B a b 5

Sistem Berkas

Tujuan Pelajaran

Setelah mempelajari bab ini, Anda diharapkan :

- Memahami konsep dasar sistem berkas
- Mengetahui metode akses pada sistem berkas
- Memahami struktur direktori
- Memahami struktur berkas
- Memahami proteksi sistem berkas
- Memahami implementasi direktori

5.1. Pengertian

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

- kumpulan berkas, sebagai tempat penyimpanan data, serta
- struktur direktori, yang mengatur dan menyediakan informasi mengenai seluruh berkas dalam sistem.

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

5.2. Berkas

5.2.1. Konsep Dasar

Seperti yang telah kita ketahui, komputer dapat menyimpan informasi ke beberapa media penyimpanan yang berbeda, seperti *magnetic disks*, *magnetic tapes*, dan *optical disks*. Agar komputer dapat digunakan dengan nyaman, sistem operasi menyediakan sistem penyimpanan dengan sistematika yang seragam. Sistem Operasi mengabstraksi properti fisik dari media penyimpanannya dan mendefinisikan unit penyimpanan logis, yaitu berkas. Berkas dipetakan ke media fisik oleh sistem operasi. Media penyimpanan ini umumnya bersifat *non-volatile*, sehingga kandungan di dalamnya tidak akan hilang jika terjadi gagal listrik mau pun *system reboot*.

Berkas adalah kumpulan informasi berkait yang diberi nama dan direkam pada penyimpanan sekunder. Dari sudut pandang pengguna, berkas merupakan bagian terkecil dari penyimpanan logis, artinya data tidak dapat ditulis ke penyimpanan sekunder kecuali jika berada di dalam berkas. Biasanya berkas merepresentasikan program (baik *source* mau pun bentuk objek) dan data. Data dari berkas dapat bersifat numerik, alfabetik, alfanumerik, atau pun biner. Format berkas juga bisa bebas, misalnya berkas teks, atau dapat juga diformat pasti. Secara umum, berkas adalah urutan bit, byte, baris, atau catatan yang didefinisikan oleh pembuat berkas dan pengguna.

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

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

5.2.2. Atribut Pada Berkas

Berkas diberi nama, untuk kenyamanan bagi pengguna, dan untuk acuan bagi data yang terkandung di dalamnya. Nama berkas biasanya berupa string atau karakter. Beberapa sistem membedakan penggunaan huruf besar dan kecil dalam penamaan sebuah berkas, sementara sistem yang lain menganggap kedua hal di atas sama.Ketika

berkas diberi nama, maka berkas tersebut akan menjadi mandiri terhadap proses, pengguna, bahkan sistem yang membuatnya. Atribut berkas terdiri dari:

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

Data tersebut dapat berguna untuk proteksi, keamanan, dan monitoring penggunaan dari berkas.

Informasi tentang seluruh berkas disimpan dalam struktur direktori yang terdapat pada penyimpanan sekunder. Direktori, seperti berkas, harus bersifat *non-volatile*, sehingga keduanya harus disimpan pada sebuah *device* dan baru dibawa bagian per bagian ke memori pada saat dibutuhkan.

5.2.3. Operasi Pada Berkas

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

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

- *Menulis pada sebuah berkas*: Untuk menulis pada berkas, kita menggunakan *system call* beserta nama berkas yang akan ditulisi dan informasi apa yang akan ditulis pada berkas. Ketika diberi nama berkas, sistem mencari ke direktori untuk mendapatkan lokasi berkas. Sistem juga harus menyimpan penunjuk tulis pada berkas dimana penulisan berikut akan ditempatkan. Penunjuk tulis harus diperbaharui setiap terjadi penulisan pada berkas.

- *Membaca sebuah berkas*: Untuk dapat membaca berkas, kita menggunakan *system call* beserta nama berkas dan di blok memori mana berkas berikutnya diletakkan. Sama seperti menulis, direktori mencari berkas yang akan dibaca, dan sistem menyimpan penunjuk baca pada berkas dimana pembacaan berikutnya akan terjadi. Ketika pembacaan dimulai, penunjuk baca harus diperbaharui.Sehingga secara umum, suatu berkas ketika sedang dibaca atau ditulis, kebanyakan sistem hanya mempunyai satu penunjuk, baca dan tulis menggunakan penunjuk yang sama, hal ini menghemat tempat dan mengurangi kompleksitas sistem.

- *Menempatkan kembali sebuah berkas*: Direktori yang bertugas untuk mencari berkas yang bersesuaian, dan mengembalikan lokasi berkas pada saat itu. Menempatkan berkas tidak perlu melibatkan proses I/O. Operasi sering disebut pencarian berkas.

- *Menghapus sebuah berkas*: Untuk menghapus berkas kita perlu mencari berkas tersebut di dalam direktori. Setelah ditemukan kita membebaskan tempat yang dipakai berkas tersebut (sehingga dapat digunakkan oleh berkas lain) dan menghapus tempatnya di direktori.

- *Memendekkan berkas*: Ada suatu keadaan dimana pengguna menginginkan atribut dari berkas tetap sama tetapi ingin menghapus isi dari berkas tersebut. Fungsi ini mengizinkan semua atribut tetap sama tetapi panjang berkas menjadi nol, hal ini lebih baik dari pada memaksa pengguna untuk menghapus berkas dan membuatnya lagi.

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

Operasi dasar ini kemudian digabung untuk melakukan operasi lainnya. Sebagai contoh misalnya kita menginginkan salinan dari suatu berkas, atau menyalin berkas ke peralatan I/O lainnya seperti *printer*, dengan cara membuat berkas lalu membaca dari berkas lama dan menulis ke berkas yang baru.

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

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

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

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

- *Penunjuk Berkas*: Pada sistem yang tidak mengikutkan batas berkas sebagai bagian dari system call baca dan tulis, sistem tersebut harus mengikuti posisi dimana terakhir proses baca dan tulis sebagai penunjuk. Penunjuk ini unik untuk setiap operasi pada berkas, maka dari itu harus disimpan terpisah dari atribut berkas yang ada pada disk.

- *Penghitung berkas yang terbuka*: Setelah berkas ditutup, sistem harus mengosongkan kembali tabel berkas yang dibuka yang digunakan oleh berkas tadi atau tempat di tabel akan habis. Karena mungkin ada beberapa proses yang membuka berkas secara bersamaan dan sistem harus menunggu sampai berkas tersebut ditutup sebelum mengosongkan tempatnya di tabel. Penghitung ini mencatat banyaknya berkas yang telah dibuka dan ditutup, dan menjadi nol ketika yang terakhir membaca berkas menutup berkas tersebut barulah sistem dapat mengosongkan tempatnya di tabel.

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

Beberapa sistem operasi menyediakan fasilitas untuk memetakan berkas ke dalam memori pada sistem memori virtual. Hal tersebut mengizinkan bagian dari berkas ditempatkan pada suatu alamat di memori virtual. Operasi baca dan tulis pada memori dengan alamat tersebut dianggap sebagai operasi baca dan tulis pada berkas yang ada di alamat tersebut. Menutup berkas mengakibatkan semua data yang ada pada alamat memori tersebut dikembalikan ke disk dan dihilangkan dari memori virtual yang digunakan oleh proses.

5.2.4. Jenis Berkas

Pertimbangan utama dalam perancangan sistem berkas dan seluruh sistem operasi, apakah sistem operasi harus mengenali dan mendukung jenis berkas. Jika suatu sistem operasi mengenali jenis dari berkas, maka ia dapat mengoperasikan berkas tersebut. Contoh apabila pengguna mencoba mencetak berkas yang merupakan kode biner dari program yang pasti akan menghasilkan sampah, hal ini dapat dicegah apabila sistem operasi sudah diberitahu bahwa berkas tersebut merupakan kode biner.

Teknik yang umum digunakan dalam implementasi jenis berkas adalah menambahkan jenis berkas dalam nama berkas. Nama dibagi dua, nama dan akhiran (ekstensi), biasanya dipisahkan dengan karakter titik. Sistem menggunakan akhiran tersebut untuk mengindikasikan jenis berkas dan jenis operasi yang dapat dilakukan pada berkas tersebut. Sebagai contoh hanya berkas yang berakhiran *.bat*, *.exe* atau *.com* yang bisa dijalankan (eksekusi). Program aplikasi juga menggunakan akhiran tersebut untuk mengenal berkas yang dapat dioperasikannya. Akhiran ini dapat ditimpa atau diganti jika diperbolehkan oleh sistem operasi.

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

Sistem UNIX tidak dapat menyediakan dukungan untuk akhiran berkas karena menggunakan angka ajaib yang disimpan di depan berkas untuk mengenali jenis berkas. Tidak semua berkas memiliki angka ini, jadi sistem tidak bisa bergantung pada informasi ini. Tetapi UNIX memperbolehkan akhiran berkas
tetapi hal ini tidak dipaksakan atau tergantung sistem operasi, kebanyakan hanya untuk membantu pengguna mengenali jenis isi dari suatu berkas.

