Concurrent Programming

CS511

Teachers

Instructor: Eduardo Bonelli

E-mail: `ebonelli@stevens.edu`

Office hours: Wed, 6-7PM.

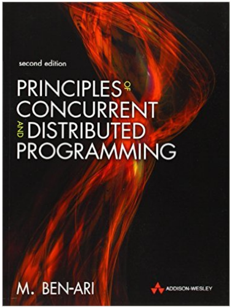
CAs: Josephine Cerino, Barry Diaz, Robert Feliciano, Matthew Turner, Federico Yacoubian, Patrick Zielinski

Ask questions!

- Feel free to interrupt and ask questions at any time
  - Your questions also help me better understand the topics
  - It also helps classmates who might have similar doubts
- Contact me by email
- Come see me during office hours

Bibliography

- Slides, above all
- Notes
- The book we use



Credits

This course has benefitted from material from the following sources:

- https://sites.google.com/site/pconctpiunq/ (Daniel Ciolek and Hernán Melgratti)
- Slides from the course on Concurrency at Chalmers (TDA382/DIT390)

General Structure of the Course

- Lectures
- Assignments:
  - Compulsory
- Exercise booklets
  - Crucial
- Quizzes
- Exams:
  - Midterm and Endterm

Read syllabus for full details

Weight of Grading Categories

Homework	(30%)
Quizzes	(20%)
Midterm	(25%)
Endterm	(25%)

Additional considerations:

- There is no curving for this course

Contents

- First half: Process synchronization in shared memory model
  - Java/Groovy
- Third Quarter: Process synchronization in message passing model
  - Erlang
- Fourth Quarter: Model checking
  - Spin (Promela)
  - Concurrency is hard! We need tool support based on formal methods

Course Essence: Process Synchronization

Course Objectives

- Understand classic problems in Concurrent Programming (CP) such as synchronization
- Understand the primary primitives used in CP
- Develop skills to be able to use these primitives in solving synchronization problems
- Understand the fundamentals of model-checking for checking properties of concurrent systems
- Apply model-checking to simple synchronization problems

The paper that began the field of concurrency[1]

Solution of a Problem in Concurrent Programming Control

E. W. Dijkstra
Technological University, Eindhoven, The Netherlands

A number of mainly independent sequential-cyclic processes with restricted means of communication with each other can be made in such a way that at any moment one and only one of them is engaged in the "critical section" of its cycle.

Introduction

Given in this paper is a solution to a problem for which, to the knowledge of the author, has been an open question since at least 1962, irrespective of the solvability. The paper consists of three parts: the problem, the solution, and the proof. Although the setting of the problem might seem somewhat academic at first, the author trusts that anyone familiar with the logical problems that arise in computer coupling will appreciate the significance of the fact that this problem indeed can be solved.

computer can only request one one-way message at a time. And only this will make the reader realize to what extent this problem is far from trivial.

The Solution

The common store consists of:

"**Boolean array** b, $c[1{:}N]$; **integer** k"

The integer k will satisfy $1 \leq k \leq N$, $b[i]$ and $c[i]$ will only be set by the ith computer; they will be inspected by the others. It is assumed that all computers are started well outside their critical sections with all Boolean arrays mentioned set to **true**; the starting value of k is immaterial.

The program for the ith computer $(1 \leq i \leq N)$ is:

```
"integer j;
Li0:  b[i] := false;
Li1:  if k ≠ i then
Li2:  begin c[i] := true;
Li3:  if b[k] then k := i;
      go to Li1
      end
        else
Li4:  begin c[i] := false;
        for j := 1 step 1 until N do
```

CACM (September, 1965)

[1] **The computer science of concurrency: the early years**, Leslie Lamport, Turing Award Lecture. CACM, Vol 58, Iss. 6, June 2015, pp 71-76, https://doi.org/10.1145/2771951

Concurrency

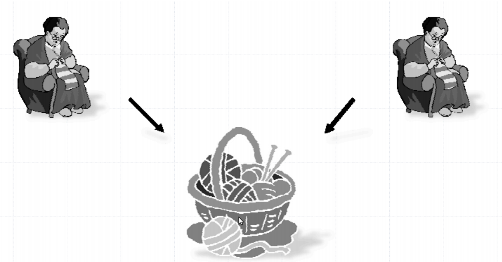
- The study of systems of interacting computer programs which share resources and run concurrently, i.e. at the same time
- Parallelism
  - Occurring physically at the same time
- Concurrency
  - Occurring logically at the same time, but could be implemented without real parallelism
- We focus on high-level synchronization
  - Distinction concurrency/parallelism is irrelevant for us

