Name: Tan Han Nguyen

NetID: TXN200004

Week 08 Lab 2

0. Log in

Pre-authentication banner message from server:

| University of Texas at Dallas

| Department of Computer Science

|

| Use of UTD Information Systems is subject to

| the UTD Information Security and Acceptable Use Policy.

|

| Pursuant to Texas Administrative Code 202:

| (1) Unauthorized use is prohibited;

| (2) Usage may be subject to security testing and monitoring;

| (3) Misuse is subject to criminal prosecution; and

| (4) No expectation of privacy except as otherwise provided by applicable

| privacy laws.

|

| ATTENTION: utdnetid != utdnetid@utdallas.edu (UTD != Google!)

|

| \*\*\*\*\* This system will require a connection to the GlobalProtect VPN startin

> g

| on the following dates:

|

| cslinux1.utdallas.edu - June 15, 2020

| cslinux2.utdallas.edu - June 22, 2020

|

| \*\*\*\*\* GlobalProtect VPN Instructions: https://www.utdallas.edu/oit/howto/vpn

> /

|

End of banner message from server

Keyboard-interactive authentication prompts from server:

| Password:

End of keyboard-interactive prompts from server

┌──────────────────────────────────────────────────────────────────────┐

│ • MobaXterm Personal Edition v24.2 • │

│ (SSH client, X server and network tools) │

│ │

│ ⮞ SSH session to txn200004@cslinux2.utdallas.edu │

│ • Direct SSH : ✓ │

│ • SSH compression : ✓ │

│ • SSH-browser : ✓ │

│ • X11-forwarding : ✓ (remote display is forwarded through SSH) │

│ │

│ ⮞ For more info, ctrl+click on help or visit our website. │

└──────────────────────────────────────────────────────────────────────┘

Last login: Thu Oct 10 20:46:27 2024 from 10.50.241.34

\*\*\*---\*\*\*---\*\*\*---\*\*\*---\*\*\*---\*\*\*

csgrads1.utdallas.edu - CentOS Linux 7.9

--All CS Graduate Students should use csgrads1--

cs1.utdallas.edu - CentOS Linux 7.9

cs2.utdallas.edu - CentOS Linux 7.9

\*\*\*---\*\*\*---\*\*\*---\*\*\*---\*\*\*---\*\*\*

This system is for use by CS students who need a general purpose Linux system

to complete homework assignments. Computationally or resource intensive

simulations will be throttled automatically.

Thank you,

CS Lab Manager

cs-labs@utdallas.edu

/scratch disk space can be used for temporary files.

All files within /scratch will be erased on a regular basis (Sunday 0300).

{cslinux2:~} mkdir week08Lab2; cd week08Lab2

1. Design and Implement myShell2.c

{cslinux2:~/week08Lab2} vim myShell2.c

{cslinux2:~/week08Lab2} cat myShell2.c

/\* myShell1.c

\* A simple shell program that uses child process and exec with redirection, process sequence and pipe.

\*/

#include <stdio.h>

#include <sys/types.h>

#include <stdlib.h>

#include <unistd.h>

#include <sys/wait.h>

#include <string.h>

#include <strings.h> //for strcasecmp

#include <glob.h> // to expand wildcard

#include <fcntl.h> // for O\_CREAT etc.

#include <sys/wait.h> // for waitpid

#include <sys/types.h> // for pid\_t

//Function to perform redirection read from stdin -> read from file, write to stdout -> write to file

void fileRedir(char\* redirection, char\* file){

int fdin;

int fdout;

//redirect standard input

if(!strcmp(redirection, "<")){

if ((fdin = open(file, O\_CREAT | O\_RDONLY)) < 0) {

perror(file); /\* open failed \*/

return; // Exit the function if open fails

}

dup2(fdin, STDIN\_FILENO);

close(fdin);

}

//redirect standard output

if(!strcmp(redirection, ">")){

if ((fdout = open(file, O\_WRONLY | O\_CREAT | O\_TRUNC, 0644)) < 0) {

perror(file); // If open fails, print an error

return; // Exit the function if open fails

}

dup2(fdout, STDOUT\_FILENO);

close(fdout);

}

return;

}

// Function to parse line input into array of arguments

void parse(char\* line, char\*\* argv) {

//Checking the line char by char until null

while(\*line != '\0'){

//Check if it's a white space -> replace it with '\0'

while (\*line == ' ' || \*line == '\t' || \*line == '\n'){

\*line++ = '\0'; //replace and move to next char

}

\*argv++ = line; // assign non-white space char to \*argv and move \*argv to next index

