CS 446/646 SIMULATOR PROJECT

INTRODUCTION

This assignment has been developed to provide you with a quality experience of the design and operational decisions made by persons developing an Operating System. However, it also incorporates the real world (i.e., advanced academia and/or industry) conditions of managing a slightly larger scale project as well as reading code during the grading component of each phase.

The simulator project will be run in three phases. Each of these phases will be specified in this document although some small changes may be made as the project progresses. The Instructor is open to changes recommended by students as long as the entire project is completed on or before 9 December (Preparation Day).

The simulator must be programmed in C or C++. 5 points of extra credit will be provided to all students (i.e., graduate and undergraduate) for each program phase that is implemented in C. This means it must be written completely in C and the make operation must compile it with **gcc** instead of **g++**. All programs (i.e., C or C++) are expected to use a make file.

CALENDAR/SCHEDULE OF ASSIGNMENTS

15 Feb: Sim01 assigned in Week 5 folder
19 Feb: PA01 grading due on paper to Michael's office
04 Mar: Sim01 due in Week 5 folder
07 Mar: Sim02 assigned in Week 8 folder
11 Mar: Sim01 grading due on paper to Michael's office
01 Apr: Sim02 due in Week 8 folder
04 Apr: Sim03 program assigned in Week 11 folder
08 Apr: Sim02 grading due on paper to Michael's office
22 Apr: Sim03 due in Week 11 folder
29 Apr: Sim03 grading due on paper to Michael's office

The schedule above is provided to help make assignments and due dates clear. Make sure you correctly turn in each assignment to the right place by the right time.

GENERAL PROGRAMMING AND DEVELOPMENT EXPECTATIONS

Specific rubrics will be provided for grading each program. However, the following are general expectations of programmers in this 400-/600- level course:

- 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

- 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. If there is any question about the level of these standards, check with Michael

- 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.)

- for each programming assignment, each student will upload the program files using his or her own secret ID. 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. As was implemented in PA01 the file must be named **Sim0X_<ID Code>.tar.gz** where **X** represents the specific project number, or as an example, **Sim01_123456.tar.gz**. The programs must be uploaded at or before 3:00 pm on the date for each specific programming project/phase and delivered to the Instructor's office before 3:00 pm for each grading component. Dates are found previously in this document

CREATING THE PROGRAM META-DATA

The program meta-data components 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**

The program meta-data descriptors are as follows:

end, hard drive, keyboard, printer, monitor, run, start

The cycle times are applied as specified here:

The cycle time represents the number of milliseconds per cycle for the program. For example, if the hard drive has a 50 msec/cycle time, and it is supposed to run for 10 cycles, the hard drive operation must run for 500 msec. You must use an onboard clock interface of some kind to manage this, and the precision must be to the microsecond level. The cycle time in the meta-data program must be set to zero when it is not applicable to the operation, but it is always required as part of the meta-data code. Also note that the simulator must represent real time; if the operations take 10 seconds, the simulator should take 10 seconds.

The form for each meta-data operation is as follows:

<component letter>(<operation>)<cycle time>; <successive meta-data operation>; . . . <last operation>.

Creating example test programs:

The program **programgenerator.cpp** has been developed to support testing and work with this assignment. It can generate test program meta-data with varying parameters. It can also be modified if you wish to use a different operations-generating algorithm(s).

RUNNING THE SIMULATOR

The simulator will input a configuration file that is accepted from the command line, as follows:

```
./sim0x config_1.cnf
```

Differing configuration files will be used for various testing purposes. Also, the simulator name **must** be the assignment name (e.g., sim01, sim02, etc.)

Phase I – Single Program Management

DESCRIPTION

This phase will require the creation of a simple, one-program OS simulator named **`Sim01`**, using three states (Enter/Start, Running, Exit). It will accept the meta-data for one program with a potentially unlimited number of meta-data operations, run it, and end the simulation. Each unique I/O operation must be conducted with its own unique thread

Each individual call to an I/O operation must be conducted by a unique, individual thread. No credit will be earned for the programming part of the assignment if threads are not used for these actions. Note that you must create and call your own threads; you may not call timer or other library threads.

