Carlos Gonzalez

Pseudocode

CIS 3207 Section 04

I will have two .c files One will be the main.c file and the other will be a utilities.c file that will function as a library later on.

Utilities.c

include “shell.h”

include stdio.h

include stdlib.h

include unistd.h

include string.h

include dirent.h

include errno.h

include sys/stat.,h

include sys/types.h

include sys/wait.h

Global extern char double pointer for enviro

Clear Function

Shell\_clr(char \*\* args){

System(clear)

Return 1

}

Simple shell quit

Int shell\_quit(char \*\*args){

Return 0;

}

Pause function

Int shell\_pause(char \*\*args){

Char buffer[500] = “”

Prints Shell is Paused. Press enter to resume

While(Strcmp(buffer, \n) != 0){

Fgets(Buffer, sizeof(Buffer), stdin

}

Prints shell has resumed

Return 1

}

Echo Function

Int shell(char \*\*args){

I = 1

While(Args[i] != null

Prints the string from args[i]

Increment i

}

Prints newline

Return 1

}

Directory Function

Int shell\_dir(char \*\*args){

DIR \*directory

Struct dirent \*reader

Directory = opendir(./)

If directory is not null

While reader is equal to readdir(Directory)

Print directory to screen

Close directory variable

}

}else{

Print error directory couldn’t be opened

}

Return 0

}

Change directory function

Int shell\_cd(char \*\*args){

If(Args[i] == NULL){

Char cwd[1024] to hold directory name

Get directory name

Prints Current Directory is cwd

Else{

If(there is a change directory error){

Prints cd error

}

}

}

Environment function

Int shell\_environ(char \*\*args){

I = 1

Char \*environmentvariable = \*environ

For I; environmentvariable; increment i{

Prints environmentvariable

Environmentvariable = \* (environ + i)

}

Return 0

}

Help function

Int shell\_help(char \*\*args){

Call clear function

Int j = 0

Struct winsize to hold dimensions of the window

Hold dimensions of wndow

Char MFile[1024[

File pointer

Open file for readme in read mode

While(fgets(MFile, 1024, filepoint) is not equal to NULL){

Prints string in MFile

J++

If j is equal to w.ws\_row – 1{

Char buffer[10]

While fgets mFile, 1024, filepointer is not equal to null{

Get string from stdin and store into buffer

If(Strcmp((buffer), \n){

Char \*token = strtok(MFile)

Prints string from MFile

}

If(Strcmp(buffer, q\n) is equal to 0){

Close file pointer

Return 1

}

}

Close file pointer

Return 1

}

}

Main.c File

Include Utilities.h

Define ASize = 1024

Read from stdin

Char \*readline(){

Char input[Asize]

Char \*read

Reads input and saves into input using fgets

Read = input

Return read

}

Printd Current directory

Void printHName{

Char name[ASize]

Char wd[1024]

If(getcwd(wd) is not equal to null){

Gethostname and save into wd

Prints hostname wd to stdou

}else{

Printss error message

}

}

Does most of the work

Int eval(char \* line)

Char \*argv[ASize]

Background = 0;

Calls Parse function with arguments line and argv

If(argv[ALength -1], “&” == 0){

Background = 1

Argv[Alength-1] = NULL;

}

X = Calls Redirect function with arguments argv and background

If(x = 2){

Return1

}

If(x = 0){

If(argv[0] == NULL{

Retrun 0

}

If(Calls isUtilities function with argument argv == 1){

Calls RunUtilities function with argument argv

}

Else{

Calls forkprogram with argv and Background

Return 1

}

Return 1

}

}

Parseline function

Void ParseLine( char \*line, char\* argv[ASize]){

Char \*savestring

Char t[] = {“\n}

Savestring = strtok(line, t);

Counter = 0

Argv[counter] = token

Counter++

While(Savestring is not equal to null){

Savestring = strtok(NULL, t)

Argv[counter] = savestring

Counter++

}

Alength++

}

RunUtilities function

Int RunUtilities(char \*argv[ASize]){

Int counter

For counter = 0; I < Csize; i++

If(Strcmp(argv[0], utilities[i]) == 0)

Return (\*utilities[i])(Argv);

