Student Name: Ch.Akhil

Student ID: 11702503

E-mail Address: akhilchigurupalli@gmail.com

Github Link: https://github.com/akhil-ch/OS-assignment

Question no:16

Question:

16.A barrier is a tool for synchronizing the activity of a number of threads. When a thread reaches a barrier point, it cannot proceed until all other threads have reached this point as well. When the last thread reaches the barrier point, all threads are released and can resume concurrent execution.

Assume that the barrier is initialized to N—the number of threads that must wait at the barrier point: init(N);

Each thread then performs some work until it reaches the barrier point:

/* do some work for awhile */ barrier point();

/* do some work for awhile */

Using synchronization tools like locks, semaphores and monitors, construct a barrier that implements the following API:

• int init(int n)—Initializes the barrier to the specified size.

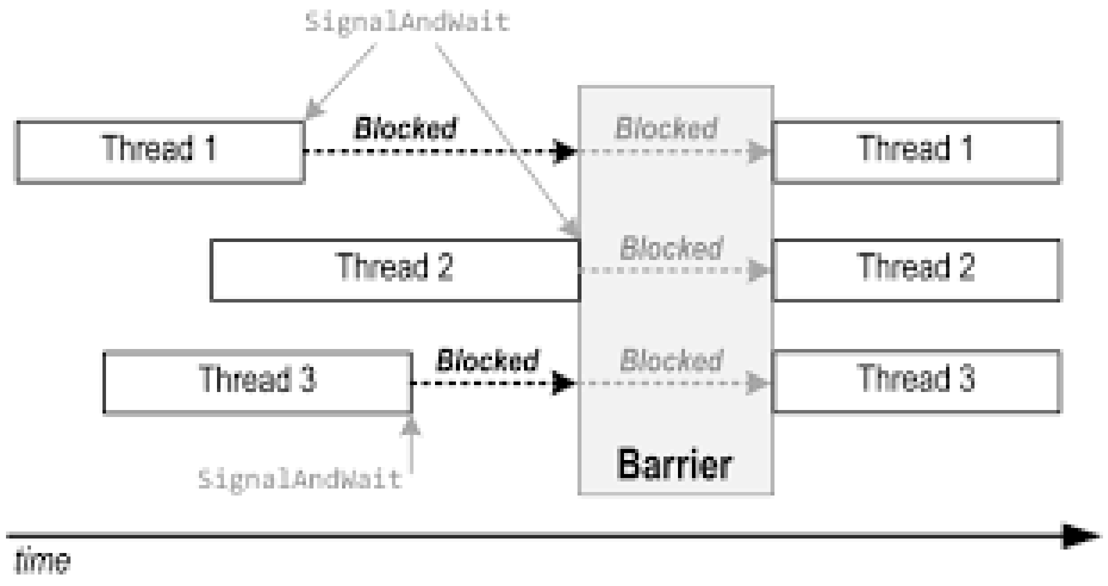
• int barrier point(void)—Identifies the barrier point.

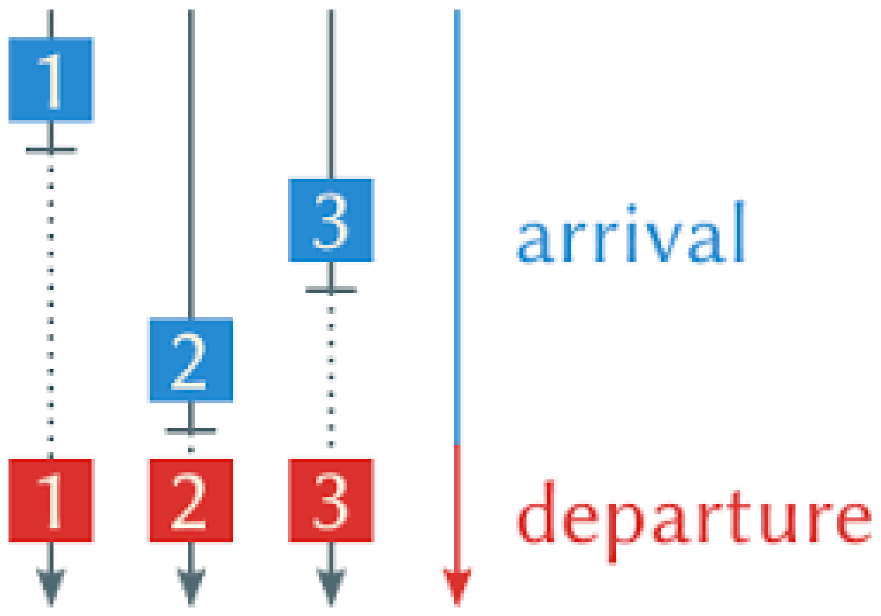
All threads are released from the barrier when the last thread reaches this point.

