Ted Timmons
tedt@pdx.edu
2009-11-04
CS333 Homework 3

My solution **to** the game parlor is incomplete. If **I** remove the monitor and use a Yield(), it works. However, **I** can’t get it working properly with the monitor.

```
$ make && blitz -g os
kpl Main -unsafe
asm Main.s
lddd System.o List.o Thread.o Switch.o Synch.o Main.o Runtime.o -o os
Beginning execution...
====================  KPL PROGRAM STARTING  ====================
Initializing Thread Scheduler...
done with init, time to test.
new barber: barber 1
barber: barber 1 is done with initloop, is now ready to cut hair
new barber: barber 2
new barber: barber 3
new customer: customer 3
new customer: customer 5
 barber 1 cutting hair.
new customer: customer 4
new customer: customer 7
barber: barber 3 is done with initloop, is now ready to cut hair
 customer 3 getting a haircut.
new customer: customer 6
new customer: customer 9
new customer: customer 10
new customer: customer 12
new customer: customer 14
barber: barber 2 is done with initloop, is now ready to cut hair
 customer 5 getting a haircut.
 barber 3 cutting hair.
 barber 1 is done cutting hair.
new customer: customer 11
new customer: customer 15
new customer: customer 8
new customer: customer 13
shop is full, customer 12 is not waiting.
 barber 2 cutting hair.
 barber 3 is done cutting hair.
 barber 1 cutting hair.
 customer 4 getting a haircut.
 customer 7 getting a haircut.
 barber 2 is done cutting hair.
shop is full, customer 15 is not waiting.
 barber 1 is done cutting hair.
shop is full, customer 8 is not waiting.
shop is full, customer 13 is not waiting.
 barber 3 cutting hair.
 customer 6 getting a haircut.
 barber 2 cutting hair.
 customer 9 getting a haircut.
 barber 1 cutting hair.
 barber 3 is done cutting hair.
 customer 10 getting a haircut.
 barber 2 is done cutting hair.
 customer 11 getting a haircut.
 barber 2 cutting hair.
 barber 1 is done cutting hair.
 barber 3 cutting hair.
 barber 2 is done cutting hair.
 customer 14 getting a haircut.
 barber 3 is done cutting hair.

*****  A 'wait' instruction was executed and no more interrupts are scheduled... halting +
emulation!  *****

Done!  The next instruction to execute will be:
```

```
000EC8: 09000000       ret
Number of Disk Reads    = 0
Number of Disk Writes   = 0
Instructions Executed   = 328887
Time Spent Sleeping     = 0
    Total Elapsed Time  = 328887
```

```
$ !mak
$ make && blitz -g os
make: Nothing to be done for ‘all’.
Beginning execution...
====================  KPL PROGRAM STARTING  ====================
Initializing Thread Scheduler...
about to test game parlor
here we are, testing the game parlor
A Backgammon requests 4
------------------------------Number of dice now avail = 8
A Backgammon proceeds with 4
------------------------------Number of dice now avail = 4
B Backgammon requests 4
------------------------------Number of dice now avail = 4
B Backgammon proceeds with 4
------------------------------Number of dice now avail = 0
D Risk requests 5
------------------------------Number of dice now avail = 0
A Backgammon releases and adds back 4
------------------------------Number of dice now avail = 4
A Backgammon requests 4
------------------------------Number of dice now avail = 4
A Backgammon proceeds with 4
------------------------------Number of dice now avail = 0
C Risk requests 5
------------------------------Number of dice now avail = 0
E Monopoly requests 2
------------------------------Number of dice now avail = 0

FATAL ERROR in D Risk: "Attempt to lock a mutex by a thread already holding it" -- +
TERMINATING!

====================  KPL PROGRAM TERMINATION  ====================

****  A ’debug’ instruction was encountered  *****
Done!  The next instruction to execute will be:
000E08: C0100000       sethi   0x0000,r1        ! 0x00000E18 = 3608 (noGoMessage)

Entering machine-level debugger...
======================================================
=====                                            =====
=====         The BLITZ Machine Emulator         =====
=====                                            =====
=====  Copyright 2001-2007, Harry H. Porter III  =====
=====                                            =====
======================================================

Enter a command at the prompt.  Type ’quit’ to exit or ’help’ for
info about commands.
