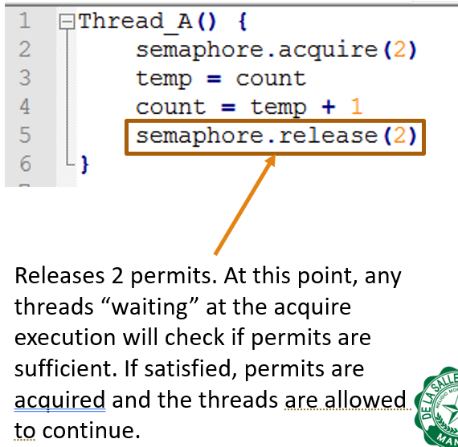
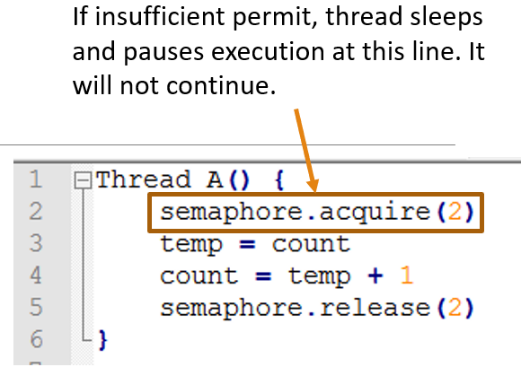
|  |  |
| --- | --- |
| **GDPARCM Lecture – Semaphores** | Instructor: Neil Patrick Del Gallego |

**Semaphores**

* Data structure that holds N number of permits. Two functions: **acquire(S)** where **S <= N**. and **release(S)**.



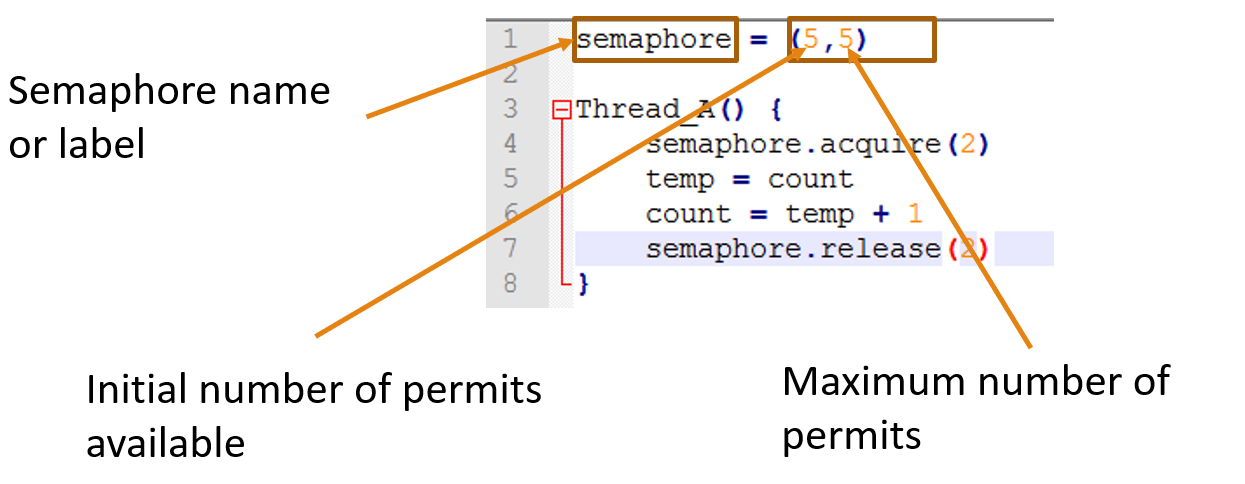
Acquire(S)

* Attempts to obtain S permits. If insufficient permits are available, thread pauses and sleeps at this line.
* Otherwise, thread proceeds as is and acquires the permits.
* Note that this process is atomic. There is no race condition.

Release(S)

* Releases S amount of permits. Allowing other threads asking for permits to continue execution.

