CSE-316

IN

ELCTRONIC AND COMMUNICATION ENGINEERING

OS ASSIGNMENT

Submitted by

NAME:MOHAN SAI

STUDENT ID:11714356

ROLL: B58

SEC:EE035

Email:boyamohansai@gmail.com

submitted to

MAM:MANJITKAUR,mam

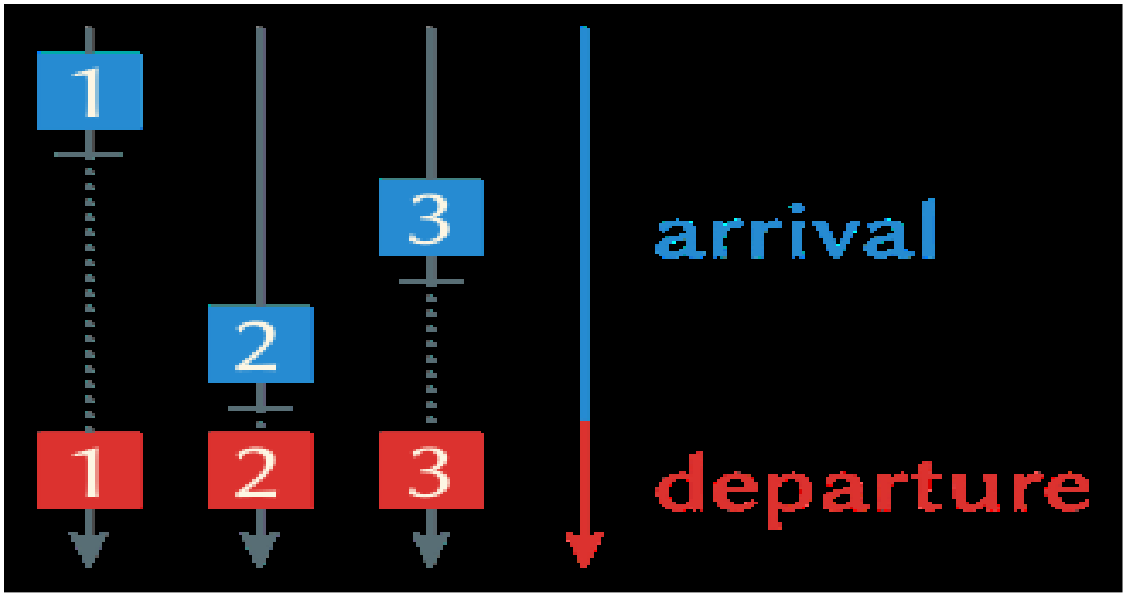
GITHUB LINK: https://github.com/Mohansai-boya/Mohan

QUESTION NO 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 described in this chapter, 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.

Barriers

a barrier is a type of synchronization method. A barrier for a group of threads or processes in the source code means any thread/process must stop at this point and cannot proceed until all other threads/processes reach this barrier.
A barrier is a method to implement synchronization. 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 threads
3. threads doing some work
4. threads waiting at the barrier.
5. barrier is released when last thread comes at the thread.
6. all threads complete thier task and exit.
7. exit.

Complexity:

O (n) complexity. “n” is no of thread_count.

CODE:

```
#include<stdio.h>

#include<pthread.h>

#include<stdlib.h>

#include <unistd.h>



pthread_mutex_t lock = PTHREAD_MUTEX_INITIALIZER;

pthread_cond_t  finish_cond  = PTHREAD_COND_INITIALIZER;

int barrier = 0;

int thread_count;

int barrier_size;

int counter=0;

int invoke_barrier = 0;
```

```
/*
 * params : number of threads a process is creating.
 * returns : none.
 *
 * Initialize barrier with total number of threads.
 */
void barrier_init(int n_threads)
{
   if ( thread_count < barrier_size ) { barrier = thread_count; return; }
   barrier = n_threads;
}




*
 * params: none.
 * returns: -1 on failure, 0 on success.
 * decrement the count by 1.
 *
 *
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;
}



/*
```

```
 *  params: none.
 *  returns: int : 0 on sucess, -1 on failure.
 *
 *
 *  wait for other threads to complete.
 */
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;
```

```
        }
    }

    /*
     * last thread will execute this.
     */-2
    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;
}

void * barrier_point(void *numthreads)
{
```

