Lecture.11.File.System.txt

- File Concept
    - OS provides uniform logical view of stored information
    - A file is a logical storage unit mapped by OS into physical devices
    - Data cannot be read from or written to secondary storage unless it is within a file
    - Data file types can be:
        - Numeric
        - Alphanumeric
        - Alphabetic
        - Binary
        - Etc.
    - Contents defined by file's creator
        - Text file
        - Source file
        - Executable file

- File Attributes
    - Each file has several attributes associated to it, such as:
        - Name
            - Only information kept in human-readable form
        - Identifier
            - Unique tag (number) identifies file within a file
        - Type
            - Needed for systems that support different types
        - Location
            - Pointer to file location on device
        - Size
            - Current file size
        - Protection
            - Controls who can do reading, writing, executing
        - Time, date, and user identification
    - Only 'Name' is stored in a human-readable format
        - All other attributes are stored as non-human-readable format
    - Information about files are kept in the directory structure, which is maintained on the disk
        - Each disk volume has its own directory structure

- File Operations
    - File is an abstract data type
        - Create
        - Write
            - At write pointer location
        - Read
            - At read pointer location
        - Reposition within file
            - Seek
        - Delete

    - Truncate
        - Shrink or extend the size of a file to the specified size
    - Open(F_i)
        - Search the directory structure on disk for entry 'F_i', and move the content of entry to memory
    - Close(F_i)
        - Move the content of entry 'F_i' from memory to directory structure on disk

- Open Files
    - Several pieces of data are needed to manage open files, such as:
        - Open-file table
            - Tracks open files
        - File pointer
            - Pointer to last read/write location, per process that has the file open
        - File-open count
            - Counter of number of times a file is open to allow removal of data from open-file table when last processes closes it
        - Disk location of the file
            - Cache of data that allows us to access information
        - Access rights
            - Per process access mode information, called permissions
                - i.e. Permissions such as read-write, read-only, etc.
            - Permissions are configurable

- Open File Locking
    - Provided by some operating systems and file systems
        - Is a type of protection mechanism
    - Similar to reader-writer locks
    - Shared lock
        - Similar to reader lock
            - Several processes can acquire concurrently
    - Exclusive lock
        - Similar to writer lock
    - File locking mechanisms
        - Mandatory
            - Access is denied depending on locks held and requested
                - Implemented in Windows
        - Advisory
            - Processes can find status of locks and decide what to do
                - Implemented in UNIX

- File Types: Names, Extension, & Function
    - i.e. Table of File Types

File Type	Usual Extension	Function

executable	exe, com, bin, or none	ready-to-run machine-language program
object	obj, o	compiled, machine language, not linked
source code	c, cc, java, perl, asm	source code in various languages
batch	bat, sh	commands to the command interpreter
markup	xml, html, tex	textual data, documents
word processor	xml, rtf, doc, docx	various word-processor formats
library	lib, a, so, dll	libraries of routines for programmers
print or view	gif, pdf, jpg	ASCII or binary file in a format for printing or viewing
archive	rar, zip, tar	related files grouped into one file, sometimes compressed, for archiving or storage
multimedia	mpeg, mov, mp3, mp4, avi	binary file containing audio or A/V information

- These file types may be implemented differently on different operating systems
  - Each file type has its own set of advantages and disadvantages

- Internal File Structure
  - Describes how files are stored in a storage medium
    - i.e. Are all files stored in one contiguous block?
    - i.e. Do files have their own distinct block of memory?
  - Internally, locating an offset within a file can be complicated for the operating system
  - Disk systems typically have a well-defined block size determined by the size of a sector
    - Modern file systems use blocks to store files
  - All disk I/O is performed in units of one block (physical record), and all blocks are the same size
  - It is unlikely that the physical record size will exactly match

the length of the desired logical record
  - This is called internal fragmentation
    - Occurs when the block is bigger than the file or bits of information to be stored
  - The benefit is that files can be stored anywhere on a storage medium, and pointers to the file are used
- Packing a number of logical records into physical blocks is a common solution to the internal fragmentation problem
- UNIX operating system defines all files to be simply streams of bytes
  - The logical record size is 1 byte
- All file systems suffer from internal fragmentation
  - Internal fragmentation loses some disk space
- Finding the best block size is a tradeoff between speed and storage space
- Note: Excessive external fragmentation can cause slower seek times in a hard drive (HDD)

