FREIE UNIVERSITÄT BERLIN

Operating Systems & Computer Networks
6. I/O & File System

Dr. Larissa Groth
Computer Systems & Telematics (CST)

Roadmap

1. Introduction and Motivation
2. Interrupts and System Calls
3. Processes
4. Scheduling
5. Memory
6. **I/O and File System**
7. Booting, Services, and Security

Lernziele I

- Sie nennen:
  - die wesentlichen Ziele eines general-purpose BS im Umgang mit Ein- und Ausgabegeräten

- Sie nennen und beschreiben:
  - Programmiertechniken im Umgang mit Ein- und Ausgabegeräten
  - warum Buffering im Umgang mit Ein- und Ausgabegeräten sinnvoll sein kann
  - Implementierungsmöglichkeiten für I/O-Buffering

- Sie beschreiben:
  - was RAID ist
  - welche 4 Methoden der File Allocation auf einer Festplatte vorgestellt wurden

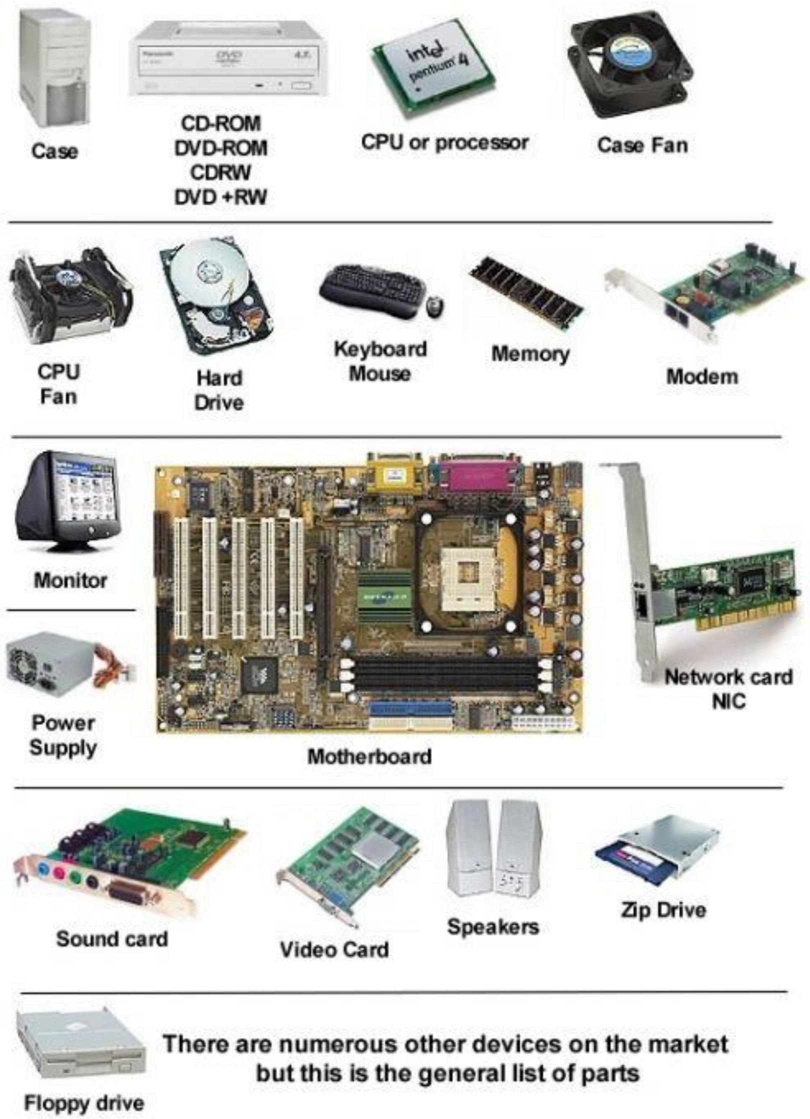
Lernziele II

- Sie wenden an:
  - I/O Scheduling-Algorithmen (FIFO, SSTF, SCAN, C-SCAN) auf eine gegebene Folge von Anfragen an eine HDD-Festplatte
  - Berechnung der Parität für RAID Level 5
  - Ausfüllen der File Allocation Table für einen gegebenen Zustand des Sekundärspeichers
  - Füllen des Sekundärspeichers anhand einer gegebenen File Allocation Table

- Sie wägen Vor- und Nachteile ab:
  - der Methoden der File Allocation

Input/Output System

Operating System Design and I/O



Operating System Design and I/O

Efficiency Problems

- I/O (usually) cannot keep up with processor speed
- Use of multiprogramming allows for some processes to be waiting on I/O while another process executes
- Most I/O devices extremely slow compared to main memory
- Swapping is used to bring in additional Ready processes (requires I/O operations)

Generality

- Desirable to handle all I/O devices in a uniform manner, i.e., provide good abstraction to application programmer
- Hide most of details of device I/O in lower-level routines
  - ➢ Processes and upper levels see devices in general terms, e.g., read, write, open, close, lock, unlock

➢Conflicting goals motivate focus on API design

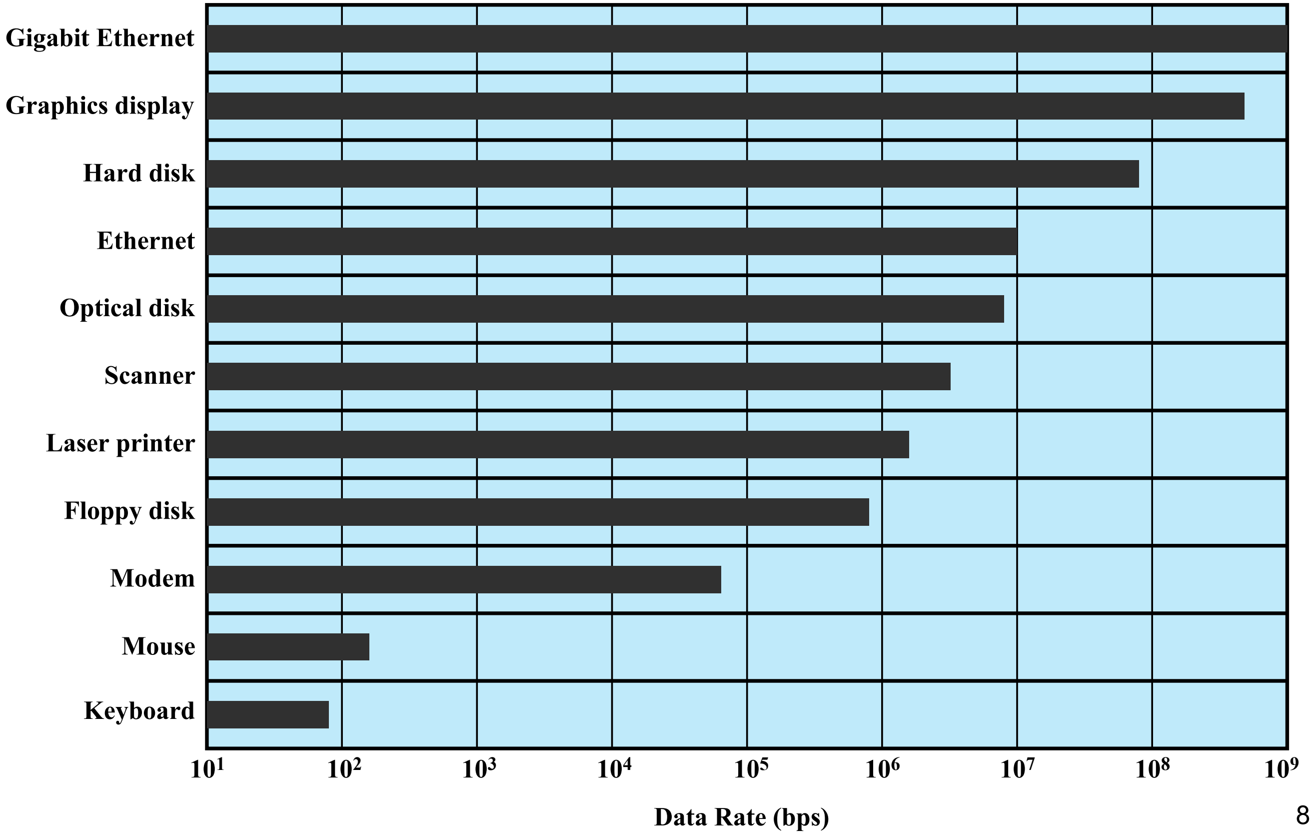
Types of I/O Devices

Wide variety of I/O devices

- Human readable, e.g., display, keyboard, mouse
- Machine readable, e.g., disk and tape drives, sensors, controllers, actuators
- Communication, e.g., digital line drivers, modems

Data rate

- Application
  (software support, priority)
- Complexity of control
- Unit of transfer
  (stream, blocks, characters)
- Data representation
- Encoding schemes
- Error conditions



Alternatives for I/O Organization

Device abstraction

- Character-based I/O
  - E.g. input devices like keyboard or mouse
- Block-based I/O
  - E.g. data storage

