Principles of Programming Languages(CS F301)

BITS Pilani
Hyderabad Campus

Prof.R.Gururaj
CS&IS Dept.

Brief Introduction to Concurrency (Ch.13 of T1)

BITS Pilani
Hyderabad Campus

Prof R Gururaj

Introduction to Concurrency (Ch.13)

Concurrency in SW execution can occur at four levels:

1. Instruction Level
2. Statement Level
3. Unit (subprogram) Level
4. Program level

Instruction level and Program level concurrency has nothing to do with Programming language design.

Statement level and subprogram level concurrency involve design decisions while designing PLs.

Multiprocessor architecture

We use the phrase *physical concurrency* when multiple processors are actually used to execute concurrent units.

If concurrent units are executed on a single processor in an interleaved fashion, it is known as *logical concurrency*.

Most multiprocessor computers either belong to SIMD or MIMD.

SIMD: Single Instruction Multiple Data. Vector processors.

MIMD : Multiple Instruction Multiple Data.

SIMD

Each processor has its own memory.

Multiple processors execute same instruction simultaneously, where each processor working on different data.

One processor called the controller, controls the operations of other processors.

Scientific computations are the most suited kind.

Widely used for graphics, video processing.

MIMD

Computers have multiple processors that operate independently, but whose operations can be synchronized are called MIMD.

Each processor executes its own instruction stream.

MIMD can be of one of the following type :

1. Distributed (each processor has its own memory)
2. Shared memory (all processors share the same memory). Memory access clashes can occur.

Categories of Concurrency

We use the phrase *physical concurrency* when multiple processors are actually used to execute concurrent units.

If concurrent units are executed on a single processor in an interleaved fashion, it is known as *logical concurrency*.

The concept of *logical concurrency*, allows the programmer and application SW to assume that there exist multiple processors providing actual concurrency, in fact the execution of programs happens on same processor in interleaved fashion.

From the programmer and language designer point of view the logical concurrency is same as physical concurrency.

It is language implementer's task, using the underlying operating system , to map logical concurrency to the host HW.

Motivations to use Concurrency

1. Speed of execution of programs on machines with multiple processors.
2. Single processor concurrent execution can be much faster.
3. Concurrency provides better solution certain problems, like how recursion can help in certain situations.
4. Applications can be distributed over multiple machines locally or through the Internet.

Introduction to Subprogram level concurrency

1. Task: is a unit of a program that can be in concurrent execution with other units of the same program.
2. Each task in a program can support one thread of control.
3. Tasks are sometimes called processes.
4. Heavyweight tasks– has its own address space
5. Lightweight tasks (ex.threads in Java) – all run under the same address space

Advantages of lightweight processes:

1. Easy to implement
2. Easy to manage execution
3. Inter task Communication is easy (through nonlocal variables)

Synchronization

Is the mechanism that controls the order in which tasks execute.

Two kinds of synchronizations:

- Cooperation Synchronization: is required between task-A and task-B when task-A must wait for task-B to finish. Ex: Producer consumer problem.

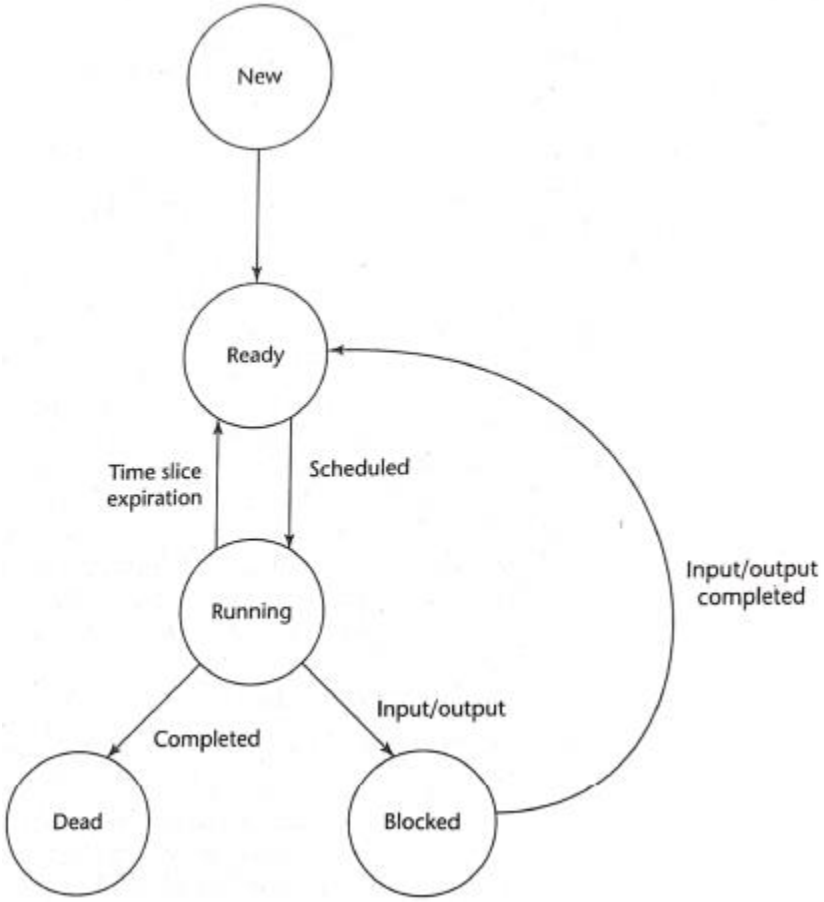
- Competition Synchronization: is required between tasks when both tasks require the use of same resource that can be used simultaneously.

Two of the primary facilities that languages that support subprogram-level concurrency are:

1. Mutually exclusive access to shared data (competition synchronization).
2. Cooperation among tasks (cooperation synchronization).
3. Task can be in any one of the following states:

   new, ready, running, blocked, dead.

Task states


New
Ready
Time slice expiration
Scheduled
Running
Input/output completed
Completed
Input/output
Dead
Blocked

The Design issues for language support for concurrency are:

1. How competition and cooperation synchronization are provided.
2. How an application can influence task scheduling.
3. How and when tasks start and end execution.
4. How and when they are created.

A program has the liveness characteristic if it continues to execute, eventually leading to completion.

Loss of liveness and Deadlock .

Rather than designing language constructs for supporting concurrency, sometimes libraries can be used.

Ex: The API **OpenMP** (**Open Multi-Processing**) supports multi-platform shared-memory multiprocessing programming in C, C++, and Fortran ; #include<omp.h>

Synchronization

innovate achieve lead

Semaphore:

Is a data structure consisting of an integer and a task description queue. Semaphores can be used for implementing both competition and cooperation synchronizations. Easy to use semaphores incorrectly resulting in errors that cannot be detected by the compiler, linker, or run-time system.

Monitors:

Are data abstractions that provide a natural way of providing mutual exclusion. Supported by Java, Ada, C# etc.

Message passing:

The underlying concept is to allow tasks to send messages to each other to synchronize their execution.

Some points to note on Concurrency:

1. Tasks in Ada are heavyweight tasks.
2. Java supports lightweight tasks relatively simple but effective.
3. Java supports monitors for implementing competition synchronization.
4. Java has – wait, notify methods for implementing cooperation synchronization.
5. Java has join, sleep, interrupt methods.
6. Java also has Semaphore class
7. C# supports concurrency based on Java but slightly sophisticated.
8. All .NET languages have use of the concurrent data structures for stacks, queues and bags where competition synchronization is implicit.

Summary of Ch.13 (Concurrency)

innovate achieve lead

1. Introduction
2. Types of concurrency
3. Synchronization
4. Subprogram level concurrency.
5. Message passing
6. Semaphores
7. Monitors