Report on

Bounded-buffer problem

Department of Information Technology

by

Aritra Ghosal 12200220009

5th Semester

Course Code:PCC-CS502

Course name :Operating System

St. Thomas' College of Engineering and Technology, Kolkata

Affiliated to

Maulana Abul Kalam Azad University of Technology, West Bengal

Session: 2022-2023

Abstract

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.

In Bounded Buffer Problem there are three entities storage buffer slots, consumer and producer. The producer tries to store data in the storage slots while the consumer tries to remove the data from the buffer storage.

It is one of the most important process synchronizing problem let us understand more about the same.

INTRODUCTION:

In the bounded buffer problem, there is a buffer of n slots, and each slot is capable of storing one unit of data. Producer and Consumer are the two processes operating on the Buffer.

The producer tries to enter data in an empty slot, and the consumer tries to remove it. When the processes are run simultaneously, they will not give the expected results.

There is a need for the producer and the consumer to run independently.

One solution to the Bounded Buffer problem is to use Semaphores.

The Semaphores that are used are as follows:

- m, a Binary Semaphore.
- empty, a Counting Semaphore.
- full, a Counting Semaphore

DETAILED DESCRIPTION:

Although this is a very popular problem that is available in virtually every textbook, let us revisit it! First, because the buffer is shared by all threads, they have to be protected so that race condition will not occur. So, this requires a mutex lock or a binary semaphore. A producer cannot deposit its data if the buffer is full. Similarly, a consumer cannot retrieve any data if the buffer is empty. On the other hand, if the buffer is not full, a producer can deposit its data. After this, the buffer contains data, and, as a result, a consumer should be allowed to retrieve a data item. Similarly, after a consumer retrieves a data item, the buffer is not full, and a producer should be allowed to deposit its data.

Putting these observations together, we know that

A producer must wait until the buffer is not full, deposit its data, and then notify the consumers that the buffer is not empty.

A consumer, on the other hand, must wait until the buffer is not empty, retrieve a data item, and then notify the producers that the buffer is not full.

Of course, before a producer or a consumer can have access to the buffer, it must lock the buffer. After a producer and consumer finishes using the buffer, it must unlock the buffer

The pseudocode of the producer function looks like this:

```
do {
  wait(empty); // wait until empty>0 and then decrement 'empty'
  wait(mutex); // acquire lock
  /* perform the
  insert operation in a slot */
  signal(mutex); // release lock
  signal(full);  // increment 'full'
} while(TRUE)
```

Looking at the above code for a producer, we can see that a producer first waits until there is atleast 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.

Consumer Operation:

The pseudocode of the consumer function looks like this:

```
do {
  wait(full); // wait until full>0 and then decrement 'full'
  wait(mutex);  // acquire the lock
  /* perform the remove operation
   in a slot */
  signal(mutex); // release the lock
  signal(empty); // increment 'empty'
} while(TRUE);
```

The consumer waits until there is atleast 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.

In this program, we insist that the number of producer and the number of consumers are equal. Since one consumer retrieves exactly one END, when the last consumer retrieves END it is the

last one and the buffer has no data. As a result, the system ends properly. What if the number of producers and consumers are not equal? Will you be able to make sure the system will end properly?

In the above program, when a producer or a consumer is accessing the buffer, all other producers and consumers are blocked to achieve mutual exclusion. Is it possible that two binary semaphores can be used, one for blocking producers and the other for blocking consumers, so that one producer and one consumer can have access to the buffer at the same time? In this way, our program would be more efficient. Please think about this suggestion to see if it works.

CONCLUTION:

The bounded-buffer problems (aka the producer-consumer problem) is a classic example of concurrent access to a shared resource. A bounded buffer lets multiple producers and multiple consumers share a single buffer. Producers write data to the buffer and consumers read data from the buffer.

This problem is generalized in terms of the Producer-Consumer problem. Solution to this problem is, creating two counting semaphores "full" and "empty" to keep track of the current number of full and empty buffers respectively.

The mutex semaphore ensures mutual exclusion. The empty and full semaphores count the number of empty and full spaces in the buffer. After the item is produced, wait operation is carried out on empty. This indicates that the empty space in the buffer has decreased by 1.

References:

www.it.uu.ee

www.scanftree.com

www.prepinsta.com

www.pagesw.mtu.edu

APPENDIX:

Producer buffer solution:

```
do
{
   // wait until empty > 0 and then decrement 'empty'
   // that is there must be atleast 1 empty slot
   wait(empty);
   // acquire the lock, so consumer can't enter
   wait(mutex);

   /* perform the insert operation in a slot */

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