➢ Not necessarily related to implementation, e.g. USB

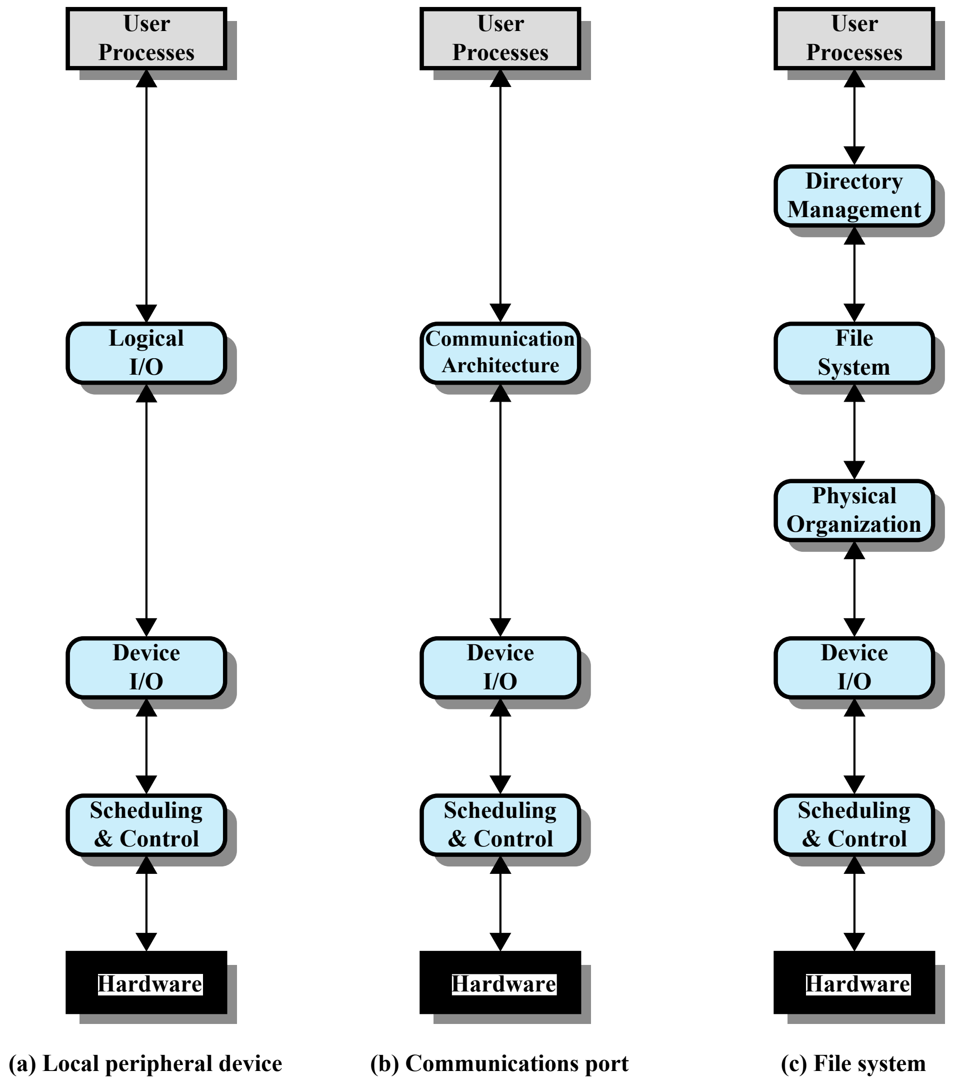
Communication endpoint (socket) abstraction

- Used for networking

➢ Second part of this lecture

File abstraction

- Structured, persistent storage
- Sometimes with additional semantics, e.g., locking, transaction support, etc.



I/O Related Programming Techniques

Programmed I/O

- Process is busy-waiting for the operation to complete

```
while (*IO_STATUS_ADDR != IO_DONE){}
```

Interrupt-driven I/O

- I/O command is issued, requesting process is blocked
- Processor continues executing instructions
- I/O module sends an interrupt when done

Direct Memory Access (DMA)

- DMA module controls exchange of data between main memory and the I/O device
- Processor interrupted only after entire block has been transferred

Comparison of I/O Techniques

Programmed I/O
- Only when there's no alternative, e.g., timing with very high accuracy

Interrupt-driven I/O
- Event-based programming, e.g., user input

Direct memory access
- Data transfer, e.g. disk I/O, graphics operations, network packet processing

	No Interrupts	**Use of Interrupts**
I/O-to-memory transfer through processor	Programmed I/O	Interrupt-driven I/O
Direct I/O-to-memory transfer		Direct memory access (DMA)

Direct Memory Access

Moving data between main memory and peripherals is a simple operation, but keeps CPU busy

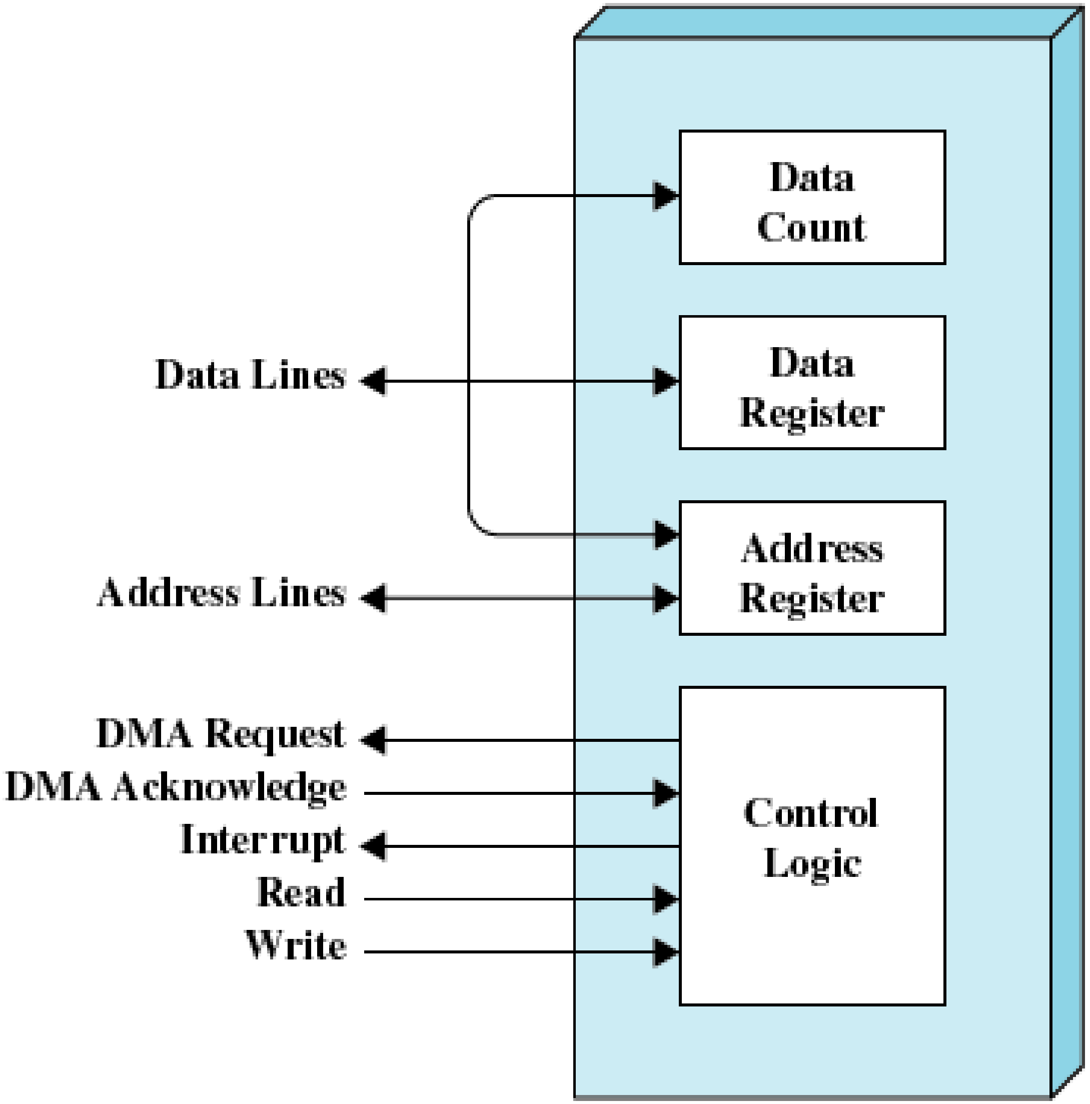
➢ Delegate I/O operation to extra hardware: DMA module

DMA module transfers data directly to or from memory

- "For-loop in hardware"

➢ Continuous memory regions

When complete, DMA module sends interrupt signal to CPU



DMA Configurations

a) Single-bus, detached DMA

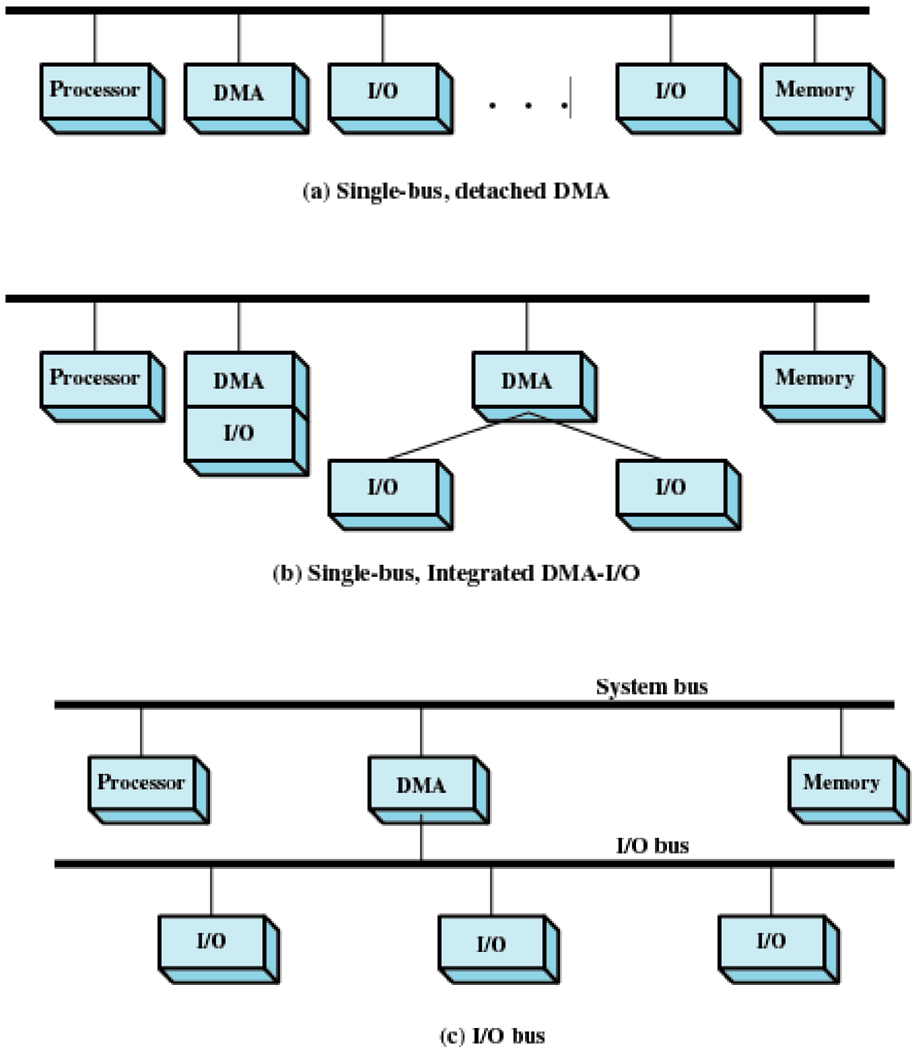
- ➢ Simple, but inefficient
- ➢ Requires multiple I/O requests to device

