COMP2007: Operating Systems & Concurrency
Week 10 – 3:00pm Monday – 27 November 2023

valid for 65 minutes from 2:55pm
generated 2023-10-17 03:13

Figure: Attendance Monitoring

Operating Systems and Concurrency

File Systems 2
COMP2007

Geert De Maere
(Dan Marsden)
{Geert.DeMaere,Dan.Marsden}@Nottingham.ac.uk

University Of Nottingham
United Kingdom

2023

Recap

Last Lecture

- **Challenges** arising from the inherent nature of devices
  - **Delays** due to seek time, rotational latency, transfer times (hard drives)
  - **Block erasing** for page writing (SSDs)
- Two level **performance improvement**:
  - Disk scheduling and cylinder skew
  - File system implementation

File Systems

What Can an OS Do For Me?

```java
import java.io.FileWriter;
import java.io.IOException;
import java.io.PrintWriter;

public class Demo1 {
  public static void main(String[] args) throws IOException {
    FileWriter fw =
      new FileWriter(("C:/Program Files (x86)/test.txt"));
    PrintWriter pw = new PrintWriter(fw);
    pw.close();
  }
}
```

File Systems

- **File system abstraction**: the logical file system is mapped onto the physical one (abstraction from the physical level)
- **Abstraction from the device**: uniform view of very different underlying storage mechanism
- **Concurrency**: what if multiple processes **access the file simultaneously**
- **Security**: why is the **access denied**

File Systems

File systems allow data to be stored, located, and retrieved easily and efficiently.

Disk Layout

Boot Sector and Partitions

- Drive is a **collection of sectors** (0 - N)
- **Boot record** is located at start of the drive:
  - Used to boot the computer (BIOS reads and executes boot sector)
  - Contains **partition table** at its end with **active partition**
  - One partition is listed as **active** containing a **boot block** to **load the operating system**
- The drive is commonly split into **multiple partitions**:
  - A different **file/operating system** may exist on each partition (occasionally none)

Figure: Disk Layout

Disk Layout

Partition Layout (File System Dependent)

- **Boot block** containing code to boot the operating system (for every partition irrespective containing an OS)
- **Super block** containing stats about the partition (partition size, number of FCBs, location of free list, . . .)
- **Free space management** contains **data structures** to indicate **free FCBs** or **data blocks**
- **Meta data** or **File Control Blocks** (e.g. i-nodes)
- **Data blocks**, including the **root directory** (the top of the file-system tree)

Figure: Disk Layout

File Systems

OS Abstractions

- A **user view** that defines a file system in terms of the **abstractions** (system calls) that the operating system provides (**files** and **directories**)
- An **implementation view** that defines the file system in terms of its **low level implementation**

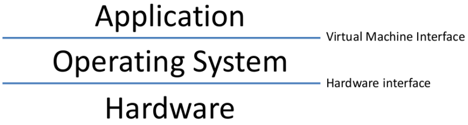


Figure: User vs. Implementation View

File System Layers

Logical Layers

- Shared layers:
  - **I/O control** interacts with the **device controller/registers** (device drivers, interrupt handlers)
  - **Basic file system** instructs device drivers "blocks" , **schedules** I/O, and manages **buffers** and **caches** for (meta-)data
- File system specific layers:
  - **File organisation** models **logical blocks** for files and **free space**
  - **Logical file system** manages **file control blocks**, **directory** structures, and **protection**
- **Application programs** define the structure of the files

Figure: File System Layers

Files

Types

- 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 are **sequential**, **random (direct) access**, **indexed access**

Files

File Control Blocks & Tables

- **File control blocks** (FCBs) are **kernel** data structures
  - 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)
- FCBs are kept in the **per process** and **system wide open file table** (array) indexed using a process specific file handle

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) (Silberschatz)

Files

File Control Blocks & Tables

- The per **process file table** contains **process specific information**, e.g.:
  - All **files currently open** to the process
  - Read/write/current **pointers**
  - A **reference** to the relevant entry in the system wide file table
- The **system wide file table** contains **general information**, e.g.:
  - One entry per open file
  - Location on disk
  - Access times
  - **Reference count**

