Kaushal

18ETCS002147

Laboratory 5

Title of the Laboratory Exercise: Solution to Producer Consumer Problem using Semaphore and Mutex

1. Introduction and Purpose of Experiment

In multitasking systems, simultaneous use of critical section by multiple processes leads to data inconsistency and several other concurrency issues. By solving this problem students will be able to use Semaphore and Mutex for synchronisation purpose in concurrent programs. 2. Aim and Objectives

Aim

- To implement producer consumer problem using Semaphore and Mutex

Objectives

At the end of this lab, the student will be able to

- Use semaphore and Mutex
- Apply semaphore and Mutex in the required context
- Develop multithreaded programs with Semaphores and Mutex

3. Experimental Procedure

i. Analyse the problem statement ii. Design an algorithm for the given problem statement and develop a flowchart/pseudo-code iii. Implement the algorithm in C language iv. Compile the C program

v. Test the implemented program vi. Document the Results vii. Analyse and discuss the outcomes of your experiment

4. Questions

Implement producer consumer problem by using the following

a) Semaphore

b) Mutex

5. Calculations/Computations/Algorithms:

To solve this problem, we need two counting semaphores – Full and Empty. “Full” keeps track of number of items in the buffer at any given time and “Empty” keeps track of number of unoccupied slots.

Initialization of semaphores –

```
mutex = 1
Full = 0 // Initially, all slots are empty. Thus full slots are 0
Empty = n // All slots are empty initially
```

Solution for Producer –

```
do{

//produce an item

wait(empty);
wait(mutex);

//place in buffer

signal(mutex);
signal(full);

}while(true)
```

When producer produces an item then the value of “empty” is reduced by 1 because one slot will be filled now. The value of mutex is also reduced to prevent consumer to access the buffer. Now, the producer has placed the item and thus the value of “full” is increased by 1. The value of mutex is also increased by 1 beacuse the task of producer has been completed and consumer can access the buffer.

Solution for Producer –

```
do{

//produce an item

wait(empty);
wait(mutex);

//place in buffer

signal(mutex);
signal(full);

}while(true)
```

As the consumer is removing an item from buffer, therefore the value of “full” is reduced by 1 and the value is mutex is also reduced so that the producer cannot access the buffer at this moment. Now, the consumer has consumed the item, thus increasing the value of “empty” by 1. The value of mutex is also increased so that producer can access the buffer now.

6. Presentation of Results

Producer-Consumer Using Semaphores:-

```c
#include <stdio.h>
#include <stdlib.h>
#include <semaphore.h>
#include <time.h>
#include <pthread.h>

#define N 10
sem_t  bin_sem,  full,  empty; int front = 0;
int rear = 0; int CQueue[N];

int produce_item()
{
    int p = 1 + rand() % 300; return p;
}

void  insert_item(int  item)
{
    if (rear == N) rear = 0;
    CQueue[rear++]  =  item;
    printf("The  Inserted  Item  is  %d\n",  item);
}

int remove_item()
{
    if (front == N) front = 0;
    int e = CQueue[front++]; return e;
}

void  consume_item(int  item)
{
    printf("The  Consumed  Item  is  %d\n",  item);
}

void *producer(void *a)
{
int item, i;
    for (i = 0; i < N; i++)
    {
        item = produce_item();
        sem_wait(&empty);
        sem_wait(&bin_sem);
        insert_item(item);
        sem_post(&bin_sem);
        sem_post(&full);
    }
}
```

```
void *consumer(void *a)
{
int item, i;
    for (i = 0; i < N; i++)
    {
    sem_wait(&full);
    sem_wait(&bin_sem);
    //  removing  item:  critical  region
    item = remove_item();
    consume_item(item);
    sem_post(&bin_sem);
    sem_post(&empty);
    }
}

int main(int argc, char **argv)
{
    //  initializing  semaphores
    sem_init(&bin_sem, 0, 1);
    sem_init(&full, 0, 0);
    sem_init(&empty, 0, N);

    //initializing pthread_t th1, th2; int a;
    pthread_t th1, th2; int a;
    // creating threads
    pthread_create(&th1, NULL, producer, NULL);
    pthread_create(&th2, NULL, consumer, NULL);
    pthread_join(th1, NULL);
    pthread_join(th2, NULL); return (EXIT_SUCCESS);
}
```

Output for producer-consumer using semaphore:-

Kaushal

18ETCS002147

```
PS F:\RUAS\5 sem\Operating system\Os lab\lab_6\PROGRAMS> cd "f:\RUAS\5 sem\Operating system\Os lab\lab_6\PROGRAMS\" ; if ($?) { gcc usingSemaphores.c -o usingSemaphores } ; if ($?) { .\usingSemaphores }
The  Inserted  Item  is  234
The  Inserted  Item  is  244
The  Consumed  Item  is  234
The  Inserted  Item  is  63
The  Consumed  Item  is  244
The  Inserted  Item  is  30
The  Consumed  Item  is  63
The  Inserted  Item  is  1
The  Consumed  Item  is  30
The  Inserted  Item  is  109
The  Consumed  Item  is  1
The  Inserted  Item  is  53
The  Consumed  Item  is  109
The  Inserted  Item  is  157
The  Consumed  Item  is  53
The  Inserted  Item  is  57
The  Consumed  Item  is  157
The  Inserted  Item  is  20
The  Consumed  Item  is  57
The  Consumed  Item  is  20
PS F:\RUAS\5 sem\Operating system\Os lab\lab_6\PROGRAMS>
```


