Operating Systems

5. CPU: IPC

Hyunchan, Park

http://oslab.chonbuk.ac.kr

Division of Computer Science and Engineering

Chonbuk National University

Contents

- Concept of IPC
- Two models of IPC: shared memory and message passing
- Message passing: Direct/indirect communications
- Communications in Client-Server Systems
  - Sockets
  - Remote Procedure Calls
  - Pipes

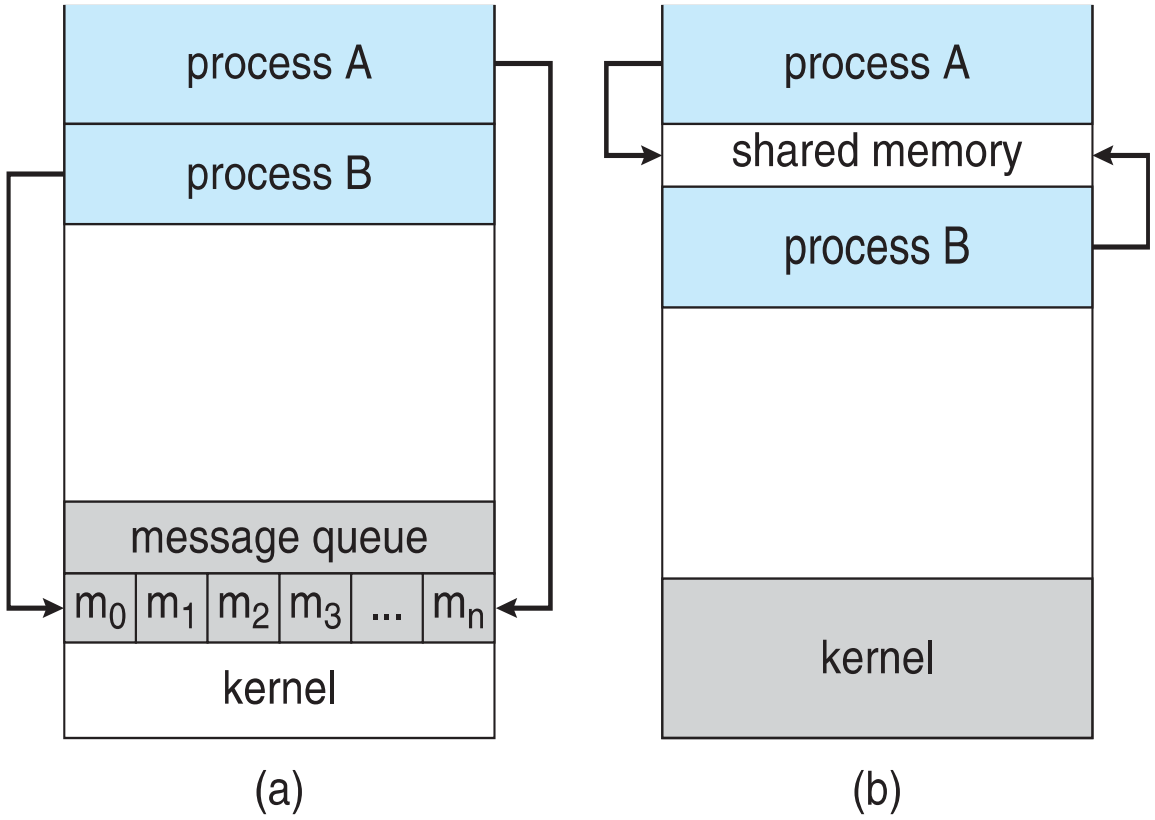
Interprocess Communication

- Processes within a system may be independent or cooperating
  - Independent process cannot affect or be affected by the execution of another process
  - Cooperating process can affect or be affected by other processes, including sharing data
- Reasons for cooperating processes:
  - Information sharing
  - Computation speedup
  - Modularity
  - Convenience
- Cooperating processes need interprocess communication (IPC)

Communications Models

- Two models of IPC

(a) Message passing. (b) shared memory.



IPC Model: Shared Memory

- An area of memory shared among the processes that wish to communicate
- The communication is
  - Under the control of the users processes
  - Not the operating system
- Major issues is to provide mechanism that will allow the user processes to synchronize their actions when they access shared memory.
  - Synchronization is discussed in great details later

IPC Model: Message Passing

- Mechanism for processes to communicate and to synchronize their actions
- Message system – processes communicate with each other without resorting to shared variables
- IPC facility provides two operations:
  - send(message)
  - receive(message)
- The message size is either fixed or variable

Message Passing (Cont.)

- If processes P and Q wish to communicate, they need to:
  - Establish a communication link between them
  - Exchange messages via send/receive
- Implementation issues:
  - How are links established?
  - 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?