Barrier:

Barrier is a synchronization construct where a set of processes synchronizes globally i.e. each process in the set arrives at the barrier and waits for all others to arrive and then all processes leave the barrier.

Synchronization ensures that concurrently executing threads or processes do not execute specific portions of the program at the same time. When a barrier is inserted at a specific point in a program for a group of threads [processes], any thread [process] must stop at this point and cannot proceed until all other threads [processes] reach this barrier.





Algorithm:

1. Initialize barrier_size and thread_count
2. Create the right number of threads specified
3. Threads performing specified task
4. Threads reach barrier point and wait
5. Barrier is released when last thread reaches barrier point
6. All threads complete their task
7. Exit

Complexity:

O(n)

Where n is number of threads

Code:

```
#include<stdio.h>

#include<pthread.h>

#include<stdlib.h>

#include <unistd.h>


int tc; //thread count

int bs; //barrier size

int barrier = 0;


pthread_mutex_t lock = PTHREAD_MUTEX_INITIALIZER;

pthread_cond_t  finish_cond  = PTHREAD_COND_INITIALIZER;


int counter=0;

int invoke_barrier = 0;
```

```
void barrier_initialise(int no_threads);

void * barrier_point(void *num_threads);

int wait_barrier();

int decrement();


int main()

{

            int i,j;

            printf("Enter Barrier Size and No. of threads \n");

    scanf("%d", &bs);

    scanf("%d", &tc);

    if (bs>=0 && tc>=0)

    {

            pthread_t thread_id[tc];

            barrier_initialise(bs);

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

        {

            pthread_create(&(thread_id[i]), NULL, &barrier_point, &tc);

        }

        for( j = 0; j < tc; j++)

        {

            pthread_join(thread_id[j], NULL);

        }

            }
```

