INTER TASK COMMUNICATION MECHANISM

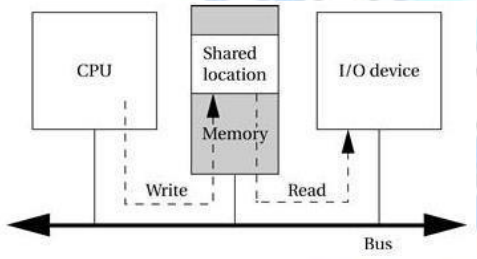
Tasks often need to communicate with each other. **Intertask communication mechanisms** are provided by the operating system as part of the process abstraction.

In general, a process can send a communication in one of two ways: **blocking** or **nonblocking.** After sending 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. Both types of communication are useful.

There are two major styles of intertask communication: **shared memory** and **message passing.** The two are logically equivalent—given one, you can build an interface that implements the other. However, some programs may be easier to write using one rather than the other. In addition, the hardware platform may make one easier to implement or more efficient than the other.

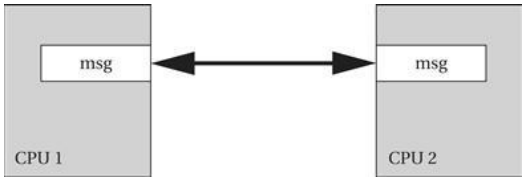
SHARED MEMORY COMMUNICATION

Figure 5.7 illustrates how shared memory communication works in a bus-based system. Two components, such as a CPU and an I/O device, communicate through a shared memory location. The software on the CPU has been designed to know the address of the shared location; the shared location has also been loaded into the proper register of the I/O device. If, as in the figure, the CPU wants to send data to the device, it writes to the shared location. The I/O device then reads the data from that location. The read and write operations are standard and can be encapsulated in a procedural interface.



MESSAGE PASSING

Message passing communication complements the shared memory model. As shown in Figure 5.8, each communicating entity has its own message send/receive unit. The message is not stored on the communications link, but rather at the senders/receivers at the endpoints. In contrast, shared memory communication can be seen as a memory block used as a communication device, in which all the data are stored in the communication link/memory.



Applications in which units operate relatively autonomously are natural candidates for message passing communication. For example, a home control system has one microcontroller per

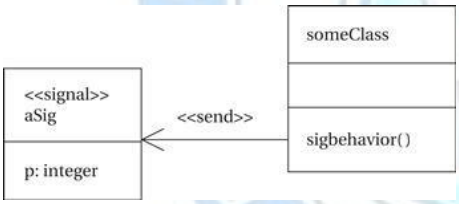
household device—lamp, thermostat, faucet, appliance, and so on. The devices must communicate relatively infrequently; furthermore, their physical separation is large enough that we would not naturally think of them as sharing a central pool of memory. Passing communication packets among the devices is a natural way to describe coordination between these devices. Message passing is the natural implementation of communication in many 8-bit microcontrollers that do not normally operate with external memory.

QUEUES

Another form of interprocess communication commonly used in Unix is the **signal.** A signal is simple because it does not pass data beyond the existence of the signal itself. A signal is analogous to an interrupt, but it is entirely a software creation. A signal is generated by a process and transmitted to another process by the operating system.

SIGNALS

A UML signal is actually a generalization of the Unix signal. While a Unix signal carries no parameters other than a condition code, a UML signal is an object. As such, it can carry parameters as object attributes. Figure 5.9 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. Some architectures define mailbox registers. These mailboxes have a fixed number of bits and can be used for small messages. We can also implement a mailbox using P() and V() using main memory for the mailbox storage. A very simple version of a mailbox, one that holds only one message at a time, illustrates some important principles in interprocess communication.

In order for the mailbox to be most useful, we want it to contain two items: the message itself and a mail ready flag. The flag is true when a message has been put into the mailbox and cleared when the message is removed. (This assumes that each message is destined for exactly one recipient.)