Interaction Models

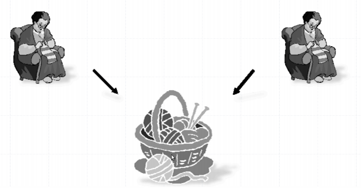
- What is process synchronization?
  - Ensure that instructions are executed in certain order
- Synchronization is irrelevant if processes do not interact with each other
- They just “do their own thing”

Process Interaction

- Concurrency, and hence process synchronization, is useful only when processes interact with each other
- What does it mean for processes to interact?
  - They share resources
- For example, two grannies and only one set of knitting needles

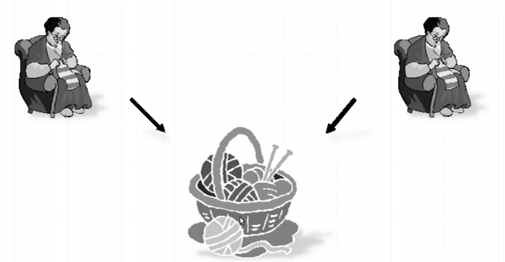


Interaction Models

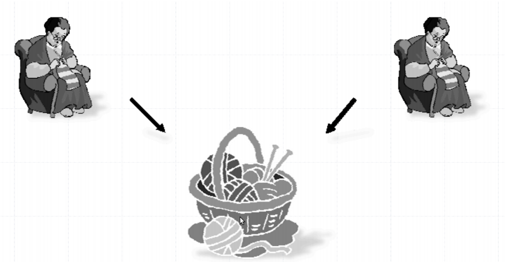


- How may the needles be shared?
- Two fundamental models of interaction
  - Shared Memory
    - Read/assign to a variable or memory location
  - Message passing
    - Send/receive

Competitive Processes

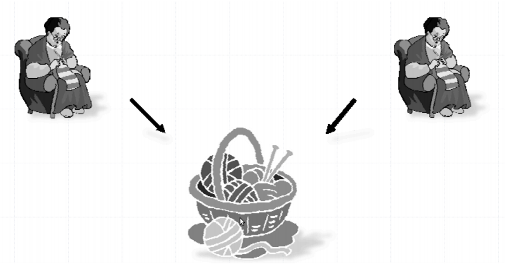


Competitive Processes



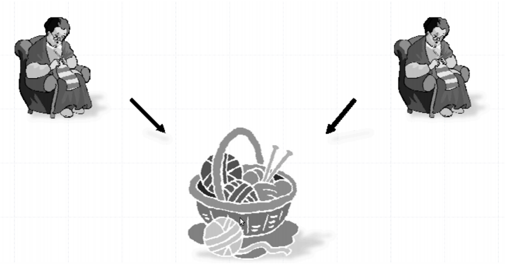
- Deadlock: each granny takes a needle and waits indefinitely until the other one has freed the one she has.

Competitive Processes



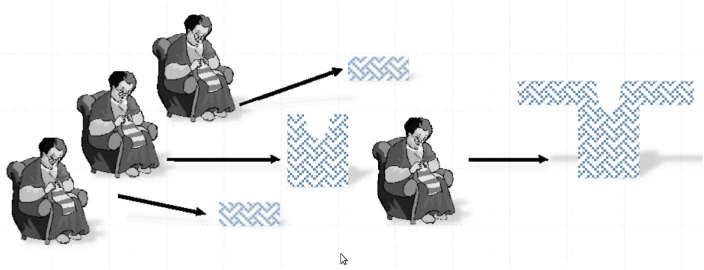
- Deadlock: each granny takes a needle and waits indefinitely until the other one has freed the one she has.
- Livelock: each granny takes a needle, sees that the other granny has the other needle and returns it (this repeats indefinitely).

Competitive Processes



- Deadlock: each granny takes a needle and waits indefinitely until the other one has freed the one she has.
- Livelock: each granny takes a needle, sees that the other granny has the other needle and returns it (this repeats indefinitely).
- Starvation: one of the grannies always takes the needles before the other one.

Cooperative[2] Processes



- Communication mechanisms are necessary for cooperation to be possible

[2] Unrelated to cooperative or non-preemptive scheduling

Modelling Program Execution

What is the value of `x` after executing this program?

```
int x=0;

// Thread P
Thread.start{
  x=1
}

// Thread Q
Thread.start{
  x=2
}
```

- Value of `x` after execution of just P?

Modelling Program Execution

What is the value of `x` after executing this program?

```
int x=0;

// Thread P
Thread.start{
  x=1
}

// Thread Q
Thread.start{
  x=2
}
```

- Value of `x` after execution of just P?
- Value of `x` after execution of just Q?

