Justin Shuck

CS333 – Fall 2014

Proj 3 – Main.c

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

code Main

-- OS Class: Project 3

--

-- Justin Shuck

--

-- Due: 10/21/2014 2:00 PM

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

function main ()

InitializeScheduler()

testSleepingBarberPart1() -- Tests part 1 of Proj 3

--testGameParlorPart2() -- Tests part 2 of Proj 3

endFunction

const

CHAIRS = 5

CUST\_COUNT = 15

BARB\_COUNT = 1

var

customers: Semaphore = new Semaphore

barbers: Semaphore = new Semaphore

mutexLock: Mutex = new Mutex

waitCounter: int = 0

threads: array[50] of Thread = new array of Thread {50 of new Thread}

barbThreads: array[1] of Thread = new array of Thread { 1 of new Thread}

-------------------------------------------------------------------------------

------------------------- PART1: Sleeping Barber ----------------------------

-------------------------------------------------------------------------------

function testSleepingBarberPart1()

var

index: int

total: int

customers.Init(0)

barbers.Init(0)

mutexLock.Init()

total = BARB\_COUNT+CUST\_COUNT

barbThreads[0].Init("Barber")

barbThreads[0].Fork(barber,1)

--------------------------------------------------------------

-- COMMENTED CODE: Useful for testing large numbers of

-- Barbers/Customers. However I couldn't implement a

-- concat of a string "Barber" with the index. Adding a static

-- test below to demenstrate meaningful output usage.

--------------------------------------------------------------

-- for index = 0 to BARB\_COUNT

-- thread[index].Init("Barber ")

-- endFor

-- for index = BARB\_COUNT to CUST\_COUNT

-- thread[index].Init("Customer ")

-- endFor

-- for index = 0 to BARB\_COUNT

-- thread[index].Fork(barber,50)

-- endFor

-- for index = BARB\_COUNT to CUST\_COUNT

-- thread[index].Fork(customer, index \* 50)

--endFor

print(" Barber 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 \n")

threads[0].Init("Customer #1")

threads[1].Init("Customer #2")

threads[2].Init("Customer #3")

threads[3].Init("Customer #4")

threads[4].Init("Customer #5")

threads[5].Init("Customer #6")

threads[6].Init("Customer #7")

threads[7].Init("Customerf#8")

threads[8].Init("Customer #9")

threads[9].Init("Customer #10")

threads[10].Init("Customer #11")

threads[11].Init("Customer #12")

threads[12].Init("Customer #13")

threads[13].Init("Customer #14")

threads[14].Init("Customer #15")

-- Iterate over the customers

for index = 0 to CUST\_COUNT - 1

threads[index].Fork(customer, index)

endFor

ThreadFinish()

endFunction

---------------------------------------------------------------------

-- BARBER

---------------------------------------------------------------------

function barber(timeToWait: int)

while (true)

customers.Down()

mutexLock.Lock()

waitCounter = waitCounter - 1

barbers.Up()

mutexLock.Unlock()

cut\_hair(timeToWait)

endWhile

endFunction

---------------------------------------------------------------------

-- CUSTOMER

---------------------------------------------------------------------

function customer(id: int)

--wait(timeToWait) -- Wait a specific amount of time before a 'new' customer arrives

mutexLock.Lock()

E(id)

-- If there is no one waiting, wake up the barber and get haircut/take a seat

if (waitCounter < CHAIRS)

waitCounter = waitCounter + 1

S(id)

customers.Up()

mutexLock.Unlock()

barbers.Down()

get\_haircut(id)

L(id)

-- The shop is full (NO seats)

else

L(id)

mutexLock.Unlock()

endIf

endFunction

---------------------------------------------------------------------

-- BUSY LOOP: Dummy function that just waits x-time

---------------------------------------------------------------------

function wait(timeToWait: int)

var index: int

for index = 1 to timeToWait

currentThread.Yield()

endFor

endFunction

function get\_haircut(custNum: int)

mutexLock.Lock()

B(custNum)

wait(50)

F(custNum)

mutexLock.Unlock()

endFunction

function cut\_hair(custNum: int)

mutexLock.Lock()

Start()

wait(75)