- Access Methods
  - An access method is the information from file accessed and read into computer memory
  - There are 3 different ways to access data or information inside a file:
    1. Sequential access
       - Information in the file processed in order; one record after the other
       - i.e. Diagram

```
                     Current
Beginning            Position                  End
|----------------------|----------------------|
|                      |                      |
|                      |                      |
|----------------------|----------------------|
<<=====rewind=========|
                       |====read-or-write===>>
```

       - Used in tape storage mediums
    2. Direct access
       - File made of fixed length of logical records that allow programs to read and write records rapdily in no particular order
         - Gives immediate access to large amounts of information, like a database
           - This is a big advantage, compared to sequential access
         - But, there needs to be a record that stores the Relative block number
           - The relative block number is the index relative to the beginning of the file
             - It allows the OS to decide where file should be placed

- Sequential Access On Direct Access File
  - Not all operating systems support both sequential and direct access for files
  - Simulation of sequential access on direct-access file
    - i.e. Table

Sequential access	Implementation for direct access
reset	cp = 0;
read next	read cp; cp = cp + 1;
write next	write cp; cp = cp + 1;

  - Need instructions like 'reset', 'read next', and 'write next'
  - Keeping a variable `cp` that defines our current position

- Other Access Methods
  - Can be built on top of base methods
  - Creation of an index for the file
    - Keep index in memory for fast determination of location of data to be operated on
      - Most commonly used method
  - IBM indexed sequential-access method (ISAM)
    - Small master index, points to disk blocks of secondary index
      - Can be kept in main memory (RAM)
    - File kept sorted on a defined key
    - All done by the OS

- Example
  - A retail price file might list the universal produce codes (UPCs) for items, with the associated prices
    - This can be used for the index
  - Each record consists of a 10-digit UPC and a 6-digit price, for a 16-byte record
  - If our disk has 1024 bytes per block, we can store 64 records per block
    - A file of 120,000 records would occupy about 2000 blocks, which is 2 million bytes
      - 1024 / 16 = 64 records
      - 120,000 / 64 = 1875
~ 2000
  - By keeping the file sorted by UPC, we can define an index consisting of the first UPC in each block
    - This index would have entries of 10 digits each, or 20,000 bytes, and thus could be kept in (main) memory

```
- Index & Relative Files
    - Master index, points to disk blocks of secondary index
    - i.e. Diagram of Index File
                     logical record
         last name   number
        |----------|----------------|
        | Adams    |                |
        |----------|----------------|
        | Arthur   |                |
        |----------|----------------|
        | Asher    |                |
        |----------|----------------|
        |          |                |
        |     *    |----------------|
        |     *    |                |
        |     *    |----------------|
        |          |                |
        |----------|----------------|
        | Smith    |                | <-----------------|
        |----------|----------------|                   |
         Index file                                     |
    - i.e. Diagram of Relative File                     |
        |-------------------------------------|         |
        |                                     |         |
        |                                     |         |
        |-------------|-----------------|-----|         |
        | smith, john | social-security | age | <-------|
        |-------------|-----------------|-----|
        |                                     |
        |                                     |
        |                                     |
        |                                     |
        |                                     |
        |                                     |
        |                                     |
        |                                     |
        |-------------------------------------|
         Relative File
    - VMS OS provides index and relative files like ISAM
        - https://www.vmssoftware.com

- Directory Structure
    - The directory can be viewed as a symbol table that translates
      file names into their file control blocks
    - Directory can be organized in many ways
        - Depends on factors such as:
            - Is the file shared?
            - How is the file protected?
            - Who has permissions?
                - i.e. Not everyone is allowed to delete a file
```

