Bahria University,

Karachi Campus



LAB EXPERIMENT NO.

\_\_\_10\_\_\_

LIST OF TASKS

|  |  |
| --- | --- |
| TASK NO | OBJECTIVE |
| **01** | Semaphore is one of the concurrency mechanisms available. Find out about more concurrency mechanisms. How do these mechanisms protect critical sections? Compare their implementations with *wait()* and *signal()* operations of semaphores. |
| **02** | Implement the algorithm of Producer-Consumer problem given above, in C language. |
|  |  |
|  |  |

Submitted On

01-06-2023

(Date: DD/MM/YY)

**Task 01:** Semaphore is one of the concurrency mechanisms available. Find out about more concurrency mechanisms. How do these mechanisms protect critical sections? Compare their implementations with *wait()* and *signal()* operations of semaphores.

**Solution:**

Semaphore is indeed one of the concurrency mechanisms used to protect critical sections in concurrent programming. Let me provide you with information about some other concurrency mechanisms and compare their implementations with the wait() and signal() operations of semaphores.

**Comparing with Semaphores:**

Semaphore wait() and signal() operations are general-purpose synchronization primitives that can be used to implement various concurrency mechanisms. Here's a comparison with the mechanisms mentioned above:

**Semaphores vs. Mutex Locks:**

Semaphores are more flexible than mutex locks as they can allow multiple threads to access a resource concurrently, whereas mutex locks provide exclusive access. Mutex locks are simpler and typically more efficient when exclusive access is sufficient.

**Semaphores vs. Read-Write Locks:**

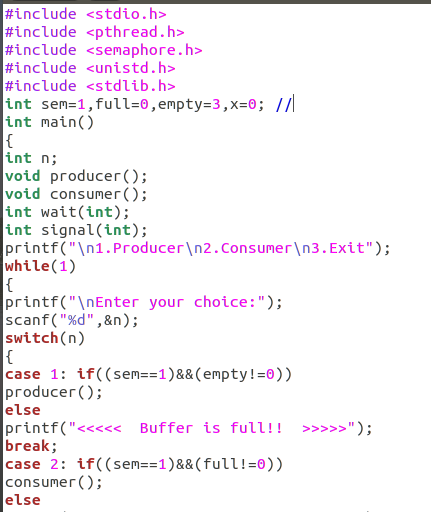
Semaphores can be used to implement read-write locks by assigning a count of 1 for the write operation and allowing multiple counts for read operations. However, read-write locks provide a higher level of abstraction specifically designed for scenarios where reads are more frequent than writes.

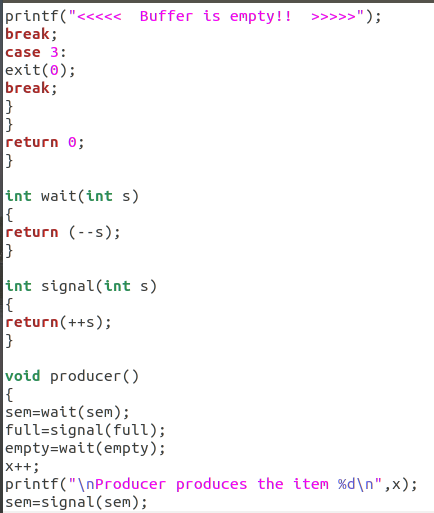
**Semaphores vs. Condition Variables:**

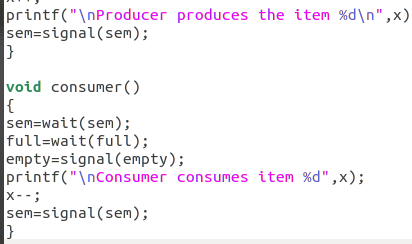
Semaphores and condition variables have different purposes. Semaphores are primarily used for signaling and synchronization between multiple threads, while condition variables are used for thread coordination based on specific conditions. Condition variables are often used in conjunction with mutex locks to protect shared data and synchronize thread activities.

In summary, semaphores provide a general-purpose mechanism for synchronization and signaling, while mutex locks, read-write locks, and condition variables offer more specialized concurrency mechanisms tailored for specific scenarios.

**Task 02:** Implement the algorithm of Producer-Consumer problem given above, in C language.

**Solution:**





**Output:**

