FINAL PROJECT REPORT

Project 1

Group Members

Nicolas Cardenas
John Gregulak
James Owens

Operating Systems

Fall 2015

Contents

Project Problem Statement

The purpose of this project is to gain familiarity with the mechanics of process control through the implementation of a shell user interface. This includes the relationship between child and parent processes, the steps needed to create a new process, including search of the path, and an introduction to user-input parsing and verification. Furthermore, the student will come to understand how input/output redirection, pipes, and background processes are implemented.

The task for the project was to design and implement a basic shell interface that supports input/output redirection, pipes, background processing, and a series of built in functions as specified from the project description. There were many things that standard Linux supports but that the programmed shell did not support; however, these features could be added in as extra features. The shell should be robust (e.g. it should not crash under any circumstance beyond machine failure). There should also be a make file used to compile the shell and delete and unnecessary/old files from the last time the shell was run. This includes any text files created and also all of the object files.

Assumptions made

The first assumption made was that special characters, quotes and globs could not be entered on the command line. Another major assumption was that no multiple redirections of the same type will appear. For example, ls –as > text.txt > out.txt will not appear and also something similar to input.txt < ./out > output.txt would not appear for testing of this project.

In addition to the defined assumptions above, it was assumed that this shell would only be compiled and run using Linux on the linprog machine.

Also, it was assumed that several things will not be run in the background. This list includes all text editors like vi, nano and emacs. In addition, it was also assumed that etime and limits would not be run in the background. Because etime waits for the child to finish inside the fork, the wait statement would need to be removed for the background processing; this would cause the background time to calculate immediately (without measuring the proper execution time). To solve this problem requires a function call back when the process is removed from the queue.

For background processing, the actual execution strings are printed out when a process completes (not what was typed into the shell).

When running grep, the assumption is that the search term is not surrounded in single quotes.

Problem Solving Steps

Initially, the project was divided into smaller parts and dispersed amongst the group members for to work on individually until the next time we met up. Once we completed our individual parts, we began to bring all of the files together one by one and integrating them into the whole shell. This required several parameter changes, and a lot of glue code to be written. Testing was done on all the individual functions and sections of code to make sure they worked individually before they were integrated to the overall shell.

For example, a series of test functions exist inside of test_functions.c to test major functionality necessary in both parsing and the glue code created.

A series of utility libraries were also constructed to make the core parts of the code easier to read. A string library was created for use in parsing, tools for probing two-dimensional character arrays (argv) were created, and a buffer data structure was made for use in tracking executing background processes.

For the more complex functions that had to be built in like pipes and background processing, we started with easy tasks. For pipes, a single pipe was created before moving onto the more complicated cases. Had we just tried, for example, to get triple pipes to work from the beginning, it would've been a nightmare trying to debug and find out why it didn't work.

We also spent a lot of time working together. As a team, we found that we could communicate more effectively and write better code. Each person could still work separately when we were together, but it was easy to obtain help from another member when we were together.

A lot of error checking was done to ensure that the code was robust and would not crash from simple errors.

Why Solution uses system calls/libraries

- #include <stdio.h>

  We used this library function for all input and output that was done in the shell, whether it be to/from files or just to the screen as stdout.

- #include <stdlib.h>

  This library does many useful things, but we mainly used it for malloc, calloc, and free that were all used for this project.

- #include <string.h>

  This library helped us manipulate C strings and arrays. The main function calls used were strcmp and strcpy.

- #include <unistd.h>

  This library was used for symbolic constants and types. This library was also used for the exec family of function calls (execv and execvp in particular) and the access function for testing the existence of files.

- #include <sys/types.h>

  Used mainly for getting process IDs using pid_t.

- #include <sys/wait.h>

  Declarations for waiting. Used mainly for waiting for the child process to finish before the parent process executes code.

- #include <fcntl.h>

  File controls. From this header we used O_CREAT to create a text file to write to and O_WRONLY to write only to a file to make sure that nothing reads from it.

