File Systems - Part I

CS 537 - Introduction to Operating Systems

File

- A named collection of related data that is recorded on secondary storage
  - text, binary, executable, etc.
- A file is broken up into chunks
  - chunk is the smallest accessible piece of a file
  - often called a *record*
- Internal Structure is dependant on OS
  - in Unix, all chunks are 1 byte in size
  - some systems may allow larger records

Naming

- Files are given human readable names
  - necessary to allow users to remember files
- Length of name
  - limits amount of description for names
- Case sensitive names or not
  - foo vs Foo
- What characters are allowed in name
  - Unix doesn't allow spaces, Windows NT does

Types

- Not all systems support types
- Type indicates how OS should handle file
- Unix supports 3 types
  - directory files, device files, regular files
  - leave it to applications to support file types
    - emacs and *.c files, javac looks for *.java, etc.
- Windows supports many types
  - extension indicates what type file is
  - word document (.doc), batch file (.bat), etc.
  - double clicking on a file launches the program

Meta Data

- Information about a file
- Usually stored with the file
- Different systems record different data
- A few examples
  - size of file
  - creator, when created, last modification
  - who is allowed to access the file
  - number of links to the file
  - etc

File Operations

- Most systems allow access to files through the OS only
- This requires a set of system calls for files
  - create, open, write, read, reposition, close, delete
- Some systems support many more functions

Creating a File

- File name is placed in the file directory
  - more on directories later
- Space on disk is found for the file
  - more on space allocation later
- Initial meta data for the file is recorded
  - creator, permissions, etc.
- File is now available for usage

Open File Table

- OS table that caches information on open files
- Prevents having to read file information from disk on every access to a file
- File Pointer (FP)
  - indicates user's current location in the file
- An entry in the open file table includes starting location of file on disk and the FP
- *File descriptor* is the index of the file in the open file table

Opening a File

- Directory is searched for the file
- Starting location of file is recorded in open file table
- FP is set to zero (beginning of file)
- File descriptor is returned to user and that is now used for accessing the file
  - use FD instead of file name
- Example

```
int fd = open("filename", READ | WRITE);
```

Opening a File

- Possible for multiple users to open a file
- Only one entry in open file table
- Each user keeps its own copy of the FP
  - each user can be in a different place in the file
- Need to be careful about consistency semantics
  - what if two users have file open and one writes

Closing a File

- When last reference to the file is closed, file is removed from the open file table
- Need to make sure all changes to the file are written back to disk
  - remember, file blocks are cached in memory for performance reasons
- Example

```
close(fd);
```

Deleting a File

- Need to reclaim all the blocks on disk that the file is using
- Need to remove the file from the directory structure
- Caution
  - may be multiple references to a file
  - need to be careful about deletion
  - more on this when discussing directories
- Unix deletes a file when there are no more references to the file

File Accesses

- Sequential
  - must go through records in the order they exist in the file
  - to get to the N^{th} record, must first search through records 0 to N-1
- Random
  - allow user to access any record in a file directly
  - sequential access can be easily simulated with random access
- Indexed
  - search an *index* to find a pointer to specific block on disk

File Access

- Choice of access type can greatly effect performance of file system
- Random and indexed access can lead a programmer to "jump" all over file
  - prefetching won't help performance
- Sequential access keeps users localized
  - prefetching can provide great benefits
- Programmers will do whatever API makes easier

Reading / Writing a File

- Location of read/write depends on type of access allowed
  - Unix requires all accesses to occur from location of FP
  - FP is updated after each read/write
- Reads and writes occur in memory
  - block is transferred to/from disk as needed
- Example

```
int count = read(fd, buffer, length);
int count = write(fd, buffer, length);
```

Repositioning Within File

- Many systems provide random access via a *seek* system call
- This allows a user to "jump" to a specific location in a file
- What it actually does is change the FP
- Example

  lseek(fd, distance, whence);

  – *whence* indicates where to seek from
    - beginning of file, current position, end of file