Kevin Andrade
Assignment 1

1)

	Process state		
	P1	P2	P3
① P1 is loaded into memory and starts executing in `main()`.	running	x	x
② P1 calls `fork()` and creates P2, but P1, the parent, keeps running.	running	Ready	x
③ P1 issues an I/O request; P2 starts executing at the return from `fork()`.	waiting	running	x
④ P2 calls `fork()` to create P3; P2 keeps running.	waiting	running	Ready
⑤ P2's time slice expires; P3 starts running.	waiting	Ready	running
⑥ P1's I/O completes (but there is no other changes)	Ready	Ready	running
⑦ P3 waits for user input. P1 runs.	running	ready	waiting

2) it would not be a wise idea to remove processes from the system manually. The reason is because the final (terminated) state allows other processes (such as the parent that created the process) to examine the return code of the just finish process (successful or unsuccessful). Normally when a process finishes, the parent makes a final call (e.x wait()) to wait for the child (hence getting rid of the terminated state would be terrible) to finish, and to indicate to the OS that it can clean up any related data structures that were referred to the now extinct process.

3)

1. Yes
2. Yes
3. No. where there are three D's sequentially there is no connection between the processes that would connect the parent to child.
4. No. After the first 2 sequential D's the corresponding c is not attached to a nearby parent. Processes do not connect.
5. Yes.
6. There are 8 processes and since variables such as I are copied, the largest value would be 4.

4)

1) ABCDDCDDBCDDCDD

2) 8 processes still (waiting will not affect the process creation, just the way they are executed)

Kevin Andrade
Assignment 1

5)

1. The first option is better. Switching to another process while p does io is much more efficient than waiting for p to be done and not using that time to do something else. On the simulator it took 9 ticks of time to wait for p to be done and then execute the other process. On the contrary when I ran it with another process running while waiting for the IO, it was much faster at only 6 ticks.

2. I would not say that IO_RUN_IMMEDIATE is “always” better. When running processes on the simulator, I noticed that it did not make a difference if I used IO_RUN_IMMEDIATE or IO_RUN_LATER if there was only one process besides the IO. It made a huge difference when there were multiple io’s and processes running. RUN_IMMEDIATE was much faster when there was multiple io’s by not spending time in the ready step and interrupting another process to run the IO, and every time the IO was waiting the other processes were running. In the IO_RUN_LATER once the IO went from waiting to ready it waited for the other processes to finish and then ran the remaining IO.