

# OPERATING SYSTEMS

## REPORT

### Assignment Simulation Based

**Student Name:** Bhaveen Reddy G

**Student ID:** 11705345

**Section No:** E1710

**Roll No. :** 13

**Email Address:** bhaveen.reddy@aiesec.net

**GitHub Link:** <https://github.com/bhaveen07/OSAssignment>

#### **Description:**

5.41 A barrier is a tool for synchronising 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 initialised 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 synchronisation tools described in this chapter, construct a barrier that implements the following API :

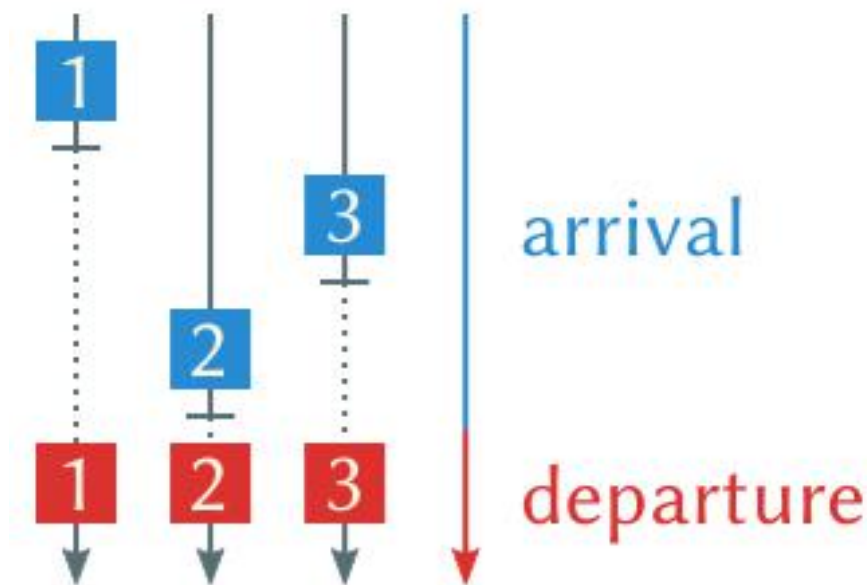
- `int init(int n)` —Initialises the barrier to the specified size.
- `int barrier point(void)` —Identifies the barrier point. All threads are released from the barrier when the last thread reaches this point.

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 synchronisation 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 synchronisation. Synchronisation 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. initialise 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 their task and exit.
7. exit.

