Processes

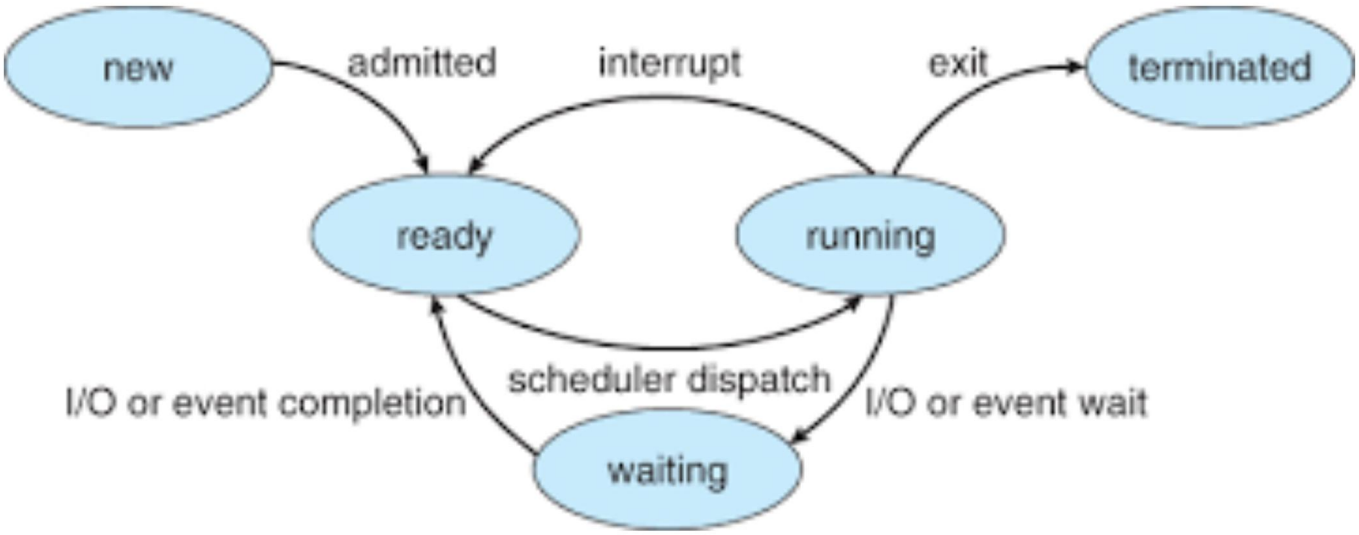
→ A process is the unit of work in a modern time-sharing system.

Modern computer systems allow multiple programs to be loaded into memory and executed concurrently.

Process State

→ 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.
- Terminated : The process has finished execution.

Process Control Block

Process ID
Process state
Process priority
Accounting information
Program counters
CPU registers
PCB pointers
List of open files
Process I/O status Information
.....

Process state : The state may be new, ready, running, waiting, halted, and so on.

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

➔ It contains information associated with a specific process like:

Program Counter : The counter indicates the address of the next instruction to be executed for this process.

CPU Registers : The registers include accumulators, index registers, stack pointers, and general-purpose registers. Along with the program counter, the state information must be saved when an interrupt occurs, to allow the process to be continued correctly afterward

CPU-Scheduling Information : This information includes a process priority, pointers to scheduling queues, and any other scheduling parameters.

