

**Operating Systems – CSE316**

*Simulation Based Assignment Report*

|  |  |
| --- | --- |
| Submitted By:  Student Name:  *Vamsi Krishna Pasam*  Student ID: *11718746*  Section: *REE031B55*  Roll No: *B55*  Email Address: [*vamsipasam2k@gmail.com*](mailto:vamsipasam2k@gmail.com)  Github link: <https://github.com/vamsipasam/OS8746assign> | **Submitted To**:  *Geetika Chatley ma’am*  *( department of computer science )* |

Question assigned is 16th .

Q. 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 threadshave 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 a while \*/ barrier point();

/\* do some work for a while \*/

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

• intinit(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.

**Description:**

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

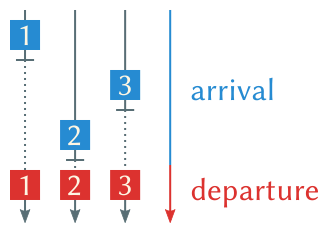
threads are released from the barrier when the last thread reaches this point.

The return value of each function is used to identify error conditions. Each function will return 0 under normal operation and will return −1 if an error occurs. A testing harness is provided in the source code download to test your implementation of the barrier.

# 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 in C++:**

#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.

\*/

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");

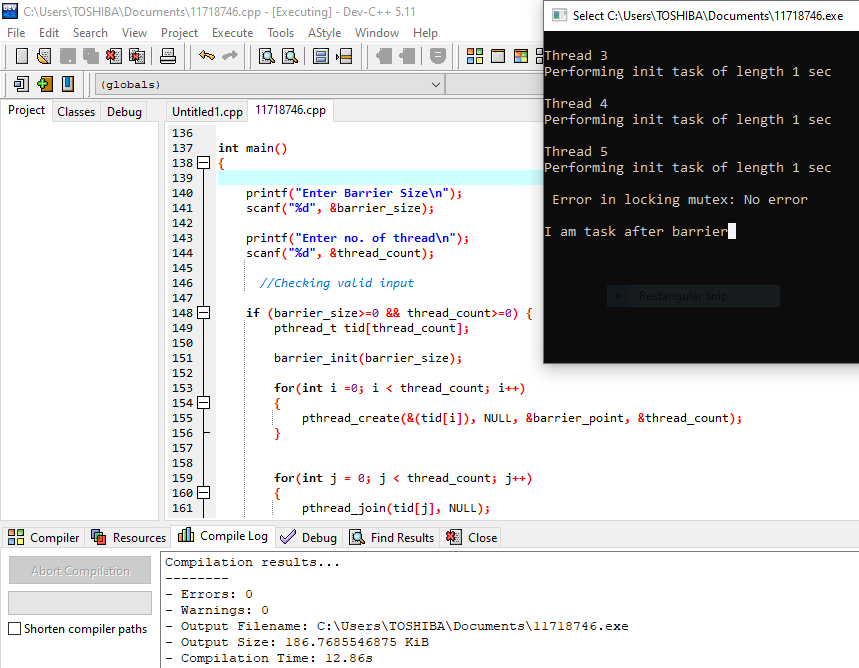
main();

}

return 0;

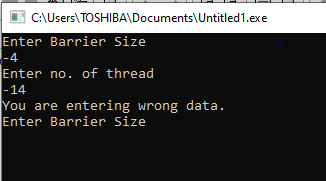
}

**Compile and Run:**

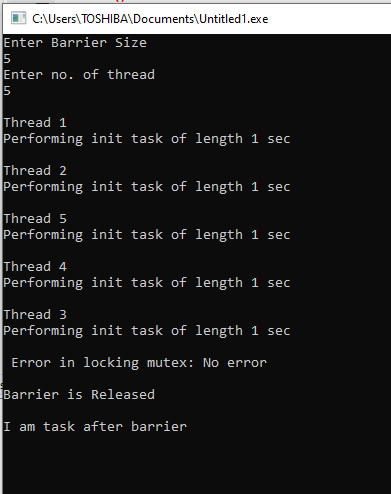


**Test Cases:**

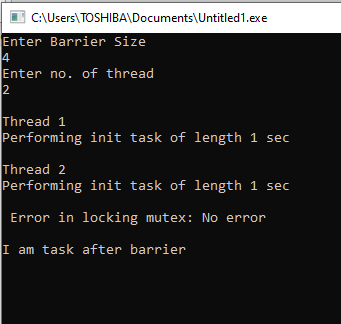
**Case 1:** when user enter invalid input like – string, double, float, negative no. etc



