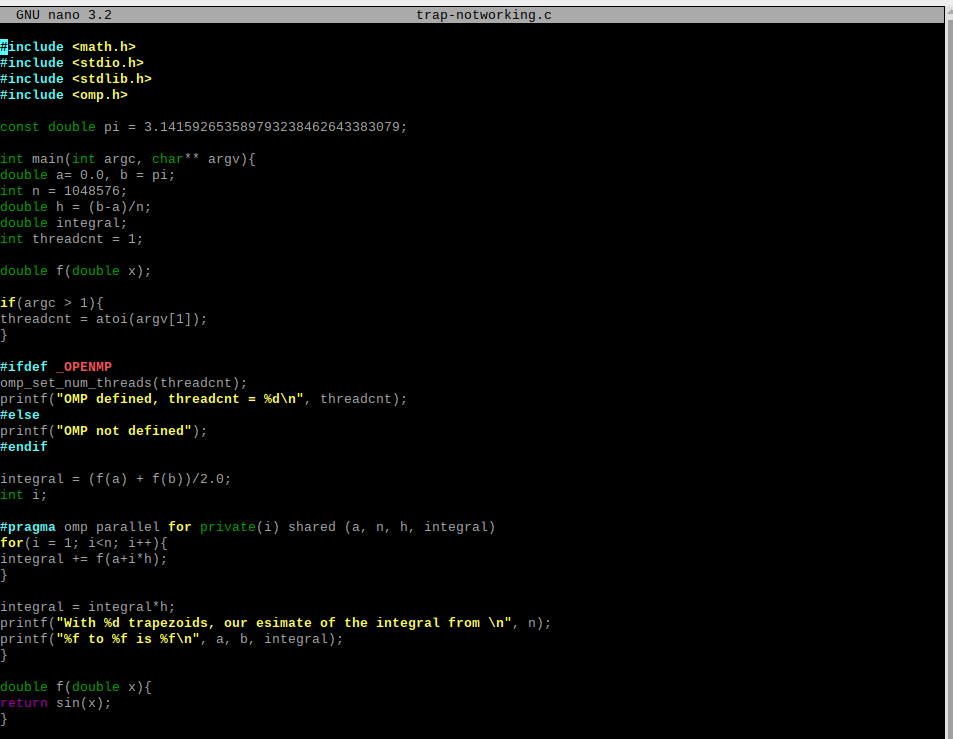
Tony Derado

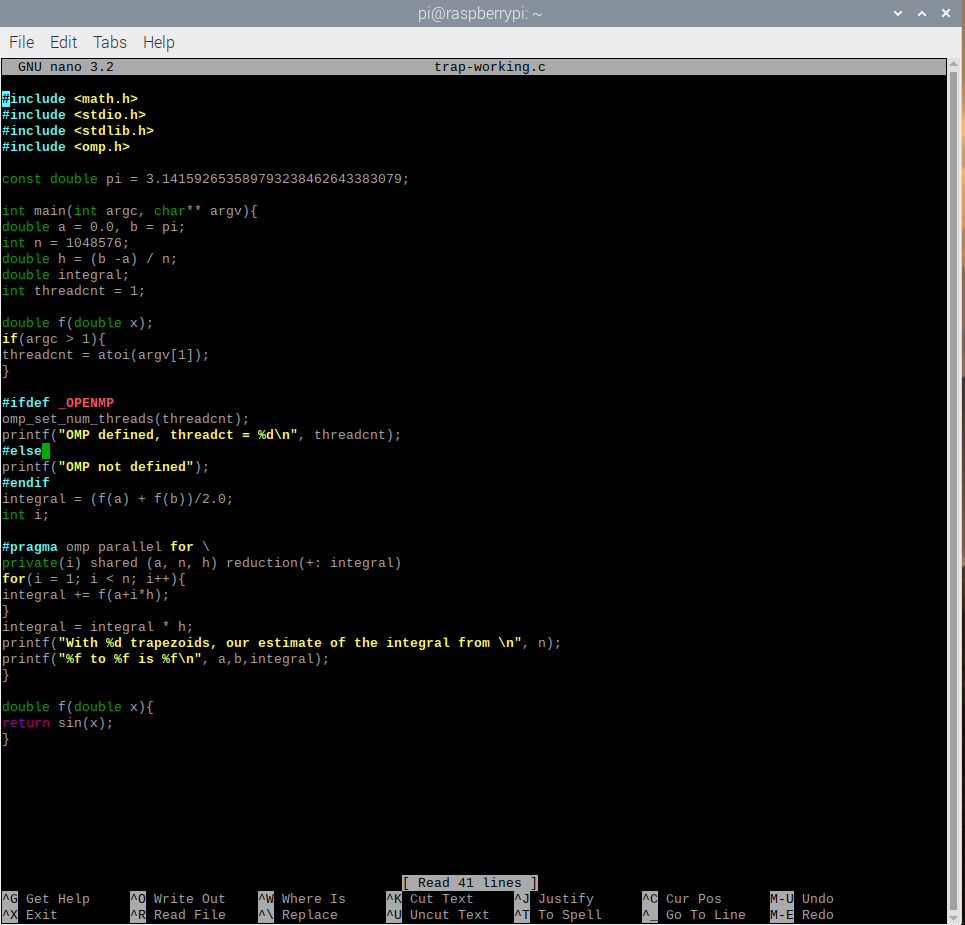
Task 3B:

