CS 446 - OS Simulator Specification

Designed by: Cayler Miley, Michael Leverington

Revised: August 2016

1 Introduction

This set of programming assignments is designed to materialize all of the major operating systems concepts in the CS 446 course by allowing you to make design decisions during development of an operating system. These assignments will increase your understanding of operating systems and incorporate common aspects of industry and/or advanced academia.

Over the course of the semester, you will complete one introductory assignment and four simulation assignments. After the completion of the fourth simulation assignment, you will have simulated the core components of a modern day operating system. Each of the assignments build tremendously upon the previous assignment, thus it is advantageous for you to design each assignment with all future assignments in mind. This will significantly reduce your workload in the long run.

This document may change throughout the semester and suggestions may be made for any changes one week prior to the assignment due date. This is however at the instructor's discretion.

All of the simulation assignments must be completed using C or C++. A total of 5% extra credit is offered to all students (i.e., graduate and undergraduate) who complete any single assignment in C. This means it must be completely written in C (including any header files) and compile with `gcc`. All programs require the use of a make file.

2 Assignment Calendar

	Assignment 1	Assignment 2	Assignment 3	Assignment 4	Assignment 5
Due Date	Sep. 12	Sep. 26	Oct. 17	Nov. 7	Nov. 28

3 Simulator Description

3.1 Expectations

A rubric will be provided for each program. In addition to the rubric, the following will be expected of each program throughout the simulation assignments:

- since you will have an overview of all of the programs, it will be worth your time to consider the subsequent phases as you develop the first program(s); if you have an overlying strategy from the beginning, extending each program will not be difficult
- you may work with any number of fellow students to develop your program design, related data structures, algorithmic actions, and so on for each phase. If you do, you must note which students with whom you worked in your upload text on WebCampus; this is for your protection
- that said, once you begin coding each phase, you may not discuss, or work, with anyone on your programming, strategy(s), debugging, and so on; it will behoove you to make sure you have a high-quality design developed prior to beginning your coding process

- all programs must be eminently readable, meaning any reasonably competent programmer should be able to sit down, look at your code, and know how it works in a few minutes. This does not mean a large number of comments are necessary; your code itself should read clearly. You are also required to follow a documentation format in your code (this must simply follow a readable and consistent format, if you have no format to follow use Doxygen). You will be graded on the readability of your code and difficulty in reading your code may result in a zero
- the program must demonstrate all the software development practices expected of a 400- (or 600-) level course. For example, all potential file failures must be resolved elegantly, any screen presentation must be of high quality, any data structures or management must demonstrate high quality, supporting actions and components must demonstrate effective modularity with the use of functions, and so on. This means your code should be tested for failure and handled accordingly, including informing the use of the errors
- you may use any I/O libraries or classes as needed, but any other classes must be created by you. In addition, you may use POSIX/pthread operations to manage your I/O operations but you may not use previously created threads such as timer threads (e.g., sleep, usleep, etc.). Additionally, you are free to use basic error libraries but the errors must be handled by you.
- for each programming assignment, each student will upload the program files through WebCampus. The file for each student must be tarred and zipped in Linux as specified below, and must be able to be unzipped on any of the ECC computers include any and all files necessary for the operation of the program. Any extraneous files such as unnecessary library or data files will be cause for credit reduction. The format for submission is `Sim0X_<LastNameFirstName>.tar.gz` where X represents the specific project number, or as an example, `Sim01_SmithJohn.tar.gz`.
- all programs must run on the computers in the ECC with no errors or warnings.

3.2 Meta-Data

All assignments will use meta-data to house the information required to run each simulation. The meta acts as the set of instructions for your simulation to run on. The meta-data codes are as follows:

- S - Operating **S**ystem, used with `start` and `end`
- A - Program **A**pplication, used with `start` and `end`
- P - **P**rocess, used with `run`
- I - used with **I**nput operation descriptors such as `hard drive`, `keyboard`
- O - used with **O**utput operation descriptors such as `hard drive`, `monitor`
- M - **M**emory, used with `allocate`

The meta-data descriptors are as follows:

- `end, hard drive, keyboard, printer, monitor, run, start, allocate`

The meta-data will always follow the format:

`<META DATA CODE>(<META DATA DESCRIPTOR>)<NUMBER OF CYCLES>`

For example, an input keyboard operation that runs for 13 cycles would look like the following:

`I(keyboard)13`

Below is an example meta-data file:

```
Start Program Meta-Data Code:
S(start)0; A(start)0; P(run)11; P(run)9; P(run)12;
P(run)9; P(run)11; P(run)8; P(run)14; P(run)14; P(run)12;
P(run)12; P(run)6; P(run)8; P(run)9; P(run)6; P(run)14;
P(run)15; P(run)12; P(run)9; P(run)6; P(run)5; A(end)0;
A(start)0; P(run)6; P(run)6; P(run)9; P(run)11; P(run)13;
P(run)14; P(run)5; P(run)7; P(run)14; P(run)15; P(run)7;
P(run)5; P(run)14; P(run)15; P(run)14; P(run)7; P(run)14;
P(run)13; P(run)8; P(run)7; A(end)0; A(start)0; P(run)6;
P(run)10; P(run)13; P(run)9; P(run)15; P(run)6; P(run)13;
P(run)11; P(run)5; P(run)6; P(run)7; P(run)12; P(run)11;
P(run)6; P(run)8; P(run)10; P(run)5; P(run)8; P(run)9; P(run)7;
A(end)0; S(end)0.
End Program Meta-Data Code.
```

3.3 Configuration

Each assignment will use a configuration file to set up the OS simulation for use. This will specify the various cycle times associated with each computer component, memory, and any other necessary information required to run the simulation correctly. All cycle times are specified in milliseconds. For example, if the hard drive cycle time is 50 ms/cycle and you must run for 5 cycles, the hard drive must run for 250 ms. These will be used by a timer to accurately display timestamps for each OS operation. You must use an onboard clock interface of some kind to manage this, and the precision must be to the microsecond level. The configuration will need to be read in prior to running any processes. The configuration file will be key to setting the constraints under which your simulation will run.