THE CONFIGURATION FILE

Contents of the configuration file are as follows:

Start Simulator Configuration File
Version/Phase: <number>
File Path: <complete file path and name of program file>
Processor cycle time (msec/cycle): <time>
Monitor display time (msec/cycle): <time>
Hard drive cycle time (msec/cycle): <time>
Printer cycle time (msec/cycle): <time>
Keyboard cycle time (msec/cycle): <time>
Log: <Log to Both> OR <Log to Monitor> OR <Log to File>
Log File Path: <complete file path and name of log file name>
End Simulator Configuration File

A specific example of the configuration file is shown here:

```
Start Simulator Configuration File
Version/Phase: 1.0
File Path: test_1.mdf
Processor cycle time (msec): 10
Monitor display time (msec): 25
Hard drive cycle time (msec): 50
Printer cycle time (msec): 500
Keyboard cycle time (msec): 100
Log: Log to Both
Log File Path: logfile_1.lgf
End Simulator Configuration File
```

Example meta-data program:

```
Start Program Meta-Data Code:
S(start)0; A(start)0; I(hard drive)14; P(run)7;
O(hard drive)5; I(keyboard)8; O(hard drive)10; I(keyboard)15;
P(run)10; O(monitor)11; I(keyboard)14; O(printer)8; A(end)0;
S(end)0.
End Program Meta-Data Code.
```

SIMULATOR OPERATIONS AND OUTPUT

The simulator will conduct the following operations:

It will load and store the configuration file as needed, it will create a Process Control Block (PCB) for the one program, it will then put the process into a running mode, and it will report on all of its activities. The program must be able to print the output to the screen, to a file, or both, as provided in the configuration file. Here is an example using the previously provided example input and configuration file; note again that this program should take about ten seconds to run:

```
0.000111 - Simulator program starting
0.000341 - OS: preparing process 1
0.000361 - OS: starting process 1
0.000471 - Process 1: start hard drive input
0.700471 - Process 1: end hard drive input
0.700587 - Process 1: start processing action
0.770587 - Process 1: end processing action
0.770666 - Process 1: start hard drive output
1.020666 - Process 1: end hard drive output
1.020735 - Process 1: start keyboard input
1.820735 - Process 1: end keyboard input
1.820811 - Process 1: start hard drive output
2.320809 - Process 1: end hard drive output
2.320948 - Process 1: start keyboard input
3.820948 - Process 1: end keyboard input
3.821096 - Process 1: start processing action
3.921095 - Process 1: end processing action
3.921189 - Process 1: start monitor output
4.196188 - Process 1: end monitor output
4.196296 - Process 1: start keyboard input
5.596297 - Process 1: end keyboard input
5.596404 - Process 1: start printer output
9.596403 - Process 1: end printer output
9.596560 - OS: removing process 1
9.596616 - Simulator program ending
```

Phase II – Batch Processing

DESCRIPTION

This phase will require the creation of a batch program OS simulator named **Sim02**, using four states (Enter/Start, Ready, Running, Exit). It will accept the meta-data for several programs (i.e., potentially unlimited number), run the programs one at a time, and then end the simulation.

Undergraduate: Must be configurable for, and implement, First Come – First Served (FCFS) or Shortest Job First (SJF) CPU scheduling algorithms depending on the code provided in the configuration file.

Graduate, or undergraduate extra credit: In addition to FCFS, and SJF, must also be configurable for, and implement, Shortest Remaining Time First - Non Preemptive (SRTF-N) CPU scheduling algorithms; SRTF-N must analyze which of the remaining processes is the shortest after the completion of each process, then select the shortest available remaining process. As a note, the resulting output should be the same as SJF for this assignment; however, the code must show an evaluation action after each process has completed

Again, each individual call to an I/O operation must be conducted by a unique, individual thread. No credit will be earned for the programming part of the assignment if threads are not used for these actions.

