Networking

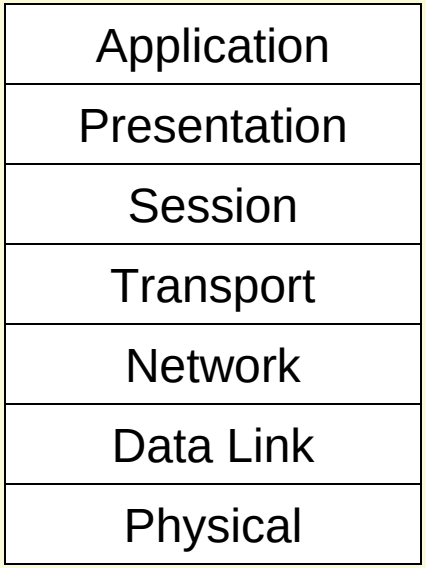
(Stack and Sockets API)

Topic Overview

- Introduction
  - Protocol Models
  - Linux Kernel Support
- TCP/IP
- Sockets
  - Usage
  - Attributes
  - Example of Client / Server
- Socket Structures
- Connection Walk-Through (socket, connect, close)
- Network Files in **`/proc`**
- Advanced Topics
- References

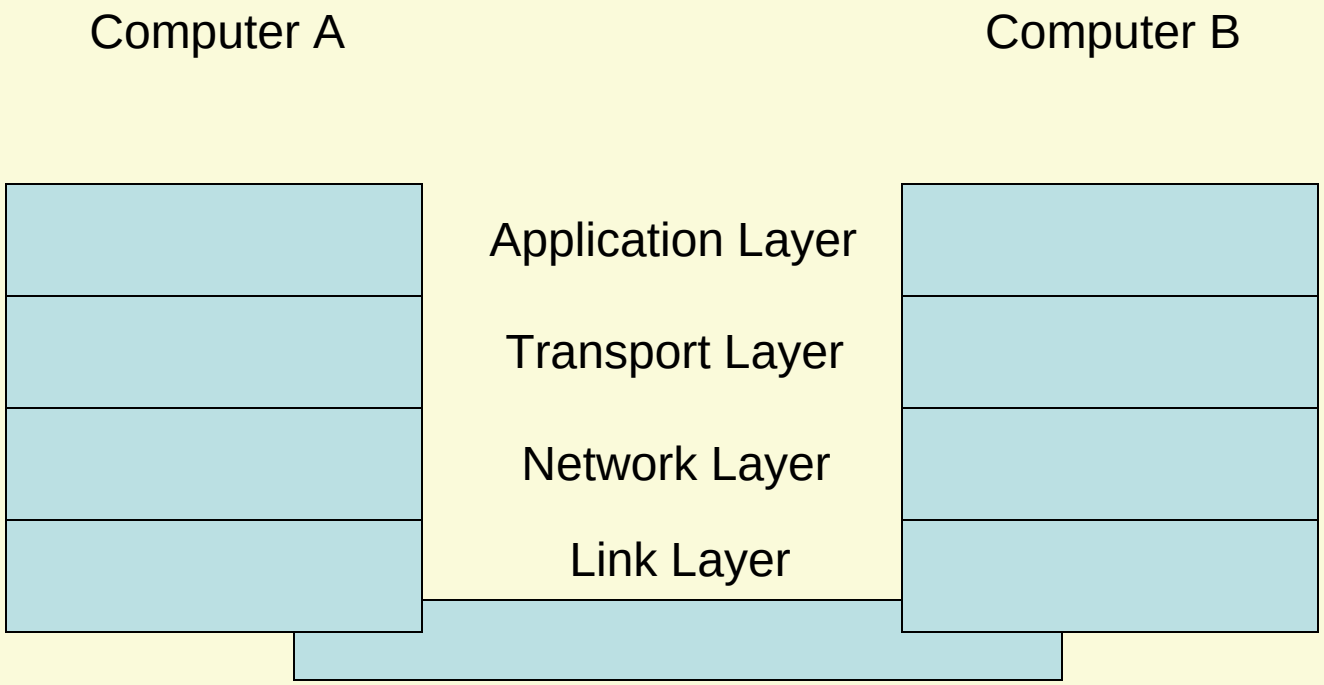
Introduction

- Protocol Models
  - OSI seven-layer model



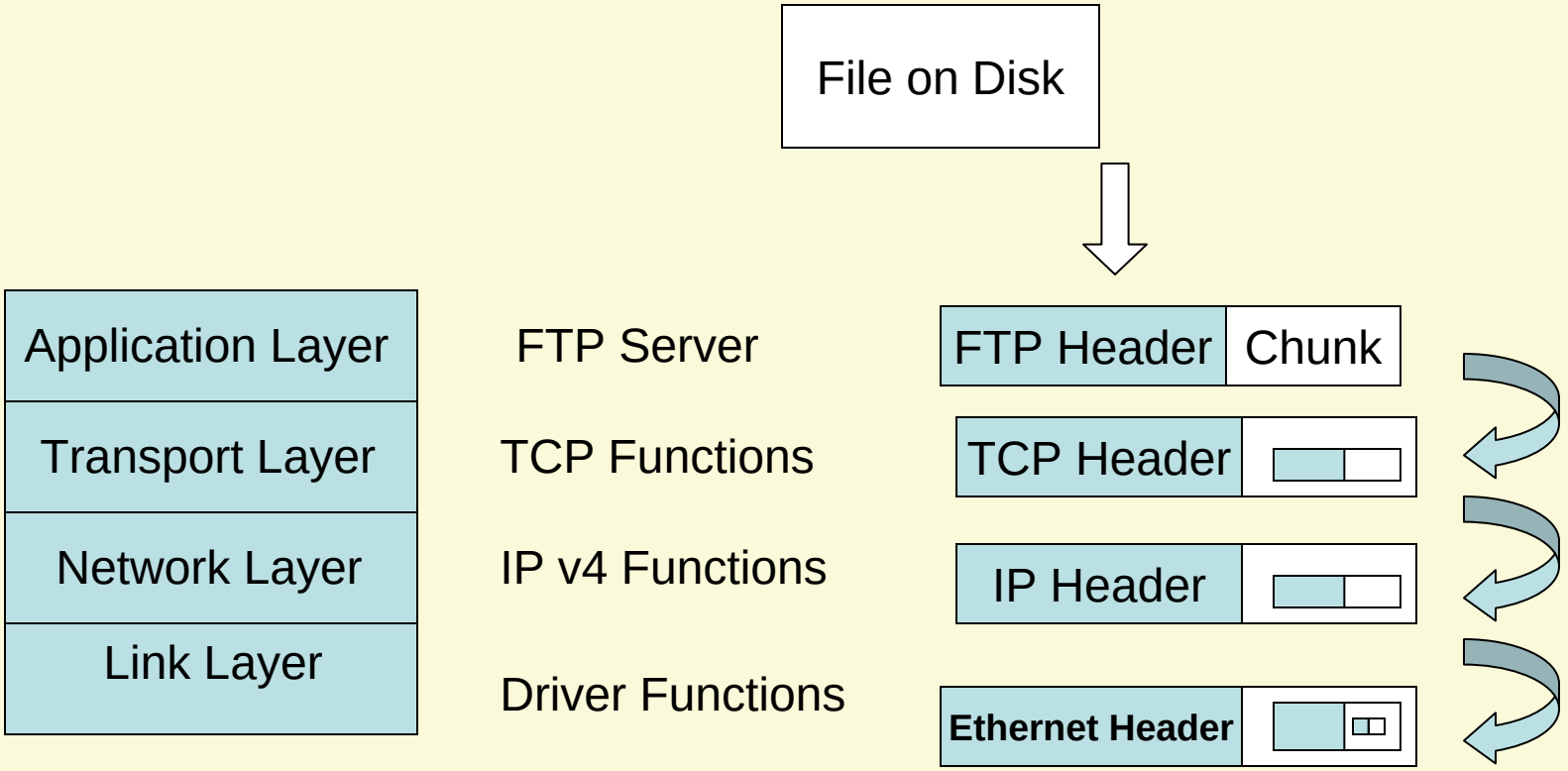
Protocol Models

- TCP/IP Protocol stack

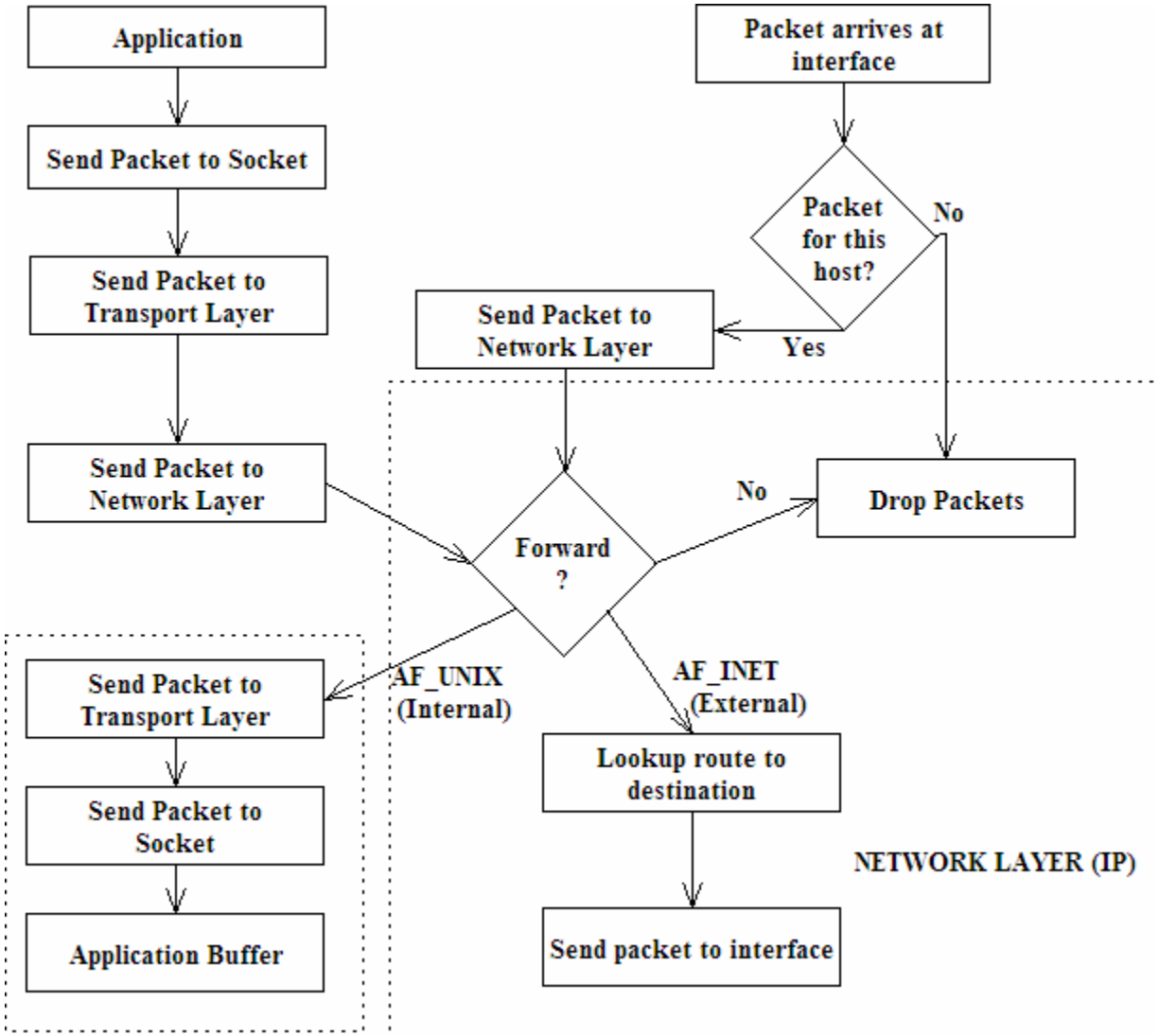


Protocol Models

- Example : File Transfer



TCP/IP : Abstracted View



Linux Kernel Support

- The TCP/IP is the heart of the Linux messaging system.
- BSD Net3 Socket API support
- Supported Network Stacks
  - AppleTalk
  - IPX (Internetwork Packet Exchange)
- IPv6 Support
  - Theoretical maximum of 2^{128} addresses

