Powerpuff Unix Shell

The Unix Shell

Course Title: Operating System
Course Code: CSE325

Project Report

Source Code: `https://github.com/shaykhsiddique/Powerpuff-UNIX-Shell`

Submitted To

Md. Ashraful Islam
Lecturer,
Department of Computer Science & Engineering,
East West University, Bangladesh.

Author

Shaykh Siddique(2016-1-60-053)
Thajiba Tabassum(2015-02-60-022),
S. M. Saikat Hossain(2016-2-60-068),
Md.Jadid Mostafiz(2014-3-60-029),
Shaik Md. Ibnay-Momen(2016-2-60-100),
Jannatul Ferdous Sorna(2016-1-60-029)

1 Introduction

A Shell provides us with a command line interface to the Unix system. It gathers input from you and executes programs based on that input. When a program finishes executing, it displays that program's output.
Shell is an environment in which we can run our commands, programs, and shell scripts. There are different flavors of a shell, just as there are different flavors of operating systems. Each flavor of shell has its own set of recognized commands and functions.
Shells are mostly used to make control of remote connection of a server.

2 Software Description and Facilities

This Software is just a sample Unix Shell developed in **C**.

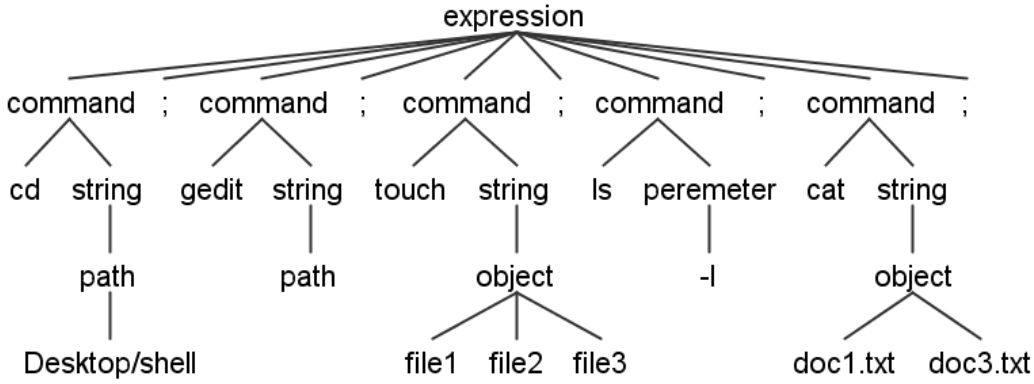
2.1 Parsing

As We are working with command line, we must need to parse all the commands. A single command will be separated by semicolon ';' and in a single command - Command words and parameters are separated by space.
We used a parsing simulation software named **ANTLR** to simulate a basic parse tree.

Input Commands:

```
cd Desktop/shell;
gedit;
touch file1 file2 file3;
ls -l;
cat doc1.txt doc3.txt;
```

Parse Tree



2.2 Implementation of Commands

cd [path]
`cd` is a command of changing directory. It expects one parameter, the path of the directory to change in. Using a function called `chdir(directory)`, where parameter directory is the path.

ls
`ls` is a command which will list all the files and directories of current working directory. Opening current directory, read and print all files and folder names. Using `opendir(param)`, where param is the path of directory.

For reading files and folder names, `readdir(dr)`, where dr is the current directory object and this function is return the names of all files and folders.

mkdir [folder]
Using a function `mkdir(param1, param2)`, where parameter one is the name of new folder and parameter two is the permission of this folder. Here we use the permission (777) read, write, and execute.

touch [file]
Creating a file just in write mode. Expecting one or more parameter.

cat [file]
Open all of those files, read and print in console. Expecting one or more parameter.

echo [string]
Just printing all strings. Expecting one or more parameter.

cp [source] [destination]
Opening the source file, load all the data into program. Then go to destination file, create that destination file and write all data.

exit
Command exit is used to terminate the program.

