Client – Server Architecture

Programming Interfaces

ARASH HABIBI LASHKARI
(April - 2011)

Sockets

- IP lets us send data between machines
- TCP & UDP are *transport layer* protocols
  - Contain **port number** to identify transport endpoint (application)
- One popular abstraction for transport layer connectivity: **sockets**
  - Developed at Berkeley

Sockets

Attempt at generalized IPC model

Goals:

- communication between processes should not depend on whether they are on the same machine
- efficiency
- compatibility
- support different protocols and naming conventions

Socket

Abstract object from which messages are sent and received

- Looks like a file descriptor
- Application can select particular style of communication
  - Virtual circuit, datagram, message-based, in-order delivery
- Unrelated processes should be able to locate communication endpoints
  - Sockets should be named
  - Name meaningful in the communications domain

Programming with sockets

Step 1

Create a socket

```
int s = socket(domain, type, protocol)
```

AF_INET

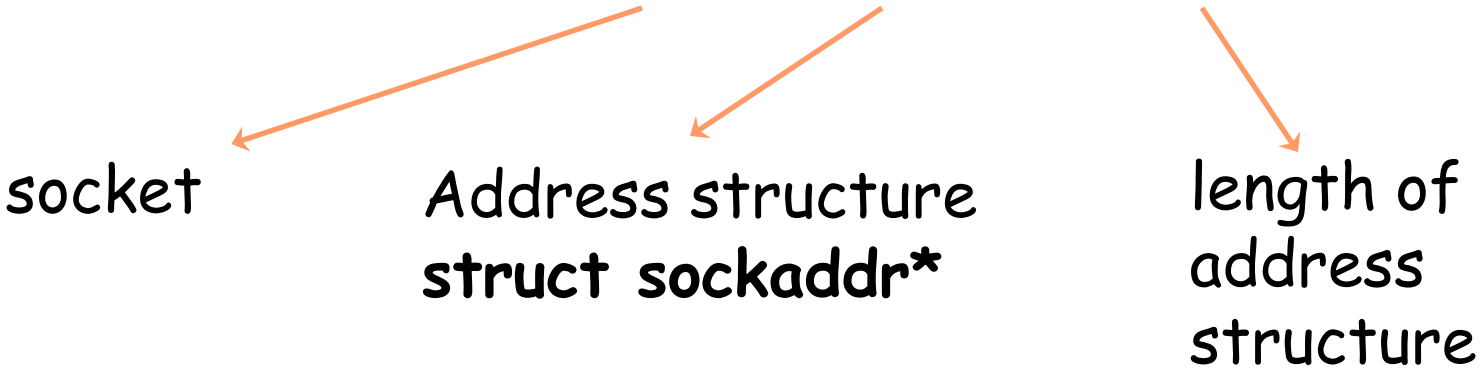
SOCK_STREAM
SOCK_DGRAM

useful if some families have more than one protocol to support a given service

Step 2

Name the socket (assign address, port)

```
int error = bind(s, addr, addrlen)
```



Step 3a (server)

Set socket to be able to accept connections

```
int error = listen(s, backlog)
```

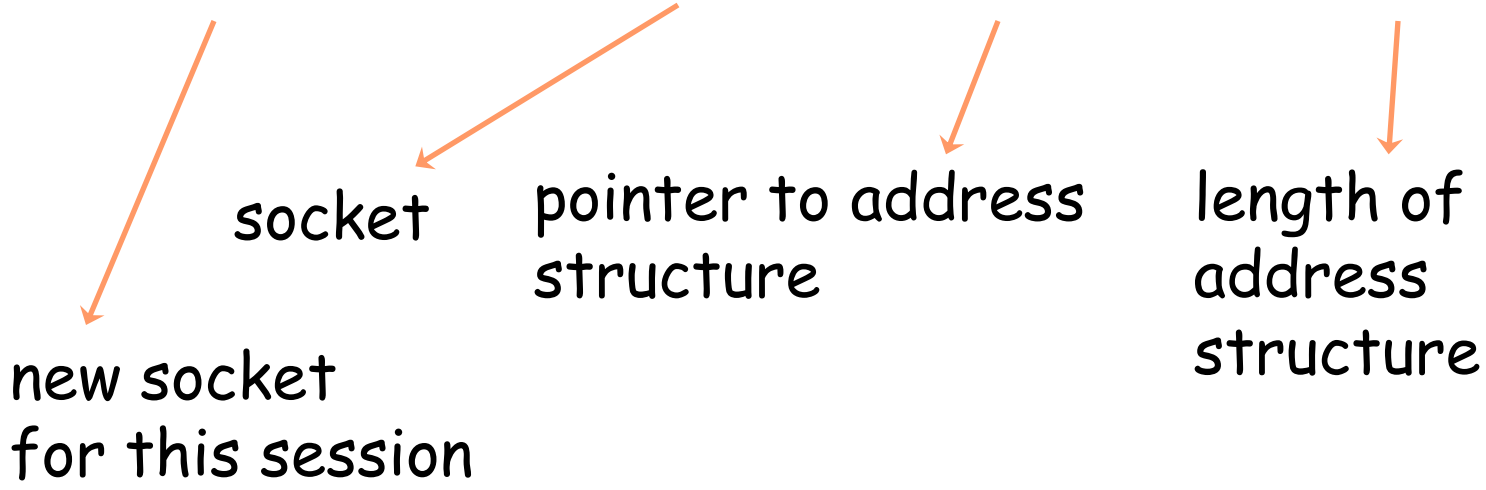
socket

queue length for
pending connections

Step 3b (server)