Tabel 5-1. Tabel Jenis Berkas

Jenis berkas	Akhiran	Fungsi
executable	exe, com, bat, bin	program yang siap dijalankan
objek	obj, o	bahasa mesin, kode terkompilasi
kode asal (source code)	c, cc, pas, java, asm, a	kode asal dari berbagai bahasa
batch	bat, sh	perintah pada shell
text	txt, doc	data text, document
pengolah kata	wpd, tex, doc	format jenis pengolah data
library	lib, a, DLL	library untuk rutin program
print, gambar	ps, dvi, gif	format aSCII atau biner untuk dicetak
archive	arc, zip, tar	beberapa berkas yang dikumpulkan menjadi satu berkas. Terkadang dimampatkan untuk penyimpanan

5.2.5. Struktur Berkas

Kita juga dapat menggunakan jenis berkas untuk mengidentifikasi struktur dalam dari berkas. Berkas berupa source dan objek memiliki struktur yang cocok dengan harapan program yang membaca berkas tersebut. Suatu berkas harus memiliki struktur yang dikenali oleh sistem operasi. Sebagai contoh, sistem operasi menginginkan suatu berkas yang dapat dieksekusi memiliki struktur tertentu agar dapat diketahui dimana berkas tersebut akan ditempatkan di memori dan di mana letak instruksi pertama berkas tersebut. Beberapa sistem operasi mengembangkan ide ini sehingga mendukung beberapa struktur berkas, dengan beberapa operasi khusus untuk memanipulasi berkas dengan struktur tersebut.

Kelemahan memiliki dukungan terhadap beberapa struktur berkas adalah: Ukuran dari sistem operasi dapat menjadi besar, jika sistem operasi mendefinisikan lima struktur berkas yang berbeda maka ia perlu menampung kode untuk yang diperlukan untuk mendukung semuanya. Setiap berkas harus dapat menerapkan salah satu struktur berkas tersebut. Masalah akan timbul ketika terdapat aplikasi yang membutuhkan struktur informasi yang tidak didukung oleh sistem operasi tersebut.

Beberapa sistem operasi menerapkan dan mendukung struktur berkas sedikit struktur berkas. Pendekatan ini digunakan pada MS-DOS dan UNIX. UNIX menganggap setiap berkas sebagai urutan 8-bit byte, tidak ada interpretasi sistem operasi terhadap dari bit-bit ini. Skema tersebut menawarkan fleksibilitas tinggi tetapi dukungan yang terbatas. Setiap aplikasi harus menambahkan sendiri kode untuk menerjemahkan berkas masukan ke dalam struktur yang sesuai. Walau bagaimana pun juga sebuah sistem operasi harus memiliki minimal satu struktur berkas yaitu untuk berkas yang dapat dieksekusi sehingga sistem dapat memuat berkas dalam memori dan menjalankannya.

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

5.2.6. Struktur Berkas Pada Disk

Menempatkan batas dalam berkas dapat menjadi rumit bagi sistem operasi. Sistem disk biasanya memiliki ukuran blok yang sudah ditetapkan dari ukuran sektor. Semua I/O dari disk dilakukan dalam satuan blok dan semua blok ('*physical record*') memiliki ukuran yang sama. Tetapi ukuran dari '*physical record*' tidak akan sama dengan ukuran '*logical*

record'. Ukuran dari '*logical record*' akan bervariasi.Memuatkan beberapa '*logical record*' ke dalam '*physical record*' merupakan solusi umum dari masalah ini.

Sebagai contoh pada sistem operasi UNIX, semua berkas didefinisikan sebagai kumpulan byte. Setiap byte dialamatkan menurut batasnya dari awal berkas sampai akhir. Pada kasus ini ukuran '*logical record*' adalah 1 byte. Sistem berkas secara otomatis memuatkan byte-byte tersebut kedalam blok pada disk.

Ukuran '*logical record*', ukuran blok pada disk, dan teknik untuk memuatkannya menjelaskan berapa banyak '*logical record*' dalam tiap-tiap 'physical record'. Teknik memuatkan dapat dilakukan oleh aplikasi pengguna atau oleh sistem operasi.

Berkas juga dapat dianggap sebagai urutan dari beberapa blok pada disk. Konversi dari '*logical record*' ke 'physical record' merupakan masalah perangkat lunak. Tempat pada disk selalu berada pada blok, sehingga beberapa bagian dari blok terakhir yang ditempati berkas dapat terbuang. Jika setiap blok berukuran 512 byte, sebuah berkas berukuran 1.949 byte akan menempati empat blok (2.048 byte) dan akan tersisa 99 byte pada blok terakhir. Byte yang terbuang tersebut dipertahankan agar ukuran dari unit tetap blok bukan byte disebut fragmentasi dalam disk.

Semua sistem berkas pasti mempunyai fragmentasi dalam disk, semakin besar ukuran blok akan semakin besar fragmentasi dalam disknya.

5.2.7. Penggunaan Berkas Secara Bersama-sama

Konsistensi semantik adalah parameter yang penting untuk evaluasi sistem berkas yang mendukung penggunaan berkas secara bersama. Hal ini juga merupakan karakterisasi dari sistem yang menspesifikasi semantik dari banyak pengguna yang mengakses berkas secara bersama-sama. Lebih khusus, semantik ini seharusnya dapat menspesifikasi kapan
suatu modifikasi suatu data oleh satu pengguna dapat diketahui oleh pengguna lain.

Terdapat beberapa macam konsistensi semantik. Di bawah ini akan dijelaskan kriteria yang digunakan dalam UNIX.

Berkas sistem UNIX mengikuti konsistensi semantik:

- Penulisan ke berkas yang dibuka oleh pengguna dapat dilihat langsung oleh pengguna lain yang sedang mengakses ke berkas yang sama.
- Terdapat bentuk pembagian dimana pengguna membagi pointer lokasi ke berkas tersebut. Sehingga perubahan pointer satu pengguna akan mempengaruhi semua pengguna *sharing*nya.

5.3. Metode Akses

5.3.1. Akses Secara Berurutan

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

Pada sistem yang lain, contohnya IBM, terdapat banyak dukungan metode akses yang berbeda. Masalah pada sistem tersebut adalah memilih yang mana yang tepat untuk digunakan pada satu aplikasi tertentu.

Sequential Access merupakan metode yang paling sederhana. Informasi yang disimpan dalam berkas diproses berdasarkan urutan. Operasi dasar pada suatu berkas adalah tulis dan baca. Operasi baca membaca berkas dan meningkatkan pointer berkas selama di jalur lokasi I/O. Operasi tulis menambahkan ke akhir berkas dan meningkatkan ke akhir berkas yang baru. Metode ini didasarkan pada tape model sebuah berkas, dan dapat bekerja pada kedua jenis *device* akses (urut mau pun acak).

5.3.2. Akses Langsung

Direct Access merupakan metode yang membiarkan program membaca dan menulis dengan cepat pada berkas yang dibuat dengan *fixed-length logical order* tanpa adanya urutan. Metode ini sangat berguna untuk mengakses informasi dalam jumlah besar. Biasanya database memerlukan hal seperti ini. Operasi berkas pada metode ini harus dimodifikasi untuk menambahkan nomor blok sebagai parameter.

Pengguna menyediakan nomor blok ke sistem operasi biasanya sebagai nomor blok relatif, yaitu indeks relatif terhadap awal berkas. Penggunaan nomor blok relatif bagi sistem operasi adalah untuk memutuskan lokasi berkas diletakkan dan membantu mencegah pengguna dari pengaksesan suatu bagian sistem berkas yang bukan bagian pengguna tersebut.

5.3.3. Akses Dengan Menggunakan Indeks

Metode ini merupakan hasil dari pengembangan metode *direct access*. Metode ini memasukkan indeks untuk mengakses berkas. Jadi untuk mendapatkan suatu informasi suatu berkas, kita mencari dahulu di indeks, lalu menggunakan pointer untuk mengakses berkas dan mendapatkan informasi tersebut. Namun metode ini memiliki kekurangan, yaitu apabila berkas-berkas besar, maka indeks berkas tersebut akan semakin besar. Jadi solusinya adalah dengan membuat 2 indeks, indeks primer dan indeks sekunder.

Indeks primer memuat pointer ke indeks sekunder, lalu indeks sekunder menunjuk ke data yang dimaksud.

5.4. Struktur Direktori

5.4.1. Operasi Pada Direktori

Operasi-operasi yang dapat dilakukan pada direktori adalah:

1. Mencari berkas, kita dapat menemukan sebuah berkas didalam sebuah struktur direktori. Karena berkas-berkas memiliki nama simbolik dan nama yang sama dapat mengindikasikan keterkaitan antara setiap berkas-berkas tersebut, mungkin kita berkeinginan untuk dapat menemukan seluruh berkas yang nama-nama berkas membentuk pola khusus.
2. Membuat berkas, kita dapat membuat berkas baru dan menambahkan berkas tersebut kedalam direktori.
3. Menghapus berkas, apabila berkas sudah tidak diperlukan lagi, kita dapat menghapus berkas tersebut dari direktori.

4. Menampilkan isi direktori, kita dapat menampilkan seluruh berkas dalam direktori, dan kandungan isi direktori untuk setiap berkas dalam daftar tersebut.

5. Mengganti nama berkas, karena nama berkas merepresentasikan isi dari berkas kepada user, maka user dapat merubah nama berkas ketika isi atau penggunaan berkas berubah. Perubahan nama dapat merubah posisi berkas dalam direktori.
6. Melintasi sistem berkas, ini sangat berguna untuk mengakses direktori dan berkas didalam struktur direktori.

5.4.2. Direktori Satu Tingkat

Ini adalah struktur direktori yang paling sederhana. Semua berkas disimpan di dalam direktori yang sama. Struktur ini tentunya memiliki kelemahan jika jumlah berkasnya bertambah banyak, karena tiap berkas mesti memiliki nama yang unik.

5.4.3. Direktori Dua Tingkat

Kelemahan yang ada pada direktori tingkat satu dapat diatas pada sistem direktori dua tingkat. Caranya ialah dengan membuat direktori secara terpisah. Pada direktori tingkat dua, setiap pengguna memiliki direktori berkas sendiri (UFD). Setiap UFD memiliki struktur yang serupa, tapi hanya berisi berkas-berkas dari seorang pengguna.

