Operating Systems: Internals and Design Principles

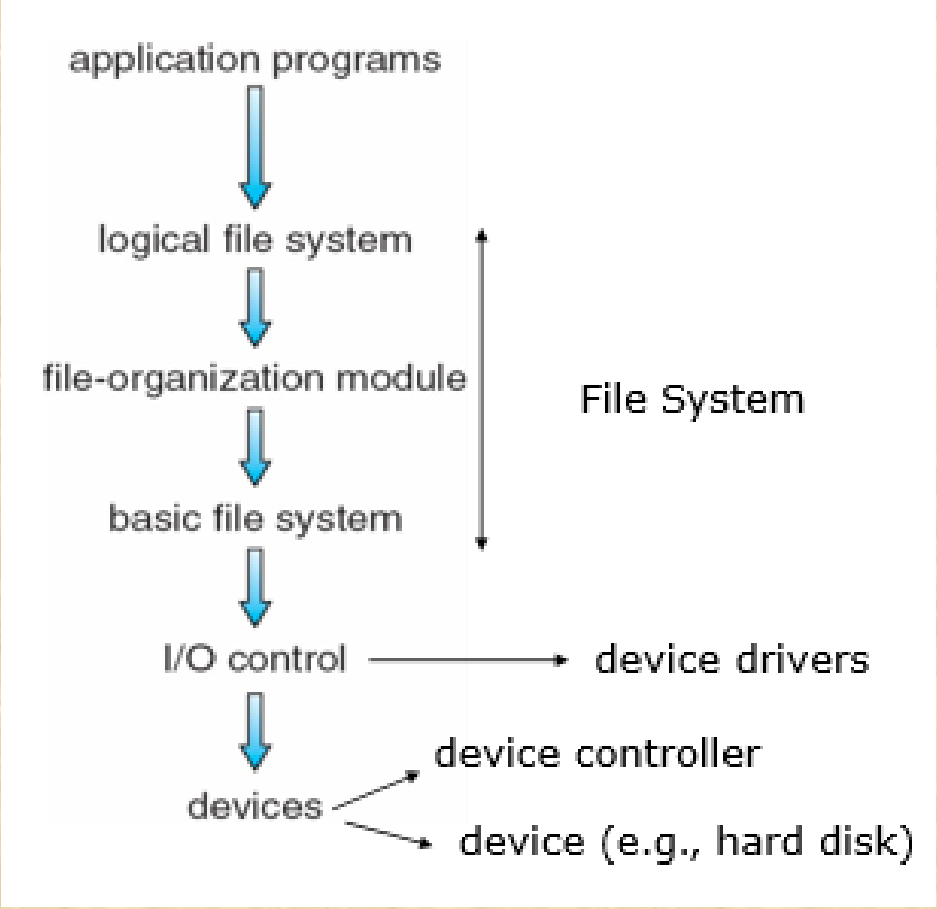
Chapter 10 File System Implementation

Andrew S. Tanenbaum, "Modern Operating Systems", 4th Edition, Pearson, 2015

Files System Structure

- File structure
  - Logical storage unit
  - Collection of related information
- File system structures and data reside on secondary storage (disks)
  - Provides efficient and convenient access to disk by allowing data to be stored, located and retrieved easily
  - Can also sit on another media (USB disk, CD-ROM, etc). Usually need a different file system
- File control block – storage structure consisting of information about a file
  - File attributes are stored here
- Device driver controls the physical device
- File system organized into layers

