

# CS 446/646 - OS Simulator Specification

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

Revised: February 2019

## Contents

<b>1</b>	<b>Introduction</b>	<b>2</b>
<b>2</b>	<b>Simulator Description</b>	<b>2</b>
2.1	Expectations . . . . .	2
2.2	Meta-Data . . . . .	3
2.3	Configuration . . . . .	4
2.4	Running the Simulator . . . . .	4
2.5	Turning in Assignments . . . . .	4
<b>3</b>	<b>Assignment 1</b>	<b>5</b>
3.1	Description . . . . .	5
3.2	Specification . . . . .	5
3.3	Example Configuration File . . . . .	6
3.4	Example Input . . . . .	6
3.5	Example Output . . . . .	6
<b>4</b>	<b>Assignment 2</b>	<b>7</b>
4.1	Description . . . . .	7
4.2	Specification . . . . .	7
4.3	Timer Usage . . . . .	8
4.4	Thread Usage . . . . .	8
4.5	Readme . . . . .	8
4.6	Example Configuration . . . . .	8
4.7	Example Input . . . . .	8
4.8	Example Output . . . . .	9
<b>5</b>	<b>Assignment 3</b>	<b>9</b>
5.1	Description . . . . .	9
5.2	Specification . . . . .	10
5.3	Readme . . . . .	10
5.4	Example Configuration . . . . .	10
5.5	Example Input . . . . .	11
5.6	Example Output . . . . .	11
<b>6</b>	<b>Assignment 4</b>	<b>12</b>
6.1	Description . . . . .	12
6.2	Specification . . . . .	12
6.3	Readme . . . . .	13
6.4	Example Configuration . . . . .	13
6.5	Example Input . . . . .	13
6.6	Example Output . . . . .	13

<b>7</b>	<b>Assignment 5</b>	<b>14</b>
7.1	Description . . . . .	14
7.2	Specification . . . . .	14
7.3	Example Configuration . . . . .	14
7.4	Example Input . . . . .	15
7.5	Example Output Format . . . . .	15

## 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 System, used with `begin` and `finish`
- A - Program Application, used with `begin` and `finish`
- P - Process, used with `run`
- I - used with Input operation descriptors such as `hard drive`, `keyboard`, `scanner`
- O - used with Output operation descriptors such as `hard drive`, `monitor`, `projector`
- M - Memory, used with `block`, `allocate`

The meta-data descriptors are as follows:

- `begin`, `finish`, `hard drive`, `keyboard`, `scanner`, `monitor`, `run`, `allocate`, `projector`, `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:

---

```

1 Start Program Meta-Data Code:
2 S{begin}0; A{begin}0; P{run}11; P{run}9; P{run}12;
3 P{run}9; P{run}11; P{run}8; P{run}14; P{run}14; P{run}12;
4 P{run}12; P{run}6; P{run}8; P{run}9; P{run}6; P{run}14;
5 P{run}15; P{run}12; P{run}9; P{run}6; P{run}5; A{finish}0;
6 A{begin}0; P{run}6; P{run}6; P{run}9; P{run}11; P{run}13;
7 P{run}14; P{run}5; P{run}7; P{run}14; P{run}15; P{run}7;
8 P{run}5; P{run}14; P{run}15; P{run}14; P{run}7; P{run}14;
9 P{run}13; P{run}8; P{run}7; A{finish}0; A{begin}0; P{run}6;
10 P{run}10; P{run}13; P{run}9; P{run}15; P{run}6; P{run}13;
```

```
11 P{run}11; P{run}5; P{run}6; P{run}7; P{run}12; P{run}11;
12 P{run}6; P{run}8; P{run}10; P{run}5; P{run}8; P{run}9; P{run}7;
13 A{finish}0; S{finish}0.
14 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:

---

