CS39002: Operating Systems Lab
Spring 2017

Assignment 4
Due: Part 1 due March 5, 6 pm, Part 2 due March 12, 6 pm

1. Consider an $n \times n$ matrix X for some $n > 0$ with rows and columns numbered from 0 to $n - 1$. Given such a matrix, we define a *k-shuffle* of X as repeating the following pair of operations k times: first circular shift the arrays from right-to-left, and then circular shift the columns from top-to-bottom. The example below shows the result of 1-shuffle of a matrix X.

X

0	3	5	1
1	4	2	2
2	1	3	2
1	2	3	4

After Row Shift

3	5	1	0
4	2	2	1
1	3	2	2
2	3	4	1

After Column Shift

2	3	4	1
3	5	1	0
4	2	2	1
1	3	2	2

We wish to do it using x threads ($x \leq n$). Each thread will take horizontal stripes of (n/x) rows (with the last thread possibly taking more if n is not divisible by x) during row shift and vertical strips of (n/x) columns during column-shift and shift the entries. For example, if you have two threads 0 and 1, the thread 0 will shift rows 0 and 1 and thread 1 will shift rows 2 and 3 during row-shift for the X shown above. Similarly, thread 0 will shift columns 0 and 1 and thread 2 will shift columns 2 and 3 during column shift. As is obvious, no thread can start a column shift until all threads have finished a row shift. Similarly, if $k > 1$, no thread can start the row shift of the i-th phase until all threads have finished the column shifts of the $(i - 1)$-th phase.

Write a program *shuffle.c* that does the following:

- Read in an $n \times n$ matrix row-by-row from the keyboard (you can store it in file and read using the < operator)
- Print the matrix nicely
- Read in k and x (assume $x \leq n$ will be entered)
- Declare and initialize any mutex/condition variables that you may need.
- Write a single *start_routine*() function that all threads will execute. The function should have (through a structure passed through the void * argument) parameters to somehow identify which part of the array it should work on, and the type of shift (row or column).
- Create x threads with appropriate parameters.
- The main process waits for all of them to finish, and then prints the final matrix and terminates

You will have to design the synchronization between the threads. You must write how you are synchronizing between the threads clearly as a comment just before

start_routine(). Use only pthread_* functions to synchronize between threads. Submit the single file *shuffle.c*.

2. In this assignment, we will simulate the working of a server that requests service requests from users continuously, and then assigns one thread to each service request. A typical example is a web server that provides some service, for example, search, ticket booking etc.

   Create two processes (not threads) A and B where A is a producer process and B is a consumer process, with a queue of size 10 between them (you can borrow your code from last assignments). Process A will act as the generator of requests, and process B as the receiver of requests. For each request received, process B will create a thread to handle that request, and then will come back to check if any other requests are there in the queue. Process A and B runs perpetually, but with the following conditions: (i) If process B is killed somehow (say ^C pressed by mistake), process A should detect that and also terminate automatically, and (ii) whenever a process terminates, it must clean up all shared memory, semaphore, thread mutex/condition variable etc. before it terminates (Use signal handlers to catch the signal for ^C).

   Process A does the following in an infinite loop: sleep for a random time between 0 and 2 seconds, generate a random number between 1 and 5, and add the number in the queue. The number put in the queue acts as a request.

   Each request contains just an integer x (positive or negative). Process B maintains a global variable *Ticket* that is initialized to 100. For each request, B creates a thread that executes a routine *book_ticket*() that takes the integer x in the request as a parameter. If $(Ticket - x) \geq 0$, *book_ticket*() sets $Ticket = \min(Ticket - x, 100)$, sleeps for a random time between 0 and 2 seconds, prints the value of *Ticket*, and returns 1. If $(Ticket - x) < 0$, *Ticket* is not changed, the function sleeps for a random time between 0 and 2 seconds, prints a message saying request cannot be fulfilled, and returns 0.

   However, handling too many requests at once can slow down the server. So B also keeps track how many requests are being serviced (threads started but not finished), and if the number is greater than some fixed value (for this assignment, use 10), B is blocked till the number of worker threads comes to less than or equal to 5 again.

   Write the codes for process A and B in two files A.c and B.c. You should run the two executables from two terminals. You should use shared memory and semaphores for all synchronization between A and B (you can use either System V or pthreads). You must use pthreads mutex/condition variables for all synchronizations between B and the threads and between the threads themselves. Submit the files A. c and B.c.