



**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 ▾ F1 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         endLoop = i;
179         for(int j = 0;j<x2.getColumnsLength();j++){//# of cols for the second matrix
180             sum=0;
181             for(int k = 0; k<x1.getColumnsLength();k++){//# of cols for the first matrix
182                 sum=sum+((x1.getMatrix())[i][k])*((x2.getMatrix())[k][j]);
183             }
184             sem_wait(&semaphore);
185             result.setValue(i,j,sum);
186             sem_post(&semaphore);
187         }
188         counter++;
189     }
190     if(startLoop == -1 && endLoop ==-1){
191         printf("ThreadID=%d\n", threadID);
192     }
}
```

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

```
main.cpp
~/OS_Assignment

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;
16     ...
17 };
18 ...
19 ...
20 ...
21 ...
22 ...
23 ...
24 ...
25 ...
26 ...
27 ...
28 ...
29 ...
30 ...
31 ...
32 ...
33 ...
34 ...
35 ...
36 ...
37 ...
38 ...
39 ...
40 ...
41 ...
42 ...
43 ...
44 ...
45 ...
46 ...
47 ...
48 ...
49 ...
50 ...
51 ...
52 ...
53 ...
54 ...
55 ...
56 ...
57 ...
58 ...
59 ...
60 ...
61 ...
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     endLoop = i;
179     for(int j = 0;j<x2.getColumnsLength();j++){//# of cols for the second matrix
180         sum=0;
181         for(int k = 0; k<x1.getColumnsLength();k++){//# of cols for the first matrix
182             sum=sum+((x1.getMatrix())[i][k])*((x2.getMatrix())[k][j]);
183         }
184         sem_wait(&semaphore);
185         result.setValue(i,j,sum);
186         sem_post(&semaphore);
187     }
188     counter++;
189 }
190 }

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

```

## **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 :

```
aqtala@ubuntu:~$ cd OS_Assignment
aqtala@ubuntu:~/OS_Assignment$ g++ main.cpp -o main.out -lpthread
aqtala@ubuntu:~/OS_Assignment$ time ./main.out 1
ThreadID=0, startLoop=0, endLoop=1024
numOfEven=1048576 numOfOdd=0 totalCells=1048576

real    0m4.594s
user    0m4.549s
sys     0m0.016s
aqtala@ubuntu:~/OS_Assignment$
```

The screenshot shows a terminal window with two tabs: "main.cpp" and "out.txt". The "main.cpp" tab contains the C++ code for the program. The "out.txt" tab shows the output of the program, which consists of a large number of memory addresses (6144) repeated many times, indicating a memory leak or corruption. The terminal also displays system performance metrics like CPU usage and memory usage.

## # OF THREADS = 2 :

```
aqtala@ubuntu:~/OS_Assignment$ cd OS_Assignment
aqtala@ubuntu:~/OS_Assignment$ g++ main.cpp -o main.out -lpthread
aqtala@ubuntu:~/OS_Assignment$ time ./main.out 1
ThreadID=0, startLoop=0, endLoop=1024
numOfEven=1048576 numOfOdd=0 totalCells=1048576

real    0m4.594s
user    0m4.549s
sys     0m0.016s
aqtala@ubuntu:~/OS_Assignment$ time ./main.out 2
ThreadID=0, startLoop=0, endLoop=943
ThreadID=1, startLoop=1, endLoop=959
numOfEven=1013837 numOfOdd=0 totalCells=1005349

real    0m3.752s
user    0m7.110s
sys     0m0.028s
aqtala@ubuntu:~/OS_Assignment$ 
```

main.cpp

out.txt

out.txt

959

960

961

Plain Text ▾ Tab Width: 8 ▾ Ln 994, Col 454 ▾ INS

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 :

