University College Dublin
An Coláiste Ollscoile, Baile Átha Cliath

Operating Systems
File Management

Dr. Vivek Nallur (vivek.nallur@ucd.ie)

File Management

- Why do we need File Management
  - Need for file systems
  - Files & directories, and operations on them
  - Management of a storage device's data and free space
  - Backup, recovery and file system integrity mechanisms

Introduction to file management

- A process may store information in its address space; issues:
  1. volatility (information lost when process terminates or corrupted when system crashes)
  2. information sharing with processes not overlapping its lifespan
  3. limited storage size (less of a problem with 64-bit addresses)

So how to solve this problem

- Create a file for a process to store information
  - named collection of data, normally residing on persistent secondary storage (disk, CD/DVD, solid-state drive, tape)
  - can be manipulated as a unit: open, close, create, destroy, copy, rename, list
  - individual data items within a file may be also manipulated:
    - read, write, update, insert, delete

Data Hierarchy

- Information stored in a system follows a data hierarchy

1. Lowest level: bits
2. Medium level: fixed-length patterns of bits
   - byte, word
   - character: mapping of bytes/words/groups of words to semantic symbols (ASCII, Unicode. . .)
   - field: group of characters
   - record: group of fields
   - file: group of related records
3. Highest level: file system or database

Data Hierarchy Cont.

- Any unit that can contain a file system can be called a Volume
  - This unit is normally a secondary storage system
  - physical vs logical volumes (groups of physical volumes)
  - If emulating older systems, this volume can be stored on in memory
  - In rare occasions a Volume can be created in memory called a RAM DISK

Files

- Files consist of one or more records (also called blocks)
  - physical record: minimal unit of information readable from/writable to a storage device (hardware)
  - logical record: amount of data treated as a unit of information (software)

Files example

- Unix and Windows files are just sequences of bytes physically grouped in blocks
  - byte: logical record
  - block of bytes: physical record

File characteristics

- Location: physically in storage device, or logically in file system’s structure
- size & type (executable, associated to an application, etc)
- accessibility: restrictions placed on data
  - Owner information
  - Read / Write / Execute status

File Systems

- A file system (FS) is the OS component responsible for:
  - file management: storage, reference, organisation, access,
  - sharing and security
  - file integrity mechanisms
  - access methods

File Characteristics

- device independence from the user's viewpoint: symbolic file names rather than physical file names (i.e. addresses in secondary storage device) are available to users (processes)
- recovery or backup capabilities
- optionally, cryptographic capabilities

Typical File Systems

A File System is typically concerned with secondary storage (e.g., disk, tape), but it can also deal with other media:

- tertiary storage: auxiliary storage (robot-operated tapes),tape storage is now in terms of 1000 TB so far larger than current generation hard drives
- primary storage: special (temporary) FSs in memory space

Directories

- To organise and quickly locate files, FSs use directories:
  - special files containing names and locations of other FS files
- Directory entries contain information fields such as:
  - file name
  - location: physical block address, or logical location of file in FS
  - size & type
  - accessed, modified and creation times

Directory Organisation

- Single-level (or flat) FS:
  - simplest FS organisation: all files stored in one directory
  - no two files can have the same name
  - FS must perform a linear search of the directory contents to
  - locate each file, which can lead to poor performance

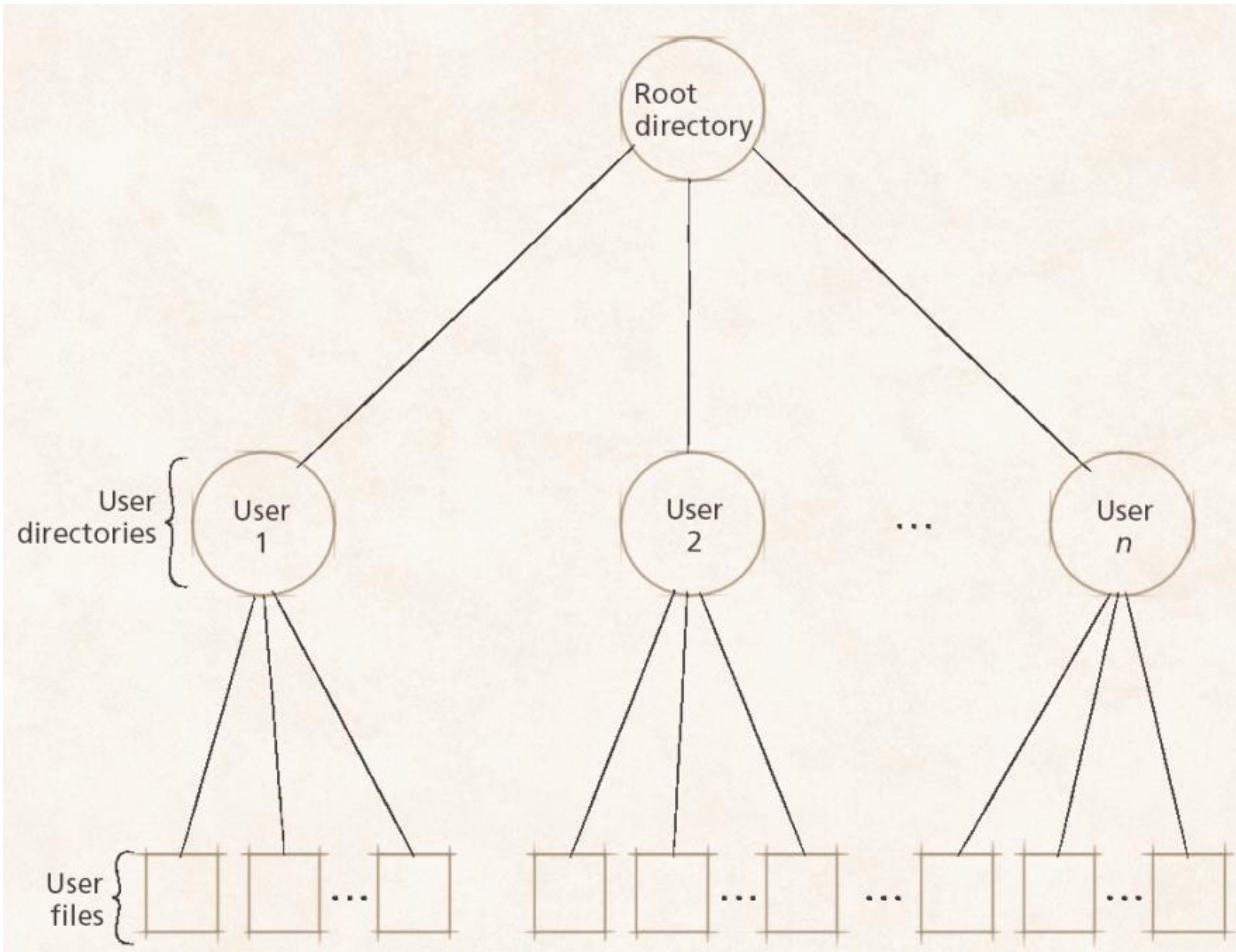
Hierarchical FS:

- nested structure:
  - root directory: start of the FS structure on the storage device
  - a directory may contain both files or other directories
  - any directory has a single parent directory

File names / Directory structure

- file names need to be unique only within a given directory
  - absolute path: file name including path name from the root
    - Unix Directory: /etc/modules/file.txt (unique)
    - Windows Directory: C:\users\admin\file.txt
  - relative path: relative to working directory, to simplify
  - navigation: ./file.txt (only necessarily unique within
  - working directory)

Directory Structure visualized



Links & Directories

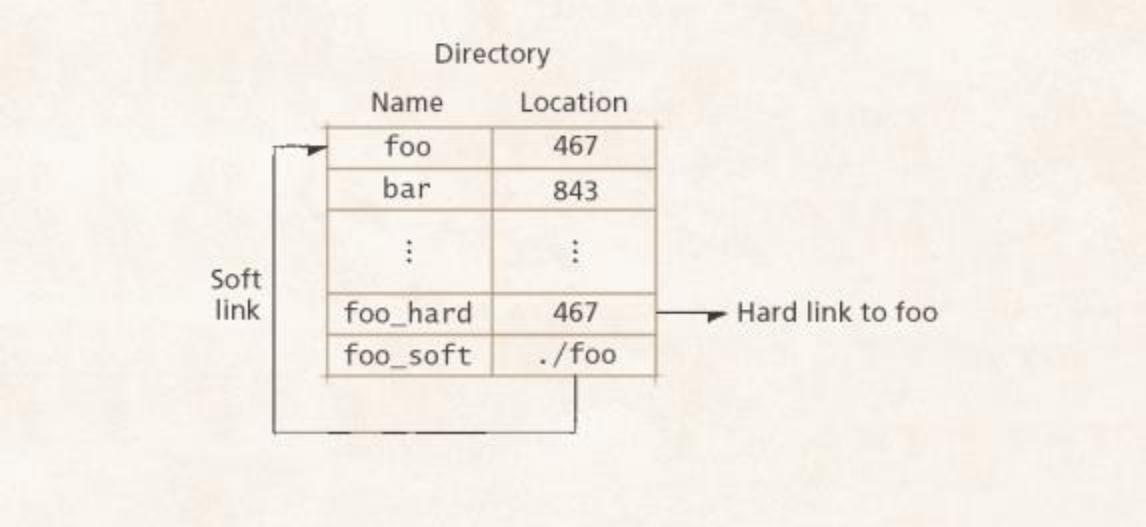
- Link: directory entry that references another directory or file (perhaps in a different directory)
- links facilitate data sharing and access
- links allow to convert hierarchical structures into
  - acyclical structures: a directory may have two different parents
  - cyclical structures

Links & Directories (cont.)

