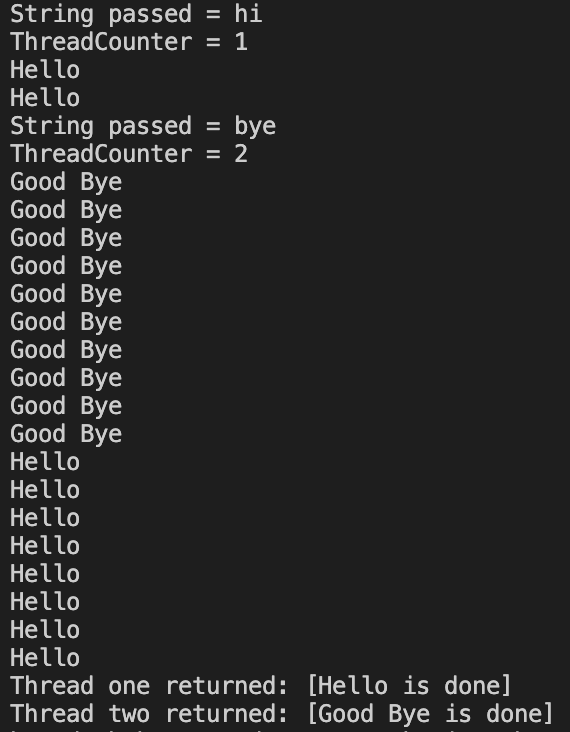
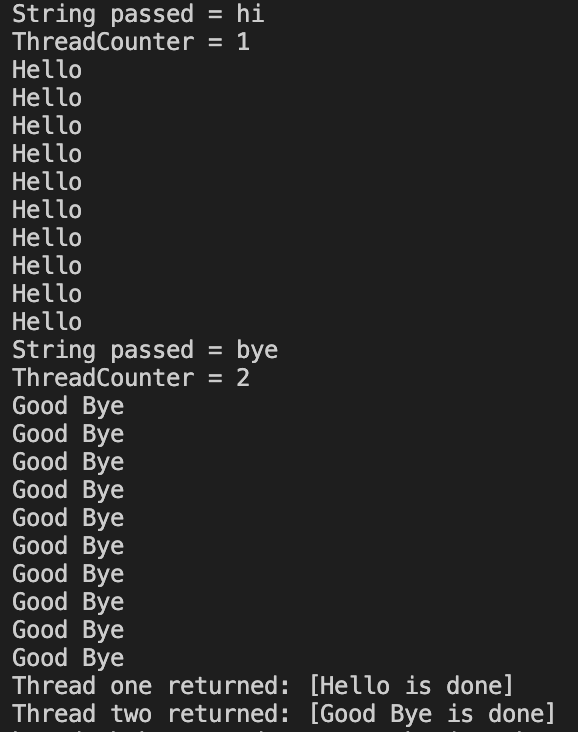
1. **Describe/explain your observations, i.e., what must have happened in the original unmodified program? (4 points)**

In the unmodified program, the program was hitting the “return 0” statement before the thread that was created had been executed. This meant that the “Thread version of Hello, world.\n” wasn’t able to print yet because the program finished before the thread ran. This can be proven by adding sleep(2) after creating the thread we were able to output the print statement inside doGreeting().

1. **What does sampleProgramTwo output? If you run it a repeated number of times does the output vary? Why? (6 points)**

