# CGS 3763: Operating System Concepts Spring 2006

## Storage Management – Part 1


Instructor : Mark Llewellyn
markl@cs.ucf.edu
CSB 242, 823-2790
http://www.cs.ucf.edu/courses/cgs3763/spr2006

School of Electrical Engineering and Computer Science
University of Central Florida




# Background

- For most users, the file system is the most visible aspect of an OS.
- It provides the mechanism for on-line storage of and access to both data and programs of the OS and all the users of the computer system.
- The file system consists of two distinct parts: a collection of files, each storing related data, and a directory structure, which organizes and provides information about all the files in the system.
- Files are mapped by the OS onto physical devices, which are typically nonvolatile.



# The File Concept

- A file is a named collection of related information that is recorded on secondary storage.
- From a user's perspective, a file is the smallest allotment of logical secondary storage; that is, data cannot be written to secondary storage unless they are within a file.
- A file is viewed as a contiguous logical address space.
- Types:
  - Data
    - numeric
    - character
    - binary
  - Program
- The information in a file is defined by its creator.



# File Structure

- A file has a certain structure, which depends on its type.
- None - sequence of words, bytes (text files)
- Simple record structure
  - Lines
  - Fixed length
  - Variable length
- Complex Structures
  - Formatted document
  - Relocatable load file
- Can simulate last two with first method by inserting appropriate control characters.
- Who decides:
  - Operating system
  - Program



# File Attributes

- Name – only information kept in human-readable form.
- Identifier – unique tag (number) identifies file within file system.
- 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 – data for protection, security, and usage monitoring.
- Information about files are kept in the directory structure, which is maintained on the disk.



# File Operations

- A file is an abstract data type. To define a file properly, we need to consider the operations that can be performed on files.
- The OS will provide system calls to:
- Create – two steps are necessary to create a file. First, space in the file system must be found for the file. Second, an entry for the new file must be made in the directory.
- Write – writing to a file requires a system call specifying both the name of the file and the information to be written to the file. Given the name of the file, the system searches the directory to find the file's location. The system must maintain a write pointer to the location in the file where the next write it to take place. The write pointer must be updated after each write to the file.



# File Operations (cont.)

- Read – reading from a file requires a system call that specifies the name of the file and where (in memory) the next block of the file should be put. Again, the directory is searched for the associated entry, and the system must maintain a read pointer to the location in the file where the next read is to occur. Once the read has taken place, the read pointer must be updated. Because a process is usually either reading from or writing to a file, the current operation location can be maintained as a per-process current-file-position-pointer. Both the read and write operation use this same pointer, saving space and reducing system complexity.
- Reposition within file – the directory is searched for the appropriate file, and the current-file-position-pointer is repositioned to a specified value. Repositioning with a file need not involve any actual I/O. This file operation is also known as a file seek.



# File Operations (cont.)

- Delete – to delete a file requires searching the directory for the named file. Having found the associated directory entry, the file space is released, so that it can be utilized by other files and the directory entry is erased.
- Truncate – this situation arises when a user wishes to delete the contents of a file without destroying the attributes of the file. Rather than forcing the user to delete the file and then recreate it, this capability allows the attributes to remain unchanged – except for the file length which is reset to 0 and the file space is released.
- *Open($F_i$)* – search the directory structure on disk for entry $F_i$, and move the content of entry to memory, The OS maintains a small table, called the open-file table, which contains information about all open files.
- *Close ($F_i$)* – move the content of entry $F_i$ in memory to directory structure on disk.



# Handling Open Files

- Several pieces of data are needed to manage 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 access information.
  - Access rights: per-process access mode information.
- Some operating systems provide facilities for locking an open file (or portions of an open file). File locks allow one process to lock a file and prevent other processes from gaining access to it. This is useful in environments where files can be shared by several processes.



# Open File Locking

- File locks are utilized to provide synchronization for file access.
- Most systems provide two types of file locks.
  - Shared locks – several processes can simultaneously hold a shared lock on the same file. Used for read access only.
  - Exclusive locks – at most one process can hold an exclusive lock on a file at any given time. Used for write access to a file.
- Locks are used to mediates access to a file.
- The OS may provide either mandatory or advisory file locking mechanisms:
  - Mandatory – access is denied depending on locks held and requested.
  - Advisory – processes can find status of locks and decide what to do.



