Name: K Srikanth Registration Number: 17ETCS002124

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

Algorithm: produce(thread_id)

1. start

2. generate random integer values between the range of 1 to 30.

3. While! pthread_equal((producers+i),self) and i < producer_count increment i*

End while loop.

4. Display item produced by producer.

5. Return item.

6. End function produce.

Algorithm:consume(item, thread_id)

1. start

2. generate random integer values between the range of 1 to 30.

3.While !pthread_equal((producers+i),self) and i < producer_count*

increment i

End while loop.

4. Print buffer elements.

5. Print item consumed by consumer and buffer length.

6. End function consume.

Name: K Srikanth Registration Number: 17ETCS002124

Algorithm: Producer ()

1. Start

2. While(True)

i. item = produce(pthread_self())

ii. decrement the number of empty slots

iii. lock critical section

iv. Increment buf_pos

v. put the item at address (buf+buf_pos)

vi. unlock critical section

vii. Increment the number of slots filled

viii. Delay for random number of seconds between 1 and 3.

3. End function producer.

Algorithm: Consumer ()

1. Start

2. While(True)

i. decrement the number of slots filled

ii. lock critical section

iii. copy value in (buf+buf_pos) to item

iv. decrement buf_pos

v. consume(c to pthread_self)

vi. unlock critical section

vii. Increment the number of slots empty

viii. Delay for random number of seconds between 1 and 3.

3. End function consumer.

Algorithm:Main()

Declare threads producer and consumer.

Declare semaphores mutex,empty,full.

Declare *buf,buf_pos=-1, producer_count, consumer_count, buf_len.

Declare i,error

1. Start.

2. Read number of producers and allocate memory using malloc.

3. Read number of consumers and allocate memory using malloc.

4. Read buffer capacity and allocate memory using malloc.

5. For i=0 to producer_count-1

1. Create thread producers+I and call function producer and store return value in error.

i. if(error)

1. then print "error"

ii. Else

1. print success

End for loop.

6. For i=0 to consumer_count-1

1. Create thread consumers+I and call function producer and store return value in error.

i. if(error)

1. then print "error"

ii. Else

1. print success

End for loop.

7. For i=0 to producer_count-1

1. Join thread consumers+I and call function producer and store return value in error.

i. if(error)

1. then print “error”

ii. Else

1. print success

End for loop.

8. For i=0 to consumer_count-1

1. Join thread consumers+I and call function producer and store return value in error.

i. if(error)

1. then print “error”

ii. Else

1. print success

End for loop.

9. Stop.

6. Presentation of Results

Code

```c
#include <stdio.h>
#include <pthread.h>
#include <stdlib.h>
#include <string.h>
#include <semaphore.h>
#include <unistd.h>
// K Srikanth 17ETCS002124
pthread_t *producers;
pthread_t *consumers;
sem_t mutex,empty,full;
int *buf,buf_pos=-1,producer_count,consumer_count,buf_len;
int item_count=-1,total_items,consumed_item_count=-1;
int produce(pthread_t self){
    int i =0;
    int item = 1 + rand()%total_items;
    while(!pthread_equal(*(producers+i),self) && i<producer_count){
        i++;
    }
    printf("Producer %d Produced Item %d \n",i+1,item);
    return item;
}
```

Figure 1 C Code for the given problem statement using semaphore and mutex

```c
void consume(int p,pthread_t self){
    int i =0,b;
    while(!pthread_equal(*(consumers+i),self) && i<consumer_count){
        i++;
    }
    b=i;
    printf("Buffer :");
    for ( i = 0; i <=buf_pos; ++i){
        printf("%d",*(buf+i));
    }
    printf("\n Consumer %d consumed item %d \n ",b+1,p);
    printf("Current buffer length is %d\n",buf_pos);
}
```

Figure 2 C Code for the given problem statement using semaphore and mutex continued

```c
void *producer (void*args){
    while (item_count<total_items){
        int item = produce(pthread_self());
        sem_wait(&empty);
        sem_wait(&mutex);
        buf_pos++;
        *(buf + buf_pos) = item;
        ++item_count;
        sem_post(&mutex);
        sem_post(&empty);
        sleep(1+rand()%3);
    }
    pthread_exit(NULL);
    return NULL;
}
```

Figure 3 C Code for the given problem statement using semaphore and mutex continued

```c
void *consumer (void*args){
    int c;
    while (consumed_item_count<total_items){
        sem_wait(&full);
        sem_wait(&mutex);
        c = *(buf + buf_pos);
        consume(c,pthread_self());
        --buf_pos;
        ++consumed_item_count;
        sem_post(&mutex);
        sem_post(&empty);
        sleep(1+rand()%3);
    }
    pthread_exit(NULL);
    return NULL;
}
```

