Network

Sanzheng Qiao

Department of Computing and Software

March, 2013

Introduction

Trend:

Large numbers of personal computers on the network.

Advantages:

- price/performance (e.g., Gflops/$M)
- speedup (e.g., parallel computing)
- reliability (e.g., distribute data)
- flexibility (e.g., microkernel)
- fault tolerance

Introduction

Goal:

Get same effect as with timesharing, except lots of CPU power.

Difficulty:

Coordination is more difficult than in centralized system.

Disadvantages:

- less software available
- less security

Examples of networks

- DARPAnet: first widely used network, developed in early 70's, used phone lines, provided mail, file transfer, remote login. still in use.
- Usenet: late 70's, UNIX systems phone each other to send mails and transfer files.
- LAN: early 80's, hook up personal computers. The most popular interconnection for LANs is Ethernet, also token ring
  (10Mbps-100Mbps).
- Internet: tying together existing networks such as DARPAnet, Usenet, LANs.

Example

Broadcast Networks

Broadcast networks use shared communication medium.

Examples: Ethernet (10 Mbits/sec); cellular phones (100Kb–1Mbit/sec).

Mechanisms

- Header on front of packet. Everyone gets packet, discards if not the target. Collision problem: two broadcast same time. Security problem: If you can break into any machine on the network, can eavesdrop (even passwords!)

Example

- Receiver sends an acknowledgement if received ok, discards if not (corrupted). Sender waits for a while, if doesn't get an acknowledgement (timeout), re-sends. Stability problem: heavy load $\rightarrow$ more collision $\rightarrow$ more re-send $\rightarrow$ more load.
  - carrier sense: don't send unless idle
  - adaptive randomized waiting

The Internet

Interconnecting local area networks (e.g., Ethernet, AppleTalk, phone wires)

Routing

Internet has no centralized state. No single machine knows entire topology (topology is constantly changing). Routing tables: Neighbors periodically exchange routing tables, if neighbor has cheaper route, use that one. (cost: number of hops, load of each link)

Point-to-Point Networks

Central idea behind ATM (asynchronous transfer mode), the first commercial point-to-point LAN.

Advantages:

Higer link performance, faster than broadcast link

Lower latency, no need for arbitration to send.

Mechanism:

Switches: inputs, buffers, crossbar (Omega network), buffers, outputs.

Examples

- Multiprocessors hooked together in a 2-D mesh, hypercube.
- Workstations connected to memory and graphics engine by a switched network, instead of a bus.

Network Protocols

Conventions between the parties on the network about how information will be transmitted between them.

Example: system calls are protocol between user programs and operating system.

Layering structure

ISO OSI (Open System Interconnect) Model

layers and transmitting units

7	application	message
6	presentation	message
5	session	message
4	transport	message
3	network	packets
2	data link	frames
1	physical	bits

Example: nachos

user

post office

network

physical simulated by sockets

Example: nachos

User level
threads/kernel.cc NetworkTest

Compose a mail: From mailbox 1 to mailbox 0 at farhost
postOfficeOut$->$Send(outPktHdr, outMailHdr, data);
Send three pieces to post office

postOfficeIn$->$Receive(0, &inPckHdr, &inMailHdr, buffer);
Receive three pieces from mailbox 0

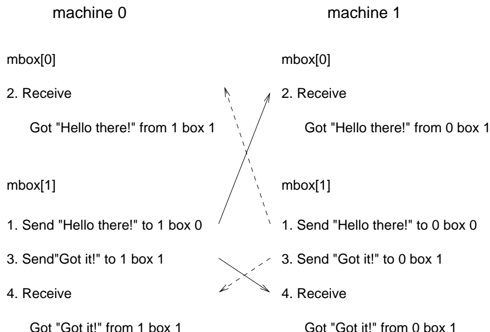
Send an acknowledgement to farhost mailbox 1
Receive acknowledgement from mailbox 1

Example: nachos

User level

nachos -N -m 0	nachos -N -m 1
Got: Hello there! : from 1, box 1	Got: Hello there! : from 0, box 1
Got: Got it! : from 1, box 1	Got: Got it! : from 0, box 1

Nachos: Nettest



Example: nachos

MailHeader structure (network/post.h)

- to, mailbox
- from, mailbox
- length

PacketHeader structure (machine/network.h)

- to, machine
- from, machine
- length

Example: nachos

Limited mail size, machine/network.h

MaxWireSize = 64

MaxPacketSize = MaxWireSize - sizeof(PacketHeader)

MaxMailSize = MaxPacketSize - sizeof(MailHeader)

The user fills

mailHdr.to, mailHdr.from, mailHdr.length
pktHdr.to

Example: nachos

Post office level

- Synchronizing with the network level
- Assuming reliable network for now

PostOfficeOutput, network/post

- Only one message can be sent to the network at any time (lock)
- Block the sender until the network is ready for the next message (semaphore)
- CallBack tells the network what to do when it is ready for the next message (semaphore V)

