CSE 120 - Discussion session 2

April 15, 2019

Logistics

- Homework 1 - April 16
- Gradescope for Homeworks

- Project 1 - April 24
- Git repo for projects
- Submission - whatever is on your repo on the day of the deadline at 11:59 pm

Threads - Recap

How to create a thread?

- All nachos threads that run kernel code - instances of the KThread class
- TCB object contained within each thread
  - Low level support for context switches, creation and destruction of threads
- To create a thread:

```
1  Runnable myRunnable = new Runnable() {
2          public void run() {
3              myFunction();
4          }
5          };
6  KThread newThread = new KThread(myRunnable);
7
8  newThread.fork();
```

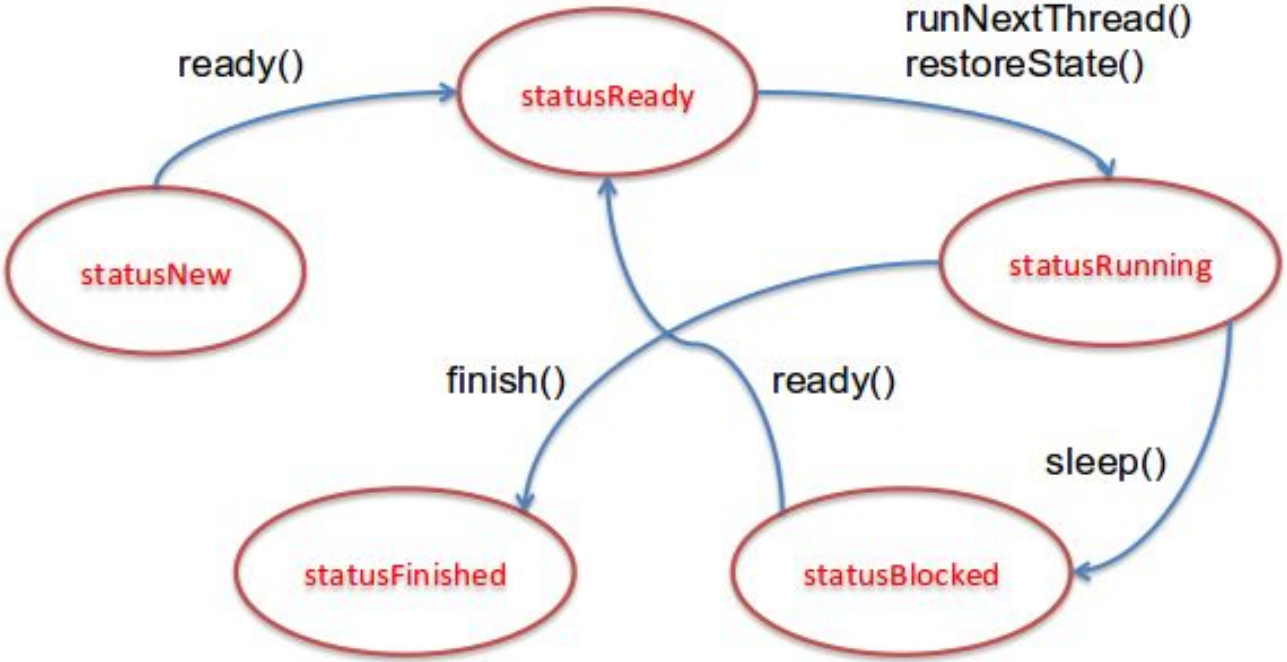
Main thread and Idle thread

- First KThread - initialize() method of ThreadedKernel
- Creates the main thread and the idle thread
- Main thread
  - treated normally by the scheduler
- Idle thread
  - Never added to ready queue
  - Only if readyQueue.nextThread() returns null, thread system switches to idle thread

Thread lifecycle

States:

1. New
2. Ready
3. Running
4. Blocked
5. Finished



Project 1

Problems

1. Alarm - waitUntil() method
2. KThread - join() method
3. Condition variables
4. Scheduled wait on Condition Variables
5. squadMatch

How to test?

1. Write a test method - myTest()
2. Call myTest() from the class.selfTest() method
3. Call class.selfTest() from ThreadedKernel.selfTest()
4. Check Testing section on project page
5. Test thoroughly!

Problem 1 - Alarm.waitUntil(x)

1. Complete the implementation of the `Alarm` class. A thread calls `waitUntil(long x)` to suspend its execution until wall-clock time has advanced to at least *now* + *x*. There is no requirement that threads start running immediately after waking up; just put them on the ready queue in the timer interrupt handler after they have waited for at least the right amount of time. You need only modify `waitUntil` and the timer interrupt handler methods. If the wait parameter `x` is 0 or negative, return without waiting (do not assert).

Understanding the problem

What should happen when a thread A calls waitUntil(x)?

- A must be blocked for at least x ticks from now

Let current time be Y. Let thread A call waitUntil(x).

- Till what time must Thread A wait?

One possible implementation:

```
1  public void waitUntil(long x) {
2      // for now, cheat just to get something working (busy waiting is bad)
3      long wakeTime = Machine.timer().getTime() + x;
4      while (wakeTime > Machine.timer().getTime())
5          KThread.yield();
6  }
7
```

Busy waiting is not good! Why?

Hints

In waitUntil(x):

- Compute wakeTime of A.
- Add A to wait queue - implement using appropriate data structure

In timerInterrupt():

- Check all threads that have called Alarm.waitUntil(x)
- Check if any thread is ready to be woken up(unblocked) - use the wakeup time computed in waitUntil(x),
- Change its status to *ready* in order to unblock it

Problem 2 - KThread.join()

2. Implement KThread.join which synchronizes the calling thread with the completion of the called thread.

Understanding the problem

Let A and B be two threads. Let A call B.join()

What does this mean?

- Block A until B is finished
- Unblock A once B is finished

Hints

1. How to block a thread?
    - Check the sleep() method
2. How to unblock a thread?
    - Check the ready() method
3. How will we know whether B has finished or not?
    - Busy waiting?
    - Check the finish() method
4. Which is the *currentThread* and which is *this*?
    - *this* - thread B
    - *currentThread* - thread A