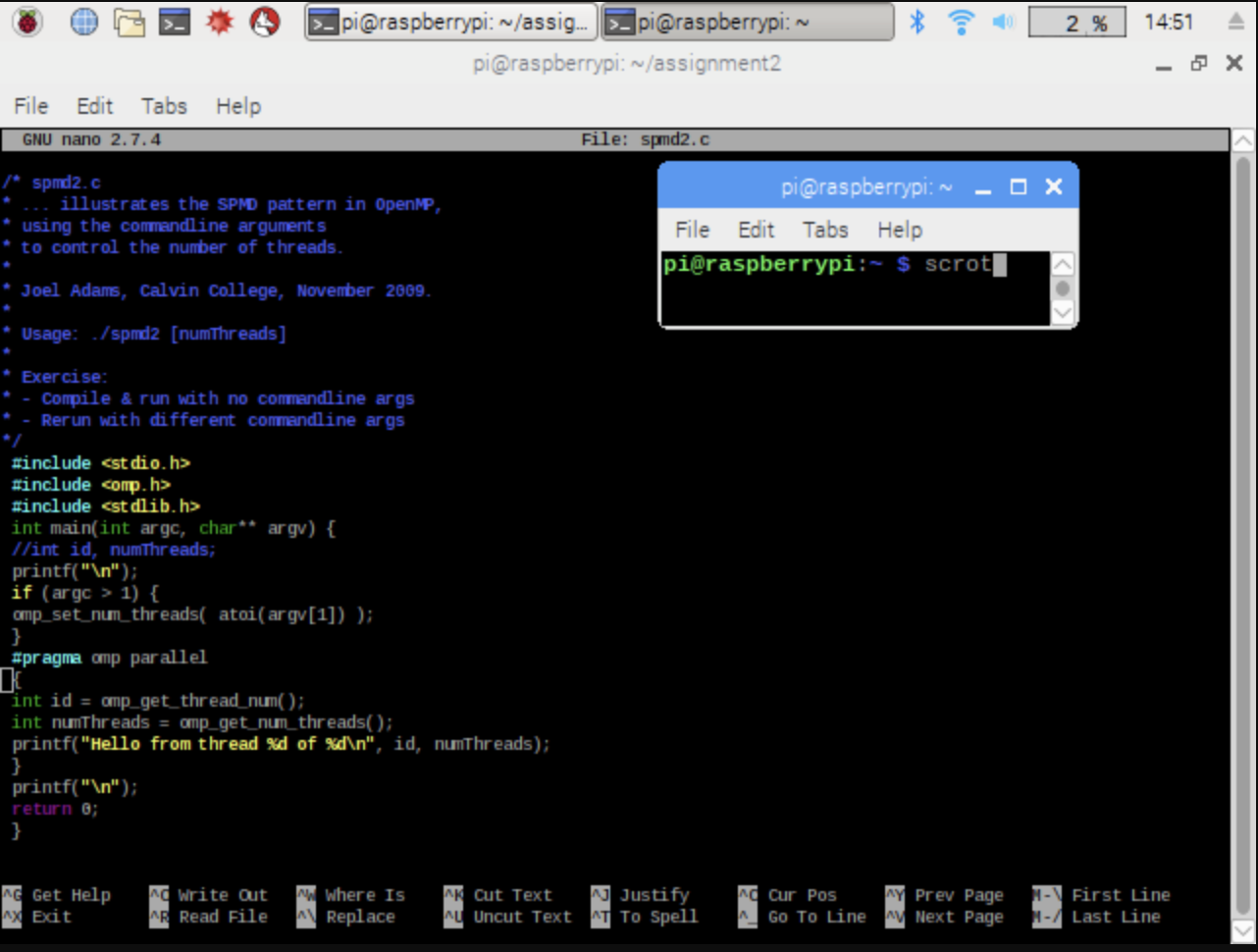
For this part of the project, we worked with a program to observe how threads behave. A thread is a lightweight process that allows a single process to be decomposed into smaller independent parts. The code used for this assignment uses OpenMP pragmas, a set of compiler directives that allow the compiler to generate a threaded code. The main benefit of threading code is that it enables the execution of different instructions on different data in memory at the same time.

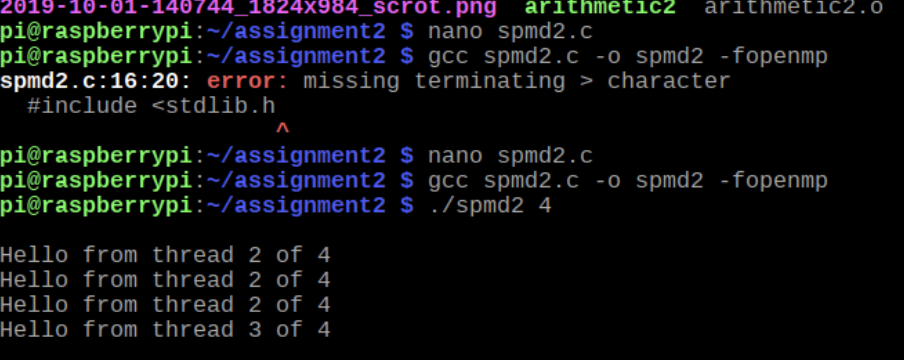
We used the nano editor from the terminal to create a file by typing *nano spmd2.c,* where we copied the following code:



Up to line 10, the code is run in one thread on one core of the Raspberry Pi, while from lines 11 to 15 the code is run on separate threads. Then again, in lines 15 to 18, the threads are joint back and are run in thread 0 like in the beginning.

After saving and exiting the editor, we compiled our program by using the command *gcc spmd2.c -o spmd2 -fopenmp*  and ran the program with *./smpd2 4.*  The number at the end of the run command indicates how many threads to fork the program in.

After running the programs, we obtained the following output:



The thread id number should not appear listed more than once as each has a unique id. The problem with the code was that all the cores in the Raspberry Pi share the same memory bank and, therefore, the variables’ declaration should occur inside the block that will be forked and run in parallel to avoid having the threads share the variable’s memory location.

To fix this problem, we modified lines 5, 12 and 13 in the following way:

*5. //int id, numThreads;*

*12. int id = omp\_get\_thread\_num();*

*13. int numThreads = omp\_get\_num\_threads();*

This way, we are stating in the code that each thread will have its own copy of the variable’s named *id* and *numThreads.* The corrected output was:

