Concurrent Programming (COMP2007) – Part II
Lecture 1: Introduction to Concurrency

Kevin Bryson
K.Bryson@cs.ucl.ac.uk
Room 8.04

Lecture Overview

- The organization of the course
  - Lectures, labs & coursework.
- Resources for the course
  - Recommended textbooks
  - Course web site

- Introduction to Concurrency
  - What is a concurrent program?
  - Historical overview and getting a ‘feel’ for CPU time
  - Important application areas
  - An example of why concurrency matters …
  - Understanding the key difference between parallelism and logical concurrency.

Course Organization

- Three lectures per week:
  Monday 14:00 -> 15:00
  Thursday 14:00 -> 16:00

- Coursework will consist of:
  - A moodle quiz about concurrency worth 5%. Will be given out after Reading Week (after Graham's last deadline).
  - A group mini-project worth 5%.
  - These projects will be 'anonymously peer reviewed' to gain marks for both reviewers and project authors.

- This year we are going to try to run a 'Question & Answer' discussion group on Moodle instead of labs … we will see how this works.

Course Textbooks

- **Concurrency: state models & Java programs.**
  by J. Magee and J. Kramer.
  John Wiley & Sons Ltd, 2006
  (ISBN 0-470-09355-2)

Any modern Java Thread book for Java 5 or 6 concurrency:

- **Java Threads** by Scott Oak & Henry Wong.
  O'Reilly, 2004
  (ISBN: 0-596-00782-5)

Course Website

- The UCL Moodle System has a COMP2007 course
  http://moodle.ucl.ac.uk/

The ENROLMENT KEY is: ATOMIC

- Moodle pages will contain:
  - These notes in PDF (single page per slide … see instructions on the Moodle site to print multiple slides per page).
  - Questions & Answers discussion board.
  - Moodle quizzes & group coursework registration/description.
  - Other bits and pieces

- But it is important to make your own notes as well since I will be saying more about concurrency than is written on the overheads …

So what is a Concurrent Program ?

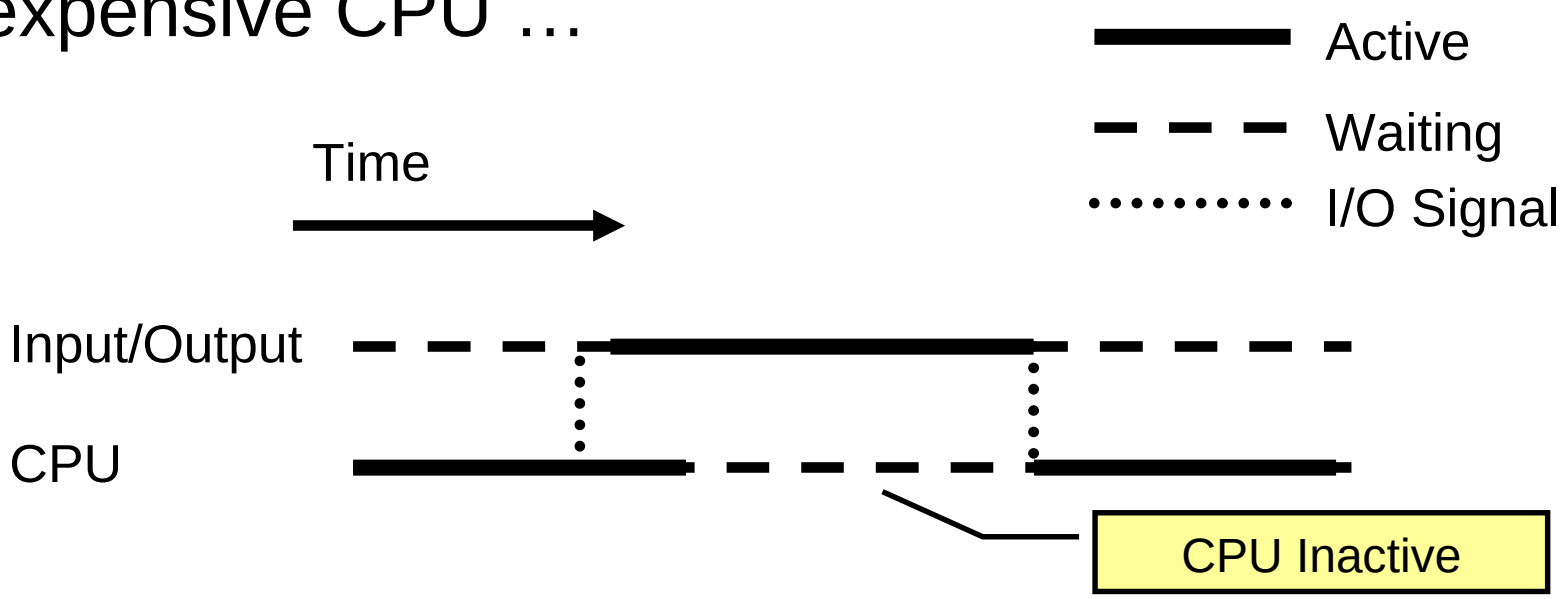
- A sequential program consists of instructions that are *totally ordered.* The instructions are carried out in a precise sequence and the process is deterministic.

