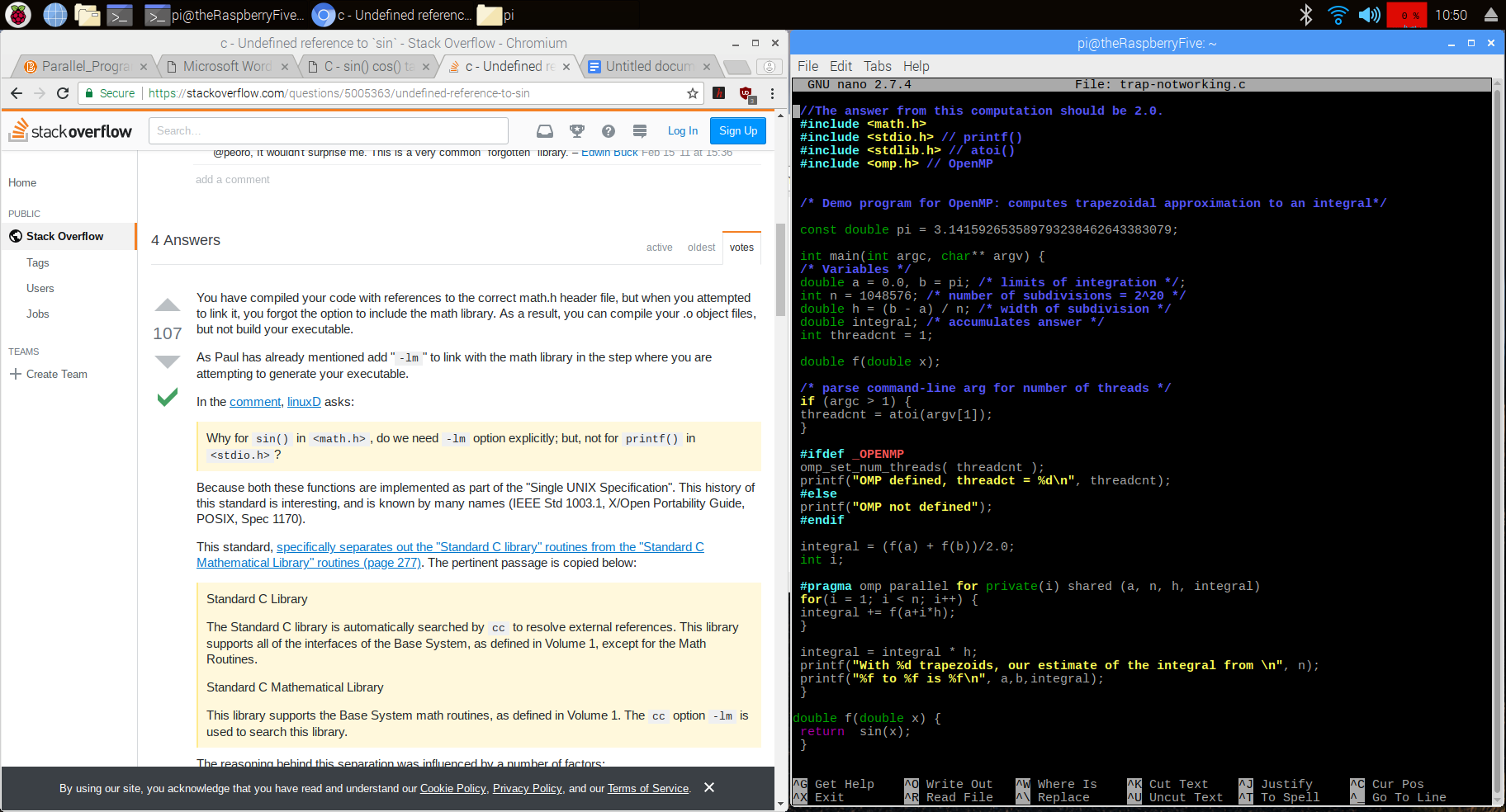
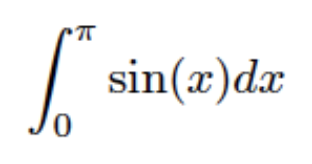
**Assignment 4 Parallel Programming**

