LECTURE 3.4
FILE SHARING AND SYSTEMS

COP4600

Dr. Matthew Gerber

4/18/2015

FILE SHARING

File Sharing: User Level (11.5.1)

- Multiple users of an operating system require at least some level of file sharing management
- At a minimum, the operating system must keep track of:
  - The *owner* of a given file
  - Which other users are allowed to access a file
  - What those others are allowed to do with the file
- We will discuss mechanisms for this later
- This information may be maintained about directories as well
  - Often, files and directories will *inherit* this information from their parent directories

File Sharing: File System Level (11.5.2)

- Over computer networks, entire file systems can be shared
  - The World Wide Web is in large part simply a file-sharing network!
  - URIs state a protocol, a *server*, and a file
  - The web browser, or *client,* connects to the server and requests the file
- *Distributed file system* protocols, by contrast, provide the ability to actually mount remote file systems – or subdirectories of them – as we would mount locally attached storage
  - A client machine connects to the *file server*, which (hopefully) performs appropriate authentication and authorization checks before allowing access to the files

File Permissions (11.6)

- We want to be able to control which users are allowed to perform which operations on files
- This requires categorizing both users and operations
- Typical operations that may have separate access requirements include:
  - **Reading from** a file
  - **Writing to** a file
  - **Executing** a program
  - **Deleting** a file
  - **Listing** a directory

File Permissions: The UNIX Model

- UNIX-like systems have three permissions assigned to three classes of users for each file and directory
- For **files**…
  - **Read** permission allows the file to be read
  - **Write** permission allows the file to be written to
  - **Execute** permission allows the file to be executed
- For **directories**…
  - **Read** permission allows the directory to be read and listed
  - **Write** permission allows files in the directory to be created, deleted, and renamed
  - **Execute** permission allows items in the directory to be located without reading it, if their name is known
- The classes of users are:
  - **User:** The user whose file it is; sometimes called the owner
  - **Group:** The group to which the file is assigned
  - **Others:** Everyone else; sometimes called the world

DO NOT USE THE OWNER/GROUP/UNIVERSE NOMENCLATURE!

File Permissions: Access Control Lists

- *Access Control Lists* allow any set of users to be arbitrarily granted a given set of access rights to any file
- All serious ACL mechanisms allow for groups to be created, and those groups to be granted access the same way as users
- UNIX-like systems often combine ACLs with the older UNIX permissions model in various ways

FILE SYSTEMS

Disks and File Systems (12.1)

- These attributes are common to essentially all file systems:
  - We divide disks into partitions
  - We format partitions using file systems
  - File systems have hierarchical directories
  - Directories provide references to files
  - Files are stored in blocks
  - Blocks are one or more sectors on disk
- That’s pretty much it
- File systems implement all of this radically differently
- We do, at least, have some ways of describing the pieces

File System Layers

- The application makes file requests through the system libraries
- ***One way*** of looking at what happens next is the following
  - Those requests eventually arrive at the **logical file system**, which manages all of the directory-level metadata, and in turn consults the…
  - **…file organization module**, which manages the actual location of files on the physical disk, including free space, and in turn consults the…
  - **…basic file system**, which brokers the commands to actually read and write blocks on the physical disk, by consulting the…
  - …**I/O control layer**, which consists of the device drivers and interrupt handlers, that in turn do their work by consulting the…
  - …disk
- We will not be able to go through everything necessary to write a file system driver in this course
  - …but we can hit the high points

Implementation: System Structures (12.2.1)

- The *boot control block*, if it exists, contains the information necessary to boot an operating system from the partition
- The *volume control block* or *superblock* contains (at least) the number of blocks in the partition, the size of the blocks, free-block information, and information about file control blocks
- The root (at least) of the partition’s directory structure must be managed at the partition level

Implementation: File Structures

- Each file has a *file control block* that contains at least the following subset of what we have referred to generically as directory information:
  - Permissions and access control
  - Dates
  - Size
  - Information required to locate the file
- FCBs are associated with directory entries

Implementation: Working with Files

- When files are opened, their FCBs are copied into the open-file table
- The operating system provides the process a pointer into the open-file table
- All file I/O is done using this pointer

