https://docs.oracle.com/javase/specs/jls/se8/html/jls-17.html

2020 Test

1 (6p) Why the details of accessing variables are often important to know in concurrent programming? What are synchronized and volatile variables in Scala and how they are used in concurrent programming?

- The details of accessing variables are important to know in concurrent programming

2 (10p) Answer *shortly* with clear definitions and descriptions. (Max. two points per subquestion.)

a) What is a race condition?
b) What is a data race?
c) What is livelock?
d) What is multitasking?
e) What is Dekker's algorithm?

a) A race condition occurs when two or more threads can access shared data and they try to change the shared data at the same time. Therefore, the result of the change in data is dependent on the thread scheduling algorithm or in other words, the threads are "racing" to access/change the data

b) If there are two unsynchronized operations performed by two different threads on a variable such that one of them is a write and which are not ordered by happens-before relationship, we have a data race

c) Livelock occurs when two or more processes continually repeat the same interaction in response to changes in the other processes without doing any useful work. These processes are not in the waiting state, and they are running concurrently

d) Multitasking is the concurrent execution of multiple tasks (also known as processes) over a certain period of time. In multitasking the processes are interleaved by giving the processor to each of them in turns

e) Dekker's algorithm is the first correct solution to the critical section problem
– Guarantees mutual exclusion, freedom from deadlock and freedom from starvation for two processes
– Freedom from starvation is irrespective of scheduling policy, as long as the weak fairness assumption holds
– Instead of a naive turn taking, the algorithm features an alternating priority which will guarantee that one process may proceed to the critical section, if both are seeking to do so. After exiting the critical section, the priority is given to the other process

3 (6p) What are monitors, what do they consist of, how are they initialized, and how they are used? Describe the basic operations of Scala monitors.

- What are monitors: Monitors are data abstraction construct that monitors how threads access some resources (shared resources). It is a synchronization construct that allows threads to have both mutual exclusion and the ability to wait (block) for a certain condition to become false
- What do monitors consist of: A monitor consist of one or more methods (operations of the data abstraction), an initialization sequence and local data. A monitor consists of a mutex (lock) object and condition variables. A condition variable essentially is a container of threads that are waiting for a certain condition
How monitors are initialized:
How monitors are used:
Basic operations of Scala monitors:
- wait() method: When executing the wait() method of an object, the thread
1. adds itself to the object's wait set, and
2. releases the object's lock (fully, even if multiply acquired), ensuring all variable values written will be visible to the thread that acquires the lock.
3. If the thread holds multiple locks, only the one for which wait() is called is resumed.

4 (6p) Explain Scala (*i.e. Java/JVM*) memory model. What is the happens-before relation and how is it used to define the JVM memory model? What kind of behaviours are allowed by the JVM memory model that are not allowed by sequential consistency? How does the JVM memory model compare to the C/C++ memory model?

- Java memory model: A memory model describes, given a program and an execution trace of that program, whether the execution trace is a legal execution of the program. The Java programming language memory model works by examining each read in an execution trace and checking that the write observed by that read is valid according to certain rules.

- What is the happens-before relation: Two actions can be ordered by a happens-before relationship. If one action happens-before another, then the first is visible to and ordered before the second.

- How happens-before relation is used to define the JVM memory model:
The happens-before relation defines when data races take place.
A set of synchronization edges, S, is sufficient if it is the minimal set such that the transitive closure of S with the program order determines all of the happens-before edges in the execution. This set is unique.
It follows from the above definitions that:
An unlock on a monitor happens-before every subsequent lock on that monitor.
A write to a volatile field (§8.3.1.4) happens-before every subsequent read of that field.

A call to start() on a thread happens-before any actions in the started thread.
All actions in a thread happen-before any other thread successfully returns from a join() on that thread.
The default initialization of any object happens-before any other actions (other than default-writes) of a program.
When a program contains two conflicting accesses (§17.4.1) that are not ordered by a happens-before relationship, it is said to contain a data race.

-Behaviors allowed by JVM memory model but not allowed by sequential consistency:

- How does JVM memory model compare to C/C++ memory model

5 (6p) Write an essay (max 50 lines) on correctness of concurrent programs.

