CS240
Homework 4
Due on or before Dec 3 at 11:59:59 PM Pacific Time

For this homework you will build a program that simulates the kernel of an operating system. This simulator, called **ksim**, will simulate the creation, destruction, and movement of processes within an operating system. The model that we will be using for this simulator is the five-state process model as described in the course textbook. A graph depicting the five-state process model is shown in Figure 1 below.

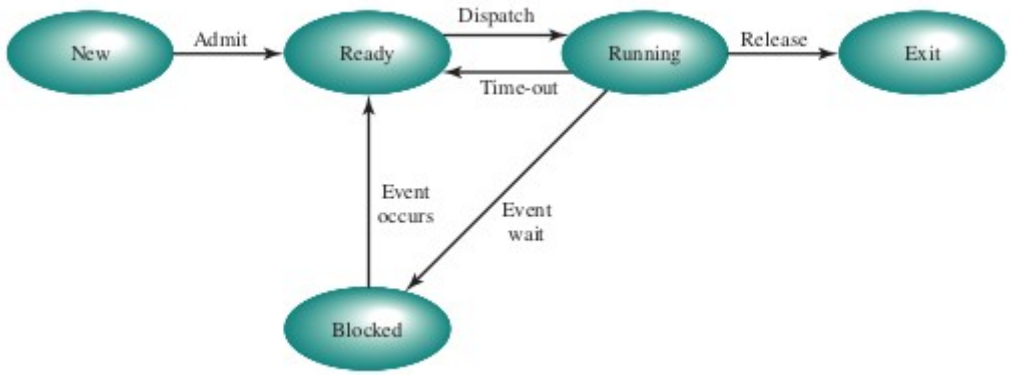

Figure 1: Five-State Process Model

The ksim simulator will be used to study existing operating systems and to explore innovations in future operating systems. The simulator will permit a user to create, destroy, and interact with processes and the kernel that is hosting them.

Overview

The primary objective of ksim is to simulate the kernel of an operating system. You are not supposed to build a kernel, but to build something that simulates how a kernel would behave. The unit of dispatch in this kernel will be a process. There will be no need to simulate threads. One of the primary duties of the simulator will be to keep track of processes and shuffle them around between queues based upon states and transitions shown in Figure 1 above.

Time

Scheduling is one of the primary tasks of any OS kernel. Time is of paramount importance when scheduling. The unit of time inside the kernel simulated by ksim is known as a *tick*. This unit of time is not fixed relative to "human" time, but will vary depending upon the resources available (e.g. processor clock speed) on the target platform. Certain operations in the kernel will consume ticks of time. Your simulator does not actually have to expend any amount of time associated with

these ticks, but must provide a representation to the user that these ticks have passed. The tick count is initialized to 0 when the simulator is started.

Commands

The user can enter commands into ksim to set up various scenarios and see how the kernel will behave. When ksim is launched it presents the user with a command prompt. The following shows ksim being launched and displaying the initial command prompt:

```
# ./ksim
ksim-000000000>
```

The ksim prompt is just the name ksim followed by a dash and the current tick. As previously mentioned, the tick count is initialized to 0 when ksim is launched. The current tick must be displayed in 9 decimal digits. This means that the highest value that can be represented in the tick counter is 999,999,999. This also means that the tick counter will "roll over" when 1,000,000,000 ticks have passed. Some commands cause ticks of time to pass. Other commands do not cause ticks to pass.

All ksim commands that you are required to implement are listed below. Each command consists of an opcode (a string of characters) and 0 or more operands.

`add <process-name>`
This command is entered by the user to create a new process. The process will be associated with the name <process-name>. The <process-name> operand is required and can be composed of any sequence of characters, but cannot include whitespace (spaces, tabs, vertical tabs, etc.) characters. If successful, this command will create a simulation process that will be placed into the New state and 32 ticks will pass, indicating the time the kernel consumed to create the new process. If a process named <process-name> already exists, your simulator must indicate that the process exists and not add any time to the tick counter. An example of this command is shown below.

```
ksim-000000000> add
Opcode "add" requires 1 operand.
ksim-000000000> add p0
ksim-000000032> add p0
Process named "p0" is already being hosted.
ksim-000000032> exit
#
```

`exit`
This command is entered by the user to terminate the simulation. This command has no operands. An example of this command is shown in the **`add`** command above.

