CS 536 Fall 2016: Lab1

Maria L. Pacheco

September 14, 2016

1 Problem 1

1.1 Unchecked `execlp()` and invalid commands

When the return value of `execlp()` is not checked, we are not able to terminate child processes that failed to execute the input command. This will cause the child to go to the beginning of the `while` loop and his `pid` will be printed in the command prompt. This means, we know have other processes receiving input commands and creating children besides the initial parent process.

1.2 Not waiting for a child process before continuing to loop

If we don't wait for a child process before continuing to loop and leave the program as it is now, there is no way we can guarantee that child processes terminate. This in one sense allows for actual concurrency, but on the other hand it fails to guarantee that zombie processes won't be left behind, as it doesn't check their exit status.

1.3 Performing `waitpid()` as a blocking call from within the parent process is not a valid approach for a file server

Having the parent process wait for each child process before continuing to receive new inputs and creating new children, causes this program to execute commands in a sequential way rather than in a concurrent way. In a file server, all client requests done between the start of a child execution and its termination would be lost and not processed. In case they were en-queued, they would be delayed and the server would incur in overhead. Using signals is a way to deal with this issue and prevent child processes from becoming zombies and check their exit status. This way, we define a signal handler in the parent process. Child processes would send a signal: be it by having finished, being interrupted, etc.

1.4 Yielding more reliable server code

Three things that could be improved in the given program would be, 1) replace the `waitpid()` approach with a signal manager to allow the program to function as a server, being available to receive all client requests and preventing child processes from becoming zombies and check their exit status and 2) keep a clean

buffer to avoid corruptions when reading command requests 3) have a server that only deals with enqueueing requests and not wasting time in processing the request before listening again, to guarantee that no requests are lost, we could have another thread that reads the shared queue and processes the requests.