- Access

  We used this to determine accessibility of a file.

- #include <sys/time.h>

  Used when getting time of day when using etime.

- #include <time.h>

  Used to get the time of day and manipulate time.

- Int fd = open(…..)

  Used to open a file into the file descriptor table. Followed with close() and also dup().

- Fork()

  Creates a new process

- Execv() and execvp()

  Execute commands in a new process.

- Stat

  Used to find if a name is a file or a directory.

Problems Encountered

The main issue we encountered was in dynamic memory allocation. The use of calloc and free caused a lot of difficulties during the project. One major problem with this was calling free on a variable multiple times, which gave a cryptic stack trace to traverse. Another problem related to memory was in memory access. There were times when one-off errors would cause a read past the end of an array (caught with valgrind). Another error was in partially initialized structs (also caught by valgrind). The partially initialized struct was fixed by appending = {0} as an initializer.

Parsing gave a lot of problems early on in the project. Mainly with the structuring of code. It often became difficult to debug due to not very modular code. To rectify this issue, a string utility library was developed and most of the functionality of parsing was delegated to a series of utility functions found in utility.c.

For piping, it was quite straight forward to get a simple command like ls | wc to work but it was harder to get double and even the triple pipes to work properly. The main issues with double and triple pipes were not properly closing file descriptors after they had already been duped into either the stdin or stdout location and the improper ordering of the waitpid() function calls in the parent process. Also, nested forking was required. This required careful code analysis when structuring how the fork() commands were called.

A very interesting roadblock encountered was, again, in memory. For a while, when working on the code, it was found that a particular function was returning a pointer, then the pointer was becoming invalid. However, this was not the case on every platform. The code ran fine on the program machine. Upon further inspection it was found that on linprog, the pointer was truncated upon return (used valgrind to find this). This led speculation that the return value on the function was incorrect. In the function definition, the return type was correct. However, it was found that the definition was not included in the file where the function was being called. The c compiler had implicitly defined a return value for the function as a 32 bit signed integer. This truncated the 64 bit character pointer and caused the pointer to point at completely different data. The problem was fixed by using the proper includes.

Known Bugs (same as README)

- When typing cd ../ at root, the shell changed PWD to //
- A non-deterministic bug was found when running the shell early on: Any command called (even valid) would not run. The arguments were parsed to absolute paths equal to $PWD/[cmd_entered]. Not even exit would work in this situation. However, later on in the project, the shell was run a few thousand times with a text file input and no error occurred. It is believed at this time that this bug is fixed.

Division of Labor

Nicolas Cardenas – Pipes, CD, Exit, Path Resolution

John Gregulak – Limits, Time, File Redirection, Echo, Prompt

James Owens – Parsing, Shell structure, all glue code, background processing

Slack Days Used

For Project 1, no slack days will be used.

Cumulative Log Entries

Group Member Name	Date of Work	Changed Made (Brief Description)
James, John, Nick	9/2	Visualizing project and dividing the project amongst the members
John	9/2-9/7	Work on File Redirection
John	9/9-9/11	Work on etime

James, John, Nick	9/13	Team Meeting
John	9/14-9/16	Work on limits
John, James, Nick	9/18-9/21	Team meeting, helping each other with parts.
John	9/27	Working prompt
John, James Nick	9/25 - 9/27	Group meeting, started assembling project into a whole and integrating smaller functions into shell
Nick	9/7 - 9/12	Working on CD
John	09/14	Echo works
Nick	9/16	Exit works
Nick	9/18-9/28	Working on Pipes
James	9/9	Initial Commit of parse.h and parse.c files
James	9/10	Initial commit of utility.h and utility.c
James	9/16	Work on string library
James	9/18	Work on splitting of arguments and environment variable
James	9/20	Path Resolution
James	9/24	Background process execution
James, John, Nick	9/28	Finallization of project

Question Responses

For Project 1, there are no questions to be answered.

Additional Features

None were added.