Interprocess Communications
Interprocess Communications

Operating systems provide facilities for interprocess communications (IPC), such as message queues, semaphores, and shared memory.

Distributed computing systems make use of these facilities to provide application programming interface which allows IPC to be programmed at a higher level of abstraction.

Distributed computing requires information to be exchanged among independent processes.
In distributed computing, two or more processes engage in IPC in a protocol agreed upon by the processes. A process may be a sender at some points during a protocol, a receiver at other points.

When communication is from one process to a single other process, the IPC is said to be a **unicast**. When communication is from one process to a group of processes, the IPC is said to be a **multicast**, a topic that we will explore in a later chapter.
Unicast vs. Multicast

unicast

multicast

P₁ → P₂

P₁ → P₂ → P₃ → ... → P₄

m

m

m
Interprocess Communications in Distributed Computing

Sender

Data

Receiver
Operations provided in an Interprocess Communications API

- **Receive** ([sender], message storage object)
- **Connect** (sender address, receiver address), for connection-oriented communication.
- **Send** ([receiver], message)
- **Disconnect** (connection identifier), for connection-oriented communication.
Interprocess Communication in basic HTTP

operations:
S1: accept connection
S2: receive (request)
S3: send (response)
S4: disconnect
C1: make connection
C2: send (request)
C3: receive (response)
C4: disconnect
Event Synchronization

Interprocess communication requires that the two processes synchronize their operations: one side sends, then the other receives until all data has been sent and received.

Ideally, the send operation starts before the receive operation.

In practice, the synchronization requires system support.
Synchronous vs. Asynchronous Communication

The IPC operations may provide the synchronization necessary using blocking. A blocking operation issued by a process will block further processing of the process until the operation is fulfilled.

Alternatively, IPC operations may be asynchronous or nonblocking. An asynchronous operation issued by a process will not block further processing of the process. Instead, the process is free to proceed with its processing, and may optionally be notified by the system when the operation is fulfilled.
Synchronous *send* and *receive*

Process 1 running on host 1

- Blocking *send* starts
- Blocking *send* returns

Process 2 running on host 2

- Blocking *receive* starts
- Blocking *receive* ends

Transparent acknowledgment provided by the IPC facility

Execution flow

Suspended period

An operation
Asynchronous *send* and Synchronous *receive*

Process 1 running on host 1

- non-blocking *send* issued

Process 2 running on host 2

- blocking *receive* starts
- blocking *receive* ends

Execution flow

Suspended period

An operation
Synchronous send and Asynchronous receive

Process 1 running on host 1

- Blocking send issued
- Blocking send returns

Process 2 running on host 2

- Non-blocking receive issued

Scenario 1

Execution flow

Suspended period

An operation

Transparent acknowledgment provided by the IPC facility
Synchronous send and Asynchronous receive

Scenario 2

Process 1 running on host 1
blocking send issued
indefinite blocking

Process 2 running on host 2
non-blocking receive issued and returned immediately

Execution flow
Suspended period
An operation
Synchronous **send** and Asynchronous **receive**

Process 1 running on host 1
- Blocking send issued

Process 2 running on host 2
- Non-blocking receive issued and returned immediately
- Process is notified of the arrival of data

Scenario 3
- Transparent acknowledgment provided by the IPC facility

Execution flow
- Process 1 to Process 2
- Suspended period
Asynchronous **send** and Asynchronous receive

- **Process 1 running on host 1**
  - non-blocking send
  - Execution flow
  - Suspended period

- **Process 2 running on host 2**
  - non-blocking receive
  - issued and returned immediately.
  - process is notified of the arrival of the data and the data may now be delivered to it.

**Suspended period**
Event Diagram

Interprocess communication
An operation
Process blocked
Execution flow
Blocking, Deadlock, and Timeouts

Blocking operations issued in the wrong sequence can cause **deadlocks**.

Deadlocks should be avoided. Alternatively, **timeout** can be used to detect deadlocks.
Using threads for asynchronous IPC

When using an IPC programming interface, it is important to note whether the operations are synchronous or asynchronous.

If only blocking operation is provided for send and/or receive, then it is the programmer’s responsibility to use child processes or threads if asynchronous operations are desired.
Deadlocks and Timeouts

Connect and receive operations can result in indefinite blocking.

For example, a blocking `connect` request can result in the requesting process to be suspended indefinitely if the connection is unfulfilled or cannot be fulfilled, perhaps as a result of a breakdown in the network.

It is generally unacceptable for a requesting process to “hang” indefinitely. Indefinite blocking can be avoided by using `timeout`.

Indefinite blocking may also be caused by a `deadlock`.
Indefinite Blocking due to a Deadlock

"receive from process 2" issued; process 1 blocked pending data from process 2.

"receive from process 1" issued; process 2 blocked pending data from process 1.
Data Representation

Data transmitted on the network is a binary stream.

An interprocess communication system may provide the capability to allow data representation to be imposed on the raw data.

Because different computers may have different internal storage format for the same data type, an external representation of data may be necessary.

Data marshalling is the process of (i) flatterning a data structure, and (ii) converting the data to an external representation.

Some well known external data representation schemes are:

- Sun XDR
- ASN.1 (Abstract Syntax Notation)
- XML (Extensible Markup Language)
Data Encoding Protocols

level of abstraction

<table>
<thead>
<tr>
<th>data encoding schemes</th>
</tr>
</thead>
<tbody>
<tr>
<td>application specific data encoding language</td>
</tr>
<tr>
<td>general data encoding language</td>
</tr>
<tr>
<td>network data encoding standard</td>
</tr>
</tbody>
</table>

Sample Standards

- XML (Extensible Markup Language)
- ASN.1 (Abstract Syntax Notation)
- Sun XDR (External Data Representation)