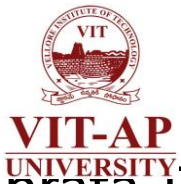
CSE2008: Operating Systems

L9: Interprocess Communication (IPC)



Dr. Subrata Saikadar

SCOPE, VIT-AP University

Recap

- Introductory Concepts
- Process Fundamentals

Outline

- Interprocess Communication (IPC)
  - Types of processes in the context of IPC
  - IPC in Shared-Memory Systems
  - IPC in Message-Passing Systems

Types of processes in the context of IPC

- Independent
  - Does not share data with any other processes executing in the system
- Cooperating
  - Shares data with other processes executing in the system

➢ *A cooperating process can affect or be affected by the other processes executing in the system*

Q: Why do we require an environment that allows process cooperation?

A: many reasons are there; presented in the next slide.

Types of processes in the context of IPC

- Reasons for allowing process cooperation (and IPC)
  - Information sharing
  - Computation speedup
  - Modularity

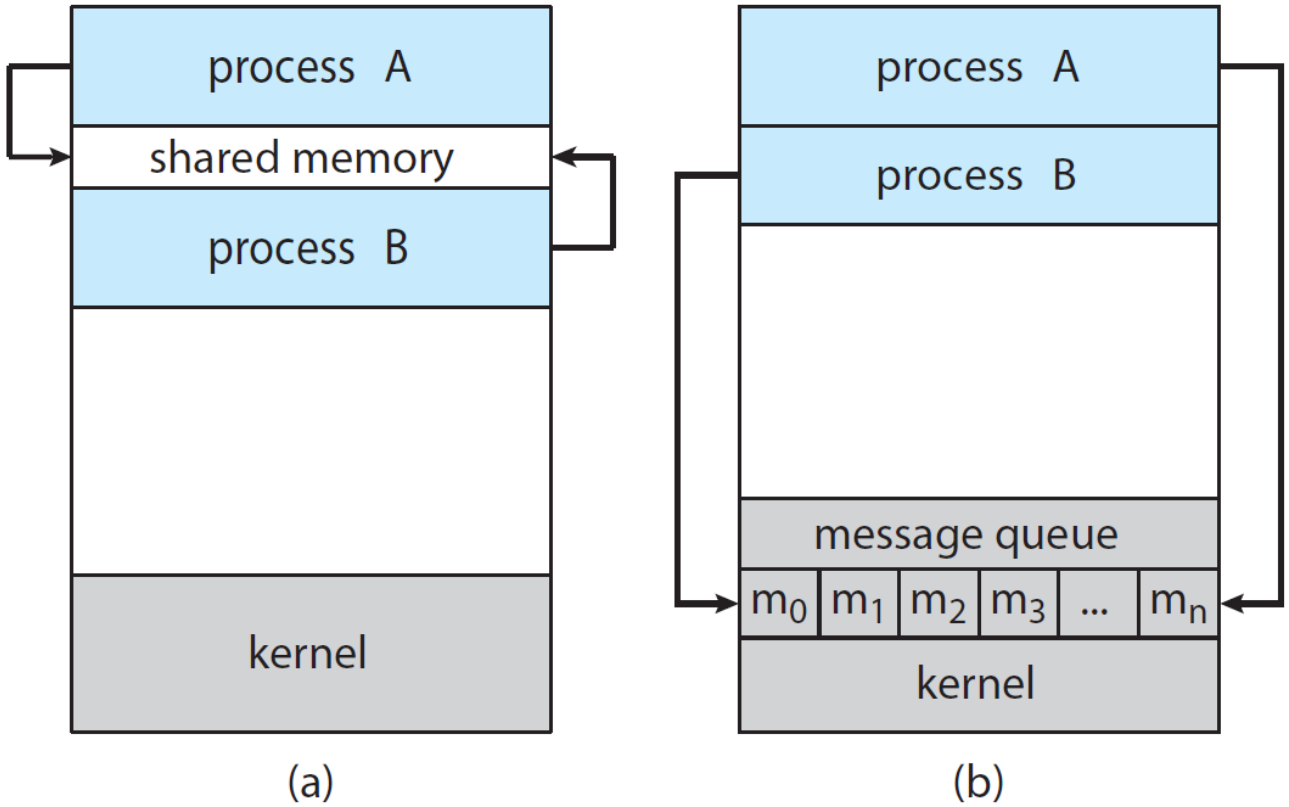
Types of processes in the context of IPC

- Independent
  - Does not share data with any other processes executing in the system
- **Cooperating**
  - **Shares data with other processes executing in the system**

➢ *Cooperating processes require an **interprocess communication (IPC) mechanism** that will allow them to exchange data— that is, send data to and receive data from each other.*

