Assignment Project Exam Help

Operating Systems and Concurrency

https://powcoder.com

Lecture 19: File Systems II

G52OSC

Add WeChat powcoder

Geert De Maere and Isaac Triguero
{Geert.DeMaere,Isaac.Triguero}@Nottingham.ac.uk

University Of Nottingham
United Kingdom

2018

- **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) $\Rightarrow$ **disk scheduling** can help to minimise the impact of this
- This **influences file system design**

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

- A **user view** that defines a file system in terms of the **abstractions** that the operating system provides
- An **implementation view** that defines the file system in terms of its **low level implementation**

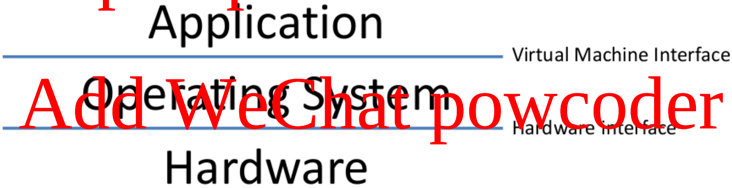

Figure: User vs. Implementation View

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** (abstracts storage details), “user file **attributes**” (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
- The **user view** defines how the file system looks like to regular users (and programmers) and relates to **abstractions**

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**
  - **Directories** group files together (but are files on an implementation level)

- Unix also has character and block special files:
  - **Character special files** are used to model **serial I/O devices** (e.g. keyboards, printers)
  - **Block special files** are used to model, e.g. hard drives

Files

File control block

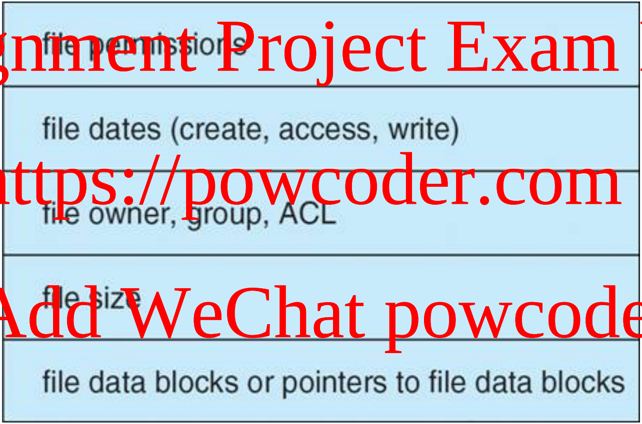

Figure: File control block (FCB) (Silberschatz)

System Calls
Types

- File control blocks (FCBs) are kernel data structures, i.e. they are protected and only accessible in kernel mode!
- Allowing user applications to access them directly could compromise their integrity

- 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**:
  - **File manipulation**: open(), close(), read(), write(), ...
  - **Directory manipulation**: create(), delete(), readdir(), rename(), link(), unlink(), list(), update()

File System Structures

Overview

- 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**
  - **Generic graph structure** in which links and **cycles can exist**
- The use of **DAGs** and **generic graph structures** results in significant **complications** in the implementation

File System Structures

Overview

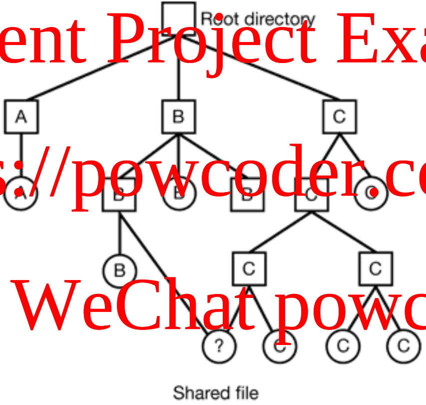

Figure: DAG Directory Implementation (Tanenbaum)

File System Structures

DAG and Graph Complications

- When searching the file system:
  - Cycles can result in **infinite loops**
  - Sub-trees can be **traversed multiple times**

- Files have **multiple absolute file names**
- **Deleting files** becomes a lot more complicated (i.e. links may no longer point to a file, **inaccessible cycles** may exist)

- A **garbage collection scheme** may be required to remove files that are no longer accessible from the file system tree (that are part of a **cycle only**)

Directories

Possible Implementations

- Directories contain a list of **human readable file names** that are mapped onto **unique identifiers** and **disk locations**
  - They provide a mapping of the logical file onto the physical location

- Retrieving a file comes down to **searching a directory file** as fast as possible:
  - A **simple random order of directory** entries might be insufficient (search time is linear as a function of the number of entries)
  - Indexes or **hash tables** can be used

- They can **store all file related attributes** (e.g. file name, disk address – Windows) or they can **contain a pointer** to the data structure that contains the details of the file (Unix)

Directories
Possible Implementations

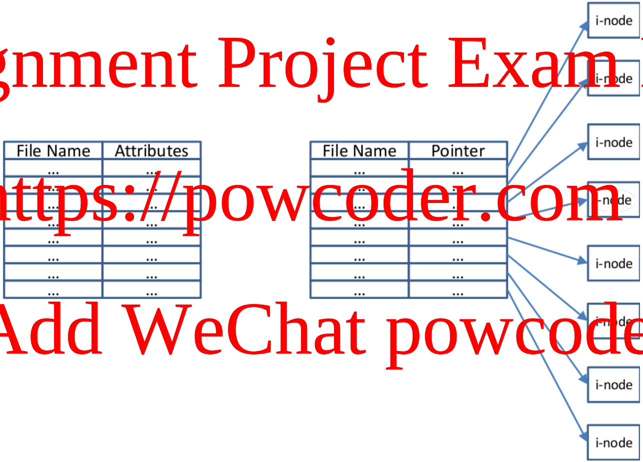

Figure: Directory Implementations

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
  - `readdir`, return the next entry in the directory file
  - Others: `rename`, `link`, `unlink`, `list`, `update`

- **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!