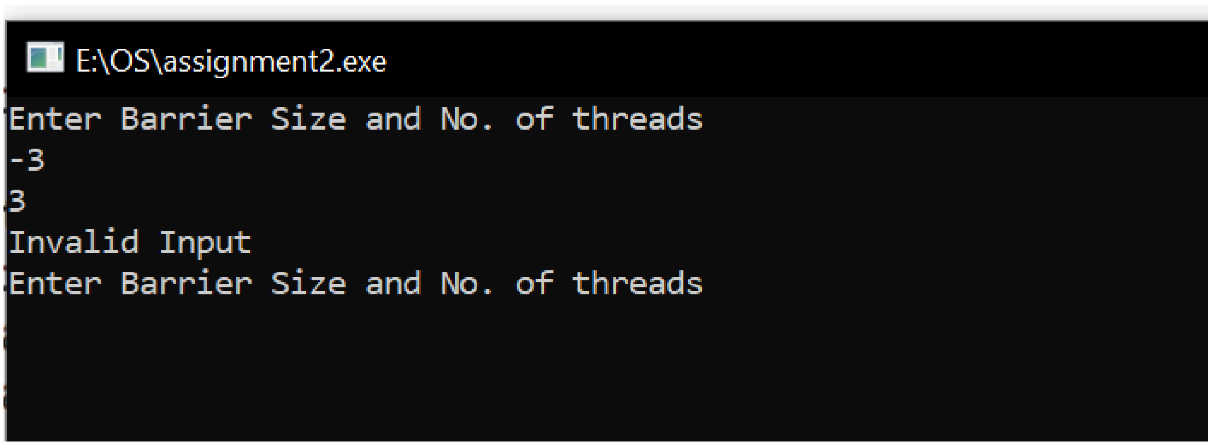
```
else
        {
                printf("Invalid Input\n");
                main();
        }
        return 0;
}
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
void barrier_initialise(int no_threads)
{
   if ( tc < bs )
        {
        barrier = tc;
        return;
        }
   barrier = no_threads;
}
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
void * barrier_point(void *num_threads)
{
    int r = rand()%9;
    printf("\nThread %d \nPerforming work of duration %d sec\n",++counter,r);
    sleep(r);
    wait_barrier();
    if (bs!=0) {
      if ((tc - (invoke_barrier++) ) % bs == 0)
              {
        printf("\nBarrier is Released\n");
      }
      printf("\nThis is task after barrier\n");
    }
    //printf("Thread completed job.\n");
    return  NULL;
}
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
int wait_barrier()
{
   if(decrement() < 0)
   {
      return -1;
   }
   while (barrier)
   {
      if(pthread_mutex_lock(&lock) != 0)
      {
         perror("\n Error in locking mutex");
         return -1;
      }
```

```
        if(pthread_cond_wait(&finish_cond, &lock) != 0)
        {
            perror("\n Error in cond wait.");
            return -1;
        }
    }
    if(0 == barrier)
    {
        if(pthread_mutex_unlock(&lock) != 0)
        {
            perror("\n Error in locking mutex");
            return -1;
        }
        if(pthread_cond_signal(&finish_cond) != 0)
        {
            perror("\n Error while signaling.");
            return -1;
        }
    }
    return 0;
}
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
```

```
int decrement()
{
 if (barrier == 0) {
        return 0;
    }
    if(pthread_mutex_lock(&lock) != 0)
    {
        perror("Failed to take lock.");
        return -1;
    }
    barrier--;
    if(pthread_mutex_unlock(&lock) != 0)
    {
        perror("Failed to unlock.");
        return -1;
    }
    return 0;
}
```

Test Cases:

