CST8177

Job and Process Management

Job and Process Management

- ***Topics to be covered***
  - Processes
    - Definition of a process
    - Different kinds of processes to manage on a running system
  - Process Management
    - process management utilities / commands
    - managing foreground & background processes
  - Managed processes
    - **`cron`** daemon
    - **`crontab`** - both file and command
    - **`cron`** system processes
    - **`at`** / **`atq`** / **`atrm`**
    - **`nohup`**
    - managing process **`nice`** levels

- In several situations, it is necessary to control current processes. Some examples include:
  - running a process in the background
    - running other commands/processes while the first one executes in the background
  - executing process at a specified time and date
    - running daily updates, weekly patches, and monthly backups
  - killing / restarting processes
    - cleaning up deadlocked processes, stopping, starting, or restarting a process currently running on the system
- It is important to remember that most processes submitted by a user will only run as long as the user remains logged in.

In order to do anything in Linux, you must:

- start an application
- use/run a command or script
- point & click to an icon
- do **something** to get the system to work for you

As far as Linux is concerned, all of the above and anything and everything that is happening at any point in time causes a "process" to execute or activate.

Upon logging in to the system, you actually end up starting several processes:

- login process (X or otherwise)
- authentication process (may include PAM)
- shell process once authenticated (usually BASH)
- login scripts (**`/etc/profile`** scripts, **`~/.bash_profile`**, etc)

- Even thought there might not be any users logged into the system, no connection the outside world, and no one at the console, there are **ALWAYS** processes running in Linux, even if it is only the kernel and the **`init`** daemon
- A Linux process consists of:
  - a single running program
  - a unique process identification number (**PID**)
  - an environment in which to run (**current shell environment or sub-shell**)
- Linux keeps track of processes using the PID, as well as the relationship between them (**child / parent**), not by the command names or user who started it, although it tracks that info as well

- It also maintains information as to the relationship between processes, if any (**child / parent** process)
- A parent process is simply a process that starts a new child process. A new or child process inherits the exported environment of its parent process
- A child process receives an exported copy of the parent’s environment. Thus, any changes the child makes to its environment does not automatically affect the parent process
- Many commands allow for viewing or listing the process currently running, and allow for examining the process current state and relationship to other processes.
- Managing these processes is one of the most important tasks given any administrator.

- A process can exist in one of many states:
  - **Runnable**
    - on run queue, in effect one of currently executing processes
  - **Sleeping**
    - waiting for event to occur to wake up and react/start up
  - **Stopped**
    - not currently executing, waiting to be killed or restarted
  - **Uninterruptable sleep**
    - cannot be woken until specific expected event
  - **Defunct** (**zombie**) **process**
    - just before a process dies, it notifies its parent and waits for an acknowledgment If it never receives it, its PID is not freed up but all other resources are freed. Zombies can usually be cleared by rebooting/restarting the system.

Process Display Commands

`ps`

- displays processes currently running on the system

  Many options are available (see man pages)

  Examples: **`ps -ef | less`** or **`ps aux | less`**

`top`

- displays processes currently running and some information about resource usage for each process

`free`

- Displays memory (RAM) and swap usage and by what

`vmstat`

- Displays virtual memory usage and by what

`pstree`

- generates a tree-like structure of process parent/child relationships

Managing Processes

- Processes can also be managed
  - **Foreground processes are interactive.**
    - Processes require you to wait until they are done before using the prompt again.
  - **Background processes are non-interactive.**
    - Processes run in the background, while you do other things; user generally cannot send information to the process directly

A System Administrator may:

- stop or restart execution of a process
  - Linux allows for direct control of a process status
- suspend a foreground process to background
  - A suspended process can be restarted or stopped
- move a process to background or foreground

  N.B.: output from background processes may show up on the screen, but does not affect the current information you are using
- terminate any process
  - Linux allows for hard termination of any process and its children. Terminating a child process does not directly affect the parent, but terminating a parent process might cause some grief to the child process(es).

Ctrl-C (also seen as **^C**)

- will terminate the process running in the foreground - the process the keyboard can access. This is not the same as **^Z**! (but see **kill -sigint**)

kill -signal pid

- allows for stopping/killing/restarting any process or daemon by sending its process ID (pid) a signal

kill -l

- List all the signals available

killall -signal processname

- allows for stopping/killing/restarting of all processes or daemons matching the name given

Note: Killing a parent process may or may not also kill all of its child processes. Killing a child process is not likely to also terminate its parent process.