//Now skip the rest of the argument until the start of next argument

while(\*line != '\0' && \*line != ' '&& \*line != '\t' && \*line != '\n'){

line++;

}

}

\*argv = NULL; //Mark the end of the argument array

}

//Function to execute command using child process

void execute(char\*\* argv){

pid\_t pid;

int status;

//Call a child process, then check its pid

//pid < 0 = forking failed

if ((pid = fork()) < 0){

printf("\*\*ERROR: Forking child process failed \n");

exit(1);

}

else if (pid == 0){

if(execvp(\*argv, argv) < 0){

printf("\*\*ERROR: exec failed\n");

exit(1);

}

}

else{

//Waiting for the child completion

while(wait(&status) != pid);

}

}

//Function to expand wildcard

int expand\_wildcards(char\*\* argv, char\* expanded\_argv[]) {

int i = 0;

int j = 0;

size\_t k;

//Iterate through each argument in argv

while (argv[i] != NULL){

glob\_t globbuf;

//Apply glob to each argument

if(glob(argv[i], 0, NULL, &globbuf) == 0){

for(k = 0; k < globbuf.gl\_pathc; k++){

expanded\_argv[j++] = strdup(globbuf.gl\_pathv[k]); //Expand wildcard

}

globfree(&globbuf); //Free memory allocated by glob

}

else {

//If no match, just copy the argument as is

expanded\_argv[j++] = strdup(argv[i]);

}

i++;//move to next argument

}

expanded\_argv[j] = NULL; // null terminator at the end

return j; //Return number of expanded arguments

}

//Define buffer size

#define MAX\_LINE 4096

#define MAX\_ARG 64

int main(void){

char line[MAX\_LINE];

char line\_backup[MAX\_LINE];

char\* prompt = "{myShell2}$ ";

char\* argv[64]; // maximum 64 argument

char\* expanded\_argv[64]; //Array to hold expanded wildcard arguments

int has\_redir = 0;

//first prompt

printf("%s", prompt); // print the prompt

//The loop

while(fgets(line, sizeof(line), stdin) != NULL){

//Remove \n from fgets

line[strlen(line) - 1] = '\0';

// Backup the line before parsing

strcpy(line\_backup, line); // Create a backup of the original input

//Parse the line into array of arguments

parse(line, argv);

// Debugging output: show parsed arguments

/\*printf("Parsed arguments:\n");

for (int i = 0; argv[i] != NULL; i++) {

printf("argv[%d] = %s\n", i, argv[i]);

}\*/

//exit case

if(strcasecmp(argv[0], "exit") == 0){

break;

}

//listall case

else if (strcasecmp(argv[0], "listall") == 0){

argv[0] = "ls";

argv[1] = "-la";

argv[2] = NULL; //Mark the end of the argv array

execute(argv);

printf("%s", prompt); // prompt after exec

continue; // skip to next loop

}

//compile

else if (strcasecmp(argv[0], "compile") == 0){

argv[0] = "gcc";

//Skip argv[1]

argv[2] = "-o";

//Getting the output file name from argv[1]

size\_t name\_length = strlen(argv[1]) - 2; // not getting .c

char \*output\_name = strndup(argv[1], name\_length); // Duplicate the "program"

argv[3] = output\_name;

argv[4] = NULL;

//expand wildcards

expand\_wildcards(argv, expanded\_argv);

execute(expanded\_argv); // execute

printf("%s", prompt); // Prompt again

continue; //Skip to next loop

}

// < > case

else if ((strchr(line\_backup, '<') != NULL) || (strchr(line\_backup, '>') != NULL)) {

//Save the original stdin and stdout

int original\_stdin = dup(STDIN\_FILENO);

int original\_stdout = dup(STDOUT\_FILENO);

// Variables to store file redirection paths

char\* input\_file = NULL;

char\* output\_file = NULL;

char\* argv2[64]; // Array to hold the actual command without redirection parts

int i = 0; // Index for argv

int j = 0; // Index for argv2

// Parse the arguments and look for redirection

for (i = 0; argv[i] != NULL; i++) {

if (strcmp(argv[i], "<") == 0) {

// Input redirection: store the filename

input\_file = argv[i + 1];

i++; // Skip the filename in the next iteration

}

else if (strcmp(argv[i], ">") == 0) {

// Output redirection: store the filename

output\_file = argv[i + 1];

i++; // Skip the filename in the next iteration

}

else {

// Normal command argument: copy to argv2

argv2[j++] = argv[i];