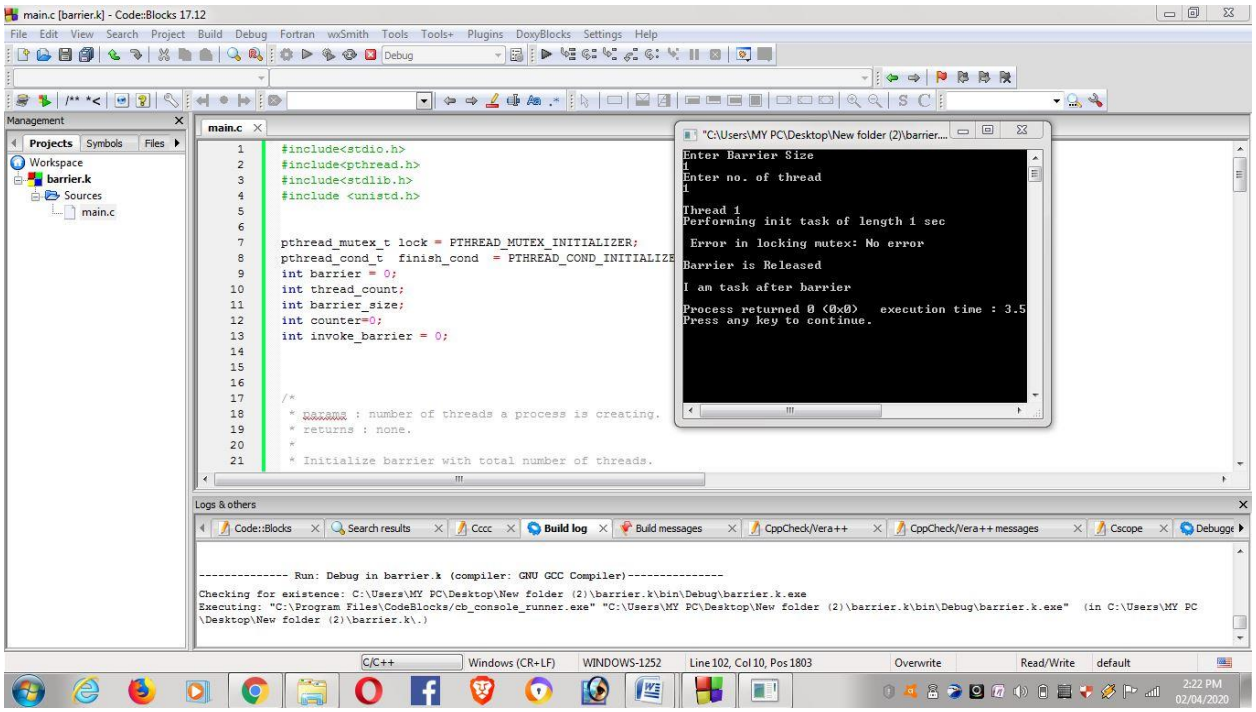
```
    int r = rand() % 5;

    printf("\nThread %d \nPerforming init task of length %d sec\n",++counter,r);
    sleep(r);

    wait_barrier();
    if (barrier_size!=0) {
      if ((thread_count - (invoke_barrier++) ) % barrier_size == 0) {
        printf("\nBarrier is Released\n");
      }
      printf("\nI am task after barrier\n");

    }
    //printf("Thread completed job.\n");

    return  NULL;
}




int main()
{
```

```
printf("Enter Barrier Size\n");
scanf("%d", &barrier_size);

printf("Enter no. of thread\n");
scanf("%d", &thread_count);

  //Checking valid input

if (barrier_size>=0 && thread_count>=0) {
   pthread_t tid[thread_count];

   barrier_init(barrier_size);

   for(int i =0; i < thread_count; i++)
   {
      pthread_create(&(tid[i]), NULL, &barrier_point, &thread_count);
   }


   for(int j = 0; j < thread_count; j++)
   {
      pthread_join(tid[j], NULL);
   }
}
//when user give wrong input then this section will execute.
```

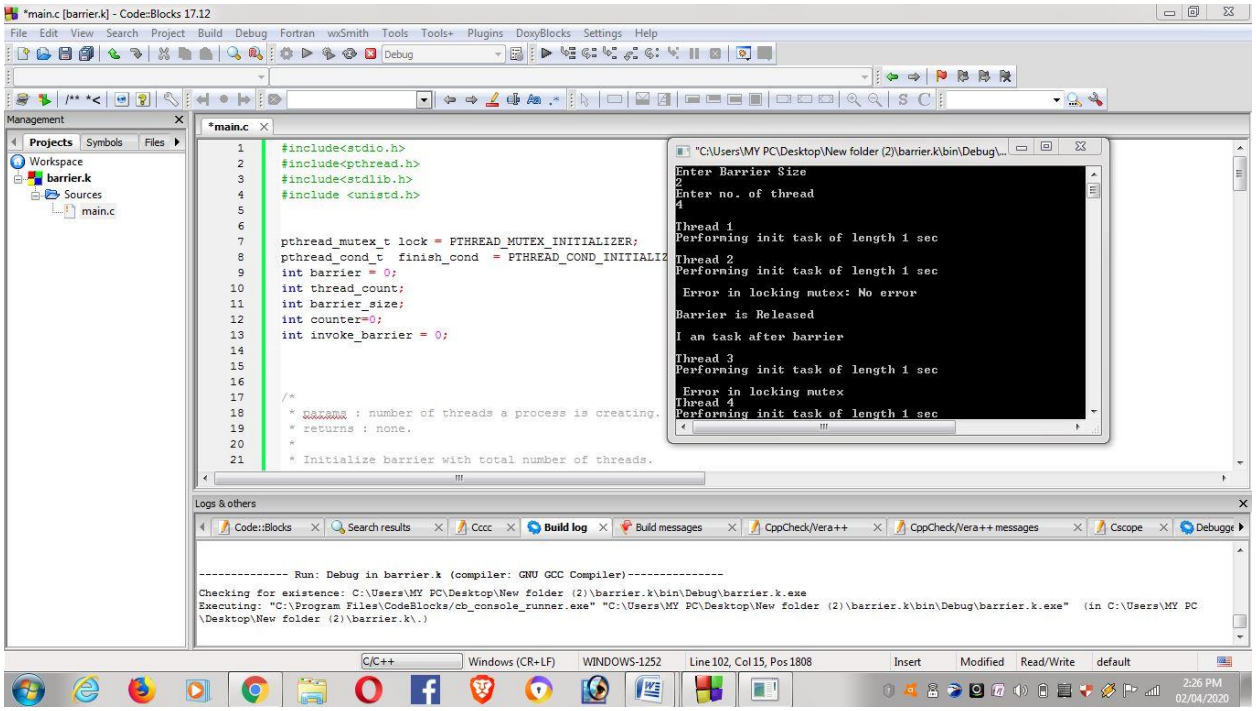
```
    else{
      printf("You are entering wrong data.\n");
      main();
    }

    return 0;
}
```

Test cases:

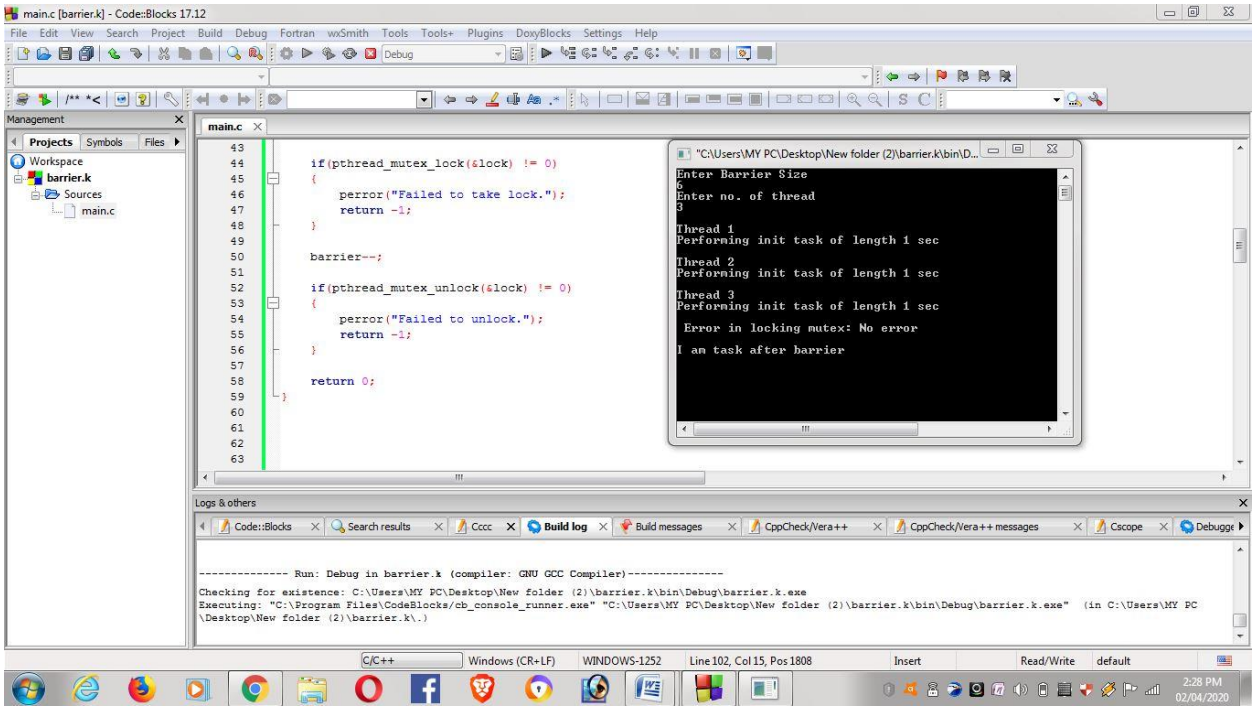
Case 1:when barrier and thread are equal



