Tonya Shulkey (018145619)

Evan Marquez (018131930)

CECS 326 Sec 04

February 19, 2021

Part 1 non-sync:

Design: For the design I used the code given for the Simple thread and changed the param to void \* n and the return type to void \*. I created a main that would check if the command line arguments are integers. If it is not it would print a message and exit the program. I made an array for the pthread ids so that the thread number would print correctly while in the Simple thread function. I also created an array that would hold the addresses of the ids. Then I created a for loop that would loop to the number given in the command line and create pthreads sending them to the Simple thread function. After the loop I made another that would loop the same amount of times and join the pthreads so that it would wait on the threads to finish. You can see the design in the source code called project1.c.

In this first part of the multithreads without synchronization the threads are unable to come up with the correct final value after adding the value together. This is because there is nothing to stop each thread from iterating the shared value and so it will iterate the value it sees without checking if the shared value was updated recently or waiting for a thread to finish its update first. As you can see in figure 1.1 the threads see the same number in multiple places so it doesn’t look like its incrementing by 1. In figure 1.2 you can see that the final value is different for each thread because it ends the loop before all the threads are actually finished.

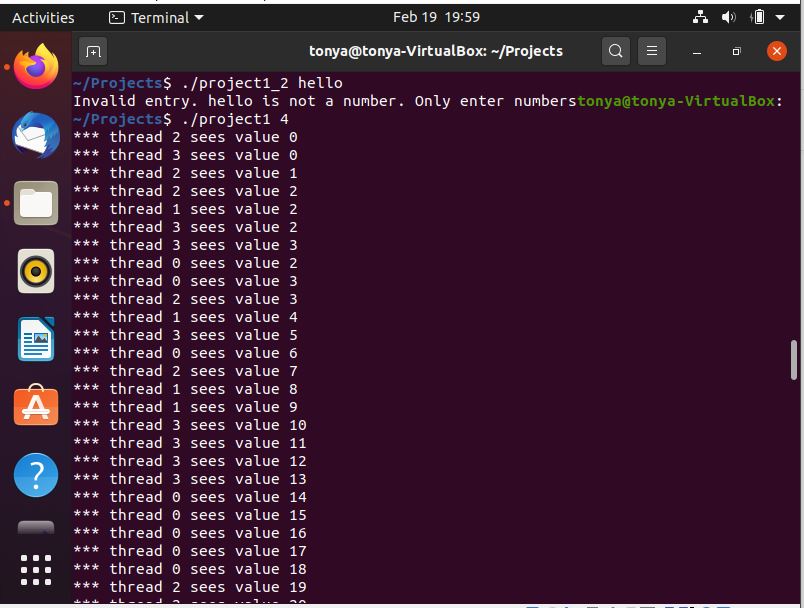


Figure 1.1

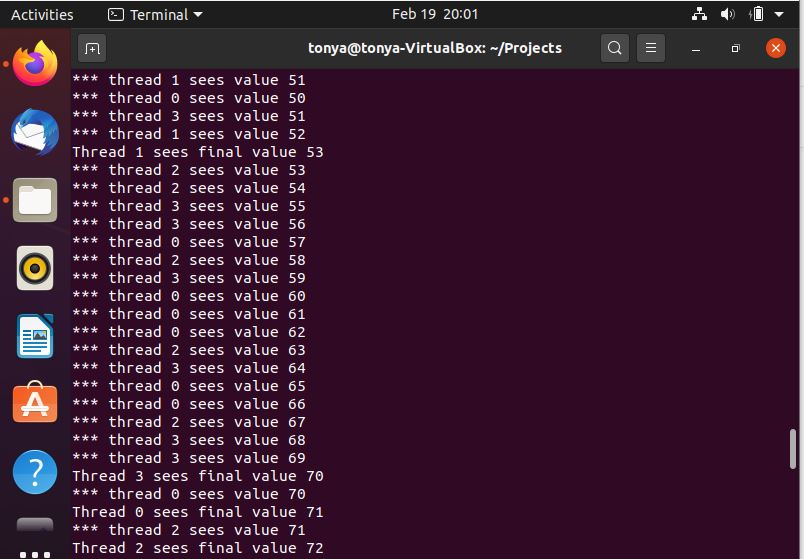


Figure 1.2

Part2 with sync:

Design: I designed the second part of the code to use mutex and barrier. I used mutex instead of sema because it was simpler to use and because we want the thread to wait for another thread to finish updating the shared value before it starts so that they are not working on the same thing at the same time and messing up the calculations. I just needed to initialize and destroy it in the main and use two lines of code in the Simple thread to lock and unlock the mutex. The code will look very similar to the first part but I included the define pthread\_sync so that it will run all the code this time. What is included in the second part is the mutex lock around the incrementing of the shared value so that threads will have to wait until the update is done to start their own. It also includes a barrier before printing the final value so that the threads would have to wait until calculations are over with all the threads. In the main I had to initialize the barrier and the mutex as well as destroy the mutex. You can see the code in project1\_2.c.