Layered File System



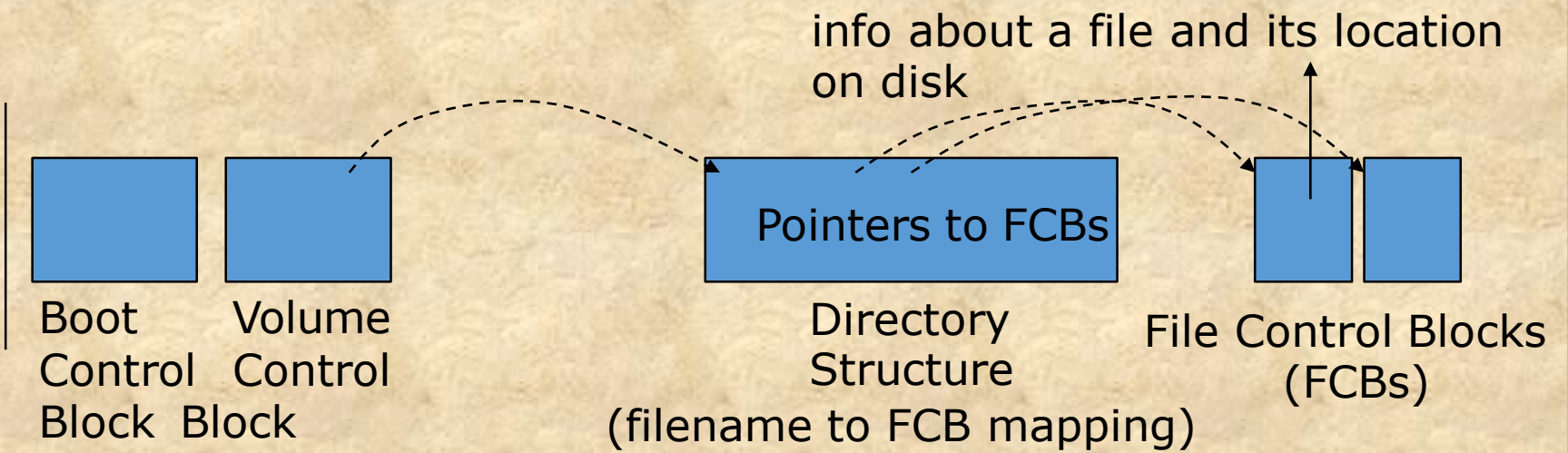
Layered File System

- ***Device drivers*** manage I/O devices at the I/O control layer
- ***Basic file system*** manage device drivers in terms of retrieving and storing raw blocks of data, without any consideration for what is in each block
- ***File organization module*** understands files, logical address, and physical blocks
  - Translates logical block # to physical block #
  - Manages free space, disk allocation
- ***Logical file system*** manages metadata information
  - Translates file name into file number, file handle, location by maintaining file control blocks (***inodes*** in UNIX)
  - Directory management
  - Protection

File System Implementation (Major on disk Structures)

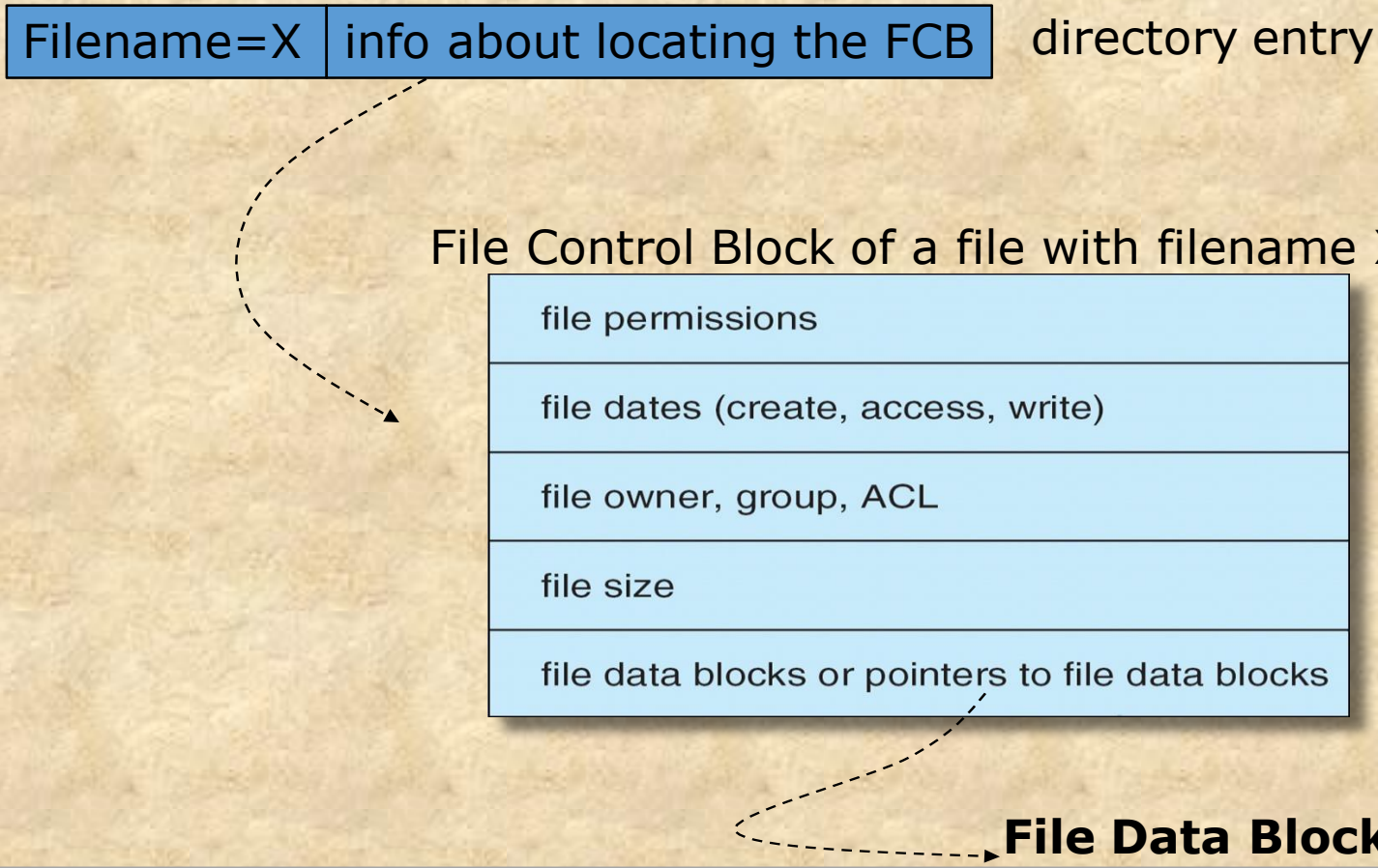
- ***Boot control block*** contains info needed by system to boot OS from that volume
  - Needed if volume contains OS, usually first block of volume
- ***Volume control block*** contains volume details
  - Total # of blocks, # of free blocks, block size, free block pointers or array
- ***Directory structure*** organizes the files
  - Names and inode numbers, master file table
- ***Per-file File Control Block (FCB)*** contains many details about the file
  - inode number, permissions, size, dates

