CS 421 - Operating Systems
OS Simulator - Assignment 1: Shell

Note: The entire OS Simulator project is heavily influenced by the "Project Manual to Accompany "A Practical Approach to Operating Systems"" by Malcolm G. Lane and James D. Mooney as well as the paper "Operating Systems I: An Operating System Simulation Developed as Homework" by Ronald Marsh.

Dr. Mooney was one of my professors at WVU. In the late 80's, he co-authored a book on operating systems. It came with a project manual and some accompanying software. The book and manual are now out of print. While some text from the projects will be taken directly from the manual, I have heavily modified the project to be more modern and open-ended.

Part 1: Design and Programming

For the majority of the semester, you will be constructing an Operating System simulator. You are free to name your simulator whatever you want (it must be appropriate in polite company) and brand it however you wish.

Your first task is to design and implement a shell and some basic functionality. You should do your best to design your code so that it is open to change. You should investigate Kent Beck's Four Rules of Simple Design. Beck says that good code:

1. Passes all tests (**Note: I will not grade your assignment unless you have automated tests**)
2. Is clear, expressive, and consistent (The code expresses every idea that we need to express)
3. Duplicates no behavior or configuration (Says everything once and only once)
4. Uses minimal methods, classes, and modules (Has no superfluous parts)

It is a mistake to try to make assumptions about the future of the code, because those assumptions may not be true. Do not try to predict the future.

Since this is a simulator, it will be running on top of another operating system. You may choose any programming language for this project. It doesn't matter to me which OS you develop on or what test framework you use, as you will demo your projects for me in person. **It must run on a Raspberry Pi outside of an IDE**.

Your shell should be text based, similar to the Command Prompt or Linux Terminal. It must display a welcome message and supply the following functionality / commands:

1. Version: Display the software version number for your simulator
2. Date: Display or set the date
   You will need to see what facilities your underlying OS and programming language have for accessing the current date.
3. Directory: Print a list of all the files in the simulator's directory.
   At this point, you may assume that there is only one directory for the OS. This command may take some research.
4. History: Display a history of the last ten commands a user used.
5. Batch files: Run commands from a batch file or script
6. Aliasing: Allow the user to map different command names (i.e. renaming Exit to Quit)
7. Exit: Exit the simulator. You should ask the user if they are sure that they want to exit.

8. Help: Display help information for each command. Should be able to display a list of all commands present on the system.

Note: *Any time an entire output from your system, such a help file, will not fit on the screen entirely, you need to display the portion that fits on the screen and pause until the user asks for the rest (i.e. "press enter or click to continue"). This will require you to determine the screen size of your simulator from the start.*

You will need to add and delete functionality in the future, so design your system to make this as easy as possible. (**Hint: Is there some sort of Design Pattern that is useful for commands?**)

Code Requirements:

It is expected that you use good programming style and practices that were discussed in previous classes and outside resources like Code Complete. You should use good naming and commenting practices. **You should write your code as if the person who ends up maintaining it is a violent psychopath who knows where you live.**

User's Manual

Your simulator should be accompanied by a User's Manual which should contain at least the following sections:

1. Table of Contents
2. Overview of the OS
3. Summary of Commands
4. Detailed Description of Each Command
    a. Syntax (if applicable)
    b. Use
    c. Example
    d. Possible Errors
5. Summary of Error Messages
6. Index

Write the User's Manual so that your grandparents would be happy with it.

Technical Support Manual

A Technical Support Manual is also required for the project. This manual describes the program structure and how it operates. It should include the following information:

1. Table of Contents
2. Overview of the Program
    a. Paragraph(s) about the system, including principal elements
3. Program Structure
    a. UML class diagrams
    b. List of the contents in each file
        i. List the functions / data structures and a brief description of each
4. Description of each function
    a. Prototype
    b. Parameters - data type, description
    c. Return value, including returns when an error is encountered

    d. Description of what the function does
5. Description of Data Structures and Classes
    a. Use
    b. Attributes - list including data types and description
6. Global Variables (if any)
    a. List including data types and description
7. Cross References
    a. For each function, which function does it call, and which functions call it?
8. Index

This manual is intended for technical staff who must maintain your simulator. Ask the question: “would I be happy with the given responsibility to maintain this program using this manual?”

We will build on these two manuals as the project progresses.