End()

mutexLock.Unlock()

endFunction

---------------------------------------------------------------------

-- Print Helper Function

---------------------------------------------------------------------

function Start()

printChairs()

print(" start \n")

endFunction

function End()

printChairs()

print(" end \n")

endFunction

function printChairs()

var

index: int

for index = 1 to waitCounter

print("X")

endFor

for index = 1 to CHAIRS - waitCounter

print("-")

endFor

endFunction

--------------------

-- Print extra spaces

--------------------

function printSpace(space: int)

var

index: int

totSpaces: int

print(" ")

totSpaces = space \* 3

for index = 1 to totSpaces

print(" ")

endFor

endFunction

--------------------

-- E: Enter

--------------------

function E(custNum: int)

printChairs()

printSpace(custNum)

print("E \n")

endFunction

--------------------

-- S: Sit in waiting chair

--------------------

function S(custNum: int)

printChairs()

printSpace(custNum)

print("S \n")

endFunction

--------------------

-- B: Begin Haircut

--------------------

function B(custNum: int)

printChairs()

printSpace(custNum)

print("B \n")

endFunction

--------------------

-- F: Finish haircut

--------------------

function F(custNum: int)

printChairs()

printSpace(custNum)

print("F \n")

endFunction

--------------------

-- L: Leave

--------------------

function L(custNum: int)

printChairs()

printSpace(custNum)

print("L \n")

endFunction

-------------------------------------------------------------------------------

------------------------- PART2: Game Parlor --------------------------------

-------------------------------------------------------------------------------

const

GROUPS = 8 -- Total available groups

DICE = 5 -- Total available dice

GAMES\_PLAYED = 5 -- Total games played

WAIT\_COUNTER = 50 -- Mock time for waiting

var

gameParlor: GameParlor

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

function testGameParlorPart2()

gameParlor = new GameParlor

gameParlor.Init()

print("-- PART 1: BEGIN TESTING -- \n")

thread[0].Init("A - Backgammon")

thread[0].Fork(mockGame, 4)

thread[1].Init("B - Backgammon")

thread[1].Fork(mockGame, 4)

thread[2].Init("C - Risk")

thread[2].Fork(mockGame, 5)

thread[3].Init("D - Risk")

thread[3].Fork(mockGame, 5)

thread[4].Init("E - Monopoly")

thread[4].Fork(mockGame, 2)

thread[5].Init("F - Monopoly")

thread[5].Fork(mockGame, 2)

thread[6].Init("G - Pictionary")

thread[6].Fork(mockGame, 1)

thread[7].Init("H - Pictionary")

thread[7].Fork(mockGame, 1)

ThreadFinish()

print("-- PART 2: END TESTING -- \n")

endFunction

---------------------------------------

-- Iterates over the total GAMES\_PLAYED

-- and uses a method similar to Part 1's

-- 'wait' method where the currentThread

-- yields until WAIT\_COUNTER is complete

----------------------------------------

function mockGame(dice: int)

var

index1: int

index2: int

for index1 = 1 to GAMES\_PLAYED

gameParlor.getDice(dice)

for index2 = 1 to WAIT\_COUNTER

currentThread.Yield()

endFor

gameParlor.releaseDice(dice)

endFor

endFunction

behavior GameParlor

-----------------------------------

-- Init method, Initializes the variables

-- that we're going to use by either

-- calling an Init or by setting its

-- value

-----------------------------------

method Init()

numDiceLeft = DICE -- Set Dice

numWaitingGroups = 0 -- Set the counter for the groups waiting

monitoringLock = new Mutex

monitoringLock.Init()

firstInLine = new Condition

firstInLine.Init()

restOfLine = new Condition

restOfLine.Init()

endMethod

-----------------------------------

-- Print method: Generic use, passes

-- in a string and the number of dice

-- remaining for the particular action

-----------------------------------

method print(printString: String, num: int)

print("")

print(currentThread.name)

print(" ")

print(printString)

print(" ")

printInt(num)

print("\n-------------------------------- Number of dice now available = ")

printInt(numDiceLeft)

print("\n\n")

endMethod

-----------------------------------

-- Get Dice method

-----------------------------------

method getDice(diceNeeded: int)

monitoringLock.Lock()

self.print("requests", diceNeeded)

numWaitingGroups = numWaitingGroups + 1

-- if there are more than one person in line,

-- then have the rest of the line wait

if (numWaitingGroups > 1)

restOfLine.Wait(&monitoringLock)

endIf

-- Wait until the appropriate number of dice

-- are available

while (numDiceLeft < diceNeeded)

firstInLine.Wait(&monitoringLock)

endWhile

-- At this point they can get dice. We need

-- to decrement the dice counter and the number

-- of groups waiting.

numDiceLeft = numDiceLeft - diceNeeded

numWaitingGroups = numWaitingGroups - 1

restOfLine.Signal(&monitoringLock)

self.print("proceeds with", diceNeeded)

monitoringLock.Unlock()

endMethod

-----------------------------------

-- Release Dice method

-----------------------------------

method releaseDice(diceReturned: int)

monitoringLock.Lock()

numDiceLeft = numDiceLeft + diceReturned

self.print("releases and adds back", diceReturned)

firstInLine.Signal(&monitoringLock)

monitoringLock.Unlock()

endMethod

endBehavior

endCode