CS 446/646 - OS Simulator Specification

Designed by: Cayler Miley, Michael Leverington Revised by: Vineeth Rajamohan

Revised: August 2019

Contents

1 Introduction

This set of programming assignments is designed to materialize all of the major operating systems concepts in the CS 446/646 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 three simulation assignments. Graduate students will have to complete all the assignments in C. After the completion of all the simulation assignments, 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++ programming language for undergraduate students and C for graduate students. All programs require the use of a make file. ALL PROGRAMS MUST RUN AND COMPILE IN THE ECC, OTHERWISE YOUR GRADE WILL RESULT IN A ZERO. Your submissions must include source code, readme, and makefile.

2 Simulator Description

2.1 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 programs; 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. Failure to do so will result in zero points.
- 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. If you would like an example on documentation, search "code documentation" in a search engine. You will be graded on the readability of your code and difficulty in reading your code may result in a reduced grade
- 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 user of the errors encountered in your simulator
- 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 or 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` Points will be deducted for incorrect file name format.
- all programs must run on the computers in the ECC with no errors or warnings.

2.2 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`, `mouse`
- O - used with **O**utput operation descriptors such as `hard drive`, `monitor`, `printer`
- M - **M**emory, used with `block`, `allocate`

The meta-data descriptors are as follows:

- `begin, finish, hard drive, keyboard, mouse, monitor, run, allocate, printer, block`

The meta-data will always follow the format:

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

Below is an example meta-data file:

```
Start Program Meta-Data Code:
S{begin}0; A{begin}0; P{run}11; I{hard drive}9; I{keyboard}12;
I{mouse}9; O{hard drive}11; O({monitor}8; O{printer}14; M{block}14; M{allocate}12;
A{finish}0; S{finish}0.
End Program Meta-Data Code.
```

2.3 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. Log File Path is the name of the new file which will display the output. 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
Projector cycle time {msec}: 25
Processor cycle time {msec}: 10
Keyboard cycle time {msec}: 50
Monitor display time {msec}: 20
Scanner cycle time {msec}: 10
Hard drive cycle time {msec}: 15
Log: Log to Both
Log File Path: logfile_1.lgf
End Simulator Configuration File
```

2.4 Running the Simulator

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

```
./simX config_1.conf
```

X represents the assignment number. Many configuration files should be used to test your program, which you may modify for testing purposes as you see fit.

2.5 Turning in Assignments

All assignments will be turned into WebCampus. You must submit a zipped `.tar.gz` 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. **Late assignments will not be accepted**.

3 Assignment 1

3.1 Description

Assignment 1 tests your knowledge of strings, reading from files, and data structures. This assignment allows you to create a library of functions/classes for use in later projects. Keep in mind that you will be using many of the functions/classes you create in this phase of the simulator in future phases. Assignment 1 is designed as a data structures problem, and is not a part of the official simulator.

3.2 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. **Note that the configuration file used for testing will be the same as the example file shown below.** 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 and run your code. Include only the makefile and any source or header files in your zipped 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 (e.g., `Ohard drive5` would run for $5 \times$ *hard drive cycle time*)

Additionally you will be required to:

- handle file failures and typos (this includes a missing file, an incorrect file path, a typo in the file name, etc.)
- handle meta-data and configuration typos (this includes misspellings in the configuration or meta data file, incorrect characters such as a colon instead of a semi-colon, etc.)

- 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 like a book, you can find examples of code documentation by running a search on it)
- The configuration file must be taken as a command line argument during code execution.
- 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.

3.3 Example Configuration File

```
Start Simulator Configuration File
Version/Phase: 1.0
File Path: Test_1a.mdf
Monitor display time {msec}: 20
Processor cycle time {msec}: 10
Mouse cycle time {msec}: 25
Hard drive cycle time {msec}: 15
Keyboard cycle time {msec}: 50
Memory cycle time {msec}: 30
Printer cycle time {msec}: 10
Log: Log to Both
Log File Path: logfile_1.lgf
End Simulator Configuration File
```

3.4 Example Input

```
Start Program Meta-Data Code:
S{begin}0; A{begin}0; P{run}11; M{allocate}2;
O{monitor}7; I{hard drive}8; I{mouse}8; O{printer}20;
P{run}6; O{printer}4; M{block}6;
I{keyboard}17; M{block}4; O{printer}8; P{run}5; P{run}5;
O{hard drive}6; P{run}18; A{finish}0; S{finish}0.
End Program Meta-Data Code.
```

3.5 Example Output

```
Configuration File Data
Monitor = 20 ms/cycle
Processor = 10 ms/cycle
Mouse = 25 ms/cycle
Hard Drive = 15 ms/cycle
```

