Jordan University of Science and Technology

Operating Systems
Assignment 2

Done By:

Tala Abu Qdais 132333
Feras Hamam 134806
Amal Rasas 132835

Instructor : Mohammad Alshboul

Assignment Tasks:

1. Screenshot of the main code:

```
200
201 int main(int argc , char* argv[]){
202   readFile();//do not change order of readfile => always the first line of main
203   //thread Number
204   int NumOfThreads = 0;//number of threads
205   NumOfThreads = atoi(argv[1]);
206   printf("\n");
207   //end of thread Number
208   //declaring threads
209   pthread_t threads[NumOfThreads];
210
211   threadWork = ceil(x1.getRowsLength()/NumOfThreads);//how much each thread should work
212   //end of declaring threads
213   //starting threads
214   result.setMatrix(x1.getRowsLength(),x1.getColumnsLength());
215   sem_init(&semaphore,0,1);
216   for(int i = 0 ; i < NumOfThreads; i++){
217       int rc = pthread_create(threads+i,NULL,*crossProduct,(void *) (intptr_t) i);
218       if (rc) {
219          cout << "Error:unable to create thread," << rc << endl;
220          exit(-1);
221       }
222   }
223   for(int i = 0 ; i < NumOfThreads; i++){
224       int rc = pthread_join(threads[i], NULL);
225       if (rc) {
226          cout << "Error:unable to create thread," << rc << endl;
227          exit(-1);
228       }
229   }
230   sem_destroy(&semaphore);
231   printf("numOfEven=%d numOfOdd=%d totalCells=%d\n",result.getNoEven(),result.getNoOdd(),result.getTotalCells());
232   writeResult();
233   //end of starting threads
234 }
```

2. Screenshot of the thread function(s):

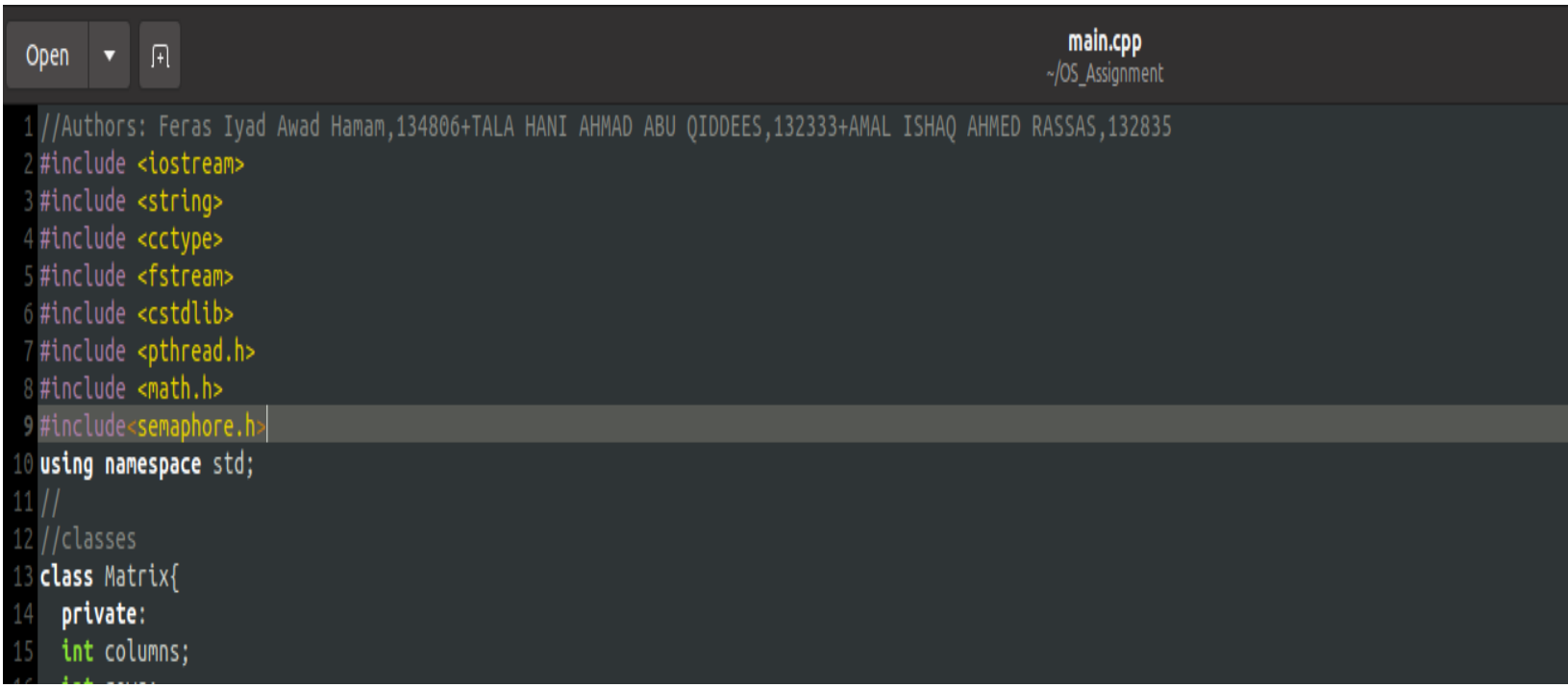
```
213  //starting threads
214  result.setMatrix(x1.getRowsLength(),x1.getColumnsLength());
215  sem_init(&semaphore,0,1);
216  for(int i = 0 ; i < NumOfThreads; i++){
217      int rc = pthread_create(threads+i,NULL,*crossProduct,(void *) (intptr_t) i);
218      if (rc) {
219         cout << "Error:unable to create thread," << rc << endl;
220         exit(-1);
221      }
222  }
223  for(int i = 0 ; i < NumOfThreads; i++){
224      int rc = pthread_join(threads[i], NULL);
225      if (rc) {
226         cout << "Error:unable to create thread," << rc << endl;
227         exit(-1);
228      }
229  }
230  sem_destroy(&semaphore);
231  printf("numOfEven=%d numOfOdd=%d totalCells=%d\n",result.getNoEven(),result.getNoOdd(),result.getTotalCells());
232  writeResult();
233  //end of starting threads
```

Open

main.cpp
~/OS_Assignment

```
152 //cross product
153 void *crossProduct(void *id){
154   int threadID = (intptr_t)id;
155   int startLoop = -1;
156   int endLoop = -1;
157   int i = 0;//rows of first matrix
158   int sum = 0;
159   int counter = 0;//counter to determine when a thread should stop
160   while(true){//# of rows for the first matrix
161     if(threadWork < 1)
162     {
163       threadWork = 1;
164     }
165     sem_wait(&semaphore);
166     if(firstMRow >= x1.getRowsLength() || counter >= threadWork){
167       endLoop = i+1;
168       sem_post(&semaphore);
169       break;//end thread if all rows have been calculated before or if the thread has finished his work(threadWork)
170     }
171     i = firstMRow;
172     firstMRow++;
173     sem_post(&semaphore);
174     if(startLoop == -1)//if it is not running yet!
175     {
176       startLoop = i;
177
178     }
179     endLoop = i;
180     for(int j = 0;j<x2.getColumnsLength();j++){//# of cols for the second matrix
181       sum=0;
182       for(int k = 0; k<x1.getColumnsLength();k++){//# of cols for the first matrix
183       sum=sum+((x1.getMatrix())[i][k])*((x2.getMatrix())[k][j]);
184       }
185       sem_wait(&semaphore);
186       result.setValue(i,j,sum);
187       sem_post(&semaphore);
188     }
189     counter++;
190   }
191   if(startLoop == -1 && endLoop ==-1){
192     printf("ThreadID=%d\n", threadID);
```

