IPC Socket

Contents

1. Introduction
2. Types
    - Stream Sockets
    - Sequential Sockets
    - Datagram Sockets
    - Raw Sockets
3. Creating a Socket
4. Binding a Socket
5. Using a Socket
    - Connection
    - Sending Data
    - Receiving Data
    - Other Operations
6. UNIX Socket Domain

Introduction

What is a socket?

- End point of communication
- Used for IPC
  - Within same machine
  - Across networks
- Connection to network protocols (TCP/IP, UDP...)
- Usage is similar to file (through file descriptors)
- This presentation will provide a very condensed explanations of sockets, although this concept is vast

Types

Types of Sockets in UNIX

- Stream Sockets
- Sequential Sockets
- Datagram Sockets
- Raw Sockets

Types

Stream Sockets

- Provides bidirectional, reliable, sequenced, and unduplicated flow of data
- Designed to prevent the loss or duplication of data
- Default protocol in AF_INET domain is TCP
- Requires a connection before communication
- When using a stream socket for data transfer, an application program needs to perform the following sequence:
  i. Create a connection to another socket using the **connect** subroutine.
  ii. Use the **read** and **write** subroutines (or **send** and **recv)** to transfer data.
  iii. Use the **close** subroutine to finish the session.

Types

Sequential Sockets

- Provides sequenced, reliable, and unduplicated flow of information
- Not very frequently used
- Difference between SOCK_STREAM and SOCK_SEQPACKET:

i. SOCK_STREAM is supported by windows and all major versions of Linux but SOCK_SEQPACKET is compatible with only certain builds.

ii. Both SOCK_SEQPACKET and SOCK_STREAM handle backpressure by tossing out the offending packet but:

- SOCK_SEQPACKET returns an error to the sending socket, whereas
- SOCK_STREAM asks for it to be retransmitted when the buffer has space

Types

Datagram Sockets

- Provides unreliable, non-sequenced and datagrams, which are connectionless messages of a fixed maximum length
- Designed for short messages
- Data packets may be duplicated
- Application program to sends datagrams to correspondents named in **send**
- Application programs can receive datagrams through sockets using the **recv**

Types

Raw Sockets

- Provides access to internal network protocols and interfaces
- allow an application to have direct access to lower-level communication protocols
- intended for advanced users who want to take advantage of some protocol feature that is not directly accessible through a normal interface, or who want to build new protocols on top of existing low-level protocols
- Like datagram sockets, these are datagram-oriented.
- Superuser privileges required

