

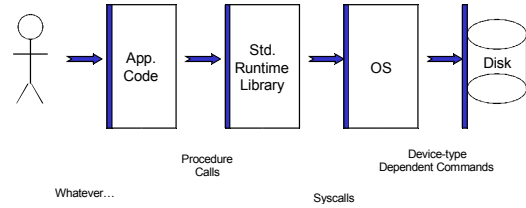
CSE 451: Operating Systems Autumn 2013

Module 15 File Systems

Ed Lazowska
lazowska@cs.washington.edu
Allen Center 570

© 2013 Gribble, Lazowska, Levy, Zahorjan

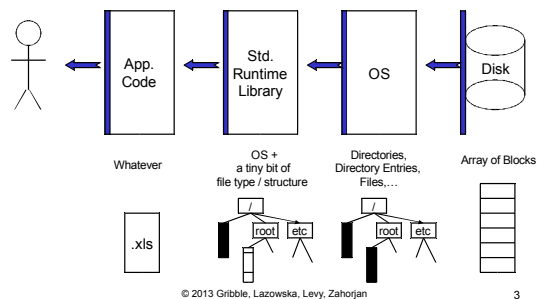
Interface Layers



© 2013 Gribble, Lazowska, Levy, Zahorjan

2

Exported Abstractions

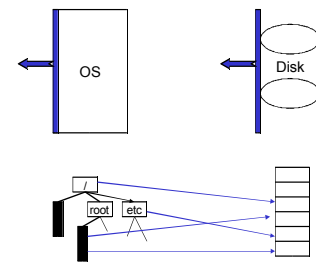


© 2013 Gribble, Lazowska, Levy, Zahorjan

3

Primary Roles of the OS (file system)

1. Hide hardware specific interface
2. Allocate disk blocks
3. Check permissions
4. Understand directory file structure
5. Maintain *metadata*
6. Performance
7. Flexibility



© 2013 Gribble, Lazowska, Levy, Zahorjan

4

File systems

- The concept of a file system is simple
 - the implementation of the abstraction for secondary storage
 - abstraction = files
 - logical organization of files into directories
 - the directory hierarchy
 - sharing of data between processes, people and machines
 - access control, consistency, ...

© 2013 Gribble, Lazowska, Levy, Zahorjan

5

Files

- A file is a collection of data with some properties
 - contents, size, owner, last read/write time, protection ...
- Files may also have types
 - understood by file system
 - device, directory, symbolic link
 - understood by other parts of OS or by runtime libraries
 - executable, dll, source code, object code, text file, ...
- Type can be encoded in the file's name or contents
 - windows encodes type in name
 - .com, .exe, .bat, .dll, .jpg, .mov, .mp3, ...
 - old Mac OS stored the name of the creating program along with the file
 - Unix has a smattering of both
 - in content via magic numbers or initial characters (e.g., #!)

© 2013 Gribble, Lazowska, Levy, Zahorjan

6

Basic operations

Unix

- create(name)
- open(name, mode)
- read(fd, buf, len)
- write(fd, buf, len)
- sync(fd)
- seek(fd, pos)
- close(fd)
- unlink(name)
- rename(old, new)

Windows

- CreateFile(name, CREATE)
- CreateFile(name, OPEN)
- ReadFile(handle, ...)
- WriteFile(handle, ...)
- FlushFileBuffers(handle, ...)
- SetFilePointer(handle, ...)
- CloseHandle(handle, ...)
- DeleteFile(name)
- CopyFile(name)
- MoveFile(name)

© 2013 Gribble, Lazowska, Levy, Zahorjan

7

File access methods

- Some file systems provide different **access methods** that specify ways the application will access data
 - sequential access
 - read bytes one at a time, in order
 - direct access
 - random access given a block/byte #
 - record access
 - file is array of fixed- or variable-sized records
 - indexed access
 - FS contains an index to a particular field of each record in a file
 - apps can find a file based on value in that record (similar to DB)
- Why do we care about distinguishing sequential from direct access?
 - what might the FS do differently in these cases?