3. Screenshot highlighting the parts of the code that were added to make the code thread-safe, with explanations on the need for them:




```
1  //Authors: Feras Iyad Awad Hamam,134806+TALA HANI AHMAD ABU QIDDEES,132333+AMAL ISHAQ AHMED RASSAS,132835
2  #include <iostream>
3  #include <string>
4  #include <cctype>
5  #include <fstream>
6  #include <cstdlib>
7  #include <pthread.h>
8  #include <math.h>
9  #include<semaphore.h>
10 using namespace std;
11 //
12 //classes
13 class Matrix{
14   private:
15   int columns;
```

```
62 };
63 //end of classes
64 //globals
65 Matrix x1 =  Matrix();
66 Matrix x2 =  Matrix();
67 Matrix result = Matrix();
68 int firstMRow = 0;//first Matrix Row Counter to use it in threads
69 int threadWork  = 0;//how much each thread should work determined in main
70 sem_t semaphore;
71 //end of globals
72
73 //Read functions
74 int lineAnalyzing(string line, int lineCounter, int matrixSize){
```

```
157   int i = 0;//rows of first matrix
158   int sum = 0;
159   int counter = 0;//counter to determine when a thread should stop
160   while(true){//# of rows for the first matrix
161     if(threadWork < 1)
162     {
163       threadWork = 1;
164     }
165     sem_wait(&semaphore);
166     if(firstMRow >= x1.getRowsLength() || counter >= threadWork){
167       endLoop = i+1;
168       sem_post(&semaphore);
169       break;//end thread if all rows have been calculated before or if the thread has finished his work(threadWork)
170     }
171     i = firstMRow;
172     firstMRow++;
173     sem_post(&semaphore);
174     if(startLoop == -1)//if it is not running yet!
175     {
176       startLoop = i;
177
178     }
179     endLoop = i;
180     for(int j = 0;j<x2.getColumnsLength();j++){//# of cols for the second matrix
181       sum=0;
182       for(int k = 0; k<x1.getColumnsLength();k++){//# of cols for the first matrix
183       sum=sum+((x1.getMatrix())[i][k])*((x2.getMatrix())[k][j]);
184       }
185       sem_wait(&semaphore);
186       result.setValue(i,j,sum);
187       sem_post(&semaphore);
188     }
189     counter++;
190   }
```

```
200
201 int main(int argc , char* argv[]){
202   readFile();//do not change order of readfile => always the first line of main
203   //thread Number
204   int NumOfThreads = 0;//number of threads
205   NumOfThreads = atoi(argv[1]);
206   printf("\n");
207   //end of thread Number
208   //declaring threads
209   pthread_t threads[NumOfThreads];
210
211   threadWork = ceil(x1.getRowsLength()/NumOfThreads);//how much each thread should work
212   //end of declaring threads
213   //starting threads
214   result.setMatrix(x1.getRowsLength(),x1.getColumnsLength());
215   sem_init(&semaphore,0,1);
216   for(int i = 0 ; i < NumOfThreads; i++){
217       int rc = pthread_create(threads+i,NULL,*crossProduct,(void *) (intptr_t) i);
218       if (rc) {
219          cout << "Error:unable to create thread," << rc << endl;
220          exit(-1);
221       }
222   }
223   for(int i = 0 ; i < NumOfThreads; i++){
224       int rc = pthread_join(threads[i], NULL);
225       if (rc) {
226          cout << "Error:unable to create thread," << rc << endl;
227          exit(-1);
228       }
229   }
230   sem_destroy(&semaphore);
231   printf("numOfEven=%d numOfOdd=%d totalCells=%d\n",result.getNoEven(),result.getNoOdd(),result.getTotalCells());
232   writeResult();
233   //end of starting threads
234 }
```

Semaphore Explanation:

A semaphore is a synchronization object that controls access by multiple processes to a common resource in a parallel programming environment. Semaphores are widely used to control access to files and shared memory.

In our code , we used semaphores to handle race conditions that occur on the following variables:
firstMrow , numOfEven, numOfOdd, totalCells.

1.include<semaphore.h>:
Header to define the sem_t type.

2.sem_t semaphore:
Semaphore type.

3. sem_wait(&semaphore):
Function to perform a semaphore lock operation.

4. sem_post(&semaphore):
Function to perform a semaphore unlock operation.

5. sem_init(&semaphore ,0 ,1):
Function that initializes the unnamed semaphore. We passed 0 as a second argument because there is only one process and multiple threads working ,and passed 1 as a third argument as the initial value of the semaphore.

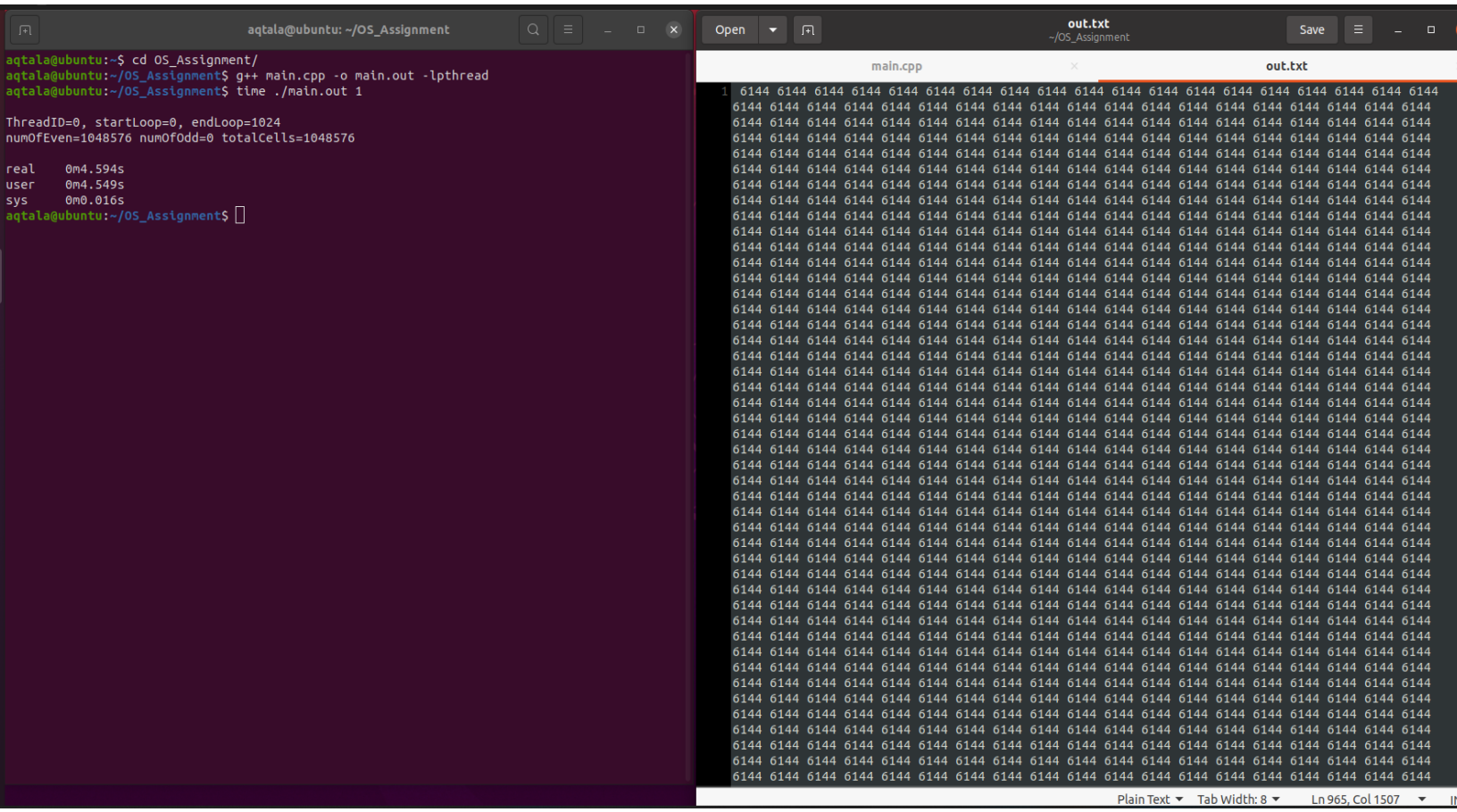
6. sem_destroy(&semaphore):
Function that is used to destroy the unnamed semaphore.

4. Screenshot of the output of the two versions of your code (thread-safe vs. non-thread-safe), when running passing the following number of threads (T): 1, 2, 4, 8, 16, 32.

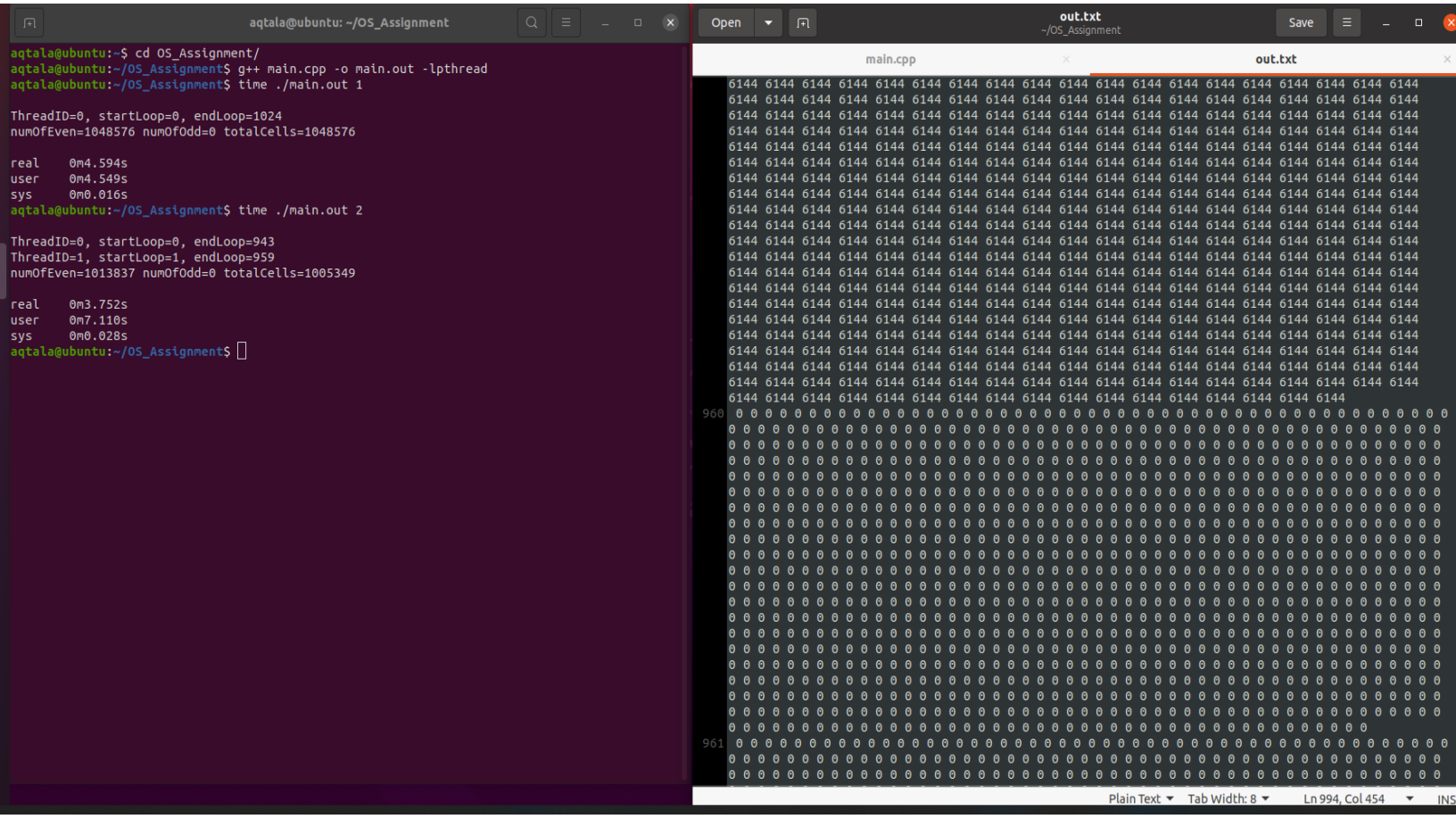
****NOTE :** the input file that we used is the one uploaded on elearning (N=1024).