```
1 Start Simulator Configuration File
2 Version/Phase: 2.0
3 File Path: Test_2e.mdf
4 Projector cycle time {msec}: 25
5 Processor cycle time {msec}: 10
6 Keyboard cycle time {msec}: 50
7 Monitor display time {msec}: 20
8 Scanner cycle time {msec}: 10
9 Hard drive cycle time {msec}: 15
10 Log: Log to Both
11 Log File Path: logfile_1.lgf
12 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., `0(hard drive)5` would run for  $5 \times \text{hard drive cycle time}$ )

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

---

```
1 Start Simulator Configuration File
2 Version/Phase: 1.0
3 File Path: Test_1a.mdf
4 Monitor display time {msec}: 20
5 Processor cycle time {msec}: 10
6 Scanner cycle time {msec}: 25
7 Hard drive cycle time {msec}: 15
8 Keyboard cycle time {msec}: 50
9 Memory cycle time {msec}: 30
10 Projector cycle time {msec}: 10
11 Log: Log to Both
12 Log File Path: logfile_1.lgf
13 End Simulator Configuration File
```

---

### 3.4 Example Input

---

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

---

### 3.5 Example Output

---

```
1 Configuration File Data
2 Monitor = 20 ms/cycle
3 Processor = 10 ms/cycle
4 Scanner = 25 ms/cycle
5 Hard Drive = 15 ms/cycle
6 Keyboard = 50 ms/cycle
7 Memory = 30 ms/cycle
8 Projector = 10 ms/cycle
9 Logged to: monitor and logfile_1.lgf
10
11 Meta-Data Metrics
12 P{run}11 - 110 ms
13 M{allocate}2 - 60 ms
14 O{monitor}7 - 140 ms
15 I{hard drive}8 - 120 ms
16 I{scanner}8 - 200 ms
17 O{projector}20 - 200 ms
18 P{run}6 - 60 ms
19 O{projector}4 - 40 ms
20 M{block}6 - 180 ms
21 I{keyboard}17 - 850 ms
22 M{block}4 - 120 ms
```

```
23 O{projector}8 - 80 ms
24 P{run}5 - 50 ms
25 P{run}5 - 50 ms
26 O{hard drive}6 - 90 ms
27 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  $6 \times \text{projector cycle time}$ . 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

---

```
1 Start Simulator Configuration File
2 Version/Phase: 1.0
3 File Path: Test_1a.mdf
4 Monitor display time {msec}: 20
5 Processor cycle time {msec}: 10
6 Scanner cycle time {msec}: 25
7 Hard drive cycle time {msec}: 15
8 Keyboard cycle time {msec}: 50
9 Memory cycle time {msec}: 30
10 Projector cycle time {msec}: 10
11 System memory {kbytes}: 1024
12 Log: Log to Both
13 Log File Path: logfile_1.lgf
14 End Simulator Configuration File
```

---

### 4.7 Example Input

---

```
1 Start Program Meta-Data Code:
2 S{begin}0; A{begin}0; P{run}11; M{allocate}2;
3 O{monitor}7; I{hard drive}8; O{projector}20;
4 P{run}6; O{projector}4; M{block}6; I{keyboard}17;
```



```
5 M{block}4; A{finish}0; A{begin}0; P{run}5; P{run}5; O{hard drive}6;  
6 P{run}18; A{finish}0; S{finish}0.  
7 End Program Meta-Data Code.
```

---

## 4.8 Example Output

---

