Sample Coding Questions for Exam #3, CS 306 Spring 2015

(1) Write a C program (main) that will create a file, start a function that can write to the file running in a child process, then wait for the function (really the subprocess) to complete (terminate):

-- The file to be created should be named output.
-- The prototype for the function is: void process(int filefd)
-- filefd must be a file descriptor that process() will use to write to the file.
-- Error check the fork() call, and terminate if it fails.
-- process() will eventually exit() and indicate success/failure via its exit status.
-- Your program must not terminate until the child has terminated; don’t leave orphans/zombies.
-- If the child indicates failure exit status, your program should too.

(2) Variation on (1): instead of having process() write directly to the file, have it send data via a pipe to the parent, and have the parent write the data to the file. Details that differ from (1):

-- The prototype for the function is: void process(int pfd)
-- pfd must be a file descriptor that process() will use to write to the pipe.
-- The parent must not make any assumptions about the amount or type of the data that will be sent by the child; it must simply duplicate it in the file.

(3) Write a C code fragment that will wait for and collect an arbitrary number of child processes, and print out the number of successful children when all children have been collected:

-- Since the number of child processes is arbitrary/unknown, you will have to use a loop to collect, checking whether any more children remain.
-- You must obviously check the exit status of each child and keep counts.
-- Print the number of successful out of total, like: 4/6 children were successful.
-- A "code fragment" means code that could be inserted into a main or function, so does not include the syntax necessary to define a main/function.
-- You must, however, include any variable, etc. declarations required by your code.
-- Do not concern yourself with what the subsequent parent code will do.

(4) Variation on (3): instead of printing the number of successful children finally, terminate the parent (with failure status) if any child is not successful.

(5) Write a program (main) that takes a number of children as a command-line argument, then creates that number of children, each of whom simply prints out which number child it is along with its PID and PPID twice, sleeping 1 sec. in between:

-- Remember that command-line arguments are strings, so you must use a function such as atoi() to convert the number argument to an integer.
-- A function can determine its PID with the function getpid() and its PPID with the function getppid().
-- Follow good practice and collect all children before terminating.

(6) Write a function that will take a command name as a parameter, and execute it in a subprocess:

-- Function prototype: int exec_command(char *command)
-- command is to be a simple command name, like ls or wc.
-- The return is to be an int Boolean indicating true if the command was able to be exec’ed, else false on any errors.
-- Make sure that the function does not return until the command has completed, and do not leave orphans/zombies.