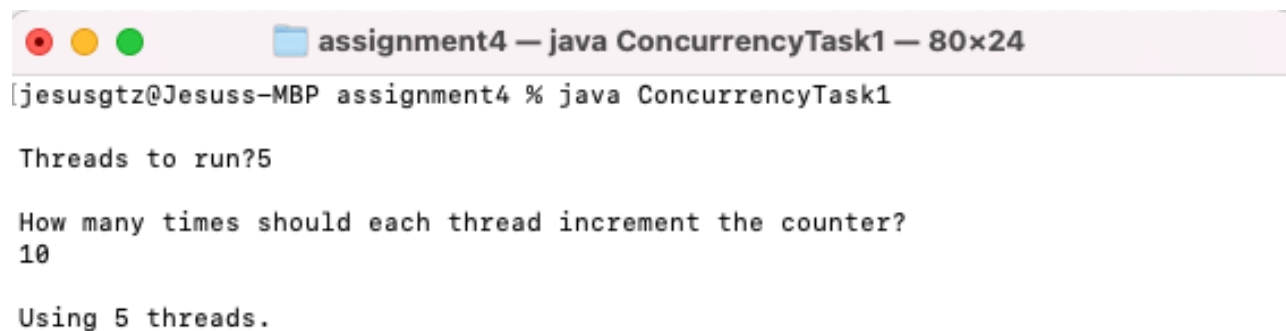


Assignment #4a
Jesus Gutierrez

Task#1:

Disclaimer: A C program was encouraged by the professor for this assignment, but I found it easier to develop half the homework in Java for the first task, and C for the second task due to personal knowledge.

This program reads the value of how many threads the user wants to run.
This program reads how many times each thread should increment the counter.
This is shown in the image below:



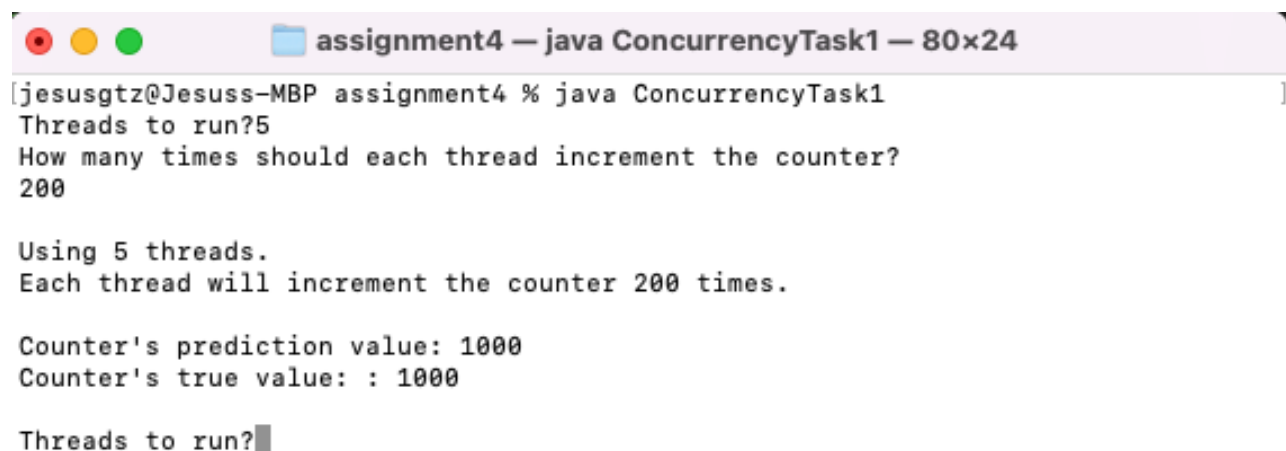
```
assignment4 — java ConcurrencyTask1 — 80x24
[jesusgtz@Jesuss-MBP assignment4 % java ConcurrencyTask1

Threads to run?5

How many times should each thread increment the counter?
10

Using 5 threads.
```

After all threads end up complete; the value of the counter will be given in the terminal as shown below, a predicted value, following its true value respectively:



```
assignment4 — java ConcurrencyTask1 — 80x24
[jesusgtz@Jesuss-MBP assignment4 % java ConcurrencyTask1

Threads to run?5
How many times should each thread increment the counter?
200

Using 5 threads.
Each thread will increment the counter 200 times.

Counter's prediction value: 1000
Counter's true value: : 1000

Threads to run?█
```

As you can see, it asks the user to run again the program. The user can continue to run the program as many times as he/she desires. The user may exit the program when he/she enters input ≤ 0 as number of threads.