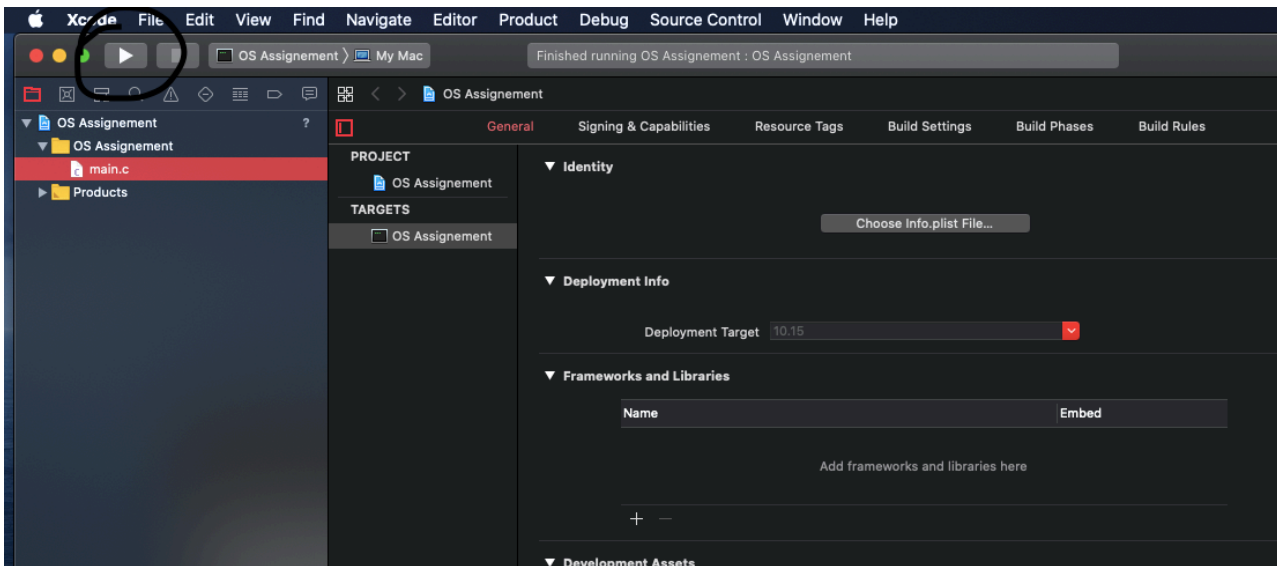
## **Complexity:**

$O(n)$  complexity. “n” is no of thread\_count.

## Compile And Run:

I used Xcode( software by apple) since I am a MacBook user.

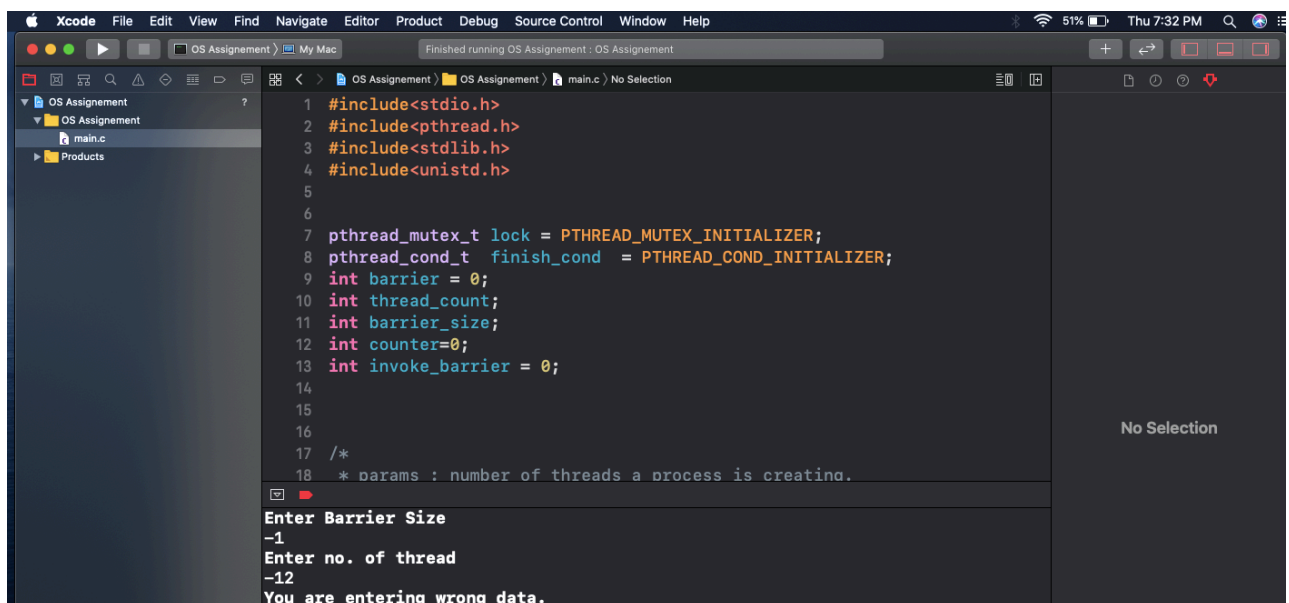
In Xcode, we can simply compile and run the programs by tapping on Play Button.



