University of Nottingham
UK | CHINA | MALAYSIA

Operating Systems and Concurrency

Lecture 19: File System 2

University of Nottingham, Ningbo China
2024

Recap
Last Lecture

- **Traditional hard drives** are still used for most of secondary storage, **solid state disks** are becoming more popular
  - SSDs require **less disk scheduling**
- Traditional hard drives have **physical limitations** (seek times, rotational latency) ⇒ **disk scheduling** can help to minimise the impact of this
- This **influences file system design**

Goals for Today

Overview

- User view of file systems
  - System calls
  - Structures, organisation, file types
- Implementation view of file systems
  - Disk and partition layout
  - File tables
  - Free space management ...

File Systems
Different Views

- A **user view** that defines a file system in terms of the **abstractions** that the operating system provides
  - How files are **named**, what **operations** allowed on them
  - What **directories** looks like
  - **Interface** issue
- An **implementation view** that defines the file system in terms of its **low level implementation**
  - How files and directories are **stored**
  - How **disk space** is managed
  - How to make everything work efficiently

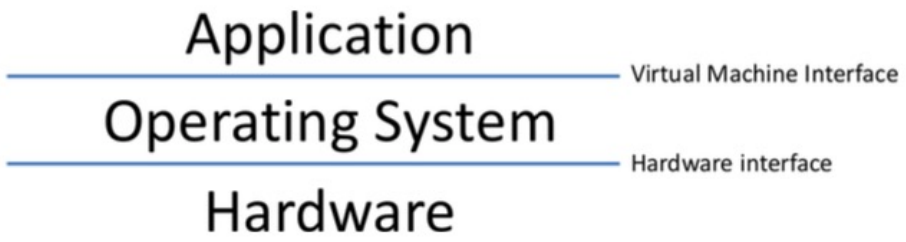
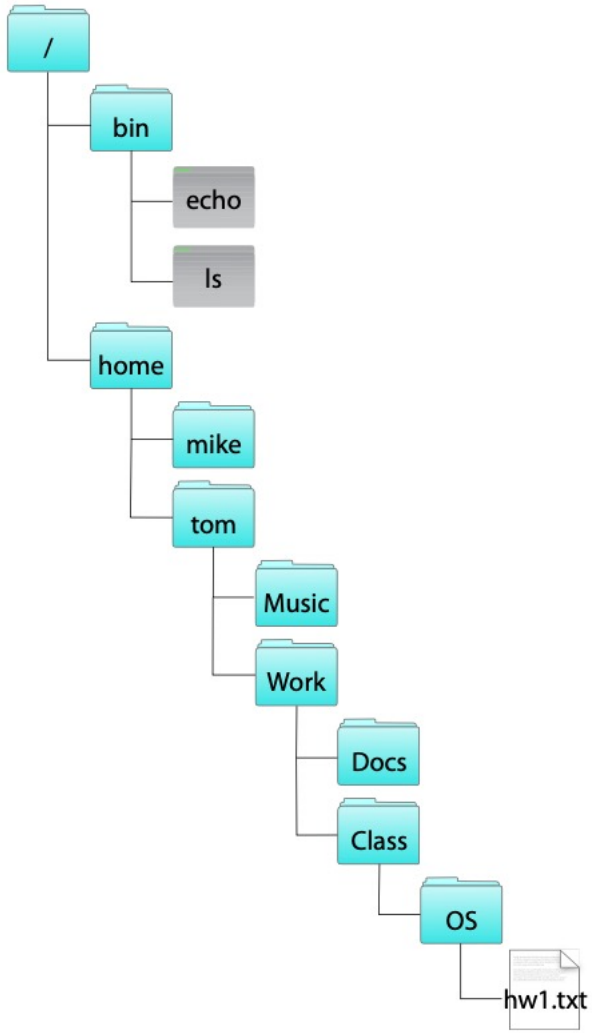


Figure: User vs. Implementation View

User View
Definition

- Important aspects of the **user view** include:
  - The **file abstraction** which **hides** away implementation details to the user (similar to processes and memory)
    - File **naming policies** (two-part file names)
    - User file **attributes** (metadata, e.g. size, protection, owner, protection, dates)
      - There are also **system attributes** for files (e.g. non-human readable file descriptors (similar to a PID), archive flag, temporary flag, etc.)
  - **Directory** structures and organisation
  - **System calls** to interact with the file system

