03/09/2022 – 03/11/2022

**Concurrency vs. Parallel**

Concurrency: multiple tasks run simultaneously. You can’t tell which task comes first and which task comes after. You can’t tell the order in general.

Parallel: multiple tasks **physically** run simultaneously. This is an implementation level concept. In real time, there are at least two executors.

If there is concurrency, there does not have to have parallel.

If there is parallel, there must have concurrency.

ConcurrentModificationException example:

for (int i : myList) {

myList.remove(i);

}

**Multi-process vs. multi-thread**

Process: an independent execution of instructions with independent memory space, stack, heap, and OS resource. Each process sees a complete memory space (pretend to be the only task of a system). Different processes communicate through inter process communication (explicit IPC).

Thread: an independent execution of instructions with shared memory space. Each thread has its private stack, program counter, and register states. Thread in the same process has shared: heap, static memory segment, OS resource. Communication performed through shared memory read/writes.

Difference: **independent memory space**. If has, process; otherwise, thread.

Multi-process: higher communication overhead, better resource isolation (fault tolerance), higher creation/destroy overhead.

Multi-thread: lower communication overhead, worse resource isolation (fault tolerance), lower creation/destroy overhead.

Multi-thread example:

Public static void main () {

Thread t = new Thread () {

@Override

Public void run () {

System.out.println(“1”);

}

};

t.start();

System.out.println(“2”);

t.join();

System.out.println(“3”);

}

Main thread --- new thread --- print “2” --- join --- print “3” --- exit

| |

---------- print “1” -----

Another example:

Public static void main () {

Thread t = new Thread () {

@Override

Public void run () {

System.out.println(“1”);

}

};

t.start();

System.out.println(“2”);

}

Main thread --- new thread --- print “2” --- exit

| |

---------- print “1” -----

**When will JVM exit? When there are no alive non-daemon threads.**

To make t a daemon thread: t.setDaemon(true);

PS: for JVM, GC is a daemon thread.

**Ways of creating threads and making them run**

1. extends Thread
2. implements Runnable
3. implements Callable

**Methods of thread**

Static methods: sleep(1000), yield() // we can see yield() as sleep(0)

**Synchronization and Race**

Data race: If two “conflicting operations” are in different threads and are not properly synchronized (concurrent), they will introduce data races. In general, two operations conflict with each other if they operate on the same memory location, and at least one of them is a write. Races are mostly treated as bugs in Java programs.

Three factors of data race:

1. more than one operation work on the same memory location
2. at least one operation is a write
3. at least two of those operations are concurrent

**Locks**

**Deadlock**

Condition to form a deadlock:

1. mutual exclusion: at least one resource must be held in a non-shareable mode. Only one process can use the resource at any given instant of time.
2. hold and wait or resource holding: a process is currently holding at least one resource and requesting additional resources which are being held by other process.
3. no preemption: a resource can be released only voluntarily by the process holding.
4. circular wait

**Livelock**

**Condition Synchronization**

Producer consumer problem:

* Consumer: if the queue is empty, wait for it to be not empty, then poll one element from it.
* Producer: if the queue is full, wait for it to be not full, then offer one element to it.

Transactions

A series of actions that follows ACID principles

Sql injection

Compare statement preparestatement and