Sravya Kuchipudi
Professor Kaisler, Hasanov
CSCI 6917
15 June 2023

Project Report: Building a GUI for Unix Commands Execution on MacOS

One of the main aspects of this project is connecting a graphical user interface (GUI) to the Mac Terminal to allow for a seamless integration of user interactions and command execution. In the last week I explored different approaches to achieving this integration. The main approaches that were researched were the subprocess module, AppleScript, and the pty module. Each option had its own advantages and considerations which was why this was a needed portion of the project to figure out which would be the best to integrate.

The subprocess module in Python provides a straightforward method to execute commands in the Terminal from the GUI application. By using the subprocess.run() function, input from the GUI can be passed as command-line arguments to the Terminal. This approach is simple to implement and leverages the existing functionality of the operating system. However, it requires executing a separate Terminal process, which may not be ideal for all scenarios.

Another approach involves using AppleScript, a scripting language provided by Apple for automation on macOS. With AppleScript, the user can directly send commands to the Terminal application. The integration between GUI and Terminal is achieved by creating an AppleScript script that takes input from the GUI and executes it as a Terminal command. This approach eliminates the need for a subprocess, making it a lightweight solution. Since it is specific to macOS and requires knowledge of AppleScript syntax that is an added portion of work that needs to be considered.

The pty module offers a different approach by allowing pseudo-terminal utilities. Instead of executing a separate Terminal process, a virtual terminal session is created using the pty.openpty() function. This approach allows for interaction with the Terminal directly from the GUI application. By writing commands to the master file descriptor and reading the output from the slave file descriptor, the GUI can replicate input as lines in the Terminal. The pty module provides greater control and flexibility, but it requires additional code to handle the output and update the GUI accordingly.

Some other approaches that I looked into briefly but discarded were Interprocess Communication (IPC) and Remote Procedure Call (RPC). IPC mechanisms, such as named pipes or sockets, can be used to establish communication between the GUI and the Terminal. The GUI application can write user input to the pipe or socket, and a separate script running in the Terminal can listen for the inputs and execute the corresponding commands. This approach allows for more decoupling between the GUI and the Terminal, enabling them to run independently. However, it requires implementing the IPC mechanisms and coordinating the communication between the two processes. RPC frameworks can be employed to establish a communication channel between the GUI and the Terminal. The GUI application can make remote procedure calls to the Terminal, passing the user input as arguments. The Terminal script, acting as an RPC server, can receive these calls, execute the commands, and return the output to the GUI. This approach provides a standardized and scalable solution but introduces additional complexity due to the need for defining and implementing the RPC interfaces. While both approaches were interesting to look into, they likely would require more research and work than can be finished within the allotted project time.

The current plan is to use the subprocess module to connect the GUI to the Mac Terminal since it has the best learning curve for someone who has not done a lot of related work to this project. It has a

balance of platform compatibility and simplicity that is key to be able to both learn new skills while finishing within the deadline.