- A collection of nodes containing information about all files
  - i.e. Diagram of Directory To Files

```
|--------------------------------------------|
|                                            |
|   |----|  |----|  |----|  |----|  |----|   |
|   |    |  |    |  |    |  |    |  |    |   | Directory
|   |----|  |----|  |----|  |----|  |----|   |
|      |       |       |       |       |     |
|------|-------|-------|-------|-------|-----|
       |       |       |       |       |
       |       |       |       |       |
       |       |       |       |       |
       |       |       |       |       |
|------|-------|-------|-------|-------|-----|
|      |       |       |       |       |     |
|   |----|  |----|  |----|  |----|  |-----|  |
|   | F1 |  | F2 |  | F3 |  | F4 |  | F_n |  | Files
|   |----|  |----|  |----|  |----|  |-----|  |
|                                            |
|--------------------------------------------|
```

- Both the directory structure and the files reside on disk
- The simplest directory structure associates each directory for a particular file
  - Everybody has access to the file

- Disk Structure
  - Disk can be sub-divided into partitions
  - Disks or partitions can be RAID protected against failure
    - RAID stands for Redundant Array of Inexpensive Disks
      - Instead of using a single disk, multiple disks are used to store the same information
        - If one disk fails, the other is used
      - Good for combatting disk failures
    - Primarily used in cloud storage
  - Disk or partition can be used raw
    - Without a file system, or formatted with a file system
  - Entity containing file system in known as a volume
    - Each volume containing file system also tracks that file system's info in device directory or volume table of contents

- A Typical File System Organization
  - We can have a single disk divided into multiple single partitions
    - i.e. File System With 1 Disk And 2 Partitions

```
                |----- |-----------| ------|
                |      | directory |       |
                |      |-----------|       |
                |      |           |       |
partition ------|      |           |       |
```

```
            A          |      |           |       |
                       |      |   files   |       |
                       |      |           |       |---- disk 1
                       |----- |           |       |
                              |-----------|       |
                       |----- | directory |       |
                       |      |-----------|       |
        partition -----|      |           |       |
            B          |      |   files   |       |
                       |      |           |       |
                       |----- |-----------| ------|
```

- 'disk 1' is divided into two partitions: A & B
    - Each partition has its own directory and files
- We can have multiple physical disks combined into one partition
    - i.e. File System With 2 Disks And 1 Partition

```
                       |----- |-----------| ------|
                       |      | directory |       |
                       |      |-----------|       |
                       |      |           |       |---- disk 2
                       |      |           |       |
                       |      |           |       |
                       |      |           | ------|
        partition -----|      |           |
            C          |      |   files   | ------|
                       |      |           |       |
                       |      |           |       |
                       |      |           |---- disk 3
                       |      |           |       |
                       |      |           |       |
                       |      |           |       |
                       |----- |-----------| ------|
```

- 'disk 2' and 'disk 3' are joined to create a single partition; partition C
    - Both disks share files and a directory
- The advantage of combining multiple disks is that it gives a single big partition

