File-System Internals

Practice Exercises

15.1 Explain how the VFS layer allows an operating system to support multiple types of file systems easily.

Answer:
VFS introduces a layer of indirection in the file system implementation. In many ways, it is similar to object-oriented programming techniques. System calls can be made generically (independent of file system type). Each file system type provides its function calls and data structures to the VFS layer. A system call is translated into the proper specific functions for the target file system at the VFS layer. The calling program has no file-system-specific code, and the upper levels of the system call structures likewise are file system-independent. The translation at the VFS layer turns these generic calls into file-system-specific operations.

15.2 Why have more than one file system type on a given system?

Answer:
File systems can be designed and implemented with specific uses in mind, and optimized for those uses. Consider a virtual memory file system vs. a secondary storage file system. The memory-based one need not concern itself with fragmentation, or persisting data structures in the face of power loss. There are also special-purpose file systems like the procfs file system, designed to give the convenient file system interface to system aspects like the process name space and process resource use.

15.3 On a Unix or Linux system that implements the procfs file system, determine how to use the procfs interface to explore the process name space. What aspects of processes can be viewed via this interface? How would the same information be gathered on a system lacking the procfs file system?

Answer:
On systems containing the procfs psuedo-filesystem, details vary but generally the file system is mounted at /proc, and exploring it with file system commands can reveal the following:

- Each process is represented by its processID, so counting them reveals the number of processes in the system.
- Within each directory under the processID, details of the process state such as its current working directory, command line used to start the process, priority information, memory use information, lock information, open file information, and so on.
- Some procfs also provide interfaces to other kernel structures such as DMA structures, device lists, file system lists and so on.

See the `proc(5)` manual page for details on a given system.

Without procfs, to provide the same information, separate system calls per information type is used, or the ability to open the kernel memory space through `/dev/kmem` or `/dev/sys` is provided. Then programs using these interfaces need to be written to extract the data and present it in human-understandable form. See http://osxbook.com/book/bonus/ancient/procfs for a nice exploration of using procfs vs. not having it available.

15.4 Why do some systems integrate mounted file systems into the root file system naming structure, while others use a separate naming method for mounted file systems?

Answer:
As with many aspects of operating system design, choices can be arbitrary or based on some small implementation detail and then exist long after any justify reason. Generally regarding file system mounting, integration with the root file system naming has proven to be more flexible and useful and separate mount point naming and therefore prevails on most file systems.

15.5 Given a remote file access facility such as `ftp`, why were remote file systems like NFS created?

Answer:
Users of computer systems value ease-of-use in most cases. The more general purpose, and more widely used and operating system is, the more seamless its operation should be. In the case of remote file access, it is easier for users to use an familiar facility (such as a file system interface) rather than separate commands. And because file systems are tried and true, well integrated, and full featured, existing tools, scripts, and use cases can apply to remote file systems just as local file systems by using a file system interface for remote file access.

Exercises

15.6 Given a mounted file system with write operations underway, and a system crash or power loss, what must be done before the file system is remounted if: (a) The file system is not log-structured? (b) The file system is log-structured?

Answer:

(a) There are many data structures involved in file system implementation (including directory structure metadata, per-file metadata, free space tracking metadata, and volume detail metadata). Before mounting a file system and depending on these data structures to be accurate, they must be examined and confirmed to be correct. As one example, if there is a number-of-files count in the volume data structures, the number of file metadata structures (inodes for example) need to be counted to determine if they match. Likewise, the list of free blocks needs to be compared to all used blocks (by looking at each file's allocated block list) to be sure the same block is not both on the free list and used. For large file systems with large numbers of files, these file system checks can take a very long time, and must complete before the file system is available for use.

One shortcut in this case is for the operating system to write a flag to the file system when the file system is consistent, for example once a set of buffers is completely written out to the file system. If the flag is set, these checks need not be run. Also, do decrease the possibility of directory structure metadata corruption that could cause the loss of much or all of the filesystems' contents, some operations such as directory structure changes can be done atomically (and thus more slowly).

(b) Log-structured file systems place all file system changes into a log file on the storage media. The changes are then replayed into the file system structures over time, with the system noting which changes are complete and which are only in the log file. Changes complete before a crash are not a worry, while changes not made can be made after the system is back up and running. No data structures are in an incorrect state, so no data structure checking is needed (in theory).

15.7 Why do operating systems mount the root file system automatically at boot time?

Answer:

Aside from the kernel, a functioning operating system has daemons and user interfaces. Some operating systems (such as TOPS-20), integrated the user interface into the kernel, but most find the flexibility, ease of debugging, ease of modification, and protection and security implementation to favor separate components.

Given separate components that make up a functioning operating system, what is the best way to store, manage, and retrieve those components? A general system for naming, finding, managing, and updating programs is needed. Fortunately there is already a facility designed to do that—a file system. So, at boot time, a root file system is mounted

providing access to other operating system components (as well as other files). Because the file system facility is sometimes too general purpose for providing operating system contents (Should it be easy to modify files in the operating system's file system? Should arbitrary programs be executable from there or only ones guaranteed to be from the operating system vendor?) some systems place limits on what can occur in the root file system or which programs can be run (see Section 17.11.4 for example).

15.8 Why do operating systems require file systems other than root to be mounted?

Answer:
There are several reasons to require an explicit mount operations for accessing other file systems.

- The user performing the mount needs to have permission to access the contents, so permission checking is part of the mount operation.
- If there is a problem with the mount (the file system type is not supported, the data structures are corrupt, and so on) human intervention may be needed, so a human should be executing the command.
- Where the file system is integrated into the system name space can vary, so an explicit command can provide that detail.
- The operating system may not be able to automatically determine the type of file system, especially in systems where third-party file system types can be loaded into a running system, so the type of the file system must be provided in the mount command.
- The mount command can have various options to optimize performance or determine uses for the file system being mounted.