Concurrency:
State Models & Design Patterns

Practical Session
Week 13
Assignment 12

Discussion
A12 - Exercise 1

Answer the following questions:

a) What is a Software Architecture? What is its benefit?

A Software Architecture defines a system in terms of computational components and interactions amongst those components. It breaks the overall complexity down to several smaller blocks that are easier to handle. This improves the maintainability and ultimately the modularity.

b) What are the potential disadvantages when using layered architectures?

It increases the development overhead in small projects and it restricts the freedom of choice in terms of coding styles.
A12 - Exercise 1

Answer the following questions:

c) Provide an example in which the pattern Specialist Parallelism could be a legitimate architectural choice. Justify your answer!

Specialist Parallelism makes very much sense for different workloads that require different computation strategies, as initiated by web-servers (networking, data analysis and management, storage, ...).

d) The concepts Result Parallelism, Specialist Parallelism and Agenda Parallelism represent three ways of thinking about the problem. Can you tell on what they focus? Provide one sentence for each one of them.

Result Parallelism focuses on the shape of the finished product, Specialist P. on the makeup of the work crew and Agenda P. on the list of tasks to be performed.
A12 - Exercise 1

Answer the following questions:

e) What is a Flow Architecture? What are Blackboard Architectures?

In Flow Architecture architectural patterns (e.g. Unix pipes) synchronization is ensured by the linear processing in which information only flows in one direction from sources over filters to sinks. Blackboard Architecture architectural patterns (e.g. Producer/Consumer) perform all synchronization in a “coordination medium” where agents can exchange messages.

f) Which blackboard style should be preferred when we have multiple processors? Why?

The Agenda Parallelism should be selected, as it allows you to instantiate as many threads as CPU cores exist. The tasks thereof are typically independent and the workers identical.
A12 - Exercise 1

Answer the following questions:

g) What are Unix pipes and how do you use them?

Unix pipes (established through the “|” symbol) are bounded buffers that connect producer and consumer processes. They can be used in shell commands (e.g. `ls -f | wc -l`) to “connect” different input / output streams together to easily perform complex data manipulations.
Now we change roles: It is your turn to **send me questions of topics you are still not familiar with** for the next practical session. The best questions will be presented in front of you. I will try my best to answer all the questions you submit. For each question you ask you will retrieve a point (maximum of 3 points, no bonus this time :).

This will be part of the second half of this practical session. 😊
You have to **attend the lecture to reveal such slides.***

*:-(

*Disclaimer:
The content that has been shown on this slide is irrelevant for the exam.
Concurrency:
State Models & Design Patterns

Q&A Session

Week 13
A12 - Exercise 2

2 questions  Safety / Liveness / Fairness
5 questions  LTS / FSP
2 questions  Petri Nets
9 questions  Modeling Concurrency
12 questions Modeling Concurrency (Java)
1 question  Concurrency Architectures
3 questions  Random
2 questions  Organizational Affairs

= 36 questions in 8 categories
Safety / Liveness / Fairness
Safety / Liveness / Fairness

Q01:
How can optimistic methods livelock?
I guess, it could be some examples in ethernet protocol, such as re-try without using random sleep-interval, and each time the optimistic methods are trapped in a loop of failed->wait->retry->failed. But I'm not sure if it is the correct answer.

A: No.

livelock = circular dependencies exist, threads are busy to give priority to other(s)
optimistic methods = execute then verify integrity

-> livelock could occur when verifying integrity (broken design?)
Safety / Liveness / Fairness

Q02: Why is the busy-wait mutex protocol fair?

A: If P1 is busy-waiting, then enter1 = true, enter2 = true and turn = “P2”. Eventually P2 will go around the loop and set turn = “P1”, letting P1 proceed and forcing P2 to busy-wait. Similarly, if P2 wants to get into its CS, P1 will eventually let it do so, as long as its CS and non-CS eventually terminate.

In short: they give priority to the other (turn variable)
Q03:
Why must safety properties be deterministic to be transparent? (Safety)

A:
The safety property must not alter the behaviour of the system being checked, that is, it must be transparent.

If a process is non-deterministic, then when it is composed with an existing system, it can arbitrarily decide to take one path rather than another. This means that such a process alters the behaviour of the base system. As a consequence such a process is not transparent and cannot be a safety property.
LTS / FSP

Q04:

About computing the number of traces in an FSP: i) is the way to do this always the binomial formula from assignment nr 2? ii) Or does it depend on the given FSP? If so, can you elaborate on how to compute the number of traces, any guidelines or tips?

A:  i) No.

    ii) The formula depends always on the given FSP.

How can you calculate that number?

- induction: start with $i=0,1,2,...$

- imagination: try to see the dimensions and map them to a meaningful system

- counting: look at the FSP and count them manually (more difficult than it seems!)
LTS / FSP

Q05:
We used a lot of LTS to model simple processes. i) But is this even realistic in real world, large-scale projects? ii) Should/can those be broken down to check the project and is this even feasible because of the complexity?

A: i) Generally no, with some exceptions. It depends on the field (and the associated financial risks). Example: it was used to verify a train track control system.

ii) For worthwhile projects (see above) it has been broken down. For example:

- green signal
- no other trains
- barriers are down

→ train can pass
Q06: Until now I do not really get the approach/way of thinking to come up with complex FSP definitions. (I know this is a vague one...)

A:
1) Use the process names provided or create reasonable ones
2) Think of actions within those processes
3) Try to build meaningful cycles: connect the actions with processes
4) Rethink the possibilities you gathered

FYI: You could also draw an FSP considering the 4 rules above from which you could derive the LTS.
LTS / FSP

Q07:
(Lecture 2, Slide 10) What is the purpose of +VarAlpha and what effect does it have on the FSP?

A:

It's an alphabet extension. It is sometimes useful to extend the alphabet of a process with actions that it does not engage in and consequently actions that are not used in its definition. This may be done to prevent another process executing the action.
Petri Nets
Petri Nets

Q08:
What constrains could you put on a petri net to make it fair?
I guess, we can maybe add constrains of number of firing to each concurrent transition.

A: You do not need any constraints. Either it is fair or not. If you want to transform an unfair Petri Net into a fair one you might need to change it.

A Petri Net is fair when the firing of any transition more than a given number of times is a sufficient condition for all the transitions in the net to have fired. When the Petri Net is fair, no process in the system can be starved; that is, resources are allocated so that all tokens progress through the net.

There exists a paper: On fairness and conflicts in Petri nets

Q09:
If you implement a Petri net model, why is it a good idea to realize transitions as “thread-per-message gateways”?

A: I assume messages can be translated to tokens in the Petri Net. Moreover, messages pass a gateway and consequently tokens pass transitions. So the concepts are very similar.
Modeling Concurrency
Modeling Concurrency

Q10:
i) When does it make sense to split locks?  ii) How does it work? (Fairness)

A:
i) If your locks cause performance issues because they are too coarse grained.

ii) For each distinct shared resource you can identify you create a distinct lock. Then you only have to acquire the locks you need.

Example: ConcurrentHashMap (which uses bucket locking)
Modeling Concurrency

Q11:
What are advantages and disadvantages of encapsulating synchronization conditions as helper methods? (Fairness)

A:
Advantages:
- synchronization code separated
- easier to integrate fairness or other priorization policies

Disadvantages:
- increases complexity
Modeling Concurrency

Q12:

Reading and writing of volatile variables causes the variable to be read or written to main memory. Reading from and writing to main memory is more expensive than accessing the CPU cache. What are realistic cases when we really need to enforce visibility of variables?

A: When you concurrently check for a flag and you need an immediate response from other threads. Without volatile your check might succeed a few iterations/context switches later. This could be especially harmful if the system is under a high load -> starvation.
Modeling Concurrency

Q13:

i) Why should busy waiting be avoided and ii) how can it be replaced? iii) Are there situations where busy waiting is necessary?

A: i) Because it is inefficient.

ii) With other synchronization strategies, e.g., wait() and notify().

iii) No. You can always use a superior synchronization measures. The only advantage is that it is rather simple to implement.

```java
int i = 0;
while (i < 5000) {
    i = i + 1;
}
```

Very bad:

```java
while (x < 5) {
    Thread.sleep(100);
}
```

Still suboptimal:

Never do this except when working on (IoT) microcontrollers with a limited architecture!
Modeling Concurrency

Q14:
How can throughput be improved on a single synchronization point with high contention?

A:
Rule of thumb #1: keep the synchronized code as short as possible.

Rule of thumb #2: if you still need more throughput, try to improve the synchronization approach (different pattern / logic / split the concerns [different thread pools for different data] / ...).
Modeling Concurrency

Q15:
What primitives do you need to implement the busy-wait mutex protocol?
A: volatile (atomic reads and writes)
Modeling Concurrency

Q16:
Why can't we use an if clause instead of a while loop to test the guard condition?

