CIS 452 - Operating Systems Concepts

Nathan Bowman

Images taken from Silberschatz book

Implementing Directories

Files and directories find make accessing secondary storage much simpler

Rather than requesting byte 2838494837, user opens `/home/me/project.c` and reads first line

However, recall that directories themselves are stored on disk

Accessing `/home/me/project.c` requires

- reading root directory to find `home`
- reading `/home` to find `/home/me`
- reading `/home/me` to find `/home/me/project.c`

Actually many more disk reads involved because directory does not store file location but rather inode location

Using file system rather than direct access to disk takes up disk space

Using file system requires additional disk accesses, slowing down process of writing and reading from disk

Filesystem designers must carefully consider tradeoffs in storage efficiency, access efficiency, and convenience for user

Recall: directory maps filename to info about file

Need some way to implement mapping that gives acceptable performance from directories

When considering how to implement directories, must consider how they will be used

- Search for file
- Create file
- Delete file
- Rename file
- List directory contents (`ls`)
- Traverse filesystem (e.g., create backup)

Simplest method is linked list

Each node contains filename, inode, and pointer to next node

Linear search is somewhat expensive however

For file creation, entire list must be searched to ensure file does not already exist

Entire list may also be searched during deletion, rename, and search

Directory information accessed often, so poor performance can noticeably slow system

Can be more efficient by keeping sorted directory

Sorted linked list will not help much -- still need to traverse links frequently

Balanced tree more appropriate for keeping sorted data

Creation, deletion, rename, search will all take logarithmic time

For even faster lookup time, hash table may be used

Hash value computed from filename used to index hash table

Hash table stores pointers to directory entries

Hash tables convenient for search, insertion, deletion

General problem with hash tables is accounting for collisions

One option is to make hash table large, but this wastes space and is not foolproof

Other common option is to store linked lists as hash table entries

Downside to hash tables with linked lists is additional disk accesses

However, collisions should be rare, so lists generally short

Still much faster than linear search through full directory

Recall that each access to directory is disk read -- very expensive

Inefficient implementation of directories can severely hurt system performance

In reality, OS will cache whenever possible, so not as dire as it seems