File System

Bojan Nokovic

Based on: "Operating Systems Concepts", 10th Edition Silberschatz Et al.

Mar. 2021

File Concept

OS provides uniform logical view of stored information

File - a logical storage unit mapped by OS into physical devices.

Data cannot be written to secondary storage unless they are within a file.

Data file types: numeric, alphanumeric, alphabetic, binary

Contents defined by file's creator

- text file
- source file
- executable file

File Attributes

- Name - only information kept in human-readable form
- Identifier - unique tag (number) identifies file within file system
- Type - needed for systems that support different types
- Location - pointer to file location on device
- Size - current file size
- Protection - controls who can do reading, writing, executing
- Time, date, and user identification
- Information about files are kept in the directory structure, which is maintained on the disk

File Operations

File is an abstract data type

- Create
- Write - at write pointer location
- Read - at read pointer location
- Reposition within file - seek
- Delete
- Truncate - shrink or extend the size of a file to the specified size
- Open(F_i) - search the directory structure on disk for entry F_i, and move the content of entry to memory
- Close (F_i) - move the content of entry F_i from memory to directory structure on disk

Open Files

Several pieces of data are needed to manage open files:

- Open-file table: tracks open files
- File pointer: pointer to last read/write location, per process that has the file open
- File-open count: counter of number of times a file is open to allow removal of data from open-file table when last processes closes it
- Disk location of the file: cache of data access information
- Access rights: per-process access mode information

Open File Locking

Provided by some operating systems and file systems

- Similar to reader-writer locks
- Shared lock similar to reader lock - several processes can acquire concurrently
- Exclusive lock similar to writer lock

File-locking mechanisms

- Mandatory - access is denied depending on locks held and requested (Windows)
- Advisory - processes can find status of locks and decide what to do (Unix)

File Types - Name, Extension

file type	usual extension	function
executable	exe, com, bin or none	ready-to-run machine-language program
object	obj, o	compiled, machine language, not linked
source code	c, cc, java, perl, asm	source code in various languages
batch	bat, sh	commands to the command interpreter
markup	xml, html, tex	textual data, documents
word processor	xml, rtf, docx	various word-processor formats
library	lib, a, so, dll	libraries of routines for programmers
print or view	gif, pdf, jpg	ASCII or binary file in a format for printing or viewing
archive	rar, zip, tar	related files grouped into one file, sometimes compressed, for archiving or storage
multimedia	mpeg, mov, mp3, mp4, avi	binary file containing audio or A/V information

Internal File Structure

Internally, locating an offset within a file can be complicated for the operating system.

Disk systems typically have a well-defined block size determined by the size of a sector.

All disk I/O is performed in units of one block (physical record), and all blocks are the same size.

It is unlikely that the physical record size will exactly match the length of the desired logical record.

Packing a number of logical records into physical blocks is a common solution to this problem.

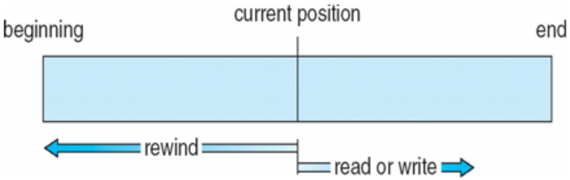
UNIX operating system defines all files to be simply streams of bytes - the logical record size is 1 byte.

All file systems suffer from internal fragmentation.

Access Methods

The information from file accessed and read into computer memory.

1. Sequential Access - information in the file processed in order, one record after the other.



2. Direct Access - file made of fixed length of logical records that allow programs to read and write records rapidly in no particular order - immediate access to large amounts of information (database).

   Relative block number (index relative to the beginning of the file) allow OS to decide where file should be placed.

Sequential Access on Direct-access File

Not all operating systems support both sequential and direct access for files.

Simulation of sequential access on direct-access file

sequential access	implementation for direct access
reset	$cp = 0;$
read next	*read cp*; $cp = cp + 1;$
write next	*write cp*; $cp = cp + 1;$

Keeping a variable *cp* that defines our current position.

Other Access Methods

Can be built on top of base methods

Creation of an index for the file

Keep index in memory for fast determination of location of data to be operated on.

IBM indexed sequential-access method (ISAM)

- Small master index, points to disk blocks of secondary index
- File kept sorted on a defined key
- All done by the OS

Example

A retail-price file might list the universal product codes (UPCs) for items, with the associated prices.

