
Simple Linux Shell

Name	Abdelrahman Ibrahim Gaber
ID	19015881
Department	Electronics and communication

1 Overall organization of the code

This lab mainly talk about implementation of simple Linux shell that execute any commands entered or commands in the built in shell we can shorten the program in this steps :

1. Setup Environment : to determine the directory that programm began on it
2. super loop that take the input from the user and check it if the input wrong or exit command to terminate the program or a real command from the user
3. The commands divides into two types :
 - Built in commands : this part include implementation of special kind of commands using system calls like "getenv" and "setenv" that handle with that commands like "export" that store a value whether it's a number or string to a certain variable , "echo" that do the functionality of printf in C programming language , "pwd" that print the current directory and "cd" that change the directory as the user like.
 - Normal commands : A child process must be forked and get the id if the id = 0 so we are in child process and we can execute the commands using system call "execvp" , if the id != 0 and return the child process id so we are now in parent process and there is two options:
 - foreground process : the parent should wait the child using waitpid
 - background process : the process run in the background
4. register Child signal and avoid zombies processes and write in log.text file using handler function if the child process terminated successfully

2 Major Functions

1. Get input : to get the command from the user

```
void Get_Input(void){
    char arr[100];
    printf("%s shell >> ",getcwd(arr , 100));
    scanf("%[^\\n]%*c", input);
}
```

2. Pares Input : to pares the input and put it in a global array of strings

```
void Parse_Input(void){
    char*token = strtok(input , " ");
    if(strcmp(token , "export") == 0){
        Clean_Export(token);
        exportFlag = 1;
    }
    else{
        if(strcmp(token , "cd") == 0) cdFlag = 1;
        if(strcmp(token , "echo") == 0) echoFlag = 1;
        if(strcmp(token , "pwd") == 0) pwdFlag = 1;
        if(strcmp(token , "exit") == 0) exitFlag = 1;
        while (token != NULL)
        {
            paresedInput[counter] = token;
            token = strtok(NULL," ");
            counter++;
        }
        paresedInput[counter] = '\0' ;
        backGroundIndex = counter - 1;
        counter = 0;
    }
}
```

2.1 Clean Export : to pares the input in case of export command

```
void Clean_Export(char * token){
    while(token != NULL){
        paresedInput[counter] = token;
        token = strtok(NULL,"=");
        counter++;
    }
    paresedInput[counter] = '\0';
    counter = 0;
}
```

3. Execute Shell built in : (export,echo,pwd and cd)

```
void Excute_Shell_Built_In(void){
    if(cdFlag){
        Excute_CD();
    }
    else if(exportFlag){
        Excute_Export();
    }
    else if(echoFlag){
        Excute_Echo();
    }
    else if(pwdFlag){
        printf("%s\n",getcwd(NULL,0));
    }
}
```

3.1 Execute CD function :

```
void Excute_CD(void){
    if((paresedInput[1] == NULL) || ((strcmp(paresedInput[1],"~")==0))){
        chdir(getenv("HOME"));
    }
    else{
        int flag = 0;
        flag = chdir(paresedInput[1]) ;
        if( flag != 0){
            printf("Error, the directory is not found\n");
        }
    }
}
```

3.2 Execute Export function :

```
void Excute_Export(void){
    char* data = paresedInput[2];
```

```

/*check the qutation marks*/
if(data[0] == '"'){
    data++;
    data[strlen(data)-1] = '\\0';
    setenv(paresedInput[1] , data , 1);
}
else{
    /*No qutation mark*/
    setenv(paresedInput[1] , paresedInput[2] , 1);
}
}

```

3.3 Execute Echo function :

```

void Excute_Echo(void){
char*echoEnv = paresedInput[1];
if(paresedInput[2] == NULL){
    /*there is only one command in echo*/
    /*there is two cases print variable or sentence*/

    /*remove the qoutation*/
    echoEnv++;
    echoEnv[strlen(echoEnv) - 1] = '\\0';
    /*case 1 print variable*/
    if(echoEnv[0] == '$'){
        /*skip dollar sign*/
        echoEnv++;
        printf("%s\\n",getenv(echoEnv));
    }
    else{
        /*case 2 print sentence*/
        printf("%s\\n",echoEnv);
    }
}
else{
    char*temp = paresedInput[2];