```
aqtala@ubuntu:~/OS_Assignment$ cd OS_Assignment/
aqtala@ubuntu:/OS_Assignment$ g++ main.cpp -o main.out -lpthread
aqtala@ubuntu:/OS_Assignment$ time ./main.out 1
ThreadId=0, startLoop=0, endLoop=1024
numOfEven=1048576 numOfOdd=0 totalCells=1048576
real    0m4.594s
user    0m4.549s
sys     0m0.016s
aqtala@ubuntu:/OS_Assignment$ time ./main.out 2
ThreadId=0, startLoop=0, endLoop=943
ThreadId=1, startLoop=1, endLoop=959
numOfEven=1013837 numOfOdd=0 totalCells=1005349
real    0m3.752s
user    0m7.110s
sys     0m0.028s
aqtala@ubuntu:/OS_Assignment$ time ./main.out 4
ThreadId=2, startLoop=3, endLoop=986
ThreadId=0, startLoop=0, endLoop=1001
ThreadId=3, startLoop=1, endLoop=1012
ThreadId=1, startLoop=2, endLoop=1016
numOfEven=1035142 numOfOdd=0 totalCells=1033826
real    0m2.977s
user    0m5.655s
sys     0m0.032s
aqtala@ubuntu:/OS_Assignment$
```

The terminal window shows the execution of a C++ program named 'main.out' which processes data in parallel using four threads. The program prints out thread IDs, start and end loops, and the number of even and odd cells processed. The final output shows the total number of cells processed by each thread.

The 'main.cpp' file contains the source code for the program, and the 'out.txt' file contains the output data, which appears to be a large grid of binary values (0s and 1s).

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 :

```
aqtala@ubuntu:~/OS_Assignment$ time ./main.out 16
ThreadID=0, startLoop=0, endLoop=1001
ThreadID=3, startLoop=1, endLoop=1012
ThreadID=1, startLoop=2, endLoop=1016
ThreadID=0numOfEven=1035142 numOfOdd=0 totalCells=1033826
real    0m2.977s
user    0m5.655s
sys     0m0.032s
aqtala@ubuntu:~/OS_Assignment$ time ./main.out 16
ThreadID=13, startLoop=7, endLoop=961
ThreadID=2, startLoop=5, endLoop=953
ThreadID=12, startLoop=6, endLoop=967
ThreadID=3, startLoop=18, endLoop=975
ThreadID=1, startLoop=4, endLoop=1002
ThreadID=0, startLoop=0, endLoop=995
ThreadID=14, startLoop=8, endLoop=999
ThreadID=5, startLoop=14, endLoop=1013
ThreadID=6, startLoop=13, endLoop=1008
ThreadID=10, startLoop=2, endLoop=1011
ThreadID=4, startLoop=16, endLoop=1016
ThreadID=9, startLoop=1, endLoop=1015
ThreadID=15, startLoop=9, endLoop=1028
ThreadID=8, startLoop=10, endLoop=1017
ThreadID=1, startLoop=3, endLoop=1018
ThreadID=7, startLoop=12, endLoop=1022
numOfEven=1040444 numOfOdd=0 totalCells=1039655
real    0m3.596s
user    0m6.804s
sys     0m0.036s
aqtala@ubuntu:~/OS_Assignment$ time ./main.out 8
ThreadID=2, startLoop=4, endLoop=990
ThreadID=1, startLoop=2, endLoop=996
ThreadID=0, startLoop=0, endLoop=1001
ThreadID=3, startLoop=6, endLoop=1007
ThreadID=7, startLoop=1, endLoop=1016
ThreadID=5, startLoop=5, endLoop=1020
ThreadID=6, startLoop=3, endLoop=1018
ThreadID=4, startLoop=7, endLoop=1022
numOfEven=1036763 numOfOdd=0 totalCells=1036092
real    0m2.719s
user    0m5.137s
sys     0m0.012s
aqtala@ubuntu:~/OS_Assignment$
```

The terminal window shows two runs of the program. The first run with 16 threads completes successfully, while the second run with 8 threads fails at line 1022. The output file 'out.txt' contains a grid of binary data, with the last few lines being all zeros.

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 :

The screenshot shows a terminal window with the following content:

```
real 0m4.594s
user 0m4.549s
sys 0m0.016s
aqtala@ubuntu:~/OS_Assignment$ time ./main.out 2
ThreadID=0, startLoop=0, endLoop=943
ThreadID=1, startLoop=1, endLoop=959
numOfEven=1013837 numOfOdd=0 totalCells=1005349

real 0m3.752s
user 0m7.110s
sys 0m0.028s
aqtala@ubuntu:~/OS_Assignment$ time ./main.out 4
ThreadID=2, startLoop=3, endLoop=986
ThreadID=0, startLoop=0, endLoop=1001
ThreadID=3, startLoop=1, endLoop=1012
ThreadID=1, startLoop=2, endLoop=1016
numOfEven=1035142 numOfOdd=0 totalCells=1033826