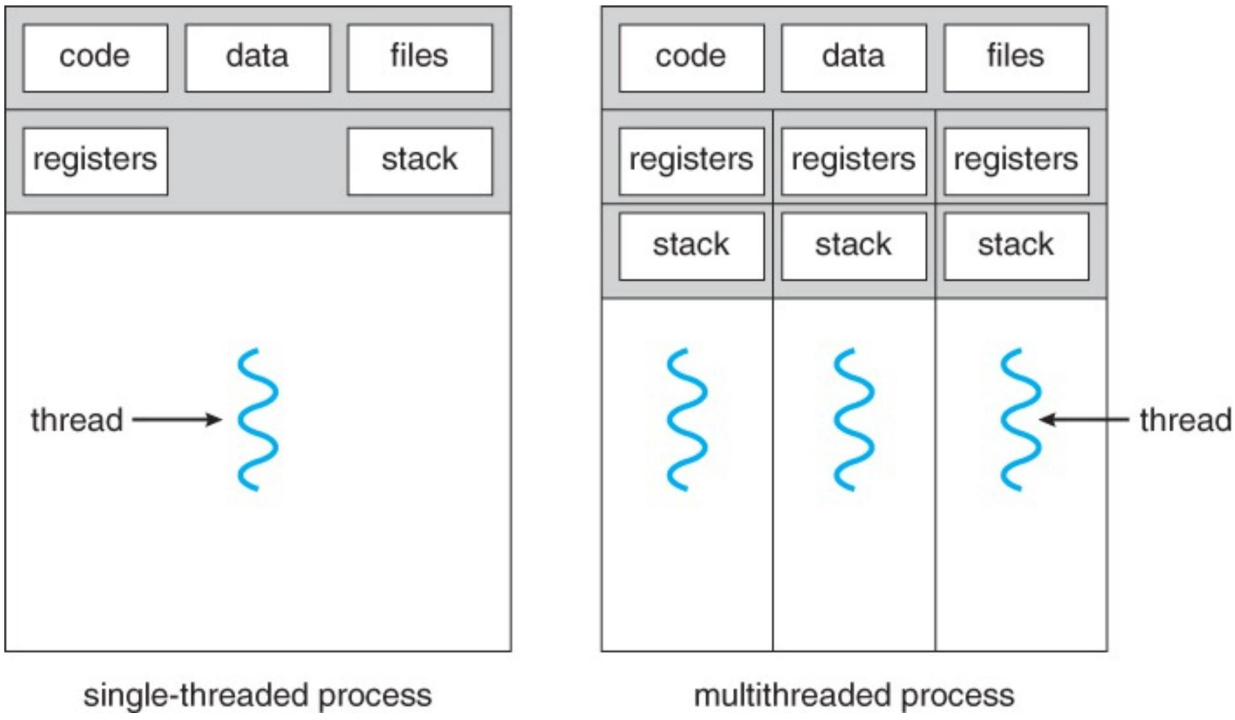
Memory-Management Information : This information may include such items such as the value of the base and limit registers and the page tables or the segment tables

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.

NOTE : PCB simply serves as the repository for any information that may vary from process to process.

Threads



single-threaded process

multithreaded process

A thread is a single sequential flow of execution of tasks of a process so it is also known as thread of execution or thread of control.

➔ For example, when a process is running a word-processor program, a single thread of instructions is being executed.

Single thread of control allows the process to perform only one task at a time. The user cannot simultaneously type in characters and run the spell checker within the same process, for example.

Most modern operating systems have extended the process concept to allow a process to have multiple threads of execution and thus to perform more than one task at a time

Multiple threads can run in parallel. On a system that supports threads, the PCB is expanded to include information for each thread.

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 (possibly from a set of several available processes) for program execution on the CPU.

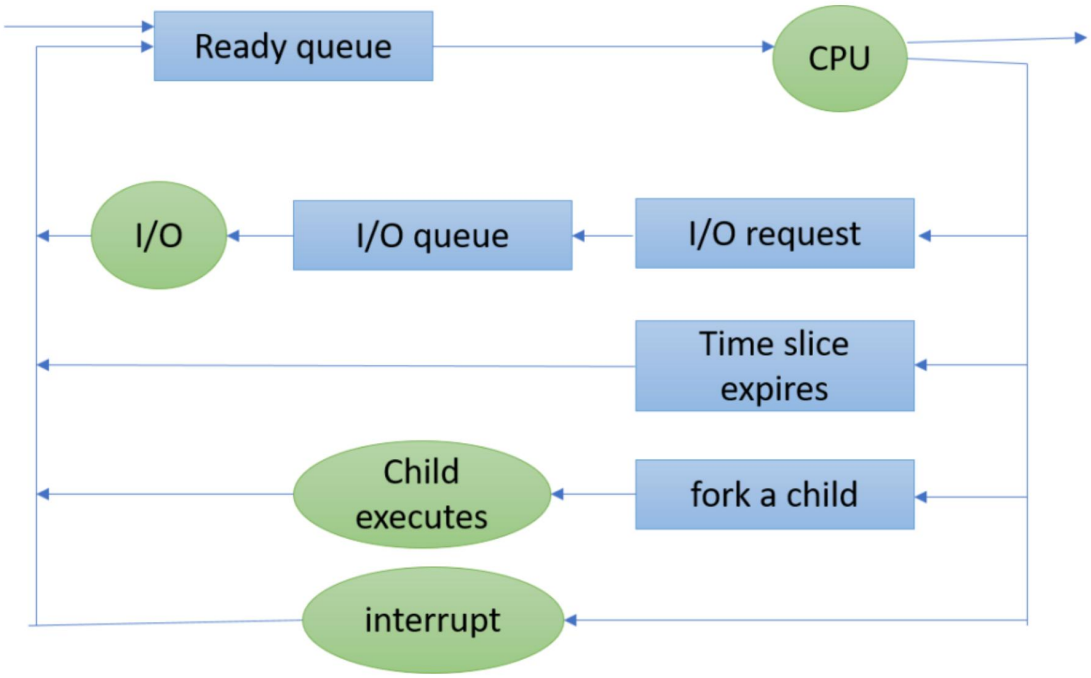
For a single-processor system, there will neverbe more than one running process.

