CPSC 410/611: File Management

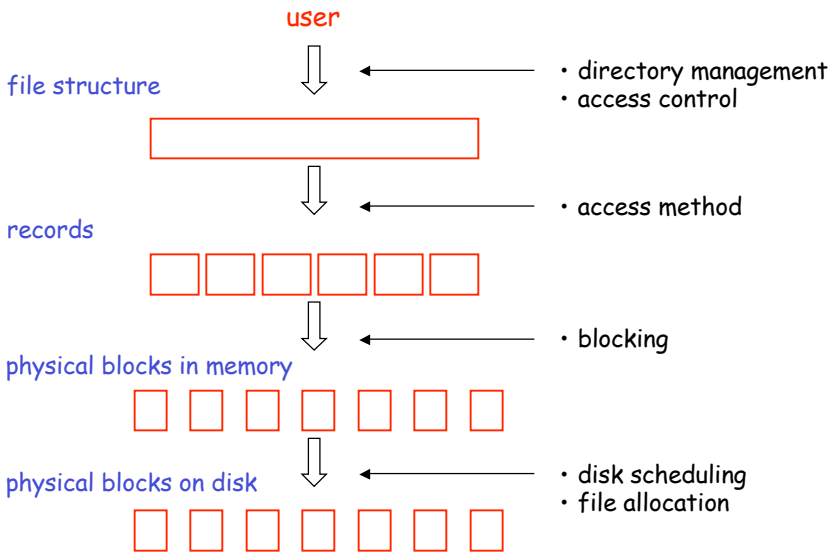
- What is a file?
- Elements of file management
- File organization
- Directories
- File allocation
- UNIX file system

- *Reading: Silberschatz, Chapter 10, 11*

What is a File?

- A **file** is a collection of data elements, grouped together for purpose of access control, retrieval, and modification
- Often, files are mapped onto physical storage devices, usually nonvolatile.
- Some modern systems define a file simply as a sequence, or stream of data units.
- A **file system** is software responsible for
  - creating, destroying, reading, writing, modifying, moving files
  - controlling access to files
  - management of resources used by files.

The Logical View of File Management

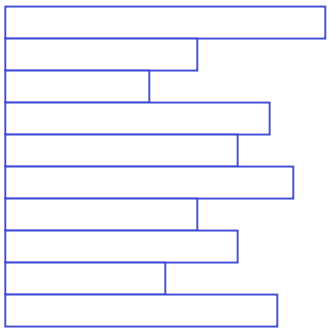


Logical Organization of a File

- A file is perceived as an ordered collection of **records**, R_0, R_1, ..., R_n.
- A record is a contiguous block of information transferred during a logical read/write operation.
- Records can be of fixed or variable length.
- Organizations:
  - Pile
  - Sequential File
  - Indexed Sequential File
  - Indexed File
  - Direct/Hashed File

Pile

- Variable-length records
- Chronological order
- Random access to record by search of whole file.
- What about modifying records?

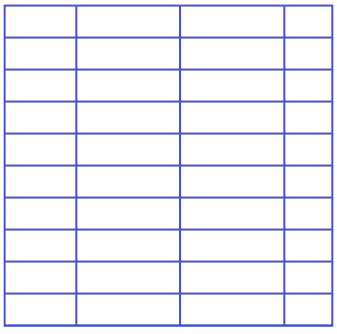


Pile File

Sequential File

- Fixed-format records
- Records often stored in order of *key field*.
- Good for applications that process all records.
- No adequate support for random access.
- What about adding new record?
- Separate pile file keeps *log file* or *transaction file*.

