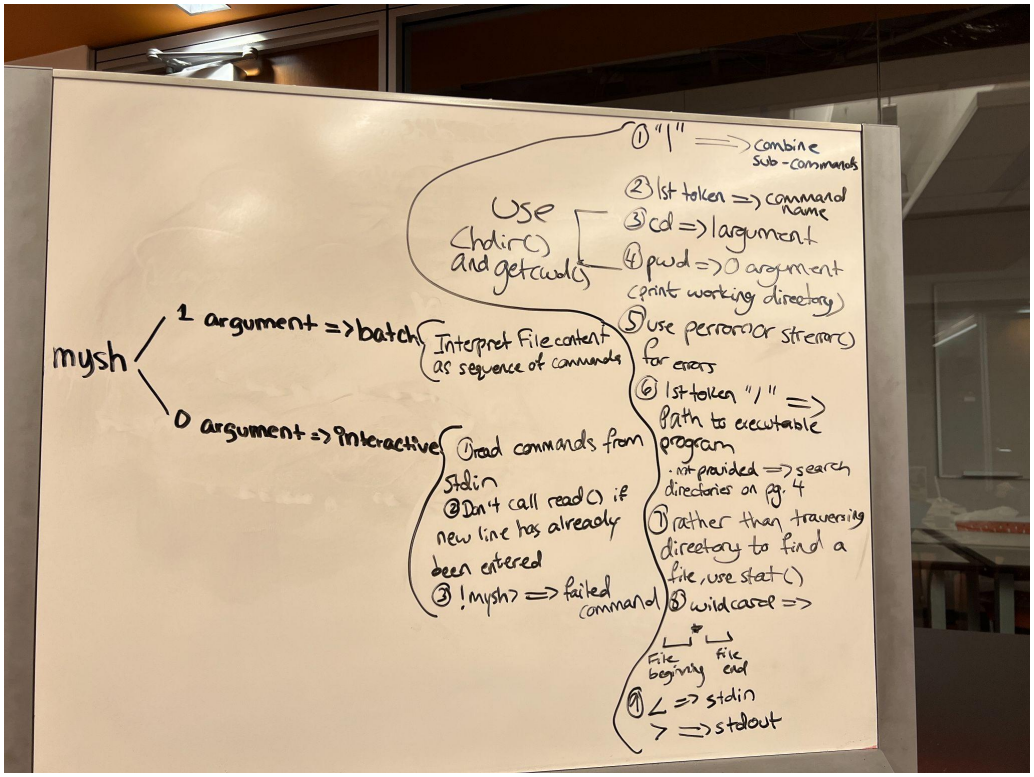
Project II: My Shell

By: Abhishek Thakare (ayt17) and Adhit Thakur (at1186)

Contents

This README discusses the processes behind the project “My Shell”, which involves implementing a simple command-line shell, similar to bash or zsh. This includes using the Posix stream IO, reading and changing the working directory, spawning child processes and obtaining their exit status, using dup2() and pipe(), and reading the contents of a directory. Our task is to implement all of these, and more, in order to fully mimic the command-line shell just as any other terminal. Below describes our test plan, proofs for the design properties, and other miscellaneous comments.



The plan for the project origiannlly started off with a base plan of tasks that needed to get done. There is the splitting up of the two modes, then within each mode, there are different implementations, but similar

executions. After implementing the commands that are built in and called using an executable, the respective files are parsed.

Test Plan

In order to complete this project, we went through each task one step at a time, by starting out large with the harder tasks and then getting into the smaller more specific challenges.

- Task one: parsing
  - For parsing, we utilized an array of arraylists. Each index in the array represents a subcommand, while each arraylist in the array represents the token list for the respective subcommand.

- Task two: executing commands in interactive mode
  - After generating the array of arraylists for the given command / commands, we looped through each arraylist and interpreted the commands.
  - The interpretation was done by looking at the token list, and then executing the respective command depending on the token located
    - An array of arguments was formed, with these being executed
- Task three: transitioning over to batch mode
  - For batch mode, the file was looped through using read(), new lines were identified, and commands were executed every new line that was encountered
- Task four: working with wildcards
  - For wildcards, we used the glob() function, and returned an adjusted argument list
  - This argument list would be then used as the new argument list in execv(), allowing for all necessary files to be included
- Task five: extensions
  - The extensions done were **multiple pipes** and **cd: home**
    - Multiple pipes were implemented by having a simple loop, and there being 3 cases
      - Case 1 is if the command is the beginning of the sequence, case 2 is if the command is at the end of the sequence, and case 3 is if the command is in the middle of the sequence

- Based on the case, STDIN, STDOUT, or both were set respectively

Test Programs

We are able to check if our code is working through the command line we created. By running specific commands that are not usual, we are testing our program to ensure that we are accounting for all edge cases. We will share our testing strategies for each of the commands (and more) that can be used in a shell down below:

- ❖ Parsing:
- ❖ Command: cd
  - ➢ Tests:
    - cd ~/Desktop
      - Testing wildcards
    - cd ..
      - Testing backwards traversal
    - cd anyPathThatIsNotInWorkingDirectory
      - Testing navigation
- ❖ Command: ls
  - ➢ Tests:
    - ls anyPathThatIsNotInWorkingDirectory
      - Testing navigation
    - ls
      - Testing current directory contents
- ❖ Command: echo
  - ➢ TestOne
    - echo anyText
      - Prints anyText to stdout
    - echo incorrectfileName
      - Returns error

- Command: cat
  - TestOne
    - cat program.c
      - Prints out the contents of program
    - cat testFile.txt
      - Prints out the contents of testFile
- Command: < & >
  - TestOne
    - fileOne.txt > fileTwo.txt
      - Puts the contents
- Command: |
  - TestOne
    - fileOne | fileTwo | fileThree
      - Tests standard input / output setting
    - fileOne < fileTwo | file
- Modes: interactive
  - TestOne
    - On error, mysh prints out an ! before the prompt
  -
- Modes: batch
  - TestOne
    - testFile.txt
      - This file traverses backwards through the directories, then prints the working directory, then changes the directory to a randomly assigned directory
    - testFileTwo.txt
      - This file tests multiple pipes in a command, and file redirections combined with pipes

Design Notes

The process of implementing this command-line shell was very long, and we had to make sure our code did not contain any bugs at any step along the way of our designing. For each step we worked on, our goal was to ensure that it was 100% correct and efficient, so we do not fall into later errors that can cause a catastrophe when debugging. Starting out with parsing, we knew this was one of the hardest parts of the project, so we wanted to start here. The parsing part involved us understanding how strncat works, and how a string needs to be initlaized to empty, since malloc allocates garbage to the different indices in a string.

With the implementation of commands, one of the primary errors that we encountered was a heap buffer overflow. This error occurs when an index of a declared object is accessed that is outside of the bounds of that object. We encountered this error many times when manipulating the argument string list and array of arraylists. Another big issue that we dealt with was memory leakage. While testing interactive mode, we did not experience any major memory issues, and went along with that. However, once we tested batch mode, we realized that there were many places where we did not free variables, and spent a large chunk of time looking at how these allocations were not freed. We were not able to finish this part of the project, but still got to understand how strcpy() works, and the nature of strings when performing array assignments.

Although the project currently does not perform the necessary functionalities optimally, those functionalities are still performed. For example, we were not able to implement the wildcard implementation into the actual execution of the commands, but when tested independently, this method worked. Furthermore, we were not able to perform in depth testing of the file redirects combined with piping, but tested each of these components individually. All in all, we feel liek some alternative strategies that could have been done is more memory checking, and making sure that the current laid out plan allows for optimal implementation in all aspects of the project.