# Some Common File Types – Name, Extension

| 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, pas, asm, a | source code in various languages |
| batch | bat, sh | commands to the command interpreter |
| text | txt, doc | textual data, documents |
| word processor | wp, tex, rtf, doc | various word-processor formats |
| library | lib, a, so, dll | libraries of routines for programmers |
| print or view | ps, pdf, jpg | ASCII or binary file in a format for printing or viewing |
| archive | arc, zip, tar | related files grouped into one file, sometimes compressed, for archiving or storage |
| multimedia | mpeg, mov, rm, mp3, avi | binary file containing audio or A/V information |



# Access Methods

- When the information in a file is to be accessed and stored into the main memory for use by a user process, the file must be accessed.
- The information in a file can be accessed in several different ways. Some systems provide only one access method for files. Other systems, support many different access methods.
- Choosing the correct access method for a file for a particular application is a major design issue.



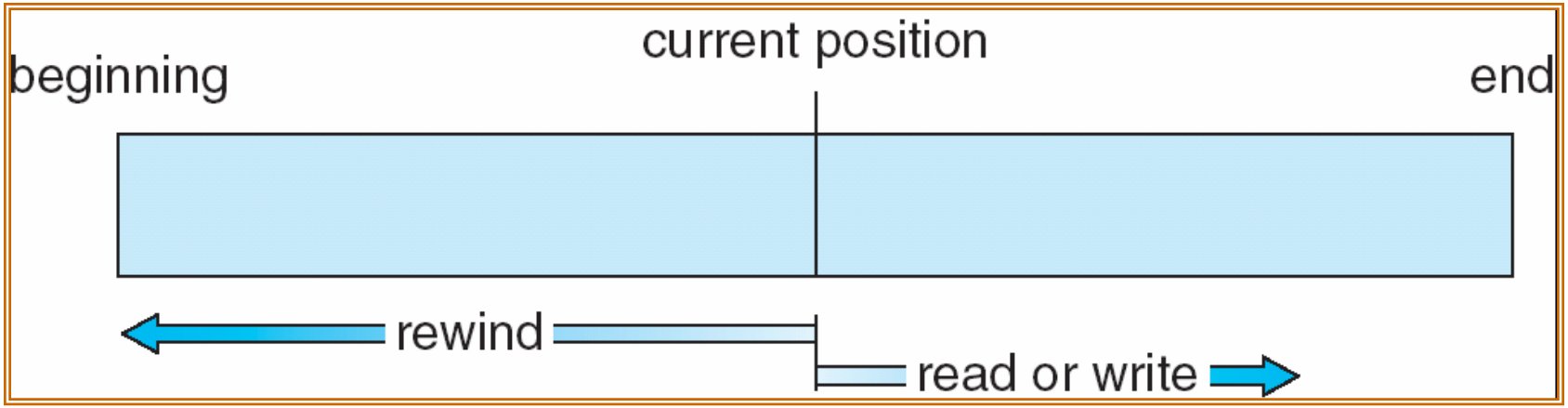
# Sequential Access Method

- The simplest access method is sequential access.
- Information in the file is processed in order, one record after another.
- This is by far the most common mode of access. Most editors and compilers access files in this fashion.
- Reads and writes make up the bulk of the operations on a file.
- A read operation – read next – reads the next portion of the file and automatically advances a file pointer, which tracks the I/O location.



# Sequential Access Method

- Similarly, a write operation – write next – appends to the end of the file and advances to the end of the newly written material (the new end of the file).
- Such a file can be reset to the beginning; and on some systems, a program may be able to skip forward or backward *n* records.
- Sequential access is based on a tape model of a file and works as well on sequential access devices as it does on random access devices.






# Direct Access Method

- Another method is direct access (sometimes referred to as relative access).
- A file made up of fixed-length logical records that allow programs to read and write records rapidly and in no particular order.
- The direct access model is based on a disk model of a file, since disks allow random access to any file block.
- For direct access, the file is viewed as a numbered sequence of blocks or records.
  - Thus, a program might read block 14, then read block 436, and then write block 16.
- There are no restrictions on the order of reading or writing for a direct file.



# Direct Access Method (cont.)

- Direct access files are useful for immediate access to large amounts of information.
- Database files are often of this type. When a query concerning a particular subject arrives, the address of the block which contains the result is calculated and then that block is read directly to provide the desired information.
- The direct access method requires that the file operations be modified to include the block number as a parameter.
  - Thus, we have *read n*, where n is the block number, rather than *read next* as with sequential access.
- An alternative approach is to retain the original read next operation, but to add an operation position to n. (See next page.)
  - Thus, a read operation would first position to n, then read next.