> quit
Number of Disk Reads    = 0
Number of Disk Writes   = 0
Instructions Executed   = 266009
Time Spent Sleeping     = 0
    Total Elapsed Time  = 266009
```

```
header Main

  uses System, Thread, Synch

  functions
    main ()
    testBarber()
    get_haircut()
    cut_hair()
    barber(waitTime: int)
    customer(waitTime: int)
    loopWait(waitTime: int)

  class GameParlor
    superclass Object
    fields
      numberDiceAvail: int
      lobbyCondition:  Condition
      gmutex:          Mutex
      diceCountMutex:  Mutex
    methods
      Init ()
      Request (numNeeded: int)
      Return (numReturned: int)
      Print (str: String, count: int)
  endClass


endHeader
```

```
code Main

  -- OS Class: Project 3
  --
  -- Ted Timmons, tedt@pdx.edu, 2009-11-04
  --

-----------------------------  Main  ---------------------------------

  function main ()
      InitializeScheduler()
      --testBarber()

print("about to test game parlor\n")
      testGameParlor()
print("done testing game parlor\n")
    endFunction


var gp: GameParlor

function testGameParlor ()
  var
    team: array[8] of Thread = new array of Thread {8 of new Thread}

  print("here we are, testing the game parlor\n")
  gp = new GameParlor
  gp.Init ()

  team[0].Init ("A Backgammon")
  team[0].Fork (play, 4)
  team[1].Init ("B Backgammon")
  team[1].Fork (play, 4)
  team[2].Init ("C Risk")
  team[2].Fork (play, 5)
  team[3].Init ("D Risk")
  team[3].Fork (play, 5)
  team[4].Init ("E Monopoly")
  team[4].Fork (play, 2)
  team[5].Init ("F Monopoly")
  team[5].Fork (play, 2)
  team[6].Init ("G Pictionary")
  team[6].Fork (play, 1)
  team[7].Init ("H Pictionary")
  team[7].Fork (play, 1)

  ThreadFinish()   -- Patiently wait for our threads to finish
endFunction

-- "playing" thread. This is a group that is playing a given game
function play (diceNeeded: int)
  var
    i: int

  for i = 1 to 5
--print("in loop: ")
--printInt(i)
--print("\n")
    -- get our dice
    gp.Request(diceNeeded)

    -- play our game
    currentThread.Yield()
    -- give them back
    gp.Return(diceNeeded)
  endFor
endFunction




var
  CHAIRS: int = 5                      -- # chairs for waiting customers
  -- semaphore: typedef int semaphore    -- use your imagination

  customers: Semaphore = new Semaphore
  barbers:   Semaphore = new Semaphore
  mutex:     Semaphore = new Semaphore
  waiting:   int = 0

  thread: array[16] of Thread = new array of Thread {16 of new Thread}

function testBarber()
    var i: int

    customers.Init(0)
    barbers.Init(0)
    mutex.Init(1)

    print("done with init, time to test.\n")
    thread[0].Init("barber 1")
    thread[1].Init("barber 2")
    thread[2].Init("barber 3")
    thread[3].Init("customer 3")
    thread[4].Init("customer 4")
    thread[5].Init("customer 5")
    thread[6].Init("customer 6")
    thread[7].Init("customer 7")
    thread[8].Init("customer 8")
    thread[9].Init("customer 9")
    thread[10].Init("customer 10")
    thread[11].Init("customer 11")
    thread[12].Init("customer 12")
    thread[13].Init("customer 13")
    thread[14].Init("customer 14")
    thread[15].Init("customer 15")

    thread[0].Fork(barber, 50)
    thread[1].Fork(barber, 500)
    thread[2].Fork(barber, 1000)

    for i = 3 to 15
      thread[i].Fork(customer, i*50)
    endFor
    --customer()
    --customer()
    --barber()

    ThreadFinish()   -- Patiently wait for our threads to finish
    print("done with testing.\n")
endFunction

function get_haircut()
  print(" ")
  print(currentThread.name)
  print(" getting a haircut.\n")
endFunction

function cut_hair()
  print(" ")
  print(currentThread.name)
  print(" cutting hair.\n")

  -- wait to cause a haircut to take a little bit of time.