real 0m2.977s
user 0m5.655s
sys 0m0.032s
aqtala@ubuntu:~/OS_Assignment$ time ./main.out 16
ThreadID=13, startLoop=7, endLoop=961
ThreadID=2, startLoop=5, endLoop=953
ThreadID=12, startLoop=6, endLoop=967
ThreadID=3, startLoop=18, endLoop=975
ThreadID=1, startLoop=4, endLoop=1002
ThreadID=0, startLoop=0, endLoop=995
ThreadID=14, startLoop=8, endLoop=999
ThreadID=5, startLoop=14, endLoop=1013
ThreadID=6, startLoop=13, endLoop=1008
ThreadID=10, startLoop=2, endLoop=1011
ThreadID=4, startLoop=16, endLoop=1016
ThreadID=9, startLoop=1, endLoop=1015
ThreadID=15, startLoop=9, endLoop=1020
ThreadID=8, startLoop=10, endLoop=1017
ThreadID=11, startLoop=3, endLoop=1018
ThreadID=7, startLoop=12, endLoop=1022
numOfEven=1040444 numOfOdd=0 totalCells=1039655

real 0m3.596s
user 0m6.804s
sys 0m0.036s
aqtala@ubuntu:~/OS_Assignment$
```

The terminal shows three runs of the program with 2, 4, and 16 threads respectively. The 16-thread run shows a bug where thread 7 stops working at line 1022, leaving the rest of the output as zeros. The status bar at the bottom right indicates "Plain Text" and "Ln 965, Col 1507".

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 :

```
aqtala@ubuntu: ~/OS_Assignment
ThreadID=6, startLoop=3, endLoop=1018
ThreadID=4, startLoop=7, endLoop=1022
numOfEven=1036763 numOfOdd=0 totalCells=1036092
real    0m2.719s
user    0m5.137s
sys     0m0.012s
aqtala@ubuntu:~/OS_Assignment$ time ./main.out 32

ThreadID=1, startLoop=4, endLoop=895
ThreadID=0, startLoop=0, endLoop=923
ThreadID=5, startLoop=31, endLoop=929
ThreadID=22, startLoop=21, endLoop=924
ThreadID=31, startLoop=32, endLoop=935
ThreadID=9, startLoop=1, endLoop=932
ThreadID=3, startLoop=38, endLoop=943
ThreadID=7, startLoop=26, endLoop=941
ThreadID=21, startLoop=20, endLoop=946
ThreadID=23, startLoop=23, endLoop=954
ThreadID=11, startLoop=3, endLoop=945
ThreadID=2, startLoop=6, endLoop=960
ThreadID=6, startLoop=29, endLoop=958
ThreadID=20, startLoop=19, endLoop=969
ThreadID=4, startLoop=33, endLoop=974
ThreadID=10, startLoop=2, endLoop=979
ThreadID=25, startLoop=22, endLoop=985
ThreadID=17, startLoop=12, endLoop=992
ThreadID=30, startLoop=30, endLoop=999
ThreadID=14, startLoop=8, endLoop=1001
ThreadID=15, startLoop=10, endLoop=994
ThreadID=29, startLoop=28, endLoop=1006
ThreadID=13, startLoop=7, endLoop=996
ThreadID=24, startLoop=18, endLoop=1003
ThreadID=27, startLoop=25, endLoop=1009
ThreadID=28, startLoop=27, endLoop=1008
ThreadID=19, startLoop=15, endLoop=1007
ThreadID=8, startLoop=17, endLoop=1014
ThreadID=16, startLoop=11, endLoop=1015
ThreadID=12, startLoop=5, endLoop=1013
ThreadID=18, startLoop=14, endLoop=1016
ThreadID=26, startLoop=24, endLoop=1018
numOfEven=1039161 numOfOdd=0 totalCells=1017895
real    0m3.840s
user    0m7.283s
sys     0m0.024s
aqtala@ubuntu:~/OS_Assignment$
```

The terminal output shows the execution of a C++ program named 'main.out' with 32 threads. The program prints thread IDs, start loops, end loops, and various system performance metrics (real, user, sys times). It also prints the total number of even and odd cells processed.

The output file 'out.txt' contains a grid of binary digits (0s and 1s) representing the state of cells. A red box highlights the last line of the grid, which corresponds to line 1018 in the terminal output. This indicates that thread 26 stopped working on line 1018, as all subsequent lines consist of 0's. The total number of cells processed (1017895) does not match the sum of even and odd cells (1039161 + 1036092 = 2075253).

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 :

The screenshot shows a terminal window on an Ubuntu system. The command line shows the user navigating to the directory ~/OS\_Assignment, compiling the main.cpp file into main.out, and running it with the command ./main.out. The output of the program includes the following information:

```
aqtala@ubuntu:~/OS_Assignment$ cd OS_Assignment
aqtala@ubuntu:~/OS_Assignment$ g++ main.cpp -o main.out -lpthread
aqtala@ubuntu:~/OS_Assignment$ time ./main.out
ThreadID=0, startLoop=0, endLoop=1024
numOfEven=1048576 numOfOdd=0 totalCells=1048576

real    0m4.797s
user    0m4.776s
sys     0m0.016s
aqtala@ubuntu:~/OS_Assignment$
```

The main output consists of a large matrix of the number 6144, filling the entire screen. The matrix is approximately 1024x1024 cells. The terminal also displays the battery status as 99% available (plugged in) and various window control buttons at the top.

## # OF THREADS = 2 :

The screenshot shows a terminal window on an Ubuntu system. The user has navigated to the directory ~/OS\_Assignment and compiled a program named main.cpp into a shared library main.out using g++ -fPIC. They then ran the command time ./main.out 2 to measure the execution time. The output shows the following details:

```
aqtala@ubuntu:~/OS_Assignment$ cd OS_Assignment
aqtala@ubuntu:~/OS_Assignment$ g++ main.cpp -o main.out -fPIC
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$
```

On the right side of the terminal, there is a code editor window titled "main.cpp" and "out.txt". The "main.cpp" tab is active, showing the source code of the program. The "out.txt" tab is also present, showing the output of the program. The output consists of a large number of "6144" characters, indicating the state of memory cells.

## # OF THREADS = 4 :

```
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$ 
```

The terminal window shows the execution of a C++ program named 'main.out'. It is built with g++ and linked against the pthread library (-lpthread). Two runs are shown: one with 2 threads (using the command 'time ./main.out 2') and one with 4 threads (using the command 'time ./main.out 4'). Both runs output the same results: the number of even cells (1048576), the number of odd cells (0), and the total number of cells (1048576). The program also prints the thread ID, start loop, and end loop for each thread. The output file 'out.txt' contains a large number of '6144' characters.

## # OF THREADS = 8 :

```
aqtala@ubuntu:~/OS_Assignment$ 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$
```

The terminal window shows the execution of a C++ program named 'main.out' with different numbers of threads (2, 4, and 8). The program prints thread IDs, loop ranges, and performance metrics (real, user, sys times). The output file 'out.txt' contains a large sequence of '6144' characters.



## # OF THREADS = 32 :

```
ThreadID=9, startLoop=1, endLoop=1022
ThreadID=0, startLoop=0, endLoop=1024
numOfEven=1048576 numOfOdd=0 totalCells=1048576

real    0m3.259s
user    0m6.183s
sys     0m0.088s
aqtala@ubuntu:~/OS_Assignment$ time ./main.out 32

ThreadID=1, startLoop=4, endLoop=958
ThreadID=8, startLoop=24, endLoop=963
ThreadID=3, startLoop=57, endLoop=952
ThreadID=10, startLoop=2, endLoop=956
ThreadID=7, startLoop=25, endLoop=971
ThreadID=22, startLoop=21, endLoop=953
ThreadID=4, startLoop=44, endLoop=969
ThreadID=17, startLoop=12, endLoop=983
ThreadID=21, startLoop=20, endLoop=966
ThreadID=0, startLoop=0, endLoop=962
ThreadID=20, startLoop=18, endLoop=992
ThreadID=14, startLoop=8, endLoop=985
ThreadID=2, startLoop=5, endLoop=981
ThreadID=6, startLoop=31, endLoop=995
ThreadID=30, startLoop=34, endLoop=977
ThreadID=5, startLoop=38, endLoop=998
ThreadID=16, startLoop=11, endLoop=1062
ThreadID=11, startLoop=3, endLoop=1003
ThreadID=9, startLoop=1, endLoop=1014
ThreadID=15, startLoop=10, endLoop=1000
ThreadID=25, startLoop=22, endLoop=1017
ThreadID=27, startLoop=27, endLoop=1066
ThreadID=13, startLoop=7, endLoop=1009
ThreadID=24, startLoop=19, endLoop=1005
ThreadID=26, startLoop=23, endLoop=1012
ThreadID=19, startLoop=15, endLoop=1013
ThreadID=12, startLoop=6, endLoop=1023
ThreadID=23, startLoop=17, endLoop=1019
ThreadID=28, startLoop=30, endLoop=1020
ThreadID=18, startLoop=13, endLoop=1022
ThreadID=29, startLoop=32, endLoop=1021
ThreadID=31, startLoop=37, endLoop=1024
numOfEven=1048576 numOfOdd=0 totalCells=1048576

