Operating Systems (EE463)

Lab6: Multi-Thread in C

Name	ID
Abdullah Essam Alghamdi	1939627

Instructor: Eng. Turkey Saderaldin

Ex1:

```
root@lamp ~/Lab6# ./ex1
Parent: My process# ---> 924
Parent: My thread # ---> 139763837777728
Child: Hello World! It's me, process# ---> 924
Child: Hello World! It's me, thread # ---> 139763837773568
Parent: No more child thread!
```

1) After running the code, I noticed that the process ID is the same in the parent and the child threads, but they are different in the thread ID.
2) The process ID of the parent and the child are the same because in multi-threaded program threads share the address and resources. Unlike multi-process programs, where each process has its own separate address space and resources.

Ex2:

```
root@lamp ~/Lab6# ./ex2
Parent: Global data = 5
Child: Global data was 5.
Child: Global data is now 15
Parent: Global data = 15
Parent: End of program.
root@lamp ~/Lab6# ./ex2
Parent: Global data = 5
Child: Global data was 10.
Child: Global data is now 15
Parent: Global data = 15
Parent: End of program.
```

3) The global_data sometimes prints 5 in the child thread which means the child precede the parent in modifying the global_data, and it prints 10 when the parent thread precedes the child thread in modifying the

global_data. This is called a race condition between the parent thread and the child thread in the critical section global_data. To avoid this, we can use mutex to synchronize between the parent and the child in modifying the critical section.

4) No, the program doesn't give the same output because there is a race condition in the critical section global_data.
5) No, the threads do not have separate copies because in the multi-threaded programs the threads shared the same address space and same resources which includes the global variables.

Ex3:

```
root@lamp ~/Lab6# ./ex3
I am the parent thread
I am thread #9, My ID #140042399807232
I am thread #8, My ID #140042408199936
I am thread #6, My ID #140042424985344
I am thread #4, My ID #140042441770752
I am thread #2, My ID #140042458556160
I am thread #0, My ID #140042475341568
I am thread #3, My ID #140042450163456
I am thread #1, My ID #140042466948864
I am thread #5, My ID #140042433378048
I am thread #7, My ID #140042416592640
I am the parent thread again
root@lamp ~/Lab6# ./ex3
I am the parent thread
I am thread #9, My ID #140148036843264
I am thread #7, My ID #140148053628672
I am thread #5, My ID #140148070414080
I am thread #3, My ID #140148087199488
I am thread #1, My ID #140148103984896
I am thread #2, My ID #140148095592192
I am thread #0, My ID #140148112377600
I am thread #4, My ID #140148078806784
I am thread #8, My ID #140148045235968
I am thread #6, My ID #140148062021376
I am the parent thread again
```

6) The Operating system assigns different IDs to the threads each time the program is executed.
7) No, it does not because the threads are assigned randomly by the Operating system.

Ex4:

```
root@lamp ~/Lab6# ./ex4
First, we create two threads to see better what context they share...
Set this_is_global to: 1000
Thread: 140017846535936, pid: 1272, addresses: local: 0X72013EDC, global: 0X4EFE707C
Thread: 140017846535936, incremented this_is_global to: 1001
Thread: 140017854928640, pid: 1272, addresses: local: 0X72814EDC, global: 0X4EFE707C
Thread: 140017854928640, 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: 1272, lobal address: 0X46D146F8, global address: 0X4EFE707C
Child : pid: 1275, local address: 0X46D146F8, global address: 0X4EFE707C
Child : pid: 1275, set local_main to: 13; this_is_global to: 23
Parent: pid: 1272, local_main = 17, this_is_global = 17
```

8) In the multi-threaded program:
    1. Each thread has its own ID.
    2. All the threads have the same PID.
    3. The address of the local variable in each thread is different.
    4. The address of the global variable is the same in all the threads.
    5. When incrementing the global variable, all the threads share the same address of this variable.

    In the multi-processing program:

    1. Each parent-child has its own PID.
    2. The local and global addresses are the same.
    3. Each parent and child have its own copy of the local and global variables, they are not sharing it together.

9) Yes, it changed because the threads share the same address of the global variable.
10) In the threads the local addresses change, but the global addresses are the same.
11) The local_main and this_is_global are changed in the copy of the child process only because the child process has its own copy of the variables.
12) They are the same in all processes, but each process copies the variables and use it within the process itself.

Ex5:

```
root@lamp ~/Lab6# ./ex5
End of Program. Grand Total = 41929232
root@lamp ~/Lab6# ./ex5
End of Program. Grand Total = 42792662
root@lamp ~/Lab6# ./ex5
End of Program. Grand Total = 52852330
root@lamp ~/Lab6# ./ex5
End of Program. Grand Total = 45344605
root@lamp ~/Lab6# ./ex5
End of Program. Grand Total = 50439326
```

13) The final answer of the global variable will be different each time the program is executed.
14) We have 50 threads each thread has 50,000 iterations. So, 50 * 50,000 = 2,500,000
15) Since each thread has a unique integer data value ranging from 1 to 50, *iptr can have any of these 50 values during its 50,000 iterations in the loop. The actual value of *iptr depends on the specific thread that is executing the thread_func function during the loop iterations.
16) The actual Grand Total = 63,750,000
17) There is a race condition between the thread each time the program executed.