Sravya Kuchipudi
Professor Kaisler, Hasanov
CSCI 6917
13 July 2023

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

Since the last report the project has been focused on adding more commands to the commands dictionary as well as beginning to integrate the more specific error handling. For the commands, a handful were added and during the testing of these additions, the issues that arose were addressed to better the usage. While for the error checking there was some progressed and the steps that have be done so far as well as the ones that will be done are addressed.

Over the break a few more commands were integrated into the commands dictionary. The chosen additional commands were pwd, cat, more, head, and tail. The pwd command stands for print working directory and when executed, it displays the full path of the current working directory by the user simply typing pwd in the terminal. It is essential for understanding the current location within the file system hierarchy and helps navigate through directories efficiently. The cat command, short for concatenate, is used to display the contents of one or more files on the terminal and is often used to read text files or to combine multiple files into one. The word cat is used with the file name(s) so cat file.txt will display its contents. The more command is used to view the contents of a file one page at a time and can be executed by more file.txt. The head command is used to display the first few lines of a file by entering head file.txt. By default, it shows the first ten lines, but a different number can be specified (such as head -n 5 file.txt to display the first five lines). The tail command is similar to the head command but displays the last few lines of a file using tail file.txt. This command is frequently used for monitoring log files, where the latest entries or changes appended to the file can be observed and helps track real-time updates without having to open and read the entire file.

These commands were initially easy to integrate but a problem that arose was that when any of the file reading commands were ran the GUI would freeze up and just keep loading. When the generated system error was looked at closely it was noticed that for docx or pdf files the output was “UnicodeDecodeError: 'utf-8' codec can't decode byte 0xc4 in position 10: invalid continuation byte”. This helped identify the issue is that there is trouble parsing these types of files that have embedded stylistic elements that confuse the Terminal. This means that currently the only file types that for sure would work are simplistic ones like txt files. These commands are not designed to work directly on binary files like pdf or docx so while a few attempts for a work around were tried, a decision was made that the investment into this fix was not currently a priority.

The next steps of the project at this point were to improve the error checking mechanisms. To enhance the code by adding error checking, several steps could be taken such as input validation and detailed error responses when possible. First, input validation was implemented to ensure that the user provides valid commands and arguments. This involves checking if the entered command exists in the commands dictionary and verifying that the arguments meet any required format or constraints. Next, potential errors that may occur during command execution will be handled. For example, when using subprocess.run() the generated process error can be displayed as an appropriate error message to the user, providing details about the error that occurred. Additionally, specific file-related errors might need to be handles. When using file operations, such as opening or reading files, there can be catch exceptions like FileNotFoundError or PermissionError to inform the user about the specific file-related issue encountered. It can be beneficial to add these informative error messages through the GUI application rather than just generated into the IDE as they currently are received. Displaying error messages in the output window or using message boxes can help the user understand and troubleshoot any issues that

occur during command execution and help them learn more about Unix commands to provide a smoother experience and better feedback to the user in case of errors.