ETH zürich

Exercise Session 14 – Concurrency

Data Structures and Algorithms

These slides are based on those of the lecture, but were adapted and extended by the teaching assistant Adel Gavranović

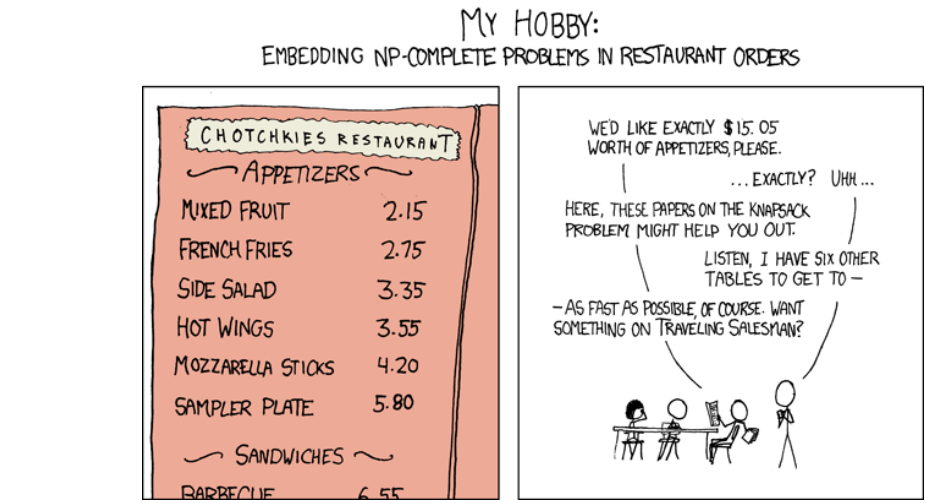
Today's Schedule

Intro
Follow-up
Feedback for **code** expert
Learning Objectives
Repetition theory (concurrent programming)
In-Class Code-Example
Information about Exam
Outro

n.ethz.ch/~agavranovic

- Exercise Session Material
- Adel's Webpage
- Mail to Adel

Comic of the Week



xkcd

1. Intro

Intro

- Some unanswered e-mail
- Some **code** expert exercises still not corrected
- Most of this I'll finish over the weekend

2. Follow-up

Follow-up from last exercise sessions

What is the “Greedy Choice Property” exactly?

- No further explanation in the lecture notes…
- “We can assemble a globally optimal solution by making locally optimal (greedy) choices.” (Chapter 16: Greedy Algorithms, Cormen et al.)

ES12::Slide16::Solution Reconstruction

- Alternative method would not yield all solutions

Exercise “Applying Maximum Flow”

- Make sure you understand this. It comes up often enough in very similar forms
- I was *very* strict with grading on this one (but all that tried got the XP)

3. Feedback for **code** expert

General things regarding **code** expert

- Did you go over and understand the “Applying Maximum Flow” exercise solution? (The garbage truck one)

Questions regarding **code** expert from your side?

4. Learning Objectives

Learning Objectives

- Understand what
  - Race Conditions
  - Bad Interleavings
  - and Data Races

  are and how to curb their ill effects

5. Repetition theory (concurrent programming)

Terminology

Race Condition
Occurs, if the observable behavior of a program depends on the sequence of events in the computer system that cannot be (directly) controlled (such as thread scheduling).

Bad Interleavings
Particular interleaving that leads to undesired results.

Data Race
Concurrent R/W or W/W access to shared memory by multiple threads, which is a bug.

Counter Problem

```
std::vector<std::thread> tv(10);
int counter = 0;

for (auto& t : tv)
  t = std::thread([&] {
    for (int i = 0; i < 100000; ++i) { counter++; } // data race
  });

for (auto& t : tv)
  t.join();

std::cout << "counter = " << counter << '\n';
```

Counter Solution 1

```
std::vector<std::thread> tv(10);
std::mutex lock;
int counter = 0;

for (auto& t : tv)
  t = std::thread([&] {
    for (int i = 0; i < 100000; ++i) {
      mutex.lock(); counter++; mutex.unlock(); // synchronized
    }
  });

for (auto& t : tv)
  t.join();

std::cout << "counter = " << counter << '\n';
```

Counter Solution 2

Note: Atomic datatypes will be introduced briefly in week 14.

```
std::vector<std::thread> tv(10);
std::atomic<int> counter = 0; // atomic integer

for (auto& t : tv)
  t = std::thread([&] {
    for (int i = 0; i < 100000; ++i) { counter++; } // atomic increment
  });

for (auto& t : tv)
  t.join();

std::cout << "counter = " << counter << '\n';
```

Quiz: What's wrong with this code?

```
void exchangeSecret(Person& a, Person& b) {
  a.getMutex()->lock();
  b.getMutex()->lock();

  Secret s = a.getSecret();
  b.setSecret(s);

  a.getMutex()->unlock();
  b.getMutex()->unlock()
}
```

Deadlock

Thread 1:
```
exchangeSecret(p1, p2);
```

Thread 2:
```
exchangeSecret(p2, p1);
```

How to resolve?