4.1 Non-thread-safe :

OF THREADS = 1 :

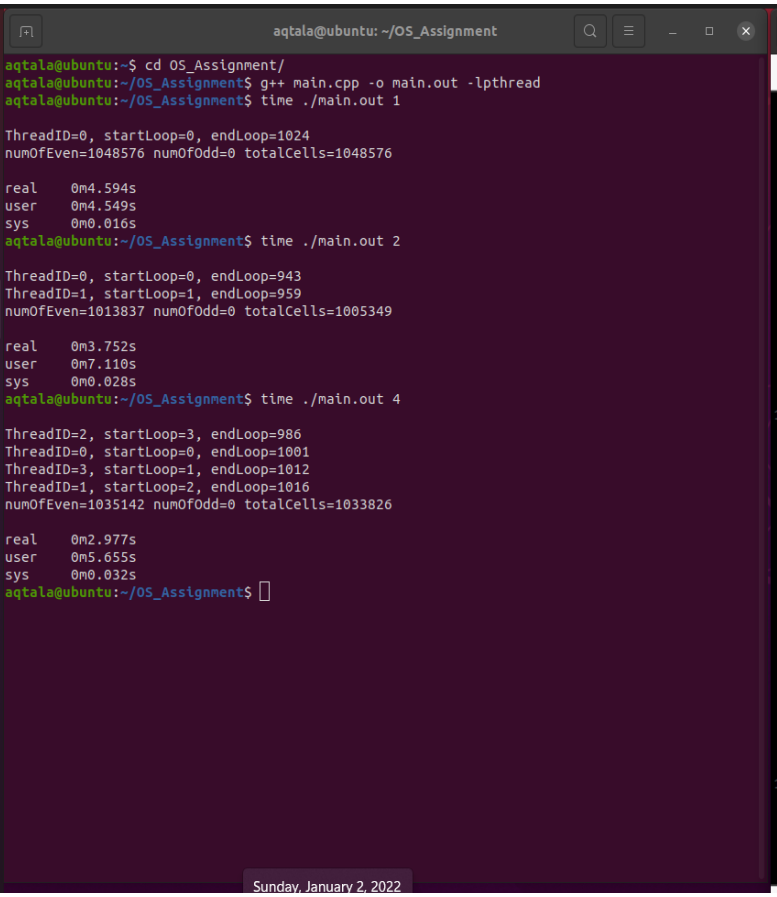


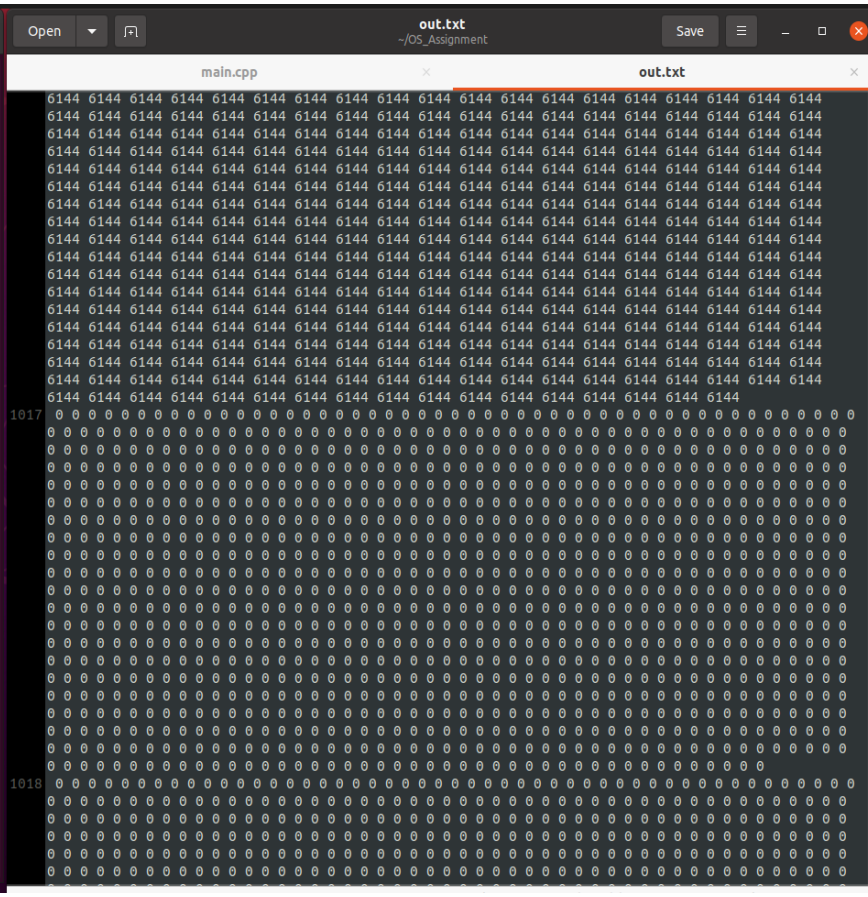
OF THREADS = 2 :



Here thread 1 stopped working on line 959(all the lines after 959 are 0's), and the number of odd and even cells does not match the total number of cells .

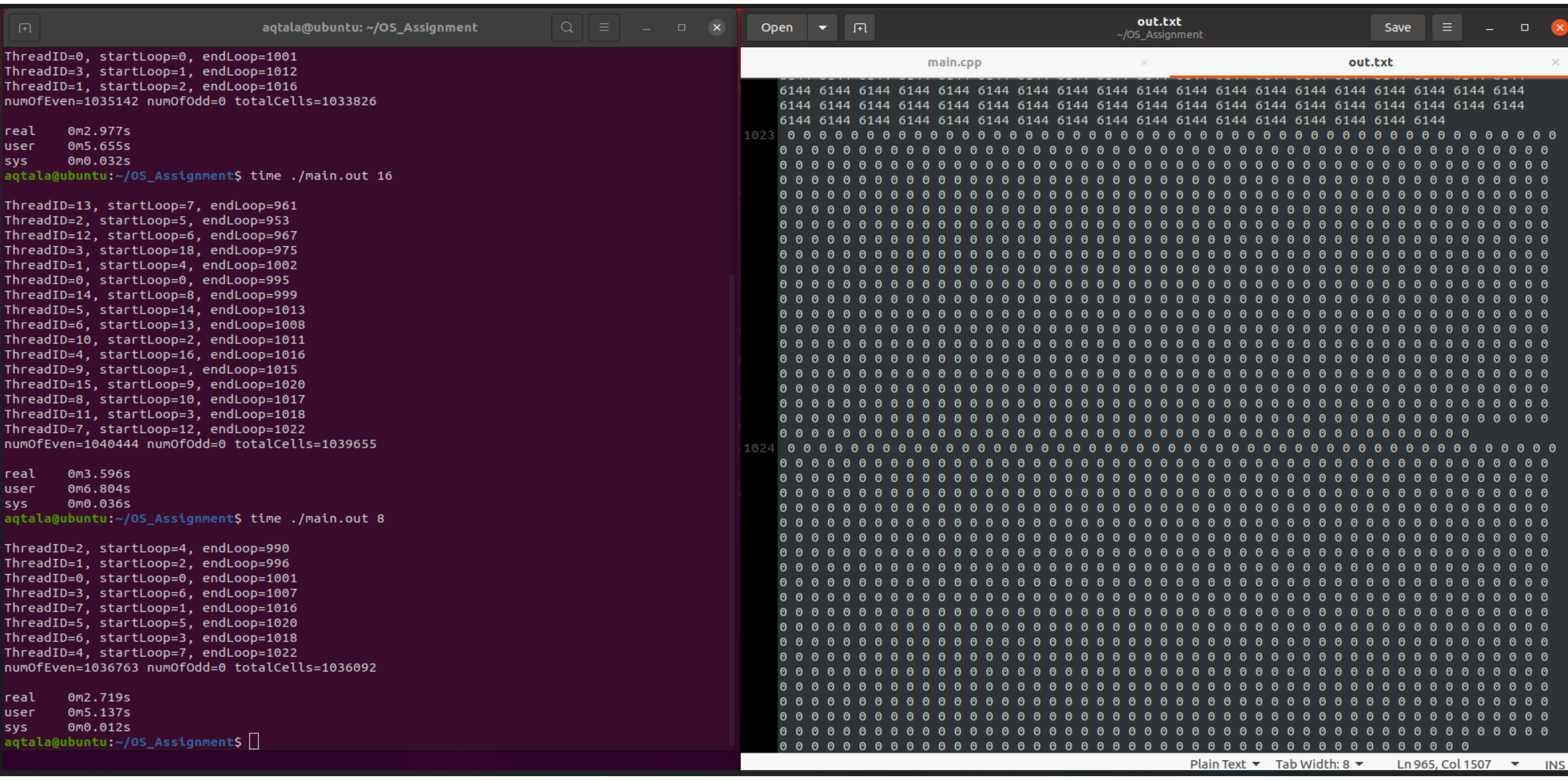
OF THREADS = 4 :