b) Single-bus, integrated DMA-I/O

- ➢ Efficient, but expensive
- ➢ One controller per device (group)

c) I/O bus

- ➢ Efficient and less expensive
- ➢ Separate bus, one controller



I/O Buffering

Main memory used to temporarily store data

- Mitigates differences in data processing speeds
  - Processes must wait for I/O to complete before proceeding
- Manage pages that must remain in main memory during I/O
  - Buffer must be accessible to low-level drivers and hardware

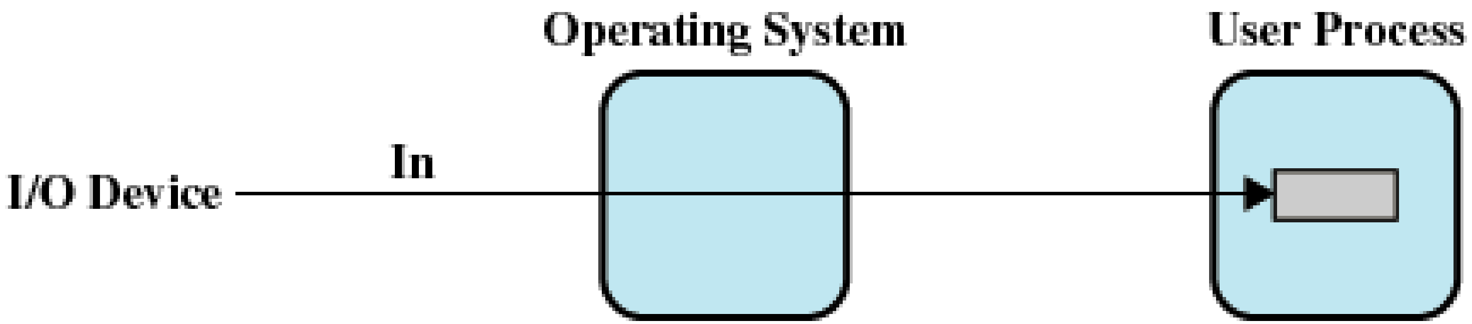
Approaches (with different buffering strategies)

- Block-oriented
  - Information is stored in fixed sized blocks
  - Transfers are made one block at a time
  - Used for disks and tapes

- Stream-oriented (stream of characters)
  - Transfer information as a stream of bytes
  - Used for terminals, printers, communication ports, mouse and other pointing devices, and most other devices that are not secondary storage

I/O Buffering Implementations (I)

No buffering

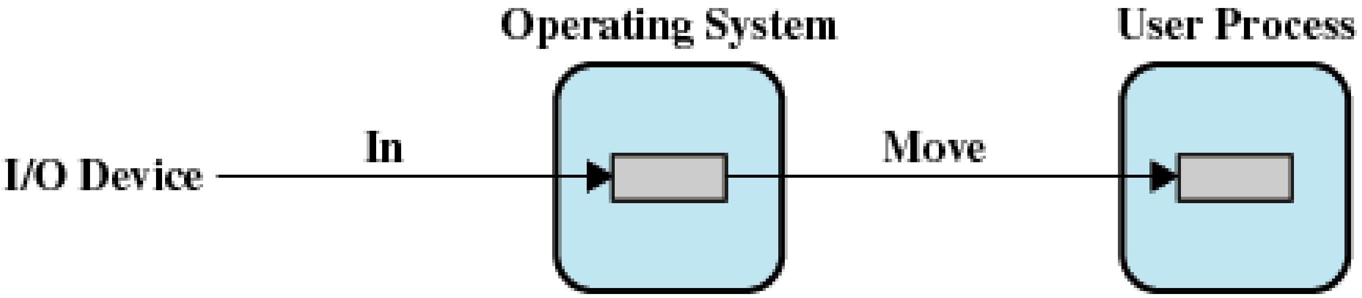


(a) No buffering

I/O Buffering Implementations (II)

Single buffering

- Block-oriented: User process can process one *fixed-sized* block of data while next block is read in
- Stream-oriented: Process one *variable-sized and delimited* line at time

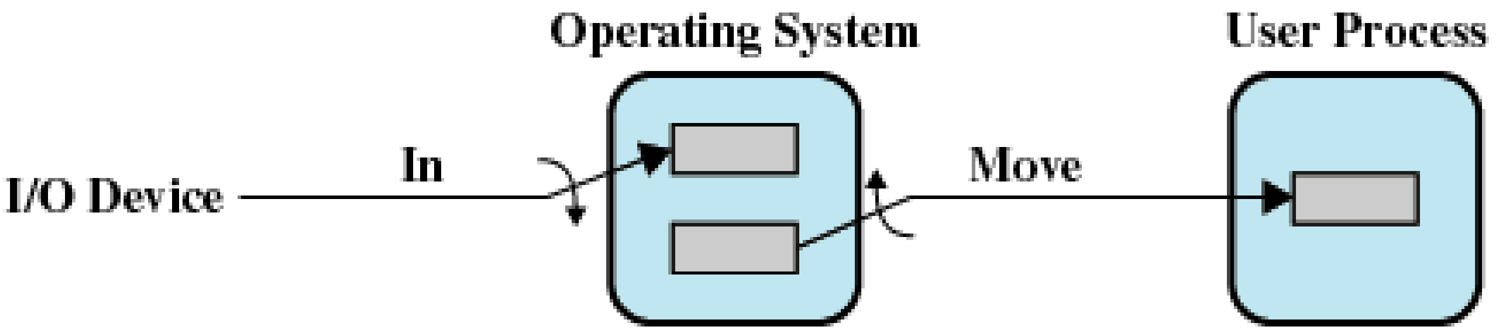


(b) Single buffering

I/O Buffering Implementations (III)

Double buffering

- Process can transfer data to or from one buffer while OS empties or fills other buffer



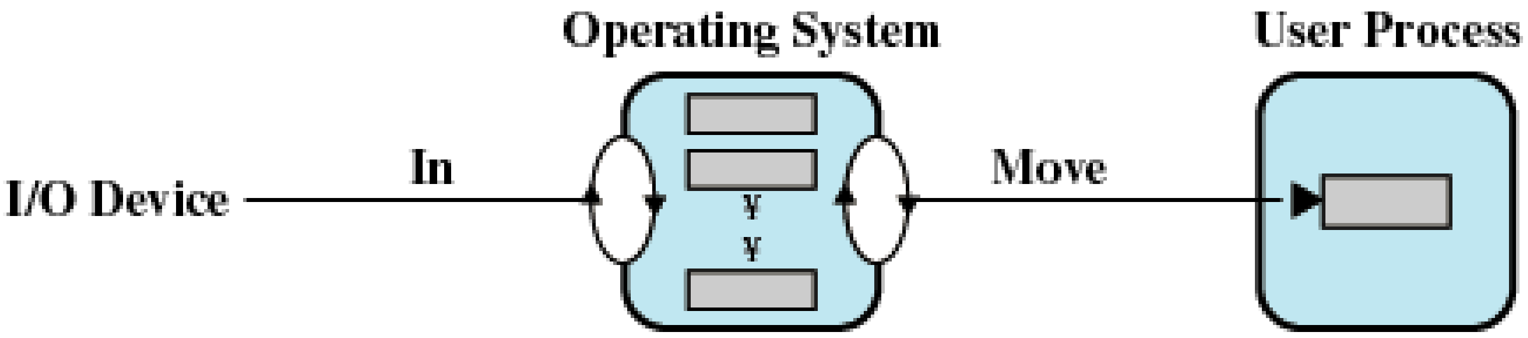
(c) Double buffering

I/O Buffering Implementations (IV)

Circular/ring buffering

- Each individual buffer is one unit in circular buffer

➢ Used when I/O operation must keep up with process

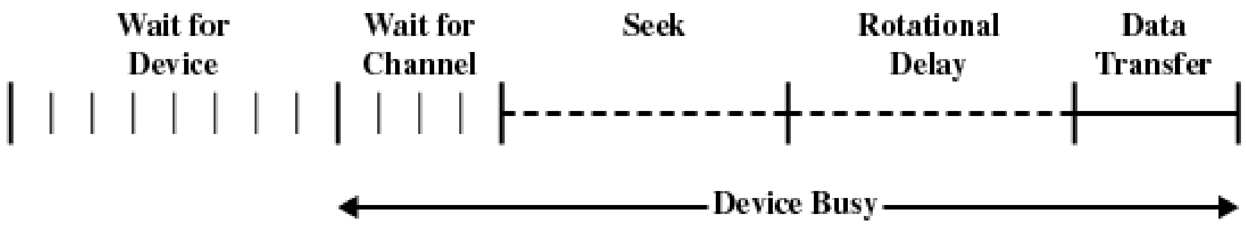


(d) Circular buffering

I/O Scheduling

For a single resource there will be a number of I/O requests
- From *one or several* processes
- Some devices keep internal state, so ordering of I/O requests matters