© 2013 Gribble, Lazowska, Levy, Zahorjan

8

Directories

- Directories provide:
 - a way for users to organize their files
 - a convenient file name space for both users and FS's
- Most file systems support multi-level directories
 - naming hierarchies (`/`, `/usr`, `/usr/local`, `/usr/local/bin`, ...)
- Most file systems support the notion of current directory
 - absolute names: fully-qualified starting from root of FS

```
bash$ cd /usr/local
```
 - relative names: specified with respect to current directory

```
bash$ cd /usr/local (absolute)
bash$ cd bin (relative, equivalent to cd /usr/local/bin)
```

© 2013 Gribble, Lazowska, Levy, Zahorjan

9

Directory internals

- A directory is typically just a file that happens to contain special metadata
 - directory = list of (name of file, file attributes)
 - attributes include such things as:
 - size, protection, location on disk, creation time, access time, ...
 - the directory list is usually unordered (effectively random)
 - when you type "ls", the "ls" command sorts the results for you

© 2013 Gribble, Lazowska, Levy, Zahorjan

10

Path name translation

- Let's say you want to open `/one/two/three`

```
fd = open("/one/two/three", O_RDWR);
```
- What goes on inside the file system?
 - open directory `"/"` (well known, can always find)
 - search the directory for "one", get location of "one"
 - open directory "one", search for "two", get location of "two"
 - open directory "two", search for "three", get loc. of "three"
 - open file "three"
 - (of course, permissions are checked at each step)
- FS spends lots of time walking down directory paths
 - this is why open is separate from read/write (session state)
 - OS will cache prefix lookups to enhance performance
 - `/a/b`, `/a/bb`, `/a/bbb` all share the `"a"` prefix

© 2013 Gribble, Lazowska, Levy, Zahorjan

11

File protection

- FS must implement some kind of protection system
 - to control who can access a file (user)
 - to control how they can access it (e.g., read, write, or exec)
- More generally:
 - generalize files to **objects** (the "what")
 - generalize users to **principals** (the "who", user or program)
 - generalize read/write to **actions** (the "how", or operations)
- A protection system dictates whether a given action performed by a given principal on a given object should be allowed
 - e.g., you can read or write your files, but others cannot
 - e.g., you can read `/etc/motd` but you cannot write to it

© 2013 Gribble, Lazowska, Levy, Zahorjan

12

Model for representing protection

- Two different ways of thinking about it:
 - access control lists (ACLs)
 - for each object, keep list of principals and principals' allowed actions
 - capabilities
 - for each principal, keep list of objects and principal's allowed actions
- Both can be represented with the following matrix:

		objects		
		/etc/passwd	/home/gribble	/home/guest
principals	root	rw	rw	rw
	gribble	r	rw	r
	guest			r

The table is annotated with a red dashed box around the bottom row (guest) labeled "ACL" and a blue dashed box around the right two columns (gribble, guest) labeled "capability".

© 2013 Gribble, Lazowska, Levy, Zahorjan

13

ACLs vs. Capabilities

- Capabilities are easy to transfer
 - they are like keys: can hand them off
 - they make sharing easy
- ACLs are easier to manage
 - object-centric, easy to grant and revoke
 - to revoke capability, need to keep track of principals that have it
 - hard to do, given that principals can hand off capabilities
- ACLs grow large when object is heavily shared
 - can simplify by using "groups"
 - put users in groups, put groups in ACLs
 - you are all in the "VMware powerusers" group on Win2K
 - additional benefit
 - change group membership, affects ALL objects that have this group in its ACL

© 2013 Gribble, Lazowska, Levy, Zahorjan

14

The original Unix file system

- Dennis Ritchie and Ken Thompson, Bell Labs, 1969
- "UNIX rose from the ashes of a multi-organizational effort in the early 1960s to develop a dependable timesharing operating system" – Multics
- Designed for a "workgroup" sharing a single system
- Did its job exceedingly well
 - Although it has been stretched in many directions and made ugly in the process
- A wonderful study in engineering tradeoffs



© 2013 Gribble, Lazowska, Levy, Zahorjan

15

(Old) Unix disks are divided into five parts ...

- Boot block
 - can boot the system by loading from this block
- Superblock
 - specifies boundaries of next 3 areas, and contains head of freelists of inodes and file blocks
- i-node area
 - contains descriptors (i-nodes) for each file on the disk; all i-nodes are the same size; head of freelist is in the superblock
- File contents area
 - fixed-size blocks; head of freelist is in the superblock
- Swap area
 - holds processes that have been swapped out of memory

© 2013 Gribble, Lazowska, Levy, Zahorjan

16

So ...