# Simulation of Sequential Access on a Direct Access File

| sequential access | implementation for direct access |
|---|---|
| *reset* | $cp = 0;$ |
| *read next* | *read cp*;<br>$cp = cp + 1;$ |
| *write next* | *write cp*;<br>$cp = cp + 1;$ |



# Comparison of Sequential and Direct Access Methods

- **Sequential Access**

  read next
  write next
  reset
  no read after last write
  (rewrite)

- **Direct Access**

  read *n*
  write *n*
  position to *n*
  read next
  write next
  rewrite *n*

  where *n* = relative block number

# Other Access Methods

- Other access methods can be built on top of the direct access method.
- Typically, these methods involve the construction of an index for the file.
- The index contains pointers to the various blocks of a file. To find a specific record, first the index is searched and then once the value is found in the index, the pointer is used to access the file directly to find the desired record.

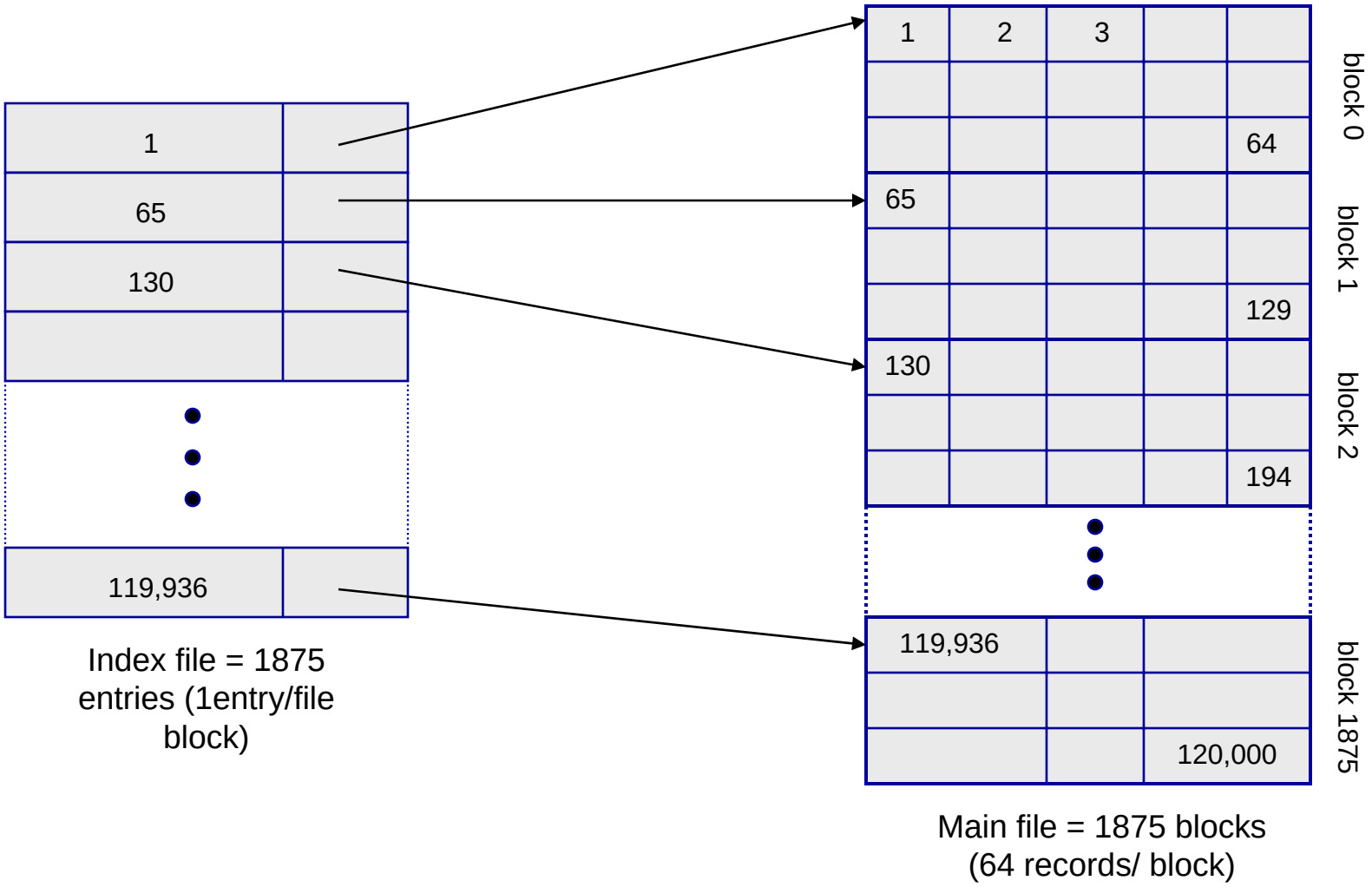


# Other Access Methods (cont.)

- For example, a retail-price file might list the UPCs for items, with the associated prices.
- Lets assume that each record consists of a 10-digit UPC and a 6-digit price, thus each record requires 16 bytes.
- Supposing our disk has 1,024 bytes per block; we can store 64 records/block.
- A file of 120,000 records would occupy $1.92 \times 10^6$ bytes (about 2 million bytes) and require 1875 blocks.
- By keeping the file sorted by UPC, we can define an index consisting of the first UPC in each block. This index would have 1875 entries of 10 bytes each, or 18,750 bytes, and as such, could be maintained in main memory.
- To find a particular item, we could perform a binary search of the index and from this search determine the exact block which contains the desired item.



# Other Access Methods (cont.)






# Directory Structure

- The file system of a computer can be extensive.
- Some systems will maintain millions of files on terabytes of disk.
- To manage all of this data, they must be organized in some fashion. This organization involves the use of directories.
- Although a disk can be used in its entirely to hold a file system, it is often desirable to use parts of a disk for a file system and other parts for other things, such as swap space, or unformatted (raw) space.
- These parts of the disk are known variously as partitions, slices, or minidisks. These various parts can also be combined to create larger structures known as volumes.
- For now, we'll simply refer to a chunk of disk storage that holds a file system as a volume.