```

```

    /*there is more than input*/
    /*remove the first qutation*/
    echoEnv++;
    if(echoEnv[0] == '$'){
        echoEnv++;
        printf("%s ",getenv(echoEnv));
        /*remove the last qutation*/
        temp[strlen(temp)-1] = '\0';
        printf("%s\n",temp);
    }
    else{
        printf("%s ",echoEnv);
        /*skip $*/
        temp++;
        /*remove the last qoutation*/
        temp[strlen(temp)-1] = '\0';
        printf("%s\n",getenv(temp));
    }
}
}

```

4. Execute command :

```

void Execute_Command (void){
    int status , foregroundId;
    int errorCommand = 1;
    int child_id = fork();
    if(child_id == -1){
        printf("System Error!\n");
        exit(EXIT_FAILURE);
    }
    else if (child_id == 0){
        if(paresedInput[1] == NULL){
            /*command consist of one word*/
            errorCommand = execvp(paresedInput[0] , paresedInput);
        }
    }
}

```

```

else if(paresedInput[1] != NULL){
    /*more than one word*/
    /*check if there is a variable in system environment or not*/
    char* env = paresedInput[1];
    if(env[0] == '$'){
        int i = 1;
        char*envTemp;
        env++;
        envTemp = getenv(env);
        char *exportTemp = strtok(envTemp , " ");
        while(exportTemp != NULL){
            paresedInput[i++] = exportTemp;
            exportTemp = strtok(NULL, " ");
        }
    }
    errorCommand = execvp(paresedInput[0] , paresedInput);
}
if(errorCommand){
    printf("Error ! unknown command\n");
    exit(EXIT_FAILURE);
}
}
else{
    /*parent process*/
    /*foreground and background*/
    if(strcmp(paresedInput[backGroundIndex] , "&")==0){
        /*we are in the backGround*/
        /*no wait*/
        return;
    }
    else{
        foregroundId = waitpid(child_id , &status , 0);
        if(foregroundId == -1){
            perror("Error in waitpad function\n");
        }
    }
}

```

```

        return;
    }
    if(errorCommand){
        FILE * file = fopen("log.text" , Append_To_File);
        fprintf(file , "%s" , "Child process terminated\n");
        fclose(file);
    }
}
}
}
}

```

5. Reap Child Zombie : to avoid zombie process and write in log file

```

void Reap_Child_Zombie(void){
    int status;
    pid_t id = wait(&status);
    /*avoid zombie process*/
    if(id == 0 || id == -1){
        return;
    }
    else{
        Write_To_Log_File();
    }
}
}

```

5.1 write in log file function:

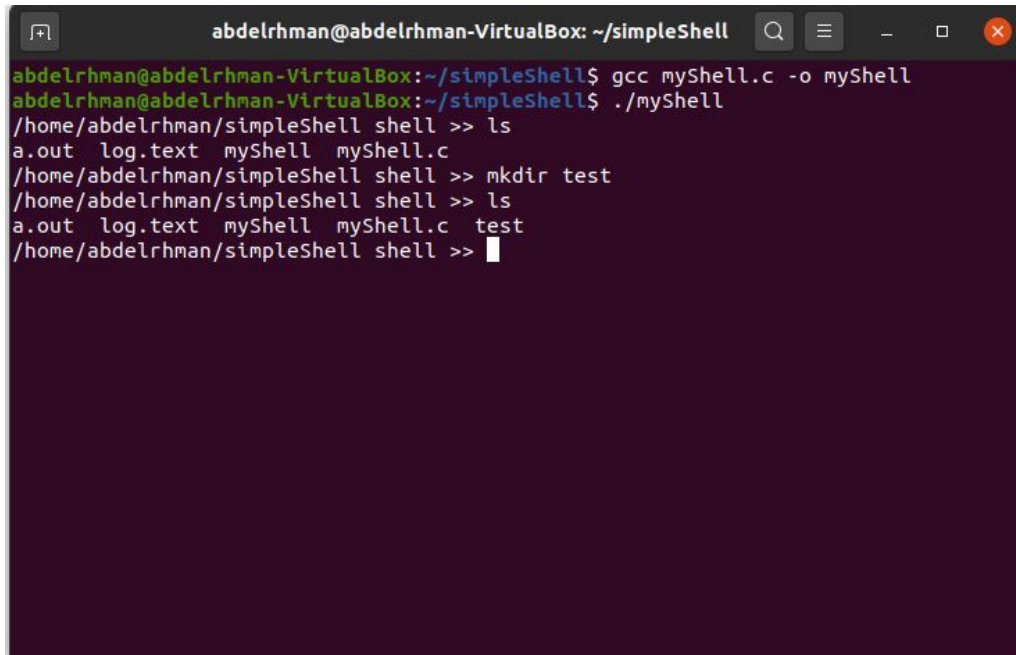
```

void Write_To_Log_File(void){
    FILE * file = fopen("log.text" , Append_To_File);
    if(file == NULL){
        printf("Error in file\n");
        exit(EXIT_FAILURE);
    }
    else{
        fprintf(file , "%s" , "Child process terminated\n");
        fclose(file);}}

