Simple Shell

OS LAB 1

Ahmed Ibrahim Ali | 4449 | Dr. Hicham El-Mongui

Description

In general ,

the shell is a program that takes commands from the keyboard and gives them to the operating system to perform. In the old days, it was the only user interface available on a Unix-like system such as Linux. Nowadays, we have *graphical user interfaces (GUIs)* in addition to *command line interfaces (CLIs)* such as the shell.

IMPLEMENTATION "HOW IT WORKS ? "

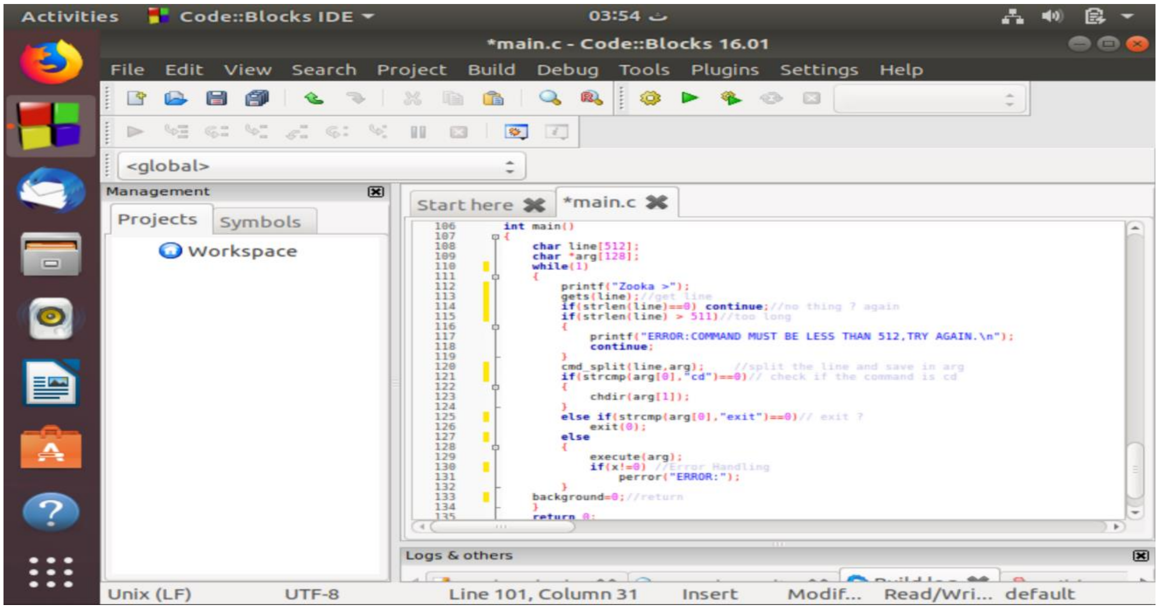
Our Simple Shell implementation is so simple to work

In this program there is four main functions which we can work through

Main Function

In main function we start program by getting the command to be executed from user and call other functions to make some operations on this command

