Mobile Application Development

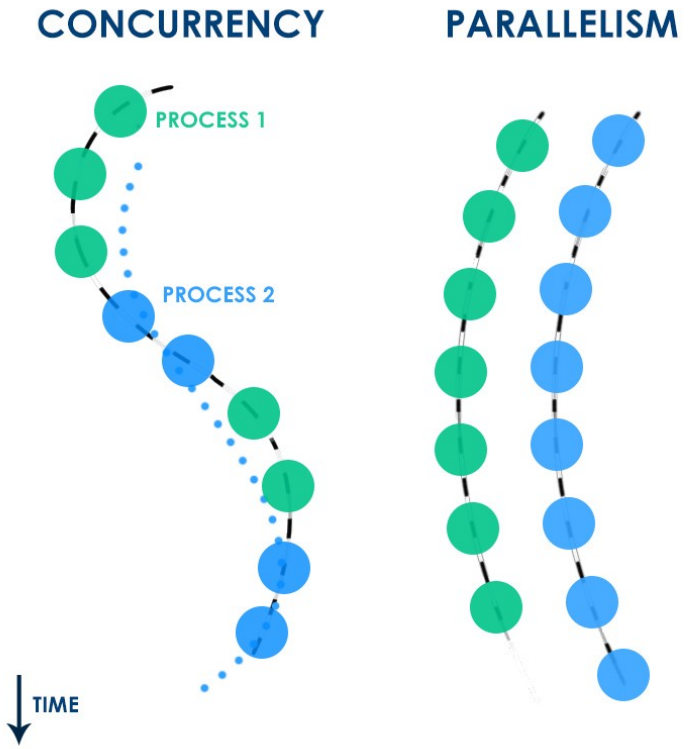
Produced by

David Drohan (ddrohan@wit.ie)

Department of Computing & Mathematics
Waterford Institute of Technology
http://www.wit.ie

Waterford Institute *of* Technology
INSTITIÚID TEICNEOLAÍOCHTA PHORT LÁIRGE

Introducing Concurrency



Agenda

- Background (What, Why & How)
- Concurrency & Parallelism
- Models for Concurrent Programming
- Processes & Threads
- Synchronous and Asynchronous Programming

Background - What

Concurrency (computer science)

From Wikipedia, the free encyclopedia

For a more practical discussion, see Concurrent computing.

In computer science, **concurrency** is the ability of different parts or units of a program, algorithm, or problem to be executed out-of-order or in partial order, without affecting the final outcome. This allows for parallel execution of the concurrent units, which can significantly improve overall speed of the execution in multi-processor and multi-core systems. In more technical terms, concurrency refers to the decomposability property of a program, algorithm, or problem into order-independent or partially-ordered components or units.[1]

- multiple computations are happening at the same time, *but not necessarily in parallel*

Background - Why

- Concurrency is everywhere in modern programming, whether we like it or not:
  - Multiple computers on a network
  - Multiple applications running on one computer
  - Multiple processors in a computer (multi-core processors)
- In fact, concurrency is essential in modern programming:
  - Web Sites / Web Apps handling multiple simultaneous users
  - Mobile apps need to do some of their processing “in the cloud”
  - GUIs/IDEs almost always require background work that does not interrupt the user. E.g, AS compiles your code while you’re still editing it

Background - Why

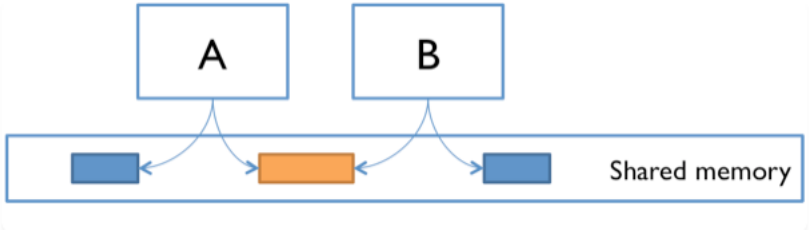
- Being able to program with concurrency is important now and will still be important in the future
- Processor clock speeds are no longer increasing, instead, we're getting more cores with each new generation of chips. (quad-core etc.)
- So in the future, in order to get a computation to run faster, we'll have to split up a computation into *concurrent pieces*.

