Operating Systems ICS 431

Week 13

Chapter 11: File System Implementation

Dr.Tarek Helmy El-Basuny

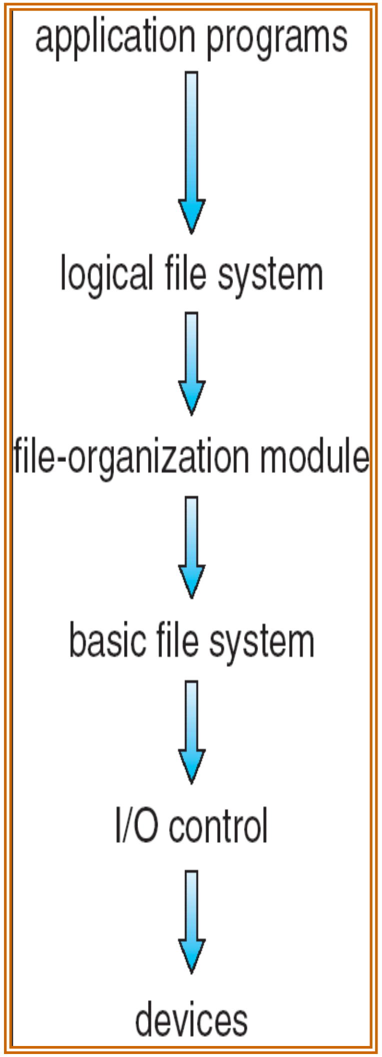
Chapter 11: File System Implementation

- How are the file system modules implemented and organized?
- What kind of on-disk and in memory data structures used to implement a file system?
- How disk blocks are allocated to files so that disk space is used effectively and files can be accessed quickly?
  - There are three allocation methods:
    - Contiguous Allocation
    - Linked Allocation
    - Indexed Allocation
- How does the file system manage the free blocks?
- How to improve the efficiency, performance, and recovery of the file system?

File System Modules Organization: Layered Approach

- File system modules organized into layers where:
  - Each layer uses the features of lower layers to offer new features to the upper layers.
- Application Programs: The programs that are making a file request.
- Logical file system: Manages the file-system through the FCB,

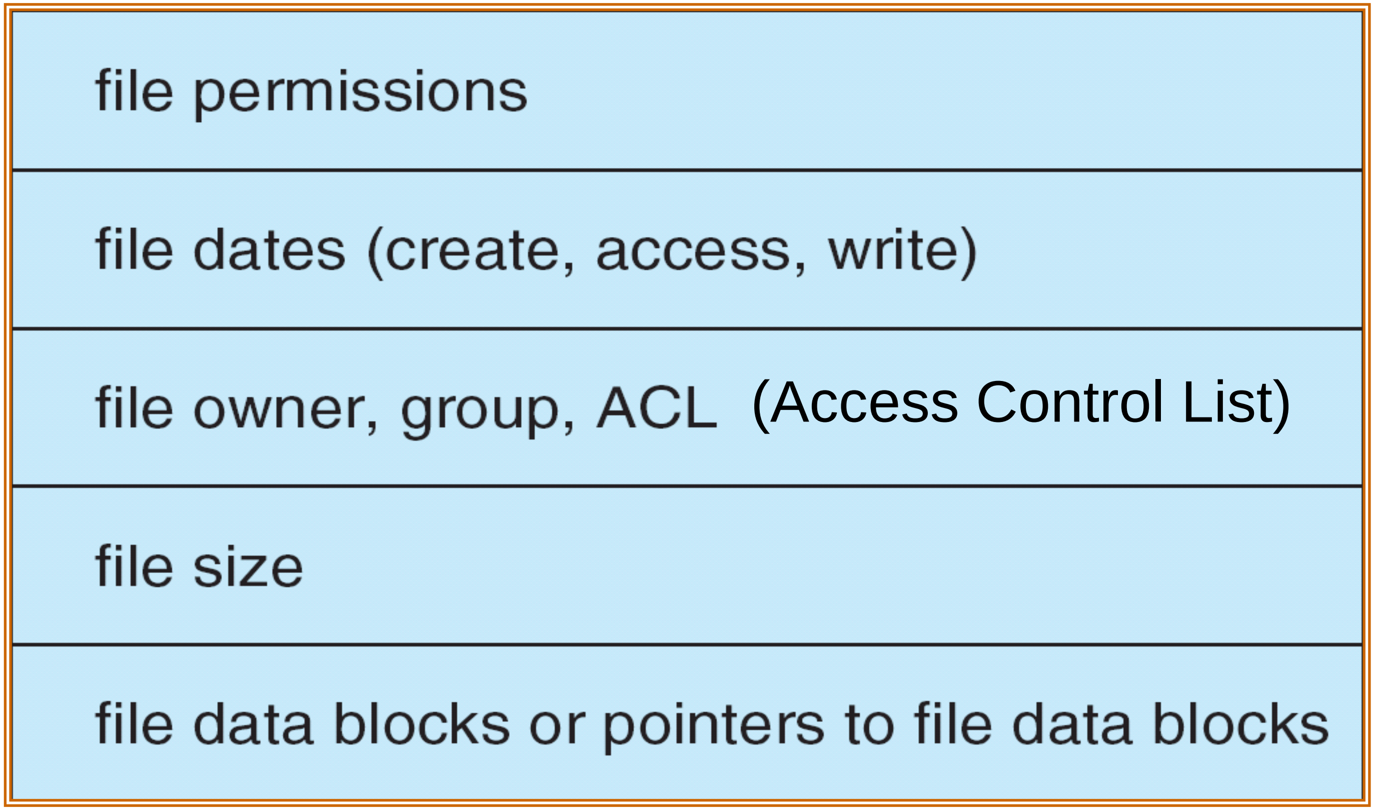
  Directory Structure Table, Protection and Security issues.
- File-organization module:
  - Reads information about the file from its FCB (i.e. logical blocks locations)
  - Translates block's logical addresses to physical addresses.
  - Manages the free spaces on the disk.
- Basic file system module:
  - Issues high level commands to a specific device driver to read & write physical blocks. i.e. (drive 1, cylinder 60, track 4, sector 10)
- I/O control:
  - Acts as a translator, it uses device drivers and interrupt handlers to transfer information between the MM and the disk system.
  - It receives high level commands i.e. retrieve specific block and outputs low level instruction.