The first part of parallel programming task four, we created both files and executables for trap-notworking and trap-working.

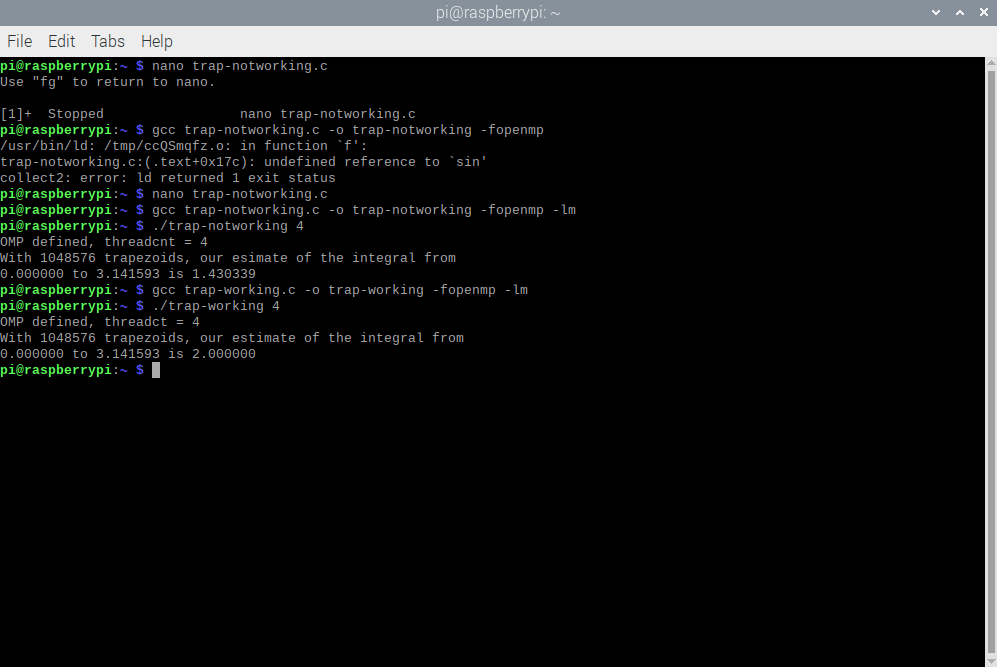
(The creation of the trap-notworking class)



(The creation of the trap-working class)



Then after their creation we created the executables for each one, however due to the use of the sin method, we had to include -lm at the end of the gcc creation command.

