

**Lab 5 Report**

**Report Subject: OS Experiment - Lab 5**

**Student ID: 2018380039**

**Student Name: Dikshya Kafle**

**Submission Date：2021/11/24**

**Computer Operating System Experiment**

**Laboratory 5**

**Synchronization**

1. **Objective:**

* Learn to work with Linux and Pthread synchronization mechanisms.
* Practice the critical-section problem
* Examine several classical synchronization problems

1. **Equipment:**

* VirtualBox with Ubuntu Linux

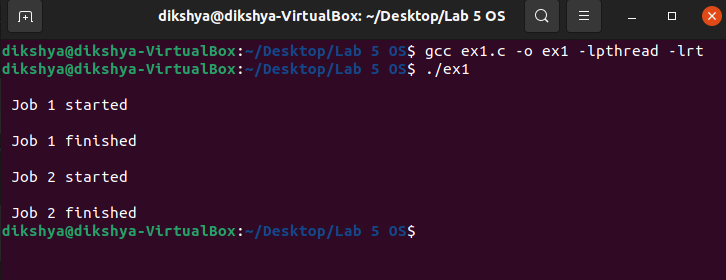
1. **Experiments:**

**Experiment 1: Thread Synchronization Problems**

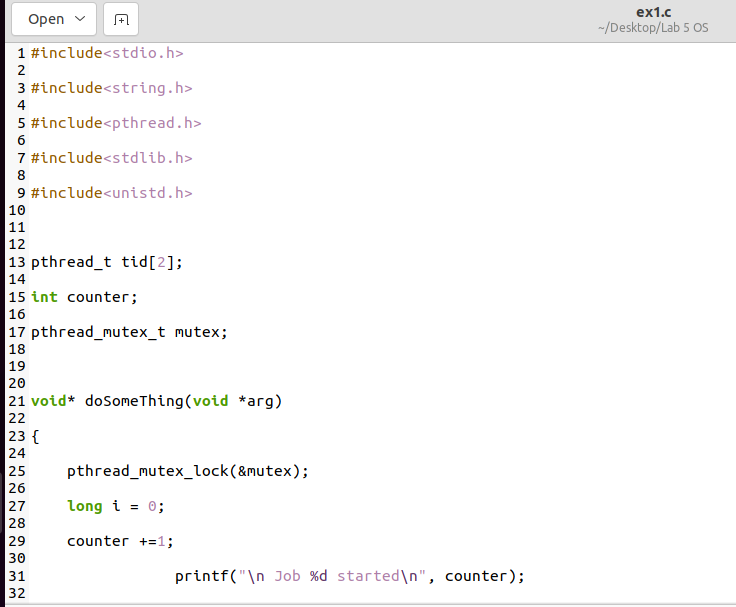
1. Do you think that the counter increase correctly? If not, what is wrong? And please increase the

