Linux Shell Project Report

# Overview

The goal of this project was to create a working Linux command shell, supportive of full I/O redirection, background processing of commands, up to two pipes, and a novel *‘ioacct’* command to see total bytes read/written by a command.

Our implementation utilizes a dynamically resizable array class to handle the arguments, allowing us to have unlimited length of command strings, and makes the program extremely scalable and modular. We’ve sectioned off all major and most minor processing to separate functions, resulting in extremely modular, scalable, and transportable code.

A comprehensive pseudocode diagram is provided at the end of this report.

# Design

Designing the program took many iterations and a few overhauls, but resulted in a remarkably clean and compact codebase.

## Dynamic Array Class

Writing our own dynamic array class, complete with create/delete/resize/print functions made reading and parsing arguments and commands effortless and quick, and made cleaning up memory trivial.

## Memory Management

Managing our memory inside the main function was made wasy by using as little arguments as possible, and consolidating. Memory cleanup via free() was handled at the end of each execution of our central while loop, while we made sure to set each dynamically allocated variable to NULL upon reset. This coupled with using gdb and valgrind allowed us to keep a close watch on any possible memory leaks, and eliminate them quickly.

## Assumptions

Our program contains user-input validation to check for malformed commands (ie. a redirect with no file, or a pipe with no command) and also ambiguous commands (ie. a pipe and input redirect on the same line).

The only assumptions we rely on are that the user not try to execute commands with more than two pipes, or more than two redirects. Other than that our input validation should be able to catch and prevent and other input errors.

# Development Process

# Pseudocode

#includes

function declarations

while true :

reap children processes (catch possible zombie processes)

get { hostname, username, current directory, $OLDPWD(NULL to start) }

print username@hostname prompt

create 3 arg lists (allocate mem properly, initialize to NULL)

read sommand line input

parse out whitespace,

do input error handling (malformed/ambiguous commands)

fill first argslist with full command line

check for IOacct/background process commands, adjust first argslist

if cmd = built-in (cd, ioacct, exit) :

if cd :

chdir(), handle special cases (-, ~, blank, &)

if exit[n] :

reap child processes, return n;

else :

parse out possible pipes in first argslist

fill second and third argslists with piped commands

if no pipes:

parse first arglist for redirects, adjust array, store redirect stream

search PATH for cmd

child = fork()

if child == 0 (inside child) :

check redirects:

if in redirect, open file RDONLY, dup STDIN, close

if out redirect, creat() file 0644 perms, dup STDOUT, close

if append redirect, open file WRONLY|APPEND, dup STDOUT, close

execv(cmd path, args)

else if background:

waitpid(-1, (int \*)NULL, WNOHANG);

else if ioacct:

read from proc/[pid]/io file while waitpid(WNOHANG) == 0

print bytes read/written

else (inside parent):

waitpid(-1, (int \*)NULL, 0)

if 1 pipe:

create pipe int array pipefd[2], call pipe()

parse second argslist for redirects

child = fork()

if child == 0 (inside second argument):

set input fd to pipefd[0], close fd's

check redirects:

if in redirect, open file RDONLY, dup STDIN, close

if out redirect, creat() file 0644 perms, dup STDOUT, close

if append redirect, open file WRONLY|APPEND, dup STDOUT, close

execv(2nd cmd path, args)

else (in parent):

child2 = fork()

if child == 0:

set output fd to pipefd[1], close fd's

execv(1st cmd path, args)

close pipes

call waitpid(-1, (int\*)NULL, 0) on both processes

if 2 pipes:

create pipe in array pipefd[4], call pipe twice

parse third arglist for redirects

child = fork()

if child == 0 (1st cmd):

set output to pipefd[1], close fd's

execv(1st cmd path, args)

else:

child2 = fork()

if child2 == 0 (2nd cmd):

set input fd to pipefd[0], output to pipefd[3], close fd's

execv(cmd path 2, args)

else:

child3 = fork()

if child3 == 0 (3rd cmd):

set input to pipefd[2], close fd's

do redirect fd dup2's

execv(cmd path 3, args)

close pipes

call waitpid(-1, (int\*)NULL, 0) on all 3 processes