```
1 0.000001 - Simulator program starting  
2 0.000051 - OS: preparing process 1  
3 0.000053 - OS: starting process 1  
4 0.000055 - Process 1: start processing action  
5 0.132061 - Process 1: end processing action  
6 0.132063 - Process 1: allocating memory  
7 0.182066 - Process 1: memory allocated at 0x00000010  
8 0.182069 - Process 1: start monitor output  
9 0.392073 - Process 1: end monitor output  
10 0.392074 - Process 1: start hard drive input  
11 0.536077 - Process 1: end hard drive input  
12 0.536079 - Process 1: start projector output  
13 5.536081 - Process 1: end projector output  
14 5.536085 - Process 1: start processing action  
15 5.608088 - Process 1: end processing action  
16 5.608089 - Process 1: start projector output  
17 6.608094 - Process 1: end projector output  
18 6.608097 - Process 1: start memory blocking  
19 6.758099 - Process 1: end memory blocking  
20 6.758101 - Process 1: start keyboard input  
21 6.843104 - Process 1: end keyboard input  
22 6.843106 - Process 1: start memory blocking  
23 6.943109 - Process 1: end memory blocking  
24 6.931074 - OS: removing process 1  
25 6.941181 - OS: preparing process 2  
26 6.942381 - OS: starting process 2  
27 6.943110 - Process 2: start processing action  
28 7.003114 - Process 2: end processing action  
29 7.003116 - Process 2: start processing action  
30 7.063119 - Process 2: end processing action  
31 7.063127 - Process 2: start hard drive output  
32 7.171130 - Process 2: end hard drive output  
33 7.171134 - Process 2: start processing action  
34 7.387137 - Process 2: end processing action  
35 7.387158 - OS: removing process 2  
36 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. **You can only use the pthread library for mutex and semaphore implementation. Any other library usage will affect your score.** 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 (projector, 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 projector and hard drive must be printed and your code should reset back to 0 after reaching the maximum quantity value.

## 5.3 Readme

The readme document must include instructions on how to compile and run the code. In addition, you must include the file name (.cpp/.c) and the line number where mutex and semaphores are used. Failure to do so will affect the score.

## 5.4 Example Configuration

---

```
1 Start Simulator Configuration File
2 Version/Phase: 3.0
3 File Path: Test_3a.mdf
4 Processor cycle time {msec}: 5
5 Monitor display time {msec}: 22
6 Hard drive cycle time {msec}: 150
7 Projector cycle time {msec}: 550
8 Keyboard cycle time {msec}: 60
9 Memory cycle time {msec}: 10
```

```
10 Scanner cycle time {msec}: 15
11 System memory {kbytes}: 2048
12 Memory block size {kbytes}: 128
13 Projector quantity: 4
14 Hard drive quantity: 2
15 Log: Log to File
16 Log File Path: logfile_1.lgf
17 End Simulator Configuration File
```

---

## 5.5 Example Input

---

```
1 Start Program Meta-Data Code:
2 S{begin}0; A{begin}0; P{run}11; M{allocate}2;
3 O{monitor}7; I{hard drive}8; O{projector}20;
4 M{allocate}4; A{finish}0; A{begin}0; O{projector}6; M{allocate}3; I{hard drive}7;
5 O{hard drive}2; O{hard drive}16; M{allocate}4;
6 P{run}6; O{projector}4; A{finish}0; A{begin}0; M{block}6; I{keyboard}17;
7 M{block}4; P{run}5; P{run}5; O{hard drive}6;
8 P{run}18; A{finish}0; S{finish}0.
9 End Program Meta-Data Code.
```