<u>Snippet code</u>

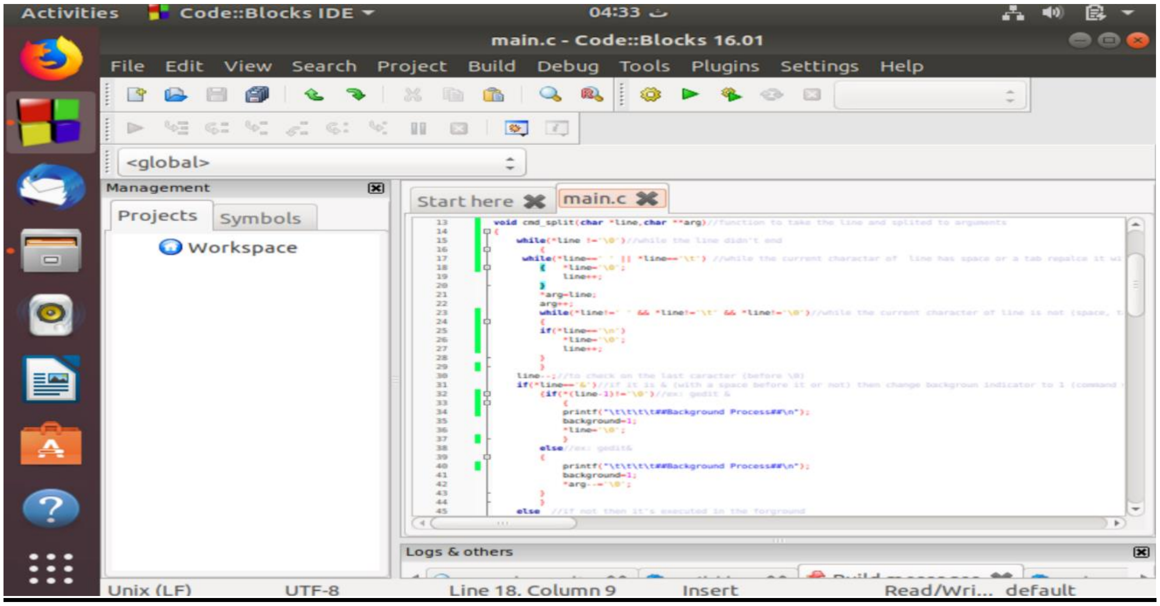


```c
int main()
{
    char line[512];
    char *arg[128];
    while(1)
    {
        printf("Zooka >");
        gets(line);//get line
        if(strlen(line)==0) continue;//no thing ? again
        if(strlen(line) > 511)//too long
        {
            printf("ERROR:COMMAND MUST BE LESS THAN 512,TRY AGAIN.\n");
            continue;
        }
        cmd_split(line,arg);     //split the line and save in arg
        if(strcmp(arg[0],"cd")==0)// check if the command is cd
        {
            chdir(arg[1]);
        }
        else if(strcmp(arg[0],"exit")==0)// exit ?
            exit(0);
        else
        {
            execute(arg);
            if(x!=0) //Error Handling
                perror("ERROR:");
        }
    background=0;//return
    }
    return 0;
```

Cmd_split

This is the most important function in this program which can take the line and split it into arguments which can be used in execution .