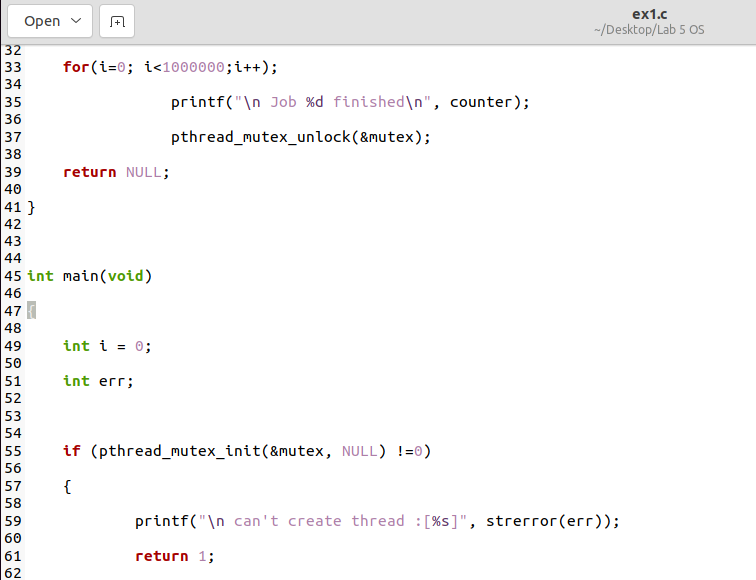
counter correctly using multithreads

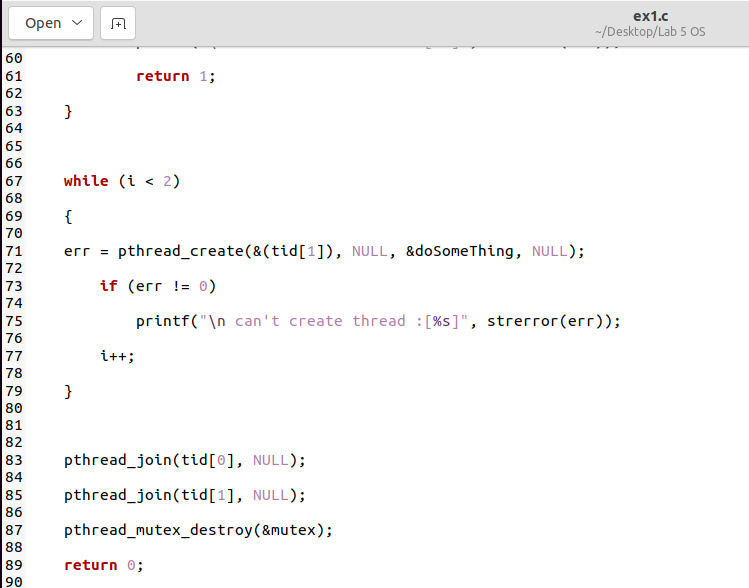
The counter is a shared variable that will be edited by the threads. However it will not increase correctly cause it is not protected. This causes a race condition where the values of “counter” are unpredictable and vary depending on the timings of context switches of the threads in this case. To illustrate this, we run the program multiple time and we notice the different behaviours:



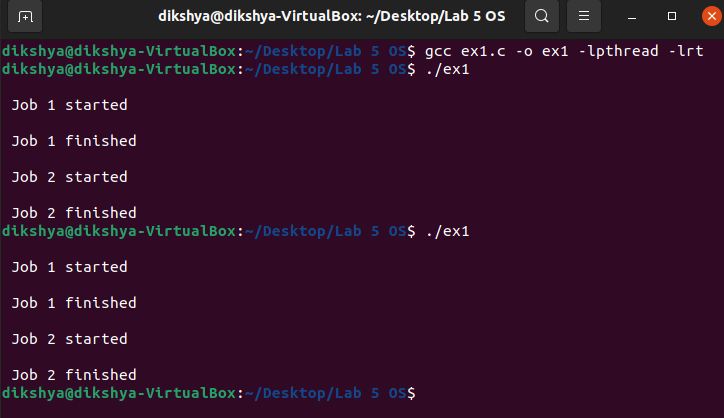
Notice how both threads will access the shared variable and result in race condition. To correct this, we need to protect the shared variable “counter” and make only one threads able to edit it at a time. For this we use the pthread\_mutex\_t data type (or a binary semaphore):







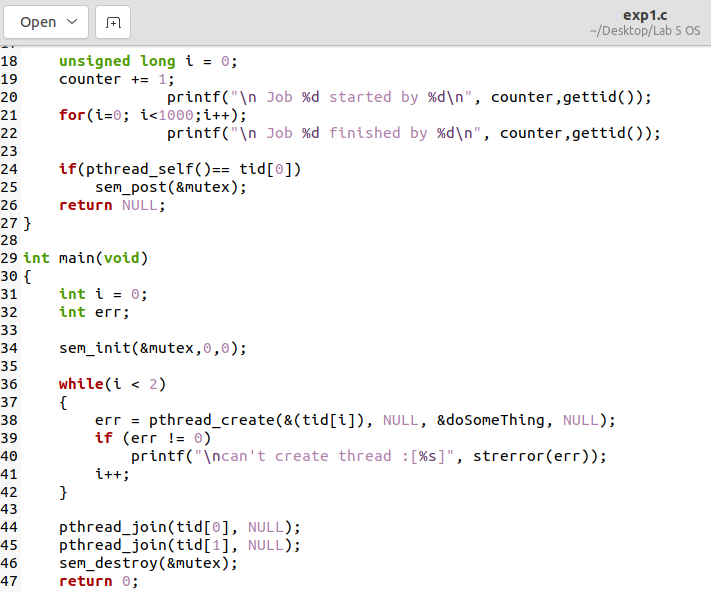
Results:



1. if Thread 1 must run before Thread 2, how do you do?

