CSE 5031 Operating Systems
2019/20 Fall Term

Project: 4 – Part 2
Topic: Multithreaded Programming
Date: 02 - 06.12.2019

Objectives:

- to coordinate threads of execution with POSIX Semaphores
- to implement alternative cooperating threads scenarios

References:

- **Lawrence Livermore National Laboratory Computing Center Pthreads tutorial portal,** https://computing.llnl.gov/tutorials/pthreads/#Pthreads
- **Linux System Programming 2d ed., Robert Love, O'Reilly 2013** (course web site, or http://pdf-ebooks-for-free.blogspot.com.tr/2015/01/oreilly-linux-system-programming.html
- **The GNU C Library Reference Manual** (course web site, or http://www.gnu.org/software/libc/manual/pdf/libc.pdf)

Section I. Project Definition

I.1 Project Scope

The project aims to develop simple **multithreaded C** applications, and experiment with the coordination of **threads** of execution, using **POSIX semaphores**.

You are asked to develop two coordination scenarios:

i. Threads cooperating according to the **Producer – Consumer** paradigm.
ii. Threads cooperating under **spawn – join** paradigm (generating several concurrent threads, and waiting for them to use the results they produced independently).

The project aims primarily at experimenting with **multithreading** and **synchronization** of **concurrent** activities; thus the complexity of the application is a secondary issue. To keep the problems simple, you are asked to program computation of the value of a single variable function.

I.2 Implementation Constraints

The **argument passing** mechanism to **threads** is rather limited.

> "*The pthread_create() function permits the programmer to pass only one argument and this argument must be passed by reference and cast to (**void ***).*
> *For cases where multiple arguments must be passed, this limitation can be overcome by creating a structure which contains all of the arguments, and then passing a pointer to that structure in the call to the pthread_create() routine.*"

You are asked to pass the address of an operand as thread's argument, if it uses only one operand to compute the result e.g computing the square of an integer or multiplying an integer by 2.

Given our focus and syntactical limitation of the *pthread_create()* function, the applications you will develop cannot be easily generalized to process any algebraic expression. As such, next sections provide ad-hoc guidelines to simplify your implementation and let you to focus on the thread synchronization issue.

Given the simplicity of the computation of the value of an algebraic function, you will use the "*sleep (seconds)*" call to mimic the heavier processing loads, and observe a more realistic execution pattern.

I.3 Implementation Platforms

The project may be developed and tested on:

- ✓ the **Oracle Virtualbox VM** platform, or
- ✓ on the **laboratory workstations** running under **CentOS 7** alike.

Section II. Implementing the Producer – Consumer Scenario

II.1 Problem Statement

To study **threads** cooperation using the **Producer – Consumer** model, write **a C** program computing the equation:

$$res = 2\,x^2$$

The scenario starts as the main thread reads an integer "x" from stdin (produces x). A second thread retrieves "x" (consumes x) and **computes** its square (produces x^2). A third thread retrieves "x^2" (consumes x^2) and multiplies it by 2 (produces $2\,x^2$). The main thread retrieves "$2\,x^2$" (consumes $2\,x^2$) and displays the result on stdout.
As the scenario depicts well, a producer may in its turn be the consumer of another service. **Producer – Consumer** paradigm is commonly implemented in Computer Network and is referred to as the "**Client-Server**" model.

II.2 Implementation Guidelines

Organize you program in **three threads** of execution, performing the following actions:

a) **main thread** the default thread running main( ) function should:
- ✓ initialize the synchronization semaphores;
- ✓ create two threads:
  - one to compute "**the square**" of an integer, and
  - the other to "**multiply**" by an integer 2,
- ✓ read the integer "x" from stdin;
- ✓ trigger the square thread to compute x^2;
- ✓ wait for the end of the "multiply by 2" and display the result;
- ✓ wait for the termination of the threads.

b) **square thread** computing power of 2 of an integer:
- ✓ waits for the value of **x** to be ready;
- ✓ computes x^2 of the argument whose address is passed in argument by reference;
- ✓ sleeps for 1 second;
- ✓ stores the result in a global variable;
- ✓ triggers the multiply by 2 thread;
- ✓ terminates

c) **multiply thread** computing **power 2** of an integer:
- ✓ waits for the value of x^2 to be ready;
- ✓ multiplies the integer whose address is passed in argument by reference;
- ✓ stores the result in a global variable known by the main thread;
- ✓ sleeps for 1 second;
- ✓ triggers the main thread;
- ✓ terminates

Section III. Implementing the Spawn and Join Scenario

III.1 Problem Statement

To study **threads** cooperation using the **Spawn-and-Join** model, write **a C** program computing the equation:

$$res = 2\,x + x^2$$

The scenario starts as the main thread reads an integer "x" from stdin. Given that the operator "+" is commutative both of its operands can be computed concurrently. Two more threads may be spawned to compute the "$2\,x$" and "x^2" concurrently. A forth thread should wait for both threads, then add both results yielding in "$2\,x + x^2$". Finally main thread should be triggered to display the result of this addition on stdout.

III.2 Implementation Guidelines

Organize you program in **four threads** of executions:

- ✓ **main thread**
- ✓ **square thread**
- ✓ **multiply thread**
- ✓ **sum thread**

You may reutilize the program you have developed in **Section II** and use the implementation of the **square**, **multiply** and **main threads** with minor modifications. All you need is to add the **sum thread**, and implement the **spawn – join** paradigm using POSIX Semaphores.

Section VI. Project IV Part 2 Report

Submit the programs implementing the **Producer – Consumer** and the **Spawn-and-Join** paradigms only if they are running correctly!

Add a comment line consisting of your name and student-id**;** and store **source code** files in the "**Prj4-Part2**" folder, located at the course web site under the tab **CSE5031-X/Assignment**; where "**X**" stands for (A,B,C,D) your laboratory session group you are registered.

Warning

You are encouraged to discuss the implementation procedures and general concepts behind the projects with your fellow students. However, **plagiarism is strictly forbidden**! Submitted report should be the result of **your personal work**!

Be advised that you are **accountable** of your submission not only for this project, but also for the mid-term, and final examinations. Your project grade may be reevaluated retrospectively, had you fail to answer correctly the same or a similar examination questions that you have solved with success in your submissions.