Introduction to Concurrency

What is concurrency?

- Usually concurrent means:
  - Adj. running together in space, as parallel lines, going on side by side, as proceedings occurring togeth-er, as events or circumstances, existing or arising togeth-er, compact, associated (English Longman Dictionary) . . .
  - Information: our mind is actually information regarding an event that occurs for a fraction of a second.

If there is no observer who can identify two events as being in strict temporal sequence (i.e. one event has fully terminated before the other one started) then those two events are considered concurrent.

A computer scientist’s view on concurrency

- Terminology for physically concurrent machines/operating systems:
  - Overlapped I/O and computation
  - Multi-processing
  - Multiprogramming
  - Multiprocessing

- Multi-tasking
  - Multiple processes run on one machine
  - Multi-processing
  - Parallel/serial processors

- General network architectures
  - Multiple communicating distributed entities
  - Distributed networks

Forms of concurrency

- SISD
  - Single instruction, single data
  - Serial processors

- MIMD
  - Multiple instruction, multiple data
  - Parallel processors

- SIMD
  - Single instruction, multiple data

A computer scientist’s view on concurrency

- Overlapped I/O and computation
- Multi-processing
- Multiprogramming
- Multi-tasking

- General network architectures
  - Multiple communicating distributed entities

Models and Terminology

Concurrent program abstraction

- Concurrent program = (multiple sequential programs (processes or threads) which are executed concurrently simultaneously).

Forms of concurrency

- Overlapped I/O and computation
- Multi-processing
- Multiprogramming
- Multi-tasking

Models and Terminology

Concurrent program on different abstraction levels/perspectives

- Network
  - Multiple CPUs/network nodes and other specialized subprocesses
  - Client/server networking
  - All-to-all process-to-process interactions

- Operating systems/distributed operating systems

- Architecture & Hardware
  - Processors & networks
  - High-level concurrent programming

- Assembly level concurrency programming
  - Individual instructions (and abstraction level)

- Instruction-level parallelism

Time-line or Sequence?

Consider the sequence of interaction points only:
- Non-real-time systems
- Real-time systems

Models and Terminology

The concurrent programming abstraction

- Concurrent program = (multiple sequential programs (processes or threads) which are executed concurrently simultaneously).

- Interaction occurs in form of:
  - Communication (implicit interactions) multiple concurrent execution units compete for a shared resource
  - Communication (explicit interactions)

Sequential programming delivers some fundamental components for concurrent programming but we need to add a number of further crucial concepts

References for this chapter

Forms of concurrency

- What is concurrency?
- Working definition:
  - Literally ‘concurrent’ means:
    - Occurring together, as events or circumstances; existing or arising together; conjoint, associated.

Sequential programming delivers some fundamental components for concurrent programming but we need to add a number of further crucial concepts.

A computer scientist’s view on concurrency

- Terminology for physically concurrent machines/operating systems:
  - Overlapped I/O and computation
  - Multi-processing
  - Multiprogramming
  - Multiprocessing

- Multi-tasking
  - Multiple processes run on one machine
  - Mult-processing
  - Parallel/serial processors

- General network architectures
  - Multiple communicating distributed entities
  - Distributed networks

Models and Terminology

Concurrent program on different abstraction levels/perspectives

- Network
  - Multiple CPUs/network nodes and other specialized subprocesses
  - Client/server networking
  - All-to-all process-to-process interactions

- Operating systems/distributed operating systems

- Architecture & Hardware
  - Processors & networks
  - High-level concurrent programming

- Assembly level concurrency programming
  - Individual instructions (and abstraction level)

- Instruction-level parallelism

Time-line or Sequence?

Consider the sequence of interaction points only:
- Non-real-time systems
- Real-time systems

Models and Terminology

The concurrent programming abstraction

- Concurrent program = (multiple sequential programs (processes or threads) which are executed concurrently simultaneously).

- Interaction occurs in form of:
  - Communication (implicit interactions) multiple concurrent execution units compete for a shared resource
  - Communication (explicit interactions)

Sequential programming delivers some fundamental components for concurrent programming but we need to add a number of further crucial concepts.

A computer scientist’s view on concurrency

- Terminology for physically concurrent machines/operating systems:
  - Overlapped I/O and computation
  - Multi-processing
  - Multiprogramming
  - Multiprocessing

- Multi-tasking
  - Multiple processes run on one machine
  - Mult-processing
  - Parallel/serial processors

- General network architectures
  - Multiple communicating distributed entities
  - Distributed networks

Models and Terminology

Concurrent program on different abstraction levels/perspectives

- Network
  - Multiple CPUs/network nodes and other specialized subprocesses
  - Client/server networking
  - All-to-all process-to-process interactions

- Operating systems/distributed operating systems

- Architecture & Hardware
  - Processors & networks
  - High-level concurrent programming

- Assembly level concurrency programming
  - Individual instructions (and abstraction level)

- Instruction-level parallelism

Time-line or Sequence?

