The Department of Electrical and Computer Engineering

The University of Alabama in Huntsville

CPE 435 Lab-03

Build Your Own Linux Shell

Introduction

Shell

The shell or command line interpreter is the fundamental user interface to an operating system, each interactive user can send commands to the OS and by which the OS can respond to the user. The command line is a sequence of ASCII text words delimited by whitespace. The first word in the command line is either the name of a built-in command or the pathname of an executable file. The remaining words are command line arguments. If the first word is a built-in command, the shell immediately executes the command in the current process. Otherwise, the word is assumed to be the pathname of an executable program. In this case, the shell forks a child process, then loads and runs the program in the context of the child.

Exercise

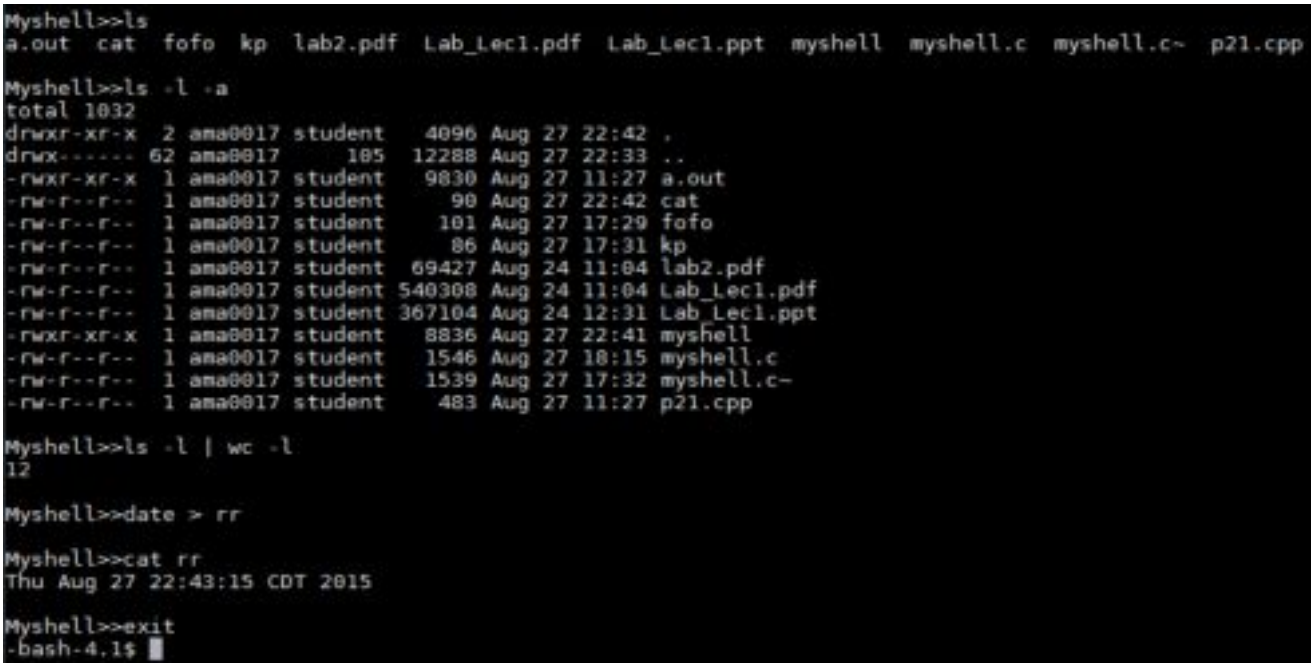
You will create your own linux shell by writing a c program. Please go through the demo codes provided and try to understand the functions mentioned in the hint section.

Your new shell should support following commands and similar commands:

user commands, such as ls ,date,ls –l –a
commands with I/O re-direction ,ex : ls –l > a.txt
commands with a single pipe ,ex : who | wc –l
command with piping and redirection; ex: ls -l | sort > b.txt

Like all Linux shells, your shell executes a loop. It prints the shell prompt, reads the command line (terminated with NULL), parses the command line and creates its arguments, executes the command with its arguments, then waits until the command finishes. It should be a child process which runs a command.

Following image shows an example solution in action where ls command is operated first, and others subsequently until the entered command is exit.



