Task 3

Part A: Foundation

**What is race condition?**

A race condition is when the functionality is dependent on the timing of uncontrollable events. It occurs in logic circuits, in multithreaded or distributed software

**Why race condition is difficult to reproduce and debug?**

It is difficult to reproduce and debug because it is an uncontrollable event.

**How can if be fixed? Provide an example from your project\_a3(see spm2.c)**

In spmd2.c the variable id is declared inside the pragma which makes it unique. This makes it so each thread has its own private copy of the variables.

**Summaries the Parallel Programming Patterns section in the “Introduction to Parallel Computing\_3.pdf”**

The two main categories of patterns for parallel programming are Strategies and Concurrent Execution Mechanisms. In strategies, the programmer asks the two questions: what algorithmic strategies and what implementation strategies to use. . Parallel algorithmic strategies are the choices to be made about what tasks can be executed concurrently using multiple cores processing units. Parallel programs can use several patterns of implementation strategies. Some of those affect the overall structure of the program, and others care about how the data can be computed by multiple processing units. The concurrent execution patterns can be grouped into two major categories: Process/Thread control patterns (which control at runtime how the processing units run), and Coordination patterns. Most software uses one of the two coordination patterns: message passing between concurrent processes, and mutual exclusion between threads executing concurrently

**In the section “Categorizing Patterns” in the “Introduction to Parallel Computer\_3.pdf” compare the following:**

**Collective synchronization(barrier) vs Collective communication(reduction)**

The barrier is the point where all the threads must stop, and the task cannot proceed until all the other threads reach that certain point. Reduction on the other hand lets you specify thread-private variables that are subject to a reduction operation at the end of the parallel region.

**Master-worker vs fork join**

The master worker model is when the work is divided. There is a block of code that is the master and the rest of the blocks are workers. Inside the same thread, they do different operations then combine the results of the thread. The fork join, the fork assigns the child thread with some piece of work, the child thread performs the work, then call the join to exit.

**Where can we find parallelism in programming?**

It can be found in the task view, the Data view and the Resource view. In task view it can be found within a Statement level or Block, Loop, Routine or Process level.

**What is dependency and what are its types (provide one example of each)**

A dependency is when you have to wait for one action to be completed before the other is started. There are three types of dependency: true dependence, output dependence, and anti-dependence.

Examples

True dependence:

S1: a = 1;

S2: b = a; // S2 depends on S1

Output dependence:

S1: a = f(x);

S2: a = b; // S2 depends on S1

Anti-dependence:

S1: a = b; // S1 depends on S2

S2: b = 1;

**When a statement is dependent and when it is independent? (Provide examples)**

A statement is dependent when it relies upon other statements to make it work

Int a = 3

Int b = a + 10

A statement is considered independent when its output is not affected by the other statements

Int a = 3

Int b = 4

**When can two statements be executed in parallel?**

Two statements can be executed in parallel if neither is dependent on the other in order to execute.

**How can dependency be removed?**

It can be removed by rearranging the statements of the program to remove the dependency

**How do we compute dependency for the following 2 loops and what are the type/s of dependency?**

The dependency can be computed by looking at the IN(memory variable) and the OUT(memory variables that can be changed by S) sets of a given statement S

for(i=0; I < 100; i++)

S1: a[i] =i

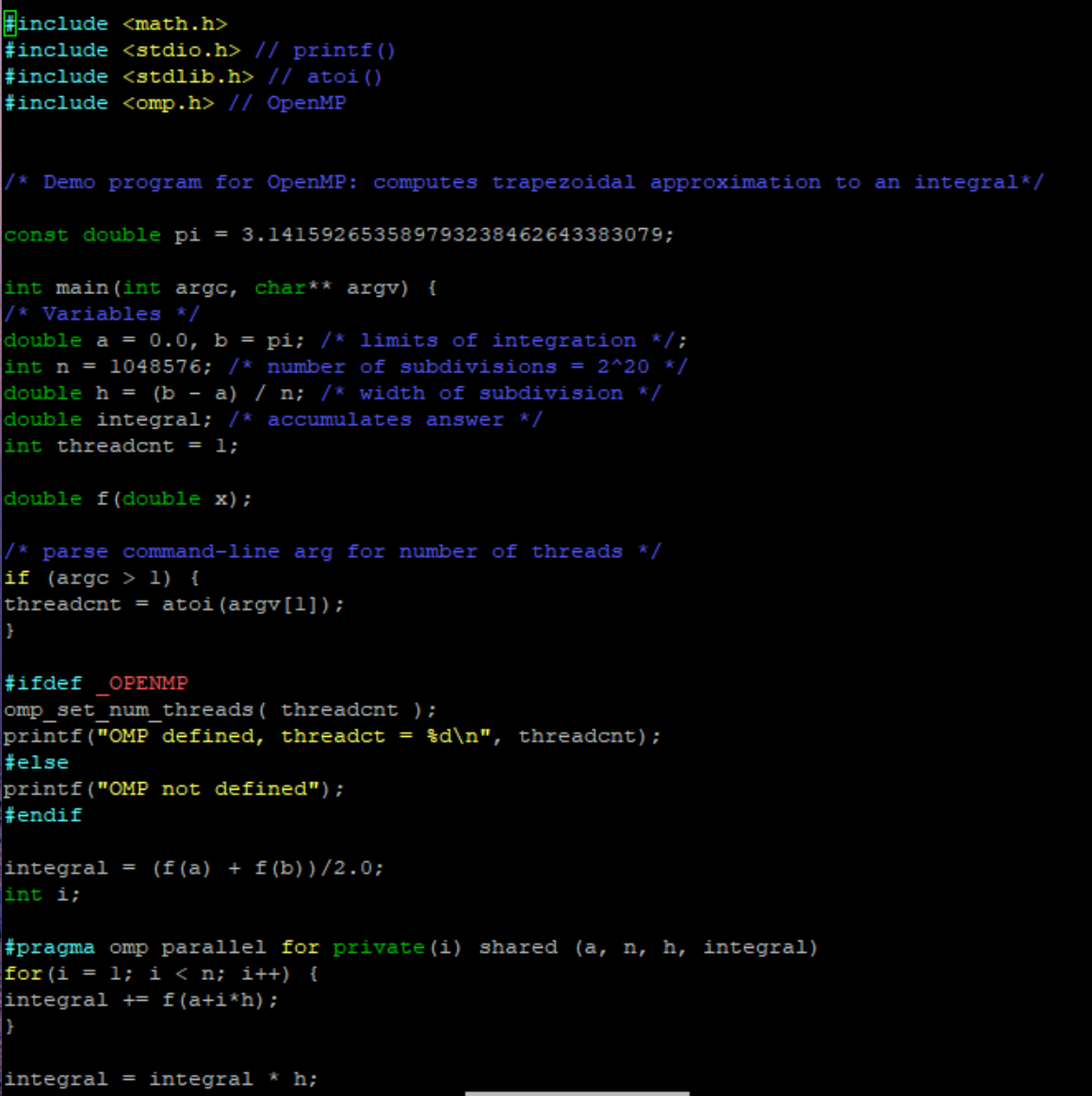
This is an example of true dependency because it is reliant upon another value to dictate the output