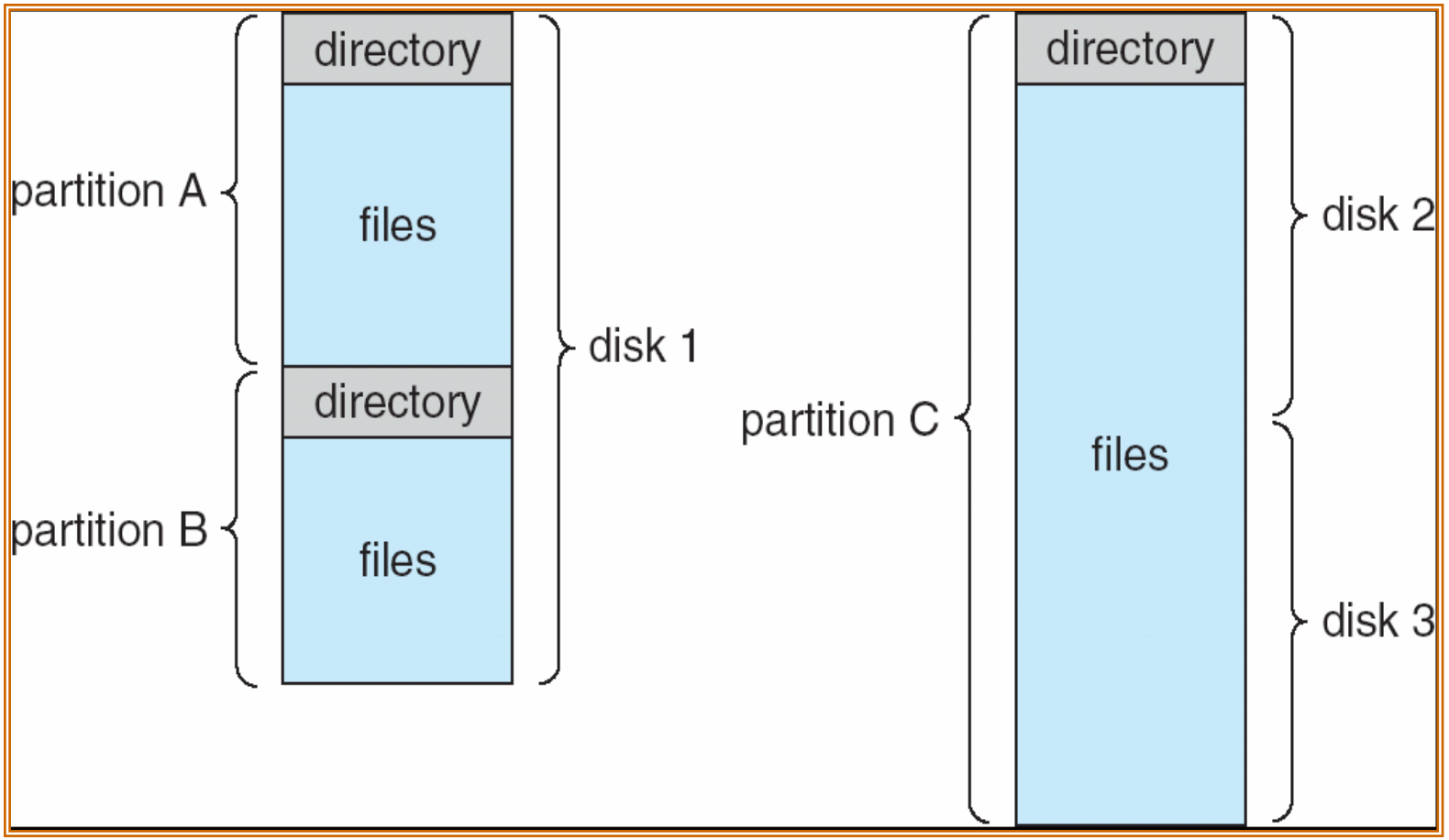


# Directory Structure (cont.)

- Each volume can be thought of as a virtual disk.
- Volumes can also store multiple operating systems, allowing a system to boot and run more than one OS.
- Each volume that contains a file system must also contain information about the files in the system. This information is maintained in a device directory or volume table of contents.
- The device directory, typically referred to simply as the directory, records information – such as name, location, size, and type – for all the files on that volume.
- The illustration on the next page shows a typical file system organization.



# A Typical File-system Organization






# Directory Overview

- The directory can be viewed as a symbol table that translates file names into their directory entries. This allows us to organize the directory in several different ways.
- When considering a particular directory structure, we need to keep in mind the operations that will be performed on a directory:
  - Search for a file – since files have symbolic names and similar names may indicate a relationship between files, we may want to be able to find all files whose names match a particular pattern.
