1. Process Concept

A process is basically a program in memory either in execution or waiting for CPU, I/O, or other service. Thus, a process is the fundamental computational unit that requests various types of services in the computer system. A process can contain one or more threads of execution.

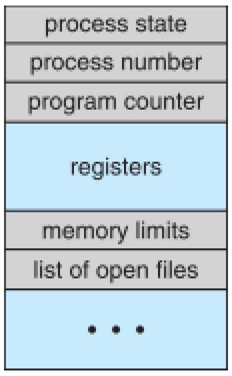
Two basic types of processes are:

1. System processes-execute operating system services
2. User processes-execute application programs

The state of a process is defined in part by the current activity of that process. A process may be in one of the following states:

1. New-The process is being created.
2. Running-Instructions are being executed.
3. Waiting-The process is waiting for some event to occur (such as an I/O completion or reception of a signal).
4. Ready-The process is waiting to be assigned to a processor.
5. Terminated-The process has finished execution.

Each process is represented in the operating system by a process control block (PCB)—also called a task control block.



1. Process state-The state may be new, ready, running, waiting, halted, and so on.
2. Program counter-The counter indicates the address of the next instruction to be executed for this process.
3. CPU registers-The registers vary in number and type, depending on the computer architecture. They include accumulators, index registers, stack pointers, and general-purpose registers, plus any condition-code
4. Information-Along with the program counter, this state information must be saved when an interrupt occurs, to allow the process to be continued correctly afterward (Figure 3.4).
5. CPU-scheduling information-This information includes a process priority,pointers to scheduling queues,and any other scheduling parameters.
6. Memory-management information-This information may include such items as the value of the base and limit registers and the page tables, or the segment tables, depending on the memory system used by the operating system.

In most operating systems, several processes are stored in memory at the same time and the OS manages the sharing of one or more processors (CPUs) and other resources among the various processes. This technique implemented in the operating system is called **multiprogramming**.

2. Process Scheduling

The objective of multiprogramming is to have some process running at all times, to maximize CPU utilization. The objective of time sharing is to switch the CPU among processes so frequently that users can interact with each program while it is running. To meet these objectives, the **process scheduler** selects an available process for program execution on the CPU.

As processes enter the system, they are put into a **job queue**, which consists of all processes in the system. The processes that are residing in main memory and are ready and waiting to execute are kept on a list called the **ready queue.**

The list of processes waiting for a particular I/O device is called a **device queue**.

A new process is initially put in the ready queue. It waits there until it is selected for **execution, or dispatched**

The selection process is carried out by the appropriate **scheduler.**

The long-term scheduler, or job scheduler, selects processes from this pool and <u>loads them into memory</u> for execution.

The short-term scheduler, or CPU scheduler, selects from among the processes that are ready to execute and <u>allocates the CPU to one of them</u>.

The long-term scheduler controls the **degree of multiprogramming** (the number of processes in memory).

An **I/O-bound process** is one that spends more of its time doing I/O than it spends doing computations.

A **CPU-bound process**, in contrast, generates I/O requests infrequently, using more of its time doing computations.

This **medium-term scheduler** remove a process from memory (and from active contention for the CPU) and thus reduce the <u>degree of multiprogramming</u>.

Later, the process can be reintroduced into memory, and its execution can be continued where it left off. This scheme is called **swapping**.

Switching the CPU to another process requires performing a state save of the current process and a state restore of a different process. This task is known as a **context switch.**

2. Operation on Process

During the course of execution, a process may create several new processes. As mentioned earlier, the creating process is called a **parent process**, and the new processes are called the **children of that process**.

Each of these new processes may in turn create other processes, forming a **tree of processes**.

Most operating systems (including UNIX, Linux, and Windows) identify processes according to a unique process identifier (or pid).

3. Threads Overview

A process can have multiple threads or sequences of executions. A thread is often called a lightweight process and is a (dynamic) component of a process.

4. Multithreading Models

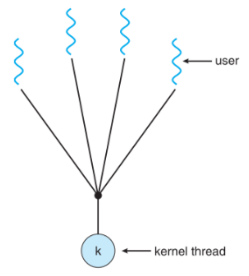
There are two general types of threads:

1.User-level threads-the thread management is carried out at the level of the application without kernel intervention.
2.Kernel-level threads.- The thread management tasks are carried out by the kernel.

Ultimately, a relationship must exist between user threads and kernel threads.

Multithreading models:

1. Many To One Model
The many-to-one model maps many user-level threads to one kernel thread. Thread management is done by the thread library in user space, so it is efficient.