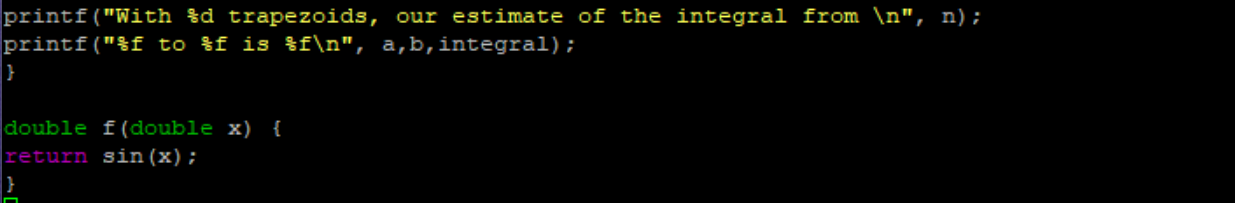
for(i=0; i<100, i++){

s1: a[i] =i

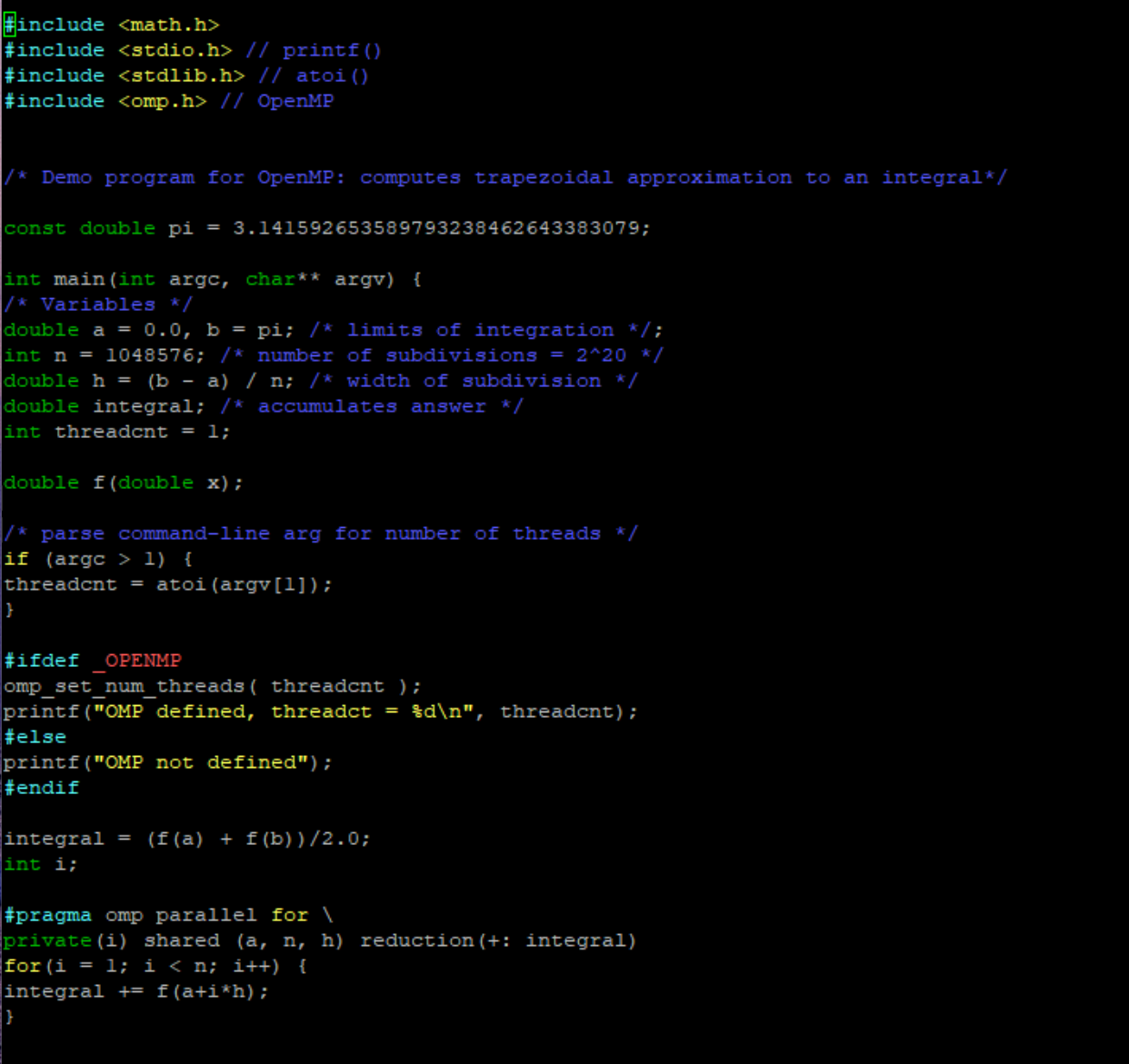
s2: b[i] =2\*i

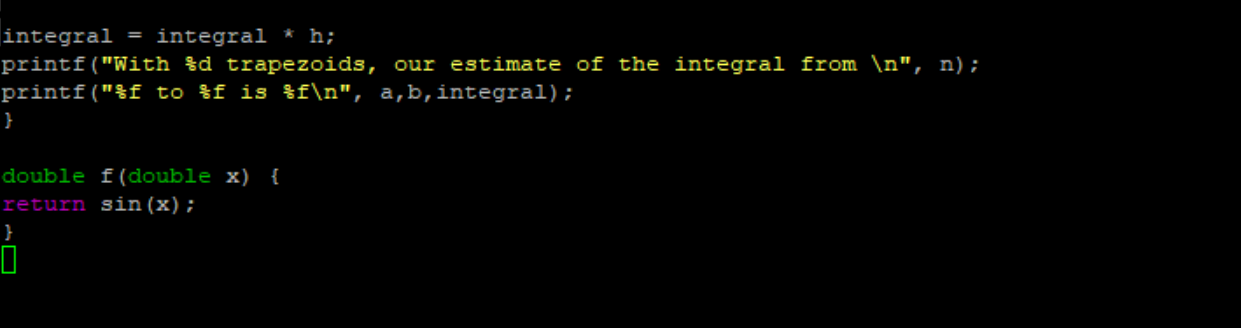
This is also an example of true dependency



