Chapter 3 Exercises

1. Compiled and ran twoprocs. The output was:

```
   I am parent 1300
   I am child 1301
```

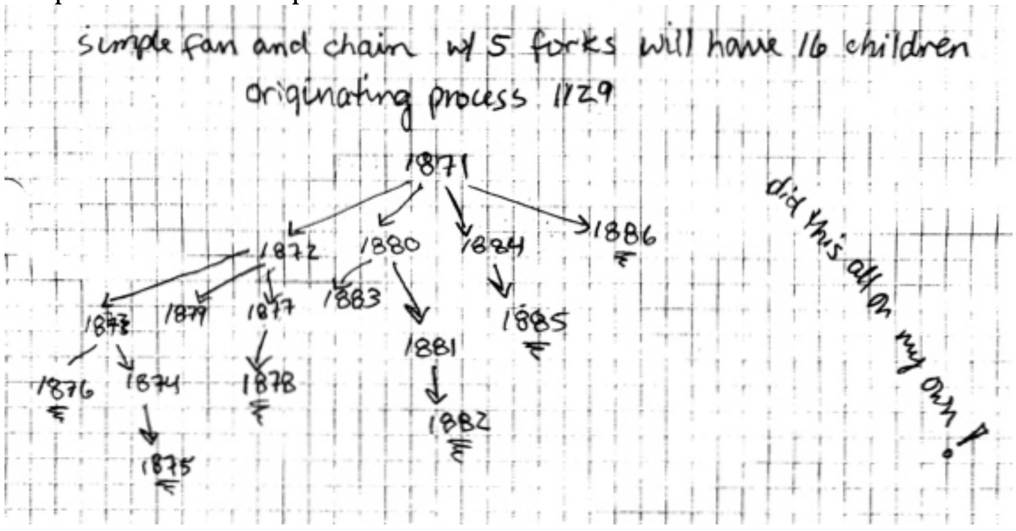
Compiled and ran simplechain. There was a warning that said I shouldn't use an assignment as a condition without parentheses. I assumed the code was wrong, but I was. C can do the assignment and condition all in the if statement. I suppressed the warning by adding parens around the statement -- if ((childpid = fork())).

Note: stdout buffers the message, so it will not print out immediately after a printf returns. stderr does not buffer messages.

Modifying fork() == -1 means the forked process does not know if the process is a parent or child. As a result, parent and child spawn a process, so at each iteration the number of processes will double.

(Put a wait(NULL) in the loop. This kept the children from being orphaned and adopted by init, which would give the child a parent of 1. Very hard to make an ancestry tree with so many parents of 1!)

```
./simplefan 5
i:5  process ID:1875  parent ID:1874  child ID:0
i:5  process ID:1874  parent ID:1873  child ID:1875
i:5  process ID:1876  parent ID:1873  child ID:0
i:5  process ID:1873  parent ID:1872  child ID:1876
i:5  process ID:1878  parent ID:1877  child ID:0
i:5  process ID:1877  parent ID:1872  child ID:1878
i:5  process ID:1879  parent ID:1872  child ID:0
i:5  process ID:1872  parent ID:1871  child ID:1879
i:5  process ID:1882  parent ID:1881  child ID:0
i:5  process ID:1881  parent ID:1880  child ID:1882
i:5  process ID:1883  parent ID:1880  child ID:0
i:5  process ID:1880  parent ID:1871  child ID:1883
i:5  process ID:1885  parent ID:1884  child ID:0
i:5  process ID:1884  parent ID:1871  child ID:1885
i:5  process ID:1886  parent ID:1871  child ID:0
i:5  process ID:1871  parent ID:1129  child ID:1886
```



2. wait() pauses the execution of its calling process until it gets the status location about the child process that terminated. waitpid() may or may not pause it calling process.

There are four ways to use wait()/ waitpid().
If pid == -1, the call waits for any child process.
If pid == 0, the call waits for any child process in the caller's process group.
If pid > 0, the call waits for the process with process id == pid
If pic < -1, the call waits for the process that has a process group == abs(pid)

3. In simplefan.c, using wait(NULL) kept the children from being orphaned, but did not keep the processes in order of 'i'. When using while(wait(NULL) >0), then the output was in numerical order.

r_wait() means to wait for ALL of the children

4. execl, execlp, execle, execv, execvp, execve, function family replace the current process image with a new process image.
execlp and execvp search for executables using the PATH variable.
execv and execvp point to an array the represents the argument list of the program.
execl, execlp, execle gives the list of arguments for the program that is to be executed.
execve executes a file. An interpreter takes all the arguments need to execute the program, using the first one as the program name.

5. execcmd.c takes any terminal command as an argument, and any arguments related to the command.

6. Comparing execcmd.c and ececcmdargv.c

Both programs have the same result. The difference is the execmdargv makes a new array of arguments from the command line arguments, which the child process executes. The trick that execcmd.c uses is to use the argv array directly rather than duplicate it.