v1.0

Sistemi Concorrenti e di Rete LS

II Facoltà di Ingegneria - Cesena
a.a 2008/2009

[module 1.1]
BASIC CONCEPTS

CONCURRENCY AND CONCURRENT SYSTEMS

- Concurrency as a main concept of many domains and systems
  - operating systems, multi-threaded and multi-process programs, distributed systems, control systems, *real-time* systems,...
  - “complex” software systems
- General definitions
  - “*In computer science, concurrency is a property of systems in which several computational processes are executing at the same time, and potentially interacting with each other.*” [ROS-97]
  - “*Concurrency is concerned with the fundamental aspects of systems of multiple, simultaneously active computing agents, that interact with one another*” [CLE-96]
- Common aspects
  - systems with multiple activities or *processes* whose execution **overlaps in time**
  - activities can have some kind of *dependencies,* therefore can **interact**

CONCURRENT PROGRAMMING

- **Concurrent programming**
  - building programs in which multiple computational activities overlap in time and typically interact in some way
- **Concurrent program**
  - finite set of *sequential* programs that can be executed in parallel, i.e. overlapped in time
    - a sequential program specifies sequential execution of a list of statements
    - the execution of a sequential program is called **process**
    - a concurrent program specifies two or more sequential programs that may be executed concurrently as *parallel processes*
  - the execution of a concurrent program is called *concurrent computation or elaboration*

CONCURRENT PROGRAMMING VS. PARALLEL PROGRAMMING

- *Parallel* programming
  - the execution of programs overlaps in time by running on separate physical processors
- *Concurrent* programming
  - the execution of programs overlaps in time *without necessarily running on separate physical processors*, by sharing for instance the same processor
    - potential or *abstract* parallelism
- *Distributed* programming
  - when processors are distributed over a network
  - no shared memory

WHY CONCURRENCY AND CONCURRENT PROGRAMMING (1/2)

- Performance improvement
  - increased application throughput
    - by exploiting parallel hardware
  - increased application responsiveness
    - by optimizing the interplay among CPU and I/O activities

- Quantitative measurement for performance: **speedup**

$$S = \frac{T_1}{T_N}$$

N is the number of processors
T_1 is the execution time of the sequential algorithm
T_N is the execution time of the parallel algorithm with N processors

AMDAHL'S LAW

- *Maximum* speedup parallelizing a system:

$$S = \frac{1}{1 - P + \frac{P}{N}}$$

P is the proportion of a program that can be made parallel
$(1 - P)$ is the proportion that cannot be parallelized (remains serial)

- Theoretically maximum for P = 1 (*linear speedup)*
  - actually there are specific cases with S > N (*super-linear)* speedup

WHY CONCURRENCY AND CONCURRENT PROGRAMMING (2/2)

- Abstraction and engineering
  - more appropriate level of abstraction for programs which interact with the environment, control multiple activities and handle multiple events
  - e.g. *reactive* systems
- Concurrency an a tool for software design and construction
  - rethinking to the way in which we solve problems
    - parallel algorithms
  - rethinking to the way in which we design and build systems
    - new level of abstraction
      - different kind of decomposition, modularization, encapsulation
- full engineering spectrum
  - modelling, implementing, testing, ...

BASIC JARGON OF CONCURRENT PROGRAMMING

- **Processes ~** a sequential program in execution
  - abstract / general concept
    - different kind of incarnation depending on the specific context
    - the basic unit of a concurrent system, single thread of control
  - synonim: *task*
    - sequence of instructions operating together as a group, unit of work
  - process execution is meant to be completely asynchronous with each other
    - different speed in executing statements ==> non-determism
- **Process interaction**
  - any non trivial concurrent program is based on *multiple* processes that need to *interact* in some way in order to achieve the objective of the system
  - basic kinds of interaction:
    - **competition / contention**, **cooperation**, **interferences**

PROCESS INTERACTION: CONTENTION (OR COMPETITION)

- Refers to interactions which are expected and necessary, but not wanted
  - typically concerns the need of coordinating the access by multiple processes to shared resources
- Two basic class of problems
  - **mutual exclusion**
    - ruling the access to shared resources by distinct processes
  - **critical sections**
    - ruling the concurrent execution of blocks of actions by distinct processes

PROCESS INTERACTION: COOPERATION

- Refers to interactions which are expected and wanted
  - they are part of the semantics of the concurrent program
- Two basic kinds
  - **synchronization**
    - concerns the explicit definition or presence of temporal relationships or dependencies among processes and among actions of distinct processes
    - introduction of specific supports for the exchange of temporal signals
  - **communication**
    - concerns the need of realising an information flow among processes, typically realised in terms of messages
    - introduction of specific supports for the exchange of messages

SYNCHRONIZATION VS. MUTUAL EXCLUSION

- Profoundly different - even if related - concepts
  - “synchronization = mutual exclusion urban legend” [BUH-05]
    - false story, still present in textbooks / research papers
  - synchronization defines a timing relationship among processes
    - mantaining time-relationships which includes actions happening at the same time or happening at the same relative rate or simply some action having to occur before another (precedence relationships)
  - mutual-exclusion defines a restriction on access to shared data
    - mutual-exclusion is meaningless if no shared data is involved
- Relationships
  - mutual-exclusion typically require some forms of *implicit synchronization*
    - blocking some actions, waiting for other actions to complete
  - synchronization does not necessarily require any kind of shared data and the mutual exclusion

ON THE DIFFICULTY OF SYNCHRONIZATION: EXAMPLE 1: “BUY-THE-MILK” PROBLEM

- “Alice and Bob live together, happily without cell-phones. Both are responsible to buy the milk when it finishes...”