File System Implementation: On-Disk Structure

- Several on-disk and in memory data structures are used to implement a file system. These structures vary depending of the OS and the file system.
- The on-disk data structures include:
- Boot Control Block: It contains:
  - Information needed by the OS to boot. **If the disk does not contain an OS, this block will be empty**. It is in zero block of first partition. It is called boot block in the Unix File System (UFS) and Partition boot sector in NTFS.
    - NTFS stands for New Technology File System. It is more better than FAT/FAT32, it supports Unicode filenames, proper security, compression and encryption.
- Volume Control Block: it contains:
  - Details information about partitions such as the number of blocks in the partition, size of blocks, free block count and free block pointers.
  - e.g., super block in UFS and Master File Table in NTFS.
- Directory Structure Table: Is used to organize the files within the directory.
- File Control Block: It contains:
  - File attributes such as file’s **owner**, **size** and **location** of data blocks, etc.
  - It is called inode in UFS.
  - In NTFS this information stored within Master File Table, NTFS uses relational database structure with a row per file.

A Typical File Control Block



file permissions

file dates (create, access, write)

file owner, group, ACL (Access Control List)

file size

file data blocks or pointers to file data blocks

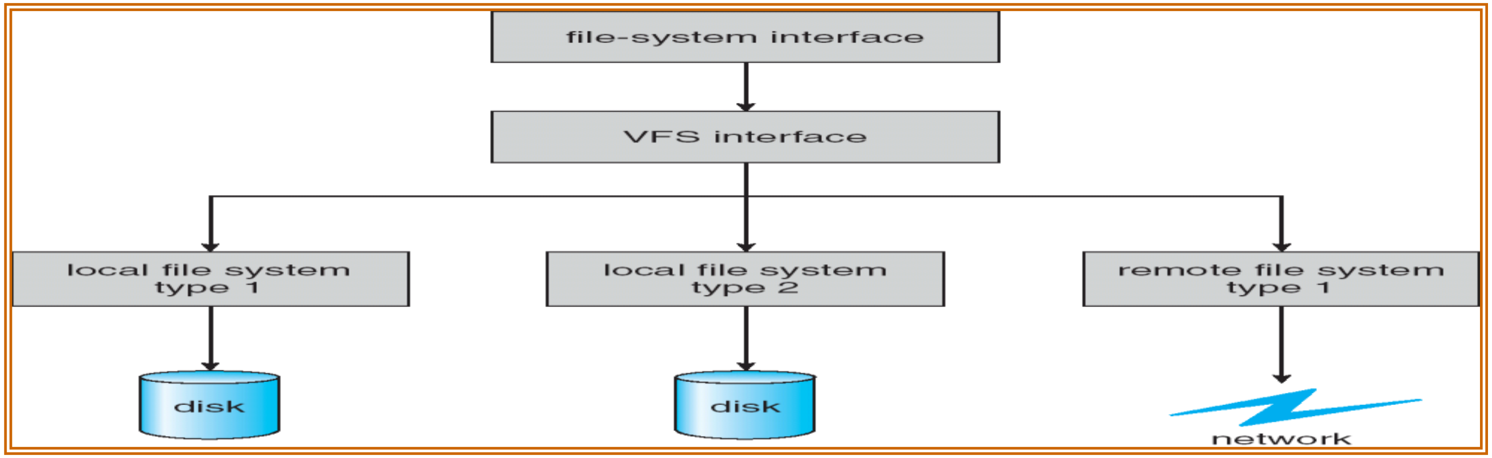
File Control Block (FCB): a storage structure contains information about a file.

