NWEN 241 Assignment 3

(Weeks 6–7 Topics)

Release Date: **18 May 2020**

Submission Deadline: **2 June 2020, 23:59**

This assignment is divided into 2 parts.

- In Part I (Tasks 1–3), you will be asked to answer questions about Weeks 6–7 topics, submitted in a plain text file named `part1.txt`. For the diagram in Task 3, to be submitted as PDF with name `part1t3.pdf`.
- In Part II (Tasks 4–8), you will be asked to implement a network server program, the specifications and guidelines for which are presented in each of the tasks. To be submitted in files named `server.c` and `server2.c`.

You must submit the required files to the Assessment System (`https://apps.ecs.vuw.ac.nz/submit/NWEN241/Assignment_3`) on or before the submission deadline. Late submissions (up to 48 hours from the submission deadline) will be accepted but will be penalized. No submissions will be accepted 48 hours after the submission deadline.

Full marks is 100. The following table shows the marks distribution:

Task Type	Part I	Part II	**Total**
Core	20	45	**65**
Completion	8	12	**20**
Challenge	5	10	**15**
Total	**33**	**67**	**100**

Part I: Concepts

This part will test your conceptual knowledge of Weeks 6–7 topics. Your answers should be submitted in a plain text file named `part1.txt`.

Task 1.

Core [20 Marks]

1. **(2 Marks)** Name the system call that is used by a parent process to obtain the exit status of a child process.
2. **(2 Marks)** Fill in the blanks: In Linux, the return value for the `fork` system call is ____ for the parent process, and ____ for the child process.
3. **(2 Marks)** Which of the following is NOT a valid process state?
   (a) `New`
   (b) `Running`
   (c) `Sleeping`
   (d) `Waiting`
4. **(2 Marks)** Briefly explain the difference between `synchronous` and `asynchronous` sockets.
5. **(4 Marks)** What is the command to `open` a socket when programming a client?
6. **(4 Marks)** Which of the following statement(s) is/are true about `fork`?
   (a) It creates a child process which shares the global variables of parent process.
   (b) It creates a child process which shares the address space of parent process.
   (c) It creates a child process which cannot be terminated until parent process is terminated.
   (d) Child process is automatically terminated if the parent process is terminated.
7. **(4 Mark)** Which of the following is the correct order in which a server process must invoke the function calls `listen`, `accept`, `bind`, and `recv`?
   (a) `accept`, `listen`, `bind`, `recv`
   (b) `bind`, `accept`, `listen`, `recv`
   (c) `bind`, `listen`, `accept`, `recv`
   (d) `listen`, `accept`, `bind`,`recv`

Task 2.

Completion [8 Marks]

1. **(3 Marks)** You are given the following C program:

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

int main(void)
{
    int var= 80;
    pid_t pidA=-1, pidB= -1;

    printf("before fork\n");
    if ((pidA = fork()) < 0) {
        printf("fork error\n");
    } else if (pidA == 0) { /* child */
        var++;
    } else { /* parent */
        pidB=wait(NULL);
    }

    printf("pidA = %ld, pidB = %ld, var = %d\n", (long)pidA,
       (long)pidB, var);
    exit(0);
}
```

   Assume that the parent process ID is 560 and the child process ID is 561. What is the output of the program?

2. **(3 Marks)** How many times will this program print "HELLO"? Briefly justify your answer.

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

int main()
{
    if( execl("/bin/ls","ls",NULL) == -1){
        printf("HELLO\n");
    }
    printf("HELLO\n");
```

```
    return 0;
}
```

3. **(2 Mark)** Which of the following system calls is NOT used for interprocess communication using TCP sockets?

   (a) `bind`

   (b) `connect`

   (c) `send`

   (d) `sendto`

Task 3.

Challenge [5 Marks]

1. Consider the following code snippet:

```
if(fork()) {
    fork();
}
if(fork()|| fork()) {
    fork();

}
printf("Nesting");
```

   (a) **(2.5 Mark)** How many times is `Nesting` printed ?

   (b) **(2.5 Mark)** Draw a tree diagram to explain your answer in (a). You can use Microsoft Word (or any other program capable of drawing) to draw the diagram, but export the document as PDF named `part1t3.pdf`.

Part II: Practical Programming

This part will test whether you can apply the conceptual knowledge you have learned in Weeks 6–7 to solve practical programming tasks. You may only use the Standard C Library to perform the tasks in this part.

In the programming tasks, you will implement a network server program, where some messages are exchanged between server program and client.

