
Creating a shell Command Interpreter

Introduction :

This project consists of designing a C program to serve as a shell interface that accepts user commands and then executes each command in a separate process. Your implementation will support input and output redirection, as well as pipes between a pair of commands. Completing this project will involve using the linux `fork()`, `exec()`, `wait()`, `dup2()`, and `pipe()` ...

1. Installation

1-installing VM VirtualBox and installing ubuntu

2- create a virtual machine

3- open the terminal and write “**sudo apt update**”. The « **sudo apt update** » command is used to download package information from all configured sources.

4- The default Ubuntu repositories contain a meta-package named “**build-essential**” that includes the GNU compiler collection, GNU debugger, and other development libraries and tools required for compiling software.

Write “**sudo apt build-essential**”: The command installs a lot of packages, including gcc, g++ and make.

5- To validate that the GCC compiler is successfully installed, use the

“**gcc -version**” command which prints the GCC version.

2. Programming:

1- To create a folder, write "**mkdir folder-name**"

2- Create a file with extension c "**touch filename.c**".

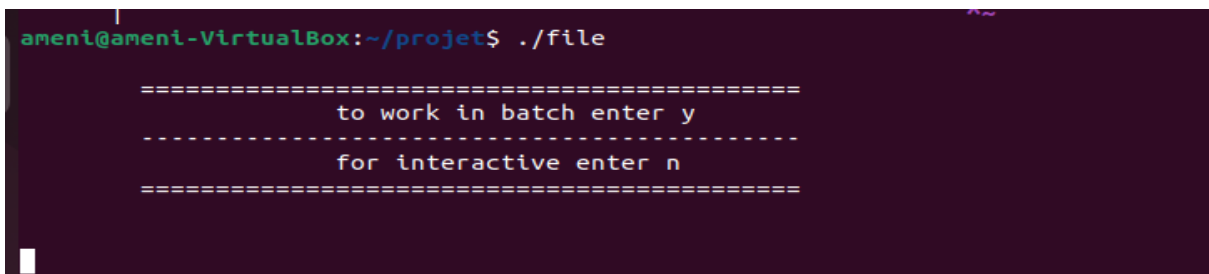
3- First, we include the library like “**#include <stdio.h>**” ...



```
1 #include <sys/wait.h>
2 #include <sys/types.h>
3 #include <unistd.h>
4 #include <stdlib.h>
5 #include <stdio.h>
6 #include <string.h>
7 #include <ctype.h>
8 #include <dirent.h>
9 #include <sys/stat.h>
10 #include <signal.h>
11 #include <fcntl.h>
12 #include <termios.h>
13 #define BUFSIZE 1024
14 #define MAXARGNUM 32
15 #define STACKSIZE 15
16 #define HISTSIZE 32
17 #define ALIASIZE 15
18 #define DELIMITER " \t\r\n\a"
--
```

Figure 1 :Library lists

- 4- We start with our main by declaring a bunch of variables like strings, pointers ...
- 5- open the terminal, after we type the command to debug the program “**gcc name file.c -o name file**” then we execute it with the help of the command “**./ file name**”



```
ameni@ameni-VirtualBox:~/projet$ ./file
=====
to work in batch enter y
-----
for interactive enter n
=====
```

Figure 2 : The output for the execute command “./file”

- 6- The program asks you if you want to work in batch or interactive mode. If you choose batch then you enter “**y**” and you execute the command that will enable you to see the output of the batch file which will have the extension of your choice (.sh or .bat). And if you choose interactive mode then you enter “**n**” for exemple and you will be presented with a custom prompt that will accept all types of command

```
n

=====
welcome to the
-----
Interactive shell
=====
```

Figure 3 : The output for choosing to work in interactive mode

```
y

=====
welcome to
-----
batch
=====
```

Figure 4 : The output for choosing to work in batch mode

7- Now we will choose interactive mode by typing “n” next to we are trying to execute a simple command like “ls” command, the execution of these commands is done by using my launch function to launch and run external programs within the program shell, and we execute with a function called execute() in C. This function is used to keep a record of the commands entered by the user in the shell program and it stores the commands in a history array, accordingly we have a list of commands used in the terminal session

8- In addition, we present our composite function:

when you enter your command the program will take it and put it in a series of characters that will be analyse it one by one first using the “line” variable we test for specific characters like “:”, “&&”, ”||”, “|”, “>” after we separate them from the space with the help of strtok with delimiter being" " . if we found the “;” we increment and we send each character before and after it to be executed

