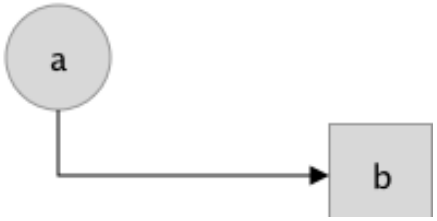
CSC 369 Worksheet 2 Solution

August 16, 2020

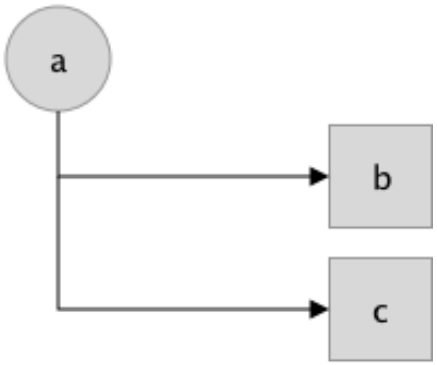
1 Homework (Simulation)

1. I need to create process trees at each step when the command `./fork.py -s 10` is run.

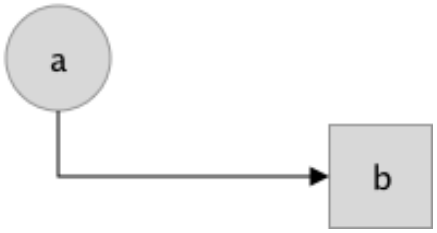
1) Action: a forks b



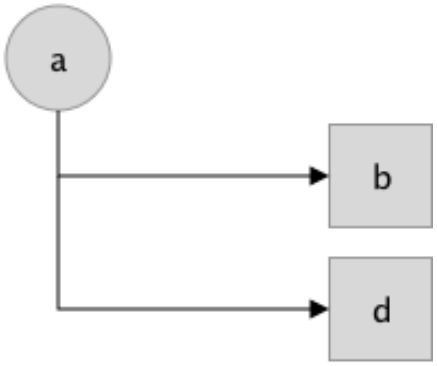
2) Action: a forks c



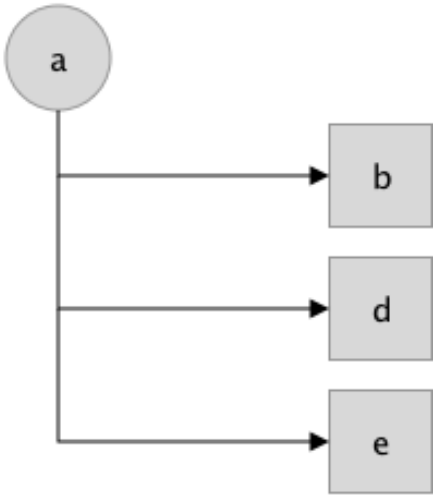
3) Action: c EXITS



4) Action: a forks d



5) Action: a forks e



Notes

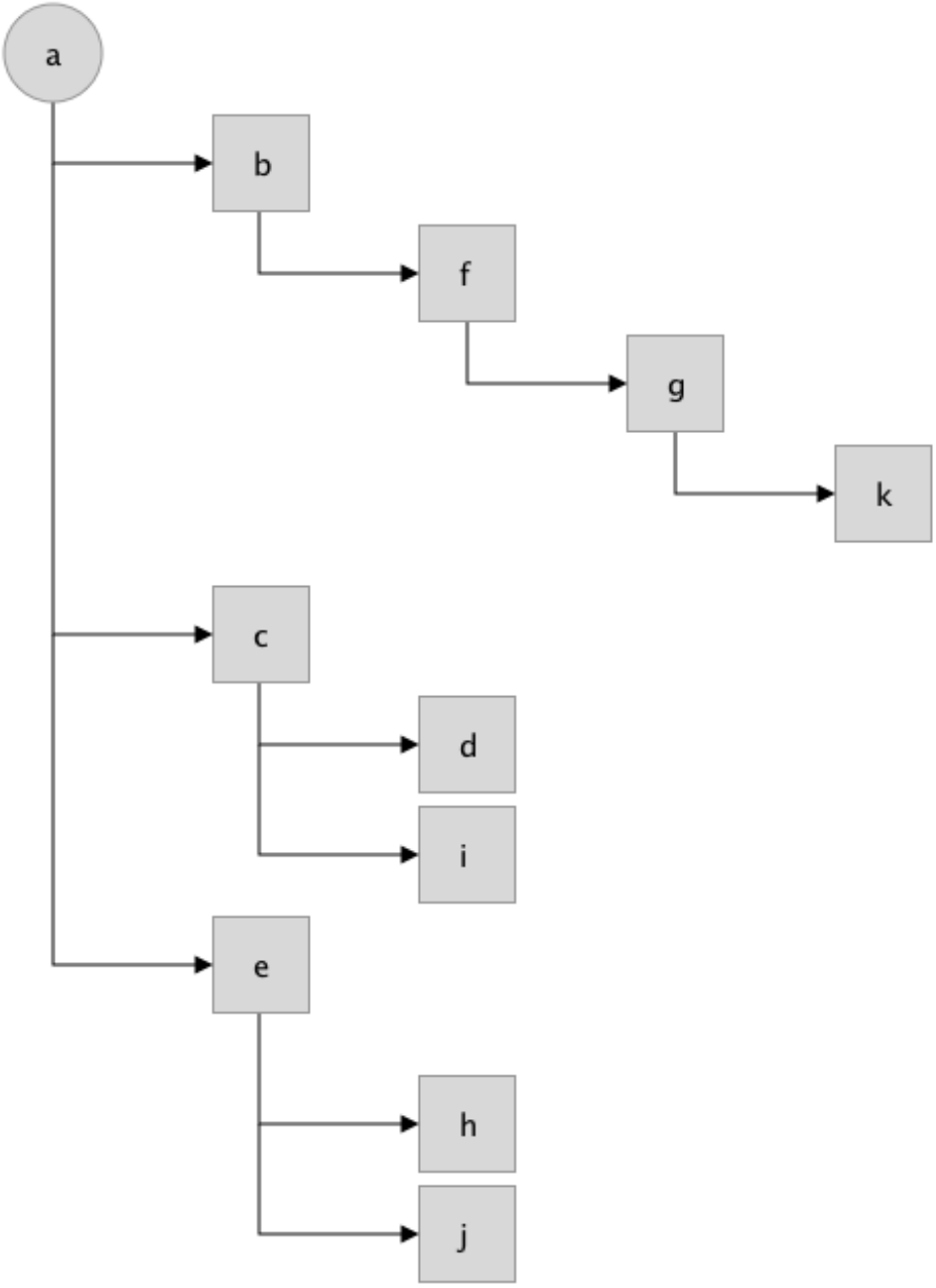
- **fork()**
  - Is used to create a new process
  - **Creator** → parent process
  - **Newly Created** → child process
  - Child process is nearly identical to parent process
- **exec()**
  - Allows a child to break free from its similarity to its parent and execute an entirely new program.
- **wait()**
  - Is used to let parent code delay its execution until the child finishes executing.
  - Makes the output deterministic

2. I need to write what the resulting final process trees will look like as the fork-percentage changes. Here I ran command (`./fork.py -s 10 -a 10 -f 0.1` and `./fork.py -s 10 -a 10 -f 0.9`)