Sample skeleton codes are provided under the `files` directory in the archive that contains this file. Use the skeleton files to perform the tasks.

Commenting

You should provide appropriate comments to make your source code readable. If your code does not work and there are no comments, you may lose all the marks.

Coding Style

You should follow a consistent coding style when writing your source code. Coding style (aka coding standard) refers to the use of appropriate indentation, proper placement of braces, proper formatting of control constructs, and many others. Following a particular coding style consistently will make your source code more readable.

There are many coding standards available (search "C/C++ coding style"), but we suggest you consult the *lightweight* Linux kernel coding style (`https://www.kernel.org/doc/html/v4.10/process/coding-style.html`). The relevant sections are 1, 2, 3, 4, 6 and 8. Note that you do not have to follow every recommendation you can find in a coding style document. If you change, for instance the tab size from 8 to 4, that is fine. You just have to apply that style consistently.

Program Specifications - Server Program

1. Load the CSV database file `scifi.csv` which contains the 25 greatest science fiction films of all time. You can use any data structure for holding the records in memory. Note that `scifi.csv` is provided under the files directory. **Note that this functionality is already provided in the skeleton file** `server.c`. You do not need to implement this.

2. Initiate the required socket operations to create and bind a socket.

3. Listen for clients at TCP port `12345`. (During testing, you may need to use a different port number to avoid conflicts with other students running tests on the same server.)

4. When a client successfully establishes a connection with the server, send a `HELLO` message back to the client .

5. Wait for a message from the client.
6. Perform the following based on the message received from the client:
   (a) If the message has the contents `BYE` (case-insensitive), close the connection to the client and go back to step 3.
   (b) If the message has the contents `GET` (case-insensitive) followed by a number (example: `GET 12`), treat the number as the row number of the record to be fetched. That is, send a message back to the client containing the record at the specified row number. You may format the content in any way you want, but all the fields of the record must be printed. If the specified row number does not exist, send back a message containing `ERROR`. Go back to step 5.

Testing

As you are using the Linux `socket()` and `fork()` system calls, you will need to test your program in Linux. You do not need to write a client program to test your server program. You can use the Linux program `nc` to receive and send messages to your server program. To do this:

1. Open a terminal. Compile and run your program.
2. Open another terminal. Run `nc localhost 12345` to establish a connection with your server program. You may need to replace `12345` with the actual port number that you specified in your code. Once `nc` is connected to the server program: whatever you type in this terminal will be sent to the server program, and whatever is sent by the server program will shown in this terminal.

If you are performing the test remotely, you will need to open 2 ssh/putty connections to the same server machine. In one ssh/putty terminal, you compile and run your program. In the other ssh/putty terminal, you run `nc localhost 12345` to establish a connection with your server program. You may need to replace `12345` with the actual port number that you specified in your code.

A video will be posted in Blackboard (by Wednesday, 20 May) to demonstrate the testing process.

Task 4.

Core [45 Marks]

Implement the server program specifications 2 – 5 above inside the file `server.c`.

Task 5.

Completion [12 Marks]

Implement the server program specifications 6(a) and 6(b) above inside the file `server.c`.

Task 6.

Challenge [10 Marks]

One of the drawbacks of the server program specified in the **Program Specifications** above is that when a client is connected with the server, no other client can connect to the server.

Provide an enhancement of the server program by using the `fork()` system call as follows: At step 4 of the Program Specifications above, when a client successfully establishes connection, fork a child process. Upon successful fork, the parent process should go back to step 3, whereas, the child process should perform the following:

1. Send a message to the client with contents `HELLO`.
2. Wait for a message from the client.
3. Perform the following based on the received message from the client:
   (a) If the message has the contents `BYE` (case-insensitive), close the connection to the client and exit the process.
   (b) If the message has the contents `GET` (case-insensitive) followed by a number (example: `GET 12`), treat the number as the row number of the record to be fetched. That is, send a message back to the client containing the record at the specified row number. You may format the content in any way you want, but all the fields of the record must be printed. If the specified row number does not exist, send back a message containing `ERROR`. Go back to Step 2.

If the fork is not successful, the process should perform steps 4 and 5 (in the original Program Specifications).

Save the enhanced server program as `server2.c`.

Marking Criteria for Tasks 4–6:

Criteria	Weight	Expectations for Full Marks
Compilation	10%	Compiles without warnings
Commenting	10%	Sufficient and appropriate comments
Coding Style	10%	Consistent coding style
Correctness	70%	Implements all specifications and handles all possible cases correctly
Total	100%	