Alternative OS – Lecture 6

Plan:

1. Jobs
2. Suspending jobs
3. Running jobs in the background
4. The **nice** utility
5. Displaying system processes (**ps)**
6. Killing your jobs (**kill**)
7. Conclusions

1. Jobs

Unix is a *multitasking* operating system. That means it can run several *programs* simultaneously. If we start a program directly from the *shell* the program is called a *job*.

For example:

% **man** jobs

The **jobs** command gives the list of currently running jobs.

2. Suspending jobs

We can *suspend* current job by pressing **Ctrl+Z.** Suspend your *man page:*

Press **Ctrl+Z**

In order to know what jobs are running we should issue following command:

% **jobs**

You will see the list of currently running *jobs*. In order to bring the suspended job back to the *foreground* use **fg** command.

In the list given by the **jobs** command, your recent job is marked with **+** symbol. Issuing **fg** without an argument will bring your *recent* job to the foreground.

Syntax of fg:

fg *%job_number*

Example:

% **fg** *%2*

You can also use a shortcut:

% **%2**

You can kill the jobs you don't need:

% **kill** *%1*

Syntax: kill %job_number

3. Running jobs in the background

There are jobs, like compiling your own programs, playing music etc. that can perfectly well be run in the *background*.

If want a program run in the background first start it, and then *suspend* it pressing **Ctrl+Z,** in order to reclaim the control over the terminal. If you start, and the suspend **xemacs**, the **xemacs** window will become indifferent to your input. That happens because the application is *suspended*. In order to run it in the background submit the **bg** command:

% **bg**

This command tells the shell to execute last suspended program in the background and leave the control over the terminal to the user. The editor is back running, and we are back to control over the terminal! You can kill the *job* of **xemacs:**

% **kill** *%1*
% **jobs**

You may have more than one *job suspended,* and thus you have to be able to make any of that jobs get running in the *background*.

% **xemacs**

Ctrl+Z

% **xeyes**

Ctrl+Z

% **jobs**

Bring the **xemacs** back to life:

% **bg** %1

Kill all the jobs:

% **kill** %1 %2

As you noticed we usually know what jobs we want to run in the background even before we actually start them. We can start a job in the background appending the **&** *(ampersand)* to the end of the command:

% **xemacs&**
% **xeyes&**

There are, however several things we should always keep in mind if we run a program in the background:

1. How to feed the input to the program in the background?
2. Where to keep the output of the program running in the background?
3. How to keep track of the jobs we put in the background?

First two issues are addressed by means of the *input and output redirection*. The third issue is a bit trickier. In the most cases the **jobs** command helps out. This should we use for now.

An example of quite time consuming command is **find:**

Syntax:

find *directory file_name*

We can run it in the background redirecting its *output* to a file:

% **find** / *dir.txt > found.txt &*

In order to have the errors printed also in the *file* instead of the *terminal display* we can redirect the *stderr* too:

% **find** / *dir.txt >& found.txt &*

or

% **find** / *dir.txt > found.txt 2> found.txt&*

4. The nice utility

You might like to decrease the *cpu time* consumed by the utility you'd like to run in the background. In order to give the most of the *processor* time to the program you run in the foreground. This can be done using the **nice** command:

Syntax:

nice [-increment] *utility*

where the increment is a number in the range [0,20].

The increment tells what *priority* must be assigned to the utility run with **nice,** 20 being the lowest priority. The lower priority is, the less *processor* time is spent for the *job*. As you remember, Unix is a multitasking OS, but usually the computers running Unix have a single *CPU*. Thus the computer must *share* the time of its *processor* between the running *tasks*. However, the switching between different *tasks* happens so fast that a human eye can not see it. It looks to us, as if the programs were run in parallel.

Try to **nice** the find utililty:

% **nice –20 find** / *dir.txt >& found.txt &*

5. Displaying system processes (ps)

The *jobs* command will list all jobs run from the terminal. But there might be other terminals, the graphical shell *X Windows*, the *taskbar,* the web browser. Apparently they too consume system resources while running on your machine. Only the *jobs* command does not show them! The **job** command does not show the *resources* consumed by the *jobs* either.

There is another command used to display the status of the active processes.

Syntax

ps [options]

This command is very *system-specific*. Please, open the *man page* for **ps**:

% **man** *ps*

Most common options are (when two possibilities are given, the system dependance is meant):

- **-a** or **–e** - all processes, all users
- **-e** - environmrnt/everything
- **-u** - show user details of the process (who owns, when started etc.)
- **-x** or **–e** - show even those processes that are not run from a terminal.
- **-l** - long output format, more information than usually is shown

Examples of the command:

% **ps**

% **ps –a**

% **ps –al**

% **ps –u**

% **ps –Afl**

% **ps –u**

6. Killing your jobs (kill)

We already used **kill** command to terminate the *jobs* run from the terminal. Similarly, this command can be used to terminate *any process you own*, even if it is not listed by the **jobs** command.

Syntax:

kill [-signal] *–process-id*

The *process-id* is given by the **ps** command. The **signals** list can be obtained by

% **kill –l**

HUP INT QUIT ILL TRAP ABRT EMT FPE KILL BUS SEGV SYS PIPE ALRM TERM URG STOP TSTP CONT CHLD TTIN TTOU IO XCPU XFSZ VTALRM PROF WINCH INFO USR1 USR2

Some of the most commonly used sigmals are:

signal	description	reaction

KILL	Kill the proceess	Always terminate
QUIT	Quit the process	Quits if possible
STOP	Similar to **Ctrl+Z**, stop the process	Stops the process
CONT	Continue the stopped process	Resumes the process
TERM	Program termination	Terminates a process "gracefully"

Run a **vi** editor in one window and try several *system sygnals* in the other one:

% **vi** sometext

In the other terminal window:

% **ps –u**
% **kill –**TERM the_number_of_the_vi_process

7. Conclusions

We learnt how to tackle the *multitasking* features of Unix. We saw how to run *jobs* in the background. We learnt how to bring them back to the foreground. We changed *priorities* of the jobs. We also learnt how to *terminate* our *jobs*.

Finally we generalize our skills to all *processes*, including those that were not run from the terminal, and thus not shown by the **jobs** command.

End of the class.