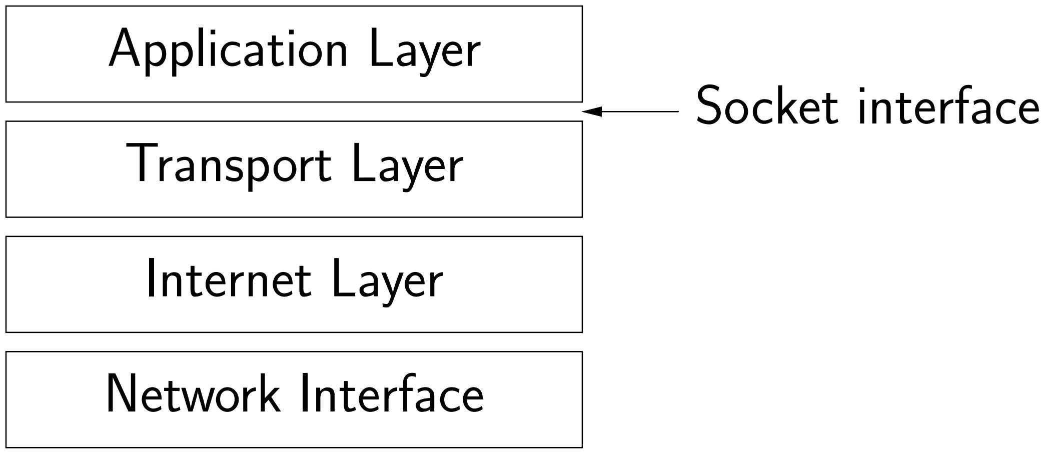
Chapter 5
Socket Interface
Overview of Berkeley Unix Sockets



An interface to the transport layer.

An end-point of network communication.

Unix file style (open–read/write–close; untyped)

Client/Server startup.

Select communications style
(connection oriented/connectionless).

Transport layer can be: TCP/IP, OpenTransport,
OSI, "Unix",...

Complicated (Internet layer shows through)

Sockets are Unix, not TCP/IP

Socket Types

Stream

— Sequenced, reliable, bi-directional delivery of bytes

— A telephone connection

— Transmission Control Protocol: TCP

Datagram

— Unreliable, unsequenced message

— Sending a letter

— User Datagram Protocol: UDP

Raw

— User defines the protocol

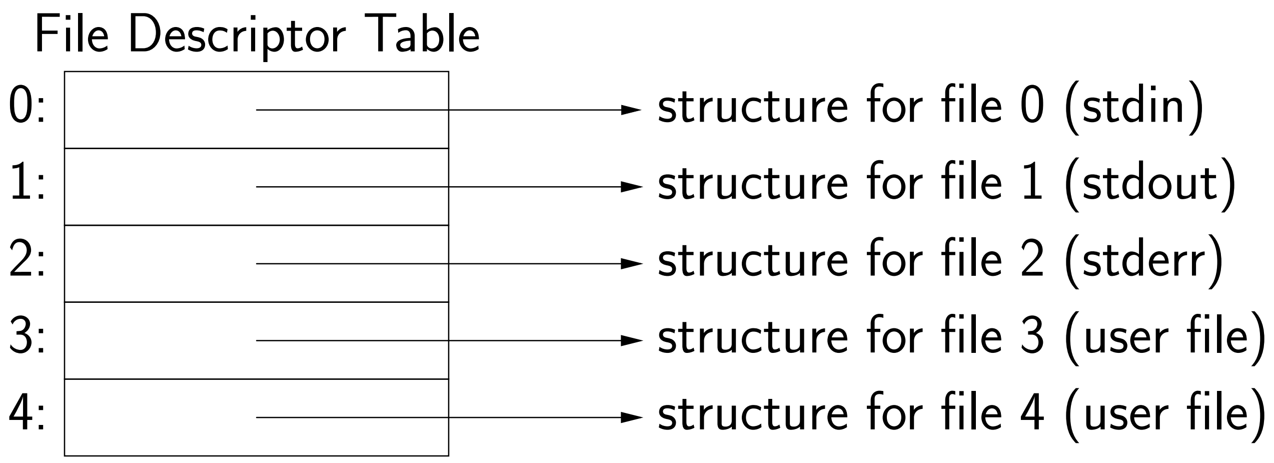
— Lumber supplied for construction

— Interface with IP, must be root

Unix file descriptors

Sockets use file descriptors:

Unix Descriptor Structure


File Descriptor Table
0:
1:
2:
3:
4:
structure for file 0 (stdin)
structure for file 1 (stdout)
structure for file 2 (stderr)
structure for file 3 (user file)
structure for file 4 (user file)

Unix: a file descriptor is an index (`int`) in to the table

The table contains a pointer to the file/socket/pipe data structure. Null pointer indicates no descriptor/file.

Typical socket data structure
(exact contents depend on socket type):

Protocol type:	INET
Connection type:	Sock_Stream
Local Machine:	cheetah
Remote Machine:	lynx

Descriptor Table and Unix Processes

Each process has its own descriptor table.

