carryOn: Pick up where you left off

by

Aditya Kumar (B18CSE002)

Devin Garg (B18CSE011)

Kartik Vyas (B18CSE020)

Department of Computer Science & Engineering

A project report submitted to the
Department of Computer Science & Engineering
in fulfilment of the requirements for the
course of Operating Systems - CS330

Supervisor:
Dr. Suchetana Chakraborty Dr. Ravi Bhandari

Department of Computer Science

May, 2021

Declaration

We hereby declare that this project report is original and has not been published and/or submitted for any other degree award to any other university before.

#	**Names**	**Registration Number**	**Signature**
1	Aditya Kumar	B18CSE002	
2	Devin Garg	B18CSE011	
3	Kartik Vyas	B18CSE020	

Date: May 8, 2021

Approval

This Project Report has been submitted along with the course project of Operating Systems (CS330), session of March '21 with the approval of the following supervisors.

Signed:

Date:

Dr. Suchetana Chakraborty
Assistant Professor, IIT Jodhpur
eMail: suchetana@iitj.ac.in
Phone: (91 291) 280 1273

Dr. Ravi Bhandari
Young Faculty Associate, IIT Jodhpur
eMail: rbhandari@iitj.ac.in
Phone: (91 291) 280 1268

Acknowledgement

We are heartily thankful to our instructors, Dr. Suchetana Chakraborty and Dr. Ravi Bhandari, for providing us the necessary guidance, the needed constant support, and helping us throughout the course of the project via continuous interaction and evaluation of the course at regular intervals.

Contents

Chapter 1

Abstract

The shell in UNIX systems is used to interpret user commands and helps in controlling and launching programs. We came up with the idea to implement our own basic shell along with additional features that a normal shell does not offer. We came up with the idea to input commands to the shell using voice commands and control and open different programs after successful recognition of voice commands. This feature will make the normal shell more advanced and interesting to use.

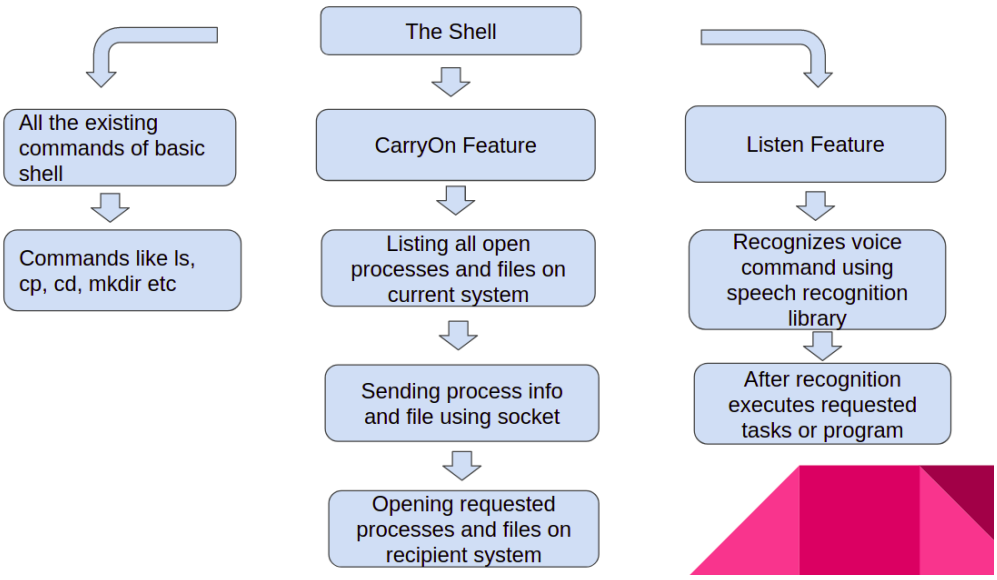

Figure 1.1: Overall flow of the project

Additionally Apple handheld devices like iPads, iPhones, etc have this feature where users can switch from one system to another and continue their work from where they left off in the previous system but it is proprietary in nature and limited to these devices. Here we propose this functionality as a programmer's aid to go from one machine to another.