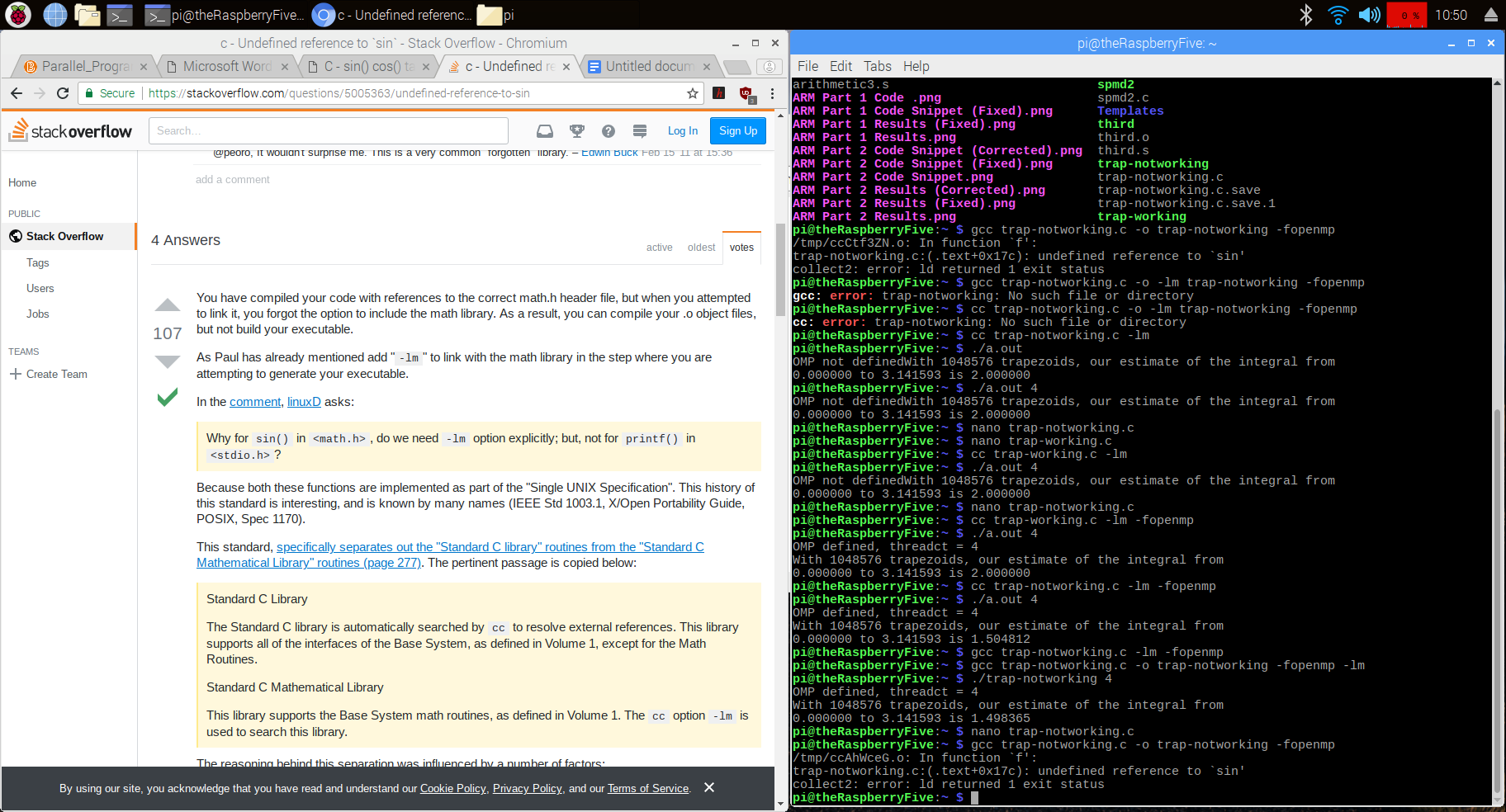
**2: Integration Using the Trapezoidal Rule**