Declaring a semaphore



**Sample Problem: Mutual Exclusion Using Semaphores**

Only 1 semaphore is needed.

|  |  |
| --- | --- |
|  | SOLUTION: |

**Sample Problem: Multiplex**

Using the solution in problem #1, allow K threads to enter at the critical section at any given point. Assume there are N threads running. K <= N. For this problem, let **N = 10** and **K = 5**.

|  |  |
| --- | --- |
|  | SOLUTION: |

**Sample Problem: Simple Thread Barrier**

Synchronize the threads to satisfy the following relation: a1 < b2 AND b1 < a2. a2 should wait for b1 to execute and b2 should wait for a1 to execute. This problem is called a **thread barrier,** where both threads must meet at a common checkpoint before a2 and b2 executes.

|  |  |
| --- | --- |
|  |  |

**Programming Discussion**

Discuss semaphore implementation through C++ mutexes – GDPARCM\_HO4 example.

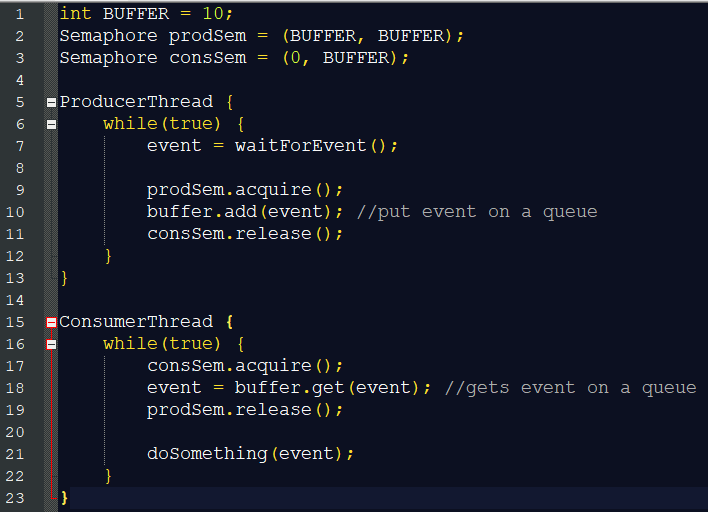
**Activities:** Use semaphores to solve the following problems.

1. Consider the **Producer-Consumer** problem where Producer thread waits for an event. When an event is detected, it is added into a buffer of MAX SIZE of 10. Consumer thread retrieves the latest event from the buffer and performs a task related to it. Assume that only 1 producer and 1 consumer thread is active.

|  |  |
| --- | --- |
|  | |
|  |  |

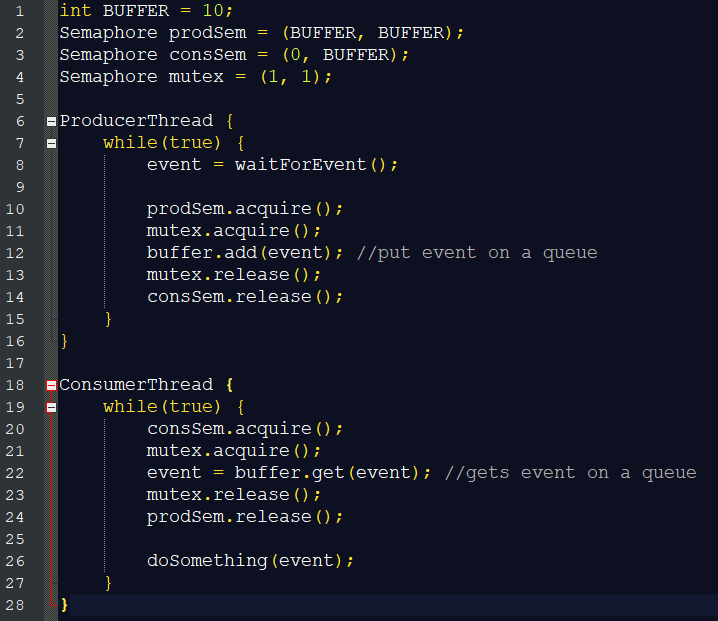
Synchronize both threads using semaphores such that **waitForEvent()** and **doSomething()** can happen concurrently. However, only 1 thread gets to access the buffer (line P6 and C12). Show the pseudocode. TIP: You will need to use two semaphores and use BUFFER\_SIZE as permit count. (10 pts)