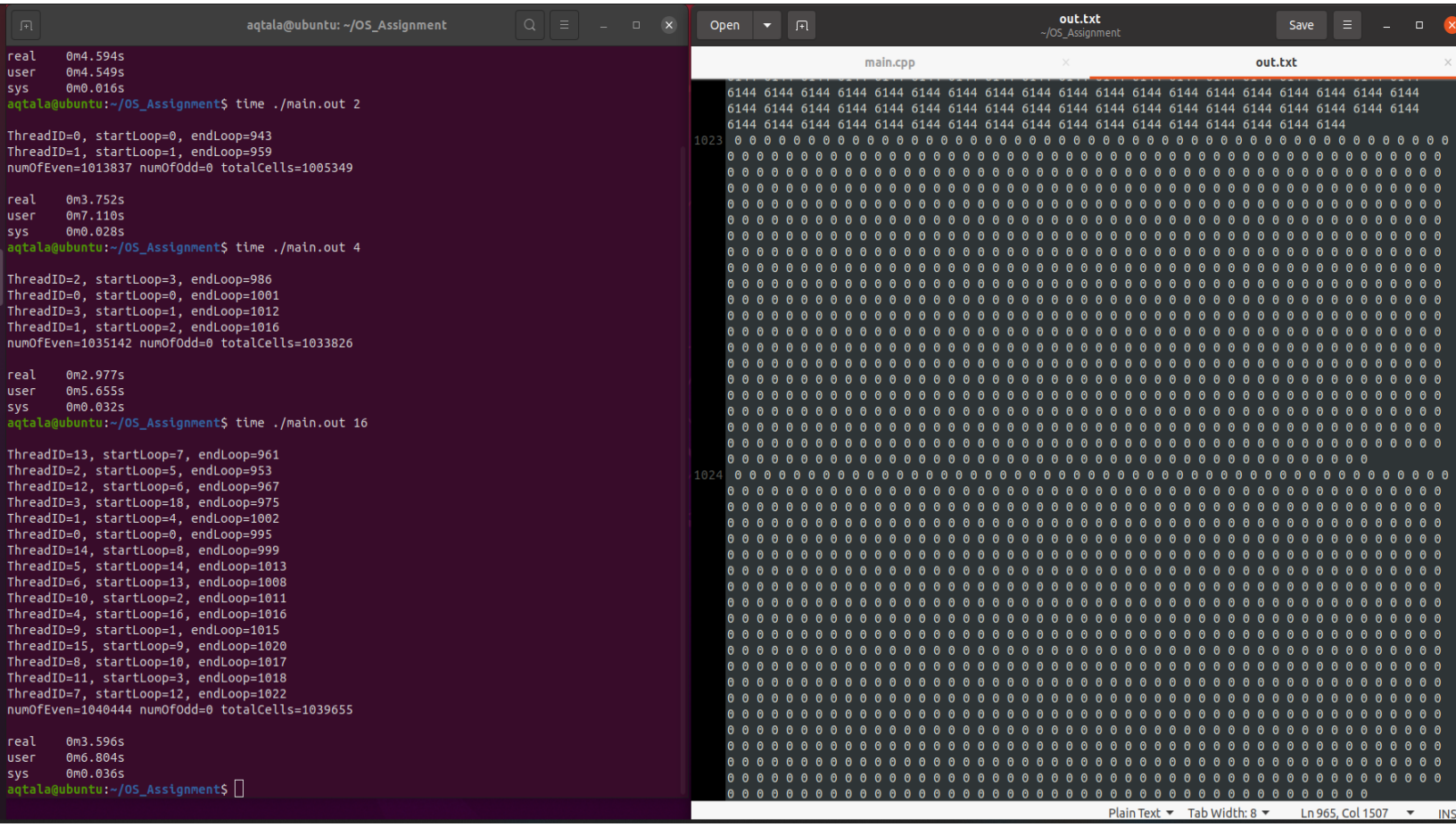
Here thread 1 stopped working on line 1016 (all the lines after 1016 are 0's), and the number of odd and even cells does not match the total number of cells .

OF THREADS = 8 :



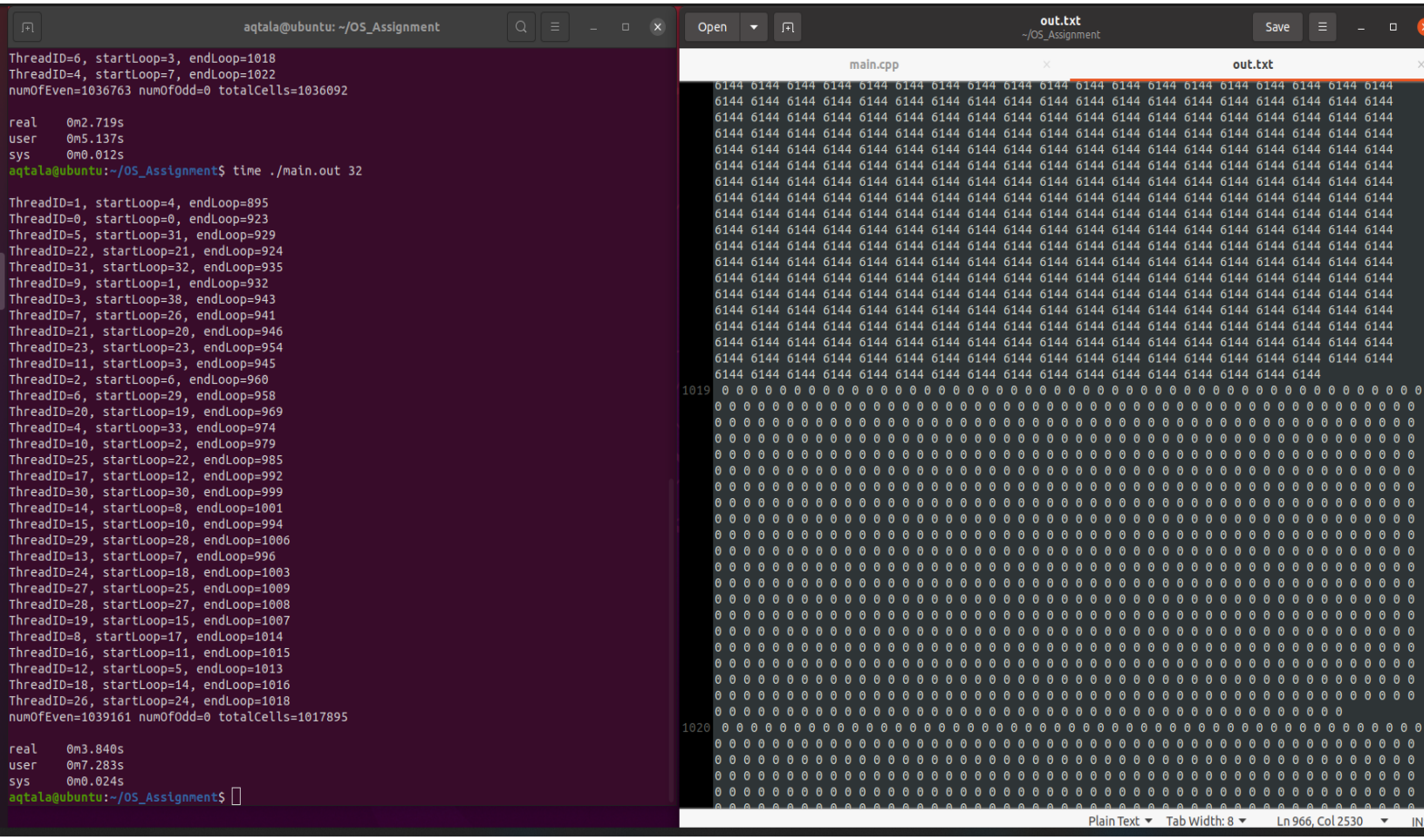
Here thread 4 stopped working on line 1022 (all the lines after 1022 are 0's), and the number of odd and even cells does not match the total number of cells .