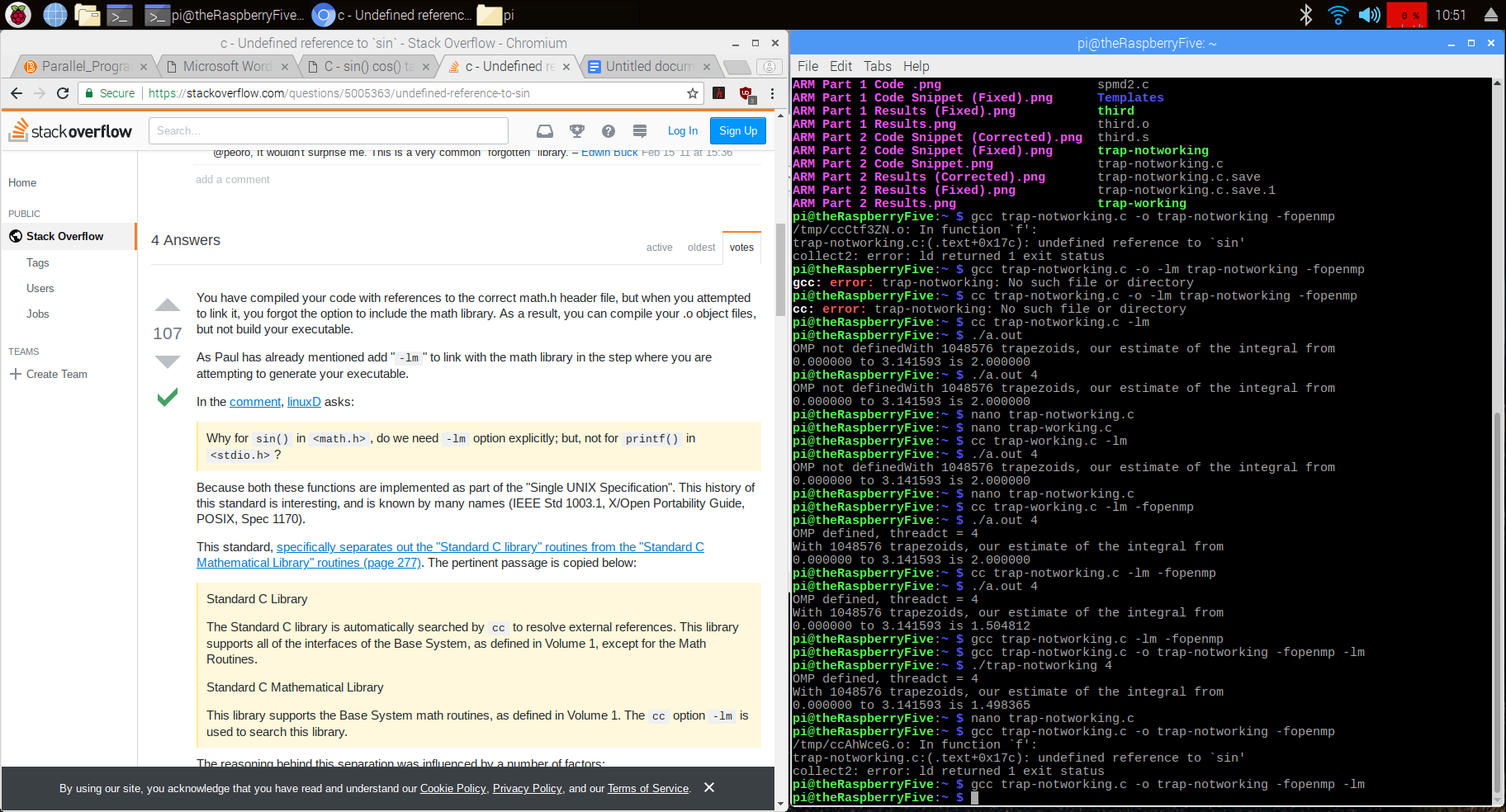
This is trap-notworking.c,

which attempts to compute an approximation of  using parallel processing.

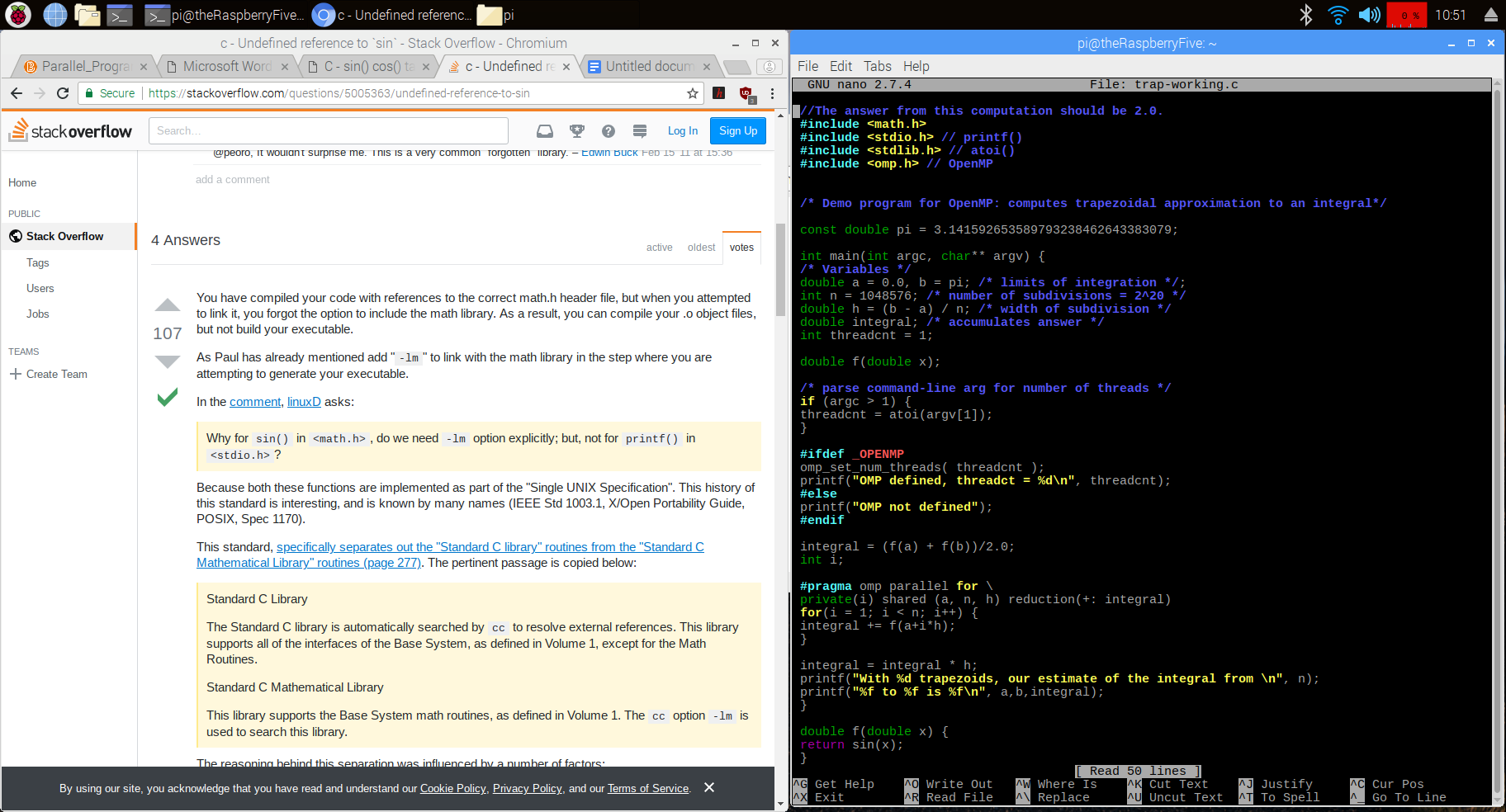
The following compile command fails because “sin(x)” is not recognized,



