

Conclusions

Lecture 19

Alma Rahat

CS-210: Concurrency

20 April 2021



Outline.

- ➊ Overview of topics covered.
- ➋ Unit testing for concurrent programs.
- ➌ Online assessment information.
- ➍ Past question paper.
- ➎ Module feedback.

Understanding concurrency and developing concurrent programs.

- ① Deconstruct (requirement analysis)
 - Actors, properties and actions of interest?
- ② Model
 - Actions?
 - **Active** and **passive** processes?
 - Variables?
 - Logic and FSP syntax?
 - Identify issues.
- ③ Implement in Java
 - Identify and apply appropriate tools.

- To understand the subtle and complex problems in concurrent system.
- To describe and apply core algorithms and strategies to reliably solve these problems.
- To analyse concurrent problems.
- To recognise the link between models of concurrency and their practical application.

- Introduction

Lecture: 1

- Definitions of concurrency.
- Real-world examples.
- Benefits.
- Issues and implications.

- Introduction
- Processes and Threads

Lectures: 2, 3, 4.1

- Abstractions, generalisation and models.
- Design and implementation workflow.
- Modelling processes in FSP:
 - Process declaration.
 - Choices.
 - Indexed processes and actions.
 - Guarded actions.
 - Examples: coin tossing, switch, drink dispenser, etc.
- Threads in Java
 - Implementation techniques (Thread and *Runnable*).
 - Thread lifecycle.

- Introduction
- Processes and Threads
- **Concurrent Execution**

Lectures: 4.2, 5, 6.1

- Execution of concurrent process.
- Process scheduling simulations.
- FSP:
 - Parallel composition.
 - Process instances and labelling.
 - Action relabelling and hiding.
 - Sharing between processes: actions and resources.
 - Action synchronisation.
 - Examples: Itch-Converse, shared printer, switch, etc.

- Introduction
- Processes and Threads
- Concurrent Execution
- **Interference**

Lectures: 6.2, 7, 8

- Ornamental garden: naive model and code.
- Testing for interference.
- Fixed model and code.
- synchronized keyword and modelling locks.
- Atomic actions and variables.

- Introduction
- Processes and Threads
- Concurrent Execution
- Interference
- **Condition Synchronisation**

Lectures: 9, 10, 11.1

- Active and passive processes.
- Conditional access to shared resources:
Car Park (model and code).
- wait-notify-notifyAll in Java.
- Semaphores: allowing multiple access.
- Example: library.

- Introduction
- Processes and Threads
- Concurrent Execution
- Interference
- Condition Synchronisation
- **Deadlock**



Lectures: 11.2, 12, 13, 14.1

- Dining philosophers problem.
- Coffman conditions.
- Deadlock handling:
 - Ignorance (do nothing)
 - Prevention (by design - break Coffman conditions)
 - Avoidance (proactive: allocate to avoid)
 - Detection and recovery (reactive: break when occurs)
- Dynamic detection: Resource Allocation Graphs.

- Introduction
- Processes and Threads
- Concurrent Execution
- Interference
- Condition Synchronisation
- Deadlock
- Amdahl's Law

Lectures: 14.2, 15.1

- Original and optimised execution time.
- Derivation of gain: proportion between original to optimised.
- Applications.

- Introduction
- Processes and Threads
- Concurrent Execution
- Interference
- Condition Synchronisation
- Deadlock
- Amdahl's Law
- Properties

Lectures: 15.2, 16, 18.1

- Safety: Deadlock free and mutual exclusion.
- Liveness: Progress, fairness and a result.
- Single lane bridge (model + code).

- Introduction
- Processes and Threads
- Concurrent Execution
- Interference
- Condition Synchronisation
- Deadlock
- Amdahl's Law
- Properties
- Software Transactional Memory

Lectures: 18.2, 19

- Data parallelism (instead of task parallelism).
- Properties of database transactions.
- Workflow for STM.
- Limitations.
- Java multiverse library.
- Examples: Bank account, binary array swapper.

The central issues in testing concurrent programs is that thread communication and scheduling are non-deterministic, and precise interleaving is unknown to the programmer of a concurrent application. Number of possibilities for interleaving grows exponentially as we introduce more and more threads.

How should you tackle this?