- What does a link contain?
  - soft link: the path name of a file (logical name)
  - hard link: the physical location of a file on the storage device (typically a block number)

Links

- Links under file location changes:
  - physical change (file physically moved on secondary storage device): hard links should be updated
  - logical change (file moved to different directory, or renamed): soft links should be updated



Metadata

- Metadata
  - information about the FS itself that enables its proper functioning and protects its integrity
  - it cannot be directly modified by users
  - Is used in cyber criminal investigations to track changes and to create a chain of evidence

Superblock

- Many FSs create a superblock to store critical metadata:
  - FS type unique identifier
  - location of root directory
  - number of blocks in FS
  - location of available free blocks on the secondary device
  - sanity check status, time of last modification. . .
- The superblock integrity is vital: to avoid data loss, most FSs
- keep redundant copies of it throughout the storage device

Metadata & File Descriptor

- File open operation returns a file descriptor (FD):
  - non-negative integer which indexes an open-file table in primary memory
  - further file access is directed through the FD
  - This open-file table also contains file attributes

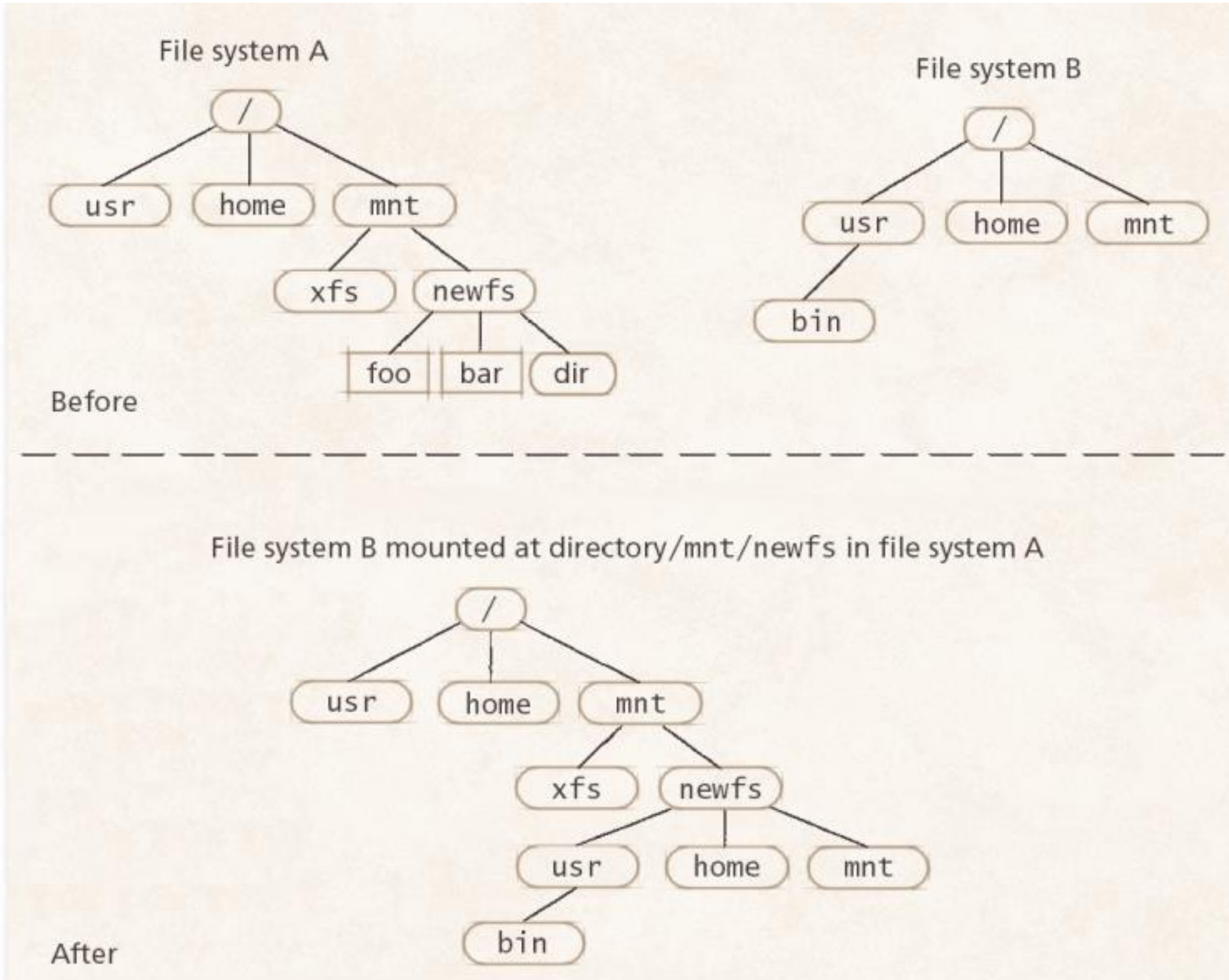
Open-file table :file attributes

- Highly system-dependent structures:
- Access control data (permissions)
- Symbolic name & location on secondary storage
- it provides fast access to information needed by FS to manage a file

Mounting

- Mount operation
  - logical combination of two or more FSs into one namespace
  - procedure: a directory in the native FS called a mount point is assigned to the root of the mounted FS
- FSs manage mounted directories through mount tables:
  - when the native FS encounters a mount point, the mount table is used to determine the device and type of the mounted FS
  - most OSs support multiple FS types for removable storage, such as UDF for DVDs and ISO 9660 for CDs
- Note:
  - soft links can be created to files in mounted FSs;
  - but: hard links cannot be created across FSs

Mounting in UNIX-compatible FSs (such as UFS, ext2,. . .)



