COMP09024 Unix System Administration

Lecture 4: Processes, Jobs and Scheduling

Duncan Thomson/Hector Marco

UWS

Trimester 1 2020/21

Outline

4.1 Processes

What is a Process?

- A *process* is an instance of a running program
- If a program is run multiple times (by same or different users) each 'instance' requires a separate process
- In Unix, every process is uniquely identified by a process ID (PID)
- When a program runs, the OS keeps track of various data:
  - The program binary or *image*
  - Memory used (.text, .data, Stack, Heap, etc.)
  - State information (registers, running threads)
  - Resources (files, network connections)
  - Security information (user/group, permissions)
- These data together form a `process`

Process Lifecycle

- A process is *spawned* by another process, known as its parent (except for PID 1 (`init`), which is started by the kernel)
  - The parent process uses the `fork` system call, which makes a copy of itself
  - One of these copies uses the `exec` system call, which replaces itself with the new program
- An active process is usually in one of three states: running, sleeping (waiting for something), or stopped
- A process informs its parent when it terminates
- If its parent isn't listening, it remains a 'zombie'
- If its parent has terminated, it becomes an 'orphan' and is adopted by PID 1 (which becomes its new parent process)

Examining Processes: ps

- The `ps` command shows a list of processes
- By default, it shows only:
  - That user's *interactive* processes
  - PID, controlling terminal, time running, and command
- The `ps` command has a wide range of options, which vary depending on the Unix variant
- The GNU version of `ps` common in Linux distributions includes
  - Unix System V options, preceded by `-`
  - BSD options, without `-`
  - GNU options, preceded by `--`

Some Options to ps

- Some options changing the processes displayed:
  - a show processes from other users
  - x show processes without a terminal
  - -e (or ax): show all processes
  - r show only running processes
- Some options changing what information is displayed:
  - -f full format (parent PID)
  - -F extra full format (PPID and memory usage)
  - -l long format (UID, status and priority)
  - u user-oriented format (username, CPU and memory usage, status
- Various flags to ps allow much finer control: selection criteria; order and format of fields; sort order; and more — man ps gives full details

The top Command

- Sometimes it is useful to see a realtime view of the system and processes, rather than the snapshot given by `ps`
- The `top` command shows a continually refreshed view of some key system statistics, and some processes, sorted (by default) by CPU usage
- Statistics include load averages, number of users, process counts, CPU usage statistics, and memory/swap usage
- `top` is exited by pressing `q` (quit)
- A variety of keystrokes allow control of fields displayed, sort order, refresh rate (pressing `?` gives a list)
- Processes may also be reprioritised or killed

Examining System Information

- System information commands vary a lot from one Unix system to another
- The following all work on Linux, some may also work on other Unices
- `uname` gives system name and kernel versions
- `free` gives memory usage; `vmstat` gives more detail
- `df` summarises disk space usage
- `uptime` shows system uptime
- `lscpu` shows basic CPU information (more in `/proc/cpuinfo`)
- `lspci` and `lsusb` show hardware connected to PCI and USB busses

4.2 Controlling Processes

Stopping Interactive Processes

- Interactive processes — those which are (in theory) interacting with a user via a terminal — may be terminated
- Termination of an interactive process is done by pressing **Ctrl-C** on the controlling terminal
- This sends an 'interrupt' (`SIGINT`) signal to the process, which normally results in it terminating
- Some other control characters are sometimes confused with:
  - Ctrl-D (End Of File) is often used to terminate input to a process (for example to logout of a system)
  - Ctrl-Z sends a `SIGTSTP` signal, which stops (or suspends) a process; this does not terminate it, but detaches it from the terminal while it waits for a signal to continue

Signals

- Signals facilitate sending a basic notification to a process
- Each type of signal has a number and an associated name
- Other than the signal number, no other data is sent to a process when a signal is sent
- Some important signals for the Unix administrator include:
  - SIGINT (2) interrupts the process, usually terminating it
  - SIGTERM (15) asks the process to terminate
  - SIGKILL (9) kills the process
  - SIGHUP (1) for 'hang up' — server processes (daemons) will often reload configuration data on receipt of this signal
  - SIGTSTP or SIGSTOP stops execution
  - SIGCONT restarts execution

The `kill` Command

- The `kill` command sends a signal to a process
- The syntax is `kill -`*`SIGNUM PID`*
  - *`SIGNUM`* is the signal number
  - *`PID`* is the process ID
  - If the signal number is omitted, the default is `SIGTERM` (15)
- Normal users may only send signals to their own processes
- `root` can send signals to any process
- An alternative to providing a PID is providing a job number (see later) with a `%` sign in front eg: `kill -9 %1`
- Note that `top` allows sending signals using the `k` key

Process Priority, `nice` and `renice`

- The exact priority processes are given depends on scheduling algorithms used by the specific Unix kernel
- The *nice* value of a process is a user interface to influence priority: the nicer, the lower priority
- The default nice value is 0, and values range from -20 to 19
- Processes may be started with positive (or negative, by `root`) nice values using: `nice` *`NICE command args`*
- The niceness of a running process can be changed with `renice` *`NICE PID`*
- Normal users can only renice their own processes, and only increase the nice value; `root` can renice arbitrarily
- In `top`, processes can be reniced with the `r` key