Ketika seorang pengguna *login*, *master* direktori berkas (MFD) dicari. Isi dari MFD adalah indeks dari nama pengguna atau nomor rekening, dan tiap entri menunjuk pada UFD untuk pengguna tersebut.

Ketika seorang pengguna ingin mengakses suatu berkas, hanya UFD-nya sendiri yang diakses. Jadi pada setiap UFD yang berbeda, boleh terdapat nama berkas yang sama.

5.4.4. Direktori Dengan Struktur *Tree*

Struktur direktori dua tingkat bisa dikatakan sebagai pohon dua tingkat. Sebuah direktori dengan struktur pohon memiliki sejumlah berkas atau subdirektori lagi. Pada penggunaan yang normal setiap pengguna memiliki direktorinya sendiri-sendiri. Selain itu pengguna tersebut dapat memiliki subdirektori sendiri lagi.

Dalam struktur ini dikenal dua istilah, yaitu *path* relatif dan *path* mutlak. *Path* relatif adalah *path* yang dimulai dari direktori yang aktif. Sedangkan *path* mutlak adalah *path* yang dimulai dari direktori akar.

5.4.5. Direktori Dengan Struktur *Acyclic-Graph*

Jika ada sebuah berkas yang ingin diakses oleh dua pengguna atau lebih, maka struktur ini menyediakan fasilitas "*sharing*", yaitu penggunaan sebuah berkas secara bersama-sama. Hal ini tentunya berbeda dengan struktur pohon, dimana pada struktur tersebut penggunaan berkas atau direktori secara bersama-sama dilarang. Pada struktur "*Acyclic-Graph*", penggunaan berkas atau direktori secara bersama-sama diperbolehkan. Tapi pada umumnya struktur ini mirip dengan struktur pohon.

5.4.6. Direktori Dengan Struktur *Graph*

Masalah yang sangat utama pada struktur direktori "*Acyclic-Graph*" adalah kemampuan untuk memastikan tidak-adanya siklus. Jika pada struktur 2 tingkat direktori,

seorang pengguna dapat membuat subdirektori, maka akan kita dapatkan direktori dengan struktur pohon. Sangatlah mudah untuk tetap mempertahankan sifat pohon setiap kali ada penambahan berkas atau subdirektori pada direktori dengan struktur pohon. Tapi jika kita menambahkan sambungan pada direktori dengan struktur pohon, maka akan kita dapatkan direktori dengan struktur *graph* sederhana.

Proses pencarian pada direktori dengan struktur "*Acyclic-Graph*", apabila tidak ditangani dengan baik (algoritma tidak bagus) dapat menyebabkan proses pencarian yang berulang dan menghabiskan banyak waktu. 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 sebagai tempat kosong. Hal ini tentunya dapat menghabiskan banyak waktu.

5.5. Proteksi Berkas

Ketika kita menyimpan informasi dalam sebuah sistem komputer, ada dua hal yang harus menjadi perhatian utama kita. Hal tersebut adalah:

1. Reabilitas dari sebuah sistem
   Maksud dari reabilitas sistem adalah kemampuan sebuah sistem untuk melindungi informasi yangtelah disimpan agar terhindar dari kerusakan, dalam hal ini adalah perlindungan secara fisik pada sebuah berkas. Reabilitas sistem dapat dijaga dengan membuat cadangan dari setiap berkas secaramanual atau pun otomatis, sesuai dengan layanan yang dari sebuah sistem operasi. Reabilitas Sistemakan dibahas lebih lanjut pada Bagian 6.10.

2. Proteksi (Perlindungan) terhadap sebuah berkas
   Perlindungan terhadap berkas dapat dilakukan dengan berbagai macam cara. Pada bagian ini, kita akan membahas secara detil mekanisme yang diterapkan dalam melindungi sebuah berkas.

5.5.1. Tipe Akses Pada Berkas

Salah satu cara untuk melindungi berkas dalam komputer kita adalah dengan melakukan pembatasan akses pada berkas tersebut. Pembatasan akses yang dimaksudkan adalah kita, sebagai pemilik dari sebuah berkas, dapat menentukan operasi apa saja yang dapat dilakukan oleh pengguna lain terhadap berkas tersebut. Pembatasan ini berupa sebuah permission atau pun not permitted operation, tergantung pada kebutuhan pengguna lain terhadap berkas tersebut. Di bawah ini adalah beberapa operasi berkas yang dapat diatur aksesnya:

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

Selain operasi-operasi berkas diatas, perlindungan terhadap berkas dapat dilakukan dengan mekanisme yang lain. Namun setiap mekanisme memiliki kelebihan dan

kekurangan. Pemilihan mekanisme sangatlah tergantung pada kebutuhan dan spesifikasi sistem.

5.5.2. Akses List dan Group

Hal yang paling umum dari sistem proteksi adalah membuat akses tergantung pada identitas pengguna yang bersangkutan. Implementasi dari akses ini adalah dengan membuat daftar akses yang berisi keterangan setiap pengguna dan keterangan akses berkas dari pengguna yang bersangkutan. Daftar akses ini akan diperiksa setiap kali seorang pengguna meminta akses ke sebuah berkas. Jika pengguna tersebut memiliki akses yang diminta pada berkas tersebut, maka diperbolehkan untuk mengakses berkas tersebut. Proses ini juga berlaku untuk hal yang sebaliknya. Akses pengguna terhadap berkas akan ditolak, dan sistem operasi akan mengeluarkan peringatan *Protection Violation*.

Masalah baru yang timbul adalah panjang dari daftar akses yang harus dibuat. Seperti telah disebutkan, kita harus mendaftarkan semua pengguna dalam daftar akses tersebut hanya untuk akses pada satu berkas saja. Oleh karena itu, teknik ini mengakibatkan 2 konsekuensi yang tidak dapat dihindarkan:

1. Pembuatan daftar yang sangat panjang ini dapat menjadi pekerjaan yang sangat melelahkan sekaligus membosankan, terutama jika jumlah pengguna dalam sistem tidak dapat diketahui secara pasti.
2. Manajemen ruang *harddisk* yang lebih rumit, karena ukuran sebuah direktori dapat berubah-ubah, tidak memiliki ukuran yang tetap.

Kedua konsekuensi diatas melahirkan sebuah teknik daftar akses yang lebih singkat. Teknik inimengelompokkan pengguna berdasarkan tiga kategori:

1. Owner: User yang membuat berkas.
2. Group: Sekelompok pengguna yang memiliki akses yang sama terhadap sebuah berkas, atau men-share sebuah berkas.
3. Universe: Seluruh pengguna yang terdapat dalam sistem komputer.

Dengan adanya pengelompokkan pengguna seperti ini, maka kita hanya membutuhkan tiga *field* untukmelindungi sebuah berkas. Field ini diasosiasikan dengan 3 buah bit untuk setiap kategori. Dalam sistem UNIX dikenal bit rwx dengan bit r untuk mengontrol akses baca, bit w sebagai kontrol menulis dan bit x sebagai bit kontrol untuk pengeksekusian. Setiap *field* dipisahkan dengan *field separator*. Dibawah ini adalah contoh dari sistem proteksi dengan daftar akses pada sistem UNIX.

Tabel 5-2. Contoh sistem daftar akses pada UNIX

drwx	rwx	rwx	1	pbg	staff	512	Apr16 22.25	bekas.txt
owner	group	universe		group	owner	ukuran	waktu	Nama berkas

5.5.3. Pendekatan Sistem Proteksi yang Lain

Sistem proteksi yang lazim digunakan pada sistem komputer selain diatas adalah dengan menggunakan password (kata sandi) pada setiap berkas. Beberapa sistem operasi mengimplementasikan hal ini bukan hanya pada berkas, melainkan pada direktori. Dengan sistem ini, sebuah berkas tidak akan dapat diakses selain oleh pengguna yang telah mengetahui password untuk berkas tersebut. Akan tetapi, masalah yang muncul dari sistem ini adalah jumlah password yang harus diingat oleh seorang pengguna untuk mengakses berkas dalam sebuah sistem operasi. Masalah yang lain

adalah keamanan password itu sendiri. Jika hanya satu password yang digunakan, maka kebocoran password tersebut merupakan malapetaka bagi pengguna yang bersangkutan. Sekali lagi, maka kita harus menggunakan password yang berbeda untuk setiap tingkatan yang berbeda.

5.6. Struktur Sistem Berkas

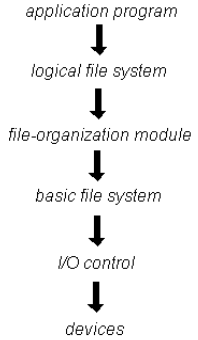
Disk menyediakan sebagian besar tempat penyimpanan dimana sistem berkas dikelola dikelola. Untuk meningkatkan efisiensi I/O, pengiriman data antara memori dan disk dilakukan dalam setiap blok. Setiap blok merupakan satu atau lebih sektor. Setiap disk memiliki ukuran yang berbeda-beda, biasanya berukuran 512 bytes. Disk memiliki dua karakteristik penting yang menjadikan disk sebagai media yang tepat untuk menyimpan berbagai macam berkas, yaitu:

• Disk tersebut dapat ditulis ulang di disk tersebut, hal ini memungkinkan untuk membaca, memodifikasi, dan menulis di disk tersebut.
• Dapat diakses langsung ke setiap blok di disk. Hal ini memudahkan untuk mengakses setiap berkas baik secara berurut mau pun tidak berurut, dan berpindah dari satu berkas ke berkas lain dengan hanya mengangkat head disk dan menunggu disk berputar.