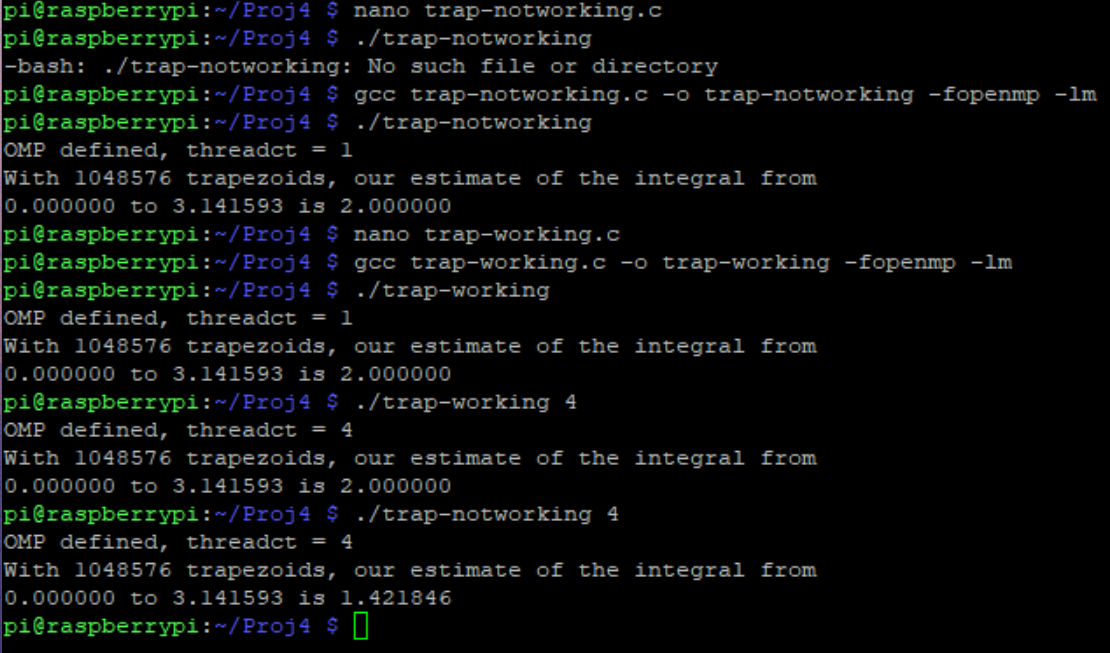


*Figure 1 -Trap-notworking.c*

**

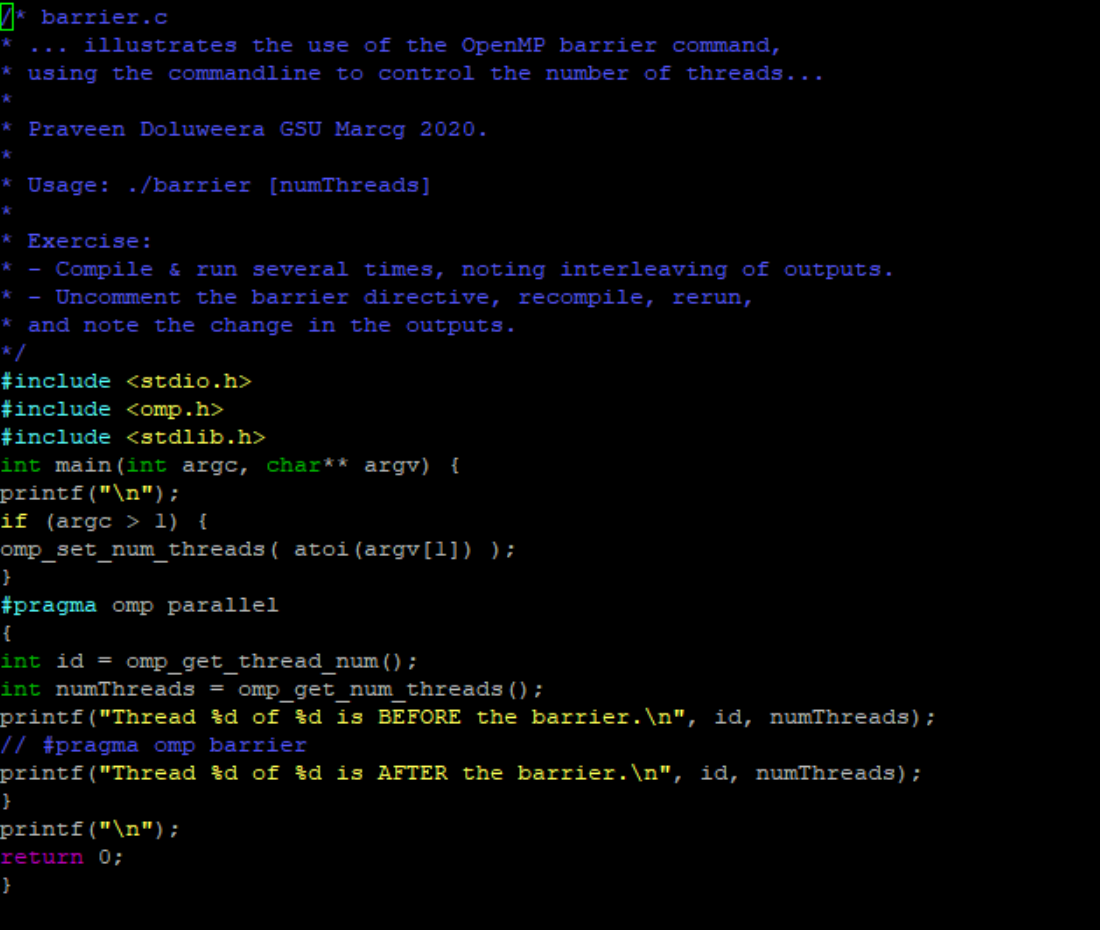