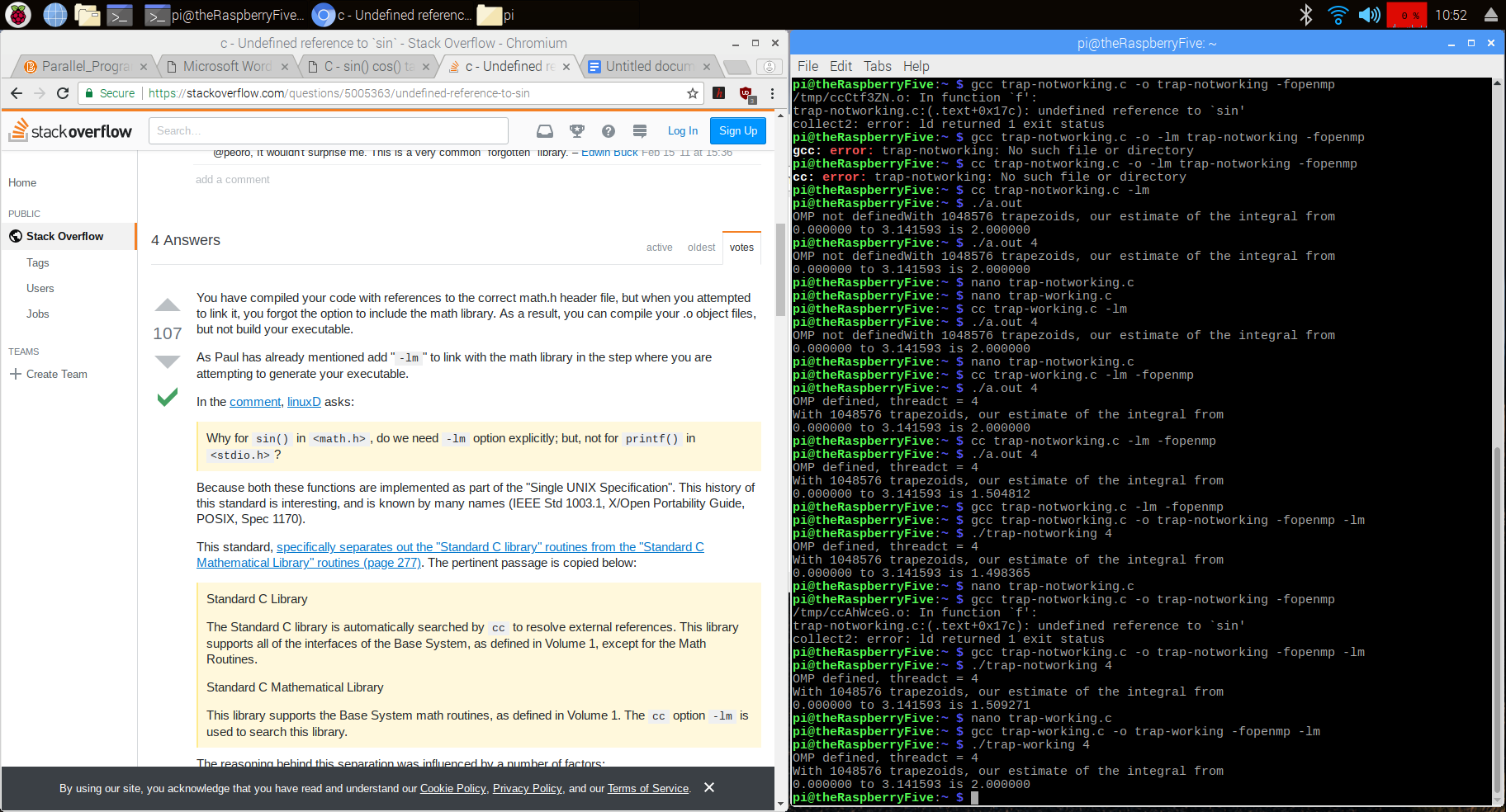
but adding “-lm” to include the math library when compiling solves the problem and allows for trap-notworking.c to be compiled successfully.



