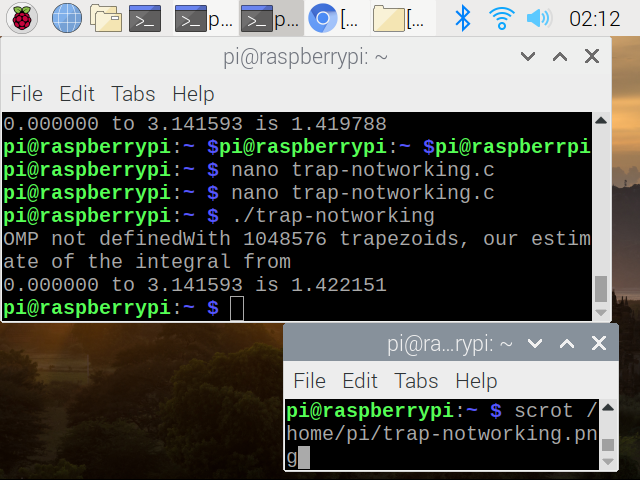
Parallel Programming Basics

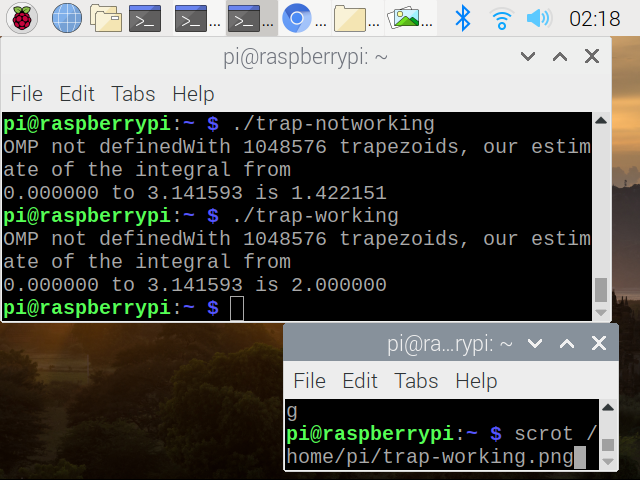
1. trap-notworking.c



In this program trap-notworking.c, the output is incorrect because the variable ‘integral’ was incorrectly declared as shared in the OpenMp directive.

Ex: #pragma omp parallel for private(i) shared (a, n, h, integral)

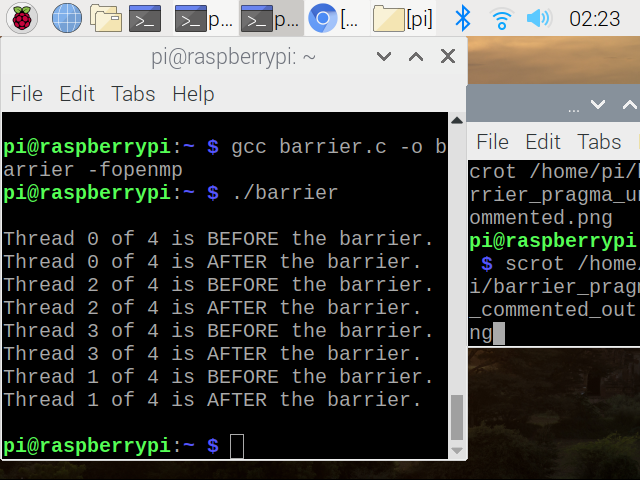
1. trap-working.c



In this program trap-working, the output is correct because we changed the variable ‘integral’ from shared to reduction.

|  |
| --- |
| Ex:  #pragma omp parallel for \ |
|  | private(i) shared (a, n, h) reduction(+: integral) |