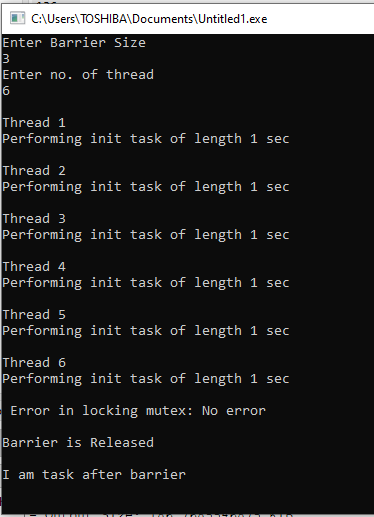
**Case 2:** when no. of thread equal to size of barrier.



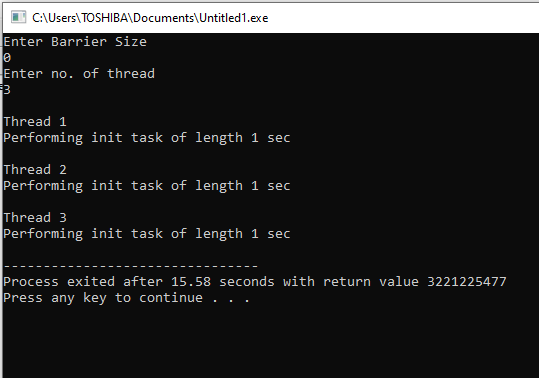
**Case 3:** when no. of thread is less than size of barrier .



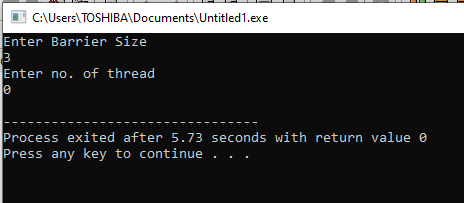
**Case 4:** when no. of thread is greater than size of Barrier.

****

**Case 5:** when size of Barrier equal to ‘0’.

****

**Case 6:** when thread equal to ‘0’.

****

**Code in Java:**

import java.io.BufferedReader;

import java.io.IOException;

import java.io.InputStreamReader;

public class TestBarrier {

//We will spawn 5 threads

public static final int THREAD\_COUNT = 5;

/\*\*

\* @param args

\* @throws IOException

\*/

public static void main(String[] args) throws IOException {

// TODO Auto-generated method stub

System.out.println("Proper output is that we should see is that all threads");

System.out.println("output an 'A' before reaching the barrier and then a 'B'");

System.out.println("after proceeding through the barrier. Therefore, output");

System.out.println("should appear as a series of 'A's followed by an equal count");

System.out.println("of 'B's. (There should not be an intermingling of 'A's and 'B's.");

System.out.println("\n\nPress Enter to begin the test:");

(new BufferedReader(new InputStreamReader(System.in))).read();

if(args.length < 1){

throw new IOException("not enterd barrier");

}

int barrier = 0 ;

try{

barrier = Integer.parseInt(args[0]);

}

catch(Exception e){

}

BarrierImpl barrierImp = new BarrierImpl(THREAD\_COUNT);

barrierImp.initBarrier(barrier);

// initialize the barrier to the number of threads

Thread[] workers = new Thread[THREAD\_COUNT];

for (int i = 0; i < workers.length; i++) {

workers[i] = new Thread(new Worker(barrierImp,i));

workers[i].start();

}

try {

for (int i = 0; i < workers.length; i++)

workers[i].join();

}

catch (InterruptedException ie) { }

}

}

Barrier Class

------------------------------------

/\*\*

\*

\* @author ramkumar

\* This class is used to implement Barrier Logic

\* Initialize barrier

\* Check Barrier limit reached

\* Move Thread to sleep if it reaches Barrier limit and others not yet

\* Wkae up all threads when all threads reach barrier point

\*/

public class BarrierImpl {

//Variable to store Barrier

private int barrier;

//Variable to check how many number of Threads reached Barrier point

private int countBarrierCrossedTheads =0;

//Variable to strore number of threads

private int numThreads = 0;

/\*\*

\* This is Object on which threads will be synchronized

\* Thread will go to Wait state on this object

\* Once all threads reach Barrier point, All Threads will be notified

\*/

private Object waitObject;

/\*

\* Constructor will take 2 arguements

\* Arg-2 ==> Barrier limit

\*/

public BarrierImpl(int numThreads){

this.numThreads = numThreads;

}

//Function to initialize Barrier

public void initBarrier(int barrier){

this.barrier = barrier;

waitObject = new Object();

}

/\*\*

\* Function to check whether barrier is reached.