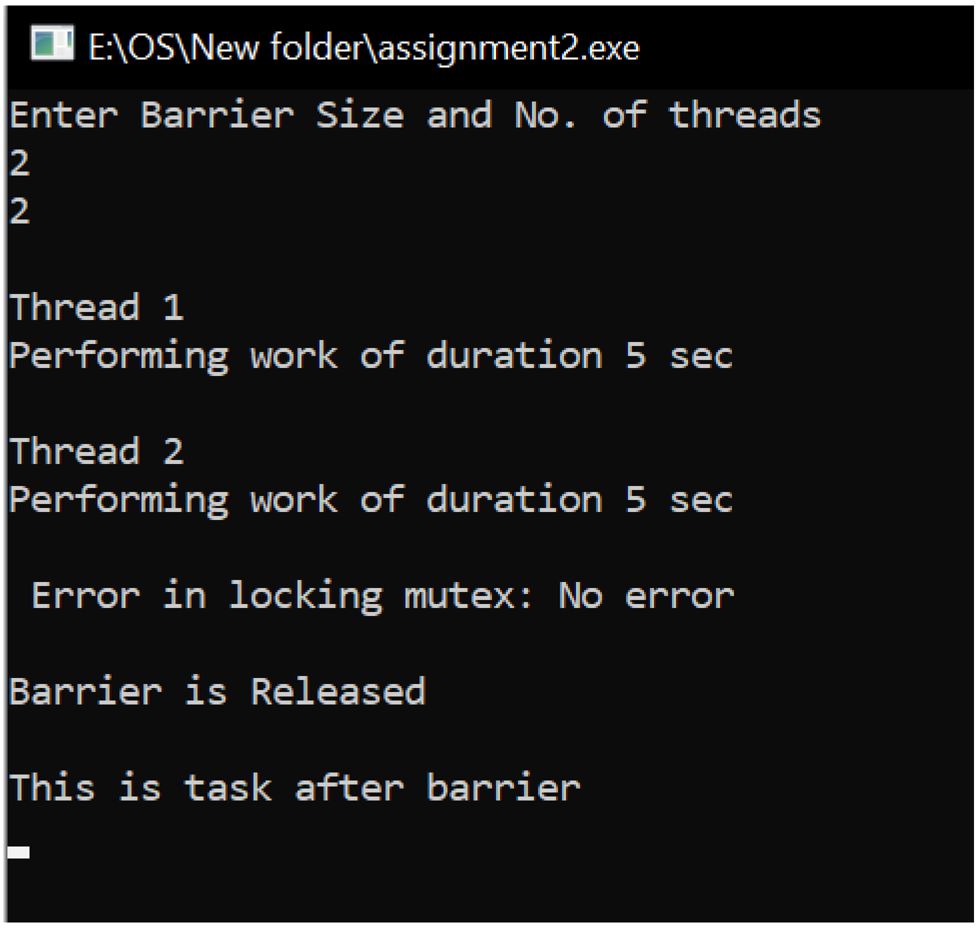
1. Invalid Input



E:\OS\assignment2.exe

```
Enter Barrier Size and No. of threads
-3
3
Invalid Input
Enter Barrier Size and No. of threads
```