If there are more processes, the rest will have to wait until the CPU is free and can be rescheduled.

Scheduling Queues

→ 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.
This queue is generally stored as a linked list.



A ready-queue header contains pointers to the first and final PCBs in the list. Each PCB includes a pointer field that points to the next PCB in the ready queue.

The system also includes other queues. When a process is allocated the CPU, it executes for a while and eventually quits or is interrupted, or waits for the occurrence of a particular event, such as the completion of an I/O request.

→ Suppose the process makes an I/O request to a shared device, such as a disk. Since there are many processes in the system, the disk may be busy with the I/O request of some other process.
The process therefore may have to wait for the disk.

The list of processes waiting for a particular I/O device are place in it's device queue.
Each device has its own device queue.

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

→ Once the process is allocated the CPU and is executing, one of several events could occur:

- The process could issue an I/O request and then be placed in an I/O queue.
- The process could create a new child process and wait for the child's termination.
- The process could be removed forcibly from the CPU, as a result of an interrupt, and be put back in the ready queue.

Note: In the first two cases, the process eventually switches from the waiting state to the ready state and is then put back in the ready queue.

→ A process continues this cycle until it terminates, at which time it is removed from all queues and has its PCB and resources deallocated.

Each rectangular box represents a queue.

Two types of queues are present: the ready queue and a set of device queues.

The circles represent the resources that serve the queues, and the arrows indicate the flow of processes in the system.

Schedulers

→ A process migrates among the various scheduling queues throughout its lifetime.

The operating system must select processes from these queues in some fashion.

The selection process is carried out by the appropriate scheduler.

→ Long-term scheduler, or job scheduler, selects processes from this pool and loads them into memory for execution.

→ Short-term scheduler, or CPU scheduler, selects from among the processes that are ready to execute and allocates the CPU to one of them.

The primary distinction between these two schedulers lies in frequency of execution. The short-term scheduler must select a new process for the CPU frequently. A process may execute for only a few milliseconds before waiting for an I/O request.

→ Medium-Term Scheduler : The key idea behind a medium-term scheduler is that sometimes it can be advantageous to remove a process from memory (and from active contention for the CPU) and thus reduce the degree of multiprogramming.

Later, the process can be reintroduced into memory,and its execution can be continued where it left off.

This scheme is called swapping. The process is swapped out, and is later swapped in, by the medium-term scheduler.

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

→ It is important that the long-term scheduler make a careful selection.
In general, most processes can be described as either I/O bound or CPU bound.

→ 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.

It is important that the long-term scheduler select a good process mix of I/O-bound and CPU-bound processes.

If all processes are I/O bound, the ready queue will almost always be empty, and the short-term scheduler will have little to do.

If all processes are CPU bound, the I/O waiting queue will always be empty, devices will go unused, and again the system will be unbalanced.

