OS2 project documentation

(Bound-Buffer Problem).

Definition of Producer and Consumer:

Two or more processes share a common buffer or queue. The producer continuously produces certain data and pushes it onto the buffer, whereas the consumer consumes those data from the buffer.

The Problem: A producer tries to insert data into an empty slot of the buffer. A consumer tries to remove data from a filled slot in the buffer. As you might have guessed by now, those two processes won't produce the expected output if they are being executed concurrently.
There needs to be a way to make the producer and consumer work in an independent manner.

Solution

One solution of this problem is to use semaphores. The semaphores which will be used here are:

• M, a binary semaphore which is used to acquire and release the lock.

• Empty, a counting semaphore whose initial value is the number of slots in the buffer, since, initially all slots are empty
.
• Full, a counting semaphore whose initial value is 0.

the current value of empty represents the number of empty slots in the buffer and full represents the number of occupied slots in the buffer.

1. Solution pseudocode Producer

```
do
{
   // wait until empty > 0 and then decrement 'empty'
   wait(empty);
   // acquire lock
   wait(mutex);

   /* perform the insert operation in a slot */

   // release lock
   signal(mutex);
   // increment 'full'
   signal(full);
}
while(TRUE):
```

Looking at the above code for a producer, we can see that a producer first waits until there is at least one empty slot.

Then it decrements the empty semaphore because, there will now be one less empty slot, since the producer is going to insert data in one of those slots.
Then, it acquires lock on the buffer, so that the consumer cannot access the buffer until producer completes its operation.

After performing the insert operation, the lock is released and the value of full is incremented because the producer has just filled a slot in the buffer.

Solution pseudocode Consumer

```
do
{
   // wait until full > 0 and then decrement 'full'
   wait(full);
   // acquire the lock
   wait(mutex);

   /* perform the remove operation in a slot */
   // release the lock
   signal(mutex);
   // increment 'empty'
   signal(empty);
}
   while(TRUE);
```

The consumer waits until there is at least one full slot in the buffer.

Then it decrements the full semaphore because the number of occupied slots will be decreased by one, after the consumer completes its operation.

After that, the consumer acquires lock on the buffer.
Following that, the consumer completes the removal operation so that the data from one of the full slots is removed.

Then, the consumer releases the lock.
Finally, the empty semaphore is incremented by 1, because the consumer has just removed data from an occupied slot, thus making it empty.

Examples of Deadlock

A deadlock occurs when two or more threads wait forever for a lock or resource held by another of the threads.

Example:

The most easily recognizable form of deadlock is traffic gridlock. Multiple lines of cars are all vying for space on the road and the chance to get through an intersection, but it has become so backed up there this no free space for potentially blocks around. This causes entire intersections or even multiple intersections to go into a complete standstill. Traffic can only flow in a single direction, meaning that there is nowhere for traffic to go once traffic has stopped. However, if the car at the very end of each line of traffic decide to back up, this frees up room for other cars to do the same and therefore the gridlock is solved.

Solution

By Semaphore (notifyAll(), wait):

This solution makes the process p1 wait until process p2 finish its critical section and notify p1 that it finished so that the p1 enters its critical section and vice versa.

Example:

If the car at the very end of each line of traffic decide to back up, this frees up room for other cars to do the same and therefore the gridlock is solved.

Examples of starvation

Starvation is a problem that is closely related to both, Livelock and Deadlock. Occur when a low priority program is requesting for a system resource, but aren't able to execute because high priority processes keep executing and low priority processes get blocked for indefinite time. In heavily loaded computer system, a steady stream of higher-priority processes can prevent a low-priority process from ever getting the CPU. In starvation resources are continuously utilized by high priority processes.
Problem of starvation can be resolved using Aging. In Aging priority of long waiting processes is gradually increased.

Example:

Person have two laptop, one with high specification and other with poor specification the person open the laptop with high specification every day and leave , whose poor specification the bad laptop with lead time will slowness and become out of work.

How did solve starvation:

By Semaphore Threads waiting on an object (called wait() on it) remain waiting indefinitely

If correctly coded producer-consumer problem is always going to be starvation free.
Consumer blocking till there is ready buffer to consume.
Producer blocking till there is empty buffer to produce.
Producer after producing immediately moving buffer to ready buffer queue.
Consumer after consuming immediately moving buffer to empty buffer queue.
This arrangement can not lead to starvation.

Solve of example

The person must open the poor laptop at least one hour a day to prevent from slow and spoiled.

Explanation for real world application and how did apply the problem

parking lot contain cars that carries customer to go work ,the cars leaves the parking lot when the parking seats are completed and after that he will leave the parking lot ,then the other one car comes in its place without stopping the transporter process and the speed of transporting people as soon as possible , simply accessing the semaphore directly.