CSCE 311

Programming Assignment #2

Logistics

This is the second Programming assignment for the course. This will be due on the department dropbox (https://dropbox.cse.sc.edu/login/index.php).

Objective

The objective of this assignment is 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 FCFS 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 FCFS 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.
4. Submit a tarball to dropbox. Assuming the folder with your project is "Project2"
    a. Linux
    ```
    tar cvf Project2.tar Project2
    gzip Project2.tar
    ```
    b. Windows
    http://gettingeek.com/how-to-create-tarball-compress-to-gzip-under-windows-tar-gz-379.html

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 the addition of any 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 Linux:** If you are using a Unix or Linux 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 Windows: 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 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 Linux 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/Linux/Windows) or use make.
(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:
(Unix) java –classpath .:OSP.jar osp.OSP
(Windows) java –classpath .;OSP.jar osp.OSP

Discussions/Programming Hints

Start by reading pages 65-66 in the OSP2 manual ***carefully*** until you reach enlightenment. It is very difficult to code something that you don't understand. This might take more than one pass.

Once you understand the goal, start with the basic structures you will need. For example, you will need some sort of data structure for the ready queue. If you read Chapter 1 of the OSP2 Manually carefully, you'll notice that OSP2 provides a generic linked list called "GenericList". If you didn't read Chapter 1, then you'll want to look over Section 1.5 in the OSP 2 manual and specifically page 14 for more details about the GenericList class.

Context Switching

Context Switching involves passing control of the CPU from one thread to another thread. This involves two separate actions.

1. Pre-empting the current/running thread.
2. Dispatching another thread.

Pre-empting the Current Thread

1. Determine which thread is the current thread. The page table base register (PTBR) point to the page table of the current thread. Use this information to figure out which thread is running. The PTBR is contained in the memory management unit (MMU).
    - Call MMU.getPTBR() to get the page table of the current thread.
    - Check to see if this is null. A null value indicates that there is no thread that is currently running.
    - Call getTask() on the page table to find out which task owns the thread.
    - Call getCurrentThread() on the task to find out what the current thread is.
    - Don't forget to put in the appropriate error checking. Either use a try/catch block to catch potential NullPointerExceptions if you're doing something like MMU.getPTBR().getTask().getCurrentThread() or make sure you check the result of each call to see if it's null.
2. Change the state of the current thread from ThreadRunning to the appropriate new state (ThreadReady, ThreadWaiting, etc.).
3. Set the page table base register to null.
    - User setPTBR(null) from MMU.
4. Set the running task to null to let the task know that the thread is no longer running.
    - Use setCurrentThread(null)

Dispatching a Thread

1. Select a thread to run.
2. If there is no thread to select, then leave PTBR set to null (see above).
3. Set PTBR to point to the page table of the task that owns the thread to be dispatched.
    - Call getTask() to get the task that owns the thread.
    - Call getPageTable() on the task to get the page table for that task.
    - Call setPTBR() on the MMU to set the PTBR.
4. Set the current thread by calling setCurrentThread(newThread) on the task.
5. If you were using RR Scheduling, you would also need to set the interrupt time with some fixed amount.

ThreadCB Methods

init(): Called once at the beginning of the simulation. Anything that needs to be initialized (variables, data structures like the ready queue, etc.) should be initialized here. Always initialize your variables!

do_create(TaskCB task): Creates a thread.

Note: Call the dispatcher before exiting this method NO MATTER WHICH path is taken.

1. Check to see if task is null. If it is, call the dispatcher and return null.
2. Does the task already have the maximum number of threads (MaxThreadsPerTask)? You can find this by using task.getThreadCount().
    a. If count > max then call dispatcher and return null.
    b. Else create the thread.
3. Set the thread priority and status: setPriority() of thread and getPriority() of task

4. Set the status: setStatus() to ThreadReady
5. Associate the task with the thread: setTask(task)
6. Associate the thread with the task: addThread(thread). If this fails, then call the dispatcher and return null.
7. Append a new thread to the Ready Queue.
8. Call the dispatcher.
9. Return the thread.

do_kill(): Destroys a thread.

1. Get the status of the thread.
2. If the thread status is ThreadReady, then remove it from the Ready Queue. If you used the GenericList class then you can use remove().
3. If the thread status is ThreadRunning, then preempt it. Instructions are above or on page 65 of the OSP2 Manual because you need to get the current thread from memory.
   - If it is the current thread, then you'll need to set the PTBR to null using setPTBR.
4. Remove the thread from its task using 'this.getTask().removeThread(this);'. Note: this needs to happen even if regardless of the earlier steps.
5. Set the thread status to ThreadKill.
6. Loop through the device table to purge any IORB associated with this thread. You can/should use cancelPendingIO() for this.
7. Release any resources currently in use using 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
5. getStatus()>= ThreadWaiting.
6. Make sure it is not in the ready queue.
7. Add this thread to the event queue using addThread(thread)
8. Call dispatch to dispatch another thread

do_resume(): Resumes/decrements the ThreadWaiting status of a thread. See page 59 of the OSP2 text for more details.

do_dispatch(): Selects a thread for dispatch. In this assignment, use FCFS scheduling.

1. Check to see if a thread is currently running. You should be familiar with how to do this at this point. If so:

   a. Let it’s task know it’s not the current thread anymore by calling 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.
2. If the ready queue is empty then set the PTBR to null and return FAILURE.
3. Since we are using FCFS scheduling, select the thread at the head of the ready queue.
4. Set the PTBR to point to the thread’s page table
5. Set the thread as the current thread of its task using setCurrentThread(thread)
6. Set the threads status to ThreadRunning
7. If we were using RR scheduling, you would set the interrupt timer at this point.
8. Return SUCCESS

How do I Earn an A?

This is very simple to do. Implement ThreadCB.java as well as TimerInterrupHandler.java following the algorithm just given for FCFS 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. (Keep in mind that the demo is performing round robin and not FCFS.) Your solution should have no core dumps and should complete successfully with no warnings or aborts of the OSP2 simulator.

Additional 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.

Write up

You must include with your code a one page write up of how you accomplished your assignment (or what you have currently and why you couldn’t get it working). In this write-up, you should describe your use, creation and manipulation of data structures used in the assignment as well as your through process as you sent about solving the problem. Include at the bottom a list of any websites or books used (besides the text) while coming to your solution.

Submission

Submit all of your files in a tarball to dropbox. You can create a tarball using 7-zip on Windows (instructions at https://linhost.info/2012/08/gzip-files-in-windows/) or by running the following commands in Linux:

```
tar cvf Project2.tar Project2
gzip Project2.tar
```

This will take all of the files in the Project2 directory and put them in a tar file called Project2.tar. The second command will gzip that tar file. Ensure that you include the following:

1. ThreadsCB.java
2. TimerInterruptHandler.java
3. Your 1-page write-up.

Late submission

You are encouraged to turn in your assignment on time. Late submission will be accepted for 5 days at a penalty of 10% per day. Make sure you have included all of the required files. “Forgotten” files will receive a late penalty.