- A concurrent program is just a set of ordinary sequential programs executed at the same time (i.e. concurrently).

- We will more formally introduce this idea of concurrency a bit later – but first some of the motivations behind it.

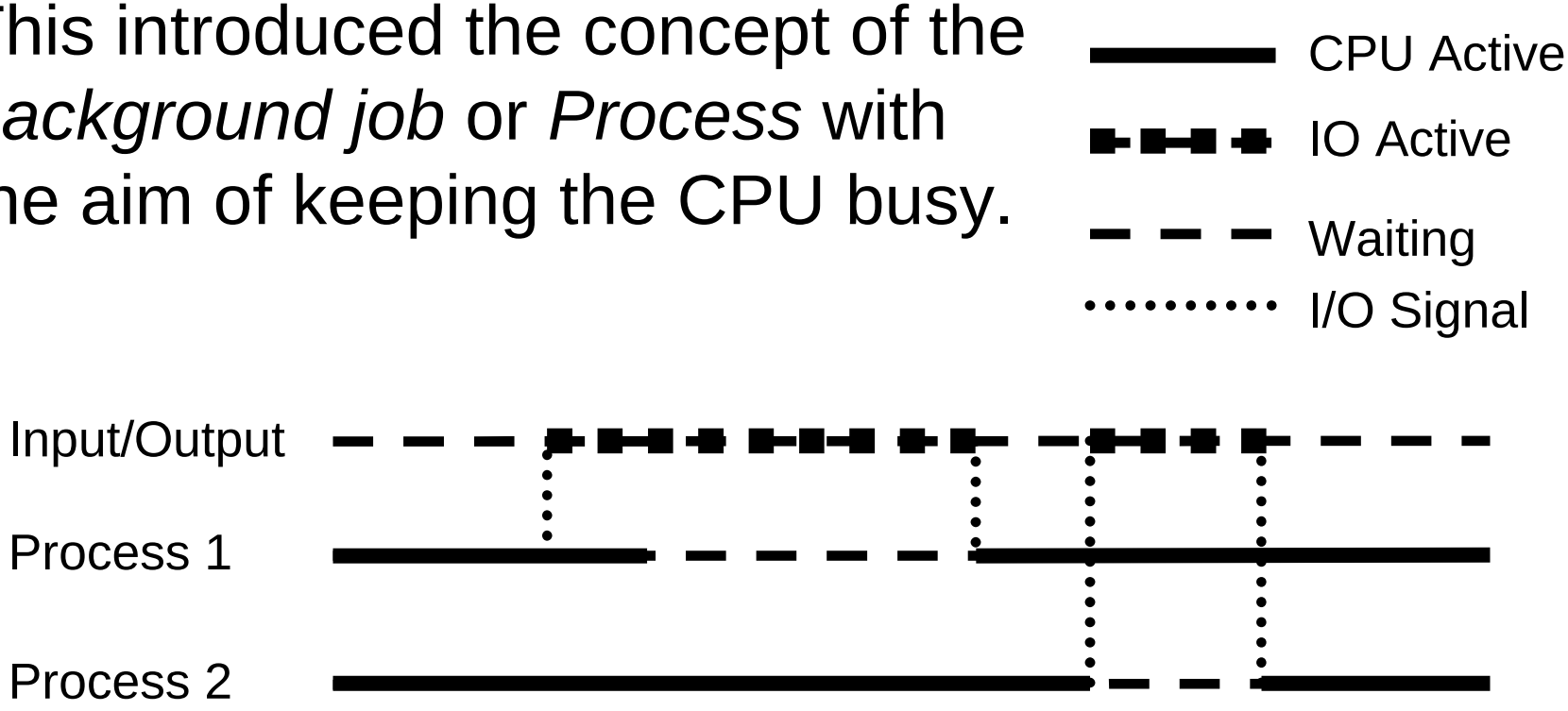
Doing more than one thing at a time …

- Concurrency was first developed within the research field of operating systems.
- Initial batch machines relied on an operator loading one program onto the machine at a time.
- This lead to very inefficient use of the very the expensive CPU …



First *Time Sharing* Operating Systems

- One of the first time-sharing or multi-tasking operating systems was CTSS (Compatible Time Sharing System) developed at MIT in 1961.
- This introduced the concept of the *background job* or *Process* with the aim of keeping the CPU busy.