The semaphore is initially to zero and we make the second thread wait first until the first thread signals the lock.





**Result:**



**Experiment 2: The Bounded-Buffer Problem**

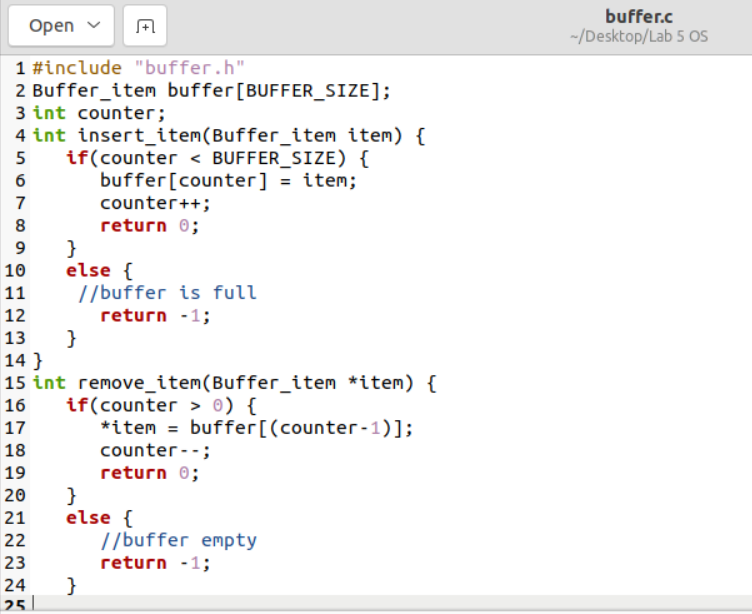
As specified, The buffer is manipulated with two functions, insert item() and remove item() , which are called by the producer and consumer threads, respectively.

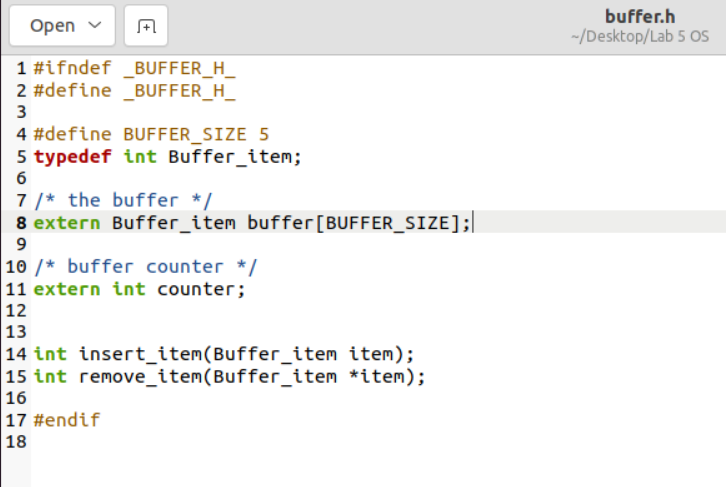
The main() function initializes the buffer and create the separate producer and consumer threads then sleeps for the determined period. Upon awakening, the program is terminated.

The producer thread alternates between sleeping for a random period of time and inserting a random integer into the buffer. Random numbers are produced using the rand() function, which produces random integers between 0 and RAND MAX . The consumer also sleeps for a random period of time and, upon awakening, it attempts to remove an item from the buffer.

* Command: Prod\_com [num\_prod] [num\_cons] [sleep\_time]

The needed application programming interface for the bounded buffer ADT (buffer.h & buffer.c):