**

*Figure 2- trap-working.c*

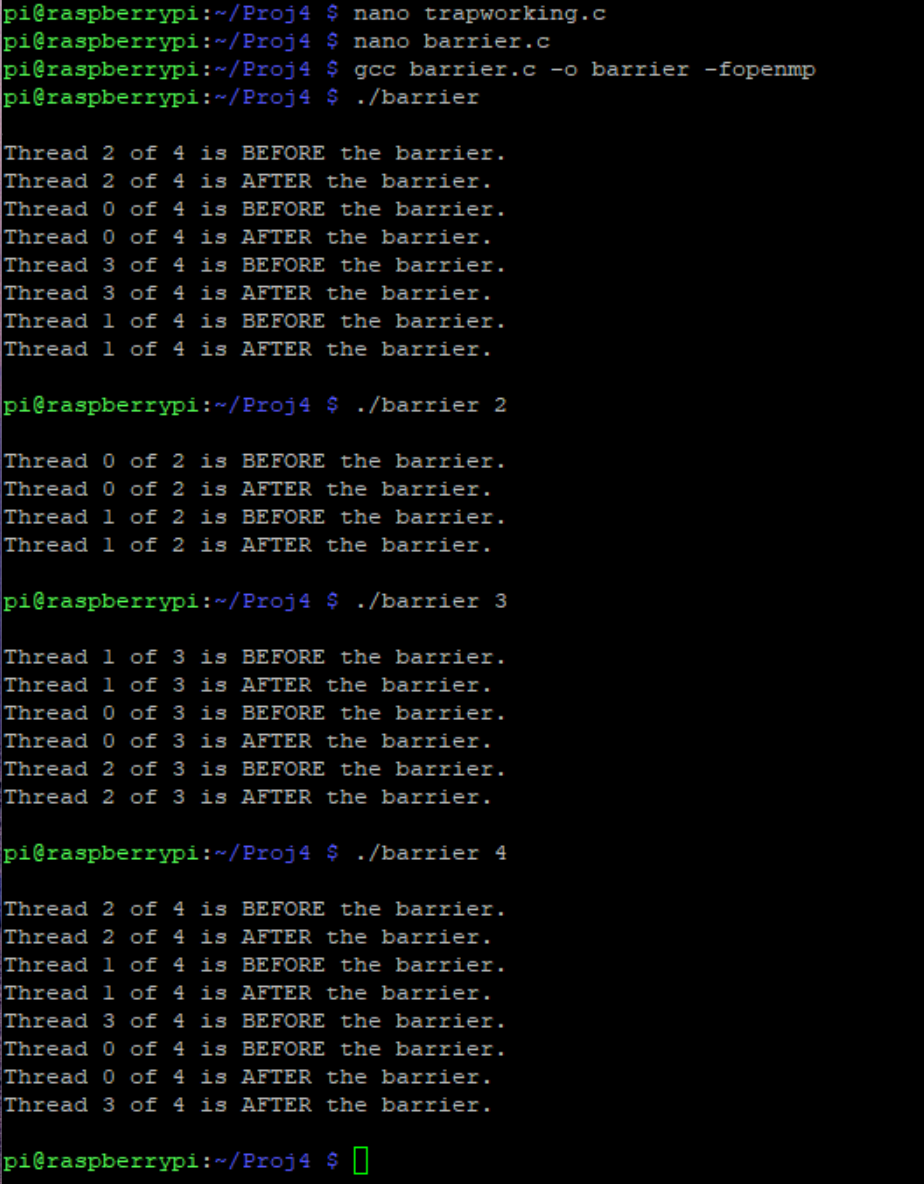


*Figure 3 – Running both files*