<u>Snippet code</u>

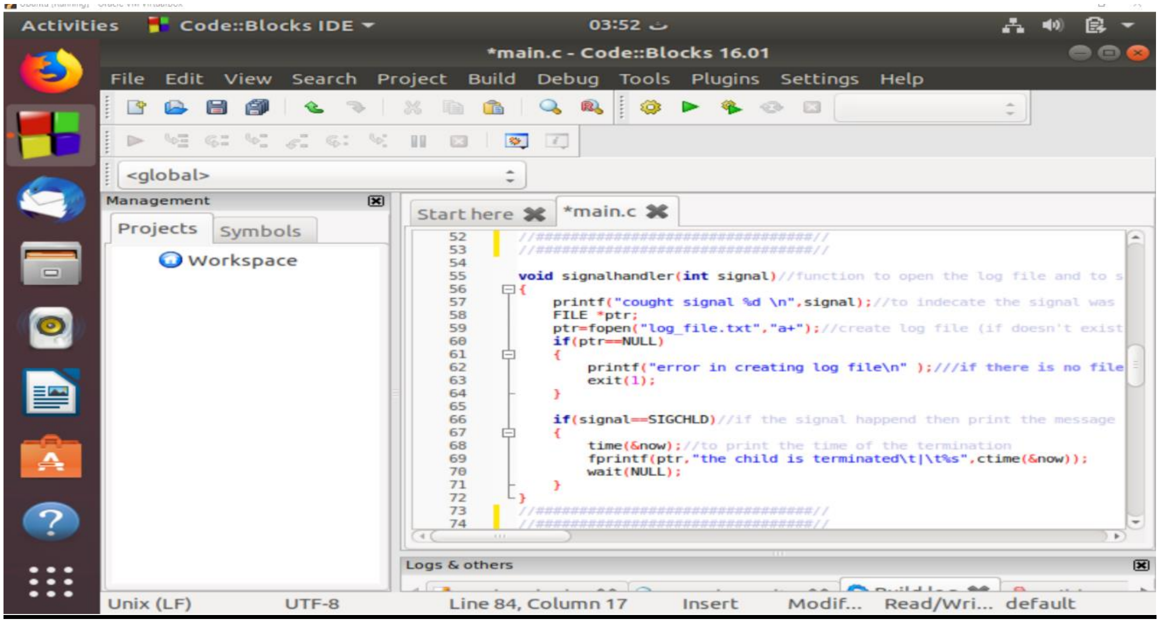


```
void cmd_split(char *line,char **arg)//function to take the line and splited to arguments
{
    while(*line !='\0')//while the line didn't end
        {
     while(*line==' ' || *line=='\t') //while the current charactar of  line has space or a tab repalce it wi
        {   *line='\0';
            line++;
        }
        *arg=line;
        arg++;
        while(*line!=' ' && *line!='\t' && *line!='\0')//while the current character of line is not (space, t
        {
        if(*line=='\n')
            *line='\0';
            line++;
        }
        }
    line--;//to check on the last caracter (before \0)
    if(*line=='&')//if it is & (with a space before it or not) then change backgroun indicator to 1 (command
        {if(*(line-1)!='\0')//ex: gedit &
            {
            printf("\t\t\t\t##Background Process##\n");
            background=1;
            *line='\0';
            }
        else//ex: gedit&
        {
            printf("\t\t\t\t##Background Process##\n");
            background=1;
            *arg--='\0';
        }
        }
    else  //if not then it's executed in the forground
```

Signalhandler

It’s just a logger function which we can use to open the log file and write the log signal on .

<u>Snippet code</u>

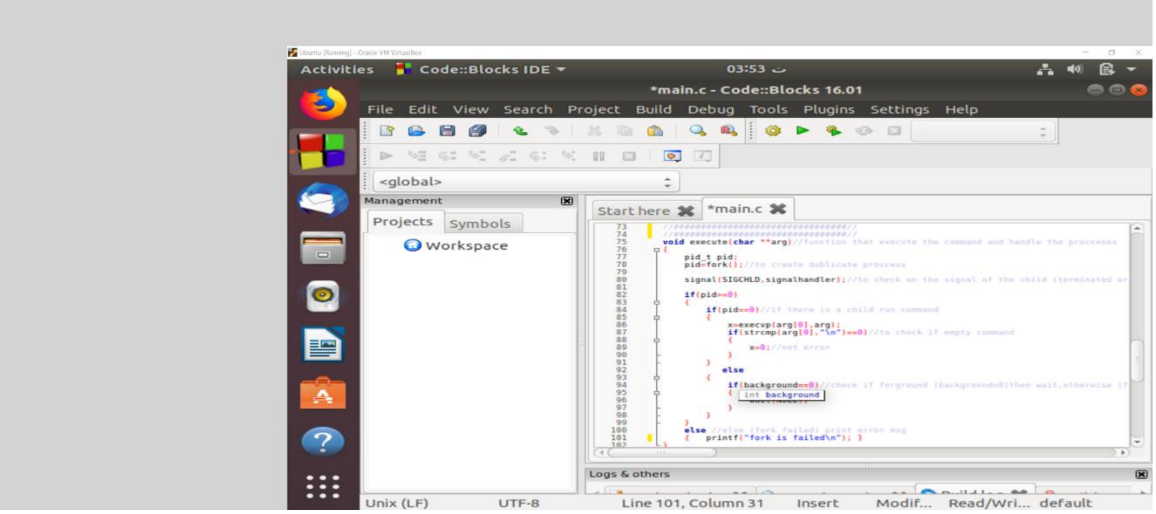


```
//##################################//
//##################################//

void signalhandler(int signal)//function to open the log file and to s
{
    printf("cought signal %d \n",signal);//to indecate the signal was
    FILE *ptr;
    ptr=fopen("log_file.txt","a+");//create log file (if doesn't exist
    if(ptr==NULL)
    {
        printf("error in creating log file\n" );///if there is no file
        exit(1);
    }

    if(signal==SIGCHLD)//if the signal happend then print the message
    {
        time(&now);//to print the time of the termination
        fprintf(ptr,"the child is terminated\t|\t%s",ctime(&now));
        wait(NULL);
    }
}
//##################################//
//##################################//
```

