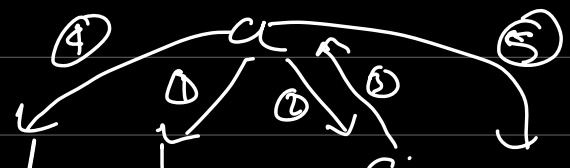
① ./fork.py -s 10

Process Tree:



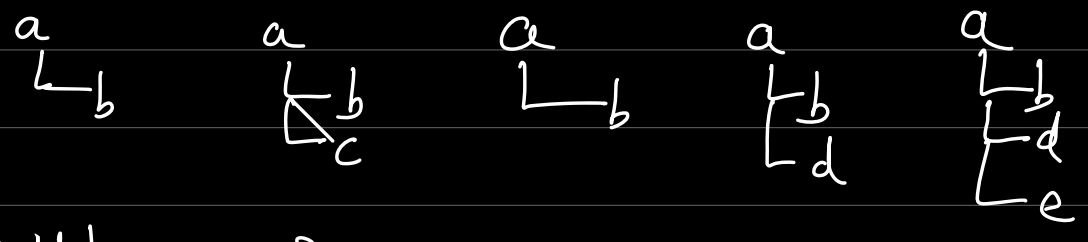
a forks b

a forks c

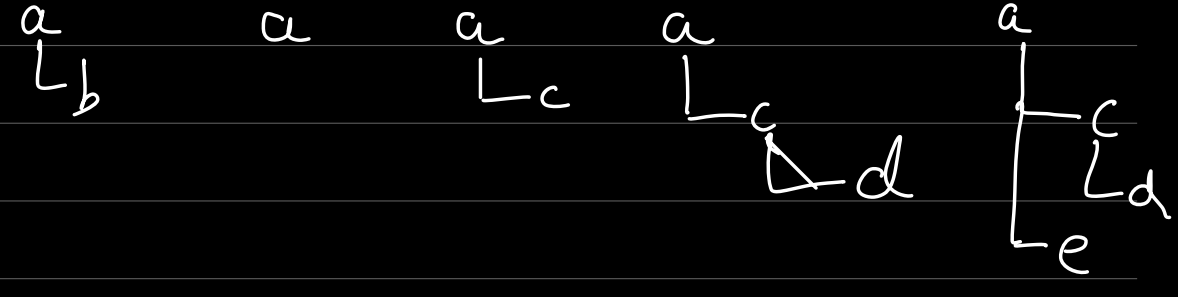
a forks d

a forks e

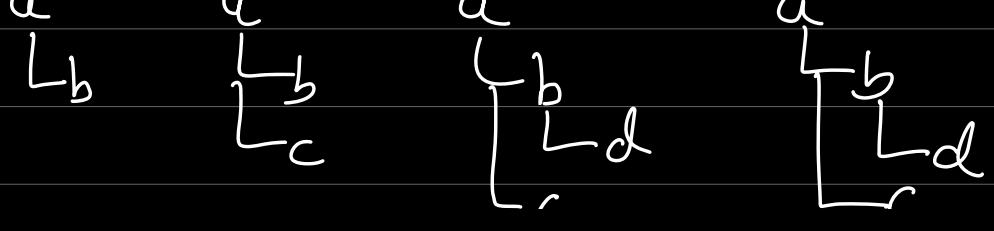
Cleaner way, "-c" showed

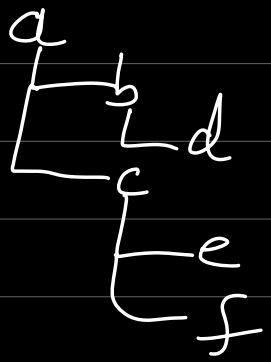


With -s 2



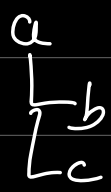
With -s 7

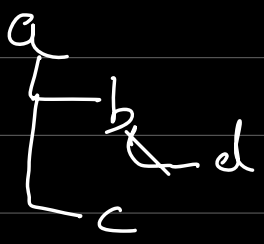


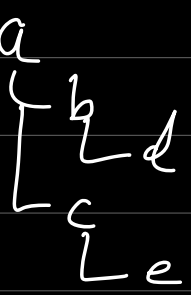


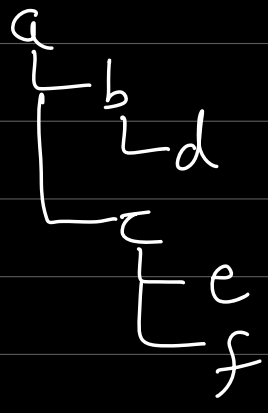
No. of actions

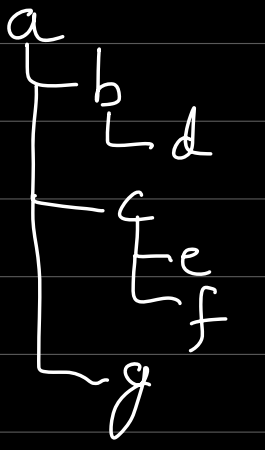
With -s 7 -a 9

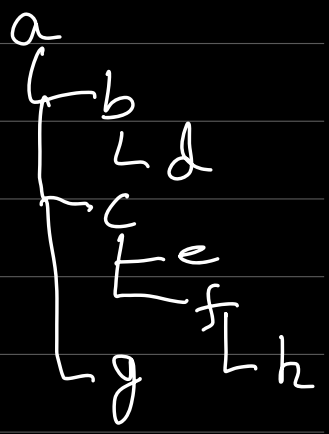


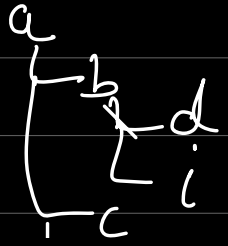


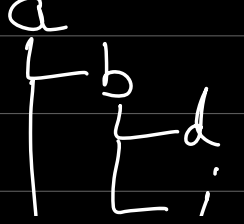


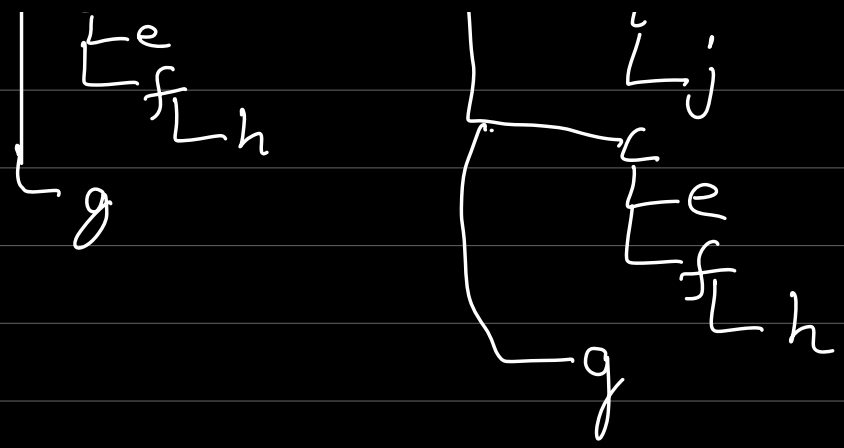












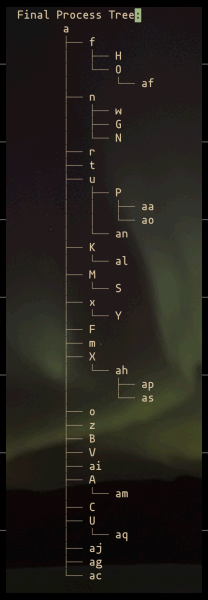
② fork_percentage = f

With higher percentage the process tree becomes denser and lower percentage with more exits, sparse the process tree.

Like for 0.1

```
Final Process Tree:
a
```

Like for 0.7 = see number of processes are high since lesser exits more forks



(3) Now can we find the actions from the process trees "-t"

a
└ b ⇒ a forks b

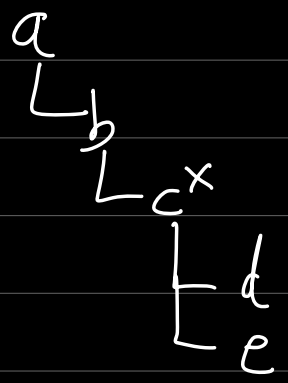
a
├ b
└ c ⇒ a forks c

a
└ c ⇒ b exits

a ⇒ c exits

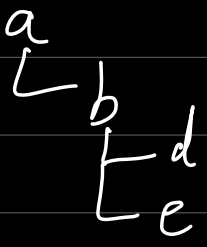
a
└ d ⇒ a forks d

(4) Child exits, what happens to its children?



I was expecting "b" is new parent instead of root "a" process

Actually using -"R" => reparent to local parent with -R



With orphaned processes it turns out the "init" process one with PID 1 becomes the parent.

Systemd => It provides a system and service manager that runs as PID 1 and starts the rest of the system.

