Operating Systems Project Phase 1 Report

Nawal Ahmed, Jordan Hasty, Tanner Jones, and Casey Sullivan

1. Abstract and Introduction

This progress report is designed to detail the current progress on the operating system simulation. The simulation is design to simulate a low level operating system and is designed to run in any c++ debugger. 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. We found that [insert findings].

2. Design Approach

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.

One of our goals during design was to keep everything in separate modules. We found a few areas where the specification was not clear 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.

3. Implementation Modules

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

- CPU
    - 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.
- 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.

The simulator is currently non-functional, so most of the modules are independent of each other.

4. Simulation

WIP

5. Data Results

WIP

6. Conclusions

From our simulation and data results, we found...

7. Demonstration

To be completed on demo day.