Bahria University,

Karachi Campus



LAB EXPERIMENT NO.

**10**

LIST OF TASKS

|  |  |
| --- | --- |
| TASK NO | OBJECTIVE |
| 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.** |
|  |  |
|  |  |

Submitted On:

31/05/2023

(Date: DD/MM/YY)

**Task No. 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.

**Solution:**

A semaphore does the allows multiple number of threads to access a particular resource until and unless there is a need of mutual exclusion.

A lock allows only one thread to enter the part that's locked and the lock is not shared with any other processes.

A mutex is the same as a lock but it can be shared by multiple processes. In mutex, acquire will let first caller through, and then block next until release

In condition synchronization, acquiring will block first caller until release.

The wait() and signal() operations of semaphores are used to acquire and release permits from the semaphore. When a thread calls wait(), it blocks until a permit becomes available. When a thread calls signal(), it releases a permit, which can then be acquired by another thread that is waiting on the semaphore.

The wait() and signal() operations can be used to implement other concurrency mechanisms, such as mutexes and condition variables. For example, a mutex can be implemented using a semaphore with a value of 1. When a thread wants to acquire the mutex, it calls wait(). This blocks the thread until the semaphore's value is 0. When a thread releases the mutex, it calls signal(). This increments the semaphore's value, which can then be used to unblock another thread that is waiting on the mutex.

**Task No. 2:** Implement the algorithm of Producer-Consumer problem given above, in C language.

**Solution:**

#include <stdio.h>

#include <stdlib.h>

int mutex = 1;

int full = 0;

int empty = 10;

int x = 0;

void producer() {

--mutex;

++full;

--empty;

x++;

printf("\nPRODUCER PRODUCED ITEM: %d", x);

++mutex;

}

void consumer() {

--mutex;

--full;

++empty;

printf("\nCONSUMER CONSUMED ITEM: %d", x);

x--;

++mutex;

}

int main() {

int n, i;

printf("1. PRESS 1 FOR PRODUCER\n");

printf("2. PRESS 2 FOR CONSUMER\n");

printf("3. PRESS 3 FOR EXIT\n");

for (i = 1; i > 0; i++) {

printf("\nENTER YOUR CHOICE: ");

scanf("%d", &n);

switch (n) {

case 1:

if (mutex == 1 && empty != 0) {

producer();

} else {

printf("BUFFER IS FULL!\n");

}

break;

case 2:

if (mutex == 1 && full != 0) {

consumer();

} else {

printf("BUFFER IS EMPTY!\n");

}

break;

case 3:

exit(0);

break;

}

}

return 0;

}

**Output:**

