

# **LAPORAN KERJA KELOMPOK**

## **SISTEM OPERASI**



*Oleh :*

**KELOMPOK 4**

**KHUMAEDI**

**JEFRI AL BUCHARI**

**HATFINA RUSTAMIN**

**SAYLANDRA AULIA**

**NUR AZISAH BASIR**

**RINA**

**SI23-A**

**PROGRAM STUDI SISTEM INFORMASI JURUSAN SAINS  
INSTITUT TEKNOLOGI BACHARUDDIN JUSUF HABIBIE**

**2024**

# Daftar Isi

<b>Daftar Isi.....</b>	<b>1</b>
<b>Kata Pengantar.....</b>	<b>1</b>
<b>1. Antarmuka Sistem File.....</b>	<b>2</b>
1.1 Konsep File.....	2
1.1.1 Atribut File.....	2
1.1.2 Operasi File.....	4
1.1.3 Tipe File.....	5
1.1.4 Struktur File.....	5
1.1.5 Struktur Internal File.....	5
1.2 Metode Akses.....	6
1.2.1 Akses Berurutan.....	6
1.2.2 Akses Langsung.....	7
1.2.3 Metode akses lainnya.....	7
1.3 Struktur Direktori.....	7
1.3.1 Single-level Directory.....	7
1.3.2 Two-level Directory.....	8
1.3.3 Tree Structured Directories.....	8
1.3.4 Acyclic Graph Directories.....	9
1.3.5 General Graph Directories.....	10
1.4 Proteksi.....	11
1.4.1 Tipe-Tipe akses.....	11
1.4.2 Kontrol Akses.....	12
1.4.3 Pendekatan lainnya.....	13
1.5 Memory-Mapped files.....	13
1.5.1 Mekanisme dasar.....	13
1.5.2 Shared Memory dalam Windows API.....	13
<b>2. Implementasi sistem file.....</b>	<b>14</b>
2.1 Struktur Sistem File.....	14
2.2 Operasi Sistem File.....	16
2.2.1 Overview.....	16
2.2.1 Usage.....	18
2.3 Implementasi Direktori.....	19
2.3.1 Linear List.....	20
2.3.2 Hash Table.....	21
2.4 Metode Alokasi.....	21
2.4.1 Contiguous Allocation.....	21

2.4.2 Linked Allocation.....	22
2.4.3 Indexed Allocation.....	24
2.4.4 Performance.....	25
2.5 Free-Space Management.....	25
2.5.1 Bit Vector.....	26
2.5.2 Linked List.....	26
2.5.3 Grouping.....	27
2.5.4 Counting.....	27
2.5.5 Space Maps.....	27
2.5.6 TRIMing Unused Blocks.....	27
2.6 Efficiency and Performance.....	28
2.6.1 Efficiency.....	28
2.6.2 Performance.....	28
2.7 Recovery.....	29
2.7.1 Consistency Checking.....	29
2.7.2 Log-Structured File Systems.....	30
2.7.3 Other Solutions.....	30
2.7.4 Backup and Restore.....	31
2.8 Example: The WAFL File System.....	32
<b>3. File-System Internal.....</b>	<b>33</b>
3.1 File Systems.....	33
3.2 File-Systems Mounting.....	35
3.3 Partitions And Mounting.....	37
3.4 File Sharing.....	39
3.4.1 Multiple Users.....	39
3.5 Virtual File Systems.....	40
3.6 Remote File Systems.....	42
3.6.1 The Client-Server Model.....	43
3.6.1 Distributed Information Systems.....	43
3.6.1 Failure Modes.....	43
3.7 Consistency Semantics.....	44
3.7.1 UNIX Semantics.....	44
3.7.2 Sessions Semantics.....	44
3.7.3 Immutable-Shared-Files Semantics.....	45
3.8 NFS.....	45
3.8.1 Overview.....	45
3.8.2 The Mount Protocol.....	46
3.8.3 The NFS Protocol.....	46
3.8.4 Path-Name Translation.....	47
3.8.5 Remote Operations.....	47

## **Kata Pengantar**

Puji dan Syukur kami panjatkan kepada tuhan yang maha esa, tuhan semesta alam. Karena atas limpahan berkah dan rahmatnya, kami dapat menyelesaikan kerja kelompok kami, serta menyelesaikan laporan ini, dengan tujuan untuk memenuhi mata kuliah sistem operasi di Institut Teknologi Bacharuddin Jusuf Habibie (ITH)

Kami menyadari bahwa laporan ini masih jauh dari kata sempurna dan masih banyak terdapat kekurangan, dan kesilapan, oleh karena itu, kami memohon kritik dan saran atas laporan kami, agar kami dapat memperbaiki laporan kami kedepan.

Akhir kata, kami ucapkan banyak terima kasih kepada pembaca yang telah menyempatkan waktu untuk membaca laporan kami.

Sabtu, 4 Mei 2024

Kelompok 4

## 1. Antarmuka Sistem File

Bagi Sebagian besar pengguna, sistem file adalah aspek yang paling terlihat dari sebuah sistem operasi itu sendiri. Sistem file menyediakan mekanisme untuk penyimpanan dan akses ke data serta program yang ada pada sistem operasi dan semua pengguna pada sistem operasi mesin yang digunakan. Sistem File sendiri terbagi atas dua komponen yang berbeda: yang pertama yaitu kumpulan file atau berkas yang menyimpan data-data terkait, dan struktur direktori, yang mengorganisir dan memberikan informasi tentang semua berkas yang ada di dalam sistem. Pada bagian ini kami akan mempertimbangkan berbagai aspek dari berkas dan juga struktur direktori utama. Kami juga akan membahas semantik atau proses bagaimana mekanisme berbagi file di antara beberapa proses, pengguna dan komputer. Selain itu kami juga akan membahas sisi keamanan perlindungan berkas saat kita memiliki beberapa pengguna, dan ingin membatasi akses pengguna terhadap berkas tersebut.

### 1.1 Konsep File

Komputer dapat menyimpan informasi ke beberapa media penyimpanan baik itu HDD, Disket magnet, dan Disket Optik. Agar Komputer dapat digunakan dengan nyaman sistem operasi akan menyediakan pandangan logis yang seragam tentang informasi yang disimpan. Sistem operasi mengabstraksi dari properti fisik perangkat penyimpanannya untuk mendefinisikan unit penyimpanan logis, yaitu berkas. Sebuah berkas adalah kumpulan informasi terkait yang diberi nama dan disimpan pada penyimpanan sekunder, karena bersifat non-volatil agar isi dari berkas atau file tetap sama meskipun mesin dimatikan.

#### 1.1.1 Atribut File

Sebuah berkas atau *file* diberikan nama untuk kenyamanan pengguna, dan dapat dipanggil sesuai dengan nama yang diberikan. Nama dari sebuah *file* biasanya ditulis dalam sebuah string. Pada beberapa sistem operasi, biasanya penamaan *file* biasanya bersifat *case-sensitive* artinya *file* dengan perbedaan huruf besar dan kecil biasanya dilihat oleh sistem operasi sebagai 2 file yang berbeda. Sebuah *File* setelah diberi nama akan bersifat independen, pemilik file dapat melakukan apa saja terhadap *file* tersebut bahkan mengirim *file* tersebut melalui *e-mail* dan *file* tersebut masih akan memiliki nama yang sama, kecuali terdapat sebuah metode sinkronisasi tertentu, dimana salinan dari suatu *file* akan menjadi independen dari salinan pertamanya aslinya, dan dapat diubah atau dimodifikasi tanpa mengubah salinan pertamanya. sebuah atribut file sangat bervariasi dimasing-masing sistem operasi, tapi pasti memiliki salah satu dari berikut:

- Nama

Nama sebagai penanda *file* adalah satu satunya informasi yang harus disimpan dalam bentuk yang dapat dibaca dan diketahui oleh manusia.

- Identifier

penanda unik dari suatu file, biasanya berupa angka. *Identifier* menandai *file* yang berada di sistem file. biasanya ditulis dalam bentuk yang tidak dapat dibaca oleh manusia

- Tipe

Informasi ini dibutuhkan oleh sistem yang mendukung berbagai jenis berkas

- Lokasi

Informasi ini merujuk ke suatu mesin dan lokasi dari file awal dimana file pertama diciptakan

- Ukuran

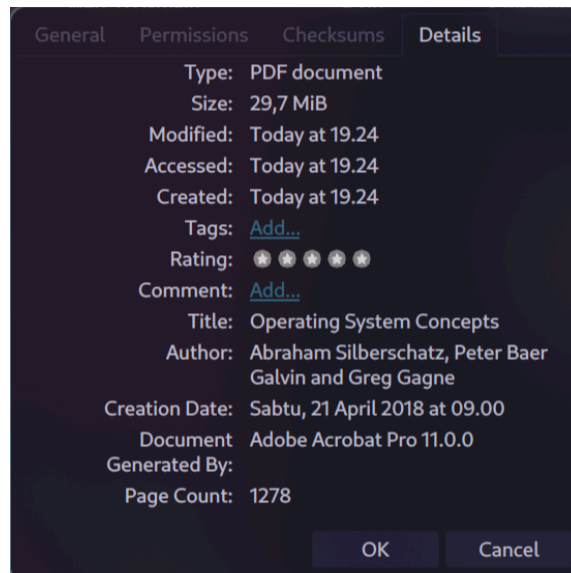
Besaran dari suatu *file* (biasanya dalam *bytes*, *words*, atau *blocks*) tidak menutup memungkinkan bagian ini juga akan menampilkan batas maksimal pada file.

- Proteksi

informasi yang menandakan siapa saja yang memiliki hak untuk membaca, menulis, mengeksekusi *file* dan seterusnya

- *Timestamps* atau penanda waktu

informasi ini biasanya mengandung, tanggal penulisan, kapan modifikasi terakhir, dan kapan terakhir *file* dibuka. Data ini juga berguna untuk proteksi, sekuritas dan melacak aktifitas dari *file*



Figur 1.1 Sebuah Informasi File pada Linux dengan Distribusi EndeavorOS

### 1.1.2 Operasi File

sebuah *file* adalah sebuah tipe data yang abstrak, untuk mendefinisikan suatu *file* dengan benar, kita perlu mempertimbangkan beberapa file yang dapat dioperasikan. Sistem operasi menyediakan *system call* untuk membuat, menulis, membaca, memindahkan, menghapus, dan memotong file

- Membuat *File*

dua hal yang wajib diperhatikan adalah pertama, ruang penyimpanan pada sistem harus tersedia untuk *file*. Kedua, entri dari *file* harus dibuat di dalam sebuah direktori penyimpanan

- Membuka *File*

Selain dari semua fungsi selain membuat dan menghapus, sebuah file harus memiliki fungsi *open()* terlebih dahulu, apabila sebuah file memiliki operasi maka akan mengembalikan sebuah argumen yang dapat digunakan di semua call yang lain. pada linux perintah yang dapat digunakan untuk membuka atau membaca *file* dapat menggunakan \$ cat

- Menulis pada *file*

untuk menulis pada *file* diperlukan *system call* yang spesifik ke membuka file, dan informasi akan ditulis pada file. sistem harus membuat sebuah pointer penulisan ke lokasi *file* dimana penulisan berikutnya akan diletakkan pada memori yang bersifat sequential. Pointer penulisan harus selalu terupdate apabila terjadi penulisan. untuk menulis pada file di linux dapat menggunakan perintah \$ nano

- Memindahkan *file*

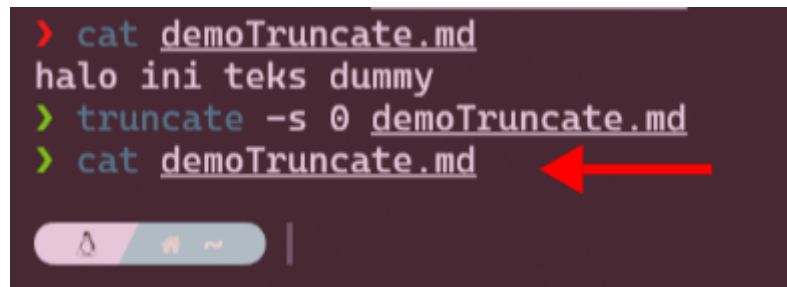
saat memindahkan *file* maka diperlukan sebuah pointer yang merujuk ke posisi *file* dan akan dipindahkan pada posisi tertentu yang telah ditentukan oleh pengguna. Memindahkn file tidak selalu membutuhkan operasi I/O. Pada Linux, memindahkan file dapat dilakukan dengan perintah \$ cp

- Menghapus *file*

untuk menghapus file perlu diketahui terlebih dahulu lokasi atau direktori sebuah *file*. Perlu diketahui juga bahwa untuk menghapus sebuah *file* pada beberapa sistem operasi diperlukan untuk benar benar menghapus file sampai ke *link* terakhirnya. Pada linux, file yang telah dihapus masih akan pindah ke ~/.Local/share/Trash maka perlu dilakukan pengecekan apakah *file* yang telah dihapus masih ada atau tidak. Pada linux dapat digunakan perintah \$ rm

- Memotong *file*

pengguna mungkin ingin menghapus sebuah konten dari sebuah *file* akan tetapi masih ingin menyimpan atributnya. Daripada harus menghapus pengguna dapat menggunakan atribut yang sama dengan operasi *truncate* atau memotong. pada sistem operasi linux dapat dilakukan operasi dengan perintah \$ truncate



```
> cat demoTruncate.md
halo ini teks dummy
> truncate -s 0 demoTruncate.md
> cat demoTruncate.md
```

A red arrow points to the second `cat demoTruncate.md` command, indicating the result of the truncation operation.

