Secure Operating System Design and Implementation
File System

Jon A. Solworth

Dept. of Computer Science
University of Illinois at Chicago

February 1, 2011

Part I

File system overview

Overview

- A file system is one of the most important components of an OS.
- It is the only OS component which is designed to withstand failure of the OS.
- For this reason, updates to the file system must be made very carefully.
- Because failure can happen at any time
- And the system should still reboot and get to a consistent state.

Part II

Disks

Disks

- Mechanical disk drives are still the primary storage for desktop systems
- A disk drive contains one or more discs
- Each disk consists of a set of concentric circles (tracks) on which data is written.
- Each track is divided in disk blocks containing data, error correcting codes, and control information
- A disk spins at constant revolution per minute
- A disk head reads (or writes) data on the disk
- In a hypothetical one track disk, a read occurs by
  - waiting for the desired block to come under the disk head and then
  - reading the block (including its error code)

Disk arms

- Of course, disks have multiple surfaces
- and each surface has multiple tracks
- A disk arm moves perpendicularly between the outer- and inner-most tracks on the disk
- So now a disk access needs first to move the arm (called a seek)
- and then wait for the disk block to rotate under the disk (called rotational latency)
- Followed by the read (or write) causing the third time component the transfer

More about disks

More about seeking

- The disk arm is mechanical and tracks are packed closely together
- Hence, disk arms must move to the track and check whether it is on the right track by reading the block header.
- If it's not, the process repeats

More about block layout

- Tracks at the outer edge are much larger than those at the inner edge
- So more blocks can be kept in an outer edge track
- Keeping approximately constant the track bit density
- But varying bit read and write speeds

More about disks (cont'd)

Disks cache data

- May reorder operations to write/read more efficiently
- May not have written data to disk before returning
- The best greedy algorithm is Shortest Service Time First (SSTF), which picks the next I/O to do with the smallest rotational latency plus seek time

Part III

Ethos file system

Ethos file system

- Ethos file system contains files and directories
- And terminal input and terminal output
- File-system-like entities also include Inter-Process Communication and Networking
- Each file system entity has a label (used to determine which processes can access it)

Ethos directories

- Directory is organized in a tree
- A directory specifies a name type
- All files and sub-directories of a directory have the same type for a name
- Directory entries are (logically) kept in sorted order and thus can be iterated through
- Directories can be very large and are (logically) kept as a B-tree
- Directories are to be viewed as dictionaries (providing a name to object map)
- Directories can be given a name to open, not a path

Ethos files

- Each file in the directory corresponds to a high level language variable, it is read or written atomically.
- Hence there is no seek and no current file position to maintain.
- The read system call does not know the size of the file to be returned, so that is dynamically negotiated with user space (in the `retire` system call)

Streams (sequences)

- In contrast to UNIX, Ethos files are *not* streams (sequences)
- But terminals are sequences
- As directories can be sequences (when index by time).
- Which means that read and write are defined over directories need to check the details here
- IPC and Networking remain sequences

Terminal I/O

- Output to the terminal of binary objects is automatically converted to strings
- There is no random garbage that occurs on the terminal screen
- The output is invertible, it can be read back in by terminal input
- The goal is to have binary and textual live seamlessly together
- This component needs to wait until we have a higher level programming language.

Part IV

File system operations

File system operations

- We aren't actually building a file system
- Instead we're borrowing Linux's file system (on Dom0)
- This saves us from doing block allocation, inode/directory implementation, file system recovery
- In return we have to build a communication to Dom0 to request file system operations
- The operation is performed by shadowdaemon process which runs as root in Dom0.
- We'll use Ethos RPC to communicate between the shadowdaemon and Ethos kernel.

Anatomy of a file system operation

- Files and directories are described by file descriptors (Fd) local to a process.
- Each valid Fd maps to a 64-bit unsigned resource descriptor ID (RdId)
- RdId's are globally unique in Ethos and never recycle
- Consider an allocation every nano-second
- 64-bits gives 2^{34} seconds or about 500 years
- The RdId maps to a FileInformation structure which contains information about the file in Ethos.
- The FileInformation, if not present as the result of a previous call, is fetched from the shadowdaemon when creating a FdId

Getting the file information

- An event is created to perform the remaining work after the shadowdaemon replies
- An RPC for fileInformation is sent to the shadowdaemon
- The shadow daemon gets information on the file
- An RPC for fileInformationReply is sent from the shadowdaemon to Ethos
- It is paired to the event
- Which then completes processing on the event