Files
Types



- Many OSs support several types of file.
- Both Windows and Unix (including OS X) have regular **files** and **directories**:
  - **Regular files** contain user data in **ASCII** or **binary** (well defined) **format**
    - A file is a **named collection of data**
    - ASCII files **consist of lines of text**. (can be displayed, can be edited by text editor)
    - Binary files have **internal structure**
  - **Directories** group files together (but are files on an implementation level)
    - They provide a **mapping** of the logical file onto the physical location

System Calls
Types

- **System calls** enable a **user application** to **ask the operating system** to carry out an **action** on its behalf (in kernel mode)
- There are two different categories of **system calls** used in file system:
  - **Regular file manipulation**: open(), close(), read(), write(), delete(). . .
  - **Directory manipulation**: create(), delete(), readdir(), rename(), link(), unlink(), list(), update()

Directories
Directory structure

- **Directories** are **special files** that **group files** together and of which the **structure is defined** by the **file system**
  - A bit is set to indicate that they are directories!

- Different directory structures have been used over the years
  - **Single level**: all files in the same directory (reborn in consumer electronics)
  - **Two or multiple level directories** (hierarchical): tree structures
    - Absolute path name: from the root of the file system
    - Relative path name: the current working directory is used as the starting point
  - **Directed acyclic graph (DAG)**: allows files to be shared (i.e. **links** to files or sub-directories) but **cycles are forbidden**

