Jonathan Carreiro – Lab 6

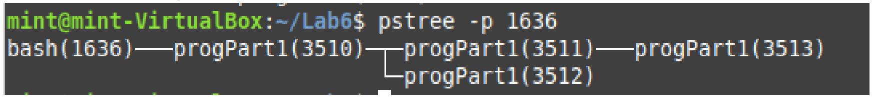
Part 1:

a. Four processes are created.

b. Eight processes are created now.

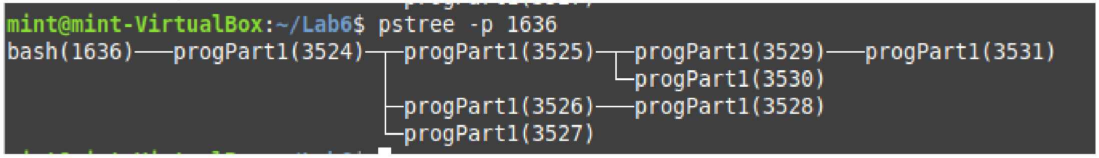
c. The relationship between the number of forks and the processes created is 2^n where n is the number of forks.

d. When running with two forks:



```
mint@mint-VirtualBox:~/Lab6$ pstree -p 1636
bash(1636)───progPart1(3510)─┬─progPart1(3511)───progPart1(3513)
                             └─progPart1(3512)
```

When running with three forks:



```
mint@mint-VirtualBox:~/Lab6$ pstree -p 1636
bash(1636)───progPart1(3524)─┬─progPart1(3525)─┬─progPart1(3529)───progPart1(3531)
                             │                 └─progPart1(3530)
                             ├─progPart1(3526)───progPart1(3528)
                             └─progPart1(3527)
```

When the sleep statement is commented, you cannot run pstree properly (no tree displays)

e. The parents terminated first, creating orphan processes

Part 2:

a. The print order is parent, child, child, parent. This is because the parent has a wait after its first print statement. This means that the parent must wait for the child to finish all its prints before finishing its own.

b. The values printed are parent: 5, child: 5, child: 6, parent: 5. This is because the parent has the value initialized at 5 before creating the child. This value is inherited by the child, printing 5. Then, when the child increments n, it only increments in the child, resulting in a 6 for the child while staying a 5 for the parent.

c. Both processes print to the output file. This is because the file descriptor is stored as an int which gets shared to the child upon creation (like n in the precious parts). Another reason is that the processes are both part of the same family so they are allowed to access the file at the same time.

d. Both processes print and the order is “parent” then “child” most of the time. I would assume that this is because the parent process reaches its write statement first since it would be a fraction of a second ahead of its child process (unless interrupted). This is why wait statements are important.