- Directory Operations & Organization
    - Search for a file, create a file, delete a file, list a directory, rename a file, traverse the file system, etc.
    - The directory is organized logically to obtain:
        - Efficiency
            - Locating a file quickly
            - Implemented via trees
        - Naming
            - Convenient to users
                - Two users can have the same name for different files
                - The same file can have several different names
    - Grouping

```
        - Logical grouping of files by properties
            - i.e. All Java programs, all games, etc.
        - Is a solution to the naming problem

- Single & Two Level Directory
    - A single level directory for all users
        - This was the first type of organization
        - i.e. Diagram of Single Level Directory
                      |-----|----|---|------|------|------|---------|
            directory | cat | bo | a | test | data | mail | records |
                      |-----|----|---|------|------|------|---------|
                         |     |    |    |      |      |        |
                         |     |    |    |      |      |        |
                         |     |    |    |      |      |        |
                         |     |    |    |      |      |        |
               files     0     0    0    0      0      0        0
        - The single level directory has a naming problem, and a
          grouping problem
    - Separate directory for each user
        - i.e. Diagram of Two Level Directory
                            |--------|--------|--------|
                master file | user 1 | user 2 | user 3 |
                directory   |----|---|----|---|----|---|
                                 |        |        |
                                 |        |        |
                        |--------|        |        |----------|
                        |                 |                   |
                        |                 |                   |
                        |                 |                   |
        user file |---|---|---|    |---|------|---|    |-----|-------|
        directory | a | b | c |    | d | data | a |    | cpp | heads |
                  |---|---|---|    |---|------|---|    |-----|-------|
                    |   |   |        |     |    |         |       |
                    |   |   |        |     |    |         |       |
                    |   |   |        |     |    |         |       |
                    |   |   |        |     |    |         |       |
                    0   0   0        0     0    0         0       0
            - Each user has a master file directory
            - Each user has his own directory
                - The folder 'a' in 'user 2' is different from the
                  folder 'a' in 'user 1' directory
        - Path name, can have the same file name for different user
        - Searching is efficient
        - No grouping capability

- Tree Structured Directories
    - i.e. Diagram of Tree Structured Directory
                         |-------|-----|----------|
                    root | spell | bin | programs |
                         |-------|-----|----------|
```

```
                             |      |       |
                             |      |       |
            |----------------|      |       |-----------|
            |                       |                   |
            |                       |                   |
          |---|---|            |-------|------|         |
          | a | b |            | ascii | keys |         |
          |-|-|---|            |--|----|------|         |
            |   |                 |       |             |
            |   0                 |       0             |
            |                     |                     |
            |                     |                     |
    |-------------------|      |------|       |------|------|--------|
    | hex | bina | deci |      | cats |       | mail | msgs | images |
    |--|------------|---|      |------|       |------|---|--|--------|
       |            |             |              |       |      |
       |            |             0              0       |      0
       |            |                                    |
    |-----|      |-----|                                 |
    | off |      | num |                                 |
    |-----|      |--|--|                                 |
       |            |                         |------|-------|-------|
       0            |                         | jims | sandy | popes |
                    |                         |------|-------|---|---|
                |------|                         |       |       |
                | nine |                         0       0       |
                |------|                                         |
                    |                                            |
                    0                                        |-------|
                                                             | hesus |
                                                             |-------|
                                                                 |
                                                                 0
```

- Tree structured directories provide:
    - Efficient searching
    - Grouping capability
        - Can have a group of files in the same folder
            - i.e. All programs in a folder called 'applications'

- Acyclic Graph Directories
    - Used when multiple users want to share the same file
        - Users need to be able to access the same sub-folder
    - Have shared subdirectories and files
        - The same file or subdirectory may be in two different directories
            - i.e. "book.pdf" is shared among multiple users
    - Two different names (aliasing)
        - Occurs when two different folders point to the same file
    - New directory entry type
        - Link

        - Another name (pointer) to an existing file
        - When we add a new directory, we add the link
    - Resolve the link
        - Follow pointer to locate the file

- General Graph Directory
    - One of the problems with sharing files is creating a cyclic graph
        - Cycles prevent the computer from traversing directories and locating files
    - How do we guarantee no cycles?
        - Allow only links to file not subdirectories
        - Garbage collection
            - When a pointer to a file is removed, the garbage collector removes the file from main memory
        - Every time a new link is added use a cycle detection algorithm to determine whether it is OK

- Protection
    - Mode of access: (r)ead, (w)rite, e(x)ecute
        - Permissions can be individually set for each file
    - Three classes of users on Unix/Linux
        - i.e Table of User Types

Access	Number	RWX
Owner	7	111
Group	6	110
Public	1	001

    - Example:
        - For a file 'game' define an access: `chmod 761 game`
            - Is this file executable?
                - Yes, but only for 'owner' and 'public', and not for 'group'