File System Implementation (Major on disk Structures)



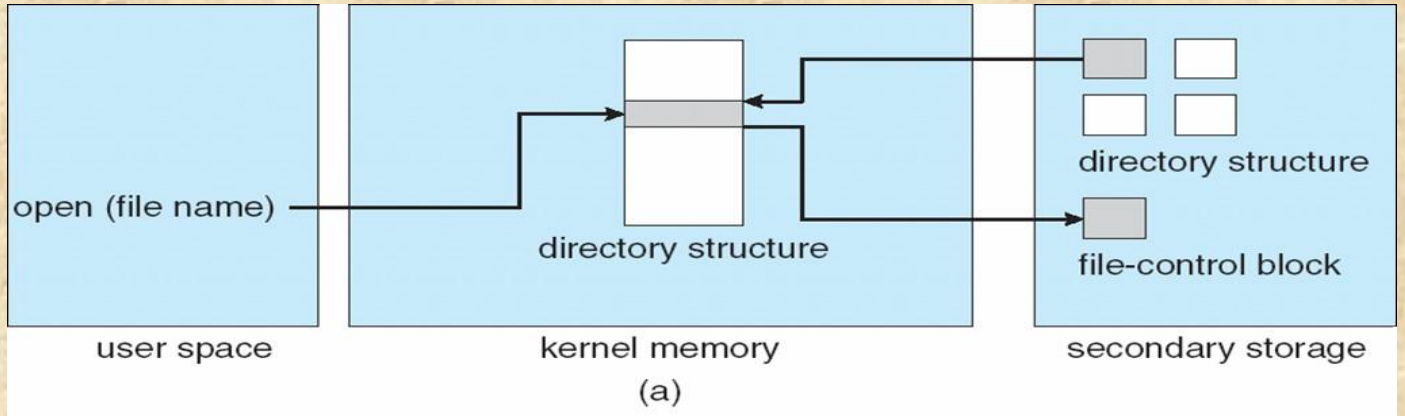
8

File System Implementation (A Typical File Control Block)

