5. Network Interface
OS Abstraction for Distributed I/O

Challenges

- Abstract multiple *layers* and multiple *networking protocols*
- Cross-system synchronization and communication primitives
- Extend classical I/O primitives to distributed systems
## 5. Network Interface

### Open Systems Interconnection (OSI)

#### Basic Reference Model

<table>
<thead>
<tr>
<th>Layer</th>
<th>Description</th>
<th>Protocols</th>
</tr>
</thead>
<tbody>
<tr>
<td>Layer 7</td>
<td>Application layer</td>
<td>RPC, FTP, HTTP, NFS</td>
</tr>
<tr>
<td>Layer 6</td>
<td>Presentation layer</td>
<td>XDR, SOAP XML, Java socket API</td>
</tr>
<tr>
<td>Layer 5</td>
<td>Session layer</td>
<td>TCP, DNS, DHCP</td>
</tr>
<tr>
<td>Layer 4</td>
<td>Transport layer</td>
<td>TCP, UDP, RAW</td>
</tr>
<tr>
<td>Layer 3</td>
<td>Network layer</td>
<td>IP</td>
</tr>
<tr>
<td>Layer 2</td>
<td>Data Link layer</td>
<td>Ethernet protocol</td>
</tr>
<tr>
<td>Layer 1</td>
<td>Physical layer</td>
<td>Ethernet digital signal processing</td>
</tr>
</tbody>
</table>

#### OS Interface

- Abstract layers 4 and 5 through special files: *sockets*
- Abstract layer 3 in kernel tables (routing, firewall, etc.): *ip, route, iptables, dhclient*
- Abstract layer 2 in kernel network interfaces: *ifconfig*
- Abstract layer 1 in device drivers: *iwconfig, hciconfig*
Socket Abstraction

What?

- Bidirectional communication channel across systems called *hosts*
Socket Abstraction

What?
- Bidirectional communication channel across systems called **hosts**

Networking Domains
- **INET**: Internet Protocol (IP)
- **UNIX**: efficient host-local communication
- And many others (IPv6, X.25, etc.)

- $ man 7 socket
- $ man 7 ip or $ man 7 ipv6 (for INET sockets)
- $ man 7 unix
Socket Abstraction

What?
- Bidirectional communication channel across systems called *hosts*

Socket Types
- **STREAM**: *connected* FIFO streams, reliable (error detection and replay), without message boundaries
- **DGRAM**: *connection-less*, unreliable (duplication, reorder, loss) exchange of messages of fixed length (datagrams)
- **RAW**: direct access to the raw, underlying protocol (not for UNIX sockets)
- Mechanism to *address* remote sockets depends on the socket type
  - $\$ man 7 tcp$ Transmission Control Protocol (TCP): for STREAM sockets
  - $\$ man 7 udp$ User Datagram Protocol (UDP): for DGRAM sockets
  - $\$ man 7 raw$ for RAW sockets
- Two classes of INET sockets
  - IPv4: 32-bit address and 16-bit port
  - IPv6: 128-bit address and 16-bit port
TCP Abstraction: Creation of a Private Channel

- The *listening server* host
  - Create a *server* socket with `new ServerSocket()`
  - Call `accept()` to wait for an incoming connection, returning a new socket associated with a private channel (or “session”) for this connection

- In the *connecting client* host
  - Create and connect a socket: `new Socket(remote_inet, remote_port)`
  - More options possible with the `connect()` method

- The server socket (object of the `ServerSocket` class) can be reused to create more private channels

- Detach a socket connection with `close()`
Establishing a Connection-Based Channel

LOCAL HOST (server socket)  REMOTE HOST (client socket)

Internet
Establishing a Connection-Based Channel

ss = new ServerSocket()
port 80 (HTTP)
Establishing a Connection-Based Channel

ss = new ServerSocket()
port 80 (HTTP)
cs = ss.accept()
Establishing a Connection-Based Channel

ss = new ServerSocket()
port 80 (HTTP)
cs = ss.accept()

REMOTE HOST (client socket)
new Socket()
to 212.27.54.252
port 80

LOCAL HOST (server socket)
212.27.54.252

CONNECTION-BASED COMMUNICATION CHANNEL
Communicating Through a Pair of Sockets

- Stream I/O work as usual *on connected sockets only*
- A connected socket without writer simulates *end-of-file*
- Methods (and specific system calls) to control socket-specific I/O (out-of-band, urgency, message structure, etc.)
Dynamic Thread Creation

1. A *main thread* listens for a *connection* request on a *predefined port*.
2. After *accepting* the request, the server creates a thread to handle the request and immediately resumes listening for another request.
3. The thread performs the request, closes the socket in response to the client’s closing and returns.
Application: Threaded Server Model

Dynamic Thread Creation

1. A main thread listens for a connection request on a predefined port.
2. After accepting the request, the server creates a thread to handle the request and immediately resumes listening for another request.
3. The thread performs the request, closes the socket in response to the client’s closing and returns.

Worker Pool

1. A main thread plays the role of a producer.
2. A bounded number of worker threads play the role of consumers.
3. The main thread listens for connection requests and asks the workers to process them (e.g., with a call-back).