# Announcements

- Should be done with identity mapping on P4
- Reading Chapter 11 (8th ed)

# 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 want to permit other users at the same time)

# 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 and 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)

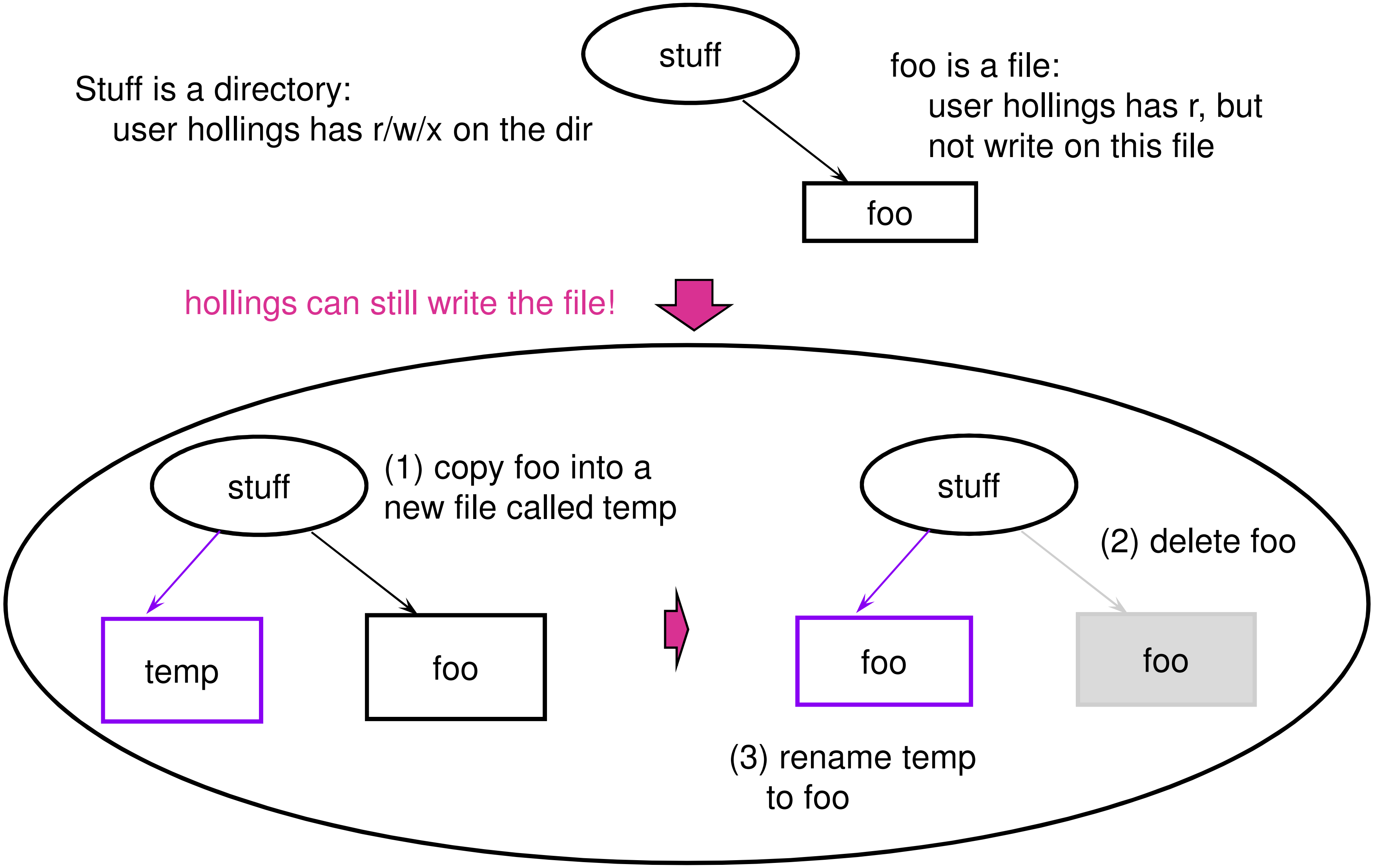
# File Protection Example (UNIX)

