakan di lokasi-lokasi tertentu, sehingga memungkinkan direktori itu menjadi bisa diakses. Perintah *mount* menginstruksikan sistem operasi untuk mengkaitkan sebuah sistem berkas ke sebuah direktori khusus.

6.3.1.2. Memahami *Mount Point*

Mount point adalah sebuah direktori dimana berkas baru menjadi dapat diakses. Untuk me-*mount* suatu sistem berkas atau direktori, titik *mount*-nya harus berupa direktori, dan untuk me-*mount* sebuah berkas, *mount point*-nya juga harus berupa sebuah berkas.

Biasanya, sebuah sistem berkas, direktori, atau sebuah berkas di-*mount* ke sebuah *mount point* yang kosong, tapi biasanya hal tersebut tidak diperlukan. Jika sebuah berkas atau direktori yang akan menjadi *mount point* berisi data, data tersebut tidak akan bisa diakses selama direktori/berkas tersebut sedang dijadikan *mount point* oleh berkas atau direktori lain. Sebagai akibatnya, berkas yang di-*mount* akan menimpa apa yang sebelumnya ada di direktori/berkas tersebut. Data asli dari direktori itu bisa diakses kembali bila proses *mounting* sudah selesai.

Gambar 6-6. Mount Point



Saat sebuah sistem berkas di-mount ke sebuah direktori, *permission* direktori *root* dari berkas yang di-mount akan mengambil alih *permission* dari *mount point*. Pengecualiannya adalah pada direktori induk akan memiliki atribut *..* (*double dot*). Agar sistem operasi bisa mengakses sistem berkas yang baru, direktori induk dari *mount point* harus tersedia.

Untuk segala perintah yang membutuhkan informasi direktori induk, pengguna harus mengubah *permission* dari direktori *mounted-over*. Kegagalan direktori *mounted-over* untuk mengabulkan *permission* bisa menyebabkan hasil yang tidak terduga, terutama karena *permission* dari direktori *mounted-over* tidak bisa terlihat (*invisible*). Kegagalan umum terjadi pada perintah `pwd`. Tanpa mengubah *permission* direktori *mounted-over*, akan timbul pesan error seperti ini:

```
pwd: Permission denied
```

Masalah ini bisa diatasi dengan mengatur agar *permission* setidaknya di-set dengan 111.

6.3.1.3. Mounting Sistem Berkas, Direktori, dan Berkas

Ada 2 tipe *mounting*: *remote mounting* dan *mounting* lokal. *Remote mounting* dilakukan dengan sistem *remote* dimana data dikirimkan melalui jalur telekomunikasi. *Remote* sistem berkas seperti Network File Systems (NFS), mengharuskan agar file diekspor dulu sebelum di-mount. *mounting* lokal dilakukan di sistem lokal.

Tiap-tiap sistem berkas berhubungan dengan piranti yang berbeda. Sebelum kita menggunakan sebuah sistem berkas, sistem berkas tersebut harus dihubungkan dengan struktur direktori yang ada (bisa *root* atau berkas yang lain yang sudah tersambung).

Sebagai contoh, kita bisa me-mount dari `/home/server/database` ke *mount point* yang dispesifikasikan sebagai `/home/user1`, `/home/user2`, and `/home/user3`:

- /home/server/database /home/user1
- /home/server/database /home/user2
- /home/server/database /home/user3

6.3.2. Sharing

Kita bisa berbagi berkas dengan pengguna lainnya yang teregistrasi. Hal pertama yang harus kita lakukan adalah menentukan dengan siapa berkas tersebut akan dibagi dan akses seperti apa yang akan diberikan kepada mereka. Berbagi berkas berguna bagi pengguna yang ingin bergabung dengan pengguna lain dan mengurangi usaha untuk mencapai sebuah hasil akhir.

6.3.2.1. Multiple User

Saat sebuah sistem operasi dibuat untuk *multiple user*, masalah berbagi berkas, penamaan berkas dan proteksi berkas menjadi sangat penting. Oleh karena itu, sistem operasi harus bisa mengakomodasikan/mengatur pembagian berkas dengan memberikan suatu struktur direktori yang membiarkan pengguna untuk saling berbagi.

Berkaitan dengan permasalahan akses berkas, kita bisa mengizinkan pengguna lain untuk melihat, mengedit atau menghapus suatu berkas. Proses mengedit berkas yang menggunakan *web-file system* berbeda dengan menggunakan aplikasi seperti Windows Explorer. Untuk mengedit sebuah file dengan *web-file system*, kita harus menduplikasi berkas tersebut dahulu dari *web-file system* ke komputer lokal, mengeditnya di komputer lokal, dan mengirim file tersebut kembali ke sistem dengan menggunakan nama berkas yang sama.

Sebagai contoh, kita bisa mengizinkan semua pengguna yang terdaftar untuk melihat berkas-berkas yang ada di direktori (tetapi mereka tidak bisa mengedit atau menghapus berkas tersebut). Contoh lainnya, kita bisa mengizinkan satu pengguna saja untuk melakukan apapun terhadap sebuah direktori dan segala isinya (ijin untuk melihat semua berkas, mengeditnya, menambah berkas bahkan menghapus isi berkas). Kita juga bisa memberikan kesempatan bagi pengguna untuk mengubah permission dan kontrol akses dari sebuah isi direktori, namun hal tersebut biasanya di luar kebiasaan, sebab seharusnya satu-satunya pengguna yang berhak mengubah permission adalah kita sendiri.

Web-file system memungkinkan kita untuk menspesifikasikan suatu akses dalam tingkatan berkas. Jadi, kita bisa mengizinkan seluruh orang untuk melihat isi dari sebuah direktori atau mengizinkan sebagian kecil pengguna saja untuk mengakses suatu direktori. Bahkan, dalam kenyataannya, kita bisa menspesifikasikan jenis akses yang berbeda dengan jumlah pengguna yang berbeda pula.

Kebanyakan pada sistem *multiple user* menerapkan konsep direktor berkas *owner/user* dan *group*.

- *Owner*: pengguna yang bisa mengubah atribut, memberikan akses, dan memiliki sebagian besar kontrol di dalam sebuah berkas atau direktori.
- *Group*: sebagian pengguna yang sedang berbagi berkas.

6.3.2.2. Remote File System

Jaringan menyebabkan berbagi data terjadi di seluruh dunia. Dalam metode implementasi pertama, yang digunakan untuk berbagi data adalah program FTP (*File Transfer Protocol*). Yang kedua terbesar adalah DFS (*Disributed File System*) yang memungkinkan remote direktori terlihat dari mesin lokal. Metode yang ketiga adalah WWW (*World Wide Web*)

FTP digunakan untuk akses anonim (mentransfer file tanpa memiliki account di sistem remote) dan akses autentik (membutuhkan ijin). WWW biasanya menggunakan akses anonim, dan DFS menggunakan akses autentik.

6.3.2.3. Client-Server Model

- server: mesin yang berisi berkas
- client: mesin yang mengakses berkas

Server bisa melayani banyak pengguna dan *client* bisa menggunakan *multiple server*. Proses identifikasi *client* biasanya sulit, dan cara yang biasa digunakan adalah melacak *IP address*, namun karena *IP address* bisa dipalsukan, cara ini menjadi kurang efektif. Ada juga yang menggunakan proses *encrypted keys*, namun hal ini lebih rumit lagi, sebab *client-server* harus menggunakan algoritma enkripsi yang sama dan pertukaran *key* yang aman.

6.3.3. Proteksi

Dalam pembahasan mengenai proteksi berkas, kita akan berbicara lebih mengenai sisi keamanan dan mekanisme bagaimana menjaga keutuhan suatu berkas dari gangguan akses luar yang tidak dikehendaki. Sebagai contoh bayangkan saja Anda berada di suatu kelompok kerja dimana masing-masing staf kerja disediakan komputer dan mereka saling terhubung membentuk suatu jaringan; sehingga setiap pekerjaan/dokumen/ berkas dapat dibagi-bagikan ke semua pengguna dalam jaringan tersebut. Misalkan lagi Anda harus menyerahkan berkas RAHASIA.txt ke atasan Anda, dalam hal ini Anda harus menjamin bahwa isi berkas tersebut tidak boleh diketahui oleh staf kerja lain apalagi sampai dimodifikasi oleh orang yang tidak berwenang. Suatu mekanisme pengamanan berkas mutlak diperlukan dengan memberikan batasan akses ke setiap pengguna terhadap berkas tertentu.

6.3.3.1. Tipe Akses

Proteksi berkaitan dengan kemampuan akses langsung ke berkas tertentu. Panjangnya, apabila suatu sistem telah menentukan secara pasti akses berkas tersebut selalu ditutup atau selalu dibebaskan ke setiap pengguna lain maka sistem tersebut tidak memerlukan suatu mekanisme proteksi. Tetapi tampaknya pengimplementasian seperti ini terlalu ekstrim dan bukan pendekatan yang baik. Kita perlu membagi akses langsung ini menjadi beberapa tipe-tipe tertentu yang dapat kita atur dan ditentukan (akses yang terkontrol).

Dalam pendekatan ini, kita mendapatkan suatu mekanisme proteksi yang dilakukan dengan cara membatasi tipe akses ke suatu berkas. Beberapa tipe akses tersebut antara lain:

- Read/Baca: membaca berkas
- Write/Tulis: menulis berkas
- Execute/Eksekusi: memasukkan berkas ke memori dan dieksekusi
- Append/Sisip: menulis informasi baru pada baris akhir berkas
- Delete/Hapus: menghapus berkas
- List/Daftar: mendaftar nama dan atribut berkas

Operasi lain seperti *rename*, *copying*, atau *editing* yang mungkin terdapat di beberapa sistem merupakan gabungan dari beberapa tipe kontrol akses diatas. Sebagai contoh, menyalin sebuah berkas dikerjakan sebagai runtutan permintaan baca dari pengguna. Sehingga dalam hal ini, seorang pengguna yang memiliki kontrol akses *read* dapat pula meng-*copy*, mencetak dan sebagainya.

6.3.3.2. Kontrol Akses

Pendekatan yang paling umum dipakai dalam mengatasi masalah proteksi berkas adalah dengan membiarkan akses ke berkas ditentukan langsung oleh pengguna (dalam hal ini pemilik/pembuat berkas itu). Sang pemilik bebas menentukan tipe akses apa yang diperbolehkan untuk pengguna lain. Hal ini dapat dilakukan dengan menghubungkan setiap berkas atau direktori dengan suatu daftar kontrol-akses (*Access-Control Lists/ACL*) yang berisi nama pengguna dan tipe akses apa yang diberikan kepada pengguna tersebut.

Sebagai contoh dalam suatu sistem VMS, untuk melihat daftar direktori berikut daftar kontrol-akses, ketik perintah "DIR/SECURITY", atau "DIR/SEC". Salah satu keluaran perintah itu adalah daftar seperti berikut ini:

```
WWW-HOME.DIR;1          [HMC2000, WWART]          (RW, RWED,,E)
  (IDENTIFIER=WWW_SERVER_ACCESS, OPTIONS=DEFAULT, ACCESS=READ)
  (IDENTIFIER=WWW_SERVER_ACCESS, ACCESS=READ)
```

Baris pertama menunjukkan nama berkas tersebut WWW-HOME.DIR kemudian disebelahnya nama grup pemilik HMC2000 dan nama pengguna WWART diikuti dengan sekelompok tipe akses RW, RWED,,E (R=Baca, W=Tulis, E=Eksekusi, D=Hapus). Dua baris dibawahnya itulah yang disebut daftar kontrol-akses. Satu-satu baris disebut sebagai masukan kontrol-akses (*Access Control Entry/ACE*) dan terdiri dari 3 bagian. Bagian pertama disebut sebagai IDENTIFIER/Identifikasi, menyatakan nama grup atau nama pengguna (seperti [HMC2000, WWART]) atau akses khusus (seperti WWW_SERVER_ACCESS). Bagian kedua merupakan daftar OPTIONS/Pilihan-pilihan. Dan terakhir adalah daftar ijin ACCESS/akses, seperti *read* atau *execute*, yang diberikan kepada siapa saja yang mengacu pada bagian Identifikasi.

Cara kerjanya: apabila seorang pengguna meminta akses ke suatu berkas/direktori, sistem operasi akan memeriksa ke daftar kontrol-akses apakah nama pengguna itu tercantum dalam daftar tersebut. Apabila benar terdaftar, permintaan akses akan diberikan dan sebaliknya bila tidak, permintaan akses akan ditolak.

Pendekatan ini memiliki keuntungan karena penggunaan metodologi akses yang kompleks sehingga sulit ditembus sembarangan. Masalah utamanya adalah ukuran dari daftar akses tersebut. Bayangkan apabila

kita mengizinkan semua orang boleh membaca berkas tersebut, kita harus mendaftar semua nama pengguna disertai ijin akses baca mereka. Lebih jauh lagi, teknik ini memiliki dua konsekuensi yang tidak diinginkan:

- Pembuatan daftar semacam itu merupakan pekerjaan yang melelahkan dan tidak efektif.
- Entri direktori yang sebelumnya memiliki ukuran tetap, menjadi ukuran yang bisa berubah-ubah, mengakibatkan lebih rumitnya manajemen ruang kosong.

Masalah ini bisa diselesaikan dengan penggunaan daftar akses yang telah disederhanakan.

Untuk menyederhanakan ukuran daftar kontrol akses, banyak sistem menggunakan tiga klasifikasi pengguna sebagai berikut:

- *Owner*: pengguna yang telah membuat berkas tersebut.
- *Group* : sekelompok pengguna yang saling berbagi berkas dan membutuhkan akses yang sama.
- *Universe*: keseluruhan pengguna.

Pendekatan yang dipakai belum lama ini adalah dengan mengkombinasikan daftar kontrol-akses dengan konsep kontrol- akses pemilik, grup dan semesta yang telah dijabarkan diatas. Sebagai contoh, Solaris 2.6 dan versi berikutnya menggunakan tiga klasifikasi kontrol-akses sebagai pilihan umum, tetapi juga menambahkan secara khusus daftar kontrol-akses terhadap berkas/direktori tertentu sehingga semakin baik sistem proteksi berkasnya.

Contoh lain yaitu sistem UNIX dimana kontrol-aksesnya dinyatakan dalam 3 bagian. Masing-masing bagian merupakan klasifikasi pengguna (yi.pemilik, grup dan semesta). Setiap bagian kemudian dibagi lagi menjadi 3 bit tipe akses *-rwx*, dimana *r* mengontrol akses baca, *w* mengontrol akses tulis dan *x* mengontrol eksekusi. Dalam pendekatan ini, 9 bit diperlukan untuk merekam seluruh informasi proteksi berkas.

Berikut adalah keluaran dari perintah "ls -al" di sistem UNIX:

```
-rwxr-x--- 1 david karyawan 12210 Nov 14 20:12 laporan.txt
```

Baris di atas menyatakan bahwa berkas laporan.txt memiliki akses penuh terhadap pemilik berkas (yi.david), grupnya hanya dapat membaca dan mengeksekusi, sedang lainnya tidak memiliki akses sama sekali.

6.3.3.3. Pendekatan Pengamanan Lainnya

Salah satu pendekatan lain terhadap masalah proteksi adalah dengan memberikan sebuah kata kunci (*password*) ke setiap berkas. Jika kata-kata kunci tersebut dipilih secara acak dan sering diganti, pendekatan ini sangatlah efektif sebab membatasi akses ke suatu berkas hanya diperuntukkan bagi pengguna yang mengetahui kata kunci tersebut.

Meskipun demikian, pendekatan ini memiliki beberapa kekurangan, diantaranya:

- Kata kunci yang perlu diingat oleh pengguna akan semakin banyak, sehingga membuatnya menjadi tidak praktis.

- Jika hanya satu kata kunci yang digunakan di semua berkas, maka jika sekali kata kunci itu diketahui oleh orang lain, orang tersebut dapat dengan mudah mengakses semua berkas lainnya. Beberapa sistem (contoh: TOPS-20) memungkinkan seorang pengguna untuk memasukkan sebuah kata kunci dengan suatu subdirektori untuk menghadapi masalah ini, bukan dengan satu berkas tertentu.
- Umumnya, hanya satu kata kunci yang diasosiasikan dengan semua berkas lain. Sehingga, pengamanan hanya menjadi semua-atau-tidak sama sekali. Untuk mendukung pengamanan pada tingkat yang lebih mendetail, kita harus menggunakan banyak kata kunci.

