

Operating System Lab Academic Task 3

Student Name: Sneh Srivastava

Student ID: 11802718

Email Address: snehsri99@gmail.com

Section: SJ

Roll No: 59

GitHub Link: https://github.com/snehsriv/OS_LAB.git

Question 1:

Create a scenario that has three threads. Two threads are reading the value of the shared variable whereas third thread is incrementing the value of the shared variable. If the writer thread is using the shared variable, no reader thread is allowed to use whereas both reader threads can access the shared variable simultaneously. Synchronize the problem.

Solution:

```
#include<stdio.h>
#include<unistd.h>
#include<pthread.h>

pthread_mutex_t l=PTHREAD_MUTEX_INITIALIZER;
pthread_mutex_t l1=PTHREAD_MUTEX_INITIALIZER;
int counter=1;
void *read1()
{
    pthread_mutex_lock(&l1);
    int a;
    a=counter;
    printf("In First Reader\n");
    printf("Value of Shared Variable after first read is %d\n",counter);
    pthread_mutex_unlock(&l1);
}
void *write1()
```

```

{
    pthread_mutex_lock(&l);
    int c;
    c=counter;
    sleep(2);
    printf("In Writer\n");
    c=c+1;
    counter=c;
    printf("Value of Shared Variable after write thread is %d\n",counter);
    pthread_mutex_unlock(&l);
}

void *read2()
{
    int a;
    a=counter;
    printf("In Second Reader\n");
    sleep(1);
    printf("Value of Shared Variable is %d\n",a);
}

int main()
{
    pthread_t t1,t2,t3;
    int i;
    for(i=1;i<=2;i++)
    {
        printf("<<<<<| CASE %d |>>>>>\n",i);
        pthread_create(&t3,NULL,write1,NULL);
        pthread_create(&t1,NULL,read1,NULL);
        pthread_create(&t2,NULL,read2,NULL);
        pthread_join(t3,NULL);
    }
}

```

```

        pthread_join(t1,NULL);

        pthread_join(t2,NULL);

        printf("\n");

        printf("The value of counter: %d\n",counter);

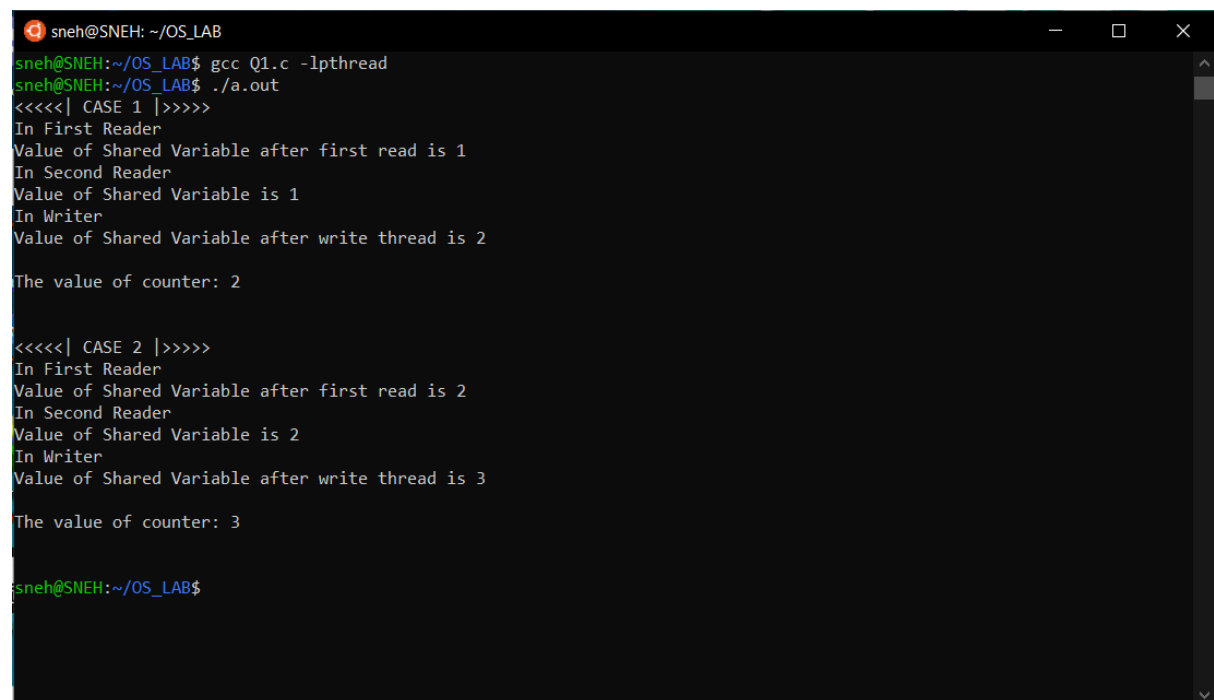
        printf("\n\n");

    }

}

