TCSS 422 — Operating Systems
Winter 2014 — Project 2

Due Date: Thursday, March 13

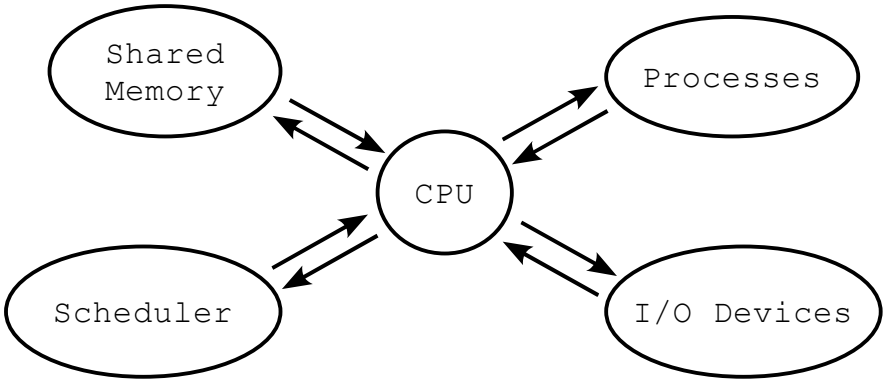
This project should be completed in groups of three to four students. You may choose to be in the same group for convenience or change groups with permission. The project deliverables (described below) should be submitted to the instructor electronically by the end of the day on the due date.

Project Description

In this project, your group will build a small operating system simulator, including a simulated CPU, processes, and process scheduler. As the simulation runs, the simulated processes will make simple system requests, such as for I/O or to access shared memory, which may cause the process to block. The simulation will display messages indicating which process is running, when service requests are generated, when an interrupt occurs, when a context switch occurs, and the state of each process on each context switch.

Program Design

The simulator will need several components. The CPU will be central to the simulation, as shown in this basic usage diagram:



This project will be implemented in C, using the `pthreads` library for thread control. Each hardware component that can take independent actions will be implemented as a thread in the simulator. In this design, only the CPU and I/O devices will have threads associated with them.

Process – You'll need to support several types of processes, including a user interface process (one waiting for keyboard input), a calculating process (possible background job), and two processes that require inter-process communications, such as a producer-consumer pair. Implement a general process ADT that can be extended by more specific types of process classes.

Each process essentially should be an endless loop of instructions of some fixed size (between 1,000 and 10,000 should work well). Most of the instructions will have no effect in the simulator (we're not simulating individual ADD, MOV, etc. instructions), but some of those instructions will be system calls for service requests, either for I/O or for communication, e.g. IPC. Each process should have different (perhaps randomly generated at startup) instruction addresses that perform system calls. From the point of view of a process, system calls will block further execution of the process until they are completed. For example, a system call that requests data from a keyboard device will not complete until the user has enter data via the keyboard. However, while a process is blocked, the simulated CPU can execute other runnable processes.

Each process will need an associated process control block with a minimum amount of information for the CPU and scheduler to use. **Note:** Individual simulated processes do not need individual threads. Processes are just data, it is the CPU that takes actions as directed by each process. One thread representing the CPU will "execute" all simulated processes.

CPU – The CPU both executes the currently running simulated process and handles low-level OS tasks, such as context switches and interrupts. The CPU will need, at a minimum, a PC (program counter) that records the address of the instruction currently being executed within the currently running process. When each instruction finishes, the PC is incremented, modulo the instruction size of the process, thus all processes will execute indefinitely.

The CPU must handle interrupts, which are generated by I/O devices. When an interrupt occurs, the PC will need to be stored in the interrupted process' PCB (this is part of a context switch) and the CPU must then handle the interrupt. For example, if a keyboard device produced an interrupt (signaling that the user typed a key) a process that was waiting for keyboard input, via a system call, may have its state changed from blocked to runnable.

I/O Devices – There should be at least two I/O devices in the simulated system that can be activated by system calls. The devices won't do much, just wait a short random interval and then generate an interrupt to signal that the system call is complete. One device should represent the keyboard and generate interrupts whenever the user types a key on the real physical keyboard.

System Timer – The system timer is an autonomous device that has a standard time delay, after which it generates an interrupt. These interrupts are used to trigger the scheduler, thereby supporting preemptive multitasking. For purposes of this simulation, you can set all process quanta to the same value.

Shared Memory – The only simulated memory will be memory shared among all processes. These shared global memory locations (which can hold simple `int`s) should behave as semaphores, and thus can be used to synchronize the IPC between two processes. The memory locations can be initialized when the simulation starts and be known to all processes in advance.

Scheduler – The scheduler will be called each time the CPU receives a system timer interrupt or a process makes a system call. It will determine what event activated it and then determine which runnable process to execute next (don't forget about the context switch). You should implement schedulers for the round-robin, priority, and lottery scheduling policies.

The user interface to your simulator should permit the user to select how many processes and of which types will be simulated, as well as which scheduling policy to use.

