SQ1:

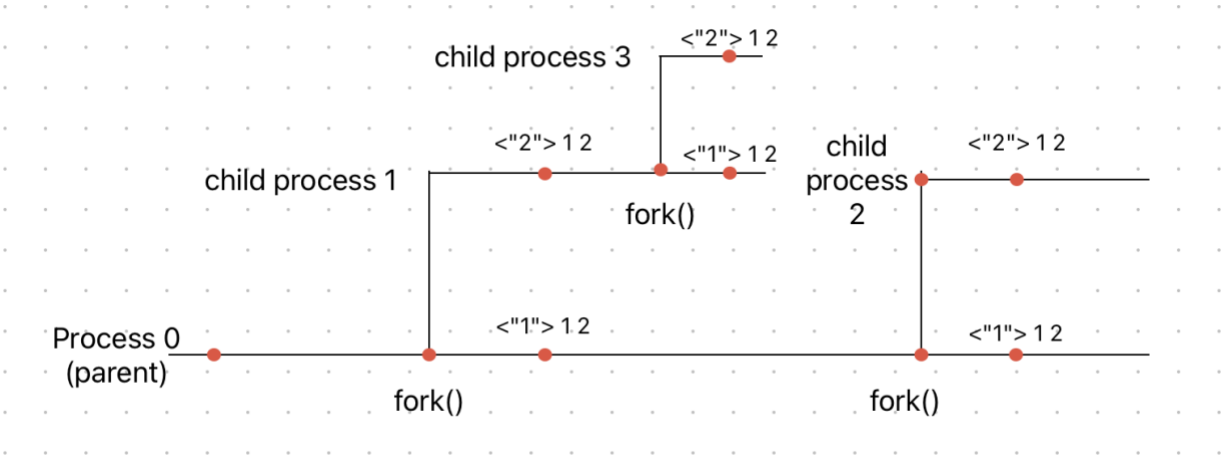
With every call to the while loop, a child is created with fork(). The fork() is thus called twice since j is incremented by 1 two times and terminates. After the fork, both the parent and child processes continue executing. In this code, the if statements differentiate the calls to parent vs. child based on their PID. All processes enter the for loop.

On the first iterations:

- Original process forks, creates child process 1
- Prints <”1”> 1 2
- Child process 1 prints <”2”> 1 2

Second iteration:

- Original process forks, creates child process 2
- Child process 1 forks, creates child process 3
- Process 0 prints <”1”> 1 2
- Process 1 prints <”1”> 1 2
- Process 2 prints <”2”> 1 2
- Process 3 prints <”2”> 1 2



SQ2:

Two processes are concurrent if their execution time overlaps.

For each pair:
A and B are not concurrent (no overlap)
A and C overlap in time so they are concurrent
A and D overlap so they are concurrent

B and C overlap in time so they are concurrent
B and D overlap in time so they are concurrent

C and D overlap in time so they are concurrent.

The concurrent pairs are:
A-C, A-D, B-C, B-D, and C-D.

A-B is the only pair that is not concurrent.

SQ3:

The ‘init’ process can terminate processes owned by other users, as it runs with root privileges, but cannot directly terminate kernel-level processes or kernel threads since they operate at a lower level than the ‘init’ process.

A special case:
‘init’ is able to terminate a zombie process if it becomes the new parent of the zombie process. This can occur if the zombie’s parent is unable to terminate it for some reason.