Example: Disk access



- Access time
  - Sum of seek time and rotational delay
    - Time it takes to get in position to read or write
  - ➢ Seek time is the reason for differences in performance
- Data transfer occurs as the sector moves under the head

➢ Reorder I/O requests according to current state of disk

Disk Scheduling Algorithms

Name	Description	Remarks
Selection according to requestor		
RSS	Random scheduling	For analysis and simulation
FIFO	First in first out	Fairest of them all
PRI	Priority by process	Control outside of disk queue management
LIFO	Last in first out	Maximize locality and resource utilization
Selection according to requested item		
SSTF	Shortest service time first	High utilization, small queues
SCAN	Back and forth over disk	Better service distribution
C-SCAN	One way with fast return	Lower service variability
N-step-SCAN	SCAN of *N* records at a time	Service guarantee
FSCAN	N-step-SCAN with *N* = queue size at beginning of SCAN cycle	Load sensitive

Disk I/O Scheduling Policies

Example

- Disk with 200 tracks
- Disk request queue has random requests
- Order of requests

55, 58, 39, 18, 90, 160, 150, 38, 184

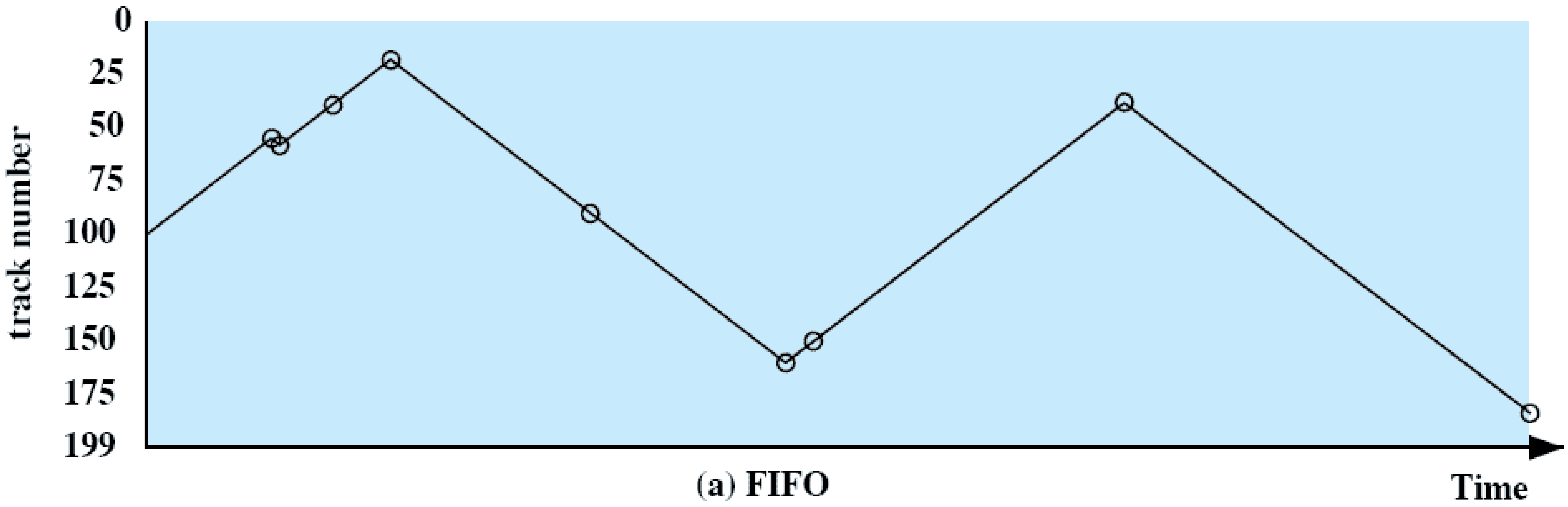
First-In, First-Out (FIFO)

Requests in sequential order

Fair to all requests

Approximates random scheduling in performance if there are many requests competing for the disk

Requests: 55, 58, 39, 18, 90, 160, 150, 38, 184



(a) FIFO

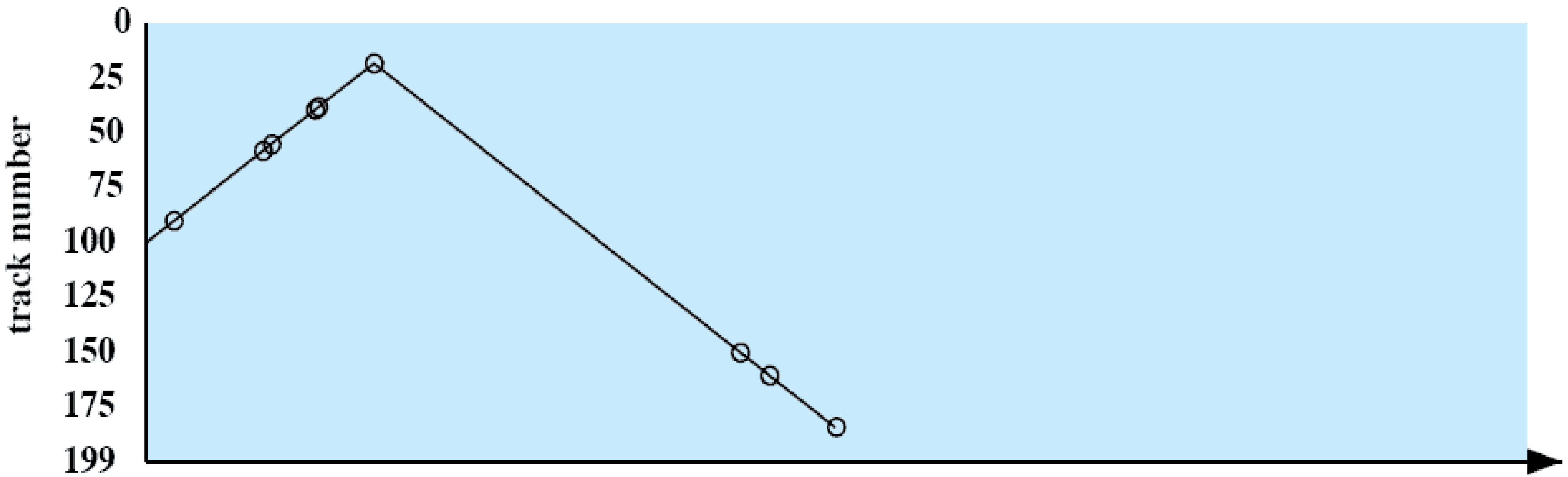
Shortest Service Time First (SSTF)

Select the disk I/O request that requires the least movement of the disk arm from its current position

Always choose the minimum seek time

Requests: 55, 58, 39, 18, 90, 160, 150, 38, 184

Service order: 90, 58, 55, 39, 38, 18, 150, 160, 184



(b) SSTF

SCAN

Also known as the elevator algorithm

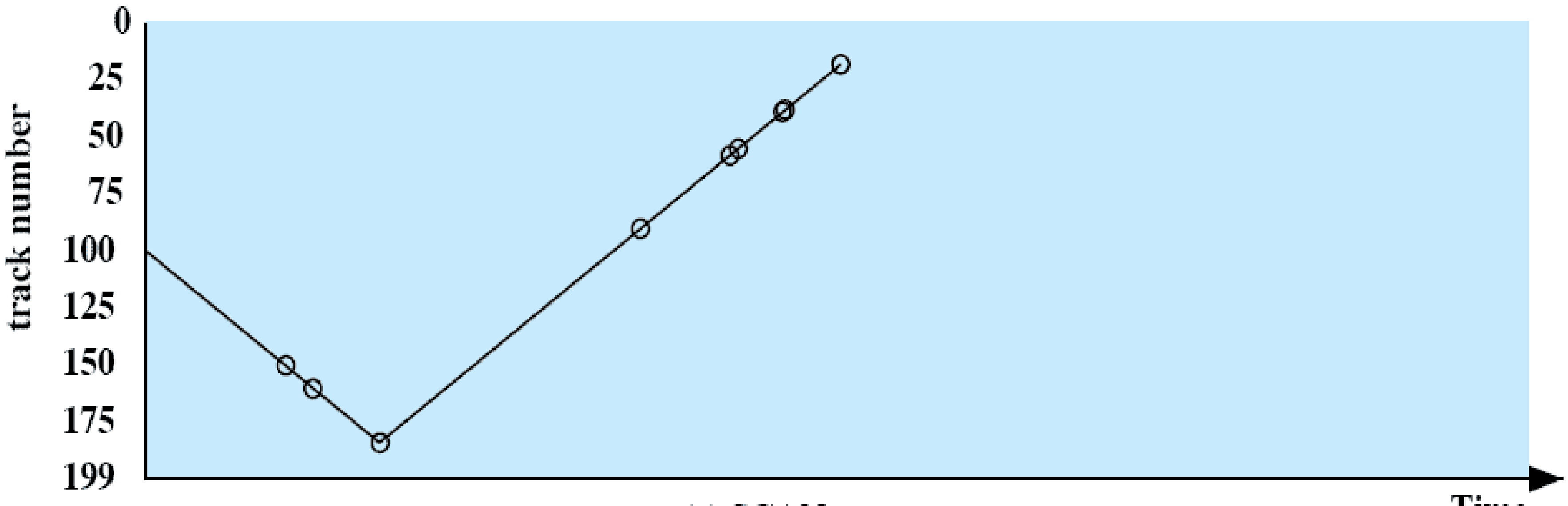
Arm moves in one direction only

- satisfies all outstanding requests until it reaches the last track in that direction then the direction is reversed

Favors jobs whose requests are for tracks nearest to both innermost and outermost tracks