}

}

argv2[j] = NULL; // Null-terminate the argv2 array

// Handle input redirection

if (input\_file != NULL) {

fileRedir("<", input\_file); // call redirect function

}

// Handle output redirection

if (output\_file != NULL) {

fileRedir(">", output\_file); // call redirect function

}

// Expand wildcards and execute the command

expand\_wildcards(argv2, expanded\_argv);

execute(expanded\_argv);

//Restore the original stdin and stdout

//If not restoring, it will read '/0' (EOF) and terminate the program

dup2(original\_stdin, STDIN\_FILENO);

dup2(original\_stdout, STDOUT\_FILENO);

close(original\_stdin);

close(original\_stdout);

printf("%s", prompt); // Prompt again

continue;

}

// ; case

else if (strchr(line\_backup, ';') != NULL) {

char\* command; // Pointer to each individual command

// Use strtok() to split line\_backup by ';'. tokenize the first command

command = strtok(line\_backup, ";");

while (command != NULL) {

// Trim leading and trailing spaces from the command

while (\*command == ' ') {

command++; // Remove leading spaces by move command pointer forward

}

char \*end = command + strlen(command) - 1;

while (end > command && \*end == ' ') {

\*end-- = '\0'; // Remove trailing space then move end pointer backward

}

// Parse the command into arguments

parse(command, argv);

// Expand wildcards

expand\_wildcards(argv, expanded\_argv);

// Execute the command

execute(expanded\_argv);

// Get the next command in the sequence use NULL to tokenize the same string

command = strtok(NULL, ";");

}

printf("%s", prompt); // Prompt again

continue;

}

// pipe case

else if (strchr(line\_backup, '|') != NULL){

//Variable to use 2 pipes for multiple commands

int i = 0;

int pfd[2]; // Pipe file descriptors

int prev\_pfd[2]; // For holding the previous pipe's file descriptors

pid\_t pid;

char\* command; // Pointer to each individual command

char\* command\_array[MAX\_ARG]; // Array to store the command pointers

int command\_count = 0; // counter to keep track of command\_array

// Use strtok() to split line\_backup by ';'. tokenize the first command

command = strtok(line\_backup, "|");

while (command != NULL) {

// Trim leading and trailing spaces from the command

while (\*command == ' ') {

command++; // Remove leading spaces by move command pointer forward

}

char \*end = command + strlen(command) - 1;

while (end > command && \*end == ' ') {

\*end-- = '\0'; // Remove trailing space then move end pointer backward

}

// Duplicate and store the command in the array and increment the counter

command\_array[command\_count++] = strdup(command);

// Get the next command in the sequence using NULL to continue tokenizing the same string

command = strtok(NULL, "|");

}

// Null-terminate the array of commands

command\_array[command\_count] = NULL;

// Loop through all commands

while (command\_array[i] != NULL) {

// Create a pipe for all but the last command. Since the last one -> stdout

if (command\_array[i + 1] != NULL) {

if (pipe(pfd) == -1) {

perror("pipe");

exit(1);

}

}

// Fork the child process for the current command

pid = fork();

if (pid == -1) {

perror("fork");

exit(1);

}

if (pid == 0) { // Child process

//If not the first command, redirect stdin to the read end of the previous pipe

if (i > 0) {

close(prev\_pfd[1]); //close the write end of prev pipe

dup2(prev\_pfd[0], STDIN\_FILENO); // redirect stdin to read end of prev pipe

close(prev\_pfd[0]); // close read end of prev pip

}

//If not the last command, redirect stdout to the write end of the current pipe

if (command\_array[i + 1] != NULL) {

close(pfd[0]); //close the read end of the curr pipe

dup2(pfd[1], STDOUT\_FILENO); //redirect stdout to write end of curr pipe

close(pfd[1]); //close write end of curr pipe

}

//parse, expand, execute each command string

parse(command\_array[i], argv);

expand\_wildcards(argv, expanded\_argv);

//NOTE: NOT using execute() because it interferes with the child process here.

if(execvp(\*argv, argv) < 0){

printf("\*\*ERROR: exec failed\n");

exit(1);

