Advanced Unix System Administration

Lecture 4
September 24, 2008

Steven Luo
<sluo+decal@OCF.Berkeley.EDU>

Input and Output

- Files
  - The file is (in principle) the fundamental abstraction behind Unix I/O
    - “Everything is a file” – the famous Unix mantra that's maybe true
  - As far as user-space programs are concerned, a “file” should be a stream of data which can be read from and written to
    - Could be a file on disk, a network socket, a device, etc.
    - Whether the file is opened via a filesystem is another story

Input and Output

- Synchronous I/O
  - At simplest: process makes syscall to I/O facility, kernel does I/O, returns
  - This is what read(), write(), and friends do
  - Because we treat network sockets and various other things as files, they can be handled in a similar way
  - This model has some inefficiencies – context switches, copies, and blocked processes

Input and Output

- Asynchronous I/O
  - Allows the process to do something else while I/O is running
  - Different ways of doing this: don't bother notifying the process, polling, event loop, signals/callbacks
- Memory-mapped I/O
  - Processes and kernel arrange to read/write from memory in orderly fashion
  - Fundamentally async

Input and Output

- I/O scheduling
  - When multiple requests to a particular I/O source come, we should try to arrange them efficiently
    - Simple first in, first out model works fine for networks – not so well for rotational disk media
    - On rotational disks, try to arrange requests so that reads and writes are near each other on the platter
    - When multiple devices are concerned, take into account which device data is on
  - If we're going to schedule, we might as well do priority scheduling too ...

Input and Output

- Filesystems
  - At the core, a FS is just a way of collecting files efficiently
  - Construction: usually laid out as blocks of various types
  - Directories contain pointers to other directories and inodes
  - inodes store filenames, metadata (permissions, ACLs, timestamps), and pointers to the actual data blocks

Input and Output

- POSIX filesystems
  - Unix filesystems traditionally make various guarantees – i.e. creating links will be atomic
  - This means that applications make assumptions about the way they operate on files (example: the standard way of safely replacing a file – especially a binary – while in use)
  - NFS breaks quite a few of these assumptions – hence random tricks and workarounds