  - Create a file – new files need to be created and added to the directory.
  - Delete a file – when a file is no longer needed, it needs to be removed from the directory.
  - List a directory – list the files in a directory and the contents of each entry.
  - Rename a file – users often change name of a file which may reposition it within the directory structure.
  - Traverse the file system – access every directory and every file within a directory structure. Often used in backing-up the system.



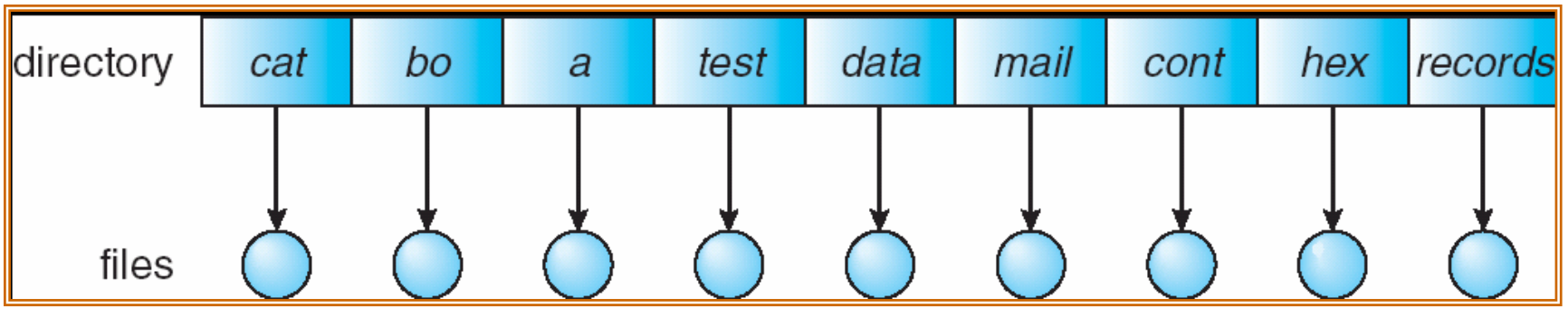
# Organize the Directory (Logically)

- Efficiency – locating a file quickly
- Naming – convenient to users
  - Two users can have same name for different files
  - The same file can have several different names
- Grouping – logical grouping of files by properties, (e.g., all Java programs, all games, …)



# Single-Level Directory

- The simplest directory structure is the single-level directory.
- All files are contained in the same directory, which is easy to support and understand.





