CSC456 HW2: Dash

Author: Nathan Ducasse
Class: CSC456 Operating Systems
Due: March 26, 2018

Compilation: > make
Usage: > ./dash

Background:

The second programming assignment is extending your shell, dash. This program will introduce the fork and exec commands as well as pipes and redirects. You will keep the executable name: dash. It should present a prompt with a prompt string: dash> and accept commands to be executed at the prompt. The approach is to fork off a child process and run the command in the child process via an exec. To send a signal, you will use the kill system call. To get information on this type: man -s2 kill. The -s2 option says look at the second section (programmer section) of the man pages. It gives the includes as well. To see a list of the signals, try man -s.

Description:

This program creates a repl after ./dash is run. This repl takes in any basic linux shell command, a cd command, signal command, piped or redirected commands, or one of three predefined commands, and the “quit” (or “exit”) command. The program will also alert if it receives a signal of any kind (except seg fault, which is used for error prevention purposes). The program does **not** support chaining of pipes or redirects, it assumes there is **only one** of any pipe or redirect with one valid command on either side. Each of the predefined commands uses the /proc directory to find and display information about the system and processes on the system.

Predefined Commands:

> cmdnm <pid>:

Displays the name of the process with the given PID. If there is no process with the desired PID, or an invalid argument is given, the console prints out *“Process not found!”* It finds the process name by checking /proc/pid/status.

> pid <string>:
Prints out a list of the PIDs for which the corresponding process name contains the given string as a substring. The underlying function checks every /proc/pid directory by iterating from -1 to the PID_MAX which is found in the /proc/sys/kernel/pid_max file.

> systat:
Prints out information about the system. It prints out Linux version and system uptime, memory information, and CPU information including vendor id and everythng up to cache size in the corresponding /proc/ file.

> quit:
Exits the repl. Any other input is considered bad input and is ignored, printing a new prompt in the repl.

Functions:

int main():
Main function. Contains the loop for the repl and calls the get_input, check_pipe_redirect, pipe_cmd, redir_cmd_out, redir_cmd_in, and parse_cmd functions, and sets up the signal listeners.

void signalHandler(int):
Handler function for the signal listeners. Prints the number of the interrupt, not used for SIGSEGV.

void signalHandlerSIGSEGV(int):
Handler function for the signal listener specifically used for the SIGSEGV signal. This is used to exit cleanly under special (read: buggy) circumstances.

int get_input(char **):
Gets the input for every repl iteration and tokenizes it, split on spaces, and gets the number of arguments, then null terminates the argument list. Returns the number of arguments.

int check_pipe_redirect(int, char **, char **, char **):
This function checks the args array for the first instance of ‘|’, ‘<’, or ‘>’, and returns 0 if it finds ‘|’, 1 if ‘>’, 2 if ‘<’, or 3 otherwise. It also separates the args on either side of the operator into cmd1 for the left, and cmd2 for the right.

void parse_cmd(int, char **):

This function expects a non-redirected/piped input, and checks first if it is one of the predefined functions, then if it is a cd command, then a signal, and finally it assumes it is a generic console command and executes accordingly.

bool process_menu(int, char **):

This function checks if it needs to run the given command as one of the predefined commands. If it does not execute any, it returns false. If it attempts to to execute, it will return true regardless of whether it was successfully executed.

void cmd_cd(int, char **):

Executes the cd command. Will work with relative and absolute directory changes, and also changes to the home directory defined in the “HOME” environment variable if there was no path given.

void pipe_cmd(char **, char **):

Executes a pipe operation. This code was implemented with the sample code under the GNU Public License. NOTE: Does not chain, either with pipes or redirects, and assumes there is only one pipe with atomic left- and right-hand sides.

void redir_cmd_out(char **, char **):

Executes a redirect out operation. This code was implemented with the sample code under the GNU Public License. NOTE: Does not chain, either with pipes or redirects, and assumes there is only one pipe with atomic left- and right-hand sides.

void redir_cmd_in(char **, char **):

Executes a redirect in operation. This code was implemented with the sample code under the GNU Public License. NOTE: Does not chain, either with pipes or redirects, and assumes there is only one pipe with atomic left- and right-hand sides.

string cmd_cmdnm(int, char **):

Returns the process name of the process with the given pid via a string.

string cmd_cmdnm(string n):

Same as the other function with the same name, but used as a utility/helper function for cmd_pid.

bool cmd_pid(int, char **):

Prints out the pids of the processes whose names contain the given substring. Returns a bool for if there was a process that contained the string or not.

void cmd_systat():

Prints out the Linux version, system uptime, memory info, and cpu info.

Testing:

The program was tested with the required tests, which were typing:

- <space>
- <tab>
- random characters
- no characters [blank line]
- <space> <space> <tab> .... <spacetab> ls

The pipe and redirection, cd, and signals were tested for functionality but not with specific tests.

Submission:

The submission contents are as follows:

- prog2.pdf
- dash.h
- dash.cpp
  - Contains main() function.
- functions.cpp
  - Contains all of the logic and extra functions for the program.
- Makefile