**UNIVERSITY OF ENGINEERING AND TECHNOLOGY, TAXILA**

**FACULTY OF TELECOMMUNICATION AND INFORMATION ENGINEERING**

**COMPUTER ENGINEERING DEPARTMENT**

**PRODUCER-CONSUMER PROBLEM USING SEMOPHERES**

**Lab 15**

**OS**

**CPED**

**Lab objective:**

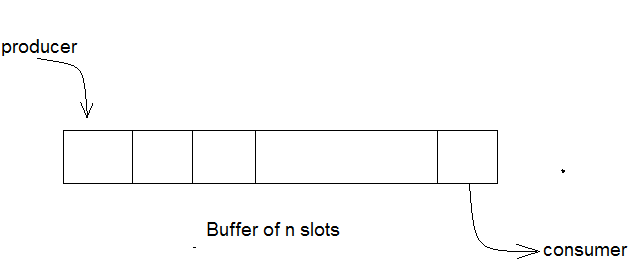
To implement producer/consumer problem using semaphore.

**Bounded Buffer Problem**

Bounded buffer problem, which is also called producer consumer problem, is one of the classic problems of synchronization. Let's start by understanding the problem here, before moving on to the solution and program code.

**What is the Problem Statement?**

There is a buffer of n slots and each slot is capable of storing one unit of data. There are two processes running, namely, producer and consumer, which are operating on the buffer.



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.

**Here's a 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.

At any instant, 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.

**The Producer Operation**

The pseudocode of the producer function looks like this:

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 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.

**The Consumer Operation**

The pseudocode for the consumer function looks like this:

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 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.

**ALGORITHM:**

1. Declare variable for producer & consumer as pthread-t-tid produce tid consume.

2. Declare a structure to add items, semaphore variable set as struct.

3. Read number the items to be produced and consumed.

4. Declare and define semaphore function for creation and destroy.

5. Define producer function.

6. Define consumer function.

7. Call producer and consumer.

8. Stop the execution.CS6413 OPERATING SYSTEM LAB

**PROGRAM: (PRODUCER-CONSUMER PROBLEM)**

#include<stdio.h>

void main()

{

int buffer[10], bufsize, in, out, produce, consume, choice=0;

in = 0;

out = 0;

bufsize = 10;

while(choice !=3)

{

printf("\n1. Produce \t 2. Consume \t3. Exit");

printf("\nEnter your choice: =");

scanf("%d", &choice);

switch(choice)

{

case 1: if((in+1)%bufsize==out)

printf("\nBuffer is Full");

else

{

printf("\nEnter the value: ");

scanf("%d", &produce);

buffer[in] = produce;

in = (in+1)%bufsize;

}

break;

case 2: if(in == out)

printf("\nBuffer is Empty");

else

{CS6413 OPERATING SYSTEM LAB

consume = buffer[out];

printf("\nThe consumed value is %d", consume);

out = (out+1)%bufsize;

}

break;

} } }

**OUTPUT:**

1. Produce 2. Consume 3. Exit

Enter your choice: 2

Buffer is Empty

1. Produce 2. Consume 3. Exit

Enter your choice: 1

Enter the value: 100

1. Produce 2. Consume 3. Exit

Enter your choice: 2

The consumed value is 100

1. Produce 2. Consume 3. Exit

Enter your choice: 3

RESULT:

Thus the producer consumer program was executed and verified successfully