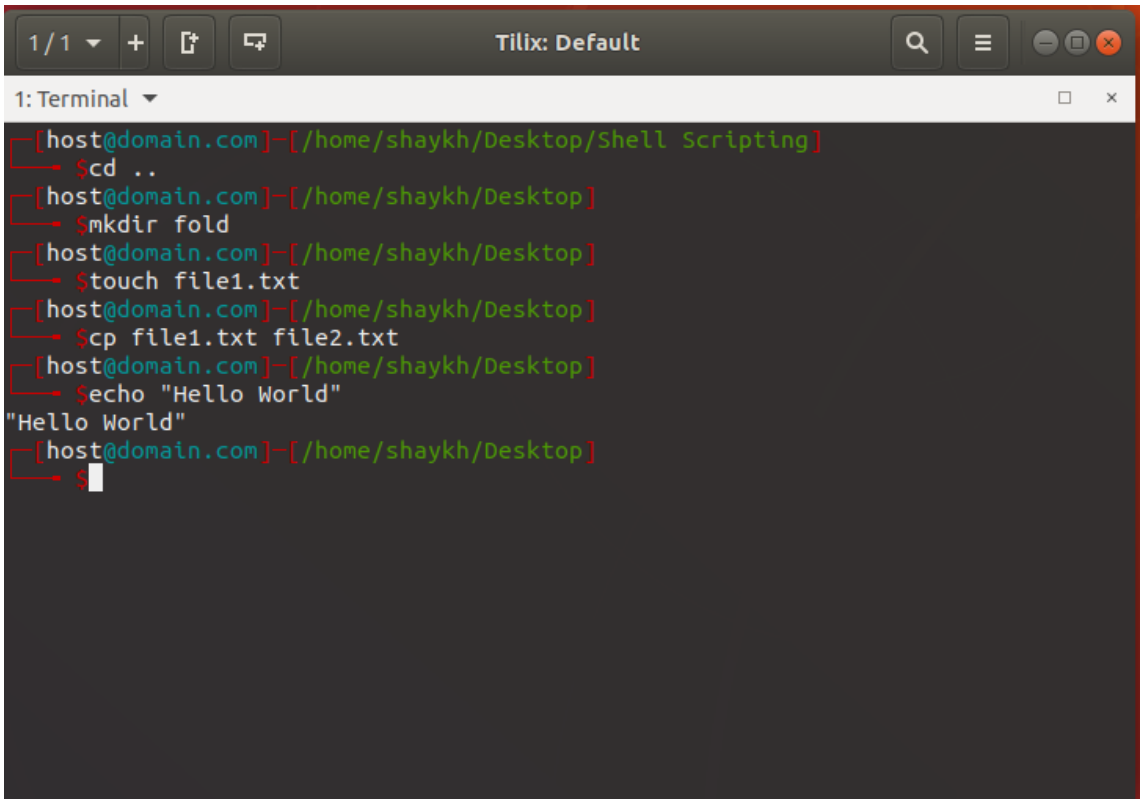
2.3 Documentation

In Linux terminal this commands can be used to setup Powerpuff shell.

```
$ wget https://raw.githubusercontent.com/shaykhsiddique/Powerpuff
$ gcc powerpuff_unix_shell.c  o powerpuff
$ ./powerpuff
```

First command is for downloading the shell, next one is for compiling and the last one is for execution.

2.4 Screenshots



3 Conclusion

3.1 Goals

The goal of this software is to introduce a unix based shell with some simple command performing on Linux operating system.

Without a shell we have just plenty of files, but we cannot access this. For remote access of a server shell is one of the best way. Many kind of web server uses shell for maintaining their client file access. All kind of VPS server uses shell to communicate with kernel.

