CS 446/646 SIMULATOR PROJECT

Here are the description and the specifications for the simulator.

RUNNING THE SIMULATOR

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

```
./simulator config_1.txt
```

Differing configuration files will be used for various testing purposes.

CREATING THE CONFIGURATION FILE

Contents of the configuration file are as follows:

Start Simulator Configuration File
Version: <number>
File Path: <file path of program file>
Quantum (cycles): <time quantum for processor>
Processor Scheduling: <FIFO> OR <SJF> OR <SRTF> OR <RR>
Processor cycle time (msec): <time>
Monitor display time (msec): <time>
Hard drive cycle time (msec): <time>
Printer cycle time (msec): <time>
Keyboard cycle time (msec): <time>
Memory type: <Fixed> OR <Unlimited/Fixed> OR <Limited/Other>
Log: <Log to Both> OR <Log to Monitor> OR <Log to File>
End Simulator Configuration File

A specific example of the configuration file is shown here:

Start Simulator Configuration File
Version: 1.0
Quantum (cycles): 3
Processor Scheduling: FIFO
File Path: c:\users\billsmith\simproj1\
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): 1000
Memory type: Fixed
Log: Log to Both
End Simulator Configuration File

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, monitor, run, start

The cycle times are applied as described here:

The cycle time may be set to zero when it is not applicable to the operation, but it is always required as part of the meta-data operation.

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

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

Example meta-data programs:

Balanced Processing and I/O:

Program Meta-Data Code:

S(start)0; A(start)0; P(run)14; I(hard drive)13; O(hard drive)15;
P(run)12; O(hard drive)11; P(run)5; I(hard drive)12; O(hard drive)12;
P(run)5; O(monitor)10; P(run)12; O(monitor)10; A(end)0; S(end)0.

I/O Bound

Program Meta-Data Code:

S(start)0; A(start)0; I(hard drive)10; P(run)13; O(monitor)15;
I(keyboard)9; O(monitor)5; I(hard drive)6; O(monitor)12; P(run)9; I(keyboard)10;
O(hard drive)7; P(run)10; I(keyboard)7; A(end)0; S(end)0.

CPU Bound

Program Meta-Data Code:

S(start)0; A(start)0; I(hard drive)12; O(hard drive)7; I(keyboard)15;
O(hard drive)11; P(run)10; P(run)9; I(hard drive)14; P(run)11; O(monitor)13;
P(run)10; O(monitor)6; P(run)15; A(end)0; S(end)0.

CREATING EXAMPLE TEST PROGRAMS

The program **programgenerator.cpp** has been developed to support testing and work with this assignment. It can generate a single program for use by the Low-Level assignment, and it can generate multiple programs for use by the High-Level assignment (shown later in this document). It can also be modified if you wish to use a different operations-generating algorithm(s).

SIMULATOR OUTPUT – LOW LEVEL

The simulator output will represent the processing as it occurs, and the output, which may be logged to the monitor, a file, or both, depending on the configuration file, must show the operations as they are conducted, as follows. This program must create and use unique processes for each operational action. Also note that your program must be able to adapt to various processor queuing systems, memory types, and logs but to start with, it only has to use FIFO queuing and Fixed memory (no real memory management; just use the memory in your functions); however, it must be able to adapt to differing logging requests

For the program:

```
S(start)0; A(start)0; P(run)14; I(hard drive)13; O(hard drive)15;
P(run)12; O(hard drive)11; P(run)5; I(hard drive)12; O(hard drive)12;
P(run)5; O(monitor)10; P(run)12; O(monitor)10; A(end)0; S(end)0.
```

With the pertinent configuration components:

```
Version: 1.0
Quantum (cycles): 3
Processor Scheduling: FIFO
File Path: c:\users\billsmith\simproj1\
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): 1000
Memory type: Fixed
Log: Log to Both
```

The output would be:

```
PID 1 – Enter system
PID 1 – Processing (140 mSec)
PID 1 – Input, hard drive (650 mSec)
PID 1 – Output, hard drive (750 mSec)
PID 1 – Processing (120 mSec)
PID 1 – Output, hard drive (550  mSec)
PID 1 – Processing (50 mSec)
PID 1 – Input, hard drive (600 mSec)
PID 1 – Processing (50 mSec)
PID 1 – Output, monitor (250 mSec)
PID 1 – Processing (120 mSec)
PID 1 – Output, monitor (250 mSec)
PID 1 – Exit system
```

SIMULATOR OUTPUT – HIGH LEVEL

For the High-Level assignment, threads must be used which will allow the I/O processes to work independently of their initializing process; in addition, you will be keeping track of the Operating System time necessary to manage the processes with at least one queuing system, and displaying it as well.