Note the addition of the CPU Scheduling option in the configuration file below (highlighted in red), and that this is now version 2.0.

Also, don't forget to change your "Version/Phase:" configuration data to 2.0.

THE CONFIGURATION FILE

Contents of the configuration file are as follows:

```
Start Simulator Configuration File
Version/Phase: <number>
File Path: <complete file path and name of program file>
CPU Scheduling Code: <scheduling code>
Processor cycle time (msec/cycle): <time>
Monitor display time (msec/cycle): <time>
Hard drive cycle time (msec/cycle): <time>
Printer cycle time (msec/cycle): <time>
Keyboard cycle time (msec/cycle): <time>
Log: <Log to Both> OR <Log to Monitor> OR <Log to File>
Log File Path: <complete file path and name of log file name>
End Simulator Configuration File
```

Here is a specific example of a configuration file:

```
Start Simulator Configuration File
Version/Phase: 2.0
File Path: Test_5.mdf
CPU Scheduling Code: FCFS
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
```

Example Metadata file:

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

SIMULATOR OPERATIONS AND OUTPUT

The simulator will conduct the following operations:

It will load and store the configuration file as needed, it will create a Process Control Block (PCB) for each program, it will then sequentially put each process into a running mode sequentially, and it will report on all of its activities. The program must be able to print the output to the screen, to a file, or both, as provided in the configuration file. Here is an example output from the example metadata file shown above.

```
0.000000 - Simulator program starting
0.000080 - OS: preparing all processes
0.000093 - OS: selecting next process
0.000274 - OS: starting process 1
0.000561 - Process 1: start hard drive input
0.165560 - Process 1: end hard drive input
0.165688 - Process 1: start processing action
0.255688 - Process 1: end processing action
0.255782 - Process 1: start monitor output
0.475783 - Process 1: end monitor output
0.475871 - Process 1: start keyboard input
1.125870 - Process 1: end keyboard input
1.125984 - Process 1: start processing action
1.255984 - Process 1: end processing action
```

1.256088 - Process 1: start hard drive output
1.436089 - Process 1: end hard drive output
1.436179 - Process 1: start hard drive input
1.556179 - Process 1: end hard drive input
1.556269 - OS: removing process 1
1.556323 - OS: selecting next process
1.556358 - OS: starting process 2
1.556431 - Process 2: start printer output
1.681431 - Process 2: end printer output
1.681504 - Process 2: start processing action
1.741503 - Process 2: end processing action
1.741576 - Process 2: start hard drive input
1.936576 - Process 2: end hard drive input
1.936660 - Process 2: start monitor output
2.076661 - Process 2: end monitor output
2.076741 - Process 2: start hard drive input
2.256741 - Process 2: end hard drive input
2.256831 - Process 2: start processing action
2.346831 - Process 2: end processing action
2.346909 - Process 2: start hard drive output
2.511908 - Process 2: end hard drive output
2.511994 - OS: removing process 2
2.512041 - OS: selecting next process
2.512074 - OS: starting process 3
2.512144 - Process 3: start hard drive input
2.737143 - Process 3: end hard drive input
2.737215 - Process 3: start processing action
2.857214 - Process 3: end processing action
2.857290 - Process 3: start processing action
2.957290 - Process 3: end processing action
2.957378 - Process 3: start monitor output
3.237378 - Process 3: end monitor output
3.237536 - Process 3: start keyboard input
3.837536 - Process 3: end keyboard input
3.837632 - Process 3: start hard drive output
3.957631 - Process 3: end hard drive output
3.957706 - Process 3: start hard drive input
4.182706 - Process 3: end hard drive input
4.182781 - OS: removing process 3
4.182831 - OS: selecting next process
4.182865 - OS: starting process 4

```
4.182936 - Process 4: start processing action
4.232937 - Process 4: end processing action
4.233017 - Process 4: start printer output
4.533016 - Process 4: end printer output
4.533092 - Process 4: start processing action
4.593092 - Process 4: end processing action
4.593169 - Process 4: start processing action
4.703170 - Process 4: end processing action
4.703251 - Process 4: start keyboard input
5.353250 - Process 4: end keyboard input
5.353337 - Process 4: start monitor output
5.653336 - Process 4: end monitor output
5.653464 - Process 4: start keyboard input
6.003464 - Process 4: end keyboard input
6.003563 - OS: removing process 4
6.003615 - Simulator program ending
```

