School of Computer Science

COMP30640

Lab 4
Programs and Processes - Multiprogramming

Teaching Assistant:	Thomas Laurent
Coordinator:	Anthony Ventresque
Date:	Friday 5th October, 2018
Total Number of Pages:	4

1 First steps with processes. *I <3 Linux*

1. use the **ps** command to find the PID of process **dhclient**. What does **ps** alone give you? What does **ps aux** give you? How can you filter the output of **ps aux** to get the info about dhclient only (think **grep**).

   Solution

   - ps alone gives us information on processes running in this session.
   - ps aux gives us all of the process.

   ```
   $> ps aux | grep dhclient
   ```

2. What is dhclient's PID?
3. Using the **ps** command, find the PID of dhclient's parent process.
4. Using the **pstree** command, find the PID and name of dhclient's parent process.
   Solution

   - 2nd column of:

   ```
   $> ps aux | grep dhclient
   ```

   - 4th column of:

   ```
   $> ps axl | grep dhclient
   ```

   - The name should be systemd and the PID 1.

   ```
   pstree -s -p \$dhclientPID
   ```

5. What does the **yes** command do? (use the man page)
6. run the following command:

   ```
   $> yes "I <3 Linux"
   ```

   While this is running (don't kill it yet unless your system requires it) open another terminal (i.e., another bash window) and run **top** to check which processes are using resources. Now stop the process associated with the **yes** command. Use the combination of keys **control** + **c** in the terminal where yes is running to kill the process.

2 Play with the state of a process. *To kill a mockingbird*

1. Run a process in the foreground

   (a) Run the following command:

   ```
   $> yes > /dev/null
   ```

The process **yes** runs in the foreground (it has the focus and you cannot access the prompt). Try for instance to run **ls** in the terminal and you'll see that the command line interface is busy running your process.

(b) Now, in the terminal, use the combination **control + z** to stop the process (it is not killed, it is just not in the foreground anymore).

(c) Try to run the **ls** command in the terminal. Everything should be fine now.

(d) run the **ps -l** command to list all the processes that have been executed from this terminal. Check the man page for **ps** – in particular the section about the state of the processes. What state is the process runnning yes in?

(e) Now bring back the process in the foreground with the command **fg**.

2. Run a processus in the background

(a) use again **control + z** to stop the process (check that the process is really in the background using **ps -l**).

(b) Now use the **bg** command to run the process in the background. Can you use the program? Can you use the prompt? What is the difference between **fg** and **bg**?

(c) run **ps -l** again to check the state of the process.

(d) stop the process (kill command). Run it again but this time, add **&** at the end of the command, such as:

```
$> yes > /dev/null &
```

Check the state of the process now.

(e) You've probably noticed a number on the terminal after starting the process – what does it refer to?

3. Kill a process

(a) start two (2)

```
$> yes > /dev/null
```

processes in the background.

(b) find the PID of the first one and use the **kill** command to stop it (kill it).

(c) Check that the process is really gone.

(d) start another **yes** process in the background and check that 2 processes are running now.

(e) Use the **killall** command to end all the running **yes** commands.

3 Multiprogramming. *Hello and Good Luck!*

In this exercise, you will start creating multiple processes for "1" program. The idea is quite simple: you have two scripts: **hello.sh** (one of the scripts you created last week) and **good_luck.sh**. Each of these scripts is a program in its own right and you can obviously use them as any normal command you've been using in your scripts (e.g., **echo**).

1. make sure you have one of the scripts from last week working – e.g., get the first, simpler, one
2. write **another** script **good_luck.sh** that does a very similar thing (but says "good luck" instead of "hello")
3. make sure they both work
4. now write the following script, **main.sh**:

   ```
   #!/bin/bash
   ./hello.sh "$@"
   ./good_luck.sh "$@"
   ```

   What does this script do? How does it work?
   Solution

   - This program passes the arguments passed to the main.sh scripts to the scripts called within it.
   - Remember, "$@" is a list of arguments, all we are doing here is running the scripts with these arguments.

4 Multiprogramming. *Goodies and Baddies*

Create 2 files **goodies** and **baddies** – and add a few names in these files (e.g., Han Solo, Voldemort, etc.). We want to create 4 scripts here:

- 1 script that says "Welcome" to the goodies
- 1 script that says "We're full" to the baddies
- 1 script that says "I don't know you" to those who are neither goodies or baddies
- 1 main script that decides which of the 3 scripts above to call depending on the parameters (see the pseudo code below)

```
#! /bin/bash
#
# if the script has no argument then
#   print "no argument given"
# else
#   for each argument from the list of arguments given as input of the script
#     if the current argument is a goodie
#        print Welcome followed by the current argument followed by !
#     else if the current argument is a baddie
#        print We're full followed by the current argument followed by !
#     else
#        print I don't know you followed by the current argument followed by !
#
```

How to test whether a name is in one of the files (goodies and baddies)? Think **grep**, and use it as is in the condition. You can use -q to silence the output of grep to the terminal. **Hint:** Be careful with quotation marks!

Solution

```
    #! /bin/bash
if [ $# -eq 0 ]; then
    echo "No arguments given"
else
    for i in "$@"; do
        # -q silences output so it won't be printed to the console.
        if grep -q "$i" goodies.txt ; then
            ./welcome.sh "$i"
        elif grep -q "$i" baddies.txt ; then
            ./wereFull.sh "$i"
        else
            ./iDontKnowYou.sh "$i"
        fi
    done
fi
```

Just as an aside, this will work, but it will also technically match any letter in a name in the file. We can use regular expressions (pattern matching) to match the names perfectly if that is our goal.

```
#! /bin/bash
if [ $# -eq 0 ]; then
    echo "No arguments given"
        else
    for i in "$@"; do
        if grep -q ^\""$i"\"$ goodies.txt ; then
            ./welcome.sh "$i"
        elif grep -q ^\""$i"\"$ baddies.txt ; then
            ./wereFull.sh "$i"
        else
            ./iDontKnowYou.sh "$i"
        fi
    done
fi
```

As an aside to the aside, we can also use grep's -w option instead of a regex.