COMP 530 Introduction to Operating Systems

Fall 2017  
Kevin Jeffay

Worksheet 9, October 2

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | Your Name: |  | You worked with: |  | +1/blank/-1: |  |
|  | Aaron Zhang |  | John Espenhahn |  | +1 |  |
|  |  |  | Brennan Proudfoot |  | +1 |  |
|  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |

1. On the next two pages is a copy of the pseudo code presented in lecture that implements a run-time system for a Hoare-style monitor (the pseudo code that implements the functions *EnterMonitor*, *ExitMonitor*, *Signal*, *Wait*, *etc*.).

Recall that Java synchronized classes are like monitors in that they provide mutual exclusion among class methods and Java provides synchronization primitives that are similar to *signal* and *wait*.

Modify the code below for a Hoare monitor run-time system to create a new runtime system for a Java synchronized class. That is, rewrite the pseudo code below to implement (at least) four new functions: *SyncClassEntry*, *SyncClassExit*, *wait*, and *notify*.

Note that you are *not* writing a Java program. You are writing operating system code that would be *used* by a Java implementation (*e.g.*, a Java JVM) to support synchronized classes.

|  |  |
| --- | --- |
| struct conditionVar  queue : systemQueue  numWaiting : integer := 0  end struct | var  monitorCodeMutex : binarySem := 1  monitorBusy : Boolean := FALSE  entryQueue : systemQueue  urgentQueue : systemQueue  numWaiters : integer := 0  numSignalers : integer := 0 |

Don’t need numSignalers b/c it was the count associated with the urgent queue. No condition variable param. b/c

Wake a waiter, basically trying to determine who would get in next

Is there someone ready to get in? If there is take them off the entry queue and put them on the ready queue

function wakeAWaiter() : boolean

begin

var waiter : process\_ID

if(numSignalers > 0) then

numSignalers -= 1

waiter := remove\_queue(urgentQueue)

elseif(numWaiters > 0) then

numWaiters -= 1

waiter := remove\_queue(entryQueue)

else

return(FALSE)

end if

DISABLE\_INTS

insert\_queue(readyQueue, waiter)

ENABLE\_INTS

return(TRUE)

end wakeAWaiter

procedure EnterMonitor()

begin

var next : process\_ID

monitorCodeMutex.down*b*()

if (monitorBusy) then

numWaiters += 1

insert\_queue(entryQueue, running)

DISABLE\_INTS

next := remove\_queue(readyQueue)

monitorCodeMutex.up*b*()

dispatch(next)

ENABLE\_INTS

else

monitorBusy := TRUE

monitorCodeMutex.up*b*()

end if

end EnterMonitor

procedure ExitMonitor()

begin

var waiterAwoken : boolean := FALSE

monitorCodeMutex.down*b*()

waiterAwoken := wakeAWaiter()

if (!waiterAwoken) then

monitorBusy := FALSE

end if

monitorCodeMutex.up*b*()

end ExitMonitor

procedure Wait(cv : conditionVar)

begin

var next : process\_ID

cv.numWaiting += 1

insert\_queue(cv.queue, running)

monitorCodeMutex.down*b*()

if (!wakeAWaiter()) then

monitorBusy := FALSE

end if

monitorCodeMutex.up*b*()

DISABLE\_INTS

next := remove\_queue(readyQueue)

dispatch(next)

ENABLE\_INTS

end Wait

Wait didn’t deal with urgent queue, so …

procedure Signal(cv : conditionVar)

begin

var next := process\_ID

if (cv.numWaiting > 0) then

cv.numWaiting -= 1

numSignalers += 1

insert\_queue(urgentQueue, running)

next := remove\_queue(cv.queue)

DISABLE\_INTS

dispatch(next)

ENABLE\_INTS

end if

end Signal