CSCE 311
Spring 2020
Project # 2
Assigned: Feb. 11, 2020
Due: Feb. 25, 2020 (timestamp must be earlier than the start of class)
N.B.: Late assignments will be penalized 10% per day

Objective: To implement CPU scheduling on the OSP2 simulator. You will implement the methods in module `ThreadCB.java` and the single method `do_handleInterrupt()` in `TimerInterrupHandler.java` to further your understanding of CPU scheduling. You are required to implement the RR scheduling algorithm.

Required to turn in: Follow all directions regarding hand-in procedures. **Points will be deducted if you do not follow directions precisely.**

1. You must also submit an electronic copy of your `RR` solution i.e. `ThreadCB.java and TimerInterrupHandler.java` (or your best try) via dropbox.
2. You must document your code. Each java file should include your name and email address at the top of the file.
3. You must provide a one-page explanation of how you accomplished the assignment (or what you have currently and why you could not complete). In this one page write-up you should describe your use, creation, and manipulation of data structures to accomplish the assignment. Also submit this via dropbox.

Building and executing the simulation

In OSP2, CPU scheduling involves the functions in file `ThreadCB.java` and `TimerInterrupHandler.java`. Chapter 4 of the OSP2 textbook describes how OSP2 interacts with the functions in `ThreadCB.java`, specifically `do_create` (which creates a thread), `do_kill` (which kills a thread and potentially the task), `do_suspend` (which modifies a threads waiting status and suspends it if it isn't already suspended), `do_resume` (which modifies a threads waiting status and moves it to the ready queue if the waiting status changes to ready), and `do_dispatch` (which removes a thread from the queue and dispatches it). These methods are where you will be concentrating your efforts. Minimize addition of global variables, if needed they should be placed just prior to `init()`. You will need to use `init()` (which is called when OSP2 is set up) to initialize any of your global variables.

Download the archive file containing the files for this project. **For unix or linix:** If you are using a unix or linix machine then you will probably want to download the tar file: Threads.tar. Download the archive and extract the files using the command **tar -xvf Threads.tar**

For a windows box: If you are using a windows box, then you will probably be happier with a zipped folder: Threads.zip. Download the archive and then extract the files by right-clicking on the file and selecting the "extract all…" option from the popup menu.
For a Mac: If you are using a Mac then you already know how to figure out which file you want to download and how to extract the files.

You should have extracted the following files:

```
Threads/Demo.jar
Threads/Makefile
Threads/Misc
Threads/OSP.jar
Threads/ThreadCB.java
Threads/TimerInterrupHandler.java
Threads/Misc/params.osp
Threads/Misc/wgui.rdl
```

As per the discussion in the OSP2 text, `Makefile` is for use in unix and linix systems. The demo file `Demo.jar` is a compiled executable that does `RR` scheduling. The only files you should have to modify are `ThreadCB.java` and `TimerInterrupHandler.java`. Modifying the other files will probably "break" OSP2. Compile the program using the appropriate command for your environment (unix/linix/windows).

(unix) `javac –g –classpath .:OSP.jar: -d . *.java`

(windows) `javac –g –classpath .;OSP.jar; -d . *.java`

This will create an executable called `OSP`. Run the simulator with the following command:

(windows) `java –classpath .;OSP.jar osp.OSP`

(unix) `java –classpath .:OSP.jar osp.OSP`

Discussion

Module: ThreadCB.java

General: need a data structure for the ready queue

Suggestion: linked list OSP2 provides a linked list “GenericList”. See section 1.5 for utility classes and page 14 specifically for the GenericList class.

Context Switching

Note: do { read pages 65-66 carefully } until you reach enlightenment

Context switching: passing control of the CPU from one thread to another.
Two parts:

1 preempting the current thread
2 dispatching another thread

Preempting the current thread

1. Determine which thread is the current thread. The page table base register (PTBR) points to the page table of the current thread. We can use this information to figure out which thread is running. The PTBR is contained in the memory management unit (MMU)
    a. The call MMU.getPTBR() returns the page table of the current thread.
        Note: if no thread is running then this call will return null.
    b. Applying the method getTask() to the page table returns the task that owns the thread
    c. Applying the method getCurrentThread() to the task returns the current thread
    d. Altogether: MMU.getPTBR().getTask().getCurrentThread()
    e. If you attempt the statement in step d) you had better put it in a try-catch construct to catch the potential NullPointerException

2. Change the state of the current running thread from ThreadRunning to the appropriate new state such as ThreadReady or ThreadWaiting.

3. Set the page table base register to null. Use the MMU method setPTBR(null)
4. Let the task know that its thread is no longer running by setting the running task to null: use the task method setCurrentThread(null)

Dispatching another thread

1. Select a thread *t* to run.
2. If there is no thread then leave PTBR set to null (from step 3 above).
3. Set PTBR to point to the page table of the task that owns the thread to be dispatched. Use the thread method getTask() to get the task owning the thread. Use task method getPageTable() to get the page table of the task owning the thread. Use the MMU method setPTBR() to set the PTBR.
4. Let the task know that its thread is the current thread using the task method setCurrentThread(*t*).
5. If you are performing RR scheduling, you would set the interrupt timer with the quantum.

ThreadCB methods:

init() – This method is called once, at the beginning of the simulation. Use it to initialize any variables or data structures that you will be using, e.g., the readyqueue.

do_create(TaskCB task) – create a thread.

1. Note: we want to call the dispatcher before exiting this method no matter which path we take.

2. First check if task is null – yes → call dispatcher and return null

3. Does the task already have the max # of threads (MaxThreadsPerTask)? Use task.getThreadCount()

3a. yes → call dispatcher and return null
3b. no, create the thread.

4. Set up the thread priority and status: setPriority() of thread/getPriority() of task
5. Set up status: SetStatus() to ThreadReady
6. Associate the task with the thread: setTask(task)
7. Associate the thread with the task: addThread(thread) If this fails thencall dispatcher and return null
8. Append new thread to readyqueue
9. Call dispatcher
10. Return thread

do_kill() – destroys a thread

1. Get status of thread.
2. If thread status is ThreadReady then remove it from the ready queue with a call to GenericList method remove()
3. If thread status is ThreadRunning then preempt it (see page 65 on context switching). First verify that it is the current thread:
    a. Compare "this" with what OSP2 thinks is the current thread. This is the thread whose page table is currently loaded:
    b. See page 65 for detail on how to do this.

   If it is the current thread then set the PTBR to null with setPTBR()
4. Remove the task from the thread (first need to get the task using getTask())
5. Set the thread status to ThreadKill.
6. Loop through the device table (Device.get(i)) to purge any IORB associated with this thread (cancelPendingIO())
7. Release all resources: ResourceCB.giveupResources()
8. Call dispatch
9. Check if the task has any threads left. If not, then invoke the Task kill() method to kill the task.

do_suspend(Event event) – suspends/increments the ThreadWaiting status of a thread

1. First we need to know what the current state of the thread is: getStatus()
2. If it is running, then suspend it.
    a. To be sure that it is running you need to find out which thread OSP thinks is the current running thread (see page 65)
    b. If this is the thread then it must lose control of the CPU.
    c. Let its task know that it is not the current thread by invoking setCurrentThread(null).
3. If its status is ThreadRunning then change its status to ThreadWaiting using setStatus().
4. If it is already waiting then increment its waiting status. To test waiting status compare getStatus()>= ThreadWaiting.
5. Make sure it is not in the ready queue.
6. add this thread to the event queue using addThread(thread)
7. call dispatch to dispatch another thread

do_resume() – resumes/decrements the ThreadWaiting status of a thread. The code for this method is on page 41 of the OSP2 text.

do_dispatch() – selects a thread to dispatch. For this first part of the project use RR scheduling.

1. Check if there is currently a running thread (page 65).
2. If so and the quantum has expired or the thread state is not ThreadRunning:
    a. Let its task know that it is not the current thread by invoking setCurrentThread(null).
    b. take away cpu control by setting the PTBR to null using setPTBR(null).
    c. set the thread's status to ThreadReady
    d. place it in the ready queue
    e. Since we are using RR scheduling, select the thread at the head of the ready queue
    f. If the ready queue is empty then set the PTBR to null and return FAILURE
    g. o/w set the PTBR to point to the thread's page table
    h. Set the thread as the current thread of its task using setCurrentThread(thread)
    i. Set the threads status to ThreadRunning
    j. Since you are using RR scheduling, you should set the interrupt timer using a quantum of 50.
3. O/W if there is a current thread, let the current thread finish its quantum, i.e., do not dispatch a different thread.
4. If was no current thread:
    a. Select the thread at the head of the ready queue
    b. If the ready queue is empty then set the PTBR to null and return FAILURE
    c. o/w set the PTBR to point to the thread's page table
    d. Set the thread as the current thread of its task using setCurrentThread(thread)
    e. Set the threads status to ThreadRunning
    f. Since you are using RR scheduling, you should set the interrupt timer using a quantum of 50.
5. Return SUCCESS

How do I get an A on this assignment?

This is very simple to do. Implement `ThreadCB.java` as well as `TimerInterrupHandler.java` following the algorithm just given for `RR` and hand in everything specified in the "Required to turn in" section on time. Your solution must prevent process starvation and should complete correctly. It should also be well documented so the grader can understand your implementation decisions. For an example of how a correct solution might look, run the demo version that you copied to your directory earlier. Your solution should have no core dumps and should complete successfully with no warnings or aborts of the OSP2 simulator.

Some more hints:

- The outlined algorithm for the solution to CPU scheduling is derived from a discussion

in your OSP2 book regarding the CPU scheduling module (Chapter 4 pgs. 57-73). Your OSP2 book (and the Silberschatz book) are your best references for conceptual questions on implementation of `ThreadCB.java` and `TimerInterrupHandler.java`.

- Re-read section 1.11 on debugging if you are having a difficult time debugging your code.
- You can also create your own functions to print out useful information if you find it necessary. Keep in mind you want to build up your skill set so you are prepared for the next assignment that will be more challenging.
- Survival hint: Since you need to turn in something that compiles and runs, you may want to work up to the final solution in stages.

Turning in your assignment via dropbox
When you are satisfied that your CPU scheduling algorithm works, use dropbox to submit your project. Use dropbox to submit the following files:

1. `ThreadCB.java`
2. `TimerInterrupHandler.java`
3. `Your 1 page writeup.`

Turning in your write-up
You must provide a one-page explanation (***using dropbox***) of how you accomplished the assignment (or what you have currently and why you could not complete). In this one page write-up you should describe your use, creation, and manipulation of data structures to accomplish the assignment

Remember to turn your assignment in on time. Late submissions will be accepted but penalized by 10% per day. Also, be sure to submit all required files and that the files that you submit are the files you intend to submit. If you mistakenly submit an incorrect file and then re-submit after the deadline you will receive a late penalty.