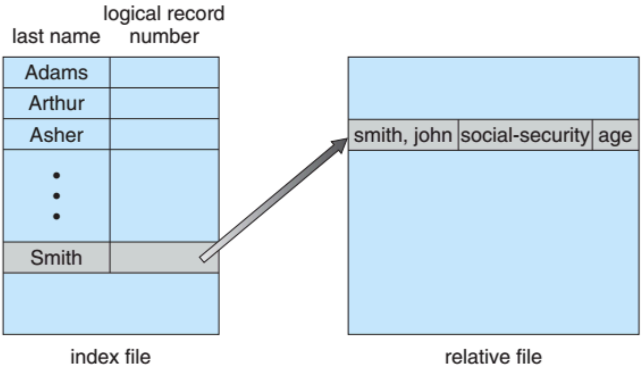
Each record consists of a 10-digit UPC and a 6-digit price, for a 16-byte record.

If our disk has 1,024 bytes per block, we can store 64 records per block. A file of 120,000 records would occupy about 2,000 blocks (2 million bytes).

By keeping the file sorted by UPC, we can define an index consisting of the first UPC in each block. This index would have 2,000 entries of 10 digits each, or 20,000 bytes, and thus could be kept in memory.

Index and Relative Files

Master index, points to disk blocks of secondary index.



VMS OS provides index and relative files like ISAM.

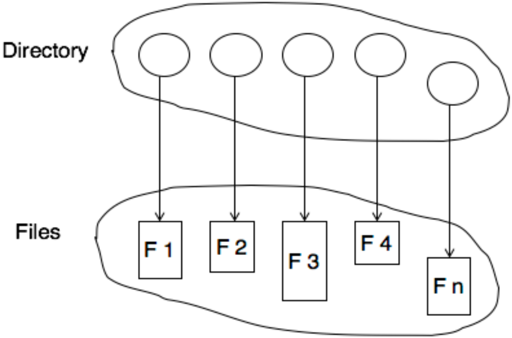
`https://www.vmssoftware.com/`

Directory Structure

The directory can be viewed as a symbol table that translates file names into their file control blocks.

Directory can be organized in many ways.

A collection of nodes containing information about all files



Both the directory structure and the files reside on disk

Disk Structure

Disk can be subdivided into partitions

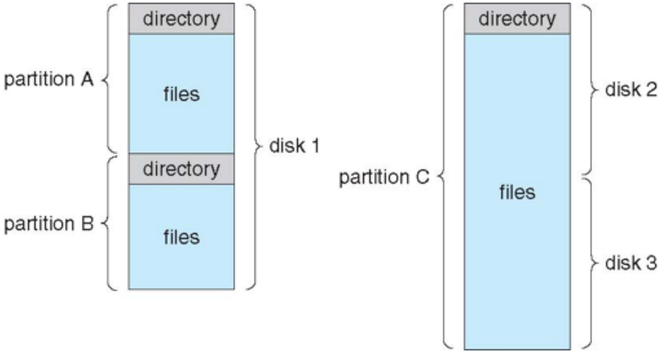
Disks or partitions can be RAID protected against failure

Disk or partition can be used raw - without a file system, or formatted with a file system

Entity containing file system in known as a volume

Each volume containing file system also tracks that file system's info in device directory or volume table of contents

A Typical File-system Organization



Directory Operations and Organization

Search for a file, create a file, delete a file, list a directory, rename a file, traverse the file system

The directory is organized logically to obtain:

Efficiency - locating a file quickly

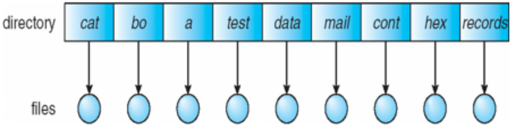
Naming - convenient to users

- Two users can have same name for different files
- The same file can have several different names

Grouping - logical grouping of files by properties, (e.g., all Java programs, all games, ...)

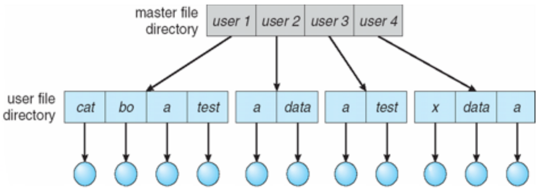
Single and Two-Level Directory

A single-level directory for all users.



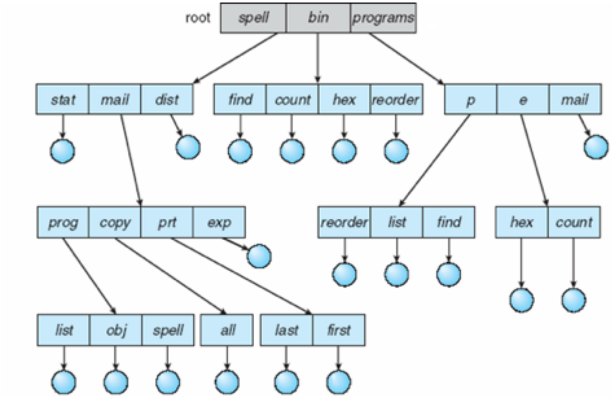
Naming problem, grouping problem

Separate directory for each user



Path name, can have the same file name for different user, efficient searching, no grouping capability

Tree-Structured Directories

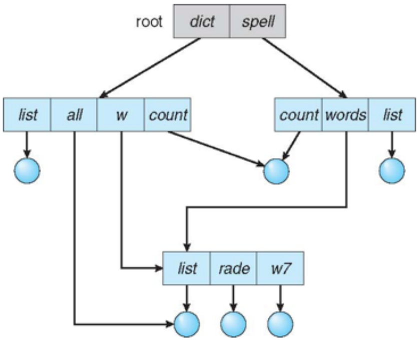


Efficient searching

Grouping capability

Acyclic-Graph Directories

Have shared subdirectories and files - the same file or subdirectory may be in two different directories.

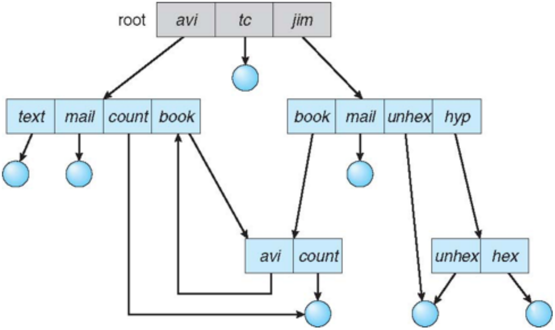


Two different names (aliasing)

New directory entry type

- Link - another name (pointer) to an existing file
- Resolve the link - follow pointer to locate the file

General Graph Directory



How do we guarantee no cycles?

- Allow only links to file not subdirectories
- Garbage collection
- Every time a new link is added use a cycle detection algorithm to determine whether it is OK

Protection

Mode of access: **r**ead, **w**rite, e**x**ecute

Three classes of users on Unix/Linux

Access			RWX
Owner	7	$\Longrightarrow$	111
Group	6	$\Longrightarrow$	110
Public	1	$\Longrightarrow$	001

Example

For a file `game` define an access:

`chmod 761 game`

Is this file executable?

Understanding Linux File Permissions

A sample directory listing from a UNIX environment.

```
drwxr-xr-sr-t  5 cupsys lp         4096 2006-11-29 08:51 cups
-rw-r--r--  1 root   root        817 2006-11-29 08:39 fstab
-rw-r--r--  1 root   root        806 2006-12-17 00:15 group
-rw-r--r--  1 root   root       1430 2006-12-17 00:15 passwd
lrwxrwxrwx  1 root   root         13 2006-11-29 08:40 motd -> /var/run/motd
drwxr-xr-x  2 root   root       4096 2006-12-22 23:36 rc0.d
```

The first element of this column is the type of the file.

- "-" means it's a normal file;
- "d" is for a directory and
- "l" is for a link pointing to a file.

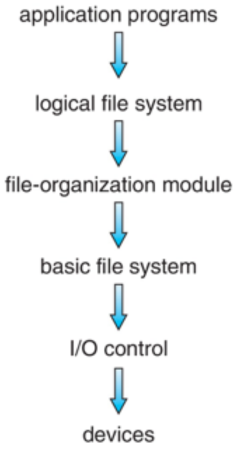
There is several other types of file, but they are much less useful to know for the casual Linux system administrator.