- 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: contents can only be modified by root user, folder owner, or file owner

# 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: contents can only be modified by root user, folder owner, or file owner
  - 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

# UNIX File Protection Example



Stuff is a directory:
user hollings has r/w/x on the dir

stuff

foo is a file:
user hollings has r, but
not write on this file

foo

hollings can still write the file!

stuff

(1) copy foo into a
new file called temp

temp

foo

stuff

(2) delete foo

foo

foo

(3) rename temp
to foo

# File Protection Example (AFS)

- Each Directory has an ACL
  - protection information applies to all files in a directory
  - file access types are:
    - lookup, insert, delete, administer, read, write, lock (k)
  - an ACL may be for a user or a group
  - ACL may contain negative rights
    - everyone but Joe Smith may read this file
- Groups
  - are collections of users
  - each user can create up to a fixed number of groups
    - users can administer their own groups
- Cells
  - collections of computers (e.g., csic, wam)

# File Operations (cont)

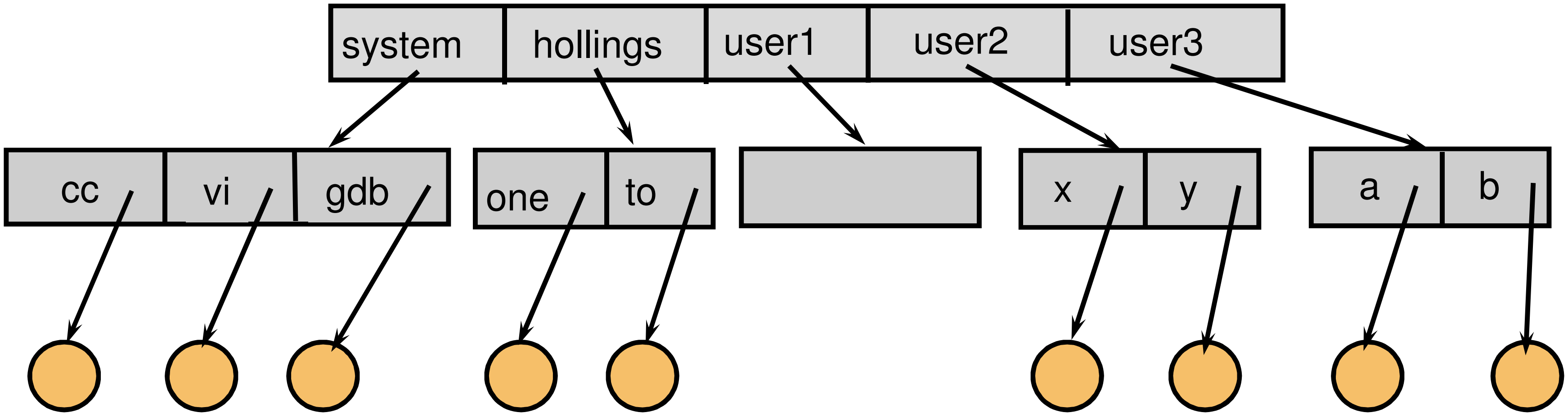
- write
  - indicate what file to write (either name or handle)
  - provide data to write
  - specify where to write the data within the file
    - generally this is implicit (file pointer)
    - could be explicit (direct access)
- read
  - indicate what file to read (either name or handle)
  - provide place to put information read
  - indicate how much to read
  - specify where to write the data within the file
    - usually implicit (sequential access via 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 metadata about time
  - free system resources (file descriptors, buffers)
- read metadata
  - get file size, time, owner, etc.
- update metadata
  - change file size, time owner, etc.