File Organisation & Record Access

- Records of a file are arranged and accessed on secondary storage in two ways:
  1. sequential (tape): records physically placed and processed in sequential order
  2. direct or random (disk): records are read or written in any suitable order

Record Access Criteria

- Important Criteria for access method
  - rapid access
  - ease of update and maintenance
  - reliability, economy of storage
- These criteria may conflict or vary in importance; examples:
  - ease of update is irrelevant in a CD-ROM
  - faster access may reduce the effective storage space (e.g. indices)

File Allocation

- Allocating and freeing space on secondary storage resembles the same issues on primary storage
- however, unlike in primary storage, physical proximity of related information is very important for performance
- the disk head has to physically move to access data, and disk access can be up to 10^6 times slower than memory access

File Allocation

- Earlier contiguous allocation systems have generally been replaced by more dynamic noncontiguous allocation systems:
  - files tend to grow or shrink over time
  - users rarely know in advance how large their files will be
- Allocation is based on the division of secondary memory into equal-sized blocks (physical records)
  - block size is usually a multiple of page/frame size for efficiency reasons

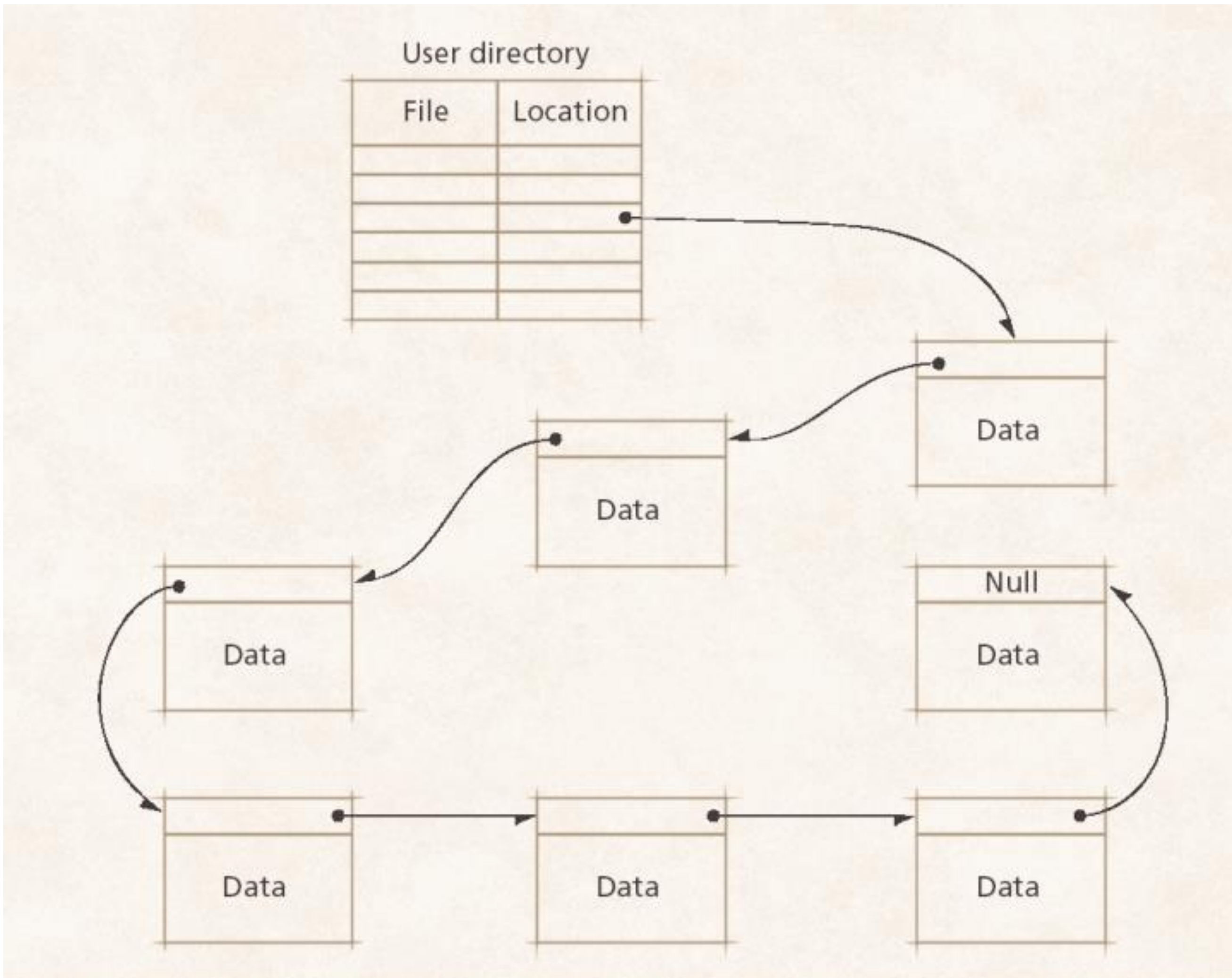
Contiguous File Allocation

- A file occupies a set of contiguous blocks on the storage device
- Advantages
  - physical adjacency of successive blocks is good for speed (a disk head does not normally need to move from block b to b + 1)
- Disadvantages
  - free hole selection issue; strategies: first fit, best fit
  - external fragmentation issue: as files are allocated and deleted free space is broken into little pieces
  - performance loss due to additional I/O operations:
    - example: if a file grows and no contiguous free blocks are available, the whole file must be transferred to a new area of adequate size

Linked Noncontiguous File Allocation

- Blocks of a same file form a linked list (sometimes, doubly linked)
- A directory entry points to the first block of a file
- Blocks divided into two portions
  - pointer portion: points to file's next block
  - data portion: stores file content
- It solves several issues of continuous allocation:
  - no external fragmentation: no compaction needed as long as there are free blocks
  - quick insertion and deletion: we only need to modify the pointer in the previous block