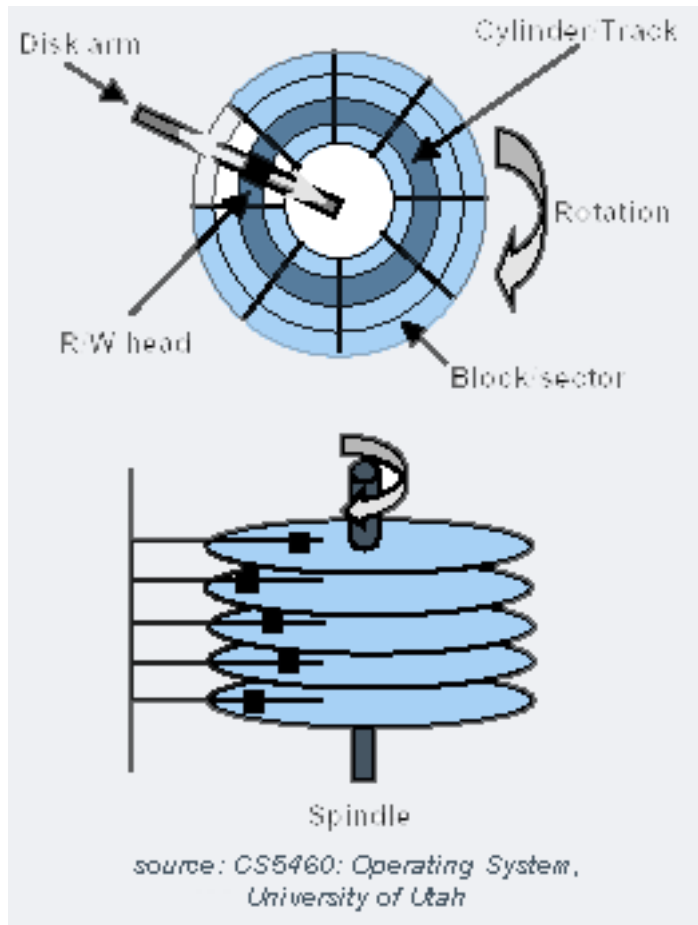
6.4. Implementasi Sistem Berkas

6.4.1. Struktur Sistem Berkas

Disk yang merupakan tempat terdapatnya sistem berkas menyediakan sebagian besar tempat penyimpanan dimana sistem berkas akan dikelola. *Disk* memiliki dua karakteristik penting yang menjadikan *disk* sebagai media yang tepat untuk menyimpan berbagai macam berkas, yaitu:

- Data dapat ditulis ulang di *disk* tersebut, hal ini memungkinkan untuk membaca, memodifikasi, dan menulis di *disk* tersebut.
- Dapat diakses langsung ke setiap *block* di *disk*. Hal ini memudahkan untuk mengakses setiap berkas baik secara berurut maupun tidak berurut, dan berpindah dari satu berkas ke berkas lain dengan hanya mengangkat *head disk* dan menunggu *disk* berputar.

Gambar 6-7. Disk Organization

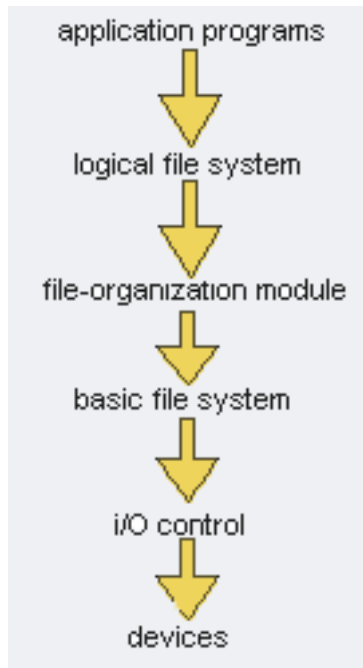


Untuk meningkatkan efisiensi *I/O*, pengiriman data antara *memory* dan *disk* dilakukan dalam setiap *block*. Setiap *block* merupakan satu atau lebih sektor. Setiap *disk* memiliki ukuran yang berbeda-beda, biasanya berukuran 512 *bytes*.

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 *secondary storage*.

Sistem berkas dari sistem operasi yang sudah *modern* diimplementasikan dengan menggunakan struktur berlapis. Keuntungan struktur berlapis ini adalah fleksibilitas yang dimilikinya. Penggunaan dari struktur berlapis ini memungkinkan adanya implementasi yang lebih dari satu secara bersamaan, terutama pada *I/O Control* dan tingkatan organisasi berkas. Hal ini memungkinkan untuk mendukung lebih dari satu implementasi sistem berkas.

Gambar 6-8. Layered File System



Lapisan struktur sistem berkas menghubungkan antara perangkat keras dengan aplikasi program yang ada, 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. *Input* didalamnya berisikan perintah tingkat tinggi seperti "ambil block 133", sedangkan *output*-nya adalah perintah tingkat rendah, instruksi spesifik perangkat keras yang digunakan oleh *controller* perangkat keras.
- *Basic file system*, diperlukan untuk mengeluarkan perintah *generic* ke *device driver* untuk *read* dan *write* pada suatu *block* 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 *pengguna* ingin menulis berkas ke dalam *disk*. Di dalam *File-organization module* juga terdapat *free-space manager*.
- *Logical file-system*, tingkat ini berisi informasi tentang simbol nama berkas, struktur dari direktori, dan proteksi dan sekuriti dari berkas tersebut. Sebuah *File Control Block (FCB)* menyimpan informasi tentang berkas, termasuk kepemilikan, izin dan lokasi isi berkas.

Di bawah ini merupakan contoh dari kerja struktur berlapis ini ketika suatu program mau membaca informasi dari *disk*. Urutan langkahnya:

1. *Application program* memanggil sistem berkas dengan *system call*.

Contoh: *read (fd, input, 1024)* akan membaca *section* sebesar 1 Kb dari *disk* dan menempatkannya ke variabel *input*.

2. Diteruskan ke *system call interface*.

System call merupakan *software interrupt*. Jadi, *interrupt handler* sistem operasi akan memeriksa apakah *system call* yang menginterupsi. *Interrupt handler* ini akan memutuskan bagian dari sistem operasi yang bertanggung jawab untuk menangani *system call*. *Interrupt handler* akan meneruskan *system call*.

3. Diteruskan ke *logical file system*.

Memasuki lapisan sistem berkas. Lapisan ini menyediakan *system call*, operasi yang akan dilakukan dan jenis berkas. Yang perlu ditentukan selanjutnya adalah *file organization module* yang akan meneruskan permintaan ini. *File organization module* yang akan digunakan tergantung dari tipe sistem berkas dari berkas yang diminta.

Contoh: Misalkan kita menggunakan LINUX dan berkas yang diminta ada di Windows 95. Lapisan *logical file system* akan meneruskan permintaan ke *file organization module* dari Windows 95.

4. Diteruskan ke *file organization module*.

File organization module yang mengetahui pengaturan (organisasi) direktori dan berkas pada *disk*. Sistem berkas yang berbeda memiliki organisasi yang berbeda. Windows 95 menggunakan VFAT-32. Windows NT menggunakan format NTFS. Linux menggunakan EXT2. Sistem operasi yang paling *modern* memiliki beberapa *file organization module* sehingga bisa membaca format yang berbeda.

Pada contoh di atas, *logical file system* telah meneruskan permintaan ke *file organization module* VFAT32. Modul ini menterjemahkan nama berkas yang ingin dibaca ke lokasi fisik yang biasanya terdiri dari *disk interface, disk drive, surface, cylinder, track, sector*.

5. Diteruskan ke *basic file system*.

Dengan adanya lokasi fisik, kita bisa memberikan perintah ke piranti keras yang dibutuhkan. Hal ini merupakan tanggungjawab *basic file system*. *Basic file system* ini juga memiliki kemampuan tambahan seperti *buffering* dan *caching*.

Contoh: Sektor tertentu yang dipakai untuk memenuhi permintaan mungkin saja berada dalam *buffers* atau *caches* yang diatur oleh *basic file system*. Jika terjadi hal seperti ini, maka informasi akan didapatkan secara otomatis tanpa perlu membaca lagi dari *disk*.

6. *I/O Control*

Tingkatan yang paling rendah ini yang memiliki cara untuk memerintah/memberitahu piranti keras yang diperlukan.

6.4.2. Implementasi Sistem Berkas

6.4.2.1. Gambaran

Untuk mengimplementasikan suatu sistem berkas biasanya digunakan beberapa struktur *on-disk* dan *in-memory*. Struktur ini bervariasi tergantung pada sistem operasi dan sistem berkas, tetapi beberapa prinsip dasar harus tetap diterapkan. Pada struktur *on-disk*, sistem berkas mengandung informasi tentang bagaimana mem-*boot* sistem operasi yang disimpan, jumlah *blocks*, jumlah dan lokasi *block* yang masih kosong, struktur direktori, dan berkas individu.

Struktur *on-disk*:

- *Boot Control Block*

Informasi yang digunakan untuk menjalankan mesin mulai dari partisi yang diinginkan untuk menjalankan mesin mulai dari partisi yang diinginkan. Dalam UPS disebut *boot block*. Dalam NTFS disebut *partition boot sector*.

- *Partition Block Control*

Spesifikasi atau detail-detail dari partisi (jumlah *block* dalam partisi, ukuran *block*, ukuran *block*, dsb). Dalam UPS disebut *superblock*. Dalam NTFS disebut tabel *master file*.

- Struktur direktori

Mengatur berkas-berkas

- File Control Block (FCB)

Detail-detail berkas yang spesifik. Di UPS disebut *inode*. Di NTFS, informasi ini disimpan di dalam tabel *Master File*.

Struktur *in-memory*:

- Tabel Partisi *in-memory*

Informasi tentang partisi yang di-*mount*.

- Struktur Direktori *in-memory*

Menyimpan informasi direktori tentang direktori yang paling sering diakses.

- Tabel *system-wide open-file*

- menyimpan *open count* (informasi jumlah proses yang membuka berkas tsb)
- menyimpan atribut berkas (pemilik, proteksi, waktu akses, dsb), dan lokasi *file blocks*.

- Tabel ini digunakan bersama-sama oleh seluruh proses.
- Tabel *per-process open-file*
 - menyimpan *pointer* ke *entry* yang benar dalam tabel *open-file*
 - menyimpan posisi *pointer* pada saat itu dalam berkas.
 - modus akses

Untuk membuat suatu berkas baru, program aplikasi memanggil *logical file system*. *Logical file system* mengetahui format dari struktur direktori. Untuk membuat berkas baru, *logical file system* akan mengalokasikan FCB, membaca direktori yang benar ke memori, memperbaharui dengan nama berkas dan FCB yang baru dan menuliskannya kembali ke dalam *disk*.

Beberapa sistem operasi, termasuk UNIX, memperlakukan berkas sebagai direktori. Sistem operasi Windows NT mengimplementasi beberapa *system calls* untuk berkas dan direktori. Windows NT memperlakukan direktori sebagai sebuah kesatuan yang berbeda dengan berkas. *Logical file system* bisa memanggil *file-organization module* untuk memetakan direktori *I/O* ke *disk-block numbers*, yang dikirimkan ke sistem berkas dasar dan *I/O control system*. *File-organization module* juga mengalokasikan *blocks* untuk penyimpanan data-data berkas.

Setelah berkas selesai dibuat, mula-mula harus dibuka terlebih dahulu. Perintah *open* dikirim nama berkas ke sistem berkas. Ketika sebuah berkas dibuka, struktur direktori mencari nama berkas yang diinginkan. Ketika berkas ditemukan, FCB disalin ke ke tabel *system-wide open-file* pada memori. Tabel ini juga mempunyai *entry* untuk jumlah proses yang membuka berkas tersebut.

Selanjutnya, *entry* dibuat di tabel *per-process open-file* dengan penunjuk ke *entry* di dalam tabel *system-wide open-file*. Seluruh operasi pada berkas akan diarahkan melalui penunjuk ini.

6.4.2.2. Partisi dan *Mounting*

Setiap partisi bisa merupakan *raw* atau *cooked*. *Raw* adalah partisi yang tidak memiliki sistem berkas dan *cooked* sebaliknya. *Raw disk* digunakan jika tidak ada sistem berkas yang tepat. *Raw disk* juga dapat menyimpan informasi yang dibutuhkan oleh sistem *disk RAID* dan *database* kecil yang menyimpan informasi konfigurasi RAID.

Informasi *boot* bisa disimpan di partisi yang berbeda. Semuanya mempunyai formatnya masing-masing karena pada saat *boot*, sistem tidak punya sistem berkas dari perangkat keras dan tidak bisa memahami sistem berkas.

Root partition yang mengandung *kernel* sistem operasi dan sistem berkas yang lain, di-*mount* saat *boot*. Partisi yang lain di-*mount* secara otomatis atau manual (tergantung sistem operasi). Sistem operasi menyimpan dalam struktur tabel *mount* dimana sistem berkas di-*mount* dan tipe dari sistem berkas.

Pada UNIX, sistem berkas dapat di-*mount* di direktori manapun. Ini diimplementasikan dengan mengatur *flag* di salinan *in-memory* dari tipe direktori itu. *Flag* itu mengindikasikan bahwa direktori adalah puncak *mount*.

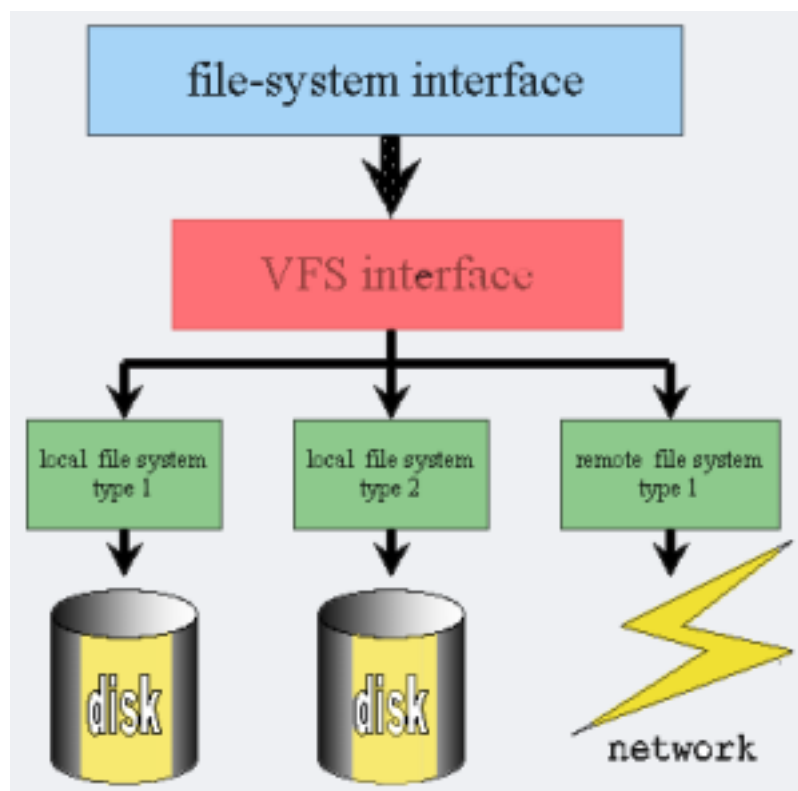
6.4.2.3. Sistem Berkas Virtual

Suatu direktori biasanya menyimpan beberapa berkas dengan tipe-tipe yang berbeda. Sistem operasi harus bisa menyatukan berkas-berkas berbeda itu di dalam suatu struktur direktori. Untuk menyatukan berkas-berkas tersebut digunakan metode implementasi beberapa tipe sistem berkas dengan menulis di direktori dan *file routine* untuk setiap tipe.

Sistem operasi pada umumnya, termasuk UNIX, menggunakan teknik *object-oriented* untuk menyederhakan, mengorganisir dan mengelompokkannya sesuai dengan implementasinya. Penggunaan metode ini memungkinkan berkas-berkas yang berbeda jenisnya diimplementasikan dalam struktur yang sama.

Implementasi spesifiknya menggunakan struktur data dan prosedur untuk mengisolasi fungsi dasar dari *system call*.

Gambar 6-9. Schematic View of Virtual File System



Implementasi sistem berkas terdiri dari 3 lapisan utama:

- *Interface* sistem berkas: perintah *open*, *read*, *write*, *close* dan *file descriptor*.
- *Virtual File System* (VFS)

Virtual file system adalah suatu lapisan perangkat lunak dalam kernel yang menyediakan *interface* sistem berkas untuk program *userspace*. VFS juga menyediakan suatu abstraksi dalam kernel yang

mengijinkan implementasi sistem berkas yang berbeda untuk muncul.

VFS ini memiliki 2 fungsi yang penting yaitu:

- Memisahkan operasi berkas *generic* dari implementasinya dengan mendefinisikan VFS *interface* yang masih baru.
 - VFS didasarkan pada struktur *file-representation* yang dinamakan *vnode*, yang terdiri dari *designator* numerik untuk berkas unik *network-wide*.
- Sistem berkas lokal dan sistem berkas *remote* untuk jaringan.

6.4.3. Implementasi Direktori

Sebelum sebuah berkas dapat dibaca, berkas tersebut harus dibuka terlebih dahulu. Saat berkas tersebut dibuka, sistem operasi menggunakan *path name* yang dimasukkan oleh pengguna untuk mengalokasikan direktori *entry* yang menyediakan informasi yang dibutuhkan untuk menemukan *block disk* tempat berkas itu berada. Tergantung dari sistem tersebut, informasi ini bisa berupa alamat *disk* dari berkas yang bersangkutan (*contiguous allocation*), nomor dari *block* yang pertama (kedua skema *linked list*), atau nomor dari *inode*. Dalam semua kasus, fungsi utama dari direktori *entry* adalah untuk memetakan nama ASCII dari berkas yang bersangkutan kepada informasi yang dibutuhkan untuk mengalokasikan data.

Masalah berikutnya yang kemudian muncul adalah dimana atribut yang dimaksud akan disimpan. Kemungkinan paling nyata adalah menyimpan secara langsung di dalam direktori *entry*, dimana kebanyakan sistem menggunakannya. Untuk sistem yang menggunakan *inodes*, kemungkinan lain adalah menyimpan atribut ke dalam *inode*, selain dari direktori *entry*. Cara yang terakhir ini mempunyai keuntungan lebih dibandingkan menyimpan dalam direktori *entry*.

Cara pengalokasian direktori dan pengaturan direktori dapat meningkatkan efisiensi, performa dan kehandalan. Ada beberapa macam algoritma yang dapat digunakan.

6.4.3.1. Algoritma

6.4.3.1.1. Linear List

Metode paling sederhana. Menggunakan nama berkas dengan penunjuk ke data *block*.

Proses:

- Mencari (tidak ada nama berkas yang sama)
- Menambah berkas baru pada akhir direktori
- Menghapus (mencari berkas dalam direktori dan melepaskan tempat yang dialokasikan)

Penggunaan suatu berkas:

Memberi tanda atau menambahkan pada daftar direktori bebas.

Kelemahan:

Pencarian secara *linier* (*linier search*) untuk mencari sebuah berkas, sehingga implementasi sangat lambat saat mengakses dan mengeksekusi berkas.

Solusi:

Linked list dan *Software Cache*

6.4.3.1.2. Hash Table

Linear List menyimpan *entry* direktori, tetapi struktur data *hash* juga digunakan.

Proses:

Hash table mengambil nilai yang dihitung dari nama berkas dan mengembalikan sebuah penunjuk ke nama berkas yang ada di *linier list*.

Kelemahan:

- Ukuran tetap:
- Adanya ketergantungan fungsi *hash* dengan ukuran *hash table*

Alternatif:

Chained-overflow hash table yaitu setiap *hash table* mempunyai *linked list* dari nilai individual dan *crash* dapat diatasi dengan menambah tempat pada *linked list* tersebut. Namun penambahan ini dapat memperlambat.

6.4.3.2. Direktori pada CP/M

Direktori pada CP/M merupakan direktori *entry* yang mencakup nomor *block disk* untuk setiap berkas. Contoh direktori ini (Golden dan Pechura, 1986), berupa satu direktori saja. Jadi, Semua sistem berkas harus melihat nama berkas dan mencari dalam direktori satu-satunya ini.

Direktori ini terdiri dari 3 bagian yaitu:

- *User Code*

Merupakan bagian yang menetapkan *track* dari *user* mana yang mempunyai berkas yang bersangkutan, saat melakukan pencarian, hanya *entry* tersebut yang menuju kepada *logged-in user* yang bersangkutan. Dua bagian berikutnya terdiri dari nama berkas dan ekstensi dari berkas.

- *Extent*

Bagian ini diperlukan oleh berkas karena berkas yang berukuran lebih dari 16 *block* menempati direktori *entry* yang banyak. Bagian ini digunakan untuk memberitahukan *entry* mana yang datang pertama, kedua, dan seterusnya.

- *Block Count*

Bagian ini memberitahukan seberapa banyak dari ke-enambelas *block disk* potensial, sedang digunakan. Enambelas bagian akhir berisi nomor *block disk* yang bersangkutan. Bagian *block* yang terakhir bisa saja penuh, jadi sistem tidak bisa menentukan kapasitas pasti dari berkas sampai ke *byte* yang terakhir.

Saat CP/M menemukan entry, CP/M juga memakai nomor *block disk*, saat berkas disimpan dalam direktori *entry*, dan juga semua atributnya. Jika berkas menggunakan *block disk* lebih dari satu *entry*, berkas dialokasikan dalam direktori yang ditambahkan.

6.4.3.3. Direktori pada MS-DOS

Merupakan sistem dengan *tree hierarchy directory*. Mempunyai panjang 32 *bytes*, yang mencakup nama berkas, atribut, dan nomor dari *block disk* yang pertama. Nomor dari *block disk* yang pertama digunakan sebagai indeks dari tabel MS-DOS direktori *entry*. Dengan sistem rantai, semua *block* dapat ditemukan.

Dalam MS-DOS, direktori bisa berisi direktori lain, tergantung dari hirarki sistem berkas. Dalam MS-DOS, program aplikasi yang berbeda bisa dimulai oleh setiap program dengan membuat direktori dalam direktori *root*, dan menempatkan semua berkas yang bersangkutan di dalam sana. Jadi antar aplikasi yang berbeda tidak dapat terjadi konflik.

6.4.3.4. Direktori pada UNIX

Struktur direktori yang digunakan dalam UNIX adalah struktur direktori tradisional. Seperti yang terdapat dalam gambar direktori *entry* dalam UNIX, setiap *entry* berisi nama berkas dan nomor *inode* yang bersangkutan. Semua informasi dari tipe, kapasitas, waktu dan kepemilikan, serta *block disk* yang berisi *inode*. Sistem UNIX terkadang mempunyai penampakan yang berbeda, tetapi pada beberapa kasus, direktori *entry* biasanya hanya *string* ASCII dan nomor *inode*.

Gambar 6-10. A UNIX directory entry



Saat berkas dibuka, sistem berkas harus mengambil nama berkas dan mengalokasikan *block disk* yang bersangkutan, sebagai contoh, nama *path* `/usr/ast/mbox` dicari, dan kita menggunakan UNIX sebagai contoh, tetapi algoritma yang digunakan secara dasar sama dengan semua hirarki sistem direktori sistem.

Pertama, sistem berkas mengalokasikan direktori *root*. Dalam UNIX *inode* yang bersangkutan ditempatkan dalam tempat yang sudah tertentu dalam *disk*. Kemudian, UNIX melihat komponen pertama dari *path*, *usr* dalam direktori *root* menemukan nomor *inode* dari direktori */usr*. Mengalokasikan sebuah nomor *inode* adalah secara *straight-forward*, sejak setiap *inode* mempunyai lokasi yang tetap dalam *disk*. Dari *inode* ini, sistem mengalokasikan direktori untuk */usr* dan melihat komponen berikutnya, dst. Saat dia menemukan *entry* untuk *ast*, dia sudah mempunyai *inode* untuk direktori */ust/ast*. Dari *inode* ini, dia dapat menemukan direktorinya dan melihat *mbox*. *Inode* untuk berkas ini kemudian dibaca ke dalam memori dan disimpan disana sampai berkas tersebut ditutup.

Nama *path* dilihat dengan cara yang relatif sama dengan yang absolut. Dimulai dari direktori yang bekerja sebagai pengganti *root directory*. Setiap direktori mempunyai *entry* untuk *.* dan *..* yang dimasukkan ke dalam saat direktori dibuat. *Entry .* mempunyai nomor *inode* yang menunjuk ke direktori di atasnya / orangtua (*parent*), *.* kemudian melihat *./dick/prog.c* hanya melihat tanda *..* dalam direktori yang bekerja, dengan menemukan nomor *inode* dalam direktori di atasnya / *parent* dan mencari direktori *disk*. Tidak ada mekanisme spesial yang dibutuhkan untuk mengatasi masalah nama ini. Sejauh masih di dalam sistem direktori, mereka hanya merupakan ASCII string yang biasa.

6.5. Filesystem Hierarchy Standard

6.5.1. Pendahuluan

Filesystem Hierarchy Standard (FHS) adalah standar yang digunakan oleh perangkat lunak dan pengguna untuk mengetahui lokasi dari berkas atau direktori yang berada pada komputer. Hal ini dilakukan dengan cara menetapkan prinsip-prinsip dasar pada setiap daerah pada sistem berkas, menetapkan berkas dan direktori minimum yang dibutuhkan, mengatur banyaknya *exceptions* dan mengatur kasus yang sebelumnya pernah mengalami konflik secara spesifik.

Dokumen FHS ini digunakan oleh pembuat perangkat lunak untuk menciptakan suatu aplikasi yang *compliant* dengan FHS. Selain itu, dokumen ini juga digunakan oleh para pembuat sistem operasi untuk menyediakan sistem yang *compliant* dengan FHS.

Komponen dari nama berkas yang dapat berubah-ubah, akan diapit oleh tanda *<* dan *>*, sedangkan komponen yang bersifat pilihan, akan diapit oleh tanda *"*[*"* dan *"]* dan dapat dikombinasi dengan *'<'* dan *'>'*. Sebagai contoh, jika nama berkas diperbolehkan untuk menggunakan atau tidak menggunakan *extension*, akan ditulis sebagai *<nama berkas>[.<extension>]*. Sedangkan, variabel *substring* dari nama direktori atau nama berkas akan ditulis sebagai *"*"*.

6.5.2. Sistem Berkas

Terdapat dua perbedaan yang saling independen dalam berkas, yaitu *shareable* vs. *unshareable* dan *variable* vs. *static*. Secara umum, berkas-berkas yang memiliki perbedaan seperti di atas sebaiknya diletakkan dalam direktori yang berbeda. Hal ini mempermudah penyimpanan berkas dengan karakteristik yang berbeda dalam sistem berkas yang berbeda.

Berkas *shareable* adalah berkas yang disimpan di satu komputer, namun masih dapat digunakan oleh komputer lainnya. Sedangkan berkas *unshareable* tidak dapat digunakan bersama-sama antar komputer yang satu dan lainnya.

Berkas *static* meliputi berkas *binary*, *library*, dokumentasi dan berkas-berkas lain yang tidak dapat diubah tanpa intervensi administrator sistem. Sedangkan, berkas *variable* adalah semua berkas yang bukan merupakan berkas *static*.

6.5.3. Sistem Berkas *Root*

6.5.3.1. Tujuan

Isi dari sistem berkas *root* harus memadai untuk melakukan operasi *boot*, *restore*, *recover* dan atau perbaikan pada sistem.

Untuk melakukan operasi *boot* pada sistem, perlu dilakukan hal-hal untuk *mounting* sistem berkas lain. Hal ini meliputi konfigurasi data, informasi *boot loader* dan keperluan-keperluan lain yang mengatur *start-up* data.

Untuk melakukan *recovery* dan atau perbaikan dari sistem, hal-hal yang dibutuhkan untuk mendiagnosa dan memulihkan sistem yang rusak harus diletakkan dalam sistem berkas *root*.

Untuk *restore* suatu sistem, hal-hal yang dibutuhkan untuk *back-up* sistem, seperti *floppy disk*, *tape*, dsb, harus berada dalam sistem berkas *root*.

Aplikasi pada komputer tidak diperbolehkan untuk membuat berkas atau subdirektori di dalam direktori *root*, karena untuk meningkatkan *performance* dan keamanan, partisi *root* sebaiknya dibuat seminimum mungkin. Selain itu, lokasi-lokasi lain dalam FHS menyediakan fleksibilitas yang lebih dari cukup untuk *package* manapun.

6.5.3.2. Persyaratan

Terdapat beberapa direktori yang merupakan persyaratan dari sistem berkas *root*. Setiap direktori akan dibahas dalam sub-bagian di bawah. */usr* dan */var* akan dibahas lebih mendetail karena direktori tersebut sangat kompleks.

Tabel 6-1. Direktori/link yang dibutuhkan dalam */*.

Direktori	Keterangan
<i>bin</i>	Instruksi dasar <i>binary</i>
<i>boot</i>	Berkas statik untuk me- <i>load boot</i>
<i>dev</i>	Berkas <i>device</i>
<i>etc</i>	Konfigurasi sistem <i>host-specific</i>
<i>lib</i>	<i>Shared libraries</i> dasar dan modul <i>kernel</i>
<i>media</i>	<i>Mount point</i> untuk media-media <i>removable</i>
<i>mnt</i>	<i>Mount point</i> untuk <i>mounting</i> sistem berkas secara temporer

Direktori	Keterangan
opt	Penambahan aplikasi <i>package</i> perangkat lunak
sbin	Sistem <i>binary dasar</i>
srv	Data untuk servis yang disediakan oleh sistem
tmp	Berkas temporer
usr	Hirarki sekunder
var	Data variabel

6.5.3.3. Pilihan Spesifik

Tabel 6-2. Direktori/link yang dibutuhkan dalam /.

Direktori	Keterangan
home	Direktori <i>home user</i>
lib<qual>	Format alternatif dari <i>shared libraries</i> dasar
root	Direktori <i>home</i> untuk <i>user root</i>

6.5.3.4. /bin: Perintah *binary* dasar (untuk digunakan oleh semua pengguna)

/bin berisi perintah-perintah yang dapat digunakan oleh administrator sistem dan pengguna, namun dibutuhkan apabila tidak ada sistem berkas lain yang di-*mount*. /bin juga berisi perintah-perintah yang digunakan secara tidak langsung oleh *script*.

6.5.3.5. /boot: Berkas statik untuk me-*load boot*

Dalam direktori ini, terdapat segala sesuatu yang dibutuhkan untuk melakukan *boot* proses. /boot menyimpan data yang digunakan sebelum *kernel* mulai menjalankan program *user-mode*. Hal ini dapat meliputi sektor *master boot* dan sektor berkas map.

6.5.3.6. /dev: Berkas *device*

Direktori /dev adalah lokasi dari berkas-berkas *device*. Direktori ini harus memiliki perintah bernama "MAKEDEV" yang dapat digunakan untuk menciptakan *device* secara manual. Jika dibutuhkan, "MAKEDEV" harus memiliki segala ketentuan untuk menciptakan *device-device* yang ditemukan dalam sistem, bukan hanya implementasi partikular yang di-*install*.

6.5.3.7. /etc: Konfigurasi sistem *host-specific*

Direktori /etc menyimpan berkas-berkas konfigurasi. Yang dimaksud berkas konfigurasi adalah berkas lokal yang digunakan untuk mengatur operasi dari sebuah program. Berkas ini harus statik dan bukan

merupakan *executable binary*.

6.5.3.8. /home: Direktori home user

/home adalah konsep standar sistem berkas yang *site-specific*, artinya *setup* dalam *host* yang satu dan yang lainnya akan berbeda-beda. Maka, program sebaiknya tidak diletakkan dalam direktori ini.

6.5.3.9. /lib: *Shared libraries* dasar dan modul *kernel*

Direktori /lib meliputi gambar-gambar *shared library* yang dibutuhkan untuk *boot* sistem tersebut dan menjalankan perintah dalam sistem berkas *root*, contohnya dengan *binary* di /bin dan /sbin.

6.5.3.10. /lib<qual>: Format alternatif dari *shared libraries* dasar

Pada sistem yang mendukung lebih dari satu format *binary*, mungkin terdapat satu atau lebih perbedaan dari direktori /lib. Jika direktori ini terdapat lebih dari satu, maka persyaratan dari isi tiap direktori adalah sama dengan direktori /lib normalnya, namun /lib<qual>/cpp tidak dibutuhkan.

6.5.3.11. /media: Mount point media removable

Direktori ini berisi subdirektori yang digunakan sebagai *mount point* untuk media-media *removable* seperti *floppy disk*, *cd rom*, dll.

6.5.3.12. /mnt: *Mount point* untuk sistem berkas yang di-*mount* secara temporer

Direktori ini disediakan agar administrator sistem dapat *mount* suatu sistem berkas yang dibutuhkan secara temporer. Isi dari direktori ini adalah *issue* lokal, dan tidak mempengaruhi sifat-sifat dari program yang sedang dijalankan.

6.5.3.13. /opt: Aplikasi tambahan untuk paket perangkat lunak

/opt disediakan untuk aplikasi tambahan paket perangkat lunak. Paket yang di-*install* di /opt harus menemukan berkas statiknya di direktori /opt/<package> atau /opt/<provider>, dengan <package> adalah nama yang mendeskripsikan paket perangkat lunak tersebut, dan <provider> adalah nama dari *provider* yang bersangkutan.

6.5.3.14. /root: Direktori *home* untuk *user root*

Direktori *home root* dapat ditentukan oleh *developer* atau pilihan-pilihan lokal, namun direktori ini adalah lokasi *default* yang direkomendasikan.