Notes

- `./fork.py -s 10 -a 10 -f 0.1`

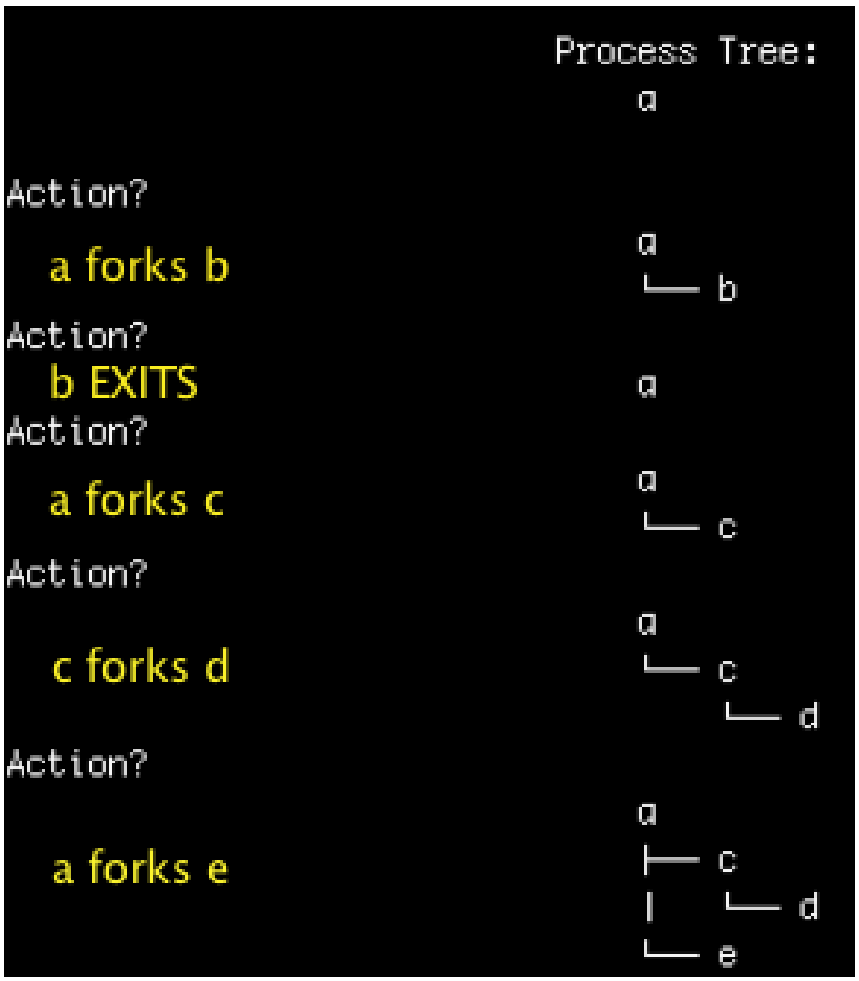
- `./fork.py -s 10 -a 10 -f 0.9`



Based on the diagram above, I can deduce that the lower the fork percentage, the more likely that `exit()` is executed by the childmost process, and the final tree will either have a single node or none.

On the other hand, the higher the fork-percentage is, the more likely that `fork()` is executed by the childmost process, and the final tree will have nodes that are deeply nested.

3. I need to fill out blank entries created by the command (`./fork.py -t`)



4. I need to write what happens when a child exits; what happens to its children in the process tree.

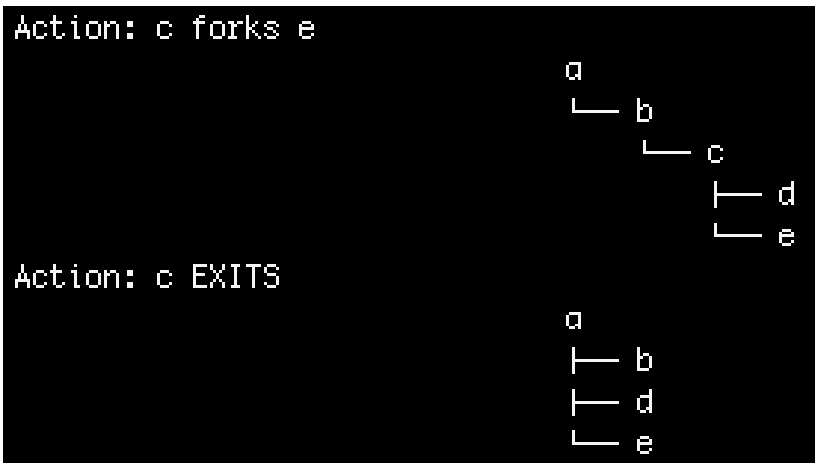
When a child exists, all of its children will also exit.

I am not sure what happens when `-R` flag is used.

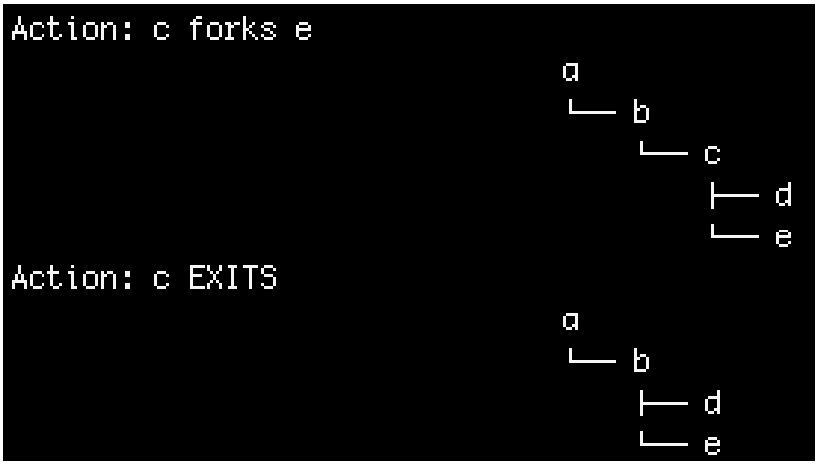
Correct Solution

I need to write what happens when a child exits; what happens to its children in the process tree.

