Curtin University — Discipline of Computing

Unix Systems Programming (COMP 2002)

2021, Semester 1

Final Assessment

Weight: 50% of the unit mark.

Released: 1430 (GMT+8) June 17th

Due Date: 4 hours after downloaded from Blackboard, within the 24hr period

Note: The reason for the 24 hour window is to accommodate for uploading issues and any timetabling clashes. It is recommended that you download and work on the test as soon as it is made available. Your time (**4 Hours** *(240 minutes)*) starts as soon as you have downloaded the test from Blackboard.

Please read this front cover carefully. If any one of these are done incorrectly, it can result in heavy penalties.

This is a **OPEN BOOK test**. There are **FOUR (4) questions**. Answer all of them. You have **4 Hours** *(240 minutes)* from the time the test is downloaded from Blackboard to complete the test. There are **100 marks** available.

Each Question is to be stored in its own file named: **Question*** where the ***** is replaced with the word version of the number (e.g., **One**, **Two**). Programs must have the appropriate extensions for that program.

All your code must conform to practices emphasised in the lectures and practicals. All code must conform to Computing's coding standard.

You must work alone on this assessment. You must not communicate with anyone other than your lecturer or the Unit Coordinator regarding any aspect of this Assessment.

All submissions will be subjected to rigorous testing for plagiarism, collusion and any other forms of cheating. You must cite any and all code from any source, including your own work submitted for a different assessment.

Upon Completion, you must **tar** and **gzip** your Declaration of Originality, **BashHistory.txt** and Question files, and submit to Blackboard.

[user@pc]$ tar -cvzf <StudentID>_TestAnswers.tar.gz TestAnswersDirectory

Question 1: Unix/Linux Knowledge(25 marks)

Answer these short answer questions in your own words.

(a) Describe the general steps required to create and use a loadable kernel module.

[4 marks]

(b) Describe what a runlevel is in Linux. Additionally, describe how they are used (with specific examples) in Linux. **[5 marks]**

(c) Create a sed command for finding occurrences of four consecutive and identical word characters (like aaaa or bbbb)

[6 marks]

(d) Specifically describe the use of the following bash script:

```
sed "s/[[:punct:]]//g;
     s/\b$1\b/@/g;
     s/[[:alnum:]]*//g;
     s/ //g;
     /^$/d;
     " $2 |
sed ':a;N;$!ba;s/\n//g;' |
sed 's/@$//g;
     s/@/\n/g' > temp;
sed = temp | sed '/^$/d;:a;N;$!ba;s/\n/ /g;
s/\(.\)* \(\b[0-9]*\) $/\2/g;'
```

[10 marks]

QUESTION 2 APPEARS ON NEXT PAGE

Question 2: C Programming (35 marks)

(a) Examine the program below and answer the following questions about it:

```c
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>

int main(void)
{
    FILE* fp;
    char* str;
    str = (char*)malloc(sizeof(char)*20);

    // Fork 1
    // fork();

    fp = fopen("test.txt", "r+");

    // Fork 2
    // fork();

    fgets(str, 20, fp);
    printf("%s", str);
}
```

i) Describe a simple test file that you might use for this program. In the context of the test file, what is the output if you uncomment Fork 1? What is the output if you uncomment Fork 2? **[2 marks]**

ii) Assuming that you only uncomment one statement, does it matter which one you uncomment? Explain in detail why or why not. If appropriate, discuss underlying data structures. **[8 marks]**

QUESTION 2 CONTINUES ON NEXT PAGE

(b) Examine the following function. This function is intended to return TRUE if the 'needle' is able to be found within 'haystack'. However, testing reveals that it is not functioning as intended. Explain why and provide a solution.

[5 marks]

```
int finder(const char *haystack, char *needle)
{
    int match = FALSE;
    regex_t re;
    if (regcomp(&re, needle, REG_EXTENDED|REG_NOSUB) == 0)
    {
       match = TRUE;
    }
    return match;
}
```

(c) Examine the program below. Describe how this program works. Additionally, explain what issues might arise when using this program.

[8 marks]

```
int i = 0;

int my_handler(void)
{
    do_something_else();
    i = i + 1;
}

int main(void)
{
    signal(SIGINT, my_handler);
    for( ; ; )
        if ( i > 0 )
        {
            do_something();
            i = i - 1;
        }
}
```

QUESTION 2 CONTINUES ON NEXT PAGE

(d) Write a simple C program to the following specification:

- Ask the user for their username and personal identification number (PIN)
- The PIN must be a 5-character sequence of characters.
- Ensure that the PIN is hidden from view as the user types.
- Do not wait for the user to press 'Enter' to accept the pin; accept the PIN the moment the user types in the 5th character.
- You do not need to actually validate or verify the information the user has typed in.
- Add a handler to the program such that whenever the user types CTRL-C into their terminal, the terminal is restored to its original setting.

[12 marks]

QUESTION 3 APPEARS ON NEXT PAGE

Question 3: Python Programming (15 marks)

For this question you may find the following documentation useful:

https://docs.python.org/3/library/os.html

Write a python script that has the following functionality:

- Accept two strings as command line arguments.
  - The first string is a **pattern**
  - The second string is a **file name**
- The script will output the names of all files that contain the **pattern** in the current directory and each of its subdirectories.
- The output of this script will be saved to the file specified by **file name**.

QUESTION 4 APPEARS ON NEXT PAGE

Question 4: System Calls (25 marks)

(a) Discuss what a system call is and how it differs from a function defined by the user.

[5 marks]

(b) Many system calls in C are provided by a glibc wrapper. If a wrapper does not exist for the system call you are interested in or you do not wish to use it, how do you use a system call?

[4 marks]

(c) Examine the program below. Convert this program into one that makes use of system calls. Raw system calls or glibc wrappers are fine. **Note:** *fork()* must be changed as well.

[16 marks]

```c
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>

int main(void)
{
    FILE* fp;
    char* str;
    str = (char*)malloc(sizeof(char)*20);

    fork();

    fp = fopen("test.txt", "r+");

    fgets(str, 20, fp);
    printf("%s", str);
    free(str);
}
```

End of Final Assessment