```


3 Sample runs

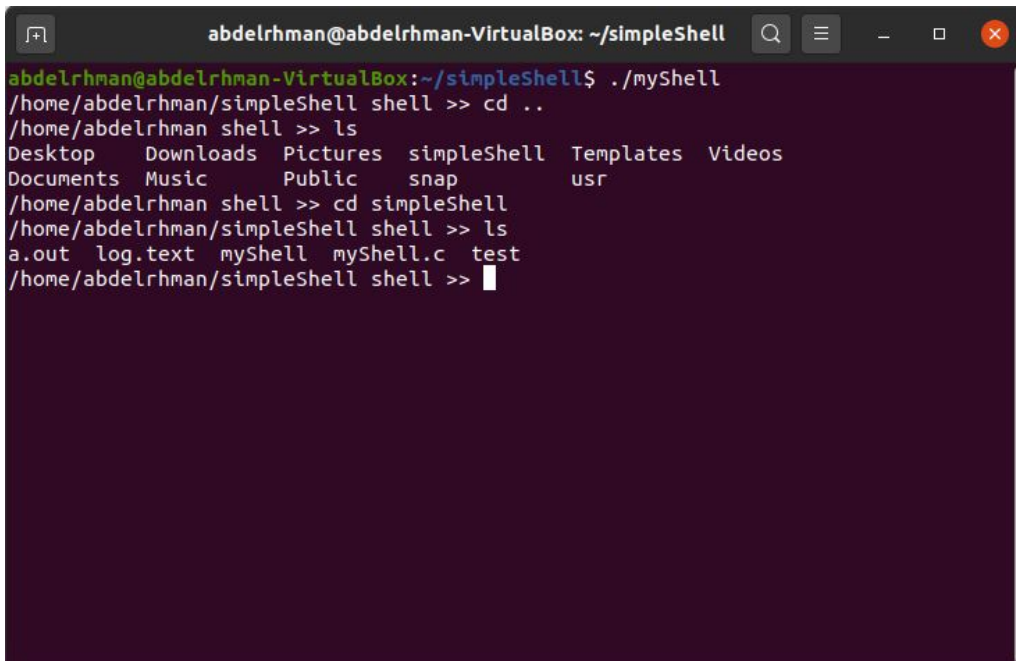
3.1 basic commands in shell



```
abdelrhman@abdelrhman-VirtualBox: ~/simpleShell
abdelrhman@abdelrhman-VirtualBox:~/simpleShell$ gcc myShell.c -o myShell
abdelrhman@abdelrhman-VirtualBox:~/simpleShell$ ./myShell
/home/abdelrhman/simpleShell shell >> ls
a.out log.text myShell myShell.c
/home/abdelrhman/simpleShell shell >> mkdir test
/home/abdelrhman/simpleShell shell >> ls
a.out log.text myShell myShell.c test
/home/abdelrhman/simpleShell shell >> 
```

Figure 1: ls and mkdir test cases

3.2 built in shell commands (cd , echo , export)



```
abdelrhman@abdelrhman-VirtualBox: ~/simpleShell
abdelrhman@abdelrhman-VirtualBox:~/simpleShell$ ./myShell
/home/abdelrhman/simpleShell shell >> cd ..
/home/abdelrhman shell >> ls
Desktop  Downloads  Pictures  simpleShell  Templates  Videos
Documents Music      Public    snap         usr
/home/abdelrhman shell >> cd simpleShell
/home/abdelrhman/simpleShell shell >> ls
a.out log.text myShell myShell.c test
/home/abdelrhman/simpleShell shell >> 
```