Sockets

- A ***socket*** is a bidirectional communication **device** that can be used to communicate with another process on the same machine or with a process running on other machines.
- Sockets are created and used differently from pipes.
- An **inode** is associated with each socket. The socket's descriptor is similar to a file descriptor (entry in the descriptor table).
- Defining Sockets

```
int socketFd;
```

Socket Attributes

- Sockets are characterized by three attributes: ***domain***, ***type***, and ***protocol***.

- ***domain****:* Domains specify the network medium that the socket communication will use. Domains are of two types; (Abstracted View)
  - **AF_INET**

    refers to Internet networking (Requires a Network Interface device)

  - **AF_UNIX**

    sockets based on a single computer (No network interface device required)

    eg: X11 Window Server

Socket Attributes

- Socket ***types*** :
  - ***Stream Sockets***
    - Provides a connection that is a sequenced and reliable two-way byte stream.
    - Specified by the type SOCK_STREAM.
    - Eg: TCP
  - ***Datagram Sockets***
    - Doesn’t establish and maintain a connection.
    - Data may be lost, duplicated, or arrive out of sequence.
    - Specified by the type SOCK_DGRAM.
    - Eg: UDP

Socket Attributes

- Socket ***Protocols***
  - Determined by Socket **type** and **domain**
  - Default protocol is 0

Using Sockets

- Server operations
  - Creating a Socket
  - Fill in the Socket Address structure
  - Naming a Socket
  - Creating a Socket Queue
  - Accepting Connections
  - Closing a Socket
- Client operations
  - Requesting Connections

Creating a Socket (2)

- **`socket()`** system call

```
#include <sys/types.h>
#include <sys/socket.h>

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

- socket system call returns a descriptor that is similar to a low-level file descriptor.
- socket created is one end point of a communication channel.

Creating a Socket (3)

- Domains types: Code Example

AF_UNIX	UNIX internal (file system sockets)
AF_INET	ARPA Internet protocols (network sockets)
AF_ISO	ISO standard protocols
AF_NS	Xerox Network Systems protocols
AF_IPX	Novell IPX protocol
AF_APPLETALK	Appletalk DDS

Creating a Socket (4)

- ***Socket Addresses*** - Each socket domain requires its own address format.

- For AF_UNIX socket [sys/un.h]

```
struct sockaddr_un {
       sa_family_t sun_family; /* AF_UNIX */
       char sun_path[]; /* pathname */
};
```

Creating a Socket (5)

- For AF_INET socket [netinet/in.h]

```
struct sockaddr_in {
        short int sin_family; /* AF_INET */
        unsigned short int sin_port; /* Port number */
        struct in_addr sin_addr; /* Internet address */
};
```

```
struct in_addr {
        unsigned long int s_addr;
};
```

Naming a Socket

- AF_UNIX sockets are associated with a file system pathname,
- AF_INET sockets are associated with an IP port number.
- **bind()** system call assigns the address specified in the parameter, address, to the unnamed socket associated with the file descriptor socket.

```
#include <sys/socket.h>

int bind(int socket, const struct sockaddr *address,
                              size_t address_len);
```

Naming a Socket (2)

- **`bind`** return value
  - 0 on success, -1 on failure
  - Error codes

EBADF	The file descriptor is invalid.
ENOTSOCK	The file descriptor doesn’t refer to a socket.
EINVAL	The named file descriptor already exists.
EADDRNOTAVAIL	The address is unavailable.
EADDRINUSE	The address has a socket bound to it already

Creating a Socket Queue

- A server program must create a queue to store pending requests.
- **`listen()`** system call.

```
#include <sys/socket.h>

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

- Allows incoming connections to be held pending while a server program is busy dealing with a previous client.
- Return values same as for bind.

