Faculty of Engineering & Technology

Electrical & Computer Engineering Department

Circuits & Electronics Lab - ENEE2103

Experiment 10: The Operational Amplifier

Prepared by:

Mohammad Shrateh _ 1201369

Instructor: Dr. Amr Slimi

Assistant: Eng. Tareq Zidan

Section: 1

Date: 31/1/2024

1. Introduction:

The goal of this project is to create a system directory for shell commands using the Python programming language. The system will create structured XML files for each command separately, where each directory contains a description of the command, release date, examples, and related commands. The project also includes verification of the generated content with directory comparison, continuous improvement using the command recommendation system and search function, and integration of object-oriented programming (OOP) principles.

2.project Overview

The project is structured into several parts:

Part one : Python script Development :

In this part, the code was written to automatically create system directories for shell commands using the Python language. It reads the commands through a file containing the commands, each command on a line, and then creates the corresponding directories in XML format.

Part Two: Verification

In this part, the contents of the files are verified for commands to create a new file containing information, and if there is a difference in the contents of the files, they are printed on the screen.

Part Three: Continuous Improvement and Guidance

In this part, there are two sections:

In the first section, it displays commands similar to the command being used. The similarity may be in terms of letters or through the function of the command.

In the second section, the name of the command is entered, and the program searches for it and quickly prints its contents on the screen.

A command recommendation system is implemented to suggest Python commands based on command names and functions.

Search function is added to the directory to quickly retrieve information.

Part 4: OOP Integration

In this part, object-oriented programming principles are applied to enhance modularity, encapsulation, and reusability.

Where 3 classes are created, namely CommandManualGenerator, CommandManual, and XmlSerializer, to manage manual creation and serialization.

Discussion:

```
# function to retern the description for commands
def command_description(self, command_name):
    try:
        # using man to return the descriptin manual
        command_descrption = subprocess.check_output(
            ["man", command_name], universal_newlines=True
        )
        # search for NAME
        command_name = command_descrption.find("NAME")
        # get command description
        command_des = command_descrption.find("DESCRIPTION", command_name)
        descrption = command_descrption[command_name:command_des].strip()

        # print first paragraph
        descrption = descrption.split("\n\n")[0]
        return descrption

    except Exception as e:
        # print error
        print(f" error occurred: {e}")
```

In this part, I used man to get the command manual, then I searched for NAME, then searched for the description section and returned the first idea from it.

```python
# function to retern the version for commands
def command_version(self, command_name):
    try:
        # using '--version' to return the the version of commands
        command_version = subprocess.check_output(
            [command_name, "--version"], universal_newlines=True
        )
        # print only the first line
        version = command_version.strip().split("\n")[0]
        return version

    except Exception as e:
        # print error
        print(f"An unexpected error occurred: {e}")
```

In this part, I used --version to get the version of the command, and then displayed the first line of it.

```python
# function to retern the related command for commands
def related_command(self, command_name):
    try:
        # use the 'compgen -c' to return the related commands
        related_command = subprocess.check_output(
            ["bash", "-c", "compgen -c"], universal_newlines=True
        )

        all_related_commands = related_command.strip().split("\n")
        # list to add related commands
        related_commands = []
        for i in all_related_commands:
            if i.startswith(command_name):
                # add related command to list
                related_commands.append(i)

        # return list of commands
        return related_commands

    except Exception as e:
        # print error
        print(f"An unexpected error occurred: {e}")
```

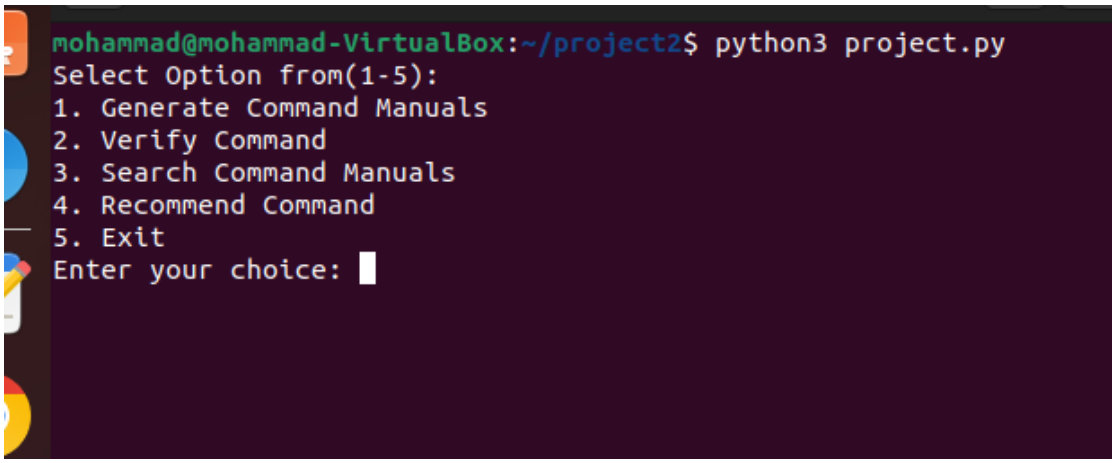
In this part, I used the command ‘compgen –c’ to get the suspicious commands for the command being used, then I defined a list to add all the similar commands and then return them.

```python
# function to retern the example for commands
def command_example(self, command_name):
    try:
        # using '--help' to return the example of commands
        example_command = subprocess.check_output(
            [command_name, "--help"], universal_newlines=True
        )
        # return the first line example for command
        return example_command.strip().split("\n")[0]

    except Exception as e:
        # print error
        print(f"An unexpected error occurred: {e}")
```

In this part, I used --help to get the examples of the command, and then displayed the first line of it.