3.2 Source Codes

Source Code link: `https://github.com/shaykhsiddique/Powerpuff-UNIX-Shell`

```
#include<stdio.h>
#include <unistd.h>
#include <sys/types.h>
#include <stdlib.h>
#include <string.h>
#include <dirent.h>
    //define custom variables
#define debugi(acb) printf("%d\n", acb)
#define debugs(acb) printf("%s\n", acb)


#define sizeofarray 100
#define ANSI_COLOR_RED      "\x1b[31m"
#define ANSI_COLOR_GREEN    "\x1b[32m"
#define ANSI_COLOR_YELLOW   "\x1b[33m"
#define ANSI_COLOR_BLUE     "\x1b[34m"
#define ANSI_COLOR_MAGENTA  "\x1b[35m"
#define ANSI_COLOR_CYAN     "\x1b[36m"
#define ANSI_COLOR_RESET    "\x1b[0m"


char current_working_dir[sizeofarray];

//one_process is a global structure of all commands and commnd_paramrt

//single command structure
struct single_commands{
    char comm_line[sizeofarray];
}all_commands[sizeofarray], all_paramtr[sizeofarray];


struct sngl_comnd{
    char commnd_word[sizeofarray];
```

```
    char commnd_paramrt[sizeofarray][sizeofarray];
    int numOfParm;
}one_process[sizeofarray];


struct single_commands input_command(){
    struct single_commands s1;
    gets(s1.comm_line);

    return s1;
}


void print_path(){
    getcwd(current_working_dir, sizeof(current_working_dir)); //path of c
    printf(ANSI_COLOR_RED "        ["ANSI_COLOR_RESET"host"ANSI_COLOR_CYAN"
}



int command_parsing(struct single_commands comd_all){

    char *onecommand[500];

    char* token_line = strtok(comd_all.comm_line, ";");
//    checking ; for finding multiple commands
    onecommand[0] = token_line;
    int ind=1;
    while (token_line != NULL) {

                token_line = strtok(NULL, ";");
            if(token_line == NULL) break;
                onecommand[ind] = token_line;
                ind++;
                //    checking [space] for finding multiple commands

    }

    for(int i=0; i<ind && onecommand[i]!=NULL; i++){
        char* token_word = strtok(onecommand[i], " ");
//        main command parsing
        strcpy(one_process[i].commnd_word, token_word);
        int j=0;
        while(token_word!=NULL){
//      parameter parsing
            token_word = strtok(NULL, " ");
            if(token_word == NULL) break;
            strcpy(one_process[i].commnd_paramrt[j], token_word);
```

```
            j++;

        }
        one_process[i].numOfParm=j;

    }
    return ind;
}



void print_process_str(int total_no_of_process){ //this is a sample of te
    for(int i=0; i<total_no_of_process && one_process[i].commnd_word!=NUI
        printf("Command: %s  ——>Perameters: ", one_process[i].commnd_word


        for(int j=0; j<one_process[i].numOfParm; j++){
            printf("%s ", one_process[i].commnd_paramrt[j]);

        }
        printf("\n");
    }

}

void cmd_cd(int process_id){
char *directory = one_process[process_id].commnd_paramrt;


   int ret = chdir (directory);    //On success, zero is returned.
On error, -1 is returned, and errno is set appropriately

    if(ret){
        perror("Error ");
    }
}


void execute_ls(int process_id){
    pid_t pid =fork();
    pid_t tpid;
    if(!pid){
        //child process started
//          char *const parmList[] = {"/bin/ls", "", current_working_dir, N

        struct dirent *de;
        DIR *dr = opendir(".");
```

```
        if (dr == NULL) // opendir returns NULL if couldn't open director
        {
            printf("Could not open current directory" );
        }
        while ((de = readdir(dr)) != NULL)
                printf("%s\n", de->d_name);

        closedir(dr);

    }else if(pid==-1){
        //error to create process. Need error msg
    }else{
//          parent process remaining part
            pid_t tpid = wait(&pid);    //parent process wait untill child

    }
}

void execute_mkdir(int process_id){
    for(int j=0; j<one_process[process_id].numOfParm; j++){
        int result = mkdir(one_process[process_id].commnd_paramrt[j], 07
        //Upon successful completion, mkdir() shall return 0. Otherwise,
        if(result==-1){
            printf("Error: Cannot create directory...\n");
            sleep(1);
        }

    }
}

void execute_touch(int process_id){
    for(int j=0; j<one_process[process_id].numOfParm; j++){
        FILE *fp = fopen(one_process[process_id].commnd_paramrt[j], "wb")
        fclose(fp);
    }
}



void execute_cp(int process_id){