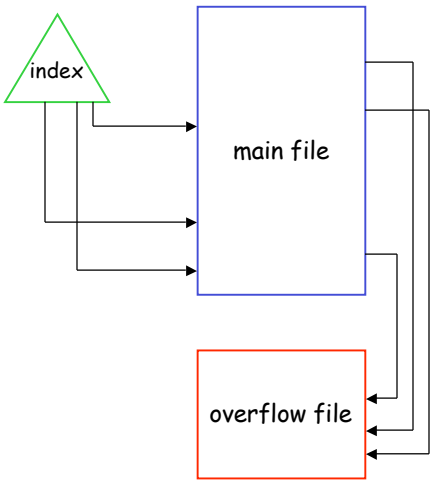
key field



Sequential File

Indexed Sequential File

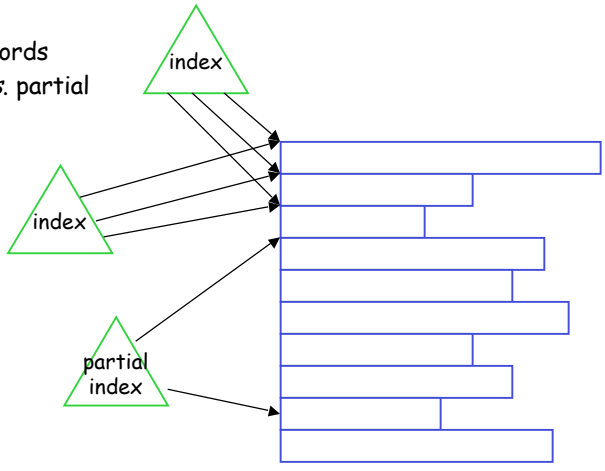
- Similar to sequential file, with two additions.
  - Index to file supports random access.
  - Overflow file indexed from main file.
- Record is added by appending it to overflow file and providing link from predecessor.



Indexed Sequential File

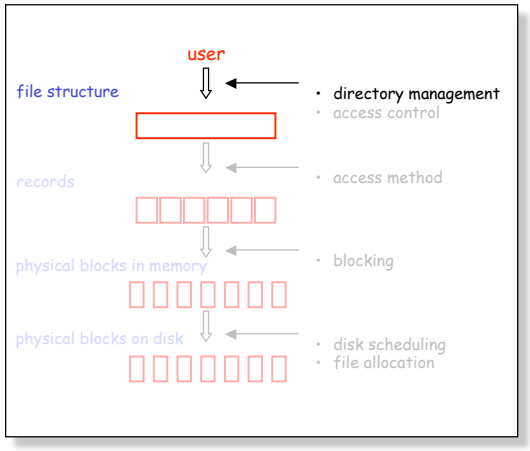
Indexed File

- Multiple indexes
- Variable-length records
- Exhaustive index *vs.* partial index



File Management

- What is a file?
- Elements of file management
- File organization
- Directories
- File allocation
- UNIX file system

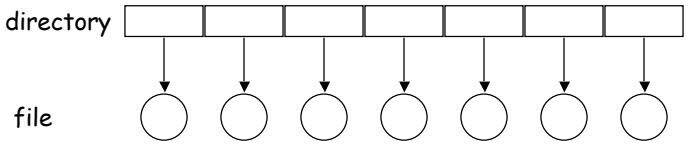


Directories

- Large amounts of data: Partition and structure for easier access.
- High-level structure:
  - *partitions* in MS-DOS
  - *minidisks* in MVS/VM
  - *file systems* in UNIX.
- Directories: Map file name to directory entry (basically a symbol table).
- Operations on directories:
  - search for file
  - create/delete file
  - rename file

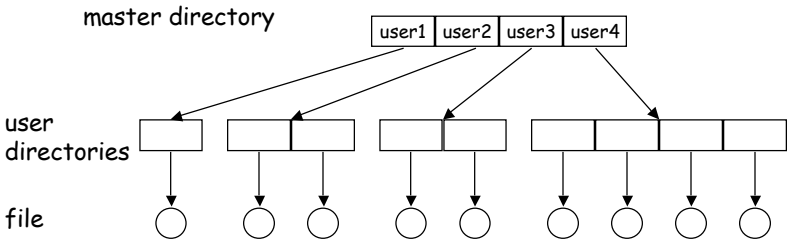
Directory Structures

- Single-level directory:



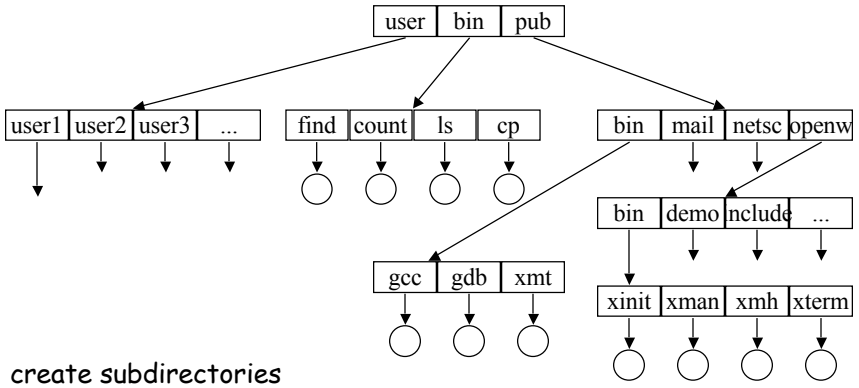
- Problems:
  - limited-length file names
  - multiple users

Two-Level Directories



- Path names
- Location of system files
  - special directory
  - search path

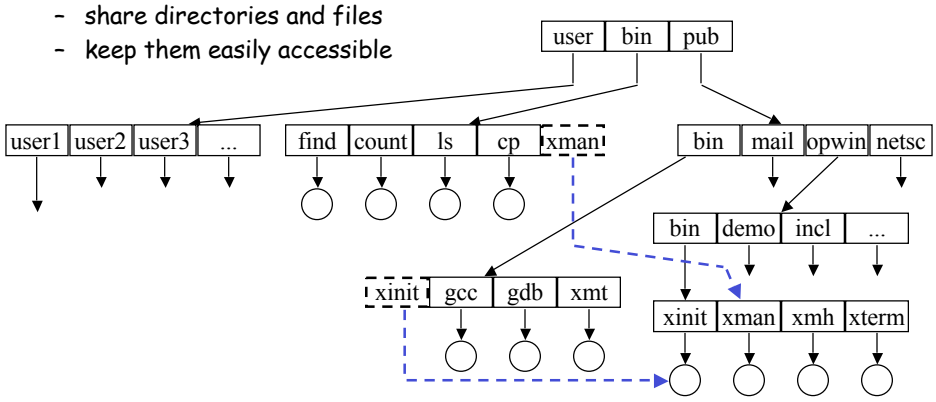
Tree-Structured Directories



- create subdirectories
- current directory
- path names: complete vs. relative

Generalized Tree Structures

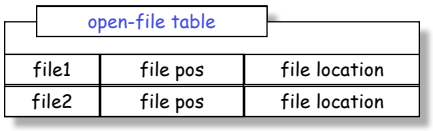
- share directories and files
- keep them easily accessible



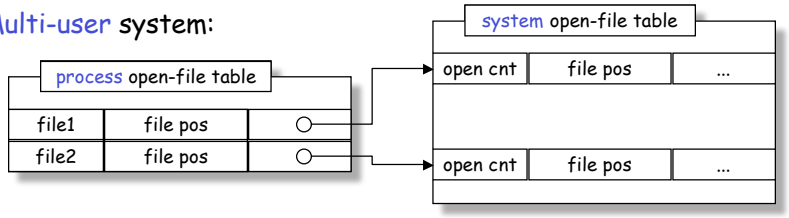
- Links: File name that, when referred, affects file to which it was linked. (hard links, symbolic links)
- Problems:
  - consistency, deletion
  - Why links to directories only allowed for system managers?

Bookkeeping

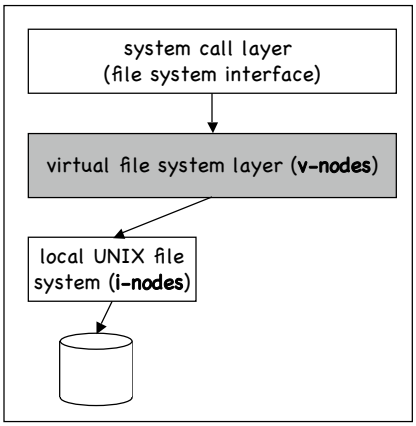
- **Open file** system call: cache information about file in kernel memory:
  - location of file on disk
  - file pointer for read/write
  - blocking information
- Single-user system:



- Multi-user system:



File System Architecture: Virtual File System



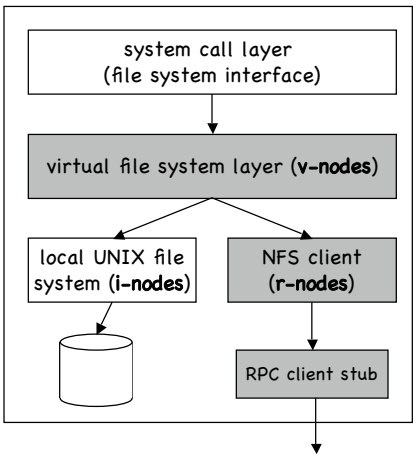
Example: **Linux Virtual File System** (VFS)

- Provides generic file-system interface (separates from implementation)
- Provides support for network-wide identifiers for files (needed for network file systems).

Objects in VFS:

- **inode objects** (individual files)
- **file objects** (open files)
- **superblock objects** (file systems)
- **dentry objects** (individual directory entries)

File System Architecture: Virtual File System



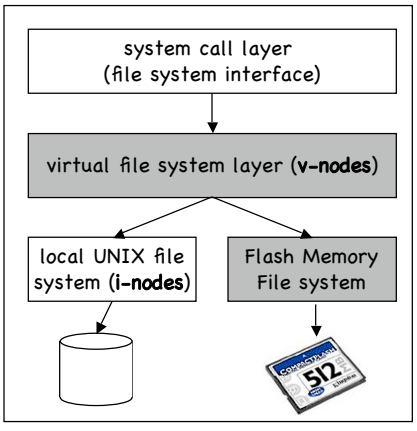
Example: Linux Virtual File System (VFS)

- Provides generic file-system interface (separates from implementation)
- Provides support for network-wide identifiers for files (needed for network file systems).

Objects in VFS:

- inode objects (individual files)
- file objects (open files)
- superblock objects (file systems)
- dentry objects (individual directory entries)

File System Architecture: Virtual File System



Example: Linux Virtual File System (VFS)

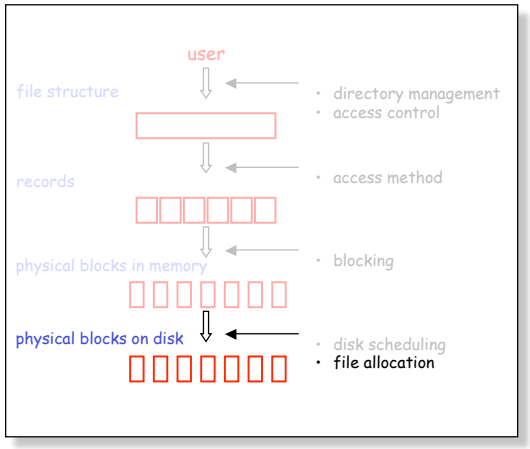
- Provides generic file-system interface (separates from implementation)
- Provides support for network-wide identifiers for files (needed for network file systems).

Objects in VFS:

- inode objects (individual files)
- file objects (open files)
- superblock objects (file systems)
- dentry objects (individual directory entries)

File Management

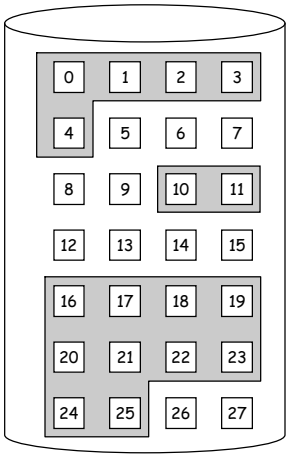
- What is a file?
- Elements of file management
- File organization
- Directories
- File allocation
- UNIX file system



Allocation Methods

- File systems manage disk resources
- Must allocate space so that
  - space on disk utilized effectively
  - file can be accessed quickly
- Typical allocation methods:
  - contiguous
  - linked
  - indexed
- Suitability of particular method depends on
  - storage device technology
  - access/usage patterns