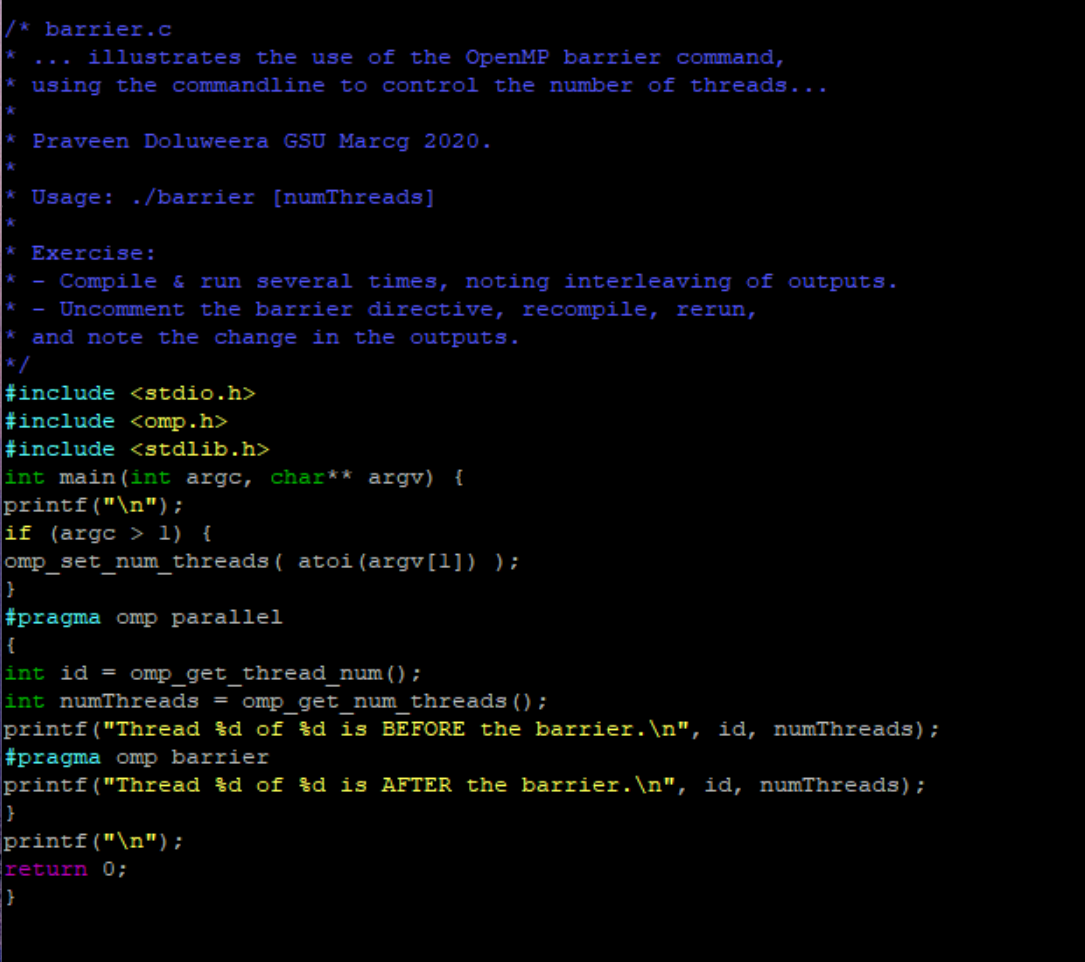
I then copied and ran the barrier code as shown in Figure 4 and ran it as shown in Figure 5

