



No E-Mail submissions will be accepted.

Submission formats and file naming:

File name : firstName_lastName_lab_8

File format: pdf or MS Word format

e.g. Donald_Trump_lab_8.pdf

1) Use the online [https://www.onlinegdb.com/online c compiler](https://www.onlinegdb.com/online_c_compiler) then run the following code.

```
1  #include <stdio.h>
2  #include <pthread.h>
3  #include <stdatomic.h>
4
5  volatile atomic_int lock = 0;
6  int counter = 0;
7
8  int TSL(volatile atomic_int *lock) {
9      return atomic_exchange(lock, 1);
10 }
11 void acquire_lock(volatile atomic_int *lock) {
12     while (TSL(lock));
13 }
14 void release_lock(volatile atomic_int *lock) {
15     atomic_store(lock, 0);
16 }
17
18 void* increment(void* arg) {
19     int *id = (int *)arg;
20     for (int i = 0; i < 1000000; i++) {
21         // acquire_lock(&lock);
22         if (i%100000 == 0)
23             printf("Thread ID : %d \n", *id) ;
24         counter++;
25         // release_lock(&lock);
26     }
27     return NULL;
28 }
29 int main() {
30     pthread_t t1, t2;
31     int T1=1, T2=2;
32     pthread_create(&t1, NULL, increment, &T1);
33     pthread_create(&t2, NULL, increment, &T2);
34     pthread_join(t1, NULL);
35     pthread_join(t2, NULL);
36     printf("Final counter value: %d\n", counter);
37     return 0;
38 }
```

- a) Attach a screenshot of your output.

```
Thread ID: 1
Thread ID: 2
Thread ID: 2
Thread ID: 1
Thread ID: 2
Thread ID: 1
Thread ID: 1
Thread ID: 1
Final counter value: 1229239
```

- b) What is the final value of the counter without a race condition, explain?

Without a race condition the final counter should be 2000000 since the loop is running 1000000 times and there are 2 threads running

- c) What do you observe? Does the counter value change if you run your code multiple times (yes/no), explain?

Yes. Because both threads are accessing the counter variable at the same time there are times when they both access the same value and some of the increments are lost due to the race condition

- d) Uncomments lines 21 and 25 and rerun your code, attach a screenshot of the output.

```
Thread ID: 2
Thread ID: 1
Thread ID: 1
Thread ID: 2
Thread ID: 1
Thread ID: 2
Thread ID: 1
Thread ID: 1
Final counter value: 2000000
```

- e) Does the counter value change if you run your code multiple times (yes/no), explain?

No. Since the threads only access the counter variable one at a time they both increment the counter 1000000 times and the output is always 2000000

- f) What is the return value of TSL(lock) if the lock is free?

It will return 0 since atomic_exchange returns the previous value stored. 0 represents a free lock

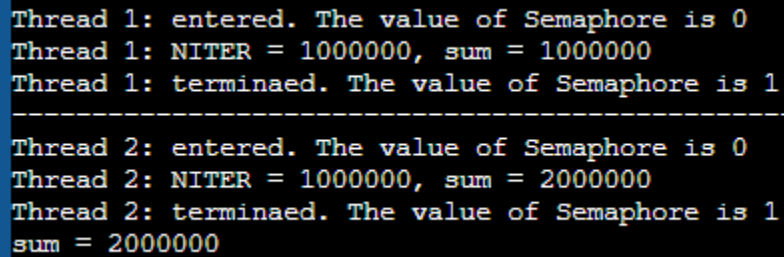
2) In the following code, the semaphores are employed to prevent the race condition. Compile and run the code and answer the following questions.

