Due Tuesday, 05 Oct 1999, at the beginning of class.

Please revisit this document regularly. It may change as we learn more about implementation details.

1 Overview

The goal is to implement a very simple shell running under the Linux operating system.

Our simple shell's life consists of

- Reading a shell command typed by the user.
- Starting a child process that executes the given command.
- If necessary, waiting until the child finishes. If not, prompting for another command.

Our shell will provide two features of more commonly used shells: it will execute commands with full pathnames and will run commands in the background.

2 Specifications

The shell should support two features:

- Given a command specified by a fully-qualified pathname and a set of arguments, it should invoke the command. For example, the command "/bin/echo hello world" should run the /bin/echo program with the given arguments.
- Given a command (again fully-qualified and possibly having arguments) followed by an ampersand ('&'), run the given command in the background. That is, start running the command, but permit the user to type additional commands. If the user types the command "fg," bring the command into the foreground, preventing typing of additional commands until the command finishes.

  You need not worry about file redirection (<, >), quoting, control-z, or bg.

3 Linux System Calls

Among the system calls you may wish to know about include

`fork` "splits" a process into two pieces: a parent and a child. `fork()` returns one value to the parent process and a different value to the child process so they can differentiate each other.

`execve` replaces the currently running program by the specified program, also giving it the specified argv values. The child process may want to use this command.

`exit` stops program execution, returning a value to the parent. Your shell will probably not directly invoke this function. The programs that the shell executes will invoke this function and could be caught by a `wait`.

`wait` suspends a process until a child process finishes.

Be sure to read the UNIX manual pages for fork(2), execve(2), exit(3), and wait(2) to learn their prototypes and necessary header files.

4 Optional Features

Optionally, you can extend your shell's features. For example, you can add backgrounding a process, searching for commands in a specified list, or file redirection. Go for it!

5 Additional Resources and Hints

The bash WWW page briefly describes shells before going into bash specifics.

This is the first time this project has been assigned so difficulties will surely be encountered. As appropriate, I will post hints and explanations here as we learn them.

6 Grading and Logistics

You can work in groups of up to two people. Grades will be assigned primarily according to correctness. Adequate documentation of the project and the code will assist in assigning partial credit. In class, we will discuss how to submit the project.