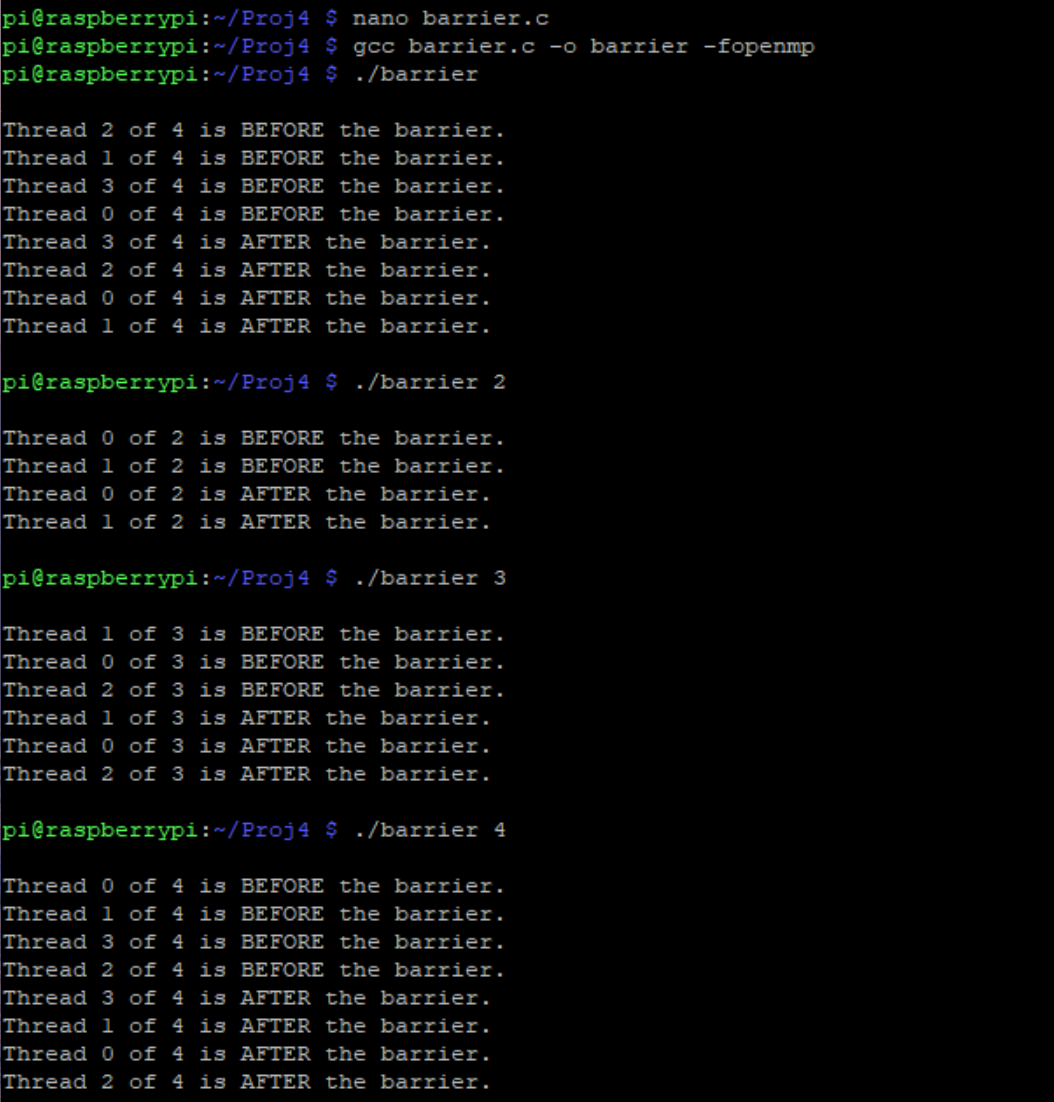


*Figure 4*

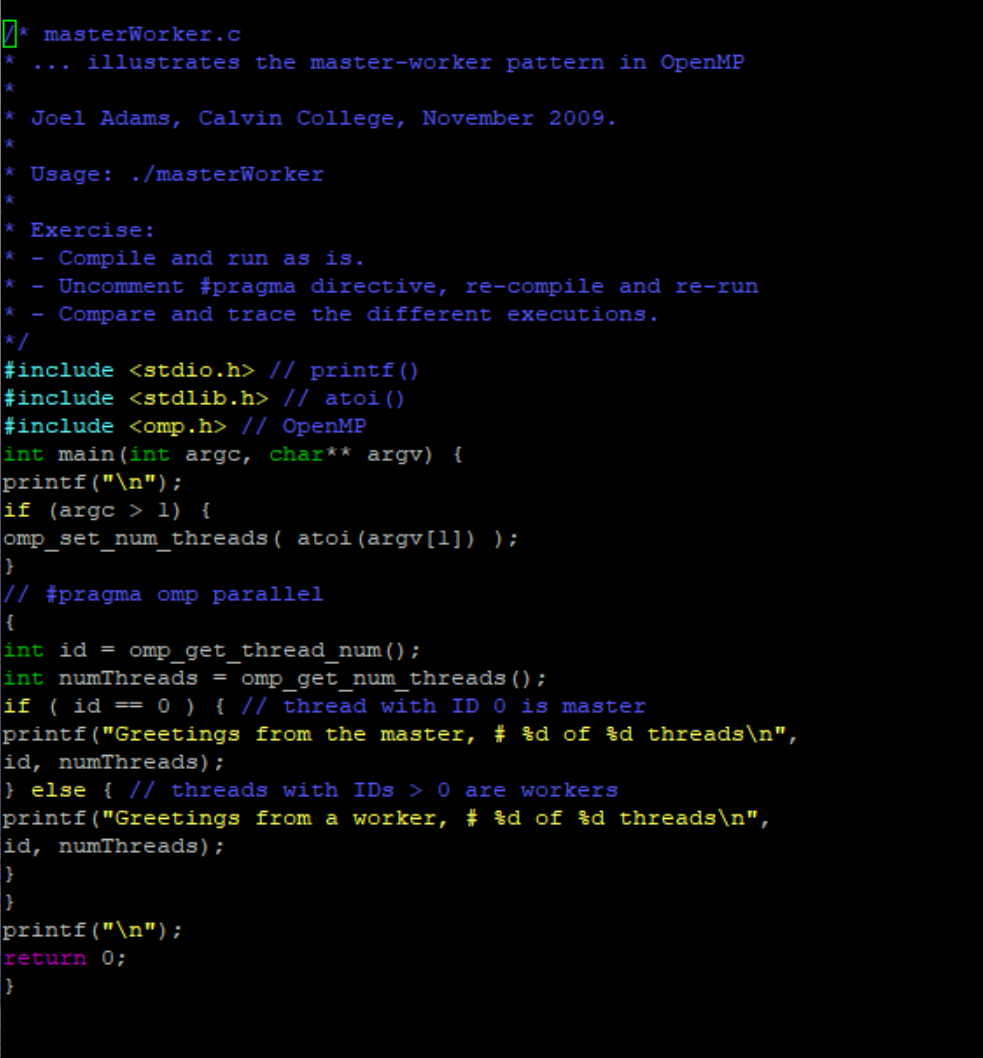


*Figure 5*