Socket Programming with TCP

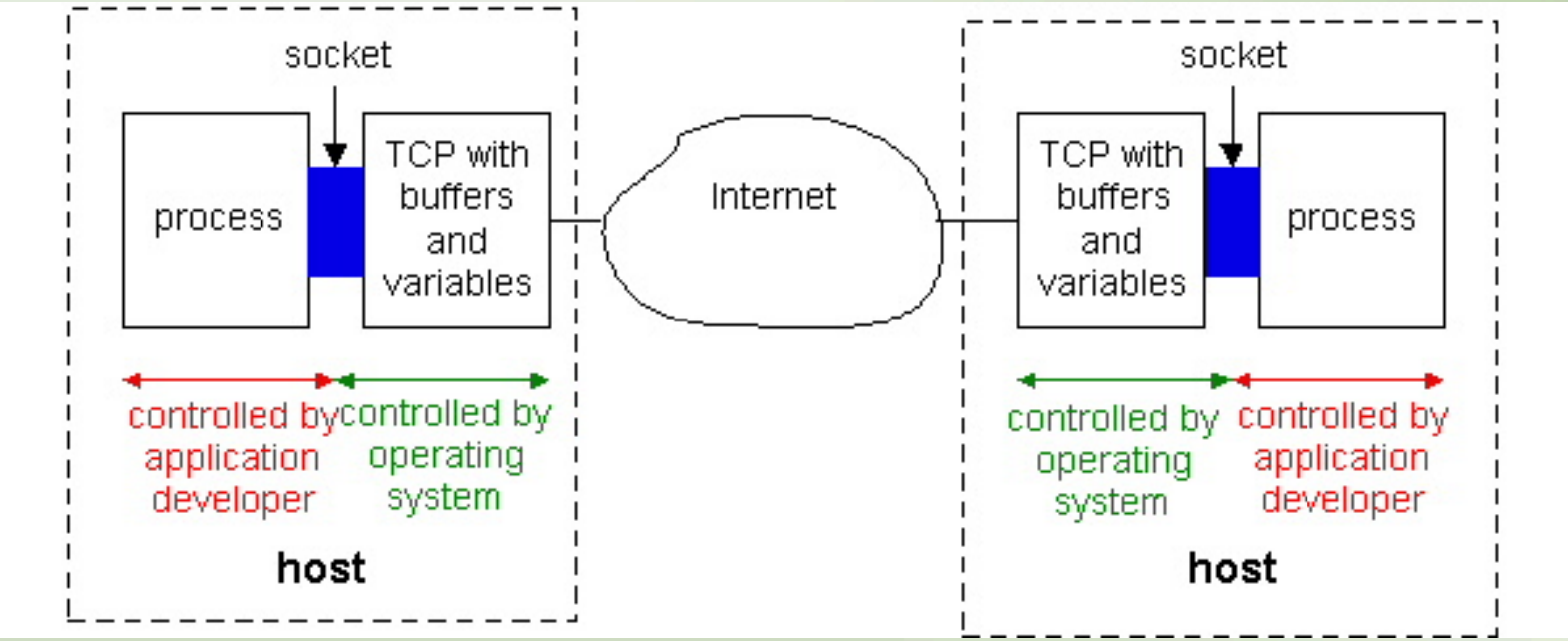


Figure 2.6-1: Processes communicating through TCP sockets♪

The application developer has the ability to fix a few TCP parameters, such as maximum buffer and maximum segment sizes.

Sockets for server and client

- Server
  - Welcoming socket
    - Welcomes some initial contact from a client.
  - Connection socket
    - Is created at initial contact of client.
    - New socket that is dedicated to the particular client.
- Client
  - Client socket
    - Initiate a TCP connection to the server by creating a socket object. (Three-way handshake)
    - Specify the address of the server process, namely, the IP address of the server and the port number of the process.

Socket functional calls

- socket (): Create a socket
- bind(): bind a socket to a local IP address and port #
- listen(): passively waiting for connections
- connect(): initiating connection to another socket
- accept(): accept a new connection
- Write(): write data to a socket
- Read(): read data from a socket
- sendto(): send a datagram to another UDP socket
- recvfrom(): read a datagram from a UDP socket
- close(): close a socket (tear down the connection)