```
Keyboard = 50 ms/cycle
Memory = 30 ms/cycle
Printer = 10 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
I{mouse}8 - 200 ms
O{printer}20 - 200 ms
P{run}6 - 60 ms
O{printer}4 - 40 ms
M{block}6 - 180 ms
I{keyboard}17 - 850 ms
M{block}4 - 120 ms
O{printer}8 - 80 ms
P{run}5 - 50 ms
P{run}5 - 50 ms
O{hard drive}6 - 90 ms
P{run}18 - 180 ms
```

4 Assignment 2

4.1 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.

4.2 Specification

You are required to run a single program through your simulator. This must be timestamped for each operation start and completion. You are also required to use a PCB(process control block) to update the state of your program: `START, READY, RUNNING,WAITING, EXIT`. Depending on what is specified by the given configuration file, all operations must be printed to the screen, a file, or both. Lastly, you must allocate memory for the single program .You will notice that there is a new line in the configuration file specifying system memory. You will be required to take in the memory of the system (in kbytes, Mbytes, or Gbytes) convert to kbytes if needed, randomly generate a number and convert that number to hexadecimal for this project.

You will be required to:

- use a 5-state PCB to change the state of your process
- use a timer to complete every operation in real time as well as timestamp the start and end of each operation (more details available at the end of the document)
- use dedicated threads for each I/O operation (other than the pthread library you may NOT use any other thread libraries)
- elegantly handle all errors including typos, file failures, missing data, etc.

Since at this point all the operations are linear one way to use the PCB to change the state of your process is:

- Have a class named PCB with an integer named processState.
- Assign integer values for start, ready, running, wait, exit.
- declare an PCB object in the main.
- at the correct instance assign the appropriate state.

Use the system timer to print the time at the start and the end of every operation. The only requirement (at this time) for the memory function is that the random value is an unsigned int which will be printed as a hexadecimal address.

4.3 Timer Usage

You will be required to use a timer to accurately timestamp all of your simulator's actions. This means that a projector operation running for 6 cycles will have to physically run for $6 \times$ *projector cycle time*. The simulator must output a timestamp at the beginning of the projector operation and another timestamp at the end of the operation. You are required to use the system clock. The timer itself must be measurable to the microsecond level and accurate to the millisecond level. For example, the output below should run for 7.387 seconds, outputting the correct timestamps for each operation. In addition, you are required to create your own timer thread to count down from a particular milliseconds to 0. You need to create a wait function to perform the count down. For example, a process is expected to run a projector operation for 7387 milliseconds, your simulator will set the timer to 7387 and the timer will automatically counting down to 0 by using the system clock. **You need to output after every operation is been completed.**

4.4 Thread Usage

You are required to complete each input and output operation (designated by I and O respectively) by creating a new thread for the operation and waiting for it to complete. The threads must only be used for I/O operations to best simulate the hands-off role of an OS in controlling external devices. For documentation on thread usage see the pthread man page or the POSIX tutorial at: https://computing.llnl.gov/tutorials/pthreads/.

4.5 Readme

The readme file for this project must contain instruction/commands on how to compile and run your code. In addition, you must also mention the file names and the lines number were you have created the threads and the PCB.
NOTE: The values mentioned in the example output may not represent the correct values. They are just to show the output format.

4.6 Example Configuration

```
Start Simulator Configuration File
Version/Phase: 2.0
File Path: Test_1a.mdf
Monitor display time {msec}: 20
Processor cycle time {msec}: 10
Mouse cycle time {msec}: 25
Hard drive cycle time {msec}: 15
Keyboard cycle time {msec}: 50
Memory cycle time {msec}: 30
Printer cycle time {msec}: 10
```

```
System memory {kbytes}: 1024
Log: Log to Both
Log File Path: logfile_1.lgf
End Simulator Configuration File
```

4.7 Example Input

```
Start Program Meta-Data Code:
S{begin}0; A{begin}0; P{run}11; M{allocate}2;
O{monitor}7; I{hard drive}8; O{printer}20;
P{run}6; O{printer}4; M{block}6; I{keyboard}17;
M{block}4; A{finish}0; A{begin}0; P{run}5; P{run}5; O{hard drive}6;
P{run}18; A{finish}0; S{finish}0.
End Program Meta-Data Code.
```

4.8 Example Output

```
0.000001 - Simulator program starting
0.000051 - OS: preparing process 1
0.000053 - OS: starting process 1
0.000055 - Process 1: start processing action
0.132061 - Process 1: end processing action
0.132063 - Process 1: allocating memory
0.182066 - Process 1: memory allocated at 0x00000010
0.182069 - Process 1: start monitor output
0.392073 - Process 1: end monitor output
0.392074 - Process 1: start hard drive input
0.536077 - Process 1: end hard drive input
0.536079 - Process 1: start printer output
5.536081 - Process 1: end printer output
5.536085 - Process 1: start processing action
5.608088 - Process 1: end processing action
5.608089 - Process 1: start printer output
6.608094 - Process 1: end printer output
6.608097 - Process 1: start memory blocking
6.758099 - Process 1: end memory blocking
6.758101 - Process 1: start keyboard input
6.843104 - Process 1: end keyboard input
6.843106 - Process 1: start memory blocking
6.943109 - Process 1: end memory blocking
6.931074 - OS: removing process 1
6.941181 - OS: preparing process 2
6.942381 - OS: starting process 2
6.943110 - Process 2: start processing action
7.003114 - Process 2: end processing action
7.003116 - Process 2: start processing action
7.063119 - Process 2: end processing action
7.063127 - Process 2: start hard drive output
7.171130 - Process 2: end hard drive output
7.171134 - Process 2: start processing action
7.387137 - Process 2: end processing action
```

```
35 7.387158 - OS: removing process 2
36 7.387433 - Simulator program ending
```