You may place other reporting statements (beyond what is shown above) into your display if it helps to better represent the OS actions.

Phase III – Multi-Programming

DESCRIPTION

This phase will require the creation of a multiprogramming OS simulator named **`Sim03`**, using five states (Enter/Start, Ready, Running, Blocked/Waiting, and Exit). It will accept the meta-data for several programs (i.e., potentially unlimited number, with a potentially unlimited number of operations in each), run the programs concurrently using a multi-programming strategy, and then end the simulation.

Undergraduate: Must conduct Round Robin (RR) management with a specified Quantum (i.e., number of cycles) time. Must be configurable for and implement a First In – First Out with pre-emption (FIFO-P) CPU scheduling algorithm.

Graduate, or undergraduate extra credit: Must include the undergraduate requirements. In addition to FIFO-P, the simulator must also be configurable for, and implement the Shortest Remaining Time First – Preemptive (SRTF-P) CPU scheduling algorithm; SRTF-P must analyze which of the remaining processes is the shortest at the end of each interrupted process whether the Quantum time was completed or not, then select and run the shortest available remaining process.

All students: For this phase, when an I/O process is called (i.e., the I/O thread is deployed), the running process must be placed in the Blocked/Waiting state. When the I/O process (i.e., thread) has completed, it must interrupt the processor at the end of the presently running process cycle* causing the OS to move the process back into its appropriate place in the Ready state. This REQUIRES that an interrupt system be implemented as an integral part of the program.

*interrupts must happen at integral cycle points; this means that if 45 mSec of a 100 mSec cycle has been run, the remaining 55 mSec must also be run so that when the process is pulled out of the running mode, it will be in a state such that it can start a new cycle (as opposed to a fraction of a cycle) when it returns to running; note that this may require queueing interrupts

As before, and as is clearly obvious for this phase, each individual call to an I/O operation must be conducted by a unique, individual thread. No credit will be earned for the programming part of the assignment if threads are not used for these actions. And to repeat a previous requirement, you must create your own sleep timers; you may not use sleep, usleep, or any other library sleeping or waiting threads.

THE CONFIGURATION FILE

Contents of the configuration file are as follows:

Start Simulator Configuration File
Version/Phase: <number>
File Path: <complete file path and name of program file>
CPU Scheduling: <scheduling code>
Quantum time (cycles): <number of cycles for quantum>
Processor cycle time (msec/cycle): <time>
Monitor display time (msec/cycle): <time>
Hard drive cycle time (msec/cycle): <time>
Printer cycle time (msec/cycle): <time>
Keyboard cycle time (msec/cycle): <time>
Log: <Log to Both> OR <Log to Monitor> OR <Log to File>
Log File Path: <complete file path and name of log file name>
End Simulator Configuration File

SIMULATOR OPERATIONS AND OUTPUT

The simulator will conduct the following operations:

It will load and store the configuration file as needed, it will create a Process Control Block (PCB) for each program, it will then put each process into the Ready queue, use the appropriate strategy to select and run the first process, place it into Running mode, and then watch for an interrupt that indicates that either: 1) the Quantum time or 2) any given I/O action, have been completed. From that point, the OS must use the configured algorithm to select the next process to be run, and place it in the Running mode. Each process must be removed as soon as it has completed, and the simulator must shut down after the last process has been completed and removed. As before, the simulator must report on as many actions as reasonably necessary to show what the system is doing at any given moment, and then be able to print the output to the screen, to a file, or both, as provided in the configuration file.