Background – How (Models for Concurrent Programming)

- There are two common models for concurrent programming:
  - **shared memory** and
  - **message passing**
- In the ***shared memory*** model of concurrency, concurrent modules interact by reading and writing **shared objects** in memory.
- In the ***message-passing*** model, concurrent modules interact by sending messages to each other through a communication channel. Modules send off messages, and incoming messages to each module are queued up for handling.

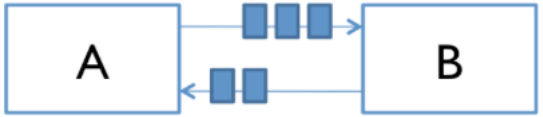
Background – How (Shared Memory Model Examples)

- A and B might be two processors (or processor cores) in the same computer, sharing the same physical memory.
- A and B might be two programs running on the same computer, sharing a common filesystem with files they can read and write.
- A and B might be two threads in the same program (we'll explain what a thread is soon), sharing the same objects.



Background – How (Message Passing Model Examples)

- A and B might be two computers in a network, communicating by network connections.
- A and B might be a web browser and a web server – A opens a connection to B, asks for a web page, and B sends the web page data back to A.
- A and B might be an instant messaging client and server.



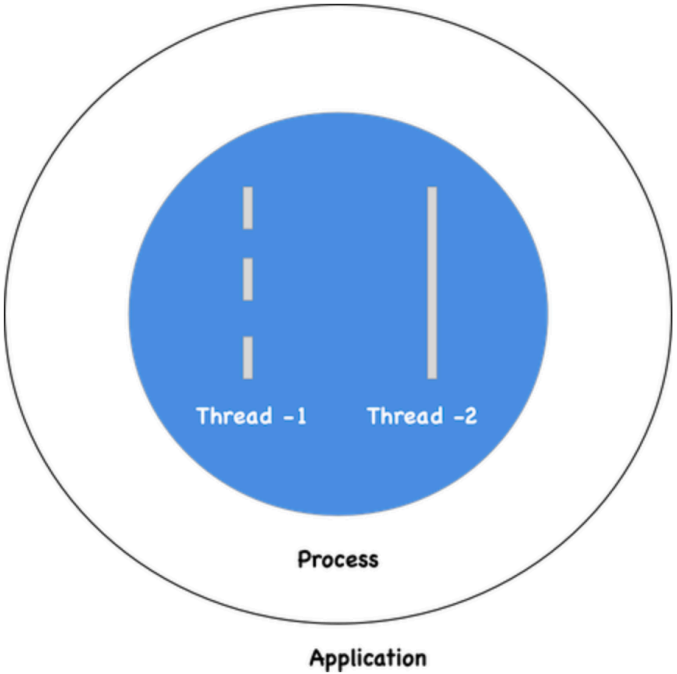
Processes & Threads

- The **message-passing** and **shared-memory** models are about how concurrent modules communicate. The concurrent modules themselves come in two different kinds:
  - **processes** and
  - **Threads**
- Both **processes** and **threads** are independent sequences of execution.
- The typical difference is that threads (of the same process) run in a **shared** memory space, while processes run in **separate** memory spaces.

Processes & Threads

- **Threads** are a sequence of execution of code which can be executed independently of one another. *It is the smallest unit of tasks that can be executed by an OS*. A program can be single threaded or multi-threaded.
- A **process** is an instance of a running program. A program can have multiple processes. **A process usually starts with a single thread** i.e. a primary thread but later down the line of execution it can create multiple threads.

Processes & Threads



Distribution of Processes and Threads in an application.

Synchronous and Asynchronous Programming

- In a **synchronous** programming model, tasks are executed one after another. Each task waits for any previous task to complete and then gets executed.

- In an **asynchronous** programming model, when one task gets executed, you could switch to a different task without waiting for the previous to get completed.