File System Implementation: in-Memory Structure

- In-memory Partition Table: Contains information about each mounted partition.
  - Mounting a file system associates it with a directory in the existing file system tree. once mounted, the file system becomes accessible.
- In-memory Directory Structure: Holds information about recently accessed directories.
- In-memory System-wide Open-file Table: Contains a copy of the FCB of each open file.
- In-memory Per-process Open-file Table: Contains a pointer to the opened file entry of this process in the system-wide open-file table.
- Buffers used to map file’s blocks for reading and writing.
- The in-memory information is used for both file-system management and performance improvement.
- Caching on desk information speeds up the searching process in the data structures used to implement the file system.

Virtual File System

- Modern OSs must support multiple types of file systems, i.e. (Hierarchical File System or HFS for Mac OS), (FAT, NTFS for Windows), (Ext* family for Linux), Network File Systems, Flash File System, Tape File Systems, etc.).
- **Virtual File Systems** (VFS) provide an object-oriented way of implementing file systems.
- A **VFS** is an abstraction layer on top of a more concrete file system.
- VFS used to bridge the differences in file systems, so that applications can access files on local file systems of those platforms without having to know what type of file system they are accessing.
- The VFS allows client applications to access different types of concrete file systems in a uniform way.
- A VFS can be used to access local and remote storage devices transparently without the client application noticing the difference.

