Unix Shell

CprE 308

Project 1

Neh Batwara

Contents

1 Summary 2

2 Code: 2

3 Included Files: 5

1 Summary

In this project we created our own version of the UNIX shell. A UNIX shell is simply an interactive interface between the OS and the user. It repeatedly accepts input from the user and initiates processes on the OS based on that input.

A challenging part of this project was string parsing since that was how the commands that the user inputs are executed by the shell.The shell uses fork( ) to call a program which creates a child process. To execute the command the child process calls the execvp( ) which exeutes the child process.

I also used system calls to process the following process/commands:
1.chdir( ) to change the working directory with the string that gets passed in by the user inpu with cd
2.getcwd( ) to get the current working directory
3.getenv/setenv to retrieve and set environment variables.
4.waitpid( ) to wait for the child process to exit or to get exit status

2 Code:

The following code is for the main Unix Shell.

```
#include <stdio.h>
#include <unistd.h>
#include "p1_Unix_Shell.h"
#include <string.h>
#include <sys/types.h>
#include <unistd.h>
#include <sys/wait.h>
#include <errno.h>
#include <stdlib.h>
/*Maximum legnth of input allowed to the shell.*/
#define INPUT_LENTH 100
/*Maximum number of argument allowed to be passed to the shell. */
#define ARGUMENT_BUFFER 100
#define CHAR_BUFFER 100

/*
 *This function should parse the user input into an array.
 */
int parse_input(char * cmd, char * argv[])
{
        int i = 1;
    argv[0] = strtok(cmd, " \n");
        while(argv[i-1] != NULL && i < (ARGUMENT_BUFFER - 1))
        {
        argv[i] = strtok(NULL, " \n");
    i++;
        }
        return i-1;
}

/*
 *This function parses the arguments passed from the command line to the shell..
 */
char * parse_argument(int argc, char ** argv)
{
        int arg;
        opterr = 0;
        char * prompt = NULL;
        while((arg = getopt(argc, argv, "p:")) != -1)
        {
                if (arg == 'p'){
                prompt = optarg;
                }
                else{
                abort();
                }

        }

        if(prompt == NULL)
        {
                prompt = "308sh";
        }

        return prompt;

}

/*
 *This is the main function with the while loop that executes the shell.
* It take the user imput from the command line and parses it to the shell program.
 */
int main(int argc, char ** argv)
{
        char * prompt = parse_argument(argc, argv);
        char input[INPUT_LENTH];
        while(1)
        {
                char * argv[ARGUMENT_BUFFER];
                printf(">>>%s> ", prompt);
                fgets(input, INPUT_LENTH, stdin);
                int lastIndex = parse_input(input, argv);
                if(lastIndex != 0)
                {
                        if(!strcmp(argv[0], "exit"))
                        {
                                break;
                        }
                        else if(!strcmp(argv[0], "cd"))
                        {
                                cd(argv);
                        }
                        else if(!strcmp(argv[0], "cwd"))
                        {
                        printf(" Current Working Directory : %s\n", getcwd(NULL,CHAR_BUFFER));
                        }

                        else if(!strcmp(argv[0], "pid"))
                        {
                                printf("PID = %i\n", getpid());
                        }
                        else if(!strcmp(argv[0], "ppid"))
                        {
                                printf("PPID = %i\n", getppid());
                        }
                        else if(!strcmp(argv[lastIndex-1], "&"))
                        {
                                argv[lastIndex-1] = NULL;
                                execute(argv);
                        }
                        else
                        {
                                execute_block(argv);
                        }
                }
                int status;
                pid_t process_child = waitpid(-1, &status, WNOHANG);
                if(process_child > 0)
                {
                        printf("process %i exited\n", process_child);
                        processStatus(process_child, status);
                }
        }
        return 0;
}

/*
 *Fucntion when cd is called.
 */
void cd(char * argv[])
{
        if(argv[1] == NULL)
        {
                printf("No paramaters supplied to cd\n");
        }
        else
        {
                if(chdir(argv[1]) == -1)
                {
                        printf("Cd failed - %s\n", strerror(errno));
                }
        }
}


/*
 * This function prints the status so the user can see it.
 */
void processStatus(pid_t childPid, int status)
{
        if(WIFSIGNALED(status))
        {
                printf("Child %i exited with signal %d\n", childPid, WTERMSIG(status));
        }
        else if(WIFEXITED(status))
        {
                printf("Child %i exited with return code %d\n", childPid, WEXITSTATUS(status));
        }

}

/*
 *execute_block waits for the process to finish.
 */
void execute_block(char * argv[])
{
        pid_t pid = fork();
        if(pid == 0)
        {
                printf("pid: %i\n", getpid());
                execvp(*argv, argv);
                printf("Unknown command.\n");
                exit(0);
        }
        else
        {
                int status;
                waitpid(pid, &status, 0);
                processStatus(pid, status);
        }

}

/*
 *Exceute does not block the main process and waits for the child process to finish.
 */
void execute(char * argv[])
{
        pid_t pid = fork();
        if(pid == 0)
        {
                printf("\n PID: %i\n", getpid());
                execvp(*argv, argv);
                printf("PID: %i failed - %s\n", getpid(), strerror(errno));
        }
}
```

3 Included Files:

1. makefileċmake
2.p1_Unix_Shellċ[Main C File]
3.p1_Unix_Shellḣ [Header File]
4.p1_Unix_Shellȯ [Object File]
5.READMEṫxt [text file for instructions]