Concurrent Programming

CS511

Teachers

Instructor: Eduardo Bonelli

E-mail: ebonelli@stevens.edu

Office hours: MTF 2-3PM/3-4PM

CA Office hours: By appointment

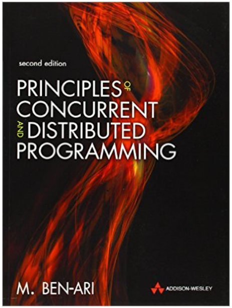
Office: North Building 318

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
- 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 by Dan Duchamp
- 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
  - Additional Makeup

Read syllabus for full details

Contents

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

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
- Get to know modern CP techniques
- Understand the fundamentals of model-checking for checking properties of concurrent systems

Concurrency

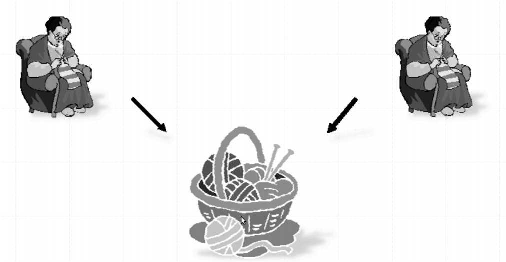
- The study of systems of interacting computer programs which share resources and run concurrently, i.e. at the same time
- Parallelism
  - Occurring physically at the same time
- Concurrency
  - Occurring logically at the same time, but could be implemented without real parallelism
- We focus on high-level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

Interaction Models

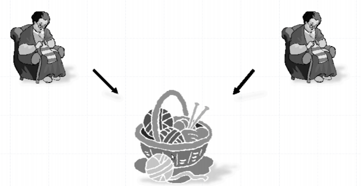
- What is process synchronization?
  - Ensure that instructions are executed in certain order
- Synchronization is irrelevant if processes do not interact with each other
- They just "do their own thing"

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



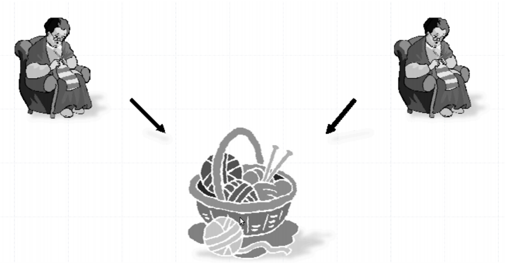
- How may the needles be shared?
- Two fundamental models of interaction
  - Shared Memory
    - Read/assign to a variable or memory location
  - Message passing
    - Send/receive

Interaction Models

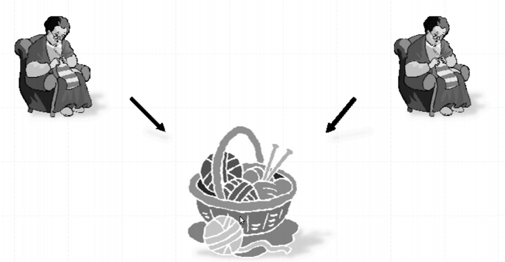
```
global int x=0;

thread P: {
  x = 1;
}

thread Q: {
  print(x);
}
```

- Two "processes" or "threads" that run concurrently
- Distinction process/thread is irrelevant to us
- Do these two thread interact? Yes!
- What is the output?
  - Depends on the scheduler
- In this course we assume (almost) nothing about the scheduler
  - Guarantees stronger synchronization properties

Competitive Processes

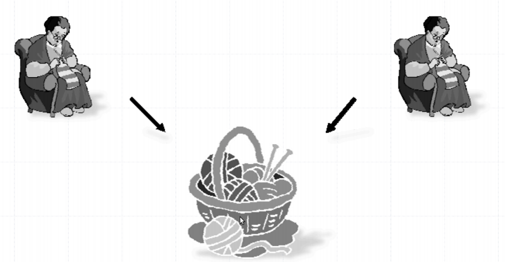


Competitive Processes



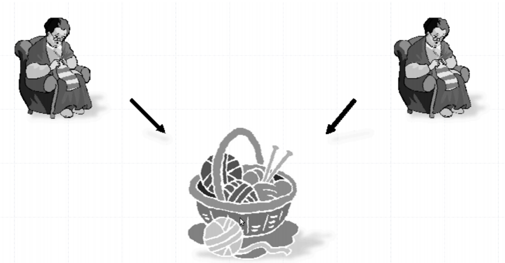
- Deadlock: each granny takes a needle and waits indefinitely until the other one has freed the one she has.