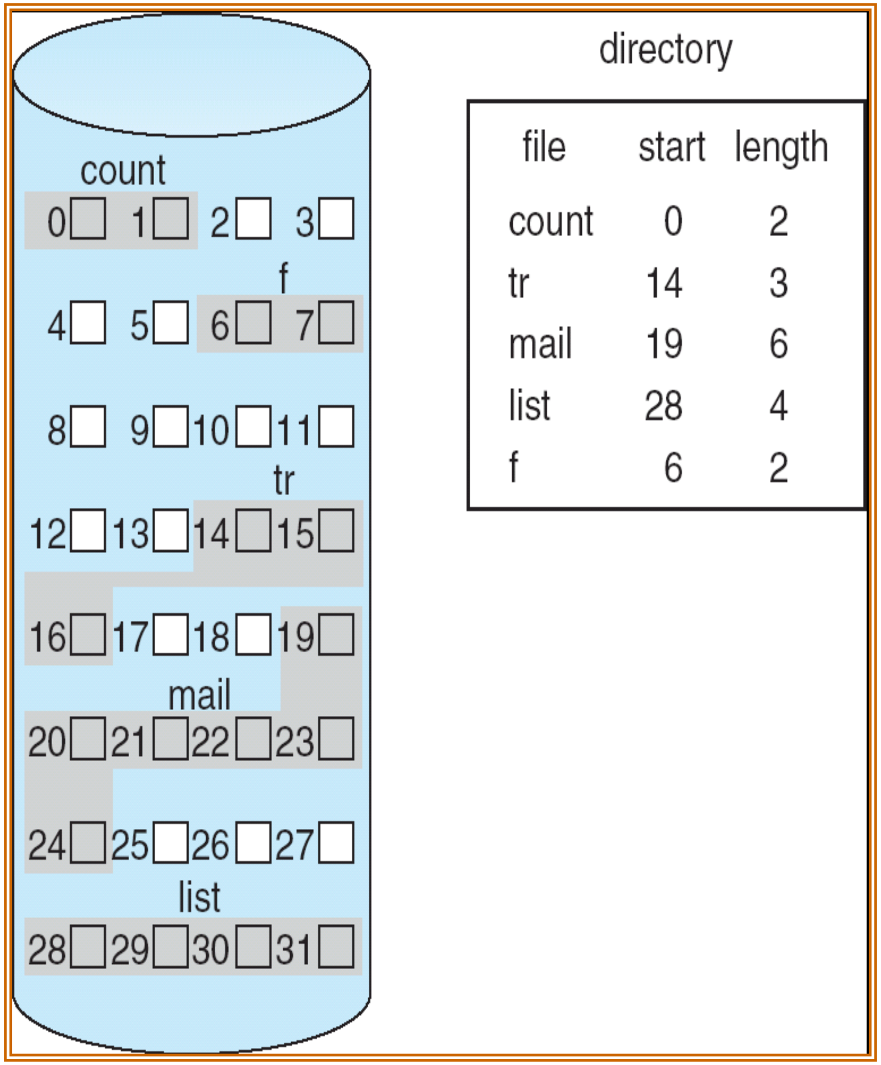


Disk Blocks Allocation Methods

- Allocation method refers to how disk blocks are allocated to files so that disk space is used effectively and files can be accessed quickly. There are three methods:
- Contiguous allocation
- Linked allocation
- Indexed allocation

Contiguous Allocation of Disk Space

- A file occupies contiguous blocks on disk.
  - This is similar to contiguous allocation of the main memory to a process's pages.
- Efficient because it offers random access to any location in a file.
  - Block i of a file is located at b+i where b is the starting location of the file.
- Faster in accessing as blocks will be quickly read one next to the other. It means the conversion of logical to physical address will be easy.
- When a new file is to be written, the file system determines where to put it.
  - Algorithms include best-fit and first-fit (which is most common).



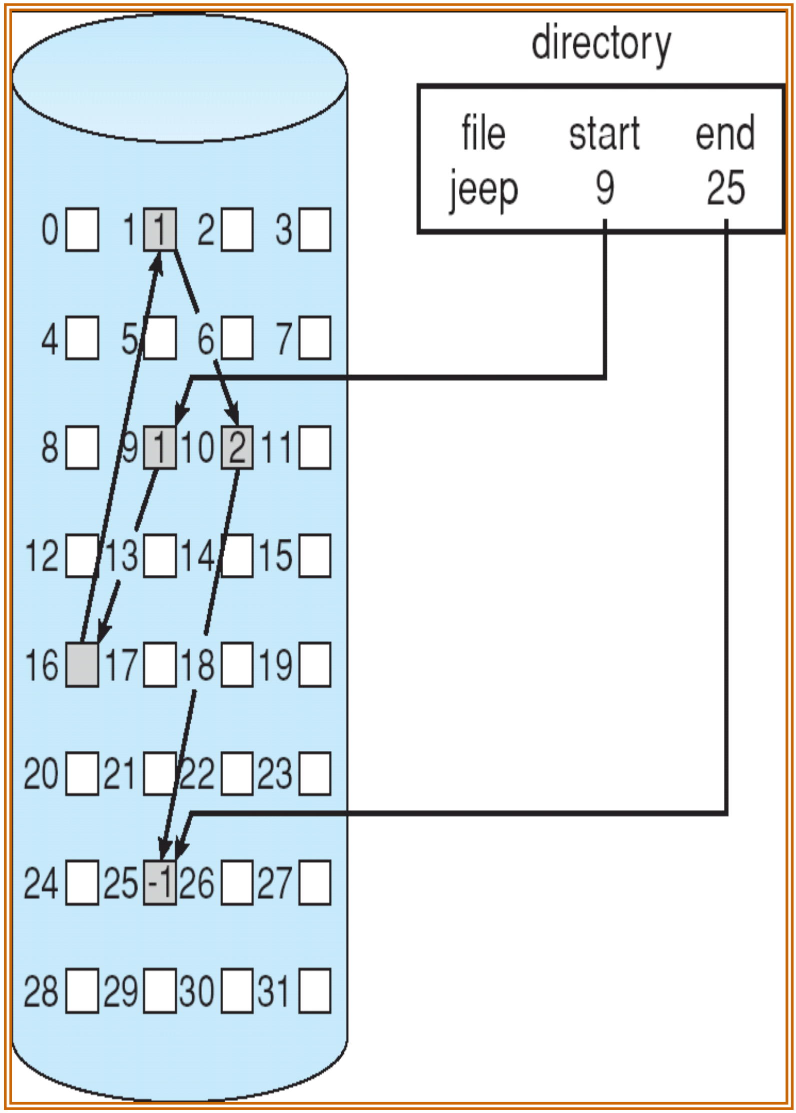
file	start	length
count	0	2
tr	14	3
mail	19	6
list	28	4
f	6	2

Drawbacks of Contiguous Allocation

- Fragmentation
  - “ Contiguous blocks” may be too big for a given file and therefore a small fragment is left. Therefore,
- Compaction/de-fragmentation will be used occasionally to eliminate fragments.
  - This requires a disk down time.
- When the file is first created, its size must be provided or estimated.
  - Program sizes can be pre-determined, but data files can not.
  - If the estimation is too low, sufficient space will not be made available later (specially if best fit was used), if it is too high, internal fragmentation occurs.