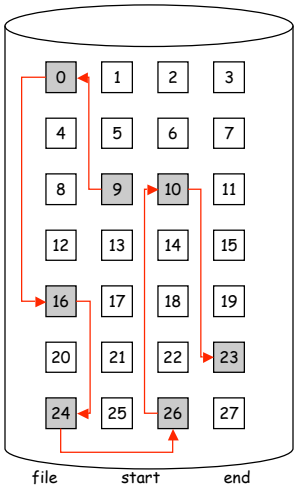
Contiguous Allocation



file	start	length
file1	0	5
file2	10	2
file3	16	10

- Logical file mapped onto a sequence of adjacent physical blocks.
- Advantages:
  - minimizes head movements
  - simplicity of both sequential and direct access.
  - Particularly applicable to applications where entire files are scanned.
- Disadvantages:
  - Inserting/Deleting records, or changing length of records difficult.
  - Size of file must be known *a priori*. (Solution: copy file to larger hole if exceeds allocated size.)
  - External fragmentation
  - Pre-allocation causes internal fragmentation

Linked Allocation

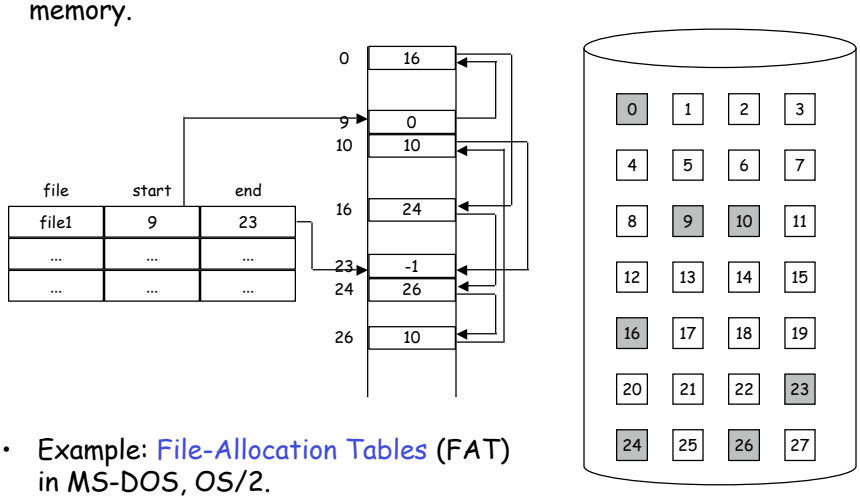


file	start	end
file 1	9	23
...	...	...
...	...	...

- Scatter logical blocks throughout secondary storage.
- Link each block to next one by forward pointer.
- May need a backward pointer for backspacing.
- Advantages:
  - blocks can be easily inserted or deleted
  - no upper limit on file size necessary *a priori*
  - size of individual records can easily change over time.
- Disadvantages:
  - direct access difficult and expensive
  - overhead required for pointers in blocks
  - reliability

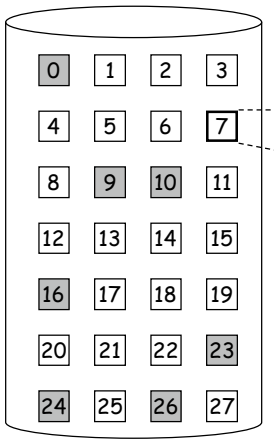
Variations of Linked Allocation

- Maintain all pointers as a separate linked list, preferably in main memory.

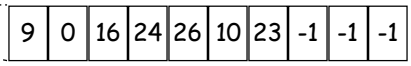


- Example: File-Allocation Tables (FAT) in MS-DOS, OS/2.