Note that using threads for the I/O operations will also require some kind of interrupt system to inform the Operating System when a thread (I/O) action has completed for a specific process. Also note that your program must be able to adapt to various processor queuing systems, memory types, and logs but to start with, it only has to use Round Robin queuing and Fixed memory (no real memory management; just use the memory in your functions); however, it must be able to adapt to differing logging requests

For the programs:

```
S(start)0; A(start)0; P(run)13; I(keyboard)5; P(run)6; O(monitor)5;
P(run)5; I(hard drive)5; P(run)7; A(end)0; A(start)0; P(run)10; I(keyboard)5; P(run)7;
O(hard drive)5; P(run)15; A(end)0; A(start)0; P(run)13; I(hard drive)5; P(run)14;
O(hard drive)5; P(run)13; I(hard drive)5; P(run)10; S(end)0.
```

With the pertinent configuration components:

```
Version: 1.0
Quantum (cycles): 3
Processor Scheduling: RR
File Path: c:\users\billsmith\simproj1\
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): 1000
Memory type: Fixed
Log: Log to Both
```

This simulator will support multi-programming and might look like the following; this will be discussed in class as there are many variables involved:

```
SYSTEM – Boot, set up (200 mSec)
PID 1 – Enter system
SYSTEM – Creating PID 1 (30 mSec)
PID 2 – Enter system
SYSTEM – Creating PID 2 (40 mSec)
PID 3 – Enter system
SYSTEM – Creating PID 3 (25 msSec)
PID 1 – Processing (130 mSec)
SYSTEM – Swapping processes (30 mSec)
PID 2 – Processing (100 mSec)
SYSTEM – Swapping processes (30 mSec)
PID 3 – Processing (130 mSec)
SYSTEM – Swapping processes (30 mSec)
SYSTEM – Managing I/O (20 mSec)
PID 1 – Input, keyboard started
SYSTEM – Swapping processes (20 mSec)
SYSTEM – Managing I/O (20 mSec)
PID 2 – Input, keyboard started
SYSTEM – Managing I/O (20 mSec)
PID 1 – Input, keyboard completed (5000 mSec)
SYSTEM – Swapping processes (20 mSec)
PID 3 – Processing, (130 mSec)
SYSTEM – Swapping processes (25 mSec)
PID 1 – Processing (60 mSec)
SYSTEM – Managing I/O (20 mSec)
SYSTEM – Swapping processes (35 mSec)
PID 2 – Input, keyboard completed (5000 mSec)
SYSTEM – Swapping processes (35 mSec)
SYSTEM – Managing I/O (20 mSec)
PID 1 – Output, monitor started
SYSTEM – Swapping processes (35 mSec)
PID 2 – Processing (70 mSec)
```

```
SYSTEM – Swapping processes (20 mSec)
SYSTEM – Managing I/O (20 mSec)
PID 1 – Output, monitor completed
.
.
.
PID 1 – Processing (70 mSec)
SYSTEM – Swapping processes (20 mSec)
PID 2 – Exit system
SYSTEM – Ending process (40 mSec)
PID 1 – Exit system
SYSTEM – Ending process (40 mSec)
PID 3 – Exit system
SYSTEM – Ending process (40 mSec)
SYSTEM – Shut down management (150 mSec)
```

Overview of Assignments

(dates are subject to change but only under exaggerated circumstances):

First (design) assignment, **24 February, due 24 March**

- you will be randomly assigned to a group of 4; if the math doesn't quite work out, a smaller or larger group may be created at the discretion of the Instructor

- your group must generate a design that clearly shows the operation of the program at the structural level but can be translated into a program and major modules (functions)

- your group may create either the Low-Level simulator design or the High-Level simulator design; your group is not required to use your own design later on when you begin the program but you are required to use one of the designs that has been presented; note that groups that include graduate students are required to implement the High-Level design

- your group will receive a maximum of 85 (out of 100) points if you develop the Low-Level design and a maximum of 100 points (out of 100) if you develop the High-Level design

- it is strongly suggested that you create a graphical representation of the design along with possibly a textual design (no code allowed in either); your primary goal is completeness, correctness, clarity, and communication (although gratuitous alliteration is not required in the design)

- grading will come in two parts for this assignment: 1) The Instructor will give the assignments an initial grade depending on the clarity and completeness (from abstract idea to module design), and then 2) if any other group uses the design from your group, your design will earn 5 extra points for each user.

