

No E-Mail submissions will be accepted.  
Submission formats and file naming:

File name :firstName\_lastName\_lab\_7

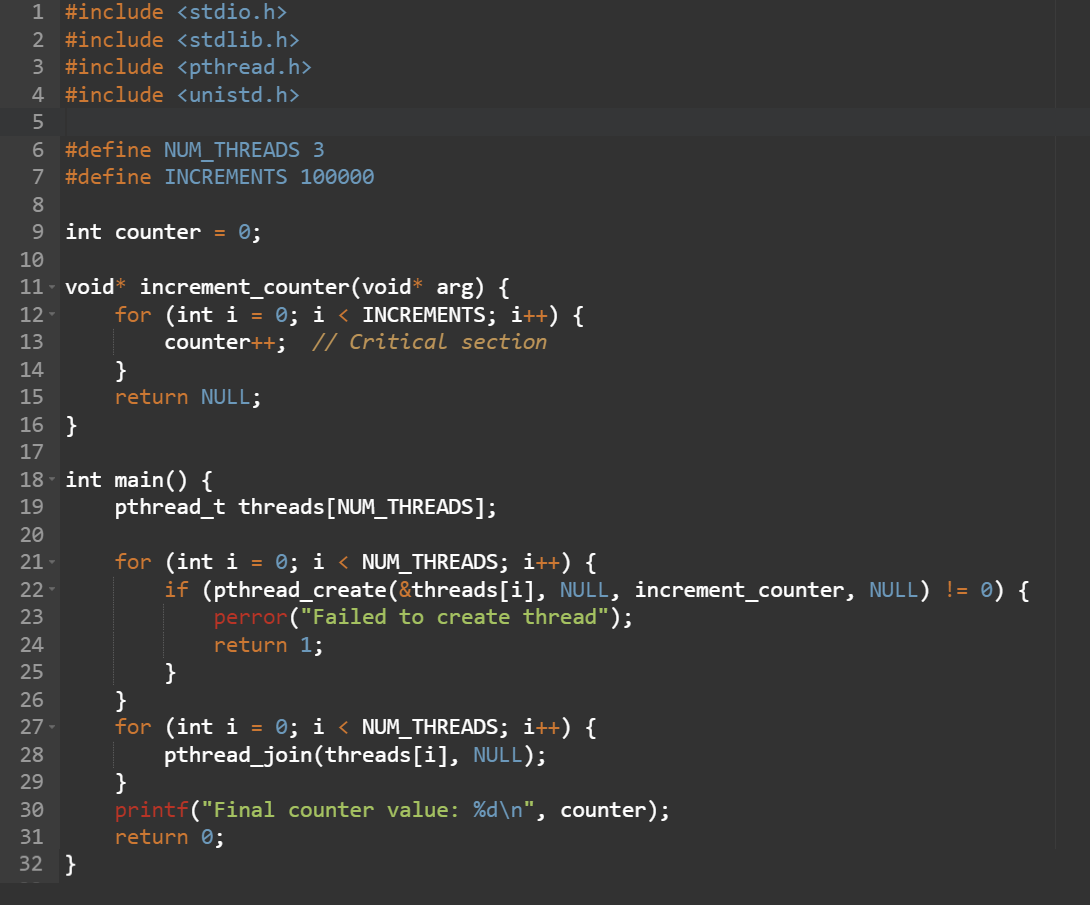
File format: pdf or MS Word format

e.g. Donald\_Trump\_lab\_7.pdf

1) Consider the following C code, use the online C compiler

<https://www.onlinegdb.com/online_c_compiler>

to run your code.



1. Attach a screenshot of your output.



1. What is the final value of the counter without a race condition, explain?

The final value without a race condition would be 100\_000 because that is the number of increments the program does

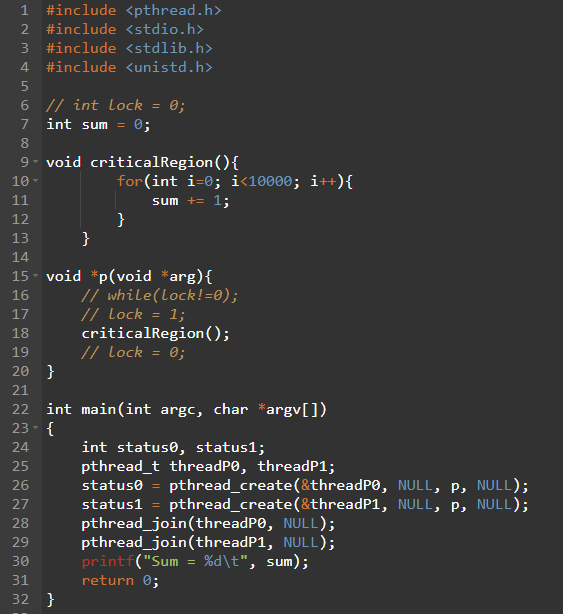
1. What is the final value of the counter with a race condition, explain?

The final value with a race condition is 139\_282 or some other random value because all the threads are incrementing the value separately. The value of the counter will be the same for some threads when they try and access it at the same time.