Requests: 55, 58, 39, 18, 90, 160, 150, 38, 184

Service order: 150, 160, 184, 90, 58, 55, 39, 38, 18



(c) SCAN

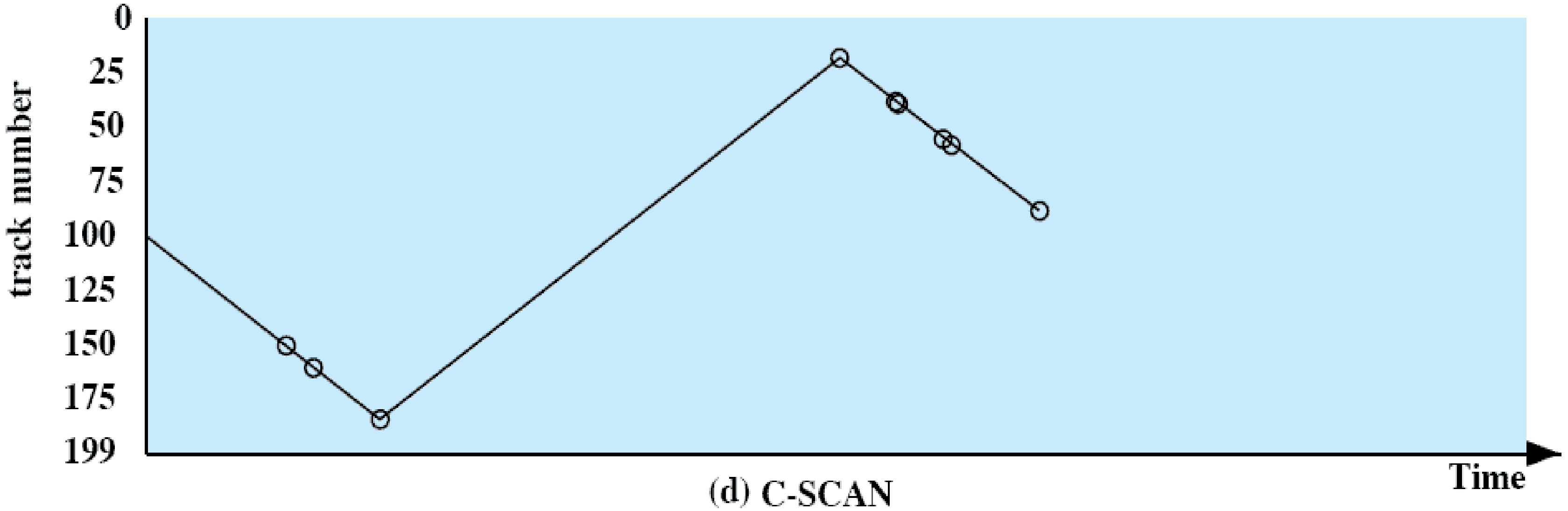
C-SCAN (Circular SCAN)

Restricts scanning to one direction only

When the last track has been visited in one direction, the arm is returned to the opposite end of the disk and the scan begins again

Requests: 55, 58, 39, 18, 90, 160, 150, 38, 184

Service order: 150, 160, 184, 18, 38, 39, 55, 58



(d) C-SCAN

Comparison of Disk Scheduling Algorithms

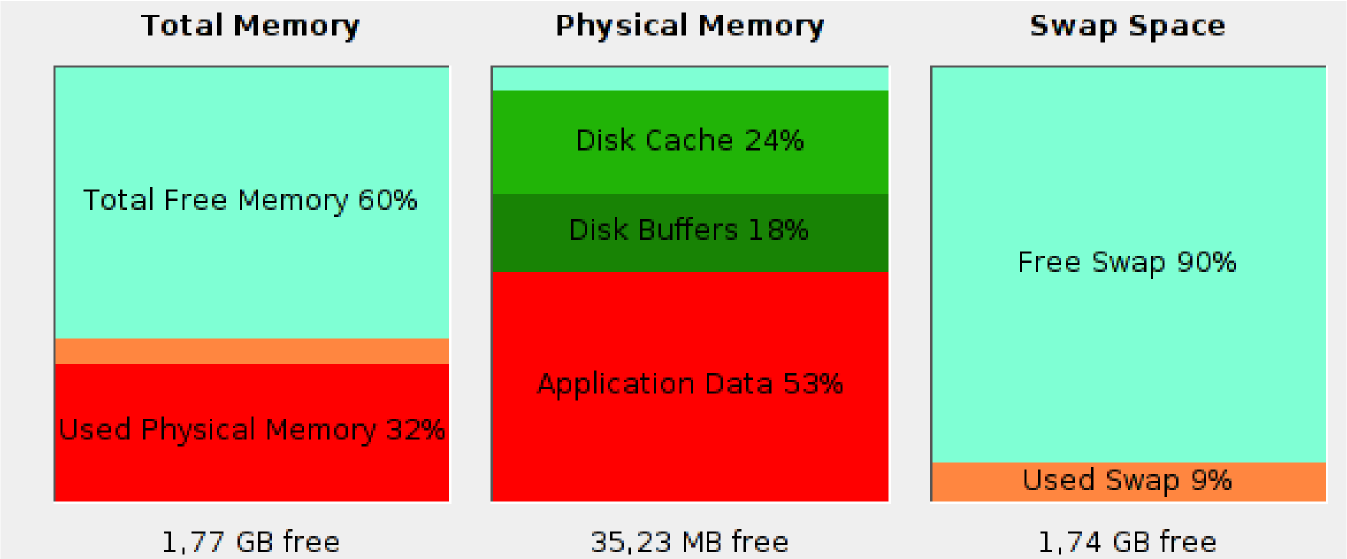
(a) FIFO (starting at track 100)		(b) SSTF (starting at track 100)		(c) SCAN (starting at track 100, in the direction of increasing track number)		(d) C-SCAN (starting at track 100, in the direction of increasing track number)	
Next track accessed	**Number of tracks traversed**	**Next track accessed**	**Number of tracks traversed**	**Next track accessed**	**Number of tracks traversed**	**Next track accessed**	**Number of tracks traversed**
55	45	90	10	150	50	150	50
58	3	58	32	160	10	160	10
39	19	55	3	184	24	184	24
18	21	39	16	90	94	18	166
90	72	38	1	58	32	38	20
160	70	18	20	55	3	39	1
150	10	150	132	39	16	55	16
38	112	160	10	38	1	58	3
184	146	184	24	18	20	90	32
Average seek length	55.3	**Average seek length**	27.5	**Average seek length**	27.8	**Average seek length**	35.8

Disk Cache

Main memory buffer for disk sectors

- Contains copy of subset of sectors on disk

➢ Speeds up I/O requests to these sectors



Policies:

- Least Recently Used
  - Block longest in cache with no reference to it is replaced
- Least Frequently Used
  - Block with fewest references is replaced
  - ➢ Reference count is misleading for bursty access patterns

RAID
– Overview

- RAID 5 = **Redundant Array of Independent Disks**, Level 5
- Combines striping with distributed parity
- Minimum 3 disks required
- Data and parity information are distributed in blocks
- Allows fault tolerance in case of one disk failure
- Good compromise between performance, fault tolerance, and storage efficiency

RAID 5
– Characteristics

- Read: Fast (parallel access possible)
- Write: Slower (parity computation required)
- Parity distribution: Evenly across all disks
- Usable storage: $(n-1) / n$ of total capacity (for n disks)
- Recovery: Possible if one disk fails (**not two or more!**)

RAID 5 – Parity Calculation

Uses exclusive OR (XOR) for parity calculation

Example:

$$\begin{aligned} &Data\ B\text{: } 1010 \\ &Data\ A\text{: } 1101 \\ &Parity\text{: } A \oplus B = 0111 \end{aligned}$$

If one disk fails, its data can be reconstructed:

$$\begin{aligned} A &= B \oplus Parity \\ &= 1010 \oplus 0111 \\ &= 1101 \end{aligned}$$

RAID 5
– Example with 3 Data Disks + 1 Parity Disk

Binary Data (4-bit):