To implement this feature we will need to list out all the processes open in the current system, along with open files and URLs in that process, and send them to another system through some method. We have extensively used GRPC and sockets to communicate between two systems and transfer the necessary details to implement this feature which further advances the basic shell.

Chapter 2

Introduction

Shell is used for interpreting commands given by the user and controlling and launching programs. We implemented all the basic commands like ls, mkdir, cd, cp, mv, and other commands using the subprocess library of python. Copying of files and directories is done recursively and also when there is conflict, the user is given the choice to overwrite or not already existing file. We have also integrated voice based shell utilities and a "Carry On" feature in our shell. On inputting voice command, the system starts listening for voice command, and after successful recognition of voice command, it performs the requested tasks. We are using python's speech recognition library to recognize voice commands. After recognizing the voice command, the system searches for the binary file of the process and executes it. Some of the tasks that are being handled by listen command are opening the camera, opening the terminal, playing a video on youtube, searching on Wikipedia and displaying the result, and many more tasks. On inputting carryon command user can switch from one system to another from where he left off. To implement this feature we list out all the open processes and the user is given the choice to select the processes which he wants to resume on another device. After selecting the processes user is given some time to save files open in the respective process. Next information about the processes and the file data are transferred to another system with help of GRPC and sockets individually. After successfully receiving the info and files on another system the processes are opened on another system along with the respective files with the help of the subprocess library of python.

Chapter 3

Background

3.1 Sockets

Simply put, a socket is an endpoint communication instance for a node that is present in a network. In general, sockets include information about the transmission protocol in use, the IP address, and the port number. So, how that works is - whenever a node acts as a server, it opens a socket and starts listening for connection requests. A client on the other hand is aware of the IP address of the server and the port on which the socket is open. With this information, the client is able to try to connect with the server, which, given some constraints are satisfied which may include authentication, then accepts the connection request, and then a connection is established between the client and the server.

3.2 GRPC

It is a modern open-source high-performance Remote Procedure Call framework suitable for any environment built by Google. In GRPC a client application can call a method on a server application on a different machine as if it were a local object, making it easier for us to create distributed applications and services. In GRPC we can define our own service, we can define our own methods and return types that can be called remotely with their parameters. On the server-side, the server implements this interface and runs a GRPC server to handle client calls. On the client side, the client has a stub that provides the same methods as the server. GRPC clients and servers can run and talk to each other in a variety of environments. GRPC uses protocol buffers, Google's open-source mechanism for serializing structured data.

Chapter 4

Methodology

4.1 The Shell

Commands Handled :

- ls command : It lists out all the files in the current working directory.
- cd command : it is used to change the current working directory. In case of too many arguments, it will indicate an error.
- mkdir command : It is used to make new directories. In case the directory already exists, it will output a message indicating the same. In the case of fewer arguments, it will indicate an error.
- cp command : It is used for copying files and directories. It first checks for the destination directory if it exists or not. In case the destination directory does not exist, the directory will be created. In case the directory exists and it's a file, then an error will be indicated. Then it checks whether the source file or directory exists or not, if it does not exist an error will be indicated. After all the validations it begins the copying process. If we are copying a directory, the system will copy all its contents and copying is done recursively. In case it is a file, it checks for conflict, if there already exists a file, the system will display information about both the files and will ask the user if the user wants to overwrite the given file or not.
- mv command : It is used to move files or directories from one location to another.