`io-event <io-dev-num>`
This command is entered by the user to simulate that the I/O device specified by the operand <io-dev-num> is ready. The operand <io-dev-num> is a number in the range [0..3]. Since a process in this simulation can only be waiting on at most one I/O device, this will cause all processes that are waiting on this device to immediately become available and enter the Ready state. If no processes are waiting on the specified device, ksim will emit a message of the form "No processes waiting on device <io-dev-num>" (where <io-dev-num> will be replaced with the device number specified by

the user). This kernel consumes 1 tick when servicing this command. An example of this command is shown below.

```
#./ksim
ksim-000000000> add p0
ksim-000000032> add p1
ksim-000000064> step
Process "p0" moved from New to Ready.
Process "p0" moved from Ready to Running.
ksim-000000320> step
Process "p1" moved from New to Ready.
Process "p0" moved from Running to Ready.
Process "p1" moved from Ready to Running.
ksim-000000576> wait 0
Process "p1" moved from Running to Blocked.
ksim-000000577> step
Process "p0" moved from Ready to Running.
ksim-000000833> step
Process "p0" moved from Running to Ready.
Process "p0" moved from Ready to Running.
ksim-000001089> io-event 0
Process "p1" moved from Blocked (iodev=0) to Ready.
ksim-000001090> step
Process "p0" moved from Running to Ready.
Process "p1" moved from Ready to Running.
ksim-000001346>
```

query <process-name>

This command is entered by the user to cause ksim to print information regarding the state of process <process-name>. If the operand <process-name> is "all" (without quotes) or is not present on the command line, ksim will print detailed information about all processes that are currently being hosted. If the name specified by <process-name> is not associated with a current process, ksim will emit a message indicating such. The kernel consumes no ticks when servicing this command. An example of the command is shown below.

```
ksim-000000579> query p0
***
    id: "p0"
    state: "Ready"
***
ksim-000000579> query all
***
    id: "p0"
    state: "Ready"
***
***
    id: "p1"
    state: "Blocked"
        waiting on device 0 since tick 000000578
***
ksim-000000579> query Dino
Process "Dino" not found.
ksim-000000579>
```

`release`

This command is entered by the user to cause the currently running process to move from state Running to state Exit. This command takes no operands. If there is a process in the Running state when this command is entered, the kernel consumes 32 ticks servicing this command. If no process is in the Running state when this command is entered, the kernel consumes no ticks servicing this command. An example of this command is shown below.

```
# ./ksim
ksim-000000000> add p0
ksim-000000032> add p1
ksim-000000064> release
No process is currently Running.
ksim-000000064> step
Process "p0" moved from New to Ready.
Process "p0" moved from Ready to Running.
ksim-000000320> release
Process "p0" moved from Running to Exit.
ksim-000000352> step
Process "p0" is banished to the void.
process "p1" moved from New to Ready.
Process "p1" moved from Ready to Running.
ksim-000000608>
```

`step`

This command is entered by the user to cause ksim to step to perform housekeeping duties and advance the simulation. This command has no operands. The housekeeping duties performed during this command are:

1. Remove from the system all processes that are in the Exit state. Once a process is removed from the system it can no longer be queried in the simulator. For each process that is removed from the system, ksim will emit a message of the form “process <process-name> is banished to the void.” where <process-name> is the name that was associated with the process when it was created using the **`add`** command.

2. Advance at most 1 process in the New state into the Ready state. If more than 1 process exists in the New pool, the process with the oldest creation time is advanced from New to Ready.

3. Advance at most 1 process from each I/O device in the Blocked state into the Ready state. This can only occur if the process that is Blocked has waited for the specified number of ticks associated with an I/O device wait (1024 ticks). See the **`wait`** command for more details. Please note that this is independent of the **`io-event`** command, which causes all processes that are waiting for a specified I/O device to immediately advance to the Ready state.

4. If a process is currently in the Running state, remove the process from the processor and place it in the Ready state.

5. If any processes are in the Ready state, dispatch a process to the processor by advancing it from the Ready state to the Running state. If more than 1 process is in the Ready state, the process that has waited the longest amount of time since last being run is advanced from the Ready state to the Running state.

