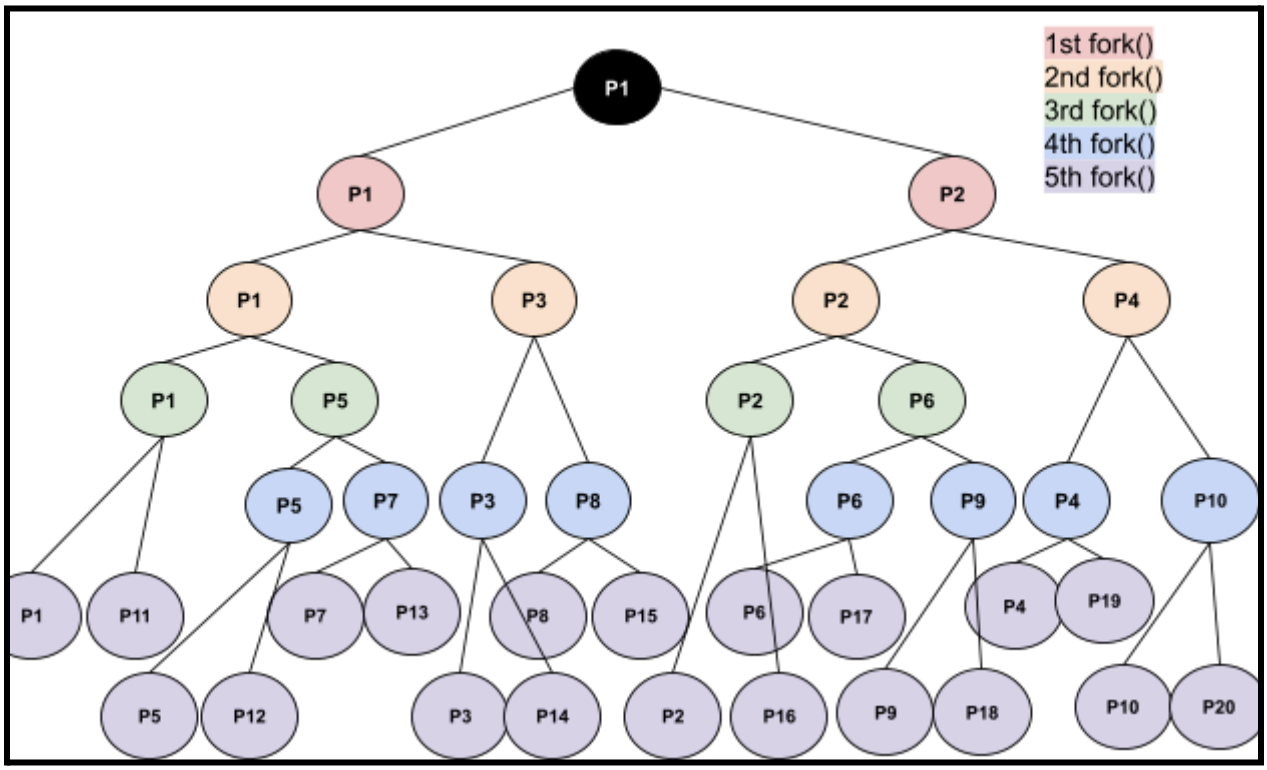
1a. There are 20 processes created from the program. With every successful fork() call, it splits the processes into two parts. One part, the parent process, will return the PID. The second part, the child process, will return 0. The first and second fork() calls apply to every process, giving us a total of 4 processes. The third fork() signaled by the && operand, will only apply to parent processes because their values are not zero. This will give us a total of 6 processes. The fourth fork() signaled by the || operand will only apply to children processes because they have a value of zero. Lastly, the fifth fork() call applies to all 10 processes which gives us a total of 20 processes.

1b. assignment01_Duff_April_Question1.c

1c. i) Processes are color coded by what fork() call created them

ii) The processes produced by the fifth fork() call do not have any children because the program has terminated. These processes are all children processes: P11 - P20.

