Practical Sheet 7

Operating Systems

This sheet uses Linux commands to explore Operating Systems. It is intended for use with the Cloud9 (c9.io) Virtual Machine (VM) environment, but similar capabilities can be achieved on other Linux systems, including other VMs and the Raspberry Pi.

You login to the Cloud9 using a web browser – Chrome recommended. Note that Cloud9 is an software development environment, so it is quite complex.

To use cloud9, complete the following steps:

1. Go to c9.io and create a free account (you may need to respond to an email)
2. Create a new workspace
    a. Use a blank template
    b. Any name is ok

1 Explore the Command Line

The command line is a more basic interface to the operating system that the familiar graphical user interface (GUI). Although GUI are easier to use, the primitive interface

- Gives a better understanding of what is happening underneath the graphical interface
- Allows multiple commands to be combined into 'scripts' for complex operations.

The Virtual Machine has a command line interface, even though it is accessed using a web browser. However, the overall interface is quite complex with the command window only one among many.

1.1 File management from the command line.

Exercise 1.1: Create a Text File

Since the system is empty there is nothing to look at. Try creating a file as follows:

```
$ ls                              Nothing to see
$ cat - > temp.txt                Type into a file …
The quick brown fox jumps             … this line
over the lazy dog                     … and this line
ctrl-d                            Keys 'ctrl' and 'd' together
temp $ ls                         Now a file exists
temp.txt
```

Exercise 1.2 Try the following commands:

Comm-and	Description	Example	
pwd	Print the current working directory	`pwd`	
ls	List the files in a directory	`ls`	List all the files in the directory (likely to be empty)

Comm-and	Description	Example	
		`ls -l`	List with more information
		`ls *.py`	List files matching *.py i.e. Python files
cd	Change directory	`cd Files`	Change to a sub-directory of the current directory
		`cd /`	Change to the root directory
		`cd /usr/bin`	Change to a directory under the root
		`cd`	Go back to your home directory
mkdir	Create a new directory in the current directory	`mkdir sheet7`	Create a directory sheet7 in the current directory

Exercise 1.3 Use commands to answer.

- What is the path to my home directory?
- Are there other home directories?
- What is in the root directory '/'
- Where are the programs stored?

1.2 Running a Python Program

You can also crate a file using the IDE. A Python program can be run from the command line:

```
$ python3 test1.py
Hello
```

Exercise 1.4 Enter and run a simple python program.

1.3 Machine and Operating System Information

The 'uname' gives information about the OS:

Exercise 1.5 Try each of the following uname options:

1. The machine make: `uname -m`
2. The Operating System name: `uname -o`
3. The Operating System release: `uname -r`
4. The node name: `uname -n`

1.4 Installed Software Information

The following command gives information about the installed packages. Installing packages is the way to provide more programs. **Warning**: other options may break things!

```
dpkg --list
```

Exercise 1.5 Try this and look for the python3.X package. Now use: 'dpkg –L *<package name>*' to list the file provided by this package. Note down the path and use 'ls' to check that it is there.

1.5 Windows Command Line

Try accessing the Window command line on the PC you are using. The command name is usually 'cmd'. The Window commands are not necessarily the same as Linux, but some are.

Exercise 1.6 Use 'mkdir' to create a directory and 'dir' to list the file is a directory. Compare what you see on the command line and in the GUI interface.

2 Compiling a C Program

A **simple C program is provided**. Download it from TeachingLondonComputing and then upload it to the virtual machine. Unzip it using the command 'unzip sheet7_files'. It contains the three files:

1. main.c: this inputs three numbers (lengths) and calls 'isTriangle'.
2. isTriangle.c: this tests whether the lengths could make a triangle.
3. isTriangle.h: is a header file declaring the isTriangle function.

Read this code. The program tests whether three length make a triangle.

2.1 Step 1: Compile C Program and Look at Executable File

Compile The Program All At Once

To compile the whole program use the following command:

```
gcc isTriangle.c main.c –o ttest
```

This command compiles the two C source files and generates an executable called 'ttest'. You can run the program as follows:

```
./ttest 3 4 5
```

Exercise 2.1 Compile and run the program.

Examine the Shared Libraries

Use the following command to see what shared libraries are used by the program:

```
ldd ttest
```

The following table explains what the libraries are for:

Shared Library	Description
linux-vdso.so.1	Used for making system calls
libc.so.6	The standard C library
/lib64/ld-linux-x86-64.so.2	The loader. This is called first to read the executable file and link to the other shared libraries.

Symbols in the Executable File

The executable file does not just contain binary data. It also contains symbols. There are two reasons for this:

1. It needs to know which functions are needed in the shared libraries (see 2.2 above).
2. It is useful (e.g.) for debugging to retain symbols from the original program[1].

The following command list global and local symbols respectively.

[1] The source code can be included too, so it can be used in debugging. Compile with gcc –g.

```
objdump -T -w ttest
objdump -t -w ttest
```

The global symbols are essential as they refer to entries in shared libraries. Check that you can associate (most) of them with the C source code. You may wish to use google to find out more about the others.