Directories
Directory structure

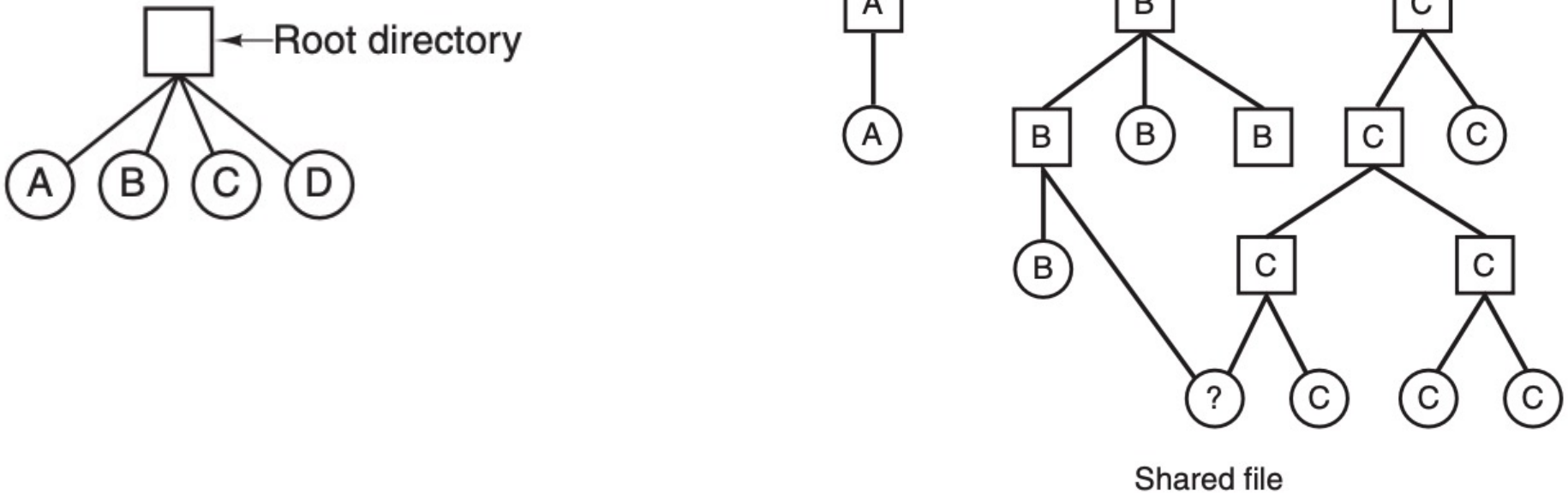
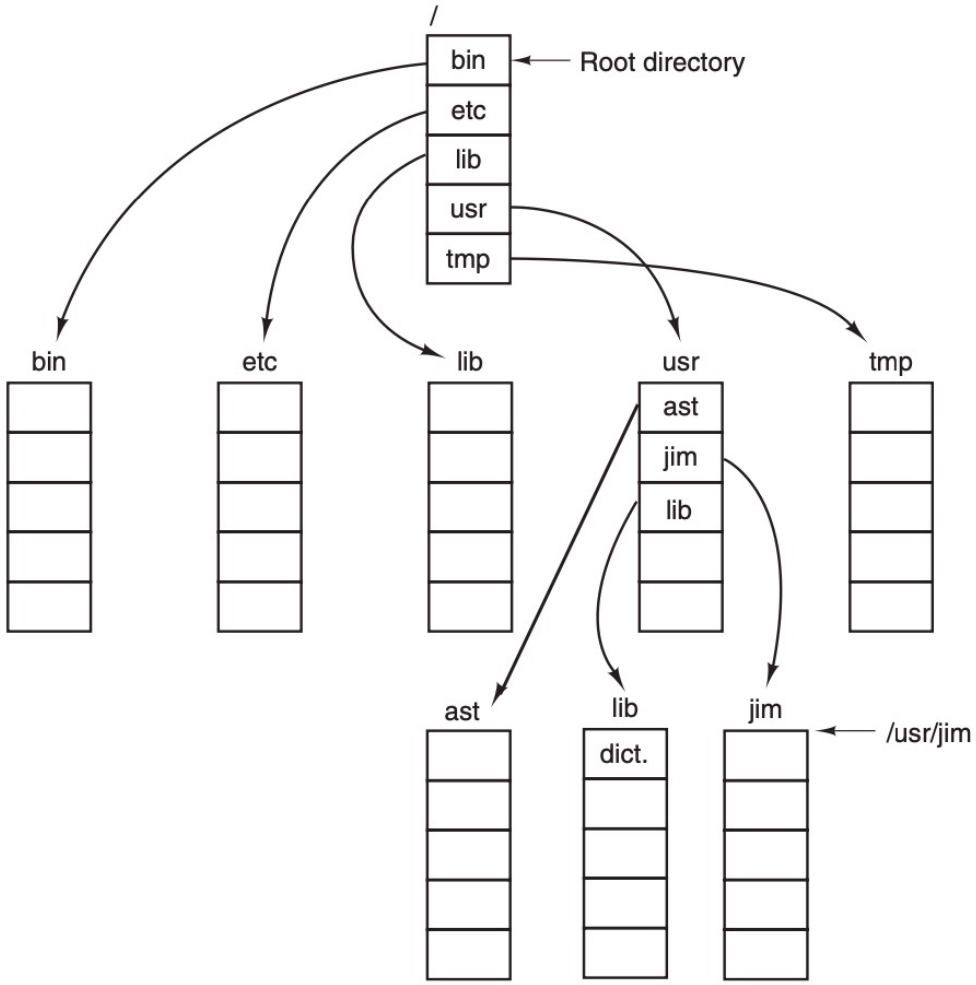


Figure: Single-level directory and DAG Directory Implementation (Tanenbaum)

Directories
DAG and Graph Complications

- The use of **DAGs** results in significant complications in the implementation
- Files have **multiple** **absolute file names**
- **Deleting files** becomes a lot more complicated (i.e. links may no longer point to a file)
- A **garbage collection scheme** may be required to remove files that are no longer accessible from the file system tree

Directories
System Calls



- Similar to files, **directories** are manipulated using **system calls**
  - create/delete: a new directory is created/deleted
  - opendir, closedir: add/free directory to/from internal tables
  - Others: rename, link, unlink, list, update

Implementation view of file system
Context

- Irrespectively of the type of file system, a number of **additional considerations** have to be addressed from the implementation view, including
  - Disk **partitions**, **partition tables**, **boot sectors**, etc.
  - Free **space management** (⇒ free memory)
  - System wide and per process **file tables** (⇒ process tables)
  - How file and directories **store** (to be introduced in the **next lecture**)

- **Low level formatting** writes sectors to the disk, **high level formatting** imposes a file system on top of this
  - File system manage **blocks** that can cover multiple **sectors**
  - The **file** is split up into a number of (not necessarily) blocks.