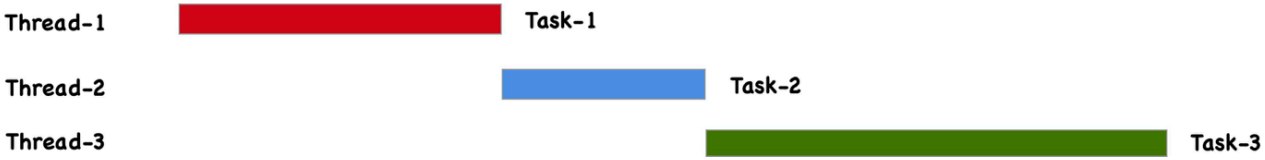
Synchronous and Asynchronous Programming

Synchronous

Single Threaded:

Each task gets executed one after another. Each task waits for its previous task to get executed.

Multi-Threaded:



Tasks get executed in different threads but wait for any other executing tasks on any other thread.

Synchronous and Asynchronous Programming

Asynchronous

Single Threaded:

Tasks start executing without waiting for a different task to finish. At a given time a single task gets executed.

Multi-Threaded:

Tasks get executed in different threads without waiting for any tasks and independently finish off their executions.

Synchronous and Asynchronous Programming

- What is the role of synchronous and asynchronous programming in concurrency and parallelism?

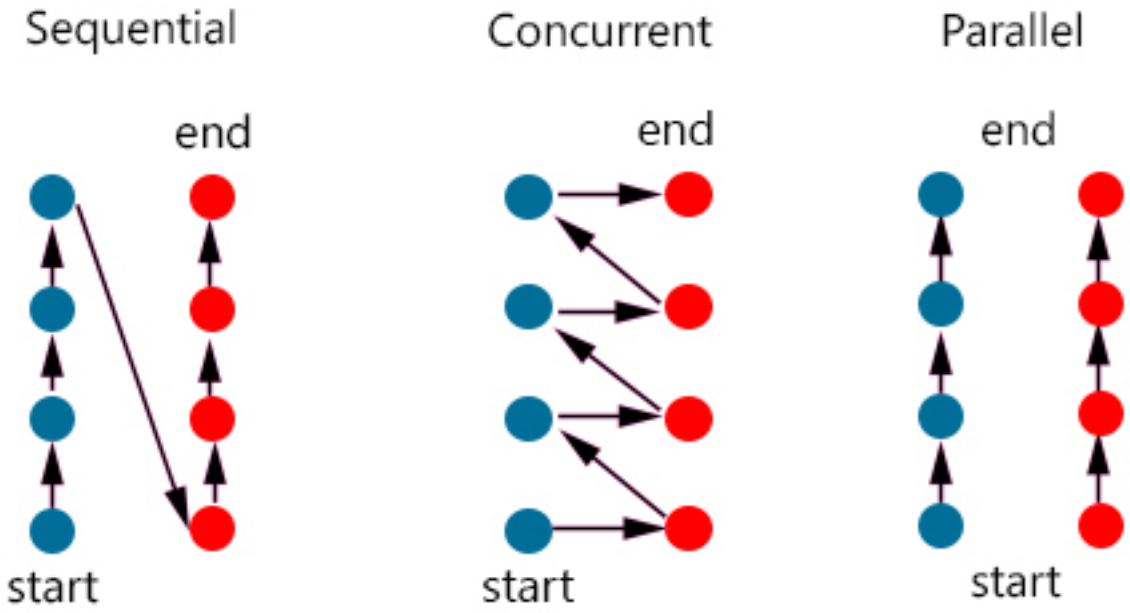
  - Asynchronous programming model helps us to achieve **concurrency**.

  - Asynchronous programming model in a multi-threaded environment is a way to achieve **parallelism**.

In a Nutshell…

- **Concurrency and Parallelism**
  - Way tasks are executed
- **Single Threaded and Multi-Threaded**
  - The environment of task execution
- **Synchronous and Asynchronous**
  - Programming model

In a Nutshell…



References

Sources:

https://web.mit.edu/6.005/www/fa14/classes/17-concurrency/

https://medium.com/swift-india/concurrency-parallelism-threads-processes-async-and-sync-related-39fd951bc61d


Thanks.