- Understanding Linux File Permissions
    - A sample directory listing from a UNIX environment
        - i.e. Output of `ls -l` On A Unix Machine

```
drwxr-sr-t  5 cupsys lp    4096  2006-11-29  08:51  cups
-rw-r--r--  1 root   root  817   2006-11-29  08:39  fstab
-rw-r--r--  1 root   root  806   2006-12-17  00:15  group
-rw-r--r--  1 root   root  1430  2006-12-17  00:15  passwd
lrwxrwxrwx  1 root   root  13    2006-11-29  08:40  lx ->
                                                  /var/run/lx
drwxr-xr-x  2 root   root  4096 2006-12-22 23:36 rc0.d
```

        - The file 'passwd' is readable for all: 'owner', 'group', and 'public'

        - But it is only writable by the 'owner'
    - The first element of this column is the type of the file
        - "-" means it is a normal file
            - i.e. `-rw-r--r--`
        - "d" is for directory
            - i.e. `drwxr-sr-t`
        - "l" is for a link, pointing to a file
            - i.e. `lrwxrwxrwx`
    - There are several other types of files
        - But they are much less useful to know for the casual Linux system administrator

- File System Structure
    - File system resides on secondary storage (disks)
        - Provided user interface to storage, mapping logical to physical
        - Provides efficient and convenient access to disk
    - File system is organized into layers
        - i.e. Flowchart of File System Layers

```
Application programs
        |
        |
        V
Logical file system
        |
        |
        V
File organization module
        |
        |
        V
Basic file system
        |
        |
        V
   I/O control
        |
        |
        V
     Devices
```

        - Application programs is at the top
        - Devices are at the bottom

- File System Layers (1)
    - The Logical file system layer:
        - Manages metadata information
            - i.e. File system structure
        - Translates symbolic file name into file number, file handle, location by maintaining file control blocks (`inodes` by UNIX)

        - The symbolic name is the name that we see
        - Each file has a unique file number because two or more files may have the same (symbolic) name, and the OS needs to be able to differentiate them
    - Manages the file system directory including protection and security
  - File organization module translates logical to physical block number, manages free space, disk allocation, etc.
  - i.e. Flowchart of File System Layers

```
Application programs
        |
        |
        V
Logical file system
        |
        |
        V
File organization module
        |
        |
        V
Basic file system
        |
        |
        V
   I/O control
        |
        |
        V
     Devices
```

- File System Layers (2)
  - The Basic file system layer:
    - Gives commands to the device driver to read/write physical block on the disk
      - Buffers hold data in transit
      - Caches hold frequently used data
  - I/O control layer consists of device drivers and interrupt handlers to transfer translator information between main memory and the disk
    - Input
      - High level commands
    - Output
      - Low level hardware specific instructions
    - Changing the device requires changing drivers; we need new drivers
      - However, the other layers remain the same
  - i.e. Flowchart of File System Layers

```
Application programs
        |
```

```
                |
                V
        Logical file system
                |
                |
                V
        File organization module
                |
                |
                V
        Basic file system
                |
                |
                V
           I/O control
                |
                |
                V
             Devices
```

- File System I/O Control Layer
    - I/O control layer commands example:
        - Input high-level command: "retrieve block 123"
        - Output low-level command: "read drive1, cylinder 72, track 2, sector 10, into memory location 1060"

- File System Operations
    - We have system calls at the API level
        - i.e. open(), close(), read(), write()
    - But how do we implement their functions?
        - Use a combination of on-disk and in-memory structures
            - On disk structures we have:
                - 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 (superblock, master file table) contains volume details
                    - Contains total number of blocks, number of free blocks, block size, free block pointers or array
                - Directory structure organizes the files
                    - Names and inode numbers, master file table