– each process can have its own files open.

fork: both parent and child get a copy of the variables

– both have a copy of the descriptors

– if you have a file/socket/pipe open when you fork, both processes have it open afterward

– one of the processes probably wants to close its access to the file/socket/pipe

Generic terminology: close the descriptor

End of file (*warning*): A reader will not see end of file unless all potential writers have closed the descriptor. (Including the reader!)

Ports

Each machine provides serveral services.

Problem: Addressing a remote service.

Solution: Assign each service a 16-bit number (port).

Well-Known Ports: Identifiers fixed `/etc/services`
ftp, telnet, www, login

It doesn't matter where you telnet from.

Internet reserves ports 0–512
Unix reserves ports 513–1024
Available 1025–65535

`getservbyname` given a service name looks up the port number

port numbers are found in `/etc/services`

An internet connection is between a machine/port pair and another machine/port pair.

Socket Structure

Sockets can interface to TCP/IP, XNS, OSI, ...

`struct sockaddr`: general form of the data structure.
first field indicates the family (TCP/IP, XNS),
remaining fields depend on which family it is

We use internet sockets (TCP/IP)

We use the internet version of `sockaddr`

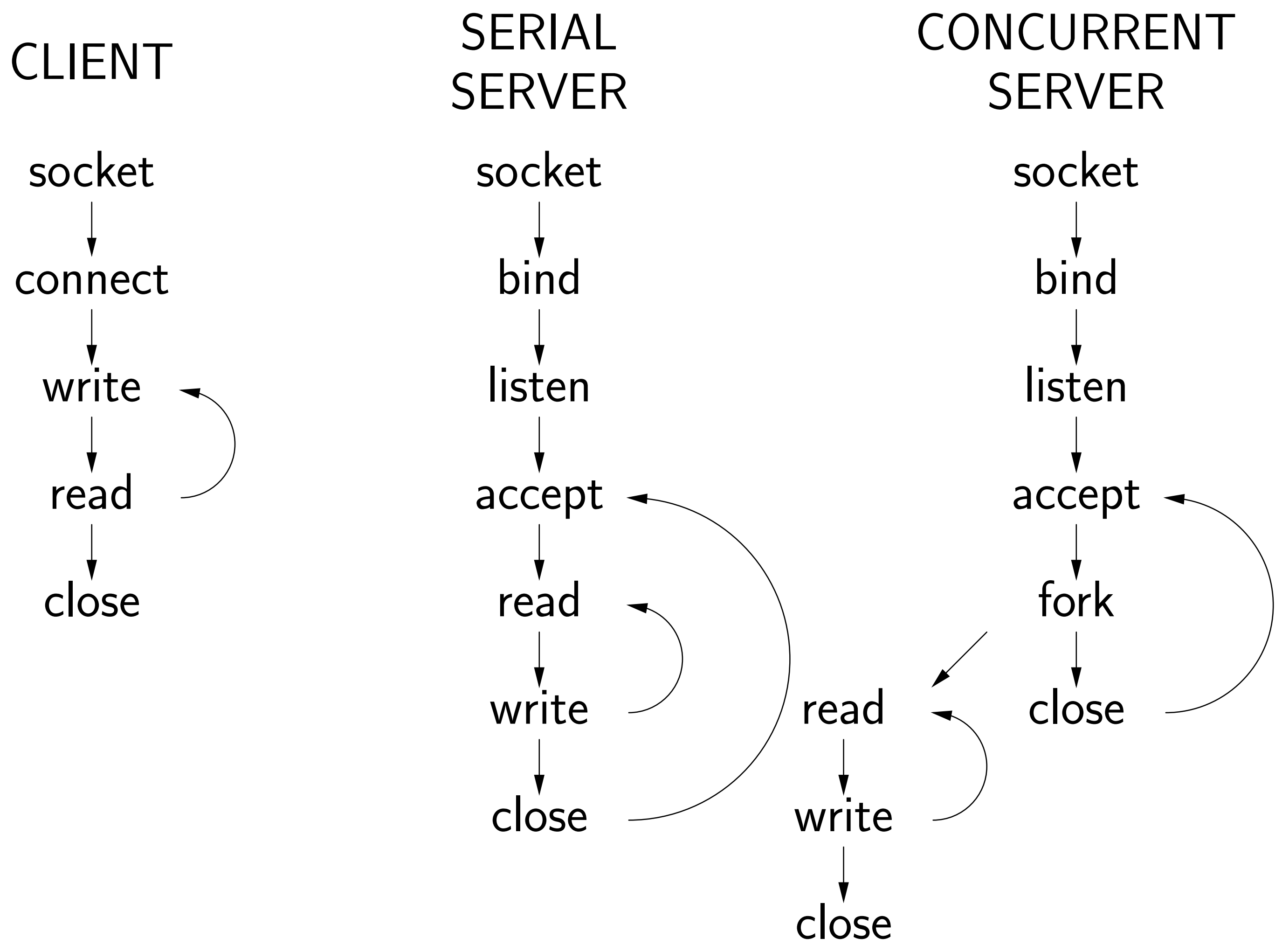
Internet Socket Structure:

```
#include <netinet/in.h>
struct sockaddr_in {
  u_char  sin_len;
  u_short sin_family;
  u_short sin_port;
  struct  in_addr sin_addr;
  char    sin_zero[8];
};
```

`sin_len` total size of the structure
`sin_family` will always be `AF_INET`
`sin_port` is the port number
`sin_addr` indicates the machine (internet number)

```
struct in_addr { _u32 s_addr;};
```

Connection-Oriented Socket Calls Overview



Client Summary

Connect to a service on a server.
Must indicate the server (internet number) and the service (port number).
This creates an open descriptor (i.e., an open socket).