Linked Noncontiguous File Allocation



Linked Noncontiguous File Allocation

- Disadvantages:
  - Pointers require space
  - Block size issues

Linked Noncontiguous File Allocation

- Disadvantages:
  - Pointers require space
    - if block is 512 bytes and disk address (pointer) takes 4 bytes, user sees blocks of 508 bytes, and 0.78 % of the disk is used for pointers and not for data
  - Block size issues

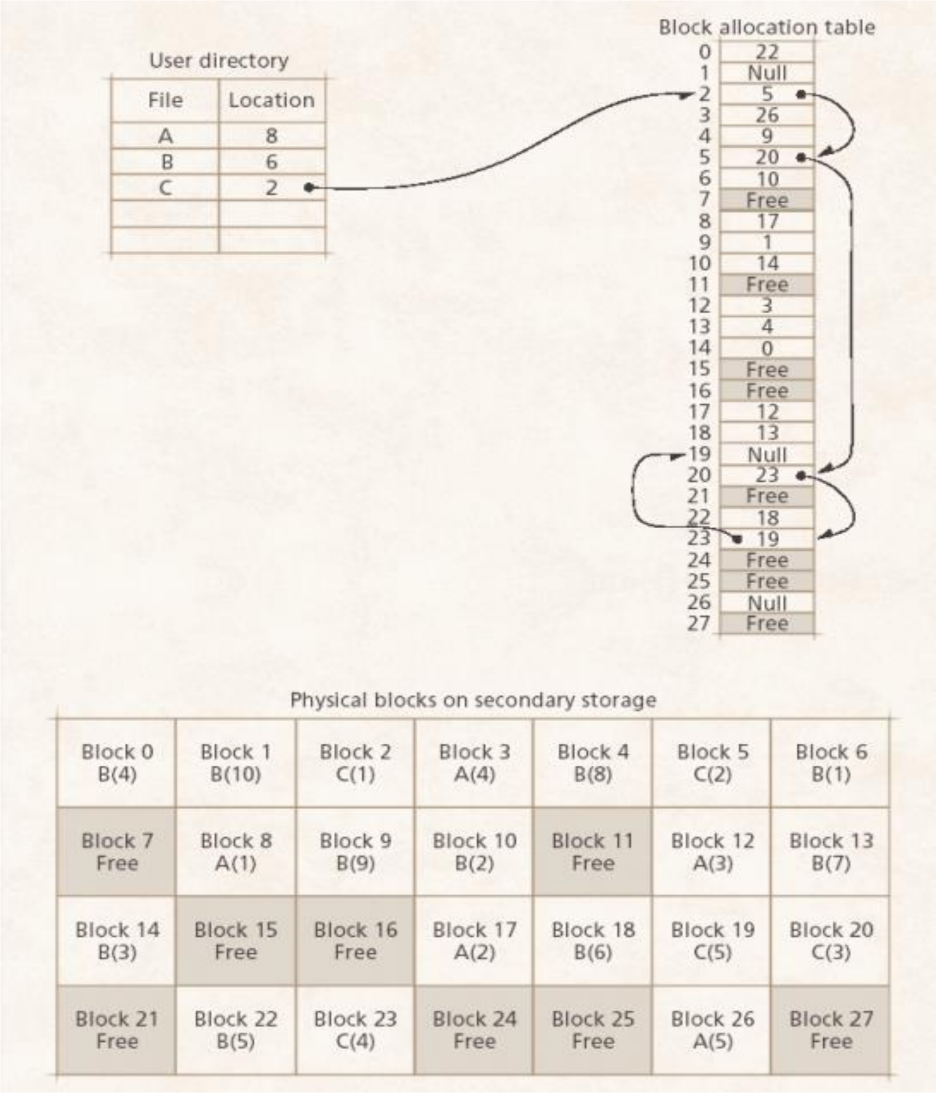
Linked Noncontiguous File Allocation

- Disadvantages:
  - Pointers require space
  - Block size issues
    - if large: there may be significant internal fragmentation
    - if small: file data may end up spread across multiple blocks dispersed throughout the storage device
      - in order to find the i-th block, the linked list must be searched from the beginning
      - this may lead to poor performance

Tabular Noncontiguous File Allocation

- Some FSs store pointers to file blocks in a central location, common to all files: block allocation table (BAT)
- Variation of the linked list approach
- A directory entry for a file points to its first BAT entry
- Features:
  - an entry in the BAT has two meanings:
    1. block number in the disk
    2. next entry in the BAT
  - the last block is marked by Null