Ctrl-Z

- suspends a foreground process to background (note that other Oses may use this as end-of-file on **stdin**, but we use **^D** for that purpose)

  Don't use ^Z to terminate a process!

jobs [%jobid]

- lists all background or suspended processes

bg %jobid

- Resume the suspended process **%jobid** in the background as if an **&** had been used to start it

fg %jobid

- moves process **%jobid** to the foreground

command1 options &

- runs command as background process

Background processing

- By using a **&** at the end of a command, the command(s) in the line will be executed in the background, returning the user immediately to the prompt.
- All output to **stdout** and **stderr** (usually the screen) will be displayed normally, possibly interleaved with output from other processes.
- Any requests from the background from **stdin** (usually the keyboard) will blocked from reading and placed in a stopped state.
- It is important to structure commands to be run in the background to avoid these limitations and ensure the proper execution of unattended commands.

The cron and anacron daemons

- System scheduling is done by the **cron** daemon, which searches **/var/spool/cron** for **crontab** files named after accounts in **/etc/passwd**; **cron** also searches for **/etc/anacrontab** and the files in the **/etc/cron.d** directory.
- **cron** examines all stored **crontab**s, checking each command to see if it should be run in the current minute.
- Each user can have their own **crontab** file, but it is often restricted to the root account (see **cron.allow** and **cron.deny** in crontab(1)).
- **anacron** runs commands with a frequency specified in days. Unlike **cron**, it does not assume that the machine is running continuously. Hence, it can be used on machines that aren't running 24 hours a day, to control regular jobs as daily, weekly, and monthly jobs.

- **anacron** reads a list of jobs from a configuration file, **/etc/anacrontab** (see anacrontab(5)). This file contains the list of jobs that **anacron** controls. Each job entry specifies a period in days, a delay in minutes, a unique job identifier, and a shell command. It uses a very different format from crontab(5).

```
SHELL=/bin/sh
PATH=/sbin:/bin:/usr/sbin:/usr/bin
MAILTO=root
RANDOM_DELAY=45
START_HOURS_RANGE=3-22

#days delay job-id       command
1     5     cron.daily  nice run-parts /etc/cron.daily
7     25    cron.weekly nice run-parts /etc/cron.weekly
@monthly 45 cron.monthly nice run-parts /etc/cron.monthly
```

Consider this entry in **anacrontab**:

```
#days delay job-id       command
1     5     cron.daily   nice run-parts /etc/cron.daily
```

Every day at 0300 (**START_HOURS_RANGE=3-22**) **anacron** will see if the job **cron.daily** has been run. If not (it tracks the last date it did run in a separate file), it waits for the 5-minute delay specified plus a random delay between 0 and 45 minutes (**RANDOM_DELAY=45**). It will then run the shell command **nice**.

This is a binary executable with the form:

```
nice [OPTION] [COMMAND [ARG]...]
```

In this case, **nice** has no options, and will run the shell script **run-parts** with a priority of 10 (the default) instead of the 0 of a typical process.

The script looks through the file **/etc/cron.daily** and executes each program found there. My system at home has 12 entries which run one at a time until all are done.

The other entries work much the same way, at weekly (7 days) and monthly (which can be 28, 29, 30, or 31 days) intervals.

- The **crontab** file has a specific format (_ means space):
  Minute _ Hour _ Day of Month _ Month _ Day of Week _ [username] _ command list

Minute	1 to 59
Hour	0 to 23
DOM	1 to 31
Month	1 to 12
DOW	0 to 7 *
Userid	System crontab only
Command(s)	Or script name

- * Both 0 and 7 represent Sunday
- Ranges of values can be specified using the dash (**-**).
- Comments can be placed in the **crontab** file by putting the **#** symbol in front of them.
- Blank lines are allowed.

- All fields must contain a value of some valid kind
- Field are separated by one or more spaces
- Continuous periods are indicated with the asterisk (*). In any given field, if no specific entry is required, the asterisk may be used in that field to indicate that the whole range is valid.

```
# .---------------- minute (0 - 59)
# |  .------------- hour (0 - 23)
# |  |   .--------- day of month (1 - 31)
# |  |   |  .------ month (1 - 12)
# |  |   |  |  .--- day of week (0 - 6)
# |  |   |  |  |
  0  6   1  *  *  /home/allisor/bin/payday "$(date)"
  1  6  15  *  *  /home/allisor/bin/payday "$(date)"
```

- The main **crontab** file **/etc/crontab** should NEVER be edited manually; you don't want **crond** to fail.
- Construct a **crontab**: as root, in **/etc/cron.d**; as a user, in **/var/spool/cron**
- A user **crontab** file can be created by using the standard editor (**vi**) to create a plain text file with proper **crontab** fields as above, or using the **crontab** command
- It is recommended to name the file something distinctive (e.g. **allisor.cron**) and for it to be stored in your home directory.
- The **crontab** file can then be submitted to the **cron** daemon using the **crontab** command.
- The **crontab** command can also be used to add, edit, list, and remove individual **crontab** files submitted previously.