  loopWait(800)
  print(" ")
  print(currentThread.name)
  print(" is done cutting hair.\n")
endFunction


function barber(waitTime: int)
    print("new barber: ")
    print(currentThread.name)
    print("\n")

    loopWait(waitTime)

    print("barber: ")
    print(currentThread.name)
    print(" is done with initloop, is now ready to cut hair\n")

    while (true)
        customers.Down()      -- go to sleep if # of customers is 0
        mutex.Down()          -- acquire access to "waiting'
        waiting = waiting - 1 -- decrement count of waiting customers
        barbers.Up()          -- one barber is now ready to cut hair
        mutex.Up()            -- release 'waiting'
        cut_hair()            -- cut hair (outside critical region)
    endWhile
endFunction



function customer(waitTime: int)
    loopWait(waitTime)

    -- it makes more sense for customers to print once they are ready,
    -- not when they are created. That helps the print statements to be
    -- in a logical order.
    print("new customer: ")
    print(currentThread.name)
    print("\n")

    mutex.Down()              -- enter critical region
    if (waiting < CHAIRS)      -- if there are no free chairs, leave
        waiting = waiting + 1 -- increment count of waiting customers
        customers.Up()        -- wake up barber if necessary
        mutex.Up()            -- release access to 'waiting'
        barbers.Down()        -- go to sleep if # of free barbers is 0
        get_haircut()         -- be seated and be served
    else
        mutex.Up()            -- shop is full, do not wait

        print("shop is full, ")
        print(currentThread.name)
        print(" is not waiting.\n")
    endIf
```

```
endFunction

-- helper function to wait by waitTime loops
function loopWait(waitTime: int)
  var i: int
  for i = 1 to waitTime
  endFor
endFunction

--  class GameParlor
--    superclass Object
--    fields
--      numberDiceAvail: int
--      lobbyCondition:  Condition
--      gmutex:          Mutex
--      diceCountMutex:  Mutex
--
--    methods
--      Init ()
--      Request (numNeeded: int)
--      Return (numReturned: int)
--      Print (str: String, count: int)
--  endClass


  behavior GameParlor

      method Init ()
        numberDiceAvail = 8

        gmutex = new Mutex
        gmutex.Init()

        lobbyCondition = new Condition
        lobbyCondition.Init()

        diceCountMutex = new Mutex
        diceCountMutex.Init()
      endMethod

      method Request (numNeeded: int)
          self.Print ("requests", numNeeded)
          diceCountMutex.Lock()

          -- make sure we have enough dice to play our game.
          -- if not, use the monitor to wait.
          while numNeeded > numberDiceAvail
            -- unlock while we are waiting so we don’t tie up the dice count
            -- (remember another request could be for a smaller number of dice)
            diceCountMutex.Unlock()

            gmutex.Lock()
            lobbyCondition.Wait(&gmutex)

            -- this isn’t right, it’s simply a placeholder until
            -- the lobbyCondition is working.
            --currentThread.Yield()

            -- relock dice count so we can check it
            diceCountMutex.Lock()
          endWhile

          numberDiceAvail = numberDiceAvail - numNeeded
          diceCountMutex.Unlock()
          self.Print ("proceeds with", numNeeded)
        endMethod

      method Return (numReturned: int)
          diceCountMutex.Lock()
          numberDiceAvail = numberDiceAvail + numReturned
          self.Print ("releases and adds back", numReturned)
          diceCountMutex.Unlock()

          -- wake up all the waiting teams so at least one can run
          -- (since they are requesting different numbers of dice, it’s
          -- incorrect to simply wake up the oldest one)
          gmutex.Lock()
          lobbyCondition.Broadcast(&gmutex)
          gmutex.Unlock()
        endMethod


      -- This method prints the current thread’s name and the arguments.
      -- It also prints the current number of dice available.
      method Print (str: String, count: int)
          var oldStatus: int

          -- Print this thread’s name.  Note that we temporarily disable
          -- interrupts so that all printing will happen together.  Without
          -- this, the other threads might print in the middle, causing a mess.
          oldStatus = SetInterruptsTo (DISABLED)


          print (currentThread.name)
          print (" ")
          print (str)
          print (" ")
          printInt (count)
          nl ()
          print ("------------------------------Number of dice now avail = ")
          printInt (numberDiceAvail)
          nl ()

          -- restore interrupt status. We’re done printing now.
          oldStatus = SetInterruptsTo (oldStatus)

        endMethod

  endBehavior

endCode
```