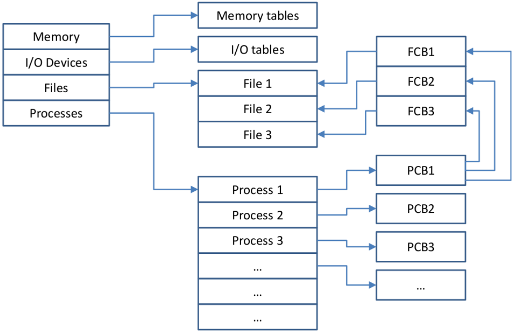


Figure: File Tables

Files

System Calls

- **System calls** for file manipulation include: `create()`, `open()`, `close()`, `read()`, `write()`, ...
- For instance:
  - The `open()` system call:
    1. Maps the **logical name** onto the **low level name** identifying the **file control block**
    2. **Retrieves** the "FCB" (from the drive)
    3. Adds it to the **process/system open file table** (increments the **reference count**)
    4. Returns a **process specific file handle** (index into the table)
  - The `close()` system call:
    1. Decrements the **reference count**
    2. **Synchronise FCB** with disk
    3. **Removes FCB** from process/system file tables (when reference count = 0)

Files

System Calls – Illustration Using “strace” (dtruss on MacOS)

```
# command = strace cat helloWorld.txt > /dev/null

execve("/usr/bin/cat", ["cat", "helloWorld.txt"], 0x7fffccb21658 /* 34 vars */) = 0
...
open("helloWorld.txt", O_RDONLY)         = 3
...
read(3, "Hello World\n", 1048576)        = 12
write(1, "Hello World\n", 12)            = 12
read(3, "", 1048576)                     = 0
...
close(3)                                 = 0
close(1)                                 = 0
close(2)                                 = 0
exit_group(0)                            = ?
```

Directories

Implementations

- **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
  - They map **human readable "logical" names** onto **unique identifiers** for **file control blocks** that detail **physical locations** and **file attributes**
- **Two approaches** exist:
  - All attributes are **stored in the directory file** (e.g. file name, disk address – Windows)
  - **A pointer** to the data structure (e.g. **i-node**) that contains the file attributes (Unix)

File Name	Attributes
...	...
...	...
...	...
...	...
...	...
...	...
...	...
...	...
...	...

Figure: Directory Implementations

Directories

Implementations

- **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
  - They map **human readable "logical" names** onto **unique identifiers** for **file control blocks** that detail **physical locations** and **file attributes**
- **Two approaches** exist:
  - All attributes are **stored in the directory file** (e.g. file name, disk address – Windows)
  - **A pointer** to the data structure (e.g. **i-node**) that contains the file attributes (Unix)

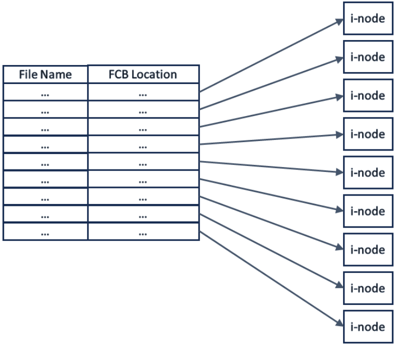


Figure: Directory Implementations

Directories

Implementations

- Directories enable to build **directed acyclic-graphs** (generalisation of a **tree structure** – links can compromise this)

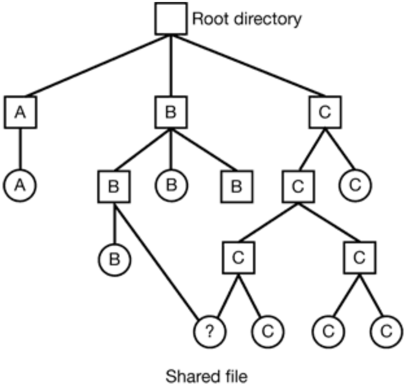


Figure: DAG Directory Implementation (Tanenbaum)

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`
- Common **operations** include, creating, deleting, searching, listing, **traversing**, . . .

