4.1 Concurrency in Java

• Unlike C/C++, Java has built in concurrency support;
• Various concurrency patterns available.

4.1.1 Java Objects and Concurrency

• Every Java Object has a lock
• Very convenient! but obvious space overhead
• Can always lock an object with synchronized reserved word
• JVM implements 'thin locks', status bit with every object, 0 or 1, only 1 if ever locked
• Benefit+ No accidental double locking
• Benefit+ Unlocks implicit, these are Scoped Locks
• Benefit+ Thread specific data becomes private local data to object

4.1.2 Built in CV

• obj.wait();
• obj.wait(long timeout);
• obj.notify();
• obj.notifyAll();

4.1.3 Using Java Threads

• Simply Extend Thread;
• implement run() method;
• call start() on the object;
• Each thread has name, priority, and more: http://java.sun.com/j2se/1.3/docs/api/java/lang/Thread.html;
4.1.4 Note on Priority

- Unlike UNIX, higher priority == higher value!
- `setPriority(int)`, `getPriority()`, example `setPriority(Thread.MAX_PRIORITY)`
- If any thread is runnable at level i, run instead of any thread less than i
- Fixed priority scheduling, although not guaranteed to always hold
  Use for performance reasons, NOT safety
- Java will not change priority levels on you

4.2 Concurrency and Java 1.5 aka 'Tiger'

- Built on Doug Lea’s concurrency library;

4.2.1 More Concurrency Constructs

- Semaphores
  Ordinary counting
  `acquire()`, `tryAcquire()`, `release()`
  Fair (FIFO) ordering
- Linked Blocking Queue
  Blocks on put() if full, Blocks on take() if empty
  Allows for producer consumer threads to add and remove work from a shared structure
  Linked implementation, queue does not need a limit
  Does allow for max capacity
  WebServer example, through put declines after some number of clients
  Solution: Use Blocking Queue, reject new clients by setting capacity of pipeline
- Array Blocking Queue
  Blocks on put() if full, Blocks on take() if empty
  Same idea, except an array implementation
  Ideal for fixed number of tasks
- Synchronus Queue
  Each put() waits for a take(), Also called a Rendezvous channel
  If you come back from a put() you can be sure there was a take()
  CSP - Communicating Sequential Processes
  Tony Hoare, inspired a language called OCCAM
- Priority Blocking Queue
  Unbounded Queue, based on heap
  Head = item with 'lowest priority'
  Useful for concurrent simulation applications
• Delay Queue
  Time based scheduling queue
  Only expired elements can be removed
  Head = Element that expired furthest into the past
  Element is expired when its getDelay(TimeUnit) method returns 0, -1
  Useful for simulators or when managing objects with timeouts

• Copy on Write ArrayList
  Mutations on this list copy the entire backing array, updating one element
  Cost of copying array
  useful when traversals vastly overwhelm new changes
  useful when you do not want to synchronize traversals

• Exchanger
  Simple rendezvous
  Each thread gives object to exchange, gets other
  yours = exchanger.exchange(mine)

• Barrier
  all threads reach synch point before continuing, 'Barrier'
  Very common for loops and scientific apps
  Also for SOR (Successive Over Relaxation) aka Gaussian Smoothing
  Each location gets possibly weighted average of neighboring locations
  Image processing, convolutions, etc..
  Barrier code example in slides

• FutureTask
  Asynchronously executes some function to compute value
  run(), get(), cancel(), isDone()
  Way to set up synch points and check if Future is complete
  if (f.isDone()) ...
  v1 = f.get(), v2 = g.get(), waits for tasks to complete

4.3 Thread Pools

• Group of always living threads used repeatedly
• Example: Servers don’t create or destroy threads, too costly
• Instead keep a ‘pool’ of threads and take a thread when a new task arrives
• Benefit+ Faster with many tasks
• Also limits max threads
4.4 Further Links

- Concurrent Java 1.5 Package Listing
  http://java.sun.com/j2se/1.5.0/docs/api/java/util/concurrent/package-summary.html

- Info about OCCAM
  http://www.wotug.org/occam/

- Original Slides for this lecture:
  http://www.cs.umass.edu/ emery/classes/cmpsci691w-spring2006/lectures/cmpsci691w-lecture04-java.pdf