**SOLUTION:**

****

1. Show the pseudocode using semaphores if there will be **multiple** producer and consumer threads active. TIP: You will need to introduce another semaphore (number of permits you must determine). Hence, 3 semaphores are required. (10 pts)

NOTE: Using the above solution, this can still result in a race condition. Multiple producer threads acquiring the prodSem can access the same buffer. Must use mutex semaphore.

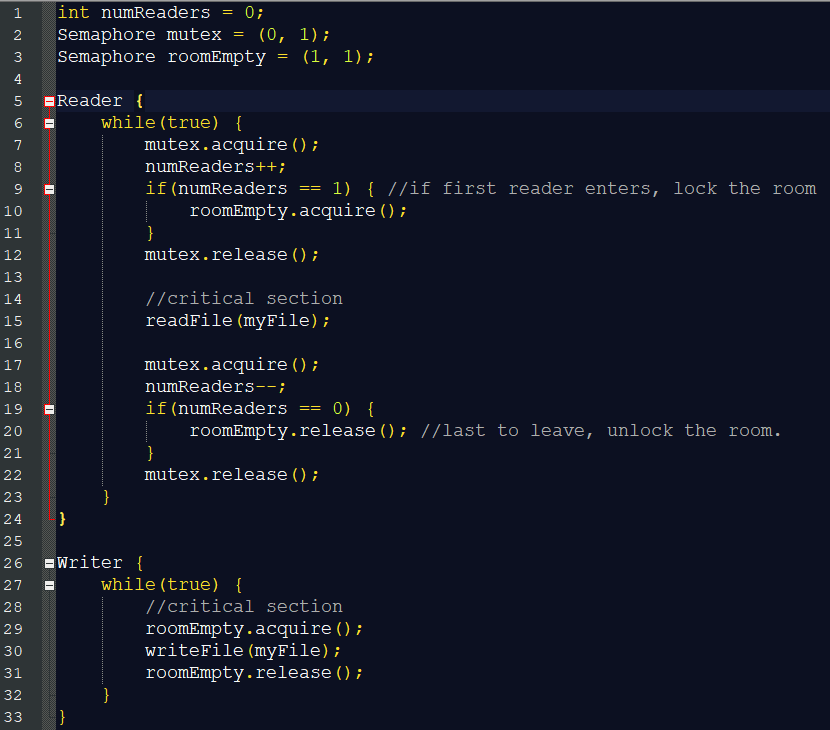


1. The **Readers-Writers** problem refers to events wherein multiple threads attempt to read or modify a shared data structure, file system, or database. While the data structure is being written or being modified, it is often necessary to bar other threads from reading, to prevent a reader from interrupting a modification in progress and reading inconsistent or invalid data. There are two types of threads, **readers,** and **writers** where they execute different lines of code before entering the critical section. Create a solution for the **Readers-Writers** problem that imposes the following synchronization rules:

* Any number of **readers** can be inside the critical section if there are no **writers**.
* **Writers** get exclusive access to the critical section.
* A **writer** cannot enter the critical section if there are **readers (or another writer)** inside the critical section.
* Only 1 **writer** gets to access the critical section. No other threads (**readers or another writer)** may enter.

|  |  |
| --- | --- |
|  | |
|  |  |

SOLUTION:



When the writer exits, can it be sure that the room is now empty? Yes, because it knows that no other thread can have entered while it was there. The code for readers is like the barrier code we saw in the previous section. We keep track of the number of readers in the room so that we can give a special assignment to the first to arrive and the last to leave. The first reader that arrives must wait for **roomEmpty**. If the room is empty, then the reader proceeds and, at the same time, bars writers. Subsequent readers