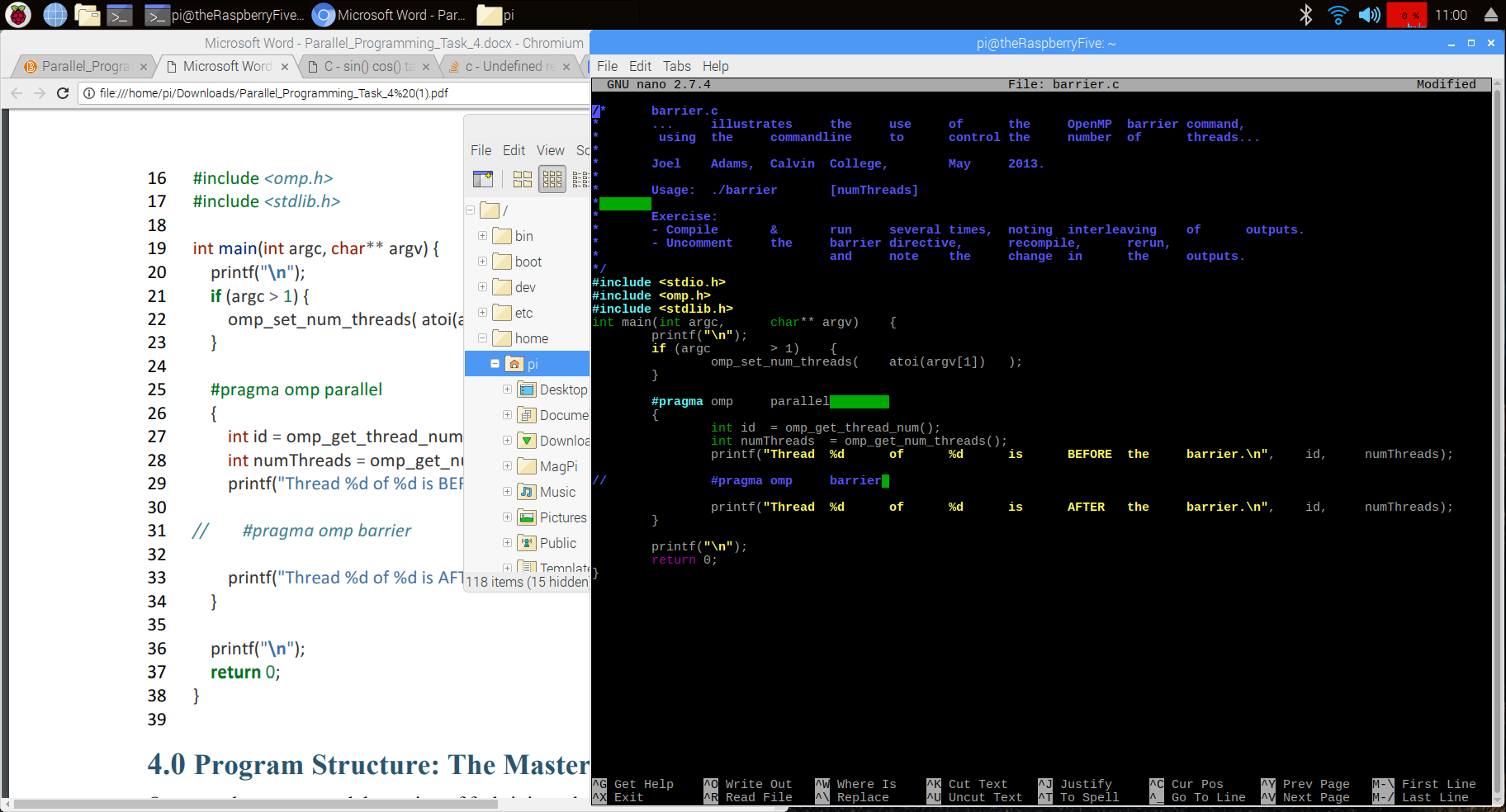
However, the program still gives an incorrect result. We need to add a reduction clause for the accumulator variable (integral) for the summation across parallel threads to work correctly.

