OS Lab

Andrew Nady

900184042 - February 23, 2021

Introduction

I created a shell which support “cd”, “ls”, “pwd”, “grep”, “cat”, piping, input and output redirection, it almost support all command as the normal shell do.

So what is my design to create all of these features to my shell, one important part that I relied on system calls like execvp, getenv, dup2 , etc.

So what I did some functions to help me implement these features

Cd

I make the default directory is the root (Home of each computer).

```
void change_dir(char *command, char *folder)
{

    char s[100];
    if (strcmp(folder, "~") == 0)
        chdir(getenv("HOME"));
    else if (chdir(folder) != 0)
        perror("chdir() to /error failed");

    printf("current Path:%s\n", getcwd(s, 100));


}
```

```
if (strcmp(argv[0], "cd") == 0)
{
    change_dir(argv[0], argv[1]);
}
```

Output redirection

a) I created a function to check if there is “>” character or not

b) I created a function to delete the character “>” and all the strings after it.

c) I created a function to get the file name in order to forward it to dup2 function and call the delete function inside the get file function.

```
bool check_for_redirection(char **argv)
{
    int i = 0;
    while (argv[i] != NULL)
    {

        if (strcmp(argv[i], ">") == 0)
        {
            return true;
        }
        i++;
    }
    return false;
}
void deleteUtil(char **argv, int pos)
{

    while (argv[pos] != NULL)
    {

        int j = pos;

        while (argv[j] != NULL)
        {
            argv[j] = argv[j + 1];
            j++;
        }
    }
}
char *getfile(char **argv)
{
    char *text;
    int x = -1;
    bool flag = 0;
    int i = 0;
    while (argv[i] != NULL)
    {

        if (strcmp(argv[i], ">") == 0)
        {
            x = i + 1;
            flag = 1;
            text = argv[i + 1];
        }
        if (flag)
            deleteUtil(argv, i);

        i++;
    }

    return text;
```

```
if (check_for_redirection(argv))
{
    directed_file_output = getfile(argv);

    if ((newfd = open(directed_file_output, O_CREAT | O_TRUNC | O_WRONLY, 0644)) < 0)
    {
        perror(directed_file_output); /* open failed */
        exit(1);
    }
    printf("writing output of the command %s to \"%s\"\n", argv[0], directed_file_output);
    dup2(newfd, 1);
}
```

Input redirection

I created 4 functions to do the input redirection:

a) check if "<" character exists or not

b) delete "<" and all after it except ">" and what follow it

c) git file function to use it in dup int the function input redirection

d) discussed in c)

```
bool check_for_redirection_input(char **argv)
{
    int i = 0;
    while (argv[i] != NULL)
    {

        if (strcmp(argv[i], "<") == 0)
        {
            return true;
        }
        i++;
    }
    return false;
}
void deleteUtil_input(char **argv, int pos)
{

    while (argv[pos] != NULL && strcmp(argv[pos], ">") != 0)
    {

        int j = pos;

        while (argv[j] != NULL)
        {
            argv[j] = argv[j + 1];
            j++;
        }
    }
}
char *getfile_input(char **argv)
{
    char *text;
    int x = -1;
    bool flag = 0;
    int i = 0;
    while (argv[i] != NULL)
    {

        if (strcmp(argv[i], "<") == 0)
        {
            x = i + 1;
            flag = 1;
            text = argv[i + 1];
        }
        if (flag)
            deleteUtil_input(argv, i);

        i++;
    }

    return text;
}

void input_redirection(char *directed_file)
{
    int newfd;
    newfd = open(directed_file, O_RDONLY);
    dup2(newfd, 0);


    return;
}
```

```
else if (check_for_redirection_input(argv))
{
    directed_file_input = getfile_input(argv);
    input_redirection(directed_file_input);
}
```

