

CSE 451: Operating Systems Autumn 2003

Lecture 13 File Systems

Hank Levy
levy@cs.washington.edu
Allen Center 596

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, ...

11/18/03

© 2003 Hank Levy

2

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, ...
 - unix has a smattering of both
 - in content via magic numbers or initial characters (e.g., #!)

11/18/03

© 2003 Hank Levy

3

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)

NT

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

11/18/03

© 2003 Hank Levy

4

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?

11/18/03

© 2003 Hank Levy

5

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)
```

11/18/03

© 2003 Hank Levy

6

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

11/18/03

© 2003 Hank Levy

7

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

11/18/03

© 2003 Hank Levy

8

Protection Systems

- 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 **principles** (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 subject on a given object should be allowed
 - e.g., you can read or write your files, but others cannot
 - e.g., your can read /etc/motd but you cannot write to it

11/18/03

© 2003 Hank Levy

9

Model for Representing Protection

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

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

ACL

capability

11/18/03

© 2003 Hank Levy

10

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 subjects that have it
 - hard to do, given that subjects 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

11/18/03

© 2003 Hank Levy

11