The central issues in testing concurrent programs is that thread communication and scheduling are non-deterministic, and precise interleaving is unknown to the programmer of a concurrent application. Number of possibilities for interleaving grows exponentially as we introduce more and more threads.

How should you tackle this?

- Weed out concurrent issues by design.
- Isolate concurrent parts as much as possible, and test non-concurrent parts as usual.
- Use concurrent testing tools:
 - Stress test: <http://tempusfugitlibrary.org/>
 - Try as many configurations of interleaving as possible:
<https://github.com/google/thread-weaver> and <http://www.cs.umd.edu/projects/PL/multithreadedtc/overview.html>

Any questions?



Contribution to the module marks. 70%.

Total marks. 50.

Indicative time required for completion. designed as a 2 hours exam.

Date and time. 10:00–13:00 BST, 10 May 2021 [consult latest exam timetable for updates].

Nature of assessment. Open book; answer all questions.

Nature of questions. A number of questions with increasing difficulty (easy, intermediate to challenging).

Syllabus. All the questions in the assessment are from within what we have covered. You would require a solid understanding to achieve a high score.

What to Expect?

Notes.

- We did not cover **Haskell**, so we are **not** going to ask you any Haskell questions **Haskell**.
- Past papers will give you a clue on what to expect! We will be testing conceptually similar topics as we have done in the past. However, since this is an open book paper, the questions will be posed differently, possibly they may be more analytical, and thus they may require a little more thinking. The most representative is last year's paper.

Question paper from 2019-20 academic years.

1. Please provide the term used to describe the following concepts:

- (a) How do you refer to *a passive process that controls access to shared resources*? [2 marks]
- (b) How do you refer to *a property that states that “something good eventually happens”, i.e. progress is made despite potential concurrency issues*? [2 marks]

Easy questions to test your basic knowledge and key concepts.

2. Using Amdahl's law, answer the following.

- (a) A particular (parallelised) program spends 55% of its time in its parallel part. Compute the bound on speedup possible with 16 cores. [2 marks]
- (b) In addition to improving performance in the parallel part, if the serial part is also optimised by a factor of 2, what is the combined bound of speedup in this case? [2 marks]
- (c) You are given the choice to either perform the optimisations above or to make the serial part *five* times faster without any speedup in the parallel part. What option would you go for? [3 marks]

Easy (or intermediate) questions to test your grasp of Amdahl's law and applications.

3. (a) Consider the following scenario: A computer has two USB disk drives, and two processes that can *cut and paste* data between the drives and *erase* the origin. Now, if we use *locking* mechanism for writing and mutual exclusion, in the light of the necessary and sufficient (or Coffman) conditions, discuss how the above system may lead to deadlock. [4 marks]
- (b) Briefly propose a solution (keeping the locking mechanism) that would fix the above problem by breaking one of the Coffman conditions. You are not required to provide code. [2 marks]

Easy to intermediate question to test your basic knowledge on deadlock analysis.

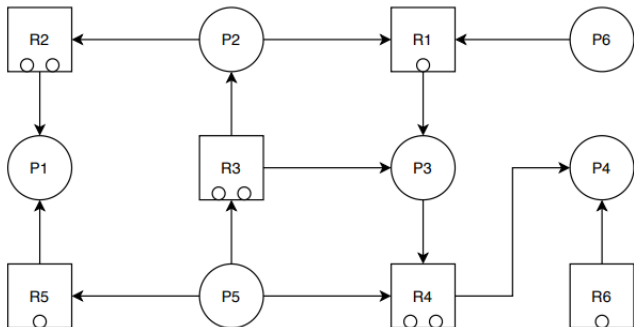
- (c) Below we have provided you with a sample code for the `UsbDrive` class. Using the `multiverse` library, modify the code below to provide a *software transactional memory (STM)* solution to the deadlock problem. You are required to provide necessary Java code here. [6 marks]

Intermediate to challenging question on your understanding of STM.

- (a) Provide Java code for an implementation of the `Switch` class that implements the `Runnable` interface adhering to the class diagram and the following specifications: [3 marks]
- The `toggle` method prints a message on the console showing the contents of `name` and `isOn` attributes, and if `isOn` is `false` then sets it to `true`, and *vice versa*.
 - The `run` method implements an infinite `while` loop within which it allows the thread to sleep for 500ms, and then call the `toggle` method.