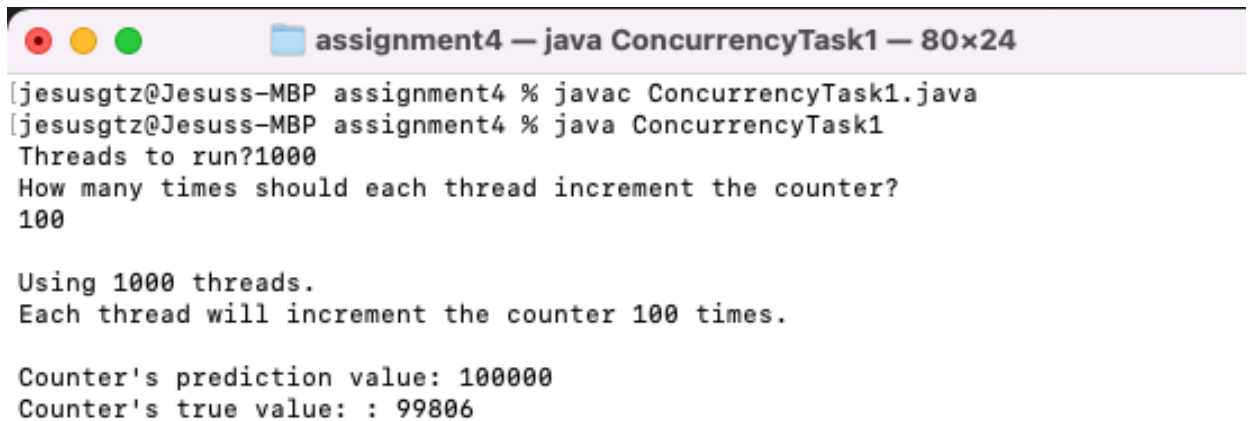
An error is given when the number of increments is large. In the image below, we run the code with 1,000,000(given with negative numbers):

```
Threads to run?100000
How many times should each thread increment the counter?
10000000

Using 100000 threads.
Each thread will increment the counter 10000000 times.

Counter's prediction value: -727379968
Counter's true value: : -796350119
```

Running the program without using locking across threads gives us the incorrect updation of the counter due to race conditions as shown here:



```
assignment4 — java ConcurrencyTask1 — 80x24
[jesusgtz@Jesuss-MBP assignment4 % javac ConcurrencyTask1.java
[jesusgtz@Jesuss-MBP assignment4 % java ConcurrencyTask1
Threads to run?1000
How many times should each thread increment the counter?
100

Using 1000 threads.
Each thread will increment the counter 100 times.

Counter's prediction value: 100000
Counter's true value: : 99806
```

Using locks to access the shared counter gives us the prediction value and the true value the same as illustrated(different number of threads and counter values):

```
assignment4 — java ConcurrencyTask1 — 80x24
[jesusgtz@Jesuss-MBP assignment4 % java ConcurrencyTask1
Threads to run?25
How many times should each thread increment the counter?
20

Using 25 threads.
Each thread will increment the counter 20 times.

Counter's prediction value: 500
Counter's true value: : 500

Threads to run?40
How many times should each thread increment the counter?
43

Using 40 threads.
Each thread will increment the counter 43 times.

Counter's prediction value: 1720
Counter's true value: : 1720


assignment4 — java ConcurrencyTask1 — 80x24
[jesusgtz@Jesuss-MBP assignment4 % java ConcurrencyTask1
Threads to run?
15
How many times should each thread increment the counter?
50

Using 15 threads.
Each thread will increment the counter 50 times.

Counter's prediction value: 750
Counter's true value: : 750
```

Task#2:

This program takes the **N threads** as an argument that follows a 'while loop' to continuously print the numbers 1,2,3....N times till user exits program. This is shown below:

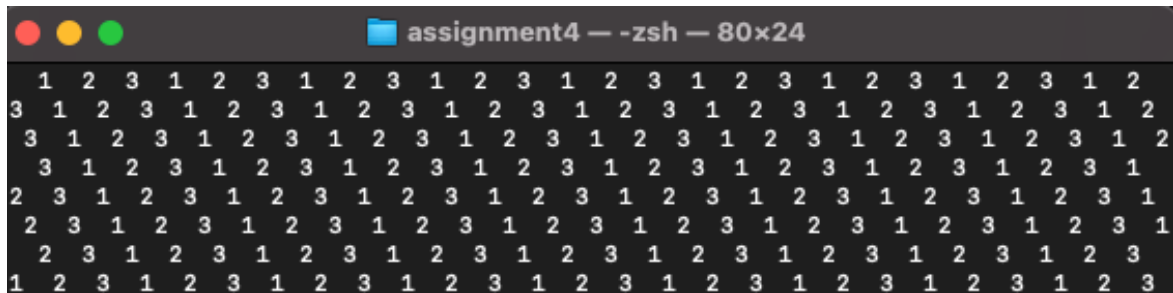


```
2 3 1 2 3 1 2 3 1 2 3 1 2 3 1 2 3 1 2 3 1 2 3 1 2 3 1
2 3 1 2 3 1 2 3 1 2 3 1 2 3 1 2 3 1 2 3 1 2 3 1 2 3 1
 2 3 1 2 3 1 2 3 1 2 3 1 2 3 1 2 3 1 2 3 1 2 3 1 2 3
1 2 3 1 2 3 1 2 3 1 2 3 1 2 3 1 2 3 1 2 3 1 2 3 1 2 3
1 2 3 1 2 3 1 2 3 1 2 3 1 2 3 1 2 3 1 2 3 1 2 3 1 2 3
 1 2 3 1 2 3 1 2 3 1 2 3 1 2 3 1 2 3 1 2 3 1 2 3 1 2
3 1 2 3 1 2 3 1 2 3 1 2 3 1 2 3 1 2 3 1 2 3 1 2 3 1 2
 3 1 2 3 1 2 3 1 2 3 1 2 3 1 2 3 1 2 3 1 2 3 1 2 3 1
2 3 1 2 3 1 2 3 1 2 3 1 2 3 1 2 3 1 2 3 1 2 3 1 2 3 1
 2 3 1 2 3 1 2 3 1 2 3 1 2 3 1 2 3 1 2 3 1 2 3 1 2 3
1 2 3 1 2 3 1 2 3 1 2 3 1 2 3 1 2 3 1 2 3 1 2 3 1 2 3
 1 2 3 1 2 3 1 2 3 1 2 3 1 2 3 1 2 3 1 2 3 1 2 3 1 2
1 2 3 1 2 3 1 2 3 1 2 3 1 2 3 1 2 3 1 2 3 1 2 3 1 2
 3 1 2 3 1 2 3 1 2 3 1 2 3 1 2 3 1 2 3 1 2 3 1 2 3 1
 3 1 2 3 1 2 3 1 2 3 1 2 3 1 2 3 1 2 3 1 2 3 1 2 3 1
  3 1 2 3 1 2 3 1 2 3 1 2 3 1 2 3 1 2 3 1 2 3 1 2 3
2 3 1 2 3 1 2 3 1 2 3 1 2 3 1 2 3 1 2 3 1 2 3 1 2 3 1
 2 3 1 2 3 1 2 3 1 2 3 1 2 3 1 2 3 1 2 3 1 2 3 1 2 3
  2 3 1 2 3 1 2 3 1 2 3 1 2 3 1 2 3 1 2 3 1 2 3 1 2
1 2 3 1 2 3 1 2 3 1 2 3 1 2 3 1 2 3 1 2 3 1 2 3 1 2 3
 1 2 3 1 2 3 1 2 3 1 2 3 1 2 3 1 2 3 1 2 3 1 2 3 1 2
  1 2 3 1 2 3 1 2 3 1 2 3 1 2 3 1 2 3 1 2 3 1 2 3 1
```

Then, we add synchronization between the threads using a lock as shown below with 'pthread_mutex_lock(&lock)':

```
while(1) {
    pthread_mutex_lock(&lock);
    if (counter != (int)*(int*)n) {
        if ((int)*(int*)n == 1) {
            pthread_cond_wait(&sync1, &lock);
        } else if ((int)*(int*)n == 2) {
            pthread_cond_wait(&sync2, &lock);
        }
        else {
            pthread_cond_wait(&sync3, &lock);
        }
    }
}
```

This implementation allows us to print in order 1,2,3...N as shown below:

A terminal window titled "assignment4 — zsh — 80x24" displays a grid of numbers. The grid consists of 8 rows and 24 columns. Each row contains a repeating sequence of the numbers 1, 2, and 3. The sequence starts with 1, followed by 2, then 3, and this pattern repeats 8 times across each row. The numbers are printed in a monospaced font, and the entire grid is centered within the terminal window.

```
1 2 3 1 2 3 1 2 3 1 2 3 1 2 3 1 2 3 1 2 3 1 2 3 1 2
3 1 2 3 1 2 3 1 2 3 1 2 3 1 2 3 1 2 3 1 2 3 1 2 3 1
3 1 2 3 1 2 3 1 2 3 1 2 3 1 2 3 1 2 3 1 2 3 1 2 3 1
3 1 2 3 1 2 3 1 2 3 1 2 3 1 2 3 1 2 3 1 2 3 1 2 3 1
2 3 1 2 3 1 2 3 1 2 3 1 2 3 1 2 3 1 2 3 1 2 3 1 2 3
2 3 1 2 3 1 2 3 1 2 3 1 2 3 1 2 3 1 2 3 1 2 3 1 2 3
2 3 1 2 3 1 2 3 1 2 3 1 2 3 1 2 3 1 2 3 1 2 3 1 2 3
1 2 3 1 2 3 1 2 3 1 2 3 1 2 3 1 2 3 1 2 3 1 2 3 1 2
```

HOW TO RUN CODES:

ConcurrencyTask1.java:

- Javac ConcurrencyTask1.java
- Java ConcurrencyTask1

ConcurrencyTask2.c:

- Gcc -o cc ConcurrencyTask2.c
- ./cc