**Task 3: Alarm**

**Introduction**

Alarm uses the hardware timer for preemption by calling its timerInterrupt() method around every 500 clock ticks, which causes the current thread to yield to another thread if there is one available. It also contains a waitUntil(long x) method, which yields a thread continuously until a certain time (x) has passed.

**Methods to change/add**

*waitUntil(long x):* We will change this method by blocking the calling thread until the time is “x + now”, instead of using its current busy-waiting implementation.

*WaitThread(long x, KThread):*

After blocking a calling thread through the waitUntil() method, the thread will be added to a “sleeping queue” along with its wake-up time. The purpose of the WaitThread class is simply to combine a sleeping thread with its wait time into one object. We will also create a priority queue using a linked list called *sleepQueue*, where we will place these WaitThread objects. WaitThread objects will be organized in the *sleepQueue* by wake-up time (shortest first) when they are added through priority queue methods.

*timerInterrupt():*

Instead of simply calling yield on the current thread every 500 ticks, this method when called will now first look at the *sleepQueue* (see above) and wake any threads that have met or exceeded their sleep time. (it will then call yield as it did previously).

**Pseudocode – WaitThread Class**

WaitThread takes the wake-up time (wakeTime) and a sleeping KThread (waitThread) as arguments.

It includes two simple methods:

KThread getWaitThread()

return waitThread;

long getWaitTime()

return waitTime;

**Pseudocode – waitUntil**

long wakeTime = Machine.timer().getTime() + x; //not changed

currentThread -> sleep

create WaitThread object with wakeTime and currentThread as arguments

insert WaitThread object into *sleepQueue*, using the .getWaitTime() for comparing priority

**no longer used:**

~~while (wakeTime > Machine.timer().getTime())~~

~~KThread.yield();~~

**Pseudocode – timerInterrupt**

For each WaitThread object in sleeepQueue,

if (head WaitThread.getWaitTime() =< current time)

wake up head thread using WaithThread.getWaitThread().ready()

remove WaitThread object from sleepQueue and put next in line as new head

else

endForLoop //since lower priority threads will automatically fail the if test

**still used after the above code:**

KThread.currentThread().yield();

**Test Cases**

Test with negative and 0 values for the time that a thread sleeps. Will it still work and simply run immediately once timerInterrupt is called?

Test with multiple threads with multiple sleeping time values in out-of-order fashion and see if the queue is properly ordered once it runs – could possibly use a print function here when waking up a thread that prints the thread’s expected wake-up time.

Does timerInterrupt still function properly when the sleepQueue is empty?

Have two or more threads with similar wake-up times in the queue to make sure that more than one thread is woken up during a single timerInterrupt call.