OPERATING SYSTEMS

Module2_Part4

Textbook : Operating Systems Concepts by Silberschatz

Message passing

Here the operating system provides the means for cooperating processes to communicate with

each other via a message-passing facility.

Message passing provides a mechanism

-to allow processes to communicate

-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.

A message-passing facility provides at least two operations:

send(message)

receive(message).

Messages sent by a process can be of either fixed or variable size.

Message passing

- If processes *P* and *Q* want to communicate, they must send messages to and receive messages from each other; a communication link must exist between them.

This link can be implemented in a variety of ways

Direct or indirect communication

Synchronous or asynchronous communication

Automatic or explicit buffering

Message passing

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.

Message passing

indirect communication, the messages are sent to and received from ***mailboxes***, or

ports.

A mail box can be viewed abstractly as an object into which messages can be placed

by processes and from which messages can be removed.

Each mailbox has a unique identification number.

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.

The 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.

Synchronous or asynchronous communication

Blocking(synchronous) or **nonblocking(asynchronous)**

- **Blocking** is considered **synchronous**
  - **Blocking send** -- the sender is blocked until the message is received
  - **Blocking receive** -- the receiver is blocked until a message is available
- **Non-blocking** is considered **asynchronous**
  - **Non-blocking send** -- the sender sends the message and continue
  - **Non-blocking receive** -- the receiver receives:
    - A valid message, or
    - Null message

Message passing

- When both send() and receive() are blocking, we have a **rendezvous** between the sender and the receiver.

The producer–consumer problem

- Producer

```
   message next_produced;
   while (true) {
/* produce an item in next_produced */

    send(next_produced);
   }
```

- Consumer

```
    message next_consumed;
   while (true) {
 receive(next_consumed)

/* consume the item in next_consumed */
  }
```

buffering

- Queue of messages attached to the link.
- Implemented in one of three ways
  1. Zero capacity – no messages are queued on a link.
     Sender must wait for receiver
  2. Bounded capacity – finite length of *n* messages
     Sender must wait if link full
  3. Unbounded capacity – infinite length
     Sender never waits