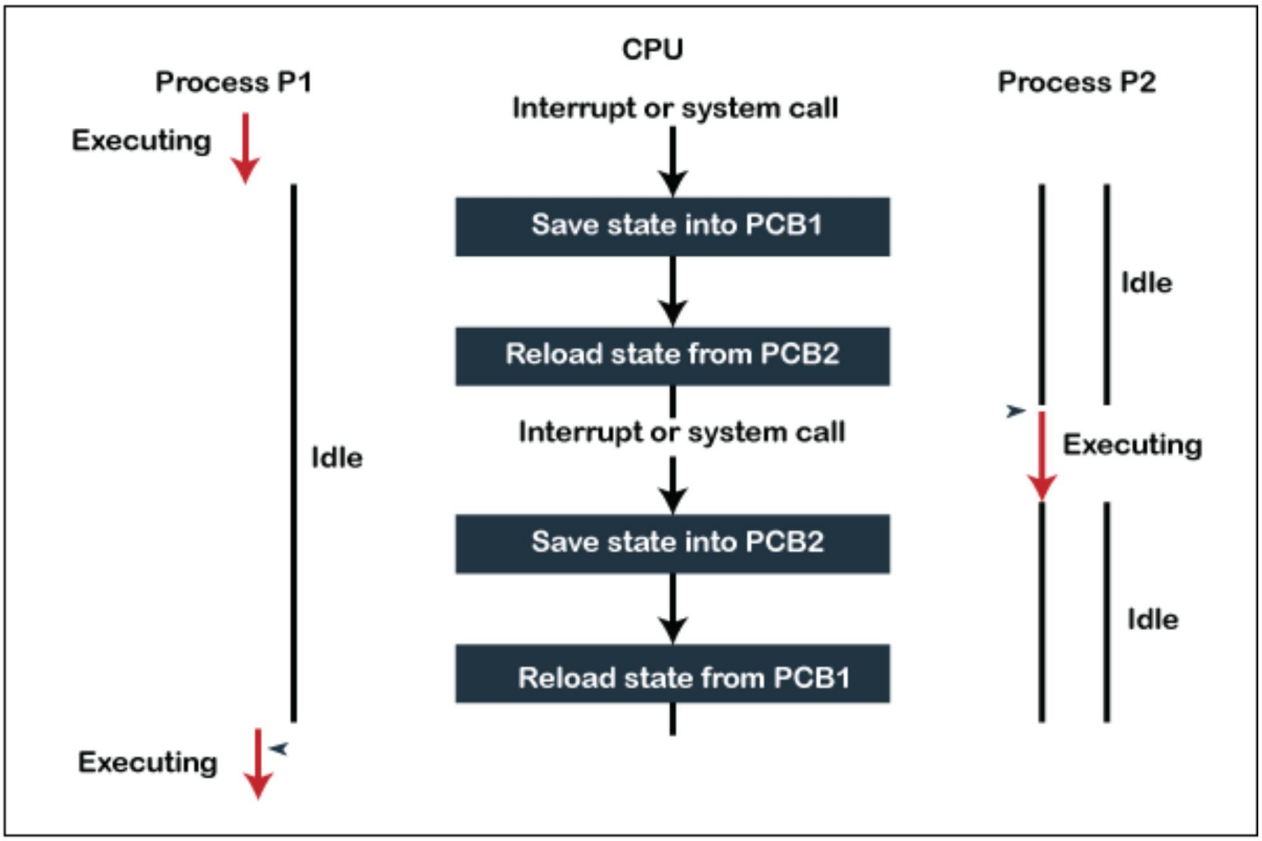
The system with the best performance will thus have a combination of CPU-bound and I/O-bound processes.

Context Switch

→ Interrupts cause the operating system to change a CPU from its current task and to run a kernel routine.

When an interrupt occurs, the system needs to save the current context of the process (state save) running on the CPU so that it can restore that context when its processing is done, essentially suspending the process and then resuming it.

we perform a state save of the current state of the CPU, be it in kernel or user mode, and then a state restore to resume operations.



→ 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.

When a context switch occurs, the kernel saves the context of the old process in its PCB and loads the saved context of the new process scheduled to run. Context-switch time is pure overhead, because the system does no useful work while switching.

Interprocess Communication

→ Processes executing concurrently in the operating system may be either independent processes or cooperating processes.

A process is independent if it cannot affect or be affected by the other processes executing in the system.

Any process that does not share data with any other process is independent.

A process is cooperating if it can affect or be affected by the other processes executing in the system. Clearly, any process that shares data with other processes is a cooperating process.

➔ Reasons for providing an environment that allows process cooperation:

Information Sharing : Since several users may be interested in the same piece of information (for instance, a shared file), 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.

Modularity : We may want to construct the system in a modular fashion, dividing the system functions into separate processes or threads.

Convenience : Even an individual user may work on many tasks at the same time. For instance, a user may be editing, listening to music,and compiling in parallel.

Cooperating processes require an interprocess communication (IPC) mechanism that will allow them to exchange data and in formation.

