Programming project# 2 – Part 2

Due: 11:55PM on March 25 2015
(*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 points 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, ./4 and ./5

- You might want to use the following Linux/Unix command to create a single zip file:
  % zip –r **program1_*yourlastname*.zip** ./3 ./4 ./5

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
[1] (50 points) Matrix Multiplication Project using C/C++ and PThread APIs.

Create a subdirectory called "2". Change to the subdirectory

Using C/C++ and Pthread APIs, complete **Project 1: Sodoku Solution Validator** described in pp. 197-199 in Textbook (9th edition). Note that you should use **C (or C++) and Pthread APIs**. (i.e., You shouldn't use Java Thread APIs or WIN32 APIs). **Note that you should NOT hard-code a solution to a 9 x 9 Sodoku puzzle in your program**. Rather, your program should read in a text files that contain a solution. In the following, the format of input text files is described. The elements of a solution contained in a text file are separated by space(s) and line feed(s). For example, a text file which contains a solution to a 9 X 9 Sodoku puzzle looks like this :

6 2 4 5 3 9 1 8 7
5 1 9 7 2 8 6 3 4
8 3 7 6 1 4 2 9 5
1 4 3 8 6 5 7 2 9
9 5 8 2 4 7 3 6 1

7 6 2 3 9 1 4 5 8
3 7 1 9 5 6 8 4 2
4 9 6 1 8 2 5 7 3
2 8 5 4 7 3 9 1 6

Assume that Sodoku solutions stored in text files to be validated have the following properties:

- All elements are "positive integer" numbers. However, each number can be larger than 9. For example, 9999 can be an element.
- Two adjacent elements in a given row are separated by a single space character or multiple space characters (e.g., "77 6 2 3 9 1 4 5 888")

Assuming that your program name is *sodoku_solution_checker*, you should be able to execute your program as follows:
% *sodoku_solution checker solution.txt*

After the successful execution of your checker program, the result should be printed on the console screen:

(1) In case of the solution could be validated, simply show "Everything looks good…" on screen.

(2) In case that the given solution is not perfect, indicate which component (e.g., row number, column number, or the index number of 3 x 3 subgrid) has an incorrect number(s).

Before you write a multithreaded code, you might want to write a single threaded version of the program first. Check if your single-threaded version works correctly using many test input data files, which you need to create.

You need to provide a half-page write-up describing your source code.

Deliverables: source code, Makefile, half-page written document (in plain text format), and **at least five test input data files**

Hints:

- If you use C, you are recommended to use strtok() and atoi() to process input data.
- If you like, you may use C++ instead of C language.