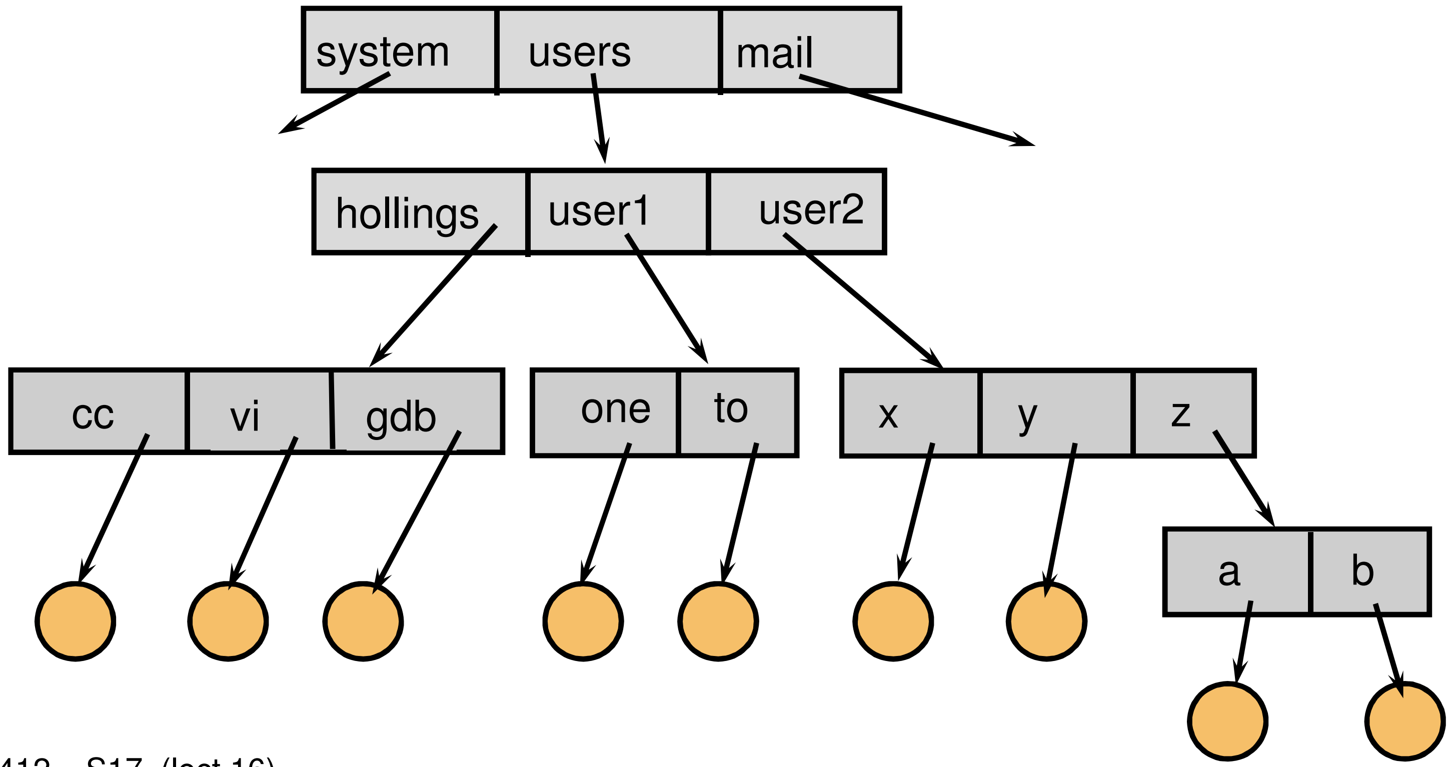
# Simple Directory Structures

- One directory
  - having all of the files in one namespace is awkward
  - lots of files to sort through
  - users 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



# OS Folder Structures (Unix)

- / (root)
  - bin *(system executables)*
  - etc *(system-wide settings)*
  - home
    - hollings
    - lam
  - lib *(shared object libraries)*
  - mnt
    - usbdrive
  - opt *(third-party software)*
  - proc *(virtual – info about processes)*
  - usr
    - bin *(applications)*
    - lib *(libraries)*
  - var *(files that change often)*

