King Abdul Aziz University

Faculty of Engineering

2023

EE-463

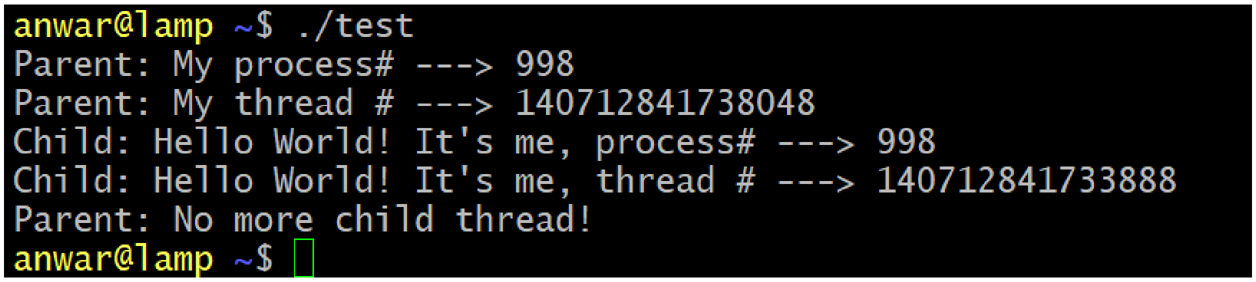
Operating Systems

Lab-6

M/No	Name	ID
1	Anwar Shukri Shawli	1937123

Instructor: Dr. Abdulghani Al-Qasimi

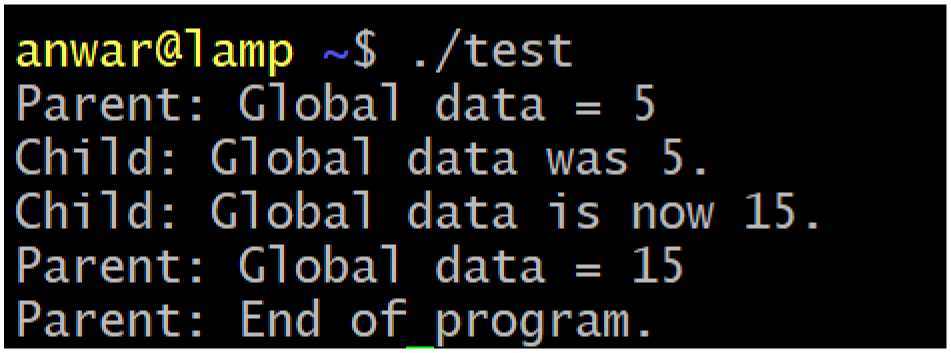
1- Run the above program, and observe its output:



```
anwar@lamp ~$ ./test
Parent: My process# ---> 998
Parent: My thread # ---> 140712841738048
Child: Hello World! It's me, process# ---> 998
Child: Hello World! It's me, thread # ---> 140712841733888
Parent: No more child thread!
anwar@lamp ~$
```

2- Are the process ID numbers of parent and child threads the same or different? Why?
The process ID numbers of the parent and child threads are the same because they belong to the same process.

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

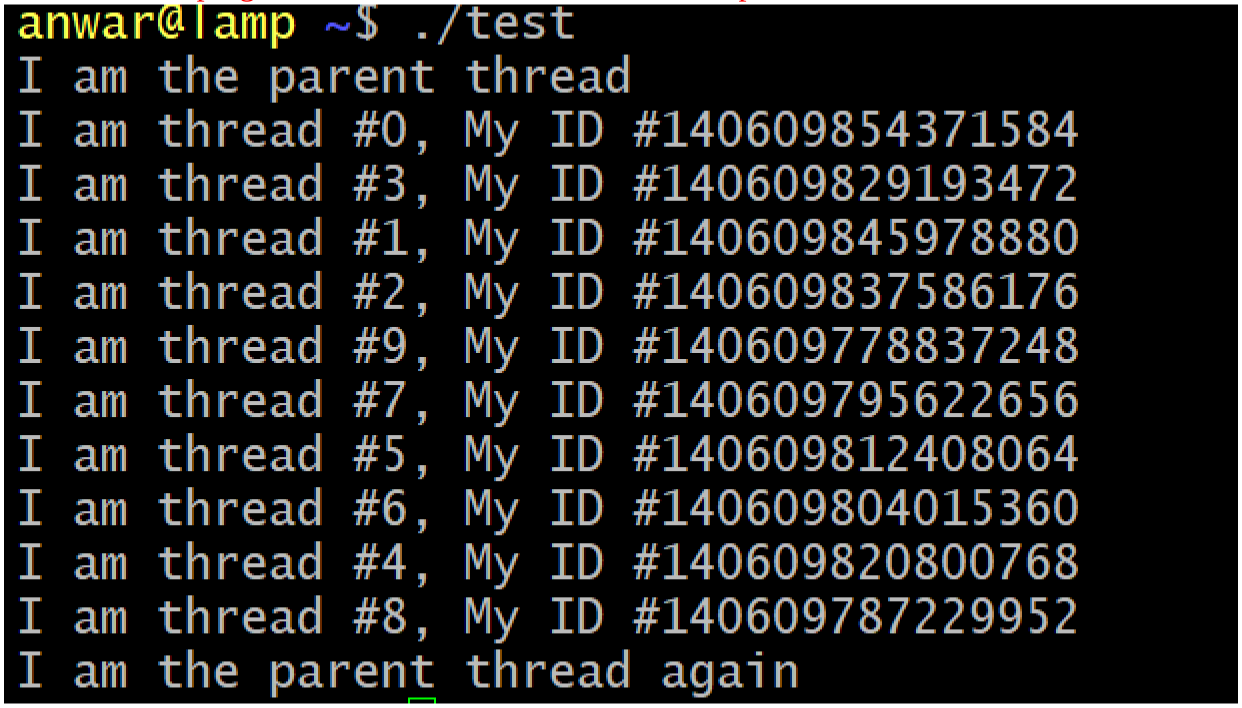


```
anwar@lamp ~$ ./test
Parent: Global data = 5
Child: Global data was 5.
Child: Global data is now 15.
Parent: Global data = 15
Parent: End of program.
```