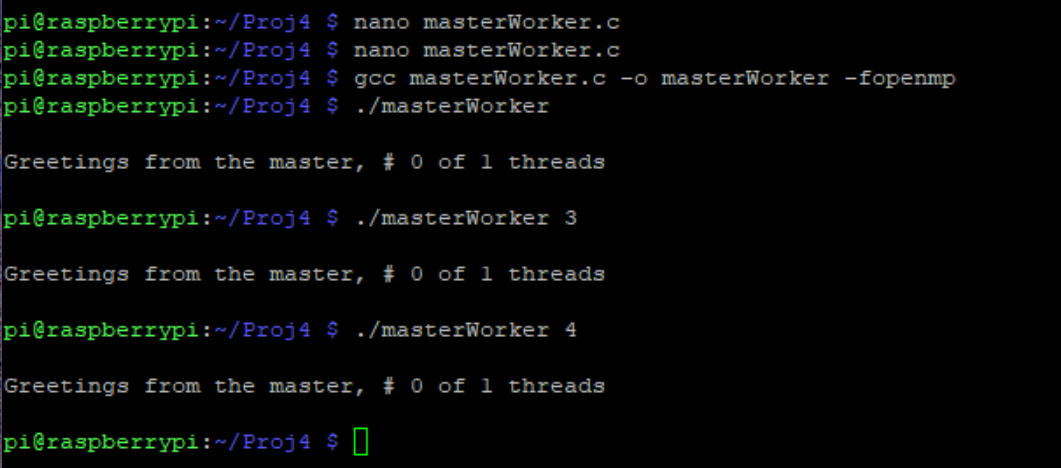
*Figure 6*



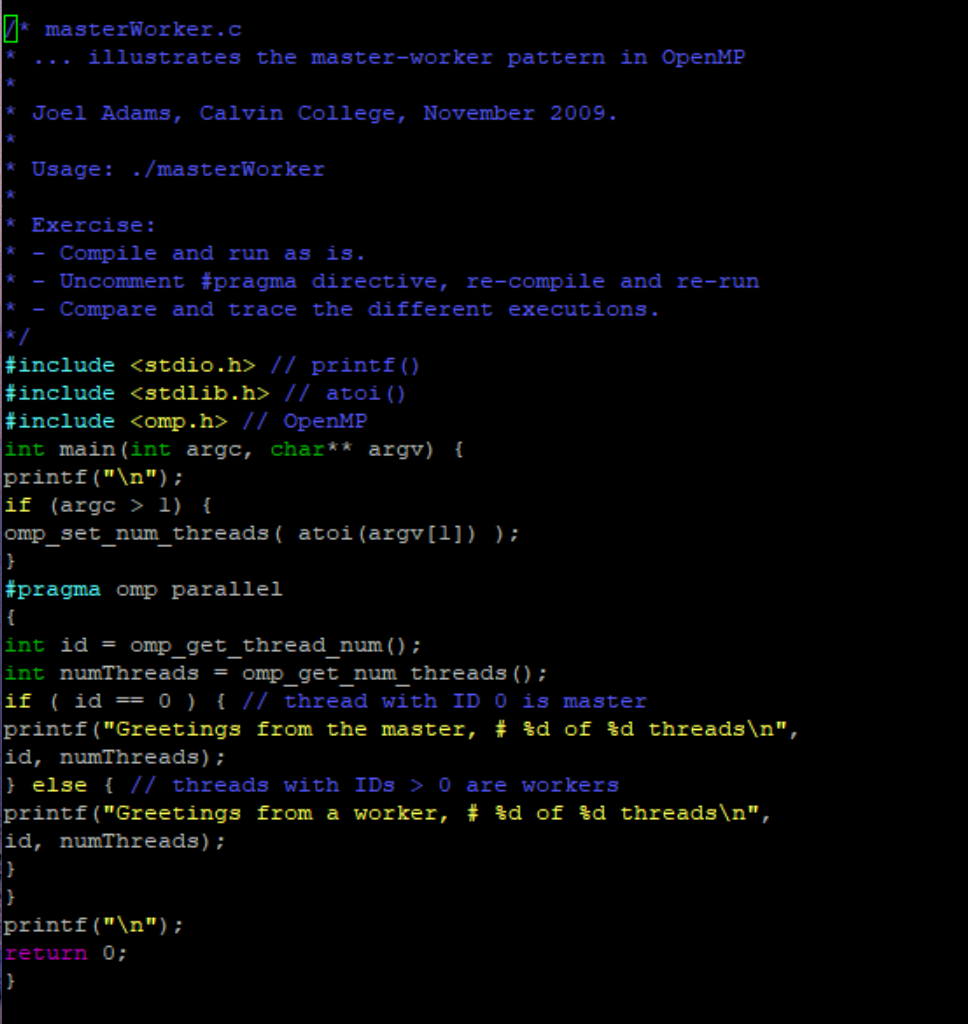
*Figure 7*