OF THREADS = 16 :



Here thread 7 stopped working on line 1022 (all the lines after 1022 are 0's), and the number of odd and even cells does not match the total number of cells .

OF THREADS = 32 :

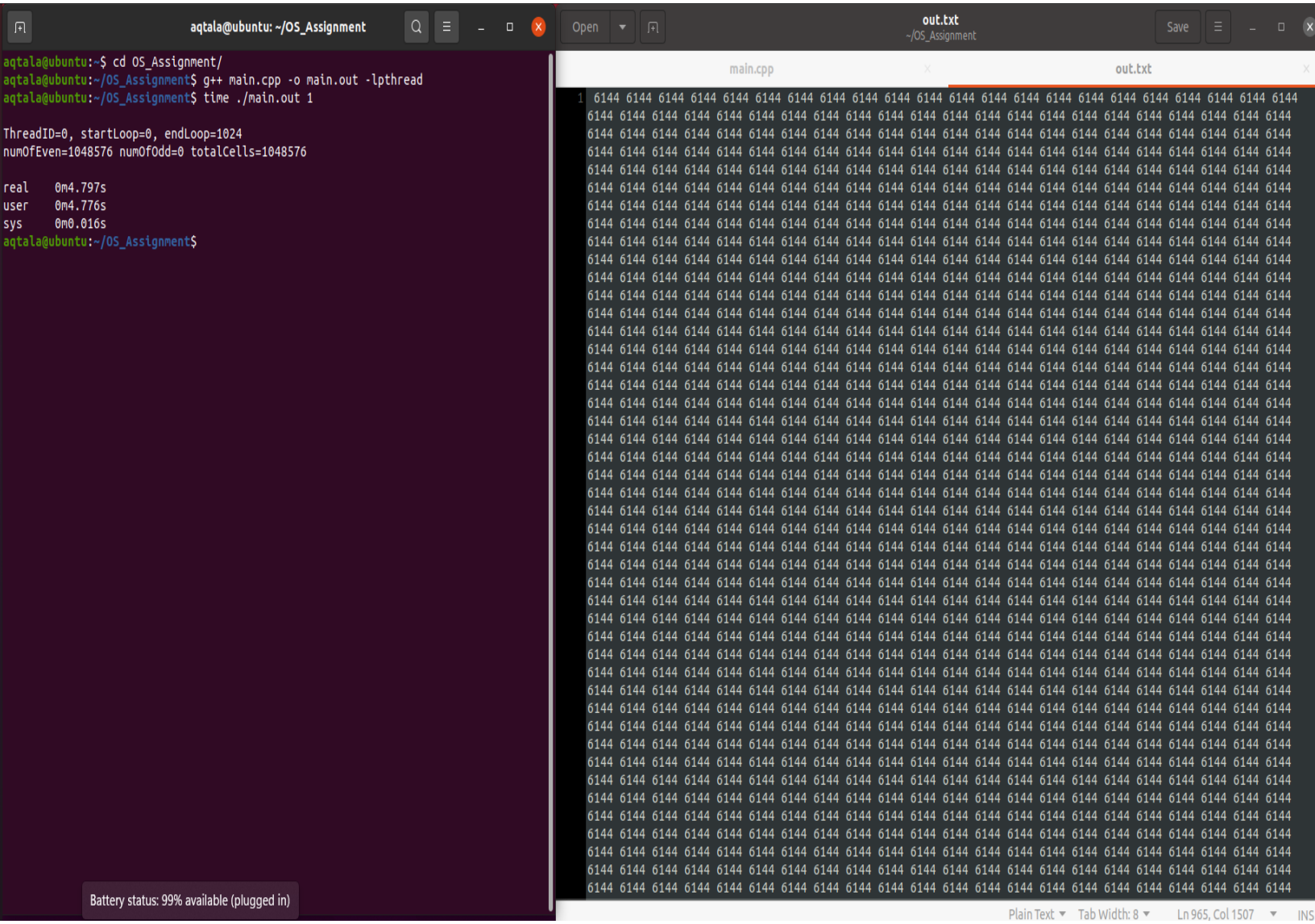


Here thread 26 stopped working on line 1018 (all the lines after 1018 are 0's), and the number of odd and even cells does not match the total number of cells .

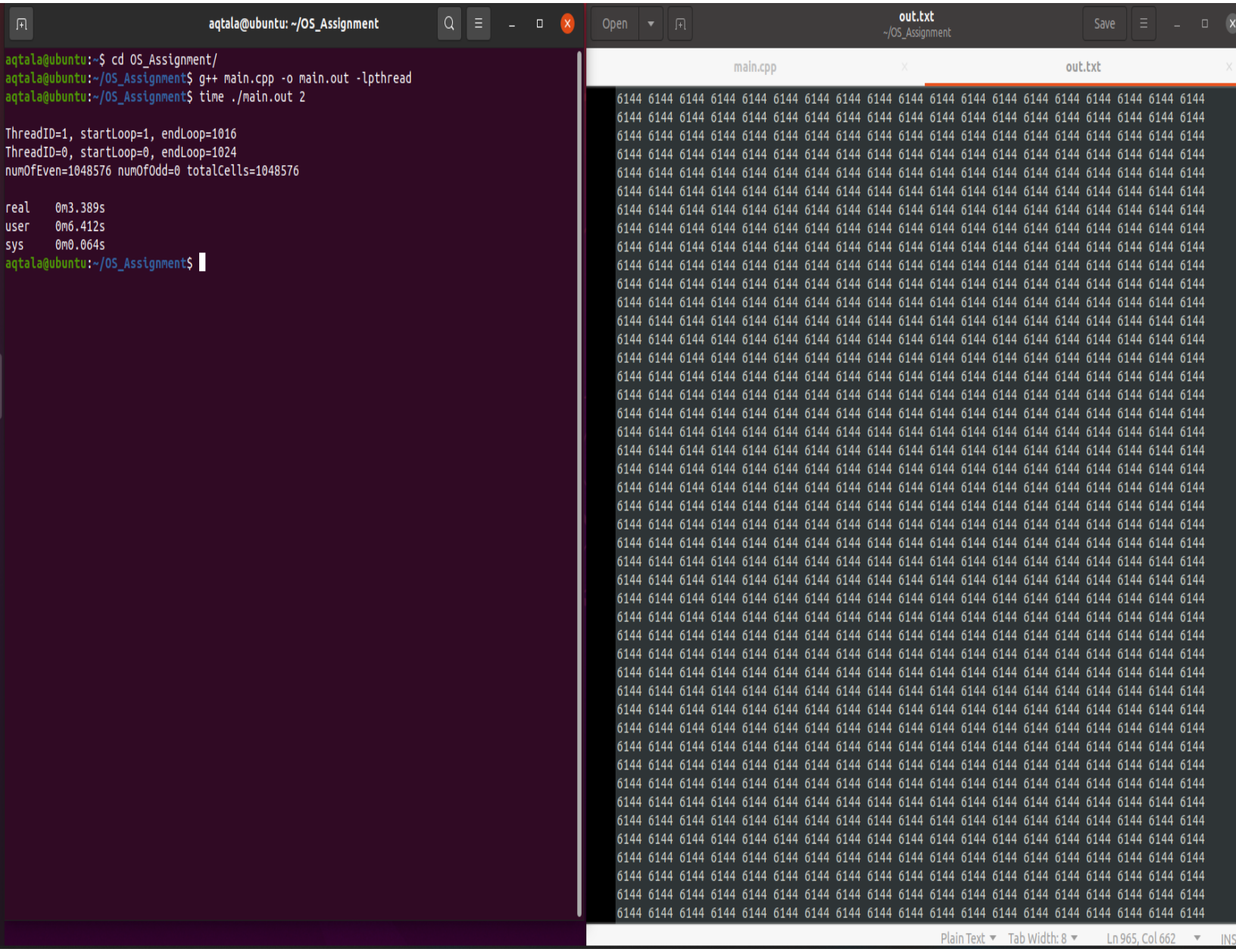
4.2 Thread-safe :

In thread-safe code one of the threads always stops executing on line 1024 (the end of the output matrix) , and the number of odd and even cells will always be equal to the number of total cells.