6.5.3.15. /sbin: Binary sistem

Kebutuhan yang digunakan oleh administrator sistem disimpan di /sbin, /usr/sbin, dan /usr/local/sbin. /sbin berisi *binary* dasar untuk *boot* sistem, mengembalikan sistem, memperbaiki sistem sebagai tambahan untuk *binary-binary* di /bin. Program yang dijalankan setelah /usr diketahui harus di-*mount*, diletakkan dalam /usr/bin. Sedangkan, program-program milik administrator sistem yang di-*install* secara lokal sebaiknya diletakkan dalam /usr/local/sbin.

6.5.3.16. /srv: Data untuk servis yang disediakan oleh sistem

/srv berisi data-data *site-specific* yang disediakan oleh sistem.

6.5.3.17. /tmp: Berkas-berkas temporer

Direktori /tmp harus tersedia untuk program-program yang membutuhkan berkas temporer.

6.5.4. Hirarki /usr

6.5.4.1. Tujuan

/usr adalah bagian utama yang kedua dari sistem berkas. /usr bersifat *shareable* dan *read-only*. Hal ini berarti /usr bersifat *shareable* diantara bermacam-macam *host FHS-compliant*, dan tidak boleh di-*write*. *Package* perangkat lunak yang besar tidak boleh membuat subdirektori langsung di bawah hirarki /usr ini.

6.5.4.2. Persyaratan

Tabel 6-3. Direktori/link yang dibutuhkan dalam /usr.

Direktori	Keterangan
bin	Sebagian besar perintah pengguna
include	Berkas <i>header</i> yang termasuk dalam program-program C
lib	<i>Libraries</i>
local	Hirarki lokal (kosong sesudah instalasi main)
sbin	Sistem <i>binary non-vital</i>
share	Data arsitektur yang independen

6.5.4.3. Pilihan spesifik

Tabel 6-4. Direktori/link yang merupakan pilihan dalam /usr.

Direktori	Keterangan
X11R6	Sistem X Window, Versi 11 <i>Release 6</i>
games	<i>Games</i> dan <i>binary educational</i>
lib<qual>	Format <i>libraries</i> alternatif
src	Kode <i>source</i>

Link-link simbolik seperti di bawah ini dapat terjadi, apabila terdapat kebutuhan untuk menjaga keharmonisan dengan sistem yang lama, sampai semua implementasi bisa diasumsikan untuk menggunakan hirarki /var:

- /usr/spool --> /var/spool
- /usr/temp --> /var/tmp
- /usr/spool/locks --> /var/lock

Saat sistem tidak lagi membutuhkan *link-link* di atas, *link* tersebut dapat dihapus.

6.5.4.4. /usr/X11R6: Sistem X Window, Versi 11 Release 6

Hirarki ini disediakan untuk Sistem X Window, Versi 11 *Release 6* dan berkas-berkas yang berhubungan. Untuk menyederhanakan persoalan dan membuat XFree86 lebih kompatibel dengan Sistem X Window, link simbolik di bawah ini harus ada jika terdapat direktori /usr/X11R6:

- /usr/bin/X11 --> /usr/X11R6/bin
- /usr/lib/X11 --> /usr/X11R6/lib/X11
- /usr/include/X11 --> /usr/X11R6/include/X11

Link-link di atas dikhususkan untuk kebutuhan dari pengguna saja, dan perangkat lunak tidak boleh di-*install* atau diatur melalui *link-link* tersebut.

6.5.4.5. /usr/bin: Sebagian perintah pengguna

Direktori ini adalah direktori primer untuk perintah- perintah *executable* dalam sistem.

6.5.4.6. /usr/include: Direktori untuk *include-files* standar

Direktori ini berisi penggunaan umum *include-files* oleh sistem, yang digunakan untuk bahasa pemrograman C.

6.5.4.7. /usr/lib: *Libraries* untuk pemrograman dan *package*

/usr/lib meliputi berkas *object*, *library* dan *binary* internal yang tidak dibuat untuk dieksekusi secara langsung melalui pengguna atau *shell script*. Aplikasi- aplikasi dapat menggunakan subdirektori *single* di bawah /usr/lib. Jika aplikasi tersebut menggunakan subdirektori, semua data yang arsitektur-*dependent* yang digunakan oleh aplikasi tersebut, harus diletakkan dalam subdirektori tersebut juga.

Untuk alasan historis, `/usr/lib/sendmail` harus merupakan *link* simbolik ke `/usr/sbin/sendmail`. Demikian juga, jika `/lib/X11` ada, maka `/usr/lib/X11` harus merupakan *link* simbolik ke `/lib/X11`, atau ke manapun yang dituju oleh *link* simbolik `/lib/X11`.

6.5.4.8. `/usr/lib<qual>`: Format *libraries* alternatif

`/usr/lib<qual>` melakukan peranan yang sama seperti `/usr/lib` untuk format *binary* alternatif, namun tidak lagi membutuhkan *link* simbolik seperti `/usr/lib<qual>/sendmail` dan `/usr/lib<qual>/X11`.

6.5.4.9. `/usr/local/share`

Direktori ini sama dengan `/usr/share`. Satu-satunya pembatas tambahan adalah bahwa direktori `/usr/local/share/man` dan `/usr/local/man` harus *synonomous* (biasanya ini berarti salah satunya harus merupakan *link* simbolik).

6.5.4.10. `/usr/sbin`: Sistem *binary* standar yang non-vital.

Direktori ini berisi *binary* non-vital manapun yang digunakan secara eksklusif oleh administrator sistem. Program administrator sistem yang diperlukan untuk perbaikan sistem, *mounting* `/usr` atau kegunaan penting lainnya harus diletakkan di `/sbin`.

6.5.4.11. `/usr/share`: Data arsitektur independen

Hirarki `/usr/share` hanya untuk data-data arsitektur independen yang *read-only*. Hirarki ini ditujukan untuk bisa di-*share* diantara semua arsitektur *platform* dari sistem operasi; sebagai contoh: sebuah *site* dengan *platform* i386, Alpha dan PPC bisa me-*maintain* sebuah direktori `/usr/share` yang di-*mount* secara sentral.

Program atau paket manapun yang berisi dan memerlukan data yang tidak perlu dimodifikasi harus menyimpan data tersebut di `/usr/share` (atau `/usr/local/share`, apabila di-*install* secara lokal). Sangat direkomendasikan bahwa sebuah subdirektori digunakan dalam `/usr/share` untuk tujuan ini.

6.5.4.12. `/usr/src`: Kode *source*

Dalam direktori ini, dapat diletakkan kode-kode *source*, yang digunakan untuk tujuan referensi.

6.5.5. Hirarki `/var`

6.5.5.1. Tujuan

`/var` berisi berkas data variabel, meliputi berkas dan direktori *spool*, data administratif dan *logging*, serta berkas *transient* dan temporer. Beberapa bagian dari `/var` tidak *shareable* diantara sistem yang berbeda,

antara lain: `/var/log`, `/var/lock` dan `/var/run`. Sedangkan, `/var/mail`, `/var/cache/man`, `/var/cache/fonts` dan `/var/spool/news` dapat di-*share* antar sistem yang berbeda.

`/var` ditetapkan di ini untuk memungkinkan operasi `mount /usr read-only`. Segala sesuatu yang melewati `/usr`, yang telah ditulis selama operasi sistem, harus berada di `/var`. Jika `/var` tidak dapat dibuatkan partisi yang terpisah, biasanya `/var` dipindahkan ke luar dari partisi `root` dan dimasukkan ke dalam partisi `/usr`.

Bagaimanapun, `/var` tidak boleh di-*link* ke `/usr`, karena hal ini membuat pemisahan antara `/usr` dan `/var` semakin sulit dan biasa menciptakan konflik dalam penamaan. Sebaliknya, buat *link* `/var` ke `/usr/var`.

6.5.5.2. Persyaratan

Tabel 6-5. Direktori/link yang dibutuhkan dalam `/var`.

Direktori	Keterangan
cache	Data <i>cache</i> aplikasi
lib	Informasi status variabel
local	Data variabel untuk <code>/usr/local</code>
lock	<i>Lock</i> berkas
log	Berkas dan direktori <i>log</i>
opt	Data variabel untuk <code>/opt</code>
run	Relevansi data untuk menjalankan proses
spool	Aplikasi data <i>spool</i>
tmp	Berkas temporer yang disimpan di dalam <i>reboot</i> sistem

6.5.5.3. Pilihan Spesifik

Direktori atau link simbol yang menuju ke direktori di bawah ini, dibutuhkan dalam `/var`, jika subsistem yang berhubungan dengan direktori tersebut di-*install*:

Tabel 6-6. Direktori/link yang dibutuhkan di dalam `/var`

Direktori	Keterangan
account	<i>Log accounting</i> proses
crash	<i>System crash dumps</i>
games	Data variabel <i>game</i>
mail	Berkas <i>mailbox user</i>
yp	Network Information Service (NIS) berkas <i>database</i>

6.5.5.4. /var/account: Log accounting proses

Direktori ini memegang *log accounting* dari proses yang sedang aktif dan gabungan dari penggunaan data.

6.5.5.5. /var/cache: Aplikasi data cache

/var/cache ditujukan untuk data *cache* dari aplikasi. Data tersebut diciptakan secara lokal sebagai *time-consuming* I/O atau kalkulasi. Aplikasi ini harus dapat menciptakan atau mengembalikan data. Tidak seperti /var/spool, berkas *cache* dapat dihapus tanpa kehilangan data.

Berkas yang ditempatkan di bawah /var/cache dapat *expired* oleh karena suatu sifat spesifik dalam aplikasi, oleh administrator sistem, atau keduanya, maka aplikasi ini harus dapat *recover* dari penghapusan berkas secara manual.

6.5.5.6. /var/crash: System crash dumps

Direktori ini mengatur *system crash dumps*. Saat ini, *system crash dumps* belum dapat di-*support* oleh Linux, namun dapat di-*support* oleh sistem lain yang dapat memenuhi FHS.

6.5.5.7. /var/games: Data variabel game

Data variabel manapun yang berhubungan dengan *games* di /usr harus diletakkan di direktori ini. /var/games harus meliputi data variabel yang ditemukan di /usr; data statik, seperti *help text*, deskripsi level, dll, harus ditempatkan di lain direktori, seperti /usr/share/games.

6.5.5.8. /var/lib: Informasi status variabel

Hirarki ini berisi informasi status suatu aplikasi dari sistem. Yang dimaksud dengan informasi status adalah data yang dimodifikasi program saat program sedang berjalan. Pengguna tidak diperbolehkan untuk memodifikasi berkas di /var/lib untuk mengkonfigurasi operasi *package*. Informasi status ini digunakan untuk memantau kondisi dari aplikasi, dan harus tetap valid setelah *reboot*, tidak berupa *output logging* ataupun data *spool*.

Sebuah aplikasi harus menggunakan subdirektori /var/lib untuk data-datanya. Terdapat satu subdirektori yang dibutuhkan lagi, yaitu /var/lib/misc, yang digunakan untuk berkas-berkas status yang tidak membutuhkan subdirektori.

6.5.5.9. /var/lock: Lock berkas

Lock berkas harus disimpan dalam struktur direktori /var/lock. *Lock* berkas untuk *device* dan *resource* lain yang di-*share* oleh banyak aplikasi, seperti *lock* berkas pada serial *device* yang ditemukan dalam /usr/spool/locks atau /usr/spool/uucp, sekarang disimpan di dalam /var/lock.

Format yang digunakan untuk isi dari *lock* berkas ini harus berupa format *lock* berkas HDB UUCP. Format HDB ini adalah untuk menyimpan pengidentifikasi proses (Process Identifier - PID) sebagai 10 byte angka desimal ASCII, ditutup dengan baris baru. Sebagai contoh, apabila proses 1230 memegang

lock berkas, maka HDO formatnya akan berisi 11 karakter: spasi, spasi, spasi, spasi, spasi, spasi, satu, dua, tiga, nol dan baris baru.

6.5.5.10. /var/log: Berkas dan direktori *log*

Direktori ini berisi bermacam-macam berkas *log*. Sebagian besar *log* harus ditulis ke dalam direktori ini atau subdirektori yang tepat.

6.5.5.11. /var/mail: Berkas *user mailbox*

Mail spool harus dapat diakses melalui /var/mail dan berkas mail spool harus menggunakan form <username>>. Berkas *user mailbox* dalam lokasi ini harus disimpan dengan format standar *mailbox* UNIX.

6.5.5.12. /var/opt: Data variabel untuk /opt

Data variabel untuk paket di dalam /opt harus di- *install* dalam /var/opt/<subdir>, di mana <subdir> adalah nama dari *subtree* dalam /opt tempat penyimpanan data statik dari *package* tambahan perangkat lunak.

6.5.5.13. /var/run: Data variabel *run-time*

Direktori ini berisi data informasi sistem yang mendeskripsikan sistem sejak di *boot* . Berkas di dalam direktori ini harus dihapus dulu saat pertama memulai proses *boot* . Berkas pengidentifikasi proses (PID), yang sebelumnya diletakkan di /etc, sekarang diletakkan di /var/run.

Program yang membaca berkas-berkas PID harus fleksibel terhadap berkas yang diterima, sebagai contoh: program tersebut harus dapat mengabaikan ekstra spasi, baris-baris tambahan, angka nol yang diletakkan di depan, dll.

6.5.5.14. /var/spool: Aplikasi data *spool*

/var/spool berisi data yang sedang menunggu suatu proses. Data di dalam /var/spool merepresentasikan pekerjaan yang harus diselesaikan dalam waktu depan (oleh program, pengguna atau administrator); biasanya data dihapus sesudah selesai diproses.

6.5.5.15. /var/tmp: Berkas temporer yang diletakkan di dalam *reboot* sistem

Direktori /var/tmp tersedia untuk program yang membutuhkan berkas temporer atau direktori yang diletakkan dalam *reboot* sistem. Karena itu, data yang disimpan di /var/tmp lebih bertahan daripada data di dalam /tmp. Berkas dan direktori yang berada dalam /var/tmp tidak boleh dihapus saat sistem di- *boot*. Walaupun data-data ini secara khusus dihapus dalam *site-specific manner* , tetap direkomendasikan bahwa penghapusan dilakukan tidak sesering penghapusan di /tmp.

6.5.5.16. /var/yp: Berkas database Network Information Service (NIS)

Data variabel dalam Network Information Service (NIS) atau yang biasanya dikenal dengan Sun Yellow Pages (YP) harus diletakkan dalam direktori ini.

6.6. Konsep Alokasi Blok Sistem Berkas

6.6.1. Metode Alokasi

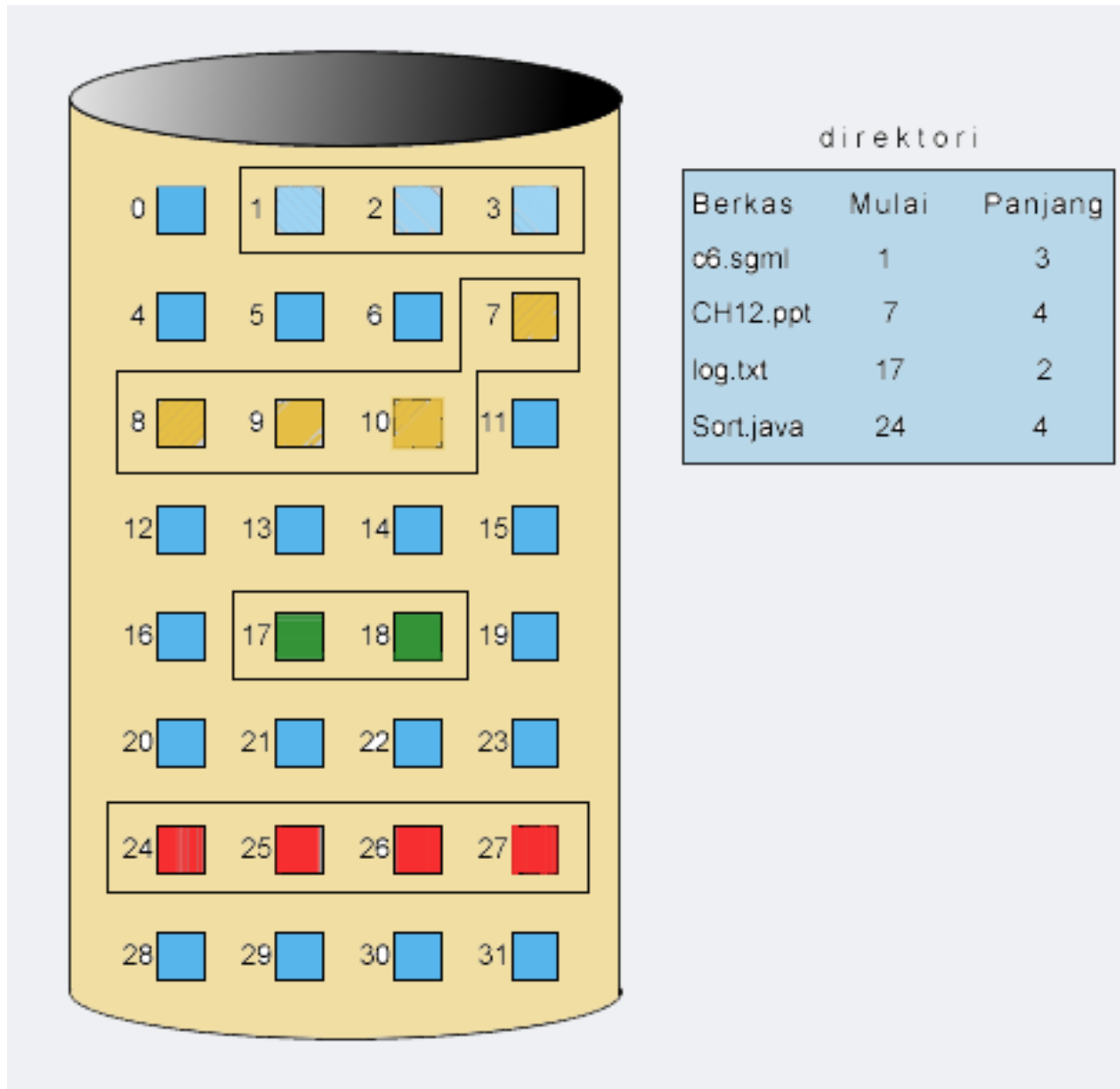
Kegunaan *secondary storage* yang utama adalah menyimpan berkas-berkas yang kita buat, karena sifat disk akan mempertahankan berkas walaupun tidak ada arus listrik. Oleh karena itu, agar kita dapat mengakses berkas-berkas dengan cepat dan memaksimalkan ruang yang ada di disk tersebut, maka lahirlah metode-metode untuk mengalokasikan berkas ke disk. Metode-metode yang akan dibahas lebih lanjut dalam buku ini adalah *contiguous allocation*, *linked allocation*, dan *indexed allocation*. Metode-metode tersebut memiliki beberapa kelebihan dan juga kekurangan. Biasanya sistem operasi memilih satu dari metode di atas untuk mengatur keseluruhan berkas.