**

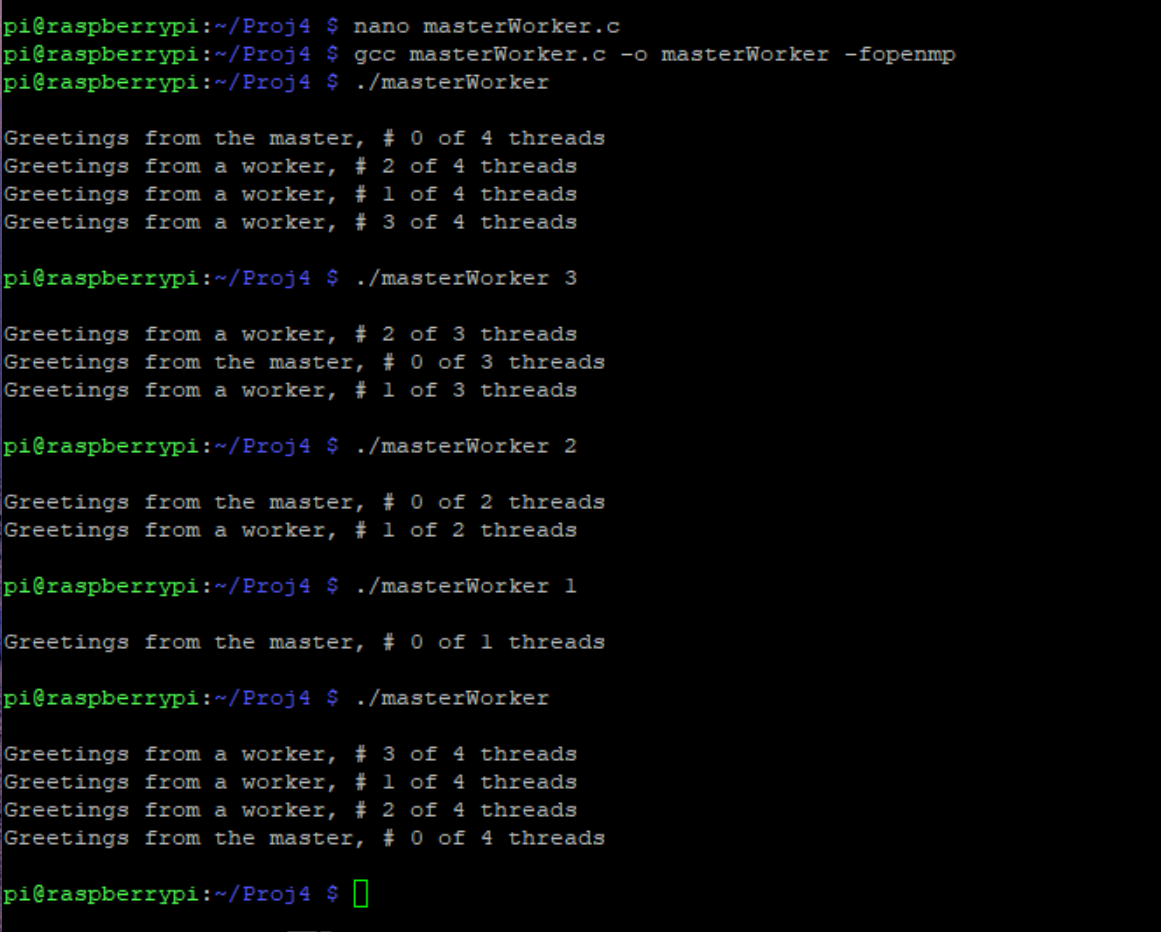
*Figure 8*

**

*Figure 9*

**

*Figure 10*

**

*Figure 11*