Tabular Noncontiguous File Allocation



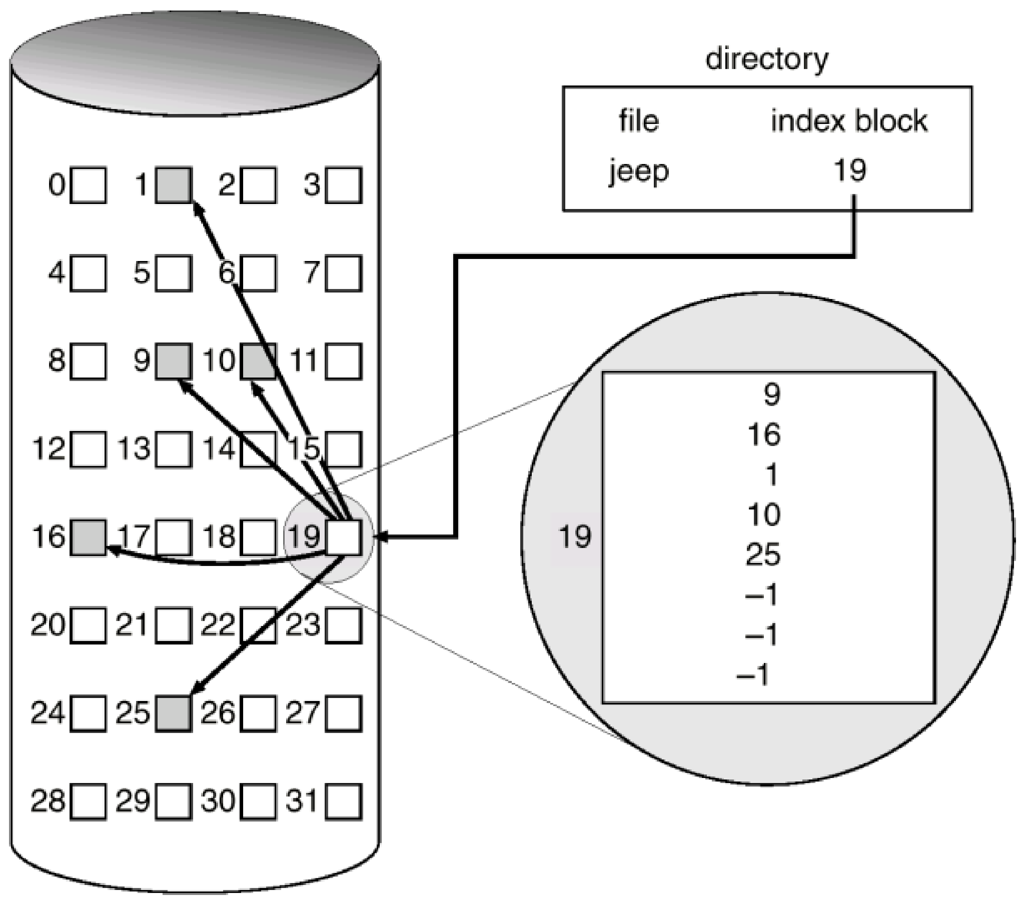
Tabular Noncontiguous File Allocation

- Increased performance with respect to linked noncontiguous allocation if BAT can be represented with a few blocks
  - physical blocks are used to its full extent
  - random access to a file’s block is faster
- Issues:
  - FS might still need to follow many pointers to reach a block
  - BAT can be cached in memory to improve access times, but this is inefficient; example:
    - 1000 GB disk and 4 KB block size → 250 million entries
  - we could exploit virtual memory, but a large BAT would generate lots of page faults
- System used in DOS and OS/2 (FAT, file allocation table)

Indexed Noncontiguous File Allocation

- In order to allow direct access, each file has an index block
  - a file's directory entry points to its index block
  - this contains a list of pointers to the file's data blocks
  - index blocks are typically cached in main memory when a file is open
- For handling large files, the index block may also point to additional index blocks (chaining)

Indexed Noncontiguous File Allocation

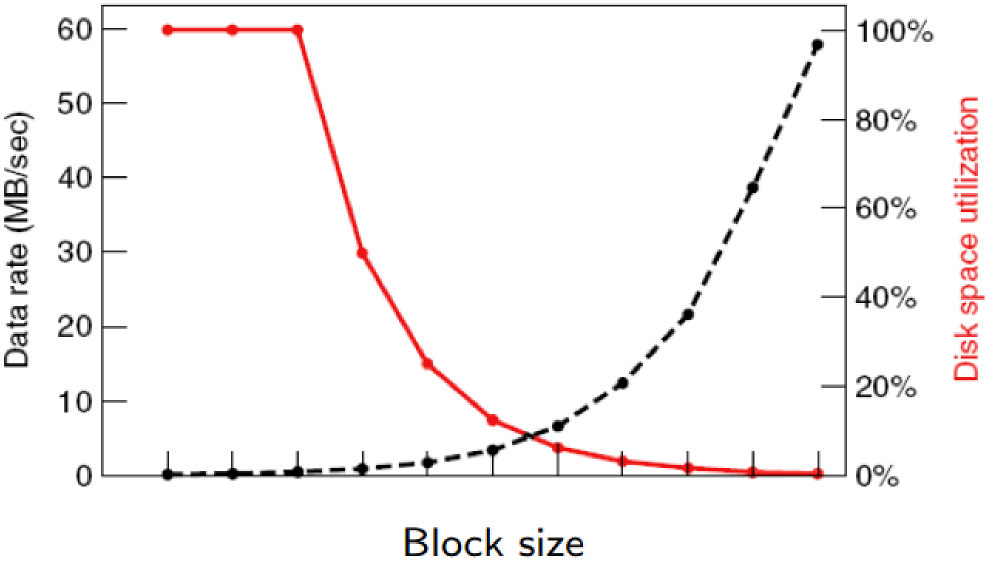


Indexed Noncontiguous File Allocation

- Advantages:
  - index nodes need only be in memory when the file is open
    - the amount of memory needed is proportional to the number of open files, not to the disk size (as with BAT)
  - as with BAT, block searching may take place in the index
  - blocks themselves
- Disadvantages:
  - index blocks waste more space than pointer overhead in linked lists (example: consider a file using only one or two blocks)
  - scattered data blocks still decrease performance: index blocks must be placed near the data blocks they reference
- Index blocks are used in UNIX-type FSs (called inodes) and in
- Microsoft’s NTFS

