UNIVERSITY OF THE WEST of SCOTLAND

COMP09024 Unix System Administration

Laboratory 4: Process Management

Process Management

Learning Outcomes

- **Monitoring processes: `ps, vmstat, pstree, top, and uptime`**
- **Prioritising processes: `nice, renice`**
- **Configuring the cron facility: `crontab`**

In this lab we will examine commands for monitoring and prioritising processes. We will also look at some commands for scheduling tasks.

Processes

Monitoring Processes

The most important command for the display of processes within the system is called **`ps`**. Type the following:

```
[root@UWS ~]# ps
```

and then

```
[root@UWS ~]# ps -ef
```

Examine the first line that is displayed and try to understand the meaning of the column headers.

Questions

- **Q4.1)** What is the difference between **ps** and **ps -ef** ?
- **Q4.2)** Explain the meaning of the column headers **UID**, **PID**, **PPID** and **CMD**
- **Q4.3)** Write down the process identifiers of at least two processes that have the PPID of 1, representing the init process
- **Q4.4)** Is the process **ps -ef** listed itself? If yes, what is its **PID** on the system?
- **Q4.5)** How many processes in total are currently being run by the root user?

As you may have seen, **ps** is a very basic but also very efficient command for examining executing processes on a Linux system. Only the superuser is able to see all active processes.

Now it's time to start a small background process like **more**.

```
[root@UWS ~]# nano &
```

Hit the return key once or twice if necessary then type in the terminal window:

```
[root@UWS ~]# ps
```

Questions

- **Q4.6)** What is the **PID** of the terminal and the **nano** (process)?
- **Q4.7)** What is the function of the ampersand **&** in the above command (Tip: What happened to the command prompt of the new window, after we typed in the command)?

Now create another process from this window by typing:

```
[root@UWS ~]# vi &
```

Have a look using the **ps -ef | grep 'nano'** and **ps –ef | grep 'vi'** command lines to answer the following question

Question

- **Q4.8)** What is the **PID** and **PPID** of the terminal and the two processes **nano** and **vi**? What is the relation between the calling terminal shell and the two processes?

To terminate a process, Linux uses the **kill** command. To actually kill a process you have to specify the **PID** of the process to the **kill** command, *not the name of the process*.

Please remember that **<PID>** is a variable that stands for process identification number. Please do not type in the command line as given below without replacing the variable **<PID>** with its actual value, something like: **5654**.

Please look again for the **<PID>** of the **vi** process and do the following:

```
[root@UWS ~]# kill <PID>
```

Question

- **Q4.9)** Did this kill the process **vi**?

Now try:

```
[root@UWS ~]# kill -9 <PID>
```

Question

- **Q4.10)** Use the **man kill** command and check for the meaning of the qualifier **-9** to find out why the process has been killed this time?

Next execute the **top** command background, later **bash** and later **top** again:

```
[root@UWS ~]# top &

[root@UWS ~]# bash

[root@UWS ~]# top &
```

Question

- **Q4.11)** What is the **PPID** and the **PID** of each **top** process?

Now kill the calling mother shell using the command **kill –9**. The **–9** qualifier ensures, that the process gets killed completely thus avoiding the possibility of any system lock-up. Normally it is not really necessary to use this qualifier, however consider **'–9'** as an option to kill a process *with extreme prejudice*.

```
[root@UWS ~]# kill -9 <PID of calling shell>
```

Question

- **Q4.12)** What happened to the two **top** processes? Were they killed too? What is the new **PPID** of the two **top** processes? Can you therefore explain what happens to orphaned processes?

As you may have concluded, **ps** is a very powerful command, allowing you to gather a great deal of information about what is executing on a running computer, especially when used together with other commands (piping).

To further observe processes in action, let us create a number crunching shell script as shown below. Make sure that you are the superuser and in the **/root** directory.

```
[root@UWS ~]# cd /root
```

```
[root@UWS ~]# nano cruncher
```

Continuing by typing in your editor the following small shell script:
(Please take care to avoid typos and use the exact signs and ASCII characters shown)

```
#!/bin/bash
for ((i=0; i<=400; i++))
do
   echo `expr $i + 1`
done
exit 0
```

Before you proceed please make sure you have used the exact symbols shown above, in particular you must use backward quotation marks: `. The position of spaces is also critical.