Hard Disk Structures
Partitions

- Disks are usually divided into **multiple partitions**
  - An independent file system may exist on each partition

- **Master boot record (MBR)** located at start of the entire drive:
  - Used to boot the computer (BIOS reads and executes MBR)
  - Contains **partition table** at its end with active partition
  - One partition is listed as **active** containing a **boot block** to load the **operating system**

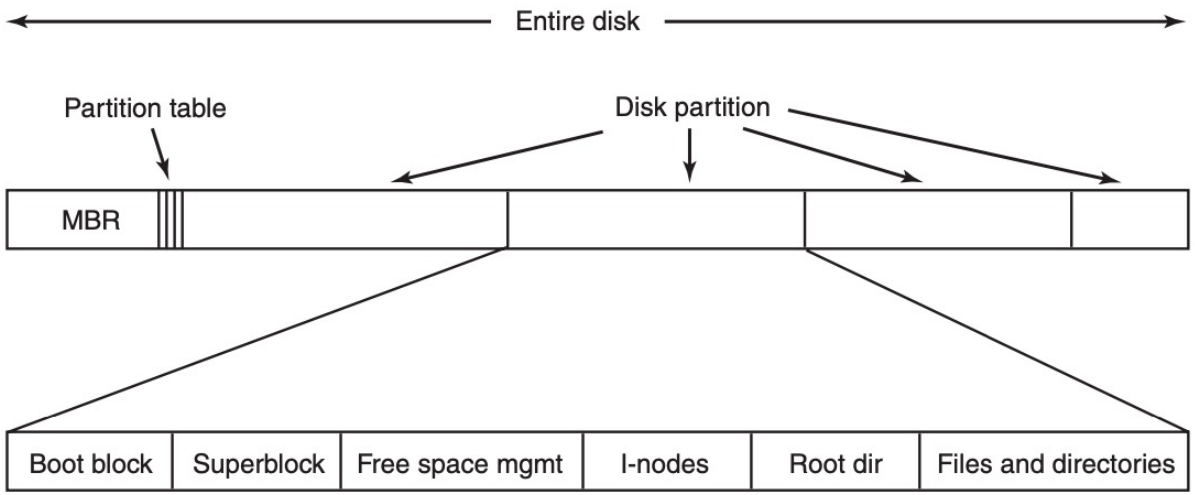


Figure: Layout of file-system

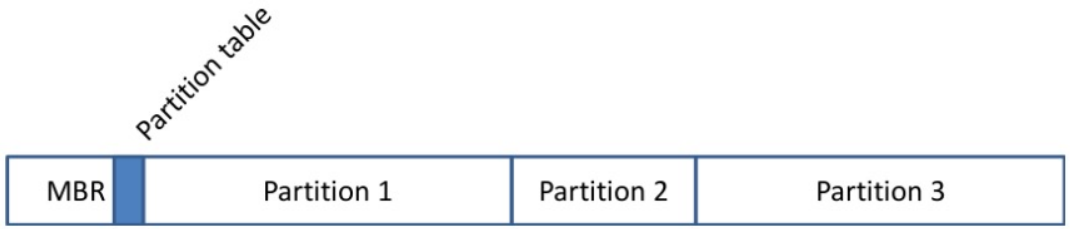


Figure: Layout of a Disk

Partition Layouts
A Unix Partition

Figure: Layout of a unix Partition

- The layout of a **partition** differs depending on the file system
  - The partition **boot block**:
    - Contains code to boot the operating system
    - Every partition has boot block – even if it does not contain OS
  - **Super block** contains the partition's (file system's) key parameters, e.g., partition size, number of blocks, I-node table size
    - Read into main memory when computer is booted
  - **Free space management** contains, e.g., a bitmap or linked list that indicates the free blocks
  - **I-nodes** contains information on files, commonly maintained in I-nodes
  - **Root directory:** contains the top of the file-system tree
  - **Data**: files and directories

Disk Space Management

Free Space Management

- Two methods are commonly used to keep track of free disk space: **bitmaps** and **linked lists**
  - Note that these approaches are very similar to the ones to keep track of free memory