File-System Structure

File system resides on secondary storage (disks)

- Provided user interface to storage, mapping logical to physical
- Provides efficient and convenient access to disk

File system is organized into layers

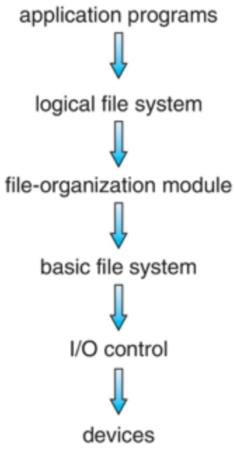


File-System Layers

Logical file system layer - manages metadata information (file-system structure)

- Translates symbolic file name into file number, file handle, location by maintaining file control blocks (`inodes` in Unix)
- Manages the file-system directory including protection and security

File organization module translates logical to physical block number, manages free space, disk allocation



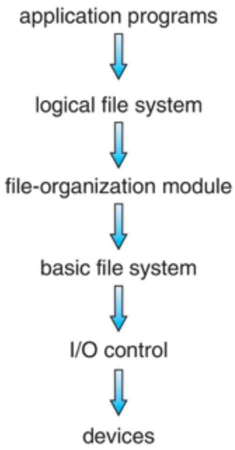
File-System Layers

Basic file system layer - gives commands to the device driver to read-/write physical block on the disk.

- Buffers hold data in transit
- Caches hold frequently used data

I/O control layer consists of device drivers and interrupt handlers to transfer translator information between main memory and the disk.

- Input - high-level commands
- Output - low-level hardware specific instructions



File System I/O control Layer

I/O control layer commands example

Input high-level command:
"retrieve block 123"

Output low-level command:
"read drive1, cylinder 72, track 2, sector 10, into memory location 1060"

File-System Operations

We have system calls at the API level ((e.g. `open()`, `close()`, `read()`, `write()`),), but how do we implement their functions?

- Use a combination of on-disk and in-memory structures

  On-disk structures:

Boot control block contains info needed by system to boot OS from that volume

- Needed if volume contains OS, usually first block of volume

Volume control block (superblock, master file table) contains volume details

- Total # of blocks, # of free blocks, block size, free block pointers or array

Directory structure organizes the files

- Names and inode numbers, master file table

Per-file File Control Block (FCB) contains many details about the file

- Permissions, inode number, size, dates
- NFTS stores into in master file table using relational DB structures

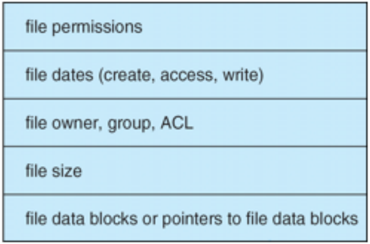


Figure: Typical File Control Block

In-Memory File System Structures

Several file system structures are maintained in memory

- Mount table storing file system mounts, mount points, file system types
- Cached portions of the directory structure
- A system-wide open-file table that contains a copy of the FCB for each open file
- A per-process open-file table that contains a pointer to the appropriate entry in the system-wide open-file table
- Buffers for assisting in the reading/writing of information from/to disk

In-Memory File System Structures

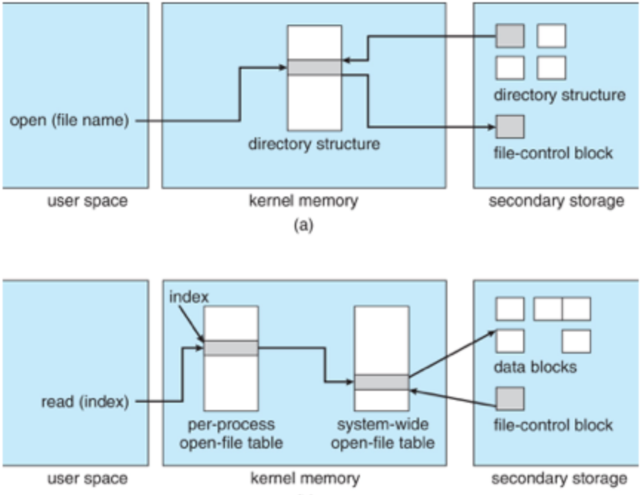


Figure (a) refers to opening a file, (b) refers to reading a file

Directory Implementation

Directory maintains a symbolic list of file names with pointers to the data blocks