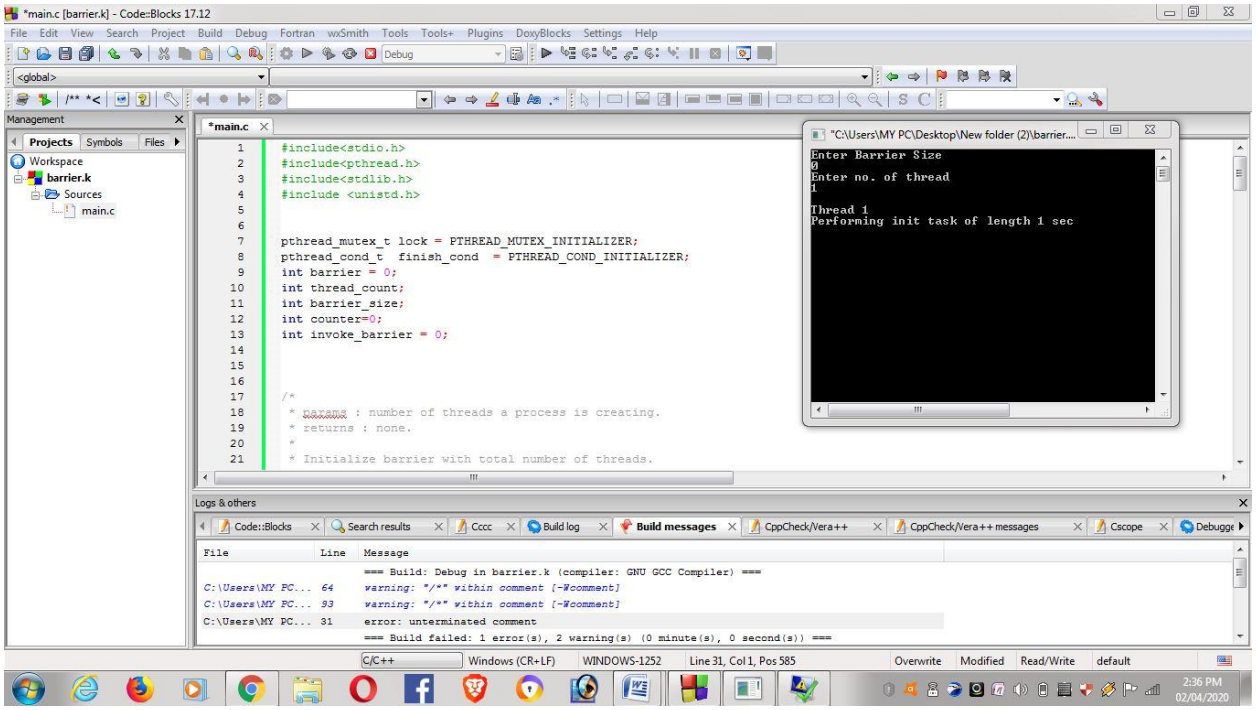
Case-2:when barrier less than thread



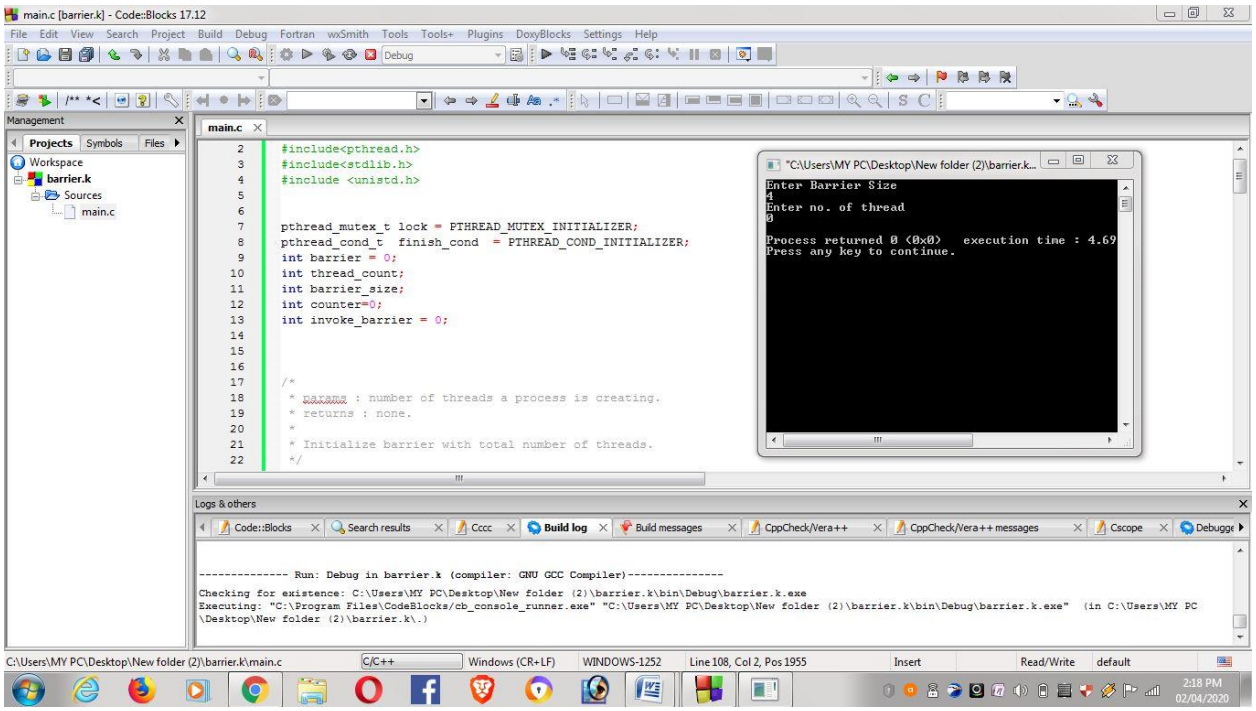
Case-3:when barrier more than thread



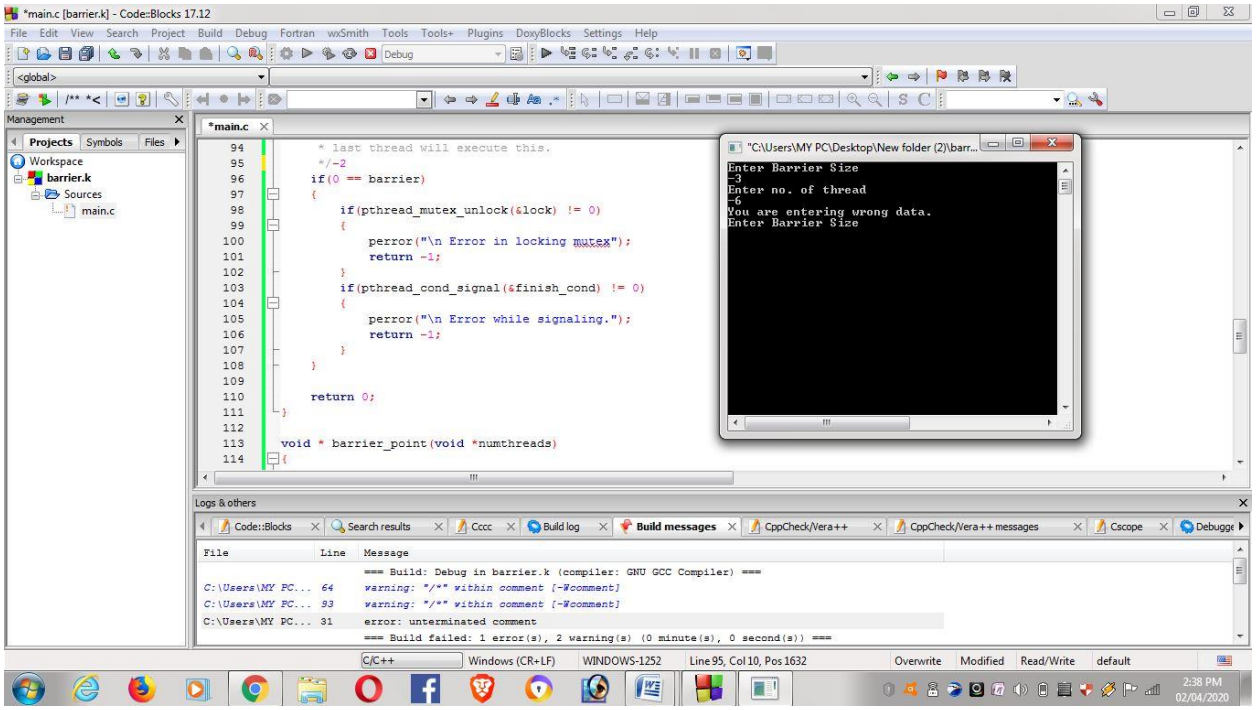
Case-4:when the barrier is zero



Case-5:when thread is given as zero



Case-6: when user enter invalid inputs like –string ,negative etc..



Github link: https://github.com/Mohansai-boya/Mohan

.