Figur 1.2 Penggunaan Truncate Pada Linux

### 1.1.3 Tipe File

Disaat mendesain sebuah sistem file, perlu dilakukan peritmbangan apakah sebuah sistem operasi perlu untuk mengetahui dan mendukung tipe -tipe file. Jika sebuah sistem operasi dapat mengenali sebuah tipe dari sebuah *file* maka *file* tersebut dapat dioperasikan tanpa adanya kendala. tipe dari sebuah *file* biasanya ditandai dengan ekstensi diakhir nama *file*

### 1.1.4 Struktur File

Tipe file juga dapat digunakan untuk mengindikasikan struktur internal dari sebuah *File*. Sumber dan objek *file* memiliki struktur yang menyesuaikan dengan ekspektasi program yang akan dibaca. Lebih lanjut, *file* diwajibkan memiliki struktur tertentu yang dapat diketahui oleh sistem operasi. agar saat dijalankan sistem operasi dapat mengetahui kemana dan seberapa besar alokasi memori yang dibutuhkan dan juga kebutuhan lainnya.



```
> debtap a.zip
Error: No such file or directory or invalid option
```

Figur 1.3 Contoh *file* yang tidak memiliki struktur yang dapat diketahui oleh fungsi yang dipanggil

### 1.1.5 Struktur Internal File

Pada dasarnya, saat berurusan dengan penyimpanan data di dalam sistem operasi, terdapat perbedaan pada ukuran data logis dan data fisik. Ini akan



menjadi rumit karena ukuran sistem operasi fisik menggunakan ukuran blok fisik tertentu untuk mengakses dan menyimpan data pada media penyimpanan. Penyimpanan data fisik bersifat konstan, dan tetap. Namun data logis itu dinamis, menyebabkan ketidakcocokan dengan ukuran blok penyimpanan data fisik. untuk menyelesaikan masalah ini, data logis dikemas dalam blok fisik saat disimpan dan, kemudian dipisahkan saat ingin kembali lagi, proess ini dikenal dengan nama *packing* dan *unpacking*

```
> stat besar.txt
File: besar.txt
Size: 1500          Blocks: 8          IO Block: 4096    regular file
Device: 0,39      Inode: 954840      Links: 1
Access: (0644/-rw-r--r--)  Uid: ( 1000/   amek)   Gid: ( 1000/   amek)
Access: 2024-05-03 22:21:53.290491326 +0800
Modify: 2024-05-03 22:21:53.300491282 +0800
Change: 2024-05-03 22:21:53.300491282 +0800
Birth: 2024-05-03 22:21:53.290491326 +0800
```

Figur 1.4 contoh *file* yang memiliki ukuran 1500 bytes tetapi memakan 4096 bytes pada penyimpanan data fisik

## 1.2 Metode Akses

*File* menyimpan informasi. Saat diggunakan, informasi ini akan diakses dan dibaca pada memori komputer. Informasi ini dapat diakses dengan berbagai cara seperti:

### 1.2.1 Akses Berurutan

Metode akses yang paling simpel adalah dengan menggunakan metode akses berurutan. Di linux dapat menggunakan \$ cat

```
> cat loren.txt

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Maecenas vehicula posuere nunc, a placerat erat tincidunt vitae. Quisque at ipsum porttitor, hendrerit augue nec, molestie diam. Nulla convallis eu arcu eget dictum. Curabitur tempor fermentum nibh, ut tristique enim ultricies vel. Quisque dignissim rhoncus venenatis. Pellentesque s
emper, purus eget malesuada vulputate, tellus velit ornare velit, nec vulputate elit ligula dictum urna. Praesent eget aliquam metus. Proin tellus ex, bibendum sit amet s
celerisque at, iaculis sed magna. Cras tellus nulla, mattis in faucibus in, pharetra sed neque. Quisque eget eleifend orci, quis viverra dui.

Vivamus vel nisl dictum, pellentesque nulla in, tempor purus. Vestibulum pharetra sapien id sem feugiat, in aliquet nulla varius. Suspendisse at nunc lobortis, vehicula d
ui sed, egestas odio. Vestibulum at nisi ullamcorper, congue purus eget, sollicitudin mauris. Proin fermentum ac purus in malesuada. Praesent rutrum nibh condimentum just
o ultrices, vitae cursus purus suscipit. Morbi sed laoreet urna, facilisis varius metus. Integer vitae nibh et odio bibendum cursus non finibus neque. Vestibulum ante ins
um primis in faucibus orci luctus et ultrices posuere cubilia curae; Duis eget luctus odio. Vestibulum ante ipsum primis in faucibus orci luctus et ultrices posuere cubil
ia curae; Suspendisse arcu eros, mattis vel est in, dignissim interdum tortor.

Mauris volutpat quam eu turpis dictum, vel luctus nisl vestibulum. Ut nec ipsum sed dui volutpat mattis. Duis ultricies dapibus tortor, vel gravida orci pulvinar ut. Quis
que pellentesque quam tortor, vel vehicula magna vehicula sit amet. Praesent at metus non nibh luctus dictum. Vestibulum facilisis dui quis maximus scelerisque. Fusce mat
tis dictum ante, in luctus leo elementum eget. Integer maximus, quam varius volutpat blandit, velit risus bibendum dolor, sit amet eleifend metus risus nec tellus.

Morbi quis neque sit amet felis ultricies pretium. Nulla volutpat nec libero at interdum. Ut ultrices ante et pellentesque feugiat. Proin nec malesuada elit. Integer nec
erat quis tellus sagittis accumsan ac quis massa. Nunc id erat a dolor eleifend blandit. Vestibulum non ultricies ex.

Vestibulum a luctus lorem. Morbi condimentum est nulla, ac convallis mauris pulvinar a. Nunc ut felis sit amet ante semper elementum sed nec diam. Quisque vulputate turpi
s purus, et feugiat dolor tempor ut. Phasellus sollicitudin ipsum vitae dignissim iaculis. Quisque volutpat mauris tortor, vel tempus quam semper et. Vestibulum ac ligula
imperdiet, sagittis massa id, sodales arcu. Nulla ut ultrices ipsum. Nam non ipsum magna. Pellentesque ornare ante nec sem tristique faucibus.
```

Figur 1.5 menggunakan perintah cat untuk akses berurutan

## 1.2.2 Akses Langsung

Metode ini memberikan pengguna akses untuk secara langsung mengakses byte atau baris yang diinginkan, pada linux dapat digunakan perintah dd dengan contoh argumen seperti berikut `dd if=lorem.txt bs=1 skip=1000 count=100`

```
> dd if=lorem.txt bs=1 skip=1000 count=100
rises, vitae cursus purus suscipit. Morbi sed laoreet urna, facilisis varius metus. Integer vitae ni100+0 records in
100+0 records out
100 bytes copied, 0,000473954 s, 211 kB/s
> grep rises, vitae cursus purus suscipit. Morbi sed laoreet urna, facilisis varius metus. Integer vitae ni" lorem.txt
Vivamus vel nisi dictum, pellentesque nulla in, tempor purus. Vestibulum pharetra sapien id sem feugiat, in aliquet nulla varius. Suspendisse at nunc lobortis, vehicula d
ui sed, egestas odio. Vestibulum at nisi ullamcorper, congue purus eget, sollicitudin mauris. Proin fermentum ac purus in malesuada. Praesent rutrum nibh condimentum just
o ultrices, vitae cursus purus suscipit. Morbi sed laoreet urna, facilisis varius metus. Integer vitae nibh et odio bibendum cursus non finibus neque. Vestibulum ante ips
um primis in faucibus orci luctus et ultrices posuere cubilia curae; Duis eget luctus odio. Vestibulum ante ipsum primis in faucibus orci luctus et ultrices posuere cubil
ia curae; Suspendisse arcu eros, mattis vel est in, dignissim interdum tortor.
```

Figur 1.6 Menggunakan perintah dd untuk langsung mengakses baris yang diinginkan

## 1.2.3 Metode akses lainnya

Metode akses lainnya juga dapat ditulis diatas metode akses secara langsung. metode ini biasanya melibatkan konstruksi index dari *file* terkait.

## 1.3 Struktur Direktori

Struktur direktori dapat dilihat sebagai suatu simbol tabel yang menerjemahkan nama-nama file menjadi blok kontrol file mereka. Dengan cara pandang seperti itu kita bisa melihat bahwa direktori itu bisa diorganisir dengan banyak cara. Bayangkan direktori sebagai kotak alat yang menyimpan daftar nama file dan referensi ke data mereka. Dan untuk menjaga semua file terorganisir, kita perlu cara untuk menambah, menghapus, mencari, dan menampilkan semua file dalam kotka alat ini. Berikutnya, kami akan jelaskan skema atau pendekatan yang paling sering digunakan untuk mendefinisikan sebuah logika struktur dari sebuah direktori

### 1.3.1 Single-level Directory

Sesuai Namanya struktur direktori hanya memiliki satu level, semua file berada di satu direkotri yang sama, memungkinkan untuk memahami dengan mudah isi dari direktori ini



Figur 1.7 Ilustrasi Single-level directory

```
> ls singleLevel  
data1.txt  
data2.txt  
data3.txt
```

Figur 1.8 Ilustrasi Single-Level Directory di Linux

### 1.3.2 Two-level Directory

Pada Two-level Directory, setidaknya terdapat dua tingkat atau level. Ini berarti bahwa ada setidaknya satu direktori yang berisi file dan direktori lain didalamnya

```
> ls twoLevel  
data1.txt  
scndFloor
```

Figur 1.9 Ilustrasi Two-Level Directory, dimana terdapat satu file dan satu direktori lainnya

### 1.3.3 Tree Structured Directories

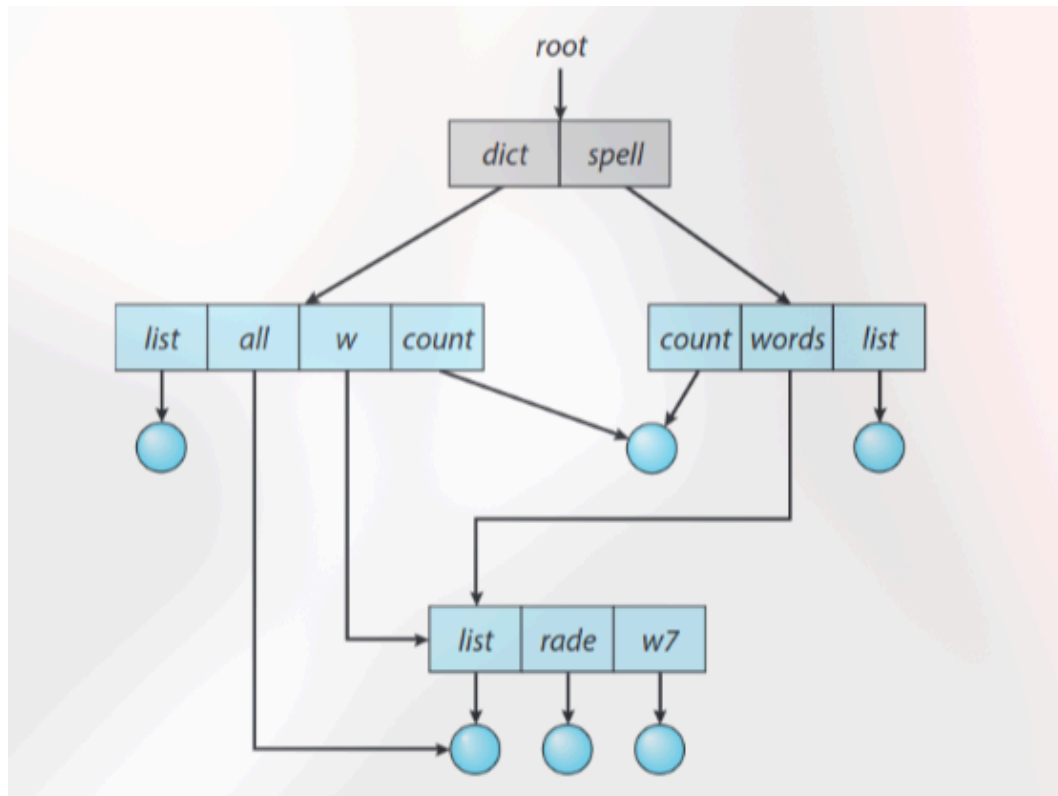
Tree Structured directories bersifat lebih kompleks, tree memiliki root direktori atau direktori akar atau awal atau main direktori, kemudian memiliki cabang didalamnya dimana masing masing direktori memiliki nama dan tugas masing-masing

```
> ls /
bin → usr/bin
boot
dev
etc
home
lib → usr/lib
lib64 → usr/lib
mnt
opt
proc
root
run
sbin → usr/bin
srv
sys
tmp
usr
var
```

Figur 1.10 Ilustrasi Tree Structured Directories

### 1.3.4 Acyclic Graph Directories

Struktur direktori graf ini memiliki hubungan yang kompleks dan tidak linear dimana satu direktori atau file dapat mengandung referensi atau *symlink* ke direktori atau file lainnya.



Figur 1.11 Ilustrasi Acyclic Graph Directory

```
lib → usr/lib
lib64 → usr/lib
```

Figur 1.12 Ilustrasi Acyclic Graph Directory di linux

### 1.3.5 General Graph Directories

Dalam struktur direktori ini, setiap direktori atau file dapat memiliki referensi atau symlink ke direktori atau file lain di dalam struktur ,yang dapat menyebabkan hubungan yang kompleks dan tidak linear di antara mereka. Tidak ada batasan tingkat kedalaman atau jenis antara file/direktori/entitas didalam struktur ini

```
> ls documents/spreadsheets
sheet1.xlsx
symlink_to_file1.txt → /home/amek/Downloads/documents/textfiles/file1.txt
```

Figur 1.13 ilustrasi sederhana direktori general graph directories

## 1.4 Proteksi

Saat informasi di letakkan di sistem komputer, hal yang harus dipastikan adalah keamanan data dari kerusakan fisik dan juga akses dari pihak yang tidak bertanggung jawab. File dapat mengalami kerusakan akibat permasalahan pada perangkat keras, kesalahan lonjakan listrik atau mati daya listrik. *File* juga dapat terhapus secara tidak sengaja. kesalahan program pada sistem file sistem operasi dapat menyebabkan kehilangan data. Oleh karena itu sangat diperlukan proteksi, untuk mesin yang menjalankan sistem operasi yang sudah modern, kebanyakan sistem operasi modern saat ini sudah memiliki keamanan yang cukup baik, dari pengguna yang ingin mengakses mesin perlu melewati proses autentikasi, enkripsi penyimpanan, dan juga keamanan jaringan sehingga orang yang ingin mengakses melalui jaringan juga akan kesulitan, untuk sistem yang memiliki banyak user diperlukan mekanisme yang lebih kompleks untuk memproteksi data yang ada.

