3C03 Concurrency: Condition Synchronisation

Wolfgang Emmerich

Goals

- *Introduce concepts of*
  - *Condition synchronisation*
  - *Fairness*
  - *Starvation*
- *Modelling:*
  - *Relationship between guarded actions and condition synchronisation?*
- *Implementation:*
  - *Condition Monitors in Java,*
  - *Semaphores as Java Monitors*

Thread Waiting Queues in Java

- `public final void notify()`

  Wakes up a single thread that is waiting on this object's queue
- `public final void notifyAll()`

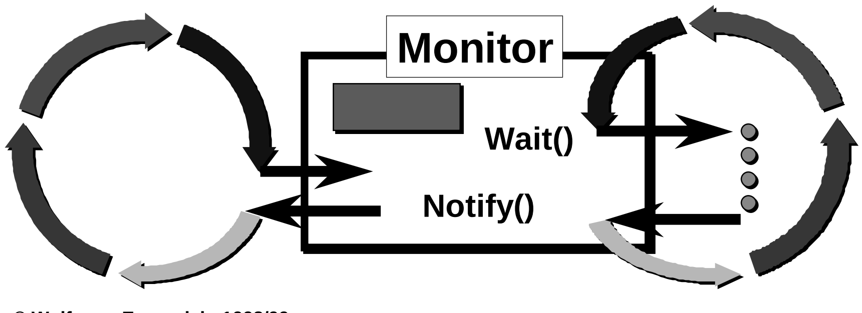
  Wakes up all threads that are waiting on this object's queue

-
  ```
  public final void wait()
           throws InterruptedException
  ```

  Waits to be notified by another thread when notified must reacquire monitor

Condition synchronisation in Java

- *Thread enters monitor when it acquires mutual exclusion lock of monitor*
- *Thread exits monitor when releasing lock*
- *Wait causes thread to exit monitor*



Semaphore as a Java Monitor

```
class Semaphore {
  private int value_;
  Semaphore (int initial) {
    value_=initial;
  }
  public synchronised up() {
    ++value_;
    notify();
  }
  public synchronised down() {
    while (value_==0) wait();
    --value;
  }
}
```

Condition Synchronisation in Java

- *FSP Model: when cond act -> NEWSTATE*
- *Java:*

```
public synchronized void act()
throws InterruptedException
{
  while (! cond) wait();
  act
  notifyAll();
}
```

- *Loop re-tests cond to make sure that it is valid when it re-enters the monitor*

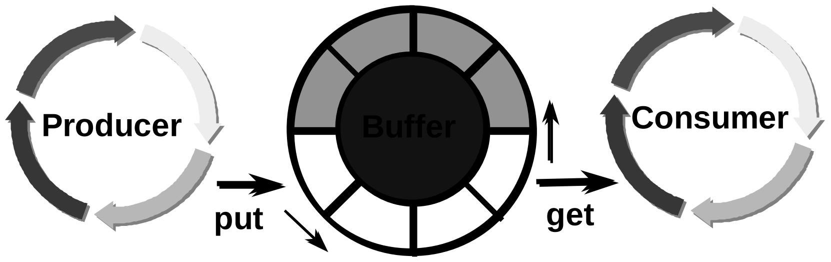
CarParkControl revisited

```
class CarParkControl {
  private int spaces;
  private int N;
  synchronized public void arrive() {
  while (spaces<=0) {
    try {
     wait();
    } catch(InterruptedException e){}
  }
  --spaces;
  notify();
 }
```

FSP and Condition Synchronisation

- ***For each guarded action in the FSP model of a monitor***
  - ***Implement action as a synchronised method***
  - ***That invokes*** `wait()` ***in a while loop before it begins***
  - ***While condition is negation of guard condition***
- ***Every change in the monitor are signalled to waiting threads using*** `notify()` ***or*** `notifyAll()`

Example: Producer/Consumer



Demo

Producer Consumer in FSP

```
PRODUCER = (put -> PRODUCER).
CONSUMER = (get -> CONSUMER).
BUFFER(SIZE=5) = BUFFER[0],
BUFFER[count:0..SIZE] = (
       when (count<SIZE) put->BUFFER[count+1]
      |when (count>0) get -> BUFFER[count-1]).
||PC=(PRODUCER||BUFFER||CONSUMER).
```

LTSA

Bounded Buffer - Outline

```
class Buffer {
  private protected Object[] buf;
  private protected int in = 0;//index put
  private protected int out = 0;//index get
  private protected int count = 0; //no items
  private protected int size;
  Buffer(int size) {
      this.size = size;
      buf = new Object[size];
  }
  synchronized public void put(Object o) {…}
  synchronized public Object get() {…}
}
```

Bounded Buffer - put

```
synchronized public void put(Object o) {
  while (count>=size) {
     try {
       wait();
     } catch(InterruptedException e){}
  }
  buf[in] = o;
  ++count;
  in=(in+1) % size;
  notifyAll();
}
```

Bounded Buffer - get

```
synchronized public Object get() {
 while (count==0) {
    try {
      wait();
    } catch (InterruptedException e){}
 }
 Object o =buf[out];
 buf[out]=null; // for display purposes
 --count;
 out=(out+1) % size;
 notifyAll(); // [count < size]
 return (o);
}
```

Monitor Invariants

- ***Monitor invariant is assertion concerning attributes encapsulated by monitor***
- ***Assertion must hold when no thread is in monitor***
- ***Examples:***
  - ***CarParkControl: 0<=spaces<=N***
  - ***Semaphore: 0<=value***
  - ***BoundedBuffer: (0<=count && 0<=in<=size && 0<=out<=size && in=(out+count) % size)***
- ***Used to reason about correctness monitors***

Summary

- *Condition synchronization*
- *In Java using* `wait()`, `notify()` *and* `notifyAll`*()*
- *Used to implement Semaphores in Java*
- *Relation between FSP model and implementation in Java monitor*
- *Monitor invariants*