The local symbols are not essential. A program 'strip' can be used to remove them, to make the executable file smaller. You need to copy the file first:

```
cp ttest ttest1
strip -s ttest1
```

Repeat the objdump command to look at symbols in the stripped executable file. Also compare the size of the file before and after being stripped.

Exercise 2.2 Explain the origin of the symbols in the executable file.

Look at the Assembly Code
You can examine the assembly code that in the program using:

```
objdump -d -w ttest
```

It is best to save this in a file as it is quite long. You will see there are 8 sections, only 2 of which (the longest ones) directly correspond to the C source code. The code is shown twice – as hexadecimal bytes and as assembler code; you will see that the number of bytes needed for each assembly code instruction (one on each line) varies.

- What are the start and end addresses of the <main> section in the program's address space?
- Match some of the assembly code statements in this section to C code statements.
- Look for a reference to a location in the '.rodata' section of the executable (see 2.3 above).
- Calculate the number of bytes used by both <main> and <isTiangle>.

The amount of code can be reduced if you ask the compiler to optimise it, as follows:

```
gcc -O2 isTriangle.c main.c -o ttest2
```

Look again at the assembly code and see how much it has changed.

Exercise 2.3 Examine the assembly code and match one part against the C source code.

2.2 Step 2: Compile Separate Object Files

Instead of compiling and linking the two files in the program separately, it is possible to compile each file separate. Rather than generating an executable file, this generates an object file for each source file.

```
gcc -c main.c
gcc -c isTriangle.c
```

You can then link these object files to generate the (same) executable as before.

```
gcc main.o isTriangle.o -o ttest3
```

It is the same these steps – compiling and then linking – are exactly what the compiler did in 2.1 above.

You can use the objdump to examine the object files, as above. For example:

`objdump -t -w main.o`	Looks at symbols
`objdump -d -w main.o`	Looks at assembly code

As an example of the differences, the following line of assembly code is taken from the ttest executable. It is a call to the isTriangle function which is at address 400778

```
400767:   e8 0c 00 00 00          callq  400778 <isTriangle>
```

Find the equivalent line in main.o and explain the difference.

Exercise 2.4 Explain what happens when files are compiled separately.

3 Users, Processes and Memory

All the operating systems we run allow multiple users to login into the same computer. It is easiest to show this with a Raspberry Pi. On almost all Windows installations this is disabled. It is also not directly possible on the web-based VM.

3.1 The Operating System Runs Processes

Exercise 3.1 List the processes on the system using the command: ps –efH

The output includes:

- UID: which user owns the process? Many are owned by 'root', which is the system.
- PID: a unique process identifier.
- PPID: the id of the process which created this process. The processes forms a parent child hierarchy. A few processes were created by 'process 0', but you will see there is no process 0. Process 1 and process 2 created lots of processes.
- STIME: the time when the process started
- TTY: whether the process is connected to a terminal (compare with result of 'who').
- TIME: Total CPU time used by the process.
- CMD: The command that is being executed. For many processes this is shown in square brackets: [nnnnnn]. These are kernel processes, meaning that they make up part of the Operating System itself.

Two other commands are 'pstree' and 'top'; 'top' shows a dynamic display of similar information. Try it. (Press 'q' to stop it). Note that nothing much will be happening. If you want to change this get another person to i) login to your machine ii) type in a program and iii) run it. Can you think of a way to make a program run for a log time?

Exercise 3.2 Find the PID of the process running the ps (or top) command.

3.2 Memory Usage

There are various ways to look at memory usage.

1. free command

Enter:

```
free –m
```

this shows the total memory, the memory used and he free memory. There are some complications:

- Some memory can be shared by several processes. This is shown as shared. It may affect the total figure.
- Extra memory is used to cache data or to buffer data going to/from the disk. The amount of cache and the size of the buffers can be reduced, freeing more memory if necessary. This is shown on the second line.

The final line shows the swap memory: i.e. pages that have been removed from memory to make space for other pages.

2. meminfo

Enter:

```
cat /proc/meminfo
```

This shows similar information about the memory usage. Note that .proc/meminfo appears like a file but is actually generate 'on the fly'.

3. vmstat

Enter:

```
vmstat -s
```

This shows various counters for events in Linux and other information. As well as data about memory pages, you may see a count of the number of interrupts and the number of context switches. A context switch is the number of time the running process has been changed.

4. top

Enter:

```
top
```

This shows data for each process, including memory usage. The display is updated.

You can use google to get more details of the information; in practice there are difference between different versions.

Exercise 3.3 Investigate the use of the above command for memory usage.

4 Summary

- "Let's get under the hood". A Virtual Machine (or a Raspberry Pi) is good for exploring Operating Systems.
- An operating system is just a program. Everything we do on the computer involves running programs.
- The most important ideas in Operating Systems are:
    - Processes: the OS runs these. They are what's happening at any moment. Lots of processes run at once.
    - User: many users can use the computer at once.
    - Files: there are many files – holding installed programs and configuration data, as well as user data.
    - Devices: the OS manages devices, making it easy to store files on a disk.