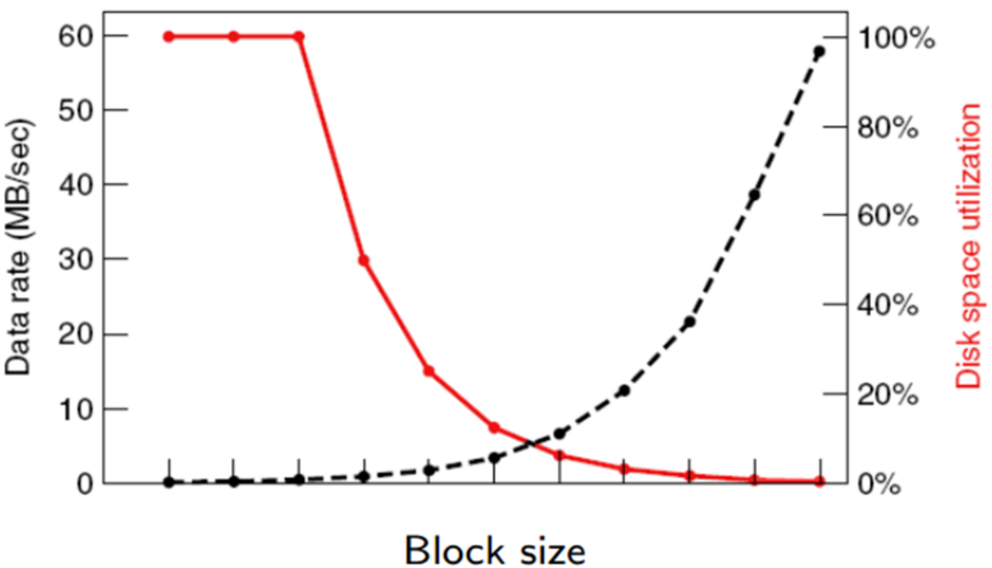
Optimal Block Size

- Block size determines performance in more than one way:



Optimal Block Size

- The plot below is experimental and system dependent, but the same pattern will occur in any system; some figures
  - median size of files in Unix environments ≈ 1 KB
  - disk transfer speeds: 800 MB/s – 1 GB/s
- It is necessary to strike a balance when choosing the block size



Free Space Management: Free List

- FS must keep track of free space for new files—including the reuse of space from deleted files
- Free list technique:
  - linked list of blocks, each one containing as many locations of free blocks as will fit

Free List cont.

- Advantage: low overhead to perform maintenance operations; two pointers suffice:
  - free blocks are allocated from the beginning of the list
  - newly freed blocks are appended to the end of the list
- Disadvantage: files are likely to be allocated in scattered blocks
  - resulting file access time will increase

Free Space Management: Bitmap

- Bitmap technique: a bitmap where each bit represents a block in memory
  - i-th bit corresponds to the i-th block, holding a 1 if it is free
  - on the storage device
  - example: if the free blocks are 2, 3, 4, 5, 8, 9, 10, 11, 12, 13, 17, 18, 25, 26 and 27, the free space bitmap is 001111001111110001100000011100000. . .

Bitmap cont.

- Advantages over free lists:
  - more compact
  - quick determination of contiguous blocks
- Disadvantage:
  - the FS may need to search the entire bitmap to find a free block (execution overhead)

Data Integrity Protection

- Data storage systems should be fault tolerant:
  - they must account for the possibility of losing critical information stored in the system, by providing techniques to recover from them
  - some scenarios that may lead to loss of data: system crashes, natural disasters, malicious programs
- Techniques involved in the preservation of the integrity of persistent data:
  - backup & recovery
  - data & FS consistency

Backup & Recovery

- Backup: periodically store redundant copies of information
- Recovery: restore most recent backup data after system failure
- Types of backups
  1. physical:
     - duplicate a storage device’s data at the bit level
     - it cannot be restored in different FS architectures or even in hard drives different from the original
  2. logical:
     - duplicate a file system’s data relying on its logical structure and using a standard archival format (often compressed)
     - it allows for incremental backups

Data Consistency

- Information written to FS after a backup is still vulnerable
  - backups cannot be done too often
- If system failure occurs during a write operation, file data may be left in an inconsistent state; examples:
  - index block which points at corrupt blocks
  - block which belongs to a file but it is in the list of free blocks
- Atomicity:
- perform a group of operations (transaction) in its entirety or
  - not at all
  - if an error occurs that prevents a transaction from completing,
  - the system stays in the state before the transaction began