2) Consider the following C code, first compile your code to obtain the binary file *lock.out*.

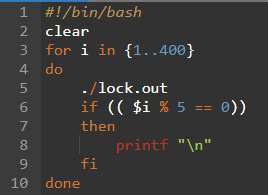
*gcc lock.c -pthread -o lock.out*

*lock.c*



In order to execute *lock.out* 400 times you need to prepare a bash script file (called *run*) as follows:

*runLock.sh*

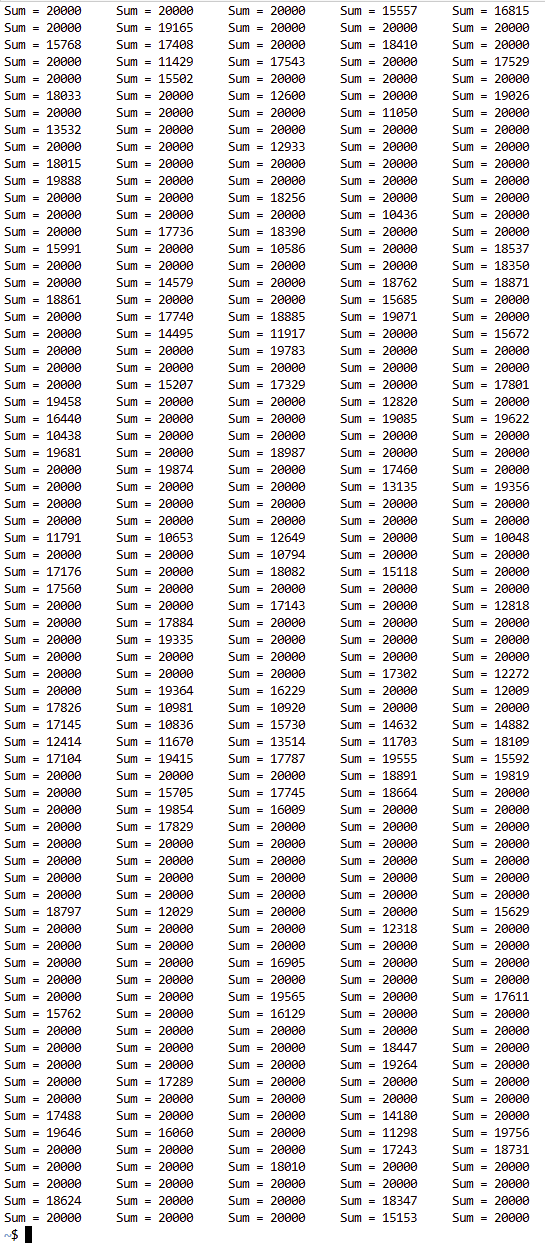


Then use the “chmod +x” command so that the system recognizes it as an executable file.

*chmod +x runLock.sh*

Now run your bash script (*./runLock.sh*) and answer the following questions:

1. Attach a screenshot of your output.

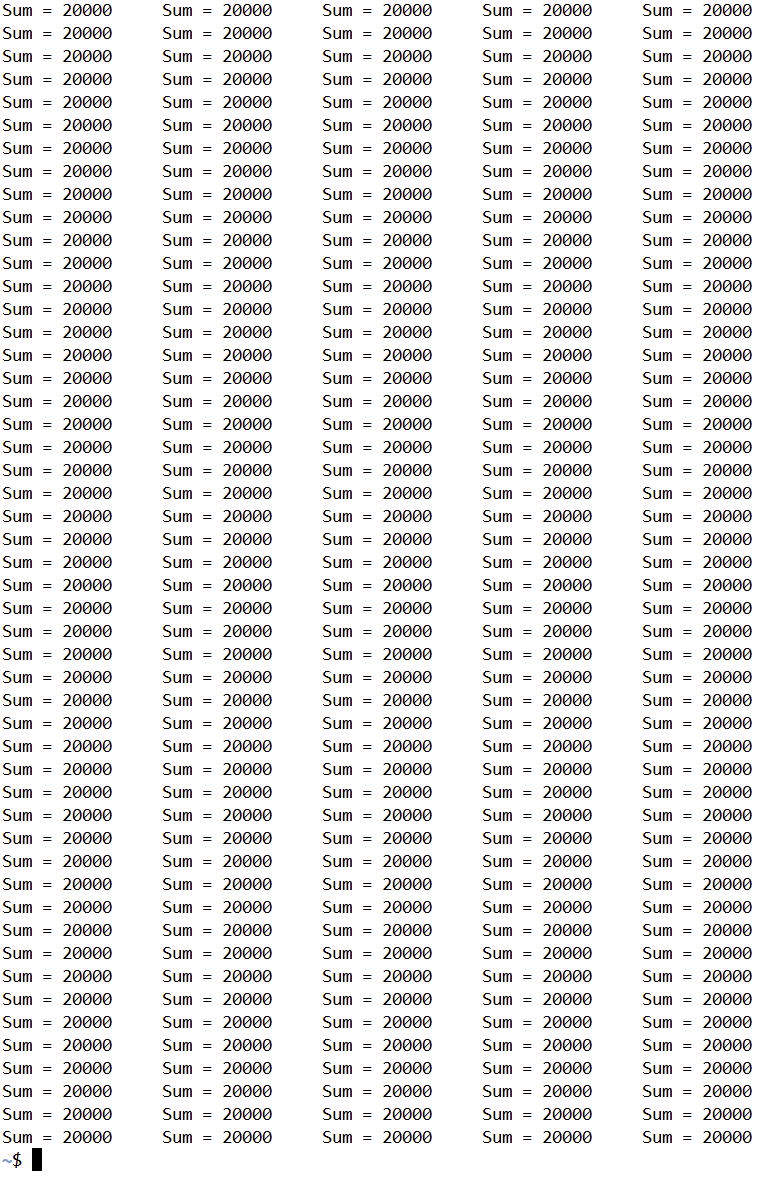


1. Why doesn’t the value of *sum* remain constant, explain?

There is a race condition between the two threads. When they add 1 to the value they sometimes increment the same value and one of the increments gets lost

Now uncomment lines 6, 16, 17,and 19 from the file *lock.c*, compile your new code and run the bash script again (*./runLock.sh*).

1. Attach a screenshot of your output.



1. Does the value of *sum* remain constant this time or not, explain?

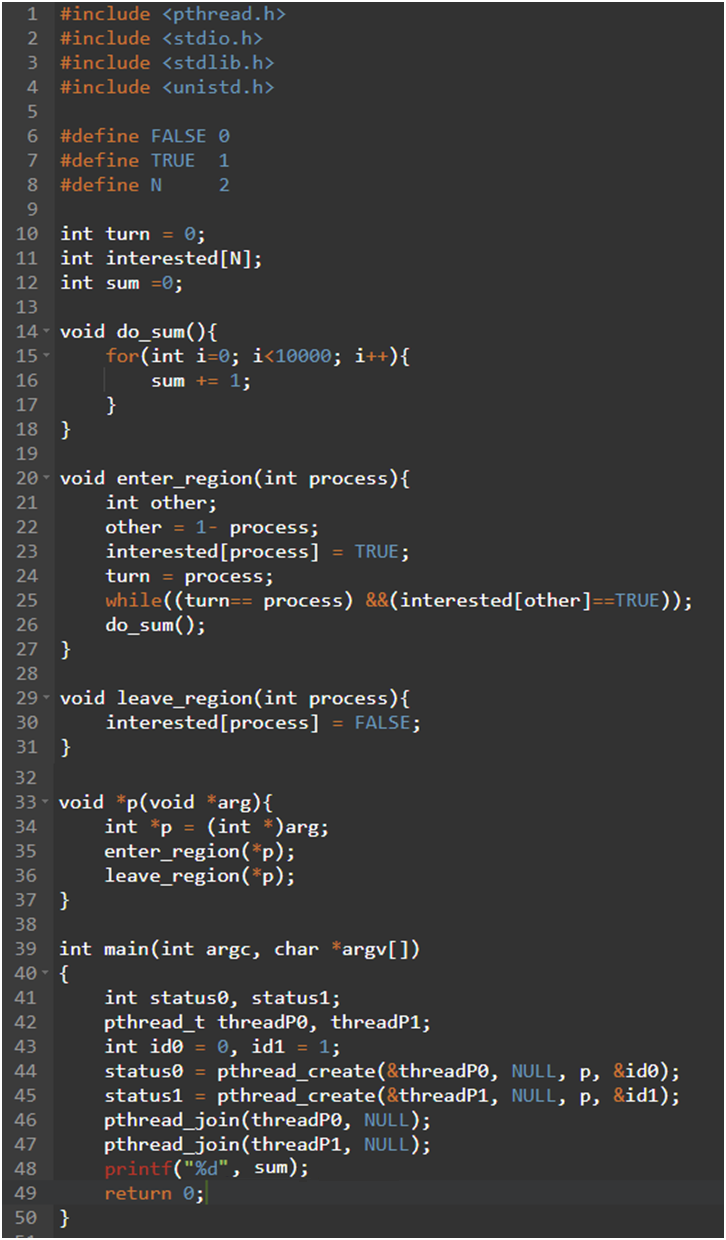
Yes because the lock prevents the threads from accessing the value at the same time. This means they always increment it properly and wait to use the value. No increments are lost this way

Hint: change {1..400} to {1..800} and run the script many times.

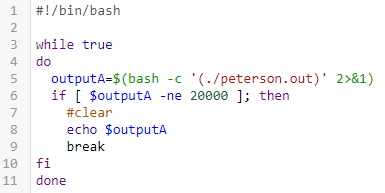
3) Repeat question 2 with the following C code.

*gcc peterson.c -pthread -o peterson.out*

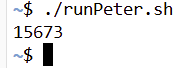
*peterson.c*



*runPeter.sh*



1. Attach a screenshot of your output.



1. Does the value of *sum* remain constant or not, explain?

No

Both threads run do\_sum() without proper locking so the sum will change