### 1.4.1 Tipe-Tipe akses

kebutuhan akan perlindungan suatu *file* adalah hasil langsung dari kemampuan untuk mengakses *file*. File yang tidak mengizinkan akses ke *file* pengguna lain tidak memerlukan perlindungan. Oleh karena itu, dapat diberikan perlindungan lengkap dengan melarang akses. atau dapat juga diberikan kebebasan dalam mengakses tanpa adanya perlindungan. Namun, kedua pendekatan itu terlalu ekstrim untuk penggunaan umum. Yang dibutuhkan adalah akses yang terkendali. Mekanisme proteksi menyediakan akses yang terkendali dengan membatasi jenis akses berkas yang dapat dilakukan. Akses diizinkan atau ditolak tergantung pada beberapa faktor. salah satunya adalah jenis akses yang diminta. Ada beberapa operasi akses yang dapat dikontrol seperti:

- Read
- Write
- Execute
- Append
- Delete
- List
- Attribute change

```

> chmod 000 cobaProteksi.txt
> cat cobaProteksi.txt
cat: cobaProteksi.txt: Permission denied
> chmod +r cobaProteksi.txt
> cat cobaProteksi.txt
halo aku teks terproteksi

```

Figure 1.14 contoh pembatasan akses *file* pada linux

### 1.4.2 Kontrol Akses

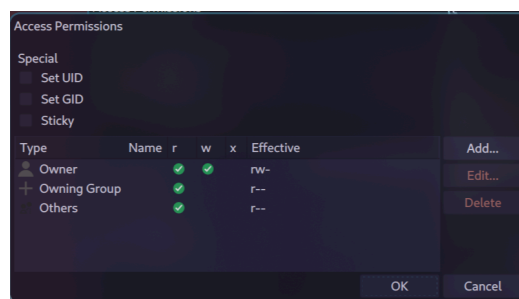
Pendekatan paling umum untuk mengatasi permasalahan keamanan adalah dengan memberikan akses kepada user atau pengguna tertentu. Beberapa pengguna mungkin tidak dapat mengakses beberapa file atau direktori. Skema paling umum adalah dengan mengimplementasikan access-control-list atau ACL dengan menspesifikasikan nama pengguna dan tipe file yang diberikan izin untuk diberikan akses

```

> setfacl -m u:amek:rw cobaProteksi.txt
> getfacl cobaProteksi.txt
# file: cobaProteksi.txt
# owner: amek
# group: amek
user::r--
user:amek:rw-
group::r--
mask::rw-
other::r--

```

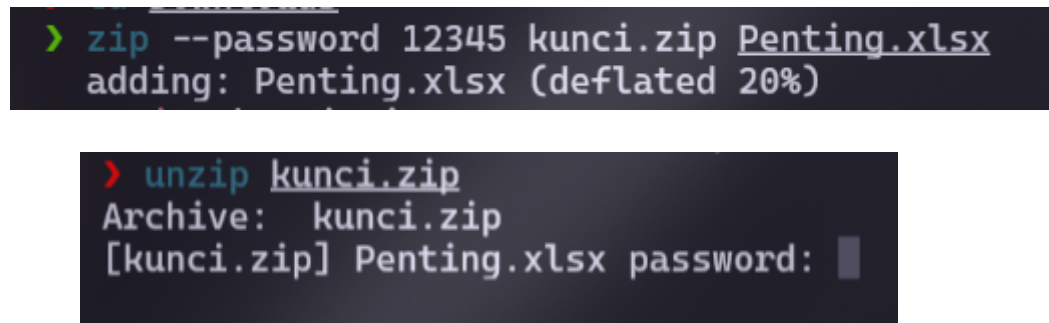
Figur 1.15 contoh pembatasan akses file dengan hanya diberikan akses read and write pada linux



Figur 1.16 Contoh pembatasan akses file di linux dengan GUI

### 1.4.3 Pendekatan lainnya

Metode lain untuk melindungi suatu *file* adalah dengan memberikan password ke file untuk proteksi lebih. jadi meskipun pengguna dapat melihat *file* tapi pengguna tidak dapat mengaksesnya karena terlindungi oleh password



```
> zip --password 12345 kunci.zip Penting.xlsx
adding: Penting.xlsx (deflated 20%)

> unzip kunci.zip
Archive:  kunci.zip
[kunci.zip] Penting.xlsx password: 
```

Figur 1.17 dan Figur 1.18 Contoh perlindungan file dengan memberikan password

## 1.5 Memory-Mapped files

Memory-Mapped Files adalah metode lain untuk mengakses *file* yang sangat umum digunakan juga. sama seperti membuka berkas secara berurutan pada disk dengan menggunakan panggilan sistem standar seperti `open()`, `read()`, dan `write()` tapi setiap akses memerlukan panggilan sistem dan akses ke disk. Prakteknya pada linux dapat dengan menggunakan perintah `cat`, karena dengan perintah ini linux secara otomatis memetakan file ke memori dan menampilkannya, meskipun tidak dilakukan secara eksplisit, tapi ini adalah contoh sederhana dari memory-mapped files di linux.

### 1.5.1 Mekanisme dasar

Memori mapping sebuah file dilakukan dengan memetakan sebuah blok disk ke suatu alamat tertentu di memori. Akses awal ke berkas dilakukan melalui paging permintaan biasa, yang menghasilkan page fault. Namun, sebuah bagian berukuran halaman dari berkas dibaca dari sistem berkas ke dalam sebuah alamat fisik. dengan menggunakan perintah `cat`, linux secara otomatis memetakan file ke memori dan menampilkannya.

### 1.5.2 Shared Memory dalam Windows API

Penggunaan Shared Memory dalam windows API melibatkan pembuatan sebuah wilayah memori bersama dengan menggunakan *memory-mapped files*. Langkah umum untuk menciptakan wilayah memori bersama ini melibatkan



pembuatan pemetaan berkas untuk berkas yang akan dipetakan, dan kemudian menetapkan pandangan dari berkas yang dipetakan ke dalam ruang alamat virtual dari sebuah proses. Proses kedua kemudian dapat membuka dan membuat pandangan dari berkas yang dipetakan ke dalam ruang alamat virtualnya. Berkas yang dipetakan mewakili objek shared memory yang akan memungkinkan komunikasi antara proses-proses tersebut.

```
> ipcmk -M 1024
Shared memory id: 65562
> ipcs -m
```

Shared Memory		Segments				
key	shmid	owner	perms	bytes	nattch	status
0x1c911f41	65562	amek	644	1024	0	

Figur 1.19 ilustrasi pembuatan shared memory di linux dengan perintah ipcmk

## 2. Implementasi sistem file

Pada topik kali ini kami akan fokus pada pembahasan implementasi sistem file.

### 2.1 Struktur Sistem File

Disk berfungsi sebagai media penyimpanan sekunder utama untuk sistem file karena dua karakteristik utama.

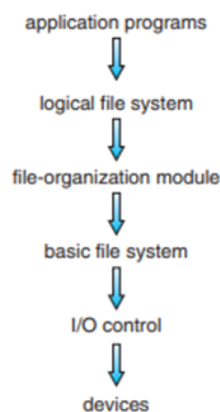
- 1) Dapat ditulis ulang di tempat. Fitur ini memungkinkan modifikasi blok dengan membacanya dari disk, mengubah isinya, dan menulis ulang ke dalam blok yang sama.
- 2) Akses langsung. Disk dapat secara langsung mengakses blok informasi apa pun yang dikandungnya, memfasilitasi akses berurutan dan acak ke file. Perpindahan antar file melibatkan pemindahan kepala baca-tulis dan menunggu media berputar.

Perangkat memori non-volatile (NVM), yang semakin banyak digunakan untuk penyimpanan file, berbeda dengan hard disk dalam hal kemampuan menulis ulang dan karakteristik kinerja. Struktur perangkat NVM dibahas lebih lanjut di Bab 11.

Untuk meningkatkan efisiensi I/O, transfer antara memori dan penyimpanan massal dilakukan dalam unit blok. Hard disk drive biasanya memiliki blok yang terdiri dari satu sektor atau lebih, dengan ukuran sektor

umumnya 512 atau 4.096 byte. Perangkat NVM biasanya memiliki blok 4.096 byte, menggunakan metode transfer yang serupa dengan drive disk.

Sistem berkas memfasilitasi penyimpanan, lokasi, dan pengambilan data yang efisien dengan mendefinisikan bagaimana sistem berkas terlihat oleh pengguna dan memetakan sistem berkas logis ke perangkat penyimpanan sekunder fisik. Sistem file biasanya terdiri dari berbagai tingkatan, yang sering kali diatur dalam desain berlapis.



Figur 2.1 Lapisan Sistem File

Tingkat kontrol I/O mencakup driver perangkat dan penanganan interupsi untuk memfasilitasi transfer informasi antara memori utama dan sistem disk. Driver perangkat bertindak sebagai penerjemah, mengubah perintah tingkat tinggi menjadi instruksi khusus perangkat keras tingkat rendah.

Sistem berkas dasar, yang juga dikenal sebagai "subsistem blok I/O" pada Linux, mengeluarkan perintah umum ke driver perangkat untuk membaca dan menulis blok pada perangkat penyimpanan. Sistem ini mengelola buffer memori, cache, dan penjadwalan permintaan I/O.

Modul pengaturan file menangani file dan blok logisnya, termasuk mengelola ruang kosong.

Sistem file logis mengelola informasi metadata, seperti struktur file, manajemen direktori, dan blok kontrol file (FCB) yang berisi atribut file.

Implementasi sistem berkas berlapis meminimalkan duplikasi kode dan memungkinkan penggunaan kembali kontrol I/O dan kode sistem berkas dasar.

Namun, hal ini dapat menimbulkan overhead sistem operasi tambahan, yang berpotensi memengaruhi kinerja.

Berbagai sistem file digunakan di berbagai sistem operasi yang berbeda, mendukung media yang dapat dipindahkan dan penyimpanan berbasis disk. Contohnya adalah ISO 9660 untuk CD-ROM, UFS untuk sistem berbasis UNIX, FAT, FAT32, dan NTFS untuk Windows, serta ext3 dan ext4 untuk Linux. Sistem berkas terdistribusi, seperti yang digunakan dalam pengaturan server-klien, juga umum digunakan.

Penelitian sistem berkas yang sedang berlangsung mengeksplorasi desain dan implementasi baru untuk memenuhi kebutuhan penyimpanan dan pengambilan yang terus berkembang. Contohnya adalah sistem berkas milik Google yang dioptimalkan untuk akses berkinerja tinggi dan sistem berkas FUSE, yang memungkinkan pengembangan dan eksekusi sistem berkas yang fleksibel sebagai kode tingkat pengguna di berbagai sistem operasi.

## **2.2 Operasi Sistem File**

sistem operasi mengimplementasikan `open()` dan `close()` untuk proses-proses yang meminta akses ke isi berkas. Pada bagian ini bagian ini, kita akan mempelajari struktur dan operasi yang digunakan untuk mengimplementasikan operasi sistem berkas.

### **2.2.1 Overview**

Sistem file dalam sistem operasi apa pun bergantung pada berbagai struktur baik dalam penyimpanan maupun memori. Struktur ini menyimpan informasi penting tentang pengaturan file dan direktori, serta memfasilitasi operasi yang efisien seperti mem-boot sistem dan mengelola akses file.

Struktur Pada Penyimpanan:

- 1) Boot control block(blok kontrol booting). Blok ini, biasanya ditemukan di awal volume, berisi informasi penting yang diperlukan untuk mem-boot sistem operasi. Misalnya, dalam UFS, blok ini disebut blok boot, sedangkan dalam NTFS, blok ini adalah sektor boot partisi.
- 2) Volume control block(blok kontrol volume). Setiap volume memiliki blok kontrol volume, yang menyimpan detail seperti jumlah total blok, ukuran blok, dan penunjuk ke blok bebas. Blok

ini dikenal sebagai superblok di UFS dan disimpan dalam tabel file master di NTFS.

- 3) Struktur direktori mengatur file di dalam sistem file dan menyertakan nama file serta nomor inode terkait di UFS, sedangkan di NTFS, struktur ini disimpan di tabel file master.
- 4) Blok Kontrol File Per File (FCB). Blok ini berisi informasi terperinci tentang setiap file, termasuk pengenalan unik. Dalam NTFS, informasi ini merupakan bagian dari tabel file master, yang mengadopsi struktur basis data relasional.

#### Struktur Dalam Memori:

- 1) Tabel Pemasangan. Memegang informasi tentang setiap volume yang dipasang di memori.
- 2) Cache Struktur Direktori. Menyimpan informasi direktori dari direktori yang baru saja diakses, sehingga memudahkan akses yang lebih cepat. Cache ini juga dapat berisi penunjuk ke tabel volume.
- 3) Tabel File Terbuka di Seluruh Sistem. Menyimpan salinan FCB dari setiap file yang terbuka bersama dengan informasi lain yang relevan.
- 4) Tabel File Terbuka Per-proses. Berisi penunjuk ke entri dalam tabel file terbuka di seluruh sistem untuk semua file yang dibuka oleh proses tertentu, bersama dengan informasi tambahan.
- 5) Buffer. Penyimpanan sementara untuk blok sistem berkas yang dibaca dari atau ditulis ke sistem berkas.

Untuk membuat berkas baru, sebuah proses berinteraksi dengan sistem berkas logis, yang memahami format struktur direktori. Sistem berkas logis mengalokasikan FCB baru, memperbarui direktori yang sesuai di memori dengan informasi berkas baru, dan kemudian menulisnya kembali ke sistem berkas.

file permissions
file dates (create, access, write)
file owner, group, ACL
file size
file data blocks or pointers to file data blocks

Figur 2.2 Tipikal blok control file

Sistem operasi yang berbeda menangani direktori secara berbeda. Beberapa, seperti UNIX, memperlakukan direktori sebagai file dengan field "type" yang menunjukkan sifatnya, sementara yang lain, seperti Windows, memperlakukannya sebagai entitas yang terpisah. Terlepas dari perbedaan ini, sistem berkas logis dapat memanggil modul organisasi berkas untuk mengelola I/O direktori dan berinteraksi dengan sistem penyimpanan yang mendasarinya.

### 2.2.1 Usage

Setelah sebuah file dibuat, file tersebut bisa digunakan untuk I/O (Input/Output). Namun, sebelum itu, file harus dibuka terlebih dahulu. Panggilan `open()` mengirimkan nama file ke sistem file logis. Panggilan sistem `open()` pertama-tama mencari file tersebut di tabel file terbuka secara sistemik untuk melihat apakah file sudah digunakan oleh proses lain. Jika sudah, entri tabel file terbuka per-proses dibuat yang menunjuk ke tabel file terbuka secara sistemik yang sudah ada. Algoritma ini dapat menghemat overhead yang signifikan. Jika file belum dibuka, struktur direktori dicari untuk nama file yang diberikan. Bagian-bagian dari struktur direktori biasanya disimpan dalam cache di memori untuk mempercepat operasi direktori. Setelah file ditemukan, FCB (File Control Block) disalin ke dalam tabel file terbuka secara sistemik di memori. Tabel ini tidak hanya menyimpan FCB tetapi juga melacak jumlah proses yang membuka file tersebut. Selanjutnya, entri dibuat di tabel file terbuka per-proses, dengan sebuah pointer ke entri di tabel file terbuka secara sistemik dan beberapa bidang lainnya. Bidang-bidang lain ini mungkin termasuk pointer ke lokasi saat ini di file (untuk operasi `read()` atau `write()` selanjutnya) dan mode akses di mana file dibuka.

Panggilan `open()` mengembalikan pointer ke entri yang sesuai di tabel sistem file per-proses. Semua operasi file kemudian dilakukan melalui pointer ini.

