Lecture -10

Managing Process and Finding Process

Listing Processes:

- View process information with **ps**
  - Shows processes on all terminals by default
  - -a includes processes on all terminals
  - -x includes processes not attached to terminals
  - -u prints process owner information
  - -f prints process parentage
  - -o **property**,... prints custom information:
  - Pid, comm, %cpu, %mem, state, tty, euser, ruser

Finding Processes:

- **Most flexible: ps options | other *commands***

```
ps axo comm, tty | Grep ttyso
```

- By predefined patterns: pgrep

```
$ pgrep  -u root

$ pgrep  -G student
```

- By exact program name: pidof

```
$ pidof bash
```

Viewing Specific process Information

Since there may be hundreds of processes on a common technique to locate a specific process is to send output from **pis** to **grep**

```
[student@stationX ~] $ ps axo pid, com I grep ‘’cups’’

     2734 cupsd
     3502 eggcups
```

Compare the above output with that from pgrep

```
[student@stationX ~] $ pgrep cups

2734

3502
```

A more exact method of obtaining PID’s for processes is the **pidof.** Which matches exactly on the program name specifyied and therefore requires that you know the specific program name?

```
[student@stationX ~] $ pidof  cupsd

2734
```

Signals:

- Most fundamental inter process communication
    - **Sent directly to processes, no user interface required**
    - **Program associate actions with each signal**
    - **Signals are specified by name or number when sent:**
        - Signal15, TERM (default) –terminate cleanly
        - Signal 9, KILL, -terminate immediately
        - Signal 1 ,HUP –RE-read configuration files
        - Man 7 signal show complete list

Signals

Signals are simple message that can be communicated to process with command like **kill.** Which will be discussed on the next page? The advantage of signals is that they can be sent to a process even if it is not attached to a terminal and display an interface. So a web server or a graphical program whose interface has ‘’frozen’’ may still be shut down by sending it the appropriate signal.

Though in most cases software developers can associate any action with the reception of a particular signal. Almost all signals have standard meanings. Signal can be specified by their name. such as kill. Or by their number. Such as 9. A called detailed list of signals including names. Numbers and meanings can be displayed with **man 7 signal.**

Sending signals to processes:

- By PID: **kill** [signal] pid…
- By Name: **Killall** [signal] comm…
- By pattern: **pkill** [-signal] pattern

The following are all identical and will send the default term signal to the process with PID number 3428:

```
[student@stationX ~] $ kill 3428

[student@stationX ~] $ kill -15 3428

[student@stationX ~] $ kill –SIGTERM 3428

[student@stationX ~] $ Kill –TERM 3428
```

Scheduling Priority:

- Scheduling Priority determines access to CPU
- Priority is affected by a process’ nice value
- Values range from -20 to 19 but default to 0
    - Lower nice value means higher CPU priority
- Viewed with **ps -0 comm, nice**

Schedule priorities

Every running process has a scheduling priority: a ranking among running process determining which should get the attention of the processor. The formula for calculating

This priority is complex, but users can affect the priority by setting the ‘’ niceness’’ value, one element of this complex formula. The niceness value defaults to zero but can be set from -20 (least nice, highest priority) to 19 (most nice, lowest priority).

Altering Scheduling Priority:

- Nice values may be altered…
    - When starting a process:

```
$  nice –n 5  command
```

    - After starting :

```
$  renice  5 PID
```

- Only root may decrease nice values

Adjusting priorities for programs

To set the niceness value to a specific value when starting a process, use **nice-n**

[student@stationX ~] $ nice -n 15 **myprog**

Non-privileged users may not set niceness value to less than zero: that is, they not request a higher than normal priority for their processes. Only root may allocate additional CPU clock cycles to a process.