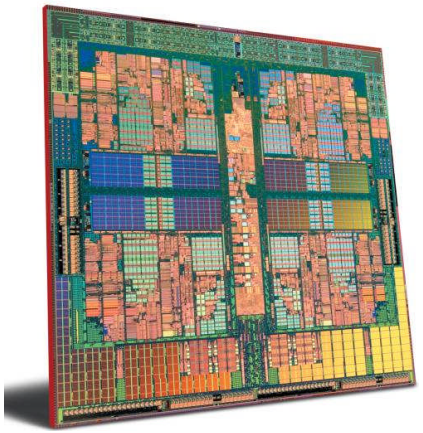


Activity: Thought Experiment

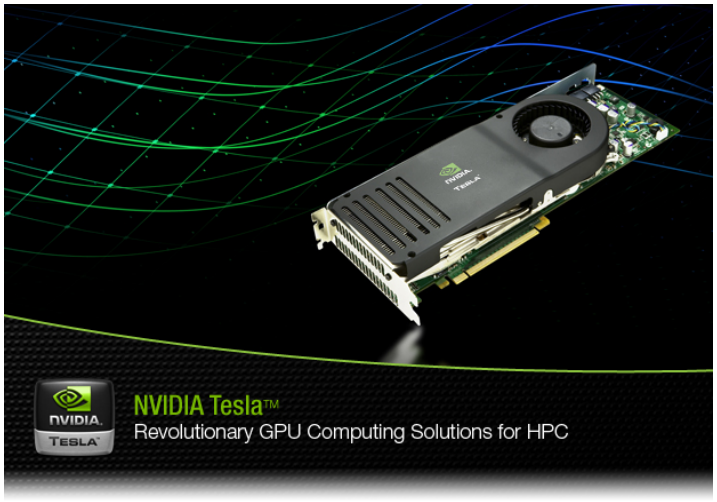
- Assume nobody thought up the idea of a time-sharing operating system. That is, a computer would only be able to run a single program at a time.
- Assume that it is only operating system research that has seen ill-fortune, our current PC technology is the same.
- We are running a (single) program which relies on input from the keyboard which then takes 10 microsecond to process.
- If we type at 240 characters per minute – how active is the processor?
- A better ‘feel’ for the times involved may be obtained if we assume that time in our universe has been ‘diluted’ so that every microsecond seems like a second – how long is the processor busy and how long does each keystroke now take between inputs?

Parallel concurrency is becoming ever more important as new multi-core CPUs and GPUs become common