# OS Folder Structures (Mac)

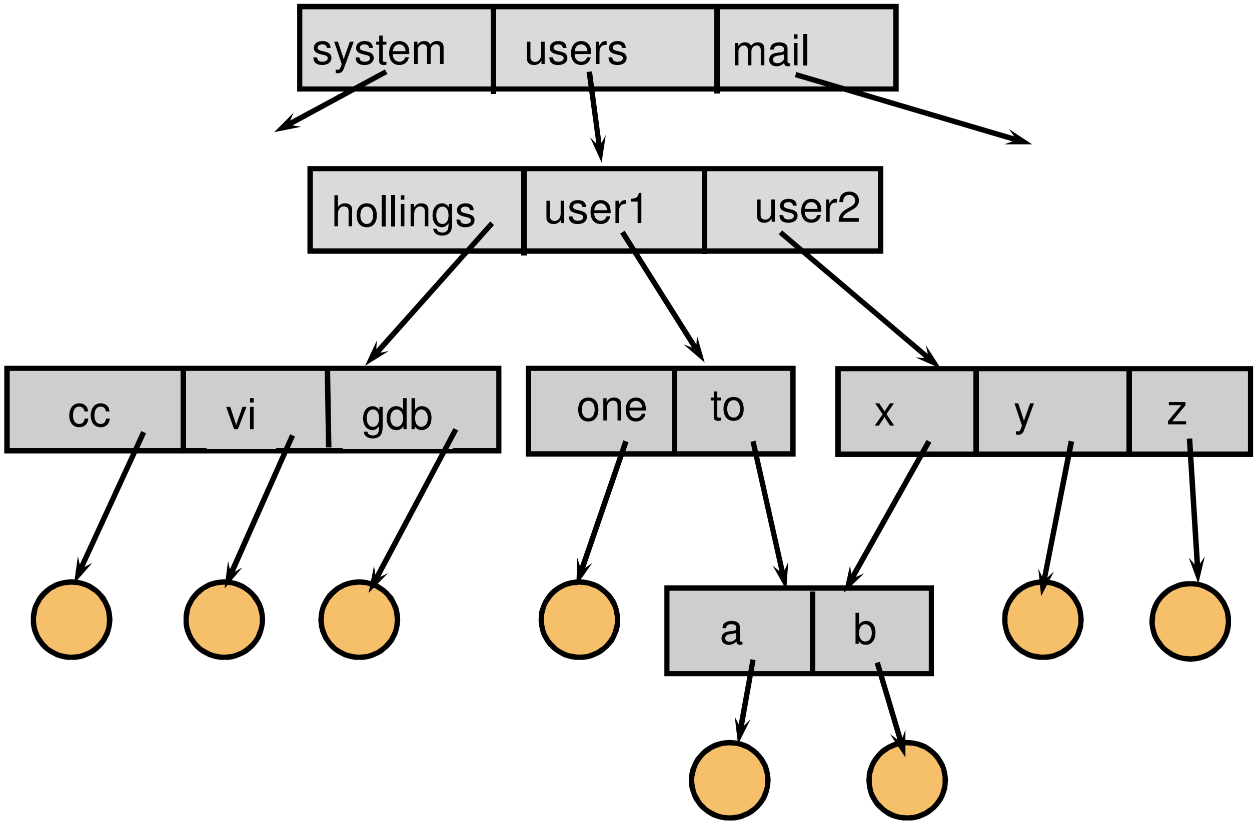
- / (root)
  - Applications
  - Library *(settings and shared object files)*
  - Users
    - hollings
    - lam
  - Volumes
    - usbdrive
  - bin
  - etc
  - opt
  - usr
  - var

# OS Folder Structures (Windows)

- C:\
  - Program Files
  - Users (previously “Documents and Settings”)
    - Hollingsworth
    - Lam
  - Windows
- D:\
  - usbdrive files