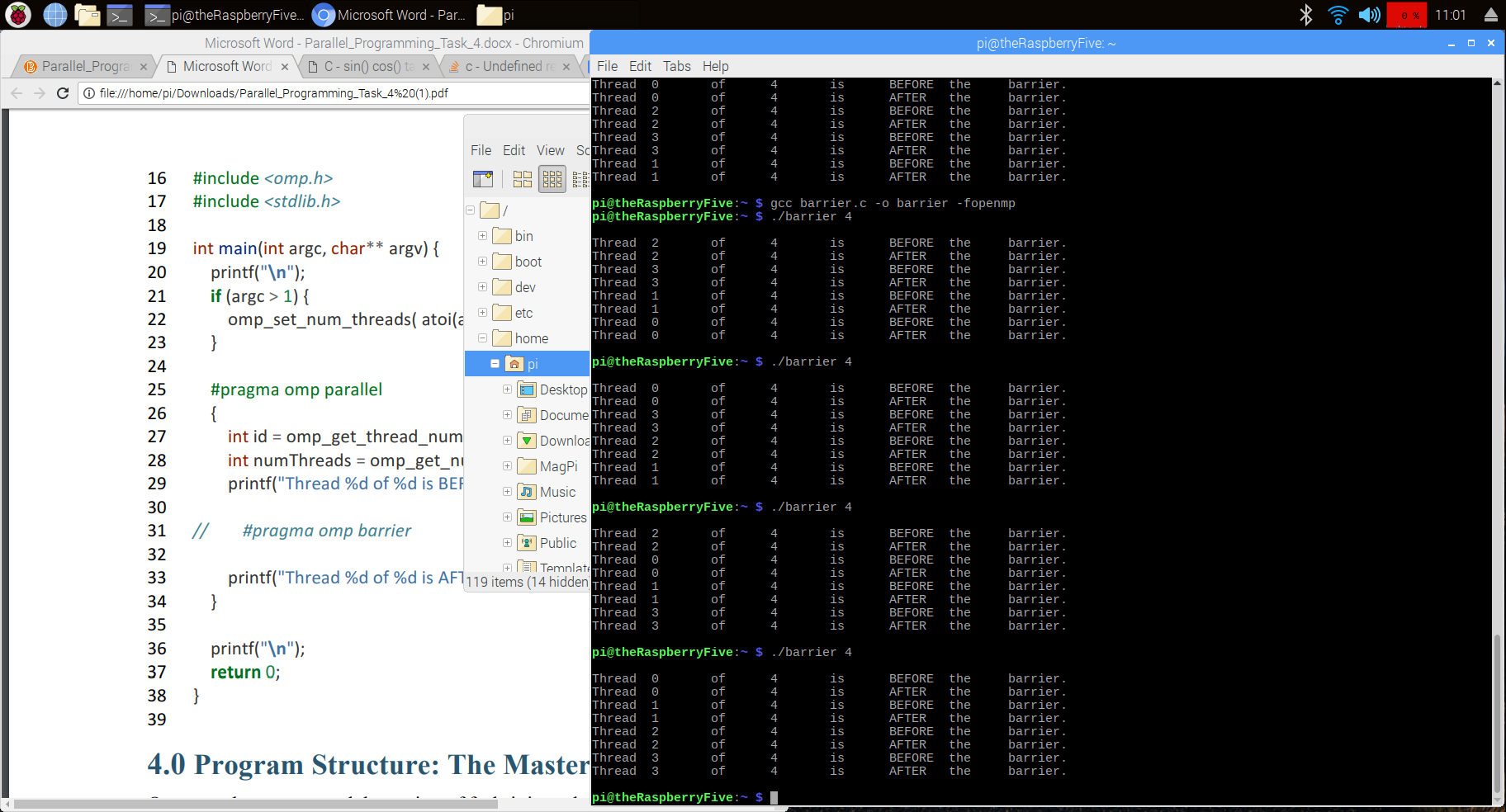


Now the program calculates the correct result.