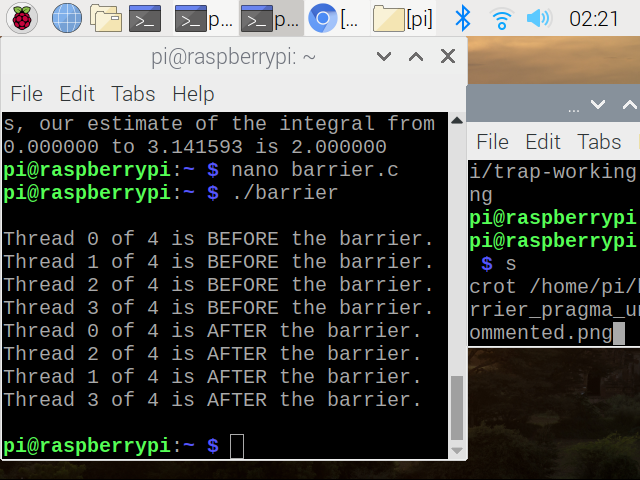
1. barrier.c pragma commented out



In this program barrier.c, the output is wrong because there are multiple threads printing one after another when the threads should wait until each unique thread has printed to continue to print duplicate numbered threads.

Ex. // #pragma omp barrier

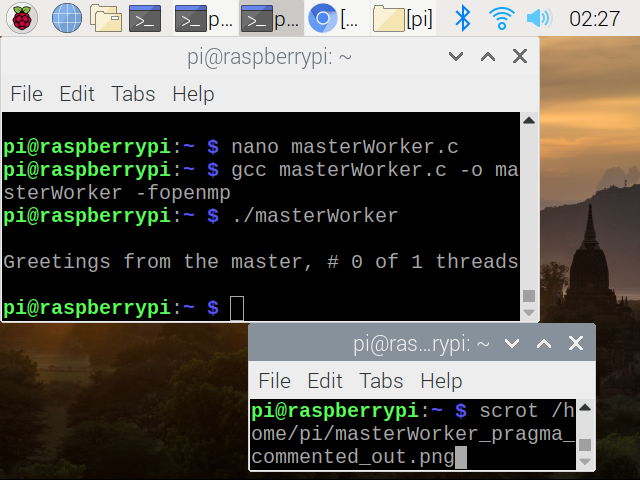
1. barrier.c pragma uncommented



The output in this program is right because we uncommented the OMP barrier so that the barrier blocks all processes (threads) until other processes reach the barrier to continue.

Ex: #pragma omp barrier

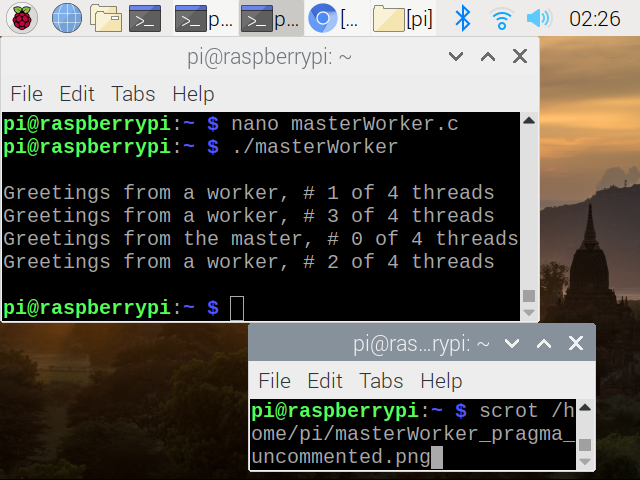
1. masterWorker.c pragma commented out



In the program masterWorker.c, the outcome is not desirable because the workers were not printed out as they should’ve been. This can be fixed by uncommenting the OMP parallel directive.

Ex: // #pragma omp parallel

1. masterWorker.c pragma uncommented



This program is fixed because the OMP parallel directive allows the program to use the fork-join pattern in which one thread called the master divides the work among the workers and outputs the aggregation of the results from the workers. In this case, the master and each of the workers all execute one block of code each.

Ex: #pragma omp parallel