Programming project# 2 – Part 2

Due: Monday, April 3 2017, at 11:55PM
(*10% penalty per day for late submission; up to 3 days)*

All the programs should be written, compiled, and executed on the IT Linux servers *unless state otherwise*. You may use your own Debian/Linux virtual machine running on Virtual Box for development and testing. However, you are recommended to test your program on IT Linux servers before you submit your program to ReggieNet (Programming projects to develop Linux kernel modules/code are exceptions).

Programs containing compilation errors will receive failing grades. Those containing run-time errors will incur a substantial penalty. You should make a serious effort to complete all programs on time. Programming assignments are individual. You should complete them with your own effort.

NOTE: Each program should be placed in its own directory. Also, you need to include a Makefile in the same directory as the source code to allow the instructor to compile your program automatically. Similarly, your half-page write-up in plain text format should be placed in the same directory. You will submit a single "ZIP" file that includes all the subdirectories where your programs and documentation are located. The name of ZIP file should be LastName_FirstNameInitial_Program2.zip. (If filename is incorrectly made, you will lose 5% automatically). The single ZIP file should contain C/C++ source files, Makefiles, and write-ups in plain text format in the following subdirectories appropriately ./3 and ./4.

- You might want to use the following Linux/Unix command to create a single zip file:
  % zip –r **program2part2*_yourlastname*.zip** ./3 ./4

To complete this assignment successfully, you will need to read Chapter 3 and Chapter 4 (Thread) in Advanced Linux Programming (NOTE: A copy of the book can be downloaded from the ReggieNet website or directly from the author's website). You might want to try the example code in the two chapters to get better understanding of Pthread APIs.

Programming

[3] (50 points) Multithreaded programming using C/C++ and PThread API.

Create a subdirectory called "3". Change to the subdirectory

Using C or C++ and Pthread APIs, complete **Project 2: Multithreaded Sorting Application** described in pp. 199 in Textbook (9th edition). The following is a copy of the paragraph in p.199 *with some modifications and clarifications:*

> Write a multithreaded sorting program that works as follows. A list of integers **that should be read from a text file** is divided into two smaller lists of equal size. Two separate threads (which we will term sorting threads) sort each sublist using a **quicksort algorithm.** The two sublists are then merged by a third thread – a *merging thread* – which merge the two sublists into a single sorted list.
> Because global data are shared cross all threads, perhaps the easiest way to set up the data is to create a global array. A second global array of the same size as the unsorted integer array will also be established. The merging thread will then merge the two sublists into this second array. Graphically, this program is structured according to Figure 4.20 in the textbook.
> This programming project will require passing parameters to each of the sorting threads. In particular, it

will be necessary to identify the starting index from which each thread is to begin sorting. Refer to the instructions in Project 1-Sodoku Solution Validator in p.198 for details on passing parameters to a thread.
The parent thread will **store** the sorted array **into a text file** once all sorting threads have exited.

- Assumption on input data file
    - Only integer data are stored in input data files.
    - The number of integer elements in input files can be any number between 50 to 5,000,000.
    - Two adjacent elements in input files are separated by a single space, multiple spaces, or new line character.

- You should use **C or C++ and Pthread APIs**. (i.e., You shouldn NOT use Java Thread APIs or WIN32 APIs).
- DO NOT assume the size of input data. The number of integer elements in the input data can be as large as 1,000,000. You will need to allocate a memory space *dynamically* (e.g., using malloc() in C or new in C++) to store all integers read from the given input file.
- Sorting will be done in increasing order. (i.e., The smallest number comes first.)
- The input data file contains integer numbers separated by a single or multiple spaces. You might want to use "strtok()" library function that you should have also used in Lab 1 and Program 0.
- The output data file should contain sorted integer numbers separated by a single space.
- Assuming that your program name is *mt_sort,* you should be able to execute your program as follows:
  % *mt_sort my_input.txt my_output.txt*
- How can I check if my multithread program has sorted the input data correctly?
  % sort –n input.txt > sorted.txt
  % diff sorted.txt my_output.txt

Before you write a multithreaded code, you might want to write a single threaded version of the program first and make sure that your single-threaded version works correctly using many input data, which you need to come up with.

You need to provide a write-up (at least 300 words) describing your source code.

Deliverables: source code, Makefile and separate written document in plain text format (at least 300 words)

[4] (15 points) **Creating Linux kernel modules**
You need to use the Linux Virtual machine that you downloaded from the textbook authors' website.

Do **Linux kernel Modules: part II assignment in Textbook p. 101.** You will need to read pp. 96-101 and pp. 31-32 in textbook carefully before you start to write the code.

Deliverables:

- source code and Makefile (They should be located under a folder called "./4")

- **DO NOT zip your video clip**. You will need to record **1~2 minute long video clip** using your smartphone or any Windows/MacOS software that you like. The video clip should show that you have run make, insmod, rmmod, dmesg commands successfully. (as described in page 98.) Upload the video clip to ReggieNet as an evidence that you have completed the mini-programming project.

Additional Requirements and hints:

- Module name should be “simple.ko”
- <linux/list.h> is located at /usr/src/linux-headers-23.16.0-4-common/include/linux/list.h
- You should add “;” to the following C struct in p.99 when you import the C struct in your code:

```
struct birthday
{
    int day;
    int month;
    int year;
    struct list_head list;
};
```

- The following code shown in p.99 should be located at the beginning of your code, where you typically declare global variables:

```
static LIST_HEAD(birthday_list);
```

- The five struct birthday elements to be added in your code should be unique. For example, the birth years of five struct birthday elements should be all different.
- The syntax of printk(…) is very similar to printf(…). For example,

```
printk(KERN_INFO "year=%d \n", ptr->year);
```