- upgrades to the design that significantly and appropriately improve the simulator will be considered for extra credit (not to exceed 10% of the total grade); however, your group should check with the Instructor before implementing these changes to make sure the Instructor considers them appropriate

- all students in the group will earn the same grade unless members of the group think this would be inappropriate; if that occurs, it is up to them to contact the Instructor and let the Instructor know the conditions

- this assignment will be turned in by uploading it to the Discussion Board **PA02-03/24** Thread at or before 3:30 pm on 24 March; note that materials uploaded after that date and time will not be graded. Each assignment will be posted for the whole group and the group number (and group name, if applicable) must be placed in the Subject Line of the message. The file containing the design materials may be zipped (tar or zip) as needed for upload. The group member names must be placed in the text area of the message along with any other supporting text deemed necessary. The Instructor will be grading this design but it will also be available for other students to review

Second (program) assignment, **24 March, due 9 April**

- you will again be randomly assigned to a different group of 4 with size adjustments made as needed

- on 24 March, you will observe a presentation of the designs developed by the groups in the class; you can choose your own or any of the ones provided; you will inform the Instructor via WebCampus email which design your team is using by Wednesday, 26 March, and you will be committed to that design from that point forward

- your group will receive a maximum of 85 (out of 100) points if you develop the Low-Level program and a maximum of 110 points (out of 100) if you develop the High-Level program; note that you do not have to develop the program you designed; you can choose to develop either the Low-Level or the High-Level simulator; also note that groups that include graduate students are required to implement the High-Level program

- your program must be eminently readable so that the grader can appropriately and correctly evaluate your code (see next assignment)

- upgrades to the program that significantly and appropriately improve the simulator will be considered for extra credit (not to exceed 10% of the total grade); however, your group should check with the Instructor before implementing these changes to make sure the Instructor considers them appropriate; in addition, these changes must accommodate themselves to the given design, which cannot be changed

- All students in the group will earn the same grade unless members of the group think this would be inappropriate; if that occurs, it is up to them to contact the Instructor and let the Instructor know the conditions

- for this assignment, each student in the group will upload the program files as if it was her or his own programming assignment, using each his or her own secret ID. The tar.gz file for each student must be zipped as was done in PA01 so that the file is named **PA1_<ID Code>.tar.gz** or as an example, **PA1_12345.tar.gz**. This means that if you were in a group of four students, there will be four identical programs uploaded to the WebCampus **PA03-04/09** assignment found in the **Week 9** folder; the difference will be that each file will have the secret ID number of the student uploading it. The programs must be uploaded at or before 3:30 pm on 09 April; assignments turned in after this date will not be graded and it is up to each individual student to verify that the file uploaded is correct and not corrupted. Again as was implemented on the PA01 assignment, your code must 1) unzip, then 2) make, then 3) run with any data provided by the evaluating student without any other modification

Third (evaluation) assignment, **11 April, due 21 April**

- this assignment will be conducted individually; the Instructor will divide all of the programs up and send them out to individual students no later than 11 April to be evaluated

- you will be provided a rubric for the assignments which you must follow but one of the components of the rubric will require that you re-create the design of the program you are evaluating into an eminently readable format making clear that the program was developed in an understandable way (and that you understand it); that said, if you can argue convincingly that the program is too hard to understand, your grade will not be reduced but the grades of the persons involved with the program will be reduced by 50%

- your (individual) grade for this will be a maximum of 100 (out of 100) points no matter which program you are required to evaluate

- for this assignment, you will print out, fill out, and then turn in the paper rubric to the Instructor's office; it can be slid under the door if the Instructor is not available

This assignment has been developed to provide you a quality experience of the design and operational decisions made by persons developing an Operating System. However, it also incorporates the real world* conditions of working on small teams and managing a somewhat large scale project

*real world is defined as advanced academia and/or industry