Figure 2: cd test case

3.3 export and echo testing

```
abdelrhman@abdelrhman-VirtualBox: ~/simpleShell
abdelrhman@abdelrhman-VirtualBox:~/simpleShell$ ./myShell
/home/abdelrhman/simpleShell shell >> export x="world"
/home/abdelrhman/simpleShell shell >> echo "Hello $x"
Hello world
/home/abdelrhman/simpleShell shell >> 
```

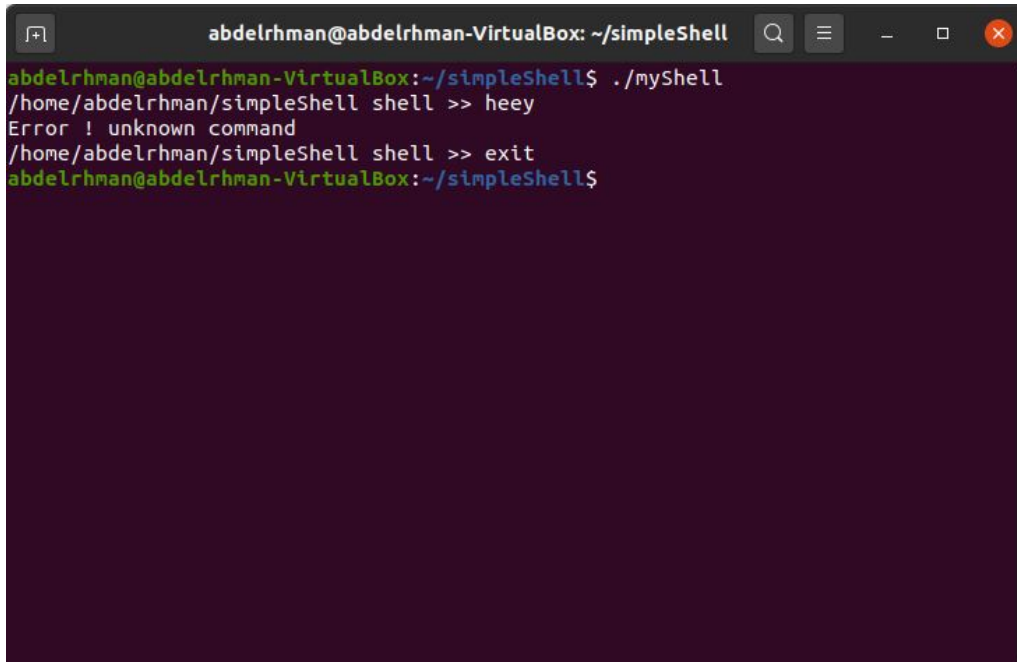
Figure 3: echo and export

3.4 general commands

```
abdelrhman@abdelrhman-VirtualBox: ~/simpleShell
abdelrhman@abdelrhman-VirtualBox:~/simpleShell$ ./myShell
/home/abdelrhman/simpleShell shell >> ls -a -l -h
total 64K
drwxr-xr-x  3 abdelrhman abdelrhman 4.0K 22:12 9  مار .
drwxr-xr-x 22 abdelrhman abdelrhman 4.0K 22:29 9  مار ..
-rwxr-xr-x  1 abdelrhman abdelrhman 18K 02:31 6  مار a.out
-rw-rw-r--  1 abdelrhman abdelrhman 375 22:29 9  مار log.text
-rwxrwxr-x  1 abdelrhman abdelrhman 18K 22:12 9  مار myShell
-rw-r--r--  1 abdelrhman abdelrhman 6.6K 19:29 9  مار myShell.c
drwxrwxr-x  2 abdelrhman abdelrhman 4.0K 22:12 9  مار test
/home/abdelrhman/simpleShell shell >> export x="-a -l -h"
/home/abdelrhman/simpleShell shell >> ls $x
total 64K
drwxr-xr-x  3 abdelrhman abdelrhman 4.0K 22:12 9  مار .
drwxr-xr-x 22 abdelrhman abdelrhman 4.0K 22:29 9  مار ..
-rwxr-xr-x  1 abdelrhman abdelrhman 18K 02:31 6  مار a.out
-rw-rw-r--  1 abdelrhman abdelrhman 400 22:38 9  مار log.text
-rwxrwxr-x  1 abdelrhman abdelrhman 18K 22:12 9  مار myShell
-rw-r--r--  1 abdelrhman abdelrhman 6.6K 19:29 9  مار myShell.c
drwxrwxr-x  2 abdelrhman abdelrhman 4.0K 22:12 9  مار test
/home/abdelrhman/simpleShell shell >> 
```