# Acylic Graph Directories

- Permit users to share subdirectories

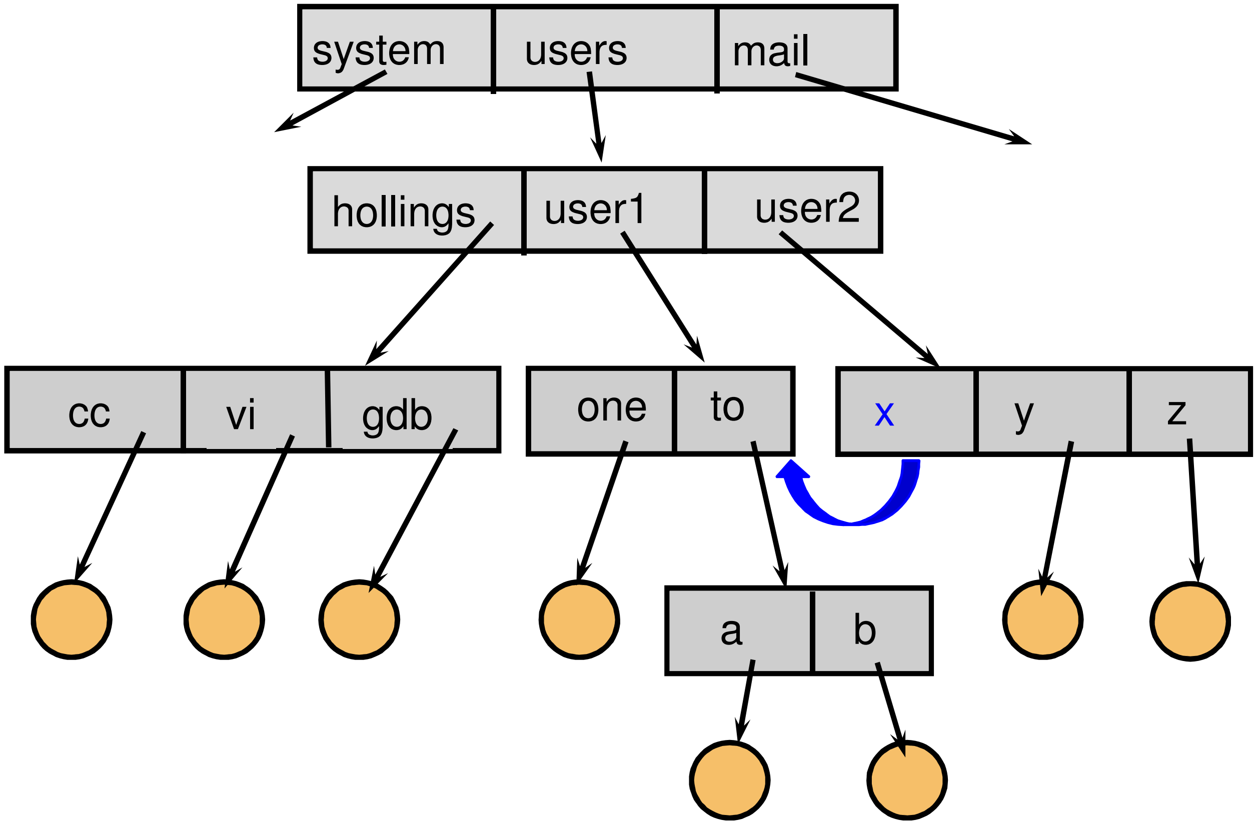


# Issues for Acylic Graph Directories

- 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

# Alternative: Linking

- Symbolic link (shortcut)



# 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
    - Older: four-byte pseudo-ASCII codes (e.g., “APPL”)
    - Newer: Uniform Type Identifier (e.g., “com.apple.application”)
  - also has an application associated with each file type
- Windows
  - has a file type in the extension of the file name (e.g., “.exe”)
  - has a table (per user) to map extensions to applications
- Unix
  - can use last part of filename like an extension (e.g., “.sh”)
  - applications can decide what (if anything) to do with it
  - look at first few bytes of file content for “magic number”
    - For example, ELF binaries begin with 7F 45 4C 46