4- Does the program give the same output every time? Why?
No, the program may not always give the same output every time. The exact behavior depends on the scheduling of the threads by the operating system. Since there is no synchronization mechanism (such as mutex or semaphore) used in this code, there is a possibility of a race condition. Race conditions can lead to unpredictable results, and the order in which the threads execute and access the glob_data variable may vary from run to run.

5- Do the threads have separate copies of `glob_data`?
No, the threads do not have separate copies of glob_data. The glob_data variable is a global variable, which means it is shared among all threads in the program. Any changes made to glob_data by one thread are immediately visible to all other threads. In this code, the change thread function modifies the value of glob_data, and the main thread (parent) also modifies it. Both threads access and modify the same global variable.

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

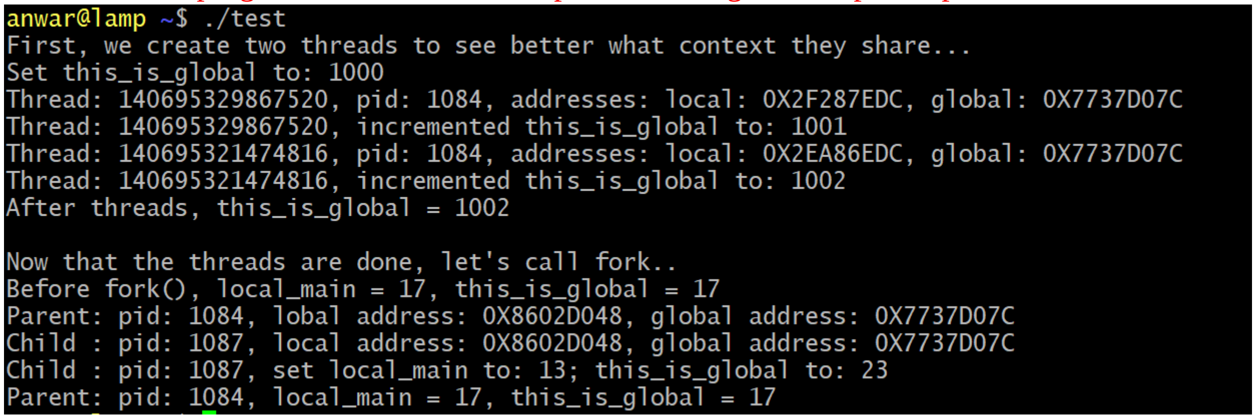


```
anwar@lamp ~$ ./test
I am the parent thread
I am thread #0, My ID #140609854371584
I am thread #3, My ID #140609829193472
I am thread #1, My ID #140609845978880
I am thread #2, My ID #140609837586176
I am thread #9, My ID #140609778837248
I am thread #7, My ID #140609795622656
I am thread #5, My ID #140609812408064
I am thread #6, My ID #140609804015360
I am thread #4, My ID #140609820800768
I am thread #8, My ID #140609787229952
I am the parent thread again
```

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

No, the output lines may not come in the same order every time. The order of thread execution and the scheduling of threads are determined by the underlying operating system and can be non-deterministic. The threads run concurrently and may be scheduled in a different order each time the program is executed.