Figure 4: export and ls

3.5 Error command and exit



```
abdelrhman@abdelrhman-VirtualBox: ~/simpleShell
abdelrhman@abdelrhman-VirtualBox:~/simpleShell$ ./myShell
/home/abdelrhman/simpleShell shell >> heey
Error ! unknown command
/home/abdelrhman/simpleShell shell >> exit
abdelrhman@abdelrhman-VirtualBox:~/simpleShell$
```

Figure 5: Errors and exit

4 The process hierarchy

4.1 foreground process : the process doesn't terminate until we close the gedit because it's a foreground process

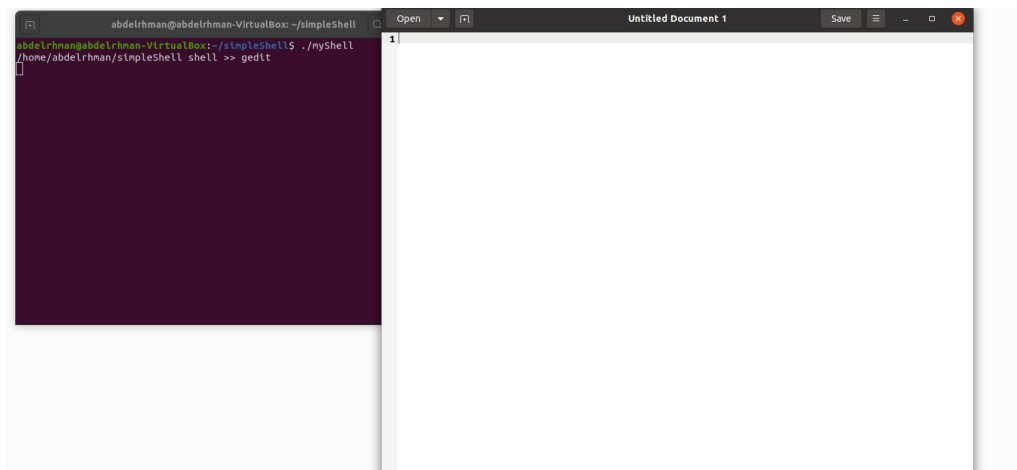


Figure 6: before termination

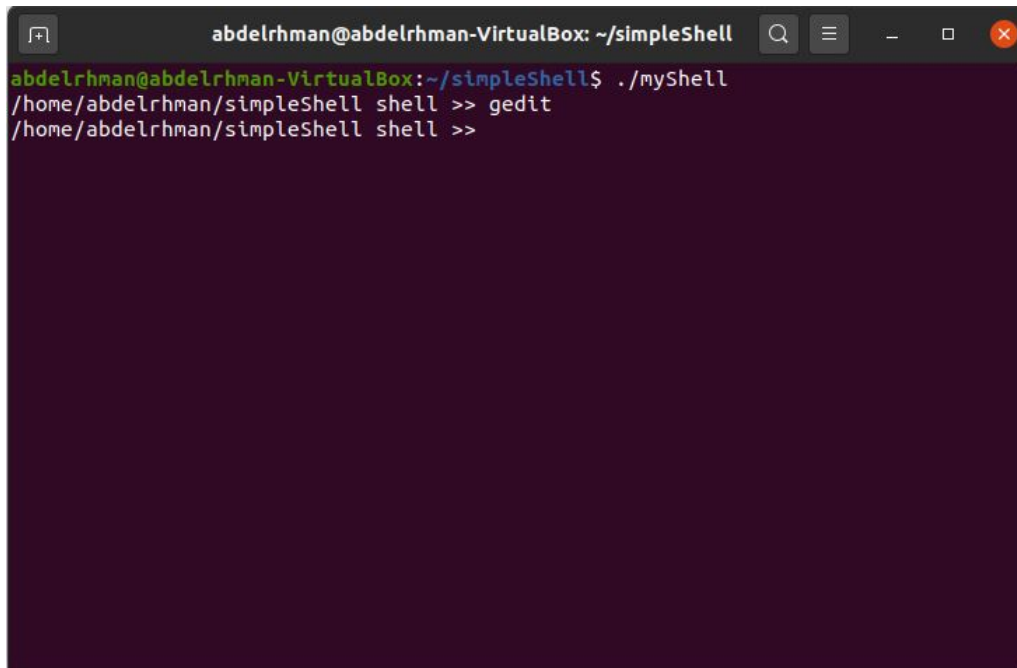


Figure 7: after termination

4.2 background process : the process work in the background and the user can enter another command

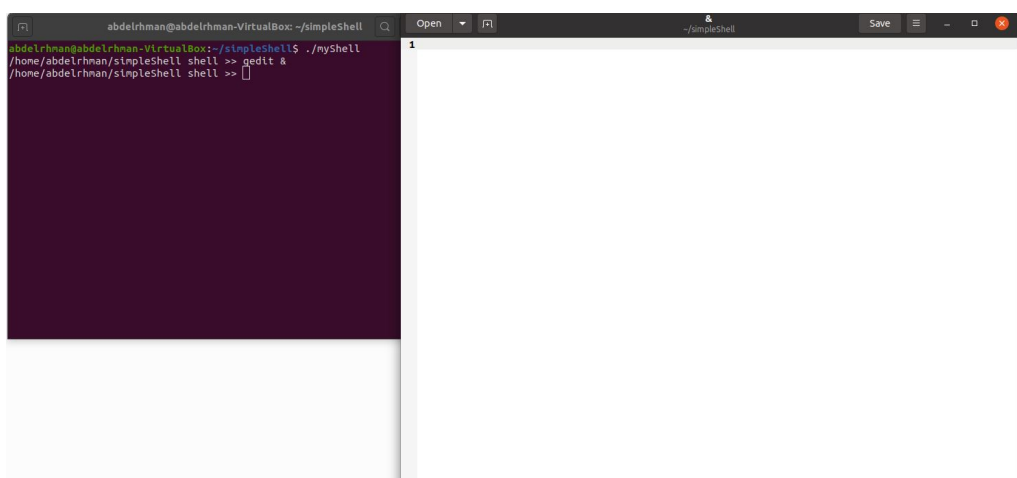


Figure 8: background process

The processes in system monitor

Processes							
Resources							
File Systems							
Q							
Q							
Process Name	User	% CPU	ID	Memory	Disk read tot	Disk writ	
