Program 2: Diagnostic Shell

CSC 456 - Operating Systems

South Dakota School of Mines and Technology
Dr. Jeff S. McGough

Christine N. Sorensen
3/14/16

Description

Diagnostic shell. A process identification program and a simple shell.
Purpose: To introduce Unix environment, system calls, signal, the proc system, Harvard commas, and for/exec system calls.
This second edition is to add sockets, pipes, and redirection.

Bugs

- PThreads were added to this program. Threads are also somewhat undeterministic. The only time seen in this program that it's become a problem is with the new 'hb' command. It will sometimes output the first line into the following `dsh>` and then leave a blank at the end. If the user just hits Enter, it should be fine.

- When you run hb, it sometimes will spit out everything at once, not doing the increments. If you just do it again, it will work.

- cmdnm was having complications working. I had to print out the PID in order for cmdnm to print the process name. It doesn't look *bad*, but it's just incorrect.

- The pipeline can only do the standard <cmd1> | <cmd2>. It can't perform multiple pipes in a row. That being said, it also can include redirects in the pipeline.

- The redirects works with the commands given in bash and defined in this program. I created a program that takes in integer and prints it out. Doing it in normal bash with the redirect in, it works fine. It however does nothing in this program.

- The remote piping is having its ups and down. It will work with server sending a ls and client doing a wc on it. But some other commands don't work with it.

Compile

```
>gcc dsh.c fork_dsh.c funcs.c globals.c misc.c –o dsh
```

A Makefile is also included with the program by running:
>make

Usage

```
>./dsh
```

Libraries

unistd.h – used for standard and miscellaneous symbolic constants, types, and functions.
Used for functions chdir(), get_current_dir_name(), fork(), and execvp()

stdlib.h – general purpose library for C that handles process control, memory allocation, and conversions.
Used for exit() and atoi()

stdio.h – standard library for file input and output.
Used for fopen(), fclose(), fgets(), printf(), and type FILE*.

string.h – library to manipulate cstrings
Used for strtok(), strlen(), strcmp(), strcat().

ctype.h – c standard library used for mapping characters.
Used for tolower()

signal.h – library for informing processes of events.
Used for kill()

sys/types.h – Copyright © 2002 by Sunmicrosystems, Inc. Contains various types.
Used for size_t.

fcntl.h – Copyright © 2007 Apple Inc. Library for file control options

sys/wait.h – Copyright © 2007 Free Software Foundation, Inc. For the delectations of waiting.
Used for wait()

pthread.h – Copyright © 1998 POSIX Threads Library
Used for all the pthread implementation.

netdb.h – library for network database operations
Used for the sockets

netinet/in.h – library for internet protocol
Used for the sockets

sys/socket.h – library for internet protocol
Used for sockets

Functions:

dsh.c

This is the center of the program. It contains main, the dsh function that handles user input, and functions that manipulate and parse the user input.

main

Start of the dsh program. Creates a flag that is used while loop. The loop continuously calls the program. Each time, >dshThe flag value will change by the user when `exit` is submitted and therefore leave the loop, thus ending the program.

dsh

The heart and soul of this program. This is called by main continuously from main(). It takes a line that the user submits, tokenizes the words from it into an array. The first word of the array is considered the command. It is compared with the commands written for this program along with the number of parameters (it only checks for the minimum needed parameters. If there is more than necessary, the program will ignore them). It calls the corresponding function and passes its parameters. If a user input does not match one of the commands, it is passed into dsh_fork().

Each function returns a value, stored in return_val. It the value is -1, it means there was an error. The commands are then sent to dsh_fork(). The return_val is returned at the end of dsh() function. This tells the while loop in main() to keep going or not.

splitRemoteServer

This takes the user line and splits the instruction from the port number. It saves the port number and returns the solo instruction.

splitRemoteClient

This takes the user line and splits the instruction from the address and port number. It saves the port number and address and returns the solo instruction.

findRedirect

This takes in the user line. It determines whether it's a direction in or out and sets the according flag. It takes the filename and saves it. Then it returns the solo instruction.

splitInstructions

This takes in the instruction and splits the instructions. It sends the second side and parses it into an array that is saved for later. The first instruction is returned solo.

func.c

These functions are specifically towards the dsh program.

cmdnm

This returns the command string that started the process from the process id it was given. The pid is passed in. The filename containing the command name is located in /proc/<pid>/cmdline. When the pid is passed in, it is concatenated with /proc/ and /cmdline in order to find that file. The file opened (or an error is returned if it fails to open). The function simply grabs the string containing the command name and prints it. 1 is returned for success.
The second edition refactored it into pthreads.

dsh_signal

This function signals a process. It is given a pid to signal and the value to signal to it. It only calls dsh_kill() with the parameters which will be explained later in the document.

sysat

This function prints out various stats about the system using the /proc/* files. It prints:

- Linux version
- System uptime
- Memory usage:
    - Memory total
    - Memory free
- CPU information:
    - Vendor ID through Cache size found in /cpuinfo

This function opens each corresponding file, retrieves the needed data, and prints it to the screen.
The second edition refactored this into PThreads.

pwd

This prints the current working directory to the screen. It utilizes the function get_current_dir_name() to retrieve it and print it.
The second edition refactored this into PThreads.

cd

This emulates the bash cd command

dsh_kill

This function uses kill() to signal a process with a number.
The signal values:

1 – Hang up
2 – Interrupt from keyboard
9 – Kill signal
15 – Termination signal
17, 19, 23 – Stop the process

hb

This is the new function. It's a "heartbeat" in which it prints the time for a number of times specified by the user and the time increments specified by the user and units in ms or s on it.

fork_dsh.c
This contains the forking of this program

dsh_fork

This is the fork/exec function handler.
The layout of this function is similar to Dr. McGough's code which was taken from "Advanced Linux Programming," by CodeSourcery LLC

This function takes the arguments from the user input as well as the number of params. The process is duplicated. It is check to see if it's the child process. If it's the parent process, the function waits. If it's the child process, the function calls execvp() with the arguments.

In the second edition, a pipe flag was added in the case of pipelining. It creates a pipe for it. A redirect flag was added and creates a one ended pipe for it. Server and client flags were added to set up a client or server for it.

misc.c

These functions were used for supporting tasks to the program.

toLowerCase

This function takes in a string and turns it to all lowercase. This is used for to avoid errors if the user mistypes a command in uppercase.

isInt

This function is passed in a character string and determines whether it is an integer or not. It first checks to see if the first character is a minus sign for negative. The it checks the other characters to see if the ASCII value is in the boundaries of zero(48) and nine (57). It returns with a -1 if it hits a non-integer character. 1 is returned if it is an integer.

This function was used in error checking from the user. If a command required integer parameters, it was checked for valid parameters.

removeNewLine

This takes in a line and removes the newline at the end of it and replaces it with a null terminator.