gcc sem.c -pthread -o sem.out

sem.c

```
1  #include <pthread.h>
2  #include <semaphore.h>
3  #include <stdio.h>
4  #include <stdlib.h>
5
6  #define NITER 1000000
7  sem_t mutex;
8
9  int sum = 0;
10
11 void * counter(void * a)
12 {
13     int i, tmp;
14     int value;
15     int *p = (int *)a;
16
17     sem_wait(&mutex);
18     sem_getvalue(&mutex, &value);
19     printf("-----\n");
20     printf("Thread %d : entered. The value of Semaphore is %d \n", *p, value);
21
22     for(i = 0; i < NITER; i++)
23     {
24         tmp = sum;
25         tmp = tmp+1;
26         sum = tmp;
27         if (i==(NITER-1))
28             printf("Thread %d : NITER = %d, sum = %d \n", *p, NITER, sum);
29     }
30     sem_post(&mutex);
31     sem_getvalue(&mutex, &value);
32     printf("Thread %d : terminated. The value of Semaphore is %d \n", *p, value);
33 }
34
35 int main(int argc, char * argv[])
36 {
37     pthread_t tid1, tid2;
38     int id1 = 1;
39     int id2 = 2;
40     sem_init(&mutex, 0, 1);
41
42     pthread_create(&tid1, NULL, counter, &id1);
43     pthread_create(&tid2, NULL, counter, &id2);
44
45     pthread_join(tid1, NULL);
46     pthread_join(tid2, NULL);
47
48     printf("sum = %d \n", sum);
49
50     pthread_exit(NULL);
51 }
```

- a) Does the sum value change if you run your code multiple times?
Attach a screenshot of your output.



```
Thread 1: entered. The value of Semaphore is 0
Thread 1: NITER = 1000000, sum = 1000000
Thread 1: terminaed. The value of Semaphore is 1

Thread 2: entered. The value of Semaphore is 0
Thread 2: NITER = 1000000, sum = 2000000
Thread 2: terminaed. The value of Semaphore is 1
sum = 2000000
```

No the value does not change

- b) What is the value of the mutex in line 18 and why?
0. When it gets initialized it is set to 1 and when we call sem_wait it decrements the value to 0
- c) What is the value of the mutex in line 31 and why?
1. When we call sem_post the value of mutex is incremented to 1 after the for loop
- d) What is the sum value in line 48? Does the sum value change if you run your code multiple times (yes/no)?
No. The value does not change when ran multiple times

3) Examine the Java code provided below and utilize the online Java compiler to execute your program.

https://www.onlinegdb.com/online_java_compiler

```
Main.java :
1 class Monitor {
2     private int counter = 0;
3     public void increment1(String name) {
4         for(int i = 0; i<100000; i++){
5             if( i%20000==0) System.out.println(name);
6             counter++;
7         }
8         System.out.println( name + " : counter = " + counter);
9     }
10    public void increment2(String name) {
11        for(int i = 0; i<100000; i++){
12            if( i%20000==0) System.out.println(name);
13            counter++;
14        }
15        System.out.println( name + " : counter = " + counter);
16    }
17 }
18 class CounterThread implements Runnable {
19     private Monitor mon;
20     public CounterThread(Monitor mon) {
21         this.mon = mon;
22     }
23     @Override
24     public void run() {
25         String threadName = Thread.currentThread().getName();
26         if (threadName=="T1"){
27             mon.increment1(threadName);
28         }else{
29             mon.increment2(threadName);
30         }
31     }
32 }
33 public class Main {
34     public static void main(String[] args) {
35         Monitor mon = new Monitor();
36         Thread thread1 = new Thread(new CounterThread(mon));
37         Thread thread2 = new Thread(new CounterThread(mon));
38         thread1.setName( "T1");
39         thread2.setName( "T2");
40         thread1.start();
41         thread2.start();
42     }
43 }
```

a) Provide a screenshot of the output.

```
T1
T2
T2
T1
T2
T1
T2
T2
T1
T1
T1: counter = 169222
T2: counter = 129460
```

b) Have you obtained an accurate counter value after executing your code (hint: Execute your code multiple times.)? Please clarify your response.

No the counter value is not accurate every time. this is due to a race condition of accessing the counter variable. This again means that some of the increments are lost

c) Implement the specified changes and then run your code again.

```
3 ▾ public synchronized void increment1(String name) {
```

```
10 ▾ public synchronized void increment2(String name) {
```

c.1 Provide a screenshot of the output.

```
T1
T1
T1
T1
T1
T1: counter = 100000
T2
T2
T2
T2
T2
T2: counter = 200000
```

c.2 Have you obtained an accurate counter value this time after running your code (hint: do not forget to run your code multiple times.)? Please clarify your response.

Yes the value is accurate. This is because the synchronized tag behaves like the lock does in c. Only one thread can access the variable at a time so each thread increments the value properly and the increments are not lost.