evolution-addressbook-factory	abdelrhman	0	1654	312.0 KiB	2.4 MiB	36.0	
evolution-alarm-notify	abdelrhman	0	1716	16.1 MiB	1012.0 KiB		
evolution-calendar-factory	abdelrhman	0	1645	992.0 KiB	6.2 MiB		
evolution-source-registry	abdelrhman	0	1636	1.4 MiB	4.6 MiB		
gdm-x-session	abdelrhman	0	1350	252.0 KiB	104.0 KiB		
gedit	abdelrhman	0	44788	18.3 MiB	N/A		
gjs	abdelrhman	0	1670	4.9 MiB	N/A		
gnome-calendar	abdelrhman	0	40643	14.8 MiB	22.7 MiB		
gnome-keyring-daemon	abdelrhman	0	1557	52.0 KiB	N/A		
gnome-keyring-daemon	abdelrhman	0	1293	636.0 KiB	N/A		
gnome-session-binary	abdelrhman	0	1577	1.5 MiB	8.1 MiB	4.0	
gnome-session-binary	abdelrhman	0	1386	1.4 MiB	4.8 MiB		
gnome-session-ctl	abdelrhman	0	1561	N/A	20.0 KiB		
gnome-shell	abdelrhman	6	1600	300.4 MiB	38.0 MiB	208.0	
gnome-shell-calendar-server	abdelrhman	0	1630	1.6 MiB	7.3 MiB		
gnome-system-monitor	abdelrhman	2	44724	15.1 MiB	4.0 KiB		
gnome-terminal-server	abdelrhman	0	44735	11.9 MiB	N/A		
End Process							

Figure 9: System monitor