

Proiect

Sisteme de operare

Terminal și implementare de comenzi

OCTAVIAN BODNARIU
OVIDIU ANDRĂȘESC
VASILE ANIȘORAC
FACULTATEA DE MATEMATICĂ ȘI INFORMATICĂ
UNIVERSIATEA DE VEST DIN TIMIȘOARA

1 Abstract:

Lucrarea de față își propune descrierea modului de realizare a unei linii de comandă simple Linux precum și implementarea unui set de comenzi interne.

2 Introducere

O linie de comanda este un program sau o secvență de instrucțiuni ce rulează dintr-un Shell sau dintr-o altă linie de comandă interfață, pe un monitor de calculator, utilizatorii având posibilitatea introducerii de comenzi (instrucțiuni) specifice comunicării cu computerul într-un mod exclusiv textual de intrare/ieșire.

Avantajul folosirii unei linii de comanda în locul unei interfețe grafice (GUI) constă tocmai în controlul pe care acesta îl ofera, având mai multe comenzi disponibile care pot fi combinate prin pipes pentru a obține comenzi sau sarcini mult mai specializate. În multe cazuri liniile de comandă sunt mai rapide și mai fiabile decât interfețele lor grafice iar uneori acestea sunt singura opțiune disponibilă în cazul sistemelor deteriorate care prezintă o funcționalitate minimală. O lista a comenzilor disponibile pentru proiectele Linux poate fi consultă la pagina [Index of Commands for Unix-like Operating Systems](#).

3 Ce este Shell-ul?

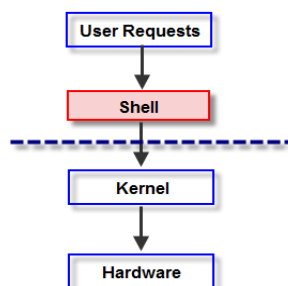
Un program **Shell**, poate fi comparat cu un interpretator de comenzi text care oferă interfeței liniei de comandă din Linux sau altor sisteme de operare Unix, posibilitatea de a citii comenzile scrise într-o consolă sau fereastra terminal printr-o interfață grafică și apoi sa le execute. Numele de Shell provine de la faptul ca acest program este o interfață externă a sistemului de operare în relația cu utilizatorul. Deci conectarea la sistemul de operare se face printr-un shell acesta fiind modul primar prin care utilizatorul interacționează cu computerul însă fără ca acesta să vedă detaliile de funcționare a proceselor sistemului.

Deși sunt de dimensiuni mici **Shell-urile** sunt programe sofisticate și puternice utilizate pentru execuția programelor (lansarea lor), pentru substituirea unor variabile și a numelor de fișiere de intrare/ieșire, pentru redirecționare a programelor (trimiterea output-ului unui program ca input pentru un altul) și servind drept mediu de control pentru schimbarea shell-ului cu un altul și ca limbaj de programare (limbaj ce poate fi folosit pentru scrierea de scripturi shell).

Un **shell prompt** este un caracter sau un set de caractere introduse la inceputul liniei de comandă care indică că Shell-ul este prompt pentru a primi comenzi, de obicei este (\$) pentru utilizatorii obișnuiți și (#) pentru administrator sau rădăcina de utilizator (root).

Sistemele de operare Unix au dezvoltat modele diferite de *shell*-uri cu multe similitudini dar și cu diferențe în cea ce privește comenzile sintaxa și funcțiile, fiecare sistem având cel puțin un shell.

sh (Bourne Shell) este shellul original pentru Unix, scris de Stephen Bourne în laboratoarele Bell Labs în 1974 este inca folosit pe scară largă deși are o dimensiune mică și doar caterva caracteristici. Astfel un proces este o instanță a unui program aflat în execuție. Flexibilitatea shell-ului este însoțită și de faptul ca acesta poate fi schimbat ușor cu un alt shell curent Bash (Bourne-shell din nou) este shell-ul implicit pentru Linux, rulând pe orice alt sistem Unix, versiuni ale acestuia aflându-se și pe alte sisteme de operare inclusiv o variantă și pe sistemele Windows ale Microsoft. Bash-ul este un super set de sh-uri abând însă mai multe comenzi și fiind mult mai intuitiv și flexibil, acesta a fost scris de către Brian Fox și Chet Ramey pentru a face Unixul compatibil și pentru a deveni o platformă open source.