}

} else { //Parent process

//Close the previous pipe in the parent

if (i > 0) {

close(prev\_pfd[0]);

close(prev\_pfd[1]);

}

//Move current pipe to previous then start new curr pipe in the next iteration

if (command\_array[i + 1] != NULL) {

prev\_pfd[0] = pfd[0];

prev\_pfd[1] = pfd[1];

}

// Wait for the child process to finish

waitpid(pid, NULL, 0);

}

i++; // Move to the next command

}

// Free memory allocated by strdup

int j = 0;

for (j = 0; j < command\_count; j++) {

free(command\_array[j]); // Free each command duplicated by strdup

}

printf("%s", prompt); // Prompt again

continue;

}

//OTHER CASES//

//expand wildcards

expand\_wildcards(argv, expanded\_argv);

execute(expanded\_argv); //child execute other commands

printf("%s", prompt); //prompt again

}

return 0;

}

2. Compile and run test cases

{cslinux2:~/week08Lab2} gcc myShell2.c -o myShell2 -std=gnu99

{cslinux2:~/week08Lab2} ./myShell2

{myShell2}$ date ; ps

Thu Oct 10 21:47:30 CDT 2024

PID TTY TIME CMD

18557 pts/0 00:00:00 bash

19390 pts/0 00:00:00 myShell2

19392 pts/0 00:00:00 ps

{myShell2}$ whoami; hostname

txn200004

cslinux2.utdallas.edu

{myShell2}$ ls -l ; uname -a

total 120

-rw------- 1 txn200004 se 517 Oct 10 21:46 fileRedirect1.c

-rw------- 1 txn200004 se 1991 Oct 10 21:45 myShell1.c

-rwx--x--x 1 txn200004 se 13840 Oct 10 21:46 myShell2

-rw------- 1 txn200004 se 15469 Oct 10 21:45 myShell2.c

-rw------- 1 txn200004 se 1421 Oct 10 21:46 shell4pipe2.c

Linux cslinux2.utdallas.edu 3.10.0-1160.119.1.el7.x86\_64 #1 SMP Tue Jun 4 14:43:51 UTC 2024 x86\_64 x86\_64 x86\_64 GNU/Linux

{myShell2}$ ls -l myShell2.c

-rw------- 1 txn200004 se 15469 Oct 10 21:45 myShell2.c

{myShell2}$ ls -l | sort -r

total 120

-rwx--x--x 1 txn200004 se 13840 Oct 10 21:46 myShell2

-rw------- 1 txn200004 se 517 Oct 10 21:46 fileRedirect1.c

-rw------- 1 txn200004 se 1991 Oct 10 21:45 myShell1.c

-rw------- 1 txn200004 se 15469 Oct 10 21:45 myShell2.c

-rw------- 1 txn200004 se 1421 Oct 10 21:46 shell4pipe2.c

{myShell2}$ grep execvp myShell2.c

if(execvp(\*argv, argv) < 0){

if(execvp(\*argv, argv) < 0){

{myShell2}$ date > out.txt

{myShell2}$ wc < out.txt

1 6 29

{myShell2}$ exit

End Lab and Log out

{cslinux2:~/week08Lab2} date

Thu Oct 10 21:49:19 CDT 2024

{cslinux2:~/week08Lab2} ls -l

total 144

-rw------- 1 txn200004 se 517 Oct 10 21:46 fileRedirect1.c

-rw------- 1 txn200004 se 1991 Oct 10 21:45 myShell1.c

-rwx--x--x 1 txn200004 se 13840 Oct 10 21:46 myShell2

-rw------- 1 txn200004 se 15469 Oct 10 21:45 myShell2.c

-rw------- 1 txn200004 se 29 Oct 10 21:49 out.txt

-rw------- 1 txn200004 se 1421 Oct 10 21:46 shell4pipe2.c

{cslinux2:~/week08Lab2} uname -a

Linux cslinux2.utdallas.edu 3.10.0-1160.119.1.el7.x86\_64 #1 SMP Tue Jun 4 14:43:51 UTC 2024 x86\_64 x86\_64 x86\_64 GNU/Linux

{cslinux2:~/week08Lab2} exit

logout

─────────────────────────────────────────────────────────────────────────────────────────────────────────────────────────────────────────────────────────────────────────────────────────────────────────────────

Session stopped

- Press <Return> to exit tab

- Press R to restart session

- Press S to save terminal output to file