Ivy Truong
04/11/2022
CSCI 3800
Homework 4

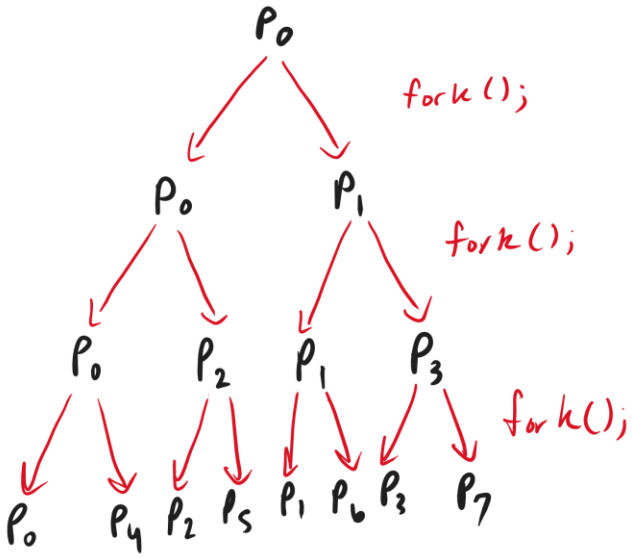
1. Code file timeLeft.c, and the .jpg files included to answer this question.

2. a) No, the code shown in problem 7.10 is not correct. This is because automatic variables can never be referenced after the function that declared it returns (according to pg. 219 in Chapter 7 of the textbook). When the function returns, the space on the stack that had that function before is freed. This allows the stack frame to use that space to store the next function call, but there will be conflicts when the IO tries to access that variable at that memory location.

   b) My solution would be to make the int *ptr variable a global variable declaring it in this format: static int *ptr. Another solution could be to move the int *ptr variable and declare it outside of the function (global). This is because local variables like automatic variables are stored on the stack memory, while global and static variables are stored in the data memory.

3. a) Important points about the fork function:
   i. Fork function allows an existing process to create a new process, a child process, that is a copy of the parent process.
   ii. The return value of the child process is 0, while the return value of the parent process is the process ID of the new child process.
   iii. A parent can have multiple children processes, but each child process can only have one parent process.
   iv. Variables in a child process do not affect the value of the variables in the parent process because the child has a copy of the parent data; child and parent processes do not share memory for these variables.
   v. Files opened in the parent are duplicated into the child process, meaning that they share a file table entry for every open file descriptor.
   vi. Cases for handling file descriptors after a fork:
      a. Parent waits for child to complete. If the child process read or wrote anything to the file the file offsets are updated.
      b. Parent and child processes run at the same time and close the file descriptors they don't need so neither process interferes with each other.
   viii. Uses for fork:
      a. A process wants to duplicate itself so processes can run different parts of the code at the same time.
      b. A process wants to execute a different program.

b) 8 new processes



4. a) The system function is a function that allows the user to execute any valid shell command that can be executed on the terminal of an operating system, returning -1 if there is an error and the termination status of the command if there is a success. The system function is implemented using fork, exec, and waitpid. First, the system function calls the fork function to create a new process. Then, the child process in the system function calls execl which specifies the path of the executable binary file of the shell, the shell, -c that specifies to take the next command line argument, the command to execute, and a null terminated string. Lastly, the parent process in the system function calls waitpid on the child process until the command is executed or the execl function fails to execute in the child process.

   b) One advantage of using system function is that it handles the required error and signal handling. Another advantage of using the system function is that it allows the programmer to execute any command on the shell, while using a custom shell using fork and exec requires a binary executable file to execute the chosen command. One disadvantage of using system function is that it cannot be run under special user permissions using a set-user-ID or set-group-ID program, which leads to a security hole in the shell. Another disadvantage of using the system function is that it is supports different platforms and not portable, so certain commands that can be executed on one platform may not execute on another platform. The last disadvantage is that the system function consumes a lot of resources because it may execute 2 processes to run one command. The system calls should be used when you only need execute a few commands on the shell from your c program since the system function uses a lot of resources to execute, when you know your c program does not need to support multiple platforms, and when your c program does not require using special permissions from the specified shell. Overall the system function should be used for simple c programs.

   Resources for Q4:
   https://stackoverflow.com/questions/1697440/difference-between-system-and-exec-in-linux
   https://stackoverflow.com/questions/19913446/why-should-the-system-function-be-avoided-in-c-and-c

https://cplusplus.com/articles/j3wTURfi/