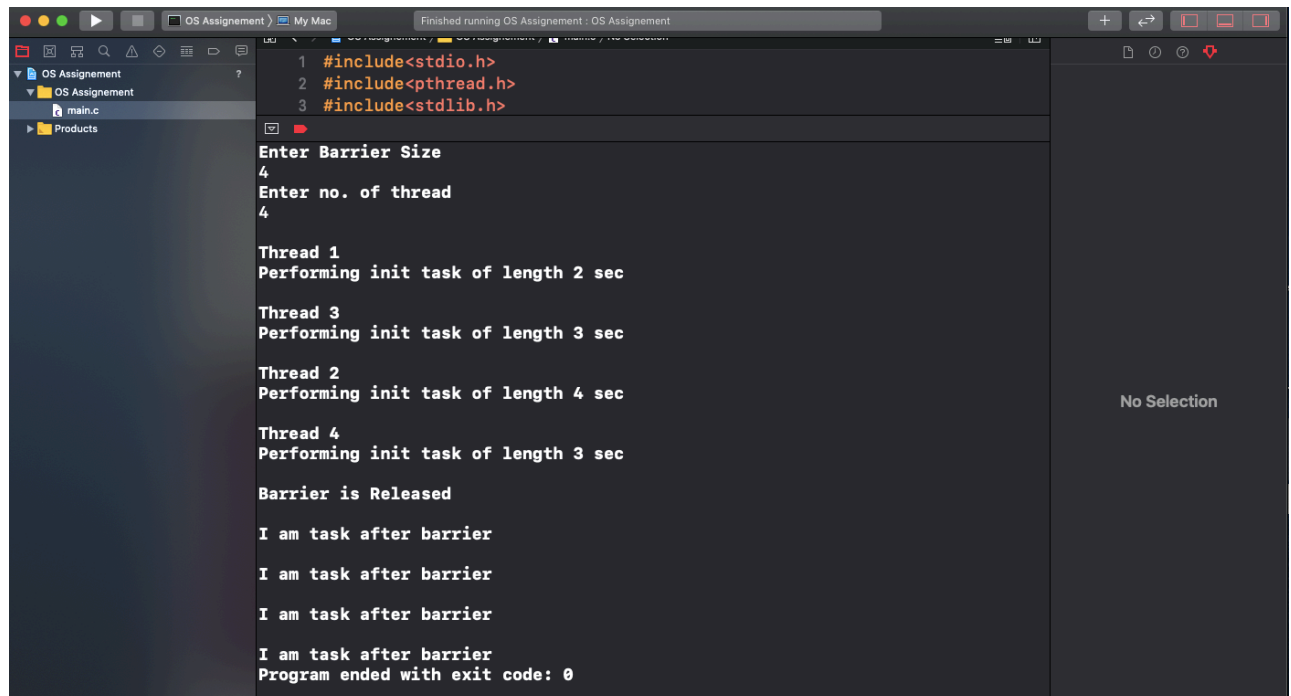
In other compilers, we have to use the particular syntax's to compile and run the program.

## 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.



The screenshot shows the Xcode IDE with a project named 'OS Assignment'. The file explorer on the left shows a folder 'OS Assignment' containing a file 'main.c'. The main editor displays the source code of 'main.c' with the following content:

```
1 #include<stdio.h>
2 #include<pthread.h>
3 #include<stdlib.h>
```

