BLG 312E HW1
Cem Yusuf Aydoğdu – 150120251

1) Following output is observed. When the fork command is executed, a child process of parent process is constructed with same context and added to the "ready queue", and a separate address space which is same as parents, is assigned to the child.
In this case, a process id which is one more than parent process id is assigned to its child. After "`fork();`", firstly the parent process is executed, but because of the "`wait(NULL);`" command, processor paused and skipped parent process, then executed child process, then returned to the parent process again.

```
Parent: My process ID: 12476
Parent: My child's process ID: 12477
   Child: My process ID: 12477
   Child: My parent's process ID: 12476
Parent: Terminating...
------------------
(program exited with code: 0)
Press return to continue
```

2) Parent's parent process id is obtained as near compared to its process id, which shows that parent's parent process has other child processes.

```
Parent: My process ID: 12681
Parent: My parent's process ID: 12679
Parent: My child's process ID: 12682
   Child: My process ID: 12682
   Child: My parent's process ID: 12681
Parent: Terminating...
------------------
(program exited with code: 0)
Press return to continue
```

3) The parent process is executed and closed firstly, then the child process is executed. Child process has a different parent process id because its parent is terminated before its execution, which means another process is assigned as its parent.

```
Parent: My process ID: 13399
Parent: My parent's process ID: 13397
Parent: My child's process ID: 13400
Parent: Terminating...
------------------
(program exited with code: 0)
Press return to continue
   Child: My process ID: 13400
   Child: My parent's process ID: 2219
```

4) Variable is defined and assigned before fork, both child and parent accessed value normally.

```
Parent: My process ID: 16226
number = 5
Parent: My parent's process ID: 16224
Parent: My child's process ID: 16227
   Child: My process ID: 16227
   number = 5
   Child: My parent's process ID: 16226
Parent: Terminating...
------------------
(program exited with code: 0)
Press return to continue
```

Different values are assigned to number in parent and child. Since there are two variables for both parent and child in their discrete address spaces, they accessed their variables distinctly.

```
Parent: My process ID: 25640
number = 7
Parent: My parent's process ID: 25638
Parent: My child's process ID: 25641
   Child: My process ID: 25641
   number = 2
   Child: My parent's process ID: 25640
Parent: Terminating...
------------------
(program exited with code: 0)
Press return to continue
```

5) A pointer to global value is allocated and assigned the value 3 before fork. Addresses are same, because of the fork command which copies parents virtual address space to the child.
Then, the parent process is accessed to the pointer in its own address space and changed the value it addresses as 32. After that, child process is accessed its distinct address space and changed the value to 16.

```
before fork: address= 0x1241010 number = 3

Parent: My process ID: 29280
address= 0x1241010 number = 32
Parent: My parent's process ID: 29278
Parent: My child's process ID: 29281
   Child: My process ID: 29281
   address= 0x1241010 number = 16
   Child: My parent's process ID: 29280
Parent: Terminating...
------------------
(program exited with code: 0)
Press return to continue
```

6) In p1.c, fork is used three times, then processes are executed. Every process increased their own tmp variables, so values are 1.

```
0: Value= 1
1: Value= 1
2: Value= 1
Main: Created 3 procs.
------------------
(program exited with code: 0)
Press return to continue
```

In p2.c, three threads are created and executed, and all threads increased the same variable. Order of the threads are observed different in each run.

```
main(): Created 3 threads.
2: Value= 1
1: Value= 2
0: Value= 3
------------------
(program exited with code: 0)
Press return to continue
```