Qian Wang

COP 4600

1089

10/13/2015

HW6

On my honor, I have neither given nor received unauthorized aid in doing this assignment.

Qian Wang

Summary:

The easiest parts?

Given the fact that this homework is already provided a skeleton code, it is easier than expected. To be honest, if we were asked to start this from scratch, I might be still working on this. But now that I have this working, I am going to make my own code.

The most difficult parts?

Understanding Dave's code is the hardest part. But his code is very clean and organized. No functions that share similar functionalities. It turns out to be very easy to follow.

The assignment's educational objectives?

Comparing to Producer and Consumer problem, this homework is focus on multi thread in a bigger scale. I haven't had a program runs this much time after data structure with Dave.

How well did I achieved them?

I believe I did the homework's requirement. Now I will spend my own time on my own code.

Qian Wang

COP 4600

1089

10/13/2015

HW5

1. Does the program compile without errors?

   No. This code has no error.

2. Does the program compile without warnings?

   No. This code has no warnings.

3. Does the program run without crashing?

   No. This code run without crashing.

4. Describe how you tested the program.

   It worked out pretty fine actually. The only few problems that I had are very easy to pick out as well. So I have to say, Dave's code has a neat print console system which make my error stands out.

5. Describe the ways in which the program does *not* meet assignment's specifications.

   If I have to say something that bothers me is that how I have nothing in the catch block of my dropdown method.

6. Does the program run correctly?

   Yes

Pseudocode:

```
        Pickup:

                If( ableToPickupRightStick() ){

                        // I am trying to initialize everyone with their right hand on the chopstick;

                }

                Eles if( hasRightHand && pickUpLeftHand){

                        // now they are trying to pick up their left chopstick

                }

                Else{

                        // See if there is anyone grab the left stick before you,

                        // If did, don't be mean and return your right stick for other to take

                }


                If( bothHandAreReady ){

                        // Start eating my man!

                }

        PutDown:

                Put down chopsticks

                // Nothing fancy here, just to put down chop sticks right then left.


ChopStick:

        Pickup:

                If(  there is anyone holding you ){

                        Assign your holder

                        Signal chopstick pick up

                }

                Else{

                        Signal chopstick not pick up

                }

        Put Down:
```

```
        If( someone else trying to put down the chopstick ){
                Signal mistake
        }
        Else{
                Clear chopsitck's holder
                Signal philosopher this chopstick is ready for grab
        }
```

Avoid deadlock and lost wake-up:

There are only 2 threads waiting for one chopstick at most. So using notify() actually does a better performance in average.