2019 Test

1 (6p) **What are busy-wait loops? Considering concurrent programming, sometimes they are considered very useful, but sometimes a really bad practice. Why? Give examples.**

- busy-waiting, busy-looping or spinning is a technique in which a process repeatedly checks to see if a condition is true, such as whether keyboard input or a lock is available

Advantages of busy-waiting:

The execution flow is usually easier to comprehend and thus less error prone.
Timing can be determined more accurately in some cases
Disadvantages:

No other code (except perhaps other interrupt routines) can be executed.
CPU-time is wasted: If no other work must be processed it is still more efficient to set some power saving-state and let a timer interrupt wake it up in time.
- Busy wait loops for process synchronization and communication are considered bad practice because (1) system failures may occur due to race conditions and (2) system resources are wasted by busy looping

A disadvantage of busy-waiting in embedded devices is the increased power consumption. In a busy wait, the processor is running full-blast, consuming power with no result. Most low power processors have the ability to put the processor to sleep while waiting for a timer interrupt, reducing power consumption dramatically. Lower power consumption = longer battery life

2 (10p) Answer *shortly* with clear definitions and descriptions. (Max. two points per subquestion.)

a) What is a race condition?
b) What is starvation?
c) What is Limited Critical Reference?
d) What is thread-safe code?
e) What is Dekker's algorithm?

a) race condition: occurs when two or more threads can access shared data and they try to change the shared data at the same time. Therefore, the result of the change in data is dependent on the thread scheduling algorithm or in other words, the threads are "racing" to access/change the data
b) starvation
c) limited critical reference:
• Reference to variable v is critical reference, if …
– Assigned value in P and read in Q
• Read directly or in a statement
• Program satisfies limited critical reference (LCR)
P v Q
kriittinen viite
• Program satisfies limited-critical-reference (LCR)
– Each statement has at most one critical reference
– Easier to analyze than without this property
– Each program is easy to transform into similar program with LCR
d) thread-safe code: Thread-safe code means that the code can be concurrently executed. When a thread is already working on an object and preventing another thread on working on the same object, this process is called Thread-Safety

e) Dekker’s algorithm: Dekker’s algorithm is the first known algorithm that solves the mutual exclusion problem in concurrent programming by Th. J. Dekker. Dekker's algorithm is used in process queuing, and allows two different threads to share the same single-use resource without conflict by using shared memory for communication.
Dekker’s algorithm will allow only a single process to use a resource if two processes are trying to use it at the same time. The highlight of the algorithm is how it solves this problem. It succeeds in preventing the conflict by enforcing mutual exclusion, meaning that only one process may use the resource at a time and will wait if another process is using it. This is achieved with the use of two "flags" and a "token". The flags indicate whether a process wants to enter the critical section (CS) or not; a value of 1 means TRUE that the process wants to enter the CS, while 0, or FALSE, means the opposite. The token, which can also have a value of 1 or 0, indicates priority when both processes have their flags set to TRUE.
This algorithm can successfully enforce mutual exclusion but will constantly test whether the critical section is available and therefore wastes significant processor time.

3 (6p) The concept of sequential correctness is not sufficient for concurrent programs. Explain how the correctness of concurrent programs can be defined and checked.

4 (6p) Write a short essay (max 30 lines) on the Akka Actor programming framework. What is the programming framework like to the programmer, what are its main uses, and what its benefits and drawbacks compared to other concurrent programming models.

5 (6p) Write an essay (max 50 lines) on low-level concurrent programming.

2017 Test

Assignment 1 Please write a short essay (1/2 page max) on semaphores. What are semaphores, what do they consist of, how are they initialized, and what are the operations you can do on a semaphore? What are the guarantees given to the user of a semaphore, when they use these operations, i.e., how does a semaphore work and what does it guarantee? What are the differences between binary and general semaphores? How can semaphores be used to implement a critical section? (4p)

Assignment 2 Consider the Java Monitors notification mechanisms and answer using 2-4 sentences for each item:

a) What happens when a thread executes a `wait()` method on an object? (1p)

b) What are the three main reasons under which a Java `waiting` thread will continue execution? (1p)

c) What is the difference between `notify()` and `notifyAll()` methods? (1p)

d) What are spurious wakeups and how to defend against them in Java? (1p)

Assignment 3 Explain the following notions and terms using 2-4 sentences for each item:

a) Amdahl's law (1p)

b) Monitor (1p)

c) LCR (1p)

d) Mutex (1p)

e) Apache Spark (1p)

f) Livelock (1p)

g) C/C++ Low level atomics (1p)

h) Volatile variable (1p)

a) Amdahl's law: Amdahl's Law states that in parallelization, if P is the proportion of a system or program that can be made parallel, and 1-P is the proportion that remains serial, then the maximum speedup S(N) that can be achieved using N processors is:

$$S(N)=1/((1-P)+(P/N))$$

As N grows the speedup tends to 1/(1-P).
b) Monitor:
c) Limited critical reference
d) Mutual exclusion (mutex):
e) Apache Spark:
f) Livelock:
g) C/C++ low level atomics:
h) volatile variable

