Advanced Unix System Administration

Lecture 3
February 7, 2007

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

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
  - Note that “everything is [supposed to be] a file” in Unix – network I/O is 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

- Files and 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

Processes

- Processes in the kernel
  - Each process is assigned a process ID number (PID)
  - Kernel keeps a huge amount of state per process: priority, whether blocked or not, credentials, execution state, etc.
  - (Linux) Pointers to these structures are stored in a hash table hashed by PID and in a linked list

Processes

- Scheduling
  - On most systems, there is a “run queue” or “ready queue” of processes that are not blocked
  - Kernel looks at processes to see which aren't blocked
  - Dispatcher looks at processes in run queue and decides which one runs next and for how long
  - When time's up, dispatcher stops the running process and performs the context switch

Processes

- Scheduling considerations
  - Priority: higher-priority tasks should run more often
  - Starvation: processes that haven't run in a long time should run
  - (SMP systems) Processor affinity
  - Locks held by processes; priority inversion
  - Different workloads benefit from different algorithms for sorting this out