ECS518U Operating Systems

Lab 5: A Simple Shell and extension with capability to run external Linux shell commands or programs

This exercise is assessed (Must be assessed the latest by your Week 7 lab but hopefully you will get it done before ☺).

Aim

This lab has two parts:

1. The aim of the first part of the lab exercise is to write a simple shell as a Python script, using Python functions (**NOT** existing Linux shell commands).
2. The aim of the second part is to extend the shell so that it can run normal Linux commands or external programs (in addition to your own shell commands).

In addition, you are to find out what happens in terms of processes when Linux shells execute commands/programs, to relate this with the part of the lectures on process control (e.g. process creation) and to answer the relevant questions on the answer sheet.

You can assess this lab **only one time**, either during your Week 5, Week 6 or Week 7 lab slot. See QMplus for details on assessment rules and marking criteria.

You should be able to:

1. Read the official Python documentation: https://docs.python.org/3.10/
2. Look at the functions available at the **`os` module** – we will be using them a lot:
3. Also look at the functions in **os.path** – we will be using them today and in the weeks to come: https://docs.python.org/3.10/library/os.html
4. Run Python scripts, as practiced in labs 3 and 4.

Sample code

Sample code is available for you to use to get you started (get it from the scripting folder on QMPlus). Download the script **`my_shell_outline.py`**. You are recommended to:

- Follow the pattern of the sample, with a function to implement each command of the shell. The command "files" is already implemented for you in the code given.
- Implement and test each command in turn.

Part A: Requirements for your shell

Your simple shell should provide the following functions (these must be implemented using **only Python functions** and NOT by running Linux shell commands from within your script):

Cmd Name	Arguments	Behaviour	Error Conditions to Check
info	1: file or directory	List information about the file or directory (see below for details)	File or directory must exist
files	(none)	List the files and directories by name, showing which are directories	(None)
delete	1: file	Delete the file	File must exist

copy	1: the 'from' file 2: the 'to' file	Copy the 'from' file to the 'to' file	The 'from' file must exist and 'to' file must not exist
where	(none)	Show the current directory	(None)
down	1: directory name	Change to the specified directory, inside the current directory	The directory must exist
up	(none)	Change to the parent of the current directory	If you are at the top of the directory tree, you should not be able to go further up – display a message
exit	(none)	Exits the shell	(none)

The 'error conditions' should be checked by your code.

For example, if the user asks for 'info' on a file that does not exist, the shell should display a helpful error message before continuing to accept the next command.

The following general error conditions must also be checked:

1. The correct number of arguments is given.
2. Operations on the file system succeed. For example, it may not be possible to delete a file (e.g. because of permissions).

   Note: it is sufficient to do this by dealing with the return values of functions or exceptions thrown if the operations fail. Look at the specification of the functions at the Python documentation. We do not require you to do specific investigation on access permissions. For those interested in this, check the information at: https://docs.python.org/3.10/library/os.html at section 16.1.5 on Files & Directories.

The following information should be listed by the info command (most of this has been done in Lab 4):

- Whether it is a directory or ordinary file
- The owner of the file or directory
- The date the file or directory was last changed (`mtime`)
- For files that are not directories
  - The size of the file in bytes
  - Whether the file is executable (Hint: check `os.access` at: https://docs.python.org/3.10/library/os.html at section 16.1.5 on Files & Directories – one way is to use os.access(os.path.abspath(filename), os.X_OK)

It is ok to include additional information.

As a sample output for the info command:

PShell>info partA.py
File Name: partA.py
Directory/File: file
Owner: tassos
Last Edited: Tue Feb 15 13:53:50 2022
Size (bytes): 5789
Executable?: True

Apart from the requirements described above, you are free to design the interface as you like. **Where you think that the requirements are ambiguous, decide how your shell will**

behave and tell us in your Answer Sheet (there is space to note this there) and when we assess your shell during your lab.

Python functions

The exercise is not challenging in terms of programming difficulty – the skeleton code gives you a very good starting point. You should just spend some time researching which Python functions to use in order to implement each command.

Some functions that may be used to implement some of the commands of your shell: `os.unlink() or os.remove(), shutil.copyfile() or shutil.copy2() or shutil.copy(), os.getcwd(), os.chdir()`, etc.

Part B: Running Linux shell commands and external programs from your shell

Your shell as implemented in Part A is not capable of running any other commands apart from the 8 commands specified. Your aim in this part is to extend it such that it can also run Linux shell commands and external programs. Paste the code for the functions from your script in Part A in the appropriate section of the provided script `my_run_shell_0.py`

You are expected to work through the steps below to enhance `my_run_shell_0.py` to run external programs and Linux commands.