Last 100 output:

```
Socrates wants left chopstick (7)
Frege finished using left chopstick (8)
Hume has both left chopstick (9) and right chopstick (0)
Hume is begining to eat for 3134 milliseconds
Frege is begining to think for 4291 milliseconds
Seneca is done thinking
Seneca wants right chopstick (1)
Aristotle is done eating
Aristotle finished using right chopstick (2)
Epicurius wants right chopstick (3)
Epicurius has right chopstick (3)
Epicurius wants left chopstick (2)
Epicurius has both left chopstick (2) and right chopstick (3)
Epicurius is begining to eat for 4583 milliseconds
Voltaire was unable to get the left chopstick (3)
Voltaire politely returned right chopstick (4)
Seneca has right chopstick (1)
Seneca wants left chopstick (0)
Seneca was unable to get the left chopstick (0)
Seneca politely returned right chopstick (1)
Aristotle finished using left chopstick (1)
Aristotle is begining to think for 2519 milliseconds
Hume is done eating
Hume finished using right chopstick (0)
Hume finished using left chopstick (9)
Seneca wants right chopstick (1)
Seneca has right chopstick (1)
Seneca wants left chopstick (0)
Seneca has both left chopstick (0) and right chopstick (1)
Hume is begining to think for 3218 milliseconds
Seneca is begining to eat for 2072 milliseconds
Aristotle is done thinking
Aristotle wants right chopstick (2)
Frege is done thinking
Frege wants right chopstick (9)
Frege has right chopstick (9)
Frege wants left chopstick (8)
Seneca is done eating
Seneca finished using right chopstick (1)
Seneca finished using left chopstick (0)
Seneca is begining to think for 1667 milliseconds
Epicurius is done eating
Epicurius finished using right chopstick (3)
Epicurius finished using left chopstick (2)
Voltaire wants right chopstick (4)
Voltaire has right chopstick (4)
Voltaire wants left chopstick (3)

Voltaire has both left chopstick (3) and right chopstick (4)

Aristotle has right chopstick (2)

Kant was unable to get the left chopstick (4)

Kant politely returned right chopstick (5)

Epicurius is begining to think for 4554 milliseconds

Aristotle wants left chopstick (1)

Aristotle has both left chopstick (1) and right chopstick (2)

Voltaire is begining to eat for 4766 milliseconds

Aristotle is begining to eat for 1670 milliseconds

Hume is done thinking

Hume wants right chopstick (0)

Hume has right chopstick (0)

Hume wants left chopstick (9)

Seneca is done thinking

Seneca wants right chopstick (1)

Aristotle is done eating

Aristotle finished using right chopstick (2)

Aristotle finished using left chopstick (1)

Seneca has right chopstick (1)

Seneca wants left chopstick (0)

Aristotle is begining to think for 4752 milliseconds

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

Voltaire ate 7 times (12400 ms) and pondered 7 times (22601ms)

Machiavelli has both left chopstick (5) and right chopstick (6)

Machiavelli is begining to eat for 3287 milliseconds

Nietzsche was unable to get the left chopstick (6)

Nietzsche politely returned right chopstick (7)

Seneca ate 7 times (16365 ms) and pondered 8 times (17631ms)
```

```
Kant ate 5 times (9311 ms) and pondered 6 times (24772ms)
Epicurius ate 5 times (16973 ms) and pondered 6 times (18364ms)
Aristotle ate 7 times (17430 ms) and pondered 8 times (19982ms)
Socrates has both left chopstick (7) and right chopstick (8)
Aristotle ate 7 times (17430 ms) and pondered 8 times (19982ms)
Nietzsche ate 6 times (16390 ms) and pondered 7 times (17458ms)
Machiavelli ate 7 times (17070 ms) and pondered 7 times (19935ms)
Epicurius ate 5 times (16973 ms) and pondered 6 times (18364ms)
Frege was unable to get the left chopstick (8)
Frege politely returned right chopstick (9)
Socrates is begining to eat for 4006 milliseconds
Frege ate 5 times (16525 ms) and pondered 6 times (22826ms)
Voltaire ate 7 times (12400 ms) and pondered 7 times (22601ms)
Socrates ate 6 times (15974 ms) and pondered 6 times (16178ms)
Hume has both left chopstick (9) and right chopstick (0)
Kant ate 5 times (9311 ms) and pondered 6 times (24772ms)
Hume is begining to eat for 3782 milliseconds
Seneca was unable to get the left chopstick (0)
Seneca politely returned right chopstick (1)
Hume ate 6 times (13834 ms) and pondered 6 times (21218ms)
Machiavelli ate 7 times (17070 ms) and pondered 7 times (19935ms)
Nietzsche ate 6 times (16390 ms) and pondered 7 times (17458ms)
Socrates ate 6 times (15974 ms) and pondered 6 times (16178ms)
Frege ate 5 times (16525 ms) and pondered 6 times (22826ms)
Seneca ate 7 times (16365 ms) and pondered 8 times (17631ms)
Hume ate 6 times (13834 ms) and pondered 6 times (21218ms)
```

Analysis:

"Seneca"	get 0, 1
"Aristotle"	get 1, 2

"Epicurius"	get 2, 3
"Voltaire"	get 3, 4
"Kant"	get 4, 5
"Machiavelli"	get 5, 6
"Nietzsche"	get 6, 7
"Socrates"	get 7, 8
"Frege"	get 8, 9
"Hume"	get 9, 0

As I believe, there two thing to check:

1. If your philosopher is using the correct chopsticks. As the above comparison it is in my case.
2. If you philosopher is actually eating, check the number of times they eat. Everyone ate around 6times which is reasonable.