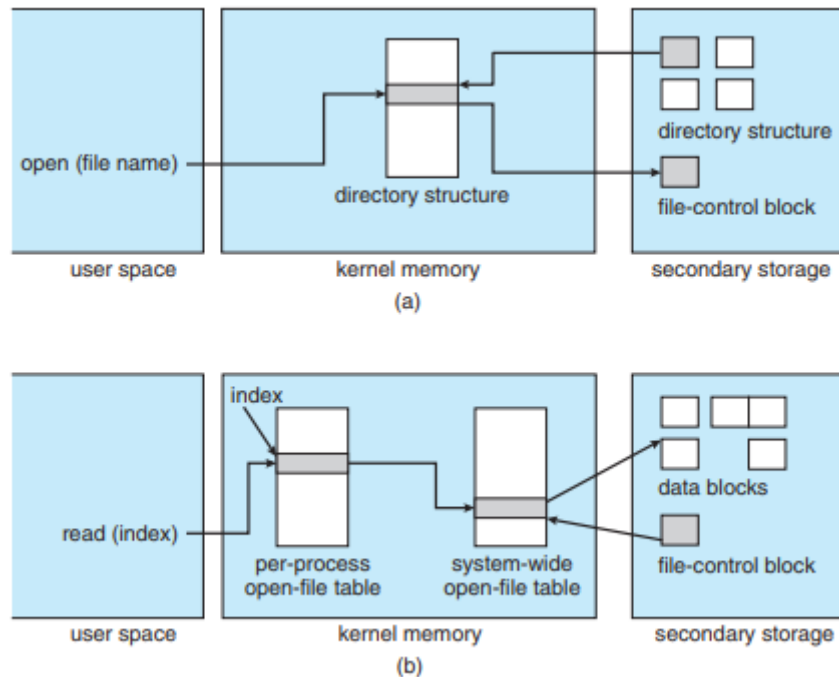
Nama file mungkin bukan bagian dari tabel file terbuka, karena sistem tidak membutuhkannya setelah FCB yang sesuai ditemukan di disk. Namun, nama tersebut bisa disimpan dalam cache untuk menghemat waktu pada pembukaan file yang sama selanjutnya. Nama yang diberikan pada entri bervariasi; sistem UNIX menyebutnya sebagai deskriptor file, sedangkan Windows menyebutnya sebagai handle file.

Ketika sebuah proses menutup file, entri tabel per-proses dihapus, dan hitungan bukaan entri sistemik dikurangi. Ketika semua pengguna yang membuka file menutupnya, metadata yang diperbarui disalin kembali ke struktur direktori berbasis disk, dan entri tabel file terbuka secara sistemik dihapus.

Aspek pengecachean dari struktur sistem file seharusnya tidak diabaikan. Kebanyakan sistem menyimpan semua informasi tentang file yang dibuka, kecuali blok data aktualnya, di memori. Sistem UNIX BSD adalah contoh tipikal dalam penggunaannya terhadap cache di mana saja I/O disk bisa dihemat. Tingkat keberhasilan cache rata-rata BSD UNIX sebesar 85 persen menunjukkan bahwa teknik-teknik ini sangat berharga untuk diimplementasikan. Struktur-operasi sistem file dari implementasi sistem file dirangkum dalam Gambar 2.3.

## **2.3 Implementasi Direktori**

Pemilihan algoritma alokasi direktori dan manajemen direktori sangat memengaruhi efisiensi, kinerja, dan kehandalan sistem file. Pada bagian ini, kita akan membahas kompromi yang terlibat dalam memilih salah satu dari algoritma-algoritma tersebut.



Figur 2.3 Struktur file sistem di memori (a)File open. (b)File read

### 2.3.1 Linear List

Metode paling sederhana untuk mengimplementasikan sebuah direktori adalah dengan menggunakan daftar linier dari nama file dengan pointer ke blok data. Metode ini mudah untuk diprogram namun memakan waktu dalam eksekusi. Untuk membuat file baru, kita harus mencari direktori untuk memastikan bahwa tidak ada file yang sudah ada dengan nama yang sama. Kemudian, kita tambahkan entri baru di akhir direktori. Untuk menghapus file, kita mencari direktori untuk file yang dinamai dan kemudian membebaskan ruang yang dialokasikan untuknya. Untuk mengulang entri direktori, kita bisa melakukan beberapa hal. Kita bisa menandai entri sebagai tidak digunakan (dengan memberikan nama khusus, seperti nama kosong, memberikan nomor inode yang tidak valid (seperti 0), atau dengan menyertakan bit digunakan-tidak digunakan dalam setiap entri), atau kita bisa mengaitkannya dengan daftar entri direktori gratis. Alternatif ketiga adalah dengan menyalin entri terakhir dalam direktori ke lokasi yang dibebaskan dan mengurangi panjang direktori. Sebuah daftar berantai juga bisa digunakan untuk mengurangi waktu yang diperlukan untuk menghapus sebuah file.

Kerugian nyata dari daftar linier entri direktori adalah bahwa menemukan sebuah file memerlukan pencarian linier. Informasi direktori digunakan secara sering, dan pengguna akan melihat jika akses ke informasi tersebut lambat. Bahkan, banyak sistem operasi mengimplementasikan cache

perangkat lunak untuk menyimpan informasi direktori yang paling sering digunakan. Cache hit menghindari kebutuhan untuk terus-menerus membaca ulang informasi dari penyimpanan sekunder. Sebuah daftar yang diurutkan memungkinkan pencarian biner dan mengurangi waktu pencarian rata-rata. Namun, kebutuhan akan menjaga daftar tetap diurutkan dapat mempersulit pembuatan dan penghapusan file, karena kita mungkin harus memindahkan sejumlah besar informasi direktori untuk menjaga direktori tetap diurutkan. Struktur data pohon yang lebih canggih, seperti pohon seimbang, mungkin bisa membantu di sini. Keuntungan dari daftar yang diurutkan adalah bahwa daftar direktori yang diurutkan dapat diproduksi tanpa langkah pengurutan terpisah.

### **2.3.2 Hash Table**

Tabel hash adalah struktur data alternatif untuk direktori file. Ini mengurangi waktu pencarian dengan menggunakan nilai yang dihitung dari nama file untuk menemukan entri dalam daftar linier. Penyisipan dan penghapusan mudah, meskipun perlu perhatian khusus untuk mengatasi tabrakan - ketika dua nama file menghasilkan nilai hash yang sama. Tabel hash memiliki kesulitan dengan ukuran yang tetap dan ketergantungan pada fungsi hash terhadap ukuran tersebut. Metode alternatif menggunakan tabel hash overflow terhubung, yang mengatasi tabrakan dengan membuat daftar terhubung. Meskipun pencarian mungkin sedikit lebih lambat, tetapi masih jauh lebih cepat daripada pencarian linier melalui seluruh direktori.

## **2.4 Metode Alokasi**

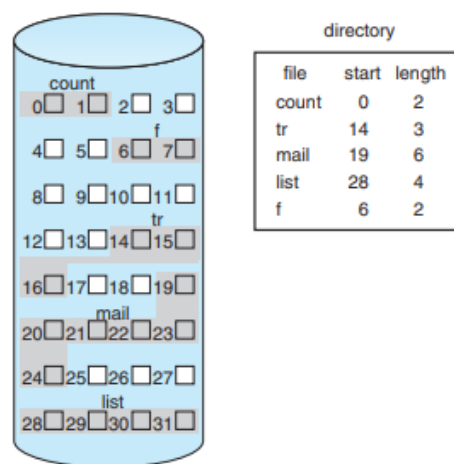
Akses langsung ke penyimpanan sekunder memberi fleksibilitas dalam implementasi file. Umumnya, banyak file disimpan pada perangkat yang sama. Masalah utamanya adalah bagaimana mengalokasikan ruang untuk file-file ini sehingga ruang penyimpanan digunakan secara efektif dan file dapat diakses dengan cepat. Ada tiga metode utama pengalokasian ruang penyimpanan sekunder yang umum digunakan: kontigu, berantai, dan berindeks. Setiap metode memiliki kelebihan dan kekurangan. Meskipun beberapa sistem mendukung ketiganya, lebih umum bagi suatu sistem untuk menggunakan satu metode untuk semua file dalam jenis sistem file tertentu.

### **2.4.1 Contiguous Allocation**

Alokasi berdekatan berarti file disimpan dalam urutan blok yang berkelanjutan pada perangkat penyimpanan. Metode ini memungkinkan akses



yang efisien karena blok-blok secara fisik berdekatan. Namun, mungkin sulit untuk menemukan ruang yang cukup berdekatan untuk file baru, yang mengarah ke fragmentasi eksternal di mana ruang kosong dipecah menjadi potongan-potongan kecil.



Figur gambar 2.4 Alokasi ruang disk yang berdekatan.

Untuk mengatasi masalah ini, beberapa sistem menggunakan skema alokasi bersebelahan yang dimodifikasi yang disebut alokasi berbasis luas. Dalam pendekatan ini, file pada awalnya dialokasikan sepotong ruang yang bersebelahan, tetapi potongan ruang tambahan yang bersebelahan, yang disebut ekstensi, dapat ditambahkan sesuai kebutuhan. Hal ini membantu mengurangi fragmentasi internal karena ruang dialokasikan dalam potongan yang lebih besar, tetapi mungkin masih mengalami fragmentasi eksternal karena ekstensi dengan berbagai ukuran dialokasikan dan didealokasikan.

Secara keseluruhan, meskipun alokasi bersebelahan sangat mudah dan efisien untuk mengakses file, sistem file modern cenderung menggunakan metode yang lebih canggih untuk mengelola ruang penyimpanan secara efektif.

### 2.4.2 Linked Allocation

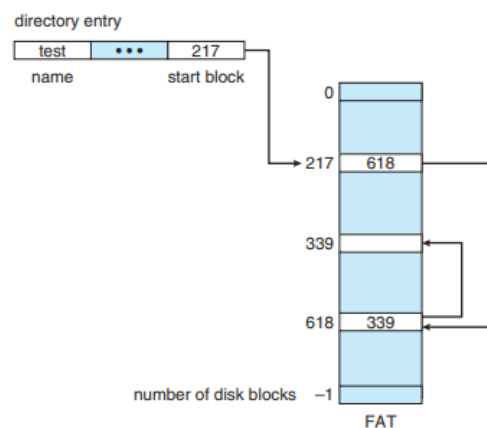
Linked allocation adalah metode penyimpanan file pada disk di mana setiap file direpresentasikan sebagai daftar terhubung dari blok penyimpanan. Berbeda dengan alokasi berurutan, di mana file menempati serangkaian blok yang kontinu, alokasi terhubung memungkinkan blok-blok file tersebar di mana saja di disk. Setiap blok mengandung sebuah pointer ke blok berikutnya dalam file. Untuk mengakses file, sistem mengikuti pointer ini dari blok ke blok.

Pendekatan ini menghilangkan fragmentasi eksternal karena blok dapat ditempatkan di mana saja di disk. File dapat tumbuh secara dinamis tanpa perlu mendeklarasikan ukurannya sebelumnya, dan ruang disk tidak perlu disusun ulang. Namun, alokasi terhubung paling efisien untuk file akses sekuensial. Mengakses blok-blok tertentu secara langsung memerlukan penelusuran seluruh daftar terhubung, yang bisa tidak efisien.

Alokasi terhubung juga memerlukan ruang untuk pointer di setiap blok, mengurangi jumlah ruang yang dapat digunakan untuk data. Untuk mengurangi hal ini, beberapa sistem menggunakan kelompok blok bersama dan mengalokasikannya sebagai satu unit.

Namun, alokasi terhubung menimbulkan kekhawatiran tentang keandalan. Jika sebuah pointer hilang atau rusak, dapat menyebabkan kesalahan dalam penghubungan file atau akses. Daftar terhubung ganda atau menyimpan metadata tambahan di setiap blok dapat mengatasi hal ini tetapi meningkatkan overhead.

Alternatif dari alokasi terhubung adalah metode tabel alokasi file (FAT), yang digunakan oleh MS-DOS. FAT menggunakan tabel yang diindeks berdasarkan nomor blok untuk mengelola alokasi ruang disk, memungkinkan penghubungan file dan pertumbuhan dinamis yang efisien. Namun, dapat menyebabkan jumlah pencarian kepala disk yang signifikan tanpa caching, memengaruhi kinerja.



Figur 2.5 Alokasi file

### 2.4.3 Indexed Allocation

Linked allocation memecahkan masalah fragmentasi eksternal dan deklarasi ukuran dari alokasi berurutan. Namun, tanpa File Allocation Table (FAT), alokasi terhubung tidak efisien untuk akses langsung karena pointer tersebar di seluruh disk. Alokasi yang diindeks mengatasi masalah ini dengan mengumpulkan semua pointer ke blok indeks.

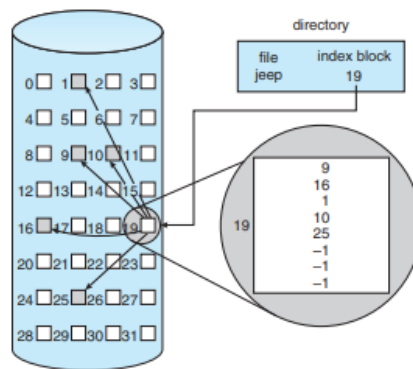


Figure 2.6 Alokasi yang diindeks di disk

Setiap file memiliki blok indeks sendiri, yang berisi alamat blok penyimpanan. Alokasi yang diindeks mendukung akses langsung tanpa fragmentasi eksternal, tetapi membuang ruang karena overhead pointer. Berbagai skema, seperti terhubung, multilevel, dan gabungan, digunakan untuk menangani masalah ukuran blok indeks. Meskipun demikian, alokasi yang diindeks memiliki masalah kinerja serupa dengan alokasi terhubung karena blok indeks dan data dapat tersebar di seluruh volume.

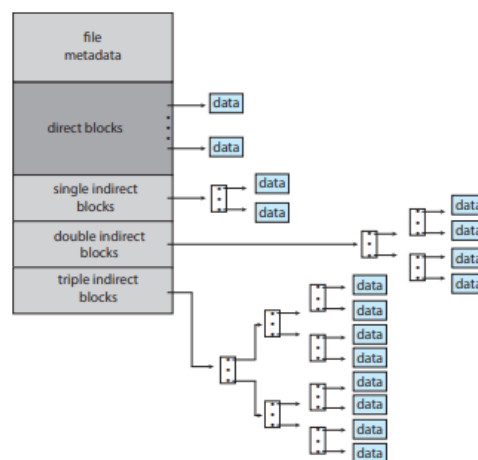


Figure 2.7 Inode pada UNIX

#### **2.4.4 Performance**

Metode alokasi yang dibahas bervariasi dalam efisiensi penyimpanan dan waktu akses blok data. Kedua faktor ini penting dalam pemilihan metode yang sesuai untuk diimplementasikan dalam sistem operasi.