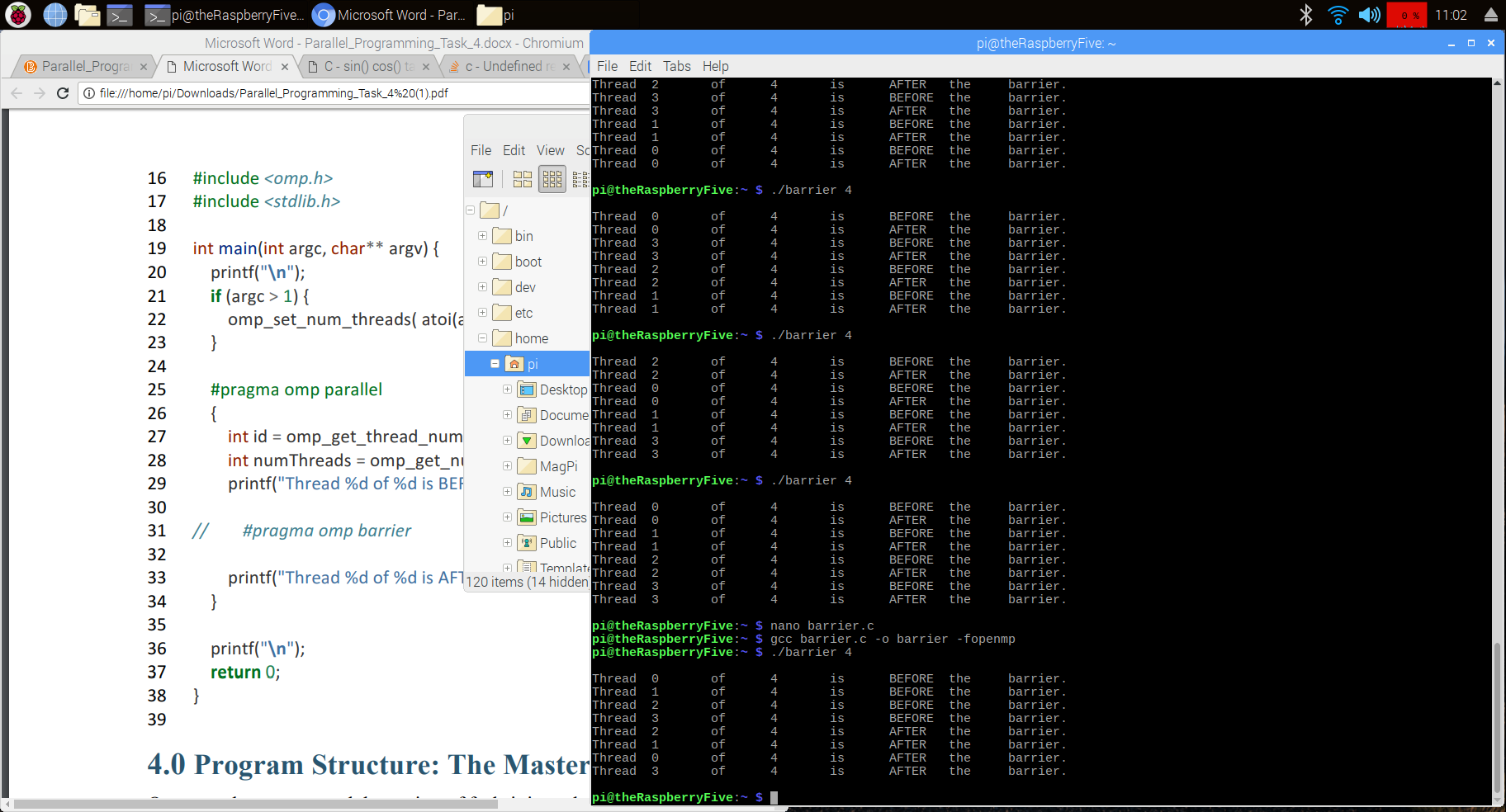
**3: Synchronization with a Barrier**

This program uses the barrier pattern (commented out in this image) to stop threads from continuing until they all reach the same point.

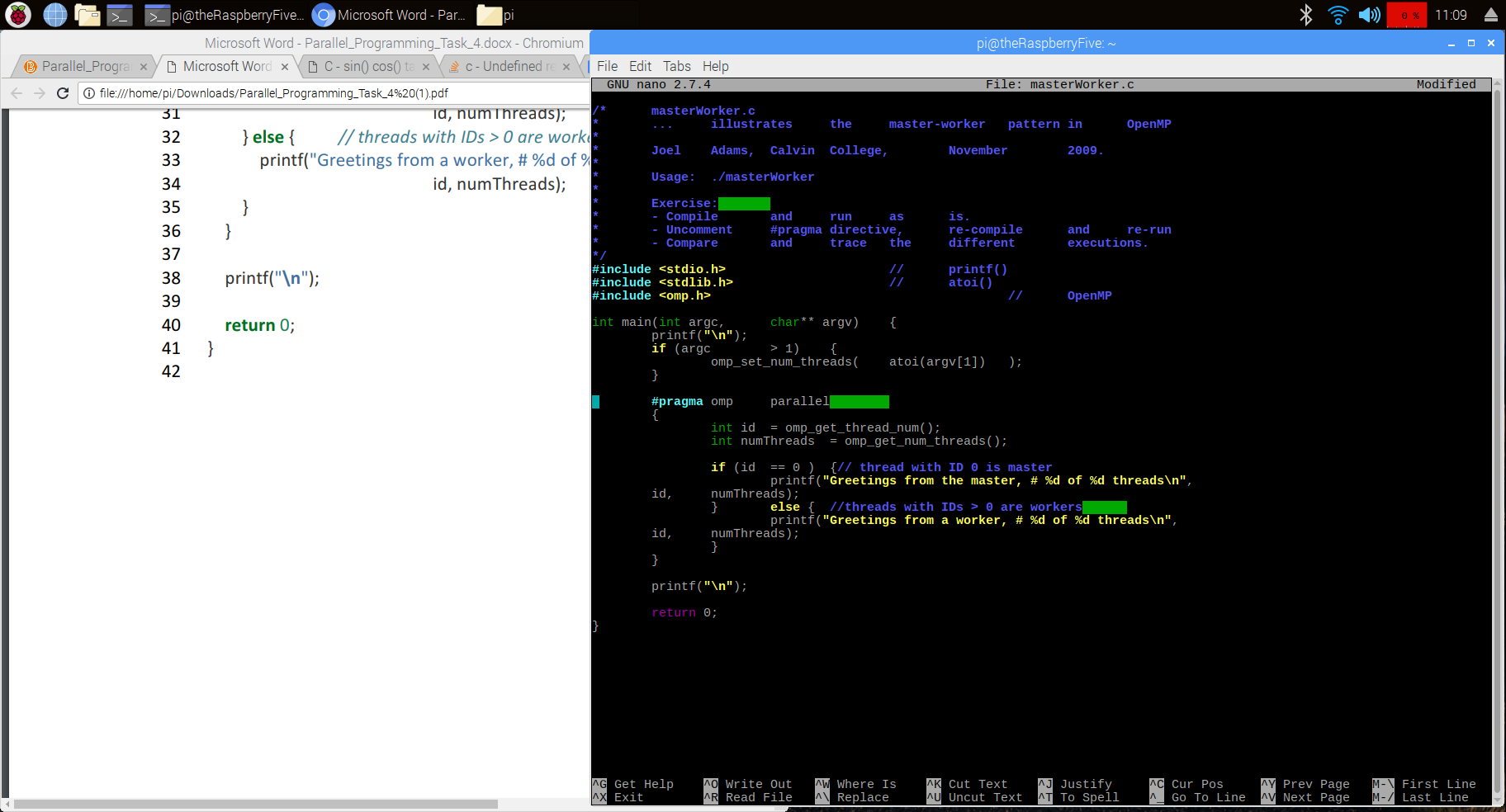


With the barrier commented out, each thread executes all of its statements before the following thread begins.