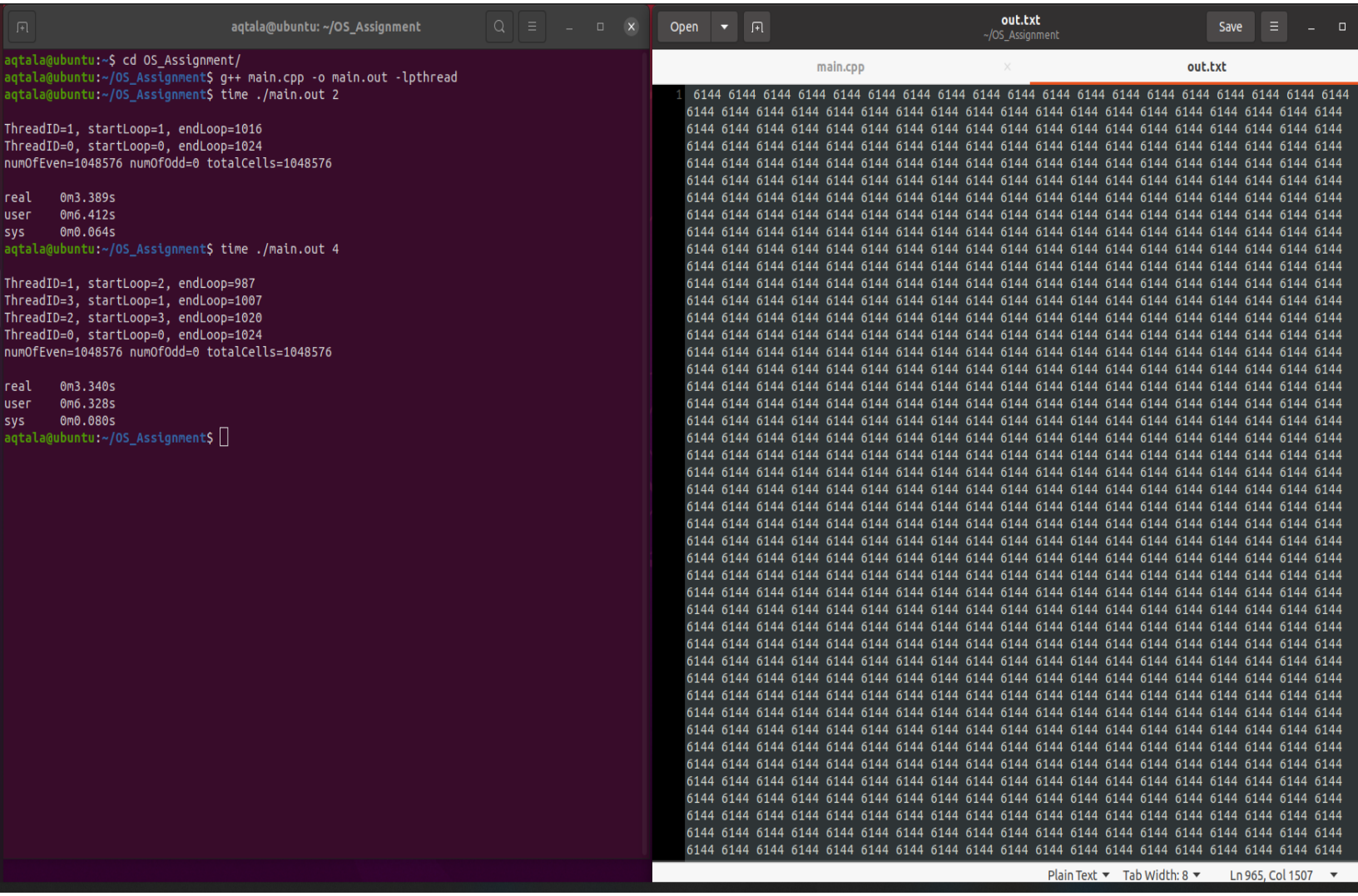
OF THREADS = 1 :



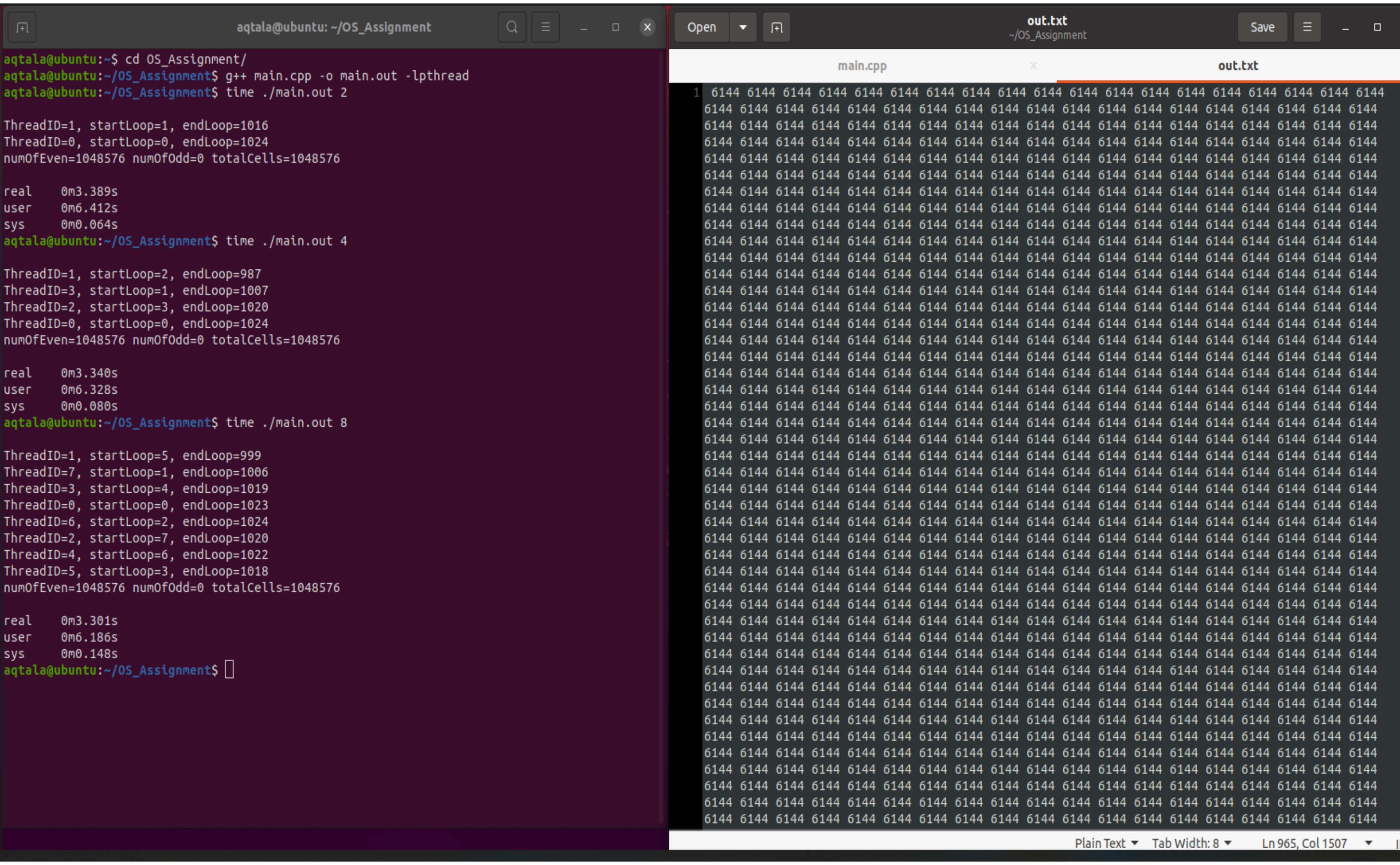
OF THREADS = 2 :


aqtala@ubuntu: ~/OS_Assignment
aqtala@ubuntu:~$ cd OS_Assignment/
aqtala@ubuntu:~/OS_Assignment$ g++ main.cpp -o main.out -lpthread
aqtala@ubuntu:~/OS_Assignment$ time ./main.out 2
ThreadID=1, startLoop=1, endLoop=1016
ThreadID=0, startLoop=0, endLoop=1024
numOfEven=1048576 numOfOdd=0 totalCells=1048576
real 0m3.389s
user 0m6.412s
sys 0m0.064s
aqtala@ubuntu:~/OS_Assignment$
Open
out.txt
~/OS_Assignment
Save
main.cpp
out.txt
6144
Plain Text
Tab Width: 8
Ln 965, Col 662
INS