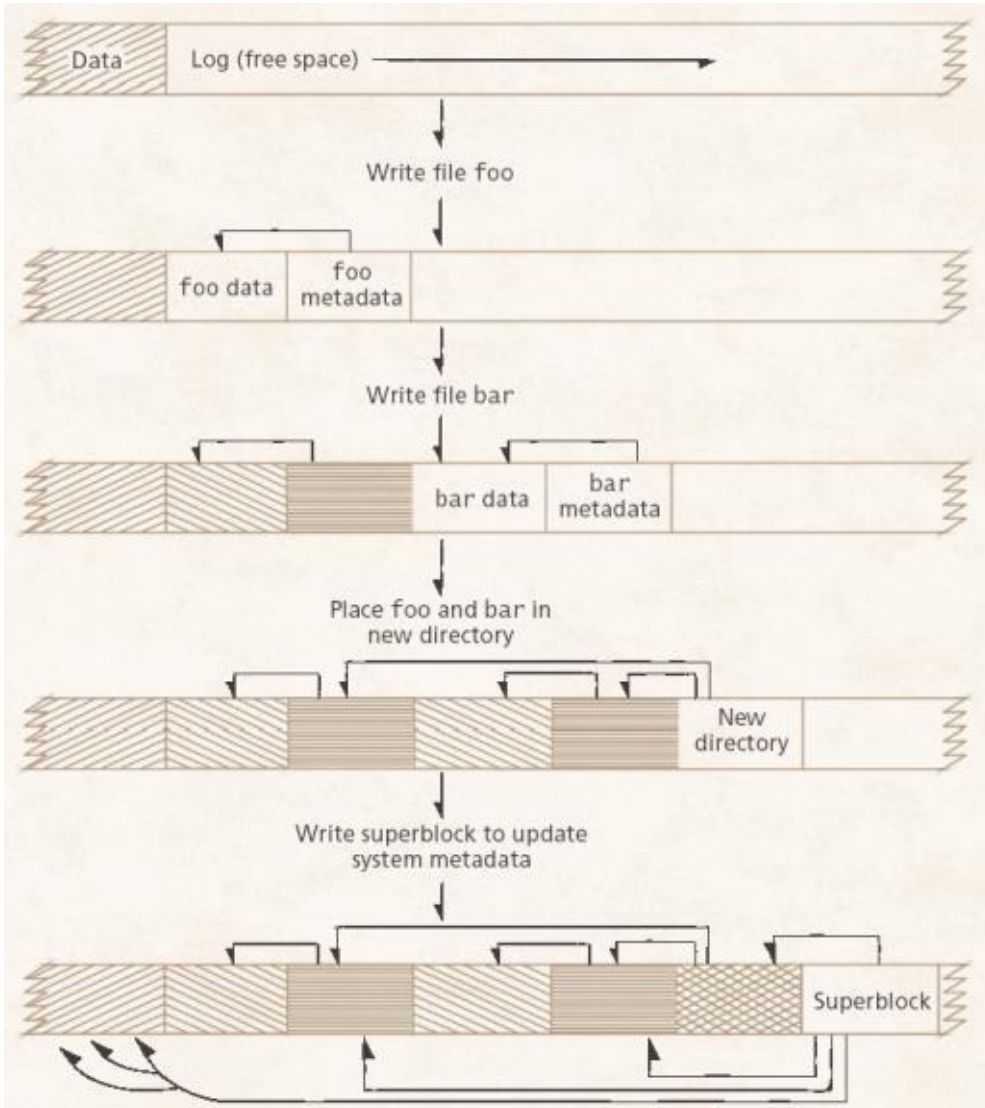
Shadow Paging & Logging

- Techniques for data consistency by means of atomic transactions:
- Shadow paging: modified data is written to a free block, instead of the original block
  - if the transaction completes, file metadata (index block, etc) is updated
- Logging: the result of each operation is recorded in a log file
    - once the transaction has completed, it is committed by
    - recording a special value in the log
    - usually, in journalling FSs only metadata operations are logged to minimise log size

Log-Structured File Systems

- Log-structured file systems (LFS)
  - all LFS operations are carried out as logged transactions
  - the entire disk acts as a circular log, at the end of which new data is sequentially written
  - consistency is guaranteed because the metadata is only written after the data, so that it can never reference invalid blocks

Log-Structured File Systems



Log-Structured File Systems (II)

- File modification in a LFS: a new version of the file and its updated metadata are written at the end of the log
- Modified directories, index blocks and superblocks (index blocks maps) are also written at the end of the log
  - issue: the entire log might have to be traversed to locate the
  - superblocks which point at the most up-to-date blocks of a file
  - to avoid this, metadata are cached in memory for quicker access
- still a special checkpoint region is needed, which contains the position of all superblocks
  - checkpoint region: fixed position (start of disk), not often written
  - at boot time, this enables the OS to quickly locate and cache metadata

Log-Structured File Systems (III)

- Consequences of sequential writes at the end of the log:
- advantage:
  - writes are very fast (consider write speed in noncontiguous file allocation)
- disadvantages:
  - log can fill up quickly (disk space is not infinite)
  - log can become fragmented, due to deleted and modified files

- Log-Structures useful for Flash storage , as they make fewer in place writes , and thus extend the life of the storage , as flash storage has a lower limit of rewrites compared to traditional magnetic storage.
- Journaling is used in many O/S for current transactions that have not been committed for example Reiser4 , this is a form of lite Log-structured file system
- Log-structured concept can be seen in other CS ideas like version control

Next week

- Study Time
  - Review Chapter 10