Project 1 - Quash

Brent Mitchell & Tyler Yoder

# Introduction

The purpose of this project was to build a shell using UNIX system calls. A shell program was designed to be similar to a basic version of bash. Features such as running executables, running built-in functions like cd and jobs, I/O redirection, and piping were implemented. This project was helpful in learning how shells and operating systems interact through system calls.

# Implementation

Our shell was implemented in c, on UNIX systems. Part of the program was developed on OSX while the other part was developed on Linux. We implemented the project in two files, main.c and jobs.c. jobs.c includes three functions that are used for job reporting, while main.c contains all of the other functions of the shell.

## General Implementation

In general, the implementation works as follows:

1. User inputs a command, up to 256 characters long.
2. The input is scrubbed, and spaces are added on either side of any symbols for easier parsing. Then the input is tokenized based on any of the possible delimiters.
3. The number of pipes needed is calculated by counting the number of | symbols in the input.
4. For each token in the input, background\_process, redirect\_output, redirect\_input, should\_pipe, and pipe\_number flags are updated depending on what symbols were input. command flag is set to 1 if the input function is not a built-in function.
5. If redirect\_output or redirect\_input is true, the token is stored in the first open space in the to\_filename or from\_filename array, respectively.
6. run\_utilities is called on the token to see if it is a built-in command or not. If it is, the built-in command runs and should\_fork is set to false. Otherwise should\_fork is set to true.
7. If should\_fork is true, jobs is incremented and fork() is called.
8. Pipes are set up between all piped processes, if any. If there is an input or output file, pipes are connected to them before continuing.
9. The command is executed.
10. If background\_process is false, quash waits on the thread to finish. Otherwise the command is added to jobs and sleeps.

## Functions

Several helper functions were created. They are explained briefly below.

### Operator

Returns 1 if &, <, >, or | is present in the string.

### Sig\_child\_handler

Handles the SIGCHLD signal to reap zombie processes.

### Init\_zombie

Uses sig\_child\_handler to kill any zombie threads at quash startup.

### Add\_space

Adds spaces before and after reading &, <, >, or | to make parsing easier.

### Run\_utilites

Handles the built-in functions set, exit, quit, cd, jobs, and kill. If any of those functions are called, they are executed and 0 is returned. Otherwise, 1 is returned. This value is used for deciding whether or not main needs to fork to execute the command (it doesn’t need to fork to execute built-in functions).

# Testing

Each required function was tested for completeness. The process for each is listed below.

### Run executables without arguments.

Ran a second instance of quash, ./quash.

### Run executables with arguments.

Ran several EECS 678 labs with arguments.

### Set command for HOME and PATH work properly.

Set PATH, then used a command that would only execute with correct PATH, i.e. in a different subdirectory but not usually in PATH. For HOME, cd with no arguments returns to HOME.

### Exit and quit commands work properly.

Used exit and quit command, both exited quash and returned to normal terminal.

### Cd (with and without arguments) works properly.

Set HOME to something noticeable, then ran cd with no arguments to return to HOME. Cd <dir> tested and moves to correct directory.

### PATH works properly. Give error messages when the executable is not found.

Attempted to run something in the current path, and it executed correctly. Attempted to run ./gibberish and it was not found.

### Child processes inherit the environment.

Using main with three arguments explicitly ensures that the same environment is used.

### Allow background/foreground execution (using &).

Ran executable with & showed correct output and ran executable in background. Foreground execution ran normally.

### Printing/reporting of background processes, (including the jobs command).

Jobs command prints each background process. To test, several executables called using background operator and then jobs was called. They all printed out correctly.

### Allow file redirection (using > and <).

Running quash < testCommandFile.txt ran the command in testCommandFile. Running echo Hello > a.txt puts “Hello” into a.txt.

### Allow (1) pipe (using |).

Attempting to pipe together two processes works as intended. cat testCommandFile.txt | less works correctly.

### Supports reading commands from prompt and from file.

Reading commands from prompt is common case and works correctly. Reading commands from input file is covered in ‘Allow file redirection’.

### Support multiple pipes in one command.

Tested in the same way as the single pipe, but with two pipes. Works for arbitrary lengths up to MAX\_LENGTH, since pipe creation is just done in a for loop.

### Kill command delivers signals to background processes.

Put a process in the background and attempted to kill it.