In the second part of the multithread with synchronization the threads are all able to come up with the same final value. They all line up at the end as well instead of finishing at different times through the calculation. There are no repeat values being seen while doing the calculation which means that the threads are waiting for the other threads to finish their updating of the shared value before it starts its own calculation. The values are able to increment the shared value by one only coming up with the correct final value. This is because of the mutex locks and the barrier which restrict the threads from continuing without checking.

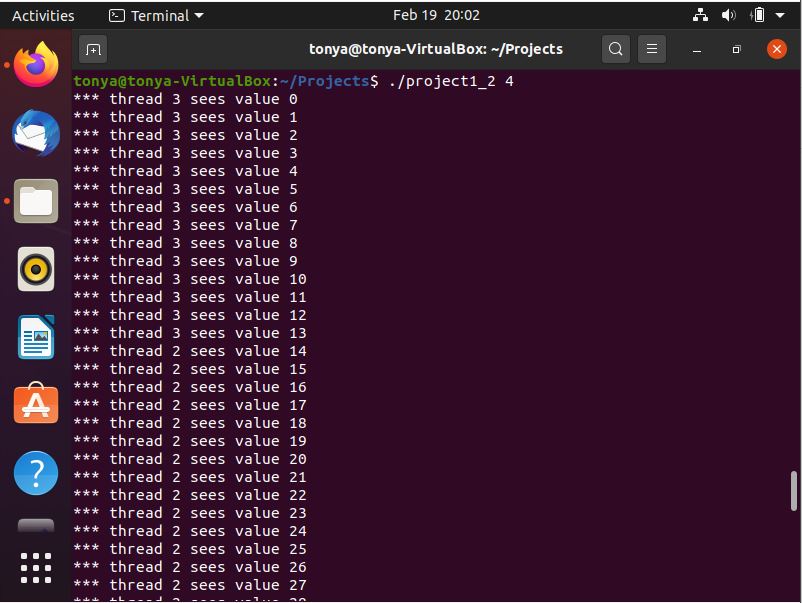


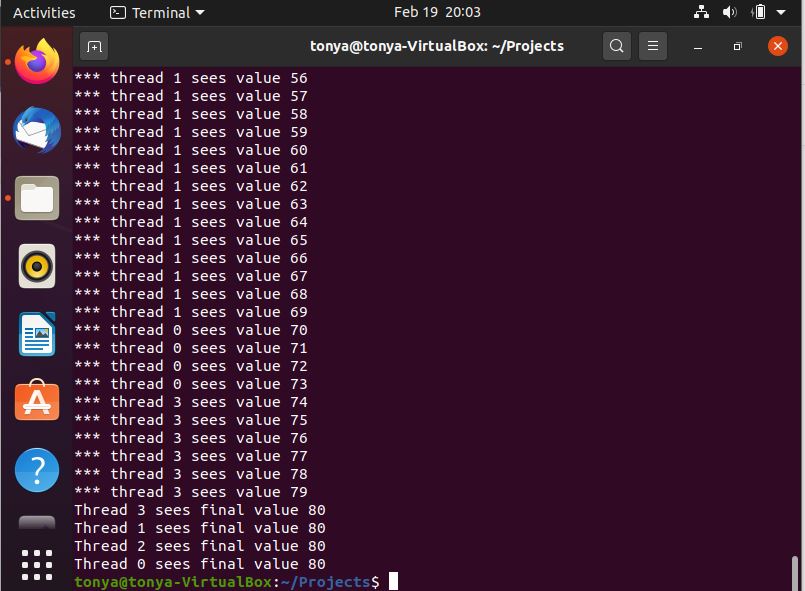
Figure 2.1

Figure 2.2

Individual contributions:

Tonya Shulkey: Worked on the code came up with the ideas and implemented it. Created the README file and the Makefile. Made a video of the code running. Wrote the Report.

Evan Marquez: Worked on the code came up with the ideas and implemented it.

We both worked on the code together while on a call.