If a process was dispatched to the processor by being advanced from the Ready state to the Running state in item 5 above, this command consumes 256 ticks. Otherwise this command consumes 1 tick. An example of this command is shown below.

```
# ./ksim
ksim-000000000> add p0
ksim-000000032> add p1
ksim-000000064> release
No process is currently Running.
ksim-000000064> step
Process "p0" moved from New to Ready.
Process "p0" moved from Ready to Running.
ksim-000000320> release
Process "p0" moved from Running to Exit.
ksim-000000352> step
Process "p0" is banished to the void.
Process "p1" moved from New to Ready.
Process "p1" moved from Ready to Running.
ksim-000000608> add p1
Process named "p1" is already being hosted.
ksim-000000608> add p0
ksim-000000640> step
Process "p0" moved from New to Ready.
Process "p1" moved from Running to Ready.
Process "p0" moved from Ready to Running.
ksim-000000896> step
Process "p0" moved from Running to Ready.
Process "p1" moved from Ready to Running.
ksim-000001152> step
Process "p1" moved from Running to Ready.
Process "p0" moved from Ready to Running.
ksim-000001408>
```

`wait <io-dev-num>`

This command is entered by the user to cause the currently running process to block waiting on the I/O device specified by the operand <io-dev-num>. The operand <io-dev-num> is a number in the range [0..3]. When a process is blocked waiting for a device, it will wait for 1024 ticks or until the user enters an **`io-event`** command signaling that the device is ready. Multiple processes can be waiting on the same I/O device. If no process is currently in the Running state or if operand <io-dev-num> is not within the specified range, ksim will emit a message indicating such and the kernel will consume no ticks for this command. If a process is currently in the Running state and operand <io-dev-num> is valid, the kernel consumes 1 tick for this command. An example of this command is shown below.

```
# ./ksim
ksim-000000000> add fred
ksim-000000032> add barney
ksim-000000064> add wilma
ksim-000000096> step
Process "fred" moved from New to Ready.
Process "fred" moved from Ready to Running.
ksim-000000352> step
Process "barney" moved from New to Ready.
Process "fred" moved from Running to Ready.
Process "barney" moved from Ready to Running.
ksim-000000608> step
Process "wilma" moved from New to Ready.
Process "barney" moved from Running to Ready.
Process "wilma" moved from Ready to Running.
ksim-000000864> step
```

```
Process "wilma" moved from Running to Ready.
Process "fred" moved from Ready to Running.
ksim-000001120> wait 0
Process "fred" moved from Running to Blocked.
ksim-000001121> query
***
    id: "fred"
    state: "Blocked"
        waiting on device 0 since tick 000001120
***
***
    id: "barney"
    state: "Ready"
***
***
    id: "wilma"
    state: "Ready"
***
ksim-000001121> step
Process "barney" moved from Ready to Running.
ksim-000001377> step
Process "barney" moved from Running to Ready.
Process "wilma" moved from Ready to Running.
ksim-000001633> step
Process "wilma" moved from Running to Ready.
Process "barney" moved from Ready to Running.
ksim-000001889> step
Process "barney" moved from Running to Ready.
Process "wilma" moved from Ready to Running.
ksim-000002145> step
Process "fred" moved from Blocked (iodev=0) to Ready.
Process "wilma" moved from Running to Ready.
Process "fred" moved from Ready to Running.
ksim-000002401>
```

Requirements

Your simulator must be built in C or C++. You must define a structure/class to represent a process and a queue and can define any other structures/classes that you need. All processes and queues in your simulator must be allocated dynamically on the heap. Your simulator must not impose an upper bound on the number of processes in a queue, the number of processes that can be active at any given time within the simulation, or anything else in the simulation. Your simulator must not have any memory leaks. You must provide a complete, working Makefile that builds your simulator on the course server. Your simulator will be tested on the course server. You must package all source code files and your Makefile into an uncompressed tar archive named “hw4.tar” (without quotes). Your hw4.tar archive must be submitted via *cscheckin*. If your submission does not successfully build on the course server **for any reason** at least 70% of the total points possible on this assignment will be deducted from your score. There will be no exceptions to this.

Hints

Carefully study the narrative in this homework. Try to make your simulator match the behavior shown as closely as possible.

This is not a difficult assignment. Do not make this problem more difficult than it actually is. Care has been taken to provide constraints that limit the amount of work that must be done to complete

this assignment. You are to build a simulator, not a kernel. As long as the user is presented with the illusion that processes are being manipulated correctly, your simulator will be well received.

Build this assignment incrementally. If you attempt to build this assignment in one sitting, you will be disappointed with the result. Give yourself enough time to do your best on this assignment. If you wait until two days before the assignment is due to start it, you will be disappointed with the result. If you give yourself enough time to work through the problems that are presented in this assignment, your ability to build quality software will be greatly improved and you will be on a good trajectory for future CS courses.

Test your assignment on the course server. Everyone has their favorite toolchain and development environment. It is fine to use another computer/toolchain/environment to build your assignment, but you **must** test your assignment on the course server. Test your assignment incrementally. Testing is part of a commitment to developing quality software. If you wait until the day before the assignment is due to start testing your simulator on the course server, you are likely going to be disappointed with the result.