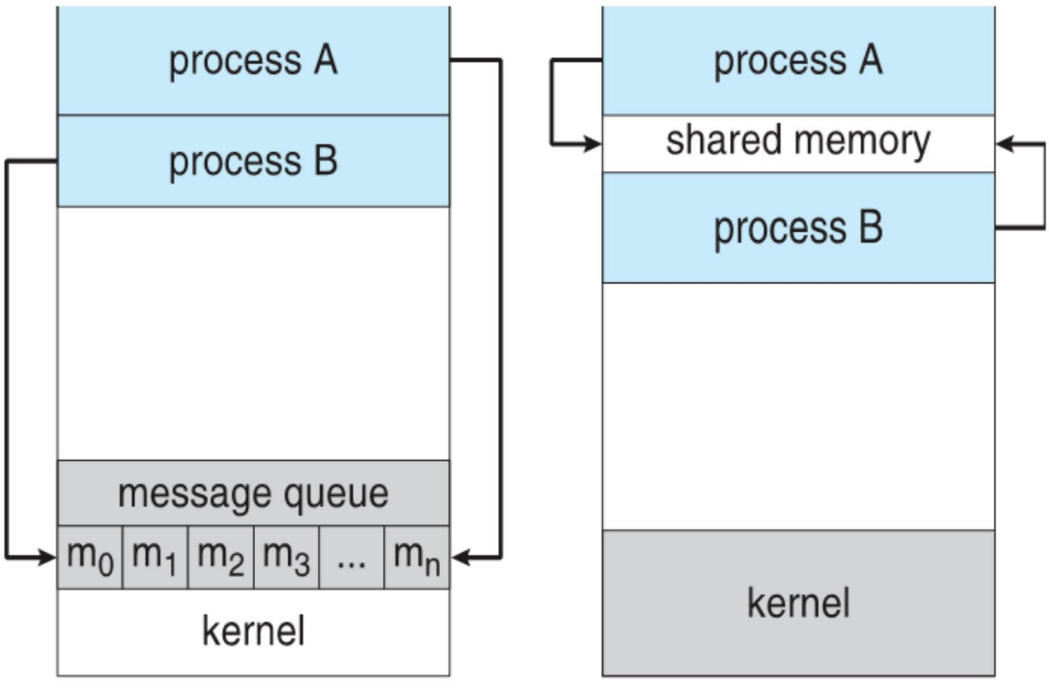
→ There are two fundamental models of interprocess communication: shared memory and message passing.

In the shared-memory model, a region of memory that is shared by cooperating processes is established.

Processes can then exchange information by reading and writing data to the shared region.

In the message-passing model, communication takes place by means of messages exchanged between the cooperating processes.

(a) Message passing. (b) shared memory.



Message passing is also easier to implement in a distributed system than shared memory.

Shared memory can be faster than message passing, since message-passing systems are typically implemented using system calls and thus require the more time-consuming task of kernel intervention.

In shared-memory systems, system calls are required only to establish shared-memory regions. Once shared memory is established, all accesses are treated as routine memory accesses, and no assistance from the kernel is required.

Shared memory suffers from cache coherency issues, which arise because shared data migrate among the several caches.

Shared-Memory Systems

→ Interprocess communication using shared memory requires communicating processes to establish a region of shared memory.

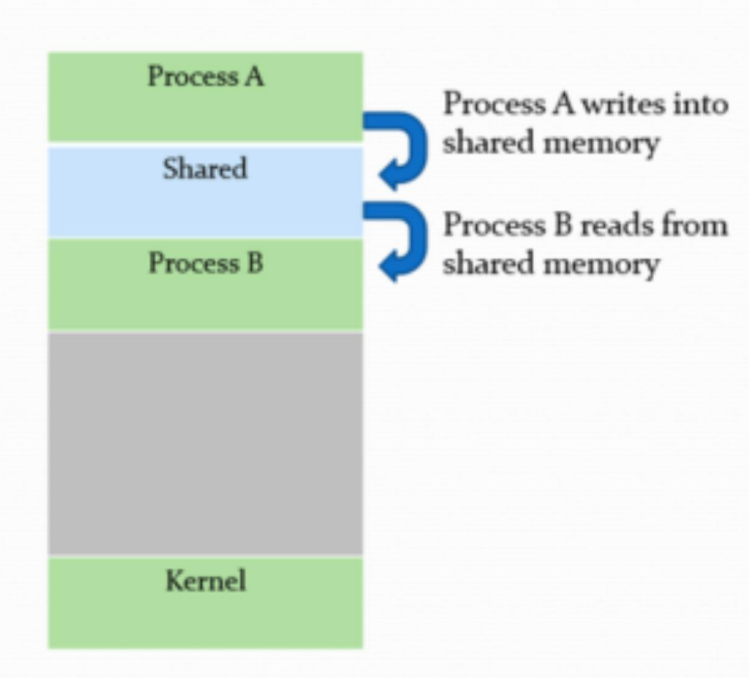
Typically, a shared-memory region resides in the address space of the process creating the shared-memory segment

