INDIAN INSTITUTE OF TECHNOLOGY PALAKKAD

COMPUTER SCIENCE AND ENGINEERING
Indian Institute of Technology Palakkad
CS3110: Operating Systems Lab
Lab 9

21-Oct-2024

1. Write a program that takes 2 integer arguments N and B. The program should spawn N producer threads, where thread i produces 1000 multiples of $(i+1)$ and pushes them onto a shared buffer of size B. The program should also spawn N consumer threads, where thread $i \in \mathbb{N}$ consumes and prints the first element in the buffer only if it a multiple of $(i+1)$. Implement the correct producer/consumer synchronization using (a) condition variables and lock; (b) only semaphores.

2. Implement a solution to the **fork/join problem** using semaphore. Test by adding `sleep(1)` calls in various locations.

3. Implement a solution to the **rendezvous problem** using semaphores. The problem is as follows: you have two threads, each of which are about to enter the rendezvous point in code. Neither should exit this part of the code before the other enters it. Test by adding `sleep(1)` calls in various locations.

4. Assume there are two points in a sequential piece of code, called P_1 and P_2. Putting a barrier between P_1 and P_2 guarantees that all threads will execute P_1 before any one thread executes P_2. Write the code to implement a `barrier()` function that can be used in this manner. It is safe to assume you know N (the total number of threads in the running program) and that all N threads will try to enter the barrier. Test by adding `sleep(1)` calls in various locations.

5. Use mutex lock and condition variables to build a semaphore.

6. Implement **reader-writer** lock using semaphore. Test by adding `sleep(1)` calls in various locations. Show the existence of writer starvation.

7. Re-implement **reader-writer** lock so that all readers and writers eventually make progress.

8. Use semaphores to build a **no-starve mutex**, in which any thread that tries to acquire the mutex will eventually obtain it.

End of document