EEE335 – Lab 2 Solutions

Your Own Shell

1. **Why are there internal functions and others located elsewhere in the operating system?**

Built-in commands are contained within the shell itself. Built-in commands are necessary to implement functionality impossible or inconvenient to obtain with separate utilities. They are implemented by the shell itself and do not to call another program.

1. **In your program, do you call system calls directly? Explain your answer.**

No, in our program, we call functions from different C libraries and it is these functions that do the system calls for us.

1. **What are the possible benefits of writing your own version of the *shell*?**

To gain experience with systems calls such as fork and exerve and with C programming in general.

1. **Why do we have a directory /usr/bin?**

The directory /usr/bin stores all the programs that are not needed during the boot process or for the core functionality of the systems.

1. **If you have a program called my\_program\_x located in /usr/my\_bin/my\_program\_x, how would your *shell* access it?**

We have to modify the environment variable PATH to include this directory.

1. **What is the purpose of the fork() function?**

To create a child process.

1. **What is the purpose of the execve() function?**

To specify the executable file to be run by the child process.

1. **Why should I use fork to call functions that are not internal (buitin)?**

These functions are implemented in a different executable files that need to be run by a process. If we used a single process, we would be able to launch the executable file using execve(), but we would not be able to return to the shell program afterward.

1. **What is the difference between the fork and execve system calls?**

Fork creates the process while execve tells the process to run a specific executable.

/\* shell.c

\*

\*

\* Author: Adrien Lapointe and Vincent Roberge and Stephen McKeon

\*/

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <unistd.h>

#include <dirent.h>

#include <sys/types.h>

#include <sys/wait.h>

#include <errno.h>

const unsigned int cmd\_max\_len = 1024;

const unsigned int max\_num\_arg = 64;

void welcome\_prompt();

void parse(char \*input, char \*\*argv);

void type\_prompt(char \*input);

void interpret\_command(char \*\*argv);

int main() {

char input[cmd\_max\_len]; // the input

char \*argv[max\_num\_arg]; // the arguments

welcome\_prompt();

while (1) {

type\_prompt(input);

parse(input, argv);

interpret\_command(argv);

}

return 0;

}

/\*

\* This functions prints the prompt message and accepts input from the user.

\*/

void type\_prompt(char \*input) {

printf("EEE335$");

if (input != NULL) {

int c = EOF;

int i = 0;

// Accepts input until user hits enter or end of file is encountered.

while ((c = getchar()) != '\n' && c != EOF) {

input[i++] = (char)c;

}

// Finish with a null terminating char

input[i] = '\0';

}

}

/\*

\* This function parses the user inputs.

\*/

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

char delimiters[] = " ";

int i = 0;

argv[i] = strtok(input, delimiters);

while (argv[i] != NULL)

{

i++;

argv[i] = strtok(NULL, delimiters);

}

}

/\*

\* This function interprets the parsed command that is entered by the user and

\* calls the appropriate built-in function or calls the appropriate program.

\*/

void interpret\_command(char \*\*argv) {

//Variables required to hold errors, process ids, statuses, and buffer size

int error; // check for errors

int pid; // process id

int status; // status of child creation

const int BUFFER\_SIZE = 500;

char buffer[BUFFER\_SIZE]; // buffer used when printing to the console

DIR \*curr\_dir; // ptr to current directory

struct dirent \*entry; // ptr to directory entry

// The 'exit' function

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

printf("Goodbye");

exit(0);

}

// The 'cd' function

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

printf("%s\n", argv[1]);

error = chdir(argv[1]);

if (error == 0)

printf("Changing to: %s\n", argv[1]);

else

printf("No such directory\n");

}

// The 'ls' function

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

// Get the current directory and open it

getcwd(buffer, BUFFER\_SIZE);

curr\_dir = opendir(buffer);

// If no error, print all entries

if(curr\_dir != NULL){

while((entry = readdir(curr\_dir)) != NULL){

printf("%s\n", entry->d\_name);

}

}

}

// The 'pwd' function

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

getcwd(buffer, BUFFER\_SIZE);

if(buffer != NULL)

printf("Current directory: %s\n", buffer);

}

//Take care of all other functions

else{

pid = fork();

//Child process

if(pid == 0){

error = execvp(argv[0], argv);

if (error != 0){

printf("%s: %s\n", argv[0], strerror(errno));

exit(0);

}

}

//Parent process

else{

//Wait for child to finish

waitpid(pid, &status, 0);

}

// If the child process is writing to stdout, this wait will ensure the text

// can be printed to the screen, before our shell prints the next prompt

usleep(10000); // wait 10 ms

}

}

/\*

\* This function prints the welcome message.

\*/

void welcome\_prompt() {

int num\_padding = 41;

int i;

// Prints the first line of padding.

for (i = 0; i < num\_padding; i++)

printf("#");

printf("\n#\tWelcome to the EEE335 shell\t#\n");

for (i = 0; i < num\_padding; i++)

printf("#");

printf("\n\n");

}