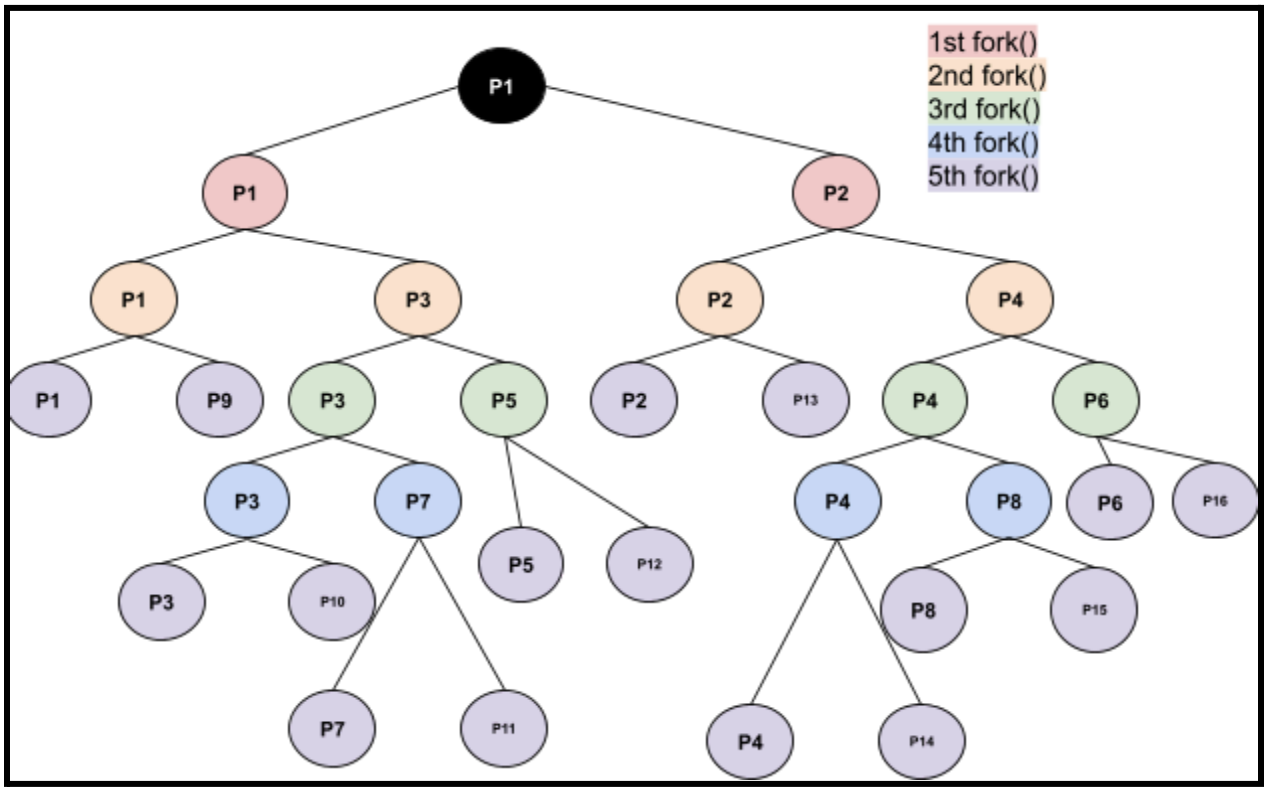


2a. There are 16 processes created by the program. The first and second fork() calls apply to every process which ends with a total of four processes. The third fork() is then called on the processes whose value is zero (child processes P3 and P4). The fork() call after the && operand now only applies to parent processes we just made with the previous fork(). This gives us 8 processes so far. The last fork() is then applied to every process, which results in a total of 16 processes.

2b. assignment01_Duff_April_Question2.c

2c. i) Processes are color coded by what fork() call created them

ii) The processes produced by the fifth fork() call do not have any children because the program has terminated. These processes are all children processes: P9 - P16.