NOTE: Normally, 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 processes are also responsible for ensuring that they are not writing to the same location simultaneously.

Eg: Producer-Consumer problem

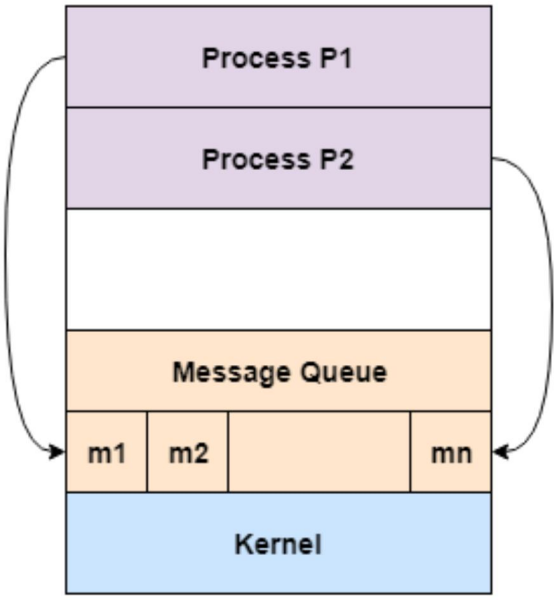


Message-Passing Systems

→ Cooperating processes to communicate with each other via a message-passing facility.

Message passing provides a mechanism to allow processes to communicate and to synchronize their actions without sharing the same address space.

It is particularly useful in a distributed environment, where the communicating processes may reside on different computers connected by a network.



Message Passing Model

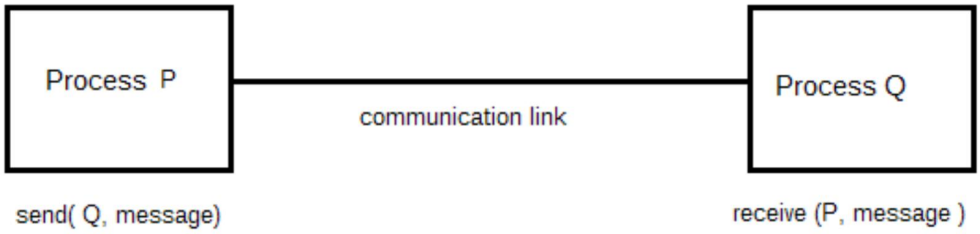
→ If processes P and Q want to communicate, they must send messages to and receive messages from each other: a communication link.

A message-passing facility provides at least two operations:

send(message) receive(message)

- Direct or indirect communication(Naming)
- Synchronous or asynchronous communication
- Automatic or explicit buffering

Naming



→ Under direct communication, each process that wants to communicate must explicitly name the recipient or sender of the communication.

In this scheme, the send() and receive() primitives are defined as:

- send(P, message)—Send a message to process P.
- receive(Q, message)—Receive a message from process Q.

→ A communication link in this scheme has the following properties:

- A link is established automatically between every pair of processes that want to communicate.
  The processes need to know only each other's identity to communicate.

- A link is associated with exactly two processes.

- Between each pair of processes, there exists exactly one link.

This scheme exhibits symmetry in addressing; that is, both the sender process and the receiver process must name the other to communicate

→ A variant of this scheme employs asymmetry in addressing. Here, only the sender names the recipient; the recipient is not required to name the sender.

