Lab 6

#	Names:	ID	Section
1	Ahmed Alkhayal	1945541	C3

Instructor:
Dr. Abdulghani M. Al-Qasimi

Date: 6/6/2023

1) Run the above program, and observe its output:

```
Parent: My process# ---> 16000
Child: Hello World! It's me, process# ---> 16000
Child: Hello World! It's me, thread # ---> 1
Parent: My thread # ---> 2
Parent: No more child thread!
PS C:\Users\Admin\Desktop\activity1>
```

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

No, they are not. We can tell because each thread has its own special ID, and the program's output confirms this.

3) Run the above program several times; observe its output every time. A sample output follows:

```
Parent: Global data = 5
Child: Global data was 5.
Child: Global data is now 15.
Parent: Global data = 15
Parent: End of program.
PS C:\Users\Admin\Desktop\activity1>  &
-ebi0myx3.l2h' '--stdout=Microsoft-MIEn
n\gdb.exe' '--interpreter=mi'
Parent: Global data = 5
Child: Global data was 5.
Child: Global data is now 15.
Parent: Global data = 15
Parent: End of program.
PS C:\Users\Admin\Desktop\activity1>  &
-dwi1xyww.lmt' '--stdout=Microsoft-MIEn
n\gdb.exe' '--interpreter=mi'
Parent: Global data = 5
Child: Global data was 5.
Child: Global data is now 15.
Parent: Global data = 15
Parent: End of program.
PS C:\Users\Admin\Desktop\activity1>
```

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

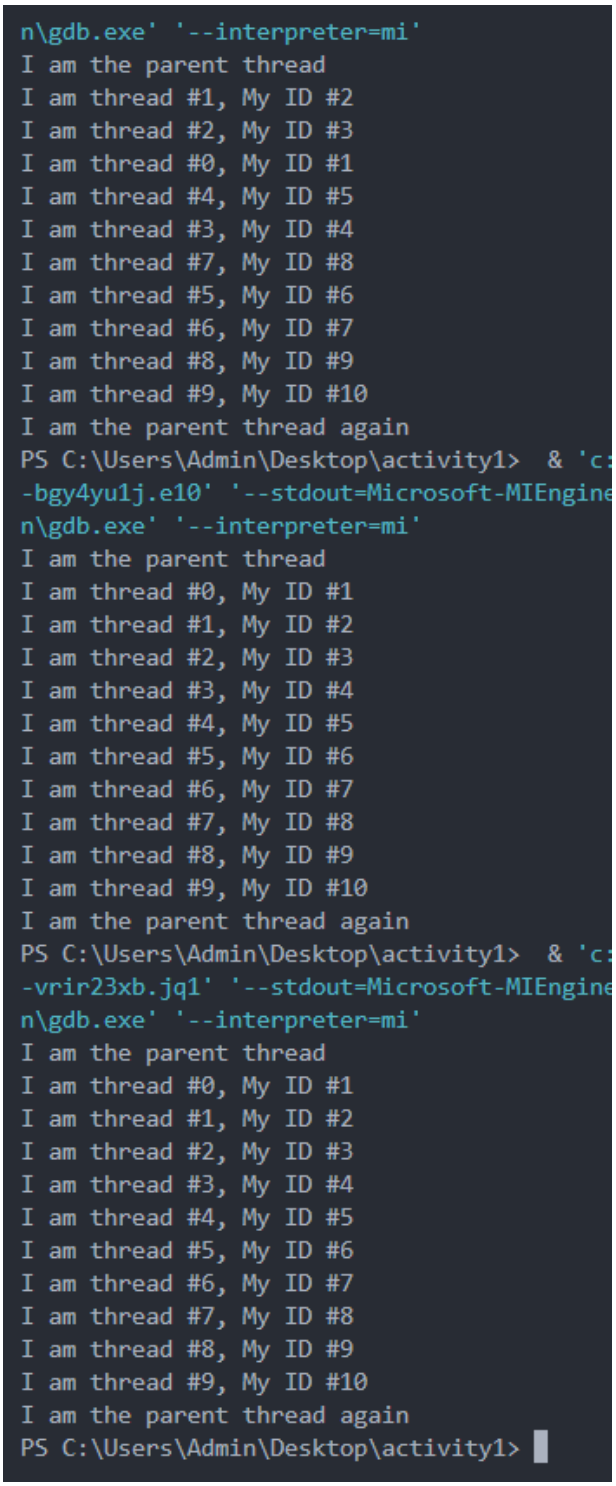
Yes, but there is no guarantee that it will consistently produce the same output every time. In this case, the output happened to be the same because my system completed the thread's task before the main thread modified the glob_data. However, it is important to note that this occurrence is

not guaranteed to happen consistently in all systems. I have tested it multiple times, and it happened in my case.