- New multicore CPU hardware means that parallel concurrency is everywhere now …

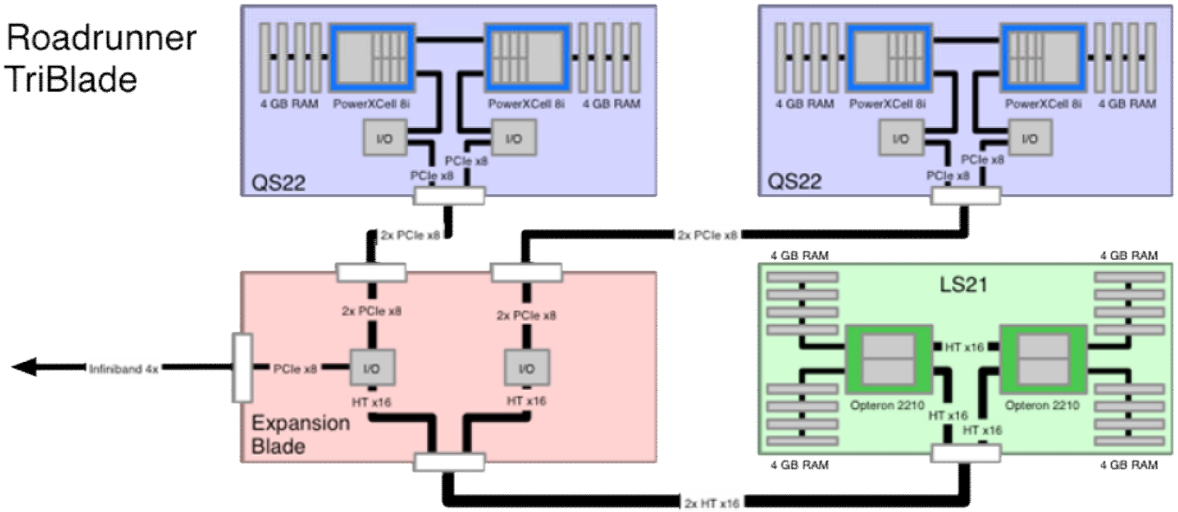


- NVIDIA Tesla C870 GPU has 128 cores and can deliver over 500 gigaflops of peak floating point performance. It can be programmed using a special C language called CUDA.



Distributed concurrency is also now common with massively parallel clusters and ‘clouds’ …

RoadRunner achieved 1.456 petaflops in Nov. 2008 making it the world’s fastest machine. Hybrid design composed of 12,960 IBM PowerXCell 8i CPUs and 6,480 AMD Opteron dual-core processors – all networked together.