5.6.1. Organisasi Sistem Berkas

Sistem operasi menyediakan sistem berkas agar data mudah disimpan, diletakkan dan diambil kembali dengan mudah. Terdapat dua masalah desain dalam membangun suatu sistem berkas. Masalah pertama adalah definisi dari sistem berkas. Hal ini mencakup definisi berkas dan atributnya, operasi ke berkas, dan struktur direktori dalam mengorganisasikan berkas-berkas. Masalah kedua adalah membuat algoritma dan struktur data yang memetakan struktur logikal sistem berkas ke tempat penyimpanan sekunder. Pada dasarnya sistem berkas tersusun atas beberapa tingkatan, yaitu (dari yang terendah):

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



Gambar 5-1. Lapisan pada sistem berkas.

5.6.2. *Mounting* Sistem Berkas

Seperti halnya sebuah berkas yang harus dibuka terlebih dahulu sebelum digunakan, sistem berkas harus di *mount* terlebih dahulu sebelum sistem berkas tersebut siap untuk memproses dalam sistem. Sistem operasi diberikan sebuah alamat mounting (*mount point*) yang berisi nama *device* yang bersangkutan dan lokasi dari *device* tersebut.

5.7. Metode Alokasi Berkas

Kemudahan dalam mengakses langsung suatu disk memberikan fleksibilitas dalam mengimplementasikan sebuah berkas. Masalah utama dalam implementasi adalah bagaimana mengalokasikan berkas-berkas ke dalam disk, sehingga disk dapat terutilisasi dengan efektif dan berkas dapat diakses dengan cepat. Ada tiga metode utama, menurut buku "*Applied Operating System Concepts: First Edition*" oleh Avi Silberschatz, Peter Galvin dan Greg Gagne untuk mengalokasi ruang disk yang digunakan secara luas yaitu, *contiguous*, *linked*, dan *indexed*.

5.7.1. Alokasi Secara Berdampingan *(Contiguous Allocation)*

Metode ini menempatkan setiap berkas pada satu himpunan blok yang berurut di dalam disk. Alamat disk menyatakan sebuah urutan linier. Dengan urutan linier ini maka *head disk* hanya bergerak jika mengakses dari sektor terakhir suatu silinder ke sektor pertama silinder berikutnya. Waktu pencarian (*seek time*) dan banyak *disk seek* yang dibutuhkan untuk mengakses berkas yang di alokasi secara berdampingan ini sangat minimal. Contoh dari sistem operasi yang menggunakan *contiguous allocation* adalah IBM VM/ CMS karena pendekatan ini menghasilkan performa yang baik.

Contiguous allocation dari suatu berkas diketahui melalui alamat dan panjang disk (dalam unit blok) dari blok pertama. Jadi, misalkan ada berkas dengan panjang n blok dan mulai dari lokasi b maka berkas tersebut menempati blok b, b+1, b+2, ..., b+n-1. Direktori untuk setiap berkas mengindikasikan alamat blok awal dan panjang area yang dialokasikan untuk berkas tersebut. Terdapat dua macam cara untuk mengakses berkas yang dialokasi dengan metode ini, yaitu:

- *Sequential access*, sistem berkas mengetahui alamat blok terakhir dari disk dan membaca blok berikutnya jika diperlukan.
- *Direct access*, untuk akses langsung ke blok i dari suatu berkas yang dimulai pada blok b, dapat langsung mengakses blok b+i.

Kesulitan dari metode alokasi secara berdampingan ini adalah menemukan ruang untuk berkas baru. Masalah pengalokasian ruang disk dengan metode ini merupakan aplikasi masalah dari *dynamic storage-allocation* (alokasi tempat penyimpanan secara dinamik), yaitu bagaimana memenuhi permintaan ukuran n dari daftar ruang kosong. Strategi-strategi yang umum adalah *first fit* dan *best fit*. Kedua strategi tersebut mengalami masalah fragmentasi eksternal, dimana jika berkas dialokasi dan dihapus maka ruang kosong disk terpecah menjadi kepingan-kepingan kecil. Hal ini akan menjadi masalah ketika banyak kepingan kecil tidak dapat memenuhi permintaan karena kepingan-kepingan kecil tidak cukup besar untuk menyimpan berkas, sehingga terdapat banyak ruang yang terbuang.

Masalah yang lain adalah menentukan berapa banyak ruang yang diperlukan untuk suatu berkas. Ketika berkas dibuat, jumlah dari ruang berkas harus ditentukan dan dialokasikan. Jika ruang yang dialokasikan terlalu kecil maka berkas tidak dapat

diperbesar dari yang telah dialokasikan. Untuk mengatasi hal ini ada dua kemungkinan. Pertama, program pengguna dapat diakhiri dengan pesan error yang sesuai. Lalu, pengguna harus mengalokasikan tambahan ruang dan menjalankan programnya lagi, tetapi hal ini *cost* yang dihasilkan lebih mahal. Untuk mengatasinya, pengguna dapat melakukan estimasi yang lebih terhadap ruang yang harus dialokasikan pada suatu berkas tetapi hal ini akan membuang ruang disk. Kemungkinan yang kedua adalah mencari ruang kosong yang lebih besar, lalu menyalin isi dari berkas ke ruang yang baru dan mengkosongkan ruang yang sebelumnya. Hal ini menghabiskan waktu yang cukup banyak. Walau pun jumlah ruang yang diperlukan untuk suatu berkas dapat diketahui, pengalokasian awal akan tidak efisien. Ukuran berkas yang bertambah dalam periode yang lama harus dapat dialokasi ke ruang yang cukup untuk ukuran akhirnya, walau pun ruang tersebut tidak akan digunakan dalam waktu yang lama. Hal ini akan menyebabkan berkas dengan jumlah fragmentasi internal yang besar.

Untuk menghindari hal-hal tersebut, beberapa sistem operasi memodifikasi skema metode alokasi secara berdampingan, dimana kepingan kecil yang berurut dalam ruang disk diinisialisasi terlebih dahulu, kemudian ketika jumlah ruang disk kurang besar, kepingan kecil yang berurut lainnya, ditambahkan pada alokasi awal. Kejadian seperti ini disebut perpanjangan. Fragmentasi internal masih dapat terjadi jika perpanjangan-perpanjangan ini terlalu besar dan fragmentasi eksternal masih menjadi masalah begitu perpanjangan-perpanjangan dengan ukuran yang bervariasi dialokasikan dan didealokasi.

5.7.2. Alokasi Secara Berangkai (*Linked Allocation*)

Metode ini menyelesaikan semua masalah yang terdapat pada contiguous allocation. Dengan metode ini, setiap berkas merupakan *linked list* dari blok-blok disk, dimana blok-blok disk dapat tersebar di dalam disk. Setiap direktori berisi sebuah penunjuk (*pointer*) ke awal dan akhir blok sebuah berkas. Setiap blok mempunyai penunjuk ke blok berikutnya. Untuk membuat berkas baru, kita dengan mudah membuat masukan baru dalam direktori. Dengan metode ini, setiap direktori masukan mempunyai penunjuk ke awal blok disk dari berkas. Penunjuk ini diinisialisasi menjadi *nil* (nilai penunjuk untuk akhir dari list) untuk menandakan berkas kosong. Ukurannya juga diset menjadi 0. Penulisan suatu berkas menyebabkan ditemukannya blok yang kosong melalui sistem manajemen ruang kosong (*free-space management system*), dan blok baru ini ditulis dan disambungkan ke akhir berkas. Untuk membaca suatu berkas, cukup dengan membaca blok-blok dengan mengikuti pergerakan penunjuk.

Metode ini tidak mengalami fragmentasi eksternal dan kita dapat menggunakan blok kosong yang terdapat dalam daftar ruang kosong untuk memenuhi permintaan pengguna. Ukuran dari berkas tidak perlu ditentukan ketika berkas pertama kali dibuat, sehingga ukuran berkas dapat bertambah selama masih ada blok-blok kosong.

Metode ini tentunya mempunyai kerugian, yaitu metode ini hanya dapat digunakan secara efektif untuk pengaksesan berkas secara sequential (*sequential-access file*). Untuk mencari blok ke-*i* dari suatu berkas, harus dimulai dari awal berkas dan mengikuti penunjuk sampai berada di blok ke-*i*. Setiap akses ke penunjuk akan membaca disk dan kadang melakukan pencarian disk (*disk seek*). Hal ini sangat tidak efisien untuk mendukung kemampuan akses langsung (*direct-access*) terhadap berkas yang menggunakan metode alokasi link. Kerugian yang lain dari metode ini adalah ruang yang harus disediakan untuk penunjuk. Solusi yang umum untuk masalah ini adalah mengumpulkan blok-blok persekutuan terkecil dinamakan *clusters* dan mengalokasikan *cluster-cluster* daripada blok. Dengan solusi ini maka, penunjuk menggunakan ruang disk berkas dengan persentase yang sangat kecil. Metode ini membuat *mapping* logikal ke fisikal blok tetap sederhana, tetapi meningkatkan *disk throughput* dan memperkecil ruang

yang diperlukan untuk alokasi blok dan management daftar kosong (*free-list management*). Akibat dari pendekatan ini adalah meningkatnya fragmentasi internal, karena lebih banyak ruang yang terbuang jika sebuah *cluster* sebagian penuh daripada ketika sebuah blok sebagian penuh. Alasan *cluster* digunakan oleh kebanyakan sistem operasi adalah kemampuannya yang dapat meningkatkan waktu akses disk untuk berbagai macam algoritma.