$$D1\colon\ 1101$$
$$D2\colon\ 1010$$
$$D3\colon\ 0110$$

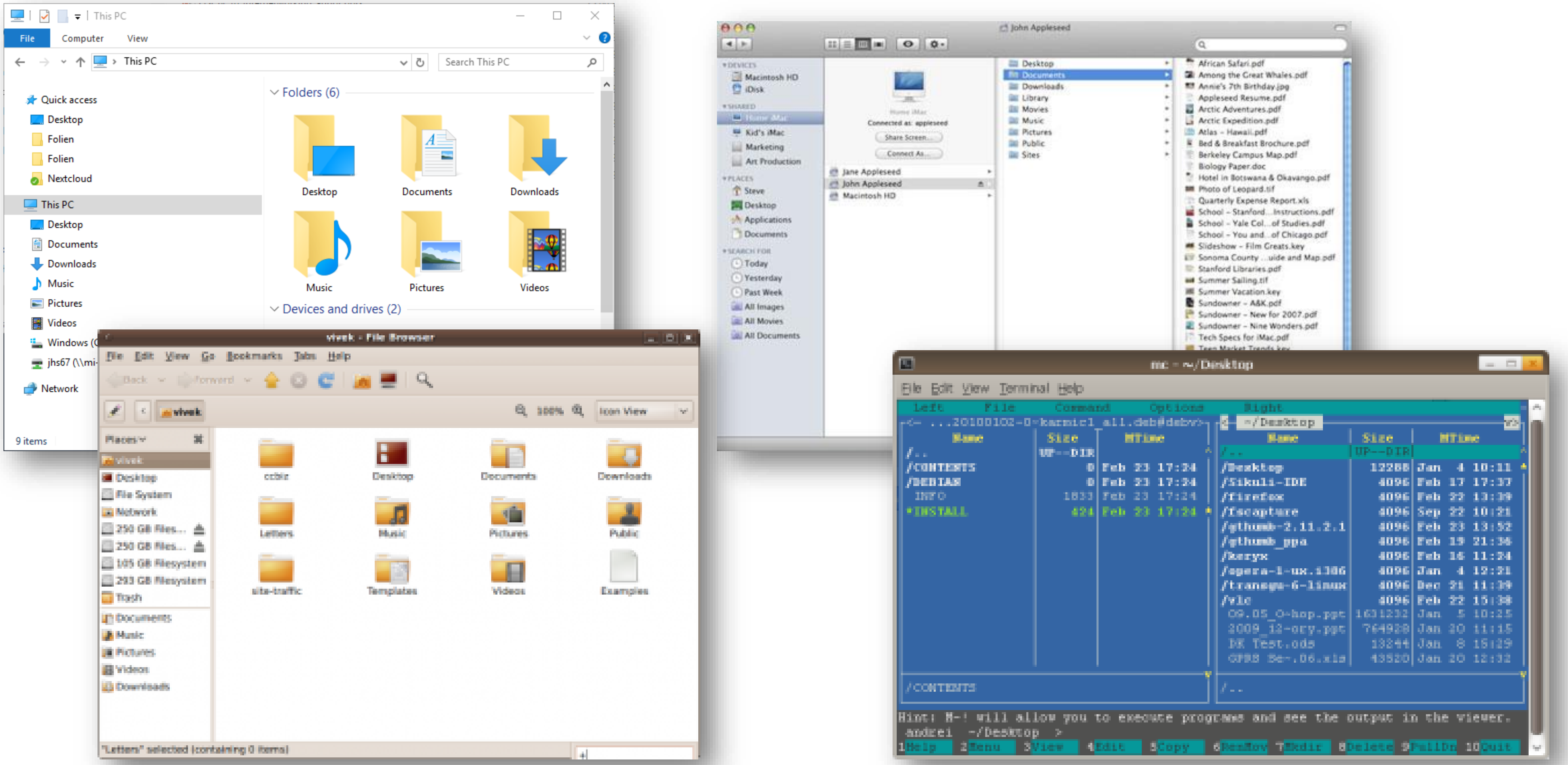
Parity calculation (bitwise XOR):

$$1101 \oplus 1010 = 0111$$
$$0111 \oplus 0110 = \mathbf{0001}$$

→ Parity block = 0001

File System

Overview



This PC
File
Computer
View
This PC
Search This PC
Quick access
Desktop
Folien
Folien
Nextcloud
This PC
Desktop
Documents
Downloads
Music
Pictures
Videos
Network
9 items
Folders (6)
Desktop
Documents
Downloads
Music
Pictures
Videos
Devices and drives (2)
John Appleseed
DEVICES
Macintosh HD
iDisk
SHARED
Kid's iMac
Marketing
Art Production
PLACES
Steve
Desktop
Applications
Documents
SEARCH FOR
Today
Yesterday
Past Week
All Images
All Movies
All Documents
Jane Appleseed
John Appleseed
Macintosh HD
Desktop
Documents
Downloads
Library
Movies
Music
Pictures
Public
Sites
African Safari.pdf
Among the Great Whales.pdf
Annie's 7th Birthday.jpg
Appleseed Resume.pdf
Arctic Adventures.pdf
Arctic Expedition.pdf
Atlas - Hawaii.pdf
Bed & Breakfast Brochure.pdf
Berkeley Campus Map.pdf
Biology Paper.doc
Hotel in Botswana & Okavango.pdf
Photo of Leopard.tif
Quarterly Expense Report.xls
Slideshow - Film Greats.key
Stanford Libraries.pdf
Summer Sailing.tif
Summer Vacation.key
Sundowner - A&K.pdf
Sundowner - New for 2007.pdf
Sundowner - Nine Wonders.pdf
Tech Specs for iMac.pdf
vivek - File Browser
File Edit View Go Bookmarks Tabs Help
Back
Forward
vivek
100%
Icon View
Places
Desktop
File System
Network
Trash
Documents
Music
Pictures
Videos
Downloads
Desktop
Documents
Downloads
Letters
Music
Pictures
Public
Templates
Videos
Examples
"Letters" selected (containing 0 items)
mc - ~/Desktop
File Edit View Terminal Help
Left File Command Options Right
Name Size MTime
/..
/CONTENTS
/DEBIAN
INSTALL
/Desktop
/firefox
/vlc
/CONTENTS
Hint: M-! will allow you to execute programs and see the output in the viewer.
1Help 2Menu 3View 4Edit 5Copy 6RenMov 7Mkdir 8Delete 9PullDn 10Quit

Goals

- Meet data management needs and requirements of user
- Guarantee that data in file is valid (over time)
- Optimize performance
- Provide I/O support for variety of storage device types
- Minimize or eliminate the potential for lost or destroyed data (redundancy)
- Provide a standardized set of I/O interface routines
- Provide I/O support for multiple users
  - Concurrency, access control, etc.

Types of File Systems

Disk File Systems

- Windows: FAT, FAT16, FAT32, NTFS
- Linux: ext, ext2, ext3
- UNIX: UFS, …
- MAC OS X: HFS, HFS+

Distributed File Systems

- NFS, AFS, SMB

Special Purpose File Systems

Properties & Typical Operations

Properties

- Long-term existence
- Sharable between processes
- Structure (internal /organizational)

Typical File Operations

• Create	Create new file
• Delete	Delete existing file
• Open	Open new/existing file
• Close	Close open file
• Read	Read data from open file
• Write	Write data to open file

File Directories

Contains information about files

- Attributes, e.g., read/write/executable bits, access time
- Ownership, e.g., user/group or Access Control List (ACL)
- Location with regard to logical structure of medium

Directory itself may be implemented as file owned by operating system

Provides mapping between file names and files themselves

- “inodes” in Unix

➢ One file can have multiple names (“hard links”)

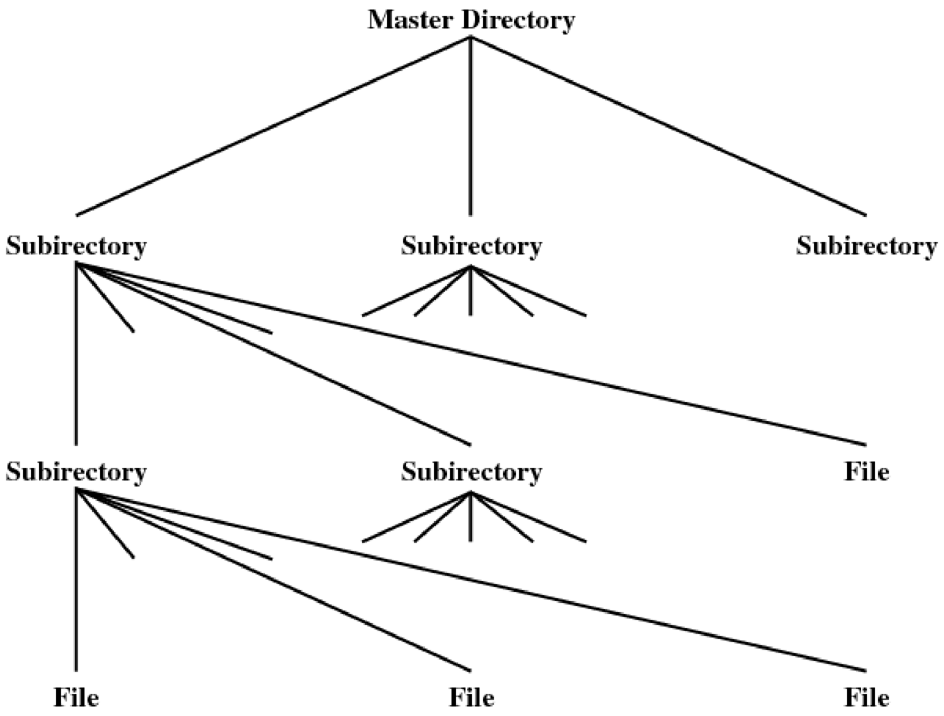
Structure

- List of entries, one for each file
- Sequential file with name of file serving as key
- Initially no support for organizing files (except for naming)
  - Forces user to be careful not to use the same name for two different files

Multi-Level Directories

Hierarchical / Tree-Structure

- Master directory with user directories underneath it
- Each user directory may have subdirectories and files as entries
- Some operating systems use multiple trees with own identifiers, e.g., drive letters (A:, C:)