Accepting Connections

- Wait for connections to be made to the socket by using the **`accept()`** system call.
- Creates a new socket to communicate with the client and returns its descriptor.

```
#include <sys/socket.h>

int accept(int socket, struct sockaddr *address,
                                size_t *address_len);
```

Accepting Connections (2)

- By default, accept will block until a client makes a connection.
- O_NONBLOCK flag on the socket file descriptor.

```
int flags = fcntl(socket, F_GETFL, 0);
fcntl(socket, F_SETFL, O_NONBLOCK|flags);
```

- Returns EWOULDBLOCK, where O_NONBLOCK has been specified and there are no pending connections.

Requesting Connections

- **`connect()`** system call.
  - Establishes connection between an unnamed socket and the server listen socket.

```
#include <sys/socket.h>
int connect(int socket, const struct sockaddr *address,
                                size_t address_len);
```

Error Codes:

EBADF	An invalid file descriptor was passed in socket.
EALREADY	A connection is already in progress for this socket.
ETIMEDOUT	A connection timeout has occurred.
ECONNREFUSED	Requested connection was refused by the server.

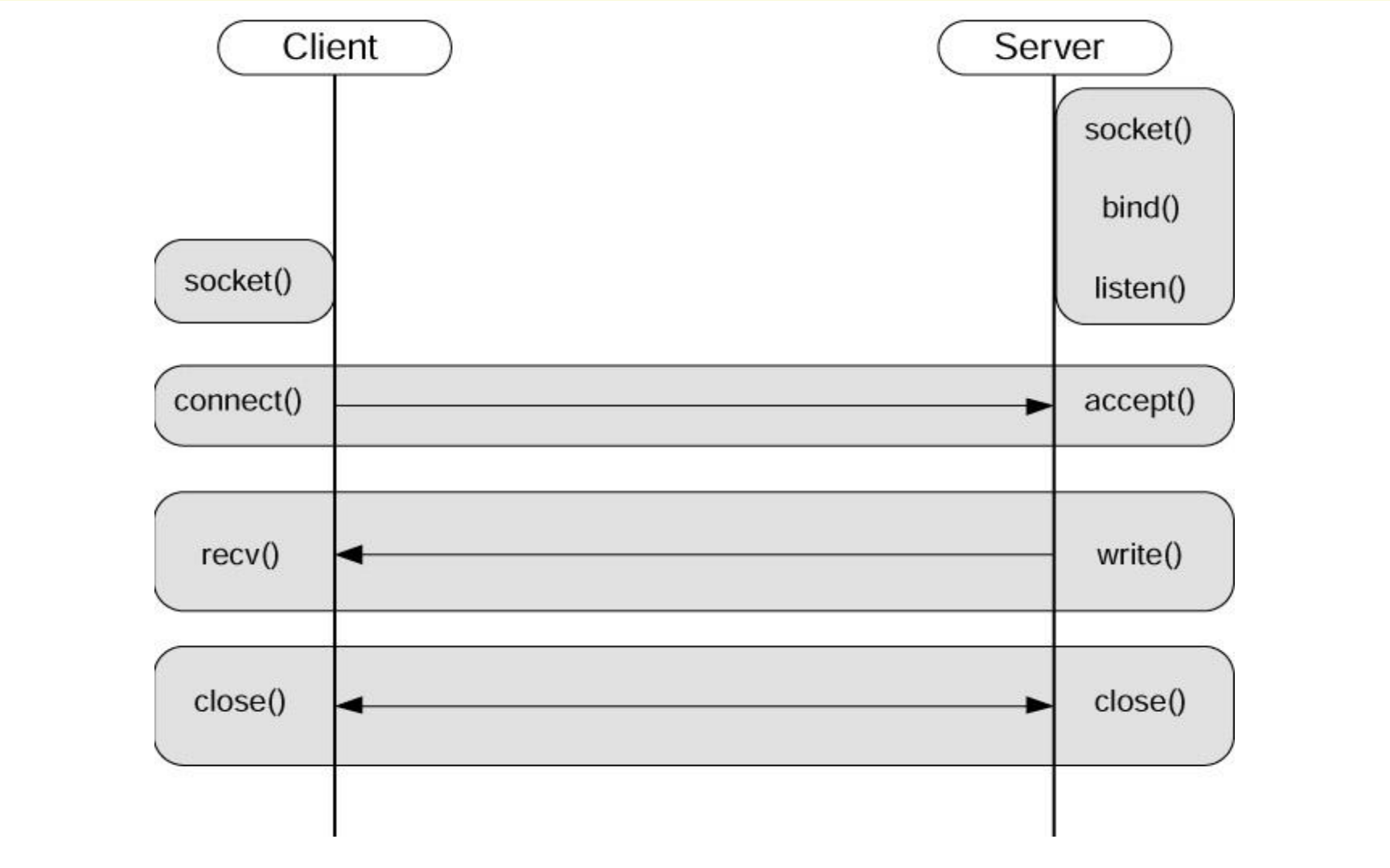
Host and Network Byte Ordering

- Client and server programs must convert their internal integer representation to the network ordering before transmission. They do this by using functions defined in netinet/in.h

Link

```
#include <netinet/in.h>
unsigned long int htonl(unsigned long int hostlong);
unsigned short int htons(unsigned short int hostshort);
unsigned long int ntohl(unsigned long int netlong);
unsigned short int ntohs(unsigned short int netshort);
```

Client-Server Request Flow



Socket Structures

- There are two main socket structures in Linux:
  - general BSD sockets
  - IP specific INET sockets.

  They are strongly interrelated; a BSD socket has an INET socket as a data member and and INET socket has a BSD socket as its owner.

Socket Structures

- BSD sockets are of type **`struct socket`** as defined in ***`include/linux/socket.h`***
  - ***`struct proto_ops *ops`*** - contains pointers to protocol specific functions for implementing general socket behavior.
  - ***`struct inode *inode`*** - this structure points to the file inode that is associated with this socket.
  - ***`struct sock *sk`*** - this is the INET socket that is associated with this socket.