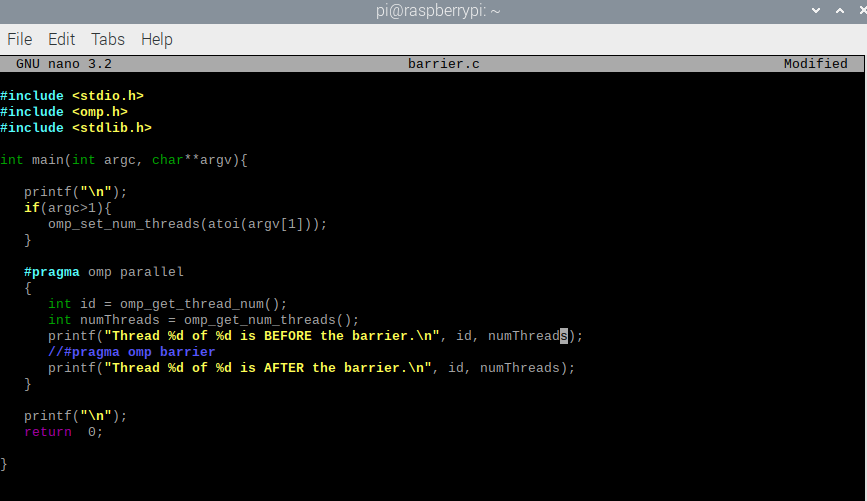


From the output of the programs we can see indeed that trap-notworking fails to receive the correct number, the difference between #pragma omp parallel for with \ and without, is that using it will save the last iterations data for use outside the #pragmas… clause.

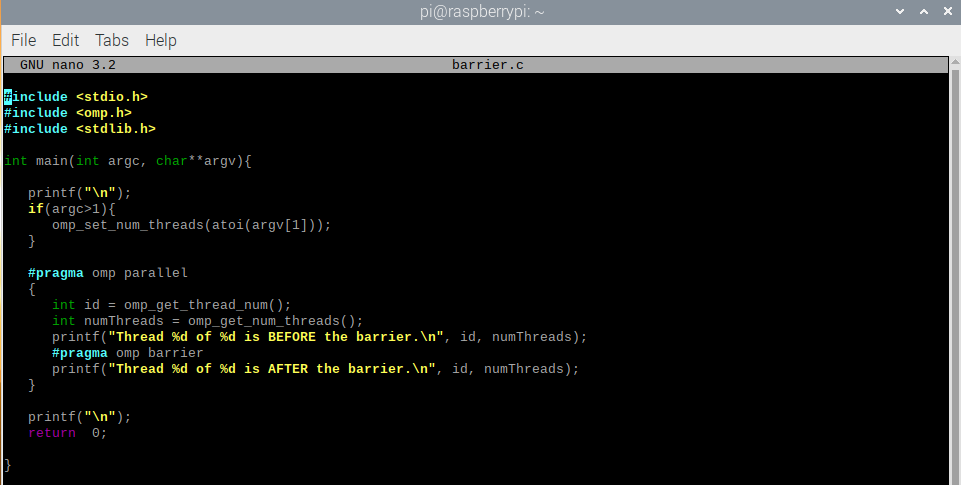
Barrier:

The barrier class had to tests to run, using it without the included #pragma omp barrier and using it with it.

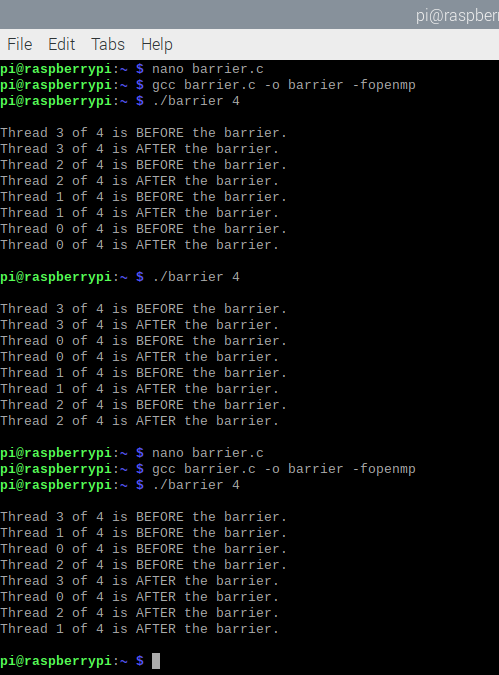
(without using the barrier)



(using barrier)



(output)