Different algorithms can be used for directory implementation

- Linear list of file names with pointer to the data blocks
  -Simple to program
  -Time-consuming to execute
  - Linear search time
  - Could keep ordered alphabetically via linked list or use B+ tree
- Hash Table - linear list with hash data structure
  -Decreases directory search time
  -Collisions, two file names hash to the same location (use chaining for collision resolution)

Allocation Methods

An allocation method refers to how disk blocks are allocated to files

- Want to utilize disk space efficiently
- Want files to be accessed quickly

Three main methods currently used for disk allocation

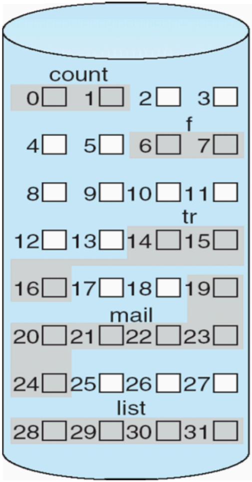
- Contiguous Allocation
- Linked Allocation
- Indexed Allocation

Allocation Methods - Contiguous Allocation

Contiguous allocation - each file occupies set of contiguous blocks

- Best performance in most cases
- Simple - only starting location (block #) and length (number of blocks) are required
- Problems include:
  - finding space for file
  - knowing file size
  - external fragmentation
  - need for compaction off-line (downtime) or on-line

Contiguous Allocation



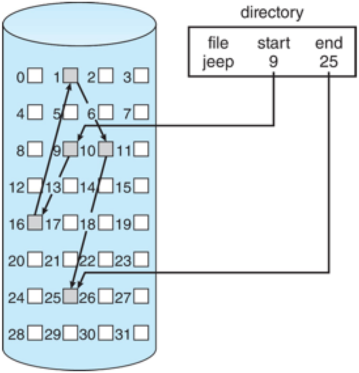
directory

file	start	length
count	0	2
tr	14	3
mail	19	6
list	28	4
f	6	2

Allocation Methods - Linked Allocation

Linked allocation - each file a linked list of blocks

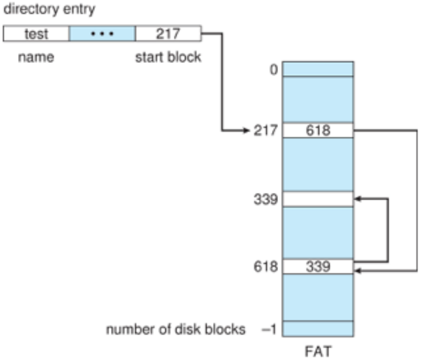
- Each block contains pointer to next block
- No external fragmentation, thus no compaction necessary
- Free space management system called when new block needed
- Improve efficiency by clustering blocks into groups, but this increases internal fragmentation
- Reliability can be a problem
- Locating a block can take many I/Os and disk seeks



Allocation Methods - Linked Allocation with FAT

FAT (File Allocation Table)

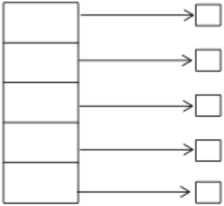
- Beginning of a FAT volume has table, indexed by block number
- Linked list of block numbers for files
- Can be cached in memory
- New block allocation simple, simply find an open slot in FAT



Indexed allocation

- Each file has its own index block(s) of pointers to its data blocks

Logical view

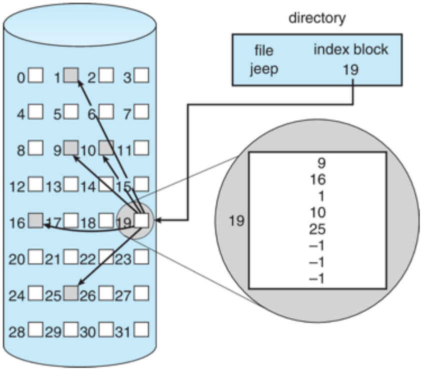


index table

Example of Indexed Allocation

Indexed allocation - each file has its own index block (or blocks) of pointers to its data blocks

- Directory contains the address of the index block
- Location *i* in the index points to block *i* of the file
- Supports direct access (like contiguous allocation) without external fragmentation
- May waste disk resources - an entire index block must be created, even for small files that only require a few pointers



Allocation Methods - Indexed Allocation (Cont.)

How large should the index be?