real    0m3.723s
user    0m6.992s
sys     0m0.080s
aqtala@ubuntu:~/OS_Assignment$
```

The terminal window shows the execution of a C++ program named 'main.out' with 32 threads. The output file 'out.txt' contains a large number of '6144' characters, indicating the execution of the program.

**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.

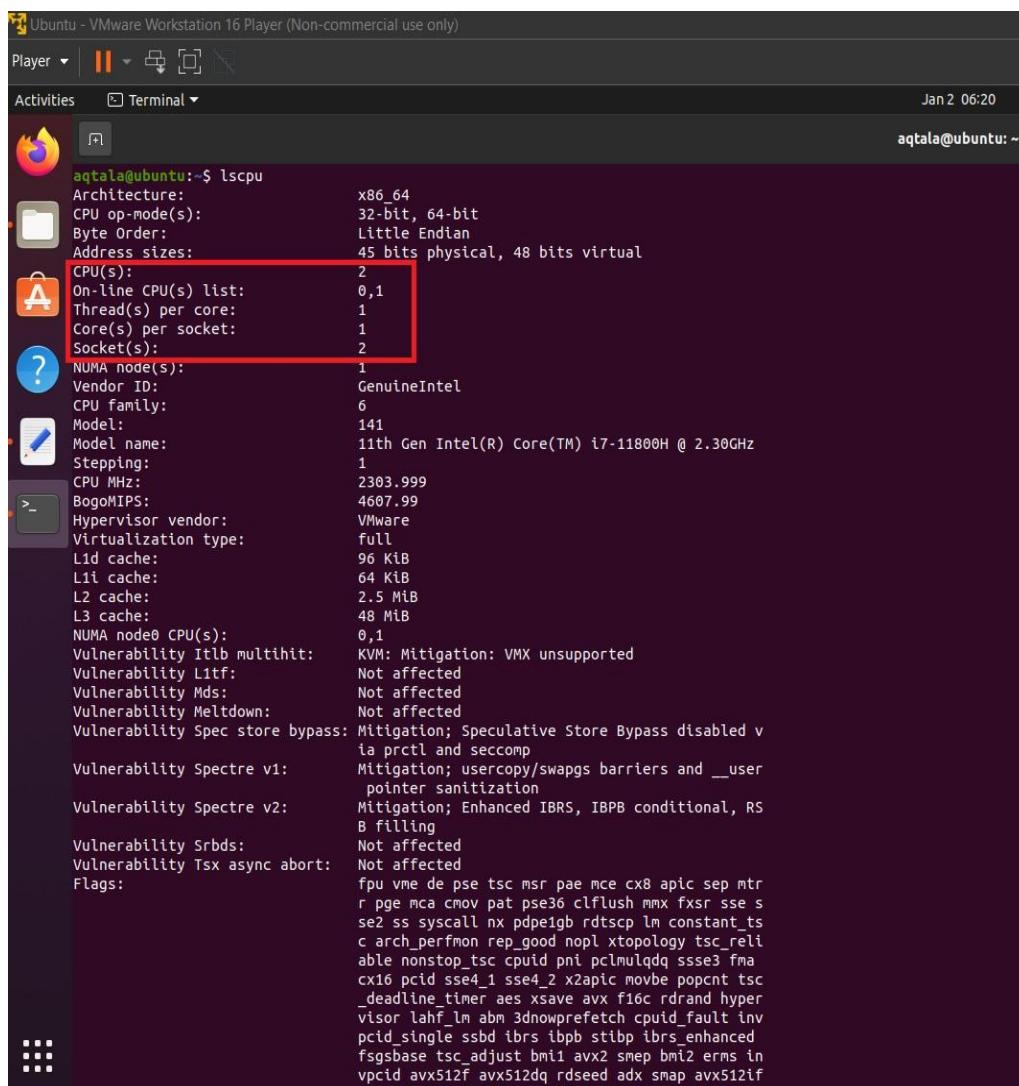
```
aqtala@ubuntu:~/OS_Assignment$ cd OS_Assignment/
aqtala@ubuntu:~/OS_Assignment$ g++ main.cpp -o main.out -lpthread
aqtala@ubuntu:~/OS_Assignment$ time ./main.out 1
ThreadID=0, startLoop=0, endLoop=1024
numOfEven=1048576 numOfOdd=0 totalCells=1048576
real    0m4.456s
user    0m4.403s
sys     0m0.044s
aqtala@ubuntu:~/OS_Assignment$ time ./main.out 2
ThreadID=1, startLoop=1, endLoop=1019
ThreadID=0, startLoop=0, endLoop=1024
numOfEven=1048576 numOfOdd=0 totalCells=1048576
real    0m3.516s
user    0m6.638s
sys     0m0.081s
aqtala@ubuntu:~/OS_Assignment$ time ./main.out 4
ThreadID=2, startLoop=6, endLoop=981
ThreadID=0, startLoop=0, endLoop=1011
ThreadID=1, startLoop=2, endLoop=1021
ThreadID=3, startLoop=1, endLoop=1024
numOfEven=1048576 numOfOdd=0 totalCells=1048576
real    0m3.518s
user    0m6.572s
sys     0m0.108s
aqtala@ubuntu:~/OS_Assignment$ time ./main.out 8
ThreadID=7, startLoop=1, endLoop=1007
ThreadID=0, startLoop=2, endLoop=1009
ThreadID=4, startLoop=6, endLoop=1008
ThreadID=2, startLoop=5, endLoop=1017
ThreadID=0, startLoop=0, endLoop=1016
ThreadID=5, startLoop=3, endLoop=1014
ThreadID=3, startLoop=7, endLoop=1023
ThreadID=1, startLoop=4, endLoop=1024
numOfEven=1048576 numOfOdd=0 totalCells=1048576
real    0m3.079s
user    0m5.741s
sys     0m0.124s
```

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** .