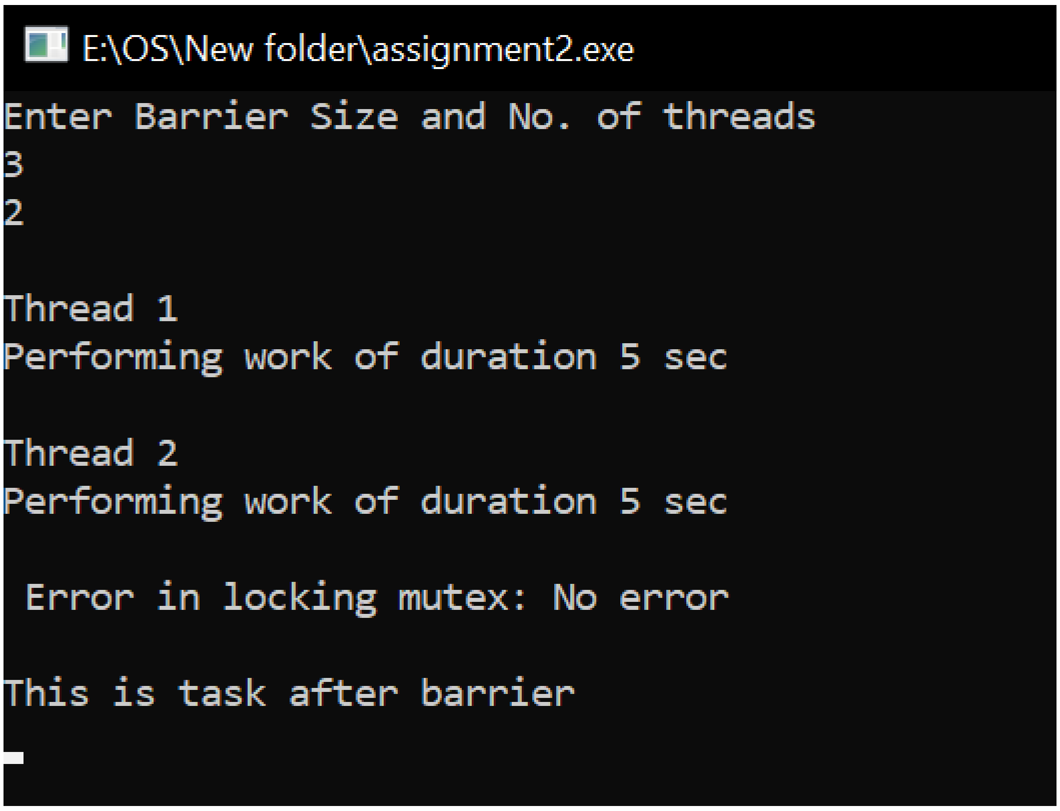
2. Number of threads = size of barrier



E:\OS\New folder\assignment2.exe

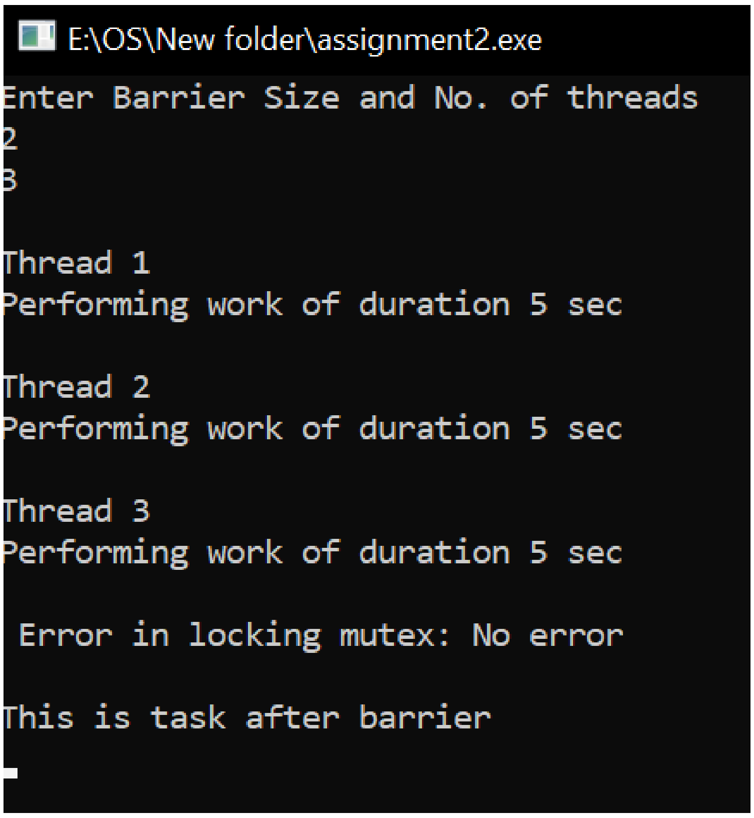
```
Enter Barrier Size and No. of threads
2
2

Thread 1
Performing work of duration 5 sec

Thread 2
Performing work of duration 5 sec

 Error in locking mutex: No error

Barrier is Released

This is task after barrier
```

3.Number of threads < size of barrier



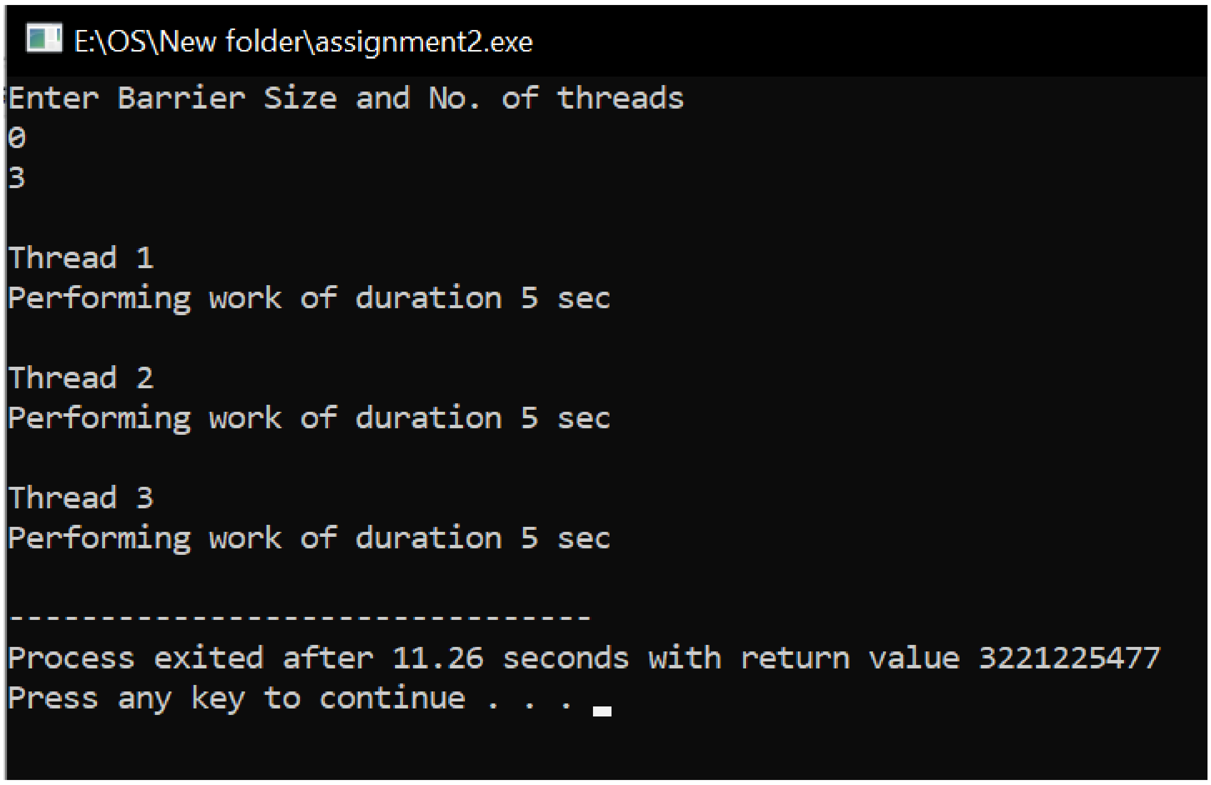
```
E:\OS\New folder\assignment2.exe
Enter Barrier Size and No. of threads
3
2

Thread 1
Performing work of duration 5 sec

Thread 2
Performing work of duration 5 sec

 Error in locking mutex: No error

This is task after barrier
```