Sebelum memilih metode alokasi, perlu ditentukan bagaimana sistem akan digunakan. Sistem dengan akses sekuensial yang dominan sebaiknya tidak menggunakan metode yang sama dengan sistem yang memiliki akses acak yang dominan.

Alokasi berurutan hanya membutuhkan satu akses untuk mendapatkan blok, karena alamat awal file dapat disimpan di dalam memori. Namun, alokasi terhubung mungkin memerlukan beberapa akses untuk mengambil blok-blok terkait.

Beberapa sistem mendukung file dengan akses langsung menggunakan alokasi berurutan dan file dengan akses sekuensial menggunakan alokasi terhubung. Jenis akses harus dideklarasikan saat pembuatan file.

Alokasi yang diindeks lebih kompleks. Jika blok indeks sudah di dalam memori, akses dapat dilakukan langsung, tetapi menyimpan blok indeks membutuhkan ruang memori yang signifikan. Kinerja alokasi yang diindeks tergantung pada struktur indeks, ukuran file, dan posisi blok yang diinginkan.

Beberapa sistem menggabungkan alokasi berurutan dengan alokasi yang diindeks, mengalokasikan berurutan untuk file kecil dan beralih ke alokasi yang diindeks untuk file yang berkembang besar.

Banyak optimasi digunakan untuk mengurangi jumlah instruksi dan jalur antara perangkat penyimpanan dan akses aplikasi ke data, terutama dengan berkembangnya perangkat NVM.

### **2.5 Free-Space Management**

Karena ruang penyimpanan terbatas, kita perlu menggunakan kembali ruang dari file-file yang telah dihapus untuk file-file baru, jika memungkinkan. Untuk melacak ruang disk yang kosong, sistem memelihara daftar ruang bebas. Daftar ruang bebas mencatat semua blok perangkat yang tidak dialokasikan untuk file atau direktori tertentu. Ketika membuat file, kita mencari daftar ruang bebas untuk jumlah ruang yang dibutuhkan dan mengalokasikan ruang tersebut untuk file baru. Ruang ini kemudian dihapus dari daftar ruang bebas. Ketika sebuah file dihapus, ruangnya ditambahkan ke daftar ruang bebas. Meskipun disebut daftar

ruang bebas, itu tidak selalu diimplementasikan sebagai daftar, seperti yang akan dibahas selanjutnya.

### 2.5.1 Bit Vector

Ruang bebas sering diwakili sebagai bitmap, di mana setiap blok memiliki nilai bit 1 jika kosong, dan 0 jika dialokasikan. Bitmap memungkinkan pencarian blok kosong dengan efisien, tetapi membutuhkan ruang memori yang signifikan terutama untuk perangkat besar. Instruksi manipulasi bit dapat digunakan untuk menemukan blok kosong pertama. Meskipun efisien, penggunaan bitmap menjadi semakin tidak praktis seiring dengan peningkatan ukuran disk, karena memori yang dibutuhkan juga meningkat secara signifikan.

### 2.5.2 Linked List

Sebuah pendekatan lain dalam manajemen ruang bebas adalah dengan mengaitkan bersama semua blok bebas, menyimpan pointer ke blok bebas pertama dalam lokasi khusus dalam sistem file dan menyimpannya di memori. Blok pertama ini berisi pointer ke blok bebas berikutnya, begitu seterusnya. Dalam contoh sebelumnya, jika blok 2, 3, 4, 5, 8, 9, 10, 11, 12, 13, 17, 18, 25, 26, dan 27 kosong, maka pointer ke blok 2 akan disimpan sebagai blok bebas pertama. Blok 2 akan berisi pointer ke blok 3, yang menunjuk ke blok 4, dan seterusnya.

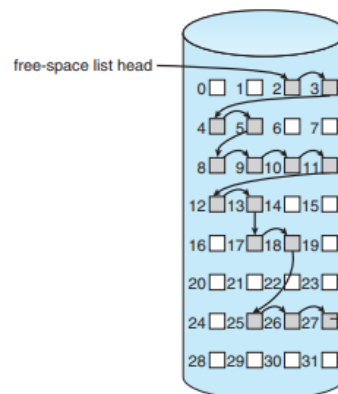


Figure 2.8 Daftar ruang kosong yang ditautkan pada disk.

Meskipun tidak efisien karena memerlukan waktu I/O yang substansial, mengunjungi daftar bebas tidak sering dilakukan. Biasanya, sistem operasi hanya membutuhkan blok bebas untuk dialokasikan ke file, jadi blok pertama dalam daftar bebas digunakan. Metode FAT menggabungkan akuntansi blok bebas ke dalam struktur data alokasi, sehingga tidak diperlukan metode terpisah.

### **2.5.3 Grouping**

Salah satu modifikasi dari pendekatan daftar bebas menyimpan alamat  $n$  blok bebas dalam blok bebas pertama.  $n-1$  blok pertama tersebut sebenarnya bebas. Blok terakhir berisi alamat  $n$  blok bebas lainnya, dan begitu seterusnya. Dengan demikian, alamat sejumlah besar blok bebas dapat ditemukan dengan cepat, tidak seperti saat menggunakan pendekatan daftar terhubung standar.

### **2.5.4 Counting**

Salah satu pendekatan lain memanfaatkan fakta bahwa seringkali beberapa blok kontigu dapat dialokasikan atau dibebaskan sekaligus. Daripada menyimpan daftar alamat  $n$  blok bebas, kita dapat menyimpan alamat blok bebas pertama dan jumlah blok bebas yang mengikuti blok tersebut. Setiap entri dalam daftar ruang bebas terdiri dari alamat perangkat dan jumlah blok. Metode ini mirip dengan penggunaan metode pemberian blok dan dapat disimpan dalam pohon seimbang untuk efisiensi.

### **2.5.5 Space Maps**

ZFS adalah sistem file yang dirancang untuk menangani volume besar file, direktori, dan sistem file. Untuk mengoptimalkan manajemen ruang bebas, ZFS menggunakan metaslab yang dibagi ke dalam bagian yang dapat dikelola dan memiliki peta ruang terkait. Informasi blok bebas disimpan dalam log secaraurut waktu dan dimuat ke dalam memori menggunakan struktur pohon seimbang. ZFS juga menggabungkan blok bebas yang bersebelahan untuk mengurangi ukuran peta. Perubahan terhadap daftar ruang bebas disertakan dalam operasi transaksi. Dengan pendekatan ini, ZFS mengurangi I/O yang dibutuhkan untuk manajemen ruang bebas, meningkatkan kinerja sistem file.

### **2.5.6 TRIMing Unused Blocks**

Pada media penyimpanan yang mendukung penimpaan seperti HDD, manajemen ruang bebas hanya membutuhkan daftar ruang bebas. Blok yang dibebaskan tetap menyimpan datanya sampai ditimpa saat alokasi berikutnya. Namun, pada perangkat yang tidak mengizinkan penimpaan seperti flash NVM, diperlukan mekanisme khusus seperti TRIM untuk memberi tahu perangkat bahwa sebuah halaman telah bebas dan dapat dihapus. Mekanisme serupa diperlukan untuk perangkat NVMe dengan perintah `unallocate`. Dengan TRIM, langkah-langkah pembersihan sampah dan penghapusan blok dapat dilakukan sebelum perangkat hampir penuh, menjaga konsistensi kinerja.

## **2.6 Efficiency and Performance**

Sekarang kita telah membahas berbagai pilihan alokasi blok dan manajemen direktori. manajemen, kita dapat mempertimbangkan lebih lanjut efeknya pada kinerja dan penggunaan penyimpanan yang efisien. Disk cenderung mewakili hambatan utama dalam sistem kinerja sistem, karena disk adalah komponen komputer utama yang paling lambat. Bahkan NVM lambat dibandingkan dengan CPU dan memori utama, sehingga kinerjanya harus dioptimalkan juga. Pada bagian ini, kami membahas berbagai teknik yang digunakan untuk meningkatkan efisiensi dan kinerja penyimpanan sekunder.

### **2.6.1 Efficiency**

Efisiensi penggunaan ruang perangkat penyimpanan sangat bergantung pada alokasi dan algoritma yang digunakan. Contoh yang menarik adalah inode UNIX yang disorot sebelumnya pada sebuah volume. Bahkan sebuah disk kosong pun memiliki persentase ruang yang hilang untuk inode. Dengan mengalokasikan inode dan menyebarkan pada volume, kami dapat meningkatkan kinerja sistem berkas. Peningkatan kinerja ini merupakan hasil dari alokasi UNIX dan algoritma ruang kosong, yang mencoba untuk menjaga file blok data di dekat blok inode file tersebut untuk mengurangi waktu pencarian.

### **2.6.2 Performance**

Untuk meningkatkan kinerja sistem berkas, beberapa cara dapat dilakukan, termasuk penggunaan cache on-board yang besar untuk menyimpan seluruh track atau blok sekaligus pengontrol pada perangkat. Pada HDD, setelah pencarian dilakukan, perjalanan dibaca ke dalam cache disk mulai dari sektor di bawah kepala disk, mengurangi latensi waktu. Pengontrol disk kemudian mentransfer sektor apa pun permintaan ke sistem operasi. Setelah blok berhasil keluar dari pengontrolan disk masuk ke memori utama, sistem operasi dapat menyimpan blok di sana.

Beberapa sistem operasi, seperti Solaris, Linux, dan Windows, menggunakan halaman cache untuk menyimpan file data. Cache halaman menggunakan teknik memori virtual untuk menyimpan file data sebagai halaman, bukan sebagai blok yang berorientasi pada sistem file. Mengakses file data menggunakan alamat virtual jauh lebih efisien daripada caching melalui blok disk fisik, karena mengakses antarmuka dengan memori virtual daripada sistem berkas. Dengan menggunakan cache halaman, sistem operasi dapat mempercepat akses ke file data dan meningkatkan kinerja sistem.

## 2.7 Recovery

Untuk memastikan keamanan dan integritas data, file dan direktori harus disimpan dalam memori utama dan volume penyimpanan. Kegagalan sistem dapat menyebabkan hilangnya data atau ketidakkonsistenan data, yang dapat disebabkan oleh kerusakan sistem yang mengganggu struktur penyimpanan file sistem data, seperti struktur direktori, blok bebas, dan penunjuk FCB bebas. Sistem berkas yang menerapkan perubahan pada struktur-struktur ini dapat mengalami masalah ketidakkonsistenan antara struktur data sistem file. Operasi yang umum, seperti membuat file, dapat melibatkan banyak perubahan struktural dalam file sistem pada disk, seperti modifikasi struktur direktori, alokasi FCB, alokasi blok data, dan pengurangan jumlah bebas untuk semua blok ini. Perubahan ini dapat diinterupsi oleh crash, dan ketidakkonsistenan antara struktur dapat terjadi.

Contoh masalah yang dapat timbul adalah ketika jumlah FCB bebas menunjukkan bahwa FCB telah dialokasikan, tetapi direktori tidak menunjuk ke FCB. Masalah ini diperparah dengan adanya caching yang dilakukan sistem operasi untuk mengoptimalkan kinerja I/O, karena beberapa perubahan mungkin terjadi langsung ke penyimpanan, sementara yang lain mungkin di-cache. Jika perubahan yang di-cache tidak mencapai perangkat penyimpanan sebelum terjadi kerusakan, lebih banyak kerusakan yang mungkin terjadi.

### 2.7.1 Consistency Checking

Untuk mendeteksi dan memperbaiki kerusakan sistem berkas, sistem berkas harus terlebih dahulu mendeteksi masalah dan kemudian memperbaikinya. Pemindaian semua metadata pada setiap file dapat mengkonfirmasi atau menyangkal konsistensi sistem. Namun, pemindaian ini dapat memakan waktu beberapa menit atau jam dan harus dilakukan setiap kali sistem melakukan booting. Alternatifnya, sistem berkas dapat merekam kondisinya di dalam metadata sistem berkas. Pada awal setiap perubahan metadata, sebuah status bit di-set untuk menunjukkan bahwa metadata sedang berubah. Jika semua pembaruan pada metadata berhasil diselesaikan, sistem file dapat menghapus bit tersebut. Namun, jika status bit tetap disetel, pemeriksaan konsistensi dijalankan.

Periksa konsistensi, seperti fsck pada UNIX, membandingkan data pada struktur direktori dan metadata lainnya dengan status pada penyimpanan dan mencoba untuk memperbaiki ketidakkonsistenan yang ditemukan. Alokasi dan algoritma manajemen ruang bebas menentukan jenis masalah apa yang dapat ditemukan oleh pemeriksa dan seberapa sukses ia akan diperbaiki. Contoh masalah yang dapat ditemukan meliputi hilangnya entri direktori pada sistem alokasi terindeks, yang dapat menyebabkan blok data tidak saling mengenal satu sama lain. Untuk mengatasi masalah ini, beberapa sistem berkas UNIX menyimpan entri direktori untuk pembacaan, tetapi setiap penulisan yang



mengakibatkan alokasi ruang atau perubahan metadata lainnya, dilakukan secara serempak, sebelum blok data yang bersangkutan ditulis.

### **2.7.2 Log-Structured File Systems**

Penggunaan algoritma pemulihan berbasis log untuk memperbarui metadata file sistem memungkinkan file sistem tetap konsisten meskipun terjadi kerusakan. Dalam pendekatan ini, semua perubahan metadata ditulis secara berurutan ke dalam log, dan setiap rangkaian operasi untuk melakukan tugas tertentu adalah sebuah transaksi. Ketika perubahan dilakukan, sebuah penunjuk diperbarui untuk menunjukkan tindakan mana yang telah selesai dan mana yang masih belum selesai. Ketika seluruh transaksi yang dilakukan selesai, dan entri dibuat dalam log yang menunjukkan itu. File log sebenarnya adalah sebuah buffer melingkar yang menulis ke akhir dari ruangnya dan kemudian melanjutkan di awal, menimpa nilai yang lebih lama yang lebih lama sambil berjalan.

Jika sistem mengalami kerusakan, file log akan berisi nol atau lebih transaksi. Setiap transaksi yang ada di dalamnya tidak diselesaikan ke file sistem, meskipun transaksi tersebut dilakukan oleh sistem operasi, sehingga sekarang harus diselesaikan. Transaksi dapat dieksekusi dari penunjuk sampai pekerjaan selesai sehingga struktur sistem berkas tetap konsisten. Satu-satunya masalah yang terjadi ketika sebuah transaksi dibatalkan - yaitu, tidak dilakukan sebelum sistem crash. Setiap perubahan dari transaksi tersebut yang diterapkan pada berkas harus dibatalkan, sekali lagi menjaga konsistensi sistem file. Pemulihan ini adalah semua yang diperlukan setelah crash, menghilangkan masalah dengan pemeriksaan konsistensi.