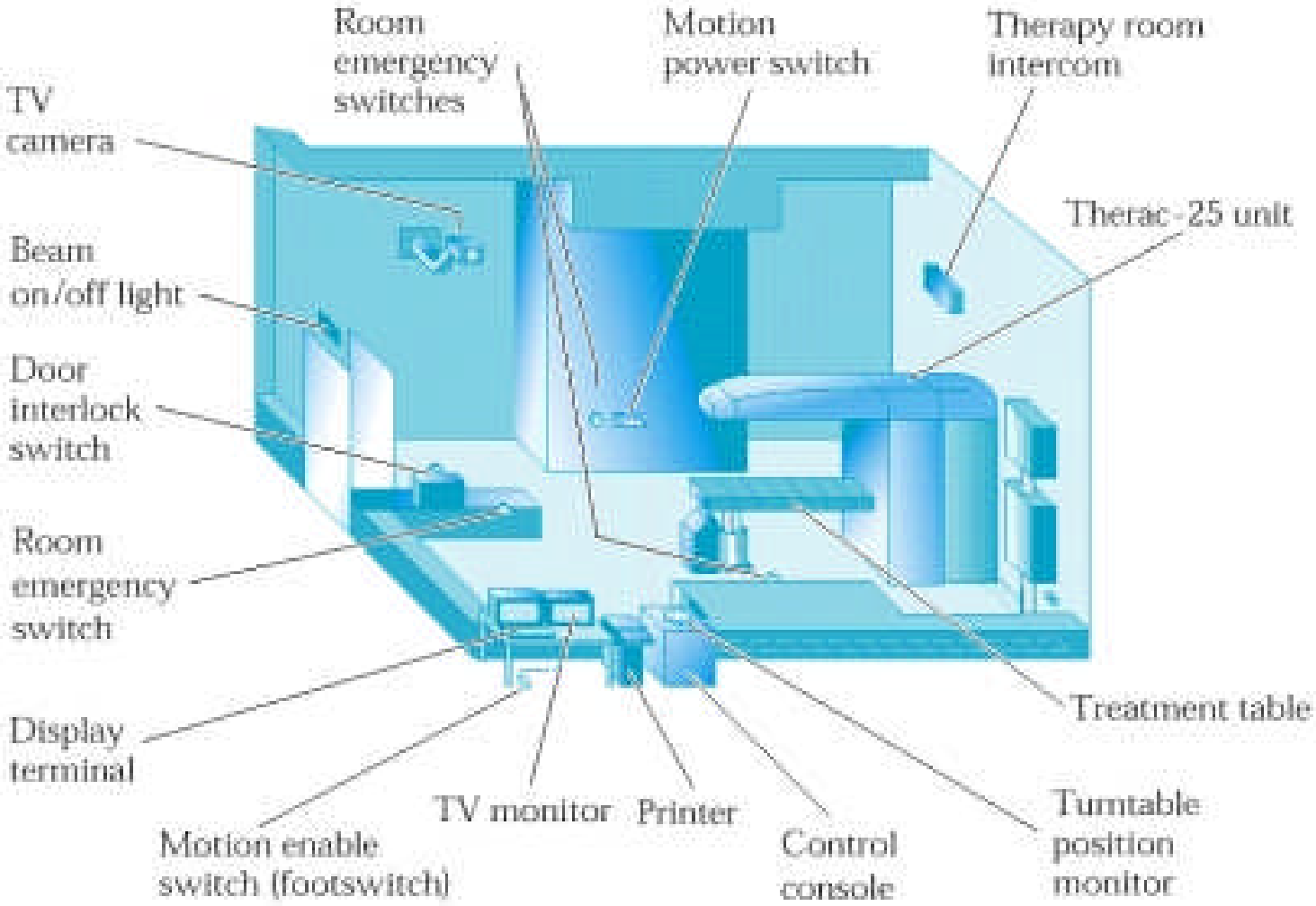


Embedded processors employ virtual (interleaved) concurrency to handle multiple tasks arising naturally from multiple sensors

- UC Berkeley Tmote Sky
- Wireless mobile sensor
- It needs to react to lots of sensors simultaneously sending info …
- Uses a special operating system (tinyos) and programming language based around a concurrency model.

Also distributed parallel concurrency arising when ad-hoc networks are created to handle environmental sensing.

Example of an embedded computer controller handling lots of concurrent sensing/tasks: Therac-25 Radiation Therapy Machine



At least 5 people died from massive radiation overdoses between 1985 and 1987 while undergoing radiation therapy on these machines

- Who killed these people ?
- A software engineer (amongst others including engineers, managers, QA testers, mechanical engineers, etc.)
- How ?
- They didn’t understand the complexity of concurrency (amongst other problems including software reuse …)

Therac-25 Radiation Therapy Software

- Fine during formal testing …
- Went into production … seemed to work well.
- The operators become more and more efficient at using the interfaces … then they became a bit too efficient … pushing buttons a bit too fast …
- And the software could not handle it because of concurrenty bugs called ‘race conditions’ … resulting in a metal target not being put into place … resulting in massive overdoses of radiation.
- But the software company showed that the software was working fine … look … must be mechanical failure ?
- It took two years and five people’s lives before the company fully investigated the software … and found concurrency bugs which had not shown up in previous systems due to hardware interlocks being present.

But maybe they were not very good software engineers? Or maybe concurrency is difficult …

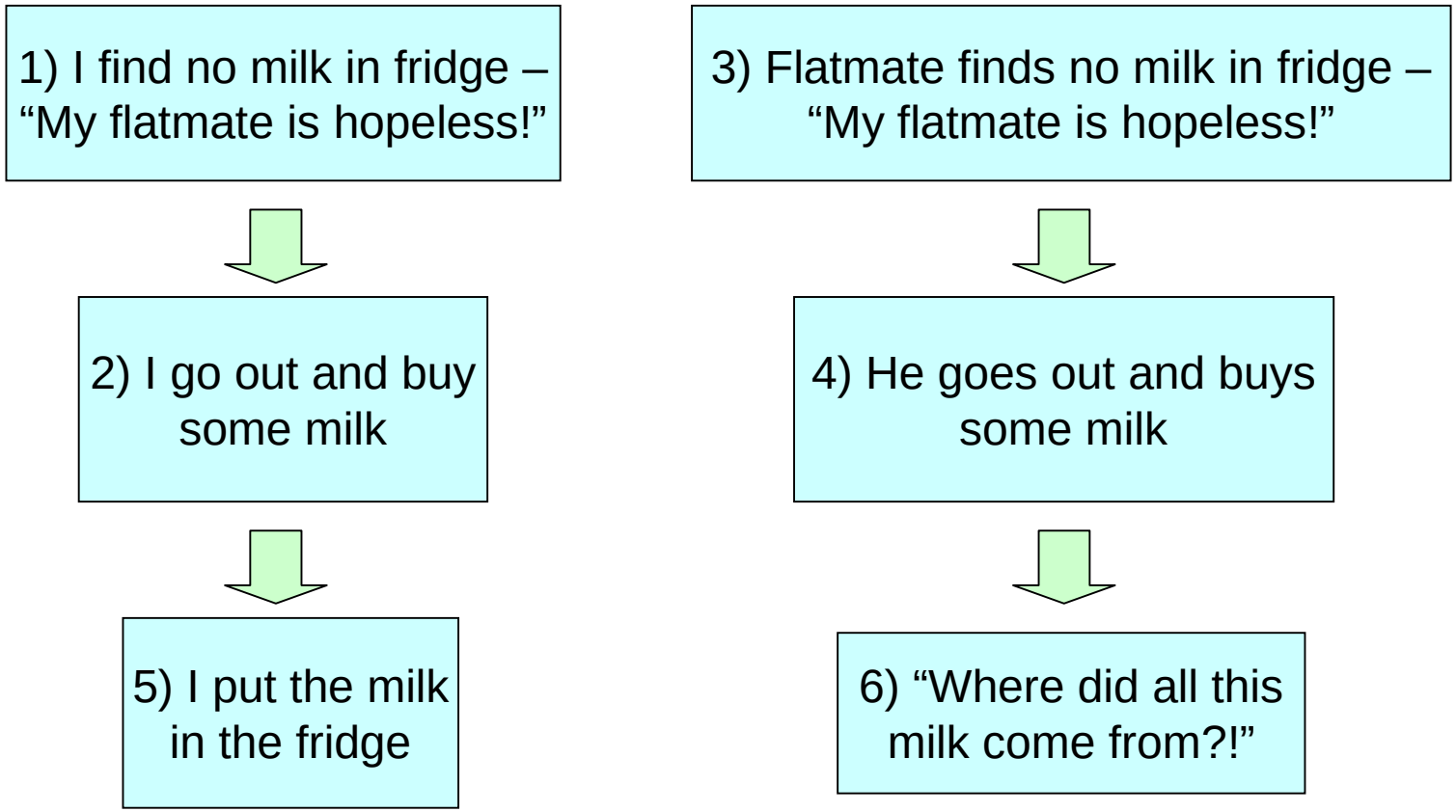
```
public class Holder {
  private int n;

  public Holder (int n) {
     this.n = n;
  }

  public void my_test() {
    if (n != n) {
      System.out.println(“I’m confused ... !");
    }
  }
}
```