Masalah yang lain adalah masalah daya tahan metode ini. Karena semua berkas saling berhubungan dengan penunjuk yang tersebar di semua bagian disk, apa yang terjadi jika sebuah penunjuk rusak atau hilang. Hal ini menyebabkan berkas menyambung ke daftar ruang kosong atau ke berkas yang lain. Salah satu solusinya adalah menggunakan *linked list* ganda atau menyimpan nama berkas dan nomor relatif blok dalam setiap blok, tetapi solusi ini membutuhkan perhatian lebih untuk setiap berkas.

Variasi penting dari metode ini adalah penggunaan *file allocation table (FAT)*, yang digunakan oleh sistem operasi *MS-DOS* dan *OS/2*. Bagian awal disk pada setiap partisi disingkirkan untuk menempatkan tabelnya. Tabel ini mempunyai satu masukkan untuk setiap blok disk, dan diberi indeks oleh nomor blok. Masukkan direktori mengandung nomor blok dari blok awal berkas. Masukkan tabel diberi indeks oleh nomor blok itu lalu mengandung nomor blok untuk blok berikutnya dari berkas. Rantai ini berlanjut sampai blok terakhir, yang mempunyai nilai akhir berkas yang khusus sebagai masukkan tabel. Blok yang tidak digunakan diberi nilai 0. Untuk mengalokasi blok baru untuk suatu berkas hanya dengan mencari nilai 0 pertama dalam tabel, dan mengganti nilai akhir berkas sebelumnya dengan alamat blok yang baru. Metode pengalokasian FAT ini dapat menghasilkan jumlah pencarian *head disk* yang signifikan, jika berkas tidak di cache. *Head disk* harus bergerak dari awal partisi untuk membaca FAT dan menemukan lokasi blok yang ditanyakan, lalu menemukan lokasi blok itu sendiri. Kasus buruknya, kedua pergerakan terjadi untuk setiap blok. Keuntungannya waktu random akses meningkat, akibat dari *head disk* dapat mencari lokasi blok apa saja dengan membaca informasi dalam FAT.

5.7.3. Alokasi Dengan Indeks (*Indexed Allocation*)

Metode alokasi dengan berangkai dapat menyelesaikan masalah fragmentasi eksternal dan pendeklarasian ukuran dari metode alokasi berdampingan. Bagaimana pun tanpa FAT, metode alokasi berangkai tidak mendukung keefisiensian akses langsung, karena penunjuk ke bloknya berserakan dengan bloknya didalam disk dan perlu didapatkan secara berurutan. Metode alokasi dengan indeks menyelesaikan masalah ini dengan mengumpulkan semua penunjuk menjadi dalam satu lokasi yang dinamakan blok indeks (*index block*). Setiap berkas mempunyai blok indeks, yang merupakan sebuah larik *array* dari alamat-alamat disk-blok. Direktori mempunyai alamat dari blok indeks. Ketika berkas dibuat, semua penunjuk dalam blok indeks di set menjadi *nil*. Ketika blok ke-*i* pertama kali ditulis, sebuah blok didapat dari pengatur ruang kosong *free-space manager* dan alamatnya diletakkan ke dalam blok indeks ke-*i*. Metode ini mendukung akses secara langsung, tanpa mengalami fragmentasi eksternal karena blok kosong mana pun dalam disk dapat memenuhi permintaan ruang tambahan. Tetapi metode ini dapat menyebabkan ada ruang yang terbuang. Penunjuk yang berlebihan dari blok indeks secara umum lebih besar dari yang terjadi pada metode alokasi berangkai.

Mekanisme untuk menghadapi masalah berapa besar blok indeks yang diperlukan sebagai berikut:

- *Linked scheme*: untuk berkas-berkas yang besar, dilakukan dengan menyambung beberapa blok indeks menjadi satu.

- *Multilevel index*: sebuah varian dari representasi yang berantai adalah dengan menggunakan blok indeks level pertama menunjuk ke himpunan blok indeks level kedua, yang akhirnya menunjuk ke blok-blok berkas.
- *Combined scheme*: digunakan oleh sistem *BSD UNIX* yaitu dengan menetapkan 15 penunjuk dari blok indeks dalam blok indeksnya berkas. 12 penunjuk pertama menunjuk ke *direct blocks* yang menyimpan alamat-alamat blok yang berisi data dari berkas. 3 penunjuk berikutnya menunjuk ke *indirect blocks*. Penunjuk *indirect blok* yang pertama adalah alamat dari *single indirect block*, yang merupakan blok indeks yang berisi alamat-alamat blok yang berisi data. Lalu ada penunjuk *double indirect block* yang berisi alamat dari sebuah blok yang berisi alamat-alamat blok yang berisi penunjuk ke blok data yang sebenarnya.

5.7.4. Kinerja Sistem Berkas

Salah satu kesulitan dalam membandingkan performa sistem adalah menentukan bagaimana sistem tersebut akan digunakan. Sistem yang lebih banyak menggunakan akses sekuensial (berurutan) akan memakai metode yang berbeda dengan sistem yang lebih sering menggunakan akses random (acak).Untuk jenis akses apa pun, alokasi yang berdampingan hanya memerlukan satu akses untuk mendapatkan sebuah blok disk. Karena kita dapat menyimpan *initial address* dari berkas di dalam memori, maka alamat disk pada blok ke-*i* dapat segera dikalkulasi dan dibaca secara langsung.

Untuk alokasi berangkai *(linked list)*, kita juga dapat menyimpan alamat dari blok selanjutnya ke dalam memori, lalu membacanya secara langsung. Metode ini sangat baik untuk akses sekuensial, namun untuk akses langsung, akses menuju blok ke-*i*kemungkinan membutuhkan pembacaan disk sebanyak *i* kali. Masalah ini mengindikasikan bahwa alokasi berangkai sebaiknya tidak digunakan untuk aplikasi yang membutuhkan akses langsung.

Oleh sebab itu, beberapa sistem mendukung akses langsung dengan menggunakan alokasi berdampingan (*contiguous allocation*), serta akses berurutan dengan alokasi berangkai. Untuk sistem-sistem tersebut, jenis akses harus dideklarasikan pada saat berkas dibuat. Berkas yang dibuat untuk akses sekuensial (berurutan) akan dirangkaikan dan tidak dapat digunakan untuk akses langsung. Berkas yang dibuat untuk akses langsung akan berdampingan dan dapat mendukung baik akses langsung mau pun akses berurutan, dengan mendeklarasikan jarak maksimum. Perhatikan bahwa sistem operasi harus mendukung struktur data dan algoritma yang sesuai untuk mendukung kedua metode alokasi di atas.

Alokasi dengan menggunakan indeks lebih rumit lagi. Jika blok indeks telah terdapat dalam memori, akses dapat dilakukan secara langsung. Namun, menyimpan blok indeks dalam memori memerlukan ruang (*space*) yang besar. Jika ruang memori tidak tersedia, maka kita mungkin harus membaca blok indeks terlebih dahulu, baru kemudian blok data yang diinginkan. Untuk indeks dua tingkat, pembacaan dua blok indeks mungkin diperlukan. Untuk berkas yang berukuran sangat besar, mengakses blok di dekat akhir suatu berkas akan membutuhkan pembacaan seluruh blok indeks agar dapat mengikuti rantai penunjuk sebelum blok data dapat dibaca. Dengan demikian, performa alokasi dengan menggunakan indeks ditentukan oleh: struktur indeks, ukuran berkas, dan posisi dari blok yang diinginkan.

Beberapa sistem mengkombinasikan alokasi berdampingan dengan alokasi indeks. Caranya adalah dengan menggunakan alokasi berdampingan untuk berkas berukuran kecil (3-4 blok), dan beralih secara otomatis ke alokasi indeks jika berkas semakin membesar.

5.8. Managemen Ruang Kosong (*Free Space*)

Semenjak hanya tersedia tempat yang terbatas pada disk maka sangat berguna untuk menggunakan kembali tempat dari berkas yang dihapus untuk berkas baru, jika dimungkinkan,karena pada media yang sekali tulis (media optik) hanya dimungkinkan sekali menulis dan menggunakannyanya kembali secara fisik tidak mungkin. Untuk mencatat tempat kosong pada disk, sistem mempunyai daftar tempat kosong (*free space list*). Daftar ini menyimpan semua blok disk yang kosong yang tidak dialokasikan pada sebuah berkas atau direktori. Untuk membuat berkas baru, sistem mencari ke daftar tersebut untuk mencarikan tempat kosong yang di butuhkan, lalu tempat tersebut dihilangkan dari daftar. Ketika berkas dihapus, alamat berkas tadi ditambahkan pada daftar.

5.8.1. Menggunakan Bit Vektor

Seringnya daftar raung kosong diimplementasikan sebagai bit map atau bit vektor. Tiap blok direpresentasikan sebagai 1 bit. Jika blok tersebut kosong maka isi bitnya 1 dan jika bloknya sedang dialokasikan maka isi bitnya 0. Sebagai contoh sebuah disk dimana blok 2, 3, 4, 5, 8, 9, 10, 11, 12, 13, 17, 18, 25, 26 dan 27 adalah kosong, dan sisanya dialokasikan.

Bit mapnya akan seperti berikut:

001111001111110001100000011100000...