Implementation
Context

- Irrespectively of the type of file system, a number of **additional considerations** have to be addressed, including
  - Disk **partitions**, **partition tables**, **boot sectors**, etc
  - Free **space management** ($\Rightarrow$ free memory)
  - System wide and per process **file tables** ($\Rightarrow$ process tables)
- **Low level formatting** writes sectors to the disk, **high level formatting** imposes a file system on top of this (using **blocks** that can can cover multiple **sectors**)

Hard Disk Structures
Partitions

- Disks are usually divided into **multiple partitions**
  - An independent file system may exist on each partition
- **Master boot record** is 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 a Disk

Partition Layouts
A Unix Partition

- The layout of a partition differs depending on the file system
- A UNIX partition contains:
  - 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 details, e.g., partition size, number of blocks, I-node table size
  - **Free space management** contains, e.g., a bitmap or linked list that indicates the free blocks

Boot block	Super block	Free space MGT	I-nodes	Root dir	Files/directories

Figure: Layout of a Partition

Partition Layouts
A Unix Partition

- A UNIX partition contains (cont'ed):
  - **I-nodes**: an array of data structures, one per file, telling all about the files
  - **Root directory**: the top of the file-system tree
  - **Data**: files and directories

Boot block	Super block	Free space MGT	I-nodes	Root dir	Files/directories

Figure: Layout of a Partition

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

- **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**

Disk Space Management

Free Space Management

- A Linked List of disk blocks (also known as Grouping)
  - We use free blocks to hold the **numbers of the free blocks** (hence, they are no longer free). E.g. with a 1KB block a 32-bit disk block number, each block will hold 255 free blocks (one for the pointer to the next block).
    - 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
  - The size of the list **grows with the size of the disk** and **shrinks with the size of the blocks**
- Linked lists can be modified by **keeping track of the number of consecutive free blocks** for each entry (known as Counting)

Disk Space Management

Free Space Management

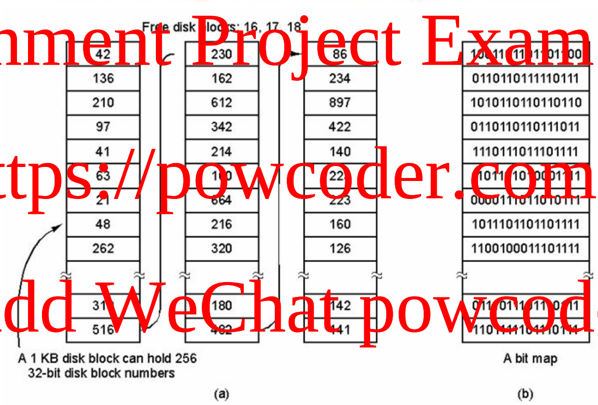

Figure: Free Block Management with Linked Lists (Tanenbaum)

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) $\Rightarrow$ bitmap size: $2^{30}/2^{12} = 2^{18}$ (32 KB)
  - 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).

File Tables
Implementation

Apart from the free space memory 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)
- A **per-process open file table**, containing a pointer to the system open file table

File Tables
Illustration

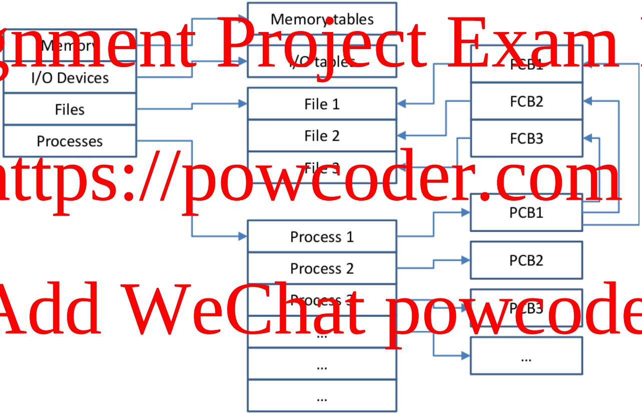

Figure: Illustration of File Tables

File Tables

Illustration

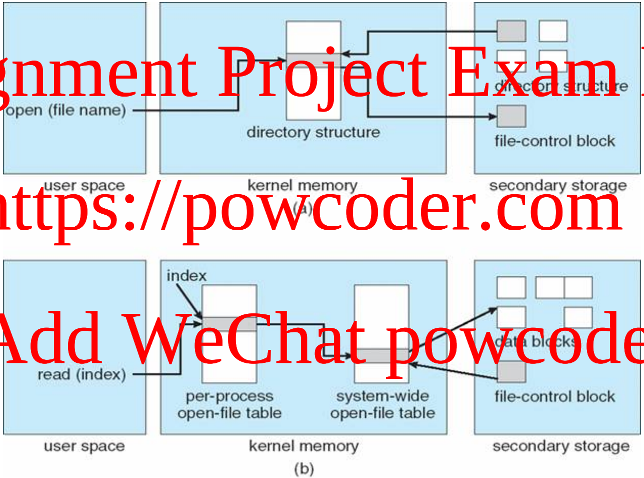

Figure: (a) Opening a file (b) reading a file (Silberschatz)

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

[1] Tanenbaum Chapter 4, Sections 4.1, 4.2, 4.3, 4.4