King Abdul Aziz University

EE-463

LAB #6

8-6-2023

BANDER SAEED ALSOLAMI
1935181

EX1

Run the above program, and observe its output:

```
bander@lamp ~$ ./ex1
Parent: My process# ---> 1317
Parent: My thread # ---> 140493006272320
Child: Hello World! It's me, process# ---> 1317
Child: Hello World! It's me, thread # ---> 140493006268160
Parent: No more child thread!
```

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 both threads belong to the same process. A process can have multiple threads, but all threads within a process share the same process ID.

EX2

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

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

Does the program give the same output every time? Why

No. The program may not give the same output every time it is executed because the order in which the parent and child threads execute is non-deterministic and may vary based on factors such as the scheduling algorithm used by the operating system and .the workload of other processes running on the system

Do the threads have separate copies of glob_data?

No. The threads share the same copy of glob_data because it is a global variable, which means that it can be accessed and modified by any thread in the program.

EX3

Run the above program several times and observe the outputs

```
bander@lamp ~$ ./ex3
I am the parent thread
I am thread #3, My ID #140045511431936
I am thread #9, My ID #140045461075712
I am thread #7, My ID #140045477861120
I am thread #8, My ID #140045469468416
I am thread #5, My ID #140045494646528
I am thread #0, My ID #140045536610048
I am thread #2, My ID #140045519824640
I am thread #4, My ID #140045503039232
I am thread #6, My ID #140045486253824
I am thread #1, My ID #140045528217344
I am the parent thread again
```

```
bander@lamp ~$ ./ex3
I am the parent thread
I am thread #1, My ID #140364096186112
I am thread #0, My ID #140364104578816
I am thread #2, My ID #140364087793408
I am thread #3, My ID #140364079400704
I am thread #4, My ID #140364071008000
I am thread #5, My ID #140364062615296
I am thread #6, My ID #140364054222592
I am thread #7, My ID #140364045829888
I am thread #8, My ID #140364037437184
I am thread #9, My ID #140364029044480
I am the parent thread again
```

```
bander@lamp ~$ ./ex3
I am the parent thread
I am thread #4, My ID #139761820419840
I am thread #9, My ID #139761778456320
I am thread #7, My ID #139761795241728
I am thread #5, My ID #139761812027136
I am thread #3, My ID #139761828812544
I am thread #1, My ID #139761845597952
I am thread #2, My ID #139761837205248
I am thread #8, My ID #139761786849024
I am thread #6, My ID #139761803634432
I am thread #0, My ID #139761853990656
I am the parent thread again
```

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

No, the output lines do not come in the same order every time because the order in which the threads execute is not deterministic and can vary each time the program is run.

EX4

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

```
bander@lamp ~$ ./ex4
First, we create two threads to see better what context they share...
Set this_is_global to: 1000
Thread: 140671886345984, pid: 1546, addresses: local: 0XB9D0BEDC, global: 0XA29
8807C
Thread: 140671886345984, incremented this_is_global to: 1001
Thread: 140671894738688, pid: 1546, addresses: local: 0XBA50CEDC, global: 0XA29
8807C
Thread: 140671894738688, 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: 1546, lobal address: 0X26917C58, global address: 0XA298807C
Child : pid: 1549, local address: 0X26917C58, global address: 0XA298807C
Child : pid: 1549, set local_main to: 13; this_is_global to: 23
Parent: pid: 1546, local_main = 17, this_is_global = 17
```

Did this_is_global change after the threads have finished? Why?

Yes, this_is_global changed after the threads had finished. This is because this_is_global is a global variable shared by all threads, and each thread increments its value by 1. Since there are two threads, the final value of this_is_global is 1002.

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

The local addresses are not the same in each thread because each thread has its own stack, and local variables are stored on the stack. The global addresses, on the other

hand, are the same in each thread because global variables are stored in a shared memory region that is accessible to all threads.

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

Yes, this_is_global changed after the child process finished, but local_main did not. When a new process is created using fork(), the child process gets a copy of the parent's memory, including all variables. Changes made to variables in the child process do not affect the parent process. In this case, the child process changed the .value of this_is_global to 23, but this change is not reflected in the parent process

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

The local addresses are different in each process because each process has its own address space. The global addresses are also different for the same reason. When a process is forked, the child process gets a copy of the parent's address space, but they are still separate and changes made in one do not affect the other.

EX5

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

```
bander@lamp ~$ ./ex5
End of Program. Grand Total = 39656818
bander@lamp ~$ ./ex5
End of Program. Grand Total = 43055916
bander@lamp ~$ ./ex5
End of Program. Grand Total = 51140027
bander@lamp ~$ ./ex5
End of Program. Grand Total = 41280425
bander@lamp ~$ ./ex5
End of Program. Grand Total = 48132714
bander@lamp ~$ ./ex5
End of Program. Grand Total = 43216409
```

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

The line tot_items = tot_items + *iptr; is executed 50 * 50000 = 2,500,000 times in total. This is because the line is inside a loop that runs 50000 times, which is executed by each of the 50 threads created in the program.

What values does *iptr have during these executions?

*iptr takes values from 1 to 50 during the executions of tot_items = tot_items + *iptr.;

What do you expect Grand Total to be?

The value of Grand Total is not deterministic due to a race condition when multiple threads modify the shared variable tot_items without synchronization.

Why you are getting different results?

The program gets different results for Grand Total due to a race condition when multiple threads modify the shared variable tot_items without synchronization. The final value of tot_items is dependent on the non-deterministic scheduling of threads and can vary between runs.