Keuntungan utama dari pendekatan ini adalah relatif sederhana dan efisien untuk mencari blok pertama yang kosong atau berturut-turut n blok yang kosong pada disk. Banyak komputer yang menyediakan instruksi manipulasi bit yang dapat digunakan secara efektif untuk tujuan ini. Sebagai contohnya, dari keluarga prosesor Intel dimulai dari 80386 dan keluarga Motorola dimulai dari 68020 (prosesor yang ada di PC dan Macintosh) mempunyai instruksi yang mengembalikan jarak di *word* dari bit pertama dengan nilai 1. Sistem operasi Apple Macintosh menggunakan metode bit vektor untuk mengalokasikan tempat pada disk. Dalam hal ini perangkat keras mendukung perangkat lunak tetapi bit vektor tidak efisien kecuali seluruh vektor disimpan dalam memori utama (dan ditulis di disk untuk kebutuhan pemulihan). Menyimpan dalam memori utama dimungkinkan untuk disk yang kecil pada mikro komputer, tetapi tidak untuk disk yang besar. Sebuah *disk* 1,3 GB dengan 512-byte blok akan membutuhkan *bit map* sebesar 332K untuk mencatat blok yang kosong.

5.8.2. *Linked List*

Pendekatan lain adalah untuk menghubungkan semua blok yang kosong, menyimpan *pointer* ke blok pertama yang kosong di tempat yang khusus pada disk dan menyimpannya di memori. Blok pertama ini menyimpan *pointer* ke blok kosong berikutnya dan seterusnya. Pada contoh sebelumnya kita akan menyimpan *pointer* ke blok ke 2 sebagai blok kosong pertama, blok 2 akan menyimpan *pointer* ke blok 3, yang akan menunjuk ke blok 4 dan seterusnya. Bagaimana pun metode ini tidak efisien karena untuk *traverse* daftar tesebut kita perlu membaca tiap blok yang membutuhkan waktu I/O. Untungnya traverse ini tidak sering digunakan. Umumnya, sistem operasi membutuhkan blok kosong untuk mengalokasikan blok tersebut ke berkas, maka blok pertama pada daftar ruang kosong digunakan.

5.8.3. *Grouping*

Modifikasi lainnya adalah dengan menyimpan alamat dari n blok kosong pada blok kosong pertama. Pada n-1 pertama dari blok-blok ini adalah kosong. Blok terakhir menyimpan alamat n blok kosong lainnya dan seterusnya. Keuntungannya dari implementasi seperti ini adalah alamat dari blok kosong yang besar sekali dapat ditemukan dengan cepat, tidak seperti pendekatan standar *linked-list*.

5.8.4. *Counting*

Pendekatan lain adalah dengan mengambil keuntungan dari fakta bahwa beberapa blok yang berkesinambungan akan dialokasikan atau dibebaskan secara simultan. Maka dari itu dari pada menyimpan daftar dari banyak alamat disk, kita dapat menyimpan alamat dari blok kosong pertama dan jumlah dari blok kosong yang berkesinambungan yang mengikuti blok kosong pertama. Tiap isi dari daftar menyimpan alamat disk dan penghitung (*counter*). Meski pun setiap isi membutuhkan tempat lebih tetapi secara keseluruhan daftar akan lebih pendek, selama *count* lebih dari satu.

5.9. Implementasi Direktori

Pemilihan dalam algoritma alokasi direktori dan manajemen direktori mempunyai efek yang besar dalam efisiensi, performa, dan kehandalan dari sistem berkas.

5.9.1. *Linear List*

Metode paling sederhana dalam mengimplementasikan sebuah direktori adalah dengan menggunakan *linear list* dari nama berkas dengan penunjuk ke blok data. *Linear list* dari direktori memerlukan pencarian searah untuk mencari suatu direktori didalamnya. Metode sederhana untuk di program tetapi memakan waktu lama ketika dieksekusi. Untuk membuat berkas baru kita harus mencari di dalam direktori untuk meyakinkan bahwa tidak ada berkas yang bernama sama. Lalu kita tambahkan sebuah berkas baru pada akhir direktori. Untuk menghapus sebuah berkas, kita mencari berkas tersebut dalam direktori, lalu melepaskan tempat yang dialokasikan untuknya. Untuk menggunakan kembali suatu berkas dalam direktori kita dapat melakukan beberapa hal. Kita dapat menandai berkas tersebut sebagai tidak terpakai (dengan menamainya secara khusus, seperti nama yang kosong, atau bit terpakai atau tidak yang ditambahkan pada berkas), atau kita dapat menambahkannya pada daftar direktori bebas. Alternatif lainnya kita dapat menyalin ke tempat yang dikosongkan pada direktori. Kita juga bisa menggunakan linked list untuk mengurangi waktu untuk menghapus berkas. Kelemahan dari linear list ini adalah percarian searah untuk mencari sebuah berkas. Direktori yang berisi informasi sering digunakan, implementasi yang lambat pada cara aksesnya akan menjadi perhatian pengguna. Faktanya, banyak sistem operasi mengimplementasikan ’*software cache’* untuk menyimpan informasi yang paling sering digunakan. Penggunaan ’*cache*’ menghindari pembacaan informasi berulang-ulang pada disk. Daftar yang telah diurutkan memperbolehkan pencarian biner dan mengurangi waktu rata-rata pencarian.

Bagaimana pun juga penjagaan agar daftar tetap terurut dapat merumitkan operasi pembuatan dan penghapusan berkas, karena kita perlu memindahkan sejumlah direktori untuk mengurutkannya. *Tree* yang lebih lengkap dapat membantu seperti B-tree. Keuntungan dari daftar yang terurut adalah kita dapatkan daftar direktori yang terurut tanpa pengurutan yang terpisah.

5.9.2. *Hash Table*

Struktur data lainnya yang juga digunakan untuk direktori berkas adalah *hash table*. Dalam metode ini linear list menyimpan direktori, tetapi struktur data hash juga digunakan. *Hash table* mengambil nilai yang dihitung dari nama berkas dan mengembalikan sebuah penunjuk ke nama berkas yang ada di-*linear list*. Maka dari itu dapat memotong banyak biaya pencarian direktori. Memasukkan dan menghapus berkas juga lebih mudah dan cepat. Meski demikian beberapa aturan harus dibuat untuk mncegah tabrakan, situasi dimana dua nama berkas pada *hash* mempunyai tempat yang sama. Kesulitan utama dalam *hash table* adalah ukuran tetap dari *hash table* dan ketergantungan dari fungsi *hash* dengan ukuran *hash table*. Sebagai contoh, misalkan kita membuat suatu *linear-probing hash table* yang dapat menampung 64 data. Fungsi *hash* mengubah nama berkas menjadi nilai dari 0 sampai 63. Jika kita membuat berkas ke 65 maka ukuran tabel *hash* harus diperbesar sampai misalnya 128 dan kita membutuhkan suatu fungsi *hash* yang baru yang dapat memetakan nama berkas dari jangkauan 0 sampai 127, dan kita harus mengatur data direktori yang sudah ada agar memenuhi fungsi *hash* yang baru.

Sebagai alternatif dapat digunakan *chained-overflow hash table*, setiap *hash table* mempunyai daftar yang terkait (*linked list*) dari pada nilai individual dan kita dapat mengatasi tabrakan dengan menambah tempat pada daftar terkait tersebut. Pencarian dapat menjadi lambat, karena pencarian nama memerlukan tahap pencarian pada daftar terkait. Tetapi operasi ini lebih cepat dari pada pencarian linear terhadap seluruh direktori.

5.10. Efisiensi dan Unjuk Kerja

Setelah kita membahas alokasi blok dan pilihan manajemen direktori maka dapat dibayangkan bagaimana efek mereka dalam keefisiensian dan unjuk kerja penggunaan disk. Hal ini dikarenakan disk selalu menjadi "bottle-neck" dalam unjuk kerja sistem.

5.10.1. Efisiensi

Disk dapat digunakan secara efisien tergantung dari teknik alokasi disk serta algoritma pembentukan direktori yang digunakan. Contoh, pada UNIX, direktori berkas dialokasikan terlebih dahulu pada partisi. Walau pun disk yang kosong pun terdapat beberapa persen dari ruangnya digunakan untuk direktori tersebut. Unjuk kerja sistem berkas meningkat akibat dari pengalokasian awal dan penyebaran direktori ini pada partisi. Sistem berkas UNIX melakukan ini agar blok-blok data berkas selalu dekat dengan blok direktori berkas sehingga waktu pencariannya berkurang.

Ada pula keefesiensian pada ukuran penunjuk yang digunakan untuk mengakses data. Masalahnya dalam memilih ukuran penunjuk adalah merencanakan efek dari perubahan teknologi. Masalah ini diantisipasi dengan menginisialisasi terlebih dahulu sistem berkasnya dengan alasan keefisiensian.

Pada awal, banyak struktur data dengan panjang yang sudah ditentu kan dan dialokasi pada ketika sistem dijalankan. Ketika tabel proses penuh maka tidak ada proses lain yang dapat dibuat. Begitu juga dengan tabel berkas ketika penuh, tidak ada berkas yang dapat dibuka. Hal ini menyebabkan sistem gagal melayani permintaan pengguna. Ukuran tabel-tabel ini dapat ditingkatkan hanya dengan mengkompilasi ulang kernel dan boot ulang sistemnya. Tetapi sejak dikeluarkannya Solaris 2, hampir setiap struktur kernel dialokasikan secara dinamis sehingga menghapus batasan buatan pada unjuk kerja sistem.

5.10.2. Kinerja

Ketika metode dasar disk telah dipilih, maka masih ada beberapa cara untuk meningkatkan unjuk kerja. Salah satunya adalah dengan mengguna kan cache, yang merupakan memori lokal pada pengendali disk, dimana cache cukup besar untuk menampung seluruh track pada satu waktu. Beberapa sistem mengatur seksi terpisah dari memori utama untuk disk-cache, yang diasumsikan bahwa blok-blok disimpan karena mereka akan digunakan dalam waktu dekat. Ada juga sistem yang menggunakan memori fisik yang tidak digunakan sebagai penyangga yang dibagi atas sistem halaman (paging) dan sistem disk-blok cache. Suatu sistem melakukan banyak operasi I/O akan menggunakan sebagian banyak memorinya sebagai blok cache, dimana suatu sistem mengeksekusi banyak program akan menggunakan sebagian besar memori-nya untuk ruang halaman.