The cruncher shell script just adds a 1 to all numbers from 1 to 400 and displays the result (**`echo`** command!) on the screen. That's all. Once you have saved the file as: **`/root/cruncher`**

Make sure it is executable by:

```
[root@UWS ~]# chmod +x cruncher
```

and execute it for the first time:

```
[root@UWS ~]# ./cruncher
```

Now run it in background redirecting the standard output to null and after that issue the **`top`** command:

```
[root@UWS ~]# ./cruncher >/dev/null &

[root@UWS ~]# top
```

`top` displays statistics for all processes and then the processes themselves in more detail: e.g. **`PID USER`** and **`PR`**. The column **`PR`** stands for the priority that is connected (not identical!) to the so called nice (**`NI`**) number of a process. We will explain this later in more detail. For the moment it is important that you know, that the *highest* priority is denoted by the *lowest* number.

Questions

- **Q4.13)** Describe the output that you see. How much approximate **`CPU`** time (in %) does the **`cruncher`** program use while it is still running? (resize the terminal to see COMMAND column).
- **Q4.14)** In what order does the **`top`** command display the processes?
- **Q4.15)** How many processes are running on your machine in total? How many of them are sleeping?
- **Q4.16)** The '**`Tasks`**' row has space for *zombie* processes. Do you have any idea what a zombie in a Unix-like system might refer to?
- **Q4.17)** What is the priority number of the program **`cruncher`**? Does it change at all?

- **Q4.18)** What is the highest priority that any of the processes gets assigned?
- **Q4.19)** What priority level is given to the majority of the processes?
- **Q4.20)** Is any swap space used at this time? (Tip: read remarks on the **`top`** command)

There are some more commands, which allow you to monitor and manipulate the performance of certain processes. This may become quite important for you, if e.g. one single process slows down the performance of your PC substantially. In a case like this CPU usage is the first factor that you should consider. The best command to do this is **`vmstat`.**
`vmstat` displays average values for a lot of features including: memory , CPU, I/O etc.

Type:

```
[root@UWS ~]# vmstat 10 4
```

Question

- **Q4.21)** Explain briefly what kind of information **`vmstat`** displays here? What do the two qualifiers: 10 and 4 stand for? (if unsure look it up using the **`man vmstat`** command)

To further explore **`vmstat,`** continue with:

```
[root@UWS ~]# vmstat 10
```

and run the crunching application **`cruncher`** 3 times in background**.**

Monitor the output of **`vmstat`** , especially the last four columns that have the common heading called **`cpu`.**

Questions

- **Q4.22)** What numbers change in the **`vmstat`** display? What does the number in the '**`r`**' column represent?
- **Q4.23)** What happens, while running the application programs, to the last three entries shown below the **`cpu`** heading that are labelled: **`us sy id`**? To which value do they always add up? Can you interpret their meaning?
- **Q4.24)** What happens to the number associated with the **`us`** column once all **`cruncher`** sessions have terminated?

As you may have concluded, **`vmstat`** is a good command for obtaining a general overview of system performance. However, there are some other useful commands worthy of mention. Try:

```
[root@UWS ~]# uptime
```

Question

- **Q4.25)** What information does **`uptime`** provide?

Another useful command is **`pstree,`** which displays system processes in a tree-like structure. This is very useful for revealing relationships between processes and for giving a quick pictorial

snapshot of what is currently running on a system. In general all processes are listed by command name, and child processes appear to the right of parent processes. Type:

```
[root@UWS ~]# pstree
```

Questions

- **Q4.26)** What is the name and the significance of the process that is displayed in the leftmost position (you may have to scroll down the terminal window to see the entry)?
- **Q4.27)** Write down all linking processes between **`init`** and **`pstree`**. Could you obtain the same information using the **`ps`** command?

Thus far we have reviewed some basic tools for monitoring processes under Linux: **`ps`** , **`vmstat`**, **`uptime`** , **`pstree`** and **`top`**.

