Homework 1 (CSE 6431 – Advanced Operating Systems, total 90 points)
This assignment must be done individually
Date Given Out: Jan. 31, 2023
Date Due: Feb 7, 2023 **before class (i.e., before 11am) No late submission is allowed!**
Submission Instruction:
For both Part I and Part II, please submit them as two separate attachments and email to both the grader (ren.450@buckeyemail.osu.edu) and projectsubmissionaddress@gmail.com. The email subject should be "CSE 6431_YourFirstName_YourLastName."

Part I (40 points)

1. [10 points] In this problem, you are to compare reading a file using a single-threaded server and a multi-threaded server. It takes 15 ms to get a request for work, dispatch it, and do the rest of the necessary processing, assuming the required data is cached in the server's memory. If a disk operation is needed, as is the case one-third of the time on this system, an additional 75ms is required, during which the thread sleeps. How many requests/sec can the server handle if it is single threaded?
   How many requests/sec can the server handle if it is multi-threaded?

2: [20 points] Five batch jobs A through E, arrive at a computer center at almost the same time. They have estimated running times of 10, 6, 2, 4 and 8 minutes. Their (externally determined) priorities are 3, 5, 2, 1, and 4, respectively, with 5 being the highest priority. For each of the following scheduling algorithms, determine the mean process turnaround time. Ignore process switching overhead.
(a) Round robin.
(b) Priority scheduling.
(c) First-come, first-served (run in order 10, 6, 2, 4, 8)
(d) Shortest job first.
For (a), assume that the system is multiprogrammed, and that each job gets its fair share of the CPU. For (b) through (d) assume that only one job at a time runs, until it finishes. All jobs are completely CPU bound.

3: [10 points] A soft real-time system has four periodic events with periods of 50, 100, 200 and 250 ms each. Suppose that the four events require 35, 20, 10 and x msec of CPU time, respectively. What is the largest value of x for which the system is schedulable?

Part II: Programming (50 points)

Objectives:

The objectives of this assignment are the following: Acquire familiarity with using POSIX thread management primitives. How to write POSIX thread programs and communicate through shared memory. Read the manual pages for the LINUX primitives for creating shared memory between two LINUX processes. These functions are -- shmget and shmat. For sample programs on how to use the POSIX thread library and its shared memory management, you can use Google to search for "POSIX Thread Programming Examples".

Problem Statement:

In this assignment you will write a POSIX program. Consider a system comprising of three producer threads, one consumer thread, and a shared buffer of size 2. Each producer thread randomly (over time) creates a specific colored item: producer_red creates red colored items, producer_black creates black colored items, and producer_white creates white-colored items. Associated with each item created by a specific producer is a local LINUX timestamp in microseconds (use the gettimeofday() function) which records the time that item was placed into the buffer. Each producer deposits its item into the buffer, and the consumer retrieves the items from the buffer. The items produced by the producers will be of any of the three strings below:

“RED timestamp”, “BLACK timestamp”, “WHITE timestamp”

Each producer threads, after successfully depositing an item into the buffer, will write that item (essentially the “COLOR timestamp” string) into its log file called Producer_COLOR.txt (e.g. Producer_RED.txt). The consumer thread will then retrieve the items from this shared buffer. When it retrieves an item, the consumer thread will write that item (essentially the “COLOR timestamp” string) into a file called Consumer.txt.

Requirements for the program:

1. Each producer thread will produce its respective colored item (represented as a string “COLOR timestamp”). If the producer thread finds the shared buffer has space, it will first create a timestamp for that item. The producer thread will then write the item into its log file (Producer_COLOR.txt) and also deposit the item into the buffer. If the buffer does not have space, the producer will have to wait. Each string (or item) written into the log file will be on a new line.

2. The consumer thread will take an item from the buffer, and write its contents to Consumer.txt file. Each string should be written in the order it was read from the buffer, and in a new line. If the buffer was initially full, the consumer thread should signal any one of the producer threades.