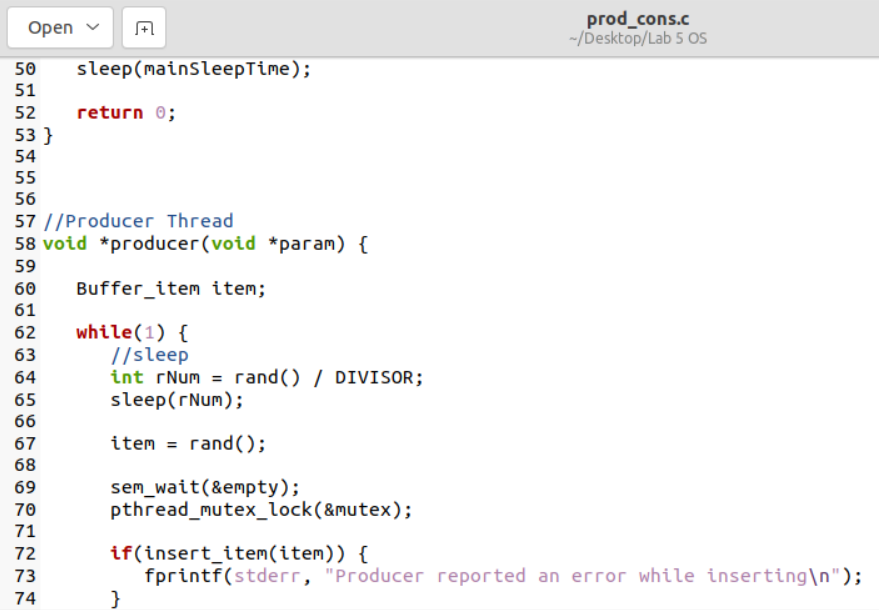


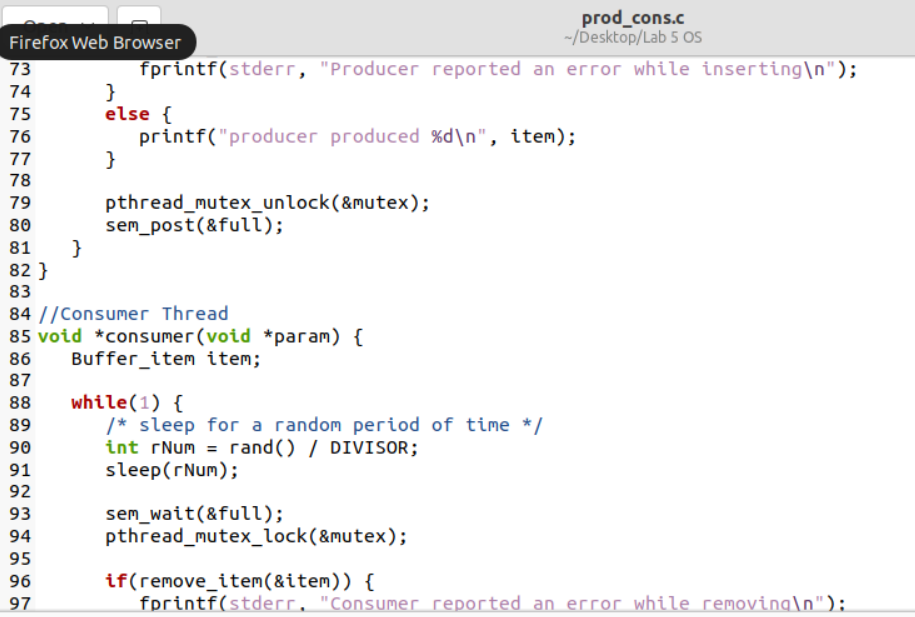


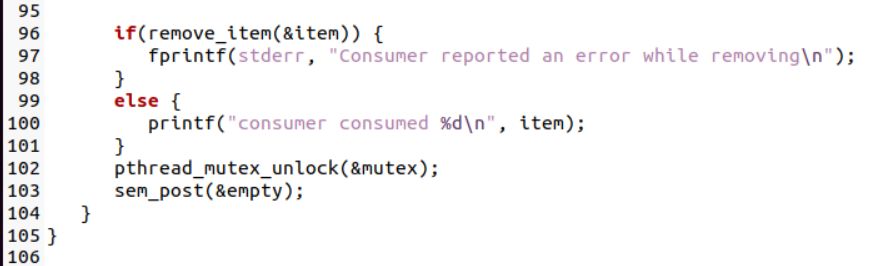
The program (prod\_cons.c):