Competitive Processes



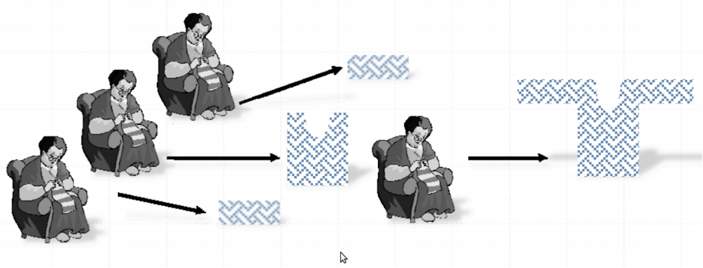
- Deadlock: each granny takes a needle and waits indefinitely until the other one has freed the one she has.
- Livelock: each granny takes a needle, sees that the other granny has the other needle and returns it (this repeats indefinitely).

Competitive Processes

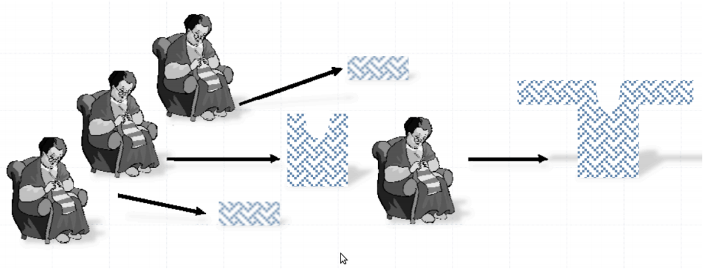


- Deadlock: each granny takes a needle and waits indefinitely until the other one has freed the one she has.
- Livelock: each granny takes a needle, sees that the other granny has the other needle and returns it (this repeats indefinitely).
- Starvation: one of the grannies always takes the needles before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

Cooperative Processes



Cooperative Processes



- Communication mechanisms are necessary for cooperation to be possible

Modelling Program Execution

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

```
global int x=0;

thread P: {
  x = 1;
  x = x + 3;
}
```

- Consider this program

```
global int x=0;

thread P:
  x = 1;

thread Q:
  x = 2;
```

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

Modelling Program Execution

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

```
global int x=0;

thread P: {
  x = 1;
  x = x + 3;
}
```

- Consider this program

```
global int x=0;

thread P:
  x = 1;

thread Q:
  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?

```
global int x=0;

thread P: {
  x = 1;
  x = x + 3;
}
```

- Consider this program

```
global int x=0;

thread P:
  x = 1;

thread Q:
  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?

```
global int x=0;

thread P: {
  x = 1;
  x = x + 3;
}
```

- Consider this program

```
global int x=0;

thread P:
  x = 1;

thread Q:
  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

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

$$(S, \to, I)$$

where

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

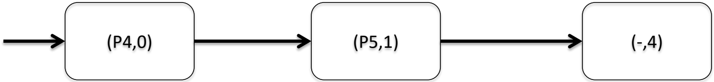
Note:

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

Example 1 – Sequential Program

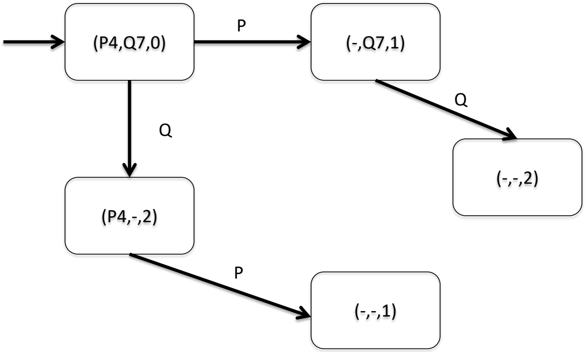
```
global int x=0;

thread 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

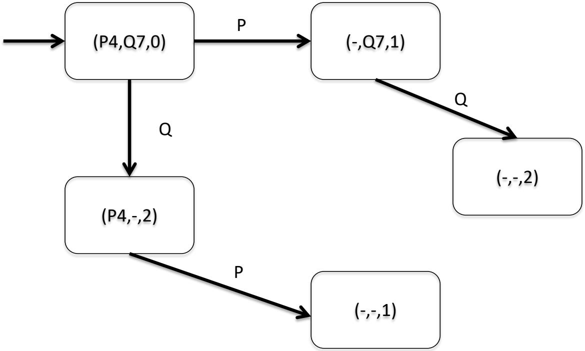


Example 3 – Concurrent Processes

```
global int x=0;

thread P:
  x = 1;

thread Q:
  x = 2;
```



Example 3 – Concurrent Processes



Examples of paths in textual notation:

- $(P4, Q7, 0) \rightarrow (-, Q7, 1) \rightarrow (-, -, 2)$
- $(P4, Q7, 0) \rightarrow (P4, -, 2) \rightarrow (-, -, 1)$
- $(P4, Q7, 0) \rightarrow (P4, -, 2)$

Execution Speed as a Synchronization Mechanism?

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

```
global int x=0;

thread P: {
  sleep(1000);
  x = 1;
}

thread 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