**CS2106 Introduction to Operating Systems**

**Lab 2 - Shell Scripting and Process Programming**

**Answer Book**

Please read the instructions in the main lab sheet before completing this document. Submission deadline is **Sunday 25 February 2024, 1 pm**. The folder will stay open slightly after this, but once the folder closes, **absolutely no submissions will be allowed.**

**Submission checklist:** A ZIP file called AxxxxxxY.zip, where AxxxxxxY is the student ID of the student submitting. The ZIP file should contain:

* This file, appropriately renamed to the submitter’s student ID.
* grade.sh
* lab2p2f.c

|  |  |
| --- | --- |
| **Student 1** | |
| Name: | Kenneth Seet |
| Student ID: | A0258173Y |
| Group: | B18 |
| **Student 2** | |
| Name: | Daniel Wang |
| Student ID: | A0255689H |
| Group: | B11 |

**Part 1 – Bash Scripting**

**Question 1.1 (1 mark)**

The first line of a Bash script, starting with #!/bin/bash, is called a shebang or hashbang. It is not a comment, and its purpose is to specify the path to the interpreter that should be used to execute the script. In this case, it indicates that the script should be interpreted and executed using the Bash shell.

The #!/bin/bash line tells the system where to find the Bash interpreter, and when you run the script, the system will use Bash to interpret and execute the commands in the script.

Without this shebang line, the system may use a default shell to execute the script, which might lead to unexpected behavior if the script relies on Bash-specific features. Including the shebang ensures that the script is executed with the intended interpreter.

**Question 1.2 (1 mark)**

We fixed the script by using double parentheses for arithmetic operation: z=$((x - y)) instead of z=$x-$y.

**Question 1.3 (1 mark)**

#!/bin/bash

echo "Hello $(whoami), today is $(date +%A), $(date +%d) $(date +%B) $(date +%Y), and the time is $(date +%T)."

**Question 1.4 (1 mark)**

The following are special variables in Bash. What do they hold? $#, $1, $2, $@, $?

$#: Holds the number of arguments passed to a script or function.

$1: Holds the first argument passed to a script or function.

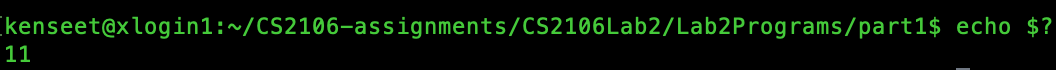
$2: Holds the second argument passed to a script or function.

$@: Stores the list of arguments passed into the function as an array.

$?: Holds the exit status of the last executed command. A value of 0 typically indicates success, while non-zero values indicate errors or other exit statuses.

**Question 1.5 (1 mark)**

I see the value 11 printed in the shell.



The final value of i in the code was 11, and when exit(i) was called, the program exited with the exit status of the value of i, which was 11.

Since $? stores the value of the exit status of the last executed command, it causes 11, which was the exit status, to be displayed on the shell.

In short, echo $? prints out the value of i from the last time exit(i) was called, which in this case, was 11.

**Question 1.6 (1 mark)**

When I ran “./slow 5 ; ./slow 10”, the output of ./slow 5 is displayed first, followed by the output of ./slow 10.

A computer screen with green text

Description automatically generated

When I ran “./slow 5 & ./slow 10”, the output of ./slow 5 and ./slow 10 both show up together, and there is no fixed order to which output appears first.

A computer screen with green text

Description automatically generated

When you run ./slow 5 ; ./slow 10, the semicolon ; is used as a command separator. This means that the second command (./slow 10) will only start executing after the first command (./slow 5) has completed.

On the other hand, when you run ./slow 5 & ./slow 10, the ampersand & is used for running the first command (./slow 5) in the background. This allows the second command (./slow 10) to start immediately without waiting for the completion of the first one.

In summary:

;: Executes commands sequentially; the second command starts only after the first one completes.

&: Executes the first command in the background, allowing the second command to start immediately without waiting for the completion of the first one.

So, the difference between the two commands lies in how they handle the execution of the commands in relation to each other (sequentially with ; or in parallel with &).

(For grader only) Part 1 total: \_\_\_\_\_\_\_\_\_\_\_ / 6

**Part 2 – Playing with POSIX Calls**

**Question 2.1 (1 mark)**

**Question 2.2 (1 mark)**

**Question 2.3 (1 mark)**

**Question 2.4 (1 mark)**

Cut and paste new code here and explain

**Question 2.5 (1 mark)**

**Question 2.6 (1 mark)**

**Question 2.7 (1 mark)**

(For grader only)

Part 2 total: \_\_\_\_\_\_\_\_\_\_\_ / 7

**REPORT TOTAL: \_\_\_\_\_\_\_\_\_\_\_\_ / 13**

**Demo: \_\_\_\_\_\_\_\_\_\_\_\_\_ /4**

**Total: \_\_\_\_\_\_\_\_\_\_\_\_\_/17**