The console output shows the program's execution:

```
Enter Barrier Size
4
Enter no. of thread
4

Thread 1
Performing init task of length 2 sec

Thread 3
Performing init task of length 3 sec

Thread 2
Performing init task of length 4 sec

Thread 4
Performing init task of length 3 sec

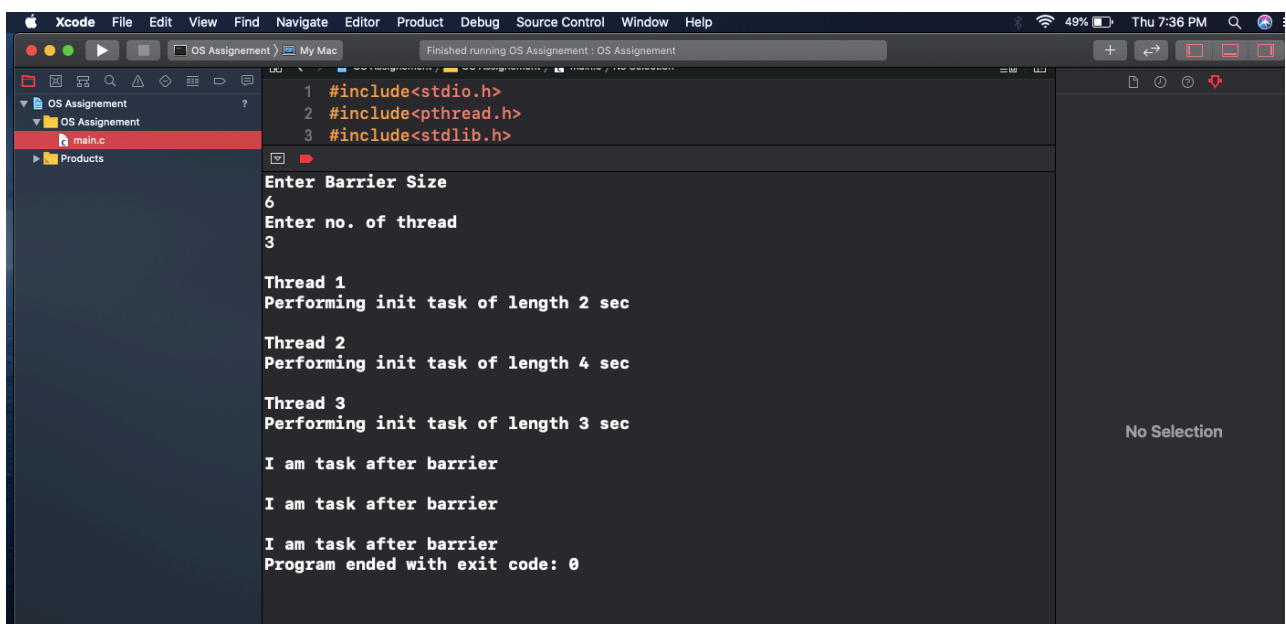
Barrier is Released

I am task after barrier
I am task after barrier
I am task after barrier
I am task after barrier

Program ended with exit code: 0
```

The right sidebar shows 'No Selection'.

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



The screenshot shows the Xcode IDE with a project named 'OS Assignment'. The file explorer on the left shows a folder 'OS Assignment' containing a file 'main.c'. The main editor displays the source code of 'main.c' with the following content:

```
1 #include<stdio.h>
2 #include<pthread.h>
3 #include<stdlib.h>
```

