# Kernel Design

Luiz Medeiros

02/27/2015

Below is a desired design that attempts to follow the format shown in the example provided.

## Overview

We begin by considering that our general I/O will be from the following:

Monitor; Keyboard;

Main memory is 10meg.

Then, we have a disk drive, which should be accessed through a bus and have it's specific index.

Finally, and most importantly, four designated registers for our purposes. (A-D).

We then consider that we have 9 system calls. They are as follows:

1. Program Exit (with and without an error)
2. Accept one character of data from keyboard
3. Output one character of data to display
4. Create a file.
5. Open a file for input
6. Read the next byte from a file.
7. Close a file.
8. Request List Of Files in Current Directory
9. Display Current Directory Name

The idea is that we have the kernel, which is a program that runs with privileges and has direct access to the registers. This kernel, which we are devising will then provide these System calls for any requesting party that may call it.

We further assume that we have shell which is in privilege mode, where the user is able to input commands, which the kernel listens to. The Commands are as follows:



What follows, is an activity chart that comprises the initial high level functionality of the kernel:



I have mentioned a common stack that takes care of pushing and popping the requests for the system calls. The reason is because I reasoned that if we have separate threads and some type of security mechanism in place, we want to restrain direct access between CPU and general user.

The idea is that the Kernel will be constantly listening to that spot in memory. In addition to the system call, there will be another set of bytes which tells where the program comes from (memory). The Kernel picks up this system call information and then executes it. While execution, it places the vector from the system call in Register A.

Let me know what you think of this approach. I can easily change it so instead of particular common place in memory it is placed directly to Register A.

# Functions

## Constructors:

The contructors will take care of different scenarios, such as checking if registers are cleared and starting the different peripherals.

## Destructors:

The destructor will clean the registers and shut down peripherals. It will prepare the system for shut down.

## Public Methods:

The system call running on the separate thread checking the system call stack will be a public method.

## Private Methods:

The private methods will be where each regular function that only pertains to the Kernel will be defined.

## Accessors:

Methods to get the values of each separate register.

## Mutators:

These will be to manipulate the values within the registers availabe to us. In addition, it will also manipulate the values in the common stack.

## System Calls:

All system calls will be private methods. Here we will defined each call, from 0 to 8.

## Errors:

The errors will all be thrown into a log file and recorded with time and information regarding request origins .

## Debug:

Here we will take care of placing test cases so to debug and test the program.

## 