```
aqtala@ubuntu:~$ lscpu
Architecture:          x86_64
CPU op-mode(s):        32-bit, 64-bit
Byte Order:            Little Endian
Address sizes:         45 bits physical, 48 bits virtual
CPU(s):                2
On-line CPU(s) list:  0,1
Thread(s) per core:   1
Core(s) per socket:   1
Socket(s):             2
NUMA node(s):          1
Vendor ID:             GenuineIntel
CPU family:            6
Model:                 141
Model name:            11th Gen Intel(R) Core(TM) i7-11800H @ 2.30GHz
Stepping:               1
CPU MHz:               2303.999
BogoMIPS:              4607.99
Hypervisor vendor:    VMware
Virtualization type:  full
L1 cache:              96 KiB
L1i cache:             64 KiB
L2 cache:              2.5 MiB
L3 cache:              48 MiB
NUMA node0 CPU(s):    0,1
Vulnerability Itlb multihit: KVM: Mitigation: VMX unsupported
Vulnerability L1tf:     Not affected
Vulnerability Mds:     Not affected
Vulnerability Meltdown: Not affected
Vulnerability Spec store bypass: Mitigation: Speculative Store Bypass disabled via prctl and seccomp
Vulnerability Spectre v1: Mitigation: usercopy/swaps barriers and __user pointer sanitization
Vulnerability Spectre v2: Mitigation: Enhanced IBRS, IBPB conditional, RS B filling
Vulnerability Srbds:    Not affected
Vulnerability Tsx async abort: Not affected
Flags:                 fpu vme de pse tsc msr pae mce cx8 apic sep mtr r pge mca cmov pat pse36 clflush mmx fxsr sse sse2 ss syscall nx pdpe1gb rdtscp lm constant_ts arch_perfmon rep_good nopl xtTopology tsc_reliable nonstop_tsc cpuid pnt pclmulqdq ssse3 fma cx16 pcid sse4_1 sse4_2 x2apic movbe popcnt tsc _deadline timer aes xsave avx f16c rdrand hyper visor lahf_lm abm 3dnowprefetch cpuid_fault invpcid_single ssbd ibrs ibpb stibp ibrs_enhanced fsgsbase tsc_adjust bmi1 avx2 smp bmi2 rmrs in vpcid avx512f avx512dq rdseed adx smap avx512if
```

Picture 2

## **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](https://github.com/FerasHamam/OS_Assignment)