Assignment 4 Please write a short essay (1/2 page max) on race conditions. How are race conditions in Java defined? What kind problems can race conditions cause on concurrent programs? What kinds of tools and algorithms can be used to detect race conditions in concurrent programs? What kind of guarantees does Java memory model give to race condition free programs? How does the existence of race conditions in a Java program change the optimizations a compiler can or can not make, or is there any difference? (4p)

Assignment 5 Please write a short essay (1/2 page max) explaining how the x86-TSO abstract machine memory model works. Why is such a model needed? Please give an example of a program behavior allowed by x86-TSO memory model which is not allowed by sequential consistency. (4p)

2016 Test

Assignment 1 Please write a short essay (1/2 page max) explaining what are weak memory models? Why are they being used as the semantics of programming languages instead of sequential consistency? Give some examples of weak memory models from hardware level, programming language level, and from databases. Discuss what kind of challenges weak memory models impose to compiler writers, and to programmers in general. (4p)

Assignment 2 Consider the Java Monitors notification mechanisms and answer using 2-4 sentences for each item:

a) What happens when a thread executes a `wait()` method on an object? (1p)

b) What are the three main reasons under which a Java `waiting` thread will continue execution? (1p)

c) What is the difference between `notify()` and `notifyAll()` methods? (1p)

d) What are spurious wakeups and how to defend against them in Java? (1p)

Assignment 3 Explain the following notions and terms using 2-4 sentences for each item:

a) RDD (1p)

b) Safety property (1p)

c) Liveness property (1p)

d) X86-TSO (1p)

e) Monitor (1p)

f) Critical section (1p)

g) Deadlock (1p)

h) Race condition (1p)

a) resilient distributed dataset (RDD):
b) safety property:
c) liveness property:
d) X86-TSO
e) monitor
f) critical section
g) deadlock
h) race condition

Assignment 4 Please write a short essay (1/2 page max) of the Akka Actor programming framework. What is the programming framework like to the programmer, what are its main uses, and what its benefits and drawbacks compared to other programming models for concurrent programs. (4p)

Assignment 5 Consider the Scala Parallel Collections programming framework, and answer using 2-4 sentences for each item:

a) What is Work Stealing, and how is it related to Scala Parallel collections implementation? (1p)

b) What are the two core abstractions used by Scala Parallel collections implementation to allow for parallelization of computation and obtaining a joint result? (1p)

c) What are the requirements imposed on the function applied by the Scala Parallel Collections `reduce` operation? (1p)

d) How do the requirements for the Apache Spark function applied by parallel `reduce` differ from the requirements imposed by Scala parallel collections? How does this affect portability between these frameworks? (1p)

a) work stealing

What are distributed systems:
A distributed system (distributed computing) is a system with various components that operate independently on different machines that communicate and coordinate actions with each other. This system appears as a single working system to the end-user.. These machines have a shared state, operate concurrently and can fail independently without affecting the whole system's process.

How distributed systems are typically operated:
In a distributed system, the components use communication channels that send messages to each other instead of sharing a memory. Thus, there is no global state nor common memory, and the system state is updated by exchanging messages.

What are some challenges of distributed systems?
Distributed system has several challenges in their operations and design. Some of them are:
• Increased failure chances: the more components added to a distributed system, the more likely failure can take place. If a system is not carefully designed and a single node/component crashes, the entire system can be at grave risk. Although on theory, a criteria of distributed systems requires the system to be fault tolerance (that is, a failure of single component does not render the whole system failing), fault tolerance is by no means available and must be carefully crafted.
• Synchronization difficulties: Distributed systems operate without a global state. In other words, distributed system requires that processes are appropriately synchronized to avoid transmission delays that result in errors and data corruption.
• Scalability: How the system is scaled or refactored is extremely complex and requires careful planning of the design from scratch. Moreover, adding more components to the system does not necessarily increase its performance.

• Complex security problems: Since distributed system works via a communication channel, it is susceptible to various attacks like sniffing, injection and middle-man attack. Attack on a single component will put the entire system at security risks
• Complex design: Distributed systems are noticeably difficult to design and manage and than normal system of components that has a global state

What are distributed systems:
A distributed system (distributed computing) is a system with various components that operate independently on different machines that communicate and coordinate actions with each other. This system appears as a single working system to the end-user.. These machines have a shared state, operate concurrently and can fail independently without affecting the whole system's process.

How distributed systems are typically operated:
In a distributed system, the components use communication channels that send messages to each other instead of sharing a memory. Thus, there is no global state nor common memory, and the system state is updated by exchanging messages. Distributed systems have evolved over time, but today's most common implementations are largely designed to operate via the internet and, more specifically, the cloud. A distributed system begins with a task, such as rendering a video to create a finished product ready for release. The web application, or distributed applications, managing this task — like a video editor on a client computer — splits the job into pieces. In this simple example, the algorithm that gives one frame of the video to each of a dozen different computers (or nodes) to complete the rendering. Once the frame is complete, the managing application gives the node a new frame to work on. This process continues until the video is finished and all the pieces are put back together. A system like this doesn't have to stop at just 12 nodes — the job may be distributed among hundreds or even thousands of nodes, turning a task that might have taken days for a single computer to complete into one that is finished in a matter of minutes.