When a child exists, its parentmost child, along with its children, will be attached to the parentmost node



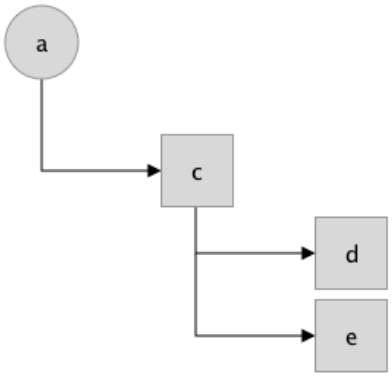
When `-R` flag is used (i.e `./fork.py -A a+b,b+c,c+d,c+e,c- -R`) and a child exists, its parentmost child, along with its children will be attached to the parent node of the child that exits



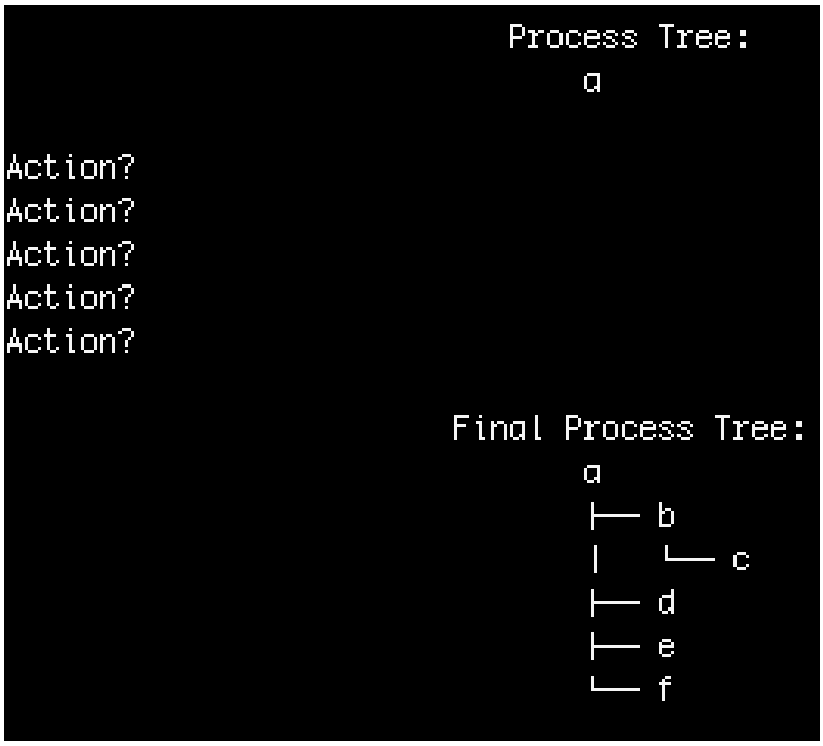
5. I need to write down the final tree by looking at the series of actions generated (here, the command `./fork.py -F` is used).

```
                          Process Tree:
                              a

Action: a forks b
Action: b EXITS
Action: a forks c
Action: c forks d
Action: c forks e

                        Final Process Tree?
```

Answer:



6. First, I need to fill the actions that took place given the final process tree.



Given the final diagram, the missing actions are:

1. Action: a forks b
2. Action: b forks c
3. Action: a forks d
4. Action: a forks e
5. Action: a forks f

Second, I need to write whether I can determine the exact actions that took place, and write where can I tell and cannot tell.

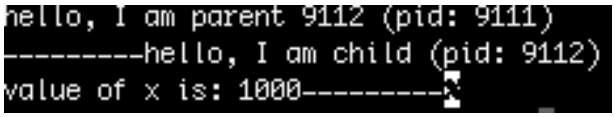
No. I cannot tell exact actions that took place. I can tell what happened upto the latest visible node in the diagram (e.g a, b, c, d, e, f in above diagram), but I cannot tell actions that took place afterwards (e.g. Action: f forks g, Action: a forks h, Action: h EXITS, and Action: g EXITS).

2 Homework (Code)

1. Let $x = 1000$.

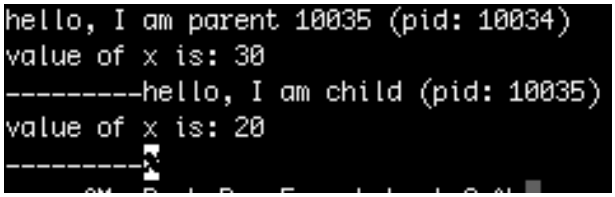
   First, I need to write the value of the variable x in the child process.

   The value of x in child process is the same as the parent (source code is provided in `question_7_part_1.c`).



   Second, I need to write what happens to variable x when both child and the parent change the value of x (source code is provided in `question_7_part_2.c`).

   When the value of x is changed in both child and parent, each possess their own values as if it's their own.



   Notes

   - C file can be compiled via command `gcc -o OUTPUT_FILE_NAME SOURCE_FILE_NAME.c`

2. Yes. Both the child and parent can access the file descriptor returned by `open()`.

   When they are writing to file concurrently, parent's write() is considered before children.