- listen command : We have integrated this new feature into the shell where the system will listen to voice commands by the user. Detailed info about this interface will be found below in the report.
- carryon command : In apple products, one can switch from one system to another and continue from where they left off in the previous system. This feature has also been integrated in the shell. We are using GRPC and Sockets for implementing this feature. Detailed info about this interface will be found below in the report.
- Rest all commands have been integrated into the shell which are part of the normal shell

4.2 Listen Feature

The system asks the user to input commands via voice. We are using python's speech recognition library for recognizing the voice of the user and further executing the command given by the user. The commands that we are currently handling are as follows

- Opening Notepad
- Opening Terminal
- Opening Camera
- Playing music requested by the user
- Displaying IP address of the system
- Searching on Wikipedia
- Opening and playing user's requested video on Youtube
- Opening Facebook
- Opening StackOverflow
- Opening Google and displaying user's query
- Sending message Over WhatsApp
- Sending Email

4.3 CarryOn Feature

Implementing handoff feature, where the user can switch from one system to another form where he left off on the previous system. On-call of the carryon command we first list out all the running processes and along with their id's, then the user is asked which processes he wants to continue on another system. Next, it finds out all the files or URLs that were open in this process. The user is given 10 seconds to save currently open files in these processes.

4.3.1 Strace

Identifying file open in a process: For the task of continuing the process on a different machine, a crucial component is the file that is currently being edited on the source machine. Identifying the open windows is performed without a lot of trouble, but the task of finding out which file is open in the process turned out to be non-trivial.

After trying out some things we took a different approach which was closer to the kernel. We took note of the fact that whenever a process does some operation on an open file, it is bound to use the system calls of read() or write() or close(). Now, we looked into how we can identify when processes are making these function calls and what are the parameters. This should, in theory, give us the file that is currently being edited by the process that is open.

For this we utilized a utility called strace. Quite literally, this utility monitors system calls. It can be invoked by passing the process ID of a running process or by passing the process that it should run and attach itself to it. When strace 'attaches' itself to a process, it in some sense snoops on all the system calls made by that process. This can be limited by specifying what kind of system calls we are interested in - in our case the above mentioned calls. This yields the system calls along with the parameters of the call which contain the address of the file that is to be read or written or closed.

There are however some limitations that we noticed which needed to be taken care of when using this utility:

- Processes don't always function in a simplistic manner of writing just the open file. This means there are some other files that are often written e.g. in case of proprietary software, processes often write to some log files or some configuration files at the same time as they edit the open file.

- Sometimes, processes disguise the file name that they are writing to. Consequently, the tracing utility does give some system calls, but the parameters are often some non-file paths or paths to some cache files which are then written to the actual file at some other random interval. Now, this is a pretty program specific thing to handle, so we looked at alternatives to the approach such as extracting the title bar information to get to the required file.

4.3.2 Socket

For transferring the files, first we extract all the file data from .json file info. Then, we use sockets to transfer these files from the source to destination system. For this, we initiate the socket server at the source system and then the client connects to the server and then authentication is done via password matching.

If the authentication is successful then, one by one we send each file first by sending the filename, the file size for smooth transferring and then the file itself. Once file transfer is completed, we add the file details, name and process type for further processing.

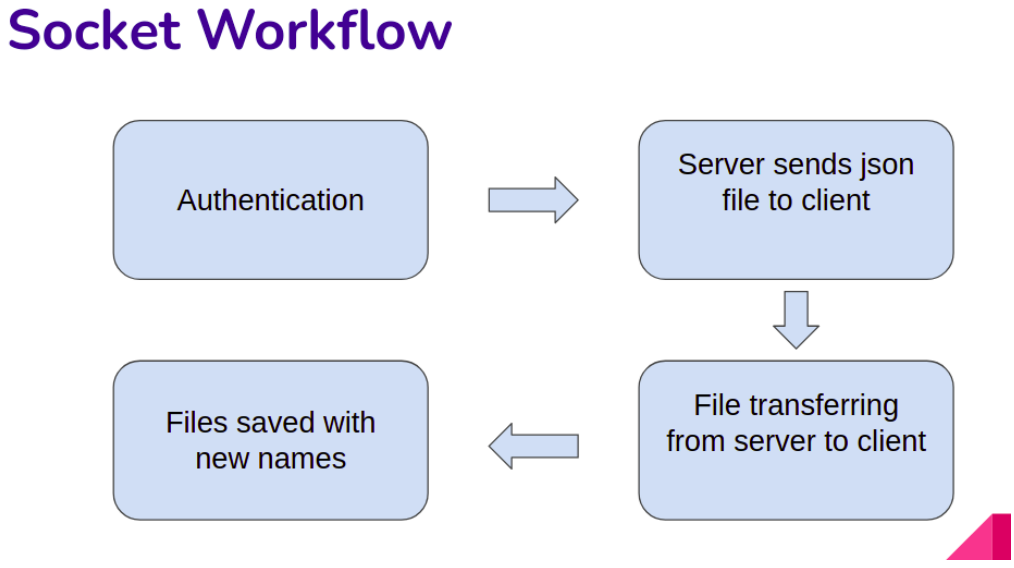

Figure 4.1: Socket workflow

4.3.3 GRPC

At the core of gRPC, we define the messages and services using protocol buffer. Then using Proto compiler, the rest of the code is generated One .proto file works for 12 different programming languages and is platform independent. The data to be transferred is efficiently serialized in bytes We are transferring each file one by one First we transfer a json file that has the details of the files to be transferred Then the client one by one requests for the respective file along with the json that contains all the file information of the files successfully transferred. The file information includes the file address and the process type name.

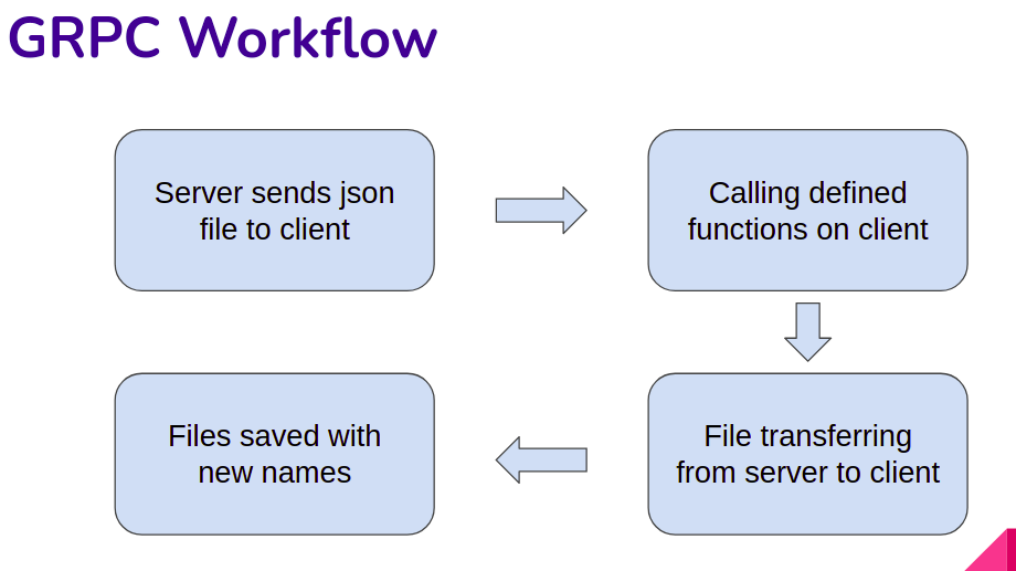

Figure 4.2: gRPC workflow

4.3.4 File Opening

Next, it opens the requested process along with the files or URLs on the requested system. The which command is used to find the location of the binary file of the process. After finding the location of binary file of the process the process is executed using subprocess library of python

Chapter 5

Experimental Setup

We used our laptop to thoroughly test all the utilities. Shell was run and commands were tested successfully. For the carryon feature, one instance of the terminal was serving as server-side while the other was serving as client-side [opened from a different folder]. A list of processes and files was transferred from the server-side to the client-side and all the processes were open along with the sent files on the client-side to test our project.