But with the barrier active (no longer commented out), each thread stops at the barrier until all threads have reached it, and then the threads continue and execute the statement after the barrier.



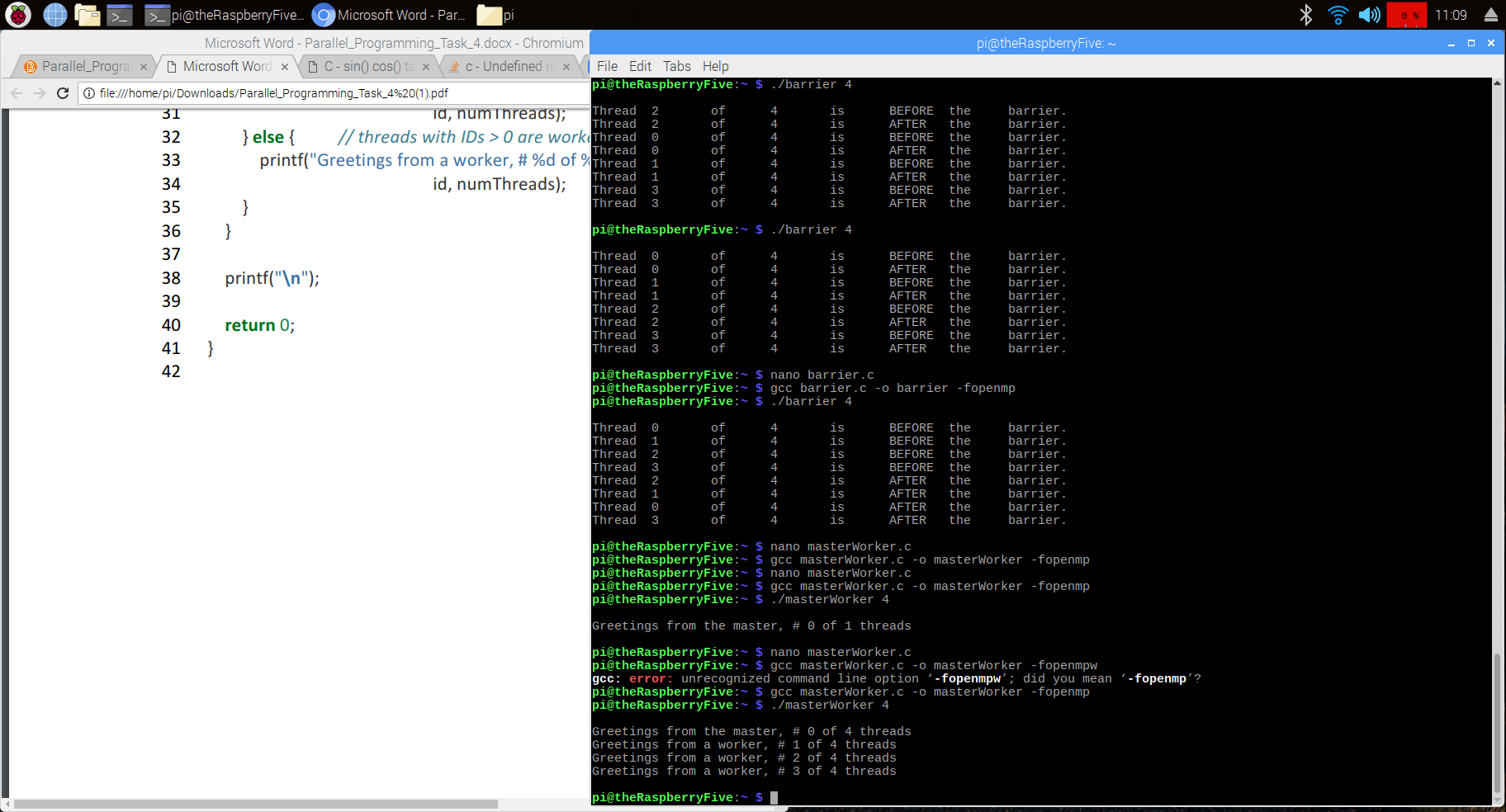
**4: Program Structure: The Master-Worker Implementation Strategy**



This program uses the

master-worker pattern to allow one thread to execute one segment of code while the other three execute a different segment of code.

With the “#pragma omp parallel” commented out, only the “master’s” statements are executed,



but with the #pragma included in the code, the first thread executes the “master” statement while the other 3 execute the “worker” statement.