- File System Operations
    - Per file File Control Block (FCB) contains many details about a file; an FCB exists for each file in the storage medium
        - Permissions, inode number, size, dates
        - NTFS stores into in-master-file-table using relational DB structures

```
    - i.e. Figure of Typical File Control Block
        |---------------------------------------------------|
        | file permissions                                  |
        |---------------------------------------------------|
        | file dates (create, access, write)                |
        |---------------------------------------------------|
        | file owner, group, ACL                            |
        |---------------------------------------------------|
        | file size                                         |
        |---------------------------------------------------|
        | file data blocks or pointers to file data blocks  |
        |---------------------------------------------------|
```

- In Memory File System Structures (1)
    - Several file system structures are maintained in memory
    - When we perform some operations on a file, we need to create some structure in memory that will maintain it; such as:
        - Mount table storing file system mounts, mount points, file system types
        - Cached portions of the directory structure
        - A system-wide open-file table that contains a copy of the FCB for each open file
        - A per process open-file table that contains a pointer to the appropriate entry in the system-wide open-file table
        - Buffers for assisting in the reading/writing of information from/to disk
    - Note: All of this stuff is under the operating system's purview and responsibility

- In Memory File System Structures (2)
    - Inside the memory is the directory structure
        - It uses the file name to get the file control block (FCB)
    - i.e. If you want to read a file, you need to first open it
        - The directory structure for the particular file is moved to the memory
            - This verifies that the file exists, and its location in memory is known
    - The system wide open-file table is used to get the data blocks

- Directory Implementation
    - Directory maintains a symbolic list of file names with pointers to the data blocks
    - Different algorithms can be used for directory implementation, such as:
        - Linear list of file names with pointer to the data blocks
            - Simple to program
            - Can be time consuming to execute because it is linear search time
                - Alternatively, we could keep ordered alphabetically via linked list or use B+ tree

- Hash table
  - Linear list with hash data structure
    - Decreases directory search time
    - Collision problem
      - Two file names hash to the same location
      - Use chaining for collision resolution

- Allocation Methods
  - An allocation method refers to how disk blocks are allocated to files
    - Want to utilize disk space efficiently
    - Want files to be accessed quickly
    - There may be problems due to external and internal fragmentation
  - Three main methods currently used for disk allocation
    - Contiguous allocation
      - One big block of files
    - Linked allocation
      - There's a link between the blocks
    - Indexed allocation

- Allocation Methods: Contiguous Allocation
  - Contiguous allocation
    - Each file occupies set of contiguous blocks
    - Best performance in most cases
    - Simple
      - Only need starting location (block #) and length (number of blocks) are required
    - Problems include:
      - Finding space for file
        - We may have enough space, but if it is not in one contiguous block, then it won't work
      - Knowing file size in advance
      - External fragmentation
        - Could be a problem
      - Need for compaction
        - The free blocks are separated from the non-free blocks
        - Free blocks are arranged together in one contiguous block
          - This process requires time, additional work, and requires the system to be offline
        - i.e. Disk defragmentation

- Contiguous Allocation
  - For contiguous allocation, the directory contains the following information:
    - File name
    - Start location
    - Length

- Allocation Methods: Linked Allocation
    - Linked allocation
        - Each file is a linked list of blocks
            - Each block contains pointer to next block
        - No external fragmentation
            - Thus, no compaction necessary
        - Free space management system called when new block needed
        - Improve efficiency by clustering blocks into groups, but this increases internal fragmentation
        - Reliability can be a problem
            - Because, if one pointer is lost, so is the entire file
        - Locating a block can take many I/Os and disk seeks
            - Linked allocation is not the best implementation
        - The directory contains the following information:
            - File name
            - Start location
            - End location

- Allocation Methods: Linked Allocation With FAT
    - FAT (File Allocation Table)
        - Beginning of a FAT volume has a table that is indexed by block number
        - Linked list of block numbers for files
        - Can be cached in memory
        - New block allocation is simple
            - Simply find an open slot in FAT

