**Fredrick Onduso Ondieki**

**CST-221**

**CST-221-Monitors and Semaphores**

**John Zupan**

**March 17, 2019**

**Scenario emphasizing the need for Synchronization**

**Semaphore**

Semaphores are data structures used for higher level synchronization. They are aimed to provide mutual exclusion to the critical sections indicated by the programmer. Semaphores are meant to accomplish two types of operations:

***Wait – Wait ()***

This operation will block a thread until an indicated condition is carried out. Essentially the thread will be allowed to continue only if the semaphore is open and it will be blocked on the queue if the semaphore is closed.

***Signal – Signal ()***

The signal operation will allow a thread to enter. When one thread is waiting in the queue, the signal () will unblock it by opening the semaphore. If there are no threads on queue, the signal will be remembered for the following thread, acting like history feature.

In addition, there are two types of semaphores; **the counting semaphore** and **the mutex semaphore**. The Mutex semaphore is the basic type of semaphore. It will provide single access to a certain resource. It will also assure mutual exclusion for a critical section. On the other side, the counting semaphore is able to allow access to multiple threads through the semaphore. It can be used to provide various types of simultaneous access that is not synchronized. This semaphore will also determine the number of threads that have passed it by using the “count” feature.

**Monitors**

A monitor is another type of data structure utilized for higher level synchronization. It is meant to control the access to shared data between users. It will also protect the data from any type of unstructured access because the information may only be accessed from within the monitor by using the provided procedures.

A monitor guarantees mutual exclusion. This means that one thread only may execute a monitor procedure at any given time. When the monitor procedure is invoked by a second thread, the second thread will be blocked.

**Essential differences between Monitors and Semaphores**

Although both semaphores and monitors are used to serve the same purpose (higher level synchronization), the are a few key differences between them that may condition certain users to only one of them. The table bellow summarizes the differences.

|  |  |  |
| --- | --- | --- |
| **#** | **Semaphores** | **Monitors** |
| 1 | Can be used anywhere in a program but should not be used in a monitor. | Monitors are based on conditional variables and can only be used in monitors |
| 2 | ***Wait()*** does not always block the caller that is when the semaphore counter is greater than zero. | ***Wait()*** always blocks the caller. It does not matter the condition of the counter variable. |
| 3 | ***Signal()*** gives either releases a blocked process for it to get a chance to execute. | ***Signal()*** either releases a blocked thread if there is one or the signal is lost as if it never happens. |
| 4 | If the Signal() releases a blocked thread, the caller and the released thread both continue. | If a ***Signal()*** releases a blocked thread, the caller yields the monitor or continues based on condition, only one of the caller or the released thread can continue but not both. |

References

Monitor Vs Semaphore – What Are the Main Differences? (n.d.). Retrieved March 17, 2019, from https://itinterviewguide.com/monitor-vs-semaphore/