execute

The function that is used to execute the commands and handle the processes .

Snippet code



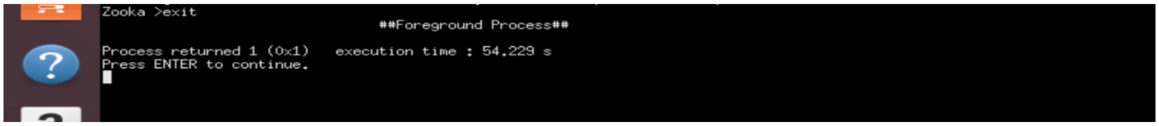
```
//##################################//
//##################################//
void execute(char **arg)//function that execute the command and handle the procceses
{
    pid_t pid;
    pid=fork();//to create dublicate proccess

    signal(SIGCHLD,signalhandler);//to check on the signal of the child (terminated or

    if(pid>=0)
    {
        if(pid==0)//if there is a child run command
        {
            x=execvp(arg[0],arg);
            if(strcmp(arg[0],"\n")==0)//to check if empty command
            {
                x=0;//not error
            }
        }
            else
        {
            if(background==0)//check if forground (background=0)then wait,otherwise if
            {
                wait(NULL);
            }
        }
    }
    else //else (fork failed) print error msg
    {   printf("fork is failed\n"); }
}
```

TEST CASES

1. The internal shell command "exit" which terminates the shell

- Concepts: shell commands, exiting the shell.
- System calls: exit()

Output :

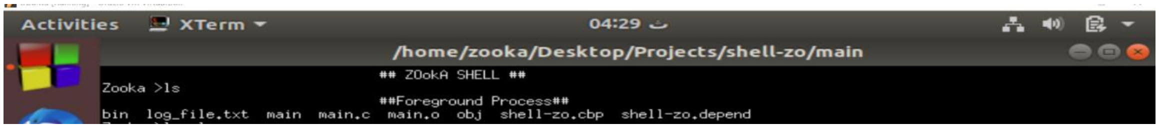


```
Zooka >exit
                                ##Foreground Process##

Process returned 1 (0x1)   execution time : 54.229 s
Press ENTER to continue.
```

2. A command with no arguments

- Example: ls, cp, rm …etc
- Details: Your shell must block until the command completes and, if the return code is abnormal, print out a message to that effect.
- Concepts: Forking a child process, waiting for it to complete and synchronous execution.
- System calls: fork(), execvp(), exit(), wait()

Output :

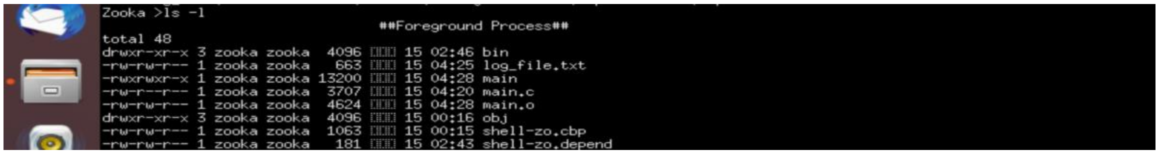


```
/home/zooka/Desktop/Projects/shell-zo/main
                                ## ZOokA SHELL ##
Zooka >ls
                                ##Foreground Process##
bin  log_file.txt  main  main.c  main.o  obj  shell-zo.cbp  shell-zo.depend
```

