

CS 537 Lecture 12 File System Interface

Michael Swift

3/5/09

© 2004-2007 Ed Lazowska, Hank Levy, Andrea and
Remzi Arpaci-Dusseau, Michael Swift

1

Motivation: I/O is Important

Applications have two essential components:

- Processing
- Input/Output (I/O)
 - What applications have no input? no output?

I/O performance predicts application performance

- Amdahl's Law: If continually improve only part of application (e.g., processing), then achieve diminishing returns in speedup
- f : portion of application that is improved (e.g., processing)
- $speedup_f$: speedup of portion of application
- $Speedup_{Application} = 1 / ((1-f) + (f/speedup_f))$
 - Example:
 - $f = 1/2$, $speedup_f = 2$, $speedup_{app} = 1.33$
 - $f = 1/3$, $speedup_f = 2$, $speedup_{app} = 1.20$

11/08/07

© 2005 Steve Gribble

Role of OS for I/O

- Standard library
 - Provide abstractions, consistent interface
 - Simplify access to hardware devices
- Resource coordination
 - Provide protection across users/processes
 - Provide fair and efficient performance
 - Requires understanding of underlying device characteristics
- User processes do not have direct access to devices
 - Could crash entire system
 - Could read/write data without appropriate permissions
 - Could hog device unfairly
- OS exports higher-level functions
 - File system: Provides file and directory abstractions
 - File system operations: mkdir, create, read, write

3/5/09

© 2004-2007 Ed Lazowska, Hank Levy, Andrea and
Remzi Arpaci-Dusseau, Michael Swift

3

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, ...

11/08/07

© 2005 Steve Gribble

4

Abstraction: File

- User view
 - Named collection of bytes
 - Untyped or typed
 - Examples: text, source, object, executables, application-specific
 - Permanently and conveniently available

3/5/09

© 2004-2007 Ed Lazowska, Hank Levy, Andrea and
Remzi Arpaci-Dusseau, Michael Swift

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
 - file extension: .com, .exe, .bat, .dll, .jpg, .mov, .mp3, ...
 - content: #! for scripts
- Operating system view
 - Map bytes as collection of blocks on physical non-volatile storage device
 - Magnetic disks, tapes, NVRAM, battery-backed RAM
 - Persistent across reboots and power failures

3/5/09

© 2004-2007 Ed Lazowska, Hank Levy, Andrea and
Remzi Arpaci-Dusseau, Michael Swift

6

File Meta-Data

- Meta-data: Additional system information associated with each file
 - Name of file
 - Type of file
 - Pointer to data blocks on disk
 - File size
 - Times: Creation, access, modification
 - Owner and group id
 - Protection bits (read or write)
 - Special file? (directory? symbolic link?)
- Meta-data is stored on disk
 - Conceptually: meta-data can be stored as array on disk

3/5/09

© 2004-2007 Ed Lazowska, Hank Levy, Andrea and
Remzi Arpaci-Dusseau, Michael Swift

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?

3/5/09

© 2004-2007 Ed Lazowska, Hank Levy, Andrea and
Remzi Arpaci-Dusseau, Michael Swift

8

File Operations

- Create file with given pathname `/a/b/file`
 - Traverse pathname, allocate meta-data and directory entry
- Read from (or write to) offset in file
 - Find (or allocate) blocks of file on disk; update meta-data
- Delete
 - Remove directory entry, free disk space allocated to file
- Truncate file (set size to 0, keep other attributes)
 - Free disk space allocated to file
- Rename file
 - Change directory entry
- Copy file
 - Allocate new directory entry, find space on disk and copy
- Change access permissions
 - Change permissions in meta-data

3/5/09

© 2004-2007 Ed Lazowska, Hank Levy, Andrea and
Remzi Arpaci-Dusseau, Michael Swift

9

Opening Files

Expensive to access files with full pathnames

- On every read/write operation:
 - Traverse directory structure
 - Check access permissions

Open() file before first access

- User specifies mode: read and/or write
- Search directories for filename and check permissions
- Copy relevant meta-data to open file table in memory
- Return index in open file table to process (file descriptor)
- Process uses file descriptor to read/write to file

Per-process open file table

- Current position in file (offset for reads and writes)
- Open mode

Enables redirection from `stdout` to particular file

11/08/07

© 2005 Steve Gribble

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)`

3/5/09

© 2004-2007 Ed Lazowska, Hank Levy, Andrea and
Remzi Arpaci-Dusseau, Michael Swift

11

Directories

- Directories provide:
 - a way for users to organize their files
 - a convenient file name space for both users and FS's
 - a map from file name to blocks of file data on disk
 - Actually, map file name to file meta-data (which enables one to find data on disk)
- 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)
```

3/5/09

© 2004-2007 Ed Lazowska, Hank Levy, Andrea and
Remzi Arpaci-Dusseau, Michael Swift

12

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

3/5/09

© 2004-2007 Ed Lazowska, Hank Levy, Andrea and Remzi Arpaci-Dusseau, Michael Swift

13

Directories: Tree-Structured

- Directory listing contains <name, index>, but name can be directory
 - Directory is stored and treated like a file
 - Special bit set in meta-data for directories
 - User programs can read directories
 - Only system programs can write directories
 - Specify full pathname by separating directories and files with special characters (e.g., \ or /)
- Special directories
 - Root: Fixed index for meta-data (e.g., 2)
 - This directory: .
 - Parent directory: ..

3/5/09

© 2004-2007 Ed Lazowska, Hank Levy, Andrea and Remzi Arpaci-Dusseau, Michael Swift

14

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

3/5/09

© 2004-2007 Ed Lazowska, Hank Levy, Andrea and Remzi Arpaci-Dusseau, Michael Swift

15

Acyclic-Graph Directories

- More general than tree structure
 - Add connections across the tree (no cycles)
 - Create [links](#) from one file (or directory) to another
- Hard link: “ln a b” (“a” must exist already)
 - Idea: Can use name “a” or “b” to get to same file data
 - Implementation: Multiple directory entries point to same meta-data
 - What happens when you remove a? Does b still exist?
 - How is this feature implemented???
 - Unix: Does not create hard links to directories. Why?

3/5/09

© 2004-2007 Ed Lazowska, Hank Levy, Andrea and Remzi Arpaci-Dusseau, Michael Swift

16

Acyclic-Graph Directories

- Symbolic (soft) link: “ln -s a b”
 - Can use name “a” or “b” to get to same file data, if “a” exists
 - When reference “b”, lookup soft link pathname
 - b: Special file (designated by bit in meta-data)
 - Contents of b contain name of “a”
 - Optimization: In directory entry for “b”, put soft link filename “a”