- Allocation Methods: Indexed
    - Indexed allocation
        - Each file has its own index block(s) of pointers to its data blocks
    - Logical view
        - i.e. Diagram of Index Table

```
|------|
|      |--------->|=|
|------|
|      |--------->|=|
|------|
|      |--------->|=|
|------|
|      |--------->|=|
|------|
|      |--------->|=|
|------|
 Index Table
```

- Example Of Indexed Allocation
    - Indexed allocation
        - Each file has its own index block (or blocks) of pointers to

its data blocks
    - The first block contains pointers to all the other blocks
  - Directory contains the address of the index block
  - Location `i` in the index points to block `i` of the file
  - Supports direct access (like contiguous allocation) without external fragmentation
  - May waste disk resources
    - An entire index block must be created, even for small files that only require a few pointers

- Allocation Methods: Indexed Allocation (1)
  - How large should the index be?
    - If too large, then wasting a lot of space (every file has an index block)
    - If too small, then may not have enough space to index all of the blocks required for very large files
  - Several schemes are available to allow index blocks to be small enough so as not to be too wasteful, but 'expandable' so large files can be handled
    - Achieved by simply adding a pointer to the next block
      - If one block is not enough, it can add a pointer to the next block
      - So the two blocks will have pointers of all blocks that contain data for the file

- Allocation Methods: Indexed Allocation (2)
  - Linked scheme
    - Allow index blocked to be linked together to handle large files
      - If the file is larger than can be handled by a single index block, the last word of an index block is used to store a pointer to the next linked block
      - If the file is small enough so as to require only a single index block, then the last word of the index block is a null pointer
  - Multilevel index
    - A first level index block points to a set of second level index blocks which, in turn, point to the data blocks for the file
      - A tree structure of index blocks that can be expanded by adding additional levels

- Allocation Methods: Indexed Allocation (3)
  - Combined Scheme
    - Contains direct blocks, single-indirect blocks, double-indirect blocks, and triple-indirect blocks
      - Direct blocks point to the data
        - Good for small files
      - In a single-indirect block, the first block is used for

the pointers
            - The block can point to the same number of data blocks as the pointer
            - Good for relatively big files
        - In a double-indirect block, the first block points to a second pointer block, and the second pointer block points to data
            - Can point to much bigger data blocks
                - Good for very big files
        - Triple indirect blocks are used for extremely big files
    - Reserves some space in the first index block to point directly to data block, while others point to multi-level indexes
        - For small files only the first index is required which points directly to data block
        - Larger files may require some single-indirect indexes
        - Even larger files may require double or triple-indirect indexes

- File System Layers
    - Example: Maximum Size
        - Consider a file system that uses inodes to represent files. Disk blocks are 4KB in size, and a pointer to a disk block requires 4 bytes. This file system has 12 direct disk blocks, as well as one single, and one double-indirect disk blocks. What is the maximum size of a file that can be stored in this file system?
        - Answer: (12 * 4KB) + (1024 * 4KB) + (1024 * 1024 * 4KB)
            - Disk blocks are 4 KB in size
            - Each pointer is 4 B in size
            - Each disk block can store 1 KB of pointers
                - (4 KB / 4) = 1 KB
                  = 1024 Bytes
            - Calculation for:
                - Direct block = (12 * 4 KB)
                - Single indirect block = (1024 * 4 KB)
                - Double indirect block = (1024 * 1024 * 4 KB)
        - Note: If there was a triple-indirect block, then the calculation for it is: (1024 * 1024 * 1024 * 4 KB)

- Free Space Management
    - File system maintains free-space list to track available block/ clusters
        - Free spaces are blocks that do not contain data
    - Free space list can be implemented in a variety of ways, such as:
        - Bit vector
        - Linked list
        - Linked list with grouping
        - Contiguous block counting