Message Passing: Buffering

- Buffer: Queue of messages attached to the link.
- implemented in one of three ways
  - Zero capacity – no messages are queued on a link.
    Sender must wait for receiver (rendezvous)
  - Bounded capacity – finite length of n messages
    Sender must wait if link full
  - Unbounded capacity – infinite length
    Sender never waits

Message Passing (Cont.)

- Implementation of communication link
  - Physical:
    - Shared memory
    - Hardware bus
    - Network
  - Logical:
    - **Direct or indirect**
    - Synchronous or asynchronous
    - Automatic or explicit buffering

Direct Communication

- Processes must name each other explicitly:
  - send (P, message) – send a message to process P
  - receive(Q, message) – receive a message from process Q
- Properties of communication link
  - Links are established automatically
  - A link is associated with exactly one pair of communicating processes
  - Between each pair there exists exactly one link
  - The link may be unidirectional, but is usually bi-directional

Indirect Communication

- Messages are directed and received from mailboxes (also referred to as ports)
  - Each mailbox has a unique id
  - Processes can communicate only if they share 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

Indirect Communication

- Operations
  - create a new mailbox (port)
  - send and receive messages through mailbox
  - destroy a mailbox
- Primitives are defined as:

  send(A, message) – send a message to mailbox A

  receive(A, message) – receive a message from mailbox A

Indirect Communication

- Mailbox sharing
  - P1, P2, and P3 share mailbox A
  - P1, sends; P2 and P3 receive
  - Who gets the message?
- Solutions
  - Allow a link to be associated with at most two processes
  - Allow only one process at a time to execute a receive operation
  - Allow the system to select arbitrarily the receiver.
    - Sender is notified who the receiver was.

Communications in Client-Server Systems

- Signal
- Pipes
- Sockets
- Remote Procedure Calls

Signal

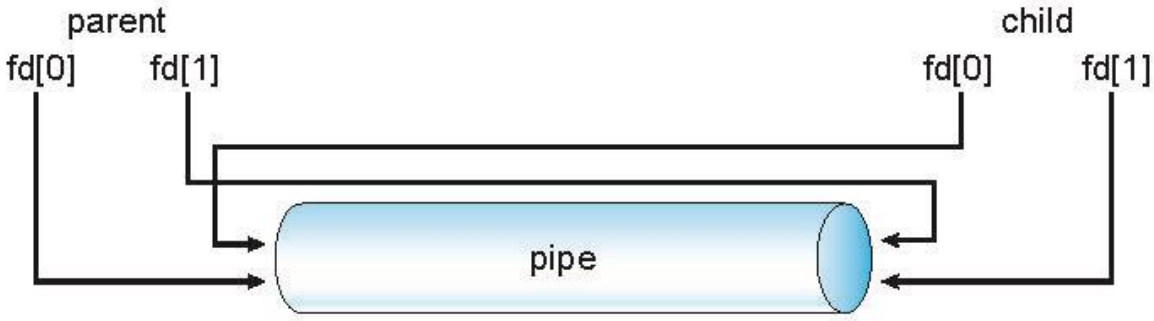
- Signal asynchronous events to one or more processes
  - Oldest IPC method used by UNIX
  - E.g) SIGINT, SIGKILL, SIGTERM, SIGUSR1 …
- A process can choose 4 ways when a signal arrived
  - *Ignore:* can ignore signals except SIGSTOP, SIGKILL
  - *Block: block and process it later*
  - *Handle:* process it immediately with signal handler
  - allow *kernel to handle* it
- Asynchronous IPC
  - When the process A sends a signal to process B,
  - B cannot actually receive the signal until B is scheduled in

Pipes

- Acts as a conduit allowing two processes to communicate
- Issues:
  - Is communication unidirectional or bidirectional?
  - In the case of two-way communication, is it half or full-duplex?
  - Must there exist a relationship (i.e., parent-child) between the communicating processes?
  - Can the pipes be used over a network?
- Ordinary pipes – cannot be accessed from outside the process that created it. Typically, a parent process creates a pipe and uses it to communicate with a child process that it created.
- Named pipes – can be accessed without a parent-child relationship.

Ordinary Pipes

- Ordinary Pipes allow communication in standard producer-consumer style
- Producer writes to one end (the write-end of the pipe)
- Consumer reads from the other end (the read-end of the pipe)
- Ordinary pipes are therefore unidirectional
- Require parent-child relationship between communicating processes