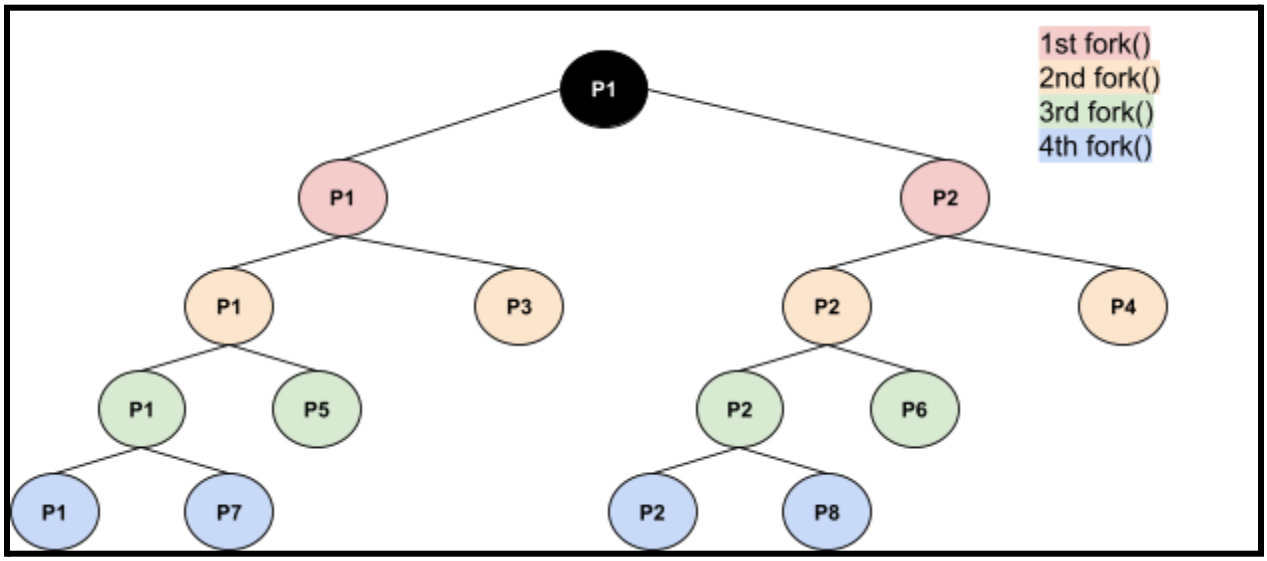


3a. There are 8 processes created by the program. The first and second fork() calls apply to each process resulting in two then four processes. The third fork() is only applied if the process value is not zero; therefore, the children do not get forked. This will give us 6 total processes. The last fork() call within the if statement is applied to parents because they do not have a value of zero. Since there are only two parent processes, they will get forked resulting in a total of 8 processes.

