MANIPAL INSTITUTE OF TECHNOLOGY
MANIPAL
A Constituent Institution of Manipal University

Linux Shell

REPORT

Submitted by

Chitrala Dhruv – 200905210

Varshith Sreepuram – 200905208

Gouraiahgari Uday Kiran – 200905190

Appala Venkata Sai Jwala Yeshwanth – 200905278

Chittoju Sai Dhanush Kumar – 200905120

OPERATING SYSTEM PROJECT

MAKING YOUR OWN LINUX SHELL IN C

We all use the built-in terminal window in Linux distributions like Ubuntu, Fedora, etc. But how do they work?

In this mini project we are going to demonstrate some under the hood features and algorithms what actually work inside a shell.

PROBLEM STATEMENT

All Linux OS have a terminal window to write in commands. But how are they executed properly after they are entered?

ALGORITHM

1. Command is entered.
2. Parsing, which is breaking up of commands into individual words and strings.
3. Checking for special characters.
4. Checking if built-in commands are asked for.
5. Executing system commands and libraries by forking a child and calling execvp.
6. Printing current directory name and asking for next input.

IMPLEMENTATION

Code :

```
#include <stdio.h>
#include <unistd.h>
#include <stdlib.h>
#include <unistd.h>
#include <string.h>
#include <errno.h>
#include <sys/stat.h>
#include <sys/types.h>
#include <sys/wait.h>
```

```
#include <time.h>


/*
    Function declarations of all commands
*/
int ushell_grep(char **args);
int ushell_chmod(char **args);
int ushell_echo(char **args);
int ushell_cd(char **args);
int ushell_pwd(char **args);
void ushell_help(char **args);
int ushell_exit(char **args);
int ushell_day(char **args);



char *ushell_list_commands[] = {
    "grep",
    "chmod",
    "echo",
    "cd",
    "pwd",
    "exit",
    "help",
    "day"
};

char *ushell_desc_commands[] = {
    "grep searches for PATTERNS in each FILE.",
    "Displays the current Permissions of the file and Changes the permission",
    "Display's Process ID if '$$' follows echo if not prints the same string followed by echo",
    "Changes to  Specified Valid Directory",
    "Display's the Present working Directory",
    "This command will close the execution of shell",
    "This will list all the commands that are built into the shell",
    "Returns day of the week"
};

int (*ushell_func_commands[])(char **) = {
    &ushell_grep,
    &ushell_chmod,
    &ushell_echo,
    &ushell_cd,
    &ushell_pwd,
    &ushell_exit,
    &ushell_help,
    &ushell_day
};

int ushell_num_commands() {
    return sizeof(ushell_list_commands) / sizeof(char *);
}

// Function where the system command is executed
int ushell_launch(char **args)
{
  pid_t pid, wpid;
  int status;

  pid = fork();
  if (pid == 0) {
    // Child process
    if (execvp(args[0], args) == -1) {
      perror("lsh");
    }
    exit(EXIT_FAILURE);
  } else if (pid < 0) {
    // Error forking
    perror("lsh");
  } else {
    // Parent process
    do {
      wpid = waitpid(pid, &status, WUNTRACED);
    } while (!WIFEXITED(status) && !WIFSIGNALED(status));
  }

  return 1;
}

// All the seven function of Seven commands go here
//exit is already done
```

```
//echo function
int ushell_echo(char**args) {
    int i = 1;
    if(strcmp(args[1], "$$")==0) {
        printf("%d\n", (int)getpid());
    }
    else {
        while (args[i] != NULL) {
            printf("%s", args[i]);
            i++;
        }
    }
    printf("\n");
    return 1;
}


//day
int ushell_day(char ** args) {
    time_t t = time(NULL);
    struct tm tm = *localtime(&t);
    int s, i;
    int month[12] = {31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31};
    char* week[] = {
        "Sunday",
        "Monday",
        "Tuesday",
        "Wednesday",
        "Thursday",
        "Friday",
        "Saturday"
    };
    int date = tm.tm_mday;
    int mon = tm.tm_mon + 1;
    int year = tm.tm_year + 1900;

    if ((year % 400 == 0) || (year% 4 == 0) && (year % 100 != 0))
        month[1] = 29;
    for (i = 0; i < mon - 1; i++)
```

```
        s = s + month[i];
    s = s + (date + year + (year/4) - 2);
    s = s%7;
    printf("%s\n", week[s]);

    return 1;
}
//cd
int ushell_cd(char**args) {
    if(args[1] == NULL) {
        printf("%s\n", getenv("PATH"));
        if(chdir(getenv("PATH")) == -1) {
            fprintf(stderr, "getenv error: %s\n", strerror(errno));
        }
    }
    else {
        if (chdir(args[1]) == -1) {
            fprintf(stderr, "chdir error: %s\n", strerror(errno));
        }
    }

    return 1;
}

//Function for pwd
#define FILENAME_MAXI 1024

int ushell_pwd(char **args) {
    char buff[FILENAME_MAXI];
    printf("%s\n", getcwd(buff, FILENAME_MAXI));
    return 1;
}

//Function for chmod
//syntax: ./chmod s.txt rwxrwxrwx
void output_permissions(mode_t m)
{
    putchar( m & S_IRUSR ? 'r' : '-' );
    putchar( m & S_IWUSR ? 'w' : '-' );