3. Result:



In this image, when the option allows us to select the menu and choose the option from among the existing options.

```
3. Search Command Manuals
4. Recommend Command
5. Exit
Enter your choice: 1
-------------------------------------------
The command generated are :
1) ls.xml
2) cat.xml
3) pico.xml
4) pwd.xml
5) echo.xml
6) mkdir.xml
7) rmdir.xml
8) sed.xml
9) grep.xml
10) rm.xml
11) find.xml
12) nano.xml
13) sed.xml
14) head.xml
15) tail.xml
16) chmod.xml
17) mv.xml
18) kill.xml
19) ps.xml
20) cp.xml
the number of generatet commands is 20
-------------------------------------------
```

In this picture, when we choose the first option, it creates files for the commands and prints them on the screen.

```
-------------------------------------------
Select Option from(1-5):
1. Generate Command Manuals
2. Verify Command
3. Search Command Manuals
4. Recommend Command
5. Exit
Enter your choice: 2
-------------------------------------------
Enter command name to verify: cat
Generated cat_new.xml
No change between the command manuals
-------------------------------------------
Select Option from(1-5):
1. Generate Command Manuals
2. Verify Command
3. Search Command Manuals
4. Recommend Command
5. Exit
```

In this image, when we choose the second option, it asks us to enter the name of the command, and then it creates a new file for the command and compares the contents with each other.

```
3. Search Command Manuals
4. Recommend Command
5. Exit
Enter your choice: 3
-------------------------------------------
Enter full command name to search: ls
Command Name: ls
Description: NAME
       ls - list directory contents

SYNOPSIS
       ls [OPTION]... [FILE]...
Version History: ls (GNU coreutils) 8.32
Example: Usage: ls [OPTION]... [FILE]...
Related Commands: lsmod, lspcmcia, lspci, lslocks, lsblk, lsirq, lsinitramfs, l
susb, lsattr, lspower, lsns, lsmod, lsof, ls, lshw, lsipc, lslogins, lsb_releas
e, lspgpot, lsmem, lscpu, lsfd, lsmod, lspcmcia, lspci, lslocks, lsblk, lsirq,
lsinitramfs, lsusb, lsattr, lspower, lsns, lsmod, lsof, ls, lshw, lsipc, lslogi
ns, lsb_release, lspgpot, lsmem, lscpu, lsfd
-------------------------------------------
Select Option from(1-5):
1. Generate Command Manuals
2. Verify Command
3. Search Command Manuals
4. Recommend Command
```

In this image, when we choose the third option, it asks us to enter the name of the command, and then it searches for the command, and if it finds it, its contents are printed on the screen.

```
3. Search Command Manuals
4. Recommend Command
5. Exit
Enter your choice: 4
-------------------------------------------
Enter command name to recommend: ps
Recommended commands for 'ps':
psicc
ps2pdf12
ps2pdf13
ps2txt
ps2ps2
pslog
ps2pdf14
pstree.x11
psfxtable
psfstriptable
ps2pdfwr
psfaddtable
ps2ascii
ps2ps
psfgettable
pstree
ps2pdf
ps2epsi
```

In this picture, when we choose the fourth option, it asks us to enter the name of the command, and then it searches for a similar command, and if it finds it, the commands are printed on the screen.

```
--------------------------------------------
Select Option from(1-5):
1. Generate Command Manuals
2. Verify Command
3. Search Command Manuals
4. Recommend Command
5. Exit
Enter your choice: 2
--------------------------------------------
Enter command name to verify: cat
Generated cat_new.xml
Changes detected:
-        cat [OPTION]... [FILE]...</CommandDescription><VersionHistory>cat (GNU
 coreutils) 8.32</VersionHistory><Example>Usage: cat [OPTION]... [FILE]...</Exa
mple><RelatedCommands>catman, cat, catman, cat</RelatedCommands></CommandManual
>
+        cat [OPTION]... [FILE]...</CommandDescription><VersionHistory>3033</Ve
rsionHistory><Example>Usage: cat [OPTION]... [FILE]...</Example><RelatedCommand
s>catman, cat, catman, cat</RelatedCommands></CommandManual>
--------------------------------------------
Select Option from(1-5):
1. Generate Command Manuals
2. Verify Command
3. Search Command Manuals
4. Recommend Command
```

This image is an example of verify if they are not the same.

4. Conclusion

At the end of this project, we were able to complete all parts correctly and obtain the correct results. We also learned how to apply OOP in writing code and how to link shell commands by creating manuals for these commands using programming commands in the Python language.