Announcements

- Reading Chapter 11
- Look for update on Web Site
  - Mem.c

File Abstraction

- What is a file?
  - A named collection of information stored on secondary storage
- Properties of a file
  - non-volatile
  - can read, read, or update it
  - has meta-data to describe attributes of the file
- File Attributes
  - name: a way to describe the file
  - type: some information about what is stored in the file
  - location: how to find the file on disk
  - size: number of bytes
  - protection: access control
    - may be different for read, write, execute, append, etc.
  - time: access, modification, creation
  - version: how many times has the file changed

File Operations

- Files are an abstract data type
  - interface (this lecture)
  - implementation (next lecture)
- create a file
  - assign it a name
  - check permissions
- open
  - check permissions
  - check that the file exists
  - lock the file (if we don’t what to permit other users a the same time)

File Operations (cont)

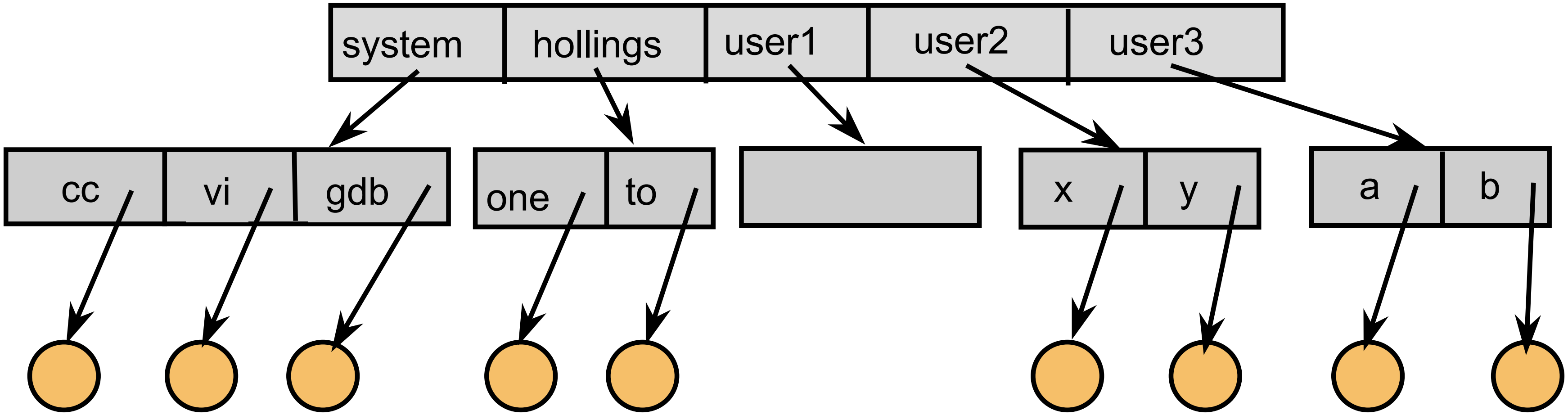
- write
  - indicate what file to write (either name of handle)
  - provide data to write
  - specify where to write the data within the file
    - generally this is implicit (file pointer)
    - could be explicit (direct access)
- read
  - indicate what file to read (either name of handle)
  - provide place to put information read
  - indicate how much to read
  - specify where to write the data within the file
    - generally this is implicit (file pointer)
    - could be explicit (direct access)
- fsync (synchronize disk version with in-core version)
  - ensure any previous writes to the file are stored on disk

File Operations (cont)

- seek
  - move the implicit file pointer to a new offset in the file
- delete
  - remove named file
- truncate
  - remove the data in the file from the current position to end
- close
  - unlock the file (if open locked it)
  - update meta data about time
  - free system resources (file descriptors, buffers)
- read meta data
  - get file size, time, owner, etc.
- update meta data
  - change file size, time owner, etc.