Consider the sequence of interaction points only:
- Non-real-time systems
- Real-time systems

Models and Terminology

The concurrent programming abstraction

- Concurrent program = (multiple sequential programs (processes or threads) which are executed concurrently simultaneously).

- Interaction occurs in form of:
  - Communication (implicit interactions) multiple concurrent execution units compete for a shared resource
  - Communication (explicit interactions)

Sequential programming delivers some fundamental components for concurrent programming but we need to add a number of further crucial concepts.

A computer scientist’s view on concurrency

- Terminology for physically concurrent machines/operating systems:
  - Overlapped I/O and computation
  - Multi-processing
  - Multiprogramming
  - Multiprocessing

- Multi-tasking
  - Multiple processes run on one machine
  - Mult-processing
  - Parallel/serial processors

- General network architectures
  - Multiple communicating distributed entities
  - Distributed networks

Models and Terminology

Concurrent program on different abstraction levels/perspectives

- Network
  - Multiple CPUs/network nodes and other specialized subprocesses
  - Client/server networking
  - All-to-all process-to-process interactions

- Operating systems/distributed operating systems

- Architecture & Hardware
  - Processors & networks
  - High-level concurrent programming

- Assembly level concurrency programming
  - Individual instructions (and abstraction level)

- Instruction-level parallelism

Time-line or Sequence?

Consider the sequence of interaction points only:
- Non-real-time systems
- Real-time systems

Models and Terminology

The concurrent programming abstraction

- Concurrent program = (multiple sequential programs (processes or threads) which are executed concurrently simultaneously).

- Interaction occurs in form of:
  - Communication (implicit interactions) multiple concurrent execution units compete for a shared resource
  - Communication (explicit interactions)

Sequential programming delivers some fundamental components for concurrent programming but we need to add a number of further crucial concepts.

A computer scientist’s view on concurrency

- Terminology for physically concurrent machines/operating systems:
  - Overlapped I/O and computation
  - Multi-processing
  - Multiprogramming
  - Multiprocessing

- Multi-tasking
  - Multiple processes run on one machine
  - Mult-processing
  - Parallel/serial processors

- General network architectures
  - Multiple communicating distributed entities
  - Distributed networks

Models and Terminology

Concurrent program on different abstraction levels/perspectives

- Network
  - Multiple CPUs/network nodes and other specialized subprocesses
  - Client/server networking
  - All-to-all process-to-process interactions

- Operating systems/distributed operating systems

- Architecture & Hardware
  - Processors & networks
  - High-level concurrent programming

- Assembly level concurrency programming
  - Individual instructions (and abstraction level)

- Instruction-level parallelism

Time-line or Sequence?

Consider the sequence of interaction points only:
- Non-real-time systems
- Real-time systems

Models and Terminology

The concurrent programming abstraction

- Concurrent program = (multiple sequential programs (processes or threads) which are executed concurrently simultaneously).

- Interaction occurs in form of:
  - Communication (implicit interactions) multiple concurrent execution units compete for a shared resource
  - Communication (explicit interactions)

Sequential programming delivers some fundamental components for concurrent programming but we need to add a number of further crucial concepts.
Introduction to Concurrency

Models and Terminology

The concurrent programming abstraction

Correctness vs. testing in concurrent systems:

Slight changes in external trigger may (and usually does) result in completely different schedules (interleavings):

- Concurrent programs only begin as one; on natural events cannot be traced back to the start at any point; and understanding differences into the past and future.
- Missing the past and future events may lead to unexpected behavior.
- Solution: explicit events and their timing are essential.
- Conclusion: understanding all joint possibilities is essential.

Introduction to Concurrency

Models and Terminology

The concurrent programming abstraction

Safety properties

\[ P(Q) \wedge \text{Process}(Q) \Rightarrow Q(0,5) \text{ where } Q \text{ is the event that } \text{the lock is held} \]

Examples:
- Mutual exclusion on resource collisions
- Absence of deadlocks
- Specifics and/or deadlocks
- Synchronization of free capabilities

Safety properties

\[ P(Q) \wedge \text{Process}(Q) \Rightarrow \neg Q(0,5) \text{ where } Q \text{ is the event that } \text{the lock is not held} \]

Examples:
- The state of the system is to be displayed eventually
- No part of the system is to be delayed forever (dynamically)

Introduction to Concurrency

Models and Terminology

The concurrent programming abstraction

Shared memory access

- Need to be coordinated.
- Typically in real-time / embedded systems or server applications.

