COMP 304- Operating Systems: Assignment 1

Due: Feb 27, 10.00 pm

Notes: This is an individual assignment. No forms of cheating will be harshly punished! You may discuss the problems with your peers but the submitted work must be your own work. No late assignment will be accepted. Submit your answers through blackboard. This assignment is worth 4% of your total grade.
Corresponding TA: Najeeb Ahmad and Nufail Farooqi, for homework related questions, please use discussion forum at Blackboard.

Problem 1

(10 points) Provide concise answers (2-3 sentences). Use your own words!
a) What is dual mode? Explain the main advantage of dual-mode of operation in an operating system.
b) Describe the steps involved from the boot-time to OS starts running.

Problem 2

(10 points) Write a C program on Unix/Linux that creates a child process. Your program will do the following:
- The parent process creates a child process and the child process must execute a built-in program Linux program (e.g. Firefox web browser).
- The parent process waits for 2 seconds for child program to execute.
- After that, the parent terminates the child process and it has to inform us by printing a message with child's process ID and its ID. Then the parent program exists.

You may use **sleep(), kill()** system call APIs in your assignment. You are required to submit your .c source code file and a snapshot of a sample run on your terminal.

Problem 3

(40 points) Write a C program on Unix/Linux that creates two children processes. Your program will do the following
- The parent process (A) creates one child process (B).
- The parent process A sends the current time to this child process B by using ordinary pipes.
- The child process B prints the time after receiving it and then the child process B sleeps for 3 secs.

- Then the child process B creates another process C and sends the current time to this child process C using ordinary pipes.
- The child process C prints the time after it receives it.
- The child process C sleeps for 3 secs and then sends the current time to its parent process B using ordinary pipes.
- When the parent process B receives the message, prints the received time.
- The child process B sleeps for 3 secs and then sends the current time to its parent process A using ordinary pipes.
- When the parent process A receives the message, prints the received time.
- The parent B terminates the child process C.
- After that, the parent A terminates the child process B.
- Then the parent terminates.

You may use **sleep(), kill(), gettimeofday()** system call APIs in your assignment. You are required to submit your .c source code file and a snapshot of a sample run on your terminal. You may refer to the ordinary pipe example in the textbook.

Problem 4

(40 points)
In this assignment, you will learn how to create a kernel module and load it into the Linux kernel. You can either use the Linux virtual machine or Linux distribution that you installed on your computer. Note that the lab machines will not allow you to run programs as a superuser thus you have to install the linux or virtual machine on your system where you can be the superuser.

Part I: Read Linux Kernel Modules under the Programming Projects from the book in Chapter 2 (Page 120 also provided in Blackboard). Perform only **Part I** of the assignment in the book.

Part II: The second part involves modifying the kernel module provided to you so that it uses the kernel linked-list data structure. We explore the most commonly used data structure, the circular, doubly linked list, that is available to kernel developers. You can examine its implementation under $< linux/list.h >$. By using this linked list, we will define a list of files and add them to the list.

Initially, you must define a struct containing information about a file that can be inserted into the linked list. Use the following C struct for file information:

```
struct fileinfo304 {
   size_t size;
   char filename[100];
   chat datecreated[100];
   int owner_id;
   int file_id;
   struct list_head list;
}
```

Note that one of the members of the struct is list head list, which is defined in $< linux/types.h >$. This list head structure only holds next and previous pointers for next and previous entries in the list. Linux provides series of macro functions (e.g. INIT_LIST_HEAD) to manipulate this list structure.

In the module entry point, create a linked list containing SIX struct $fileinfo304$ elements. Traverse the linked list and output its contents to the kernel log buffer. Invoke the dmesg command to ensure the list is properly constructed once the kernel module has been loaded. In the module exit point, delete the elements from the linked list and return the free memory back to the kernel. Again, invoke the dmesg command to check that the list has been removed once the kernel module has been unloaded. You may refer to the example in the book (Part II) but you will perform the steps for $fileinfo304$.

Part III: now, you will pass an argument, owner ID, to your module. You will remove those files owned by this ID from the list. The module should output the number of files removed and number of files remain in the list to the kernel log buffer. You can find more information about how to pass arguments to a kernel module in this link:
http://www.tldp.org/LDP/lkmpg/2.6/html/x323.html

Notes: You are required to submit your .c source code files and a snapshot of a sample run on your terminal.