Return 0

Redirect function

Void redirect(char\*argv[ASize], int flag, int i){

J = 1

In, out

Pid\_t pid

Savedin = dup(0)

Savedout = dup(1)

Pid = fork()

If(pid < 0){

Prints error

}

Else if(pid == 0){

If(flag is equal to 1){

Output = open(argv[j + 1], O\_wronly O\_creat, S\_irwxu, S\_irwig, S\_irwxo)

Dup2(output, 1)

Close(output)

Argv[j] Is set to null

Argv[j + 1] is set to null

J++

}

If(flag is equal to 2){

Input = open(argv[j + 1], O\_wronly O\_creat, S\_irwxu, S\_irwig, S\_irwxo)

Dup2(input, 0)

Close(input)

Argv[j] is set to null

Argv[j + 1] is set to null

J++

}

If(flag is equal to 3){

Output = open(argv[j + 1], O\_wronly O\_creat, S\_irwxu, S\_irwig, S\_irwxo)

Dup2(output, 1)

Close(output)

Argv[j] is set to null

Argv[j + 1] is set to null

J++

}

Execvp(argv[0], argv)

}

Else if(pid is greater than 0){

Waitpid(pid, NULL, wcontinued)

}

Dup2(savedin, 0)

Close(Savedin)

Dup2(savedout, 1)

Close(savedout)

}

ForkProgram

Void ForkProgram(char \*argv[Asize], int Background){

Pid\_t pid;

Pid = fork()

If pid is less than = 0{

Prints error

}

If pid is equal to 0{

If execvp returns an int less than 0{

Prints error

Exit

}

}

Else if( pid is greater than 0 and background is equal to 0){

Status

If waitpid is less than 0{

Prints error

}

}

}

Redirect function

Int redirect(char \*argv[ASize], int Background){

Redirected = 0

Char \*leftside[ASize]

Char \*rightside[ASize]

I = 0

While(Argv[i} is not equal to null){

If(ispipe returns 1){

pipeE(leftside, rightside, background)

return 2

}

If(strcmp(Argv[i] , “>”) is equal to 0){

Redirect(Argv, 1, i)

Redirected = 1

}

If(Strcmp(Argv[i] , “<” is equal to 0){

Redirect(argv, 1, i)

Redirected = 1

}

If(Strcmp(Argv[i] , “>>” is equal to 0){

Redirect(argv, 2, i)

Redirected = 1

}

I++

}

Return redirected

}

Ispipe function

Int ispipe(char \*argv[ASize[, char\* leftside[assize], char \*rightsize[ASize]){

Pipenum = 0

I = 0

J, k

While(Argv[i] is not equal to null){

If(Strcmp(Argv[i], |) is not equal to 0){

For j = 0 j less than I j++{

Leftside[j] = malloc for specified element

Char \*strarg = argv[j]

Leftside[j] = strarg

}

M = 0

For(k+ 1 k less than actual length k++W){

Rightside[m] = malloc for specified element

\*strarg = argv[k]

Rightside[m] = strarg

M++

}

Pipenum = 1

}

I++

}

Return pipenum

}

PipeE function

Void PipeE(char \*leftside[assize], char \*rightside){

Int fd[2]

Pid\_t pid

Pid\_t pid2

Int Sin = dup(0)

SOut = dup(1)

Pipe(fd)

Pid = fork()

If(pid is less than 0){

Prints error

Exit

}

If(pid equals 0){

Dup2(fd[1], stdout\_fileno)

Close(fd[0])

If(isUtilities(leftside) is equal to 1){

RunUtilities(leftside)

}

E;se{

If(Execvp(leftside[0], leftside) is less than 0){

Print error

}

}

}

Else{

Pid2 forks

If(pid2 is less than 0){

Prints error

Exit

}

Else if(pid2 is equal to 0){

Dup2(fd[0], stdin\_fileend)

Close fd[1]

If(isUtilities(rightside) equal to 1){

runUtilities(rightside)

}

Else{

If(execvp(rightside[0], rightside){

Prints error

}

}

}

Else if(pid 2 is greater than 0 and background == 0){

Close(fd[0]

Close(fd[1])

Waitpid(pid, null, 0)

Waitpid(pid2, null, 0)

}

}

}