Describe the actions taken by a kernel to context-switch between processes.

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

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



|  |  |
| --- | --- |
| **shared-memory model** | **Message passing 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. |
| faster than message passing | \*Slower because it’s implemented using system calls and thus require the more time-consuming task of kernel intervention. \*Useful for exchanging smaller amounts of data, because no conflicts need be avoided. |
| 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. | Provides a **mechanism** to allow processes to communicate and to synchronize their actions without sharing the same address space.\*useful (easier to implement)in a distributed environment, where the communicating processes may reside on different computers connected by a network. |
| system calls are required only to establish shared memory regions.Then: all accesses are treated as routine memory accesses, and no assistance from the kernel is required. | A message-passing facility provides at least two operations:send(message) receive(message) |
| For example, a web server produces (that is, provides) HTML files and images, which are consumed (that is, read) by the client web browser requesting the resource. | We generally think of a server as a producer and a client as a consumer. For example, a web server produces (that is, provides) HTML files and images, which are consumed (that is, read) by the client web browser requesting the resource. |
| Shared memory suffers from cache coherency issues, which arise because shared data migrate among the several caches. | Research indicates that it provides better performance than shared memory on systems with several cores |

If processes *P* and *Q* want to communicate: a ***communication link*** must exist between them.

Several methods for logically implementing a link and the send()/receive() operations:

•Direct or indirect communication (Naming)

• Synchronous or asynchronous communication

• Automatic or explicit buffering

**1- Direct or indirect communication (Naming)**

|  |  |  |
| --- | --- | --- |
| **Direct:** Each process that wants to communicate must explicitly **name** the recipient or sender of the communication.(symmetry in addressing) | ***Direct: asymmetry*** in addressing. Here, only the sender names the recipient; the recipient is not required to name the sender. | **Indirect**: the messages are sent to and received from***mailboxes***, or ***ports****A mailbox*: like an object into which messages can be placed by processes and from which messages can be removed. |
| 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. | 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. | send() and receive() primitives are defined as follows:• send(A, message)—Send a message to mailbox A.• receive(A, message)—Receive a message from mailbox A. |
| * Each mailbox has a unique ID. Example, POSIX message queues use an integer value to identify a mailbox.
* 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.
 |
| •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. |  | • 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.• Between each pair of communicating processes, a number of different links may exist, with each link corresponding to one mailbox. |
| The disadvantage in both of these schemes (symmetric and asymmetric)is the limited modularity of the resulting process definitions (Changing the identifier of a process may necessitate examining all other process definitions). |  |

**2-Synchronization**

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

There are different design options for implementing each primitive:

• **Blocking send**.

• **Nonblocking send**.

• **Blocking receive**.

• **Nonblocking receive**.

|  |  |  |
| --- | --- | --- |
|  | Sending process | Receiving process |
| Blocking(**synchronous)** | Blocked  | Waiting the massage to recive |
| Waiting the message to be available | blocked |
| **rendezvous** | blocked | Blocked(both) |
| Nonblocking (**asynchronous)**. | Sends the message and waits  |  |
|  | Retrieves either a valid or null message |

**3-Buffering**

Messages exchanged by communicating processes reside in a temporary queue whether communication is direct or indirect.

Such queues can be implemented in three ways:

|  |  |  |  |
| --- | --- | --- | --- |
|  | **Zero capacity**. | **Bounded capacity**. | **Unbounded capacity**. |
| Queue length | queue has a maximum length of zero | queue has finite length *n;* at most *n* messages can reside in it | queue’s length is infinite |
| link | The link cannot have any messages waiting in it. | A: Queue is not full: any new message is placed in it when arrived.B:the link is full: | Any number of messages can wait in the queue. |
| sender | The sender must block until the recipient receives the message. | A:sender can continue execution without waitingB: the sender must block until space is available in the queue. | The sender never blocks |