CS 446/646 - OS Simulator Specification

Revised: January 2017

Contents

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 points can be earned for each assignment based on your contribution to the discussion thread in WebCampus. Contribution can either be pointing out any errors in the assignment or responding to fellow student's questions. Note: Do not share any code in the discussion thread if so you will not be awarded any extra points. 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

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 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 <u>your</u> 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. 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.). You may use the C standard template library and C standard string library. 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. To remotely access the ECC (if you wish to test your programs) you can SSH to the ecc, copy your files, and run your files. Instructions for how to do this are listed at: `https://source2.cse.unr.edu/w/cse/student_resources/`

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`, `speaker`
- M - **M**emory, used with `block`, `allocate`

The meta-data descriptors are as follows:

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

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.
```

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
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
Mouse cycle time (msec): 10
Speaker 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:

```
./simOX config_1.conf
```

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.

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 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. 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. 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 (e.g., `O(hard drive)5` 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)
- 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.

3.3 Example Configuration File

```
Start Simulator Configuration File
Version/Phase: 1.0
File Path: Test_1a.mdf
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
Memory cycle time (msec): 30
Mouse cycle time (msec): 10
Speaker cycle time (msec): 20
Log: Log to Both
Log File Path: logfile_1.lgf
End Simulator Configuration File
```

3.4 Example Input

```
Start Program Meta-Data Code:
S(start)0; A(start)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(speaker)8; P(run)5; P(run)5;
O(hard drive)6; P(run)18; A(end)0; S(end)0.
End Program Meta-Data Code.
```

3.5 Example 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
Mouse = 10 ms/cycle
Speaker = 20 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 - 80 ms
O(printer)20 - 500 ms
P(run)6 - 60 ms
O(printer)4 - 100 ms
M(block)6 - 180 ms
I(keyboard)17 - 850 ms
M(block)4 - 120 ms
O(speaker)8 - 160 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 using the function given through WebCampus. 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) and call the given memory function with the number of kbytes.

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 the given memory allocation function and display the location at which your newly allocated memory resides
- use threads for each I/O operation (you may NOT use threads created by a library, you must create the threads yourself)
- 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 struct named PCB with an integer named processState.
- Assign integer values for start, ready, running, wait, exit at the beginning of your Sim02.cpp.
- 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. I have provided the memory function in C. If you are doing the project in C++, I would recommend modifying them for your purpose. The only requirement (at this time) for the memory function is that the returned 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 printer operation running for 6 cycles will have to physically run for 6 × *printer cycle time*. The simulator must output a timestamp at the beginning of the printer 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.

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 Example Configuration

```
Start Simulator Configuration File
Version/Phase: 2.0
File Path: Test_2a.mdf
Processor cycle time (msec): 12
Monitor display time (msec): 30
Hard drive cycle time (msec): 18
Printer cycle time (msec): 250
Keyboard cycle time (msec): 5
Memory cycle time (msec): 25
System memory (kbytes): 1024
Log: Log to File
Log File Path: logfile_1.lgf
End Simulator Configuration File
```

4.6 Example Input

```
Start Program Meta-Data Code:
S(start)0; A(start)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; P(run)5; P(run)5; O(hard drive)6;
P(run)18; A(end)0; S(end)0.
End Program Meta-Data Code.
```

4.7 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 0x00000000
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
6.943110 - Process 1: start processing action
7.003114 - Process 1: end processing action
7.003116 - Process 1: start processing action
7.063119 - Process 1: end processing action
7.063127 - Process 1: start hard drive output
7.171130 - Process 1: end hard drive output
7.171134 - Process 1: start processing action
7.387137 - Process 1: end processing action
7.387158 - OS: removing process 1
7.387433 - Simulator program ending
```

5 Assignment 3

5.1 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.

5.2 Specification

You will be required to implement a resource management system. You will use your own implementation of mutex locking and semaphores to manage the quantity of resources in your program. For all resources except the resources specified in the configuration file, you may assume their quantity to be one. The only resources you need to manage are the input and output resources. For example, when the keyboard is accessed you must manage the use of a keyboard in your program in a creative way. Since the keyboard is an example of a resource that cannot be accessed while it is being used, you need to design a mutex lock to guarantee that no other threads or processes could access it. As nothing is running concurrently (but will be in later assignments) it is in your best interest to design this system with concurrency in mind. This assignment does not require the use of resource management to function, but will ease your time spent programming for the final assignment.