The console output shows the program's execution:

```
Enter Barrier Size
6
Enter no. of thread
3

Thread 1
Performing init task of length 2 sec

Thread 2
Performing init task of length 4 sec

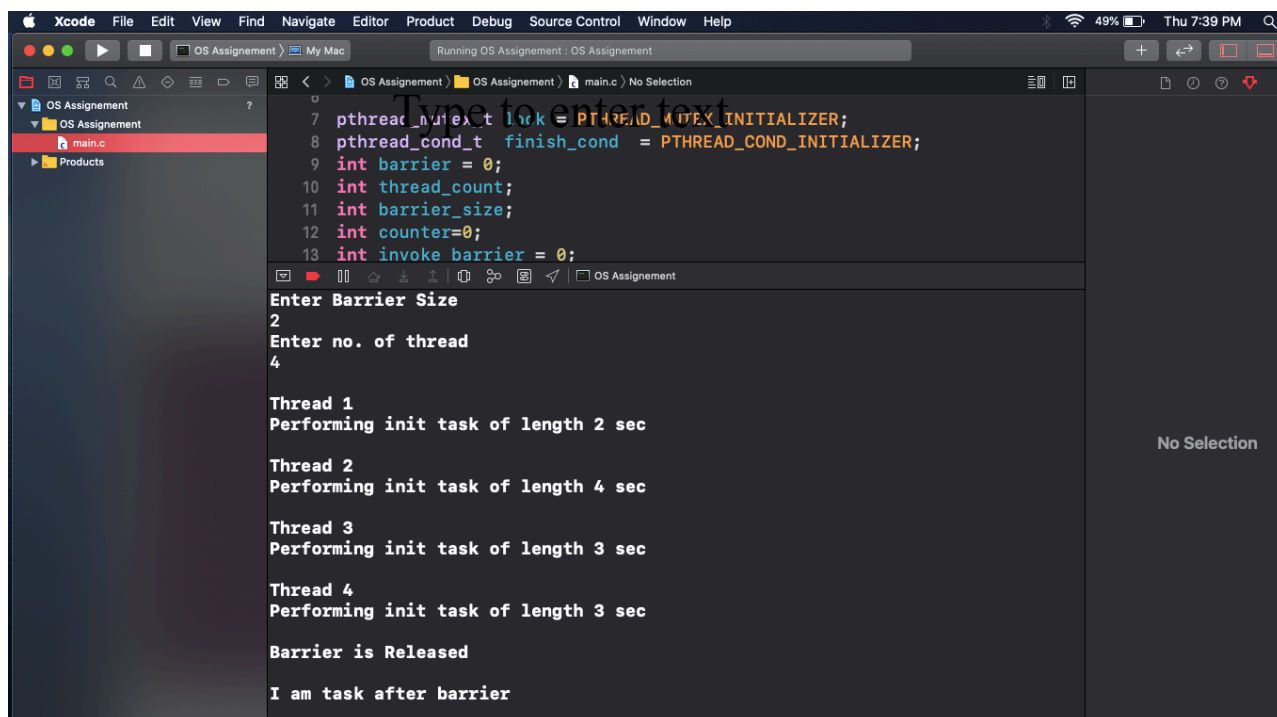
Thread 3
Performing init task of length 3 sec

I am task after barrier
I am task after barrier
I am task after barrier

Program ended with exit code: 0
```

The right sidebar shows 'No Selection'.

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



The screenshot shows the Xcode IDE with a project named 'OS Assignment'. The file 'main.c' is open, displaying C code for a multi-threaded program. The code includes pthread.h and defines a mutex, a condition variable, and several integer variables: barrier, thread\_count, barrier\_size, counter, and invoke\_barrier. The main function calls pthread\_create to create four threads. Each thread performs an initialization task of a specific length (2, 4, 3, and 3 seconds respectively). After the tasks, the threads reach a barrier. The output console shows the execution flow: 'Enter Barrier Size 2', 'Enter no. of thread 4', followed by individual thread messages indicating their task lengths. Finally, it shows 'Barrier is Released' and 'I am task after barrier'.

```
7 pthread_mutex_t lock = PTHREAD_MUTEX_INITIALIZER;
8 pthread_cond_t finish_cond = PTHREAD_COND_INITIALIZER;
9 int barrier = 0;
10 int thread_count;
11 int barrier_size;
12 int counter=0;
13 int invoke_barrier = 0;
```