Code:

```
// DiningPhiliphers.java (skeleton)
//
// - a classic synchronization problem
//
// Skeleton code derived from Dave Small's DiningPhiliphers.java v4.0

import java.util.Random;

//================================================ class DiningPhilosophers

class DiningPhilosophers
{
  public static void main( String[] arg )
  {
    new DiningPhilosophers( 10, 60000 );
  }

  private String[] name = { "Seneca", "Aristotle", "Epicurius", "Voltaire",
                            "Kant", "Machiavelli", "Nietzsche", "Socrates",
                            "Frege", "Hume" };

  private Philosopher[] thinker;
  private Chopstick[]   chopstick;
```

```
public DiningPhilosophers( int numPhilosophers, int duration )
{
  initialize( numPhilosophers );  // construct the philosophers & chopsticks
  startSimulation();
  sleep( duration );              // let simulation run for desired time
  shutdownPhilosophers();         // *gracefully* shut down the philosophers
  printResults();
}

private void initialize( int n )  // handles 2 to 10 philosophers
{
  if ( n > 10 )
    n = 10;
  else if ( n < 2 )
    n = 2;

  thinker = new Philosopher[n];
  chopstick = new Chopstick[n];

  for ( int i = 0 ; i < n ; i++ )
    chopstick[i] = new Chopstick(i);

  for ( int i = 0 ; i < n ; i++ )
    thinker[i] = new Philosopher( name[i], chopstick[i], chopstick[(i+1)%n] );
}

private void startSimulation()
{
  int n = thinker.length; // the number of philosophers

  System.out.print( "Our " + n + " philosophers (" );
  for ( int i = 0 ; i < (n-1) ; i++ )
    System.out.print( name[i] + ", " );
  System.out.println( "and " + name[n-1] +
                                      ") have gather to think and dine" );

  System.out.println( "-----------------------------------------------");

  for ( int i = 0 ; i < n ; i++ )
    thinker[i].start();
}

private void sleep( int duration )
{
    try
    {
      Thread.currentThread().sleep( duration );
    }
    catch ( InterruptedException e )
    {
      Thread.currentThread().interrupt();
    }
  }

  private void shutdownPhilosophers()
  {
    /********* YOUR CODE GOES HERE **********/
    for(int i = 0; i != thinker.length; ++i){
       thinker[i].interrupt();
    }
  }

  private void printResults()
  {
    System.out.println( "----------------------------------------------");

    int n = thinker.length; // the number of philosophers

    for ( int i = 0 ; i < n ; i++ )
      System.out.println( thinker[i] );

    System.out.flush();
  }
}

//================================================== class Philosopher

class Philosopher extends Thread
{
  static private Random random = new Random();

  private String name;
  private Chopstick leftStick;
  private Chopstick rightStick;

  private int eatingTime   = 0;
  private int thinkingTime = 0;
  private int countEat     = 0;
private int countThink   = 0;

public Philosopher( String name, Chopstick leftStick, Chopstick rightStick )
{
  this.name = name;
  this.leftStick = leftStick;
  this.rightStick = rightStick;
}

public String toString()
{
  return name + " ate " + countEat + " times (" +
    eatingTime + " ms) and pondered " + countThink + " times (" +
    thinkingTime + "ms)";
}

public void run()
{
  /********* YOUR CODE GOES HERE **********/
  try{
      while(true){
        countThink++;
        thinkingTime += doAction( "think" );
        pickupChopsticks();
        countEat++;
        eatingTime += doAction( "eat" );
        putdownChopsticks();
    }
  }
  catch(InterruptedException e){
      System.out.println(toString());
  }


  /********* YOUR CODE GOES HERE **********/
}

private int doAction( String act ) throws InterruptedException
{
  int time = random.nextInt( 4000 ) + 1000 ;
  System.out.println( name + " is begining to " + act + " for " + time +
                                       " milliseconds" );
  sleep( time );
```

```
    System.out.println( name + " is done " + act + "ing" );

    return time;
  }

    private void pickupChopsticks() throws InterruptedException
  {

      outter:
      while(true){
        synchronized(rightStick){
            System.out.println( name + " wants right " + rightStick );
             while(!rightStick.pickUp()){
                rightStick.wait();
            }
              System.out.println( name + " has right " + rightStick );
              System.out.println( name + " wants left " + leftStick );
              if(!leftStick.pickUp()){
                System.out.println( name + " was unable to get the left " + leftStick );
                System.out.println( name + " politely returned right " + rightStick );
                rightStick.putDown();
                synchronized(leftStick){
                  leftStick.wait();
                }
              }
              else{
                System.out.println( name + " has both left " + leftStick +
                  " and right " + rightStick );
                break outter;
              }
          }
        }
}

  private void putdownChopsticks()
  {
    try{
      rightStick.putDown();
      System.out.println( name + " finished using right " + rightStick );
      leftStick.putDown();
      System.out.println( name + " finished using left " + leftStick );
    }
    catch(RuntimeException e){
    }
  }
}
//============================================== class Chopstick

class Chopstick
{
  private final int id;
  private Philosopher heldBy = null;

  public Chopstick( int id )
  {
    this.id = id;
  }

  public String toString()
  {
    return "chopstick (" + id + ")";
  }

  synchronized public boolean pickUp()
  {
    if(this.heldBy == null){
      // No one is holding the chopstick
      heldBy = (Philosopher) Thread.currentThread();
      return true;
    }
    else{
      return false;
    }
  }

  synchronized public void putDown()
  {
    Philosopher p = (Philosopher) Thread.currentThread();

    if( p != heldBy){
       throw new RuntimeException( "Exception: " + p + " attempted to put " +
          "down a chopstick he wasn't holding." );
    }
    else{
      //letting go
      heldBy = null;
      this.notify();
    }

  }
}
```