Threading: Java vs. C#

Facility	Java	C#
Thread embedding	`Runnable` object	`void` → `void` procedure
Critical section	`synchronized` P`( ... )`	
	`synchronized (`o`) { ... }`	`lock (`o`) { ... }`
		`Monitor.Enter(`o`)`
		`Monitor.Exit(`o`)`
		`Monitor.TryEnter(`o`,`t`)`
Condition queue	o`.wait()`	`Monitor.Wait(`o`)`
	o`.notify()`	`Monitor.Pulse(`o`)`
	o`.notifyAll()`	`Monitor.PulseAll(`o`)`

Threading Platforms

Pthreads	Java	C#/.NET	Win32
semaphores/events			
`sem_wait(`s`)`	s`.aquire(`n`)`	s`.WaitOne()`	`WaitForSingle(`s`)`
`sem_post(`s`)`	s`.release(`n`)`	s`.Release()`	`ReleaseSem(`s`,`n`)`
	e`.unpark()`	e`.Set()`	`SetEvent(`e`)`
		e`.Reset()`	`ResetEvent(`e`)`
			`PulseEvent(`e`)`
	e`.park()`	e`.WaitOne()`	`WaitForSingle(`e`)`
		`WaitAll(`$\overline{e}$`)`	`WaitForMult(`$\overline{e}$`,`**all**`)`
		`WaitAny(`$\overline{e}$`)`	`WaitForMult(`$\overline{e}$`,`**any**`)`
critical regions			
	R/W-locks	R/W-locks	
		m`.ReleaseMutex()`	`ReleaseMutex(`m`)`
		m`.WaitOne()`	`WaitForSingle(`m`)`
	`synchronized` P`( )`		
	`synchronized(`o`) {...}`	`lock (`o`) {...}`	
`mutex_lock(`m`)`	m`.lock()`	`Monitor.Enter(`o`)`	
`mutex_unlock(`m`)`	m`.unlock()`	`Monitor.Exit(`o`)`	
	m`.trylock(`t`)`	`Monitor.TryEnter(`o`,`t`)`	
condition queues			
`cond_wait(`c`,`m`)`	c`.await()` o`.wait()`	`Monitor.Wait(`o`)`	
`cond_signal(`c`)`	c`.signal()` o`.notify()`	`Monitor.Pulse(`o`)`	
`cond_broadcast(`c`)`	c`.signalAll()`	`Monitor.PulseAll(`o`)`	
	o`.notifyAll()`		

Concurrency Extensions in Java 1.5

`java.util.concurrent`

- Adoptation of Dough Lea's utility classes [Lea 00]
- Synchronization: *semaphores*, *barriers*, *latches*, *buffers*
- Execution control: *thread pools*, *futures*

`java.util.concurrent.atomic`

- Simple atomic operations on scalar values
- *Read-compare-update* operations for non-blocking synchr.
- May be implemented by monitors, OS, or hardware

`java.util.concurrent.locks`

- Basic locking mechanisms for critical regions (as in C#)
- Provides *condition queues* (as in Pthreads)
- Uses a primitive event-mechanism for implementation

Java Semaphores

- Generalized Dijkstra semaphores
- Lots of *peek-and-poke* operations

Operations

- A Java `Semaphore` consists of:
  - A *permission count* s $(s \geq 0)$
  - A queue of waiting threads
  - Queue may be *(strongly) fair* (FIFO)
- Operations

  `s.acquire()` $\langle s > 0 \rightarrow s := s - 1 \rangle$

  `s.release()` $\langle s := s + 1 \rangle$

  `s.acquire(`n`)` $\langle s \geq n \rightarrow s := s - n \rangle$

  `s.release(`n`)` $\langle s := s + n \rangle$

Java Condition Queues

```
class BoundedBuffer {
  final Lock mutex = new ReentrantLock();
  final Condition notFull  = mutex.newCondition();
  final Condition notEmpty = mutex.newCondition();

  final Object[] items = new Object[100];
  int putptr, takeptr, count;

  public void put(Object x) throws InterruptedException {
    mutex.lock();
    try {
      while (count == items.length)
        notFull.await();
      items[putptr] = x;
      if (++putptr == items.length) putptr = 0;
      ++count;
      notEmpty.signal();
    } finally {
      mutex.unlock();
    }
  }
```

⋮

Synchronization primitives Win32 I

Semaphore

- A *semaphore* M consists of:
  - A semphore value s $(s \geq 0)$
- Operations

  `WaitFor`(S) $\langle\, s > 0 \rightarrow s := s - 1 \,\rangle$

  `Release`(S, n) $\langle\, s := s + n \,\rangle$

Synchronization primitives Win32 II

Mutex

- A *mutex* M consists of:
  - An owner thread o
  - A reservation count c $\quad (c \geq 0)$
- Operations (callled by thread tid):

`WaitFor`(M) $\quad \langle\, o = NIL \vee o = tid \rightarrow o := tid;\ \ c := c + 1 \,\rangle$

`Release`(M) $\quad \langle\, c := c - 1;$ **if** $c = 0$ **then** $o := NIL \,\rangle \qquad [o = tid]$

Synchronization primitives Win32 III

Event

- An *event (semaphore)* E consists of:
  - A flag c $\quad (c \in \{0, 1\})$
- May be created as:
  - Auto reset event E_A
  - Manual reset evetn E_M
- Operations

`Set`(E) $\quad \langle\, c := 1 \,\rangle$

`Reset`(E) $\quad \langle\, c := 0 \,\rangle$

`WaitFor`(E_A) $\quad \langle\, c = 1 \rightarrow c := 0 \,\rangle$

`Release`(E_M) $\quad \langle$ **await** $c = 1 \,\rangle$