Enter Barrier Size  
2  
Enter no. of thread  
4

Thread 1  
Performing init task of length 2 sec

Thread 2  
Performing init task of length 4 sec

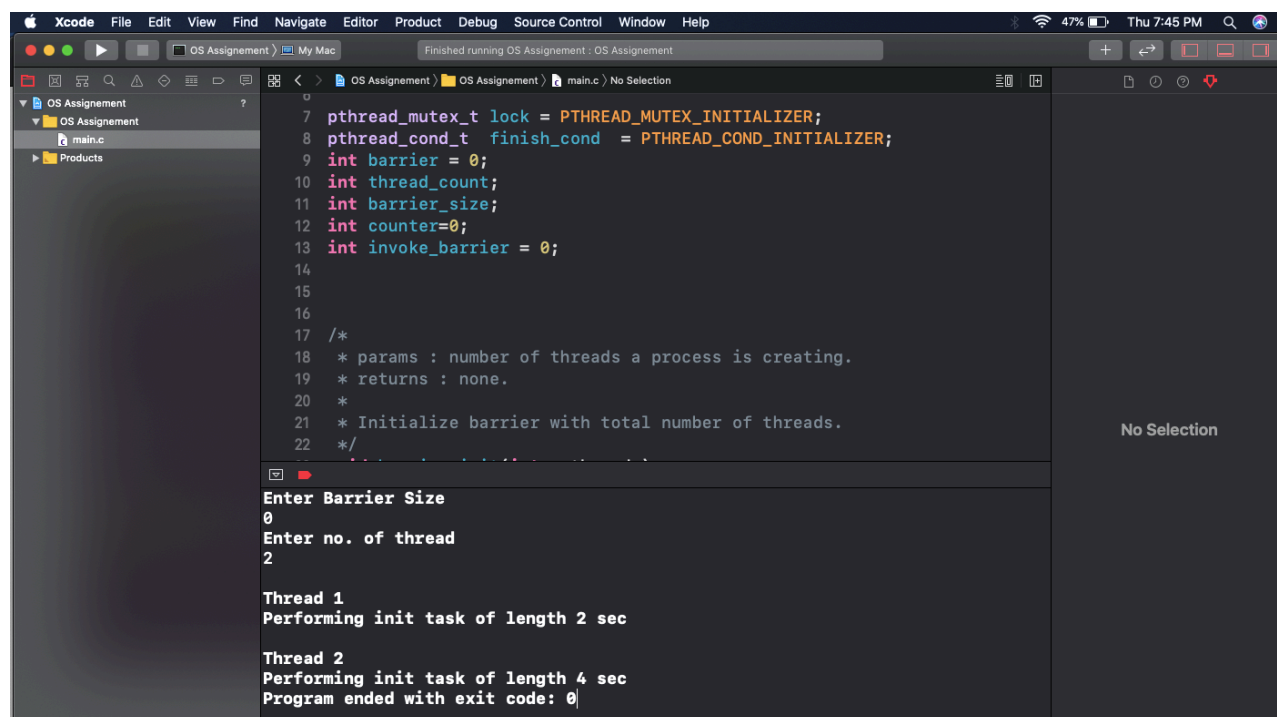
Thread 3  
Performing init task of length 3 sec

Thread 4  
Performing init task of length 3 sec

Barrier is Released

I am task after barrier

Case 5: when size of Barrier equal to '0'.



The screenshot shows the Xcode IDE with the same 'OS Assignment' project. The 'main.c' file is open, showing the same code as in Case 4, but with 'barrier\_size' set to 0. The output console shows the execution flow: 'Enter Barrier Size 0', 'Enter no. of thread 2', followed by messages for Thread 1 and Thread 2 performing their tasks. The program ends with 'Program ended with exit code: 0'.

```
7 pthread_mutex_t lock = PTHREAD_MUTEX_INITIALIZER;
8 pthread_cond_t finish_cond = PTHREAD_COND_INITIALIZER;
9 int barrier = 0;
10 int thread_count;
11 int barrier_size;
12 int counter=0;
13 int invoke_barrier = 0;
```