File Access

Reading /home/pszgd/COMP2007/helloWorld.txt

```
#Steps to read /home/pszgd/COMP2007/helloWorld.txt

- read FCB for /
- find FCB location for /home
- read FCB for /home
- find FCB location of /home/pszgd
- read FCB for /home/pszgd
- find FCB location for /home/pszgd/COMP2007
- read FCB for /home/pszgd/COMP2007
- find FCB location for /home/pszgd/COMP2007/helloWorld.txt
- read FCB /home/pszgd/COMP2007/helloWorld.txt
  => update per process/system file tables
- read data for /home/pszgd/COMP2007/helloWorld.txt
- close the file
  => update per process/system file tables

(last access times may need updating on disk)
```

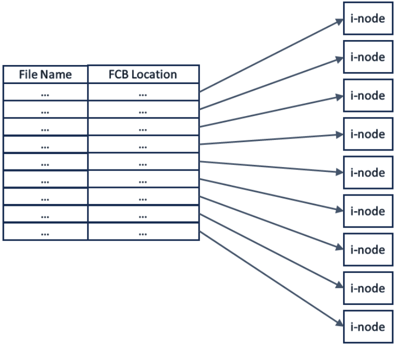


Figure: Directory Implementations

Directories

System Calls

- 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 for large directories

Free Space Management

Bitmaps

- Similar to memory management, **bitmaps** and **linked lists** can be used for 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**

Figure: Disk Layout

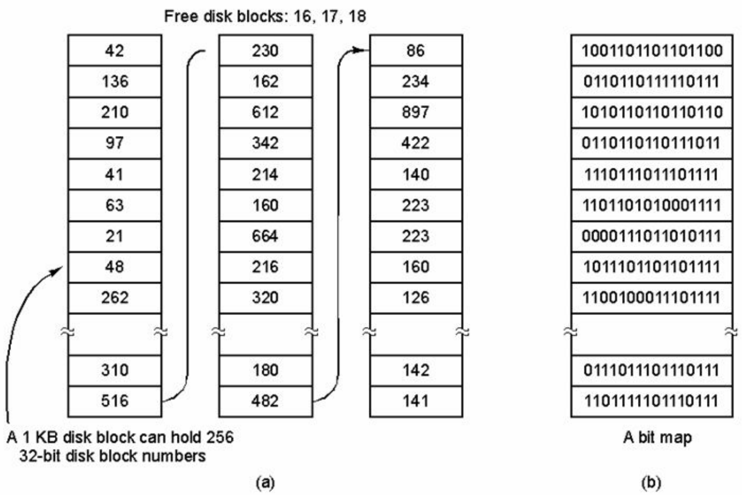
Free Space Management

Linked Lists

- **Linked List** of list free groupings
  - Use **free blocks** to hold the **locations of the free blocks** (hence, they are no longer free)
  - The size of the list **grows with the size of the disk** and **shrinks with the size of the blocks**
    - E.g., with a 1KB block a 32-bit/4 byte 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
- Linked lists can be modified by **keeping track of the number of consecutive free blocks** for each entry (known as Counting)

Free Space Management

Comparison



Free Space Management

Bitmap vs. linked list

- Bitmaps:
  - Require extra space. E.g: If block size = 2^{12} bytes (4KB) and disk size = 2^{30} bytes (1 GB) $\Rightarrow$ bitmap size: $2^{30}/2^{12} = 2^{18}$ (32KB)
  - Keeping it in main memory is possible only for small disks.
- Linked lists:
  - **Grows** with the number of empty blocks
  - **No waste** of disk space (uses empty space)
  - We only need to keep in memory **one block of pointers** (load a new block when need).

Summary

Take-Home Message

- File System Layers and Disk layouts
- Implementation of files, directories, and OS data structures
- Free space management, partitions, boot sectors, etc.