Indexed Allocation



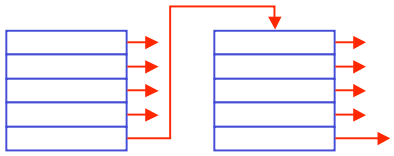
- Keep all pointers to blocks in one location: index block (one index block per file)



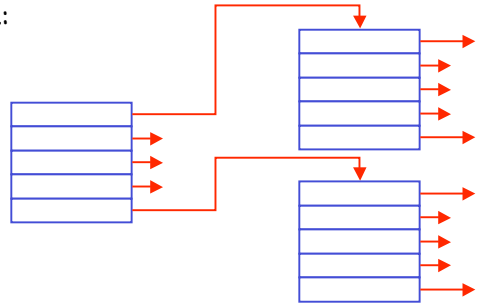
- Advantages:
  - supports direct access
  - no external fragmentation
  - therefore: combines best of continuous and linked allocation.
- Disadvantages:
  - internal fragmentation in index blocks
- Problem:
  - what is a good size for index block?
  - fragmentation *vs.* file length

Solutions for the Index-Block-Size Dilemma

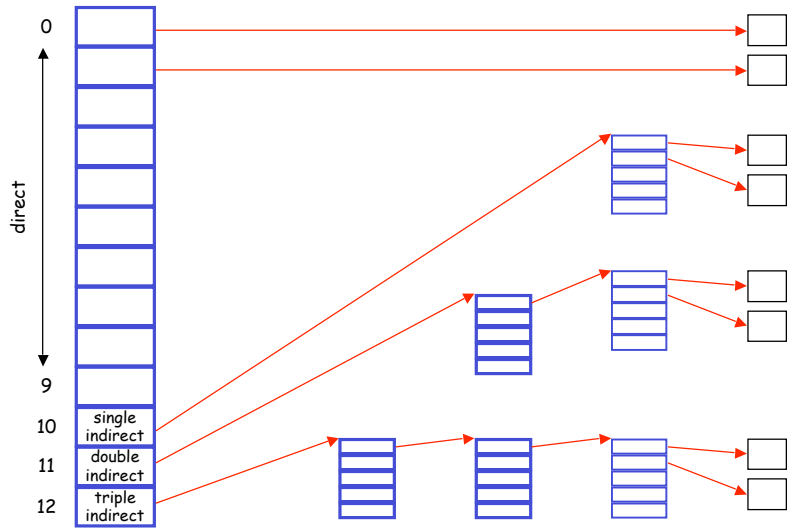
- Linked index blocks:



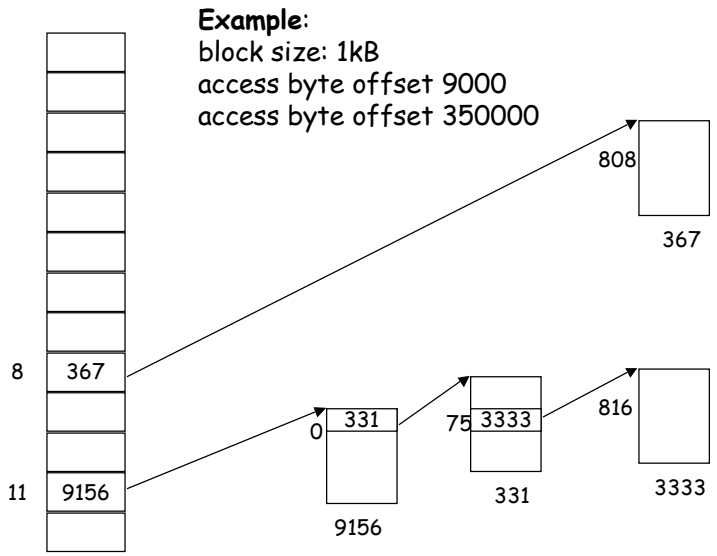
- Multilevel index scheme:



Index Block Scheme in UNIX

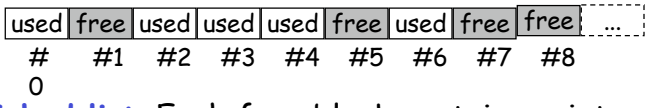


UNIX (System V) Allocation Scheme



Free Space Management

- Must keep track where unused blocks are.
- Can keep information for free space management in unused blocks.
- **Bit vector**:



- **Linked list**: Each free block contains pointer to next free block.
- Variations:
  - **Grouping**: Each block has more than on pointer to empty blocks.
  - **Counting**: Keep pointer of first free block and number of contiguous free blocks following it.