---

## 5.6 Example Output

---

```
1 X.XXXXXX - Simulator program starting
2 X.XXXXXX - OS: preparing process 1
3 X.XXXXXX - OS: starting process 1
4 X.XXXXXX - Process 1: start processing action
5 X.XXXXXX - Process 1: end processing action
6 X.XXXXXX - Process 1: allocating memory
7 X.XXXXXX - Process 1: memory allocated at 0x00000000
8 X.XXXXXX - Process 1: start monitor output
9 X.XXXXXX - Process 1: end monitor output
10 X.XXXXXX - Process 1: start hard drive input on HDD 0
11 X.XXXXXX - Process 1: end hard drive input
12 X.XXXXXX - Process 1: start projector output on PROJ 0
13 X.XXXXXX - Process 1: end projector output
14 X.XXXXXX - Process 1: allocating memory
15 X.XXXXXX - Process 1: memory allocated at 0x00000080
16 X.XXXXXX - OS: removing process 1
17 X.XXXXXX - OS: preparing process 2
18 X.XXXXXX - OS: starting process 2
19 X.XXXXXX - Process 2: start projector output on PROJ 1
20 X.XXXXXX - Process 2: end projector output
21 X.XXXXXX - Process 2: allocating memory
22 X.XXXXXX - Process 2: memory allocated at 0x00000100
23 X.XXXXXX - Process 2: start hard drive input on HDD 1
24 X.XXXXXX - Process 2: end hard drive input
25 X.XXXXXX - Process 2: start hard drive output on HDD 0
26 X.XXXXXX - Process 2: end hard drive output
27 X.XXXXXX - Process 2: start hard drive output on HDD 1
28 X.XXXXXX - Process 2: end hard drive output
```

```

29 X.XXXXXX - Process 2: allocating memory
30 X.XXXXXX - Process 2: memory allocated at 0x00000180
31 X.XXXXXX - Process 2: start processing action
32 X.XXXXXX - Process 2: end processing action
33 X.XXXXXX - Process 2: start projector output on PROJ 2
34 X.XXXXXX - Process 2: end projector output
35 X.XXXXXX - OS: removing process 2
36 X.XXXXXX - OS: preparing process 3
37 X.XXXXXX - OS: starting process 3
38 X.XXXXXX - Process 3: start memory blocking
39 X.XXXXXX - Process 3: end memory blocking
40 X.XXXXXX - Process 3: start keyboard input
41 X.XXXXXX - Process 3: end keyboard input
42 X.XXXXXX - Process 3: start memory blocking
43 X.XXXXXX - Process 3: end memory blocking
44 X.XXXXXX - Process 3: start processing action
45 X.XXXXXX - Process 3: end processing action
46 X.XXXXXX - Process 3: start processing action
47 X.XXXXXX - Process 3: end processing action
48 X.XXXXXX - Process 3: start hard drive output on HDD 0
49 X.XXXXXX - Process 3: end hard drive output
50 X.XXXXXX - Process 3: start processing action
51 X.XXXXXX - Process 3: end processing action
52 X.XXXXXX - OS: removing process 3
53 X.XXXXXX - Simulator program ending

```

---

## 6 Assignment 4

### 6.1 Description

Assignment 4 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 projectors 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.
- For this assignment you will need to read the entire meta data content from  $S_{\text{begin}}0$  to  $S_{\text{finish}}0$  before performing any scheduling operation.

### 6.3 Readme

The readme document must include instructions on how to compile and run the code. In addition, you must include the file name (.cpp/.c) and the line number where scheduling algorithms are used. Failure to do so will affect the score.

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.

### 6.4 Example Configuration

---

```
1 Start Simulator Configuration File
2 Version/Phase: 4.0
3 File Path: Test_4a.mdf
4 Processor Quantum Number: 1
5 CPU Scheduling Code: SJF
6 Processor cycle time {msec}: 5
7 Monitor display time {msec}: 22
8 Hard drive cycle time {msec}: 150
9 Projector cycle time {msec}: 550
10 Keyboard cycle time {msec}: 60
11 Memory cycle time {msec}: 10
12 Scanner cycle time {msec} : 30
13 System memory {kbytes}: 2048
14 Memory block size {kbytes}: 128
15 Projector quantity: 4
16 Hard drive quantity: 2
17 Log: Log to File
18 Log File Path: logfile_1.lgf
19 End Simulator Configuration File
```

---

### 6.5 Example Input

---

```
1 Start Program Meta-Data Code:
2 S{begin}0; A{begin}0; P{run}11; M{allocate}2; A{finish}0;
3 A{begin}0; O{hard drive}6; A{finish}0; A{begin}0; P{run}4;
4 I{keyboard}18; M{allocate}4; P{run}6; A{finish}0; S{finish}0.
5 End Program Meta-Data Code.
```

