



# King Abdulaziz University

## Faculty of Engineering



### EE463

## Operating Systems

### Semester 3 - 2023

### Lab #6

Name	ID
Nahar Khayyat	1936718

1)

```
nahar@lamp ~$ ./lab6
Parent: My process# ---> 823
Parent: My thread # ---> 140075492198208
Child: Hello World! It's me, process# ---> 823
Child: Hello World! It's me, thread # ---> 140075492194048
Parent: No more child thread!
```

2) The process ID numbers of the parent and child threads will be the same. This is because both the parent and child threads are running within the same process.

3)

```
nahar@lamp ~$ ./lab6.2
Parent: Global data = 5
Child: Global data was 10.
Child: Global data is now 15.
Parent: Global data = 15
Parent: End of program.
```

4) The program may not give the same output every time due to the unpredictable nature of thread scheduling. The operating system's scheduler decides when each thread runs, so the order of execution may vary between runs.

5) No, the threads do not have separate copies of glob\_data. The variable glob\_data is a global variable, which means it is shared among all threads within the process.

6)

```
nahar@lamp ~$ ./lab6.3
I am the parent thread
I am thread #1, My ID #139628992907008
I am thread #0, My ID #139629001299712
I am thread #2, My ID #139628984514304
I am thread #3, My ID #139628976121600
I am thread #4, My ID #139628967622400
I am thread #5, My ID #139628959229696
I am thread #6, My ID #139628950836992
I am thread #7, My ID #139628942444288
I am thread #8, My ID #139628934051584
I am thread #9, My ID #139628925658880
I am the parent thread again
```

```
nahar@lamp ~$ ./lab6.3
I am the parent thread
I am thread #0, My ID #139950893246208
I am thread #2, My ID #139950876460800
I am thread #4, My ID #139950859675392
I am thread #5, My ID #139950851282688
I am thread #6, My ID #139950771140352
I am thread #7, My ID #139950762747648
I am thread #8, My ID #139950754354944
I am thread #9, My ID #139950745962240
I am thread #3, My ID #139950868068096
I am thread #1, My ID #139950884853504
I am the parent thread again
```

7) The output lines may not come in the same order every time. This is because the execution order of the threads is determined by the operating system's scheduler, which can vary between runs. The scheduler decides when to switch between threads, and this can lead to different orders of execution for the threads in different runs of the program.

8)

```
nahar@lamp ~$ ./lab6.4
First, we create two threads to see better what context they share...
Set this_is_global to: 1000
Thread: 140083869259520, pid: 1076, addresses: local: 0XD1440EDC, global: 0XD4B5907C
Thread: 140083869259520, incremented this_is_global to: 1001
Thread: 140083860866816, pid: 1076, addresses: local: 0XD0C3FEDC, global: 0XD4B5907C
Thread: 140083860866816, 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: 1076, local address: 0X509A95F8, global address: 0XD4B5907C
Child : pid: 1079, local address: 0X509A95F8, global address: 0XD4B5907C
Child : pid: 1079, set local_main to: 13; this_is_global to: 23
Parent: pid: 1076, local_main = 17, this_is_global = 17
```

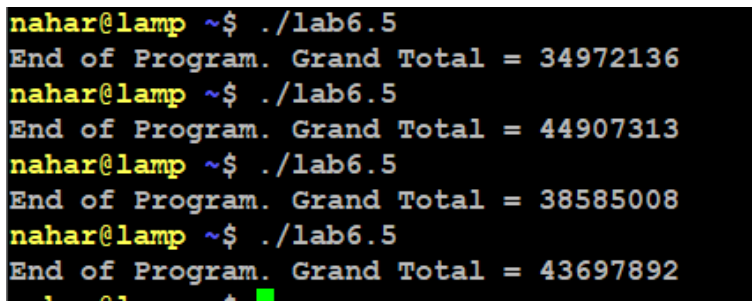
9) Yes, `this_is_global` changes after the threads have finished. This is because both threads increment the global variable `this_is_global`.

**10)** The local addresses in each thread are different. Each thread has its own stack, and the local variables are stored on the thread's stack. Thus, the local variables' addresses are different for each thread.

**11)** `local_main` does not change after the child process has finished, but `this_is_global` does change. This is because, when a new process is created using `fork()`, the child process gets a copy of the parent process's memory space. Any changes made by the child process to the memory space (including local and global variables) are not reflected in the parent process's memory space.

**12)** The local addresses are the same in each process, but they point to different memory locations. This is because, when a new process is created using `fork()`, the child process gets a copy of the parent process's memory space.

**13)**



```
nahar@lamp ~$ ./lab6.5
End of Program. Grand Total = 34972136
nahar@lamp ~$ ./lab6.5
End of Program. Grand Total = 44907313
nahar@lamp ~$ ./lab6.5
End of Program. Grand Total = 38585008
nahar@lamp ~$ ./lab6.5
End of Program. Grand Total = 43697892
```

**14)** The line `tot_items = tot_items + *iptr;` is executed 50,000 times for each of the 50 threads, resulting in a total of 2,500,000 executions.

**15)** The value of `*iptr` will vary from 1 to 50, as it is set to the value of `(m+1)` in the for loop that creates the threads.

**16)** The expected Grand Total is the sum of the arithmetic series from 1 to 50, multiplied by 50,000 (the number of iterations in each thread). The sum of the arithmetic series from 1 to 50 is  $(50 * (50 + 1)) / 2 = 1275$ . So, the expected Grand Total is  $1275 * 50,000 = 63,750,000$ .

**17)** We are getting different results because of race conditions between the threads. Multiple threads are accessing and modifying the global variable `tot_items` simultaneously, leading to unpredictable behavior.