Manfaat sampingan dari penggunaan pencatatan pada pembaruan metadata disk adalah pembaruan tersebut berjalan lebih cepat daripada ketika diterapkan langsung ke struktur data disk. Alasannya ditemukan dalam keuntungan kinerja dari I/O berurutan dibandingkan I/O acak. Metadata acak sinkron yang mahal penulisan acak sinkron yang mahal diubah menjadi penulisan berurutan sinkron yang jauh lebih murah ke area pencatatan sistem file terstruktur log. Perubahan itu, pada gilirannya, diputar ulang secara asinkron melalui penulisan acak ke struktur yang sesuai. Secara keseluruhan keseluruhan adalah peningkatan yang signifikan dalam kinerja operasi yang berorientasi pada metadata, seperti pembuatan dan penghapusan file,

### **2.7.3 Other Solutions**

Alternatif lain untuk pengecekan konsistensi digunakan oleh Network Appli-sistem berkas WAFL milik Network Appli dan sistem berkas Solaris ZFS. Sistem-sistem ini tidak pernah menimpa blok dengan data baru. Sebaliknya,

sebuah transaksi menulis semua data dan meta perubahan data ke blok baru. Ketika transaksi selesai, struktur metadata yang menunjuk ke versi lama dari blok-blok ini diperbarui untuk menunjuk ke blok yang baru. Sistem berkas kemudian dapat menghapus penunjuk lama dan lama dan blok lama dan membuatnya tersedia untuk digunakan kembali. Jika penunjuk dan blok lama disimpan, snapshot dibuat; snapshot adalah tampilan file sistem pada waktu tertentu (sebelum ada update setelah waktu tersebut). Solusi ini seharusnya tidak memerlukan pemeriksaan konsistensi jika penunjukan pembaruan dilakukan secara atomis. Namun, WAFL memiliki pemeriksa konsistensi, sehingga beberapa kegagalan masih dapat menyebabkan kerusakan metadata.

ZFS mengambil pendekatan yang lebih inovatif untuk konsistensi disk. Seperti WAFL, ZFS tidak pernah menimpa blok. Namun, ZFS melangkah lebih jauh dan menyediakan checksum dari semua metadata dan blok data. Solusi ini (bila dikombinasikan dengan RAID) memastikan bahwa data selalu benar. Oleh karena itu, ZFS tidak memiliki konsistensi pemeriksa.

#### **2.7.4 Backup and Restore**

Perangkat penyimpanan terkadang mengalami kegagalan, dan untuk memastikan bahwa data yang hilang dalam kegagalan tersebut tidak hilang selamanya, sistem program dapat digunakan untuk mencadangkan data dari satu perangkat penyimpanan ke perangkat penyimpanan lainnya, seperti tape atau perangkat penyimpanan sekunder lainnya. Pemulihan dari kehilangan satu file, atau seluruh perangkat, mungkin merupakan masalah memulihkan data dari cadangan.

Untuk meminimalkan penyalinan yang diperlukan, kita dapat menggunakan informasi dari setiap entri file direktori. Misalnya, jika program pencadangan mengetahui kapan pencadangan terakhir suatu file dilakukan, dan tanggal penulisan terakhir file dalam direktori menunjukkan bahwa file tersebut tidak berubah sejak tanggal tersebut, maka file tersebut tidak perlu diijinkan lagi. Siklus pencadangan umum yang dilakukan adalah sebagai berikut:

- Hari 1. Salin ke media cadangan semua file dari file sistem. Ini disebut pencadangan penuh.
- Hari 2. Salin ke media lain semua file yang diubah sejak hari pertama. Ini adalah pencadangan tambahan.
- Hari 3. Salin ke media lain semua file yang diubah sejak hari ke-2.
- ...
- Hari N. Salin ke media lain semua file yang berubah sejak hari N-1. Lalu pergi kembali ke hari ke-1.

Dengan menggunakan metode ini, kita dapat memulihkan seluruh file sistem dengan memulai pemulihan dengan pencadangan penuh dan melanjutkannya melalui setiap pencadangan tambahan. Tentu saja, semakin besar nilai N, semakin besar jumlah media yang harus dibaca untuk pemulihan lengkap. Keuntungan tambahan dari siklus pencadangan ini adalah kami dapat memulihkan

file apa pun yang terhapus secara tidak sengaja selama siklus dengan mengambil file yang terhapus dari pencadangan hari sebelumnya.

Panjang siklus adalah kompromi antara jumlah cadangan yang dibutuhkan dan jumlah hari yang diambil oleh pemulihan. Untuk mengurangi jumlah kaset yang harus dibaca untuk melakukan pemulihan, salah satu pilihannya adalah melakukan pencadangan penuh dan kemudian setiap hari mencadangkan semua file yang telah berubah sejak pencadangan penuh. Dengan cara ini, pemulihan dapat dilakukan melalui pencadangan tambahan terbaru dan pencadangan penuh, tanpa memerlukan pencadangan tambahan lainnya. Pengorbanannya adalah bahwa lebih banyak file akan dimodifikasi setiap hari, sehingga setiap pencadangan tambahan berturut-turut melibatkan lebih banyak file dan lebih banyak media cadangan.

Seorang pengguna mungkin menyadari bahwa file tertentu hilang atau rusak lama setelah kerusakan itu terjadi. Karena alasan ini, kami biasanya merencanakan untuk membuat cadangan penuh dari waktu ke waktu yang akan disimpan "selamanya". Disarankan kita menyimpan cadangan ini cadangan permanen ini jauh dari cadangan reguler untuk melindungi dari bahaya, seperti kebakaran yang menghancurkan komputer dan semua cadangan juga. Dalam acara TV "Mr. Robot," peretas tidak hanya menyerang sumber data utama bank, tetapi juga situs cadangan mereka. Memiliki beberapa situs cadangan mungkin bukan ide yang buruk jika data Anda penting.

## **2.8 Example: The WAFL File System**

Tata letak file tulis di mana saja (WAFL) dari NetApp, Inc. adalah contoh dari penggilingan file sistem yang dioptimalkan untuk penulisan acak. WAFL digunakan secara eksklusif pada jaringan file server yang diproduksi oleh NetApp dan dimaksudkan untuk digunakan sebagai file sistem terdistribusi. Sistem ini dapat menyediakan file ke klien melalui protokol NFS, CIFS, iSCSI, ftp, dan http, meskipun dirancang hanya untuk NFS dan CIFS.

WAFL digunakan pada server berkas yang menyertakan cache NVRAM untuk penulisan. Perancang WAFL mengambil keuntungan dengan menjalankannya pada arsitektur tertentu untuk mengoptimalkan sistem berkas untuk I/O acak, dengan penyimpanan cache yang stabil di depan. Kemudahan penggunaan adalah salah satu prinsip panduan WAFL. Para penciptanya juga mendesainnya untuk menyertakan fungsionalitas snapshot baru yang membuat beberapa salinan hanya-baca dari sistem berkas pada waktu yang berbeda.

Sistem berkas ini mirip dengan Sistem Berkas Cepat Berkeley, dengan banyak modifikasi. Sistem ini berbasis blok dan menggunakan inode untuk mendeskripsikan berkas. Setiap inode berisi 16 pointer ke blok (atau blok tidak langsung) milik file yang dideskripsikan oleh inode. Setiap sistem berkas memiliki inode root. Semua metadata berada di dalam berkas. Semua inode berada dalam satu berkas, peta blok bebas di berkas lain, dan peta inode bebas di berkas ketiga.

Dengan demikian, file sistem WAFL adalah pohon blok dengan root inode sebagai dasarnya. Untuk mengambil snapshot, WAFL membuat salinan inode

root. Setiap file atau pembaruan metadata setelah itu masuk ke blok baru daripada menimpa mereka menimpa blok yang sudah ada. Root inode yang baru menunjuk ke metadata dan data yang berubah sebagai hasil dari penulisan ini. Sementara itu, snapshot (inode root lama) masih mengarah ke blok lama, yang belum diperbarui. Oleh karena itu, snapshot menyediakan akses ke sistem berkas seperti pada saat snapshot dibuat - dan hanya membutuhkan sedikit ruang penyimpanan untuk melakukannya. Intinya, ruang ekstra yang ditempati oleh snapshot hanya terdiri dari blok-blok yang telah dimodifikasi sejak snapshot diambil.

Perubahan sistem file WAFL yang signifikan adalah penggunaan blok bebas yang memiliki lebih dari satu bit per blok. Bitmap ini menunjukkan setiap snapshot yang menggunakan blok tersebut. Ketika semua snapshot yang menggunakan blok dihapus, bitmap menjadi nol, dan blok tersebut dapat digunakan kembali. Penulisan sangat cepat karena dapat terjadi pada blok bebas yang terdekat dengan lokasi kepala saat ini. Fasilitas snapshot WAFL juga sangat efisien karena tidak memerlukan salinan salin-tulis setiap blok data sebelum dimodifikasi.

Versi WAFL yang lebih baru memungkinkan snapshot baca-tulis, yang dikenal sebagai klon. Klon juga efisien, menggunakan teknik yang sama dengan snapshot. Dalam hal ini, snapshot hanya-baca menangkap status berkas sistem, dan klon Merujuk kembali ke snapshot hanya-baca tersebut. Setiap penulisan pada klon disimpan dalam blok baru, dan penunjuk klon diperbarui untuk Merujuk ke blok baru. Cuplikan asli tidak dimodifikasi, masih memberikan tampilan ke dalam file sistem seperti sebelum klon diperbarui. Klon juga dapat dipromosikan untuk menggantikan sistem berkas asli; Hal ini melibatkan pembuangan semua pointer lama dan semua blok lama yang terkait. Klon berguna untuk pengujian dan peningkatan, karena versi asli tidak terhenti dan klon akan dihapus ketika pengujian selesai atau jika pemutakhiran gagal

### **3. File-System Internal**

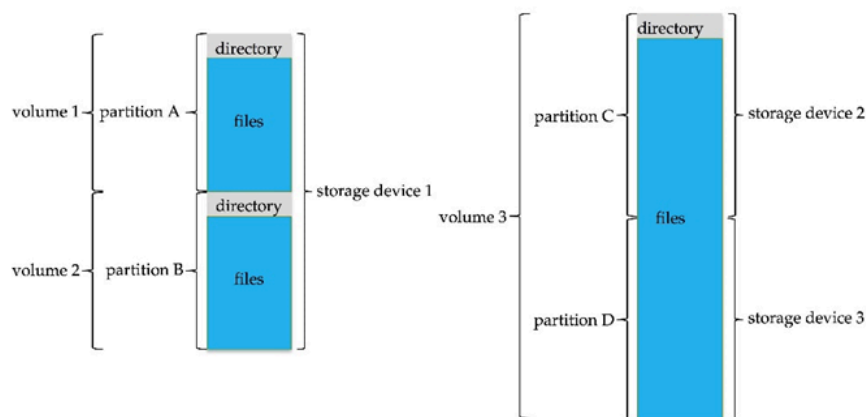
Materi ini berkaitan dengan struktur internal dan operasi sistem file. Kita akan membahas secara detail cara-cara untuk menyusun struktur penggunaan file, mengalokasikan ruang penyimpanan, memulihkan ruang yang kosong, melacak lokasi data, dan menghubungkan bagian lain dari sistem operasi ke penyimpanan sekunder.

#### **3.1 File Systems**

Tentu saja, pasti semua komputer menyimpan banyak file di dalamnya. Semua file disimpan di perangkat penyimpanan akses acak, termasuk hard disk drive, disk optik.

Komputer serba guna memiliki beberapa perangkat penyimpanan yang dapat dibagi menjadi partisi, dan setiap partisi dapat menampung volume dengan sistem file yang berbeda. Volume juga dapat mencakup beberapa partisi tergantung pada manajer volume yang digunakan.

Berikut pembahasan mengenai sistem berkas untuk keperluan umum. Perlu dicatat, bahwa ada banyak sistem berkas dengan tujuan khusus.



- tmpfs-sebuah sistem berkas "sementara" yang dibuat di memori utama yang mudah menguap dan isinya akan terhapus jika sistem melakukan boot ulang atau crash
- objfs-sebuah sistem berkas "virtual" (pada dasarnya adalah sebuah interface ke kernel yang terlihat seperti sistem berkas) yang memberikan akses kepada debugger ke simbol-simbol kernel
- ctfs-sebuah sistem berkas virtual yang menyimpan informasi "kontrak" untuk manusia usia proses yang dimulai saat sistem melakukan booting dan harus terus berjalan selama operasi.
- lofs-sebuah sistem berkas "loop back" yang memungkinkan satu sistem berkas diakses untuk menggantikan sistem berkas yang lain.
- procfs-sistem berkas virtual yang menyajikan informasi tentang semua proses sebagai sistem berkas.
- ufs, zfs-sistem berkas untuk tujuan umum

Sistem berkas komputer, dengan demikian, bisa sangat luas. Bahkan di dalam sebuah file sistem file, akan berguna untuk memisahkan file ke dalam kelompok-kelompok dan mengelola serta bertindak terhadap kelompok-kelompok tersebut.

### 3.2 File-Systems Mounting

Sama seperti file yang harus dibuka sebelum dapat digunakan, sistem file harus dipasang sebelum dapat tersedia untuk proses pada sistem. Secara lebih spesifik, struktur direktori dapat dibuat dari beberapa volume yang berisi sistem berkas, yang harus di-mount agar dapat tersedia di dalam ruang nama sistem berkas.

Prosedur mount sangatlah mudah. Sistem operasi diberi nama perangkat dan titik mount - lokasi di dalam struktur berkas di mana sistem berkas akan disambungkan. Beberapa sistem operasi mengharuskan jenis sistem file disediakan, sementara yang lain memeriksa struktur perangkat dan tentukan jenis sistem file. Biasanya, titik mount adalah direktori kosong. Sebagai contoh, pada sistem UNIX, sistem berkas yang berisi direktori home pengguna mungkin di-mount sebagai */home*; kemudian, untuk mengakses struktur direktori dalam sistem berkas tersebut, kita dapat mengawali nama direktori dengan */home*, seperti pada */home/jane*. Memasang sistem file tersebut di bawah */users* akan menghasilkan nama path */users/jane*, yang dapat kita gunakan untuk mencapai direktori yang sama.