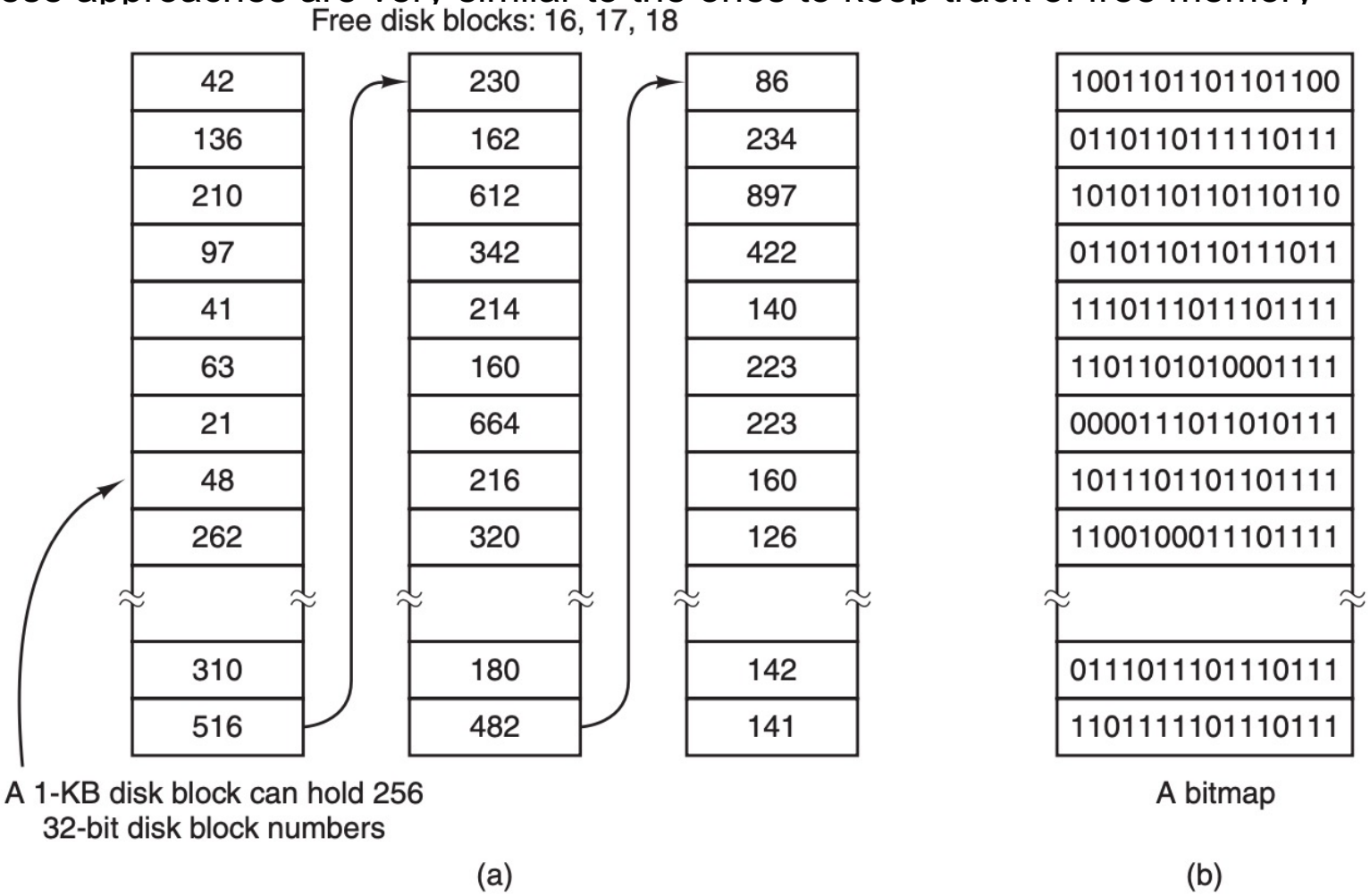


Figure: Free Block Management (Tanenbaum)

Disk Space Management

Free Space Management

- **Bitmaps** represent each block by a single bit in a map
  - The **size of the bitmap** grows with the size of the disk but is **constant** for a given disk
  - Bitmaps take **comparably less space than linked lists**
    - 1 bit per block **vs.** 1 pointer (32-bit/64-bit) per block
- **Linked Lists:**
  - Free blocks are used to hold the **numbers(pointers) of the free blocks** (hence, they are no longer free)
    - Since the free list shrinks when the disk becomes full, this is not wasted space
  - **Blocks are linked together**, i.e., multiple blocks list the free blocks
  - Linked lists can be modified by **keeping track of the number of consecutive free blocks** for each entry

Disk Space Management

Free Space Management: bitmap vs. linked list

- Bitmaps:
  - Require extra space. E.g: If block size = 2^{12} bytes and disk size = 2^{30} bytes (1 GB) ⇒ bitmap size: $2^{30}/2^{12} = 2^{18}$ (32KB)
  - Proportional to number of block
  - Keeping it in main memory is possible only for small disks.

