LAB # 09
Inter Process Communication I

Semaphores

Semaphores are a good way to learn about synchronization, but they are not as widely used in practice. Nevertheless, there are some synchronization problems that can be solved simply with semaphores, yielding solutions that are more demonstrably correct.

A semaphore S is an integer variable that, apart from initialization, is accessed only through two standard atomic operations: wait() and signal(). All modifications to the integer value of the semaphore in the wait() and signal() operations must be executed indivisibly. That is, when one process modifies the semaphore value, no other process can simultaneously modify that same semaphore value.

The semaphore is initialized to the number of resources available. Each process that wishes to use a resource performs a wait() operation on the semaphore (thereby decrementing the count). When a process releases a resource, it performs a signal() operation (incrementing the count). When the count for the semaphore goes to 0, all resources are being used. After that, processes that wish to use a resource will block until the count becomes greater than 0.

```
struct semaphore {
      int count;
      queueType queue;
};

void semWait(semaphore s)
{
      s.count--;
      if (s.count < 0) {
            /* place this process in s.queue */;
            /* block this process */;
      }
}

void semSignal(semaphore s)
{
      s.count++;
      if (s.count<= 0) {
            /* remove a process P from s.queue */;
            /* place process P on ready list */;
      }
}
```

Each semaphore has an integer value and a list of processes list. When a process must wait on a semaphore, it is added to the list of processes. A signal() operation removes one process from the list of waiting processes and awakens that process.

Note that in this implementation, semaphore values may be negative, whereas semaphore values are never negative under the classical definition of semaphores with busy waiting. If a semaphore value is negative, its magnitude is the number of processes waiting on that semaphore. It is critical that semaphore operations be executed atomically. We must guarantee that no two processes can execute wait() and signal() operations on the same semaphore at the same time. This is a critical-section problem.

Producer-Consumer Problem

Producer-consumer problem, is a common paradigm for cooperating processes. A producer process produces information that is consumed by a consumer process. One solution to the producer-consumer problem uses shared memory. To allow producer and consumer processes to run concurrently, there must be available a buffer of items that can be filled by the producer and emptied by the consumer. This buffer will reside in a region of memory that is shared by the producer and consumer processes. A producer can produce one item while the consumer is consuming another item. The producer and consumer must be synchronized, so that the consumer does not try to consume an item that has not yet been produced.

Algorithm

1. Set a buffer size. The Semaphore *lock* is initialized. *full* & *empty* are initialized as well.
2. In the case of *producer* process
    i) Produce an item into temporary variable.
    ii) If there is empty space in the buffer check the *lock* value for entering into the critical section.
    iii) If the *lock* value is 0, allow the *producer* to add value in the temporary variable to the buffer.
3. In the case of *consumer* process
    i) It should wait if the buffer is *empty*.
    ii) If there is any item in the buffer check for *lock* value, if the *lock* ==0, remove item from buffer.
    iii) Signal the *lock* value and reduce the *empty* value by 1.
    iv) Consume the item.
4. Print the result.

Time Boxing

Activity Name	Activity Time	Total Time
Login Systems + Setting up Linux Environment	3 mints + 7 mints	10 mints
Walk through Theory & Tasks	60 mints	60 mints
Implement Tasks	80 mints	80 mints
Evaluation Time	30 mints	30 mints
	Total Duration	180 mints

Objectives/Outcomes

- To learn a inter process communication with Semaphore.

Lab Tasks/Practical Work

1) 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.

2) Implement the algorithm of Producer-Consumer problem given above, in C language.