Linked Allocation

- File blocks are going to be scattered across the disk (non-contiguously) where one block points to the next block in the file.
- Each block contains a pointer to the next block and the last block contains a NIL (-1) pointer.
  - Files can grow or shrink without fragmentation and **without the need to know the file size in advance.**
  - No waste of space except for pointers.
- Pointers take up a great portion of the file space.
  - Perhaps as much as 1% of storage is now pointers.
- This method does not support random access into a file block.
  - Instead, **sequential access must be performed** from the first block, following pointers.

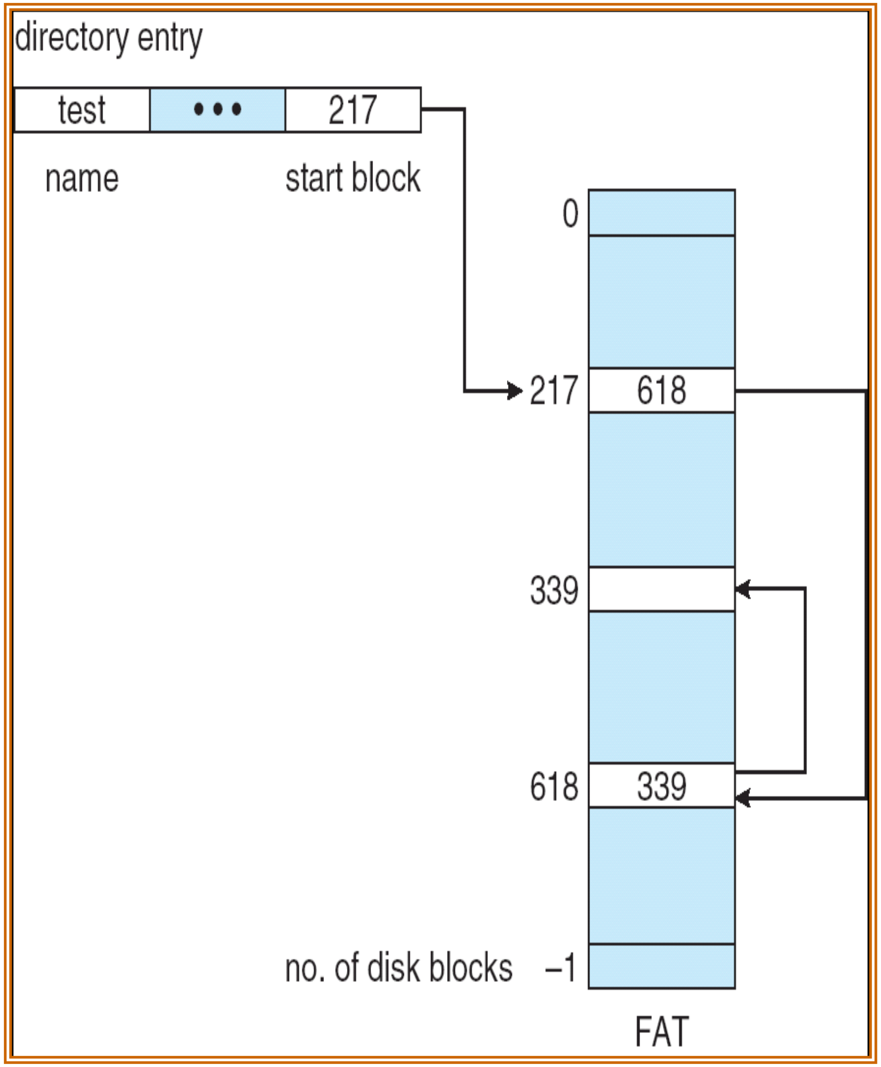


Linked Allocation Advantages and Disadvantages

- Advantages
- This method does not suffer from external fragmentation, every block wherever it is can be linked and used.
- This makes it relatively better in terms of disk space utilization.
- Any free block can be used to satisfy a request.
- There is no need to declare the size of a file when that file is created.
- A file can continue to grow as long as there are free blocks
- Disadvantages
  - Long seek time is needed to access every block individually, because the file blocks are distributed randomly on the disk.
  - This makes linked allocation slower (unless FAT is used and cached).
  - A lot of space used for pointers of the blocks,
    - One way of solving this problem is to cluster the blocks and to use pointers for clusters not for blocks.
  - Does not to support direct-access.
  - It is not reliable, since the pointers are linked, if a pointer lost or damaged a trap will occur.

