CSE 5031 Operating Systems
2020/21 Fall Term

Project: 4 – Part 1
Topic: Multithreaded Programming
Date: 01 - 07.12.2020

Objectives:

- to implement multi-thread application with Pthreads
- to coordinate threads of execution using shared variables

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 A. Project Definition

A.1 Problem Statement

The project aims at developing two simple **multithreaded** applications in **C**, using POSIX **Pthread API**, and implement **threads'** coordination with shared state variables and busy wait loops.

Multithreaded program computing the **res = 2 x^2** equation will be implemented in two phases, using the **Producer – Consumer** paradigm.

→ In **phase one** you will define one thread that computes the **square** of a number **x^2**.

→ In **phase two** you will define two threads

- ✓ the first thread computing the square of a number **x^2**.
- ✓ the second thread multiplying the value computed by the first one by two yielding in **2 x^2** .

In both phases the **main thread** will read from the keyboard an integer until the **end-of-file** (left ctrl+d) is entered, and display the result computed by the first thread in phase I, the second thread in phase II.

A.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."*

To simplify your implementation you will implement this project using **shared global variables**:

- ✓ to pass the parameters. and
- ✓ to set/unset synchronization events.

Thread synchronization will be implemented using **busy loops**. Each thread will check if the event it is waiting for is posted until the computation ends by entering the **end-of-file** (left ctrl+d) character.
If the **event is posted**, that is a new value is produced, the thread will consume it and perform its computation; then **post the event** for thread that will consume it.
If the **event is not posted**, thread will sleep for 10 seconds by calling the function "***sleep* (seconds)"**.

Refer to the "t-norace.c" program stored at the course portal under Resources / Reference C programs.

Section B. Implementing Phase I

B.1 Problem Scope

Write a C program in which the **main thread** reads an integer from stdin until **end-of-file** and waits the thread that computes the square of this number to display the result on stdout.

As the scenario depicts well, a **producer** (the main thread) may in its turn be the **consumer** of another thread (square) while the square thread also plays both roles in its turn.
Note that the **Producer** – **Consumer** paradigm depicted herein is referred to as the "**Client-Server**" model in Computer Networks.

B.2 Implementation Guidelines

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

a) **main thread** - *the default thread running main( ) function-* should:

- ✓ initialize the synchronization variables;
- ✓ create the **square** thread;
- ✓ read the integer "**X**" from stdin until end-of-file;
  - trigger the square thread to compute $\mathbf{X^2}$;
  - wait for the result of the "**square**" thread and check every other second if it is computed;
  - display the result when ready;
- ✓ signal the **square** thread the **end-of file** event;
- ✓ wait for the termination of the **square** thread;
- ✓ terminate.

b) **square thread** computing **power of 2** of an integer:

- ✓ waits for a new **X** by checking producer event every 10 seconds till the end-of-file is posted;
  - computes $\mathbf{X^2}$;
  - stores the result in a global variable;
  - posts the event for the process waiting the result;
- ✓ terminates on end-of-file signal.

Section C. Implementing Phase II

C.1 Problem Scope

Expand the C program you have developed in section B to compute $\mathbf{2\ X^2}$ using 3 threads.

- ✓ The **main thread** will read an integer from stdin until **end-of-file** and wait the thread that the result is computed and display it on stdout.
- ✓ The **square** thread will compute $\mathbf{X^2}$ **;** and
- ✓ the **multiply by 2 thread** will compute $\mathbf{2\ X^2}$ and notify the main thread.

C.2 Implementation Guidelines

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

a) **main thread** the main (default)t thread is similar to the one defined at the previous step except that

- ✓ it creates the **multiply by 2 thread** in addition; and
- ✓ waits the computation result from **multiply by 2 thread.**

b) **square thread** computes power of 2 of an integer operates as before, except that it triggers **multiply by 2** instead of the main.

c) **multiply by 2 thread**:

- ✓ waits for the value x^2 to be ready by checking producer event every 10 seconds till the end-of-file is posted;
  - computes $2\ x^2$;
  - stores the result in a global variable;
  - posts the event for the process waiting the result;
- ✓ terminates on end-of-file signal.

Section D. Project IV Part 1 Report

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

- ✓ Name the **source codes** you have developed for phase I and II as **phase1.c** and **phase2.c** respectively; add a comment line in each stating your name and student-id.
- ✓ Store your code files in the "**Prj4-Part1**" 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.