A:

a) Because the guard condition does not block in most cases. Hence, you have to repeat the checks -> loop.

b) If they do block (= wait) they might wake up due to external events (interrupts) although the guard condition is still not fulfilled.
Modeling Concurrency

Q17:
When is the complexity of delegating notifications worthwhile? (Liveness Guarded Methods) (Use helper objects to maintain aspects of state and have these helpers issue the notifications.)

A:
Think large: A globally distributed system uses notifications for finished threads. Those notifications take time to transmit and acknowledge, hence you could use helpers for the notification, so that the worker thread is free for other jobs.
Modeling Concurrency

Q18:
When implementing a concurrent algorithm with the need for synchronization, how can we test the implementation to make sure that there are i) no deadlocks / ii) no race conditions can occur / iii) the algorithm is fair?

A: Testing does never guarantee (make sure) bug freeness, it is just a measure that can possibly find some bugs.

So the question is: How to make sure ...

i) no deadlocks: avoid one of the four necessary conditions
ii) no race conditions: model checking, no shared variables
iii) it is fair: model checking, keep track of requests (e.g., using a queue)
Modeling Concurrency (Java)
Modeling Concurrency (Java)

Q19:
How would you model message-passing agents in Java? (Architecture)

A:

https://www.geeksforgeeks.org/message-passing-in-java/
Q20:

On the exercises there were also some Java based questions. I had to use google for all the Java Questions. i) How much is it expected to know java-specific terms and theory and to what extent is it expected from us to ii) code in Java in the exam?

A:

i) You should know the content from the slides and the lecture. We won’t ask something that was not covered in the lecture (as it was the case for some of the exercises).

ii) You should be able to fix / inspect / implement some Java code snippets. There is no time for super complicated examples (exam is only 60 mins).
Q21: When should you use synchronize(this) rather than synchronize(someObject)? (Safety)

A: There is not much of a technical difference, but:
- if you need multiple synchronization objects use “someObject1”, “someObject2”,...
- if you only need one synchronization object use “this”

Be careful when you use synchronized and synchronize(this) in the same class, as they use the same object to lock when instantiated.
Q22: How can concurrency invalidate a class invariant? (Java Concurrency)

A: By using code that can corrupt an object’s state. That is, using no (or wrong) synchronization measures for shared variables.
Modeling Concurrency (Java)

Q23:
What could happen if any of the ExpandableArray methods were not synchronized? (Safety Pattern)

A: It would work for single threaded access, but as soon as concurrent accesses occur you would immediately run into various problems caused by different threads changing independently values in the ExpandableArray:
- Nullpointer exceptions (something has been deleted which should not have)
- Out of bounds exceptions (something should be available which is not)
- ...
Modeling Concurrency (Java)

Q24:

What are the dangers in letting the (Java) scheduler choose which writer may enter a critical section? (Fairness)

A: You never know which one will get through. If you do not enforce fairness (some Java synchronization methods have a boolean flag for that) you could end up having a starving writer.
Q25:  
For ThreadLocal if we do not clean up, any references it holds to classes loaded will remain in the permanent heap and will never get garbage collected. Why does it happen?

A: Because it is (intended) to be used with “static”. This keyword makes the variable permanently available.

More general from StackOverflow: Static variables cannot be elected for garbage collection while the class is loaded. They can be collected when the respective class loader (that was responsible for loading this class) is itself collected for garbage. [https://stackoverflow.com/questions/453023/are-static-fields-open-for-garbage-collection](https://stackoverflow.com/questions/453023/are-static-fields-open-for-garbage-collection)
Q26: What is the purpose of yield() in normal Java programming situations? In the lectures, yield was mostly used to provoke race conditions.

A: It is used to give priority to another thread. For example, in a busy wait.

That is also the reason why it was used for that: the other thread is supposed to create a race condition.
Modeling Concurrency (Java)

Q27:
Why decided the developers of Java to use a Monitor based design and not message passing? What were the determining reasons.

A:
From an implementer’s perspective: A monitor based design is much simpler to implement and faster during execution (less overhead). A message passing approach always needs a queue for the messages, and then there is the risk of losing messages if you use a bounded buffer which would have a cap on the memory demand.