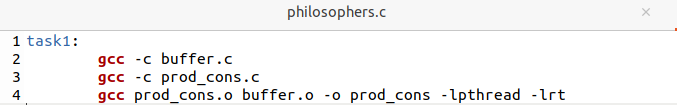




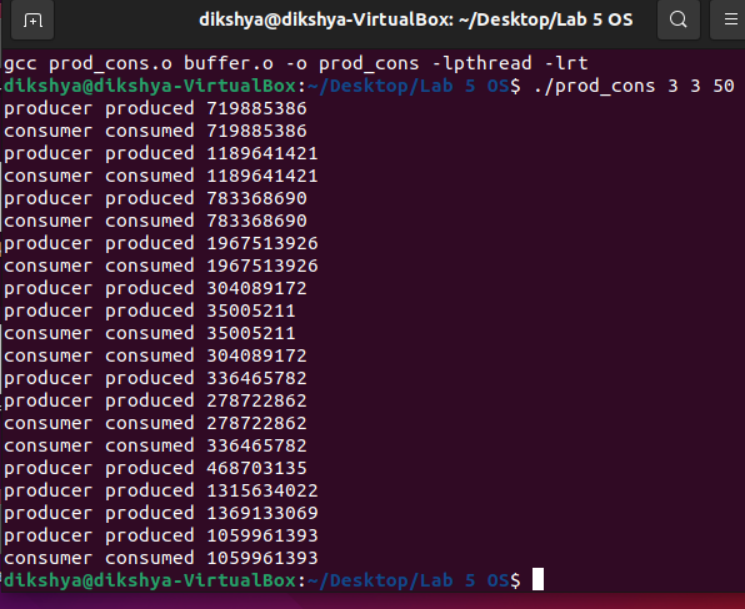




The Makefile:



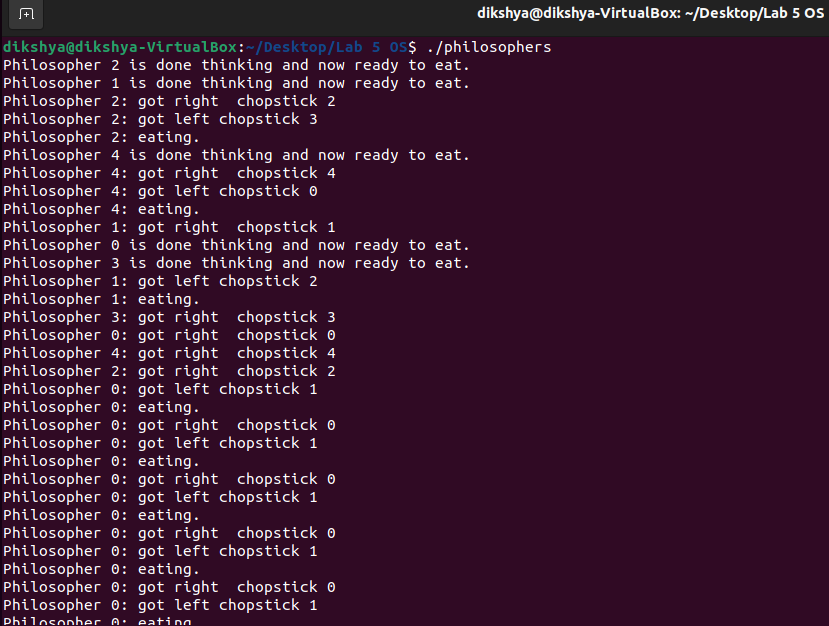
The result:



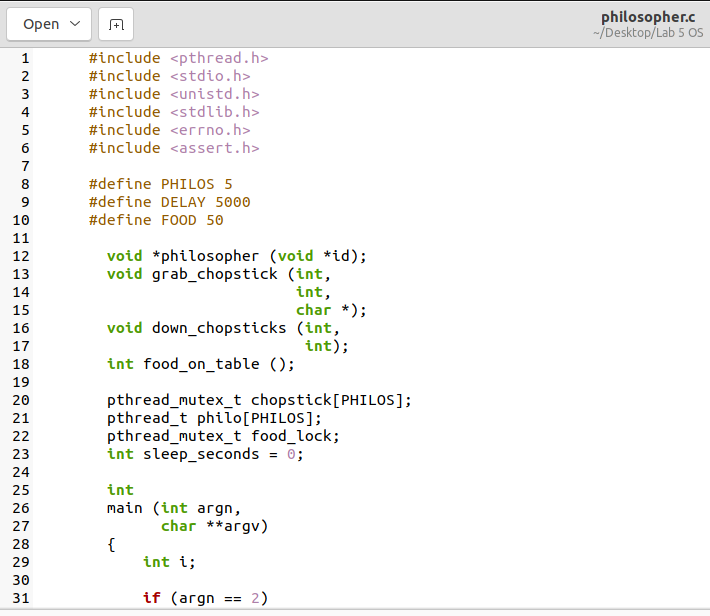
### 3.3 Experiment 3: Dining Philosophers (optional if you are still available)