    putchar( m & S_IXUSR ? 'x' : '-' );
    putchar( m & S_IRGRP ? 'r' : '-' );
    putchar( m & S_IWGRP ? 'w' : '-' );
    putchar( m & S_IXGRP ? 'x' : '-' );
    putchar( m & S_IROTH ? 'r' : '-' );
    putchar( m & S_IWOTH ? 'w' : '-' );
    putchar( m & S_IXOTH ? 'x' : '-' );
    putchar('\n');
}

int ushell_chmod(char **argv)
{
    const char *filename;
    struct stat fs;
    int r;

    filename = argv[1];
    printf("Permissions for '%s':\n",filename);
    r = stat(filename,&fs);
    if( r==-1)
    {
        fprintf(stderr,"Error reading '%s'\n",filename);
        exit(1);
    }

    /* output the current permissions */
    puts("Current permissions:");
    output_permissions(fs.st_mode);

    char perm[10];
    strcpy(perm, argv[2]);
    mode_t mode = 0;

        if (perm[0] == 'r')
            mode |= 0400;
        if (perm[1] == 'w')
            mode |= 0200;
        if (perm[2] == 'x')
            mode |= 0100;
        if (perm[3] == 'r')
            mode |= 0040;
        if (perm[4] == 'w')
            mode |= 0020;
        if (perm[5] == 'x')
            mode |= 0010;
        if (perm[6] == 'r')
            mode |= 0004;
        if (perm[7] == 'w')
            mode |= 0002;
        if (perm[8] == 'x')
            mode |= 0001;
    r = chmod( filename,mode );
    if( r!=0)
    {
        fprintf(stderr,"Unable to reset permissions on '%s'\n",filename);
        exit(1);
    }

    puts("Updated permissions:");
    stat(filename,&fs);
    output_permissions(fs.st_mode);

    return 1;
}

//Function for grep
//syntax: ./grep line s.txt
int ushell_grep(char** argv)
{
    char fn[30],pat[30],temp[200];
    FILE *fp;
    int line=0;
    fp=fopen(argv[2],"r");
    while(!feof(fp))
    {
    fgets(temp,1000,fp);
    line++;
    if(strstr(temp,argv[1])!=NULL)
    printf("Line-%d: %s",line,temp);
    }
    fclose(fp);
```

```
        return 1;
}
//Function for Help
void ushell_help(char **args) {
    int i;
    printf("-------------------------Welcome to UNIX shell-------------------------\n");
    printf("Type program names and arguments, and hit enter.\n");
    printf("The following are built-in commands\n");

    for(i = 0; i < ushell_num_commands(); i++) {
        printf(" %s - %s\n\n", ushell_list_commands[i], ushell_desc_commands[i]);
    }
}
//Function for exit
int ushell_exit(char **args) {
    printf("Exiting.....\n");
    return 0;
}

/*
    This fucntion will check the given input with all the available
    functions and executes them
    Input - array of tokens generated from ushell_splitline
    Output - Integer (return 0 only when the command is exit).
*/
int ushell_execute(char **args) {
    int i;

    //Empty argument
    if (args[0] == NULL) {
        return 1;
    }

    for (i = 0; i < ushell_num_commands(); i++){
        if (strcmp(args[0], ushell_list_commands[i]) == 0) {
            return (*ushell_func_commands[i])(args);
        }
    }
    return ushell_launch(args);
}
```

```
#define USHELL_TOK_DELIM " \t\r\n\a"
/*
    parameters - A variable line of type char *
    return - array of tokens from the input line

    This function splits the line into array of tokens at " \t\r\n\a"
*/
char **ushell_splitline(char *line){
    int bufsize = 64, position = 0;
    char **tokens = (char **)malloc(bufsize * sizeof(char *));
    char *token;

    if(!tokens) {
        fprintf(stderr, "ushell: Memory allocation error\n");
        exit(EXIT_FAILURE);
    }

    token = strtok(line, USHELL_TOK_DELIM);
    while(token != NULL) {
        tokens[position] = token;
        position++;
        if (position >= bufsize) {
            bufsize += 64;
            tokens = (char **)realloc(token, bufsize *sizeof(char *));
            if(!tokens) {
                fprintf(stderr, "ushell: Memory allocation error\n");
                exit(EXIT_FAILURE);
            }
        }

        token = strtok(NULL, USHELL_TOK_DELIM);
    }
    tokens[position] = NULL;
    return tokens;
}
    char* ushell_readline(){
    char *line = NULL;
    ssize_t bufsize = 0;

    if (getline(&line, &bufsize, stdin) == -1) {
        if(feof(stdin)){
            //When user types ctrl+D EOF occurs and shell exit with EXIT_SUCCESS
            exit(EXIT_SUCCESS);
        }else{
            perror("readline");
            exit(EXIT_FAILURE);
        }
    }
    return line;
}

void ushell_loop(){
    char *line;
    char **args;
    int status;

    do{
        int x = ushell_pwd(args);
        printf(">>");
        line = ushell_readline();
        args = ushell_splitline(line);
        status = ushell_execute(args);

        free(line);
        free(args);
    }while(status);
}

int main(int argc, char** argv){
    ushell_loop();
    return EXIT_SUCCESS;
}
```

▪ REFERENCES

www.geeksforgeeks.org

www.digitalocean.com

and some random GitHub repositories.