# I. Process Concept.

A process is a program in execution. The status of the current activity of a process is represented by the value of the **program counter** and the contents of the processor’s registers.

The memory layout of a process is typically divided into multiple sections. These sections include:

• **Text section**—the executable code.  
• **Data section**—global variables.

• **Heap section**—the memory that is dynamically allocated during program run time.  
• **Stack section**—temporary data storage when invoking functions (such as function parameters, return addresses, and local variables).

As a process executes, it changes **state**. 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:

• **New**. The process is being created.  
• **Running**. Instructions are being executed.  
• **Waiting**. The process is waiting for some event to occur (such as an I/O completion or reception of a signal).  
• **Ready**. The process is waiting to be assigned to a processor.

Each process is represented in the operating system by a **process control block** (**PCB**)—also called a **task control block.** It contains many pieces of information associated with a specific process, including these:  
• **Process state**. The state may be new, ready, running, waiting, halted, and  
so on.  
• **Program counter**. The counter indicates the address of the next instruction to be executed for this process.

• **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 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 when it is rescheduled to run.  
• **CPU-scheduling information**. This information includes a process priority, pointers to scheduling queues, and any other scheduling parameters.  
(Chapter 5 describes process scheduling.)  
• **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 (Chapter 9).  
• **Accounting information**. This information includes the amount of CPU and real-time used, time limits, account numbers, job or process numbers, and so on.  
• **I/O status information**. This information includes the list of I/O devices allocated to the process, a list of open files, and so on.

# II. Process Scheduling.

Process Scheduling is to have some process running at all times to maximize CPU utilization. The objective of time sharing is to switch a CPU core among processes so frequently that users can interact with each program while it is running.

III. Operations on Processes.

**Process Creation:**

During the course of execution, a process may create several new processes. 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.

**Process Termination:**

A process terminates when it finishes executing its final statement and asks the operating system to delete it by using the exit() system call.

A process can cause the termination of another process via an appropriate system call (for example, TerminateProcess() in Windows).

# IV. Interprocess Communication.

Processes executing concurrently in the operating system may be either independent processes or cooperating processes. A process is ***independent*** if it does not share data with any other processes executing in the system. A process is ***cooperating*** if it can affect or be affected by the other processes executing in the system.  
There are several reasons for providing an environment that allows process cooperation:  
• **Information sharing**. Since several applications may be interested in the same piece of information (for instance, copying and pasting), we must provide an environment to allow concurrent access to such information.  
• **Computation speedup**. If we want a particular task to run faster, we must break it into subtasks, each of which will be executing in parallel with the others. Notice that such a speedup can be achieved only if the computer has multiple processing cores.  
• **Modularity**. We may want to construct the system in a modular fashion, dividing the system functions into separate processes or threads.

Cooperating processes require **interprocess communication** (**IPC**) the mechanism that will allow them to exchange data—that is, send data to and receive data from each other.

There are two fundamental models of IPC: **shared memory** and **message passing**.

# V. IPC in Shared-Memory Systems.

Interprocess communication using shared memory requires communicating processes to establish a region of shared memory. Typically, a shared-memory region resides in the address space of the process creating the shared memory segment. Other processes that wish to communicate using this shared-memory segment must attach it to their address space.

The operating system tries to prevent one process from accessing another process’s memory. Shared memory requires that two or more processes agree to remove this restriction. They can then exchange information by reading and writing data in the shared areas. The form of the data and the location are determined by these processes and are not under the operating system’s control. The processes are also responsible for ensuring that they are not writing to the same location simultaneously.

# VI. IPC in Message-Passing Systems.

Message-Passing Systems is for the operating system to provide the means for cooperating processes to communicate with each other via a message-passing facility.

# VII. Communication in Client-Server Systems.

**Sockets:**

A **socket** is defined as an endpoint for communication. A pair of processes communicating over a network employs a pair of sockets (one for each process.)

A socket is identified by an IP address concatenated with a port number. In general, sockets use client-server architecture. The server waits for incoming client requests by listening to a specified port. When received, the server accepts a connection from the client socket to complete the connection.

**Remote Procedure Calls:**

One of the most common forms of remote service is the RPC paradigm, which was designed as a way to abstract the procedure-call mechanism for use between systems with network connections. However, because we are dealing with an environment in which the processes are executing on separate systems, we must use a message-based communication scheme to provide remote service.