- Windows calls these anonymous pipes

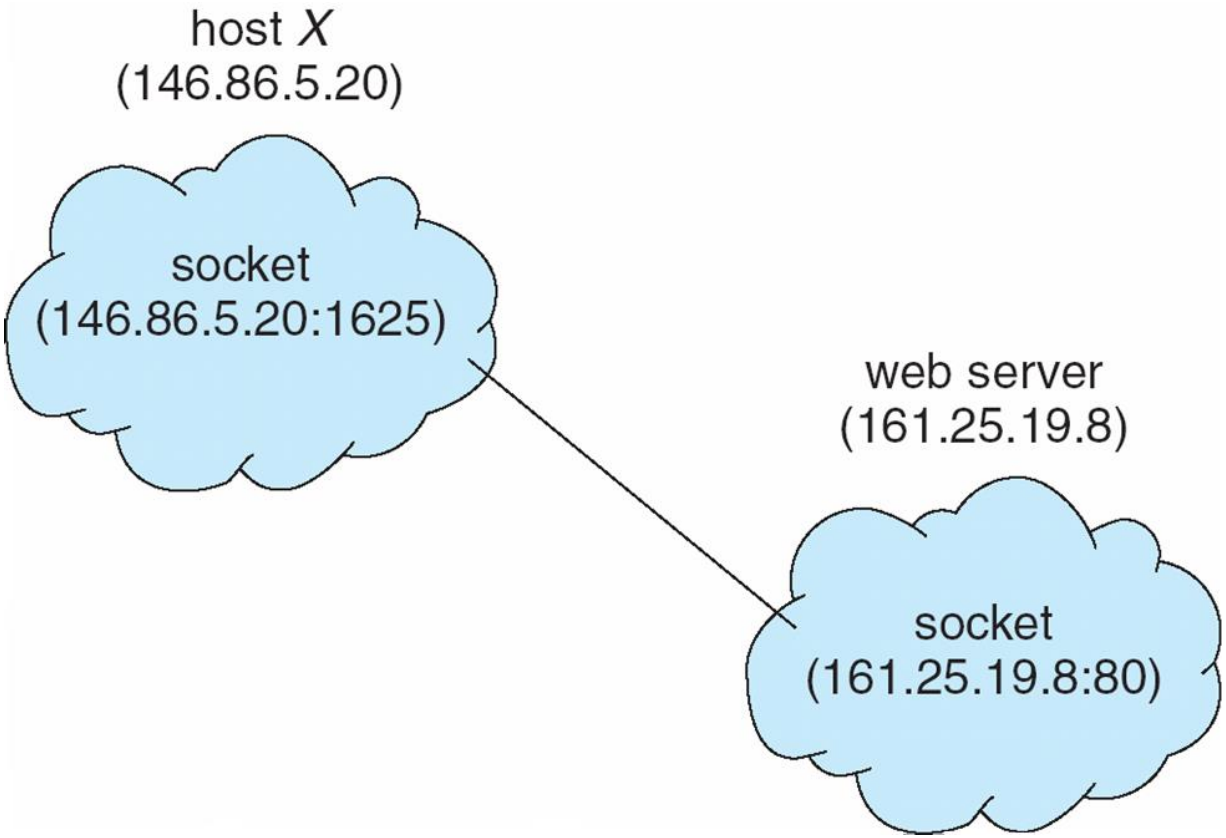
Named Pipes

- Named Pipes are more powerful than ordinary pipes
- Communication is bidirectional
- No parent-child relationship is necessary between the communicating processes
- Several processes can use the named pipe for communication
- Provided on both UNIX and Windows systems

Sockets

- A socket is defined as an endpoint for communication
- Concatenation of IP address and port – a number included at start of message packet to differentiate network services on a host
  - E.g) The socket 161.25.19.8:1625 refers to port 1625 on host 161.25.19.8
- Communication consists between a pair of sockets
- All ports below 1024 are well known, used for standard services
- Special IP address 127.0.0.1 (loopback) to refer to system on which process is running

Socket Communication



Sockets in Java

- Three types of sockets
  - Connection-oriented (TCP)
  - Connectionless (UDP)
  - MulticastSocket class
    - data can be sent to multiple recipients
  - Consider this “Date” server:

```
import java.net.*;
import java.io.*;

public class DateServer
{
   public static void main(String[] args) {
      try {
         ServerSocket sock = new ServerSocket(6013);

         /* now listen for connections */
         while (true) {
            Socket client = sock.accept();

            PrintWriter pout = new
             PrintWriter(client.getOutputStream(), true);

            /* write the Date to the socket */
            pout.println(new java.util.Date().toString());

            /* close the socket and resume */
            /* listening for connections */
            client.close();
         }
      }
      catch (IOException ioe) {
         System.err.println(ioe);
      }
   }
}
```

Remote Procedure Calls

- Remote procedure call (RPC) abstracts procedure calls between processes on networked systems
  - Again uses ports for service differentiation
- Stubs – client-side proxy for the actual procedure on the server
- The client-side stub locates the server and marshalls the parameters
- The server-side stub receives this message, unpacks the marshalled parameters, and performs the procedure on the server

Remote Procedure Calls (Cont.)

- Data representation handled via External Data Representation (XDL) format to account for different architectures
  - e.g. Big-endian and little-endian
- Remote communication has more failure scenarios than local
  - Messages can be delivered exactly once rather than at most once
- OS typically provides a rendezvous (or matchmaker) service to connect client and server