Beberapa sistem mengoptimalkan disk-cache nya dengan menggunakan berbagai macam algoritma penempatan ulang (replacement algorithms), tergantung dari macam tipe akses dari berkas. Pada akses yang sekuensial dapat dioptimasi dengan teknik yang dikenal dengan nama free-behind dan read-ahead. Free-behind memindahkan sebuah blok dari penyangga secepatnya ketika blok berikutnya diminta. Hal ini dilakukan karena blok sebelumnya tidak lagi digunakan sehingga akan membuang ruang yang ada di penyangga. Sedangkan dengan read ahead, blok yang diminta dan beberapa blok berikutnya dibaca dan ditempatkan pada cache. Hal ini dilakukan karena kemungkinan blok-blok berikutnya akan diminta setelah blok yang sedang diproses. Hal ini juga mem beri dampak pada waktu yang digunakan akan lebih cepat.

Metode yang lain adalah dengan membagi suatu seksi dari memori untuk disk virtual atau RAM disk. Pada RAM disk terdapat operasi-operasi standar yang terdapat pada disk, tetapi semua operasi tersebut terjadi di dalam suatu seksi memori, bukan pada disk. Tetapi, RAM disk hanya berguna untuk penyimpanan sementara, karena jika komputer di boot ulang atau listrik mati maka isi dalam RAM disk akan terhapus.

Perbedaan antara RAM disk dan disk cache adalah dalam masalah siapa yang mengendalikan disk tersebut. RAM disk dikendalikan oleh peng guna sepenuhnya, sedangkan disk cache dikendalikan oleh sistem operasi.

5.11. *Recovery*

Karena semua direktori dan berkas disimpan di dalam memori utama dan disk, maka kita perlu memastikan bahwa kegagalan pada sistem tidak menyebabkan hilangnya data atau data menjadi tidak konsiten.

5.11.1. Pemeriksaan Rutin

Informasi direktori pada memori utama pada umumnya lebih up to date daripada informasi yang terdapat di disk dikarenakan penulisan dari informasi direktori cached ke disk tidak langsung terjadi pada saat setelah peng-update-an terjadi. Consistency checker membandingkan data yang terdapat di struktur direktori dengan blok-blok data pada disk, dan mencoba memperbaiki semua ketidak konsistensian yang terjadi akibat crash-nya komputer. Algoritma pengalokasian dan management ruang kosong menentukan tipe dari masalah yang ditemukan oleh checker dan seberapa sukses dalam memperbaiki masalah-masalah tersebut.

5.11.2. *Back Up and Restore*

Karena kadang-kadang magnetik disk gagal, kita harus memastikan bahwa datanya tidak hilang selamanya. Karena itu, kita menggunakan program sistem untuk mem-back up data dari disk ke alat penyimpanan yang lain seperti floopy disk, magnetic tape, atau optical disk. Pengembalian berkas-berkas yang hilang hanya masalah menempatkan lagi data dari back up data yang telah dilakukan.

Untuk meminimalisir penyalinan, kita dapat menggunakan informasi dari setiap masukan direktori berkas. Umpamanya, jika program back up mengetahui bahwa back up terakhir dari berkas sudah selesai dan penulisan terakhir pada berkas dalam direktori menandakan berkas tidak terjadi perubahan maka berkas tidak harus disalin lagi. Penjadualan back up yang umum sebagai berikut:

- Hari 1: Salin ke tempat penyimpanan back up semua berkas dari disk, disebut sebuah full backup.
- Hari 2: Salin ke tempat penyimpanan lain semua berkas yang berubah sejak hari 1, disebut incremental backup.
- Hari 3: Salin ke tempat peyimpanan lain semua berkas yang berubah sejak hari 2.
- Hari N: salin ke tempat penyimpanan lain semua berkas yang berubah sejak hari N-1, lalu kembali ke hari 1.

Keuntungan dari siklus backup ini adalah kita dapat menempatkan kembali berkas mana pun yang tidaksengaja terhapus pada waktu siklus dengan mendapatkannya dari back up hari sebelumnya. Panjang dari siklus disetujui antara banyaknya tempat penyimpanan backup yang diperlukan dan jumlah hari kebelakang dari penempatan kembali dapat dilakukan.

Ada juga kebiasaan untuk mem-backup keseluruhan dari waktu ke waktu untuk disimpan selamanya daripada media backupnya digunakan kembali. Ada bagusnya menyimpan backup-backup permanent ini di lokasi yang jauh dari backup yang biasa, untuk menghindari kecelakaan seperti kebakaran dan lain-lain. Dan jangan menggunakan kembali media backup terlalu lama karena media tersebut akan rusak jika terlalu sering digunakan kembali.

5.12. Macam-macam Sistem Berkas

5.12.1. Sistem Berkas Pada Windows

Direktori dan Berkas

Sistem operasi Windows merupakan sistem operasi yang telah dikenal luas. Sistem operasi ini sangat memudahkan para penggunanya dengan membuat struktur direktori yang sangat *user-friendly*. Para pengguna Windows tidak akan menemui kesulitan dalam menggunakan sistem direktori yang telah dibuat oleh Microsoft. Windows menggunakan sistem *drive letter* dalam merepresentasikan setiap partisi dari *disk*. Sistem operasi secara otomatis akan terdapat dalam partisi pertama yang diberi label *drive* C.

Sistem operasi Windows dibagi menjadi dua keluarga besar, yaitu keluarga Windows 9x dan keluarga Windows NT (New Technology).

Direktori yang secara otomatis dibuat dalam instalasi Windows adalah:

1. Direktori C:\WINDOWS

Direktori ini berisikan sistem dari Windows. Dalam direktori ini terdapat pustaka-pustaka yang diperlukan oleh Windows, *device driver*, *registry*, dan program-program esensial yang dibutuhkan oleh Windows untuk berjalan dengan baik.

2. Direktori C:\Program Files
   Direktori ini berisikan semua program yang diinstal ke dalam sistem operasi. Semua program yang diinstal akan menulis *entry* ke dalam *registry* agar program tersebut dapat dijalankan dalam sistem Windows.

3. Direktori C:\My Documents
   Direktori ini berisikan semua dokumen yang dimiliki oleh pengguna sistem.
   Sistem operasi Windows dapat berjalan diatas beberapa macam sistem berkas. Setiap sistem berkas memiliki keunggulan dan kekurangan masing-masing. Semua keluarga Windows yang berbasis Windows NT dapat mendukung sistem berkas yang digunakan oleh keluarga Windows 9x, namun hal tersebut tidak berlaku sebaliknya.

Sistem Berkas yang terdapat dalam sistem operasi Windows adalah:

1. FAT 16: Sistem berkas ini digunakan dalam sistem operasi DOS dan Windows 3.1
2. FAT 32: Sistem ini digunakan oleh keluarga Windows 9x.
3. NTFS: Merupakan singkatan dari *New Technology File System*. Sistem berkas ini adalah sistem berkas berbasis *journaling* dan dapat digunakan hanya pada keluarga Windows NT. Keunggulan dari sistem berkas ini adalah fasilitas *recovery* yang memungkinkan dilakukannya penyelamatan data saat terjadi kerusakan pada sistem operasi.

5.12.2. Sistem Berkas pada UNIX (dan turunannya)

Ketika kita *login* ke UNIX, kita akan ditempatkan di direktori *root* kita. Direktori *root* kita dikenal sebagai direktori *home* kita dan dispesifikasi dengan *environment variable* yang dinamakan HOME. *Environment variable* ini menentukan karakteristik dari *shell* kita dan interaksi pengguna dengan *shell* tersebut. *Environment variable* yang umum adalah variabel PATH, yang mendefinisikan dimana *shell* akan mencari ketika perintah dari pengguna. Untuk melihat daftar *environment variable*, gunakan saja perintah printenv. Sedangkan untuk mengatur *environment variable*, gunakan setenv.

Ada beberapa direktori yang umum terdapat dalam instalasi UNIX:

1. Direktori "/" (*root*)
   Direktori ini terletak pada level teratas dari struktur direktori UNIX. Biasanya direktori *root* ini diberi tanda / atau *slash*. Direktori ini biasanya hanya terdiri dari direktori-direktori lainnya yang terletak pada level dibawah level direktori *root*. Berkas-berkas dapat disimpan pada direktori *root* tetapi usahakan tidak menyimpan berkas-berkas biasa sehingga direktori ini tetap terjaga keteraturannya. Perubahan penamaan direktori-direktori yang ada pada direktori *root* akan menyebabkan sebagian besar dari sistem menjadi tidak berguna. Karena sebagian besar dari direktori-direktori ini berisi fungsi-fungsi yang sifatnya kritikal yang dimana sistem operasi dan semua aplikasi memerlukan direktori-direktori ini dengan nama yang sudah diberikan pada awal instalasi. Tetapi kita bisa membuat direktori lain pada level ini. Direktori *home* juga bisa ditemukan pada level ini hasil pembuatan oleh administrator sistem.