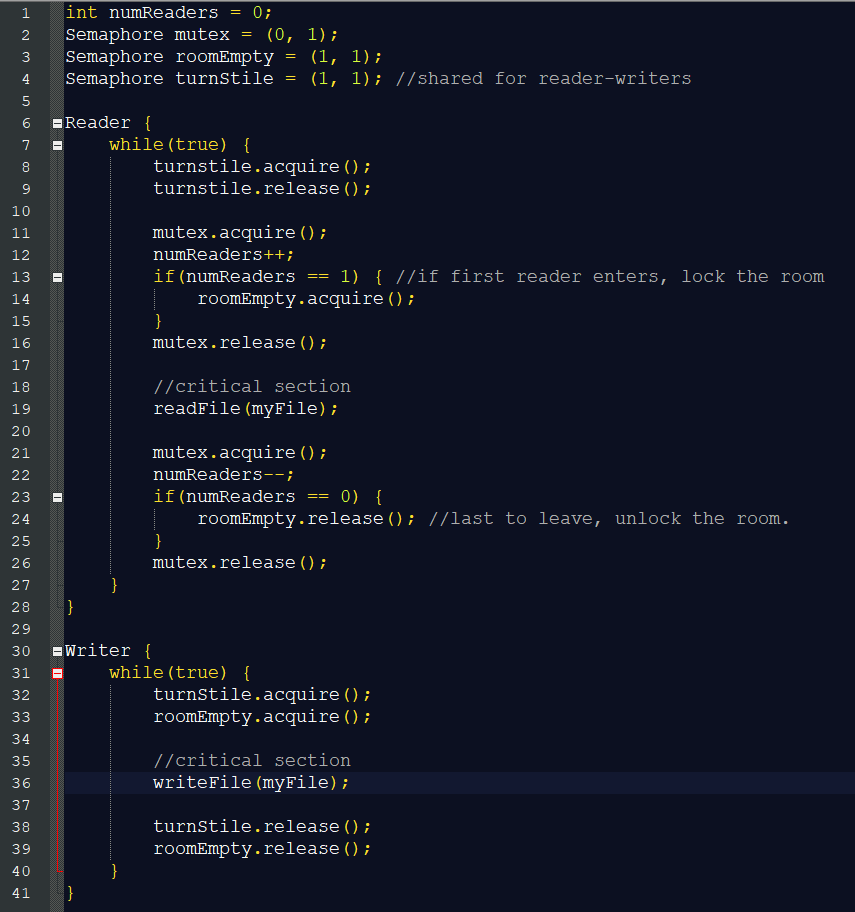
can still enter because none of them will try to wait on **roomEmpty**. If a reader arrives while there is a writer in the room, it waits on **roomEmpty**. Since it holds the mutex, any subsequent readers queue on mutex.

The code after the critical section is similar. The last reader to leave the room turns out the lights—that is, it signals **roomEmpty**, possibly allowing a waiting writer to enter.

1. No deadlock can occur in the previous solution for the **Reader-Writers** problem. However, **starvation** can occur. Starvation occurs if a thread is paused indefinitely never getting the chance to execute. In the above solution, the writer can starve. If a writer arrives while there are readers in the critical section, it might wait in queue forever while readers come and go. If a new reader arrives before the last of the current readers departs, there will always be at least one reader in the room.

Extend the solution above for the **No-Starve Writer**.

SOLUTION:



If a writer arrives while there are readers in the room, it will block at Line 8, which means that the turnstile will be locked. This will bar readers from entering while a writer is queued. When the last reader leaves, it signals **roomEmpty**, unblocking the waiting writer. The writer immediately enters its critical section since none of the waiting readers can pass the turnstile. When the writer exits it signals turnstile, which unblocks a waiting thread, which could be a reader or a writer. Thus, this solution guarantees that at least one writer gets to proceed, but it is still possible for a reader to enter while there are writers queued.

**Programming Discussion**

Discuss semaphore implementation through C++20 documentation. **IETSemaphore** in GDPARCM – HO4.

Discuss the light switch semaphore pattern. Lightswitch in GDPARCM – HO4.