Assingment-2

Thanjida Akhter

201691489

Solution: 1

Thread to synchronize with its pairing thread is given in Table-1.

Thread *i*	Round(r), Distance = (2^{r-1})			
	r =1	***r =2***	***r =3***	***r =4***
i = 0	1	2	4	8
i = 1	1	2	4	8
i = 2	1	2	4	8
i = 3	1	2	4	8
i = 4	1	2	4	8
i = 5	1	2	4	8
i = 6	1	2	4	8
i = 7	1	2	4	8

Table-1: Synchronization Distance for $0 < r < 5$

Interconnection n must be power of 2 since each thread at round r is synchronized with thread ***2^{r-1}*** away **Table-2** gave the illustration of synchronization distance domain of threads n=6. For round 4 there is redundant synchronization domain which is unnecessary r=2 and r=4 same ways r=5 and r=3.

Table – 2: 6 threads limiting in round 3

Thread i	***r =1***	***r =2***	***r =3***	***r =4***	***r =5***
	i + 1 (mod n)	***i + 2 (mod n)***	***i + 4 (mod n)***	***i + 8 (mod n)***	***i + 16 (mod n)***
i = 0	1	2	4	2	4
i = 1	2	3	5	3	5
i = 2	3	4	0	4	0
i = 3	4	5	1	5	1
i = 4	5	0	2	0	2
i = 5	0	1	3	1	3

From Table-2 conjecture that number of rounder to go for given barrier synchronization are *$\log_2(n)$.*

n=6 and 8

number of rounds = $\log_2(6) = 2.58 \sim 3$

number of rounds = $\log_2(8) = 3$

If we have 8 threads rounds above *$\log_2(8)$*

Table – 2: Synchronization Domain of 8 threads

Thread i	***r =1***	***r =2***	***r =3***
	i + 1 (mod n)	***i + 2 (mod n)***	***i + 4 (mod n)***
i = 0	1	2	4
i = 1	2	3	5

i = 2	3	4	6
i = 3	4	5	7
i = 4	5	6	0
i = 5	6	7	1
i = 6	7	8	2
i = 7	0	1	3

Dissemination barrier implementation is regardless of n which power of 2 or not. This is an only difference for n if power of two reverse directed synchronization points will be reduced but there is no impact of strength of synchronization. Whereas if n is not exact power of 2 then these reverse directed synchronization are minimal.

Solution: 2

```
public class DinPh
{
public static void main(String[] args)
{
            int a=10;
            Log.msg(String.valueOf(a));
            chop_st[] chop_s = new Chop_st[5];
            for(int i=0; i< chop_s.length; i++)
            {
                  chop_s[i] = new Chop_st("C: "+i);
            }
            Ph[] phs = new Ph[5];
            Phs[0] = new Ph("P: 0 - ", chop_s[0], chop_s[1]);
            Phs[1] = new Ph("P: 1 - ", chop_s[1], chop_s[2]);
            Phs[2] = new Ph("P: 2 - ", chop_s[2], chop_s[3]);
            Phs[3] = new Ph("P: 3 - ", chop_s[3], chop_s[4]);
            Phs[4] = new Ph("P: 4 - ", chop_s[0], chop_s[4]);
            for(int i=0;i<Phs.length;i++)
            {
                  Log.msg("Thred "+ i);
                  Thread t= new Thread( Phs[i]);
                  t.start();
            }
      }
}
class Ph extends Thread
{
      private final chop_st left_Chop_s;
      private final chop_st right_Chop_s;
      private final String nm;
      private int st;
      public Ph ( String name, Chop_stick _l, Chop_stick _r)
      {
            this.st = 1;
            this.nm = name;
            left_Chop_s = lf;
            right_Chop_s = rh;
      }
      public void eat()
      {
            if(! left_Chop_s.used){
                  if(!right_Chop_s.used){
                        left_Chop_s.take();
                        left_Chop_s.take();
                        Log.msg(nm + " : Eat");
                        Log.Delay(1000);
                        left_Chop_s.release();
                        right_Chop_s.release();
                  }
            }
            think();
      }
      public void think()
      {
                  this.st = 1;
                  Log.msg(nm + " : Think");
                  Log.Delay(1000);
      }
        @Override
      public void run()
        {
            for(int i=0; i<=10; i++)
            {
                  eat();
            }
      }
}
```

Solution: 3

```
import java.util.concurrent.BrokenBarrierException;
import java.util.concurrent.Barrier_cy;
import java.util.concurrent.TimeUnit;
package advancedConcurrentPackage;
import java.io.IOException;
import java.net.URL;
import java.util.Collections;
import java.util.List;
import java.util.concurrent.locks.lock_r;

public class Thread
{
      public value[][] flags;
      int to_th, round,total;
      static int  counter=0;
      static lock_r lock_c = new lock_r(true);
      Barrier_cy barrier;

      public Thread(int th_num)
      {
            to_th=th_num;
            total=power(th_num);
            round=(int)(Math.log(total)/Math.log(2));
             flags=new value[th_num][round];
      }

      public void th_create()
      {
            barrier=new Barrier_cy(to_th);
            for(int i=0;i<to_th;i++)
            {
                  new Thread()
                  {
                        public void run()
                        {
                              for(int j=0;j<round;j++)
                              {
                                    th_notify(counter,  j);
                                    round_sp(counter,j);
                              }
                              count_inc();
                              }
                  }.start();
            }
      }
      public void th_wait(int i) throws InterruptedException
      {

      }

      public void th_notify(int i, int r)
      {
            flags[i][r]=(i+(int)Math.pow(2,r))%to_th;

            System.out.println(Thread.currentThread().getName()+" finished 
round " + r+" waiting for others: ");
      }

      public void round_sp(int i,int r)
      {
            while(flags[i][r]!=(i-(int)Math.pow(2,r))%to_th)
            {
                  try
                  {
                        Thread.sleep(10000);
                  }
```