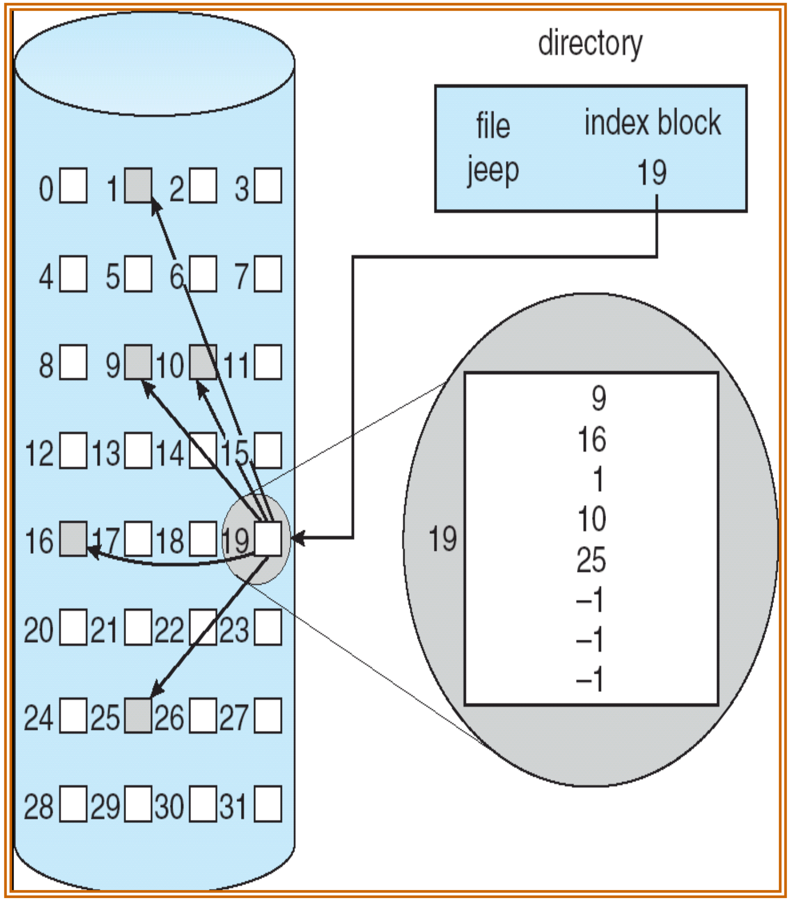
File-Allocation Table (FAT)

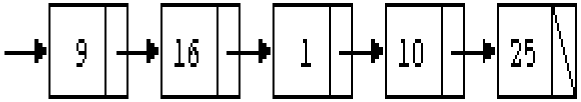
- Instead of having the pointer to the next block as a part of the block itself, why do not we have a file that contains the pointers only.
- Each entry in the FAT contains the block number of the next block in the file.
- Unused blocks are indicated by a 0 table entry.
- Example: file consisting of disk blocks 271, 618 and 339.
- The FAT can be cached or can be protected or copied to enhance the reliability.



Indexed Allocation

- Can we get the benefits of both contiguous and linked allocations.
- Allocation of blocks is still scattered across the disk like linked, but access to each block is provided by an index where we can support random access.
- Each file has its own index of pointers.
  - This allows random access to a given block without external fragmentation.
- Each file's index is stored in one block on disk and pointed to by the directory.
  - If a file can be stored in n block s, then the file can only consume n+1 blocks, 1 block for the index.





A view of the linked list

Disadvantages of Indexed Allocation

- Problems with the indexed allocation method are:
  - If a file can be stored in n blocks, then the file can needs n+1 blocks, 1 block for the index.
  - For very small files, say files that expand only 2-3 blocks, the indexed allocation would keep one entire block (index block) for the pointers which is inefficient in terms of memory utilization.
  - For files that are very large, single index block may not be able to hold all the pointers. One proposed solution is to use two or more index blocks together for holding the pointers. Every index block would then contain a pointer or the address to the next index block.