The crontab command

crontab [-u username] -e | -l | -r | file

-u user is the user login name; used only by root

-e edits the **crontab** file; created if needed

-l lists the **crontab** file content

-r deletes the **crontab** file

file submits the **crontab** file to the **cron** daemon for placement in **/var/spool/cron/usename**

crontab -l

lists contents of the current user's **crontab** file

crontab -u allisor -e

edits the **crontab** file for user **allisor** in **vi**

crontab allisor.cron

submits the **crontab** file **allisor.cron** to **cron**

crontab -r

remove the **crontab** file for the current user

Scheduling Processes

- System processes use the **cron** and **anacron** capabilities to invokes processes and scripts maintained in periodic directories:

  /etc/cron.hourly (not used by **anacron**)
  /etc/cron.daily }
  /etc/cron.weekly } **anacron**
  /etc/cron.monthly }

- Access to the **cron** and **at** systems can be managed
  - **/etc/cron.allow** and **/etc/cron.deny** control access to the **cron** system
  - **/etc/at.allow** and **/etc/at.deny** control access to the **at** commands **at**, **batch**, **atq**, and **atr**

Running Processes after `logout`

Processes running in the background by shell are usually allowed to continue after **`logout`**. Non-redirected output is lost, however.

`nohup` command

- allows for saving this output by appending it to the **`nohup.out`** file in the user's home directory.
- terminals need to shutdown with **`exit`**

`at time`

- specifies a time for a set of commands (read from **`stdin`** with an **`at>`** prompt; **`^D`** to end) to be run as the user, whether the user is online or not.

`atq`

- lists pending **`at`** jobs for the user (root gets everyone)

`atrm at-jobid`

- removes specific **`at`** jobs by job number if owner or root

The at command

The **at** command can be used to submit single-run jobs, scripts, or commands to the **cron** daemon to be executed at a later time (anywhere from 10 minutes to a few days). Anything having a more frequent requirement than that or to be done on a regular basis should be done through **crontab**.

at time

- specifies the time for commands to be run as the user, whether the user is online or not.
- commands are read from **stdin** at an **at>** prompt
- non-redirected output is automatically mailed to user's local mail account
- used for one time run scenario only
- provides job number given when queued.

The at command

at [-f script] [-m -l -r] [time] [date]

-f script actual script name to submit

-l lists jobs waiting to run (same as **atq**)

-r jobno removes a job (same as **atrm**)

-m mails the user when job is finished

time H, HH.MM, HH:MM, H:M format

date MonthDay format, day name, or **today** or **tomorrow**

at 3.00pm tomorrow -f /apps/bin/db_table.sh

- runs **db_table.sh** tomorrow afternoon at 3

at -l or **atq**

- lists pending **at** jobs for user

at -r at-jobid or **atrm at-jobid**

- removes specific at jobs from user's queue

The **nohup** command

- The **nohup** command can be used to submit jobs to **cron** you don’t think can be completed by the time you log out. The process will continue processing even after you log out.
- This can be particularly useful for a long-running tasks and/or when you can't wait around while the command executes.
- **nohup command &** is the command structure to accomplish this.
- If you DO log out of the terminal, any output from the job will be redirected to a file called **nohup.out** in your current directory by default.
- Terminals need to be closed with **exit** command.

Process Priority

- Processes are scheduled with a 0 (zero) priority by default, unless the process itself is coded to select a higher priority.
- Priority range from -20 (**highest**) to 19 (**lowest**)

`nice [-n adjustment] command`

- allows for fine tuning the priority when starting a process or daemon on the system
- only root can decrease a process priority
- default value of **`10`** for **`-n`** if omitted

`renice [-n pri] [-p pids] [-g pgrp] [-u user]`

- Allows for altering current process priority

`-n pri` priority value as above

`-p pids` alter only listed processes priority, not any child processes

`-g pgrp` alter the priority of an entire process group