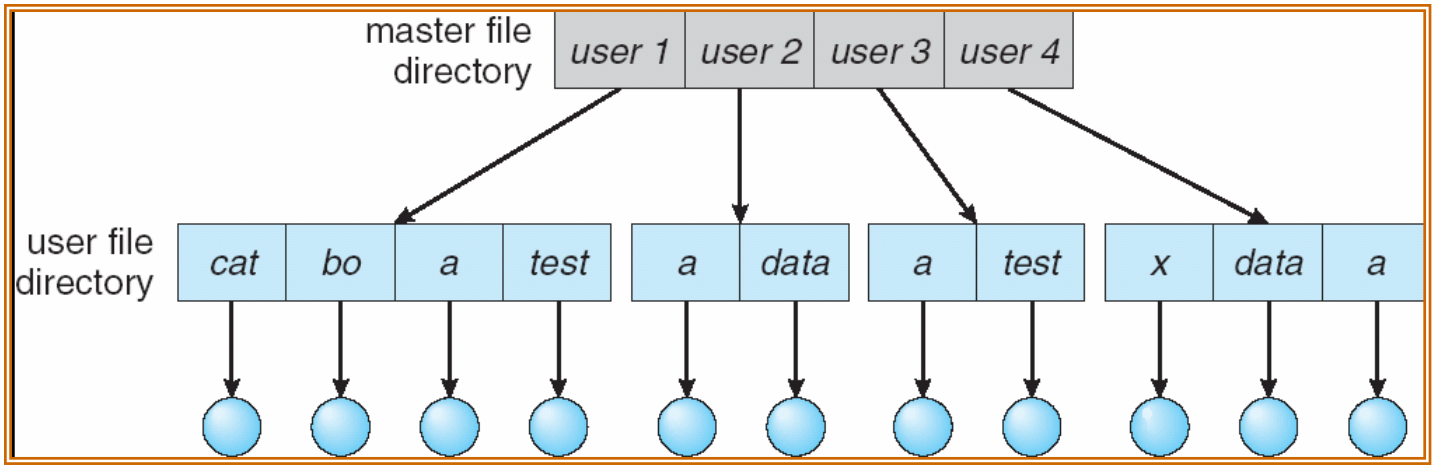

# Single-Level Directory (cont.)

- The single-level directory has serious limitations, however, when the number of files increases or when the system has more than one user.
- Since all files are contained in the same directory, they must have unique names.
  - Two different users cannot call their files "test". Think about a programming class where maybe all 50 students would call their project something like "program 2"! This is called file name collision.
- Even a single user on a single-level directory may find it difficult to remember the names of all the files as the number of files increases. It is not uncommon for a single user to have several hundred files or more on one computer system.



# Two-Level Directory

- Since the single-level directory leads to file name collision among different users, the standard solution is to create a separate directory for each user.
- In the two-level directory structure, each user has their own user file directory (UFD). When a user job starts or a user logs in, the system's master file directory (MFD) is searched. The MFD is indexed by username or account number and each entry points to the UFD for that user.






# Two-Level Directory (cont.)

- Although the two-level directory structure solves the name-collision problem, it still has disadvantages.
- This structure effectively isolates one user from another.
- Isolation is an advantage when the users are completely independent but is a disadvantage when the users want to cooperate on some task and to access one another's files.
  - Some systems simply do not allow local user files to be accessed by other users!
- If access is to be permitted, one user must have the ability to specify a file in another user's directory. To specify a particular file uniquely in a two-level directory, you must specify both the user name and the file name.



# Two-Level Directory (cont.)

- A two-level directory can be thought of as a tree (trees are inverted in computer science), of height 2.
- The root of the tree is the MFD and its children are the UFDs. The children of the UFDs are the files themselves. The files are the leaves of the tree.
- Specifying a user name and a file name defines a path in the tree from the root (the MFD) to a leaf (the specified file).
- Thus, a user name and a file name define a path name.
- Every file in the system has a path name.
- To name a file uniquely, a user must know the path name of the desired file.