3. Each producer thread should produce 1000 items. Hence at the end of running the program, the consumer should have written 3000 items into its log file.

Extra Guidelines:

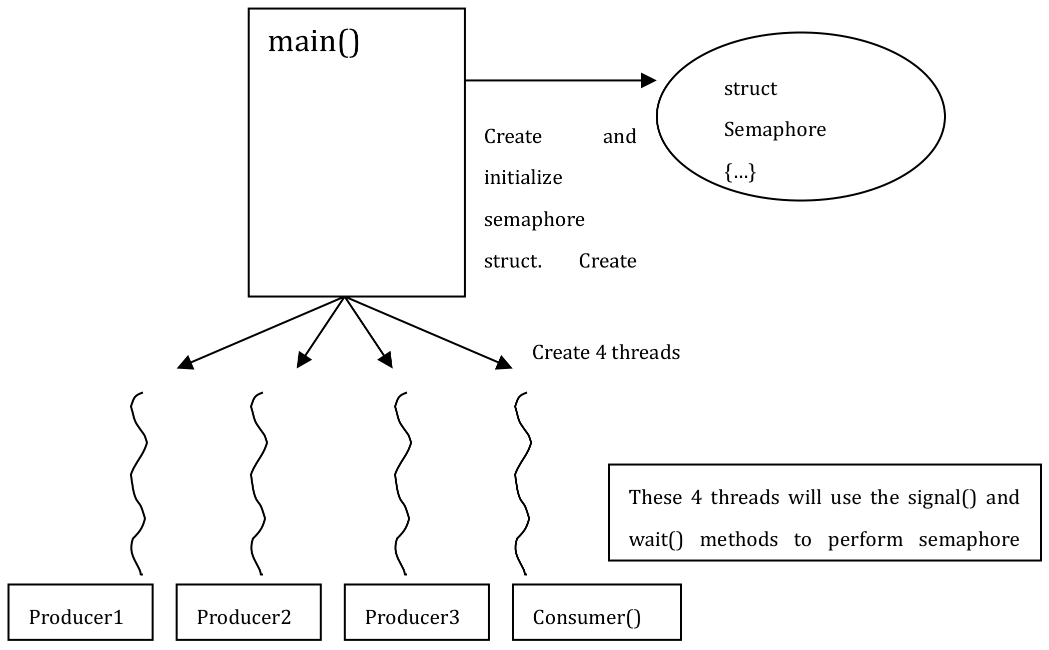
- One of the objectives of this assignment is to implement counting semaphores using POSIX mutex and condition variables. Hence, you *cannot* use **semaphore.h** (which is part of the 3RT library) or any other counting semaphore library.

You can only use the following .h files:

```
#include<pthread.h>
#include<stdio.h>
#include<string.h>
#include<stdlib.h>
#include<unistd.h>
#include <sys/time.h> //for gettimeofday()
```

To do tasks: Write the code for this system as a single LINUX process containing four POSIX threads. The functions of each producer and consumer should be performed by one separate POSIX thread.

Recipe:


main()
Create and initialize semaphore struct. Create
struct Semaphore {...}
Create 4 threads
These 4 threads will use the signal() and wait() methods to perform semaphore
Producer1
Producer2
Producer3
Consumer()

1.) In your program, define a semaphore struct (which would obviously have a counter, a pthread_mutex, and a pthread_condition variable).
2.) Define signal() and wait() methods which will be used on the semaphores.
3.) Your main() program can create and initialize the required semaphores. It will then spawn 4 threads, which will perform their functions as specified in the homework. The threads will obviously have to use the implemented signal() and wait() methods to perform semaphore operations.

Items to be submitted in a LINUX tar file:

(1) Code for your program. This should be in a file named prod_cons.c.

(2) A makefile used to compile your code

(3) A Readme.txt file with instructions on how to run your program