    //cointains 2 parameter one is source and another is destination
    if(one_process[process_id].numOfParm == 2){
        //copying files
        int last_char_id =strlen(one_process[process_id].commnd_paramrt[1
        if(one_process[process_id].commnd_paramrt[1][last_char_id] == '/'
            printf("No destination file name...\n");
            return;
        }
```

```
            char ch;
            char file_datas[100000];  //temopary data copying
            FILE *fp_src;
    //        file_name = one_process[process_id].commnd_paramrt[j];
            fp_src = fopen(one_process[process_id].commnd_paramrt[0], "r"
            if (fp_src == NULL){
              perror("Error in source file.");
            }else{

                int id=0;
                while((ch = fgetc(fp_src)) != EOF){
                    file_datas[id] = ch;
                    id++;
                    //strcat(file_datas, ch);
                }

            }
            fclose(fp_src);
//destination file copying
            FILE *fp_des;
            fp_des = fopen(one_process[process_id].commnd_paramrt[1], "wb
            fprintf(fp_des, "%s", file_datas); //printing in new file
            fclose(fp_des);
    }else{
        //parameter error.
        printf("Error: Expect 2 parameter. Command: \"cp [source] [destin

    }
}

void execute_cat(int process_id){
    for(int j=0; j<one_process[process_id].numOfParm; j++){
        char ch;
        FILE *fp;
//        file_name = one_process[process_id].commnd_paramrt[j];
        fp = fopen(one_process[process_id].commnd_paramrt[j], "r"); // re
        if (fp == NULL){
            perror("Error: \n");
            exit(EXIT_FAILURE);
        }
        //writing on console
        while((ch = fgetc(fp)) != EOF)
            printf("%c", ch);
        printf("\n");
        fclose(fp);  //close file
    }
}
```

```
void execute_echo(int process_id){
    for(int j=0; j<one_process[process_id].numOfParm; j++){
        int len = strlen(one_process[process_id].commnd_paramrt[j]);
        for(int k=0; k<len; k++){
            if(one_process[process_id].commnd_paramrt[j][k] != '"');
            printf("%c", one_process[process_id].commnd_paramrt[j][k]);
        }
//          printf("%s", one_process[process_id].commnd_paramrt[j]);
        printf(" ");
    }
    printf("\n");
}


// for all precess executing commands
void all_process_management(int numofProcess){
        //cd, ls, mkdir, cat, touch, cp, echo
        int errr=0;
    for(int i=0; i<numofProcess && one_process[i].commnd_word!=NULL; i++
        int pharmNo = one_process[i].numOfParm;
        if(strcmp(one_process[i].commnd_word, "cd") == 0){
            if(pharmNo) cmd_cd(i);
            else errr = 1;
        }else if(strcmp(one_process[i].commnd_word, "ls") == 0){
            if(!pharmNo) execute_ls(i);
            else errr = 1;
        }else if(strcmp(one_process[i].commnd_word, "mkdir") == 0){
            if(pharmNo) execute_mkdir(i);
            else errr = 1;
        }else if(strcmp(one_process[i].commnd_word, "touch") == 0){
            if(pharmNo) execute_touch(i);
            else errr = 1;
        }else if(strcmp(one_process[i].commnd_word, "cp") == 0){
            execute_cp(i);
        }else if(strcmp(one_process[i].commnd_word, "cat") == 0){
            if(pharmNo) execute_cat(i);
            else errr = 1;
        }else if(strcmp(one_process[i].commnd_word, "echo") == 0){
            if(pharmNo) execute_echo(i);
            else errr = 1;
        }else if(strcmp(one_process[i].commnd_word, "clear") == 0){
            if(!pharmNo) system("clear");
            else errr = 1;
        }else if(strcmp(one_process[i].commnd_word, "exit") == 0){
            exit(0);
        }
        else {
            printf("No command found..\n");
```

```
        };
    }

    if(errr) printf("Command Error. \n");
}

void clear_commands(int numofProcess){
    for(int i=0; i<numofProcess; i++){
        strcpy(one_process[i].commnd_word, "");
    }
}

int main(int argc, char *argv[]){
    struct single_commands cmmd;

    while(1){
        print_path();
        cmmd = input_command();
        int noOfProcess = command_parsing(cmmd);
        all_process_management(noOfProcess);
//          print_process_str(noOfProcess);
        clear_commands(noOfProcess);  //clear all command in global varia
    }
    return 0;
}
```