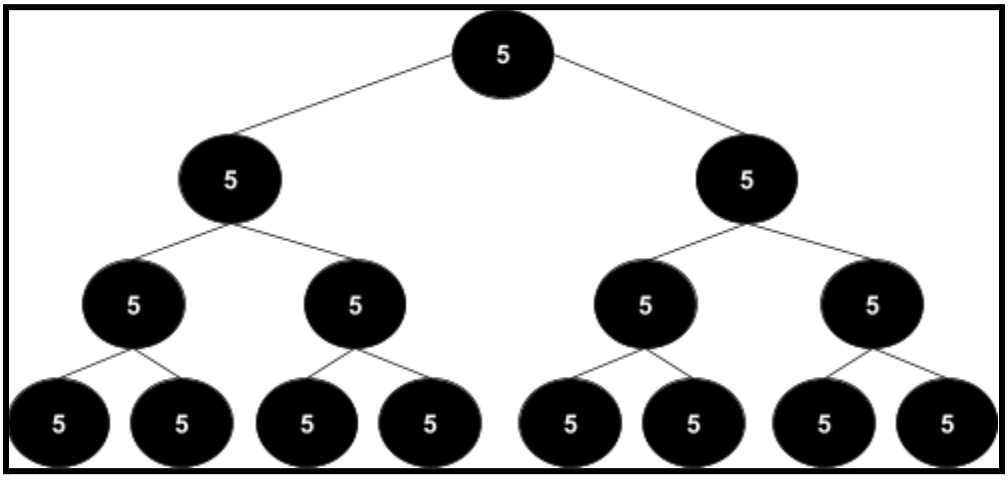
3b. assignment01_Duff_April_Question3.c

3c. i) Processes are color coded by what fork() call created them

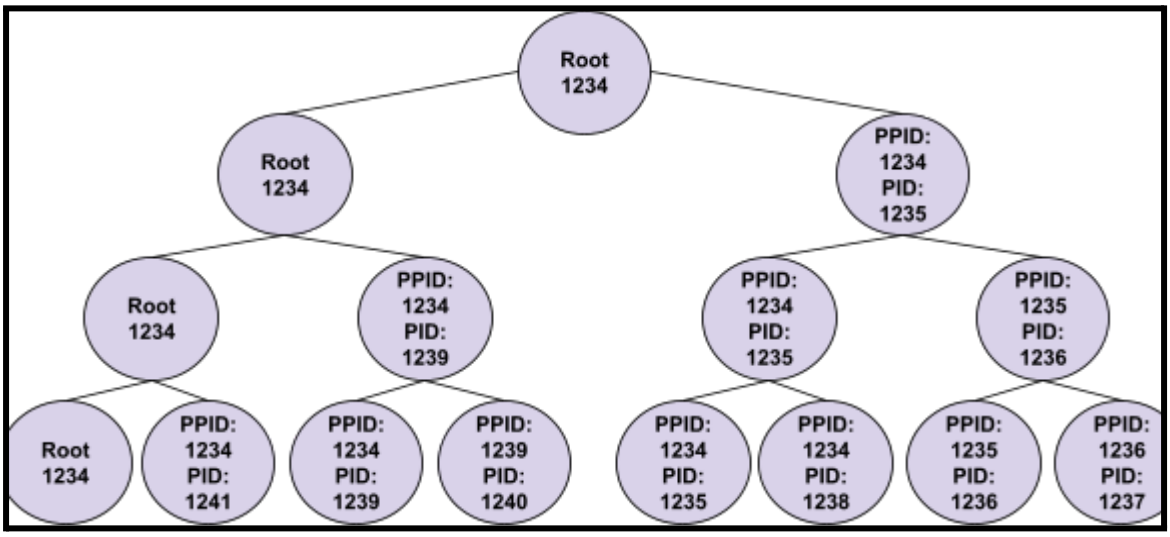
ii) Like before, the processes resulting from the last fork() call do not have any children. In this program, there are also processes P3 - P6 that did not produce any children processes. That is because there is an if statement that did not apply to these processes.