Two fundamental models of IPC

(a) Shared memory, and (b) message passing



IPC in Shared-Memory Systems

- Consider the "producer–consumer" problem: *A producer process produces information that is consumed by a consumer process*.
  - Examples:
    - A compiler may produce assembly code that is consumed by an assembler. The assembler, in turn, may produce object modules that are consumed by the loader.
    - Client–server paradigm - we generally think of a server as a producer and a client as a consumer
      - E.g., a web server produces (that is, provides) web content such as HTML files and images, which are consumed (that is, read) by the client web browser requesting the resource

One Solution → Shared memory – to allow producer and consumer processes to run concurrently, we must have available a buffer of items that can be filled by the producer and emptied by the consumer [*two types of buffers can be used: unbounded buffer/* ***bounded buffer***]

IPC in Shared-Memory Systems

- Bounded buffer for IPC using shared memory
  - variables reside in a region of memory shared by the producer and consumer processes

```
#define BUFFER SIZE 10

typedef struct {
        . . .
} item;

item buffer[BUFFER SIZE];

int in = 0;

int out = 0;
```

```
item next_produced;

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

     while (((in + 1) % BUFFER_SIZE) == out)
        ; /* do nothing */

     buffer[in] = next_produced;
     in = (in + 1) % BUFFER_SIZE;
}
```

```
item next_consumed;

while (true) {
     while (in == out)
        ; /* do nothing */

     next_consumed = buffer[out];
     out = (out + 1) % BUFFER_SIZE;

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

IPC in Message-Passing Systems

- It is particularly useful in a distributed environment, where the communicating processes may reside on different computers connected by a network
  - Example: an **Internet chat program** could be designed so that chat participants communicate with one another by exchanging messages

    Provides at least two operations:

    send(message)

    and

    receive(message)

IPC in Message-Passing Systems

- Several methods for logically implementing a link and the send()/receive() operations
  - Direct or indirect communication
  - Synchronous or asynchronous communication
  - Automatic or explicit buffering

IPC in Message-Passing Systems

VIT-AP UNIVERSITY

- Several methods for logically implementing a link and the send()/receive() operations
  - **Direct** or indirect communication
    - Each process that wants to communicate must explicitly name the recipient or sender of the communication
    - 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.

symmetry in addressing

IPC in Message-Passing Systems

- Several methods for logically implementing a link and the send()/receive() operations
  - **Direct** or indirect communication
    - Each process that wants to communicate must explicitly name the recipient or sender of the communication
    - The send() and receive() primitives are defined as:
      - 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.
    - 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.

asymmetry in addressing

IPC in Message-Passing Systems

VIT-AP UNIVERSITY

- Several methods for logically implementing a link and the send()/receive() operations
  - Direct or **indirect** communication
    - Each process that wants to communicate must explicitly name the recipient or sender of the communication
    - The send() and receive() primitives are defined as:
      - send(A, message)—Send a message to mailbox A.
      - receive(A, message)—Receive a message from mailbox A.
    - A communication link in this scheme has the following properties:
      - 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.

IPC in Message-Passing Systems

- Several methods for logically implementing a link and the send()/receive() operations
  - Synchronous or asynchronous communication
    - **Blocking send.** The sending process is blocked until the message is received by the receiving process or by the mailbox.
    - **Nonblocking send.** The sending process sends the message and resumes operation.
    - **Blocking receive.** The receiver blocks until a message is available.
    - **Nonblocking receive.** The receiver retrieves either a valid message or a null.

```
message next_produced;

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

     send(next_produced);
}
```

```
message next_consumed;

while (true) {
     receive(next_consumed);

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

IPC in Message-Passing Systems

- Several methods for logically implementing a link and the send()/receive() operations
  - Automatic or explicit **buffering**
    - **Zero capacity**

      The queue has a maximum length of zero; thus, the link cannot have any messages waiting in it. In this case, the sender must block until the recipient receives the message.
    - **Bounded capacity.**

      The queue has finite length n; thus, at most n messages can reside in it. If the queue is not full when a new message is sent, the message is placed in the queue (either the message is copied or a pointer to the message is kept), and the sender can continue execution without waiting. The link's capacity is finite, however. If the link is full, the sender must block until space is available in the queue
    - **Unbounded capacity**

      The queue's length is potentially infinite; thus, any number of messages can wait in it. The sender never blocks.

Self Study:

- Examples of IPC Systems
- Communication in Client–Server Systems

Reference

- Abraham Silberschatz , Peter B. Galvin, Greg Gagne, “Operating System Concepts”, Addison Wesley, 10th edition, 2018
  - Chapter 3: Section 3.4 – 3.8

Next

- CPU Scheduling Algorithms

VIT-AP UNIVERSITY

Thank You