3. A command with arguments

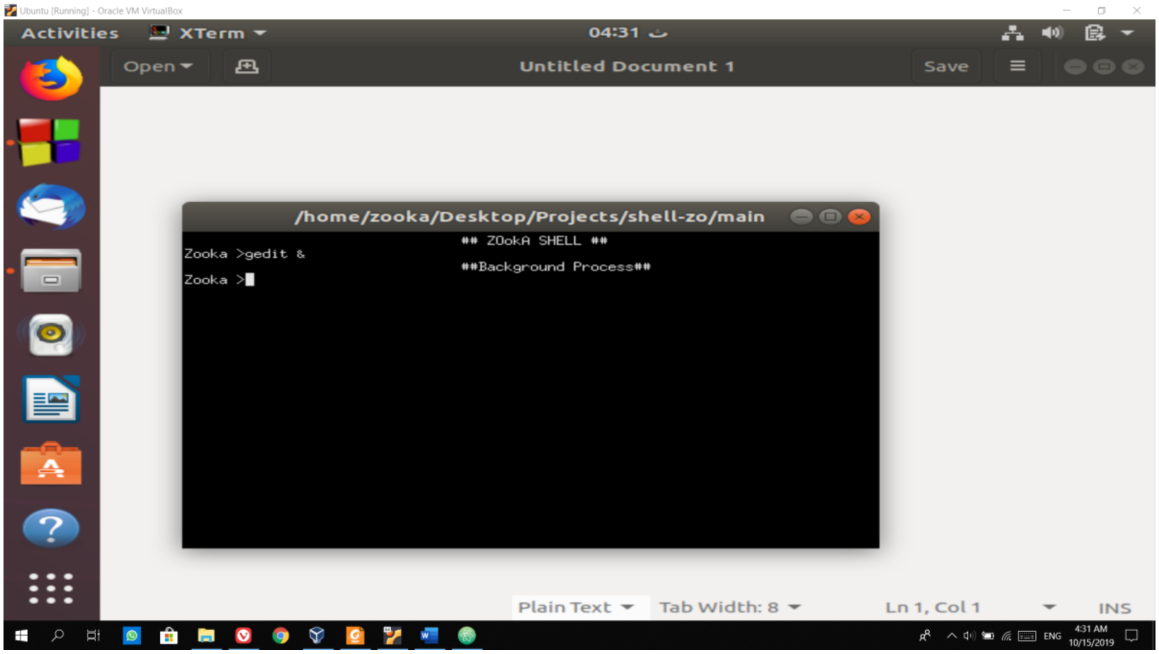
- Example: ls –l
- Details: Argument 0 is the name of the command.
- Concepts: Command-line parameters.

Output :



```
Zooka >ls -l
                                ##Foreground Process##
total 48
drwxr-xr-x 3 zooka zooka  4096 □□□ 15 02:46 bin
-rw-rw-r-- 1 zooka zooka   663 □□□ 15 04:25 log_file.txt
-rwxrwxr-x 1 zooka zooka 13200 □□□ 15 04:28 main
-rw-r--r-- 1 zooka zooka  3707 □□□ 15 04:20 main.c
-rw-rw-r-- 1 zooka zooka  4624 □□□ 15 04:28 main.o
drwxr-xr-x 3 zooka zooka  4096 □□□ 15 00:16 obj
-rw-rw-r-- 1 zooka zooka  1063 □□□ 15 00:15 shell-zo.cbp
-rw-rw-r-- 1 zooka zooka   181 □□□ 15 02:43 shell-zo.depend
```

4. **A command, with or without arguments, executed in the background using &.**
    a. Example: firefox &
    b. Details: In this case, your shell must execute the command and return immediately, not blocking until the command finishes.



```
                                     ## ZOokA SHELL ##
Zooka >gedit &
                                     ##Background Process##
Zooka >
```