6.6.1.1. Contiguous Allocation

Metode ini akan mengalokasikan satu berkas ke dalam blok-blok disk yang berkesinambungan atau berurutan secara linier dari disk, jadi sebuah berkas didefinisikan oleh alamat disk blok pertama dan panjangnya dengan satuan blok atau berapa blok yang diperlukannya. Bila suatu berkas memerlukan n buah blok dan blok awalnya adalah a , berarti berkas tersebut disimpan dalam blok di alamat a , $a + 1$, $a + 2$, $a + 3$, ..., $a + n - 1$. Direktori mengidentifikasi setiap berkas hanya dengan alamat blok pertama berkas tersebut disimpan yang dalam contoh di atas adalah a , dan banyaknya blok yang diperlukan untuk mengalokasikan berkas tersebut yang dalam contoh di atas adalah n .

Berkas yang dialokasikan dengan metode ini akan mudah diakses, karena pengaksesan alamat $a + 1$ setelah alamat a tidak diperlukan perpindahan head, jika diperlukan pemindahan head, maka head tersebut akan hanya akan berpindah satu track. Hal tersebut menjadikan metode ini mendukung pengaksesan secara berurutan, tapi metode ini juga mendukung pengaksesan secara langsung, karena bila ingin mengakses blok ke i berarti kita akan mengakses blok $a + i$.

Gambar 6-11. *Contiguous allocation*



Metode *contiguous allocation* juga mempunyai beberapa masalah. Diantaranya adalah mencari ruang untuk berkas baru, menentukan seberapa besar ruang yang diperlukan untuk sebuah berkas. Untuk masalah mencari ruang untuk berkas baru, akan di implementasikan oleh manajemen ruang kosong.

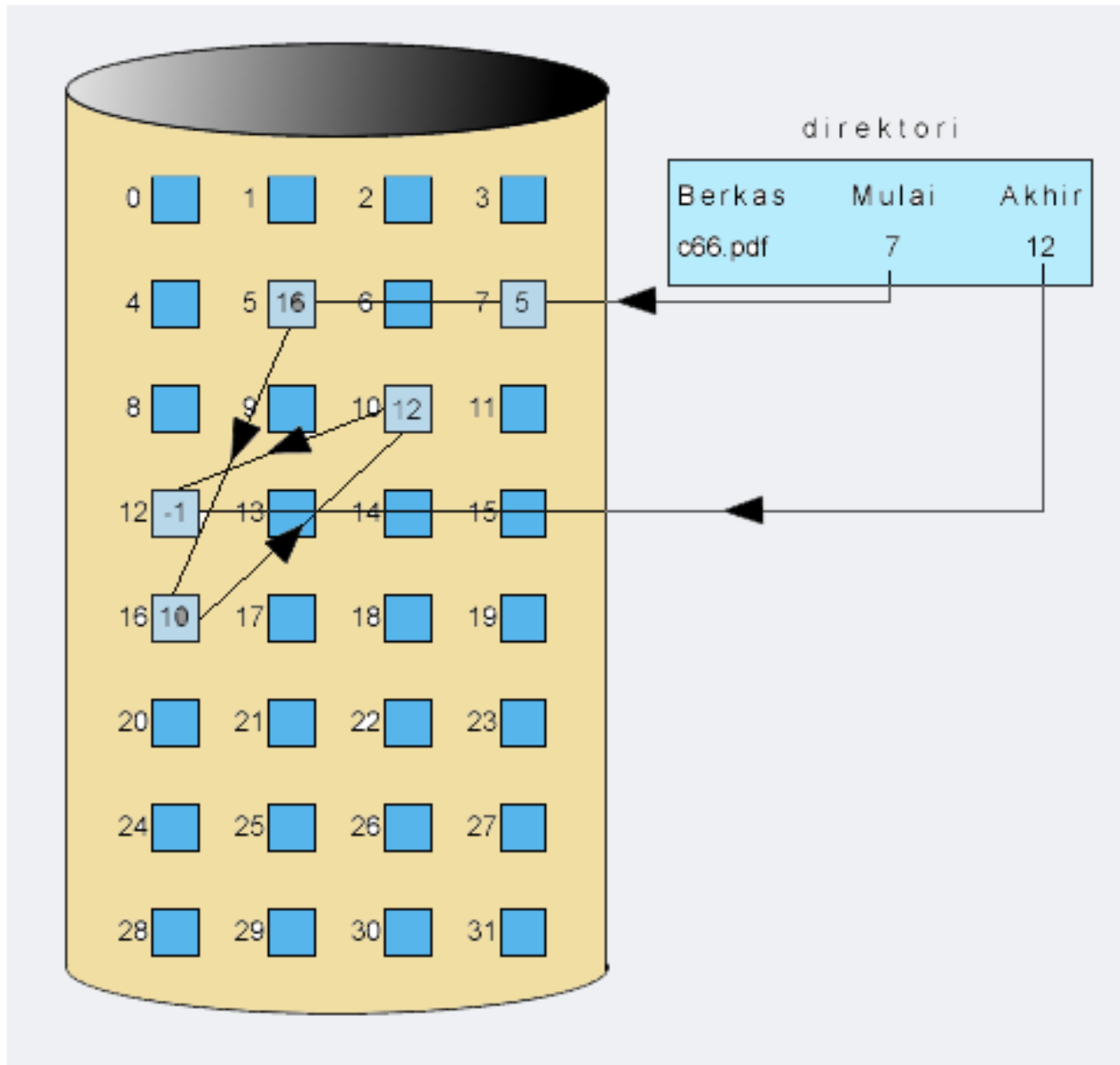
Untuk penentuan ruang kita tidak boleh terlalu kecil atau terlalu besar, bila kita menentukannya terlalu kecil maka ada saatnya berkas tersebut tidak bisa dikembangkan, tapi bila terlalu besar maka akan ada ruang yang sia-sia bila berkas tersebut hanya memerlukan ruang yang kecil.

Metode ini dapat menimbulkan fragmentasi eksternal disaat ruang kosong yang ada diantara berkas-berkas yang sudah terisi tidak cukup untuk mengisi berkas baru. Hal ini terjadi karena blok

pertama dari suatu berkas itu ditentukan oleh sistem operasi, bila berkas pertama blok pertamanya itu di 1 dan memerlukan 9 blok untuk pengalokasiannya dan berkas kedua blok pertamanya di 11 dan memerlukan 5 blok untuk pengalokasiannya, berarti ruang-kosong diantara berkas tersebut ada 1 blok, yaitu di alamat 10. Blok tersebut dapat untuk menyimpan berkas, tetapi hanya berkas yang berukuran 1 blok yang dapat disimpan di blok tersebut.

6.6.1.2. *Linked Allocation*

Metode ini dapat mengatasi masalah yang terjadi pada metode *contiguous allocation*. Dalam metode ini setiap berkas diidentifikasi dengan *linked list* dari blok-blok, jadi blok-blok tersebut tidak harus berkesinambungan dengan blok yang lain. Direktori hanya menyimpan alamat blok pertama dan alamat blok terakhir. Jika kita ingin mengakses blok kedua, maka harus melihat alamatnya di blok pertama dan begitu seterusnya. Oleh karena itu, metode ini hanya mendukung pengaksesan secara berurutan.

Gambar 6-12. *Linked allocation*

Metode *linked allocation* memiliki beberapa kerugian, karena petunjuk ke blok berikutnya memerlukan ruang. Bila ukuran petunjuknya 4 byte dari blok yang ukurannya 512 byte, berarti 0,78% dari ruang disk hanya digunakan untuk petunjuk saja. Hal ini bisa diminimalisasikan dengan menggunakan *cluster* yang menggabungkan 4 blok dalam satu *cluster*, jadi jumlah petunjuknya akan berkurang dari yang tidak memakai *cluster*.

Paling penting dalam metode ini adalah menggunakan *file-allocation table* (FAT). Tabel tersebut menyimpan setiap blok yang ada di disk dan diberi nomor sesuai dengan nomor blok. Jadi, direktori hanya menyimpan alamat dari blok pertama saja, dan untuk selanjutnya dilihat dari tabel tersebut yang menunjukkan ke blok berikutnya. Jika kita memakai metode ini, akan menyebabkan mudahnya untuk

membuat berkas baru atau mengembangkan berkas sebelumnya. Mencari tempat kosong untuk berkas baru lebih mudah, karena kita hanya mencari angka 0 yang pertama dari isi tabel tersebut. Dan bila kita ingin mengembangkan berkas sebelumnya carilah alamat terakhirnya yang memiliki ciri tertentu dan ubahlah isi dari tabel tersebut dengan alamat blok penambahan. Alamat terakhir berisi hal yang unik, sebagai contoh ada yang menuliskan -1, tapi ada juga yang menuliskannya EOF (*End Of File*).

Metode *linked allocation* yang menggunakan FAT akan mempersingkat waktu yang diperlukan untuk mencari sebuah berkas. Karena bila tidak menggunakan FAT, berarti kita harus ke satu blok tertentu dahulu dan baru diketahui alamat blok selanjutnya. Dengan menggunakan FAT kita dapat melihat alamat blok selanjutnya disaat kita masih menuju blok yang dimaksud. Tetapi bagaimanapun ini belum bisa mendukung pengaksesan secara langsung.

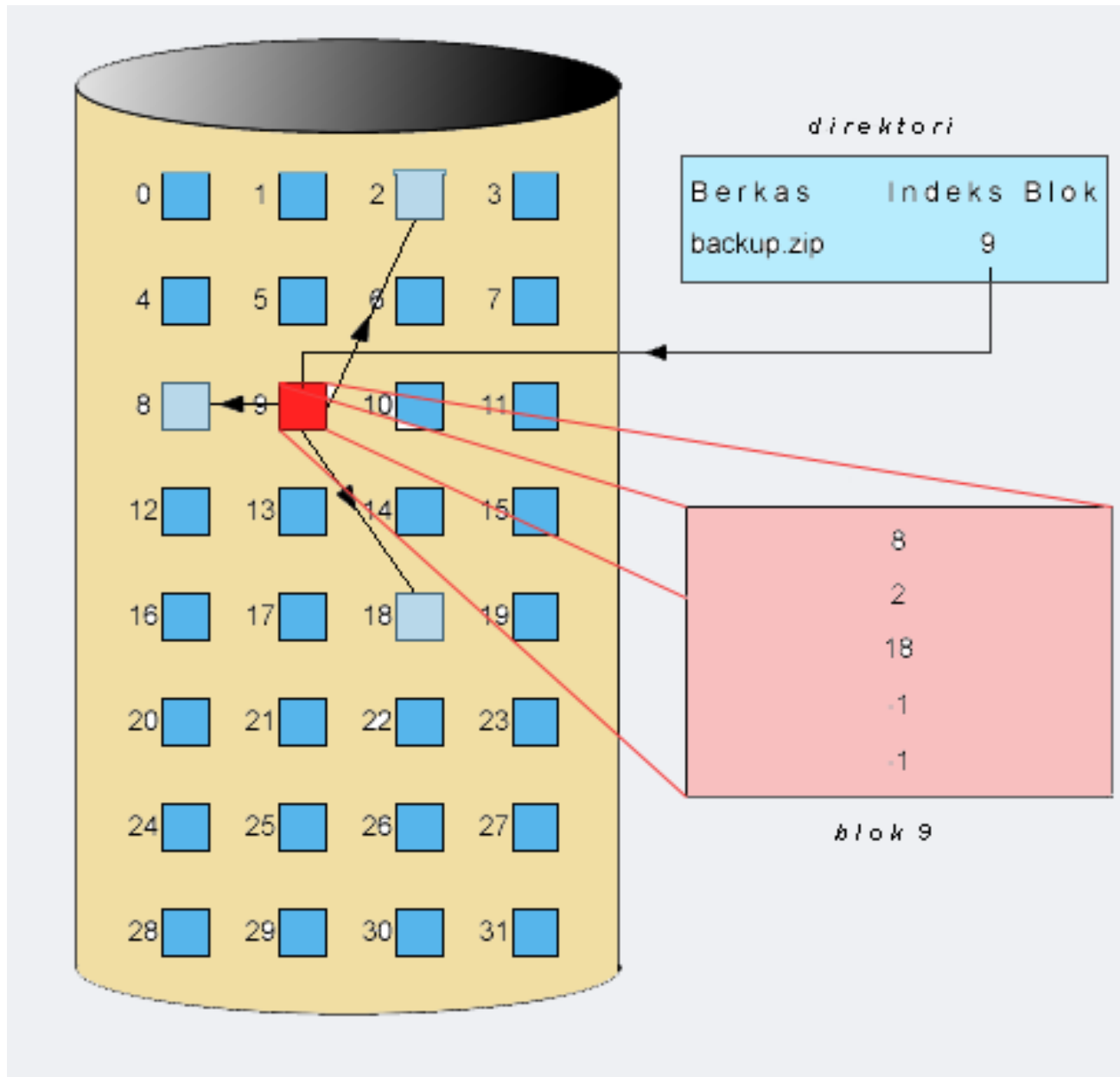
6.6.1.3. *Indexed Allocation*

Metode yang satu ini memecahkan masalah fragmentasi eksternal dari metode *contiguous allocation* dan ruang yang cuma-cuma untuk petunjuk pada metode *linked allocation*, dengan cara menyatukan semua petunjuk kedalam blok indeks yang dimiliki oleh setiap berkas. Jadi, direktori hanya menyimpan alamat dari blok indeks tersebut, dan blok indeks tersebut yang menyimpan alamat dimana blok-blok berkas berada. Untuk berkas yang baru dibuat, maka blok indeksnya di set dengan *null*.

Metode ini mendukung pengaksesan secara langsung, bila kita ingin mengakses blok ke-*i*, maka kita hanya mencari isi dari blok indeks tersebut yang ke-*i* untuk dapatkan alamat blok tersebut.

Metode *indexed allocation* tidak menyia-nyiakannya ruang disk untuk petunjuk, karena dibandingkan dengan metode *linked allocation*, maka metode ini lebih efektif, kecuali bila satu berkas tersebut hanya memerlukan satu atau dua blok saja.

Gambar 6-13. Indexed allocation



Metode ini juga memiliki masalah. Masalah itu timbul disaat berkas berkembang menjadi besar dan blok indeks tidak bisa menampung petunjuk-petunjuknya itu dalam satu blok. Salah satu mekanisme dibawah ini dapat dipakai untuk memecahkan masalah yang tersebut. Mekanisme-mekanisme itu adalah:

- *Linked scheme*: Untuk mengatasi petunjuk untuk berkas yang berukuran besar mekanisme ini menggunakan tempat terakhir dari blok indeks untuk alamat ke blok indeks selanjutnya. Jadi, bila berkas kita masih berukuran kecil, maka isi dari tempat yang terakhir dari blok indeks berkas tersebut adalah *null*. Namun, bila berkas tersebut berkas besar, maka tempat terakhir itu berisikan alamat untuk

ke blok indeks selanjutnya, dan begitu seterusnya.