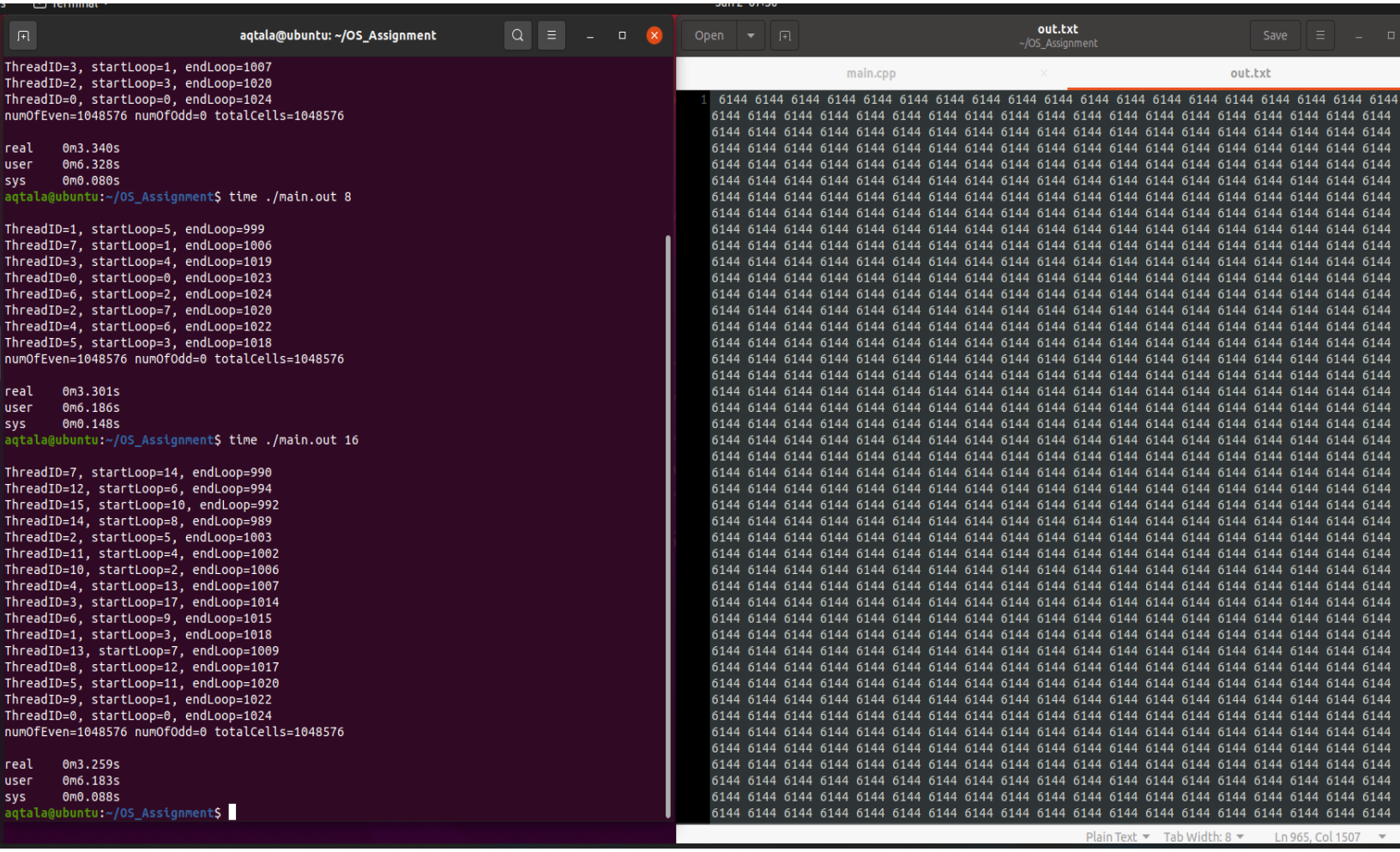
OF THREADS = 4 :


aqtala@ubuntu: ~/OS_Assignment
aqtala@ubuntu:~$ cd OS_Assignment/
aqtala@ubuntu:~/OS_Assignment$ g++ main.cpp -o main.out -lpthread
aqtala@ubuntu:~/OS_Assignment$ time ./main.out 2
ThreadID=1, startLoop=1, endLoop=1016
ThreadID=0, startLoop=0, endLoop=1024
numOfEven=1048576 numOfOdd=0 totalCells=1048576
real 0m3.389s
user 0m6.412s
sys 0m0.064s
aqtala@ubuntu:~/OS_Assignment$ time ./main.out 4
ThreadID=1, startLoop=2, endLoop=987
ThreadID=3, startLoop=1, endLoop=1007
ThreadID=2, startLoop=3, endLoop=1020
ThreadID=0, startLoop=0, endLoop=1024
numOfEven=1048576 numOfOdd=0 totalCells=1048576
real 0m3.340s
user 0m6.328s
sys 0m0.080s
aqtala@ubuntu:~/OS_Assignment$
Open
out.txt
~/OS_Assignment
Save
main.cpp
out.txt
6144
Plain Text
Tab Width: 8
Ln 965, Col 1507

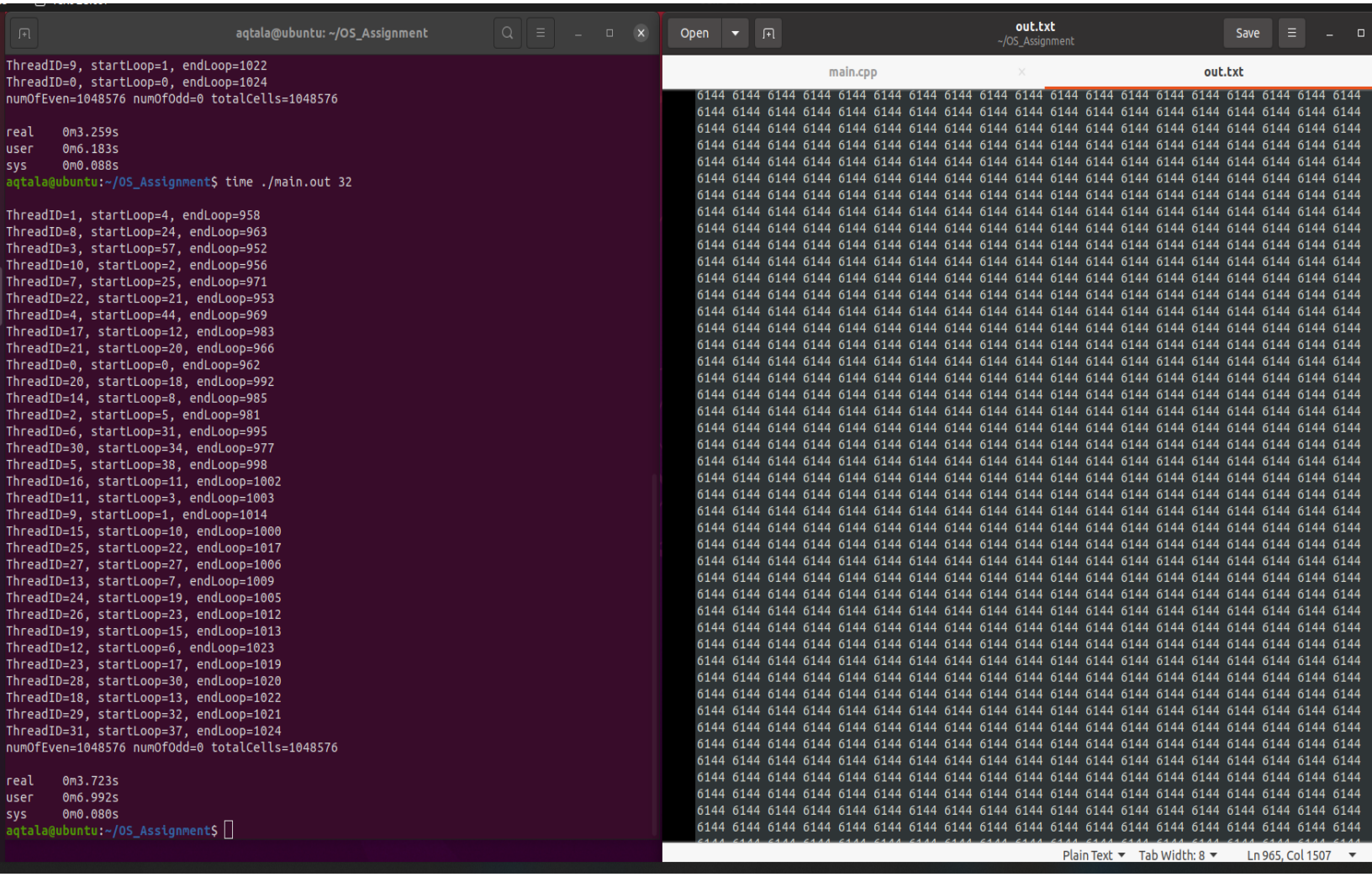
OF THREADS = 8 :


aqtala@ubuntu: ~/OS_Assignment
aqtala@ubuntu:~$ cd OS_Assignment/
aqtala@ubuntu:~/OS_Assignment$ g++ main.cpp -o main.out -lpthread
aqtala@ubuntu:~/OS_Assignment$ time ./main.out 2
ThreadID=1, startLoop=1, endLoop=1016
ThreadID=0, startLoop=0, endLoop=1024
numOfEven=1048576 numOfOdd=0 totalCells=1048576
real 0m3.389s
user 0m6.412s
sys 0m0.064s
aqtala@ubuntu:~/OS_Assignment$ time ./main.out 4
ThreadID=1, startLoop=2, endLoop=987
ThreadID=3, startLoop=1, endLoop=1007
ThreadID=2, startLoop=3, endLoop=1020
ThreadID=0, startLoop=0, endLoop=1024
numOfEven=1048576 numOfOdd=0 totalCells=1048576
real 0m3.340s
user 0m6.328s
sys 0m0.080s
aqtala@ubuntu:~/OS_Assignment$ time ./main.out 8
ThreadID=1, startLoop=5, endLoop=999
ThreadID=7, startLoop=1, endLoop=1006
ThreadID=3, startLoop=4, endLoop=1019
ThreadID=0, startLoop=0, endLoop=1023
ThreadID=6, startLoop=2, endLoop=1024
ThreadID=2, startLoop=7, endLoop=1020
ThreadID=4, startLoop=6, endLoop=1022
ThreadID=5, startLoop=3, endLoop=1018
numOfEven=1048576 numOfOdd=0 totalCells=1048576
real 0m3.301s
user 0m6.186s
sys 0m0.148s
aqtala@ubuntu:~/OS_Assignment$
Open
out.txt
~/OS_Assignment
Save
main.cpp
out.txt
Plain Text
Tab Width: 8
Ln 965, Col 1507