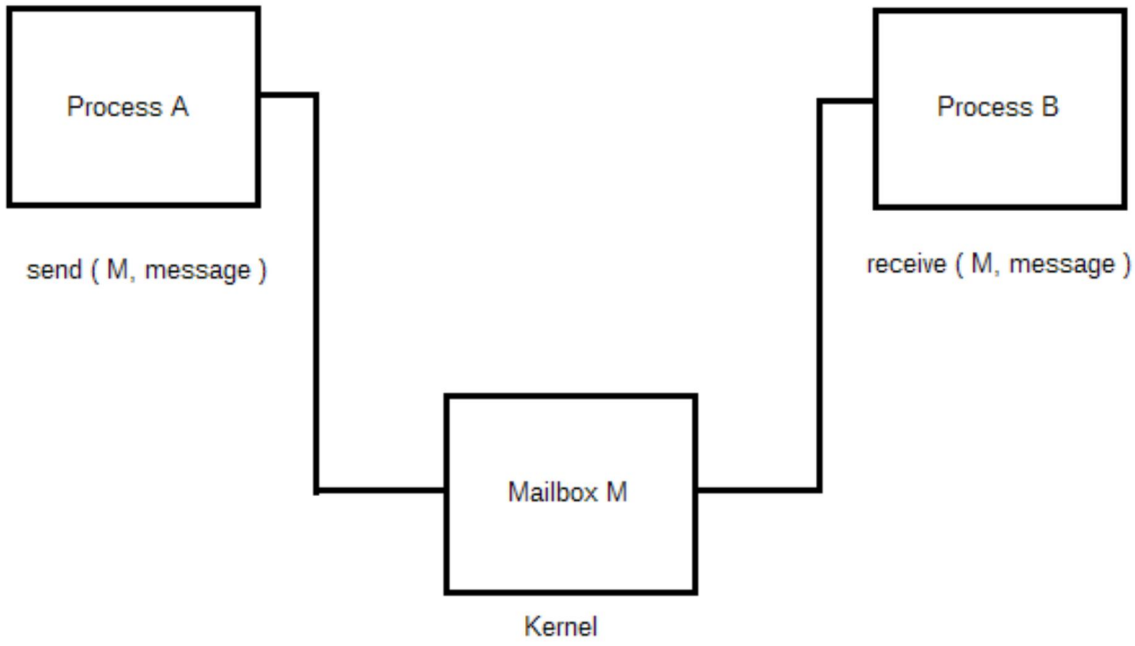
In this scheme, the send() and receive() primitives are defined as follows:

- send(P, message)–Send a message to process P.

- receive(id, message)–Receive a message from any process.

The variable id is set to the name of the process with which communication has taken place.

→ With indirect communication, the messages are sent to and received from mailboxes, or ports.



→ A mailbox can be viewed abstractly as an object into which messages can be placed by processes and from which messages can be removed.

→ A process can communicate with another process via a number of different mailboxes, but two processes can communicate only if they have a shared mailbox

- send(A, message)—Send a message to mailbox A.
- receive(A, message)—Receive a message from mailbox A.

→ In this scheme, a communication link has the following properties:

- A link is established between a pair of processes only if both members of the pair have a shared mailbox.
- A link may be associated with more than two processes.

Synchronization

→ Communication between processes takes place through calls to send() and receive() primitives.

Message passing may be either blocking or nonblocking also known as synchronous and asynchronous.

- Blocking send : The sending process is blocked until the message is received by the receiving process or by the mailbox.
- Nonblocking send : The sending process sends the message and resumes operation.
- Blocking receive : The receiver blocks until a message is available.
- Nonblocking receive : The receiver retrieves either a valid message or a null.

Buffering

Whether communication is direct or indirect, messages exchanged by communicating processes reside in a temporary queue. Basically, such queues can be implemented in three ways:

Zero capacity : The queue has a maximum length of zero; thus, the link cannot have any messages waiting in it

Bounded capacity : The queue has finite length n ;thus, at most 'n' messages can reside in it.If the queue is not full when a new message is sent, the message is placed in the queue.

Unbounded capacity : The queue's length is potentially infinite thus , any number of messages can wait in it. The sender never blocks.

NOTE: The zero-capacity case is sometimes referred to as a message system with no buffering

Get my Book "The Art of Data Structures and Algorithms" and Access all the most important DSA Problems and Solutions.

Download Interview Cafe Android App and Get this book

Get the Book from Amazon

Santosh Kumar Mishra
SDE @ Microsoft

Follow for more

iamsantoshmishra

iamsantoshmishra

Interview Cafe

Interview Cafe Notes