- You can attach a disk to a dead system ...
- Boot it up ...
- Find, create, and modify files ...
 - because the superblock is at a fixed place, and it tells you where the i-node area and file contents area are
 - by convention, the second i-node is the root directory of the volume

© 2013 Gribble, Lazowska, Levy, Zahorjan

17

i-node format

- User number
- Group number
- Protection bits
- Times (file last read, file last written, inode last written)
- File code: specifies if the i-node represents a directory, an ordinary user file, or a "special file" (typically an I/O device)
- Size: length of file in bytes
- Block list: locates contents of file (in the file contents area)
 - more on this soon!
- Link count: number of directories referencing this i-node

© 2013 Gribble, Lazowska, Levy, Zahorjan

18

The flat (i-node) file system

- Each file is known by a number, which is the number of the i-node
 - seriously – 1, 2, 3, etc.!
 - why is it called “flat”?
- Files are created empty, and grow when extended through writes

© 2013 Gribble, Lazowska, Levy, Zahorjan

19

The tree (directory, hierarchical) file system

- A directory is a flat file of fixed-size entries
- Each entry consists of an i-node number and a file name

i-node number	File name
152	.
18	..
216	my_file
4	another_file
93	oh_my_god
144	a_directory

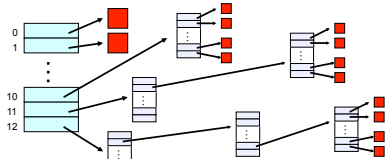
- It's as simple as that!

© 2013 Gribble, Lazowska, Levy, Zahorjan

20

The “block list” portion of the i-node (Unix Version 7)

- Points to blocks in the file contents area
- Must be able to represent very small and very large files. **How?**
- Each inode contains 13 block pointers
 - first 10 are “direct pointers” (pointers to 512B blocks of file data)
 - then, single, double, and triple indirect pointers



© 2013 Gribble, Lazowska, Levy, Zahorjan

21

So ...

- Only occupies 13 x 4B in the i-node
- Can get to 10 x 512B = a 5120B file directly
 - (10 direct pointers, blocks in the file contents area are 512B)
- Can get to 128 x 512B = an additional 65KB with a single indirect reference
 - (the 11th pointer in the i-node gets you to a 512B block in the file contents area that contains 128 4B pointers to blocks holding file data)
- Can get to 128 x 128 x 512B = an additional 8MB with a double indirect reference
 - (the 12th pointer in the i-node gets you to a 512B block in the file contents area that contains 128 4B pointers to 512B blocks in the file contents area that contain 128 4B pointers to 512B blocks holding file data)

© 2013 Gribble, Lazowska, Levy, Zahorjan

22

- Can get to 128 x 128 x 128 x 512B = an additional 1GB with a triple indirect reference
 - (the 13th pointer in the i-node gets you to a 512B block in the file contents area that contains 128 4B pointers to 512B blocks in the file contents area that contain 128 4B pointers to 512B blocks in the file contents area that contain 128 4B pointers to 512B blocks holding file data)
- Maximum file size is 1GB + a smidge

© 2013 Gribble, Lazowska, Levy, Zahorjan

23

- A later version of Bell Labs Unix utilized 12 direct pointers rather than 10
 - Why?
- Berkeley Unix went to 1KB block sizes
 - What's the effect on the maximum file size?
 - 256x256x256x1K = 17 GB + a smidge
 - What's the price?
- Subsequently went to 4KB blocks
 - 1Kx1Kx1Kx4K = 4TB + a smidge

© 2013 Gribble, Lazowska, Levy, Zahorjan

24

Protection

- **Objects:** individual files
- **Principals:** owner/group/world
- **Actions:** read/write/execute
- This is pretty simple and rigid, but it has proven to be about what we can handle!

© 2013 Gribble, Lazowska, Levy, Zahorjan

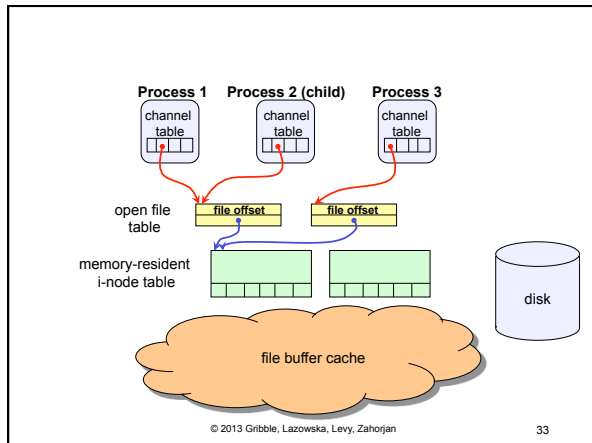
31

File sharing

- Each user has a “channel table” (or “per-user open file table”)
- Each entry in the channel table is a pointer to an entry in the system-wide “open file table”
- Each entry in the open file table contains a file offset (file pointer) and a pointer to an entry in the “memory-resident i-node table”
- If a process opens an already-open file, a new open file table entry is created (with a new file offset), pointing to the same entry in the memory-resident i-node table
- If a process forks, the child gets a copy of the channel table (and thus the same file offset)

© 2013 Gribble, Lazowska, Levy, Zahorjan

32



© 2013 Gribble, Lazowska, Levy, Zahorjan

33