\* This will be called by each thread.

\* Thread will go to wait state if Barrier is reached and others not

\*/

public boolean checkBarrierReached(int threadId,int curValue){

//check barrier reached

if(curValue == barrier){

//Current Thread reached barrier so increment count

countBarrierCrossedTheads++;

/\*\*

\* Current Thread reached barrier so check whether all others readched or not

\*/

if(countBarrierCrossedTheads != numThreads){

System.out.println("Thread "+threadId+" Reached Barrrie and all others not sso wait ");

//All threds not treached barrier, so wait

try {

synchronized(waitObject){

waitObject.wait();

}

} catch (InterruptedException e) {

// TODO Auto-generated catch block

e.printStackTrace();

}

}

else if(countBarrierCrossedTheads == numThreads){

System.out.println("All Threads reached barrier, notify all");

synchronized(waitObject){

waitObject.notifyAll();

}

}

return true;

}

//Thread already reached barrie

else if(curValue > barrier){

return true;

}

else{

return false;

}

}

}

Explanation:

﻿Worker Thread Class

---------------------------------------

public class Worker implements Runnable{

//Threads current barrier

private int currentBarrier;

//Object of BarrierImpl

BarrierImpl barrier;

//Id for Thread , this is just for identification only

int id;

public Worker(BarrierImpl barrier,int id){

this.barrier = barrier;

this.id = id;

}

@Override

public void run() {

// TODO Auto-generated method stub

//Run infinetly

while(true){

//increment barrier

currentBarrier++;

try {

Thread.sleep(id\*100);

} catch (InterruptedException e) {

// TODO Auto-generated catch block

e.printStackTrace();

}

//Check Barrier reached

/\*\*

\* IF Barrier reached Print "B"

\* Else Print "A"

\* IF currentBarrier < Barrier limit, false will be returned

\* IF currentBarrier = Barrier limit, if other threds not reached then excution will be sleep here

\* IF currentBarrier > Barrier limit "true" will be retuned

\*/

if(barrier.checkBarrierReached(id, currentBarrier)){

System.out.println("B ==> "+id);

}else

{

System.out.println("A ==> "+id);

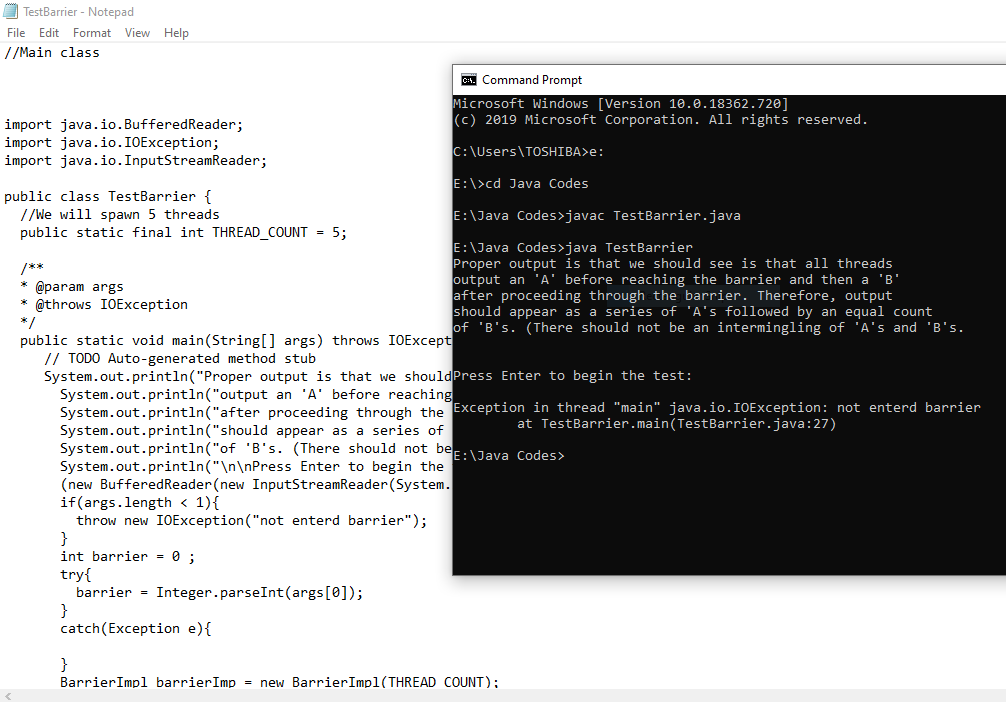
}

}

}

}

**Compilation Results:**

******

***Thank You***