Saint Petersburg Electrotechnical University LETI

Faculty of Computer Science and Technology, Department of Computer Science and Engineering

Master of Computer Science and Knowledge discovery

Course name: Parallel Computing

Student Name: Sayed Moeid Heidari

Professor : Natalya V. Razmochaeva

Concurrency

Concurrency is the ability to run several programs or several parts of a program in parallel. Concurrency enables a program to achieve high performance and throughput by utilizing the untapped capabilities of underlying operating system and machine hardware. e.g. modern computers have several CPUs or several cores within one CPU, program can utilize all cores for some part of processing; thus completing tasks much before in time in comparison to sequential processing.
Java is a multi-threaded programming language which means we can develop multi-threaded programs using Java. A multi-threaded program contains two or more parts that can run concurrently and each part can handle a different task at the same time making optimal use of the available resources specially when your computer has multiple CPUs. Multi-threading enables you to write in a way where multiple activities can proceed concurrently in the same program.

Processes and Threads

There are two principle execution units in concurrency programming which are Processes and Threads.Although processes are important in java Concurrent programming, we do concurrent programming by threads in java programming language usually.

Processes

In concurrent programming each process has its own specific execution environment which contains private and public resources which are the resources process uses to execute the program.
There are some main properties of a process which are:

- A program in execution is often referred to as a process. A thread is a subset(part) of the process.
- A process consists of multiple threads. A thread is a smallest part of the process that can execute concurrently with other parts(threads) of the process.
- A process is sometimes referred to as a task. A thread is often referred to as a lightweight process.
- A process has its own address space. A thread uses the process's address space and shares it with the other threads of that process.

- A thread can communicate with other threads (of the same process) directly by using methods like wait(), notify(), notifyAll(). A process can communicate with other processes by using inter-process communication.
- New threads are easily created. However the creation of new processes requires duplication of the parent process.
- Threads have control over the other threads of the same process. A process does not have control over the sibling process, it has control over its child processes only.

Threads

Threads are sometimes called lightweight processes. Both process and thread have their own execution environment but threads need less resources than processes to be executed. Each process contains a number of threads so by definition threads exist in processes. Every application has at least one thread.Each program contains some other system threads which their responsibilities are memory management and signal handling.

Thread Objects

In java programming a thread can be created using Thread Object and we can use threads by two basic strategies to make concurrent programs :

- Create an instance of Thread object to do the tasks asynchronously
- Pass a task to an exist executer

Defining and Starting a Thread

To create and start an instantiation of thread object we should provide a code which we want to be executed by the thread. Generally speaking there are two ways to do that:

- Runnable object
- Thread subclass

Pausing Execution with Sleep

There is a possibility to stop or suspend a thread in multithreading in JAVA programming language. The Sleep function is able to do such an operation in concurrent programming with a parameter for the sleeping time (In Milli seconds usually).

Interrupts

An interruption is an indication to stop or suspend the currently running thread to stop what it is doing now and do something else instead. The interrupt() method of thread class is used to interrupt the thread. If any thread is in a sleeping or waiting state (i.e. sleep() or wait() is invoked) then using the interrupt() method, we can interrupt the thread execution by throwing InterruptedException.For the interrupt mechanism to work correctly, the interrupted thread must support its own interruption.

Joins

Method join() makes the thread wait until the termination of other running threads. It pauses the current threads execution until the termination of t's thread termination which is running now.

Synchronization

The variety of threads can work together and execute different tasks at the same time seperately as there is a necessity of communication among these threads. We need to provide shared resources and variables which can cause diverse problems: thread interference and memory consistency errors. Synchronization is a tool to prevent these kinds of problems which can cause a number of threads to communicate in a shared environment.

Thread Interference

When multiple threads share the same memory, there is a chance that two or more different threads performing different operations on the same data interleave with each other and create inconsistent data in the memory. Threads interleave when they are performing operations with multiple steps and their sequence of steps overlap. This is called thread interference.