Repeat the loop until exit command is entered

Hint

char *strtok(char *str, const char *delim);

The strtok() function parses a string into a sequence of tokens. On the first call to strtok() the string to be parsed should be specified in str. In each subsequent call that should parse the same string, str should be NULL.The delim argument specifies a set of bytes that delimit the tokens in the parsed string. The caller may specify different strings in delim in successive calls that parse the same string.

int dup2(int oldfd, int newfd);

dup2() makes newfd be the copy of oldfd, closing newfd first if necessary, but note the following: If oldfd is not a valid file descriptor, then the call fails, and newfd is not closed. If oldfd is a valid file descriptor, and newfd has the same value as oldfd, then dup2() does nothing, and returns newfd.

char *gets(char *s) :

Reads a line from stdin into the buffer pointed to by s until either a terminating newline or EOF, which it replaces with a null byte

int pipe(int pipefd[2]);

pipe () creates a pipe, a unidirectional data channel that can be used for interprocess communication. The array pipefd is used to return two file descriptors referring to the ends of the pipe. pipefd[0] refers to the read end of the pipe. pipefd[1]refers to the write end of the pipe. Data written to the write end of the pipe is buffered by the kernel until it is read from the read end of the pipe.

Deliverables

Lab Report

The following material in each section is expected:

1. Cover page with your name, lab number, course name, and dates
2. Theory/Background (Material or methods relevant to the lab, a few sentences on each)
    a. shells
    b. strtok() function
    c. dup() and dup2()
    d. pipe ()
    e. execvp ()
3. Observations (Show output demonstrating the shell works as intended for the following)
    a. user commands, such as ls ,date,ls –l –a
    b. commands with I/O re-direction ,ex : ls –l > a.txt
    c. commands with a single pipe ,ex : who | wc –l
    d. command with piping and redirection; ex: ls -l | sort > b.txt
4. Conclusion (Did your program work as expected, what can you take away from the lab?)
5. Appendix (for source code, submit the text in a table)

The report should be submitted as a single pdf document with the source code for your program within it.

Recorded Demonstration

The following material in each section is expected:

1. Introduce yourself and give the name of the lab
2. Show and discuss how your program works
3. Compile the program
4. Show that the shell can take:
    a. user commands, such as ls ,date,ls –l –a
    b. commands with I/O re-direction ,ex : ls –l > a.txt
    c. commands with a single pipe ,ex : who | wc –l
    d. command with piping and redirection; ex: ls -l | sort > b.txt
5. Recordings should be around 5 to 7 minutes on average

The recorded demonstration should be submitted as an mp4 file. You may use Zoom or any other software you are familiar with to record it, though most of the recording will be focused on recording the screen, discussing the program, and showing the desired functionality.

Demo Codes

Demo Code 1

```
/*
Written By: Prawar Poudel

        This program is intended to showcase the use of strtok() function
        Please study about the strtok function first and compare the three outputs that you will receive here
*/
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

int main(int argc,char* argv[])
{

        printf("Demo Number 1\n");
        printf("================\n");
        char myString[100] = "i,want to break,this string using, both,comma and space";

        //following is the temporary string that I want to keep my char[] read from breaking the above char[]
myString
        //following breaks based on space character
        char *temp;
        temp = strtok(myString," "); //include <space> inside ""
        while(temp!=NULL)
        {
                printf("%s\n",temp);
                temp = strtok(NULL," "); //include <space> inside ""
        }


        //following breaks based on comma character
        printf("\n\nDemo Number 2\n");
        printf("================\n");
        strcpy(myString,"i,want to break,this string using, both,comma and space");
        temp = strtok(myString,","); //include , inside ""
        while(temp!=NULL)
        {
                printf("%s\n",temp);
                temp = strtok(NULL,","); //include , inside ""
        }

        //following breaks based on space or comma character
        printf("\n\nDemo Number 3\n");
        printf("================\n");
```

```
        strcpy(myString,"i,want to break,this string using, both,comma and space");
        temp = strtok(myString,", "); //include both space and , inside "" while(temp!=NULL)
        {
                printf("%s\n",temp);
                temp = strtok(NULL,", "); //include both space and , inside ""
        }
        return 0;
}
```