/	ufs
/devices	devfs
/dev	dev
/system/contract	ctfs
/proc	proc
/etc/mnttab	mntfs
/etc/svc/volatile	tmpfs
/system/object	objfs
/lib/libc.so.1	lofs
/dev/fd	fd
/var	ufs
/tmp	tmpfs
/var/run	tmpfs
/opt	ufs
/zpbge	zfs
/zpbge/backup	zfs
/export/home	zfs
/var/mail	zfs
/var/spool/mqueue	zfs
/zpbg	zfs
/zpbg/zones	zfs

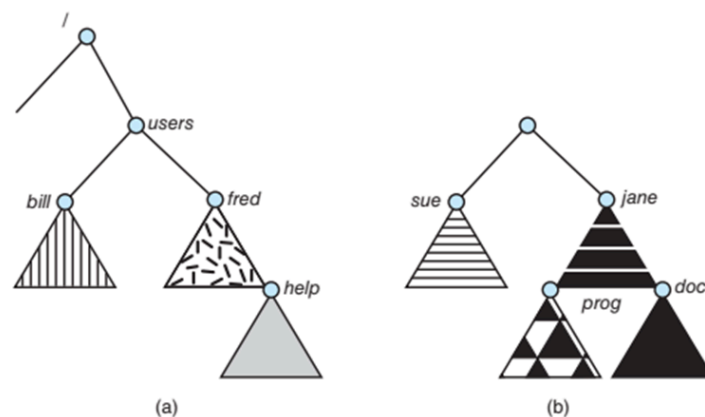
Figur File Sistem di Sistem Operasi Solaris

```
> df -T
```

Filesystem	Type	1K-blocks	Used	Available	Use%	Mounted on
/dev/nvme0n1p2	btrfs	239798208	25323412	212134652	11%	/
devtmpfs	devtmpfs	4096	0	4096	0%	/dev
tmpfs	tmpfs	2955404	4456	2950948	1%	/dev/shm
efivarfs	efivarfs	178	47	127	27%	/sys/firmware/efi/efivars
tmpfs	tmpfs	1182164	1564	1180600	1%	/run
/dev/nvme0n1p2	btrfs	239798208	25323412	212134652	11%	/home
/dev/nvme0n1p2	btrfs	239798208	25323412	212134652	11%	/var/cache
/dev/nvme0n1p2	btrfs	239798208	25323412	212134652	11%	/var/log
tmpfs	tmpfs	2955404	42100	2913304	2%	/tmp
/dev/nvme0n1p1	vfat	1021984	584	1021400	1%	/boot/efi
tmpfs	tmpfs	591080	64	591016	1%	/run/user/1000

Figur File Sistem di Arch Linux

Selanjutnya, sistem operasi memeriksa apakah perangkat memiliki sistem file yang valid. Hal ini dilakukan dengan meminta driver perangkat untuk membaca direktori perangkat dan memverifikasi bahwa direktori tersebut dalam format yang diinginkan. Terakhir, sistem operasi mencatat dalam struktur direktori bahwa sistem file dipasang pada titik pemasangan yang ditentukan. Model ini memungkinkan sistem operasi untuk menavigasi direktori, beralih antar sistem file, dan bahkan antar jenis sistem file yang berbeda jika diperlukan.

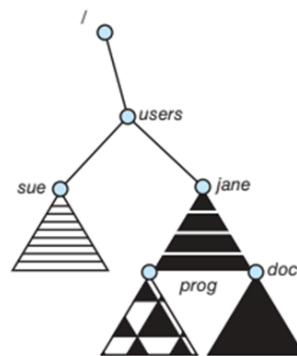


Gambar (a) Existing System      (b) Unmounted Volume

Untuk mengilustrasikan mount file, perhatikan sistem file yang digambarkan pada Gambar diatas, dimana segitiga mewakili subtree dari direktori yang diinginkan. Gambar (a) menunjukkan sistem berkas yang sudah ada, sedangkan Gambar (b) menunjukkan volume yang belum disambungkan yang berada di /device/dsk. Pada titik ini, hanya berkas pada sistem berkas yang ada yang dapat diakses.

Sistem menerapkan semantik untuk menjelaskan fungsionalitas. Misalnya, sistem mungkin tidak mengizinkan pemasangan ke direktori yang berisi file. atau

dapat menyediakan sistem file yang melekat pada direktori tersebut dan mengganti file dalam direktori tersebut dengan sistem file yang ada hingga sistem file dilepas. Dalam hal ini, sistem file dihapus dan file asli dalam direktori diperbolehkan. Sebagai contoh lain, sistem mungkin mengizinkan sistem file yang sama untuk dipasang berulang kali ke titik pemasangan yang berbeda; atau izinkan hanya satu mount per sistem file.



Volume mounted at /users

Keluarga sistem operasi Microsoft Windows memelihara direktori dua tingkat yang diperluas dengan huruf kandar yang ditetapkan ke perangkat dan volume. Setiap volume memiliki direktori grafis umum yang terkait dengan huruf drive. Jalur ke file tertentu berbentuk surat perintah: \path\file. Versi Windows yang lebih baru memungkinkan sistem file dipasang di direktori mana pun, seperti UNIX. Sistem operasi Windows secara otomatis mendeteksi semua perangkat dan memasang sistem file apa pun yang ada saat startup. Pada beberapa sistem, seperti UNIX, perintah mount bersifat eksplisit. File konfigurasi sistem berisi daftar perangkat dan antarmuka untuk instalasi otomatis saat startup, tetapi antarmuka lain dapat dibuat secara manual.

### 3.3 Partitions And Mounting

Tata letak disk dapat memiliki banyak variasi tergantung pada sistem operasi dan perangkat lunak manajemen volume. Sebuah disk dapat dipartisi, atau suatu volume dapat menjangkau beberapa partisi pada beberapa disk. Pengaturan pertama dibahas di sini, sedangkan pengaturan kedua lebih tepat dianggap sebagai bentuk RAID. Setiap partisi bisa berupa "raw", yang tidak berisi sistem file, atau "cooked", yang berisi sistem file. Jika tidak ada sistem file yang sesuai, disk mentah digunakan. Misalnya, ruang swap UNIX dapat menggunakan partisi mentah karena menggunakan formatnya sendiri pada disk dan tidak menggunakan



sistem file. Disk mentah juga dapat menyimpan informasi yang diperlukan untuk sistem diskRAID, seperti bitmap yang menunjukkan blok mana yang dicerminkan dan mana yang telah berubah serta perlu dicerminkan. Demikian pula, disk mentah dapat berisi database mini yang menyimpan informasi konfigurasi RAID, seperti disk mana yang menjadi anggota setiap kumpulan RAID.

Jika partisi berisi sistem file yang dapat di-booting yang memiliki sistem operasi yang terinstal dan terkonfigurasi dengan benar maka partisi tersebut juga membutuhkan informasi boot. Informasi ini memiliki formatnya sendiri, karena pada saat booting, sistem tidak memiliki kode sistem berkas yang dimuat sehingga tidak dapat menginterpretasikan format sistem berkas. Sebaliknya, informasi boot biasanya berupa serangkaian blok berurutan yang dimuat sebagai image ke dalam memori. Eksekusi image dimulai pada lokasi yang telah ditentukan, seperti byte pertama. Image ini, bootstrap loader, pada gilirannya mengetahui cukup banyak tentang struktur sistem berkas untuk dapat menemukan dan memuat kernel dan mulai mengeksekusinya.

Boot loader dapat berisi lebih dari sekadar instruksi untuk memboot sistem operasi tertentu. Sebagai contoh, banyak sistem yang dapat di-booting ganda, sehingga kita dapat menginstal beberapa sistem operasi pada satu sistem. Setelah dimuat, boot loader dapat mem-boot salah satu sistem operasi yang tersedia di drive. Drive dapat memiliki beberapa partisi, masing-masing berisi jenis sistem file dan sistem operasi yang berbeda. Perhatikan bahwa jika boot loader tidak memahami format sistem file tertentu, sistem operasi yang tersimpan pada sistem file tersebut tidak dapat di boot. Ini adalah salah satu alasan mengapa hanya beberapa sistem file yang didukung sebagai sistem file root untuk sistem operasi tertentu.

Jika partisi berisi sistem file yang dapat di-boot dengan sistem operasi yang diinstal dan dikonfigurasi dengan benar, partisi tersebut juga memerlukan informasi boot. Informasi ini memiliki formatnya sendiri karena sistem tidak memiliki kode sistem file yang dimuat saat startup, sehingga tidak dapat menafsirkan format sistem file. Sebaliknya, data boot biasanya berupa rangkaian blok berurutan yang dimuat ke dalam memori sebagai gambar. Pengisian gambar dimulai pada tempat tertentu, misalnya suku kata pertama. Gambar ini, bootloader, pada gilirannya mengetahui cukup banyak tentang struktur sistem file untuk menemukan dan memuat kernel dan mulai mengeksekusinya. Bootloader dapat berisi lebih dari sekedar instruksi untuk mem-boot sistem operasi tertentu. Misalnya, banyak sistem yang bersifat dual-boot, sehingga kita dapat menginstal beberapa sistem operasi dalam satu sistem. Bagaimana sistem mengetahui sistem mana yang harus dijalankan? Bootloader yang memahami banyak sistem file dan

beberapa sistem operasi dapat mengekspor ruang boot. Setelah dimuat, bootloader dapat mem-flash sistem operasi apa pun di drive. Sebuah disk dapat memiliki beberapa partisi, masing-masing berisi jenis sistem file dan sistem operasi yang berbeda. Perhatikan bahwa jika bootloader tidak memahami format sistem file tertentu, sistem operasi yang disimpan pada sistem file tersebut tidak dapat di-boot. Inilah salah satu alasan mengapa hanya sedikit sistem file yang didukung sebagai sistem file root di sistem operasi tertentu.

### **3.4 File Sharing**

Para pengguna tentu saja ingin berbagi file untuk berkolaborasi dan mengurangi upaya untuk mencapai tujuan komputasi. Oleh karena itu, sistem operasi berorientasi pada pengguna harus mengakomodasi kebutuhan untuk berbagi file.

Pada bagian ini, pembahasan mengenai berbagi file. Dimulai dari membahas masalah yang dihadapi pengguna saat berbagi file, kemudian mempertimbangkan apa yang harus dilakukan terhadap tindakan yang saling bertentangan yang terjadi pada file bersama.

#### **3.4.1 Multiple Users**

Ketika sebuah sistem operasi mengakomodasi banyak pengguna, masalah berbagi file, penamaan file, dan proteksi file menjadi hal yang utama. Dengan adanya struktur direktori yang memungkinkan file dibagikan oleh pengguna, sistem harus menjadi perantara dalam berbagi file. Sistem dapat mengizinkan pengguna untuk mengakses file pengguna lain secara default atau mengharuskan pengguna untuk secara khusus memberikan akses ke file tersebut.

Untuk berbagi dan melindungi, sistem harus menyimpan lebih banyak atribut file dan direktori daripada yang diperlukan pada sistem pengguna tunggal. Kebanyakan sistem menggunakan konsep pemilik file atau direktori dan grup untuk memenuhi kebutuhan ini. Pemilik adalah pengguna yang memiliki kontrol tertinggi atas file dan dapat mengubah atribut serta memberikan akses. Atribut grup menentukan pengguna mana yang dapat berbagi akses ke file.

Pemilik dan grup dari suatu file disimpan dengan atribut file lainnya. Saat pengguna meminta operasi pada sebuah file, ID pengguna dibandingkan dengan atribut pemilik untuk menentukan apakah pengguna tersebut merupakan pemilik file. Hal yang sama juga dilakukan dengan ID grup. Hasilnya menunjukkan izin

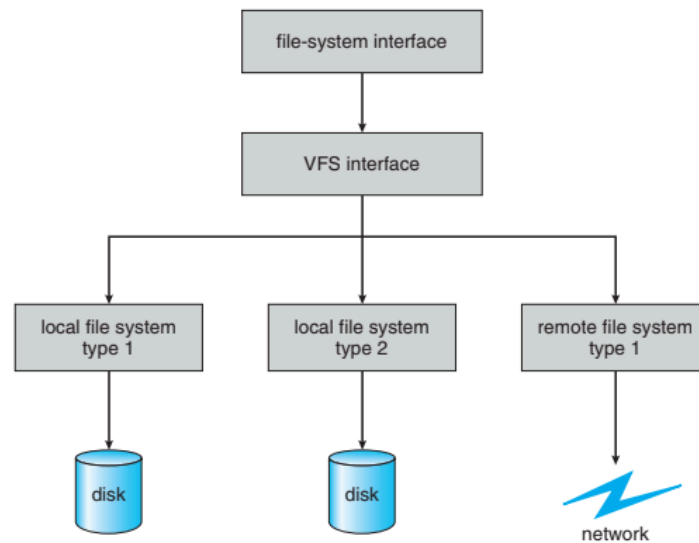
yang berlaku, dan sistem menerapkan izin tersebut pada operasi yang diminta. Sistem dengan beberapa sistem file lokal harus memeriksa ID dan mencocokkan izin setelah sistem file terpasang. Namun, ketika disk eksternal dipindahkan antar sistem, perlu memastikan bahwa ID cocok di antara sistem atau mendapatkan ulang kepemilikan file.

### **3.5 Virtual File Systems**

Seperti yang telah kita lihat, sistem operasi modern harus mendukung beberapa jenis sistem file secara bersamaan. Tetapi bagaimana sebuah sistem operasi mengizinkan beberapa jenis sistem file untuk diintegrasikan ke dalam struktur direktori? Dan bagaimana pengguna dapat berpindah antar jenis sistem berkas dengan lancar saat mereka menavigasi ruang sistem berkas?

Metode yang jelas tetapi kurang optimal untuk mengimplementasikan beberapa jenis sistem berkas adalah dengan menulis direktori dan rutinitas berkas untuk setiap jenis. Akan tetapi, kebanyakan sistem operasi, termasuk UNIX, menggunakan teknik berorientasi objek untuk menyederhanakan, mengorganisir, dan memodulasi implementasi. Penggunaan metode ini memungkinkan jenis sistem berkas yang sangat berbeda diimplementasikan dalam struktur yang sama, termasuk sistem berkas jaringan, seperti NFS. Pengguna dapat mengakses berkas yang terdapat dalam beberapa sistem berkas pada drive lokal atau bahkan pada sistem berkas yang tersedia di jaringan.

Struktur dan prosedur data digunakan untuk mengisolasi fungsionalitas pemanggilan sistem dasar dari detail implementasi. Dengan demikian, implementasi sistem berkas terdiri dari tiga lapisan utama, seperti yang digambarkan secara skematis pada Gambar 15.5. Lapisan pertama adalah antarmuka sistem berkas, berdasarkan pada panggilan `open()`, `read()`, `write()`, dan `close()` dan pada deskriptor berkas.



Gambar tampilan skematik sistem file virtual.

Lapisan kedua dalam sistem file disebut sebagai lapisan sistem file virtual (VFS).

Lapisan ini memiliki dua fungsi penting, yaitu:

1. Lapisan ini memisahkan operasi umum sistem berkas dan implementasinya melalui penggunaan antarmuka VFS yang bersih. Hal ini memungkinkan akses transparan ke berbagai jenis sistem file yang terpasang secara lokal.
2. VFS menyediakan mekanisme untuk merepresentasikan file secara unik di seluruh jaringan menggunakan struktur vnode.

