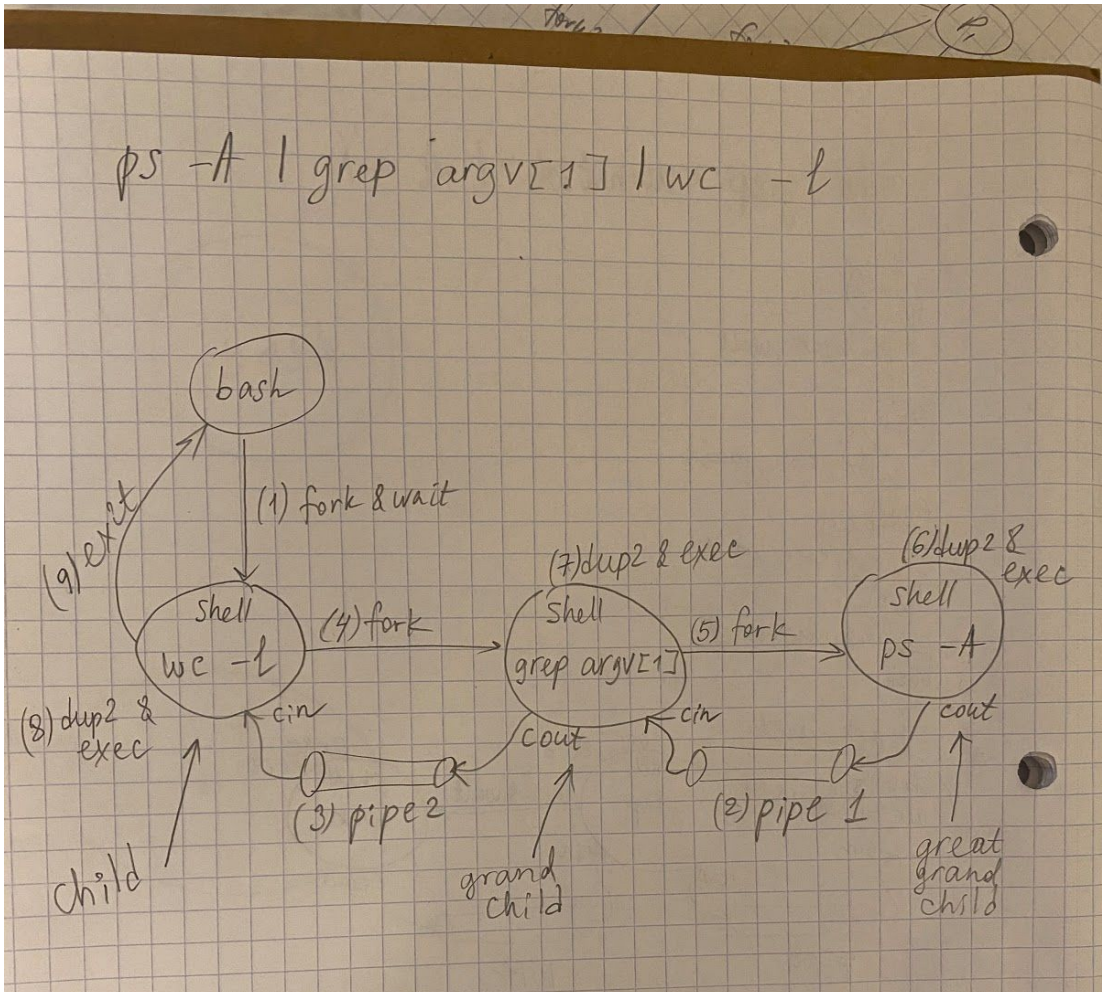
Processes testing on my computer:

```
niko@niko-UX331UA:~/Documents/CSS430$ ./processes kworker
38
niko@niko-UX331UA:~/Documents/CSS430$ ps -A | grep kworker | wc -l
38
niko@niko-UX331UA:~/Documents/CSS430$ ./processes sshd
0
niko@niko-UX331UA:~/Documents/CSS430$ ps -A | grep sshd | wc -l
0
niko@niko-UX331UA:~/Documents/CSS430$ ./processes scsi
6
niko@niko-UX331UA:~/Documents/CSS430$ ps -A | grep scsi | wc -l
6
niko@niko-UX331UA:~/Documents/CSS430$
```

Location of shell commands on my computer:

```
niko@niko-UX331UA:~/Documents/CSS430$ which ps
/bin/ps
niko@niko-UX331UA:~/Documents/CSS430$ which grep
/bin/grep
niko@niko-UX331UA:~/Documents/CSS430$ which wc
/usr/bin/wc
```

I tested processes with cout statements and process ids.To explain my implementation for processes.cpp, I decided to draw a flowchart. I also put some comments in the processes.cpp file so it should make sense together.
My flowchart:



Shell.java:
To test Shell.java, first compile it with javac Shell.java command in the ThreadOS folder. Then run java Boot to boot ThreadOS. Once ThreadOs is loaded, run I Shell command to load Shell. Input commands, for example *PingPong abc 100 ; PingPong xyz 50 ; PingPong 123 100* to test it (if PingPong.class is in the same folder). I compared my output with the one I did before compiling my own Shell.java(using Shell.class that was already in ThreadOS)

Implementation:
When Shell.java runs it constantly checks for new commands. Every time it takes in a new command it parses it based on ";" sign (each of such commands is sequential). Then it sends them for execution in order, one by one. Each sequential command can still contain concurrent commands so I first parse it by “&” sign (each of such commands is concurrent).Then I execute every concurrent command and if there is more than one I join them. To keep track of thread ids to join I use hashmap. To quit enter exit.