Inter-Task Communication Mechanisms

1. Introduction

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

1.1 Blocking and Non-Blocking Communication

Style	Behaviour
Blocking	After sending, the process goes to the waiting state until it receives a response
Non-blocking	The process continues execution after sending

Both types are useful.

1.2 Two Major Styles

Shared memory and **message passing** are logically equivalent – given one you can build an interface that implements the other. Some programs are easier to write with one, and the hardware platform may make one easier or more efficient.

2. Shared Memory Communication

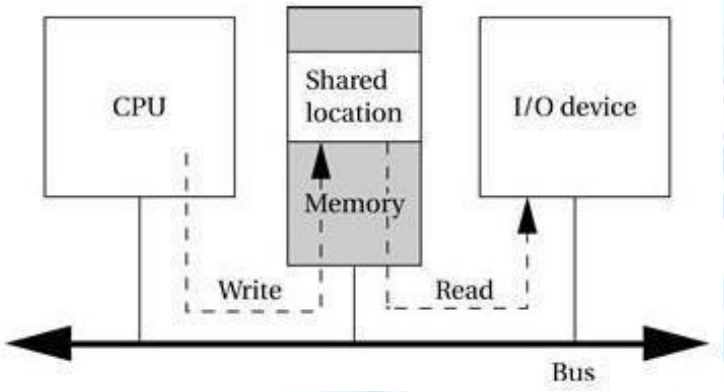


Figure 3.3 Shared memory communication in a bus-based system

- Two components, such as a CPU and an I/O device, communicate through a shared memory location.
- The CPU software knows the address of the shared location; the same address is loaded into the proper register of the I/O device.
- To send data, the CPU writes to the shared location and the I/O device reads it. Reads and writes are standard operations that can be encapsulated in a procedural interface.

3. Message Passing

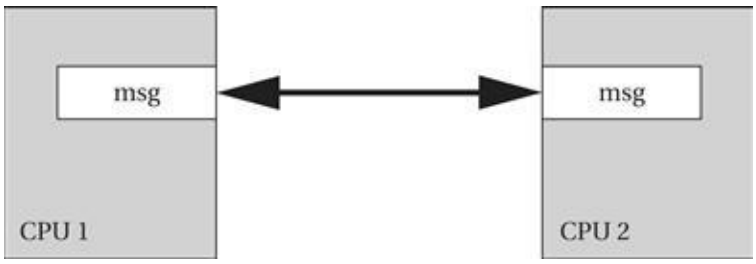


Figure 3.4 Message passing between two CPUs

- Each communicating entity has its own message send/receive unit; the message is stored at the sender/receiver endpoints, **not** on the link. (In shared memory, all data are stored in the link/memory.)
- Units that operate relatively autonomously are natural candidates, e.g. a home control system with one microcontroller per device (lamp, thermostat, faucet, appliance): they communicate infrequently and are physically far apart, so passing packets is a natural way to coordinate.
- Message passing is the natural implementation of communication in many 8-bit microcontrollers that do not normally operate with external memory.

4. Signals

4.1 Unix Signal

- A simple form of inter-process communication that passes no data beyond the existence of the signal itself.
- Analogous to an interrupt but entirely a software creation: generated by a process and delivered to another process by the OS.

4.2 UML Signal

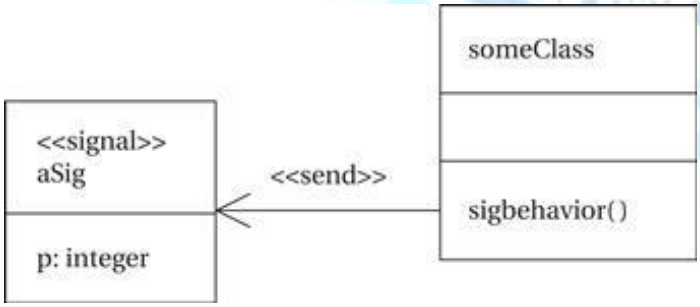


Figure 3.5 Use of a signal in UML

- A generalisation of the Unix signal. A Unix signal carries no parameters other than a condition code, whereas a UML signal is an object and can carry parameters as object attributes.
- In the figure the sigbehavior() behaviour of the class throws the signal (<<send>>); the signal object is marked with the <<signal>> stereotype.

5. Mailboxes

- A simple mechanism for **asynchronous** communication. Some architectures define mailbox registers with a fixed number of bits for small messages; a mailbox can also be built with P() and V() using main memory for storage.

- A simple mailbox holds one message at a time and has two items: the **message** and a **mail-ready flag**.
- The flag is true when a message has been placed in the mailbox and cleared when the message is removed (assuming each message has exactly one recipient).

6. Summary

Mechanism	Key point
Shared memory	Both sides read/write a common memory location
Message passing	Messages held at sender/receiver endpoints; suits autonomous units
Signal	Carries no data (Unix) or an object with attributes (UML)
Mailbox	Asynchronous; message plus mail-ready flag