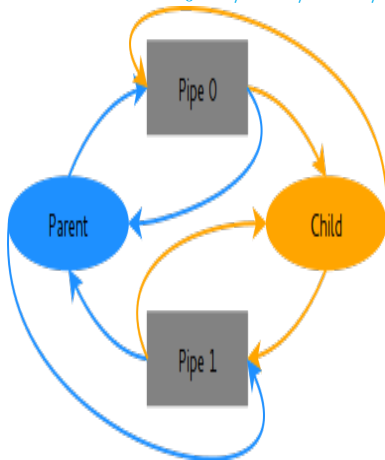


Printre alte shell-uri mai cunoscute putem aminti ash (scris de Kenneth Almquist in 1989) este o clona a sh având cerințe de memorie mici fiind utilizat pentru sisteme incorporate in alte produse, disponibil pe majoritatea sistemelor Unix de acest fel. Un alt shell cunoscut este si csh (C shell 1978) preferat de programatori datorită faptului ca folosește limbajul C.

4 Ce este un Pipe?

Un *pipe* este o formă de transfer sau de redirecționare folosită în Linux și alte sisteme de operare Unix pentru a trimite output-ul unui program pentru a fi procesat mai departe de un altul printr-un protocol standard output (stdout). Comenzile pipeline sunt conexiuni temporare între două sau mai multe programe simple. Această conexiune face posibilă efectuarea de sarcini de un înalt nivel de specializare realizate simultan și în mod continuu. Alături de expresiile regulate și de sistemul de ierarhizare al fișierelor, pipe-urile sunt una dintre cele mai importante și puternice caracteristici ale sistemelor de operare Unix.

Pipe-urile sunt canale de comunicație unidirecționale ce permit transmiterea de fluxuri de octeți între procese (dacă se dorește realizarea unei comunicații bidirecționale este nevoie de două pipe-uri). Citirea și scrierea în pipe se face conform algoritmului FIFO (primul intrat, primul ieșit). Pipe-urile sunt de fapt fișiere, dar care sunt gestionate în mod diferit decât fișierele obișnuite. Pentru construirea unui pipe se folosește funcția de sistem `pipe()` (pipe-urile construite cu funcția `pipe()` se mai numesc și pipe-uri fără nume). Funcția `pipe()` returnează o pereche de descriptori. Unul dintre aceștia este conectat la capătul de scriere al pipe-ului iar celălalt este conectat la capătul de citire al pipe-ului. Funcția `pipe()` primește ca argument un vector de două elemente de tip `int`. Dacă pipe-ul a fost construit cu succes atunci funcția `pipe` returnează în primul element din vector (`pfd[0]`) descriptorul de fișier pentru citire și în cel de al doilea element al vectorului (`pfd[1]`) descriptorul de fișier pentru scriere. Astfel scrierea și citire din pipe se face în același mod în care se face scrierea și citirea din și în fișiere obișnuite. control.aut.utcluj.ro/iatr/lab5/pipe.htm



5 Codul comentat

Comenzile implementate de noi sunt: sort, yes, ls, help, cal, echo, rename, fiecare având descrierea oficială de tip manual in comanda Help.

Codul a fost explicat prin comentariile oferite.

Comenzile Shell-ului vor explica structura și modul în care proiectul a fost realizat de către membrii săi.

```
#include <sys/wait.h> // waitpid()
#include <unistd.h> // fork(), exec(), pid
#include <stdio.h>
#include <stdlib.h> // exit(), execvp()
#include <string.h> //Pentru strtok()
#include <time.h> //Pentru randomize
#include <fcntl.h>
#include <errno.h>
#include <dirent.h>

#include "my_sort_Octavian.h"
#include "my_yes_Octavian.h"
#include "my_outside_Octavian.h"
#include "my_ls.h" //Vasile
#include "my_help.h" //Vasile
#include "my_cal.h" //Ovidiu
#include "my_echo.h" //Ovidiu
//#include "my_rename.h"//Ovidiu
#include "loh.h"

#define BUFFSIZE 64
#define SEPARATOR " \t\r\n\a"
#define ALOC_ERR fprintf(stderr,"Allocation error !")

/*
Basic loop :
    1.Citim comanda din stdin
    2.Parsare, separam stringul in comanda si argumente
    3.Executam comanda parsata
*/

/*
Info:
Autor : Octavian Bodnariu
Implementari: Shell, yes, sort, outside, ajustare cod
Autor : Vasile Anisorac
Implementari: ls, help + documentatie
Autor : Ovidiu Andrasesc
Implementari: cal, rename, locate
Versiune 1.4.4
*/
```

```

int my_rename(char** argumente)
{

}

int my_exit(char** argumente)
{
return 0;
}

int my_version(char** argumente)
{
return 1;
}

//Functii specifice shell-ului
char *citireLinie();
char *parsareLinie(char *linie);
int lansare(char **argumente);
int executa(char **argumente);
void buclaPrincipala();

//Lista de comenzi(String)
char *comenzi[] = {
    "my_help",
    "my_Version",
    "my_yes",
    "my_sort",
    "my_cal",
    "my_ls",
    "my_echo",
    "my_rename",
    "outside",
    "exit"
};

//Lista in care vor intra functiile cu comenzile construite
int (*comenzi_construite[])(char**) = {
    &my_help,
    &my_version,
    &my_yes,
    &my_sort,
    &my_cal,
    &my_ls,
    &my_echo,
    &my_rename,
    &my_outside,
    &my_exit
};