- *Multilevel index*: Pada mekanisme ini blok indeks itu bertingkat-tingkat, blok indeks pada tingkat pertama akan menunjukkan blok-blok indeks pada tingkat kedua, dan blok indeks pada tingkat kedua menunjukkan alamat-alamat dari blok berkas, tapi bila dibutuhkan bisa dilanjutkan kelevel ketiga dan keempat tergantung dengan ukuran berkas tersebut. Untuk blok indeks 2 level dengan ukuran blok 4.096 byte dan petunjuk yang berukuran 4 byte, dapat mengalokasikan berkas hingga 4 GB, yaitu 1.048.576 blok berkas.
- *Combined scheme*: Mekanisme ini menggabungkan *direct block* dan *indirect block*. *Direct block* akan langsung menunjukkan alamat dari blok berkas, tetapi pada *indirect block* akan menunjukkan blok indeks terlebih dahulu seperti dalam mekanisme *multilevel index*. *Single indirect block* akan menunjukkan ke blok indeks yang akan menunjukkan alamat dari blok berkas, *double indirect block* akan menunjukkan suatu blok yang bersifat sama dengan blok indeks 2 level, dan *triple indirect block* akan menunjukkan blok indeks 3 level. Dimisalkan ada 15 petunjuk dari mekanisme ini, 12 pertama dari petunjuk tersebut adalah *direct block*, jadi bila ukuran blok 4 byte berarti berkas yang dapat diakses secara langsung didukung sampai ukurannya 48 KB. 3 petunjuk berikutnya adalah *indirect block* yang berurutan dari *single indirect block* sampai *triple indirect block*. Yang hanya mendukung 32 bit petunjuk berkas berarti akan hanya mencapai 4 GB, namun yang mendukung 64 bit petunjuk berkas dapat mengalokasikan berkas berukuran sampai satuan terabyte.

6.6.1.4. Kinerja Sistem Berkas

Keefisienan penyimpanan dan waktu akses blok data adalah kriteria yang penting dalam memilih metode yang cocok untuk sistem operasi untuk mengimplementasikan sesuatu. Sebelum memilih sebuah metode alokasi, kita butuh untuk menentukan bagaimana sistem ini akan digunakan.

Untuk beberapa tipe akses, *contiguous allocation* membutuhkan hanya satu akses untuk mendapatkan sebuah blok disk. Sejak kita dapat dengan mudah menyimpan alamat inisial dari sebuah berkas di memori, kita dapat menghitung alamat disk dari blok ke-*i* (atau blok selanjutnya) dengan cepat dan membacanya dengan langsung.

Untuk *linked allocation*, kita juga dapat menyimpan alamat dari blok selanjutnya di memori dan membacanya dengan langsung. Metode ini bagus untuk akses secara berurutan; untuk akses langsung, bagaimanapun, sebuah akses menuju blok ke-*i* harus membutuhkan pembacaan disk ke-*i*. Masalah ini menunjukkan mengapa alokasi yang berurutan tidak digunakan untuk aplikasi yang membutuhkan akses langsung.

Sebagai hasilnya, beberapa sistem mendukung berkas-berkas yang diakses langsung dengan menggunakan *contiguous allocation* dan yang diakses berurutan dengan *linked allocation*. Di dalam kasus ini, sistem operasi harus mempunyai struktur data yang tepat dan algoritma untuk mendukung kedua metode alokasi.

Indexed allocation lebih kompleks. Jika blok indeks sudah ada di memori, akses dapat dibuat secara langsung. Bagaimanapun, menyimpan blok indeks tersebut di memori membutuhkan tempat yang dapat ditolerir. Dengan begitu, kinerja dari *indexed allocation* tergantung dari struktur indeks, ukuran file, dan posisi dari blok yang diinginkan.

Beberapa sistem menggabungkan *contiguous allocation* dengan *indexed allocation* dengan menggunakan *contiguous allocation* untuk berkas-berkas yang kecil (diatas tiga atau empat berkas), dan secara otomatis

mengganti ke *indexed allocation* jika berkas bertambah besar.

6.6.2. Manajemen Ruang Kosong

Sejak ruang disk terbatas, kita butuh menggunakan lagi ruang tersebut dari berkas yang sudah dihapus menjadi berkas yang baru, jika memungkinkan. Untuk menyimpan *track* dari ruang disk yang kosong, sistem membuat daftar ruang-kosong. Daftar ruang-kosong tersebut merekam semua blok-blok disk yang kosong itu semua tidak dialokasikan di beberapa berkas atau direktori.

6.6.2.1. Bit Vector

Seringkali, daftar ruang yang kosong diimplementasikan sebagai sebuah *bit map* atau *bit vector*. Setiap blok direpresentasikan dengan 1 bit. Jika bloknya kosong, bitnya adalah 1; jika bloknya ditempati, bitnya adalah 0.

Sebagai contoh, mempertimbangkan sebuah disk dimana blok-blok 2, 3, 4, 5, 8, 9, 10, 11, 12, 13, 17, 18, 25, 26, dan 27 kosong, dan sisa dari blok-blok tersebut ditempati. *Bit map* dari ruang-kosong yaitu

```
00111100111111000110000011100000...
```

Keuntungan utama dari pendekatan ini adalah relatif sederhana dan keefisienan dalam menemukan blok kosong yang pertama, atau blok-blok kosong n yang berurutan di dalam disk. Sayangnya, *bit vectors* tidak efisien kecuali seluruh vektor disimpan di memori utama (dan ditulis ke disk secara rutin untuk kebutuhan *recovery*). Menyimpan vektor tersebut di memori utama memungkinkan untuk disk-disk yang kecil, seperti pada *microcomputers*, tetapi tidak untuk disk-disk yang besar.

6.6.2.2. Linked List

Pendekatan yang lainnya untuk manajemen ruang-kosong adalah menghubungkan semua blok-blok disk kosong, menyimpan sebuah penunjuk ke blok kosong yang pertama di lokasi yang khusus di disk dan menyimpannya di memori. Blok pertama ini mengandung sebuah penunjuk ke blok disk kosong selanjutnya, dan seterusnya. Sebagai contoh, kita akan menyimpan sebuah penunjuk ke blok 2, sebagai blok kosong pertama. Blok 2 mengandung sebuah penunjuk ke blok 3, yang akan menunjuk ke blok 4, yang akan menunjuk ke blok 5, yang akan menunjuk ke blok 8, dan seterusnya.

Bagaimanapun, skema ini tidak efisien untuk mengakses daftar tersebut, kita harus membaca setiap blok, yang membutuhkan tambahan waktu I/O. Untungnya, mengakses daftar kosong tersebut itu tidak eksekusi yang teratur. Biasanya, sistem operasi tersebut membutuhkan sebuah blok kosong supaya sistem operasi dapat mengalokasikan blok tersebut ke berkas, lalu blok yang pertama di daftar kosong digunakan.

6.6.2.3. Grouping

Sebuah modifikasi dari pendekatan daftar-kosong adalah menyimpan alamat-alamat dari n blok-blok kosong di blok kosong yang pertama. $n-1$ pertama dari blok-blok ini sebenarnya kosong. Blok terakhir mengandung alamat-alamat dari n blok kosong lainnya, dan seterusnya. Pentingnya implementasi ini adalah alamat-alamat dari blok-blok kosong yang banyak dapat ditemukan secara cepat, tidak seperti di pendekatan *linked-list* yang standard.

6.6.2.4. Counting

Daripada menyimpan daftar dari n alamat-alamat disk kosong, kita dapat menyimpan alamat dari blok kosong yang pertama tersebut dan angka n dari blok *contiguous* kosong yang diikuti blok yang pertama. Setiap masukan di daftar ruang-kosong lalu mengandung sebuah alamat disk dan sebuah jumlah. Meskipun setiap masukan membutuhkan ruang lebih daripada alamat-alamat disk yang sederhana, daftar kesemuanya akan lebih pendek, selama jumlahnya rata-rata lebih besar daripada 1.

6.6.3. Efisiensi dan Kinerja

Kita sekarang dapat mempertimbangkan mengenai efek dari alokasi blok dan manajemen direktori dalam kinerja dan penggunaan disk yang efisien. Di bagian ini, kita mendiskusikan tentang bermacam-macam teknik yang digunakan untuk mengembangkan efisiensi dan kinerja dari penyimpana kedua.

6.6.3.1. Efisiensi

Penggunaan yang efisien dari ruang disk sangat tergantung pada alokasi disk dan algoritma direktori yang digunakan. Sebagai contoh, UNIX mengembangkan kinerjanya dengan mencoba untuk menyimpan sebuah blok data berkas dekat dengan blok inode berkas untuk mengurangi *seek time*.

Tipe dari data normalnya disimpan di masukan direktori berkas (atau inode) juga membutuhkan pertimbangan. Biasanya, sebuah "last write date" direkam untuk memberikan informasi kepada user dan untuk menentukan jika berkas ingin di *back up*. Beberapa sistem juga menyimpan sebuah "last access date", supaya seorang user dapat menentukan kapan berkas terakhir dibaca. Hasil dari menyimpan informasi ini adalah ketika berkas sedang dibaca, sebuah field di struktur direktori harus ditulis. Prasyarat ini dapat tidak efisien untuk pengaksesan berkas yang berkala. Umumnya setiap persatuan data yang berhubungan dengan berkas membutuhkan untuk dipertimbangkan efeknya pada efisiensi dan kinerja.

Sebagai contoh, mempertimbangkan bagaimana efisiensi dipengaruhi oleh ukuran penunjuk-penunjuk yang digunakan untuk mengakses data. Bagaimanapun, penunjuk-penunjuk membutuhkan ruang lebih untuk disimpan, dan membuat metode alokasi dan manajemen ruang-kosong menggunakan ruang disk

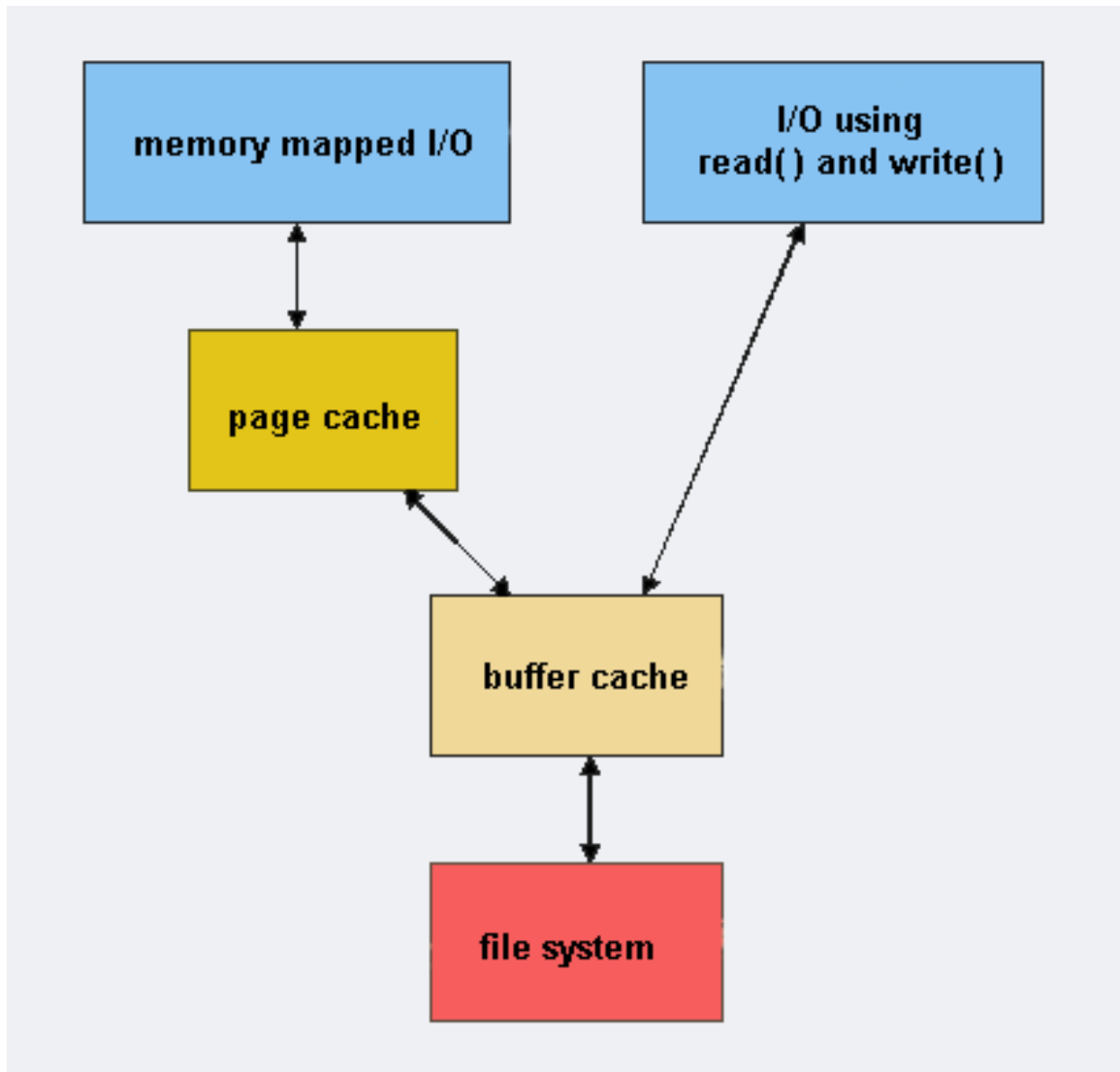
yang lebih. Satu dari kesulitan memilih ukuran penunjuk, atau juga ukuran alokasi yang tetap diantara sistem operasi, adalah rencana untuk efek dari teknologi yang berubah.

6.6.3.2. Kinerja

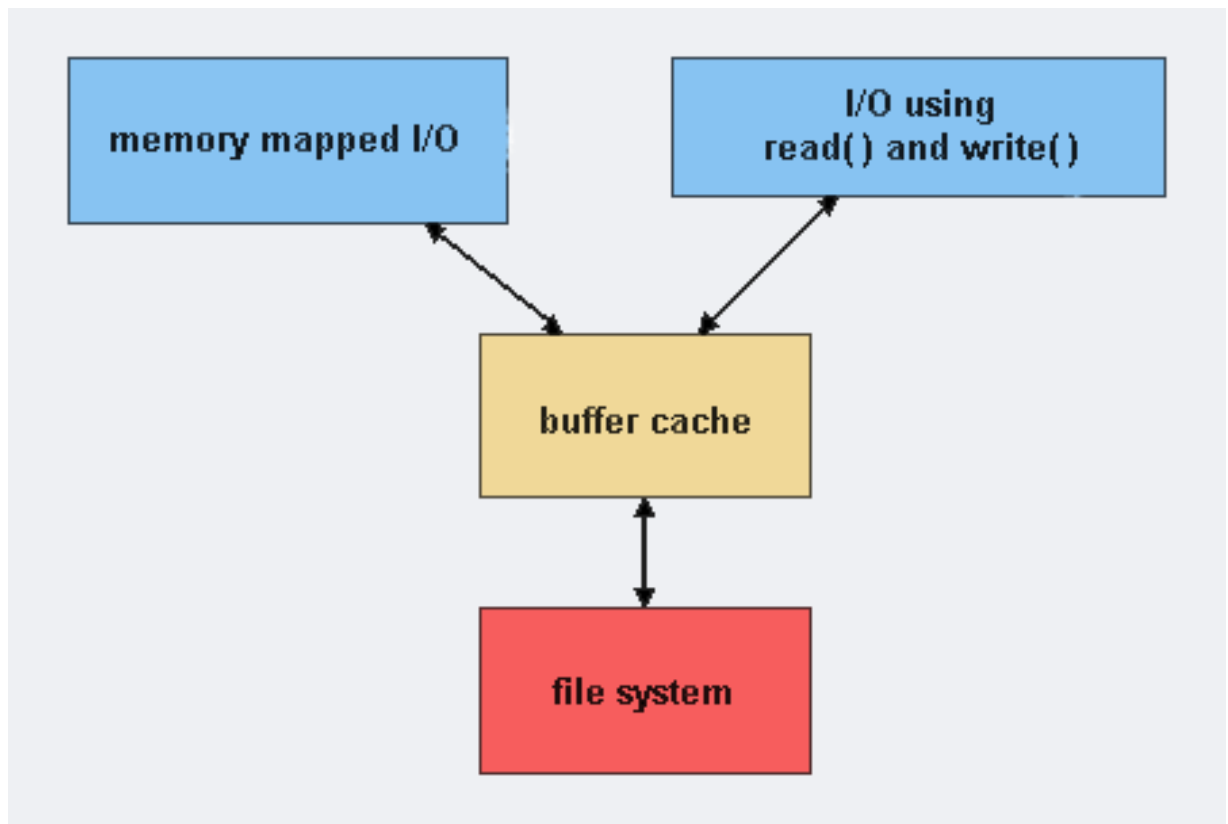
Sekali algoritma sistem berkas dipilih, kita tetap dapat mengembangkan kinerja dengan beberapa cara. Kebanyakan dari *disk controller* mempunyai memori lokal untuk membuat *on-board cache* yang cukup besar untuk menyimpan seluruh *tracks* dengan sekejap.

