Advanced Unix System Administration

Lecture 2
September 17, 2008

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

Administrative Stuff

- I've set up a mailing list for the class
  - sluo+decalfa08@OCF
  - Feel free to use it to contact each other
  - If you don't get email from the list, let me know!
- These slides will be posted on website
  - http://www.ocf/sysadmin-class/2008-fall/advanced/

Processes

- In some sense, the process is the “fundamental actor” on a Unix system
  - Everything that happens on the system is done by or for a process
  - Most of the attributes and resources that the system assigns or keeps track of belong to a process in some way or another
  - Users interact with the system via their processes

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, owning user and group, permissions, execution state, etc.
  - (Linux) Pointers to these structures are stored in a hash table hashed by PID and in a linked list

Processes

- Process creation
  - fork() and friends – creates a copy of the parent
  - If a new program is being invoked, a following call to one of exec family of functions will overwrite the address space with the code of the new program
  - Dynamic binaries: the dynamic linker loads code (more later)
  - Start of program execution

Processes

- strace(1), truss(1), ktrace(1)
  - Provides a view of the syscalls used by a program
  - Can be run on new processes, follow their children, or be attached to an existing process
  - Output is valuable when process is doing I/O, sleeping, or otherwise talking to the kernel; of no use when purely userspace
  - Can filter out selected syscalls – useful because output is very noisy

Processes

- The process tree
  - Every process has a parent – the process from which it fork()ed
  - Parent has privileges (and responsibilities) with regards its children
  - Parent and children form a process group, which has an ID number (usually parent's PID)
  - The start of the process tree is init (always PID 1)
  - Orphaned processes are inherited by init

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

Processes

- Signals
  - Allow processes to communicate with each other and the kernel
  - Provide primitive mechanism for implementing callbacks – signals can be trapped and a “signal handler” called
  - If not handled, signals perform a default action (usually exit)
  - Signal programming is tricky because of synchronization and syscall restarting issues
  - Try `man kill` or `kill -L` for more information