LAB 6
Multi-Threading

Student	ID
Abdullah Alhethili	**1855911**

EE463 | Operating Systems

Dr. Abdulghani M. Al-Qasimi

Eng. Turky Saderaldin

2) Are the process ID numbers of parent and child threads the same or different? Why?

They are the same. Both threads created by and belong to the same process. As it can be seen bellow.

```
Parent: My process# ---> 616785
Parent: My thread # ---> 140545566250816
Child: Hello World! It's me, process# ---> 616785
Child: Hello World! It's me, thread # ---> 140545566246464
Parent: No more child thread!
```

4) Does the program give the same output every time? Why?

No, this is probably to a phenomenon known as race condition. Since there are multiple threads accessing the same variable sometimes a thread access it before another changes it and other times after the other thread changes it. This due to them running at the same time and whomever is faster at accessing the data get that data at that point. You can see different results for the same code bellow.

```
Parent: Global data = 5
Child: Global data was 10.
Child: Global data is now 15.
Parent: Global data = 15
Parent: End of program.

[Done] exited with code=0 in 0.116 seconds

[Running] cd "/home/a115/463/Lab/" && gcc Lab6.c -o
Parent: Global data = 5
Child: Global data was 5.
Child: Global data is now 15.
Parent: Global data = 15
Parent: End of program.
```

5) Do the threads have separate copies of glob_data?

As stated in the previous question both threads have the same copy of the glob data variable and that why results differ.

7) Do the output lines come in the same order every time? Why?

No, they are not in the same order. This probably due to the way they were scheduled to access the CPU. Depending on who is scheduled to access the CPU next get to execute their function first. As can be seen below.

```
I am the parent thread
I am thread #0, My ID #139672433296960
I am thread #1, My ID #139672424904256
I am thread #2, My ID #139672416511552
I am thread #3, My ID #139672336463424
I am thread #4, My ID #139672328070720
I am thread #5, My ID #139672319678016
I am thread #6, My ID #139672311285312
I am thread #7, My ID #139672302892608
I am thread #8, My ID #139672294499904
I am thread #9, My ID #139672286107200
I am the parent thread again
```

```
I am the parent thread
I am thread #0, My ID #140578350691904
I am thread #1, My ID #140578342299200
I am thread #2, My ID #140578333906496
I am thread #4, My ID #140578316990016
I am thread #3, My ID #140578325382720
I am thread #5, My ID #140578239018560
I am thread #6, My ID #140578230625856
I am thread #7, My ID #140578222233152
I am thread #8, My ID #140578213840448
I am thread #9, My ID #140578205447744
I am the parent thread again
```

9) Did this_is_global change after the threads have finished? Why?

Yes, because both threads have access to it and inside their function they incremented it.

10) Are the local addresses the same in each thread? What about the global addresses?

No, the local addresses are not the same. However, the global addresses are the same.

```
First, we create two threads to see better what context they share...
Set this_is_global to: 1000
Thread: 139694260454976, pid: 624150, addresses: local: 0X1AC57E34, global: 0XA90FB014
Thread: 139694260454976, incremented this_is_global to: 1001
Thread: 139694252062272, pid: 624150, addresses: local: 0X1A456E34, global: 0XA90FB014
Thread: 139694252062272, incremented this_is_global to: 1002
After threads, this_is_global = 1002
```

11) Did local_main and this_is_global change after the child process has finished? Why?

No, the parents variables stayed the same. The child gets the same copy of the variables the parent has. However, after any modification to the original copy those modifications are saved only to the process that changed them.

```
First, we create two threads to see better what context they share...
Set this_is_global to: 1000
Thread: 140567824439040, pid: 976, addresses: local: 0X7F3E0EDC, global: 0XCC206
07C
Thread: 140567824439040, incremented this_is_global to: 1001
Thread: 140567832831744, pid: 976, addresses: local: 0X7FBE1EDC, global: 0XCC206
07C
Thread: 140567832831744, incremented this_is_global to: 1002
After threads, this_is_global = 1002

Now that the threads are done, let's call fork..
Before fork(), local_main = 17, this_is_global = 17
Parent: pid: 976, lobal address: 0X181FD8E8, global address: 0XCC20607C
Child : pid: 979, local address: 0X181FD8E8, global address: 0XCC20607C
Child : pid: 979, set local_main to: 13; this_is_global to: 23
Parent: pid: 976, local_main = 17, this_is_global = 17
```

12) Are the local addresses the same in each process? What about global addresses? What happened?

Yes, as discussed in the previous question, the variables that the child gets are the exact same ones the parent has with the same address. When the child modified the data in them the modification are saved to his own addresses.

14) How many times the line tot_items = tot_items + *iptr; is executed?

50*50000 = 2,500,000 times

15) What values does *iptr have during these executions?

It varies from thread to thread from 1 to 50**.**

16) What do you expect Grand Total to be?

(1+2+3…+50)*(50000) = 63750000

17) Why you are getting different results?

This due to the race condition discussed in earlier answer. Where to threads access the same resources but sometimes the thread access that variable while the other during the modification process of that variable so they get the wrong copy and overwrite the modified copy made by the other thread.