Demo Code 2

```
/*
Written By: Prawar Poudel

        This program is supposed to demonstrate the execution of dup2() function

        Please read the manual page first before jumping to run this program */

#include <stdio.h>
#include <unistd.h>
#include <sys/wait.h>
#include <stdlib.h>
#include <fcntl.h>

int main()
{
        printf("You would expect this to go to your stdout, and it does\n");

        //we will create a file using open function
        char myFileName[] = "test.txt";
        //lets open the file by the name of test.txt
        int myDescriptor = open(myFileName,O_CREAT|O_RDWR|O_TRUNC,0644);

        int id;

        //creating a child that redirects the stdout to test.txt
        // you can use similar functionality for '>' operator
        if((id=fork())==0)
        {
                //lets call dup2 so that out stdout (second argument) is now copied to (points to) test.txt (first argument)
                // what this essentially means is that anything that you send to stdout will be sent to myDescriptor
                dup2(myDescriptor,1); //1 is stdout, 0 is stdin and 2 is stderr
                printf("You would expect this to go to your stdout, but since we called dup2, this will go to test.txt");

                close(myDescriptor);
                exit(0);
        }else
```

```
        wait(0);

    printf("This is also printed to the console\n");
    return 0;
}
```

Demo Code 3

```
/*
Written By: Prawar Poudel
    This program demonstrates the use of pipe() function in C
    Please man pipe and have understanding before going through this code

    Pipe passes information from one process to another, similar to water-pipes there is a read-end
    and a write-end of pipe
*/
#include <stdio.h>
#include <unistd.h>
#include <stdlib.h>
#include <fcntl.h>
#include <sys/wait.h>

int main()
{
    int myPipingDescriptors[2];
    if(pipe(myPipingDescriptors)==-1)
    {
        printf("Error in calling the piping function\n");
        exit(0);
    }

    //at this point two pipe ends are created
    // one is the read end and other is write end
    // [0] will be the read end, [1] will be the write end


    //now lets fork two process where one will make use of the read end and other will make
    // use of write end
    // they can communicate this way through the pipe
    int id;
    if((id=fork())==0)
    {
        dup2(myPipingDescriptors[1],1); //second argument 1 is stdout
        close(myPipingDescriptors[0]); //read end is unused to lets close it

        //this following statement will not be printed since we have copied the stdout to write end of pipe
```

```
            printf("I am child, and sending this message.\n");
            exit(0);
        }else if (id>0)
        {
            wait(0);
            char myRead[100];

            //basically what's written to the write-end of pipe stays there until we read the read-end of pipe
            read(myPipingDescriptors[0],myRead,37);
            printf("I am parent. I read following statement\n\t%s\n",myRead);
            close(myPipingDescriptors[1]);

        }else
        {
            printf("Failed to fork so terminating the process\n");
            exit(-1);
        }
        close(myPipingDescriptors[0]);
        close(myPipingDescriptors[1]);
        return 0;
}
```

Demo Code 4

```
/*
Written By: Prawar Poudel

        execvp runs a program that you pass as argument
        Please study about the execvp function before going to run this program

        After you understand the things, please run and watch them
        Please make sure that execvp has the right executable provided
*/
#include <unistd.h>
#include <string.h>
#include <stdio.h>
#include <sys/wait.h>

int main()
{
        char myArgument[100];

        //you can change the content of myArgument using any user typed input using gets()
```

```
        strcpy(myArgument,"ls -a");

        //execvp expects the arguments to be provided as char[][]
        //so please make sure you understand strtok before coming here
        //we will use strtok() to break the sequence of command and argument in myArgument to convert to char[][]
        char* myBrokenArgs[10]; //this will hold the values after we tokenize
        printf("Starting tokenization...\n");
        myBrokenArgs[0] = strtok(myArgument," ");
        int counter = 0;
        while(myBrokenArgs[counter]!=NULL)
        {
                counter+=1;
                myBrokenArgs[counter] = strtok(NULL," ");
        }
        myBrokenArgs[counter] = NULL;
        printf("\ttokenization complete....\n\nNow executing using execvp\n");

        printf("Following will be the output of execvp\n");
        printf("=======================================\n");
        int id;

        //I will spawn a child that will run my execvp command
        if((id=fork())==0)
                execvp(myBrokenArgs[0],myBrokenArgs);
        else if(id<0)
                printf("Failed to make child...\n");
        else
        {
                //parent shall wait until the child is killed
                wait(0);
                printf("=======================================\n");
                printf("Completed execution\n");
        }

        return 0;
}
```