2. Direktori "/bin"
   Direktori ini berisi program-program yang esensial agar sistem operasi dapat bekerja dengan benar. Dalam direktori ini dapat ditemukan perintah-perintah navigasi, program-program *shell*, perintah pencarian dan lain-lainnya. bin adalah singkatan dari

kata *binary*. Di UNIX, sebuah *binary* adalah berkas yang dapat dieksekusi. Sebagian besar dari perintah dalam UNIX merupakan *binary*, perintah-perintah tersebut merupakan program-program kecil yang dapat dieksekusi oleh pengguna. Ada beberapa perintah yang disebut perintah *built-in* dimana fungsi mereka dikendalikan oleh program *shell* sehingga mereka tidak beroperasi sebagai *binary* yang terpisah. Terkadang direktori *bin* terhubung ke direktori lain yang dinamakan /usr/bin. Direktori /usr/bin biasanya adalah lokasi sebenarnya dari *binary-binary* pengguna disimpan.Dalam hal ini, /bin adalah gerbang untuk mencapai /usr/bin.

3. Direktori "/dev"
   Direktori ini berisi berkas-berkas alat atau alat I/O. Sistem UNIX menganggap semua hal sebagai berkas. Hal-hal seperti monitor, CD-ROM, printer dan lain-lainnya dianggap hanya sebagai berkas saja oleh sistem operasi. Jika UNIX memerlukan perangkat-perangkat tersebut maka UNIX akan mencarinya ke direktori *dev*.

4. Direktori "/etc"
   Direktori yang dibaca et-see ini berisi beberapa konfigurasi berkas pengguna dan sistem, dan berkas yang ditunjuk sistem sebagai operasi normal seperti berkas kata sandi, pesan untuk hari ini, dan lain-lainnya.

5. Direktori "/lib"
   Direktori ini berisi pustaka-pustaka (*libraries*) yang dibagi (*shared*). Pustaka ini adalah rutin perangkat lunak (*software routines*) yang digunakan lebih dari satu bagian dari sistem operasi. Ketika kita menginstalasi perangkat lunak yang baru maka ada pustaka-pustaka baru yang ditambahkan ke direktori lib. Jika pada waktu berusaha menjalankan aplikasi terdapat pesan *error*, hal ini diakibatkan ada pustaka yang hilang dari direktori lib. Aplikasi-aplikasi di UNIX biasanya memeriksa lib ketika menginstalasi untuk memeriksa apakah pustaka-pustaka yang diperlukan oleh aplikasi sudah tersedia atau belum. Jika sudah tersedia, UNIX biasanya tidak menimpa pustaka tersebut.

6. Direktori "/sbin"
   Direktori ini berisi *binary-binary* juga seperti pada direktori *bin*.Tetapi,bedanya adalah *binary-binary* pada direktori ini berhubungan dengan fungsi-fungsi sistem administrasi pada sistem operasi UNIX. *Binary-binary* ini bukan yang biasa digunakan oleh pengguna tetapi digunakan agar komputer dapat beroperasi secara efisien.

7. Direktori "/usr"
   Direktori ini terdiri dari banyak direktori seperti pada direktori *root*. Direktori ini berisi berkas-berkas yang dapat diakses oleh para pengguna biasa. Struktur dari direktori ini mirip dengan struktur direktori "/". Beberapa direktori yang terdapat dalam direktori ini berhubungan dengan direktori yang ada di direktori /.

8. Direktori "/var"
   Direktori ini berisi data yang bermacam-macam (vary). Perubahan data dalam sistem yang aktif sangatlah cepat. Data-data seperti ini ada dalam waktu yang singkat. Karena sifatnya yang selalu berubah tidak memungkinkan disimpan dalam direktori seperti "/etc". Oleh karena itu, data-data seperti ini disimpan di direktori var.

5.12.3. Perbandingan antara Windows dan UNIX

Sistem berkas UNIX berbeda dengan sistem berkas Windows (DOS) karena sistem berkas UNIX lebih hebat dan mudah diatur daripada Windows (DOS). Penamaan dalam UNIX dan Windows berbeda.Karena sistem Windows ingin memudahkan pengguna maka sistem mereka mengubah nama menjadi nama yang lebih mudah bagi para

pengguna. Contohnya adalah nama *folder* dalam adalah perubahan dari *directory* yang masih digunakan oleh UNIX. Penggunaan *back slash* (\) digunakan untuk memisahkan direktori-direktori dalam Windows, tetapi hal ini tidak ada dalam UNIX. Sistem UNIX menggunakan *case sensitive*, yang artinya nama suatu berkas yang sama jika dibaca, tetapi penulisan namanya berbeda dalam hal ada satu file yang menggunakan huruf kapital dalam penamaan dan satu tidak akan berbeda dalam UNIX. Contohnya ada berkas bernama berkasdaku.txt dan BerkasDaku.txt, jika dibaca nama berkasnya sama tetapi dalam UNIX ini merupakan dua berkas yang jauh berbeda. Jika berkas-berkas ini berada di sistem Windows, mereka menunjuk ke berkas yang sama yang berarti Windows tidak *case sensitive*.

Hal lain yang membedakan sistem berkas UNIX dengan Windows adalah UNIX tidak menggunakan *drive letter* seperti C:, D: dalam Windows. Tetapi semua partisi dan drive ekstra di *mount* didalam sub-direktori di bawah direktori *root*. Jadi pengguna tidak harus bingung di *drive letter* mana suatu berkas berada sehingga seluruh sistem seperti satu sistem berkas yang berurutan dari direktori root menurun secara hierarki.

5.12.4. Macam-macam Sistem Berkas di UNIX

Secara garis besar, sistem berkas di sistem UNIX terbagi menjadi dua, yaitu sistem berkas dengan fasilitas *journaling* dan yang tidak memiliki fasilitas tersebut. Dibawah ini adalah beberapa sistem berkas yang digunakan dalam sistem UNIX pada umumnya:

1. EXT2
2. EXT3
3. JFS (*Journaling File System*)
4. ReiserFS
5. Dan Lain-lain.

5.13. Ringkasan

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

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

Berkas adalah kumpulan informasi berkait yang diberi nama dan direkam pada penyimpanan sekunder.

Atribut berkas terdiri dari:

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

- modifikasi terakhir yang dilakukan pada berkas, dan
- modifikasi terakhir yang dilakukan pada berkas, dan
- penggunaan terakhir berkas

Operasi Pada Berkas

1. Membuat sebuah berkas.
2. Menulis pada sebuah berkas.
3. Membaca sebuah berkas.
4. Menempatkan kembali sebuah berkas.
5. Menghapus sebuah berkas.
6. Memendekkan berkas.

Metode Akses

1. Akses Berurutan.
2. Akses Langsung.
3. Akses menggunakan Indeks.

Operasi Pada Direktori

1. Mencari berkas.
2. Membuat berkas.
3. Menghapus berkas.
4. Menampilkan isi direktori.
5. Mengganti nama berkas.
6. Melintasi sistem berkas.

Macam-macam Direktori

1. Direktori Satu Tingkat
2. Direktori Dua Tingkat.
3. Direktori Dengan Struktur "Tree".
4. Direktori Dengan Struktur "Acyclic-Graph".
5. Direktori Dengan Struktur Graph.

Metode Alokasi Berkas

1. Alokasi Secara Berdampingan (*Contiguous Allocation*).
2. Alokasi Secara Berangkai (*Linked Allocation*).
3. Alokasi Dengan Indeks (*Indexed Allocation*).

Manajemen Free Space

1. Menggunakan Bit Vektor.
2. Linked List.
3. Grouping.
4. Counting.

Implementasi Direktori

1. Linear List.
2. Hash Table.

Sistem Berkas pada Windows

Direktori yang secara otomatis dibuat dalam instalasi Windows adalah:

1. Direktori C:\WINDOWS
2. Direktori C:\Program Files
3. Direktori C:\My Documents

Sistem Berkas yang terdapat dalam sistem operasi Windows adalah:

1. FAT 16
   Sistem berkas ini digunakan dalam sistem operasi DOS dan Windows 3.1

2. FAT 32
   Sistem ini digunakan oleh keluarga Windows 9x
3. NTFS
   Merupakan singkatan dari New Technology File System. Sistem berkas ini adalah sistem berkas berbasis journaling dan dapat digunakan hanya pada keluarga Windows NT. Keunggulan dari sistem berkas ini adalah fasilitas recovery yang memungkinkan dilakukannya penyelamatan data saat terjadi kerusakan pada sistem operasi.

Sistem Berkas pada UNIX (dan turunannya)
Ada beberapa direktori yang umum terdapat dalam instalasi UNIX:

1. Direktori /root.
2. Direktori /bin.
3. Direktori /dev.
4. Direktori /etc.
5. Direktori /lib.
6. Direktori /sbin.
7. Direktori /usr.
8. Direktori /var.

Macam-macam Sistem Berkas di UNIX

1. EXT2.
2. EXT3.
3. JFS (Journaling File System).
4. ReiserFS.
5. Dan Lain-lain.

5.14. Soal-Soal Sistem Berkas

1. Sebutkan macam-macam atribut pada berkas!
2. Operasi apa sajakah yang dapat diterapkan pada sebuah berkas?
3. Sebutkan informasi yang terkait dengan pembukaan berkas!
4. Sebutkan dan jelaskan metode alokasi pada sistem berkas!
5. Sebutkan dan jelaskan operasi pada direktori?
6. Sebutkan dan Jelaskan tentang tipe akses pada berkas?
7. Sebutkan dan jelaskan bagaimana cara mengatur free space?
8. Bagaimanakah implementasi dari sebuah direktori dalam disk
9. Sebutkan keunggulan dari sistem berkas dalam UNIX dengan sistem berkas pada WINDOWS?
10. Bagaimanakah langkah-langkah dalam proses back-up?