- Linked lists:
  - No waste of disk space
  - We only need to keep in memory one block of pointers (load a new block when need).
  - If block size = 2^{12} (4K) bytes, 32-bit disk block number and disk size = 2^{30} bytes
    - block number $2^{30}/ 2^{12} = 2^{18}$,
    - each block store has 2^{10} block number
    - number of linked-list block = $2^{18}/2^{10}=2^{8}$
    - linked-list size = 2^{8} * 4KB = 1024KB

File managed by OS

Implementation

- Apart from the free disk space tables, there is a number of key data structures stored in **memory**:
  - An in-memory **mount table**
  - An in-memory **directory cache** of recently accessed directory information
  - A **system-wide open file table**, containing a copy of the FCB for every currently open file in the system, including location on disk, file size, and “open count’ **(#processes that use the file**)
    - File control blocks (FCBs) are kernel data structures, i.e. they are protected and only accessible in kernel mode!
  - A **per-process open file table**, containing a pointer to the system open file table

File managed by OS
File tables

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

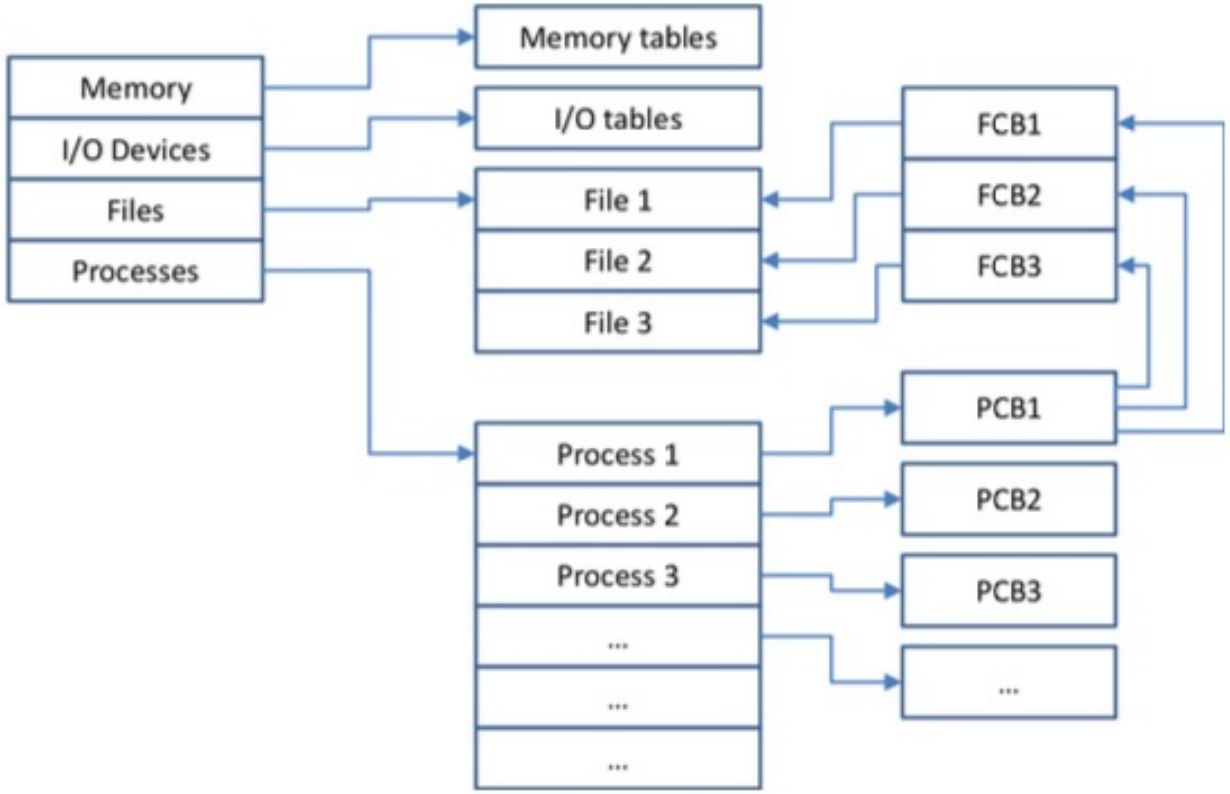


Figure: File control block (FCB) and File table

File Tables
Illustration

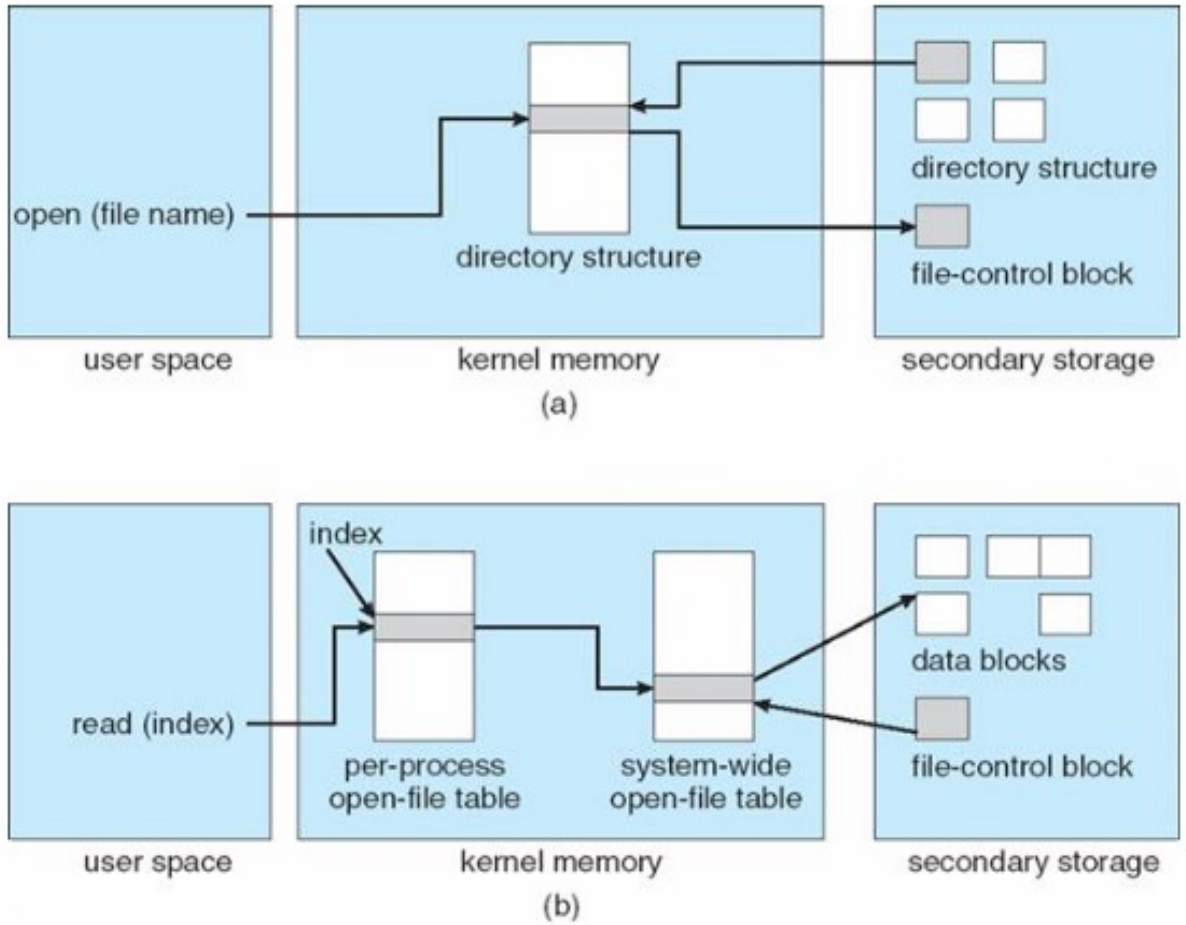


Figure: (a) Opening a file (b) reading a file (Silberschatz)

Summary
Take-Home Message

- User vs. implementation view
- System calls for file and directory management
- File tables, free space management, partitions, boot sectors, etc.