The following represents an example of the configuration file:

```
Start Simulator Configuration File
Version/Phase: 3.0
File Path: Test_3.mdf
CPU Scheduling Code: FIFO-P
Quantum Time (cycles): 3
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
```

The following represents an example of a meta-data file:

Start Program Meta-Data Code:
S(start)0; A(start)0; I(keyboard)5; O(hard drive)14; P(run)11;
P(run)11; I(hard drive)14; A(end)0; A(start)0; P(run)8;
O(monitor)8; I(keyboard)7; P(run)13; O(hard drive)6; A(end)0;
A(start)0; I(hard drive)5; P(run)5; O(hard drive)5;
I(keyboard)15; O(hard drive)10; A(end)0; A(start)0; P(run)7;
P(run)15; I(hard drive)14; O(monitor)5; P(run)13; A(end)0;
A(start)0; I(keyboard)10; O(monitor)11; I(hard drive)10;
O(monitor)13; I(hard drive)14; A(end)0; S(end)0.
End Program Meta-Data Code.

The following represents a contrived* output from the above configuration and metadata files:

0.000001 - Simulator program starting
0.000041 - OS: preparing all processes
0.000046 - OS: selecting next process
0.000303 - Process 1: keyboard input - start
0.250302 - Process 1 : block for keyboard input
0.250394 - OS: selecting next process
0.250451 - Process 2: processing action - start
0.400451 - Interrupt: Process 1 - keyboard input completed
0.400504 - OS: selecting next process
0.400552 - Process 1: hard drive output - start
0.610552 - Process 1 : block for hard drive output
0.610609 - OS: selecting next process
0.610658 - Process 2: processing action - continue
0.775657 - Interrupt: Process 2 - quantum time out
0.775710 - OS: selecting next process
0.775760 - Process 3: hard drive output - start
0.895759 - Process 3: block for hard drive output
0.895811 - OS: selecting next process
0.895862 - Process 2: processing action - continue
0.970862 - Interrupt: Process 1 - hard drive output completed
0.970915 - OS: selecting next process
0.970949 - Process 1: processing action - start

1.015948 - Interrupt: Process 1 - quantum time out
1.016002 - OS: selecting next process
1.016038 - Process 2: processing action - continue
1.126038 - Interrupt: Process 3 - hard drive output completed
1.126149 - OS: selecting next process
1.126244 - Process 1: processing action - continue
1.206243 - Interrupt: Process 1 - quantum time out
1.206375 - OS: selecting next process
.
.
.
11.985197 - Simulator program ending

*I put this output together by hand since I did not have a running program with which to work. The example output shows what the results should look like, with the following considerations:
- the times are not correct in this example; I just used times from another output to make this up
- for this example the processes are started and queued in FIFO order, as specified in the configuration file; in this case it happens that the lower the Process ID, the sooner it was dequeued from waiting if it was available; however, with more processes (and using FIFO scheduling), it is likely that processes will fall to the back of the queue and not be dequeued as quickly
- virtually all processing actions are interrupted, if not by time then by one or more completed I/O actions; this example does not show more than one I/O action completed at one time but the frequent interruption is a likely event
- no matter which processor queueing/scheduling algorithm is used, the OS should report when it is making the decision as to which process will be moved to a running state next; in fact, it should be reporting ANY decisions it makes and/or actions it takes
- also note that if you have other presentation components (i.e., more complete OS operation data, more complete I/O operation actions, etc.) that will better represent your simulator operation, you are welcome to add them to your output