Wait for a connection from client

```
int snew = accept(s, clntaddr, &clntalen)
```



Step 3 (client)

Connect to server

```
int error = connect(s, svraddr, svraddrlen)
```

socket

Address structure
struct sockaddr*

length of address structure

Step 4

Exchange data

Connection-oriented

`read/write`

`recv/send` *(extra flags)*

Connectionless

`sendto`, **`sendmsg`**

`recvfrom`, **`recvmsg`**

Step 5

Close connection

shutdown(s, *how*)

how:
0: can send but not receive
1: cannot send more data
2: cannot send or receive (=0+1)

Sockets in Java

java.net package

Two major classes:

- **Socket**: client-side
- **ServerSocket**: server-side

Step 1a (server)

Create socket and name it

```
ServerSocket svc =
              new ServerSocket(port)
```

Step 1b (server)

Wait for connection from client

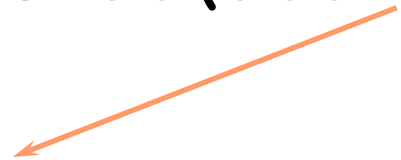
```
Server req = svc.accept()
```

new socket for client session

Step 1 (client)

Create socket and name it

```
Socket s = new Socket(address, port);
```



obtained from:
getLocalHost, getByName,
or getAllByName

```
Socket s =
    new Socket("cs.rutgers.edu", 2211);
```

Step 2

Exchange data

obtain InputStream/OutputStream from Socket object

```
BufferedReader in =
      new BufferedReader(
            new InputStreamReader(
                  s.getInputStream()));
PrintStream out =
      new PrintStream(s.getOutputStream());
```

Step 3

Terminate connection

close streams, close socket

```
in.close();
out.close();
s.close();
```

Socket Internals

Protocol Control Block

Client only sends data to {machine, port}

How does the server keep track of simultaneous sessions to the same {machine, port}?

OS maintains a structure called the
Protocol Control Block (PCB)

Server: `svr=socket()`

Create entry in PCB table

Local addr	Local port	Foreign addr	Foreign port	L?	Client

Server	Local addr	Local port	Foreign addr	Foreign port	L?
svr					

Server: `bind(svr)`

Assign local port and address to socket
bind(addr=0.0.0.0, port=1234)

Local addr	Local port	Foreign addr	Foreign port	L?

Client

Server	Local addr	Local port	Foreign addr	Foreign port	L?
svr	**0.0.0.0**	**1234**			

Server: `listen(svr, 10)`

Set socket for listening

Local addr	Local port	Foreign addr	Foreign port	L?

Client

Server	Local addr	Local port	Foreign addr	Foreign port	L?
svr	**0.0.0.0**	**1234**			*****

Server: `snew=accept(svr)`

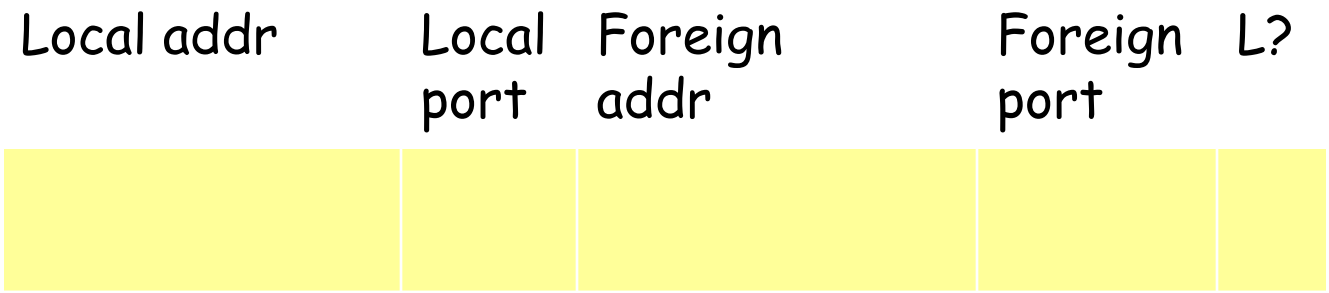
Block - wait for connection

Local addr	Local port	Foreign addr	Foreign port	L?	Client

Server	Local addr	Local port	Foreign addr	Foreign port	L?
svr	**0.0.0.0**	**1234**			*****

Client: `s=socket()`

Create PCB entry



Server	Local addr	Local port	Foreign addr	Foreign port	L?
svr	0.0.0.0	1234			*

Client: `s=bind(s)`

Assign local port and address to socket
bind(addr=0.0.0.0, port=7801)

Local addr	Local port	Foreign addr	Foreign port	L?
0.0.0.0	**7801**			

Client

s

Server	Local addr	Local port	Foreign addr	Foreign port	L?
svr	**0.0.0.0**	**1234**			*****

Client: `connect(s)`

Send *connect* request to server

[135.250.68.3:7801] to [192.11.35.15:1234]

Local addr	Local port	Foreign addr	Foreign port	L?
0.0.0.0	**7801**			

Client

s

Server	Local addr	Local port	Foreign addr	Foreign port	L?
svr	**0.0.0.0**	**1234**			*
snew	**192.11.35.15**	**1234**	**135.250.68.3**	**7801**	

Client: `connect(s)`

Server responds with acknowledgement

[192.11.35.15:1234] to [135.250.68.3 :7801]

Client

0.0.0.0	7801	192.11.35.15	1234	

s

Server

svr

snew

Communication

Each message from client is tagged as either *data* or *control* (e.g. *connect*)

If data – search through table where FA and FP match incoming message and *listen*=false

If control – search through table where *listen*=true

Server

svr

snew

Questions