Contents

☐ File Operations
☐ File System Disk Layout
☐ File Allocation

☐ Summary

Some Definitions

☐ File descriptor (fd)
  ■ an integer used to represent a file – easier than using names

☐ Metadata
  ■ data about data - bookkeeping data used to eventually access the "real" data

☐ Open file table
  ■ system-wide list of descriptors in use
Types of Metadata

- Inode – index node, or a specific set of information kept about each file
  - Two forms – on disk and in memory
- Directory – names and location information for files and subdirectories
  - Note: stored in files in Unix
- Superblock – contains information to describe the file system, disk layout
- Information about free blocks/inodes on disk

Contents of an Inode

- Disk inode:
  - File type, size, blocks on disk
  - Owner, group, permissions (r/w/x)
  - Reference count
  - Times: creation, last access, last mod
  - Inode generation number
  - Padding & other stuff
- 128 bytes on classic Unix
Data Structures for A Typical File System

Open-file Table Information

- **File Pointer**
  - current file position pointer

- **File Open Count**
  - counter which tracks the number of file opens and closes. Why?

- **Disk Location**
  - information needed to locate the file on disk (in inode).
Opening A File

- File name lookup and authentication
- Copy the file metadata into the in-memory data structure, if it is not in yet
- Create an entry in the open file table (system wide) if there isn’t one
- Create an entry in PCB
- Link up the data structures
- Return a pointer to user

\[ \text{fd} = \text{open(FileName, access)} \]

Allocate & link up data structures
File name lookup & authenticate
File system on disk

Read And Write

What happens when you...
- read 10 bytes from a file?
- write 10 bytes into an existing file?
- write 4096 bytes into a file?

Disk works on blocks (sectors)
Reading A Block

- read(fd, userBuf, size)
- Get physical block to sysBuf copy to userBuf
- read(device, phyBlock, size)
- Logical → physical
- Disk device driver
- Buffer cache

File Allocation in Disk Space

- Low level access methods depend upon the disk allocation scheme used to store file data
  - Contiguous allocation
  - Linked list allocation
  - Indexed allocation
Contiguous Allocation

- Request in advance for the size of the file
- Search bit map or linked list to locate a space
  - best fit, first fit, etc
- File header
  - first sector in file
  - number of sectors
- Pros
  - Fast sequential access
  - Easy random access
  - Easy to recover in case of crash
- Cons
  - External fragmentation
  - Hard to grow files
Linked Allocation

- Directory

<table>
<thead>
<tr>
<th>File</th>
<th>Address</th>
</tr>
</thead>
</table>

Linked Files

- File header points to 1st block on disk
- Each block points to next
- Example:
  - FAT (MS-DOS)
- Pros
  - Can grow files dynamically
  - Space efficient, little fragmentation
- Cons
  - Random/direct access: horrible
  - unreliable: losing a block means losing the rest
  - Need some bytes to store pointers
Indexed Allocation

- Solves external fragmentation
- Supports sequential, direct and indexed access
- Access requires at most one access to index block first. This can be cached in main memory
- File can be extended by rewriting a few blocks and index block
- Requires extra space for index block, possible wasted space
- Extension to big files issues
Other Forms of Indexed File Linked

Link full index blocks together using last entry.

An Example of Indexed Allocation

UNIX i-node

File Attributes
- Address of disk block 0
- Address of disk block 1
- Address of disk block 2
- Address of disk block 3
- Address of disk block 4
- Address of disk block 5
- Address of disk block 6
- Address of disk block 7
- Address of block of pointers

Disk block containing additional disk addresses
Summary

- File Operations
- File System Disk Layout
- File Allocation

- Next lecture: Review