Hierarchical/Tree-Structured Directory

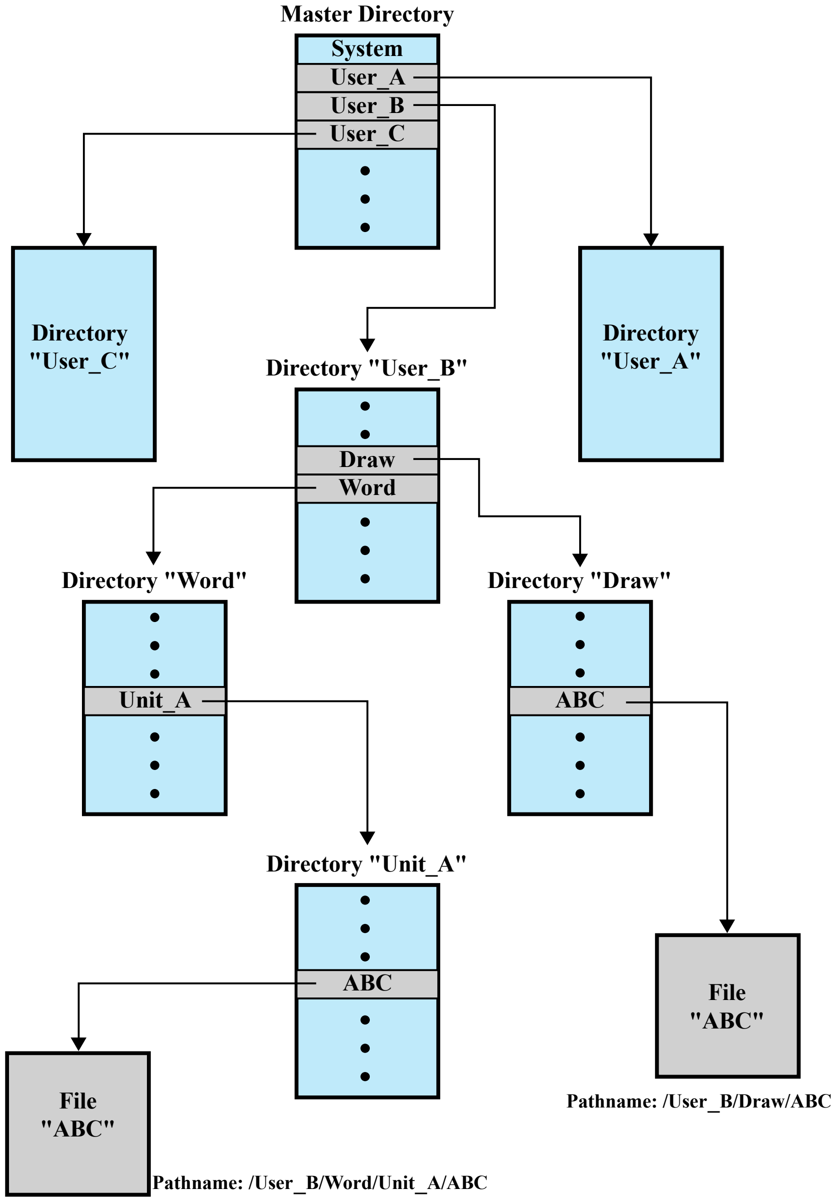
Files are located by following path from root (master) directory down various branches

- *Pathname* of file

Supports several files with same file name as long as path names differ

Per-process current directory is working directory

Files are referenced relative to current working directory (CWD)



Access Rights

None
- User may not know of existence of file
- User is not allowed to read user directory that includes file

Knowledge
- User can only determine that file exists and who its owner is

Execution
- User can load and execute program but cannot copy it

Reading
- User can read file for any purpose, including copying and execution

Appending
- User can add data to file but cannot modify or delete any of its contents

Updating
- User can modify, delete and add to file’s data
- Includes creating file, rewriting it and removing all or part of its data

Changing protection
- User can change access rights granted to other users

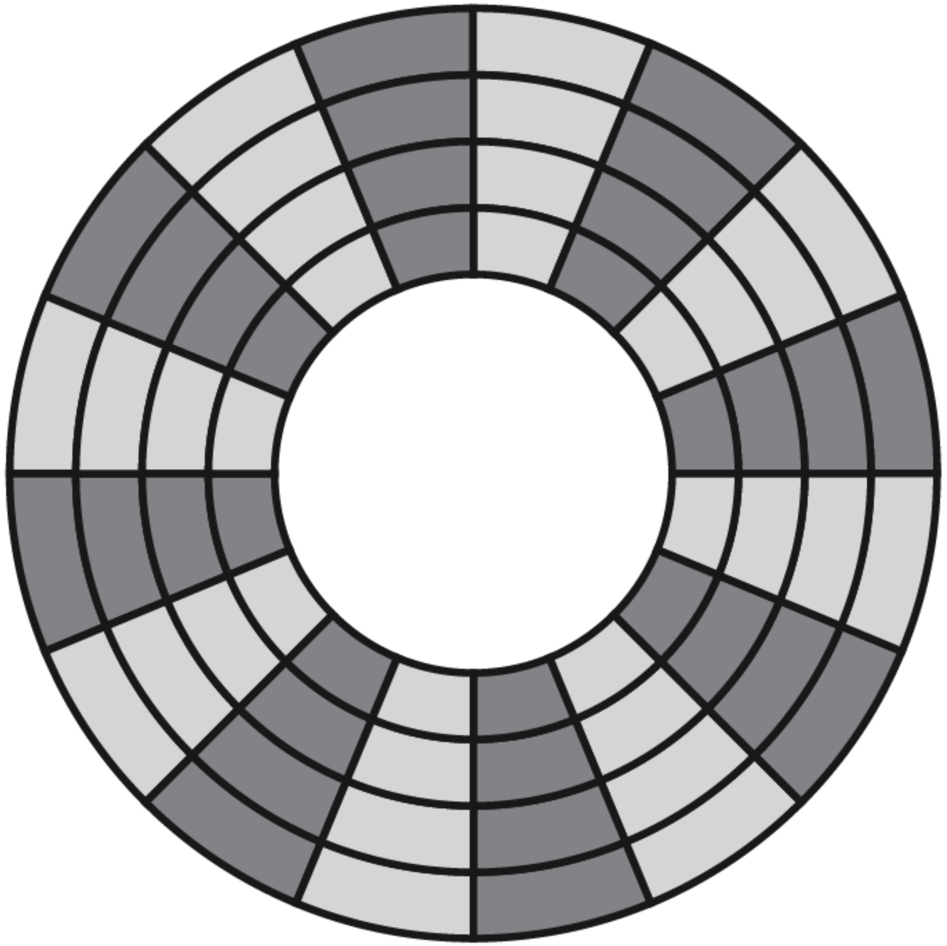
Deletion
- User can delete a file

Owner
- All rights previously listed
- Grant rights to others using classes of users:
  - Specific user
  - User groups
  - Everybody

- Complex access policies implemented with Access Control Lists (ACLs)
- ➢ Watch out for semantic differences between files and directories!

Secondary Storage Management

Secondary storage space must be allocated to files
Must keep track of space available for allocation



File Allocation

On secondary storage, a file consists of a collection of blocks

The operating system or file management system is responsible for allocating blocks to files

The approach taken for file allocation may influence the approach taken for free space management

Space is allocated to a file as one or more portions (contiguous set of allocated blocks)

File allocation table (FAT)

- data structure used to keep track of the portions assigned to a file

Secondary Storage Management - File Allocation Methods

Contiguous allocation

- Single set of blocks is allocated to a file at time of creation
- Single entry in file allocation table (starting block, length of file)
- ➢ Incurs fragmentation; changing size of a file is expensive

Chained allocation

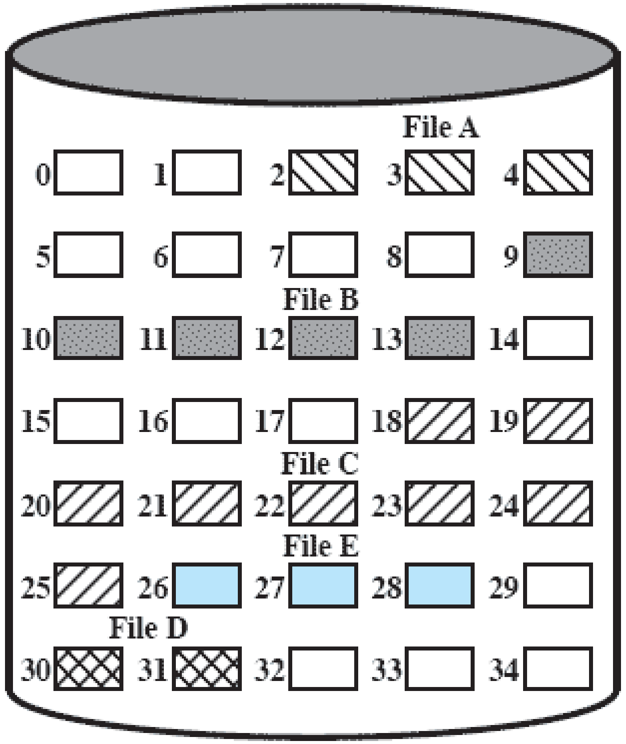
- Allocation on basis of individual block
- Each block contains a pointer to next block in chain
- Single entry in file allocation table (starting block, length of file)
- ➢ Seeking within file (random access) is expensive

Indexed allocation

- File allocation table contains a separate one-level index for each file
- The index has one entry for each portion allocated to file
- The file allocation table contains block number for index
- ➢ Avoids problems mentioned above, incurs some storage overhead

Methods of File Allocation
- Contiguous File Allocation

- A single contiguous set of blocks is allocated to a file at the time of file creation
- Preallocation strategy using variable-size portions
- Is the best from the point of view of the individual sequential file



File Allocation Table

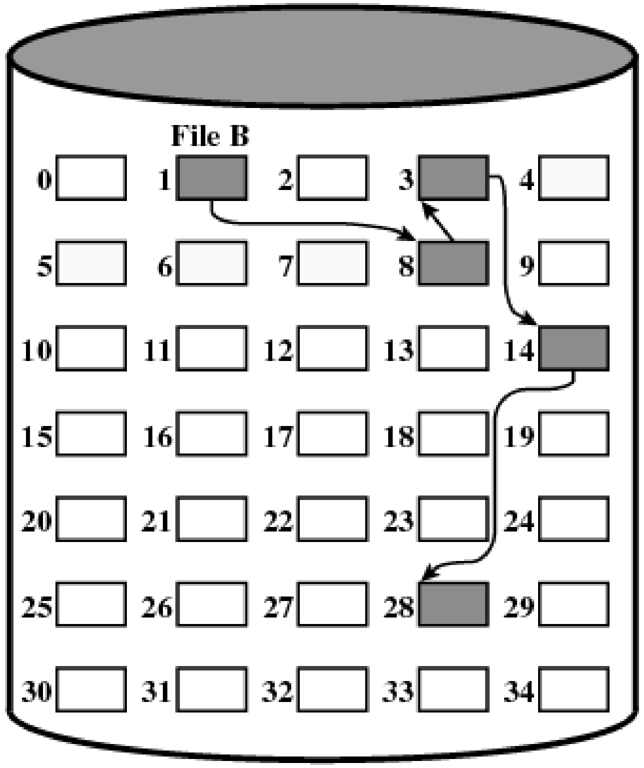
File Name	Start Block	Length
File A	2	3
File B	9	5
File C	18	8
File D	30	2
File E	26	3