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



```
anwar@lamp ~$ ./test
First, we create two threads to see better what context they share...
Set this_is_global to: 1000
Thread: 140695329867520, pid: 1084, addresses: local: 0X2F287EDC, global: 0X7737D07C
Thread: 140695329867520, incremented this_is_global to: 1001
Thread: 140695321474816, pid: 1084, addresses: local: 0X2EA86EDC, global: 0X7737D07C
Thread: 140695321474816, 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: 1084, lobal address: 0X8602D048, global address: 0X7737D07C
Child : pid: 1087, local address: 0X8602D048, global address: 0X7737D07C
Child : pid: 1087, set local_main to: 13; this_is_global to: 23
Parent: pid: 1084, local_main = 17, this_is_global = 17
```

9- Did `this_is_global` change after the threads have finished? Why?

Yes, the value of this_is_global changed after the threads have finished. The initial value was 1000, and each thread incremented it by 1. Therefore, the final value of this_is_global is 1002.

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

No, the local addresses are different in each thread. Each thread has its own stack space, so the addresses of local variables (local_thread) are different. However, the global address (this_is_global) is the same for all threads since it refers to the same global variable in the shared memory.

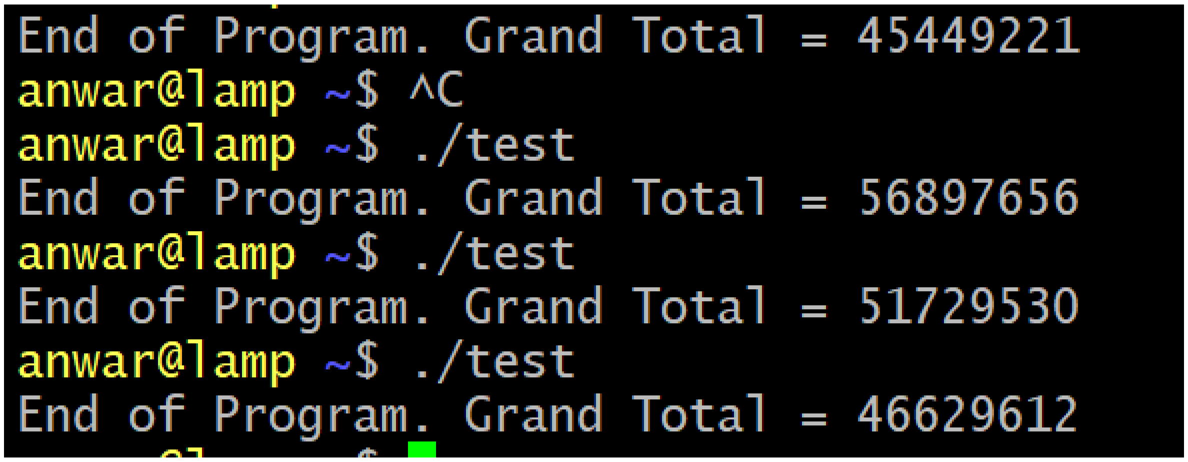
11- Did local_main and this_is_global change after the child process has finished? Why?

No, the values of local_main and this_is_global did not change after the child process has finished. The child process modified its own copies of these variables, which were separate from the copies in the parent process. Therefore, the values of local_main and this_is_global remain unchanged in the parent process.

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

No, the local addresses are different in each process. When the child process is created using fork, it gets its own separate memory space. Therefore, the addresses of local variables (local_main) are different in the child process compared to the parent process. However, the global address (this_is_global) remains the same in both processes since they refer to the same global variable in shared memory. The child process inherits the value of this_is_global from the parent process.

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



```
End of Program. Grand Total = 45449221
anwar@lamp ~$ ^C
anwar@lamp ~$ ./test
End of Program. Grand Total = 56897656
anwar@lamp ~$ ./test
End of Program. Grand Total = 51729530
anwar@lamp ~$ ./test
End of Program. Grand Total = 46629612
```

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

The line tot_items = tot_items + *iptr; is executed 50,000 times for each of the 50 threads, so it is executed a total of 50,000 * 50 = 2,500,000 times.

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

The values of *iptr during these executions range from 1 to 50. Each thread is assigned a unique data value ranging from 1 to 50 when the thread is created. The *iptr pointer points to this data value.

16- What do you expect Grand Total to be?

By adding together all the data values and multiplying by 50,000, we can determine the expected grand total. To calculate the expected Grand Total, we can use the formula for the sum of the first n natural numbers (n * (n + 1) / 2) and multiply it by 50,000 (the number of iterations per thread). Here, n = 50 (the number of threads):

Sum = 50 * (50 + 1) / 2 = 50 * 51 / 2 = 1275
Now, multiply the sum by the number of iterations per thread:
Grand Total = 1275 * 50,000 = 63,750,000
the Grand Total should consistently be 63,750,000.

17- Why you are getting different results?

The reason for getting different results is that the code suffers from a race condition. Multiple threads are accessing and modifying the tot_items variable concurrently without any synchronization mechanism.