Changing Process Priority

All Unix-like systems use a priority based scheduling algorithm to distribute CPU resources among competing processes. All processes have an execution priority assigned to them – an integer value that is dynamically computed and updated on the basis of several different factors. When the CPU is free, the scheduler selects the next most favoured process to begin or resume executing. This corresponds to the process with the *lowest* priority number. An upper limit for process priority is set by the system (or sometimes by the user) as determined by the so-called *nice number* (where *niceness* relates to a process's ability/willingness to share CPU. Thus a low number equates to a high priority). Multiple processes with the same nice number are placed into a run queue for that priority level. After a process has had a slice of the CPU 'cake' it is assigned a lower priority thus placing it temporarily at the end of the run queue.

From the scheduler's viewpoint we can classify processes into just three types:

Processes where *no user interaction* is required: For example processes involving database updates, scientific number crunching or perhaps program compilation. These processes are given a low priority by the scheduler and left in the background. Please remember that if you schedule a process to run in the background and you choose the default priority (not overridden with the **`nice`** or **`renice`** commands) then your process will run very slowly, especially on a busy machine as all other processes will have a higher priority by default.

Processes that are *waiting for I/O*: If a user response is required a process will `wait()` for a response, but once a response is received the process in question is reinstated on the run queue. It is said that the delay from user response to result should be only between 50 and 150 ms.

Real-time processes: For these the scheduler drops everything else, to the point that if a process is already on the processor, the scheduler switches the process out to the so-called TASK_RUNNING state in which it waits to run again on the CPU - this in order to allow for the real-time process to run. Generally these are system processes (daemons) but they could also be processes that interact with hardware devices**.**

To examine process *nice numbers* (add the –l qualifier to the **ps** command) and refer to the output column headed **NI**:

```
[root@UWS ~]# ps -elf
```

Question

- **Q4.28)** What are the nice numbers of some processes: e.g. **`init`** , **`kthread`**, **`cron`** and **`udevd`**? (Tip: use the **`grep`** command together with the **`ps -elf`** command!)

You can set the nice level of a command at start time by typing:

```
nice  -number command
```

The supplied number can vary from -20 to +19, but remember the higher the nice number the lower the priority (nice means nice to others!). To give a process the highest priority the user initiates the job as follows (yes it contains two minus signs!):

```
nice  -n -20 command
```

The **`--20`** is reserved for top priority. To give a process the lowest priority the user initiates a job as follows:

```
nice  -n 19 command
```

The **`-n`** and **`--adjustment=N`** equivalent. For example:

```
nice –adjustment=19 command
```

To examine the effect of using different nice numbers let's run the **`cruncher`** shell script again and measure the execution time for the script using the **`time`** command.

```
[root@UWS ~]# cd /root
[root@UWS /root]# time ./cruncher >/dev/null
```

While cruncher is running in background, use the **top** command to find out the default nice number allocated to **`cruncher`** by the system. Take a note of the final output (of the time command) once **`cruncher`** has finished executing. The output should look something like:

```
real   1m9.995s
user  0m12.453s
sys   0m22.433s
```

(where '**`m`**' stands for minutes, '**`s`**' for seconds)

The **real** time is the elapsed time according to the system clock, the **user** time is the execution time for the code inside the script and the **sys**(tem) time is the time taken by kernel code (system calls). CPU time can be found by adding together user and sys time.

Questions

- **Q4.29)** What is the nice number (**NI**) given by the system to the **cruncher** script?
- **Q4.30)** Note the actual outputs for **real**, **user** and **sys** times for the **cruncher** script after it has finished executing.

The above shows the performance characteristics of **cruncher** when using the system defaults. Now let us change the default with the **nice** command. First of all let us be nice to others!

```
[root@UWS ~]# time nice  -n 19  ./cruncher
```

Questions

- **Q4.31)** Note the **real**, **user** and **sys** times again and comment on any difference in the **real** time value
- **Q4.32)** Why do you think, **user** and **sys** time are not so different when compared to the values obtained after the first run of **cruncher**?

Now let's get cheeky to the other processes that run on the system.

```
[root@UWS ~]# time nice  -n -20  ./cruncher
```

Take care not to put a space between or after the two hyphens!