➢External fragmentation on disk

Methods of File Allocation
- Chained Allocation

- Allocation is on an individual block basis
- Each block contains a pointer to the next block in the chain
- The file allocation table needs just a single entry for each file
- No external fragmentation to worry about
- Best for sequential files

➢Low random access performance

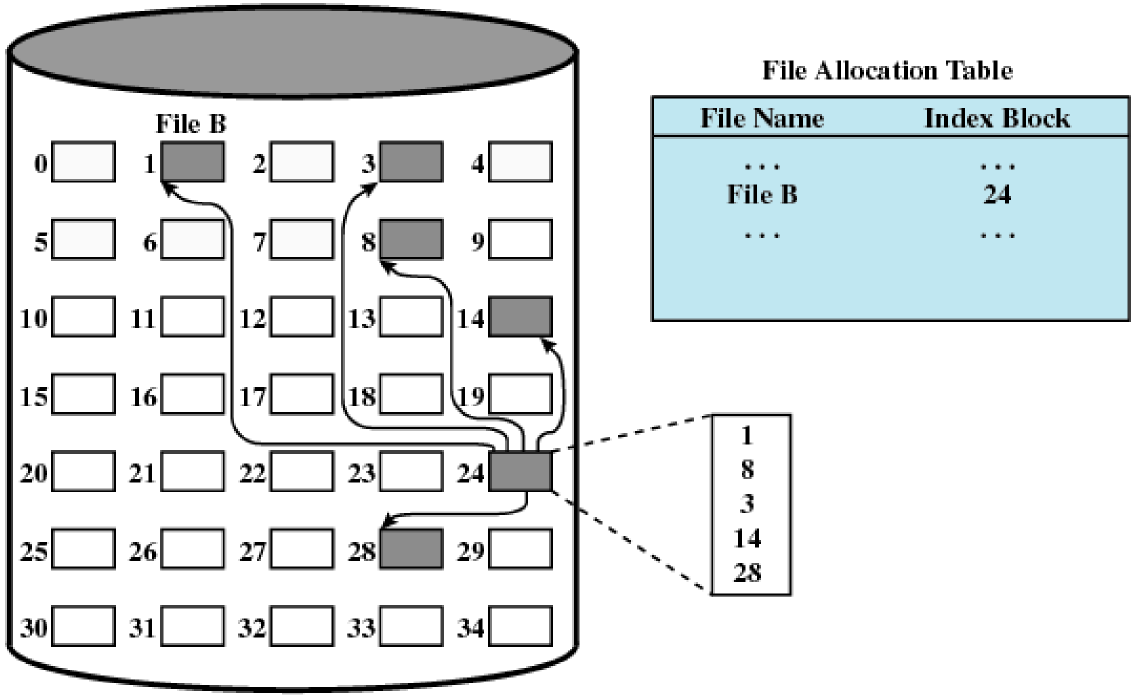


File Allocation Table

File Name	Start Block	Length
...	...	...
File B	1	5
...	...	...

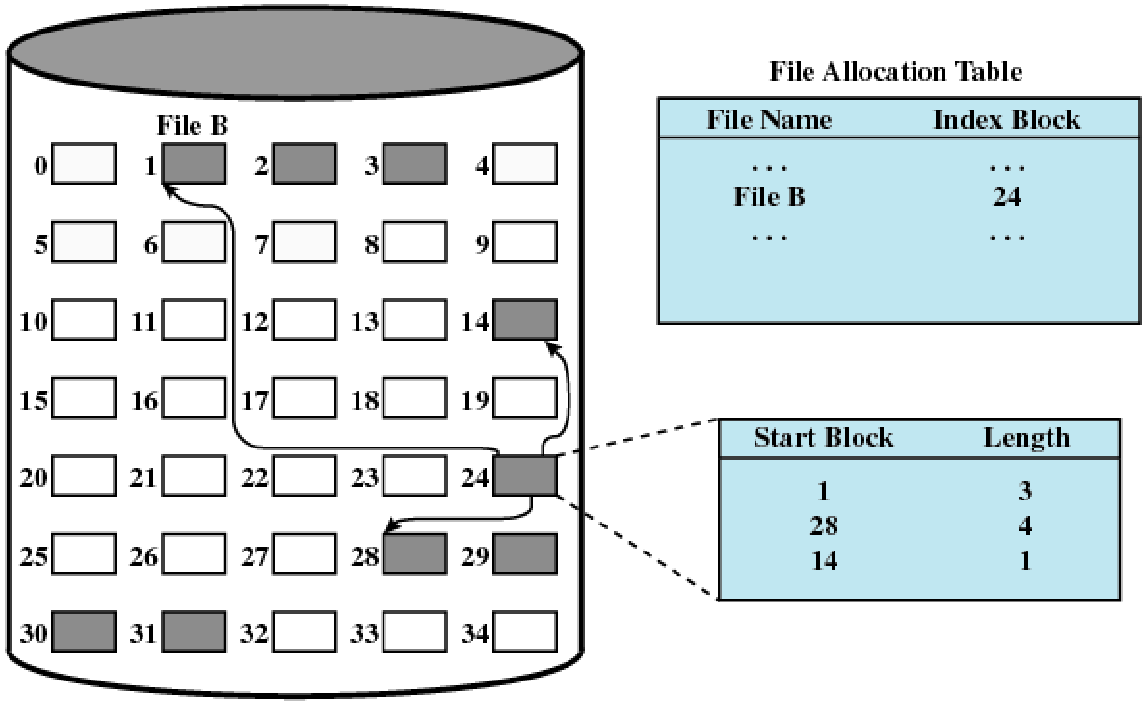
Methods of File Allocation

Indexed Allocation with Block Portions



- ➢Indexing overhead

Index Allocation with Variable-Length Portions



- ➢Good compromise

Inodes

All types of UNIX files are administered by the OS by means of **inodes**

An inode (index node) is a control structure that contains the key information needed by the operating system for a particular file

Several file names may be associated with a single inode

- an active inode is associated with exactly one file
- each file is controlled by exactly one inode

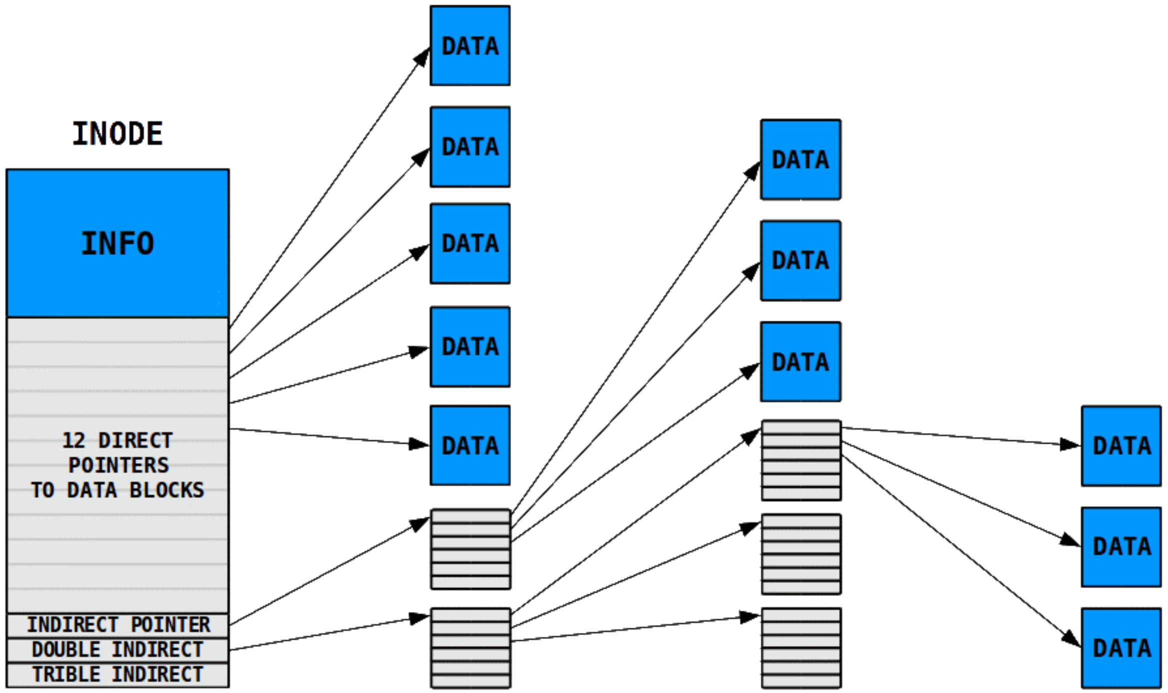
File Mode	16-bit flag that stores access and execution permissions associated with the file. 12-14 File type (regular, directory, character or block special, FIFO pipe 9-11 Execution flags 8 Owner read permission 7 Owner write permission 6 Owner execute permission 5 Group read permission 4 Group write permission 3 Group execute permission 2 Other read permission 1 Other write permission 0 Other execute permission
Link Count	Number of directory references to this inode
Owner ID	Individual owner of file
Group ID	Group owner associated with this file
File Size	Number of bytes in file
File Addresses	39 bytes of address information
Last Accessed	Time of last file access
Last Modified	Time of last file modification
Inode Modified	Time of last inode modification

Ext2: Inodes

Basic concept of the Ext2 system (and of all Unix file systems) is the structure called **inode** (index node)

A file is represented by one inode

The length of files is variable but all inodes are of the same length (128 Byte)



Smaller files are more quickly accessed than larger files

Roadmap

1. Introduction and Motivation
2. Interrupts and System Calls
3. Processes
4. Scheduling
5. Memory
6. **I/O and File System**
7. Booting, Services, and Security