VFS membedakan antara berkas lokal dan berkas remote, serta membedakan berkas lokal berdasarkan tipe sistem berkas mereka.

Empat tipe objek utama yang didefinisikan oleh VFS Linux adalah:

- Objek inode, yang mewakili sebuah file individual
- Objek fil, yang merepresentasikan sebuah berkas terbuka
- Objek superblok, yang mewakili seluruh sistem file

- Objek dentry, yang mewakili entri direktori individual

Untuk masing-masing dari keempat tipe objek ini, VFS mendefinisikan serangkaian operasi yang dapat diimplementasikan. Setiap objek dari salah satu tipe ini berisi penunjuk ke tabel fungsi. Tabel fungsi mencantumkan alamat fungsi aktual yang mengimplementasikan operasi yang ditentukan untuk objek tertentu. Sebagai contoh, API yang disingkat untuk beberapa operasi untuk objek file termasuk:

- `int open( . . . )` - Membuka sebuah file.
- `int close( . . . )` - Menutup file yang sudah terbuka.
- `ssize_t read( . . . )` - Membaca dari sebuah file.
- `ssize_t write( . . . )` - Menulis ke sebuah file.
- `int mmap( . . . )` - Memetakan memori sebuah file.

Implementasi objek file untuk tipe file tertentu diperlukan untuk mengimplementasikan setiap fungsi yang ditentukan dalam definisi objek file. (Definisi lengkap definisi objek file ditentukan dalam operasi file struct file, yang terletak pada berkas `/usr/include/linux/fs.h`).

Lapisan perangkat lunak VFS dapat melakukan operasi pada objek dengan memanggil fungsi yang sesuai. VFS tidak perlu mengetahui jenis objek yang ditangani. Fungsi yang sesuai untuk operasi `read()` pada berkas akan selalu berada di tempat yang sama dalam tabel fungsinya. VFS akan memanggil fungsi tersebut tanpa memperhatikan bagaimana data sebenarnya dibaca.

### **3.6 Remote File Systems**

Sistem berkas jarak jauh telah mengalami evolusi dari mentransfer file secara manual melalui program seperti ftp hingga menggunakan sistem file terdistribusi (DFS) dan World Wide Web. Metode berbagi file ini melibatkan

akses anonim dan akses yang diautentikasi, dengan ftp umumnya digunakan untuk akses anonim dan World Wide Web untuk pertukaran file anonim. Integrasi yang lebih ketat terjadi dalam DFS antara mesin yang mengakses dan menyediakan file, menambah kompleksitas dalam berbagi file jarak jauh.

### **3.6.1 The Client-Server Model**

Sistem berkas jarak jauh memungkinkan komputer untuk mengakses sistem berkas dari mesin jarak jauh. Mesin dengan berkas-berkas disebut server, sedangkan mesin yang mencari akses disebut klien. Hubungan klien-server umum dalam jaringan. Identifikasi klien sulit karena dapat dipalsukan. Solusi keamanan meliputi otentikasi klien melalui kunci terenkripsi, namun ini memiliki tantangan dalam kompatibilitas algoritma enkripsi dan pertukaran kunci yang aman. Pada kasus UNIX dan sistem berkas jaringan (NFS), autentikasi biasanya dilakukan melalui informasi jaringan klien. NFS memungkinkan hubungan banyak-ke-banyak, di mana banyak server dapat menyediakan berkas ke banyak klien. Setelah sistem berkas jarak jauh dipasang, permintaan operasi file dikirim ke server melalui protokol DFS, biasanya dengan ID pengguna yang meminta. Server menerapkan pemeriksaan akses standar untuk menentukan hak akses pengguna, kemudian menyetujui atau menolak permintaan

### **3.6.1 Distributed Information Systems**

Sistem informasi terdistribusi menyediakan akses terpadu ke informasi yang diperlukan untuk komputasi jarak jauh. Contohnya, DNS menerjemahkan nama host ke alamat jaringan. NIS dan NIS+ dari Sun Microsystems, serta Active Directory dari Microsoft, adalah contoh sistem informasi terdistribusi yang menyimpan informasi seperti nama pengguna, kata sandi, dan informasi host. Mereka memiliki tantangan keamanan, seperti pengiriman kata sandi tanpa enkripsi. Microsoft menggunakan Active Directory dengan protokol autentikasi jaringan Kerberos. Industri beralih ke LDAP untuk keamanan yang lebih baik. LDAP digunakan dalam Oracle Solaris dan sistem operasi utama lainnya untuk otentikasi pengguna dan pengambilan informasi sistem secara terdistribusi. LDAP memungkinkan satu direktori terdistribusi digunakan untuk menyimpan informasi pengguna dan sumber daya, menciptakan sistem masuk tunggal yang aman dan menyederhanakan administrasi sistem.

### **3.6.1 Failure Modes**

Sistem berkas lokal dapat mengalami kegagalan karena berbagai alasan seperti kegagalan drive, kerusakan struktur direktori, atau pengontrol disk. Kegagalan ini memerlukan intervensi manusia untuk memperbaikinya. Sistem berkas jarak jauh

memiliki lebih banyak mode kegagalan karena kompleksitas jaringan. Jaringan yang terganggu atau kerusakan server dapat mengganggu operasi sistem file jarak jauh. Protokol DFS mengizinkan penundaan operasi atau penghentian operasi ke server yang tidak tersedia. Informasi status dapat membantu pemulihan dari kegagalan, dan protokol NFS Versi 3 mengasumsikan bahwa permintaan dari klien adalah sah, yang membuatnya mudah diimplementasikan tetapi kurang aman. Masalah ini ditangani dalam standar NFS Versi 4 dengan membuat NFS stateful untuk meningkatkan keamanan dan kinerja.

### **3.7 Consistency Semantics**

Sistem berkas lokal dapat mengalami kegagalan karena berbagai alasan seperti kegagalan drive, kerusakan struktur direktori, atau pengontrol disk. Kegagalan ini memerlukan intervensi manusia untuk memperbaikinya. Sistem berkas jarak jauh memiliki lebih banyak mode kegagalan karena kompleksitas jaringan. Jaringan yang terganggu atau kerusakan server dapat mengganggu operasi sistem file jarak jauh. Protokol DFS mengizinkan penundaan operasi atau penghentian operasi ke server yang tidak tersedia. Informasi status dapat membantu pemulihan dari kegagalan, dan protokol NFS Versi 3 mengasumsikan bahwa permintaan dari klien adalah sah, yang membuatnya mudah diimplementasikan tetapi kurang aman. Masalah ini ditangani dalam standar NFS Versi 4 dengan membuat NFS stateful untuk meningkatkan keamanan dan kinerja.

#### **3.7.1 UNIX Semantics**

Semantik konsistensi dalam sistem berkas UNIX adalah sebagai berikut: perubahan yang dilakukan oleh satu pengguna pada sebuah berkas yang terbuka akan langsung terlihat oleh pengguna lain yang membuka berkas tersebut. Mode berbagi tertentu memungkinkan pengguna untuk berbagi penunjuk lokasi saat ini dalam file, sehingga perubahan penunjuk oleh satu pengguna akan mempengaruhi semua pengguna yang berbagi. Dalam semantik UNIX, sebuah berkas diasosiasikan dengan gambar fisik tunggal yang diakses secara eksklusif, yang menyebabkan penundaan dalam proses pengguna.

#### **3.7.2 Sessions Semantics**

Semantik konsistensi dalam sistem berkas UNIX adalah sebagai berikut: perubahan yang dilakukan oleh satu pengguna pada sebuah berkas yang terbuka akan langsung terlihat oleh pengguna lain yang membuka berkas tersebut. Mode berbagi tertentu memungkinkan pengguna untuk berbagi penunjuk lokasi saat ini dalam file, sehingga perubahan penunjuk oleh satu pengguna akan mempengaruhi semua pengguna yang berbagi. Dalam semantik UNIX, sebuah berkas

diasosiasikan dengan gambar fisik tunggal yang diakses secara eksklusif, yang menyebabkan penundaan dalam proses pengguna.

### **3.7.3 Immutable-Shared-Files Semantics**

Semantik file bersama yang tidak dapat diubah adalah pendekatan di mana sebuah berkas, setelah dideklarasikan sebagai file bersama oleh pembuatnya, tidak dapat dimodifikasi. File ini memiliki dua sifat utama: nama file tidak dapat digunakan kembali, dan isinya tidak dapat diubah. Dengan demikian, nama file yang tidak dapat diubah menandakan bahwa isi file tersebut tetap tidak berubah. Implementasi semantik ini dalam sistem terdistribusi adalah sederhana karena pembagian hanya bersifat baca

## **3.8 NFS**

NFS adalah sistem berkas jaringan yang digunakan secara luas untuk mengakses file jarak jauh di LAN atau WAN. Implementasi NFS adalah bagian dari ONC+ dan didukung oleh banyak vendor UNIX serta beberapa sistem operasi PC. Versi yang paling umum digunakan adalah Versi 3, meskipun ada versi yang lebih baru seperti Versi 4. NFS diimplementasikan dalam sistem operasi Solaris dengan protokol TCP atau UDP/IP. Versi yang dijelaskan di sini adalah Versi 3.

### **3.8.1 Overview**

NFS memperlakukan sekelompok workstation yang saling terhubung sebagai sejumlah independen sistem berkas. Tujuannya adalah untuk memungkinkan berbagi antara sistem berkas ini secara transparan berdasarkan permintaan eksplisit. Berbagi didasarkan pada hubungan klien-server, di mana sebuah mesin bisa bertindak sebagai klien, server, atau keduanya. Agar direktori jarak jauh dapat diakses secara transparan dari sebuah mesin, klien harus melakukan operasi mount, yang melibatkan me-mount sebuah direktori jarak jauh di atas direktori sistem berkas lokal. Setelah proses mount selesai, direktori yang di-mount akan terlihat seperti bagian integral dari sistem berkas lokal, menggantikan subpohon yang diturunkan dari direktori lokal. Ini memungkinkan pengguna untuk mengakses berkas-berkas pada direktori remote secara transparan. Gambaran pada Gambar 15.6 dan 15.7(a) mengilustrasikan efek dari pemasangan berkas dari sebuah mesin server ke sebuah mesin klien. Dengan izin hak akses yang sesuai, sistem file atau direktori apa pun dalam sistem file dapat dipasang dari jarak jauh di atas direktori lokal manapun. NFS memungkinkan pemasangan sistem berkas jarak jauh, memungkinkan pengguna mengakses



berkas dari workstation tanpa disk. Mekanisme mount tidak transitif, namun memungkinkan pengguna mengakses berkas di dalam sistem berkas jarak jauh. NFS dirancang untuk lingkungan heterogen dan memisahkan layanan pemasangan dan akses berkas menggunakan protokol RPC yang terpisah.

### **3.8.2 The Mount Protocol**

Protokol mount menghubungkan klien dengan server, memungkinkan pemetaan nama direktori jarak jauh dan nama mesin server. Permintaan mount diteruskan ke server mount, yang mempertahankan daftar ekspor dan tabel mount untuk akses. Setiap permintaan mount menghasilkan pegangan file yang berisi informasi untuk mengakses file dalam sistem berkas yang disambungkan. Server juga menyimpan daftar klien dan direktori yang terpasang untuk tujuan administratif. Protokol mount mencakup operasi unmount dan mengembalikan daftar ekspor

### **3.8.3 The NFS Protocol**

Protokol NFS menyediakan sekumpulan RPC untuk operasi berkas jarak jauh. Prosedur-prosedur ini mendukung operasi-operasi berikut

- Mencari file di dalam direktori
- Membaca sekumpulan entri direktori
- Memanipulasi tautan dan direktori
- Mengakses atribut file
- Membaca dan menulis file

Server NFS tidak menyimpan informasi klien antara akses, menjadikannya server tanpa kewarganegaraan. Setiap permintaan NFS harus menyediakan argumen lengkap, termasuk pengenalan file unik dan offset absolut. Kerusakan server tidak mempengaruhi operasi klien karena data dimodifikasi harus disinkronkan ke disk server sebelum dikembalikan ke klien. Performa dapat terpengaruh karena penulisan sinkron, tetapi dapat ditingkatkan dengan penggunaan cache yang tidak mudah menguap. Panggilan prosedur penulisan NFS dijamin bersifat atomik tetapi tidak menyediakan mekanisme kontrol konkurensi, sehingga pengguna disarankan untuk mengkoordinasikan akses menggunakan mekanisme di luar NFS. NFS diintegrasikan ke dalam sistem operasi melalui Virtual File System (VFS). Ketika sebuah operasi pada berkas jarak jauh yang sudah terbuka dilakukan, panggilan sistem dari klien dipetakan ke operasi VFS yang sesuai pada vnode. Lapisan VFS mengidentifikasi berkas sebagai berkas jarak jauh dan

memanggil prosedur NFS yang sesuai. Panggilan Remote Procedure Call (RPC) dilakukan ke layanan NFS di server jauh. Hasilnya kemudian dikembalikan melalui jalur yang sama. Keuntungan arsitektur ini adalah klien dan server dapat berperan secara identik, dan layanan pada setiap server dilakukan oleh thread kernel

### **3.8.4 Path-Name Translation**

Penerjemahan nama jalur di NFS melibatkan penguraian path menjadi komponen terpisah dan melakukan pencarian terpisah untuk setiap komponen pada server. Ini memakan waktu karena setiap pencarian membutuhkan RPC terpisah ke server. Cache pencarian nama direktori di sisi klien mempercepat proses dengan menyimpan vnode untuk direktori jarak jauh. Tembolok ini dibuang ketika atribut dari server tidak cocok dengan atribut dari vnode. Beberapa implementasi NFS memungkinkan pemasangan bertingkat, di mana lebih dari satu server terlibat dalam penelusuran nama path. Namun, saat klien melakukan pencarian pada direktori di mana server melakukan mount sistem berkas, klien akan melihat direktori mendasari, bukan yang di-mount.

### **3.8.5 Remote Operations**

Penerjemahan nama jalur di NFS melibatkan penguraian path menjadi komponen terpisah dan melakukan pencarian terpisah untuk setiap komponen pada server. Ini memakan waktu karena setiap pencarian membutuhkan RPC terpisah ke server. Cache pencarian nama direktori di sisi klien mempercepat proses dengan menyimpan vnode untuk direktori jarak jauh. Tembolok ini dibuang ketika atribut dari server tidak cocok dengan atribut dari vnode. Beberapa implementasi NFS memungkinkan pemasangan bertingkat, di mana lebih dari satu server terlibat dalam penelusuran nama path. Namun, saat klien melakukan pencarian pada direktori di mana server melakukan mount sistem berkas, klien akan melihat direktori mendasari, bukan yang di-mount.