- **Any command name** that is not the name of one of the 8 commands in the shell script from Part A is assumed to be that of a Linux command / external program (e.g. `ls -l`, `ps`, `python3 fork.py`, etc.).
- The shell script should **check that the command/program name is an executable file** on the path.
- The command/program must be executed using the Python function **`os.execv()`** and the results displayed. When the command/program has completed, the shell can accept further commands and external programs.
- **The shell script should show whether the external command completed normally, giving its return code, or was interrupted and exited "abnormally"**. There are Python functions that can check whether a process exited normally.

Step 1: Investigating the Outline Solution (`my_run_shell_0.py`)

An outline solution is provided. It has the following functions:

- It recognises that a program/command is not an internal command in the shell (i.e. not one of the 8 shell commands) and checks that an executable file exists somewhere on the path.
- It runs the program/command using `os.execv()`.

However, it does not meet all the requirements for Part B. **You should investigate how it behaves and explain its behaviour clearly.** To do this, you should observe the processes that exist when a command is run from within your shell. To do that you can use the `ps` command with appropriate arguments from another terminal window (look at material from the Week 2 lab for process monitoring).

☛ **To Do: Answer the relevant questions** on the Answer Sheet.

Step 2: Running in a Child Process

To resolve the non-meeting of requirements that you found in Step 1 you need to create a child process, which runs the external program using `os.fork()`: first use `os.fork()` to fork a child process and then run the external program in the child process using `os.execv()`. The parent process must **wait** for the child to complete before continuing.

☛ **To Do:**

1. **Implement this change.** Sample code using `os.fork()` is available in the scripting folder for Lab 3 and was discussed in the lectures of Week 3.
2. **Answer the relevant questions on the Answer Sheet.**

Note: it is a requirement of this lab to use `os.execv()` to run the external command/program as this corresponds to the underlying system call in Unix. **You will not receive any marks if you use another Python function to run external programs.**

Testing, Demonstration and Assessment

You are expected to:

- Answer the questions in the Answer sheet and be ready to explain your answers.
- Be ready to answer questions about the sample programs we have made available.
- Be ready to demonstrate your own shell working.
- Be ready to answer questions about your code. **The code is expected to be clearly commented and properly formatted.**

Answer Sheet

Testing, Demonstration and Assessment

You are expected to:

- Answer the questions in the Answer sheet and be ready to explain your answers.
- Be ready to answer questions about the sample programs we have made available.
- Be ready to demonstrate your own shell working.
- Be ready to answer questions about your code. **The code is expected to be clearly commented and properly formatted.**

PART A

Your Shell: Features Completed

Complete a table similar to the following showing the status of each of the required commands. You should describe how you resolved any ambiguities you found in the implementation of each command. This is mostly for your own record of what was done in the implementation of the Shell.

(example entries) :

Command	**Status (Complete / Partial / Not Complete)**	**Notes** (If you found any ambiguity in the specification for a command, say how you dealt with it. If something is left not implemented, make a note. Else say None)
files	Complete	none
copy	Complete	Currently not checking whether the “to” file does not exist

Command	**Status (Complete / Partial / Not Complete)**	**Notes** (If you found any ambiguity in the specification for a command, say how you dealt with it. If something is left not implemented, make a note. Else say None)
files		
info		
copy		
delete		
where		
up		
down		
exit		

Question: Linux Shells: How do they execute commands/programs?

Describe what happens in terms of processes when you run a command/program in a Linux shell (the answer can easily fit into 2-3 sentences). Copy and paste output of appropriate `ps` and/or `pstree` commands that explain what happens in terms of processes.

You must relate this to the lecture material on process control that was covered during the Week 3 lecture, so we obviously expect something more than, for example, ("the program that corresponds to the command is executed").

Ideas: Open one or more terminals, execute some command or program and try to find a way to examine what happens in terms of processes. You can use commands such as `ps` (look at what command line options give you information that helps you, e.g. `ps -ef`, `ps -Oppid`), `pstree`, etc. to see what processes are created. Try to figure out how the command or program you ran relates to the shell program. You can also research online but you must relate all this to lecture material on process control (e.g. fork) that we covered in Week 3.

Answer and pasted output:

PART B

(These questions are in addition to code that meets the requirements for Part B – answering these questions with no implementation will not give you full credit for this Part of the lab)

Step 1: Behaviour of 'my_run_shell_0.py' as supplied (i.e. without any changes from you)

Answer the following questions about the behaviour of `my_run_shell_0.py`

Question	Answer
What happens when the 'my_run_shell_0.py' script is used to run an external program (e.g. a Linux shell command)?	
What requirement does this behaviour not implement?	
Why does this behaviour occur? (Refer to the documentation of `os.execv()`)	

Step 2: Use of `os.fork()`

Answer the following questions about the use of `os.fork()`:

Question	Answer
What happens when the `os.fork()` function is executed?	
What Python function is used to make the parent process wait for the child process to complete? Why is this good practice?	