4.3 Job Control from the Shell

Foreground and Background Tasks

- The shell normally starts processes in the *foreground*
- This means the shell must wait for the process to terminate before anything else can be done
- For non-interactive batch tasks, there is generally no need to wait for such processes to terminate — these can be run in the *background*
- Interactive processes can be run in the background, but
  - Output from the process will be mixed in with output from the foreground task
  - If the process requires input, it will stop and wait
- A process is run in the background by appending `&` to the command line, eg: `sleep 50 &`
- The shell prints a job number (in brackets) and the PID

jobs, fg and bg

- When a number of tasks are running, it can become difficult to keep track of them all
- The `jobs` shell command gives a list of jobs, each with a job number and a status (running or stopped) and an indication of which is the 'current' job (with a `+` sign)
- Jobs can be moved to the foreground or background with the `fg` and `bg`
- By default these act on the current job, but a job number can be given by preceding it with `%`
- If a job's status changes to stopped (waiting for input) or completed, the shell will report this at the next opportunity (usually between commands)

Process I/O and Terminals

- All Unix processes have standard input, output and error streams,
- For batch processes, these would typically be files, or possibly other processes (we will look at this next week)
- For interactive processes, these are terminal devices
- Originally, terminal devices files represented serial ports on the Unix machine (connected to 'dumb terminals')
- Nowadays, terminals usually represent CLI sessions with the machine via non-physical connections (shell windows or network connections)
- Linux supports virtual terminals (via Alt-F1–Alt-F8)
- The `screen` utility is a useful virtual terminal with multiplexing and attach/detach capability from terminals

4.4 Scheduling Regular Tasks

The cron System

- Often batch tasks need to be scheduled on a regular basis
- The cron system allows such tasks to be scheduled
- Allows both system tasks and user tasks to be scheduled
- Consists of:
  - A daemon (server process): crond
  - A system configuration file: /etc/crontab
  - Individual user crontabs in /var/spool/cron/crontabs
- Crontabs specify when and what jobs are to be run

System Tasks and /etc/crontab

- The /etc/crontab file is the system crontab
- Comment lines begin with #
- Each line has seven space or TAB-separated fields:
  - The first five are time fields, which are (in order): minute (0–59), hour (0–23), day of month (1–31), month (1–12) and day of week (0–7, or names)
  - Time fields may include the * wildcard, lists, or ranges (eg 1-7,15-21)
  - The sixth field is the username to run the job as
  - The seventh field is the command to run

```
# run as root on Sundays at 03:30
30  3  *  *  0  root  /root/backup.sh
# run on weekdays in April, every 8 hours
0  1,9,17  *  4  1-5  root  /root/taxbackup.sh
```

User Crontabs

- Depending on the system setup, users may also be able to schedule jobs with `cron`
- The `/etc/cron.allow` and `/etc/cron.deny` files can specify which users can/cannot schedule jobs
- Users use the `crontab` command with appropriate flag:
  - `-l` lists personal crontab
  - `-e` edits personal crontab
  - `-r` removes personal crontab
- Fields are as for system crontab, but without ‘user’ field

cron and Debian

- Debian extends the system crontabs in two ways for easy management of regular tasks
- Four directories in /etc: cron.hourly/, cron.daily/, cron.weekly/ and cron.monthly/ hold shell scripts which require to be run at the given intervals
- Additional (system) crontabs are included from files in /etc/cron.d/ — this allows packages to add their own crontab entries easily
- Debian also uses anacron by default — a version of cron which can be used in environments where the system is not always switched on

4.5 One-off Tasks and System Time

The at System

- The at system (under the control of atd - the at daemon) allows a job or sequence of commands to be run at a specific time
- Jobs are submitted using at with a time:
  - at *HH*:*MM*
  - at *DAY HH*:*MM*
  - at *YYYY*-*MM*-*DD HH*:*MM*
  - More complex time specifications are possible, eg: teatime + 3 days
- Commands can be typed into standard input (ending with Ctrl-D) or supplied in a separate file with the -f flag, eg:

```
root@debian:~$ at -f jobfile.sh 20:15
```

More at Commands

- The atq command shows jobs waiting in the at queue
- Each job has a number (in first column of list)
- Jobs can be removed using atrm with the job number
- The batch command runs the given job when system load is less than 1.5 (set when the at system is started)
- at has several queues (the -q flag can select which one):
  - a is the default
  - b is for batch jobs
  - c–z are used for running with increasing nice values
- /etc/at.allow and /etc/at.deny allow control over which users may use the at system

System Time: date

- How do cron and at know what time it is?
- System time can be examined using the date command (format may be specified with + along with field specifiers):

```
$ date
Thu Jan 2 03:06:21 GMT 2014
$ date +"%Y-%m-%d %H:%M:%S"
2014-01-02 03:08:22
```

- root may also set the system time manually with date

```
# date 01020310.30
Thu Jan 2 03:10:30 GMT 2014
```

- The cal command can be used to display a basic calendar

Keeping Time

- Usually system time is read from hardware clock on boot
- The hardware clock can be accessed with `hwclock`
- The hardware clock is usually running on UTC
- System time is calculated from this using the time zone
- Summer time adjustments are made automatically
- Even better than using the hardware clock is using Network Time Protocol (NTP)
- Normally there are NTP servers within an organisation
- NTP clients can set their time from these
- NTP servers use external servers to remain accurate

Summary

- Processes overview
- `ps` and `top`
- Signals and `kill`
- Process priority, `nice` and `renice`
- Job control: `jobs`, `fg` and `bg`
- Backgrounding tasks with `&`
- Terminating and pausing processes with Ctrl-C and Ctrl-Z
- Interactive and batch processes and terminal devices
- Scheduling regular tasks with `cron`
- System and user crontabs
- The `at` system
- System time and `date`