EXP5 - Producer Consumer Problem in C

Theory:-

The producer-consumer problem is an example of a multi-process synchronization problem. The problem describes two processes, the producer and the consumer that share a common fixed-size buffer and use it as a queue.

- The producer's job is to generate data, put it into the buffer, and start again.
- At the same time, the consumer is consuming the data (i.e., removing it from the buffer), one piece at a time.

What is the Actual Problem?

Given the common fixed-size buffer, the task is to make sure that the producer can't add data into the buffer when it is full and the consumer can't remove data from an empty buffer. Accessing memory buffers should not be allowed to producers and consumers at the same time.

Solution of Producer-Consumer Problem

The producer is to either go to sleep or discard data if the buffer is full. The next time the consumer removes an item from the buffer, it notifies the producer, who starts to fill the buffer again. In the same manner, the consumer can go to sleep if it finds the buffer to be empty. The next time the producer transfer data into the buffer, it wakes up the sleeping consumer.

CODE:

```
#include &lt;stdio.h&gt;
#include &lt;stdlib.h&gt;
int mutex = 1;
int full = 0;
int empty = 10, x = 0;
void producer()
{
--mutex;
++full;
--empty;
x++;
printf("\nProducer produces""item %d",x);
++mutex;
}
void consumer()
{
--mutex;
--full;'
```

```
++empty;
printf("\nConsumer consumes""item %d",x);
x--;
}
int main()
{
int n, i;
printf("\n1. Press 1 for Producer""\n2Press 2 for Consumer""t\n3. Press 3 for Exit"");
for (i = 1; i &gt; 0; i++) {
        printf(&quot;\nEnter your choice:&quot;);
        scanf(&quot;%d&quot;, &amp;n);
        switch (n) {
        case 1: if ((mutex == 1)&amp;&amp; (empty != 0)) {
        producer();
}else {
printf(&quot;Buffer is full!&quot;);
}
break;
case 2: if ((mutex == 1)&amp;&amp; (full != 0)){
consumer();
}else {
printf(&quot;Buffer is empty!&quot;);
}
break;
case 3:
exit(0);
break;
}}}
```

Output:-

```
1.Producer
2.Consumer
3.Exit
Enter your choice:2
Buffer is empty!!
Enter your choice:1

Producer produces item 1
Enter your choice:1

Producer produces item 2
Enter your choice:1

Producer produces item 3
Enter your choice:2

Consumer consumes item 3
Enter your choice:1

Producer produces item 3
Enter your choice:2

Consumer consumes item 3
Enter your choice:2

Consumer consumes item 2
Enter your choice:2

Consumer consumes item 1
Enter your choice:2
Buffer is empty!!
Enter your choice:3
```

Learning Outcomes: Upon completing this experiment, students will be able to:
LO1: Define Synchronization Concepts
LO2: Identify and Implement Constraints in Multi-threaded Programs
LO3: Apply Synchronization Mechanisms for Process Control

Course Outcomes: Upon completion of the course, students will be able to:

1. Solve and build basic multi-threaded programs and implement synchronization mechanisms like mutexes and semaphores in C programming.
2. Understand the concepts of process synchronization and concurrency control while working with shared resources in a system.
3. Implement and solve real-world synchronization problems such as the Producer-Consumer problem by designing efficient algorithms that handle multiple processes simultaneously.

Conclusion:

In conclusion, the Producer-Consumer problem demonstrates the challenges of synchronization and managing access to shared resources between multiple processes. Through the use of mutex locks, the producer and consumer processes are synchronized to prevent data inconsistencies.

This experiment allowed students to understand the core concepts of inter-process communication and synchronization mechanisms. By implementing these solutions in C, students gain hands-on experience with practical challenges that arise in multi-process systems, preparing them for more complex system-level programming and problem-solving in real-world scenarios.

For Faculty Use

Correction Parameters	Formative Assessment [40%]	Timely completion of Practical [40%]	Attendance / Learning Attitude [20%]	