Additionally, the pipe” |” command allows you to pipe the output of one command to another as mentioned in the screenshot below.

```
/home/ameni/projet% ls | wc
/home/ameni/projet%      6      6      52
```

Figure 5 : The output for the command pipe

```

/home/ameni/projet% ls ; data ; echo "projet"
ameni.txt batch1.sh file file.c Makefile projet.txt
child process: execution error: No such file or directory
"projet"
/home/ameni/projet% ls ; date ; echo "projet"
ameni.txt batch1.sh file file.c Makefile projet.txt
19 2023 م ٠٨:١٨:١٥ CET جاني
"projet"
/home/ameni/projet%

```

Figure 6 : The output for the command “;” with it error management

```

/home/ameni/projet% ls && date && pwd
ameni.txt batch1.sh file file.c Makefile projet.txt
19 2023 م ٠٨:٢١:١٩ CET جاني
/home/ameni/projet
/home/ameni/projet% ls && data && pwd
ameni.txt batch1.sh file file.c Makefile projet.txt
child process: execution error: No such file or directory
/home/ameni/projet%

```

Figure 7 : The output for the command “&&” with it error management

```

/home/ameni/projet% ls || data
ameni.txt batch1.sh file file.c Makefile projet.txt
child process: execution error: No such file or directory
/home/ameni/projet%

```

Figure 8 : The output for the command or “||”with it error management

In comparison to the last command is The “>” operator is used to change the source of the standard input. This method can be useful to take user input from file contents and store it in program buffer with the help of the function fork and dup and dup2 where the out STDOUT will be put in a file with the name is given by the user after the “>” as shown in the screenshot of the output below.

```

/home/ameni/projet% ls > ameni.txt
ameni.txt batch1.sh file file.c Makefile projet.txt
/home/ameni/projet% cat ameni.txt
ameni.txt
batch1.sh
file
file.c
Makefile
projet.txt
/home/ameni/projet%

```

Figure 9 : The output for the command “>”

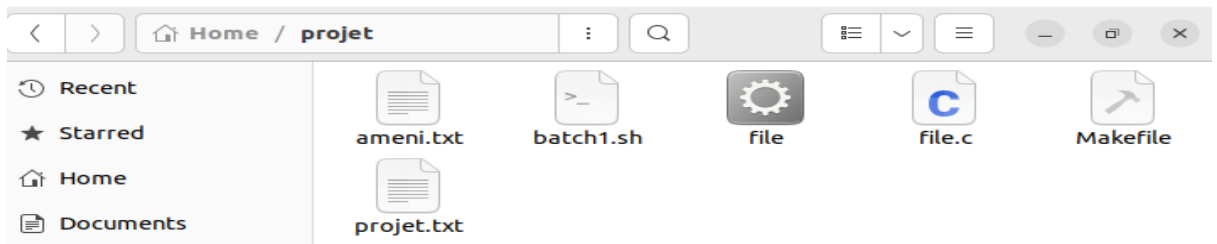


Figure 10 : The contents of the folder “Projet”

At last the quit function terminates the program by specifying a return code. This return code, passed as a parameter of the quit function, is used to specify how the program ends. A null value (0 or constant THANKS_FOR_USING_THE_SHELL!)

```
/home/ameni/projet% /home/ameni/projet% quit
*****
*                                     *
*              THANKS FOR USING THE SHELL!              *
*                                     *
*****
ameni@ameni-VirtualBox:~/projet$
```

Figure 11 : The output for the command “quit”

3. Setup guide:

Before you do any of the steps below you should open the folder and access it in the terminal. Using command “cd” then the folder name.

1. We should download the code in file with extension.c.
2. Debug the program with the help of the command “**gcc filename.c -o filename**”.
3. Run the code with the help of the command “**./filename**”.
4. Type some commands, simple complex ones.
5. For the batch file you download the file with the command in it and save it
6. To debug it type the command **chmod 777 filename**.
7. To run it type “**./filename.extension**”.
8. For the make file download the script then save it as "**Makefile**".

To automate the commands we use:

1. **"make"** for it to generate a file with extension **".o"**.
2. To remove the file you generated use the command **"make clean"**.
3. To debug the file.c type **"make debug"**.
4. To run the file.c type **"make execute"**.
5. To debug the batch file type **"make debugBatch"**.
6. To run the batch tile type **"make executeBatch"**.