Easy question to test your basic knowledge on implementing Java threads.

5. (a) Using the resource allocation graph, discuss whether the system below will lead to deadlock or not. [4 marks]



Easy to intermediate questions on RAGs.

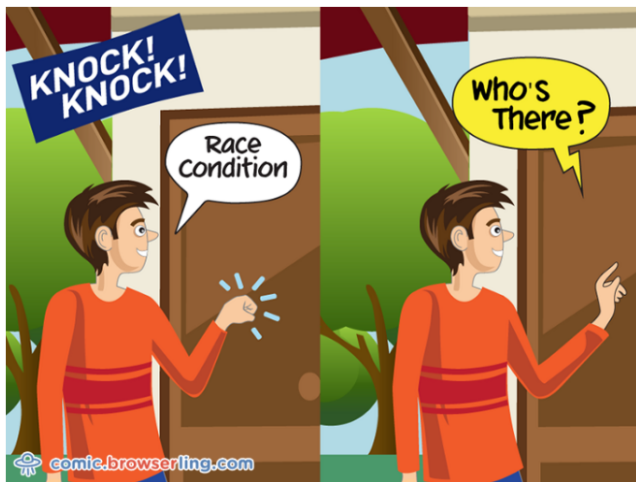
6. In a highly specialised lab, there are *five* computers. The lab **Controller** registers a student with a photographic ID, and allows the student to **enter** if there is at least one *space left*. Access is blocked if a student does not have an ID or the lab is full. A student in the lab can **leave** at any point in time. Given this scenario, answer the following questions.
- (a) Write the Finite State Process (FSP) code that models the system. [5 marks]
 - (b) Specify a safety property in FSP that ensures that there are at most *five* users in the lab at any point in time, and check the **Lab** system. [5 marks]
 - (c) Provide Java code for the monitor in this problem. [5 marks]

Intermediate and challenging questions to test your grasp of FSP. We may have similar tests for you.

What were tested?


- Introduction ✓
 - Processes and Threads ✓
 - Concurrent Execution ✓
 - Interference ✓
 - Condition Synchronisation ✓
 - Deadlock ✓
 - Amdahl's Law ✓
 - Properties ✓
 - Software Transactional Memory ✓
-
- We will find ways to test your understanding and analytical ability for various parts of what we have covered.
 - Both Java coding and FSP are important.

An open book assessment is likely to be easier than a closed book test, but more analytical. If you have a solid understanding of the topics covered, you will do well.



I sincerely hope that you learned something useful, and had some fun along the way. Thank you for being patient and engaging with us.

Could you please give us some feedback?



Dashboard

Published Courses (3)

MODULE FEEDBACK

My Module Surveys
My Module Surveys

CPD - Canvas Essentials
CPD - Canvas Essentials

Health & Safety COVID Recovery L...
Health & Safety COVID Recov...

Coming up
Nothing for the next week

Recent feedback
Nothing for now

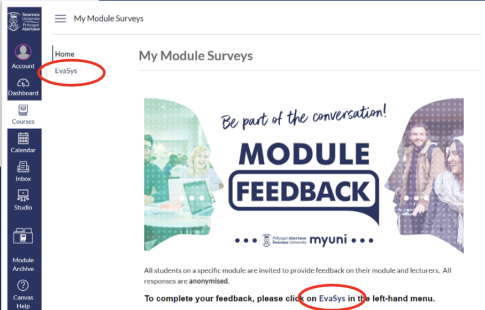
View Grades

You will see 'My Module Surveys' on your Canvas dashboard. All surveys for your modules will appear here.

Click on the course.

You can complete your survey(s) by clicking on EvaSys either in the left hand menu or in the main text.

EvaSys is the software used to create the surveys



My Module Surveys

Home
EvaSys

My Module Surveys

Be part of the conversation!

MODULE FEEDBACK

... myuni ...

All students on a specific module are invited to provide feedback on their module and lecturers. All responses are **anonymous**.

To complete your feedback, please click on **EvaSys** in the left-hand menu.

Your input is indispensable for improving the module. In your comments, please let us know what we have done well, and how we can improve.