Sockets

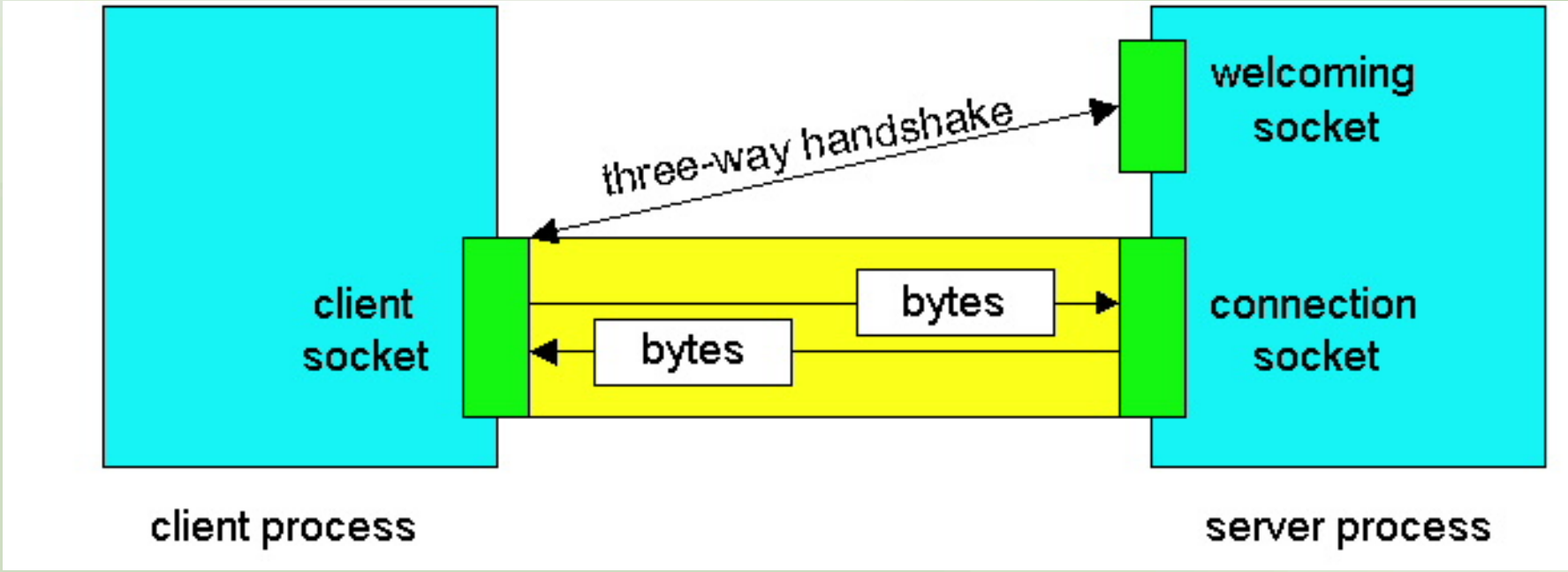
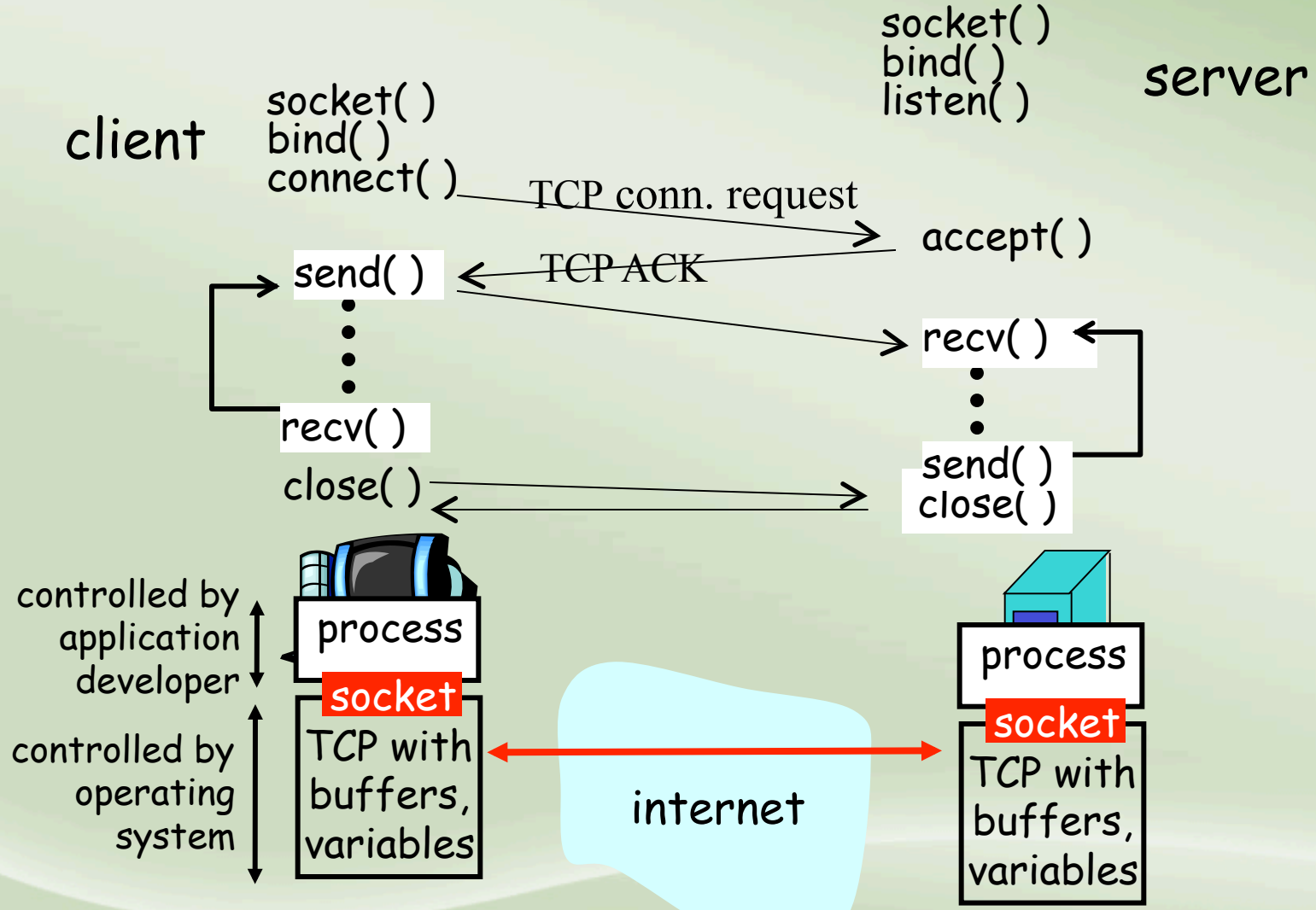