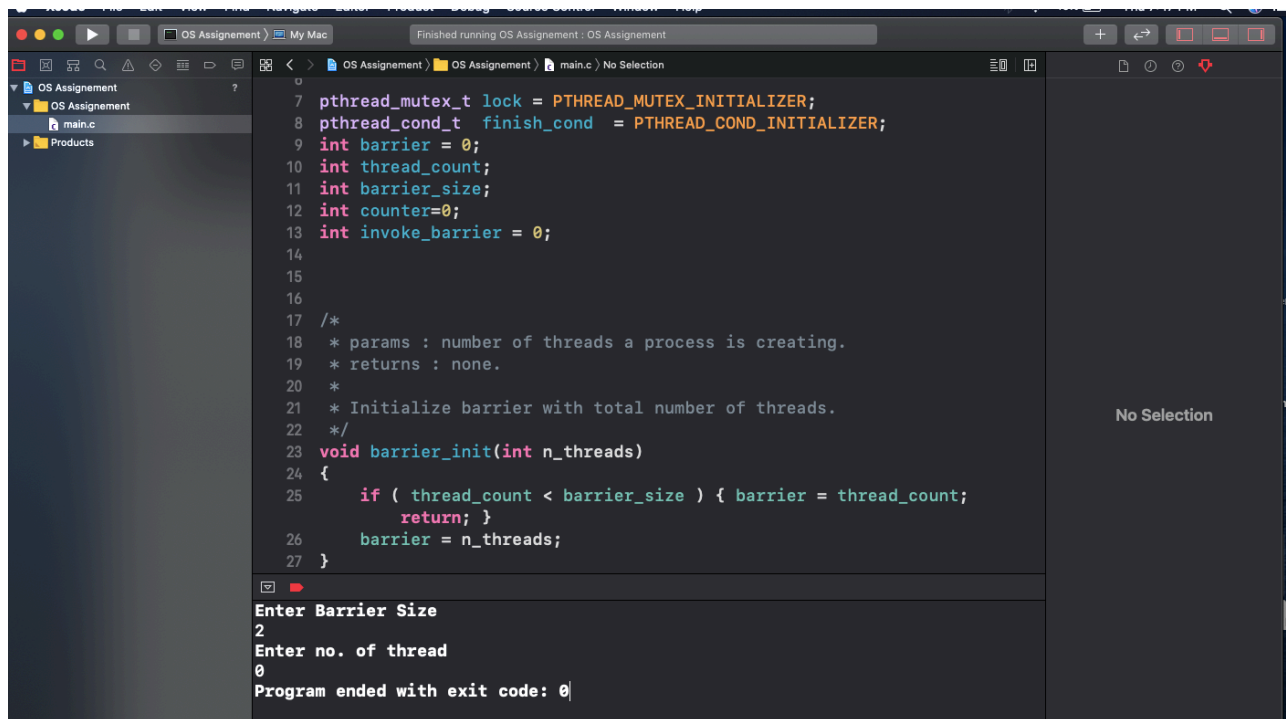
Enter Barrier Size  
0  
Enter no. of thread  
2

Thread 1  
Performing init task of length 2 sec

Thread 2  
Performing init task of length 4 sec

Program ended with exit code: 0

Case 6: when thread equal to '0'.



```
7 pthread_mutex_t lock = PTHREAD_MUTEX_INITIALIZER;
8 pthread_cond_t finish_cond = PTHREAD_COND_INITIALIZER;
9 int barrier = 0;
10 int thread_count;
11 int barrier_size;
12 int counter=0;
13 int invoke_barrier = 0;
14
15
16
17 /*
18  * params : number of threads a process is creating.
19  * returns : none.
20  *
21  * Initialize barrier with total number of threads.
22  */
23 void barrier_init(int n_threads)
24 {
25     if ( thread_count < barrier_size ) { barrier = thread_count;
26         return; }
27     barrier = n_threads;
28 }
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100
```

Enter Barrier Size  
2  
Enter no. of thread  
0  
Program ended with exit code: 0

**GitHub Link:** <https://github.com/bhaveen07/OSAssignment>