Memory Consistency Errors

In multithreading, there can be possibilities that the changes made by one thread might not be visible to the other threads and they all have inconsistent views of the same shared data. This is known as memory consistency error.

Synchronized Methods

There are two different synchronization types in java programming language:

- synchronized methods:
    - To make a method synchronized we just add synchronized keywords to the method’s declaration easily.
    - Making a method synchronized has two effects in concurrent programming:
        - It brings the improbability of accessing one shared memory location or shared resource by two invocations at the same time.
        - when a synchronized method exits, it automatically establishes a happens-before relationship with any subsequent invocation of a synchronized method for the same object. This guarantees that changes to the state of the object are visible to all threads.

- synchronized statements

Intrinsic Locks and Synchronization

Intrinsic lock or monitor lock plays an important role in different aspects of synchronization :

- It enforces exclusive access to the object's state .
- It establishes a Happens-before relationship that is essential for visibility .

Reentrant Synchronization

Synchronized blocks in Java are reentrant. This means, that if a Java thread enters a synchronized block of code, and thereby takes the lock on the monitor object the block is synchronized on, the thread can enter other Java code blocks synchronized on the same monitor object.

Atomic Access

In concurrent programming atomic operations means those kinds of operations which can be done completely or they don't happen at all and no middle action such as reading from or writing to a memory variable. And there is no side effect of doing the action until the end of the operation.

- Reads and writes are atomic for reference variables and for most primitive variables (all types except long and double).
- Reads and writes are atomic for all variables declared volatile (including long and double variables).

Liveness

A liveness property asserts that program execution eventually reaches some desirable state. While termination has been studied extensively, many other liveness properties are important for concurrent programs: deadlock,starvation and livelock.

Deadlock

Deadlock is a situation when two or more threads are waiting for each other. Each of them are waiting for others to get an event about the resource to be freed up. And they with wait in lock forever.

Starvation and Livelock

Starvation

Starvation is a situation when a specific thread is not able to get the desired resource to be fed up with and isn't able to go further and the thread’s progress will stop.

Livelock

Livelock is a situation when a thread wants to send a response to another thread and the target thread is also sending a response to that thread at the same time.

Guarded Blocks

Guarded block is a mechanism of coordinating the execution of multiple threads in a multithreaded environment. Guarded block keeps checking for a particular condition to become true and only in that case the actual execution of the thread resumes.

Immutable Objects

An immutable object is an object whose state cannot be changed after it’s creation or construction (Constructor calling).

High Level Concurrency Objects

Lock Objects

If a thread wants to execute a synchronized method on the given object. First, it has to get the lock of that object. Once thread gets the lock then it is allowed to execute any synchronized method on that object. Once method execution completes automatically thread releases the lock.

Executors

Making threads by Runnable Objects and Thread Objects are discussed previously and they work well for small applications. But for larger applications the thread making and management should be separated to have better efficiency which is implemented by encapsulation concept in executors.

Thread Pools

In a multithreadedWhen the thread pool is created, it will either instantiate a certain number of threads to make available or create new ones as needed depending on the needs of the implementation. application, thread pool is a "pool of available threads" that can be used by your application.Thread pool is a collection of managed threads usually organized in a queue, which execute the tasks in the task queue.

Fork/Join

fork/join framework helps you to get benefit from multiprocessing applications through the ExecutorService interface. It is useful if you want to break down the application into a number of pisces recursively. The first step of using the join fork framework is to write the part of the code which is going to be executed as a single segment of the whole application.

Concurrent Collections

All the classes present in Concurrent Collections are synchronized in nature. Therefore In Concurrent classes, we don't have to take care about Thread-safety.
There are a number of Concurrent Collections provided by java in java.util.concurrent package.

- BlockingQueue
- ConcurrentMap
- ConcurrentNavigableMap
- ConcurrentSkipListMap

Atomic Variables

They are a kind of classes provided by java to work with atomic operations on some variables like volatile variables and they have provided get and set like read and write.

Concurrent Random Numbers

It is useful for concurrent applications which want to generate random numbers for multiple threads. And it has been implemented in java.util.concurrent package.