Below is an example configuration file:

```
Start Simulator Configuration File
Version/Phase: 2.0
File Path: Test_2e.mdf
CPU Scheduling Code: SJF
Processor cycle time (msec): 10
Monitor display time (msec): 20
Hard drive cycle time (msec): 15
Printer cycle time (msec): 25
Keyboard cycle time (msec): 50
Log: Log to Both
Log File Path: logfile_1.lgf
End Simulator Configuration File
```

3.4 Running the Simulator

When running the simulator you will be required to input a single configuration file (extension `.cnf`). You will run the simulator from the command line similar to the following:

```
./sim0X config_1.cnf
```

The name of the assignment must be the simulator number. Many configuration files should be used to test your program, which you may modify for testing purposes as you see fit.

3.5 Turning in Assignments

All assignments will be turned into WebCampus. You must submit a zipped `.tar` archive as specified above. Inside the archive there should only be the files required to run the simulator (e.g., all source files, all header files). No resource files are allowed. Assignments will be due at noon on Wednesdays, specified by the calendar. **Late assignments will not be accepted**.

4 Assignment 1

4.1 Description

Assignment 1 tests your knowledge of strings, reading from files, and data structures. This assignment allows you to create a library of functions for use in later projects. Keep in mind that you will be using many of the functions you create in this phase of the simulator in future phases.

4.2 Specification

You will be given an arbitrary number of configuration files to read into your simulation program. Each configuration file will contain a version number (from 1-4), which will change the content of the configuration file and must be handled accordingly. Along with the configuration files, a number of test meta-data files will be given. You will need to read in the information on each file and display the metrics for them. The grader should be able to easily read your code, and run your program using the commands: `make` and `./Sim01 <CONFIG_FILE>`. Name your file `Sim01` for this assignment and include only the makefile and any source or header files in your gzipped archive. Refer to the Expectations Section for how to submit your archive to Webcampus.

For the configuration file you will:

- Output all of the cycle times in the format below
- Log to a file/monitor as specified
- Read from the meta-data file specified
- Log to the specified file location (ONLY if logging to the file)

For the meta-data file you will:

- Output each operation and the total time for which it would run

Additionally you will be required to:

- handle file failures and typos
- handle meta-data and configuration typos
- correctly identify and handle missing data (such as a missing processor cycle time or a time of 0)
- utilize a (set of) data structure(s) to organize information and compute information through the data structure
- open and close any files only once (for reading/writing only)
- document EVERY function and data structure used throughout the program (anyone should look at your code and be able to read it as well as a book, you can find examples of code documentation by running a search on it)
- specify the configuration file as a command line argument
- use a makefile

As a reminder, all of the functions created in this assignment will be used for your future assignments and are designed to help you easily transition from understanding data structures to actually applying them in the context of an operating system.

4.3 Graduate Requirements

There are no additional graduate requirements for this project.

4.4 Output

```
Configuration File Data
Processor = 10 ms/cycle
Monitor = 20 ms/cycle
Hard Drive = 15 ms/cycle
Printer = 25 ms/cycle
Keyboard = 50 ms/cycle
Memory = 30 ms/cycle
Logged to: monitor and logfile_1.lgf

Meta-Data Metrics
P(run)11 - 110 ms
M(allocate)2 - 60 ms
O(monitor)7 - 140 ms
I(hard drive)8 - 120 ms
O(printer)20 - 500 ms
P(run)6 - 60 ms
```

4.5 Grading

All grading will be completed with the given rubric. The rubric is very general, thus it would be best to find and list all of the requirements you feel you must meet in the expectations section and assignment section. You will be given 3 configuration and test files for your use. Your program must work correctly with these files and any files the grader(s) may create.

5 Assignment 2

5.1 Description

This will be the first "phase" of your operating systems simulator. Phase 1 of the simulator will allow you to run a single program. You are tasked with running a stand-alone program through your simulator using many of the operations seen previously in test files.

6 Assignment 3

6.1 Description

Assignment 3 will test your knowledge of multiprogramming and resource management. You will run multiple processes and handle their use of resources through mutexes and semaphores.

7 Assignment 4

7.1 Description

Assignment 4 will allow you to increase throughput in your simulator by implementing scheduling algorithms and by handling basic memory operations.

8 Assignment 5

8.1 Description

Assignment 5 will complete your OS simulator by allowing you to preempt processes through interrupts and use virtual memory. You will also handle basic security procedures.

9 Notes and Revisions

Any questions or concerns may be emailed to cmiley@nevada.unr.edu. Let me know if any typos are found and I will update them with the following revision.

9.1 Version 1.0 - August 22 2016

Updated assignment descriptions and specifications.

9.2 Version 1.1 - August 30 2016

Updated expectations section and included descriptions for all 5 assignments.