Modelling Program Execution

What is the value of `x` after executing this program?

```
int x=0;

// Thread P
Thread.start{
  x=1
}

// Thread Q
Thread.start{
  x=2
}
```

- Value of `x` after execution of just P?
- Value of `x` after execution of just Q?
- Value of `x` after execution of $P \parallel Q$?

Modelling Program Execution

What is the value of `x` after executing this program?

```
int x=0;

// Thread P
Thread.start{
  x=1
}

// Thread Q
Thread.start{
  x=2
}
```

- Value of `x` after execution of just P?
- Value of `x` after execution of just Q?
- Value of `x` after execution of P $\parallel$ Q? $\{x = 0, x = 1\}$ More than one result is possible!

Modelling Program Execution

What is the value of `x` after executing this program?

```
// Thread P
Thread.start{
  println "A";
  println "B"
}
```

```
// Thread Q
Thread.start{
  println "C";
  println "D"
}
```

- Number of interleavings is exponential in the number of instructions
- If P has m instructions and Q has n instructions, then there are
$$\binom{m+n}{m} = \frac{(m+n)!}{m!n!}$$
interleavings

Modelling Program Execution

A Transition System $\mathcal{A}$ is a tuple

$$(S, \rightarrow, I)$$

where

- S is a set of states
- $\rightarrow \subseteq S \times S$ is a transition relation
- $I \subseteq S$ set of initial states

Note:

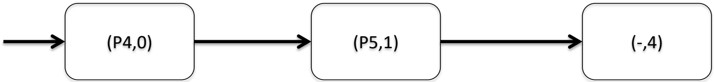
- $\mathcal{A}$ is said to be finite if S is finite
- We write $s \rightarrow s'$ for $(s, s') \in \rightarrow$.

Example 1 – Sequential Program

```
int x=0;

Thread.start { //P
  x = 1;
  x = x + 3;
}
```

- S: tuples that include
  1. the program pointer of each thread at a given point in time
  2. the value of the variables and
- $s \to t$ if executing a statement in s results in the state t

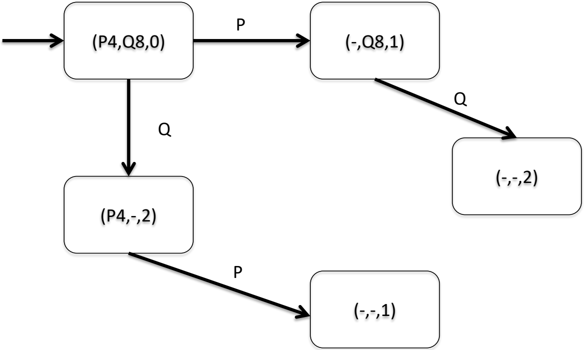


Example 3 – Concurrent Processes

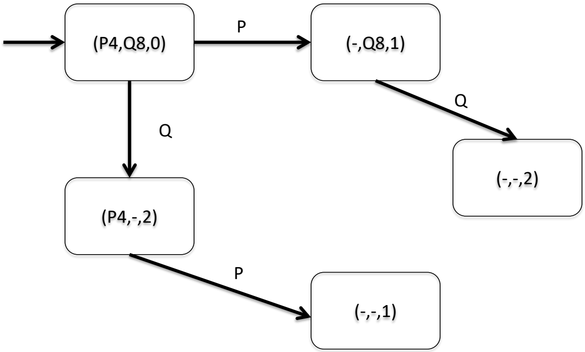
```
int x=0;

Thread.start{ // P
  x=1
}

Thread.start{ // Q
  x=2
}
```



Example 3 – Concurrent Processes



Examples of paths in textual notation:

- $(P4, Q8, 0) \rightarrow (-, Q8, 1) \rightarrow (-, -, 2)$
- $(P4, Q8, 0) \rightarrow (P4, -, 2) \rightarrow (-, -, 1)$
- $(P4, Q8, 0) \rightarrow (P4, -, 2)$

Execution Speed as a Synchronization Mechanism?

- No
- Eg. The following still has two possible results

```
int x=0;

Thread.start{ // P
  sleep(500);
  x=1
}

Thread.start{ // Q
  x=2
}
```

Summary

- We need concurrency to exploit the processor
- Concurrent programs are non-deterministic
- In this course we will study different synchronization mechanisms that will allow us to control the behavior of concurrent programs
- In particular, we will use synchronization mechanisms to ensure that our programs satisfy desirable properties to be introduced later