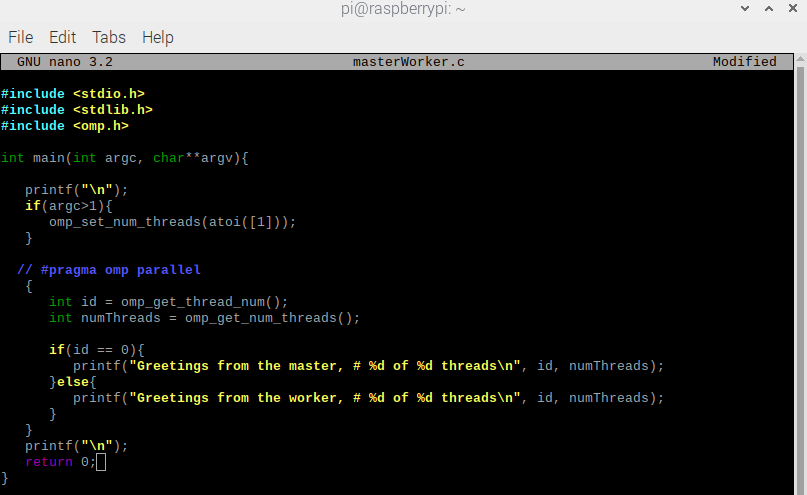


The threads without the barrier printed out their before and after statements in the order they finished. When running with the barrier however, all the threads stopped once they got to the barrier in the code until all other threads caught up.

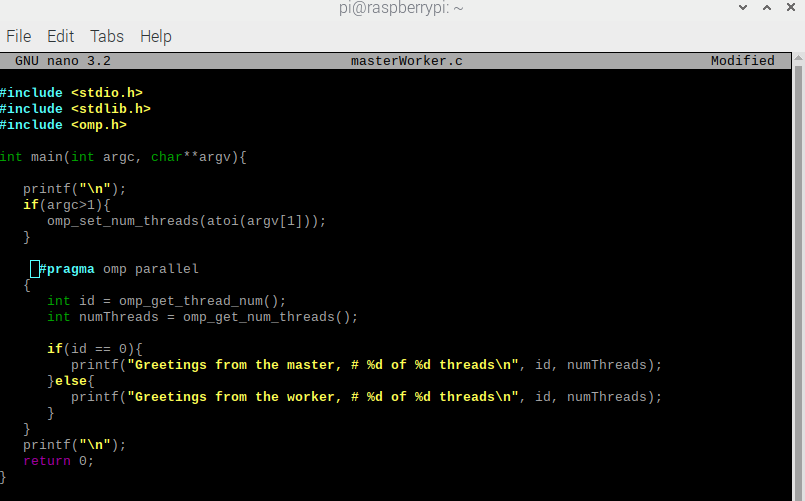
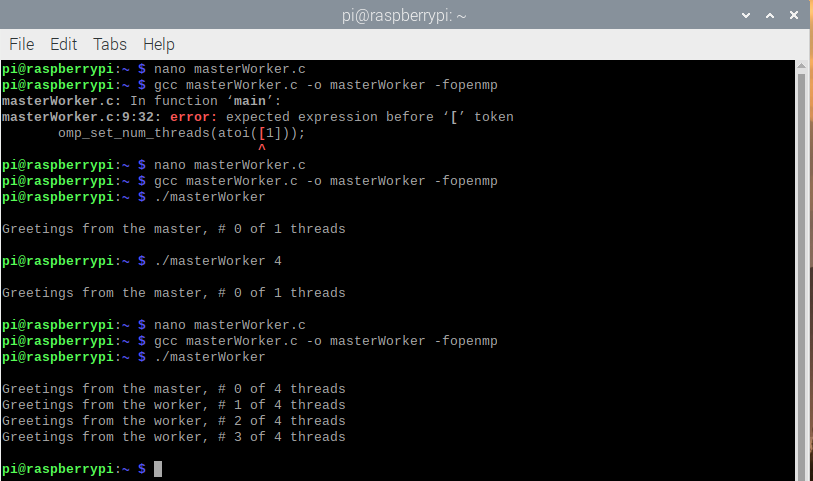
WorkerMaster:

In the workerMaster program, we created a program and ran it with and without the use of #pragma omp parallel.

(workerMaster without pragma)



(workerMaster with pragma)

(output)

After creating the excutable and running the code, we saw that without the use of parallel, the code would of course only output one line, the master, as the code never splits or forks into separate threads so only thread 0, the master, runs. Once we use the parallel the code output the separate threads are created thus allowing for the worker threads to also reply.