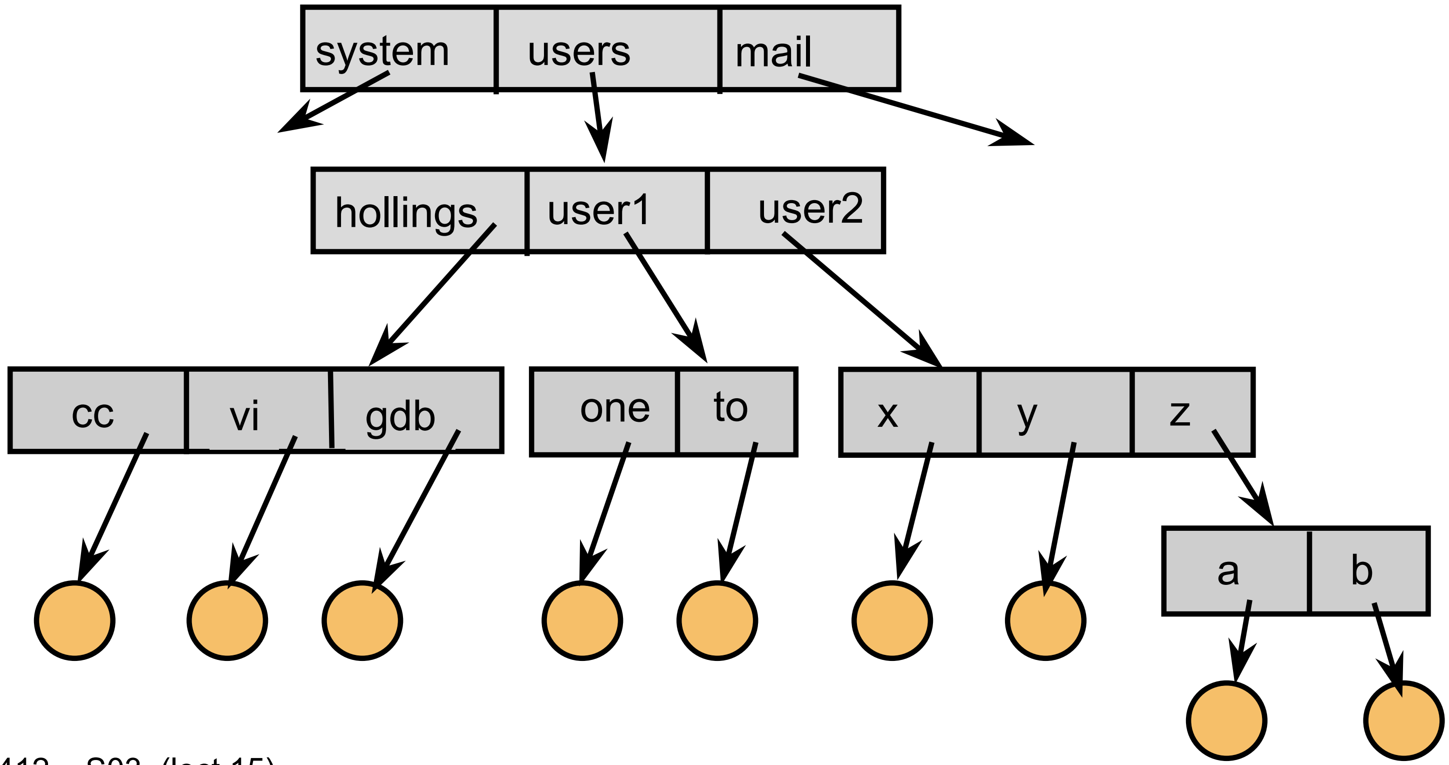
Simple Directory Structures

- One directory
  - Having all of the files in one name space is awkward
  - lots of files to sort through
  - different users would have to coordinate file names
  - each file has to have a unique name
- Two level directory
  - top level is users
  - second level is files per user