Chapter 6

Challenges and Future Prospects

6.1 Challenges

- Listing the files opened in active applications
- File Transfer using GRPC

6.2 Future Prospects

- Enhanced GUI
- Adding more features for voice enabled shell
- Checking robustness of 'carryOn' feature

Chapter 7

Results and Conclusion

We were able to implement the basic shell with additional features of voice commands and carry on feature. We were able to open processes, search on google, send mail, and other tasks using the voice feature. We were able to send a list of processes and files from one system to another with help of sockets and GRPC and were able to open these processes with respective files on another system and were able to implement carryon feature.

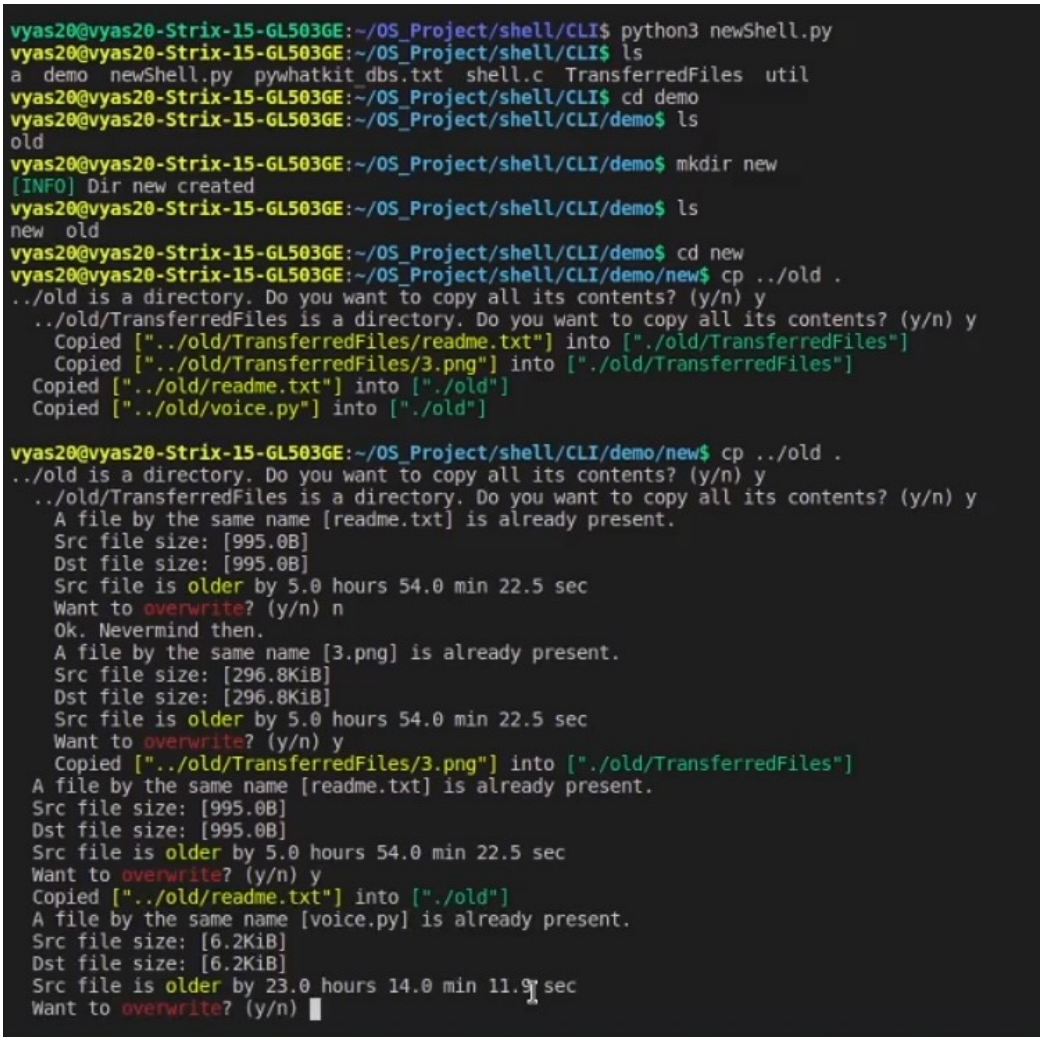