```

```

/**
@return : Numarul de comenzi pe care il stie terminalul
*/
int terminal_num_comenzi_construite()
{
    return sizeof(comenzi) / sizeof(char*);
}
/**
Main-ul
*/
int main(int argc, char* argv[])
{
    buclaPrincipala();
    return EXIT_SUCCESS;
}
/**
Citirea unei linii din terminal
*/
char *citireLinie()
{
    int bufsize = BUFFSIZE;
    int pozitie = 0;
    char *buffer = malloc(sizeof(char) * bufsize);
    int c;

    if(!buffer)
    {
        ALOC_ERR;
    }
    while(1)
    {
        c = getchar();

        if(c == EOF || c== '\n')
        {
            buffer[pozitie] = '\0';
            return buffer;
        }
        else
        {
            buffer[pozitie] = c;
        }
        pozitie++;

        if(pozitie >= bufsize)
        {
            bufsize += BUFFSIZE;
            buffer = realloc(buffer, bufsize);
        }
    }
}

```

```

        if(!buffer)
        {
            ALOC_ERR;
            exit(EXIT_FAILURE);
        }
    }
}

/**
@param : 0 linie care s-a citit din stdin
@return : Linia parsata cu strtok()
*/
char *parsareLinie(char *linie)
{
    int buffer = BUFFSIZE;
    int pozitie = 0;
    char **tokens = malloc(buffer * sizeof(char*));
    char *token;

    if(!tokens)
    {
        ALOC_ERR;
        exit(EXIT_FAILURE);
    }

    token = strtok(linie, SEPARATOR);
    while(token != NULL)
    {
        tokens[pozitie] = token;
        pozitie++;

        if(pozitie >= buffer)
        {
            buffer+= BUFFSIZE;
            tokens = realloc(tokens, BUFFSIZE * sizeof(char*));
            if(!tokens)
            {
                ALOC_ERR;
                exit(EXIT_FAILURE);
            }
        }
    }

    token = strtok(NULL, SEPARATOR);
}

/**

```

```

@param : Un string de tip array parsat
@return : Intotdeauna 1
*/
int lansare(char **argumente)
{
    pid_t pid;
    int status;

    pid = fork();

    if(pid == 0)
    {
        //Proces copil
        if(execvp(argumente[0], argumente) == -1)
        {
            perror("LNS");
        }
        exit(EXIT_FAILURE);
    }
    else if(pid < 0)
    {
        //EROARE LA FORK() !!!
        perror("LNS-fork");
    }
    else
    {
        do
        {
            waitpid(pid, &status, WUNTRACED);
        }while(!WIFEXITED(status) && !WIFSIGNALED(status));
    }

    return 1;
}
/**
@param :
@return : Adresa unei functii definite (poate fi 0 sau 1)
*/
int executa(char **argumente)
{
    int i;

    if(argumente[0] == NULL)
    {
        return 1;
    }
    for(i = 0; i < terminal_num_comenzi_construite(); i++)
    {

```



```

        if(strcmp(argumente[0], comenzi[i]) == 0)
            return (*comenzi_construite[i])(argumente);
    }
    return -1;
}
/**
Aceasta procedura grupeaza toate functiile scrise anterior si functioneaza
cat timp statusul este diferit de 0. Adica nu s-a activat comanda exit
*/
void buclaPrincipala()
{
    char *linie;
    char **argumente;
    int status;

    do
    {
        printf("%s$> ", "Octavian");
        linie = citireLinie();
        argumente = parsareLinie(linie);
        status = executa(argumente);

        if(status == -1)
        {
            printf("Comanda \'%s\' nu exista !\n", argumente[0]);
        }
        free(linie);
        free(argumente);
    }while(status);
}

```