Possible Solution

```
void exchangeSecret(Person& a, Person& b) {
  // order
  std::mutex* first; std::mutex* second;
  if (a.name < b.name)
    first = a.getMutex(); second = b.getMutex();
  else
    first = b.getMutex(); second = a.getMutex();

  first->lock(); second->lock(); // lock

  Secret s = a.getSecret();
  b.setSecret(s);

  first->unlock(); second->unlock(); // unlock
}
```

Deadlocks and Races

- Not easy to spot
- Hard to debug
- Might happen only very rarely
- Testing is usually not good enough
- Reasoning about code is required

Lesson learned: Need to be *very* careful when programming with locks!

Quiz

```
void print(char c); // output c
std::mutex m1, m2;
char value;

void B() {
  m1.lock(); m2.lock();
  print(value++);
  m2.unlock(); m1.unlock();
}
void A() {
  m2.lock(); m1.lock();
  print(value++);
  m1.unlock(); m2.unlock();
}
```

```
int main() {
  value = 'A';
  print(value++);
  std::thread t1(A);
  std::thread t2(B);
  t1.join();
  t2.join();
}
```

possible output(s)?

- ABC
- A, and the program won't terminate!

Condition Variables

Condition variables

Condition variables allow a thread to wait efficiently on a specific condition.

Once the condition has changed (or could have been changed), the changing thread notifies the waiting one(s).

Condition Variables

```
class Buffer { // Recall Buffer class from the lecture
...
public:
    void put(int x) {
        guard g(m);
        buf.push(x);
        cond.notify_one();
    }
    int get() {
        guard g(m);
        cond.wait(g, [&]{return !buf.empty();});
        int x = buf.front(); buf.pop();
        return x;
    }
};
```

Condition Variables

```
class Buffer {
...
public:
    void put(int x) {
        guard g(m);
        cond.notify_one();
        buf.push(x);
    }
    int get() {
        guard g(m);
        cond.wait(g, [&]{return !buf.empty();});
        int x = buf.front(); buf.pop();
        return x;
    }
};
```

Is this correct as well?

Answer

- Here it is irrelevant where the signalling is executed.
- The signalling effect takes place, when the thread leaves the critical section, i.e. when the guard is dropped.

6. In-Class Code-Example

The Bridge ⟶ **code** expert

7. Information about Exam

Exam on 13.8.2024, 09:30h

Relevant for the exam

Material for the exam comprises

- Course content (lectures, lecture notes)
- Exercises content (coding and text exercises, exercise sessions)
- Written exam (150 min)
- Examination aids: four A4 pages (both sides printable)
- No constraints regarding content and layout
  - text,
  - images,
  - single/double page,
  - margins,
  - font size,
  - etc.

Old Exams

Exam Collection

first solve, then check the solution!

Structure

Roughly like this

Question	1	2	3	4	5	6	Total
Points	20	10	15	15	20	20	100
Score							

- around 4 Theory tasks (around 70 points):
  - $[1]$ short tasks
  - $[2]$ asymptotics and recurrence equations
  - $[3, 4]$ 2 bigger tasks
- $[5, 6]$ 2 CodeExpert tasks (around 40 points)

Personal Tips for Exam Prep

- Write a personal cheat sheet[1] to take to the exam
- Practice coding (quickly) in an exam-like setting (i.e. same keyboard layout, on **code** expert)
- Solve old exams *and then* have a look at their solution
- Save the entirety of the lecture slides in a single PDF and `cmd+F`/`ctrl+F` through it if you're looking for something
- Make use of the lecture document
- Make use of the coding examples
- *Consider* going to a PVK
- Try not to go insane…

[1] up to 8 pages on up to 4 sheets. Code snippets and drawings are allowed!

8. Outro

General Questions?

Goodbye and…

…good luck with your exams!