4. Number of threads > size of barrier



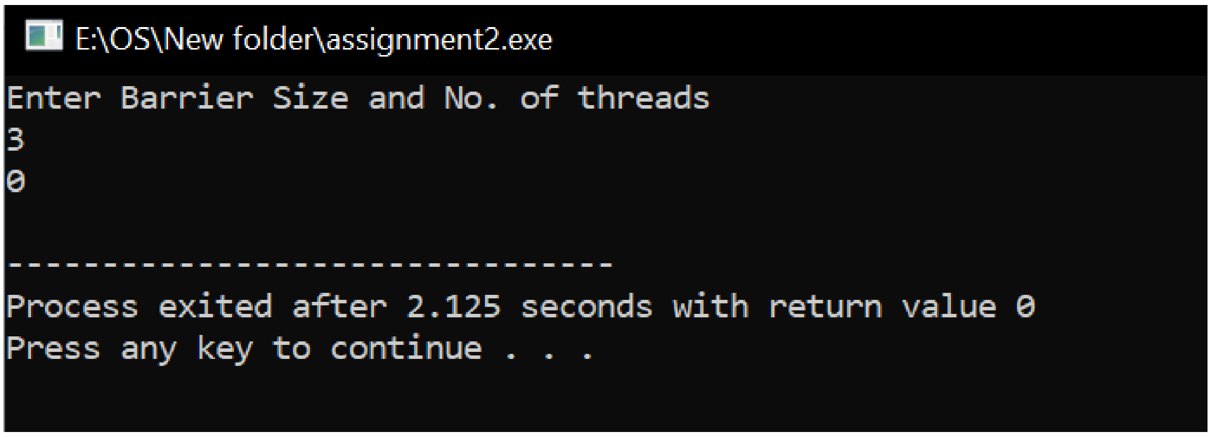
```
E:\OS\New folder\assignment2.exe
Enter Barrier Size and No. of threads
2
3

Thread 1
Performing work of duration 5 sec

Thread 2
Performing work of duration 5 sec

Thread 3
Performing work of duration 5 sec

 Error in locking mutex: No error

This is task after barrier
```

5. Barrier Size = 0


E:\OS\New folder\assignment2.exe

```
Enter Barrier Size and No. of threads
0
3

Thread 1
Performing work of duration 5 sec

Thread 2
Performing work of duration 5 sec

Thread 3
Performing work of duration 5 sec

--------------------------------
Process exited after 11.26 seconds with return value 3221225477
Press any key to continue . . .
```

6. Thread count= 0


E:\OS\New folder\assignment2.exe

```
Enter Barrier Size and No. of threads
3
0

--------------------------------
Process exited after 2.125 seconds with return value 0
Press any key to continue . . .
```