Reaping => Cleaning up resources used by a child process after its

terminated. Parent does this so for orphaned child processes new parent (PID 1) does it.

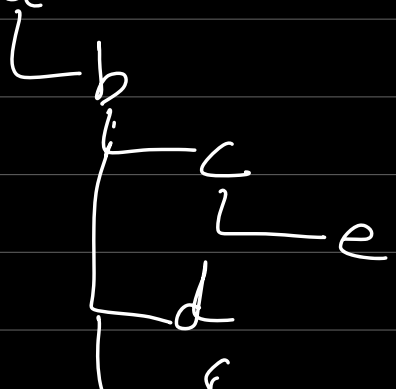
When a process exits, it enters a zombie state until its parent process retrieves its exit status, made possible using "wait()" or "waitpid()"

- Remove zombie process from process table
- Free up allocated resources

(5) "-F" flag final process tree

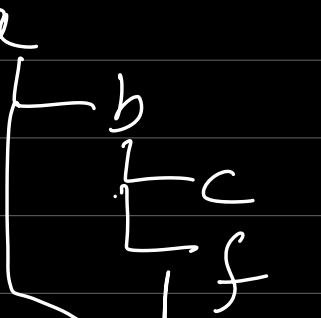
(6) "-t", "-F"

-> a



Actions
- a forks b
- b forks c
- c forks e
- b forks d
- b forks f

-> 3



Actions
- a forks b
- b forks c
- b forks f
- a forks d
- d forks e

Lets reduce fork_percentage = 0.3

Actions
- a forks d

looking at "d" you guess that b, c were processes but exited.
- You can't know exactly which

parent they had
- lets write anyway
few guesses

- a forks b
- a forks c
- c forks d
- c exits
- b exits

since d becomes orphan
goes to "init" ("a")

- a forks b
- b exits
- a forks c
- c exits
- a forks d

Homework (Code)

Code in notes repo

(1) Same value of variable in child process.
When value of variable changed in child and parent, both have different values due to separate address space

(2) Yes, both can access the file descriptor. Though printing of content moves the file pointer to end and needs to be reset to read the whole file. Writing to file adds data from both processes (child & parent)

(3) Need to use wait/waitpid to ensure deterministic execution. Without it I'm not aware but a search shows potential ways like signal, pipes. Basically a form of process

synchronization.