---

### 6.6 Example Output

---

```

1 X.XXXXXX - Simulator program starting
2 X.XXXXXX - OS: preparing process 2
3 X.XXXXXX - OS: starting process 2
4 X.XXXXXX - Process 2: start hard drive output
5 X.XXXXXX - Process 2: end hard drive output
6 X.XXXXXX - End process 2
7 X.XXXXXX - OS: preparing process 1
8 X.XXXXXX - OS: starting process 1
9 X.XXXXXX - Process 1: start processing action
10 X.XXXXXX - Process 1: end processing action
11 X.XXXXXX - Process 1: allocating memory
12 X.XXXXXX - Process 1: memory allocated at 0x00000000
13 X.XXXXXX - End process 1
14 X.XXXXXX - OS: preparing process 3
15 X.XXXXXX - OS: starting process 3
16 X.XXXXXX - Process 3: start processing action
17 X.XXXXXX - Process 3: end processing action
18 .
19 .
20 .
21 .
22 X.XXXXXX - End process 3
23 X.XXXXXX - Simulator program ending

```

---

## 7 Assignment 5

### 7.1 Description

You will now be required to implement your simulator with two scheduling algorithms using interrupts.

### 7.2 Specification

All of your processes must be interruptible and never exceed the number of cycles specified by the quantum number. All of the I/O operations must continue to be threaded and should be handled by your simulator as they complete. The possible scheduling algorithms are Round Robin (RR) and Shortest time remaining (STR). All resources must be allocated according to the specified scheduling algorithm. In addition you need to use a dedicated load thread to reload the meta-data every 100ms for about 5 times to simulate the processes' arrive time. Every time the meta-data is loaded, the applications must be treated independent of the previously loaded applications. Round Robin needs to have a dedicated thread to record time (50ms to interrupt)

### 7.3 Example Configuration

---

```

1 Start Simulator Configuration File
2 Version/Phase: 5.0
3 File Path: Test_5a.mdf
4 Processor Quantum Number {msec}: 50
5 CPU Scheduling Code: RR
6 Processor cycle time {msec}: 5
7 Monitor display time {msec}: 22
8 Hard drive cycle time {msec}: 150

```

```
9 Projector cycle time {msec}: 550
10 Keyboard cycle time {msec}: 60
11 Memory cycle time {msec}: 10
12 System memory {kbytes}: 2048
13 Memory block size {kbytes}: 128
14 Projector quantity: 4
15 Hard drive quantity: 2
16 Log: Log to File
17 Log File Path: logfile_1.lgf
18 End Simulator Configuration File
```

---

## 7.4 Example Input

---

```
1 Start Program Meta-Data Code:
2 S{begin}0; A{begin}0; P{run}11; A{finish}0;
3 A{begin}0; O{hard drive}6; A{finish}0; A{begin}0; P{run}4;
4 I{keyboard}18; M{allocate}4; P{run}6; A{finish}0; S{finish}0.
5 End Program Meta-Data Code.
```

---

## 7.5 Example Output Format

At this point you should be familiar with the structure of the output. **Below is the just the format on how the output should look like and it does not represent the correct order of scheduling.**

---

```
1
2 X.XXXXXX - Simulator program starting
3 X.XXXXXX - OS: preparing process 1
4 X.XXXXXX - OS: starting process 1
5 X.XXXXXX - Process 1: start processing action
6 X.XXXXXX - Process 1: interrupt processing action
7 X.XXXXXX - OS: preparing process 2
8 X.XXXXXX - OS: starting process 2
9 X.XXXXXX - Process 2: start hard drive output
10 X.XXXXXX - OS: preparing process 3
11 X.XXXXXX - OS: starting process 3
12 X.XXXXXX - Process 3: start processing action
13 X.XXXXXX - Process 2: end hard drive output
14 X.XXXXXX - Process 3: interrupt processing action
15 .
16 .
17 .
18 X.XXXXXX - OS: process 3 completed
19 X.XXXXXX - Simulator program ending
```

---