- Free Space Management: Bit Vector
  - This is the simplest implementation
  - Utilize a bit vector in which each bit represents a block
    - The vector indicates if a particular block is free or occupied
  - Bit vector or bit map (n blocks)
    - If we have `n` blocks, they are labelled from 0 to `n - 1`
  - i.e. Diagram of Bit Blocks

```
      0   1   2                            n - 1
    |---|---|---|---|---|---|-------------|-------|
    |   |   |   |   |   |   |    . . .    |       |
    |---|---|---|---|---|---|-------------|-------|

             |---
             | 1 => block[i] free
    bit[i] = |
             | 0 => block[i] occupied
             |---
```

  - When looking for a free block, find the first '1' in the bit vector
  - Bit vector requires extra space for storage

- Linked Free Space List On Disk
  - A linked list can be used to maintain the free-list
    - A head pointer identifies the first free block
      - Each block contains a pointer to the next free block
      - i.e. 2 -> 3 -> 4 -> 5 -> 8 -> 9 -> 11 -> 13 -> 18
    - The logic is the same as linked allocation
    - One problem is: Cannot get contiguous space easily
    - No wasted space like with bit vector implementation
      - The bit vector implementation required extra space to store the vector

- Free Space Management: Linked List With Grouping
  - In the linked list implementation, getting a large group of free blocks requires traversing the linked list to find each free block
  - Rather than having only a reference to a single unallocated block in each linked list node, each node can contain a list of free blocks
  - Modify linked list to store address of next `n - 1` free blocks in first free block, plus a pointer to next block that contains free-block-pointers
    - Same logic as linked list allocation, but instead a linked list stores free-blocks
  - An issue with this approach is that every time you use a free block, you need to update the list so there is no dis-continuity

- Free Space Management: Contiguous Block Counting

- Space is frequently used and freed contiguously, with contiguous allocation or clustering
  - When freeing a contiguous section of blocks, no need to maintain information about all of them
  - Keep address of first free block in a contiguous section of free space and a count of how many free blocks follow it
  - Free space list then has entries containing addresses and counts
- This is the same logic used for managing data in files

- Recovery
  - Crashes, bugs, power outages, etc. may leave the file system in an inconsistent state
  - Consistency checking
    - Compares data in directory structure with data blocks on disk, and tries to fix inconsistencies
      - Can be slow and sometimes fail
      - Associate a status bit with each file, set that bit prior to making changes to the file
        - Unset that bit only when all changes are complete
  - Restoring data from a backup is one way to get back to a consistent system
    - Not having a backup will hinder the recovery process
    - Restoring from a backup is time consuming
      - A better solution is to use a log file

- Log Structured File Systems
  - Log structured (or journaling) file systems record each meta-data update to the file system as a transaction
  - All transactions are written to a log
    - A transaction is considered committed once it is written to the log (sequentially)
      - Sometimes to a separate device or section of disk
      - However, the file system may not yet be updated
  - The transactions in the log are asynchronously written to the file system structures
    - When the file system structures are modified, the transaction is removed from the log
      - The comment from the log file is removed after the transaction has been successfully carried out
  - If the file system crashes, all remaining transactions in the log must still be performed
    - This will help restore the file system to a consistent state
      - This approach is better than restoring from a backup
  - Faster recovery from crash, removes chance of inconsistency of metadata
    - Only the remaining transactions in the log file are needed to restore the system to a consistent state
      - If there is nothing in the log file, then the system is in a consistent state

- The disadvantage to this mechanism is that we need to continuously maintain a log file and record all transactions

- Questions Answered
    - Why are there many different kinds of file system formats, like FAT32, NTFS, exFAT, APFS, Etc.
        - Each of these is a different way of implementating the file allocation table (FAT)
            - Each format has its own set of advantages and disadvantages
            - i.e. One may perform better for smaller storage mediums, while another performs better for large storage mediums
        - 'NTFS' is the name of the company that invented the structure
    - Is parity checking used in modern operating systems?
        - Yes, parity checking is a type of error detection
        - It is possible to correct errors in files using advanced techniques like Reed-Solomon algorithm

- End
    - Operating systems are among the most complex pieces of software ever developed!