Talk to server (using the descriptor)

Write to server
Read from server

Close the connection (descriptor)

Server Summary

Listen on a well known port waiting for a client to request service.

Accept a connection from a client
This creates a new open descriptor

Talk to the client (fork a child to talk to the client)
Read from client
Write to client

Close the connection

Accept another connection from a client ...

Socket Types

Connection-Orient Server: has two types of sockets.

Master Socket: This socket is used only for accepting connections from clients.
There is one master socket per server.
Master socket is never used for talking to clients.

Slave Socket: A new slave socket is created for each client that is accepted.
There is one slave socket per client.
Slave sockets are used for talking to clients, they are never used for accepting new clients.

Connectionless Server: has only one socket
Socket is used to send and receive messages.

Client: has only one socket

Connection-Oriented Socket Calls Summary

Socket: (client/server)

Creates a socket.

Returns a socket descriptor.

Bind: (server) (Build a master socket.)

Bind (associate) the socket to a port on a machine.

The server uses this to associate itself with a well-known port.

Specify the servers interfaces and port number.

Listen: (server)

Wait for a client to `connect` to this socket.

Listen is only done once on the master socket.

Socket Calls Summary (con'd)

Accept: (server) (Get a slave socket)

Accept a connection from a client.

Returns a new socket associated with this particular client/server connection.

Close: (client/server)

Breaks a client server connection.

Server doesn't close the master socket.

Connect: (client)

Makes a connection to a particular port on a particular machine.

Client uses this to connect to a server.

Specify the servers internet address and port number

Client is assigned any available (unreserved) port number.

Socket Calls Summary (con'd)

Read/Write: (client/server)

Read from or write to a socket that has been connected (by client) and accepted (by server).

If the client writes to the socket, the server can read that data from the slave socket.

If the server writes to the slave socket, the client can read that data from the socket.

Primitive Client

```
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
void main() {
  int s;
  char message[80];
  struct sockaddr_in      srv_addr;
  /* Get a socket. */
  s = socket(PF_INET, SOCK_STREAM, 0);
  /* Put server's address into a socket structure */
  memset(&srv_addr, 0, sizeof(srv_addr));
  srv_addr.sin_addr.s_addr = htonl(0x868bf811);
    /*134.139.248.17*/
  srv_addr.sin_family = AF_INET;
  srv_addr.sin_port = htons(7654);
  /* Request the connection to the server */
  connect(s, (struct sockaddr *) &srv_addr,
    sizeof(srv_addr));
  strcpy(message,"Client speaks");
    /*Send a message to server*/
  write(s, message, 80);
    /*Get server's reply*/
  read(s, message, 80);
  close(s);
}
```

Primitive Server

```
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
void main(){
  int master, client, len; char message[80];
  struct sockaddr_in my_addr, his_addr;
  /* get a socket allocated */
  master = socket(PF_INET, SOCK_STREAM, 0);
  /* bind to the well-known port on our machine */
  memset(&my_addr, 0, sizeof(my_addr));
  my_addr.sin_family = AF_INET;
  my_addr.sin_addr.s_addr = INADDR_ANY;
  my_addr.sin_port = htons(7654);
  bind(master, (struct sockaddr *) &my_addr,
    sizeof(my_addr));
  listen(master, 5);
  len=sizeof(his_addr);
  /* get the connection to the client */
  client = accept(master,
    (struct sockaddr *) &his_addr, &len);
    /* get the message from the client */
  read(client,message,80);
  strcpy(message,"Server replies");
  write(client, message, 80); /* send reply */
  close(client); close(master);
}
```

Connectionless Socket Calls

The client does not connect to the server.

The clients connect, does not do a connect,
it sets up a default destination for write and send and
it restricts the source of any read.

The server does not do an accept,
It does a read.
If the server wants to reply,
it must remember (save) the return address.

read/write are available for connectionless

Special procedures for reading/writing of connectionless sockets:

send, sendmsg, sendto: send a packet/message to another machine
write, send: send to default destination
sendmsg: specify address in header
sendto: specify address in separate parameter

recv, recvmsg, recvfrom: get a packet/message from another machine
read, recv: forget return address
recvmsg: source address is in message header

recvfrom: source address delivered in separate parameter

Other Utility Calls

`getpeername`: who am I talking to

`getsockopt`, `setsockopt`: gives you control over the socket options

`gethostbyname:` find internet numbers from the computer's name.

`getservbyname`: find port number from the services name.

`inet_addr`: converts a dotted IP string into a 4 byte internet number in network order.

`inet_ntoa`: converts an internet adress structure into a dotted IP string.