Special Partitions (12.2.2)

- We described mounting file systems in our last lecture
- Partitions do not *have* to have file systems
- *Raw disk* partitions are used when no file system is necessary or desirable
  - Easy example: Swap space
  - Another common example: High-end databases
    - Some report as much as 10% performance gain from using raw partitions

Virtual File Systems (12.2.3)

- *Virtual File Systems* are OS-internal programming interfaces that allow support for new file systems to be easily added to the kernel
- A VFS specification describes a means by which a file system will provide, in part:
  - Information to the kernel about:
    - Files on disk
    - Open files
    - File systems
    - Directory entries
  - Means of:
    - Traversing, reading and writing directory entries
    - Opening, closing, reading and writing files
- Importantly, these do not *actually* have to reside on disks

File Allocation

- *Contiguous* allocation requires that all the blocks of a file reside one next to another on the disk, in order
- It mostly isn't worth talking about
- That means we need a method for storing pieces of files in various locations on the disk

Linked Allocation

- Each file is a linked list of disk blocks
  - Each block contains a pointer to the next block
  - Solves all the usual problems of contiguous allocation, but…
  - Seeking is slow – $O(n)$ reads
- The *File Allocation Table* method implements linked allocation, but keeps all the pointers at the beginning of the disk
  - Doubles as the list of unused blocks
  - Theoretically can double the number of reads necessary, but the FAT is *aggressively* cached
  - Seek time is still theoretically $O(n)$ reads, but likely to result in a very small number of actual disk reads
- Developed in 1981, still in active use for small storage devices with simple requirements

Indexed Allocation

- Alternate method of solving the seek problem
- Each file is given its own *index block*, linked to by its FCB
- The n^{th} entry in the block points to the n^{th} block of the file
- Two methods of handling what happens when a file has more blocks than can fit in a block
  - Linked indexing – the last entry in a block links to another index block
  - Multi-level indexing – make the blocks hierarchical
- UNIX-based file systems combine the two
  - The first few *direct block* pointers sit right in the FCB
  - The last three pointers in the FCB point to blocks of increasing levels of *indirection*
    - The *single indirect* block is a block of pointers
    - The *double indirect* block is a block of pointers to blocks of pointers
    - The *triple indirect* block is a block of pointers to blocks of pointers to blocks of pointers

Free Space Management

- We need a way to keep track of what blocks are free
- FAT makes it easy; indexed systems less so
- Bit vectors are simple but break down quickly with large devices
- We can repurpose the pure linking approach, with each free block containing only a link to the next free block
- We can also use any of the indexed approaches, and store the indexes *in the free blocks*
  - This requires some bookkeeping when we start allocating enough blocks to need to reclaim index blocks
  - …but in the real world, if we have the disk that close to full, we're already in trouble

Performance Considerations

- Caching is a necessity for reasonable disk performance
  - This applies even with SSDs
- Essentially all current systems use *unified memory* techniques, where all physical memory not in use by processes is available for use by the disk cache
- Caches must distinguish between *synchronous* writes that must occur immediately when received, and *asynchronous* writes that can occur in any order
  - Synchronous writes appear in database transactions and in other conditions where reaching stable storage immediately is essential
- Thought exercise: *for a process*, is it faster to read or to write?

Recovery and Consistency Checking

- Crashing in the middle of updating metadata is an easy way to create undefined behavior
- Operating systems must include *consistency checking* programs that examine partitions and recover them from standard patterns of inconsistent metadata
- Crashes at the wrong time can still cause serious problems – not all patterns of inconsistent metadata are recoverable
- Caching makes all of this even worse

Journaling

- *Journaling* greatly alleviates the problem of inconsistent metadata
- Metadata changes are not actually performed directly, but are written synchronously to a journal then – at a later time – updated asynchronously on the disk from there
- On a crash, the system simply completes all changes that are still in the journal
- This actually *improves* performance by simplifying the need for synchronous writing
- Data can be corrupted, but metadata cannot be
  - Variant allow for all data to be journaled, but this *does* come at a performance cost

NEXT TIME: NETWORKS