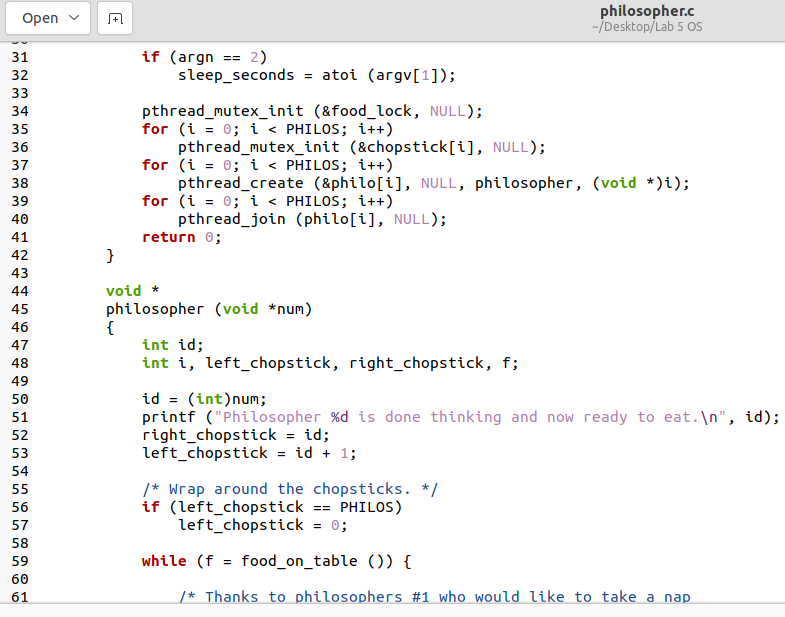
Questions:

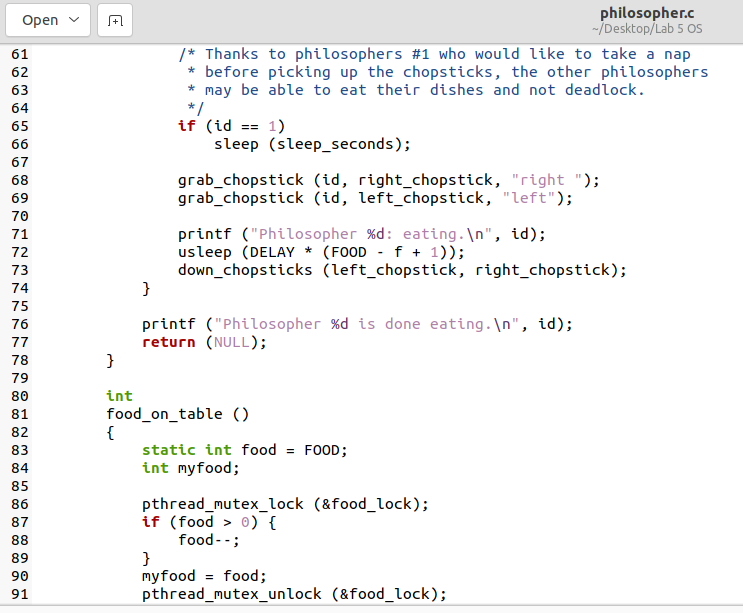
1. Do you think that this solution is good? If not, what is wrong?

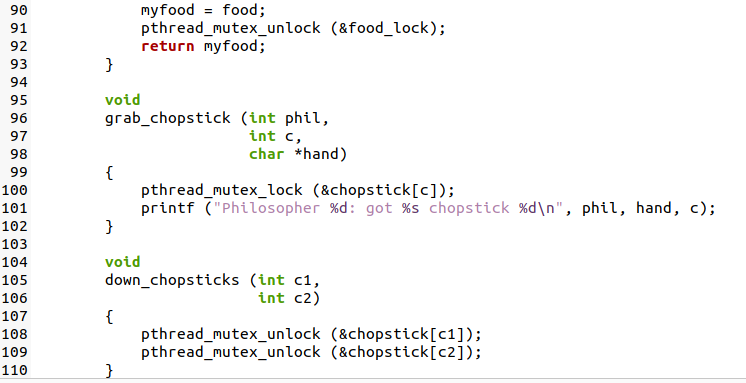


1. Can you modify this solution to meet the requirements? And use semaphores to implement it.









Results:

