Donghyun Park

Prof. Hakner

ECE 357

10/29/2017

PSet 3: The Shell

Source Code (231 lines)

#include <sys/time.h>

#include <sys/resource.h>

#include <sys/wait.h>

#include <sys/types.h>

#include <string.h>

#include <unistd.h>

#include <stdlib.h>

#include <stdio.h>

#include <errno.h>

#include <fcntl.h>

char \*readInput(FILE \*file){

char \*line = NULL;

size\_t buf1 = 0;

ssize\_t readStatus;

//reading in from stdin

if((readStatus = getline(&line, &buf1, file)) < 0){

fprintf(stderr, "Error: Failed to read from stdin! Moving to next command\n");

}

return line;

}

char \*\*parseArg(char \*line){

int pos = 0;

char \*\*tokenArray = malloc(BUFSIZ);

char \*token, \*tempbuf = malloc(BUFSIZ);

//checking memory allocation failure

if(!tokenArray){

fprintf(stderr, "Error: Failed to allocate memory!\n");

}

//tokenizing the line from stdin

strcpy(tempbuf, line);

token = strtok(tempbuf, " \r\n");

while (token != NULL){

if(!(token[0] == '<' || token[0] == '>' || (token[0] == '2' && token[1] == '>'))){

tokenArray[pos] = token;

pos++;

}

token = strtok(NULL, " \r\n");

}

tokenArray[pos] = NULL;

return tokenArray;

}

char \*\*parseRedir(char \*line){

int pos = 0;

char \*\*tokenArray = malloc(BUFSIZ);

char \*token, \*tempbuf = malloc(BUFSIZ);

//checking memory allocation failure

if(!tokenArray){

fprintf(stderr, "Error: Failed to allocate memory!\n");

}

//tokenizing the line from stdin

strcpy(tempbuf, line);

token = strtok(tempbuf, " \r\n");

while (token != NULL){

if((token[0] == '<' || token[0] == '>' || (token[0] == '2' && token[1] == '>'))){

tokenArray[pos] = token;

pos++;

}

token = strtok(NULL, " \r\n");

}

tokenArray[pos] = NULL;

return tokenArray;

}

int redirect(char \*file, int ioFd, int flags, mode\_t mode){

int fileFd;

if((fileFd = open(file, flags, mode)) < 0){

fprintf(stderr, "Error: Failed to open [%s] - %s\n", file, strerror(errno));

return 1;

}

if(dup2(fileFd, ioFd) < 0){

fprintf(stderr, "Error: Failed to dup [fd = %d] to io [fd = %d] - %s\n", fileFd, ioFd, strerror(errno));

return 1;

}

if(close(fileFd) < 0){

fprintf(stderr, "Error: Failed to close [fd = %d] - %s\n", fileFd, strerror(errno));

return 1;

}

return 0;

}

void ioRedir(char \*\*redir){

int pos = 0;

char \*trueIO, \*fullIO;

while(redir[pos] != NULL){

fullIO = redir[pos];

if(strstr(fullIO, "<") != NULL){

trueIO = fullIO + 1;

if(redirect(trueIO, 0, O\_RDONLY, 0666)){

exit(1);

}

}else if(strstr(fullIO, ">") != NULL){

trueIO = fullIO + 1;

if(redirect(trueIO, 1, O\_RDWR|O\_CREAT|O\_TRUNC, 0666)){

exit(1);

}

}else if(strstr(fullIO, "2>") != NULL){

trueIO = fullIO + 2;

if(redirect(trueIO, 2, O\_RDWR|O\_CREAT|O\_TRUNC, 0666)){

exit(1);

}

}else if(strstr(fullIO, ">>") != NULL){

trueIO = fullIO + 2;

if(redirect(trueIO, 1, O\_RDWR|O\_CREAT|O\_APPEND, 0666)){

exit(1);

}

}else if(strstr(fullIO, "2>>") != NULL){

trueIO = fullIO + 3;

if(redirect(trueIO, 2, O\_RDWR|O\_CREAT|O\_APPEND, 0666)){

exit(1);

}

}

pos++;

}

}

int execute(char \*\*arg, char \*\*redir){

pid\_t pid, wpid;

int status;

struct rusage ru;

struct timeval tic, toc;

if(arg[0] == NULL || strstr(arg[0], "#") != NULL){

//no command specified

return 1;

}else if(!strcmp(arg[0], "cd")){

//cd built-in

if(arg[1] == NULL){

fprintf(stderr, "Error: Expected a directory argument!\n");

}else{

if(chdir(arg[1]) < 0){

fprintf(stderr, "Error: Failed to change directory to [%s] - %s\n", arg[1], strerror(errno));

}

}

return 1;

}else if(!strcmp(arg[0], "exit")){

//exit built-in

if(arg[1] == NULL){

exit(0);

}else{

exit(atoi(arg[1]));

}

}else if(!strcmp(arg[0], "pwd")){

char currentDir[BUFSIZ];

if((getcwd(currentDir, sizeof(currentDir))) == NULL){

fprintf(stderr, "Error: Could not retrieve current directory\n");

}else{

printf("%s\n", currentDir);

}

return 1;

}else{

gettimeofday(&tic, NULL);

//launch command program

switch(pid = fork()){

case -1:

fprintf(stderr, "Error: Failed to fork() process [%s] - %s\n", arg[0], strerror(errno));

exit(1);

break;

case 0:

//in child process

//io redirection

ioRedir(redir);

if(execvp(arg[0], arg) < 0){

fprintf(stderr, "Error: Failed to exec() process [%s] - %s\n", arg[0], strerror(errno));

}

break;

default:

//in parent process

if(wpid = wait3(&status, 0, &ru) < 0){

fprintf(stderr, "Error: Failed to wait() process on pid=[%d] - %s\n", pid, strerror(errno));

}else{

gettimeofday(&toc, NULL);

fprintf(stderr, "Command returned with return code %d\n", WEXITSTATUS(status));

fprintf(stderr, "Real Time: %ld.%06ld sec\n", (toc.tv\_sec-tic.tv\_sec), (toc.tv\_usec-tic.tv\_usec));

fprintf(stderr, "User Time: %ld.%06ld sec\n", ru.ru\_utime.tv\_sec, ru.ru\_utime.tv\_usec);

fprintf(stderr, "System Time: %ld.%06ld sec\n", ru.ru\_stime.tv\_sec, ru.ru\_stime.tv\_usec);

}

break;

}

return 1;

}

}

void shellLoop(FILE \*file){

char \*line;

char \*\*arg, \*\*redir;

int check;

do{

if(file == stdin){

printf("S H E L L B O Y E$ ");

}

line = readInput(file);

arg = parseArg(line);

redir = parseRedir(line);

check = execute(arg, redir);

free(arg);

free(redir);

}while(check);

}

int main(int argc, char \*\*argv){

FILE \*inputFile;

if(argc == 1){

inputFile = stdin;

shellLoop(inputFile);

}else{

if((inputFile = fopen(argv[1], "r")) == NULL){

fprintf(stderr,"Error: Failed to open file [%s] - %s", argv[1], strerror(errno));

return -1;

}else{

shellLoop(inputFile);

}

}

return 0;

}

Shell Interpreter Scripts:

testme.sh:

#!/home/dhpark/school/OS/OSHW3/myshell

#This is an example of a shell script that your shell must execute correctly

#notice that lines starting with a # sign are ignored as comments!

#let’s say this here file is called testme.sh. you created it with say

#vi testme.sh ; chmod +x testme.sh

#you invoked it with

#./testme.sh

cat >cat.out

#at this point, type some lines at the keyboard, then create an EOF (Ctrl-D)

#your shell invoked the system cat command with output redirected to cat.out

cat cat.out

#you better see the lines that you just typed!

exit 123

#after your shell script exits, type echo $? from the UNIX system shell

#the value should be 123. Since your shell just exited, the following

#bogus command should never be seen

test2.sh:

#!/home/dhpark/school/OS/OSHW3/myshell

#here is another example, say it is called test2.sh

#you invoked it with

#./test2.sh <input.txt

cat >cat2.out

#since you invoked the shell script (via the system shell such as bash)

#with stdin redirected, your shell runs cat which gets stdin from input.txt

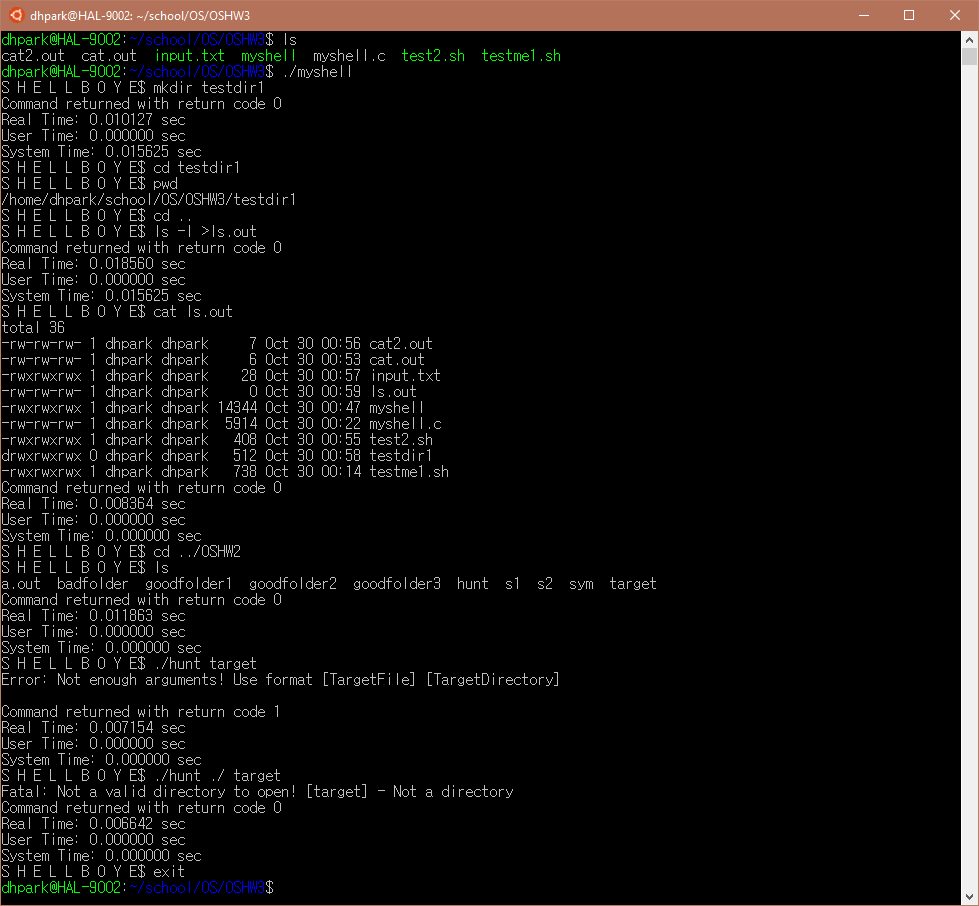
exit

#the above exit had no specified return value, so your shell exited with 0

#again, test this with echo $?

Sample Run and Error Handling

Shell:



Interpreter:

