Message Passing Model

Based on original slides by Silberschatz, Galvin and Gagne

Overview

- Message Passing Model
- Addressing
- Synchronization
- Example of IPC systems

Objectives

- To introduce an alternative solution (to shared memory) for process cooperation
- To show pros and cons of message passing vs. shared memory
- To show some examples of message-based communication systems
**Inter-Process Communication (IPC)**

- Message system – processes communicate with each other without resorting to shared variables.
- IPC facility provides two operations:
  - `send(message)` – fixed or variable message size
  - `receive(message)`
- If P and Q wish to communicate, they need to:
  - establish a communication link between them
  - exchange messages via send/receive
- The communication link is provided by the OS

**Implementation Issues**

Physical implementation

- Single-processor system
  - Shared memory
- Multi-processor systems
  - Hardware bus
- Distributed systems
  - Networking System + Communication networks

Logical properties

- Can a link be associated with more than two processes?
- How many links can there be between every pair of communicating processes?
- What is the capacity of a link?
- Is the size of a message that the link can accommodate fixed or variable?
- Is a link unidirectional or bi-directional?
Implementation Issues

Other Aspects
- Addressing
- Synchronization
- Buffering

Overview
- Message Passing Model
- Addressing
- Synchronization
- Example of IPC systems

Direct Addressing
- Processes must name each other explicitly.
- Symmetric scheme
  - send \((D, \text{message})\) – send a message to process \(D\)
  - receive\((S, \text{message})\) – receive a message from process \(S\)
- Logical properties
  - A communication link exits between exactly two processes
  - Links are established automatically
  - Links are usually FIFO
Direct Addressing

- Asymmetric scheme
  - send \((D, \text{message})\) – send a message to process \(D\)
  - receive \((\text{proc}, \text{message})\) - receive a message from any process \(\text{proc}\)

Indirect Addressing

- Messages are sent/received through mailboxes
  - shared data structures where messages are queued temporarily. Sometimes referred to as ports
- Processes can communicate only if they share a mailbox
  - Each mailbox has a unique id
  - Processes can communicate only if they share a mailbox
- Primitives are defined as:
  - send \((mb, \text{message})\) – send a message to mailbox \(A\)
  - receive \((mb, \text{message})\) – receive a message from mailbox \(mb\)

Indirect Communication

- Operations
  - create a new mailbox
  - send and receive messages through mailbox
  - destroy a mailbox
- Properties of communication link
  - Link established only if processes share a common mailbox
  - A link may be associated with many processes
  - Each pair of processes may share several communication links
  - Link may be unidirectional or bi-directional
- Relationships
  - One-to-one (private communication)
  - Many-to-one (client-server communication)
  - Many-to-many (multicast communication)
Overview
- Message Passing Model
- Addressing
- **Synchronization**
- Example of IPC systems

Synchronization
- **Send** operations may be
  - Synchronous
  - Asynchronous
- **Receive** operations may be
  - Blocking
  - Non-blocking

- Blocking send, blocking receive
  - Rendez-vous between sender and receiver

- Non-blocking send, blocking receive
  - Most useful combination (used by servers)
  - Variations: receive with timeout, select, proactive test

- Non-blocking send, Non-blocking receive
  - Neither party is required to wait
**Buffering**

- Queue of messages attached to the link; implemented in one of three ways.
  1. Zero capacity – 0 messages
     Sender must wait for receiver (rendezvous di fatto).
  2. Bounded capacity – finite length of \( n \) messages
     Sender must wait if link full.
  3. Unbounded capacity – infinite length
     Sender never waits.

---

**Producer-Consumer: Solution (1)**

```c
Mailbox mb;

Process Producer {
    while (TRUE) {
        // message in nextProduced
        send(mb, nextProduced);
    }
}

Process Consumer {
    while (TRUE) {
        receive(mb, msg);
        // consume message
    }
}
```

---

**Producer-Consumer: Solution (2)**

```c
Mailbox mb1, mb2;

Process Producer {
    while (TRUE) {
        // message in nextProduced
        receive(mb2, ack);
        send(mb1, nextProduced);
    }
}

Process Consumer {
    while (TRUE) {
        send(mb2, READY);
        receive(mb1, msg);
        // consume message
    }
}
```
Overview

- Message Passing Model
- Addressing
- Synchronization
- Example of IPC systems

Mach

- Mach communication is message based
  - Even system calls are messages
  - Each task gets two mailboxes at creation (Kernel and Notify)
  - Only three system calls needed for message transfer
    - `msg_send()`, `msg_receive()`, `msg_rpc()`
  - Mailboxes needed for communication, created via
    - `port_allocate()`

Windows XP

- Message-passing centric via local procedure call (LPC) facility
  - Only works between processes on the same system
  - Uses ports (like mailboxes) to establish and maintain communication channels
  - Communication works as follows:
    - The client opens a handle to the subsystem’s connection port object
    - The client sends a connection request
    - The server creates two private communication ports and returns the handle to one of them to the client
    - The client and server use the corresponding port handle to send messages or callbacks and to listen for replies
Client-Server Communication

Request

Client

Server

Response

Local Procedure Calls in Windows XP

Questions?