In addition to designing a resource management system, you are also required to design a memory function that allocates memory for your simulator. It must resemble the memory function given in Assignment 2, but you must find a memory address at the beginning of every block (specified once again by the configuration file). For example, the first memory access should output the address of 0x00000000. The next memory access should output the starting address of the next block. If/when you run out of blocks, the function should start again at the first block (0x00000000). You should output each address in 32-bit hexadecimal, using an unsigned int.

Note: The times for each operation should equate to the physical time for each discrete operation and the corresponding cycle time. Here I have placed an "X" to indicate that the time should be calculated by you.

For each operation, the following should be considered:

- All `I` and `O` operations must be threaded

- All `M(allocate)` should have a hexadecimal address returned (within the constraints of the total memory given by the configuration)
- Each resource (`printer`, `keyboard`, etc.) should use mutex locks and semaphores to control access from other simulator operations
- The `M(block)` operation doesn't need to be handled other than running it for the allotted time until later assignments
- The quantity value for printer and hard drive must be printed and your code should reset back to 0 after reaching the maximum quantity value.

5.3 Example Configuration

```
Start Simulator Configuration File
Version/Phase: 3.0
File Path: Test_3a.mdf
Processor cycle time (msec): 5
Monitor display time (msec): 22
Hard drive cycle time (msec): 150
Printer cycle time (msec): 550
Keyboard cycle time (msec): 60
Memory cycle time (msec): 10
System memory (kbytes): 2048
Memory block size (kbytes): 128
Printer quantity: 4
Hard drive quantity: 2
Log: Log to File
Log File Path: logfile_1.lgf
End Simulator Configuration File
```

5.4 Example Input

```
Start Program Meta-Data Code:
S(start)0; A(start)0; P(run)11; M(allocate)2;
O(monitor)7; I(hard drive)8; O(printer)20;
M(allocate)4; O(printer)6; M(allocate)3; I(hard drive)7;
O(hard drive)2; O(hard drive)16; M(allocate)4;
P(run)6; O(printer)4; M(block)6; I(keyboard)17;
M(block)4; P(run)5; P(run)5; O(hard drive)6;
P(run)18; A(end)0; S(end)0.
End Program Meta-Data Code.
```

5.5 Example Output

```
X.XXXXXX - Simulator program starting
X.XXXXXX - OS: preparing process 1
X.XXXXXX - OS: starting process 1
X.XXXXXX - Process 1: start processing action
X.XXXXXX - Process 1: end processing action
X.XXXXXX - Process 1: allocating memory
```

```
X.XXXXXX - Process 1: memory allocated at 0x00000000
X.XXXXXX - Process 1: start monitor output
X.XXXXXX - Process 1: end monitor output
X.XXXXXX - Process 1: start hard drive input on HDD 0
X.XXXXXX - Process 1: end hard drive input
X.XXXXXX - Process 1: start printer output on PRNTR 0
X.XXXXXX - Process 1: end printer output
X.XXXXXX - Process 1: allocating memory
X.XXXXXX - Process 1: memory allocated at 0x00000080
X.XXXXXX - Process 1: start printer output on PRNTR 1
X.XXXXXX - Process 1: end printer output
X.XXXXXX - Process 1: allocating memory
X.XXXXXX - Process 1: memory allocated at 0x00000100
X.XXXXXX - Process 1: start hard drive input on HDD 1
X.XXXXXX - Process 1: end hard drive input
X.XXXXXX - Process 1: start hard drive output on HDD 0
X.XXXXXX - Process 1: end hard drive output
X.XXXXXX - Process 1: start hard drive output on HDD 1
X.XXXXXX - Process 1: end hard drive output
X.XXXXXX - Process 1: allocating memory
X.XXXXXX - Process 1: memory allocated at 0x00000180
X.XXXXXX - Process 1: start processing action
X.XXXXXX - Process 1: end processing action
X.XXXXXX - Process 1: start printer output on PRNTR 2
X.XXXXXX - Process 1: end printer output
X.XXXXXX - Process 1: start memory blocking
X.XXXXXX - Process 1: end memory blocking
X.XXXXXX - Process 1: start keyboard input
X.XXXXXX - Process 1: end keyboard input
X.XXXXXX - Process 1: start memory blocking
X.XXXXXX - Process 1: end memory blocking
X.XXXXXX - Process 1: start processing action
X.XXXXXX - Process 1: end processing action
X.XXXXXX - Process 1: start processing action
X.XXXXXX - Process 1: end processing action
X.XXXXXX - Process 1: start hard drive output on HDD 0
X.XXXXXX - Process 1: end hard drive output
X.XXXXXX - Process 1: start processing action
X.XXXXXX - Process 1: end processing action
X.XXXXXX - OS: removing process 1
X.XXXXXX - Simulator program ending
```