Beberapa sistem membuat seksi yang terpisah dari memori utama untuk digunakan sebagai *disk cache*, dimana blok-blok disimpan dengan asumsi mereka akan digunakan lagi dengan secepatnya. Sistem lainnya menyimpan data berkas menggunakan sebuah *page cache*. *Page cache* tersebut menggunakan teknik memori virtual untuk menyimpan data berkas sebagai halaman-halaman daripada sebagai blok-blok *file-system-oriented*. Menyimpan data berkas menggunakan alamat-alamat virtual jauh lebih efisien daripada menyimpannya melalui blok disk fisik. Ini dikenal sebagai *unified virtual memory*.

Gambar 6-15.



Gambar 6-16.



Sebagian sistem operasi menyediakan sebuah *unified buffer cache*. Tanpa sebuah *unified buffer cache*, kita mempunyai situasi panggilan *mapping* memori butuh menggunakan dua cache- *page cache* dan *buffer cache*. Karena sistem memori virtual tidak dapat menggunakan dengan *buffer cache*, isi dari berkas di dalam *buffer cache* harus diduplikat ke *page cache*. Situasi ini dikenal dengan *double caching* dan membutuhkan menyimpan data sistem-berkas dua kali. Tidak hanya membuang-buang memori, tetapi ini membuang CPU dan perputaran I/O dikarenakan perubahan data ekstra diantara memori sistem. Juga dapat menyebabkan korupsi berkas. Sebuah *unified buffer cache* mempunyai keuntungan menghindari *double caching* dan menunjuk sistem memori virtual untuk mengatur data sistem berkas.

6.6.4. Recovery

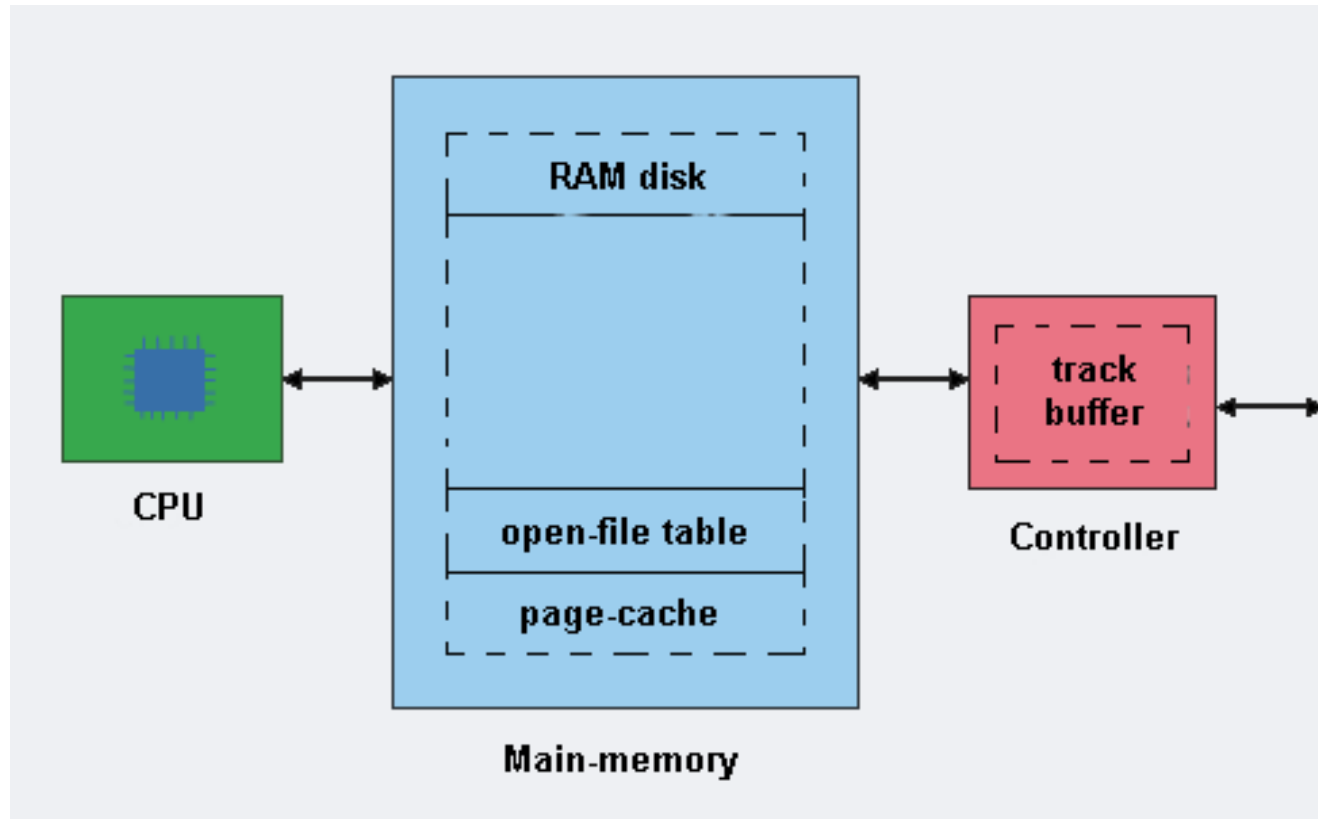
Sejak berkas-berkas dan direktori-direktori dua-duanya disimpan di memori utama dan pada disk, perawatan harus dilakukan untuk memastikan kegagalan sistem tidak terjadi di kehilangan data atau di tidakkonsistennya data.

6.6.4.1. Pengecekan Rutin

Informasi di direktori di memori utama biasanya lebih baru daripada informasi yang ada di disk, karena penulisan dari informasi direktori yang disimpan ke disk tidak terlalu dibutuhkan secepat terjadinya

pembaharuan. Mempertimbangkan efek yang memungkinkan terjadinya *computer crash*. Secara berkala, program khusus akan dijalankan pada saat waktu *reboot* untuk mengecek dan mengoreksi disk yang tidak konsisten. Pemeriksaan rutin membandingkan data yang ada di struktur direktori dengan blok data pada disk, dan mencoba untuk memperbaiki ketidakkonsistenan yang ditemukan.

Gambar 6-17. Macam-macam lokasi *disk-caching*



6.6.4.2. Backup dan Restore

Dikarenakan disk magnetik kadang-kadang gagal, perawatan harus dijalankan untuk memastikan data tidak hilang selamanya. Oleh karena itu, program sistem dapat digunakan untuk *back up* data dari disk menuju ke media penyimpanan yang lainnya, seperti sebuah *floppy disk*, tape magnetik, atau disk optikal. *Recovery* dari kehilangan sebuah file individu, atau seluruh disk, mungkin menjadi masalah dari *restoring* data dari *backup*.

Untuk meminimalis kebutuhan untuk menduplikat, kita dapat menggunakan informasi dari, masing-masing masukan direktori. Sebagai contoh, jika program *backup* mengetahui kapan *backup* terakhir dari file telah selesai, dan tanggal terakhir file di direktori menunjukkan bahwa file tersebut tidak dirubah sejak tanggal tersebut, lalu file tersebut tidak perlu diduplikat lagi.

Sebuah tipe jadwal *backup* yaitu sebagai berikut:

- Day 1:

Menduplikat ke sebuah medium *back up* semua berkas ke disk. Ini disebut sebuah *full backup*.

- Day2:

Menduplikat ke medium lainnya semua berkas yang dirubah sejak hari pertama. Ini adalah *incremental backup*.

- Day3:

Menduplikat ke medium lainnya semua berkas yang dirubah sejak hari ke-2.

- Day N:

Menduplikat ke medium lainnya semua berkas yang dirubah sejak hari ke N-1.

Perputaran baru dapat mempunyai *backupny*aditulis ke semua set sebelumnya, atau ke set yang baru dari media *backup*. N yang terbesar, tentu saja memerlukan tape atau disk yang lebih untuk dibaca untuk penyimpanan yang lengkap. Keuntungan tambahan dari perputaran *backup* ini adalah kita dapat menyimpan berkas apa saja yang tidak sengaja terhapus selama perputaran dengan mengakses berkas yang terhapus dari *backup* hari sebelumnya.

6.6.5. Log-Structured File System

Algoritma *logging* sudah dilakukan dengan sukses untuk menangani masalah dari pemeriksaan rutin. Hasil dari implementasinya dikenal dengan *log-based transaction-oriented* (atau *journaling* sistem berkas).

Pemanggilan kembali yang mengenai struktur data sistem berkas pada disk--seperti struktur-struktur direktori, pointer-pointer blok-kosong, pointer-pointer FCB kosong--dapat menjadi tidak konsisten dikarenakan adanya *system crash*. Sebelum penggunaan dari teknik *log-based* di sistem operasi, perubahan biasanya dipakaikan pada struktur ini. Perubahan-perubahan tersebut dapat diinterupsi oleh *crash*, dengan hasil strukturnya tidak konsisten.

Ada beberapa masalah dengan adanya pendekatan dari menunjuk struktur untuk memecahkan dan memperbaikinya pada *recovery*. Salah satunya adalah ketidakkonsistenan tidak dapat diperbaiki. Pemeriksaan rutin mungkin tidak dapat untuk *recover* struktur tersebut, yang hasilnya kehilangan berkas dan mungkin seluruh direktori.

Solusinya adalah memakai teknik *log-based-recovery* pada sistem berkas metadata yang terbaru. Pada dasarnya, semua perubahan metadata ditulis secara berurutan di sebuah *log*. Masing-masing set dari operasi-operasi yang menampilkan tugas yang spesifik adalah sebuah *transaction*. Jika sistemnya *crashes*, tidak akan ada atau ada kelebihan *transactions* di berkas *log*. *Transaction* tersebut tidak akan pernah lengkap ke sistem berkas walaupun diamsukkan oleh sistem operasi, jadi harus dilengkapi. Keuntungan yang lain adalah proses-proses pembaharuan akan lebih cepat daripada saat dipakai langsung ke struktur data pada disk.

6.7. Rangkuman

Di dalam sebuah sistem operasi, salah satu hal yang paling penting adalah sistem berkas. Sistem berkas ini muncul karena ada tiga masalah utama yang cukup signifikan: kebutuhan untuk menyimpan data dalam jumlah yang besar, kebutuhan agar data tidak mudah hilang (non-volatile), dan informasi harus berdiri sendiri tidak bergantung pada proses. Pada sistem berkas ini, diatur segala rupa macam yang berkaitan dengan sebuah berkas mulai dari atribut, tipe, operasi, struktur, sampai metode akses berkas.

Beberapa sistem komputer menyimpan banyak sekali berkas-berkas dalam *disk*, sehingga diperlukan suatu struktur pengorganisasian data-data sehingga data lebih mudah diatur. Dalam struktur direktori satu tingkat, semua berkas diletakkan pada direktori yang sama, sehingga memiliki suatu keterbatasan karena nama berkas harus unik. Struktur direktori dua tingkat mencoba mengatasi masalah tersebut dengan membuat direktori yang terpisah untuk tiap pengguna yang disebut dengan *user file directory (UFD)*. Sedangkan dalam struktur direktori *tree* setiap pengguna dapat membuat subdirektori sendiri dan mengorganisasikan berkas-berkasnya. Direktori dengan struktur *tree* melarang berbagi berkas atau direktori. Oleh karena itu, struktur dengan *acyclic-graph* memperbolehkan direktori untuk berbagi berkas atau sub-direktori. Struktur Direktori *general graph* mengatasi masalah yang timbul dalam struktur *acyclic* dengan metode *Garbage Collection*.

Mounting adalah proses mengaitkan sebuah sistem berkas yang baru ditemukan pada sebuah piranti ke struktur direktori utama yang sedang dipakai. *Mount point* adalah sebuah direktori dimana berkas baru menjadi dapat diakses.

Sebagai implementasi direktori yang merupakan implementasi dari Implementasi Sistem Berkas, implementasi direktori memiliki algoritma seperti *Linear List* dan *Hashtable*. Direktori pada MS/Dos merupakan sistem dengan direktori hirarki tree. Direktori pada UNIX merupakan struktur direktori tradisional.

Standar Hirarki Sistem Berkas (*File Hierarchy Standard*) adalah rekomendasi penulisan direktori dan berkas-berkas yang diletakkan di dalamnya. FHS ini digunakan oleh perangkat lunak dan *user* untuk menentukan lokasi dari berkas dan direktori.

Informasi yang disimpan di dalam suatu berkas harus disimpan ke dalam disk. Artinya, sistem operasi harus memutuskan tempat informasi itu akan disimpan. Ada 3 method untuk menentukan bagaimana sistem operasi menyimpan informasi ke disk yakni manajemen ruang kosong (mengetahui seberapa luang kapasitas disk yang tersedia), efisiensi dan kinerja, dan *recovery*.

6.8. Latihan

1. Berikan gambaran umum mengenai sistem berkas!
2. Operasi apa saja yang dijalankan untuk melakukan operasi *copy*?
3. Sebutkan salah satu cara mengimplementasikan tipe berkas!
4. Sebutkan dan jelaskan 3 contoh struktur berkas!
5. Apa bedanya *sequential access* dan *direct access*?
6. Apa kelebihan struktur direktori *Acyclic Graph* dengan struktur direktori *Tree*?
7. Jelaskan yang dimaksud dengan *Garbage Collection Scheme*!

8. Struktur Direktori apa saja yang menyediakan fasilitas *sharing*?
9. Apa yang terjadi jika kita mengubah nama suatu berkas?
10. Pada sistem UNIX, apa yg terjadi saat kita ingin menghapus suatu direktori yang masih mengandung suatu berkas?
11. Kemanakah sistem berkas akan selalu di-*mount* ketika sistem sedang berjalan?
12. Apakah yang dimaksud dengan pesan *error* berikut?
pwd: Permission denied

13. Apa perbedaan antara metode implementasi *sharing* antara FTP, DFS dan WWW?
14. Sebutkan pendekatan apa saja yang dipakai untuk mengatasi masalah proteksi berkas beserta keuntungan dan kerugiannya?
15. Berikan sebuah contoh struktur direktori selain yang ada di buku dan jelaskan cara implementasi pada direktori!
16. Sebutkan 2 sistem operasi yang implementasi sistem berkasnya menggunakan *Layered File System*!
17. Jelaskan cara membuat suatu berkas baru!
18. Apakah hubungan partisi dan *mounting* dengan implementasi sistem berkas?
19. Apa perbedaan algoritma *Linear List* dan *Hash Table* untuk implentasi direktori? Menurut anda manakah yang lebih unggul? Jelaskan!
20. Apa yang Anda ketahui mengenai *Filesystem Hierarchy Standard*?
21. Jelaskan 3 tujuan dari sistem berkas *root*!
22. Jelaskan tujuan dan persyaratan dari hirarki */usr*!
23. Jelaskan tujuan dan persyaratan dari hirarki */var*!
24. Apakah kegunaan dari direktori di bawah ini:
 - */boot*
 - */media*
 - */mnt*
 - */root*
 - */usr/lib*
 - */var/cache*

25. Sebutkan 3 metode yang sering digunakan untuk mengalokasikan berkas?
26. Bandingkan masalah-masalah yang dapat terjadi di metode alokasi *contiguous allocation*
27. Bandingkan masalah-masalah yang dapat terjadi di metode alokasi *contiguous allocation* dan *linked allocation*?
28. Sebutkan 3 contoh mekanisme dari *Indexed Allocation*?
29. Jelaskan dengan singkat mengenai *Combined Scheme*!

30. Sebutkan akses berkas yang didukung oleh sistem yang menggunakan metode alokasi *contiguous allocation* dan *linked allocation*?
31. Jika ruang kosong di *bit map* sebagai berikut: 00111011101100010001110011 maka blok mana saja yang kosong?
32. Sebutkan keuntungan dari *I/O* yang menggunakan *unified buffer cache*?
33. Jelaskan cara membuat *backup* data!
34. Solusi apa yang diberikan untuk memastikan adanya *recovery* setelah adanya *system crash*?

6.9. Rujukan

6.9.1. Rujukan Buku:

Silberschatz, Galvin, Gagne. 2002. *Operating System Concepts 6th ed.* John Wiley & Sons.