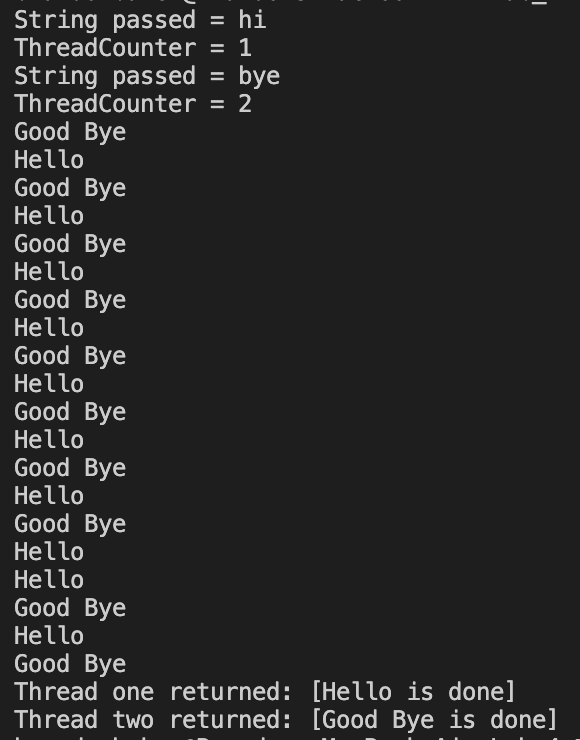
sampleProgramTwo will output the string passed/argument, the thread number created (counted through global variable), “Hello” (10 times), “Goodbye” (10 times), “Hello is done”, “Goodbye is done”, and the return values from waiting for thread1 and thread2. This program has varying output due to when the threads are created and executed in the scheduler.



In the screenshot on the left, the first thread is created and executed before the second thread exists. This allows the Hello to print without being interrupted by the print statements in the second thread for “bye”.On the right, the first thread starts execution, the second thread gets created in the middle of the first thread’s execution, and the first thread waits for the second to finish before continuing its sequence. The output varies because depending on when the scheduler decides to make the second thread, the first thread may get interrupted and wait for the second’s execution before continuing.

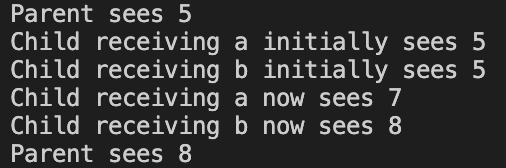
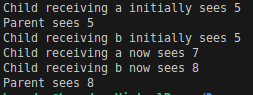
1. **Report your results again. Explain why they are different from the results seen in question 2. (4 points)**

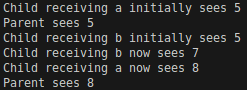
In the results after adding sleep(1) in the beginning of the for loop, thread1 is created first and both the print statements before the loop execute before the creation of thread2. Once thread2 is created, the loop executes and waits 1 second. Similarly, thread1 is executing at the same time and also runs into the loop which tells it to wait a second. By doing this, whatever thread the scheduler has running first will print, sleep 1 second, and the other thread will print, then sleep 1 second. This repeats until the loop is complete causing the thread to print the string passed every other line. Since we wait a second causing the print statement in each thread to print every other line (due to the other thread running at the same time), the results are different from question 2, where we have the threads run at their own speed when the scheduler decides to make and execute them. Below is the observed output from running sampleProgramTwo.



1. **Compile the sample program and run it multiple times (you may see some variables between runs). Choose one particular sample run. Describe, trace, and explain the output of the program. (6 points)**

The program starts with an initial sharedData variable of 5. When the main loop starts it creates two threads. Each of the threads run based on the scheduler. Therefore, the first print statement of Child receiving a/b may appear in a different order. Both will see the sharedData of 5 due to the sleep statement allowing time for the main loop to increment it by 1. Then the main thread waits until thread1 finishes. This allows for either child to finish execution first and increment the sharedData variable. Once thread1 finishes the main thread waits until thread2 finishes. This results in the threads printing 7 and 8 in order every time; however, the thread printing the shared data value can change depending on scheduling. Then once threads 1 and 2 join the parent prints the shared data which will always be 8. Below are sample outputs from what was described above.





1. **Explain in your own words how the thread-specific (not shared) data is communicated to the child threads. (4 points)**

When the main program runs and creates a child thread, it uses the fourth parameter in the pthread\_create method to pass an argument to the thread. This allows the child thread to access the value stored in the argument enhancing the communication between the main program and child thread. Once the child thread is executed, the parameter of the function it references will be the argument passed in the pthread\_create method. Once the thread returns (finishes execution), it can then return a value to the main function through the pthread\_join() method in the address of the second parameter. This allows data to be passed to and from (communicated) the child thread to the main program.

**Blocking Multi-threaded Server output**