```

Output:



```

sneh@SNEH: ~/OS_LAB
sneh@SNEH:~/OS_LAB$ gcc Q1.c -lpthread
sneh@SNEH:~/OS_LAB$ ./a.out
<<<<<| CASE 1 |>>>>>
In First Reader
Value of Shared Variable after first read is 1
In Second Reader
Value of Shared Variable is 1
In Writer
Value of Shared Variable after write thread is 2

The value of counter: 2

<<<<<| CASE 2 |>>>>>
In First Reader
Value of Shared Variable after first read is 2
In Second Reader
Value of Shared Variable is 2
In Writer
Value of Shared Variable after write thread is 3

The value of counter: 3

sneh@SNEH:~/OS_LAB$

```

Question 2:

Create a scenario where there are two threads where one thread is acting as a producer thread producing a random number and the other thread is acting as a consumer thread which printing the random number generated by producer thread. Producer thread can only produce if the global array to place random thread is having an empty index and consumer thread can only consume if there is element in the array. Take a shared variable counter which counts the total no of items in the array at any time. You need to synchronize both the threads using mutex locks for accessing the shared variable counter.

Solution:

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

#define MaxItems 5
#define BufferSize 5

sem_t empty;
sem_t full;
int in = 0;
int out = 0;
int buffer[BufferSize];
sem_t mutex;

void *producer(void *pno)
{
    int item,i;
    for(i = 0; i < MaxItems; i++)
    {
        item = rand();
        sem_wait(&empty);
```

```

        sem_wait(&mutex);

        buffer[in] = item;

        printf("Producer %d: Insert Item %d at %d\n", *((int *)pno),buffer[in],in);

        in = (in+1)%BufferSize;

        sem_post(&mutex);

        sem_post(&full);

    }
}

void *consumer(void *cno)
{
    int i;

    for(i = 0; i < MaxItems; i++)

        {

            sem_wait(&full);

            sem_wait(&mutex);

            int item = buffer[out];

            printf("Consumer %d: Remove Item %d from %d\n",*((int *)cno),item, out);

            out = (out+1)%BufferSize;

            sem_post(&mutex);

            sem_post(&empty);

        }

}

int main()
{
    pthread_t pro[5],con[5];

    sem_init(&mutex,0,1);

    sem_init(&empty,0,BufferSize);

    sem_init(&full,0,0);

    int a[5] = { 1,2,3,4,5};

    int i;

```

```
for(i = 0; i < 5; i++)
    pthread_create(&pro[i], NULL, (void *)producer, (void *)&a[i]);
for(i = 0; i < 5; i++)
    pthread_create(&con[i], NULL, (void *)consumer, (void *)&a[i]);
for(i = 0; i < 5; i++)
    pthread_join(pro[i], NULL);
for(i = 0; i < 5; i++)
    pthread_join(con[i], NULL);
sem_destroy(&mutex);
sem_destroy(&empty);
sem_destroy(&full);
return 0;
}
```

Output:

```
sneh@SNEH: ~/OS_LAB
sneh@SNEH:~/OS_LAB$ gcc Q2.c -lpthread
sneh@SNEH:~/OS_LAB$ ./a.out
Producer 1: Insert Item 1804289383 at 0
Consumer 1: Remove Item 1804289383 from 0
Producer 3: Insert Item 1681692777 at 1
Producer 4: Insert Item 1714636915 at 2
Producer 5: Insert Item 1957747793 at 3
Producer 2: Insert Item 846930886 at 4
Producer 1: Insert Item 424238335 at 0
Consumer 2: Remove Item 1681692777 from 1
Consumer 3: Remove Item 1714636915 from 2
Consumer 4: Remove Item 1957747793 from 3
Consumer 5: Remove Item 846930886 from 4
Producer 2: Insert Item 1189641421 at 1
Consumer 2: Remove Item 424238335 from 0
Producer 4: Insert Item 1649760492 at 2
Producer 5: Insert Item 596516649 at 3
Consumer 1: Remove Item 1189641421 from 1
Producer 2: Insert Item 1350490027 at 4
Producer 1: Insert Item 1025202362 at 0
Consumer 3: Remove Item 1649760492 from 2
Consumer 4: Remove Item 596516649 from 3
Producer 3: Insert Item 719885386 at 1
Consumer 1: Remove Item 1350490027 from 4
Producer 2: Insert Item 2044897763 at 2
Producer 4: Insert Item 783368690 at 3
Consumer 4: Remove Item 1025202362 from 0
Consumer 5: Remove Item 719885386 from 1
Producer 3: Insert Item 1365180540 at 4
Consumer 3: Remove Item 2044897763 from 2
Producer 5: Insert Item 1102520059 at 0
Consumer 4: Remove Item 783368690 from 3
Producer 4: Insert Item 304089172 at 1
Consumer 1: Remove Item 1365180540 from 4
Producer 3: Insert Item 1303455736 at 2
Consumer 3: Remove Item 1102520059 from 0
Consumer 2: Remove Item 304089172 from 1
Producer 4: Insert Item 521595368 at 3
Producer 2: Insert Item 1540383426 at 4
Producer 1: Insert Item 1967513926 at 0
Consumer 3: Remove Item 1303455736 from 2
Consumer 5: Remove Item 521595368 from 3
Consumer 4: Remove Item 1540383426 from 4
Consumer 1: Remove Item 1967513926 from 0
Producer 5: Insert Item 35005211 at 1
Producer 5: Insert Item 336465782 at 2
Producer 1: Insert Item 1726956429 at 3
Producer 3: Insert Item 294702567 at 4
Consumer 2: Remove Item 35005211 from 1
Consumer 2: Remove Item 336465782 from 2
Consumer 5: Remove Item 1726956429 from 3
Consumer 5: Remove Item 294702567 from 4
```

Question 3:

WAP that has three threads. Using semaphore, allow only two threads to access the shared variable at one time.

Solution:

```
#include<stdio.h>
```

```
#include <unistd.h>
```

```
#include<pthread.h>
```

```
#include <semaphore.h>
```

```
sem_t sem1;
```

```
int shared=10;
```

```
void *thread1()
```

```
{
```

```
    int x;
```

```
    sem_wait(&sem1);
```

```
    x=shared;
```

```
    x++;
```

```
    sleep(2);
```

```
    shared=x;
```

```
    printf("Thread 1, shared = %d\n", shared);
```

```
    sem_post(&sem1);
```

```
}
```

```
void *thread2()
```

```
{
```

```
    int x;
```

```
    sem_wait(&sem1);
```

```
    x=shared;
```

```
    x++;
```



```
        sleep(2);
        shared=x;
        printf("Thread 2, shared = %d\n", shared);
        sem_post(&sem1);
    }
```