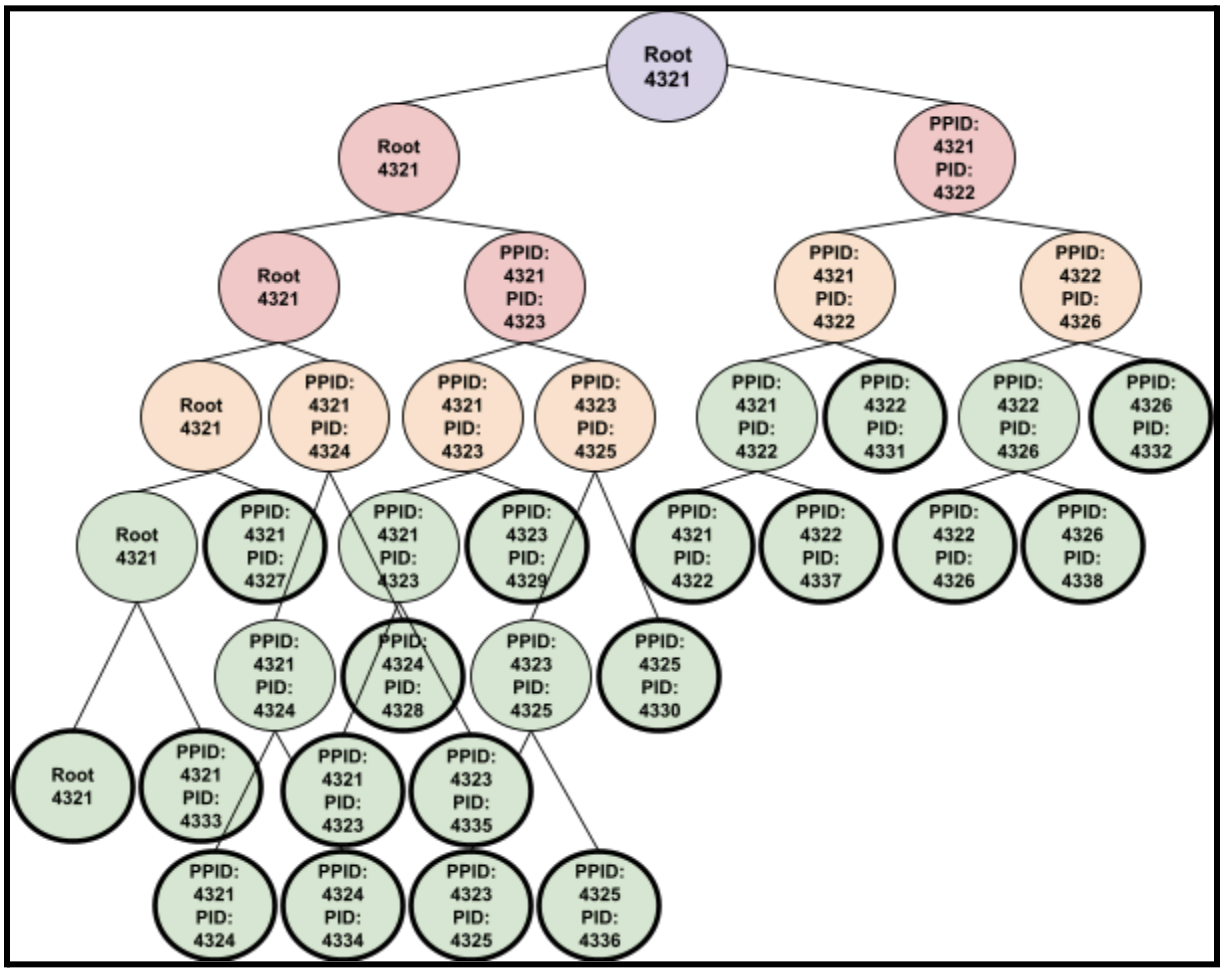
4. The only possible value of x for each process is 5. When the fork() call is successful, the parent processes data gets copied into the child process. Therefore, when x decremented by one and each process was forked, they all stayed the same value.



5. Draw the process tree diagram clearly showing the process ID and the parent process ID at each node in the tree. Assume the root process has a process ID of 1234.



6a. Draw the process tree diagram clearly showing the process ID and the parent process ID at each node in the tree. Assume the root process has a process ID of 4321.



6b. The word “Hello” will be displayed 18 times. The if loop is initialized to run three times. The first fork() call is applied to each process every time (so three times). The fork() call within the if statement is only applied if both of the following conditions are true: the node is a parent (does not equal zero) and the loop number is divisible by 2. With the for loop running three times starting at 0, it can be said that the inside fork() call will be executed 2 out of the three loops. When it is executed, it will fork() all parent processes.

7. Draw the process tree diagram clearly showing the process ID and the parent process ID at each node in the tree. Also at each node, show the value of the "variable". Assume the root process ID is 5521.