Operating Systems Project Phase 1 Full Report

Nawal Ahmed & Jordan Hasty

CVM++ OS

Abstract & Introduction

This progress report is designed to detail the current progress on the operating system simulation, CVM++ OS. The simulation is design to simulate a low level operating system and is designed to run in any c++ debugger. Like any operating system, the goal of the OS is to provide a consistent and reliable method of accessing input/output devices. To run the program, simply run “main” in CodeLite or Visual Studio. The simulation is intended to accurately demonstrate how operating system modules interact with data and each other in the kernel.

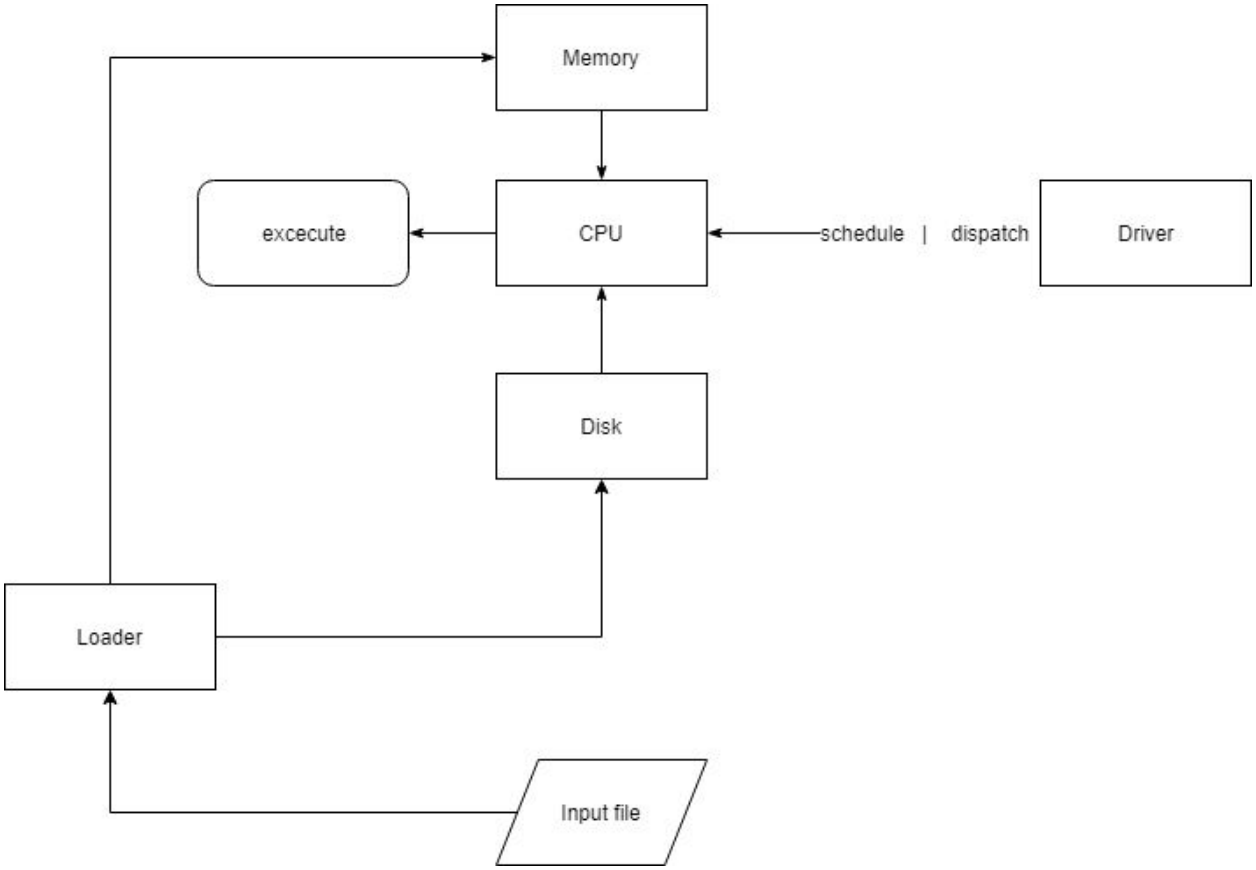
Design Approach

The program, written in C++, is designed to simulate the key components of an operating system in a virtual environment, just as it would in a regular non-simulated system.

To design the software, we initially referenced the project specification and block diagram. We made a few design decisions that were not included in the specification and ended up with the modules outlined in part 3. See the block diagram below for an overview of the interactions between the modules.

Our key goal during the design phase was to keep everything in separate modules. We found a few areas where the specification was not clear such as whether a function should have it’s own module or be contained within a different module. In some cases where it made sense, such as with helper functions, we implemented some functions inside of a module rather than in their own module.

Design Diagram:



Implementation Modules

All of the modules listed have their own properties and have their own specific functions that are used for the contribution of the entire operating system.

Currently, the project is made up of the following modules:

- CPU
    - The core module of the operating system. The CPU decodes instructions fetched from memory and performs the actions described by the instructions in the processes. This is designed to be able to run as several instances. Each instance shares a ready queue and memory.
- Disk
    - The disk simulates a hard drive. It contains a function for initialization, deletion, as well as a print function for debugging.
- Driver
    - This modules will call the loader to load job into memory and then assign them to a queue. Interprets commands from the OS and manages requests.
- Loader

    - The loader is responsible for reading the input file line by line. The module classifies each line as data or job and extracts meta-data, load jobs to memory, and data to the disk.
- Main
    - The main module initialized the program modules including the ram, disk, cpu, and loader. Currently, it also calls driver::disk to print the contents of the disk.
- Memory
    - This module simulates RAM in a computer. Data is in hexadecimal format and bundled into words. It contains basic functions for initializing and deleting itself, as well as printing the contents of itself.
- Memory Manager
    - This module helps find an empty space in memory to insert data.
- Utilities
    - This module helps parse raw data in the form of words into vectors that are pushed to an internal queue for easier processing.

Simulation Runs

During the runs of our simulation, we mainly tested our memory management paging system and loader functions. We found that programs were properly loaded into the disk memory when related functions were called. We also tested the CPU during its early stage of development and ensured that our fetching and decoding functions executed properly.

Data Results

[Insert Graph of data here]

Analysis

[Analysis of data]

Conclusion

From our simulation runs, we found that the modules which have been completed are functional. The only currently-non functional module is the CPU.

Appendix

Although the simulator is currently incomplete, a core dump file has been provided, though no programs have been executed. The core dump only contains the data of the first program loaded into memory which is allocated through paging. In this case, the program is allocated a

contiguous block of memory since all frames are empty and frames are allocated by finding the first available.