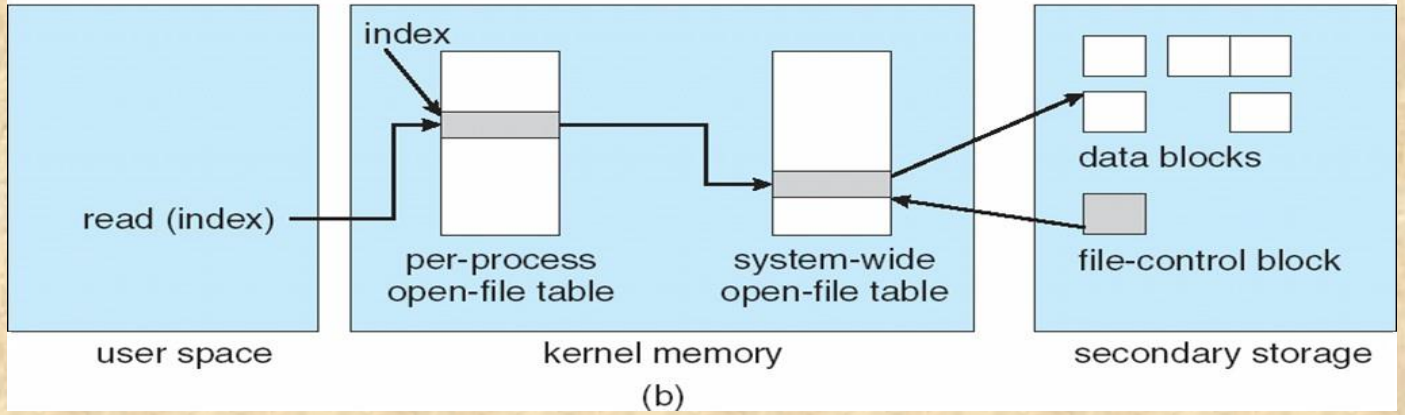


File System Implementation (In Memory File System Structures)

opening a file



reading a file



Directory Implementation

- **Linear list** of file names with pointer to the data blocks
  - Simple to program
  - Time-consuming to execute
    - Linear search time
- **Hash Table** – linear list with hash data structure
  - Decreases directory search time
  - *Collisions* – situations where two file names hash to the same location
  - Only good if entries are fixed size, or use chained-overflow method

Free-Space Mangement

- Another important aspect of disk management is keeping track of and allocating free space. One of the idea to keep allocating free space is Bit Vector.

Bit Vector

•One simple approach is to use a ***bit vector***, in which each bit represents a disk block, set to 1 if free or 0 if allocated.

•Fast algorithms exist for quickly finding contiguous blocks of a given size

•The down side is that a 40GB disk requires over 5MB just to store the bitmap. (For example.)

Free-Space Mangement

- **Grouping and Counting**

- **Grouping** – Modify linked list to store address of next free blocks in first free block, plus a pointer to next block that contains free-block pointers.

- **Counting** – Because space is frequently contiguously used and freed, with contiguous-allocation, extents, or clustering

- Keep address of first free block and count of following free blocks

- Free space list then has entries containing addresses and counts.

Efficiency and Performance

Efficiency

- UNIX pre-allocates inodes, which occupies space even before any files are created.
- UNIX also distributes inodes across the disk, and tries to store data files near their inode, to reduce the distance of disk seeks between the inodes and the data.
- Some systems use variable size clusters depending on the file size.
- The more data that is stored in a directory (e.g. last access time), the more often the directory blocks have to be re-written.
- Kernel table sizes used to be fixed, and could only be changed by rebuilding the kernels. Modern tables are dynamically allocated, but that requires more complicated algorithms for accessing them.

Efficiency and Performance

Performance

- Some OSes cache disk blocks they expect to need again in a ***buffer cache.***
- A ***page cache*** connected to the virtual memory system is actually more efficient as memory addresses do not need to be converted to disk block addresses and back again.
- Some systems (Solaris, Linux, Windows 2000, NT, XP) use page caching for both process pages and file data in a ***unified virtual memory.***

Efficiency and Performance

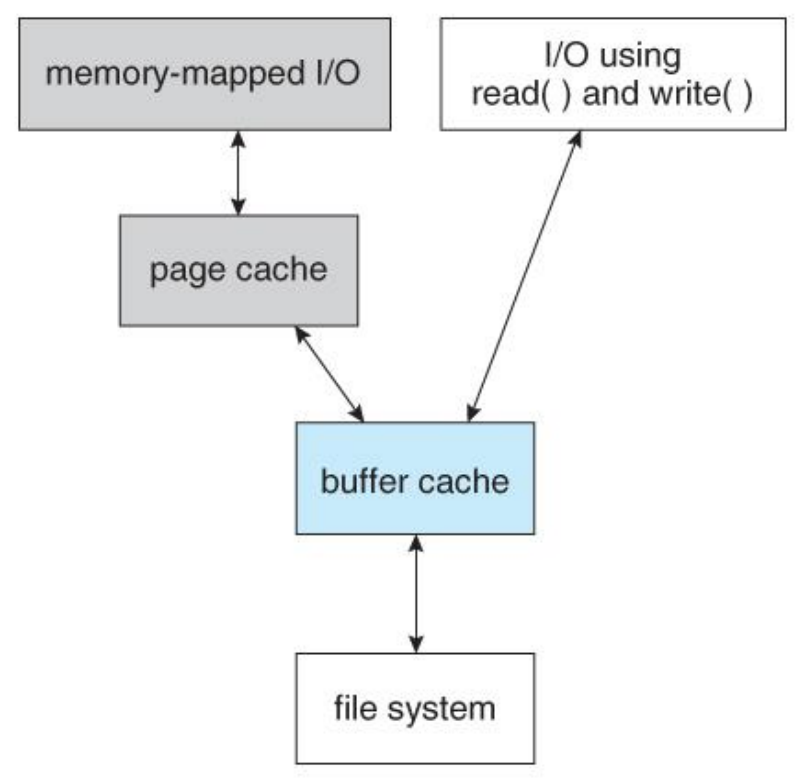


Figure 1.0 - I/O without a unified buffer cache.