- If too large, then wasting a lot of space (every file has an index block)
- If too small, then may not have enough space to index all of the blocks required for very large files

Several schemes are available to allow index blocks to be small enough so as not to be too wasteful, but ’expandable’ so large files can be handled.

Allocation Methods - Indexed Allocation (Cont.)

Linked scheme - allow index blocked to be linked together to handle large files

- If the file is larger than can be handled by a single index block, the last word of an index block is used to store a pointer to the next linked block
- If the file is small enough so as to require only a single index block, then the last word of the index block is a null pointer

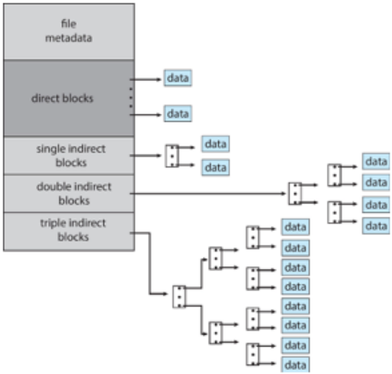
Multilevel index - a first-level index block points to a set of second-level index blocks which, in turn, point to the data blocks for the file

- A tree structure of index blocks that can be expanded by adding additional levels

Allocation Methods - Indexed Allocation (Cont.)

A Combined scheme - reserves some space in the first index block to point directly to data block, while others point to multilevel indexes

- For small files only the first index is required which points directly to data block
- Larger files may require some single-indirect indexes
- Even larger files may require double or triple-indirect indexes



Example: Maximum size

Consider a file system that uses inodes to represent files. Disk blocks are 4 KB in size, and a pointer to a disk block requires 4 bytes. This file system has 12 direct disk blocks, as well as one single, and one double indirect disk blocks. What is the maximum size of a file that can be stored in this file system?

Answer: 12*4KB + 1024*4KB + 1024*1024*4KB

Free-Space Management

File system maintains free-space list to track available blocks/clusters

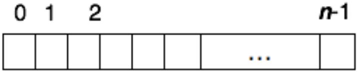
Free-space list can be implemented in a variety of ways

- Bit Vector
- Linked List
- Linked List with Grouping
- Contiguous Block Counting

Free-Space Management - Bit Vector

Utilize a bit vector in which each bit represents a block

Bit vector or bit map (n blocks)



$$\text{bit}[i] = \begin{cases} 1 \Rightarrow \text{block}[i] \text{ free} \\ 0 \Rightarrow \text{block}[i] \text{ occupied} \end{cases}$$

When looking for a free block, find the first '1' in the bit vector

Bit vector requires extra space for storage

Linked Free Space List on Disk

A linked list can be used to maintain the free-list

- A head pointer identifies the first free block
- Each block contains a pointer to the next free block
- Cannot get contiguous space easily
- No wasted space like with the bit vector implementation

Free-Space Management - Linked List with Grouping

In the linked list implementation, getting a large group of free blocks requires traversing the linked list to find each free block

Rather than having only a reference to a single unallocated block in each linked list node, each node can contain a list of free blocks

Modify linked list to store address of next $n - 1$ free blocks in first free block, plus a pointer to next block that contains free-block-pointers.

Free-Space Management - Contiguous Block Counting

Space is frequently used and freed contiguously, with contiguous-allocation or clustering

- When freeing a contiguous section of blocks, no need to maintain information about all of them
- Keep address of first free block in a contiguous section of free space and a count of how many free blocks follow it
- Free space list then has entries containing addresses and counts

Recovery

Crashes, bugs, power outages, etc. may leave the file system in an inconsistent state.

Consistency checking - compares data in directory structure with data blocks on disk, and tries to fix inconsistencies

- Can be slow and sometimes fails
- Associate a status bit with each file, set that bit prior to making changes to the file, unset that bit only when all changes are complete

Restoring data from backup

Log Structured File Systems

Log structured (or journaling) file systems record each metadata update to the file system as a transaction

All transactions are written to a log

- A transaction is considered committed once it is written to the log (sequentially)
- Sometimes to a separate device or section of disk
- However, the file system may not yet be updated

The transactions in the log are asynchronously written to the file system structures

- When the file system structures are modified, the transaction is removed from the log

If the file system crashes, all remaining transactions in the log must still be performed

Faster recovery from crash, removes chance of inconsistency of metadata.

Thank you !

Operating Systems are among the most complex pieces of software ever developed !