What is Concurrent Programming?

Concurrent Program: When two or more ‘processes’ cooperate to achieve a common goal.
‘Processes’ are also called ‘threads’ or ‘tasks’. In this course we use the terms interchangibly.

• Multiple threads of control
  * A number of sequential programs (i.e. ‘ordinary’ programs work together to achieve a goal

• Inter-process Communication
  * Shared variables
  * Message passing

• Synchronization
  * Mutual exclusion — processes must execute their critical sections one at a time.
  * Conditional synchronization — processes wait until a condition is true.

Note: Concurrent programming does not require multi-processors.

Computing Today
Parallel hardware

- Moore’s law meets the law of diminishing returns
- It’s hard to use 1 billion transistors effectively for a single CPU
- Instead manufacturers are putting 2 or 4 CPUs on a chip (multi-core)
- How many will it be in 10 years?
- And they are executing multiple threads with each CPU
  * Either by context switching or by
  * simultaneous multithreading (Intel calls it hyperthreading)
- Even the PS3 game system has 9 processors on one chip.
• Programs won’t go any faster on parallel hardware unless they are parallel programs.

**Distributed Computing**

• The internet and other networks mean that applications are distributed across a network.

• The different parts of such applications must communicate with each other.

**Big problems**

Advances in computing only fuel the need for “big iron” computing.

• Such problems require parallel computation for timely solution

• Example: airplane wings used to be designed in 2-D cross-
Computers allow 3-D modelling of the whole plane, leading to novel designs.

- Example: Modelling climate change.
• Example: Sesmic oil exploration.

• Example: Bioinformatics. The Blue Gene super computer has 32000 processors
Hardware is inherently parallel

As we move to behavioural hardware design, we find that hardware design is parallel programming.

**Why Use Concurrent Programs?**

- Faster processing (when $\geq 1$ processor)
- More effective use of resources (e.g. disks)
- Faster response to user
- System is conceptually concurrent
- Fault tolerance
- System is distributed
What’s different about it?

- Program steps from different processes may be inter-leaved or concurrent.
- Need to consider other processes.
- Usual reasoning rules don’t apply.
- Programs may fail partially.
- Testing is never sufficient.
Course Outline (Approximate)

- Following text reasonably closely
- Programming will be done in Java

<table>
<thead>
<tr>
<th>Topic</th>
<th>Lectures (Very Approx)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Architectures &amp; Applications</td>
<td>2</td>
</tr>
<tr>
<td>Processes &amp; Synchronization</td>
<td>3</td>
</tr>
<tr>
<td>Locks &amp; Barriers</td>
<td>3</td>
</tr>
<tr>
<td>Semaphores</td>
<td>2</td>
</tr>
<tr>
<td>Monitors</td>
<td>3</td>
</tr>
<tr>
<td>Message passing</td>
<td>3</td>
</tr>
<tr>
<td>Software Transactional Memory</td>
<td>2</td>
</tr>
<tr>
<td>RPC &amp; Rendezvous</td>
<td>2</td>
</tr>
<tr>
<td>Interaction Paradigms</td>
<td>4</td>
</tr>
<tr>
<td>Scientific Computing</td>
<td>4</td>
</tr>
<tr>
<td>Real-time systems (?)</td>
<td>4</td>
</tr>
<tr>
<td>Model checking (?)</td>
<td>3</td>
</tr>
<tr>
<td>Transaction processing (?)</td>
<td>2</td>
</tr>
</tbody>
</table>