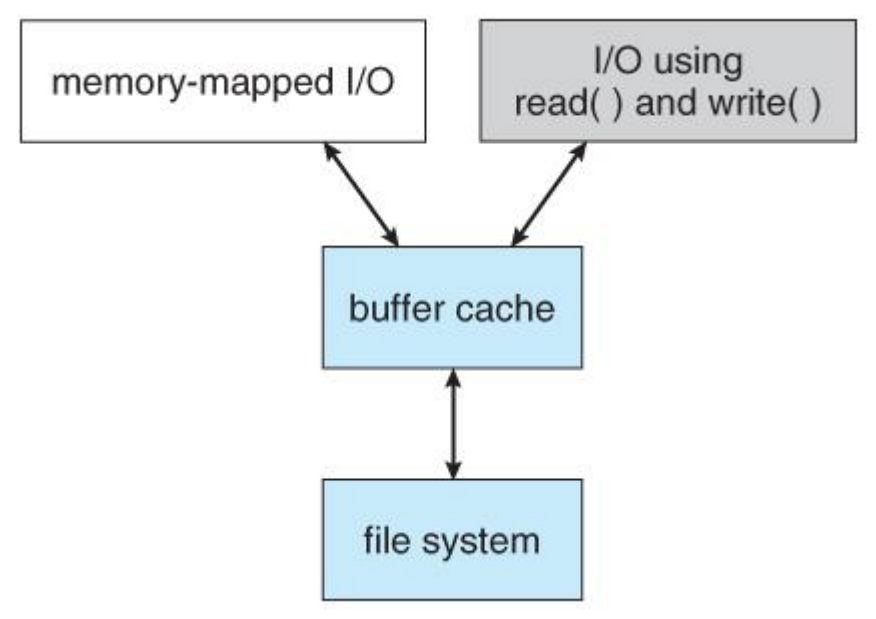


Figure 2.0 - I/O using a unified buffer cache

Recovery (Consistency Checking)

- A ***Consistency Checker*** (fsck in UNIX, chkdsk or scandisk in Windows) is often run at boot time or mount time, particularly if a filesystem was not closed down properly. Some of the problems that these tools look for include:
  - Disk blocks allocated to files and also listed on the free list.
  - Disk blocks neither allocated to files nor on the free list.
  - Consistency checkers will often collect questionable disk blocks into new files with names such as chk00001.dat. These files may contain valuable information that would otherwise be lost, but in most cases they can be safely deleted, (returning those disk blocks to the free list.)
- UNIX caches directory information for reads, but any changes that affect space allocation or metadata changes are written synchronously, before any of the corresponding data blocks are written to.

Backup and Restore

- A full backup copies every file on a file system.
- Incremental backups copy only files which have changed since some previous time.
- A combination of full and incremental backups can offer a compromise between full recoverability, the number and size of backup tapes needed, and the number of tapes that need to be used to do a full restore. For example, one strategy might be:
  - At the beginning of the month do a full backup.
  - At the end of the first and again at the end of the second week, backup all files which have changed since the beginning of the month.
  - At the end of the third week, backup all files that have changed since the end of the second week.
  - Every day of the month not listed above, do an incremental backup of all files that have changed since the most recent of the weekly backups described above.

Backup and Restore

- Backup tapes are often reused, particularly for daily backups, but there are limits to how many times the same tape can be used.
- ***Backup tapes should be tested, to ensure that they are readable!***
- For optimal security, backup tapes should be kept off-premises, so that a fire or burglary cannot destroy both the system and the backups.
- ***Keep your backup tapes secure - The easiest way for a thief to steal all your data is to simply pocket your backup tapes!***
- Note that incremental backups can also help users to get back a previous version of a file that they have since changed in some way.

Summary

- File systems structure
- File system implementation
- Directory Implementation
  - Linear List
  - Hash Table
- Free-Space Management
  - Bit Vectore
  - Grouping
  - Counting
- Efficiency and Performance
- Recovery
  - Consistency Checking
- Backup and Restore