Summary of Allocation Methods

- **Contiguous Allocation:**
  - Efficient because it offers random access to any location in a file
  - **This method suffers from both internal and external fragmentation**. A block may be too big and therefore a small internal fragment is left. Or some blocks may not be used and external fragment will be left also.
  - When the file is first created, its size must be estimated. Increasing file size is difficult because it depends on the availability of contiguous memory at a particular instance.
- **Linked Allocation:**
  - Files can grow or shrink without fragmentation and the need to know the file size in advance.
  - It does not support random access.
  - Pointers take up a great portion of the file space.
  - Not reliable as pointers may be lost or damaged and causes trap errors.
- **Indexed Allocation:**
  - Allocation of blocks is still scattered across the disk like linked, but access to each block is provided by an index rather than linked pointers.
  - Each file has its own index of pointers, this allows random access to a given block without external fragmentation.
  - If a file can be stored in n block s, then the file can only consume n+1 blocks, 1 block for the index regardless the size of the file.

Selection of Allocation and Access Methods

- Source code files that need to be compiled should be sequentially accessed, while data base files should be randomly accessed.
- Media files needs to be contiguously allocated for faster access (requires only one access to get a disk block, keep the initial address in memory and increment it).
- For linked allocation, access the i^{th} block needs i disk access.
- Some systems use contiguous allocation for small files (up to 4 blocks) and automatically switch to indexed allocation if the size of the file grows.
- The type of access to be made must be declared when the file is created.
  - A file created for sequential access will be linked and can not be used for direct access.
  - A file created for direct access will be contiguous and can support both direct and sequential access but its maximum length must be declared when it is created and the OS provide algorithms to support both methods.

Free Space Management

- The file system must keep track of the free disk space,
- The operating system **maintains a free-space list**.
- The free-space list records all free disk blocks, those not allocated to files or directories.

- **The list must be managed so that a new block can easily be allocated and deleted file`s space can be returned to this list.**

- To create a file, OS searches the free space list for the required amount of space and allocates that space to the new file.
- This allocated space is then removed from the free list.
- When the file is deleted, its disk space is added to the free list.

- How does the file system know where a free block of disk space is located? There are two ways to implement that.

Bit Vector

- Using a bit vector to **indicate every block in the file system**.
- 1 indicates a free block and 0 indicates a used block.
- Example, a disk of 32 blocks where blocks 2, 3, 4, 5, 8, 9, 10, 11, 12, 13, 17, 18, 25, 26, 27 are **free** and the rest are **allocated**. The bit vector map is:

  00111100111111000110000001110000
- To allocate a new block, OS looks for the first 1 and changes it to a 0.
- This can be done by looking for the first word of first bit that is not 0.
- Block number calculation = (number of bits per word) * (number of 0-value words) + offset of first 1 bit.
- The Macintosh OS uses this technique for managing the free blocks.
- Unfortunately, bit vectors are inefficient unless the entire vector is kept in memory, this will consume more memory specially for large disks.
- **Bit map requires extra space**, example:
  - Disk size = 2^{40} bytes (1 Terabyte)
  - Block size = 2^{15} bytes = 32 KB
  - Bit vector size = $2^{40}/2^{15} = 2^{25}$ bits (or $2^{2} * 2^{3} * 2^{10} * 2^{10} = 4$ MB)

Bit Vector Variations

- The Bit victor problem:
  - The bit victor stored on disk and will be cached in memory
  - The copy in memory and on disk may differ.
  - May lead into **inconsistency** where bit [i] = 0 (used) in memory and bit [i] = 1 (free) on disk.
- **Solution**:
  - Set bit [i] = 0 in memory.
  - Allocate block [i] to the disk.
  - Set bit [i] = 0 in memory.