Tananbaum, Andrew S. 1992. *Modern Operating System 2nd ed.* Englewood cliffs, New Jersey: Prentice Hall Inc.

Stallings, Williem. 2000. *Operating System 4th ed.* Prentice Hall.

6.9.2. Rujukan Internet:

http://infocom.cqu.edu.au/Courses/aut2001/85349/Resources/Study_Guide/10.pdf

<http://www.cs.utah.edu/classes/cs5460/lectures/lecture19-2up.pdf>

http://support.sitescape.com/forum/support/dispatch.cgi/_help/showHelp/page/help/en/webfiles_tabs/share_files.html

>http://www.ncsa.uiuc.edu/UserInfo/Resources/Hardware/IBMp690/IBM/usr/share/man/info/en_US/a_doc_lib/aixbman/admnconc/mount_overview.htm

<http://www.atnf.csiro.au/people/rgooch/linux/docs/vfs.txt>

Bab 7. I/O

Bab 8. Studi Kasus: GNU/Linux

Daftar Pustaka

- [Silberschatz2000] Avi Silberschatz, Peter Galvin, dan Rag Gagne, 2000, *Applied Operating Systems: First Edition*, Edisi Pertama, John Wiley & Sons.
- [introGramacomp] Team Gramacomp, 1996, *Introduksi Komputer*, Edisi Pertama, Gramedia.
- [KennethRosen1999] Kenneth H. Rosen, 1999, *Discrete Mathematics and Its Application*, McGraw Hill.
- [pok1] Carl V. Hamacher dan dkk, 2002, *Computer Organization: Fifth Edition*, Edisi Kelima, McGraw-Hill.
- [pok2] David Patterson dan John Hennessy, 1991, *Computer Organization & Design: The Hardware/Software Interface*, Edisi Pertama, Morgan Kaufmann Publishers, Inc..
- [webopedia] Team Jupitermedia, 2003, *HTML: Webopedia*, 2003, Jupitermedia Corporation.
- [msdnoc2003] Press Microsoft, 2003, *MSDN Library*: , October 2003, Microsoft Press.
- [Stallings2001] William Stallings, 2001, *Operating Systems*, Prentice Hall.
- [Tanenbaum1992] Andrew S. Tanenbaum, 1992, *Modern Operating Systems*, Prentice-Hall Inc..

Lampiran A. *GNU Free Documentation License*

Version 1.1, March 2000

Copyright © 2000 Free Software Foundation, Inc. 59 Temple Place, Suite 330, Boston, MA 02111-1307 USA
Everyone is permitted to copy and distribute verbatim copies of this license document, but changing it is not allowed.

A.1. PREAMBLE

The purpose of this License is to make a manual, textbook, or other written document "free" in the sense of freedom: to assure everyone the effective freedom to copy and redistribute it, with or without modifying it, either commercially or noncommercially. Secondly, this License preserves for the author and publisher a way to get credit for their work, while not being considered responsible for modifications made by others.

This License is a kind of "copyleft", which means that derivative works of the document must themselves be free in the same sense. It complements the GNU General Public License, which is a copyleft license designed for free software.

We have designed this License in order to use it for manuals for free software, because free software needs free documentation: a free program should come with manuals providing the same freedoms that the software does. But this License is not limited to software manuals; it can be used for any textual work, regardless of subject matter or whether it is published as a printed book. We recommend this License principally for works whose purpose is instruction or reference.

A.2. APPLICABILITY AND DEFINITIONS

This License applies to any manual or other work that contains a notice placed by the copyright holder saying it can be distributed under the terms of this License. The "Document", below, refers to any such manual or work. Any member of the public is a licensee, and is addressed as "you".

A "Modified Version" of the Document means any work containing the Document or a portion of it, either copied verbatim, or with modifications and/or translated into another language.

A "Secondary Section" is a named appendix or a front-matter section of the Document that deals exclusively with the relationship of the publishers or authors of the Document to the Document's overall subject (or to related matters) and contains nothing that could fall directly within that overall subject. (For example, if the Document is in part a textbook of mathematics, a Secondary Section may not explain any mathematics.) The relationship could be a matter of historical connection with the subject or with related matters, or of legal, commercial, philosophical, ethical or political position regarding them.

The "Invariant Sections" are certain Secondary Sections whose titles are designated, as being those of Invariant Sections, in the notice that says that the Document is released under this License.

The "Cover Texts" are certain short passages of text that are listed, as Front-Cover Texts or Back-Cover Texts, in the notice that says that the Document is released under this License.

A "Transparent" copy of the Document means a machine-readable copy, represented in a format whose specification is available to the general public, whose contents can be viewed and edited directly and straightforwardly with generic text editors or (for images composed of pixels) generic paint programs or (for drawings) some widely available drawing editor, and that is suitable for input to text formatters or for automatic translation to a variety of formats suitable for input to text formatters. A copy made in an otherwise Transparent file format whose markup has been designed to thwart or discourage subsequent modification by readers is not Transparent. A copy that is not "Transparent" is called "Opaque".

Examples of suitable formats for Transparent copies include plain ASCII without markup, Texinfo input format, LaTeX input format, SGML or XML using a publicly available DTD, and standard-conforming simple HTML designed for human modification. Opaque formats include PostScript, PDF, proprietary formats that can be read and edited only by proprietary word processors, SGML or XML for which the DTD and/or processing tools are not generally available, and the machine-generated HTML produced by some word processors for output purposes only.

The "Title Page" means, for a printed book, the title page itself, plus such following pages as are needed to hold, legibly, the material this License requires to appear in the title page. For works in formats which do not have any title page as such, "Title Page" means the text near the most prominent appearance of the work's title, preceding the beginning of the body of the text.

A.3. VERBATIM COPYING

You may copy and distribute the Document in any medium, either commercially or noncommercially, provided that this License, the copyright notices, and the license notice saying this License applies to the Document are reproduced in all copies, and that you add no other conditions whatsoever to those of this License. You may not use technical measures to obstruct or control the reading or further copying of the copies you make or distribute. However, you may accept compensation in exchange for copies. If you distribute a large enough number of copies you must also follow the conditions in section 3.

You may also lend copies, under the same conditions stated above, and you may publicly display copies.

A.4. COPYING IN QUANTITY

If you publish printed copies of the Document numbering more than 100, and the Document's license notice requires Cover Texts, you must enclose the copies in covers that carry, clearly and legibly, all these Cover Texts: Front-Cover Texts on the front cover, and Back-Cover Texts on the back cover. Both covers must also clearly and legibly identify you as the publisher of these copies. The front cover must present the full title with all words of the title equally prominent and visible. You may add other material on the covers in addition. Copying with changes limited to the covers, as long as they preserve the title of the Document and satisfy these conditions, can be treated as verbatim copying in other respects.

If the required texts for either cover are too voluminous to fit legibly, you should put the first ones listed (as many as fit reasonably) on the actual cover, and continue the rest onto adjacent pages.

If you publish or distribute Opaque copies of the Document numbering more than 100, you must either include a machine-readable Transparent copy along with each Opaque copy, or state in or with each Opaque copy a publicly-accessible computer-network location containing a complete Transparent copy of the Document, free of added material, which the general network-using public has access to download

anonymously at no charge using public-standard network protocols. If you use the latter option, you must take reasonably prudent steps, when you begin distribution of Opaque copies in quantity, to ensure that this Transparent copy will remain thus accessible at the stated location until at least one year after the last time you distribute an Opaque copy (directly or through your agents or retailers) of that edition to the public.

It is requested, but not required, that you contact the authors of the Document well before redistributing any large number of copies, to give them a chance to provide you with an updated version of the Document.

A.5. MODIFICATIONS

You may copy and distribute a Modified Version of the Document under the conditions of sections 2 and 3 above, provided that you release the Modified Version under precisely this License, with the Modified Version filling the role of the Document, thus licensing distribution and modification of the Modified Version to whoever possesses a copy of it. In addition, you must do these things in the Modified Version:

- A. Use in the Title Page (and on the covers, if any) a title distinct from that of the Document, and from those of previous versions (which should, if there were any, be listed in the History section of the Document). You may use the same title as a previous version if the original publisher of that version gives permission.
- B. List on the Title Page, as authors, one or more persons or entities responsible for authorship of the modifications in the Modified Version, together with at least five of the principal authors of the Document (all of its principal authors, if it has less than five).
- C. State on the Title page the name of the publisher of the Modified Version, as the publisher.
- D. Preserve all the copyright notices of the Document.
- E. Add an appropriate copyright notice for your modifications adjacent to the other copyright notices.
- F. Include, immediately after the copyright notices, a license notice giving the public permission to use the Modified Version under the terms of this License, in the form shown in the Addendum below.
- G. Preserve in that license notice the full lists of Invariant Sections and required Cover Texts given in the Document's license notice.
- H. Include an unaltered copy of this License.
- I. Preserve the section entitled "History", and its title, and add to it an item stating at least the title, year, new authors, and publisher of the Modified Version as given on the Title Page. If there is no section entitled "History" in the Document, create one stating the title, year, authors, and publisher of the Document as given on its Title Page, then add an item describing the Modified Version as stated in the previous sentence.
- J. Preserve the network location, if any, given in the Document for public access to a Transparent copy of the Document, and likewise the network locations given in the Document for previous versions it was based on. These may be placed in the "History" section. You may omit a network location for a work that was published at least four years before the Document itself, or if the original publisher of the version it refers to gives permission.

- K. In any section entitled "Acknowledgements" or "Dedications", preserve the section's title, and preserve in the section all the substance and tone of each of the contributor acknowledgements and/or dedications given therein.
- L. Preserve all the Invariant Sections of the Document, unaltered in their text and in their titles. Section numbers or the equivalent are not considered part of the section titles.
- M. Delete any section entitled "Endorsements". Such a section may not be included in the Modified Version.
- N. Do not retitle any existing section as "Endorsements" or to conflict in title with any Invariant Section.

If the Modified Version includes new front-matter sections or appendices that qualify as Secondary Sections and contain no material copied from the Document, you may at your option designate some or all of these sections as invariant. To do this, add their titles to the list of Invariant Sections in the Modified Version's license notice. These titles must be distinct from any other section titles.

You may add a section entitled "Endorsements", provided it contains nothing but endorsements of your Modified Version by various parties—for example, statements of peer review or that the text has been approved by an organization as the authoritative definition of a standard.

You may add a passage of up to five words as a Front-Cover Text, and a passage of up to 25 words as a Back-Cover Text, to the end of the list of Cover Texts in the Modified Version. Only one passage of Front-Cover Text and one of Back-Cover Text may be added by (or through arrangements made by) any one entity. If the Document already includes a cover text for the same cover, previously added by you or by arrangement made by the same entity you are acting on behalf of, you may not add another; but you may replace the old one, on explicit permission from the previous publisher that added the old one.

The author(s) and publisher(s) of the Document do not by this License give permission to use their names for publicity for or to assert or imply endorsement of any Modified Version.

A.6. COMBINING DOCUMENTS

You may combine the Document with other documents released under this License, under the terms defined in section 4 above for modified versions, provided that you include in the combination all of the Invariant Sections of all of the original documents, unmodified, and list them all as Invariant Sections of your combined work in its license notice.

The combined work need only contain one copy of this License, and multiple identical Invariant Sections may be replaced with a single copy. If there are multiple Invariant Sections with the same name but different contents, make the title of each such section unique by adding at the end of it, in parentheses, the name of the original author or publisher of that section if known, or else a unique number. Make the same adjustment to the section titles in the list of Invariant Sections in the license notice of the combined work.

In the combination, you must combine any sections entitled "History" in the various original documents, forming one section entitled "History"; likewise combine any sections entitled "Acknowledgements", and any sections entitled "Dedications". You must delete all sections entitled "Endorsements."

A.7. COLLECTIONS OF DOCUMENTS

You may make a collection consisting of the Document and other documents released under this License, and replace the individual copies of this License in the various documents with a single copy that is included in the collection, provided that you follow the rules of this License for verbatim copying of each of the documents in all other respects.

You may extract a single document from such a collection, and distribute it individually under this License, provided you insert a copy of this License into the extracted document, and follow this License in all other respects regarding verbatim copying of that document.

A.8. AGGREGATION WITH INDEPENDENT WORKS

A compilation of the Document or its derivatives with other separate and independent documents or works, in or on a volume of a storage or distribution medium, does not as a whole count as a Modified Version of the Document, provided no compilation copyright is claimed for the compilation. Such a compilation is called an "aggregate", and this License does not apply to the other self-contained works thus compiled with the Document, on account of their being thus compiled, if they are not themselves derivative works of the Document.

If the Cover Text requirement of section 3 is applicable to these copies of the Document, then if the Document is less than one quarter of the entire aggregate, the Document's Cover Texts may be placed on covers that surround only the Document within the aggregate. Otherwise they must appear on covers around the whole aggregate.

A.9. TRANSLATION

Translation is considered a kind of modification, so you may distribute translations of the Document under the terms of section 4. Replacing Invariant Sections with translations requires special permission from their copyright holders, but you may include translations of some or all Invariant Sections in addition to the original versions of these Invariant Sections. You may include a translation of this License provided that you also include the original English version of this License. In case of a disagreement between the translation and the original English version of this License, the original English version will prevail.

A.10. TERMINATION

You may not copy, modify, sublicense, or distribute the Document except as expressly provided for under this License. Any other attempt to copy, modify, sublicense or distribute the Document is void, and will automatically terminate your rights under this License. However, parties who have received copies, or rights, from you under this License will not have their licenses terminated so long as such parties remain in full compliance.

A.11. FUTURE REVISIONS OF THIS LICENSE

The Free Software Foundation may publish new, revised versions of the GNU Free Documentation License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns. See <http://www.gnu.org/copyleft/>.

Each version of the License is given a distinguishing version number. If the Document specifies that a particular numbered version of this License "or any later version" applies to it, you have the option of following the terms and conditions either of that specified version or of any later version that has been published (not as a draft) by the Free Software Foundation. If the Document does not specify a version number of this License, you may choose any version ever published (not as a draft) by the Free Software Foundation.

A.12. How to use this License for your documents

To use this License in a document you have written, include a copy of the License in the document and put the following copyright and license notices just after the title page:

Copyright © YEAR YOUR NAME.

Permission is granted to copy, distribute and/ or modify this document under the terms of the GNU Free Documentation License, Version 1.1 or any later version published by the Free Software Foundation; with the Invariant Sections being LIST THEIR TITLES, with the Front-Cover Texts being LIST, and with the Back-Cover Texts being LIST. A copy of the license is included in the section entitled "GNU Free Documentation License".

If you have no Invariant Sections, write "with no Invariant Sections" instead of saying which ones are invariant. If you have no Front-Cover Texts, write "no Front-Cover Texts" instead of "Front-Cover Texts being LIST"; likewise for Back-Cover Texts.

If your document contains nontrivial examples of program code, we recommend releasing these examples in parallel under your choice of free software license, such as the GNU General Public License, to permit their use in free software.

Indeks

- Allocation
 - Contiguous
 - Indexed, 40
 - Indexed
 - Multilevel, 44
 - Linked
 - FAT, 42
- Backup
 - Restore, 53
- Berkas, 6
 - Atribut, 7
 - Jenis, 6
 - Operasi, 7
 - Shareable, 30
 - Static, 30
 - Struktur, 8
 - Tipe, 7
 - Unshareable, 30
 - Variable, 30
- bit map
 - bit vector, 47
- Contiguous
 - Allocation, 40
- Direktori
 - CP/M, 28
 - MS-DOS, 29
 - Operasi, 9
 - Struktur, 10
 - Asiklik, 12
 - Dua Tingkat, 11
 - Satu Tingkat, 10
 - Single Level, 10
 - Tree, 11
 - Two Level, 11
 - Umum, 12
 - UNIX, 29
- File, 6
- FTP
 - DFS
 - WWW, 17
- Garbage-Collection Scheme, 12
- Grouping
 - Counting, 47
- Hirarki
 - /usr, 34
 - /var, 36
- Implementasi
 - Direktori, 27
- Implementasi Sistem Berkas, 24
- Kinerja
 - efisiensi, 49
- Linear List
 - Hash Table, 27
- Metode
 - Akses, 9
- mount
 - umount
 - mount point, 13
 - mount point, 14
- Nama berkas
 - Komponen, 30
- NFS, 15
- On disk
 - In mmemory, 24
- owner
 - group, 16
 - universe, 18
- page cache
 - disk cache
 - buffer cache, 50
- Partisi
 - Mounting, 25
- Path
 - Mutlak, 11
 - Relatif, 11
- Pengecekan rutin
 - Restore
 - Backup, 52
- Persyaratan /usr
 - Direktori, 34
- Persyaratan /var
 - Direktori, 37
- server
 - client, 17
- Sistem Berkas
 - Persyaratan
 - Direktori, 31
 - Root
 - Operasi, 31
- Struktur Berkas
 - Layered File System, 20