

Threads

Hsuan-Tien Lin

Department of CSIE, NTU

OOP Class, June 14-15, 2010

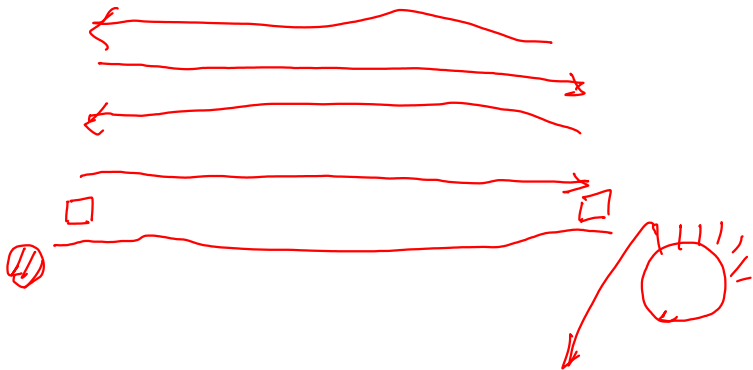
A Story of the Black and White Goats: Deadlock



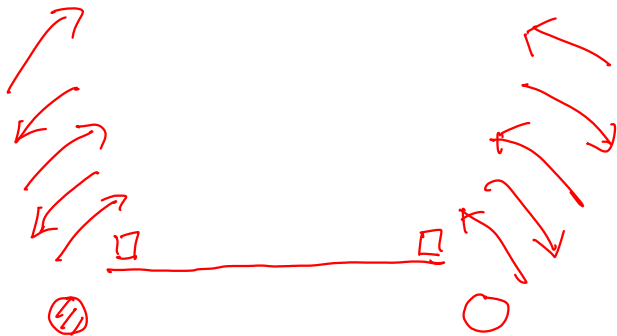
```
synchronized(b){  
  synchronized(w){  
    gogogo();  
  }  
}
```

```
synchronized(w){  
  synchronized(b){  
    gogogo();  
  }  
}
```

A Story of the Black and White Goats: Starvation



A Story of the Black and White Goats: Livelock



Ways to be Polite

```
1 synchronized void make_payment(int amount){
2     while(no_money()){
3         this.complain(1000);
4     }
5 }
```

```
1 synchronized void make_payment(int amount){
2     while(no_money()){
3         Thread.sleep(1000);
4     }
5 }
6 // ...
```

```
1 synchronized void make_payment(int amount){
2     while(no_money()){
3         this.wait(1000);
4     }
5 }
6 // ...
```

Wait until Notified

Object.

```
1   synchronized void make_payment(int amount){
2       while(no_money()){
3           this.wait();
4       }
5   }
6
7   synchronized void get_money(int amount){
8       money += amount;
9       notifyAll();
10  }
```

notify()

- difference between `wait` and `sleep`: the former **releases** the lock temporarily

Producer/Consumer Threads I

```
1 //from Java Tutorial
2 public class Drop {
3     //message sent from producer to consumer
4     private String message;
5     //true if consumer should wait for producer to send msg;
6     //false if producer should wait for consumer to retrieve msg
7     private boolean empty = true;
8
9     public synchronized String take() {
10        //wait until message is available
11        while (empty) {
12            try {
13                wait();
14            } catch (InterruptedException e) {}
15        }
16        //toggle status
17        empty = true;
18        //notify producer that status has changed
19        notifyAll();
20        return message;
21    }
```

Producer/Consumer Threads II

```
22
23 public synchronized void put(String message) {
24     //wait until message has been retrieved
25     while (!empty) {
26         try {
27             wait();
28         } catch (InterruptedException e) {}
29     }
30     //toggle status
31     empty = false;
32     //store message
33     this.message = message;
34     //notify consumer that status has changed
35     notifyAll();
36 }
37 }
```


Producer/Consumer Threads III

```
1 //from Java Tutorial
2
3 import java.util.Random;
4
5 public class Producer implements Runnable {
6     private Drop drop;
7
8     public Producer(Drop drop) {
9         this.drop = drop;
10    }
11
12    public void run() {
13        String importantInfo [] = {
14            "I_love_OOP",
15            "I_love_Java",
16            "I_love_PTT",
17            "I_love_POOCasino",
18        };
19
20
21
```

Producer/Consumer Threads IV

```
22     Random random = new Random();
23
24     for (int i = 0; i < importantInfo.length * 3; i++) {
25         synchronized(drop){
26             drop.put(importantInfo[i % importantInfo.length]);
27             System.out.printf("MESSAGE_SENT:_%s%n",
28                 importantInfo[i % importantInfo.length]);
29         }
30         try {
31             Thread.sleep(random.nextInt(5000));
32         } catch (InterruptedException e) {}
33     }
34     drop.put("DONE");
35     System.out.println("Producer_Exits");
36 }
37 }
```

Producer/Consumer Threads V

```
1 //from Java Tutorial
2 import java.util.Random;
3 public class Consumer implements Runnable {
4     private Drop drop;
5
6     public Consumer(Drop drop) { this.drop = drop; }
7
8     public void run() {
9         Random random = new Random();
10        for (String message = drop.take();
11            !message.equals("DONE");
12            message = drop.take()) {
13            System.out.format("MESSAGE_RECEIVED:_%s%n", message);
14            try {
15                Thread.sleep(random.nextInt(5000));
16            } catch (InterruptedException e) {}
17        }
18        System.out.println("Consumer_Exits");
19    }
20 }
```

Producer/Consumer Threads VI

```
1 //from Java Tutorial
2 public class ProducerConsumerExample {
3     public static void main(String [] args) {
4         Drop drop = new Drop();
5         (new Thread(new Producer(drop))).start();
6         (new Thread(new Consumer(drop))).start();
7     }
8 }
```