Pipes

It was the hardest part in the assignment according to me

It contains the largest code, so I wont be able to get the code here,

but basically the ideas:
A) parsing the argue array to make 3d of command of strings which is array of chars and then use this 3d array in the pipe
B). Initializing the number of pipes = size of commands -1
C) then create the pipes, and loop over them
D) in the parent, it execute the first command and then input the output to the next pipe to execute the next command and so on

Variables

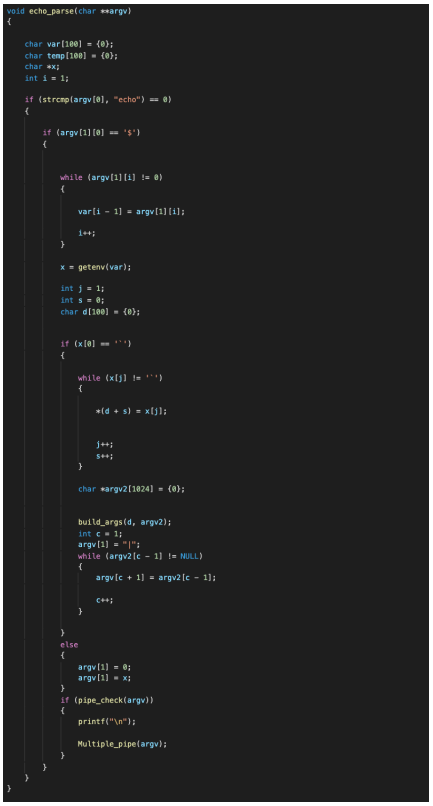
I supported the assignments of variables if they are connected “X=y” or contain spaces “X = y”

So I used all theses functions to handle both tracks

A) one check if there is equality in the argv[0] which indicates that it is connected

B) One parsing_assignment is using setenv and handle some cases in case it is not connected

C) One for the echo, I use inside it multiple pipes function in order to handle the cases that need to do more that command like “echo $x” where x=`cat /etc/hosts`.



```
void echo_parse(char **argv)
{

    char var[100] = {0};
    char temp[100] = {0};
    char *x;
    int i = 1;

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

        if (argv[1][0] == '$')
        {


            while (argv[1][i] != 0)
            {

                var[i - 1] = argv[1][i];

                i++;
            }

            x = getenv(var);

            int j = 1;
            int s = 0;
            char d[100] = {0};


            if (x[0] == '`')
            {

                while (x[j] != '`')
                {

                    *(d + s) = x[j];


                    j++;
                    s++;
                }

                char *argv2[1024] = {0};


                build_args(d, argv2);
                int c = 1;
                argv[1] = "|";
                while (argv2[c - 1] != NULL)
                {
                    argv[c + 1] = argv2[c - 1];

                    c++;
                }

            }
            else
            {
                argv[1] = 0;
                argv[1] = x;
            }
            if (pipe_check(argv))
            {
                printf("\n");

                Multiple_pipe(argv);
            }
        }
    }
}
```

```
void Parsing_assignments(char **argv)
{
    int i = 0;
    char var[100] = {0};
    int j = 1;
    int s = 0;

    char str[100] = {0};

    while (argv[i] != NULL)
    {

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


            int f = i + 1;
            while (argv[f] != NULL)
            {
                strcat(str, argv[f]);
                strcat(str, " ");


                f++;
            }

            setenv(argv[i - 1], str, 1);
            break;
        }
        i++;
    }

}
```

```
bool equality(char *str)
{
    int i = 0;

    while (str[i] != 0)
    {
        if (str[i] == '=')
            return true;

        i++;
    }
    return false;
}

int parsing_equality(char* cmd,char **argv)
{


    char *token;
    token = strtok(cmd, "=");
    int i = 0;
    while (token != NULL)
    {
        argv[i] = token;
        token = strtok(NULL, "=");
        i++;
    }
    argv[i] = NULL;
    return i;
}
void appending(char*str,char **argv)
{


        int i = 0;
        while (argv[i] != NULL)
        {

            strcat(str, argv[i]);
            strcat(str, " ");

            i++;

        }


}
```