INTERPROCESS COMMUNICATION MECHANISMS

Introduction

Processes often need to communicate with each other. Inter process communication mechanisms are provided by the operating system as part of the process abstraction.

In general, a process can send a communication in one of two ways:

(i) Blocking, and

(ii) Nonblocking.

After sending in a blocking communication, the process goes into the waiting state until it receives a response. Nonblocking communication allows the process to continue execution after sending the communication.

There are two modes through which processes can communicate with each other:

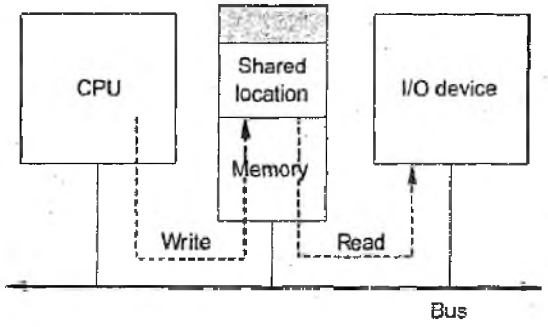
(i) Shared memory, and

(ii) Message passing.

The shared memory region shares a shared memory between the processes. On the other hand, the message passing lets processes exchange information through messages.

Shared Memory Communication

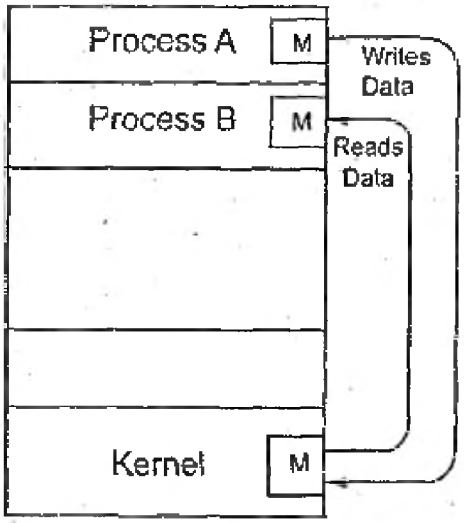
Shared memory communication works in a bus-based system. Here, a CPU and an I/O device is communicated through a shared memory location. The software on the CPU has been designed to know the address of the shared location which has also been loaded into the proper register of the I/O device.



If the CPU wants to send data to the device, it first writes to the shared location. The I/O device then reads the data from that location.

Message Passing

In the message passing mode, processes interact with each other through messages with assistance from the underlying operating system.

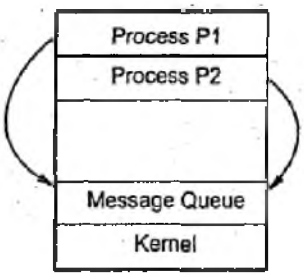


A, and B are communicating with each other through message passing. Process A sends a message M to the operating system (kernel). This message is then read by process B. Each communication entity (CPU or process) has its own message send/receive unit.

(1) Queues

A queue is a common form of message passing. The queue uses a FIFO discipline and holds records that represent the messages.

The **FreeRTOS.org** system provides a set of queue functions which allows queues to be created and deleted so that the system may have as many queues as necessary.



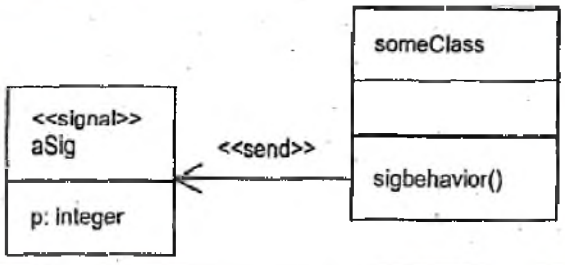
Signals

An interprocess communication commonly used in Unix is the signal which does not pass data.

A siganls a notification to a process indicating the occurrence of an event. Signal is also called software interrupt and it is not predictable to know its occurrence; hence it is also called as an asynchronous event.

UML signal is an object which can carry parameters as object attributes.

Fig shows the use of a signal in UML. The sigbehavior () behavior of the class is responsible for throwing the signal, as indicated by «send». The signal object is indicated by the «signal» stereotype.



Mailboxes

The mailbox is a simple mechanism for asynchronous communication. It has a fixed number of bits and can be used for small messages. Some architectures define mailbox registers.

We can implement a mailbox by using P() and V( ) in main memory for the mailbox storage. A very simple version of a mailbox is that it holds only one message at a time.