2. One To One Model
The one-to-one model (Figure 4.6) maps each user thread to a kernel thread.
It also allows multiple threads to run in parallel on multiprocessors.
The only drawback to this model is that creating a user thread requires creating the corresponding
kernel thread. Because the overhead of creating kernel threads can burden the performance of an application,

3. Many To Many Model
The many-to-many model multiplexes many user-level threads to a smaller or equal number of kernel threads.

5. Multicore Programming

A **multi-core** processor is a single computing component with two or more independent actual processing units (called "cores"), which are the units that read and execute **program** instructions.

The operating system with multiprogramming manages the system to support multiple processes in memory.

While one process receives service from the CPU, another process receives service from an I/O device, and the other processes are waiting in the queues; all of this occurs at the same time.

Multiprogramming is also the ability of an operating system to execute more than one program on a single processor machine.

More than one task/program/job/process can reside into the main memory at one point of time.

A computer running excel and firefox browser simultaneously is an example of multiprogramming

6. Process Synchronization

The focus of synchronization is the coordination of the activities of the active processes in the computer system.

Processes need to be synchronized when they compete for shared resources or when they need to cooperate in carrying out their activities.

No Synchronization:
When two or more processes execute and independently attempt to simultaneously share the same resources, their executions generally will not produce correct results.

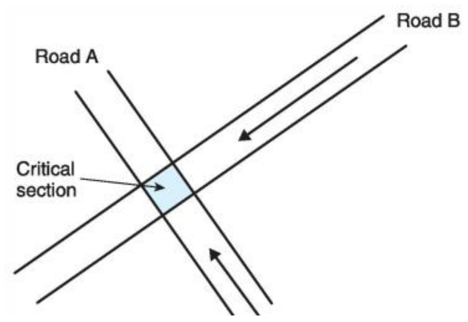
A simple example is a global variable shared by several processes. One process increases the value of the global variable, while another is taking its value to add to a local variable. This leads to unpredictable results and is sometimes called a **race condition**.

Mutual Exclusion:
To solve the race condition problem when a group of processes are competing to use a resource, only one process must be allowed to access the shared resource at a time.

At any given time, only one process is allowed to access a shared resource all other processes must wait.

Critical Sections:

A simple analogy to help understand the concept of a critical section is the example of a road intersection shown in Figure



Two vehicles, one moving on Road A and the other moving on Road B, are approaching the intersection. If the two vehicles reach the intersection at the same time, there will be a collision, which is an undesirable event.Therefore, mutual exclusion should be applied on the road intersection.

The part or segment of code in a process that accesses and uses a shared resource is called the critical section.

There are several approaches for implementing solutions to the critical section problem. These can be grouped into three categories:

1. Busy waiting (Peterson's algorithm)
2. Hardware support
3. Operating system support

Semaphores:

The solution to the road intersection analogy shown in Figure is the installation of a traffic light to coordinate the use of the shared area in the intersection.

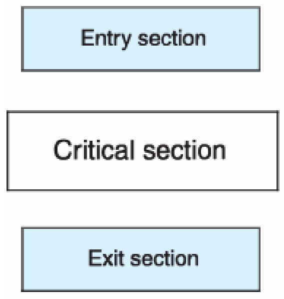
It is an abstract data type that functions as a software synchronization tool that can be used to implement a solution to the critical section problem.

A semaphore is an object whose methods are invoked by the processes that need to share a resource.

There are two general types of semaphores-

1.binary semaphore, whose integer attribute sem can take only two values, 0 or 1, and is used basically to allow mutual exclusion.

2.The counting semaphore, whose integer attribute sem can take any nonnegative integer value.



Monitors:

A monitor is a mechanism that implements mutual exclusion. It is a synchronization tool that operates on a higher level than a semaphore in the synchronization of processes.

Monitors are abstract data types implemented as a class.

Only a single process can execute an operation of the monitor at any given time. The monitor object provides mutual exclusion, and its member function execution is treated as a critical section.

An arriving process must wait in the entry queue of the monitor before entering. After entering, a process can be suspended and later reactivated.

Several queues are maintained by the monitor to hold waiting processes that have entered the monitor but have been suspended.

7. Inter Process Communication

Interprocess communication (IPC) consists of mechanisms that enable processes to exchange data.
This data is normally formatted into messages according to some predefined protocol.
The transfer of messages among the communicating processes requires synchronization.

1. Asynchronous Communication-

in which processes do not directly interact because the message sending and receiving do not occur at the same time. Mailboxes are used as intermediate storage of messages. The mailboxes are maintained by the operating system.

2. Synchronous Communication-

Synchronous communication implies that the communication of a message between two processes involves a stronger synchronization
This is also called direct communication because the sender process and the receiver process have a joint activity and both processes need to participate simultaneously during the period of cooperation.

3. Atomic Transactions