Time	Alice	Bob
5:00	Arrive home	
5:05	Look in the fridge; no milk	
5:10	Leave for a grocery	
5:15		Arrive home
5:20		Look in the fridge; no milk
5:25	Buy milk	Leave for grocery
5:30	Arrive home; put milk in fridge	
5:40		Buy milk
5:45		Arrive home; oh no!

A SOLUTION: NOTES IN THE FRIDGE (1/2)

- Looking for a solution to ensure that:
  - only one person buys the milk, when there is no milk
  - someone always buys the milk, when there is no milk
- Tentative solution: using notes on the fridge!

PROGRAM for Alice & Bob

```
if (no note) then
  if (no milk) then
    leave note
    buy milk
    remove note
  fi
fi
```

  - “if you find that there is no milk and there is no note on the door of the fridge, then leave a note on the fridge’s door, go and buy milk, put the milk in the fridge, and remove your note.”
- Does it work? Not always actually...

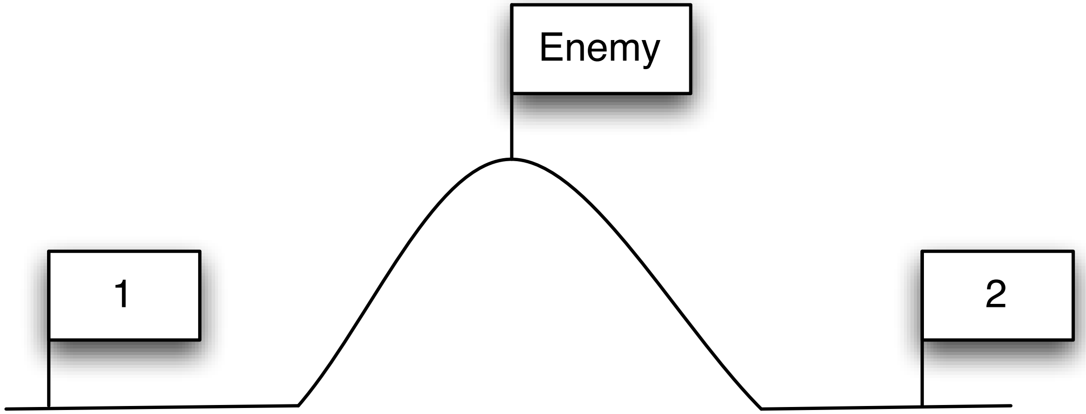
A SOLUTION: NOTES IN THE FRIDGE (2/2)
(..NOT SO EASY, ACTUALLY..)

Time	Alice	Bob
5:00	Arrive home	
5:05	Look at the fridge; no note	
5:10	...ops! need a toilet	
5:15	...still in the toilet...	Arrive home
5:20	...still in the toilet...	Look at the fridge; no note
5:21	...still in the toilet...	Look in the fridge; no milk (argh)
5:22	...still in the toilet...	leave note
5:25	...still in the toilet...	go and buy milk
5:45	look in the fridge: no milk (*)	...
5:50	leave note...	

[*] Alice does not realize that a note was put on the fridge (she is not really a good observer) and strictly follows the esablished program

ON THE DIFFICULTY OF SYNCHRONIZATION: EXAMPLE 2: “COORDINATED ATTACK” PROBLEM

- “Two camps of the same army in different locations need to decide on the exact time for a coordinated attack on the enemy camp”



- Communication between 1-2 through messangers
  - each general: send a message (messanger), wait for an ack
- If communication is not reliable, the problem has no solution

PROCESS INTERACTION: INTERFERENCES

- Refers to interactions which are neither expected, not wanted
  - producing bad effects only when the ratio among the process speeds assumes specific values (time-dependent errors)
  - the “nightmare” of concurrent programming
- **race condition** or **race hazard** or simply race
  - whenever two or more processes concurrently access and update shared resources, and the result depends on the specific order occurring in process access
- Related to two main types of programming errors
  - bad management of expected interactions
  - presence of spurious interactions not expected in the problem
- Interferences and errors in concurrent programs can lead to *critical situations* for the concurrent system in the overall.
  - **deadlock**, **starvation**, **livelock**

CRITICAL SITUATIONS

- **deadlock**
  - situation wherein two or more competing actions (processes) are waiting for the other to finish, and thus neither ever does
  - such actions typically concerns the release of a locked shared resource, the receiption of a temporal signal or a message
- **starvation**
  - situation wherein a process is blocked in an infinite waiting
  - *resource starvation* = the process is perpetually denied in accessing necessary resources.
    - without those resources, the program can never finish its task
- **livelock**
  - a livelock is similar to a deadlock, except that the states of the processes involved in the livelock constantly change with regard to one another, none progressing
  - livelock is a special case of resource starvation: the general definition only states that a specific process is not progressing

BIBLIOGRAPHY

- **[HAN-73]**
  - Per Brinch Hansen - “Concurrent Programming Concepts”, *ACM Computing Surveys*, Vol. 5, No. 4, Dec. 1973
- **[AND-83]**
  - Gregory Andrews and Fred Schneider - “Concepts and Notations for Concurrent Programming”, *ACM Computing Survey*s, Vol. 15, No. 1, March 1983
- **[CLE-96]**
  - Rance Cleaveland, Scott Smolka et al - “Strategic Directions in concurrency Research”, *ACM Computing Survey*s, Vol. 28, No. 4, Dec. 1996
- **[ROS-97]**
  - Roscoe, A. W. (1997). *The Theory and Practice of Concurrency*. Prentice Hall. ISBN 0-13-674409-5.
- **[BUH-05]**
  - Peter Buhr and Ashif Harji. “Concurrent Urban Legends”. *Concurrency and Computation: Practice and Experience.* 2005. 17:1133-1172.