Introduction to Concurrency

Models and Terminology

The concurrent programming abstraction

Process states

- Process ready
- Process waiting
- Process swapped
- Process finishing

Introduction to Concurrency

Models and Terminology

The concurrent programming abstraction

Process Control Blocks

- Process ID
- Process name
- Process status
- Process control information
- Process state

Introduction to Concurrency

Models and Terminology

The concurrent programming abstraction

Atomic operations

- Atomic operations are the most suitable but have limited benefits, being tied to small systems.

Introduction to Concurrency

Models and Terminology

The concurrent programming abstraction

Correctness proofs / designs in concurrent systems rely on the assumptions of atomic operations (detailed discussion later).

- Examples:
  - Requests need to be completed eventually
  - The state of the system needs to be displayed eventually
  - No part of the system is to be delayed forever (dynamically)

Introduction to Concurrency

Models and Terminology

The concurrent programming abstraction

Safety properties

\[ P(Q) \wedge \text{Process}(Q) \Rightarrow Q(0,5) \text{ where } Q \text{ is the event that } \text{the lock is held} \]

Examples:
- Mutual exclusion on resource collisions
- Absence of deadlocks
- Specifics and/or deadlocks
- Synchronization of free capabilities

Safety properties

\[ P(Q) \wedge \text{Process}(Q) \Rightarrow \neg Q(0,5) \text{ where } Q \text{ is the event that } \text{the lock is not held} \]

Examples:
- The state of the system is to be displayed eventually
- No part of the system is to be delayed forever (dynamically)

Introduction to Concurrency

Models and Terminology

The concurrent programming abstraction

Safety properties

\[ P(Q) \wedge \text{Process}(Q) \Rightarrow Q(0,5) \text{ where } Q \text{ is the event that } \text{the lock is held} \]

Examples:
- Mutual exclusion on resource collisions
- Absence of deadlocks
- Specifics and/or deadlocks
- Synchronization of free capabilities

Safety properties

\[ P(Q) \wedge \text{Process}(Q) \Rightarrow \neg Q(0,5) \text{ where } Q \text{ is the event that } \text{the lock is not held} \]

Examples:
- The state of the system is to be displayed eventually
- No part of the system is to be delayed forever (dynamically)

Introduction to Concurrency

Models and Terminology

The concurrent programming abstraction

Atomic operations

- Atomic operations are the most suitable but have limited benefits, being tied to small systems.

Introduction to Concurrency

Models and Terminology

The concurrent programming abstraction

Correctness proofs / designs in concurrent systems rely on the assumptions of atomic operations (detailed discussion later).

- Examples:
  - Requests need to be completed eventually
  - The state of the system needs to be displayed eventually
  - No part of the system is to be delayed forever (dynamically)
Introduction to Concurrency

UNIX processes

In UNIX systems tasks are created by ‘cloning’ existing in a duplication of the current process. Essentially, the 'clone' system call creates a new process that duplicates the parent's process, including the kernel threads and memory. The parent process continues to run as usual, while the new process begins executing from the location specified by the task's initial stack.

Languages with implicit concurrency:

- Pure functional programming is side-effect free and supports short-circuit evaluations.

Explicit concurrency:

- Ada, C++, Rust, Chill, Erlang, Go, Chapel, X10, Occam, CSP, All .net languages, Java, Scala, Clojure, Modula-2, Modula-3, …

Implicit concurrency:

- Functional programming – e.g. Haskell, Caml, Miranda, and any other functional language.

Concurrent programming languages:

- Concurrent program parts: (message passing, shared memory, rpc …)
- Support for protection: (tasks, memory, devices, …)
- Support for communication: (message passing, shared memory, sp …)
- Support for synchronization: (semaphores, monitors, …)
- Support for termination management: (mutual exclusion, …)
- Support for task management: (create, terminate, …)

Concurrent programming languages:

- Concurrent program parts: (message passing, shared memory, rpc …)
- Support for protection: (tasks, memory, devices, …)
- Support for communication: (message passing, shared memory, sp …)
- Support for synchronization: (semaphores, monitors, …)
- Support for termination management: (mutual exclusion, …)
- Support for task management: (create, terminate, …)