6 Assignment 5

6.1 Description

Assignment 5 will allow you to increase throughput in your simulator by implementing scheduling algorithms.

6.2 Specification

You will now be required to implement your simulator with three scheduling algorithms.

Note: The times for each operation should equate to the physical time for each discrete operation and the corresponding cycle time. Here I have placed an "X" to indicate that the

time should be calculated by you.

For this project, the following should be considered:

- All processes must be scheduled based on the given algorithm.
- All I/O operations must continue to be threaded.
- Each resource used by a process must be allocated, de-allocated, and re-allocated based on the maximum resource number. Only printers and hard drives will be manageable resources. Remember to lock your resources through mutexes while you are using them.
- All memory allocated during a function must be kept available until the ENTIRE application completes, when it will be freed. If you run out of memory and need to allocate more for the current process, the freed memory will need to be re-allocated for the application that no longer has memory (but is not yet complete).
- The possible scheduling algorithms possible are FIFO (first in first out), PS (priority schedule) and SJF (shortest job first). For PS the process with more number of input and output operations will be performed first and continue in such a manner until the process with least number of input and output operations will be performed in the end. The SJF should count the total number of tasks in a process and the process with least number of tasks will be completed first.
- For this assignment you do not need to do anything with the processor quantum number.

You are expected to create your own meta data files and configuration files at this point. You will be tested with multiple versions of meta data and configuration files that are not given to you. Your program should be robust enough to handle all of the test files previously and any variation upon them.

6.3 Example Configuration

```
Start Simulator Configuration File
Version/Phase: 4.0
File Path: Test_4a.mdf
Processor Quantum Number: 4
CPU Scheduling Code: RR
Processor cycle time (msec): 5
Monitor display time (msec): 22
Hard drive cycle time (msec): 150
Printer cycle time (msec): 550
Keyboard cycle time (msec): 60
Memory cycle time (msec): 10
System memory (kbytes): 2048
Memory block size (kbytes): 128
Printer quantity: 4
Hard drive quantity: 2
Log: Log to File
Log File Path: logfile_1.lgf
End Simulator Configuration File
```

6.4 Example Input

```
Start Program Meta-Data Code:
S(start)0; A(start)0; P(run)11; M(allocate)2; A(end)0;
A(start); O(hard drive)6; A(end); A(start); P(run)4;
```

```
I(keyboard)18; M(allocate)4; P(run)6; S(end)0.
End Program Meta-Data Code.
```

6.5 Example Output

```
X.XXXXXX - Simulator program starting
X.XXXXXX - OS: preparing process 1
X.XXXXXX - OS: starting process 1
X.XXXXXX - Process 1: start processing action
X.XXXXXX - Process 1: interrupt processing action
X.XXXXXX - OS: preparing process 2
X.XXXXXX - OS: starting process 2
X.XXXXXX - Process 2: start hard drive output
X.XXXXXX - OS: preparing process 3
X.XXXXXX - OS: starting process 3
X.XXXXXX - Process 3: start processing action
X.XXXXXX - Process 2: end hard drive output
X.XXXXXX - Process 3: interrupt processing action
.
.
.
X.XXXXXX - OS: process 3 completed
X.XXXXXX - Simulator program ending
```