Concurrency in software execution can occur at four different levels:

1. Instruction level: Executing two or more machine instructions simultaneously.
2. Statement level: Executing two or more high-level language statements simultaneously.
3. Unit level: Executing two or more program units simultaneously.
4. Program level: Executing two or more programs simultaneously.

A concurrent algorithm is realizable if the speed of its execution increases when more processors are available.

Two distinct categories of concurrent unit control are:

1. Physical Concurrency:
   - Assuming that more than one processor is available, several program units from the same program literally execute simultaneously.
2. Logical Concurrency:
   - Assuming that there are multiple processors providing actual concurrency when in fact the actual execution of programs is taking place in interleaved fashion on a single processor.

Programs that have concurrency but no concurrent sub-programs, though, they are sometimes called quasi-concurrent.

- A thread of control in a program is the sequence of program points reached as control flows through the program.
- A program designed to have more than one thread of control is said to be multithreaded.

Use of Concurrency can:

1. Speed of execution of programs on machines with multiple processors.
2. Programs written to use concurrent execution can be faster than the same program written for sequential execution.
3. Different method for conceptualizing program solutions to problems.
4. Program applications are distributed over several machines.
2. Subprogram level concurrency

Fundamental concepts:

1. Task - Unit of a program. Also called process.
2. Thread - Methods that are executed in objects.
3. Characteristics of Task:
   a. Implicitly started
   b. It need not wait for the task to complete its execution before continuing its own execution.
   c. When execution of task is completed, it may or may not return to the unit that started that execution.
4. Categories of Tasks:
   a. Heavyweight tasks - executes in its own address space.
   b. Lightweight tasks - all run in the same address space.
5. Synchronization:
   a. There is a mechanism that controls the order in which tasks execute.
6. Design:
   a. Synchronization synchronization: cooperation between tasks
   b. Synchronization synchronization: cooperation between processes
7. Task states: (5 states)
   a. New
   b. Ready
   c. Running
   d. Completed
   e. Blocking

Concurrency is implemented through libraries.
3. **Synchronization**

   - It is a simple mechanism that can be used to provide
     synchronization of tasks. Specifically, a semaphore is a
     data structure that consists of an integer and a queue that
     stores task descriptors.

   - **Task Descriptor** - It is a data structure that stores all the
     relevant information about the execution state of a task.

   - **Guard** - It is a linguistic device that allows the guarded
     code to be executed only when a specified condition is true.

4. **Cooperation Synchronization**

   - **RECEIVE subroutine** - fetch the item contained in the buffer
   - **DEPOSIT subroutine** - deposit the item to the buffer
   - **WAIT subroutine** - Used to test the counter of a given semaphore
   - **RELEASE subroutine** - Release the counter for another task

3. **Competition Synchronization**

   - Binary semaphore - Requires only a binary-valued counter

4. **Monitors**

   - All synchronization operations on shared data be gathered into a
     single program unit called monitors.

   - **Competition Synchronization** - Synchronized access by allowing only
     one access at a time

   - **Cooperation Synchronization** - It is still the task of the programmer

   ![Diagram of Monitor with operations](image)
5. **Message Passing** - It can be either synchronous or asynchronous.
   - A rendezvous can occur only if both the sender and receiver want it to happen.
   - Both cooperation and completion synchronization of tasks can be conveniently handled with.

6. **Java Threads**
   - Concurrent units in Java are methods named *run*. The order in which the *run* methods execute is called a *thread*.
   - Java's thread are lightweight tasks.
   - **The Thread Class** -
     - class *MyThread* extends *Thread*?
     - public void *run*() {
     - Thread *myTh* = new *MyThread*();
     - *myTh*.start();
   - Several methods for controlling the execution of threads are: 
     1. *yield* method - takes no parameters.
     2. *sleep* method - takes a single parameter.
     3. *join* method - used to force a method to delay its execution.
   - Three basic thread methods: *stop*, *suspend* and *resume*.

   **Priorities**
   - 5 - *NORMAL_PRIORITY*, 10 - *MAX_PRIORITY* and 1 - *MIN_PRIORITY*.
   - *set* *Priority* and *get* *Priority* methods are used.
   - **Semaphore** - java.util.concurrent.Semaphore package.
   - **Synchronization** - Using lock and keyword *synchronized*.
   - **Coopertation Synchronization** - Using *wait*, *notify* and *notifyAll* methods.
   - **Non-blocking Synchronization** - No waiting, java.*,util.*concurrent.*atomic*.*lock*.
   - **Explicit locks** - Using lock, unlock and trylock methods.
C# Threads - (lightweight tasks)

Using predefined delegate, ThreadStart
& public void MyRun1() { ... }

Thread myThread = new Thread (new ThreadStart (MyRun1));
myThread.Start();

→ A thread can be terminated using Abort method
→ myThread. BeginInvoke (10, null, null); // Asynchronously.

→ Synchronizing Threads -
  Three different ways that C# threads can be synchronized -
  Interlocked class, Monitor class (System.Threading) & lock statement (for the integer) (for the critical section)

****
→ Any method can be run in its own thread

Types of Storage Management -

1. **Static Storage Management** -
   Simplicity form of allocation. Ordinarily storage for the code segments of user and system programs is allocated statically.
   **Advantages** -
   (1) Requires no run-time storage management
   (2) Efficient because no time or space is expended for storage management during execution.
   **Disadvantages** -
   Incompatible with recursive subprogram calls, with data structures whose range is dependent on computed or input data.

2. **Stack based Storage Management** -
   Sequential locations in the stack. A single stack pointer is needed to control storage management using stack. Stack pointer always points to free storage block.
3. Heap storage management - (Two techniques - fixed-sized variable space)

A heap is a block of storage within which pieces are allocated and freed in some relatively unstructured manner.

Advantages -
- Language permits storage to be allocated and freed at arbitrary points during program execution.

Disadvantages -
- The problems of storage allocation, recovery, compaction, and release may be severe.

4. Garbage Collection -

It is a form of automatic memory management.

This frees the programmer from needing to delete objects explicitly when they are no longer needed.

It is used for storage reclamation.

Abstract Data Types -

1. An abstraction is a view or representation of an entity that includes only the most significant attributes.

An abstract data type is an enclosure that includes only the data representation of one specific data type and the subroutines that provide the operations for that data type.

An instance of an abstract data type is called an object.

2. Floating Point as an abstract data type

3. User-defined abstract datatype -

Characteristics - type definition and have a set of operations

Encapsulation - a single unit containing collection of data and functions.

Object-oriented programming in Smalltalk, C++, Java, C#, PHP, Perl.