Figure 2.6-2: Client socket, welcoming socket and connection socket♪

Socket-programming using TCP

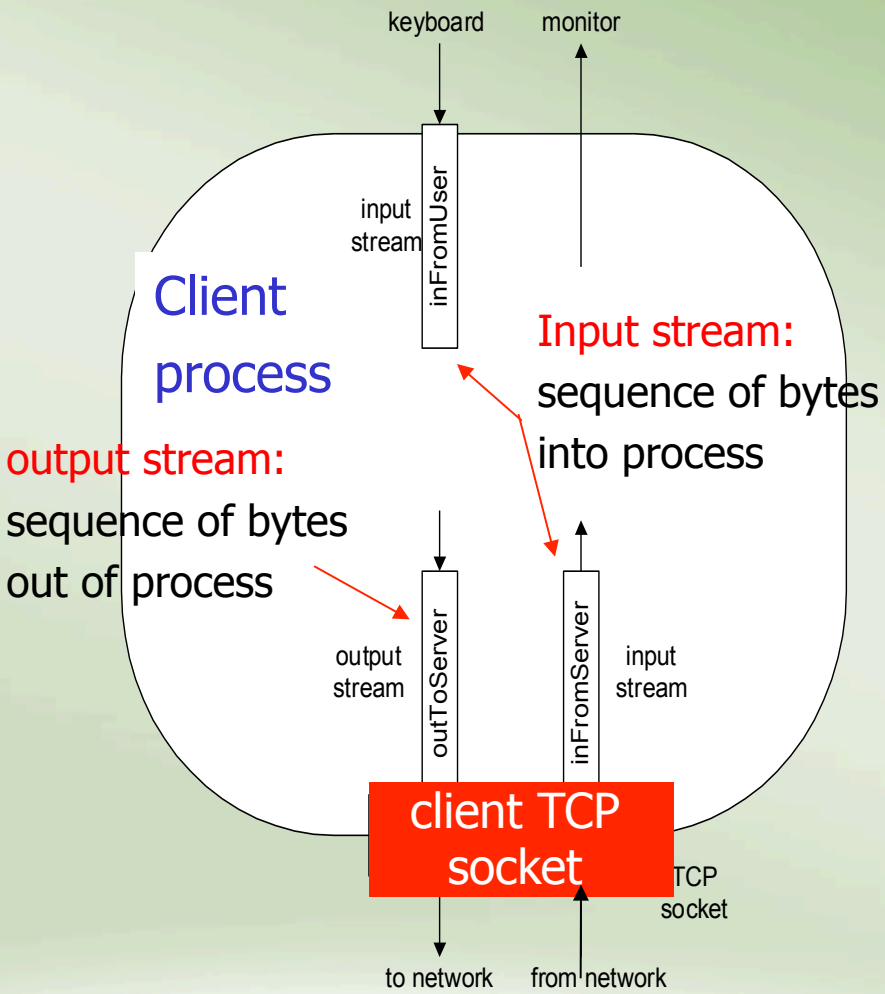
TCP service: reliable byte stream transfer ♪