Altering priorities of running programs

All users may raise the niceness (reduce the priority) of their own jobs using the renice command:

```
[student@stationX ~] $ renice 15 –p pid
```

Note that non-privileged users may start a process at any positive nice value but cannot lower it once raised. Root is permitted to reduce the niceness (raise the priority) of currently running process

```
[root@stationX ~] $ renice -15 –p pid
```

Interactive Process Management Tools:

- CLI: **top**
- **GUI: gnome-system-monitor**
- Capabilities
    - Display real-time process information
    - Allow sorting, killing and re-nicing

*The **top** command*

Running **top** present a list of the process running on your system, updated every 5 seconds. You can use keystrokes to kill, renice and change the sorting order of process. That are in the running state are highlighted and you can colorize processes and create multiple windows to view more than one sorted list of process at a time, press the? key while in top to view the complete list of hotkeys. You can exit top by pressing the q key.

The gnome-system-monitor command

The gnome-system-monitor, which can be run from the cc:: sole or by selecting Application system Toolsystem Monitor from the menu system, is a graphical tool that also offers the ability to view sorted lists of process and to kill or renice those process.

By default the tool only display processes that are in the ‘’Running’’ state, However this can be changed by selecting All process or my process from the dropdown menu in the upper-right. The preferences dialog (Editpreferences) allows you to customize which fields are displayed and you can sort by a particular field by clicking on its heading. You can send a process the TERM signal by right-clicking on it and selecting End process or the kill-9) by selecting kill process. A process can be re-niced by right-clicking on it and selecting Change priority.

Scheduling a process To Execute Later:

- One-time jobs use at, recurring jobs use **crontab**

Create	**At time**	**crontab –e**
List	**At-l**	**crontab –l**
Details	**At –c jobnum**	**N/A**
Remove	**At –d jobnum**	**crontab –r**
Edit	**N/A**	**crontab -e**

- Non-redirected output is mailed to the user
- Root can modify jobs for other users

Schedule a one-time job with at

Commands are entered, one per line, and terminated with a Ctrl-d on a line by itself. In the following example, you would like to compare network services on your machine to a baseline that is stored on non-writable media:

```
[student@stationX ~] $ at 0200
```

- netstat –tulpn I diff - /media/cdrom/netstat_baseline
- Ctrl-d

```
Job 1 at 2007-01-08 22:45

[student@stationX ~] $ at  -1

1                    2007-01-09  02:00 a student
```

At-1 (an alias to **atq**) lists the job currently pending, **at –c jobnum** cats the full environment for the specified job number, and **at-d jobnum** deletes the job.

root can modify other user's jobs by first getting a login shell as that user (**su-username**).

The time argument has many formats which are illustrated by the following example.

At 8:00pm December 7	**at 7 am Thursday**
At midnight + 23 minutes	**at now + 5 minutes**

See **man at** for information.

Schedule recurring jobs with **crontab**

The cron mechanism is controlled by a process named **crond.** This process runs every minute and determines if an entry in users cron tables need to be executed. If the time has passed for a entry to be started, it is started. A cron job can be scheduled as often as once a minute or as infrequently as once a year.

root can modify recurring jobs for any user with **crontab –u username** and any of the other options. Such as –e

See **man crontab** for more information.

Crontab File Format:

- Entry consists of five space-delimited fields followed by a command line
    - One entry per line, no limit to line length
- Field are minut, hour, day of month, month and day of week
- Comment lines begin with #
- See **man 5 crontab** for details

Crontab examples

Entry fields can be separated by any number of tabs or spaces. Valid field values are as follows:

Minute	0-59
Hour	0-23
Day of month	0-31
Month	0-12
Day of week	0-6 (0=Sunday)

Multiple values may be separated by commas. An asterisk in a field represents all valid values. A user’s crontab may look like the following:

```
# Min    Hour     DoM     Month    DoW     Command
   0       4        *        *      1, 3, 5     find  ~ -name core I xargs rm –f  { }
   0       0        31       10      1, 3 ,5     mail –s  ‘’boo’’  $LOGNAME <
boo.txt
   0       2        *        *        *        netstat –tulpin   I  diff
-/media/cdrom/baseline
```

Grouping commands

- Two ways to group commands:
    - Command: date;who |wc-l
        - Commands run back to back
    - Subshell: (date; who | wc-l) >> /tmp/trace

All output is sent to a signal STDOUT and STDERR