Linked List

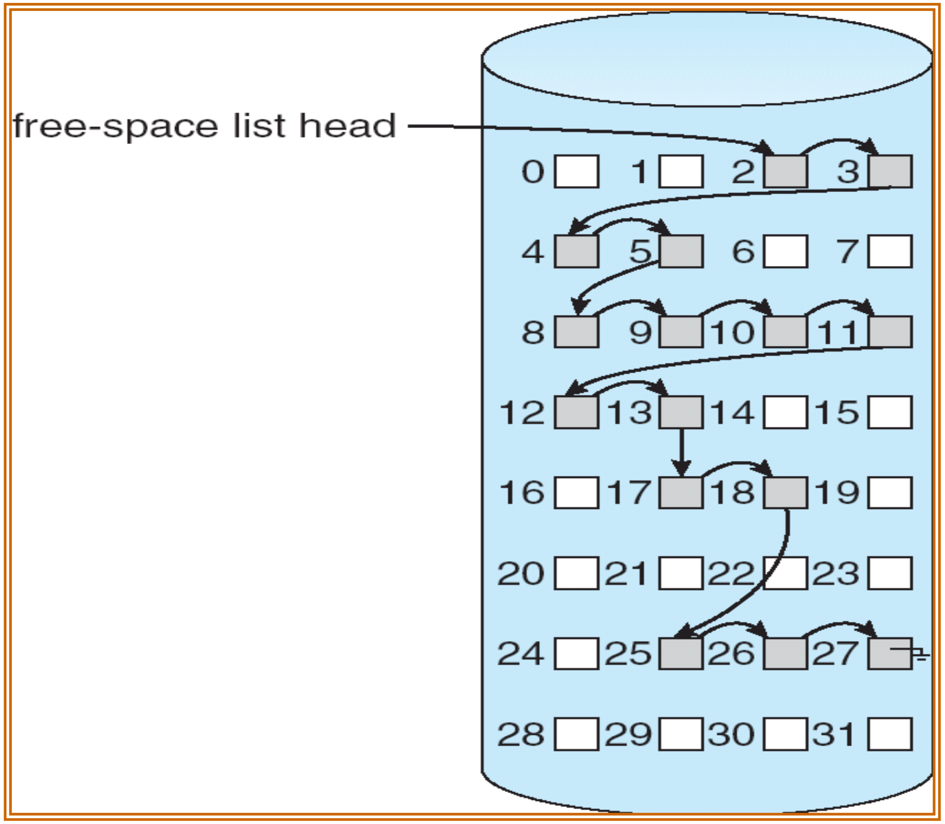
- Another way to manage the free-blocks is to link together all the free blocks, keeping a pointer to the first free block in a special location on the disk and caching it in memory.

  00111100111111000110000001110000

  - The first free block contains a pointer to the next free block and so on.
    - Allocating one block is simple.

- Two variations to avoid the previous problem:
  - Grouping: If we have n free blocks, store pointer to the n-1 free blocks in the first free block, where the last pointer points to another n blocks, etc…
  - Counting: Store a pointer and an integer which denotes how many contiguous blocks are available.

Linked Free Space List on Disk



- Os keeps a pointer to block 2 as a first free block, block 2 contains a pointer to block 3, block 3 points to block 4, and so on...

Efficiency and Performance

- Efficiency depends on:
  - Disk allocation and directory algorithms.
  - Types of data kept in file’s directory entry.
- Disks are commonly a system bottleneck
  - Ways to improve performance include
    - Cache disk requests by including a small cache in the device itself.
    - Use all currently unused main memory as a disk spool for disk caching.
    - Include a variety of cache replacement strategies.
    - Include a RAM Disk
      - Set aside part of RAM to act as a small disk
      - Since RAM is volatile, a power outage destroys information.

Consistency Checking

- The caching of certain data structures in memory can speed up the performance, but what happens in the result of a system crash?
- All volatile memory structures are lost, and the information stored on the hard drive may be left in an inconsistent state.
- A ***Consistency Checker*** (fsck in UNIX, chkdsk or scandisk in Windows) is often run at boot time or mount time, particularly if a file system 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.
  - Disk blocks allocated to more than one file.
  - The number of disk blocks allocated to a file inconsistent with the file's stated size.
  - Properly allocated files which do not appear in any directory entry.
  - Two or more identical file names in the same directory.
  - Illegally linked directories, e.g. cyclical relationships where those are not allowed, or files/directories that are not accessible from the root of the directory tree.

Recovery of Lost Data

- In order to recover lost data in the event of a disk crash, it is important to conduct backups regularly.
- Files should be copied to some removable medium, such CDs, DVDs, or external removable hard drives.
- **A full backup** copies every file on a file system.
- **Incremental backups** copy only files which have changed since some previous time.
- 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
- Recover lost file or disk by restoring data from backup.
    - A useful backup strategy is required!

Recovery

- In a system crash, some files may have been opened and partially altered.
  - A consistency checker is often used to determine lost files and tries to fix them (compares data in directory structure with data blocks on disk, and tries to fix inconsistencies.
- Use system programs to back up data from disk to another storage device.
- Recover lost file or disk by restoring data from backup.
  - Backup usually stored on external HDs
    - A useful backup strategy is required!

The End!!

Thank you

Any Questions?