Lab 1: C System Programming Review

CS536
Duc Viet Le

September 14, 2016

Problem 1.

1. What happens if the return value of `execlp()` is not checked and you give an invalid command, say, `lssss`, as input to the shell?
   Ans. if `execlp()` is not checked, if an invalid command is passed to the shell, the child process will not be terminated, and the code will create a new child process to handle next user input.

2. What happens if the parent process does not perform `waitpid()` and immediately returns to the beginning of the `while-loop`?
   Ans. Even when a child exits, the Linux system will still keeps some information of child processes. The parent is required to clean up child processes (i.e using `waitpid()`).
   If the parent process does not perform `waitpid()`, its child processes are still there and do nothing (i.e becoming zoombies). They waste memory and may cause the kernel to run out of processes.

3. If the concurrent server were a file server that receives client requests from processes on the same host/OS (or over a network), why is performing `waitpid()` as a blocking call from within the parent process not a valid approach? Describe an asynchronous method for performing `waitpid()` so child processes are prevented from becoming zombies and their exit status can be checked.
   Ans. If the concurrent server is a file server with clients, using `waitpid(k, &status, 0)` will make the parent wait its child process; therefore, it prevents parent from receiving incoming requests from other clients while child process is still running.
   Sol: Using `WNOHANG` instead of 0, we allow parent to return if no child has exited instead of waiting for its one child to exit. Therefore, it can handle request from multiple clients concurrently.

4. Ignoring the functional simplifications of the shell code, point out at least two programming bugs that should be fixed to yield more reliable server code.
   Ans.

   - Did not check if `k < 0`, when the code fails to create child process (i.e fail to `fork()`). Therefore, it will `waitpid()` will yield error because there is no child process.

- Did not check status return by `waitpid()`. We don't know if `waitpid` fails or not.

Problem 2.
Interleaving Problem: Interleaving will not be a problem when we use sequential server because sequential server handles request from client one at a time.
However, for concurrent server, interleaving may be an problem because when server receives request from a client, it creates a child process to handle request from different clients. For example:

- Client $\mathcal{A}$ and client $\mathcal{B}$ both want to update same file or perform same request at same time, and almost at the same time. $\mathcal{A}$ requests to server first, and server create child process to handle $\mathcal{A}$'request. $\mathcal{B}$ requests to server right after $\mathcal{A}$; similarly, server create a new process to handle $\mathcal{B}$'s request. However, due to unpredictable scheduling, process that handles $\mathcal{A}$ terminates after $\mathcal{B}$'s process. Therefore, changes made by $\mathcal{B}$ will get overwritten.

- It will not be problem otherwise (i.e $\mathcal{B}$ sees and updates file after $\mathcal{A}$ changes it).