Socket Structures

- INET sockets are of type `struct sock` as defined in *include/net/sock.h*

socket() Call Walk-Through

- Check for errors in call
- Create (allocate memory for) socket object
- Put socket into INODE list
- Establish pointers to protocol functions (INET)
- Store values for socket type and protocol family
- Set socket state to closed
- Initialize packet queues
- **socket()** is a glibc-2.0 library function which ends up calling sys_socket()

connect() Call Walk-Through

- Check for errors
- Determine route to destination:
  - Check routing table for existing entry
  - Look up destination in FIB
  - Build new routing table entry
  - Put entry in routing table and return it
- Store pointer to routing entry in socket
- Call protocol specific connection function
- Set socket state to established

Close connection Walk-Through

- Check for errors (does the socket exist?)
- Change the socket state to disconnecting to prevent further use
- Do any protocol closing actions (e.g., send a TCP packet with the FIN bit set)
- Free memory for socket data structures (TCP/UDP and INET)
- Remove socket from INODE list.

Network Files in */proc/net*

- ***arp***
  - displays the neighbor table (arp_tbl); (arp_get_info() : *net/ipv4/arp.c*)
- ***dev***
  - displays reception and transmission statistics for each registered interface
- ***netstat***
  - displays ICMP statistics (netstat_get_info() : *net/ipv4/proc.c*)
- ***raw***
  - displays address, queue, and timeout information for each open RAW socket from struct proto raw_prot (get__netinfo() : *net/ipv4/proc.c*)
- ***sockstat***
  - displays number of sockets that have been used and statistics on how many were TCP, UDP, and RAW (afinet_get_info() : *net/ipv4/proc.c*)
- ***tcp***
  - displays address, queue, and timeout information for each open TCP socket from struct proto tcp_prot (get__netinfo() : *net/ipv4/proc.c*)
- ***udp***
  - displays address, queue, and timeout information for each open UDP socket from struct proto udp_prot (get__netinfo() : *net/ipv4/proc.c*)

Advanced Topics

- Multiple Clients (**`select`** system call)
- Multithreaded Sockets

References

- The Linux Documentation Project
  - (*http://www.tldp.org*)
- Advanced Linux Programming (GPL)
  - (*http://www.advancedlinuxprogramming.com*)
- TCP/IP Illustrated, Volume 2
  - *by Gary Wright, W. R. Stevens*