Example: nachos

PostOfficeInput, network/post

- An array of mail boxes, each of which is a synchList of mails
- A helper (a thread), which is blocked until it is signaled when a message arrives from the network (semaphore)
- CallBack tells the network what to do when a message arrives (semaphore V)

Example: nachos

Network level

NetworkOutput, machine/network

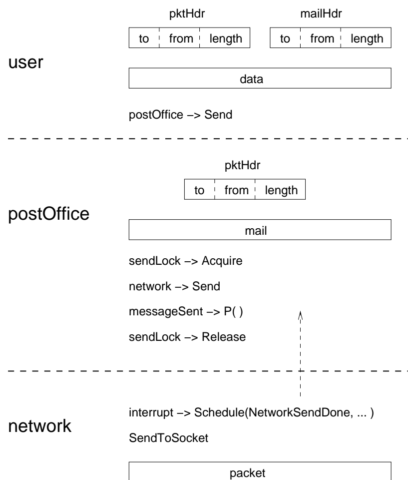
- A network with specified reliability
- Connection to physical level (implemented by socket)
- CallBack defines the network send interrupt handler, that is, what to do when a network send interrupt occurs (PostOfficeOutput CallBack, i.e., semaphore V the sender)
- Send schedules a network send interrupt; randomly drop packets (reliability); send one piece to the physical level (socket)

Example: nachos

NetworkInput, machine/network

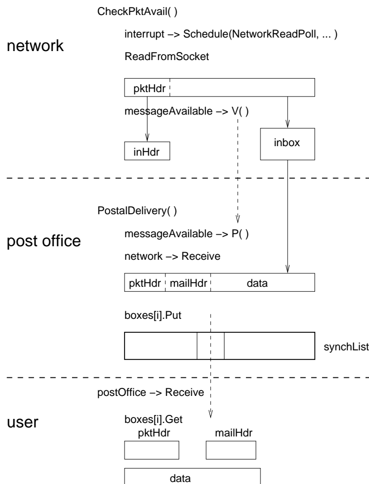
- Connection to physical level (socket)
- CallBack defines the network receive interrupt handler, that is, what to do when a network receive interrupt occurs
  - Schedules next network receive interrupt to poll for a packet
  - Polls a packet, if arrived, puts it in buffer
  - Signals post office (PostOfficeInput CallBack, i.e., semaphore V the post helper)
- Receive reads a packet in the buffer

Nachos: Send



Nachos: Receive

Event driven



OSI model

Physical layer:

electrical mechanism for transmitting bits.

Data link layer:

checking errors (check sum), getting packets between two directly connected components.

Network layer:

Routing packets from one network to another. Forwarding machines are called gateways. Unreliable (lost, delayed, different speeds, out of order). Basic network protocol: datagram protocols.

OSI model

Transport layer:

guarantee delivery and order. Simple acknowledgement-based protocol: Sender: assign a serial number to each packet, send packet, wait for acknowledgement before sending next packet, if time out resend packet. Receiver: when get a message, send back an acknowledgement; when get an acknowledgement for the current serial number, signal the sender; ignore duplicates; order packets.

OSI model

Session layer:

data exchange and synchronization.

Presentation layer:

convert different data formats.

Application layer:

provide services such as e-mail, FTP, remote command, etc.

OSI model

Physical level:

limited size (checksum), unreliable (lost packets), asynchronous

Application level:

arbitrary size, reliable, synchronous

Fragmentation

Sender splits up message into fixed size packets. Receiver assembles fixed size packets into message.

Reliability

Check packet at receiver via checksum, discard if corrupted.
Receiver acknowledges if received properly.
Timeout at sender. If no acknowledgement, re-send

OSI model

Issues

- If the sender doesn't get an ack, does that mean the receiver didn't get the original message? No. What if ack gets dropped? What if message gets delayed?
- Sender doesn't get ack, re-sends. Receiver gets duplicate messages.
  Acknowledge each?

Solution: put sequence number in packet. Receiver checks for duplicate sequence number, if so, discards.

Sender must hold the message that has not been acknowledged yet.
Receiver must keep track of every message that could be a duplicate.

Approaches

- Alternating bit protocol. One bit sequence number. Send one packet at a time; don't send next packet until ack received. Sender only holds the copy of last packet sent; receiver keeps track of sequence number of last packet received.
  - simple
  - small overhead
  - packets arrive in order
  - poor performance
- Window-based protocol (TCP). Send up to N packets at a time. Receiver can get packets out of order.

TCP: Transmission Control Protocol

Reliable byte stream between two processes on different machines over Internet (read, write, flush).

Fragments byte stream into packets and hands them to IP.

TCP/IP services

- FTP: file transfer protocol
- telnet: remote login
- e-mail: computer mail

Interprocess Communication

Communication link:

Shared memory, hardware bus, network.

Shared address space:

Communicate through global variables shared by threads.