5) Do the threads have separate copies of glob_data?

No, threads have a shared memory.

6) Run the above program several times and observe the outputs:



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

No, thread creating time and priority differs every time, so it will not have the same order every time.

8) Run the above program and observe its output. Following is a sample output:

```
First, we create two threads to see better what context they share...
Set this_is_global to: 1000
Thread: 139751728932416, pid: 18, addresses: local: 0X7C28DE34, global: 0X4812B014
Thread: 139751728932416, incremented this_is_global to: 1001
Thread: 139751720539712, pid: 18, addresses: local: 0X7BA8CE34, global: 0X4812B014
Thread: 139751720539712, 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: 18, lobal address: 0XEDE2A1DC, global address: 0X4812B014
Child : pid: 24, local address: 0XEDE2A1DC, global address: 0X4812B014
Child : pid: 24, set local_main to: 13; this_is_global to: 23
Parent: pid: 18, local_main = 17, this_is_global = 17
```

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

Yes, that is correct. We have modified it after invoking the join() function for all the threads.

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

Local addresses differ, but global addresses are indeed the same.

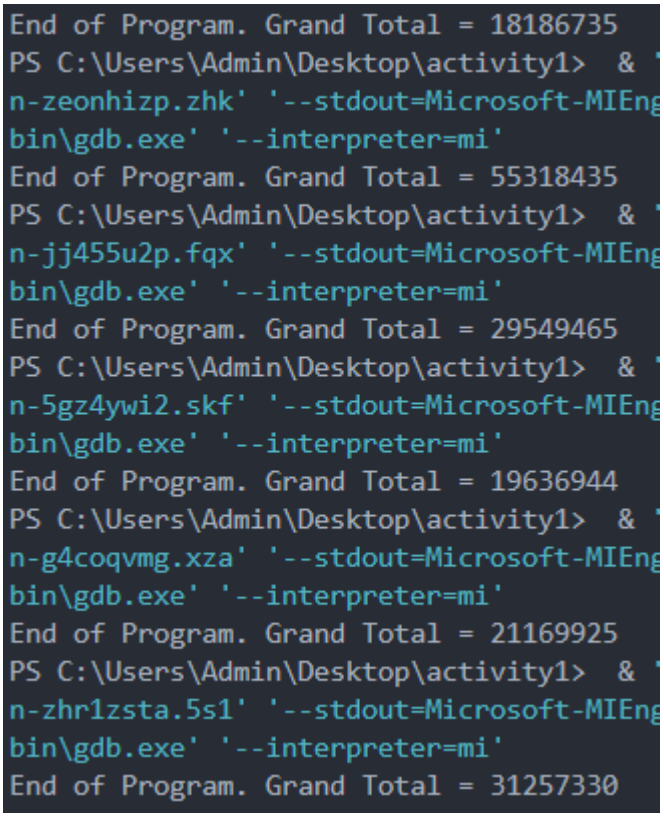
11) Did local_main and this_is_global change after the child process has finished? Why?

Yes, once the child has completed its task, it will wait for the parent to finish. Therefore, the parent can modify them after the child has finished its job.

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

Yes, processes have separate memory spaces, which means they do not share the same memory space. As a result, they can have different memory addresses. Therefore, it is possible for the addresses to be the same.

13) Run the above program several times and observe the outputs, until you get different results.



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

We have a total of 50 threads, with each thread executing it 50,000 times. Therefore, the overall execution count is 50 * 50,000, which equals 2,500,000 times.

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

It refers to the value of the data in the tidrec struct specific to the thread.

16) What do you expect Grand Total to be?

By summing the numbers from 1 to 50 and then multiplying the result by 50,000, we obtain the value of 63,750,000.

17) Why you are getting different results?

The tot_items variable is susceptible to a race condition. Since all threads attempt to modify it simultaneously, the value does not increment in an orderly manner as expected. Consequently, the output varies and cannot be considered accurate.