Figure 7.1: Basic shell commands

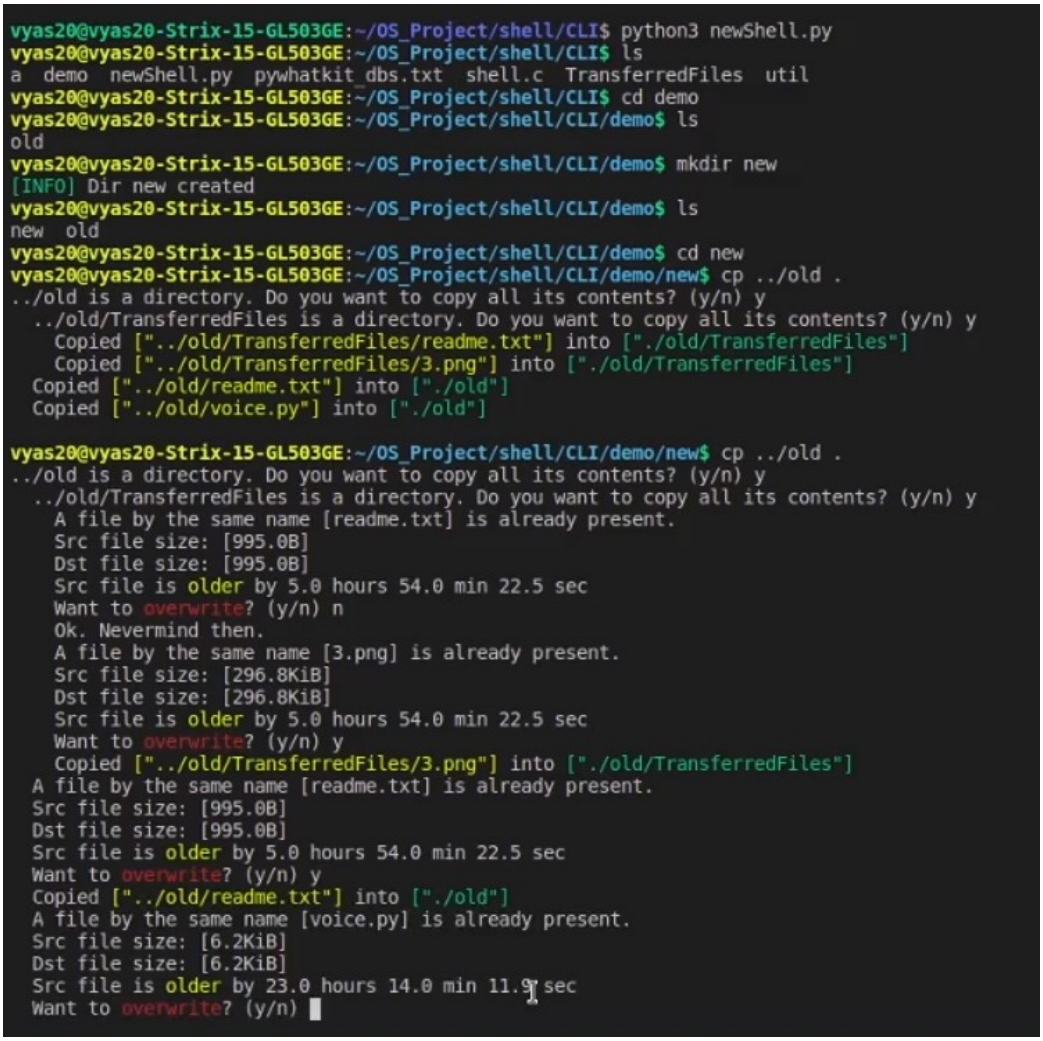

Figure 7.2: carryOn working

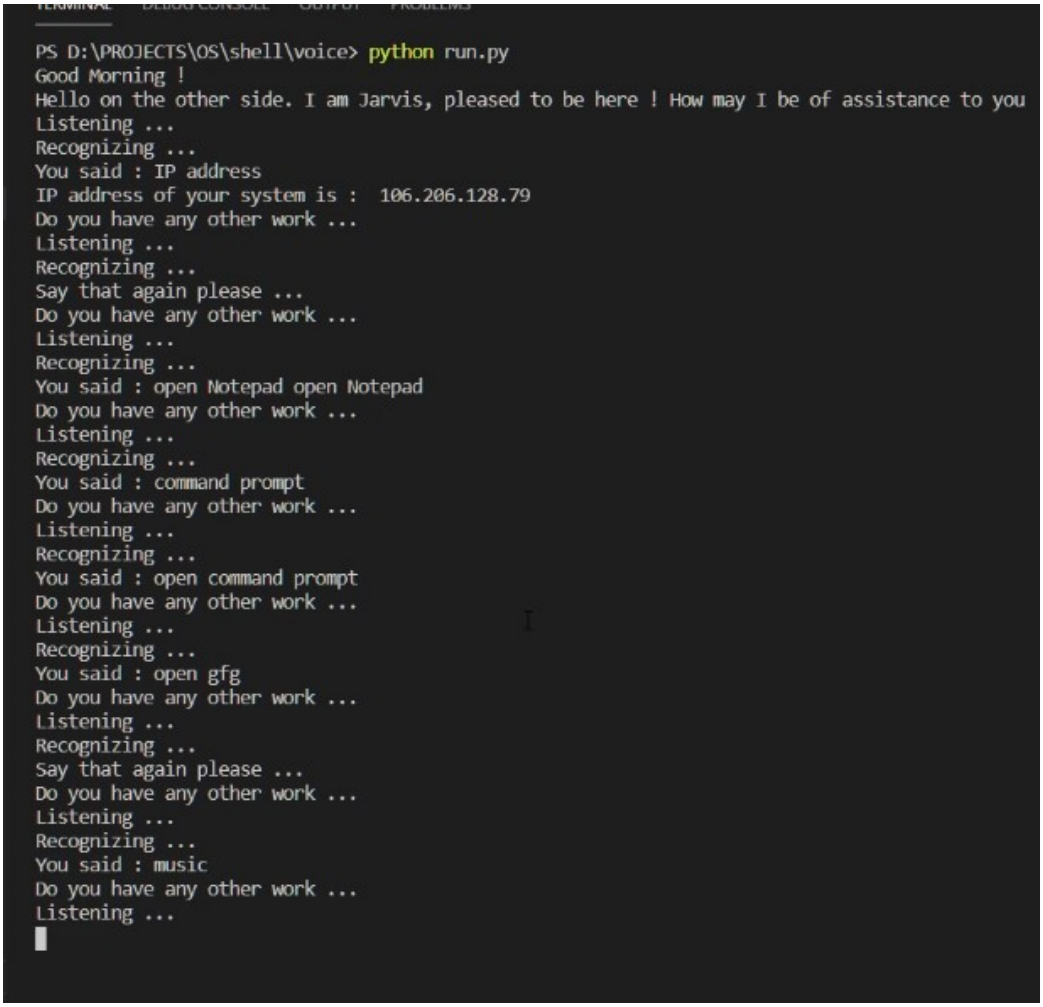

Figure 7.3: listen command working

Chapter 8

Links and References

- Project Github Link
- Project Demo Link
- Subprocess library
- Binary File of any process
- Human Readable Version of File Size
- GRPC
- Sockets