For an alternative description of this assignment, please review Prof. Mobus' version at:

http://faculty.washington.edu/gmobus/Academics/TCSS422/MoodleFiles/Project2.html

Sample Output

The following is sample output for a simulator with one user interface process, one calculating process, one pair of producer/consumer processes, one keyboard device, one auxiliary device, and using the round-robin scheduling policy (only system timer interrupts and blocking system calls cause a context switch, other interrupts do not). The display of the state for all processes is shown after the scheduler has chosen which process to run next. For this implementation, there are two system calls related to shared memory, one that decrements the value of a shared memory

location (and blocks if the value was originally 0), and one that increments the value of a shared memory location (and unblocks any processes blocked on that location), effectively implementing a semaphore.

```
Scheduler: Running process 0 (Calculator) next.
  Process 0 (Calculator)'s state: RUNNING
  Process 1 (UI)'s state: RUNNABLE
  Process 2 (Consumer)'s state: RUNNABLE
  Process 3 (Producer)'s state: RUNNABLE
Device 0 (System Timer) produced an interrupt.
Scheduler: Running process 1 (UI) next.
  Process 0 (Calculator)'s state: RUNNABLE
  Process 1 (UI)'s state: RUNNING
  Process 2 (Consumer)'s state: RUNNABLE
  Process 3 (Producer)'s state: RUNNABLE
Process 1 (UI) made a system call to device 1
(Keyboard).
Scheduler: Running process 2 (Consumer) next.
  Process 0 (Calculator)'s state: RUNNABLE
  Process 1 (UI)'s state: BLOCKED
  Process 2 (Consumer)'s state: RUNNING
  Process 3 (Producer)'s state: RUNNABLE
Process 2 (Consumer) attempted to decrement memory
location 0.
Scheduler: Running process 3 (Producer) next.
  Process 0 (Calculator)'s state: RUNNABLE
  Process 1 (UI)'s state: BLOCKED
  Process 2 (Consumer)'s state: BLOCKED
  Process 3 (Producer)'s state: RUNNING
Process 3 (Producer) incremented memory location 0.
Scheduler: Running process 3 (Producer) next.
  Process 0 (Calculator)'s state: RUNNABLE
  Process 1 (UI)'s state: BLOCKED
  Process 2 (Consumer)'s state: RUNNABLE
  Process 3 (Producer)'s state: RUNNING
Process 3 (Producer) incremented memory location 0.
Scheduler: Running process 3 (Producer) next.
  Process 0 (Calculator)'s state: RUNNABLE
  Process 1 (UI)'s state: BLOCKED
  Process 2 (Consumer)'s state: RUNNABLE
  Process 3 (Producer)'s state: RUNNING
Device 0 (System Timer) produced an interrupt.
```

```
Scheduler: Running process 0 (Calculator) next.
  Process 0 (Calculator)'s state: RUNNING
  Process 1 (UI)'s state: BLOCKED
  Process 2 (Consumer)'s state: RUNNABLE
  Process 3 (Producer)'s state: RUNNABLE
Process 0 (Calculator) made a system call to device 2
(Auxiliary).
Scheduler: Running process 2 (Consumer) next.
  Process 0 (Calculator)'s state: BLOCKED
  Process 1 (UI)'s state: BLOCKED
  Process 2 (Consumer)'s state: RUNNABLE
  Process 3 (Producer)'s state: RUNNABLE
Process 2 (Consumer) attempted to decrement memory
location 0.
Scheduler: Running process 2 (Consumer) next.
  Process 0 (Calculator)'s state: BLOCKED
  Process 1 (UI)'s state: BLOCKED
  Process 2 (Consumer)'s state: RUNNING
  Process 3 (Producer)'s state: RUNNABLE
Process 2 (Consumer) attempted to decrement memory
location 0.
Scheduler: Running process 2 (Consumer) next.
  Process 0 (Calculator)'s state: BLOCKED
  Process 1 (UI)'s state: BLOCKED
  Process 2 (Consumer)'s state: RUNNING
  Process 3 (Producer)'s state: RUNNABLE
Device 1 (Keyboard) produced an interrupt.
Scheduler: Running process 2 (Consumer) next.
  Process 0 (Calculator)'s state: BLOCKED
  Process 1 (UI)'s state: RUNNABLE
  Process 2 (Consumer)'s state: RUNNING
  Process 3 (Producer)'s state: RUNNABLE
Device 0 (System Timer) produced an interrupt.
Scheduler: Running process 3 (Producer) next.
  Process 0 (Calculator)'s state: BLOCKED
  Process 1 (UI)'s state: RUNNABLE
  Process 2 (Consumer)'s state: RUNNABLE
  Process 3 (Producer)'s state: RUNNING
```

Deliverables

The following items should be archived together, e.g. placed in a

`.zip` file, and submitted to the instructor.

1) All C source code. Do not include any extraneous files, such as Eclipse IDE files or object files.

2) A brief document describing how to use your simulator, e.g. descriptions of any command-line options, descriptions of expected user input, etc.

3) A brief document comparing the simulations of a specific combination of processes and devices under all three scheduling policies. For the combination of processes you've selected, does one of the scheduling policies utilize the CPU more efficiently than the others? Is there some property of the processes that explains this behavior?

Only one member of a group need submit the group's deliverable. Please make sure that group members' names appear somewhere within the deliverables.