Course Project

Computer Simulator

Quin'darius Lyles-Woods
qlyleswo@students.kennesaw.edu

Operating Systems
Professor Emadad Ahmed
Section W01
3502

Bachelors of Computer Science
Kennesaw State University
1100 South Marietta Pkwy SE
Marietta, GA 30060
November 29, 2021

Contents

0.1 Introduction

Within this report will be a write up of the construction of an Operating System Simulator. Throughout I will describe the components necessary for such an simulator. There are approximately 13-15 individual modules that make up such a program and I will shew to explain them all within their proper specifications.

Part I

Programs

0.2 Memory System

The memory system varies from computer to computer but in the particular program we will have the following.

- Simulated Registers
- Simulated Ram (To be called memory)
- Program to get data from the ram efficiently
- Program to convert between hex values

0.3 Driver

The Driver is a fairly simple piece of the operating system. The Driver is meant to be the main thread of the this simulated operating system. Its job is this operating system will be to call the loader into the program. This will allow the operating system to load user programs in a program-file. Typically the driver will be the software that loads the software for our **hardware**. Alas, since we are simulating the software on top of software it will be written to the program file.

0.3.1 Pseudocode

```
Driver
        {
                loader();
                loop
                        scheduler();
                        dispatcher();
                        CPU();
                        waitforinterrupt();
                endloop;
        }
```

Within the pseudocode there are a lot of things that are not defined. This is because they will be described a little bit later on in the report. Although you can view all the components within the **Content** section of this paper.

0.4 Loader

The loader has another simple but crucial role with the simulated operating system. The loader will load all the needed program files that are in the program

0.4.1 Pseudocode

```
while (scan != EOF) do
{
        read-file();
        program-attributes();
        hex-instructions();
}
```

0.5 Scheduler

The scheduler is a program that loads programs from the disk into the **RAM**. The scheduler does this out of priority of the program. The scheduler is able to load one or multiple programs into the **memory** or **RAM**.

0.6 Dispatcher

The dispatcher, like that of the loader, is a very simple one. The job of the dispatcher is to assign the simulated CPU jobs. Our Operating System will call the CPU to execute such jobs.

0.7 Memory

The memory is the **RAM** that we have been talking about throughout the report so far. Of course the basic function of ram is to be Random-Access-Memory. It would be just that within our operating system as well. Taking the program given to it be the scheduler.

0.8 Effective Address

The effective address will help us identify the actual address of the unit that is being called. So you can say maybe the CPU can be one of the units.

0.8.1 Pseudocode

```
function direct-addressing( int num) => EA
{
        return C(base-reg)+D;
}

function indirect-addressing( int num) => EA
{
        return C(base-reg)+C(I-reg)+D;
}
```

0.9 Fetch

The part of the operating system will get information from the **RAM**. This is all dependent on the CPU's program counter not to be confused with the acronym PC. This program will execute before the effective address if this helps the reader with the order of operations.

0.10 Decode

The part of the operating system called decode will do just that. The decode part of the operating system will be working in concert with the CPU. This will have to be loaded correctly in the registers or whatever data-structures.

0.11 Execute

The execute method of the operating system will be a switch loop for the CPU. The execute method will also be incrementing the program counter that we were discussing earlier. This part of the operating system will also help the DMA-channel as well.

Part II

The CPU

0.12 DMA-Channel

Speaking of the devil, the DMA-Channel also works in concert with the CPU. The DMA-Channel will work with the CPU as it called read and write from the ram to the Input and Output instructions. Below we will have some examples of these functions.

0.12.1 Pseudocode

```
function read()
{
        char character;
        return character, next(p—rec),buf[next−io]);
}
function write()
{
        char character;
        return character, next(p—rec),buf[next−io]);
}
```

The character is the DMA controller. The p-rec is the **RAM** address of where the physical data is going to be ”transferred”. The buf is the startign address of the **RAM** buffer.

```
DMA()
{
        loop
        {
                switch(int type)
                {
                        case−one: read(character, next(p—rec),buf[next−io]));
                        case−two: write(character, next(p—rec),buf[next−io]));
                }
                next−io = next−io + 4;
        }
        signal(ComputeOnly)
}
```

0.13 ComputerOnly

This section will define the algorithm that will be the instruction cycle with a dynamic relocation of the program.

Part III

Multiprocessing Architecture

0.14 Multiprocessor Architecture

So for the project we are designing this architecture while being able to be a distributed and parallel. This will be done by getting the CPU the we designed in the fist part of the report. We will then to make it a multiprocessing and distributed and paralleled architecture. The multiprocessor dispatcher or the m dispatcher, is going to extend the single CPU dispatcher and use that to scale off the original CPU. This allows the separation of the **RAM** We need to make sure that we have all the data separated because many things have depend on having a true data source.

0.15 Multiprocessor Memory Management

Since we will be splitting the memory across all of how ever many CPU's we want to have in the simulator. With this specification we will need to have the data be appropriately separated. This is done for the simplicity because the other solutions could be their own paper.

0.16 Multiprocessor Program Cache

Unlike the memory the cache needs to a unified program for the CPU. This is needed because the m-dispatcher assigns a process to the CPU. A CPU will obtain task in this distributed and paralleled operating system by using the shared cache memory. When the CPU executes the task for the operating system a note will be made. The note will tell where the address is within the cache memory. Then that is when the information is sent to the Memory Management System. Within the memory management system the data will be split between the CPU's.