Let’s have votes – who thinks the message could be printed ?

“Why is there too much milk in the fridge?” An example of two concurrent processes …



Can you think of a mechanism to avoid this undesirable behaviour?

The influence of uniprocessor architectures on our current programming mind-set

- Sequential programming was strongly influenced by uniprocessor architectures:
  - Single CPU+ RAM and bus
  - Programs and data in RAM
  - Processor executes a repeating cycle fetching instructions from the RAM
- Execution of one instruction at a time (uni-processor)

Sequential Programming

- Sequential programs are *totally ordered*
  - The programs tells you in which order the instructions will be executed
- Sequential programs are *deterministic*
  - With the same input data they will execute the same sequence of instructions and will always produce the same result

- ... this is **not true** for concurrent programming
- “Concurrent programming frees the programmer from the uni-processor (Von Neumann) architecture mind-set”

So what is Concurrency?

- It is a key abstraction in Computer Science
- Relevant to hardware design, operating systems, multiprocessors, distributed computing, programming and design

- *Concurrency is the ability to run multiple activities simultaneously or in parallel*

Why is Concurrent Programming required?

- Performance gain from multiprocessing hardware
  - Parallelism
- Increased application throughput
  - A blocking I/O call only blocks one thread
- Increased application responsiveness
  - High priority thread for user requests
- More appropriate structure
  - For programs which control multiple activities and handle multiple events
- Embedded systems for driving hardware (like Therac-25)
  - Equipment might naturally consist of multiple sensors/activators.
- Distributed systems

Concurrency isn't just parallelism …

- Parallelism
  - Physically simultaneous processing
  - Involves multiple processing elements or independent device operations
  - Example: processes running on a multiprocessor machine
- Interleaved Concurrency
  - Logically simultaneous processing
  - Does not imply multiple processing elements.
  - Could be interleaved execution on single processing element.
  - Example: processes running on a single processor (like the original operating system scenario)

Parallelism vs Concurrency

- *Parallelism* means concurrent tasks running on different processors (i.e. ongoing at exactly the same time).
- *Interleaved or logical concurrency* means a single processor that is carrying out multiple tasks simultaneously by switching rapidly between them.

- Most of the same principles and techniques apply to both parallelism and interleaved execution and so the *abstraction* of ‘logical concurrency’ is useful for both scenarios.

- The next lecture will introduce a more formal definition of this.

Lecture Summary

- You should now be aware that concurrency is everywhere in computer science from the tiniest multi-core CPU chips to the coordinated interaction of different pieces of software over the vastness of the Internet.

- Concurrency is a ‘challenging’ subject and concurrent programs can have some very counter-intuitive behaviour (the rather odd program shown previously).
- Concurrency bugs ARE VERY DIFFICULT TO DETECT and can have fatal consequences (Therac-25).

- So concurrency is everywhere … concurrent software is difficult to write … and concurrency bugs are virtually impossible to detect once they are present … mmm.