Introduction

Process management involves the creation and monitoring of processes in an operating system. Each process on an operating system effectively runs in parallel to other process, allowing for multiple things to happen simultaneously.

This is integral for all modern operating systems, as users expect to be able to run many programs simultaneously.

Implementation summary

Overall structure

My program has a very basic structure. It creates 16 child processes that all perform different commands (aside from the two that abort). The commands for these process are stored in array for convenience.

After all child processes complete, the parent program outputs a summary of the children's exit statuses.

fork()

The parent process calls *fork()* 16 times to create 16 child processes. If *fork* returns 0, the loop will break. This is to prevent children from creating their own children. The number of the iteration *c* is used by a child to indicate which child it is.

execvp()

execvp() is used by each child process to execute a Linux command. This command replaces the current process (a child process) with a new process running the command.

If *execvp* fails, the child will return the status returned by *execvp*.

waitpid()

The parent process uses *waitpid* to wait until the specified child process completes. It does this in the order of process creation, so it won't wait for child 2 until child 1 has completed.

Results and Observations

Child processes were created using fork, and their PIDs were stored in an array. The processes were created in the order defined in the array, but are very unlikely to terminate in that same order, as the commands take varying amounts of time to complete.

The parent uses *WIFEXITED* and *WIFSIGNALED* to determine if a child exited normally or was terminated by a signal.

Conclusion

The *fork* function operates in a way that feels really weird. It's rather strange the way a child process picks up at the exact moment fork is called, and continues from there. I would have expected *fork* to be passed a function or something, which child process would start in. But I also understand that the way *fork* works has many benefits.

waitpid is very intuitive, but I really don't know why the macros are called stuff like *WIFEXITED*. Kind of a funny name.