From a user’s perspective: You need more parameters (e.g., the relevant queue) that would complicate its use compared to the use of monitors.
Modeling Concurrency (Java)

Q28:

About Lecture 02 about Java Concurrency: In the notes after slide 15 about synchronized blocks in Java, the following is written: Aside: actually, an arbitrary object be used as the lock, not necessarily one of the resources accessed. So does this explain the possibility of using dedicated conditional lock objects, such as in our Java assignment with the rectangle creator? In other words, can we create just an empty object, without any other references, and just use that as a lock in a synchronized block, even though the resource which we lock on does not do anything in the program itself?

A: Yes! That’s probably the only real purpose of using

```
Object lock = new Object();
```
Q29:
Why exactly are wait and notify only available when inside the synchronized method?

A:
Because they rely on the monitor of the Java class:
- wait() gives away the monitor (thus you must have it acquired before) and it also could retrieve it back when another thread calls notify
- notify() gives away the monitor (thus you must have it acquired before) without waiting (hence after you finished your stuff)
Modeling Concurrency (Java)

Q30:

When inside a synchronized method, no other method should be able to access it. But if we hit the wait inside this synchronized method, others are allowed to enter again. So does wait disable the lock mechanism of synchronized and is it re-enabled once notify/notifyAll is called?

```java
public synchronized foo() {
    wait();
}
```

A: Yes, just revise what happens:

1) You acquire the monitor and enter the synchronized block
2) You wait() and give the monitor away.
3) Another thread can grab the monitor and execute the foo() method
4) If the first thread receives the notify (released wait) it will be again exclusively active within the foo method.
Concurrency Architectures
Concurrency Architectures

Q31:
Differences between software architecture and design patterns. I guess, They are 2 different definitions. However, there are some intersection sets between them. Design patterns are something more general, it can be some small patterns focus on class levels (singleton), or MVC, pipe-filter as same as architecture. Software architecture but may be also some inter-project pattern, such as SOA or micro-services architecture, which but not belongs to any design-patterns. I’m not sure, if i thought it in a correct way.

A: You are close: design patterns are generally applied within a specific environment (intra-environment: programming language, framework, network,...), whereas software architectures rather describe patterns that apply to a combination of different environments (inter-environment: client-server, full stack web development, database integration, ...).
Random
Q32:
What will happen if 2 concurrent Fibonacci computations started. I guess, it should work properly and even better, due to caching.

A:
Will it work properly? Yes. (A08E02: no inter-process communication, no harm)

Will it run faster? Probably not. See discussion here:
https://stackoverflow.com/questions/4802565/multiple-threads-and-cpu-cache
Q33:
When to use Petri nets and when LTS/FSP? What are the strengths and weaknesses of each?

A: LTS/FSP:
+ better tool support
+ better flexibility
- not very intuitive, syntax
- very complex implementation

Petri Nets:
+ easy to sketch
+ live testing
+ less complex implementation
- less scalable
- limited tool support

-> rather for real life examples, system validation
-> to learn and understand the concepts
-> as add-in for existing software products (due to easier implementation)
Q34:
i) What is exponential backoff and ii) how can this principle be used in synchronization?

A:

i) Retry with an exponential back-off time. Used in the (wired) Ethernet protocol for shared medium access.

ii) Try to enter critical section, if impossible, wait for exp(x) seconds, then retry. If again impossible, wait for exp(x+1) seconds, then retry. If again impossible, wait for exp(x+2) seconds, then retry, ...
Organizational Affairs
Organizational Affairs

Q35:
Which materials are important for the exam?

A:
Lecture slides, exercises, Git code samples, Petri net examples, this slides.
Exercises about slides of chapter 13 (actors) or the lab won't appear in the exam.
Organizational Affairs

Q36:
What else? I should ...

A:
... know important terms and be able to explain them (nec. deadlock conditions, nested monitor, ...)
... be able to write LTS code for a given FSP diagram
... understand provided LTS code and adapt such code to other problems
... be able to find problems in Java code and to propose (valid) solutions
... how Petri Nets work, their construction/use, understanding of the related properties

... really review all the content in the slides and the exercise sessions
... still more questions?

Send us an email!
Next Time: Exam
60 minutes / CLOSED BOOK

Lecture room (003), 18-Dec-2019, 10:30am until 11:30am

1. Arrive on time

2. Don't forget your student ID

3. Don't forget a blue or black ball pen

4. Not allowed: pencils, internet, notebook, books, any printouts, additional blank pages, pocket calculators, mobiles, smart anything, ...
Thank you VERY MUCH for your participation!
Good Luck!