```
void *thread3()
{
    int x;
    sem_wait(&sem1);
    x=shared;
    x++;
    sleep(2);
    shared=x;
    printf("Thread 3, shared = %d\n", shared);
    sem_post(&sem1);
}
```

```
int main()
{
    pthread_t t1, t2, t3;
    sem_init(&sem1, 0, 2);

    printf("Initially, shared = %d\n", shared);

    pthread_create(&t1, NULL, thread1, NULL);
    pthread_create(&t2, NULL, thread2, NULL);
    pthread_create(&t3, NULL, thread3, NULL);

    pthread_join(t1, NULL);
```

```
pthread_join(t2, NULL);
```

```
pthread_join(t3, NULL);
```

```
return 0;
```

```
}
```

Output:

```
sneh@SNEH: ~/OS_LAB
sneh@SNEH:~/OS_LAB$ gcc Q3.c -lpthread
sneh@SNEH:~/OS_LAB$ ./a.out
Initially, shared = 10
Thread 2, shared = 11
Thread 1, shared = 11
Thread 3, shared = 12
sneh@SNEH:~/OS_LAB$
```

Question 4:

A parent process creates a child process. The child process after its creation will send a message "Hello parent, this is child process" to its parent through pipe. Once the message is received by the parent, the parent will execute and print "This is Parent process".

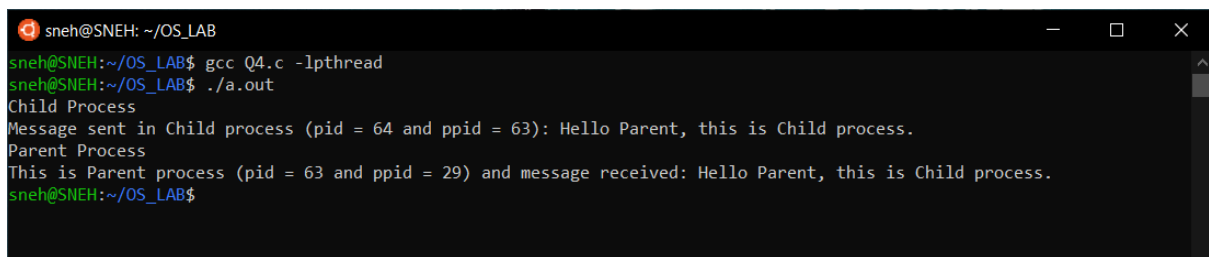
Solution:

```
#include<stdio.h>
#include<unistd.h>

int main()
{
    int pipe1[2];
    int status1;
    int pid;
    char message1[50]="Hello Parent, this is Child process.";
    char message2[50]="This is Parent process";
    char read_message[50];
    status1 = pipe(pipe1);
    if (status1 == -1)
    {
        printf("Unable to create pipe 1\n");
        return 1;
    }
    pid = fork();
    if (pid==0)
    {
        printf("Child Process\n");
        printf("Message sent in Child process (pid = %d and ppid = %d): %s\n", getpid(), getppid(),
        message1);
        write(pipe1[1], message1, sizeof(message1));
    }
    else
```

```
        {  
            read( pipe1[0], read_message, sizeof(read_message));  
            printf("Parent Process\n");  
            printf("%s (pid = %d and ppid = %d) and message  
received: %s\n", message2, getpid(), getppid(), read_message);  
        }  
  
    return 0;  
}
```

Output:



```
sneh@SNEH: ~/OS_LAB  
sneh@SNEH:~/OS_LAB$ gcc Q4.c -lpthread  
sneh@SNEH:~/OS_LAB$ ./a.out  
Child Process  
Message sent in Child process (pid = 64 and ppid = 63): Hello Parent, this is Child process.  
Parent Process  
This is Parent process (pid = 63 and ppid = 29) and message received: Hello Parent, this is Child process.  
sneh@SNEH:~/OS_LAB$
```