Message passing system:

Communication primitives: send and receive

- Direct communication. One-to-one link. Processes are identified by their ids.

```
send(pid, msgid, msg)
receive(pid, msgid, msg)
```

Interprocess Communication

- Indirect communication. Send/receive messages to/from mail boxes.

```
send(mbx, msg)
receive(mbx, msg)
```

- Synchronization
  - Blocking. The sender (receiver) is blocked until the message is received (available).
  - Unblocking. The sender (receiver) sends (receives) a message and returns immediately. The receiver may receive a null message.

Client-Server System

- NFS: network file system provides the illusion that disks or other devices from one system are directly connected to other systems.
- Remote execution: allow you to request that a particular program be run on a different machine, e.g., RPC, rsh and rexec (UNIX), distributed computing (MPI).
- Name server: keep track of host names and Internet addresses.
- Network-oriented window systems: allow a program to display on a different computer, e.g., X-windows

Well-known ports: /etc/services

Client-Server System

Iterative server:

1. start on the system
2. wait for a request from client
3. receive request
4. serve
5. deliver service to client
6. go to step 2

Client-Server System

Concurrent server:

1. start on the system
2. wait for a request from client
3. receive request
4. fork a child process

5.1. child handles service

5.2. parent goes to step 2

Client:

1. start on the system
2. send request to server
3. receive service

Berkeley sockets

Nachos-4.02: lib/sysdep.*

`OpenSocket()`:

```
int socket(int family,
           int type,
           int protocol);
```

`family`: AF_UNIX or AF_INET
`type`: SOCK_DGRAM or SOCK_STREAM
`protocol`: usually 0
returns a socket id (similar to file descriptor)

Berkeley sockets

AssignNameToSocket(...):

```
int bind(int sockfd,
         struct sockaddr *myaddr,
         int addrlen);
```

`myaddr`: socket address
Note. The system call bind() can be used in both Unix domain and Internet domain, socket address can have different structures. So, type cast is necessary.

Berkeley sockets

Unix domain

```
struct sockaddr_un {
   short sun_family;     /* AF_UNIX */
   char  sun_path[108]; /* path name */
};
```

Berkeley sockets

Internet domain

```
struct in_addr {
    u_long s_addr; /* 32-bit net id */
};

struct sockaddr_in {
    short sin_family; /* AF_INET */
    u_short sin_port; /* 16-bit port number */
    struct in_addr  sin_addr;
    char sin_zero[8]; /* unused */
};
```

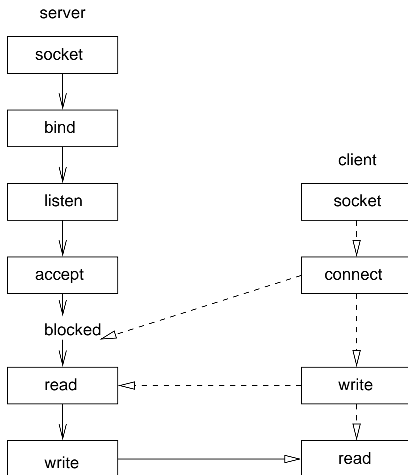
A client-server model

A connection-oriented client-server model.

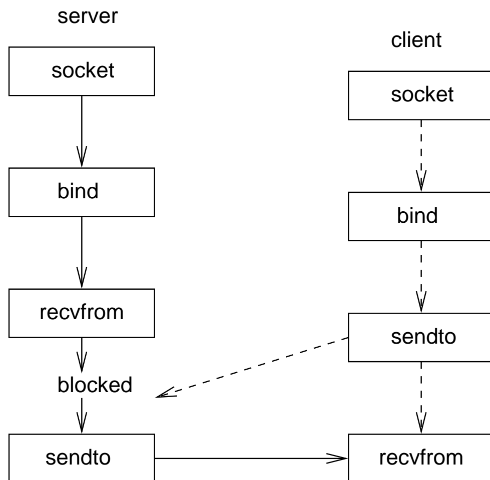
```
int listen(int sockID, int backlog);

int accept(int sockID,
           struct sockaddr *cli_addr,
           int *addrlen);
```

backlog: number of requests that can be queued by the system before the server executes the `accept()` system call. The system call `accept()` returns a new socket descriptor.

A connection-oriented client-server model



A connectionless client-server model



A concurrent server

```
sockID = socket(AF_INET, SOCK_STREAM, 0);
bind(sockID, ...);
listen(sockID, 5);
for( ; ;) {
   newsockID = accept(sockID, ...);
   if (fork() == 0) {
      close(sockID);
        <do whatever using newsockID>
      exit(0);
   }
   close(newsockID);
}
```

Client

After `socket()` and `bind()`

```
int connect(int sockfd,
            struct sockaddr *servaddr,
            int addrlen);
```

Note. A client does not have to bind a local address to the socket descriptor.

Client

Read/write on a stream socket

Similar to file,

```
read(sockID, buf, nbytes);
```

Different from file, read/write on a stream socket might read/write fewer bytes than requested. It is programmer's responsibility to ensure the actual number of bytes are read/written on the socket.