Questions

- **Q4.33)** Note the actual outputs for **real, user** and **sys** time again. Comment on any improvement in performance.
- **Q4.34)** You may have noticed that during the execution of the **cruncher** script with such a low nice number, the screen seemed to freeze. Is it a good idea, to give a standard process such a low nice number?

We have been viewing performance on a fairly quiet system where the **nice** command had little effect. To see how the command impacts a busier system we will start TWO competing processes in background, namely two instances of **cruncher**. You may find it helpful to first edit the **cruncher** script to increase the number of iterations from **400** to say **600** by changing the expression **i<=600** in the script's for-loop. Remember to save the amended script as **cruncher** again after making any edits.

Now, run the following three commands in a row:

```
[root@UWS ~]# time nice -n 19 ./cruncher >/dev/null &
```

and run a second cruncher:

```
[root@UWS ~]# time nice -n -20 ./cruncher >/dev/null &
```

Now please run the **top** command to monitor the processes.

```
[root@UWS ~]# top
```

Question

- **Q4.35)** Note the actual outputs of the **`time`** command for **`real`**, **`user`** and **`sys`**. How do the time differences between **`nice 19`** and **`nice -20`** compare to the time differences on a quiet system? (Try to generalise rather than give the actual times. E.g. your answer should look like: *On a heavily loaded system the* **`nice -20`** *command was 3 times faster than the* **`nice 19`** *run and on a quiet system 'only' 1.25 times* faster.

You can also change the priority of a process after it has started. The corresponding command is called **`renice`**. However, there are certain obvious restrictions attached to **`renice`** :

- You can only **`renice`** a process you own
- The superuser can **`renice`** any process
- Only the superuser can increase the priority of a process

To demonstrate this, please start the low priority **`cruncher`** again in a terminal:

```
[root@UWS ~]# time nice -n 19 ./cruncher >/dev/null &
```

Use the **`ps`** command in another terminal to find out the **<PID>** of the process (or view it via the **`top`** command). Change the process priority with the following command:

```
[root@UWS ~]# renice -20 <PID>
```

PID should be the number of the process, please do not type **<PID>** literally! Also note that unlike the **`nice`** command the **`renice`** command does not expect a leading – in front of the priority.

Questions

- **Q4.36)** Note the output of the **`renice`** command.
- **Q4.37)** Estimate the performance gain of the above action. E.g. the program seems to run twice as fast in real time units after the 'renicing'.

In a later laboratory we will create a bash shell script to **`renice`** a process given the nice number, and the **<PID>,** or even just the nice number and process name.

Scheduling Tasks

The cron Facility

The most important tool for scheduling Linux commands is called the **`cron`** facility. We call it *facility* rather than command, because for historic reasons its structure is less straightforward than other Linux commands. The **`cron`** facility is widely used by systems managers and normal users to

schedule unattended tasks (e.g. routine maintenance, backing up or systems monitoring) To see the the current **cron** jobs associated with the root user type (in a root terminal window):

```
[root@UWS ~]# crontab -l
```

This command should print out the current **cron** jobs on the screen associated with the calling user. Read the **crontab** manuals to find out which file you have to create or update in order to allow root to use the cron facility. (Tip: the necessary files are in **/etc**).

Questions

- **Q4.38)** Which file contains a list of users who are allowed to submit requests to the **crontab** facility?
- **Q4.39)** Which file contains a list of users who are NOT allowed to submit request to the **crontab** facility? If a user's name appears in both lists, which list takes precedence?

Initially your system should be clear of any user related **cron**-jobs. As it is not permissible to edit the appropriate **cron** entry directly we have to create a file to hold the commands to be invoked by the cron daemon. Let us name the file **/tmp/entries**. Please type:

```
[root@UWS ~]# nano /tmp/entries
```

If the **nano** does not appear, use the menu to invoke it or use a different editor.