Producer-consumer using mutex:-

```c
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#include <pthread.h>
#include <sys/unistd.h>

#define N 10

// defining mutexes
pthread_mutex_t mutex;
pthread_cond_t condc, condp;

int front = 0;
int rear = 0;
int CQueue[N];

int current_buffer_size = 0;
int p, sum = 0;

int produce_item()
{
    p = 1 + rand() % 40; return p;
}

void  insert_item(int  item)
{
    if (rear == N)
        rear = 0;
    CQueue[rear++]  =  item;
    current_buffer_size++;
    printf("The  Item  Inserted  is  %d\n",  item);
}
```

```
int remove_item(int item)
{
    if (front == N) front = 0;
    int e = CQueue[front++]; current_buffer_size--;
    printf("%d  was  removed  !\n\n",  item);
    return e;
}
void  consume_item(int  item)
{
    sum += item;
    printf("The Item %d is Consumed\n", item);
}
void *producer(void *a)
{
    int i, item;
    for (i = 0; i < N; i++)
    {
    // locks the critical region
    pthread_mutex_lock(&mutex); item = produce_item();
    while (current_buffer_size == N) pthread_cond_wait(&condp, &mutex);

    insert_item(item);
    pthread_cond_signal(&condc);
    pthread_mutex_unlock(&mutex);
    }
}
void *consumer(void *a)
{
    int i, item;
    for (i = 0; i < N; i++)
    {
        pthread_mutex_lock(&mutex);
        while (current_buffer_size == 0)
            pthread_cond_wait(&condc, &mutex);
        item = remove_item(item);
        consume_item(item);
        pthread_cond_signal(&condp);
        pthread_mutex_unlock(&mutex);
    }
}
int main(int argc, char **argv)
{

    pthread_mutex_init(&mutex, 0);
    pthread_cond_init(&condp, 0);
    pthread_cond_init(&condc, 0);

    pthread_t th1, th2; int a;

    pthread_create(&th1, NULL, producer, NULL); pthread_create(&th2, NULL, consumer, NU
LL);

    pthread_join(th1, NULL);
```

```
    pthread_join(th2, NULL);

    pthread_cond_destroy(&condc);
    pthread_cond_destroy(&condp);
    pthread_mutex_destroy(&mutex);
}
```

Output for producer-consumer using mutex:-

```
The Item Inserted is 24
The Item Inserted is 7
The Item Inserted is 18
The Item Inserted is 36
The Item Inserted is 34
The Item Inserted is 16
The Item Inserted is 27
The Item Inserted is 13
The Item Inserted is 10
The Item Inserted is 22
0 was removed !

The Item 24 is Consumed
24 was removed !

The Item 7 is Consumed
7 was removed !

The Item 18 is Consumed
18 was removed !

The Item 36 is Consumed
36 was removed !

The Item 34 is Consumed
34 was removed !

The Item 16 is Consumed
16 was removed !

The Item 27 is Consumed
27 was removed !

The Item 13 is Consumed
13 was removed !
```

```
The Item 10 is Consumed
10 was removed !

The Item 22 is Consumed
```

7. Analysis and Discussions:

Semaphores are integer variables that are used to solve the critical section problem by using two atomic operations, wait and signal that are used for process synchronization. The definitions of wait and signal are as follows − Wait. The wait operation decrements the value of its argument S, if it is positive.

Mutex is a mutual exclusion object that synchronizes access to a resource. It is created with a unique name at the start of a program. The Mutex is a locking mechanism that makes sure only one thread can acquire the Mutex at a time and enter the critical section.

8. Conclusions

The Producer-Consumer problem is a classic problem this is used for multi-process synchronization i.e. synchronization between more than one processes.

In the producer-consumer problem, there is one Producer that is producing something and there is one Consumer that is consuming the products produced by the Producer. The producers and consumers share the same memory buffer that is of fixed-size.

The job of the Producer is to generate the data, put it into the buffer, and again start generating data. While the job of the Consumer is to consume the data from the buffer.

9. Comments

1. Limitations of Experiments

The following are the problems that might occur in the Producer-Consumer: The producer should produce data only when the buffer is not full. If the buffer is full, then the producer shouldn't be allowed to put any data into the buffer. The consumer should consume data only when the buffer is not empty.

2. Limitations of Results

The code continues running until the user interfere with the execution yet the code can be adjusted such that it requests that user to enter the time/cycle it should run before it comes to end so user can pre decide the kind of output he/she needs as to interrupt on the program randomly.

3. Learning happened

We learned the working of producer consumer problem and its working also we learn how to code the following using c programming language and from doing that in-depth knowledge of the topic is gained

4. Recommendations

Utmost care has been taken while creating the document and the program in the given time constraint but there is always a chance of improvement and occurrence of error.

Kaushal

18ETCS002147

Kaushal

18ETCS002147