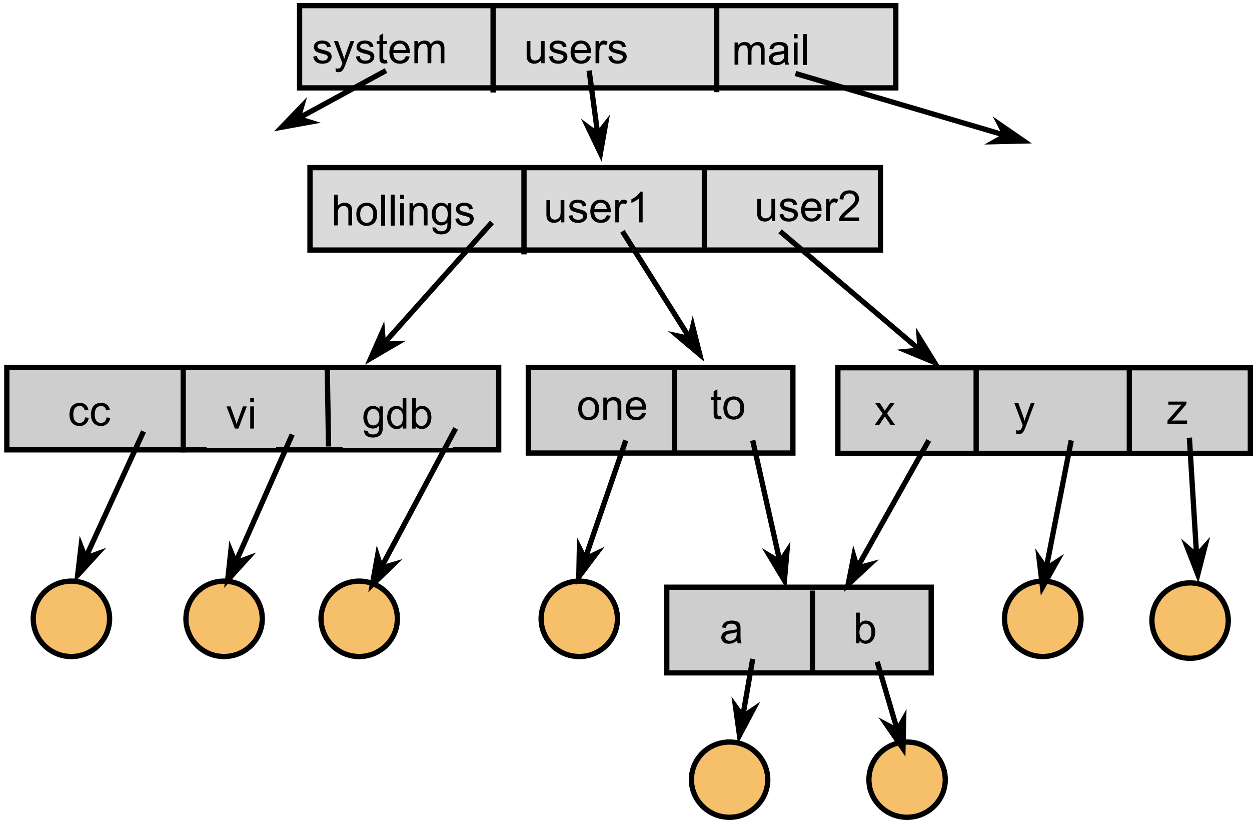
Tree Directories

- create a tree of files
- each directory can contain files or directory entries
- each process has a current directory
  - can name files relative to that directory
  - can change directories as needed



Acylic Graph Directories

- Permit users to share subdirectories



Issues for Acylic Graph Directories

- Same file may have several names
  - absolute path name is different, but the file is the same
  - similar to variable aliases in programming languages
- Deletion
  - if one user deletes a file does it vanish for other users?
    - yes, it should since the directory is shared
  - what if one user deletes their entry for the shared directory
    - no, only the last user to delete it should delete it
    - maintain a reference count to the file
- Programs to walk the DAG need to be aware
  - disk usage utilities
  - backup utilities

Does the OS know what is stored in a file?

- needs to know about some types of files
  - directories
  - executables
- should other file types be visible to the OS?
  - Example: word processing file vs. spreadsheet
  - Advantages:
    - OS knows what application to run
    - Automatic make (tops-20)
      - if source changed, re-compile before running
  - Problems:
    - to add new type, need to extend OS
    - OS vs. application features are blurred
    - what if a file is several types
      - consider a compressed postscript file

Example of File Types

- Macintosh
  - has a file type that is part of file meta-data
  - also has an application associated with each file type
- Windows 95/NT
  - has a file type in the extension of the file name
  - has a table (per user) to map extensions to applications
- Unix
  - can use last part of filename like an extension
  - applications can decide what (if anything) to do with it

File Protection

- How to give access to some users and not others?
- Access types:
  - read, write, execute, append, delete, list
  - rename: often based on protection of directory
  - copy: usually the same as read
- Degree of control
  - access lists
    - list for each user for each file the permitted operations
  - groups
    - enumerate users in a list called a group
    - provide same protection to all members of the group
    - depending on system:
      - files may be in one or many groups
      - users may be in one or many groups
  - per file passwords (tedious and a security problem)

File Protection Example (UNIX)

- each file has three classifications
  - user: the user who owns the file
  - group: a named group of other users
  - world: all others
- each file has three access types:
  - read, write, execute
- directory protection
  - read: list the files in the sub dir
  - write: delete or create a file
  - execute: see the attributes of the files in the subdir
  - sticky bit: can only modify directory entries owned by yourself

Unix File Protection (cont)

- Files have 12 bits of protection
  - 9 bits are user, group, and world for:
    - read: list the files in the sub dir
    - write: delete or create a file
    - execute: see the attributes of the files in the subdir
  - sticky bit: leave executable in memory after is done
  - setuid: run the program with the uid of the file’s owner
    - used to provide extra privilege to some processes
      - example: passwd command
  - setgid: run the program with the group id of the file’s owner