For illustrative purposes, we will set up a job to monitor disk usage in the **/home** directory at 5 minute intervals. Open another terminal window and type:

```
[root@UWS ~]# du -k /home
```

Question

- **Q4.40)** Explain the result of the above command in detail. (Tip: if you are unsure, use the man pages to check the meaning)

To obtain a sorted list, let us sort the output by the number of allocated blocks (=size) like:

```
[root@UWS ~]# du -k /home | sort -nr | head -5
```

Question

- **Q4.41)** What does the piping in the **sort –nr** and the **head –5** do to the **du -k /home** output? Explain in detail.

Having established that **du –k | sort –nr** is a good command to check the disk usage of a directory it is time to include this command in the **cron**-table.

To do so, let us go back to the editor and save the following line in the file **/tmp/entries:**

```
0,5,10,15,20,25,30,35,40,45,50,55 * * * * du –k  /home | sort –nr | head –5 >> /root/test.dat
```

Be sure to type a carriage return at the end of the line and take extra care with the syntax!

What does this line mean?

- The 1st part (**minute**), corresponding to the sequence of numbers separated by commas, tells **`cron`** to perform the task (**`du -k | sort -nr | head -5`**) every 5 minutes. The minutes have an input range of 0-59 (not 60!) so the range =[0,59].
- The 2nd part (**hour**), represented here by the wildcard '***'**, gives the hour of the day in which the task has to be performed. Range=[0,23] and wildcard '*' meaning *every* hour.
- The 3rd part (**day**) represents the day of the month. Range=[1,31] and wildcard '*****' meaning *every* month.
- The 4th part (**month**) represents the month of the year. Range=[1,12] and wildcard '*****' meaning *every* month. Also accepts Jan, Feb, Mar, etc.
- The 5th part (**weekday**) represents the day of the week (e.g. 0=sunday, 1=Monday). Range [0,6] and wildcard '*****' meaning *every* day. Also accepts Mon, Tue, Wed, Thu, Fri, Sat, Sun.

(Note: There is an extended format that includes a 6th (year) field. Range=[1900-3000]).

the final entry **`du -k /home | sort -nr | head -5 >> /root/test.dat`** is the command line that we wish to execute.

Once we enter this line in the **`/tmp/entries`** file we have to upload the file to the actual **`cron`**-table which is then read by the **`cron`** daemon (every minute!). To do so, we have to introduce the last step in 'croning', the **`crontab`** command itself:

```
[root@UWS ~]# crontab /tmp/entries
```

This should upload our command with the appropriate time settings in the crontab file. To check type:

```
[root@UWS ~]# crontab  -l
```

Now our line should be visible (if not there was a mistake).

Question

- **Q4.42)** Note the output of the above command.

As previously stated the cron facility is not especially straightforward. To examine the results set in the **`/root`** directory and wait for the occurrence of the file **`test.dat`** (which should take -at the most- 5 minutes). Once and again just try:

```
[root@UWS ~]# ls -lt test.dat
```

Repeat this command several times and check the contents of **`/root/test.dat`** every so often within the next 30 minutes. (Tip: it should change every 5 minutes)

Questions

- **Q4.43)** What happens to the size of **`/root/test.dat`** over time? Do you see any danger in having a steadily growing output file in a directory?
- **Q4.44)** What would be the entry in **`/tmp/entries`** to perform the command once every day at 23:15 (=11:15pm)?
- **Q4.45)** What would be required to perform it at 4:30pm on the 21st of January every year?
- **Q4.46)** Find an alternative **`cron`** syntax that could be used to specify that a job be performed every 5 minutes. Note the alternative notation as the answer in your logbook.

Actually the system already has some build in cron tasks that are silently running in the background. To see them set in the **`/etc`** directory:

```
[root@UWS ~]# cd /etc
```

and list all files starting with **`cron`** by typing:

```
[root@UWS /etc]# ls cron*
```

Questions

- **Q4.47)** In which different time-periods are the system **`cron`**-jobs ordered? (Tip: Check the names of the corresponding directories)
- **Q4.48)** Note an example of a **`cron`**-job service task that is found in the **`cron.daily`** directory.

To repeal the current **`cron`**-job enter the command:

```
[root@UWS /etc]# crontab -r
```

Question

- **Q4.49)** What is the exact meaning of the **`-r`** character in the **`crontab -r`** command line?

– END OF LAB –