Figure 4 C Code for the given problem statement using semaphore and mutex continued

```c
int main(void){
    int i, error;
    srand(time(NULL));
    sem_init(&mutex,0,1);
    sem_init(&full,0,0);
    printf("Home Many items to be produced ?");
    scanf("%d",&total_items);
    printf("Home Many number of producers?");
    scanf("%d",&producer_count);
    producers = (pthread_t*) malloc(producer_count*sizeof(pthread_t));
    printf("Enter the Number of Consumers");
    scanf("%d",&consumer_count);
    consumers = (pthread_t*) malloc(consumer_count*sizeof(pthread_t));
    printf("Enter the Buffer Capacity : ");
    scanf("%d",&buf_len);
    buf = (int*) malloc (buf_len*sizeof(int));
    sem_init(&empty,0,buf_len);
    for ( i = 0; i < producer_count; i++){
        error = pthread_create(producers+i,NULL,&producer,NULL);
        if(error!=0){
            printf("Error Creating Producer %d: %s\n",i+1,strerror(error))
        }else{
            printf("Created Producer !! %d \n",i+1);
        }}
    for ( i = 0; i < consumer_count; i++){
    error = pthread_create(consumers+i,NULL,&consumer,NULL);
    if(error!=0){
        printf("Error Creating Consumer %d: %s\n",i+1,strerror(error));
    }else{
        printf("Created Consumer!! %d \n",i+1);
    }}
    for ( i = 0; i < producer_count; i++){
        pthread_join(*(producers+i),NULL);
    }
    for ( i = 0; i < consumer_count; i++){
        pthread_join(*(consumers+i),NULL);
    }
    return 0;
    }
```

Figure 5 C Code for the given problem statement using semaphore and mutex continued

Result

To Compile

```
>> gcc filename.c -o filename -lpthread
```

To Run the executable file

```
>> ./filename
```

```
Home Many items to be produced ? 6
Home Many number of producers ? : 2
Enter the Number of Consumers : 2
Enter the Buffer Capacity : 3

Created Producer !! 1

Created Producer !! 2

Created Consumer!! 1

Created Consumer!! 2
Producer 2 Produced Item 6
Producer 1 Produced Item 5
Buffer :65
 Consumer 2 consumed item 5
 Current buffer length is 1
Buffer :6
 Consumer 1 consumed item 5
 Current buffer length is 0
Producer 1 Produced Item 5
Producer 2 Produced Item 2
Buffer :52
 Consumer 2 consumed item 5
 Current buffer length is 1
Buffer :5
 Consumer 1 consumed item 5
 Current buffer length is 0
Producer 2 Produced Item 2
Producer 1 Produced Item 5
Buffer :25
 Consumer 2 consumed item 5
 Current buffer length is 1
Producer 2 Produced Item 3
Buffer :23
 Consumer 1 consumed item 3
 Current buffer length is 1
Buffer :2
 Consumer 1 consumed item 2
 Current buffer length is 0
```

Figure 6 C output for the given problem statement using semaphore and mutex continued

7. Analysis and Discussions

Semaphore

A semaphore is a signalling mechanism and a thread that is waiting on a semaphore can be signalled by another thread. This is different than a mutex as the mutex can be signalled only by the thread that called the wait function. A semaphore uses two atomic operations, wait and signal for process synchronization. The wait operation decrements the value of its argument S, if it is positive. If S is negative or zero, then no operation is performed. The signal operation increments the value of its argument S.

Mutex

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. This thread only releases the Mutex when it exits the critical section. A Mutex is different than a semaphore as it is a locking mechanism while a semaphore is a signalling mechanism. A binary semaphore can be used as a Mutex but a
Mutex can never be used as a semaphore.

Srand(time(NULL)) makes use of the computer's internal clock to control the choice of the seed, Since the time is continually changing, the seed is forever changing

sem_init(sem_t *sem, int pshared,unsigned int value)

if pshared is 0, the value is shared among all the threads of the process, otherwise, the semaphore is shared but should be in the shared memory

Important points to be considered in the main program:

Initialize mutex value to 1, initialize full value to 0, and initialize empty value to the length of the buffer. Error variable lets you understand the status of thread creation. mutex is the semaphore type variable to lock and unlock the mutex, so that one thread doesn't interrupt the other while accessing critical section. full value indicates number of filled slots in the buffer. empty value indicates the empty slots in the buffer, buff_pos to store the last filled slot position and buff to store buffer item.

8. Conclusions

Advantages of using mutex:

- Easy to implement: Mutexes are just simple locks that a thread obtains before entering its critical section, and then releases it.
- Guarantee synchronization: Since only one thread is in its critical section at any given time, there are no race conditions and data remain consistent.

Advantages of using Semaphores:

- Multiple processes and threads are allowed in their critical sections at the same time, without

interrupting one another.

- A binary semaphore can be used as a mutex. Systems that do not provide mutexes, the only

way out is to use a binary semaphore.

Based on their advantages and our problem semaphore is best suited for Producer consumer problem.

9. Comments

1. Limitations of Experiments

- Using semaphore there is no thread ownership. Any thread can take the ownership based on

the priority

- Using Mutex, if the main process dies then the racing thread which was sleeping will also die.
- Current program implements solution for producer and consumer problem using semaphores

only.

2. Limitations in output

The current program does not avoid racing conditions, that is why we have 8 items being

produced although the input given was 6 items.

c. Learning happened

- To implement producer consumer problem using Semaphore and Mutex.
- Differences, advantages and disadvantages.