Justin Shuck

CS333 – Fall 2014

Proj 2 – Synch.c

######################################################################

code Synch

-- OS Class: Project 2

--

-- Justin Shuck

----------------------------- Semaphore ---------------------------------

behavior Semaphore

-- This class provides the following methods:

-- Up() ...also known as "V" or "Signal"...

-- Increment the semaphore count. Wake up a thread if

-- there are any waiting. This operation always executes

-- quickly and will not suspend the thread.

-- Down() ...also known as "P" or "Wait"...

-- Decrement the semaphore count. If the count would go

-- negative, wait for some other thread to do an Up()

-- first. Conceptually, the count will never go negative.

-- Init(initialCount)

-- Each semaphore must be initialized. Normally, you should

-- invoke this method, providing an 'initialCount' of zero.

-- If the semaphore is initialized with 0, then a Down()

-- operation before any Up() will wait for the first

-- Up(). If initialized with i, then it is as if i Up()

-- operations have been performed already.

--

-- NOTE: The user should never look at a semaphore's count since the value

-- retrieved may be out-of-date, due to other threads performing Up() or

-- Down() operations since the retrieval of the count.

---------- Semaphore . Init ----------

method Init (initialCount: int)

if initialCount < 0

FatalError ("Semaphore created with initialCount < 0")

endIf

count = initialCount

waitingThreads = new List [Thread]

endMethod

---------- Semaphore . Up ----------

method Up ()

var

oldIntStat: int

t: ptr to Thread

oldIntStat = SetInterruptsTo (DISABLED)

if count == 0x7fffffff

FatalError ("Semaphore count overflowed during 'Up' operation")

endIf

count = count + 1

if count <= 0

t = waitingThreads.Remove ()

t.status = READY

readyList.AddToEnd (t)

endIf

oldIntStat = SetInterruptsTo (oldIntStat)

endMethod

---------- Semaphore . Down ----------

method Down ()

var

oldIntStat: int

oldIntStat = SetInterruptsTo (DISABLED)

if count == 0x80000000

FatalError ("Semaphore count underflowed during 'Down' operation")

endIf

count = count - 1

if count < 0

waitingThreads.AddToEnd (currentThread)

currentThread.Sleep ()

endIf

oldIntStat = SetInterruptsTo (oldIntStat)

endMethod

endBehavior

----------------------------- Mutex ---------------------------------

behavior Mutex

-- This class provides the following methods:

-- Lock()

-- Acquire the mutex if free, otherwise wait until the mutex is

-- free and then get it.

-- Unlock()

-- Release the mutex. If other threads are waiting, then

-- wake up the oldest one and give it the lock.

-- Init()

-- Each mutex must be initialized.

-- IsHeldByCurrentThread()

-- Return TRUE iff the current (invoking) thread holds a lock

-- on the mutex.

---------- Mutex . Init ----------

method Init ()

waitingThreads = new List [Thread]

heldBy = null

endMethod

---------- Mutex . Lock ----------

method Lock ()

var oldIntStat: int

oldIntStat = SetInterruptsTo (DISABLED)

--Check to see if the thread we're trying to lock is already locked

if currentThread == heldBy

FatalError("Error: The lock is already being held")

endIf

if heldBy == null

heldBy = currentThread

else

waitingThreads.AddToEnd(currentThread)

currentThread.Sleep()

endIf

oldIntStat = SetInterruptsTo (oldIntStat)

endMethod

---------- Mutex . Unlock ----------

method Unlock ()

var

oldIntStat: int

nextThread: ptr to Thread = null

oldIntStat = SetInterruptsTo (DISABLED)

--Ensures we're not unlocking a thread we're not holding in heldBy

if heldBy != currentThread

FatalError("Error: Attempting to unlock a thread you don't own")

endIf

nextThread = waitingThreads.Remove ()

heldBy = nextThread

if nextThread != null

nextThread.status = READY

readyList.AddToEnd (nextThread)

endIf

oldIntStat = SetInterruptsTo (oldIntStat)

endMethod

---------- Mutex . IsHeldByCurrentThread ----------

method IsHeldByCurrentThread () returns bool

return currentThread == heldBy

endMethod

endBehavior

----------------------------- Condition ---------------------------------

behavior Condition

-- This class is used to implement monitors. Each monitor will have a

-- mutex lock and one or more condition variables. The lock ensures that

-- only one process at a time may execute code in the monitor. Within the

-- monitor code, a thread can execute Wait() and Signal() operations

-- on the condition variables to make sure certain condions are met.

--

-- The condition variables here implement "Mesa-style" semantics, which

-- means that in the time between a Signal() operation and the awakening

-- and execution of the corrsponding waiting thread, other threads may

-- have snuck in and run. The waiting thread should always re-check the

-- data to ensure that the condition which was signalled is still true.

--

-- This class provides the following methods:

-- Wait(mutex)

-- This method assumes the mutex has alreasy been locked.

-- It unlocks it, and goes to sleep waiting for a signal on

-- this condition. When the signal is received, this method

-- re-awakens, re-locks the mutex, and returns.

-- Signal(mutex)

-- If there are any threads waiting on this condition, this

-- method will wake up the oldest and schedule it to run.

-- However, since this thread holds the mutex and never unlocks

-- it, the newly awakened thread will be forced to wait before

-- it can re-acquire the mutex and resume execution.

-- Broadcast(mutex)

-- This method is like Signal() except that it wakes up all

-- threads waiting on this condition, not just the next one.

-- Init()

-- Each condition must be initialized.

---------- Condition . Init ----------

method Init ()

waitingThreads = new List [Thread]

endMethod

---------- Condition . Wait ----------

method Wait (mutex: ptr to Mutex)

var

oldIntStat: int

if ! mutex.IsHeldByCurrentThread ()

FatalError ("Attempt to wait on condition when mutex is not held")

endIf

oldIntStat = SetInterruptsTo (DISABLED)

mutex.Unlock ()

waitingThreads.AddToEnd (currentThread)

currentThread.Sleep ()

mutex.Lock ()

oldIntStat = SetInterruptsTo (oldIntStat)

endMethod

---------- Condition . Signal ----------

method Signal (mutex: ptr to Mutex)

var

oldIntStat: int

t: ptr to Thread

if ! mutex.IsHeldByCurrentThread ()

FatalError ("Attempt to signal a condition when mutex is not held")

endIf

oldIntStat = SetInterruptsTo (DISABLED)

t = waitingThreads.Remove ()

if t

t.status = READY

readyList.AddToEnd (t)

endIf

oldIntStat = SetInterruptsTo (oldIntStat)

endMethod

---------- Condition . Broadcast ----------

method Broadcast (mutex: ptr to Mutex)

var

oldIntStat: int

t: ptr to Thread

if ! mutex.IsHeldByCurrentThread ()

FatalError ("Attempt to broadcast a condition when lock is not held")

endIf

oldIntStat = SetInterruptsTo (DISABLED)

while true

t = waitingThreads.Remove ()

if t == null

break

endIf

t.status = READY

readyList.AddToEnd (t)

endWhile

oldIntStat = SetInterruptsTo (oldIntStat)

endMethod

endBehavior

endCode