CSE 5031 Operating Systems
2020/21 Fall Term

Project: 4 – Part 2
Topic: Multithreaded Programming
Date: 07 - 14.12.2020

Objectives:

- to implement multi-thread application with **Pthreads**
- to coordinate threads of execution using POSIX **Semaphores**

References:

- **Lawrence Livermore National Laboratory Computing Center Pthreads tutorial portal,** https://computing.llnl.gov/tutorials/pthreads/#Pthreads
- **Linux The Textbook. S.M. Sarwar, R.M/ Koretsky. CRC Press 2019. ISBN 978-1-138-71008-5**
- **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 A. Project Definition

A.1 Problem Statement

The project aims at reprogramming the **multithreaded** application you have developed in part1, using POSIX **Semaphores API** to implement the **thread coordination**.

The **multithreaded** program computing the value of the algebraic expression **$2x^2 + x / 2$** will be implemented in two phases.

→ **Phase one** will implement the **Producer** – **Consumer** paradigm to compute the expression **$2x^2$** with two threads as in part1.

→ In **phase two** you will compute the expression **$2x^2 + x / 2$** with four threads. As the operands of "**+**"operation can be computed in parallel, you will usethe **Spawn** – **Join** paradigm to organize their execution.

The **main thread** will spawn the evaluation of the expression in two flows once x value is entered:

- ✓ the first flow will compute **$2x^2$**; and
- ✓ the second the division **$x / 2$**.

The **addition thread** will **join** both flows; and evaluate the sum. It will also notify the main thread following the "**+**"operation.

For both phases the **main thread** will read from the keyboard an integer and display the result of the computation.

A.2 Implementation Constraints

To simplify your implementation you will use **shared global variables**:

- ✓ to pass the parameters to threads and return computed results; and
- ✓ to define POSIX semaphores.

To sketch heavy processing, computing threads will suspend their execution for 10 seconds by calling the function **"*sleep (seconds)*"** before delivering their result.

You are expected to display the actions taken by each thread:

- ✓ once they are activated and they have copied the shared variable; and
- ✓ before they go to **'sleep'** (perform heavy computing) or they end.

This will help you to **trace** the evolution of the threads, and hopefully to detect **faulty behavior** without using a thread-enabled debugger.

Section B. Implementing the Producer – Consumer Scenario

B.1 Coordination Design

Define all the **semaphores** and **variables** used by the **threads** to compute **2 x^2** . Draw **thread coordination schema** as presented in lectures to depict the events that trigger the actions and the workflow –course of action-.

B.2 Implementation Guidelines

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

a) **main thread** - *the default thread running main( ) function-* should:
- ✓ initialize the **semaphores**;
- ✓ create the **square** and **multiply by 2** threads, and display their **TID**;
- ✓ read the integer "**X**" from stdin;
- ✓ trigger the **square thread** to compute **x^2**;
- ✓ wait for the result produced by **multiply by 2**;
- ✓ display the result when ready;
- ✓ wait for the termination of both **threads**;
- ✓ terminates.

b) **square thread**:
- ✓ displays its **TID** and starting message;
- ✓ waits for the **production** of **x** value;
- ✓ copies the value of global variable **x** to a local variable;
- ✓ displays its **TID** and the start of the computation process;
- ✓ sleeps for 10 seconds
- ✓ computes **x^2**, stores the result in a global variable and displays its **TID**, the computed value;
- ✓ posts the event for the process waiting the result;
- ✓ displays its **TID** and termination message;
- ✓ terminates.

c) **multiply by 2 thread**:
- ✓ displays its **TID** and starting message;
- ✓ waits for the **production** of **x^2** value;
- ✓ copies the value of global result variable to a local variable;
- ✓ displays its **TID** and the start of the computation process;
- ✓ sleeps for 10 seconds;
- ✓ computes **2 x^2**, stores the result in a global variable and displays its **TID** and the computed value;
- ✓ posts the event for the process waiting the result;
- ✓ displays its **TID** and termination message;
- ✓ terminates.

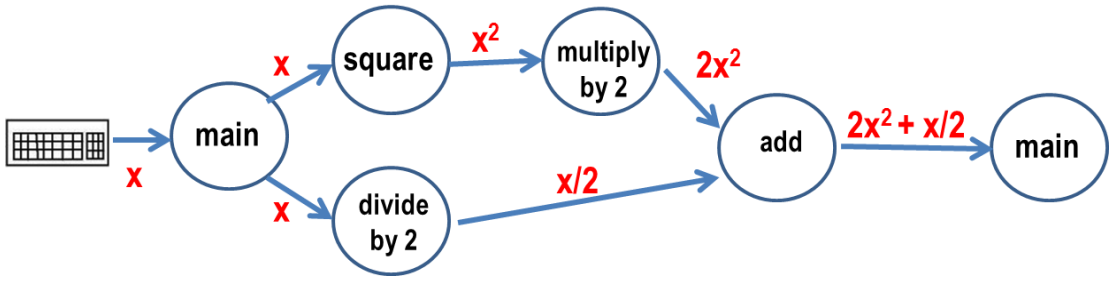
B.3 Reporting

Once the program performs as specified,
- ✓ add a comment line in each stating your name and student-id ;
- ✓ name the source code as "**prod_cons.c**".

Section C. Adding the Spawn and Join Paradigm

C.1 Problem Scope

Expand the C program you have developed in **section B** to compute the expression **2 x^2 + x / 2** with four threads. Implement the **Spawn** – **Join** paradigm depicted here after.



C.2 Implementation Guidelines

Organize your program in **five threads** of execution:

- ✓ main,
- ✓ square,
- ✓ multiply by 2,
- ✓ divide by 2, and
- ✓ add.

Implement your **threads** using the guidelines outlined in **section B.2**.

C.3 Reporting

Once the program performs as specified,

- ✓ add a comment line in each stating your name and student-id ;
- ✓ name the source code as "**spawn_join.c**".

Section D. Project IV Part 2 Report Submission

Do not submit a result if your program does not work as specified.

Store your code files

- ✓ "**prod_cons.c**".
- ✓ "**spawn_join.c**".

in the "**Prj4-Part2**" folder, located at the course web site under the tab **CSE5031 - OS Section -X/Assignment**; where "**X**" stands for (1,2,3,4) your laboratory session group.

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.