What are some challenges of distributed systems?
Distributed system has several challenges in their operations and design. Some of them are:
• Increased failure chances: the more components added to a distributed system, the more likely failure can take place. If a system is not carefully designed and a single node/component crashes, the entire system can be at grave risk.
• Synchronization difficulties: Distributed systems operate without a global state. In other words, distributed system requires that processes are appropriately synchronized to avoid transmission delays that result in errors and data corruption.

• Complex security problems: Since distributed system works via a communication channel, it is susceptible to various attacks like sniffing, injection and middle-man attack. Attack on a single component will put the entire system at security risks
• Complex design: Distributed systems are noticeably difficult to design and manage and than normal system of components that has a global state

How such problems can be resolved:
• Increased failure chances: Although on theory, a criteria of distributed systems requires the system to be fault tolerance (that is, a failure of single component does not render the whole system failing), fault tolerance is by no means available and must be carefully crafted. Each component designs should be ensured with concurrent correctness
• Synchronization difficulties: Constantly checking for the communication channels to be updated frequently and make sure they are not prone to data corruption
• Complex security problems: Make sure that each component of the distributed system are not easily prone to attack. Security of the entire system should be tested by professional technicians
• Complex design: Careful planning and documentations of the system is a must to keep track of the design.

What are memory models?
In concurrent programming, a memory model is a set of rules and protocols that dictate how the working threads interact with each other via the common shared memory and how they access and use the shared data. There are various memory models like JVM/Scala, x86-TSO, Power and ARM Memory Model

Why memory models are useful in concurrent programming?
Memory models are useful for concurrency because it is the framework that provides us knowledge on the understanding of the concurrent correctness of shared-memory. The memory model specifies how and when different threads can see values written to shared variables by other threads, and how to synchronize access to shared variables when necessary

How memory models relate to program code and program execution?
In program code, the memory model defines the concurrency semantics of shared-memory systems. For example, for reading code, the memory model regulates which possible values that read operation is allowed to return for any orderings of write operations performed by other processes within a concurrent program. Therefore, the memory model only allows a certain set of outputs of a program's reading and writing code and thus the memory model defines the possible outcomes of a concurrent programs.
In program execution, the memory model generally permits executions whose outcome is unpredictable from the order of read and write operations in the concurrent program. So without

a clear memory model, we can not know definitely the result of a program execution, even if we are aware of the program's implementations. Moreover, the memory model dictates which re-orderings are allowed and constrains the program execution to run only such re-orderings

What are memory models?
In concurrent programming, a memory model is a set of rules and protocols that dictate how the working threads interact with each other via the common shared memory and how they access and use the shared data. There are various memory models like JVM/Scala, x86-TSO, Power and ARM Memory Model

Why are memory models useful in concurrent programming?
Memory models are useful for concurrency because it is the framework that provides us knowledge on the understanding of the concurrent correctness of shared-memory. The memory model specifies how and when different threads can see values written to shared variables by other threads, and how to synchronize access to shared variables when necessary

Important properties of memory models in concurrent programming?
• Consistency model: there are several models, most common are strict consistency, sequential consistency, processor consistency and weak consistency. Knowing the concurrency model helps us understand the memory model better
• Data race definition of the memory model: in JMM, a data race occurs when a variable is read by more than one thread, and written by at least one thread, but the reads and
writes are not ordered by happens before
• happens-before relationship: a set of regulations on the relationship of two working threads, where one thread must definitely work/start before another
• Locking mechanism: in the TSO-x86 memory model, a lock prefix specifies as follows:
- Read operations are only allowed when the global lock is not held by another hardware thread
- Write operations appending to the store buffer FIFO are always allowed

How memory models relate to program code and program execution?
In program code, the memory model defines the concurrency semantics of shared-memory systems. For reading operation, the memory model regulates which possible values that the read is allowed to return for any orderings of write operations performed by other processes within a concurrent program. Therefore, the memory model only allows a certain set of outputs of a program's reading and writing code.
In program execution, the memory model generally permits executions whose outcome is unpredictable from the order of read and write operations in the concurrent program. Without a memory model, it’s impossible to know the result of a program execution, even if the program's

implementations are available. Moreover, the memory model dictates which re-orderings are allowed and constrains the program execution to run only such re-orderings