4. Makefile

A makefile can be used to control the build process of an object file such as "file.o" in Linux. The makefile would typically contain a rule that specifies how the object file should be built from its corresponding source file, for example:

file.o: file.c

gcc -c file.c -o file.o

This rule states that the object file "file.o" depends on the source file "file.c" and that the command **"gcc -c file.c -o file.o"** should be executed to build **"file.o" from "file.c"**.

The make tool uses these rules to automatically execute, build, debug or even delete.

make it build file.o

```
# file Makefile

CC = gcc
CFLAGS = -Wall -g
OBJ = file.o

all: file

file: $(OBJ)
    $(CC) $(CFLAGS) -o file $(OBJ)

%.o: %.c
    $(CC) $(CFLAGS) -c $<
```

Figure12: The code for the command "make"

```
moenes@moenes-VirtualBox:~/Pro$ make
gcc -Wall -g -c file.c
```

Figure13: The output for the command “make”

“**make clean**” it will remove the file specified

```
clean:
    rm $(OBJ)
```

Figure14: The code for the command “make clean”

```
moenes@moenes-VirtualBox:~/Pro$ ls
batch1.sh batch.sh file file.c file.o Makefile
moenes@moenes-VirtualBox:~/Pro$ make clean
rm file.o
```

Figure15: The output for the command “make clean”

“**make debug**”, it debug using the command “file.c gcc -c file.c -o file”.

```
debug:|
    gcc file.c -o file
```

Figure16: The code for the command “make debug”

```
moenes@moenes-VirtualBox:~/Pro$ make debug
gcc file.c -o file
```

Figure17: The output for the command “make debug”

“**make execute**”, it runs the file.c using the command “./file”.

```
execute:
    ./file
```

Figure18: The code for the command “make execute”

The entire code of the Makefile:

```
# file Makefile

CC = gcc
CFLAGS = -Wall -g
OBJ = file.o

all: file

file: $(OBJ)
    $(CC) $(CFLAGS) -o file $(OBJ)

%.o: %.c
    $(CC) $(CFLAGS) -c $<

clean:
    rm $(OBJ)

debug:|
    gcc file.c -o file

execute:
    ./file

debugBatch:
    chmod 777 batch1.sh

executeBatch:
    ./batch1.sh
```

Figure24: The code for Makefile

5. Function used:

Next we move to show some of the functions proper to c language to help us and others we developed it.

buffer	to output or input variables.
strtok	breaks string str into a series of tokens using the delimiter delim
opendir	defined in the header file and is part of the POSIX (Portable Operating System Interface) specification
strlen	used to calculate the length of a string
strncpm	takes two strings as arguments and compares these two strings lexicographically.
cwd	places an absolute pathname of the current working directory in the array pointed to by buf, and returns buf
fork	used for creating a new process
exec	replaces the current running process with a new process
stdr	is a general-purpose polymorphic function wrapper.
wait	blocks the calling process until one of its child processes exits or a signal is received

closedir	closes the directory specified by dirp, and frees the memory allocated by opendir()
tokens	Keywords, Identifiers, Constants, Special Characters, Strings and Operators
free	allows you to release or deallocate the memory blocks which are previously allocated by calloc(), malloc() or realloc() functions
chdir	is used to change the current working directory.
perror	prints a descriptive error message to stderr.
execte	The execution of a C program begins from the main() function.
Getcwd	places an absolute pathname of the current working directory in the array pointed to by buf, and returns buf.
dirstack	uses the built-in command pushd to change the directory
spiltline	is used to split a string into a series of tokens based on a particular delimiter
strcpy	used to copy the character array pointed by the source to the location pointed by the destination
strtol	converts the initial part of the string in str to a long int value
Strcat	concatenates string2 to string1 and ends the resulting string with the null character
aliaslist	refers to the situation where the same memory location can be accessed using different names
aliascount	used to write a line or string to the output
getopt	used to parse command line arguments
pipefds	is used for reading data from the pipe
splitline	used to split a string into a series of tokens based on a particular delimiter.
execvp	This function takes in the name of the UNIX command to run, as the first argument
pid	you can get the pid for a process via the getpid system cal
Wait pid	allows the calling thread to obtain status information for one of its child processes
dup/dup2	creates copy of file descriptor
Strtok_r	is used to break string str into a series of tokens

Table1:table describing some of the function in c built in and developed