#OF THREADS = 16 :



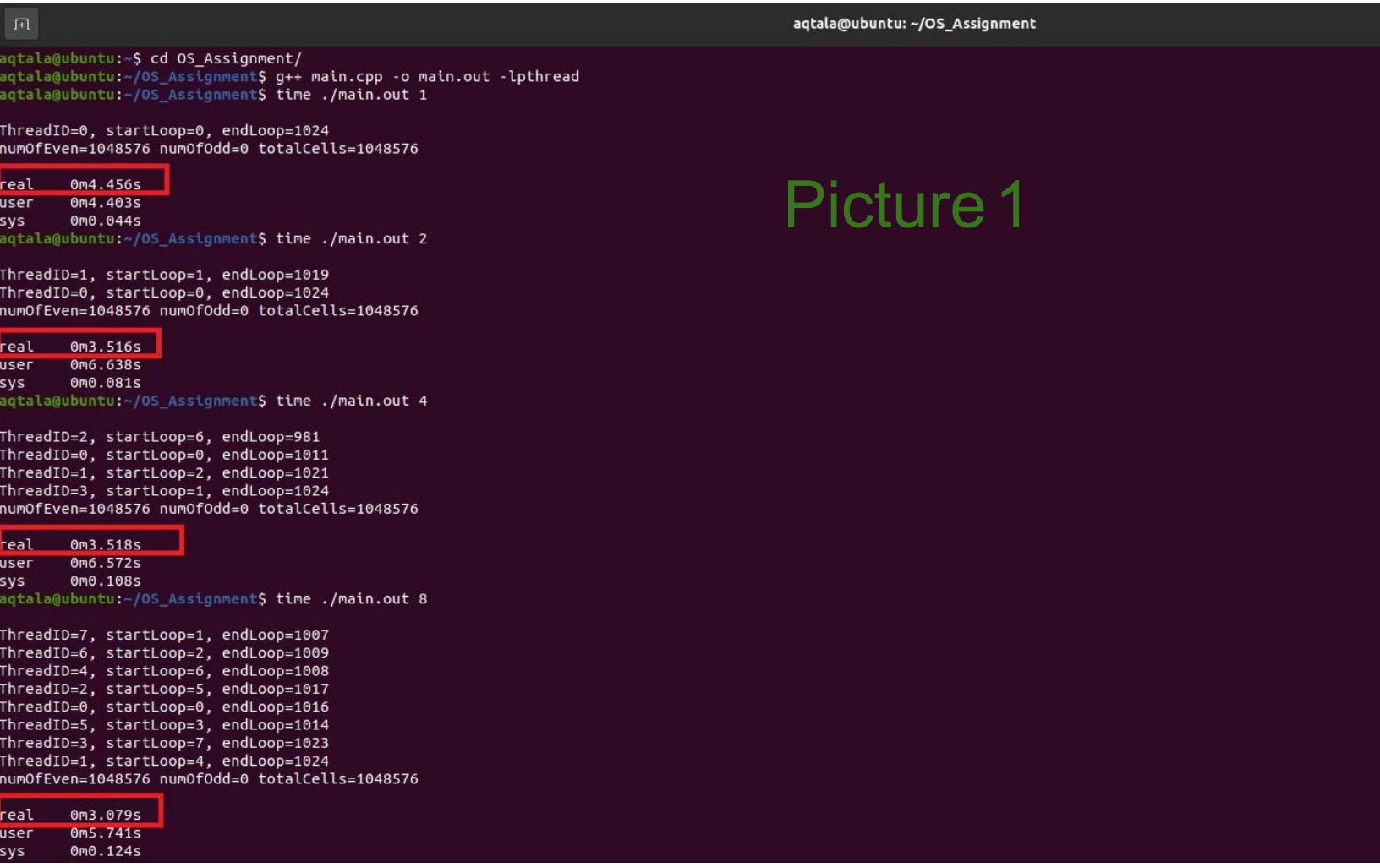
OF THREADS = 32 :



5. Based on your code, how many computing units (e.g. cores, hyper-threads) does your machine have? Provide screenshots of how you arrived at this conclusion, and a screenshot of the actual properties of your machine to validate your conclusion.

Diminishing returns is a concept in economics that means : in productive processes, increasing a factor of production by one unit, while holding all other production factors constant, will at some point return a lower unit of output per incremental unit of input (the output remains positive however productivity and efficiency decrease).

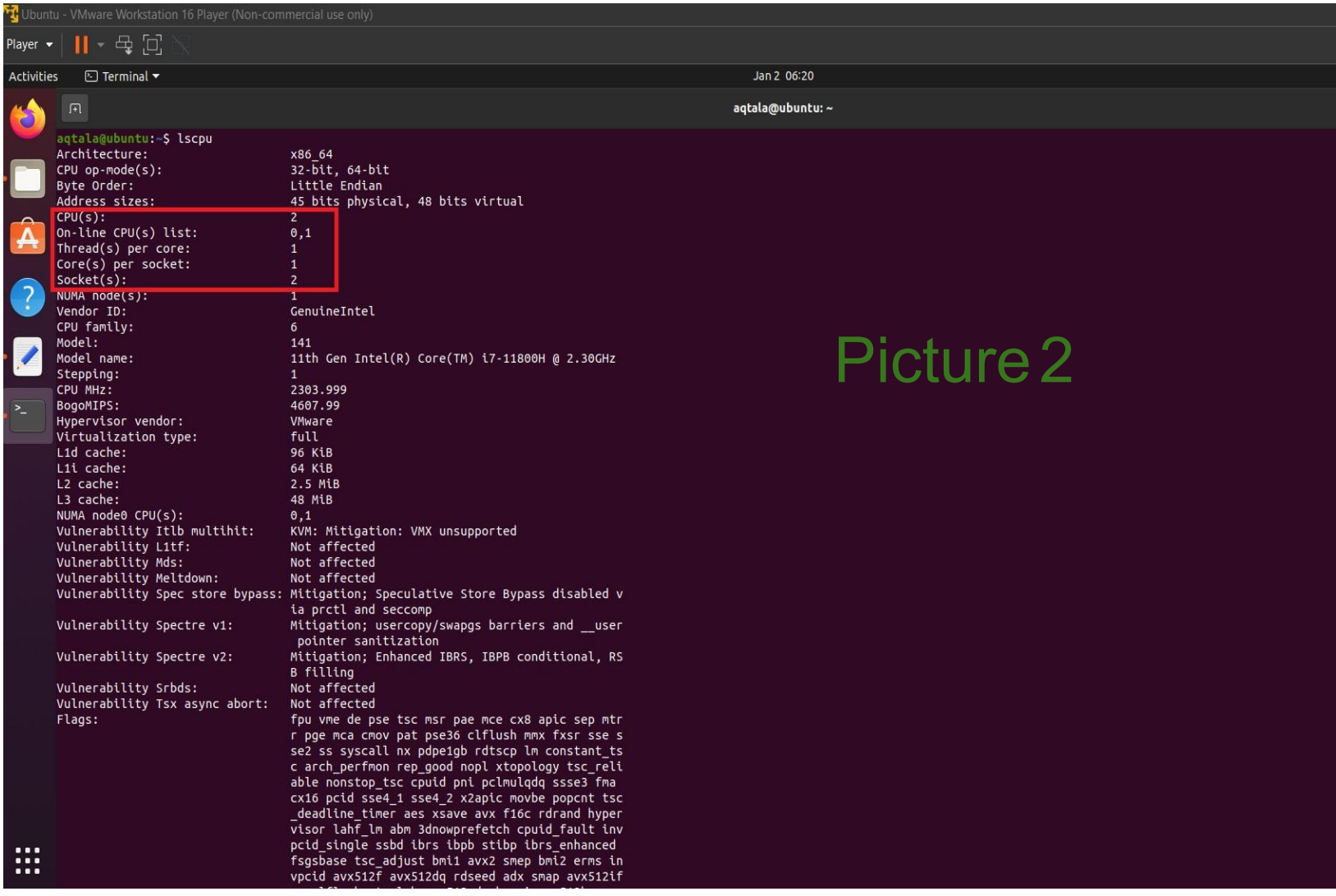
In our case the increasing factor would be the number of threads,and the output would be the execution time.



Picture 1

Assumptions:

As we can see in the picture above (picture 1) when we use only 1 thread the execution time is 4.456s and when we use 2 threads the execution time decreases approximately by 1 sec.
After using more than 2 threads the execution time is kind of steady (slight change) when compared with the above cases . So we can assume that the machine has **2 threads** .



The actual properties :

As seen in the picture above (picture 2) this machine has **2 cores** and **1 thread** per core
(**2 threads in total**).

** **NOTE :**This is a virtual machine and it is not using all the cores the actual machine has.

GITHUB REPOSITORY LINK :

https://github.com/FerasHamam/OS_Assignment