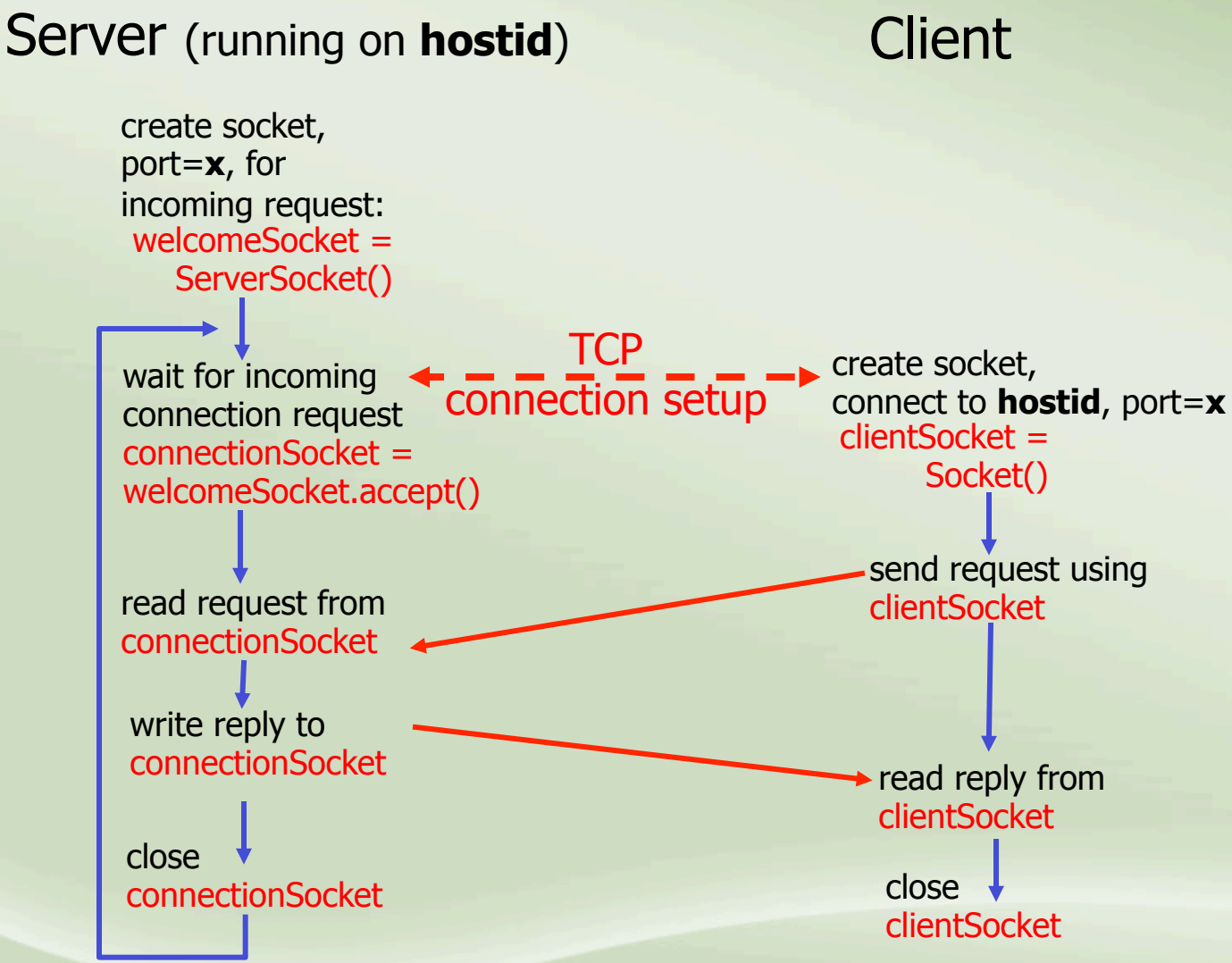
Socket programming with TCP

Example client-server app:

- client reads line from standard input (**inFromUser** stream) , sends to server via socket (**outToServer** stream)
- server reads line from socket
- server converts line to uppercase, sends back to client
- client reads, prints modified line from socket (**inFromServer** stream)



Client/server socket interaction: TCP



Creating a Socket

- Before using a socket, it has to be made
- Its access is similar to a file descriptor
- For socket creation, the socket function is called:

```
#include <sys/types.h>
#include <sys/socket.h>
int socket(int domain, int type, int protocol);
```

- Returns a socket descriptor if creation is successful or -1 on error
- Domain specifies the nature of the communication where the values include:
  - AF_INET and AF_INET6 for Internet Domain (IPV4 and IPV6 respectively)
  - AF_UNIX for the UNIX domain
  - AF_UNSPEC for unspecified or “any” domain

Creating a Socket

- Type is for the type of socket used:
  - SOCK_DGRAM – Datagram socket
  - SOCK_STREAM – Stream socket
  - SOCK_SEQPACKET – Sequential socket
  - SOCK_RAW – Raw socket
- Protocol is usually 0 indicating that default protocol is used.
- For example, to create a stream socket in the Internet domain:

```
s = socket(AF_INET, SOCK_STREAM, 0);
```

- Similar to calling open and obtaining file descriptor
- Some functions which accept file descriptor can also be used with a socket including close(), read(), write(), fchmod(),….etc.
- close() is used for closing (or deallocating) a socket when its usage is finished

Binding a Socket

- Created sockets cannot be used without associating with an address (or a name, basically)
- An address identifies a socket endpoint in a particular communication domain
- Informs a process “WHERE” to look for incoming messages
- For binding a socket with an address:

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

int bind(int sock_fd,const struct sockaddr *addr);
```

- Returns 0 if success or -1 if error in binding

Binding a Socket

- Format for the address is:

```
struct sockaddr {
        sa_family_t sa_family;
        char sa_data[];
        ..
        …
};
```

- Internet address is specified in <netinet/in.h> by the in_addr structure.

Using a Socket

Connection

- For connection-oriented service like Sequential and Stream Socket
- Prior connection between client and server is required before data exchange
- For connection:

  #include <sys/socket.h>

  int connect(int *sock_fd*,const struct sockaddr **addr*, socklen_t *len*);

- Returns 0 if successful or -1 if error occurs
- If a socket is not bound before connect() is called then the system will bind a default address to it
- Can also be used with connection-less sockets for optimization

Using a Socket

Sending Data

- For connection-oriented service write() can also be used, provided a connection is established
- Three specific socket data transfer functions are available if we want to specify options, or receive packets from multiple clients,...etc
  - ssize_t send(int *sock_fd*,const void **buff*, size_t *nbytes,* int *flags*);
  - ssize_t sendto(int *sock_fd*,const void **buff*, size_t *nbytes,* int *flags*,const struct sockaddr *des_*addr*, socklen_t *deslen*);
  - ssize_t sendmsg(int *sock_fd*,const struct msghdr **msg*, int *flags*);
- Returns number of bytes sent if successful or -1 if error occurs
- *buff* and *nbytes* have the same meaning as with write
- *des_addr* specifies the destination address for a connection-less service

Using a Socket

Sending Data

- Sendmsg is for specifying multiple buffers from which to transmit data.
- Structure of msghdr is:

```
struct msghdr {
        void *msg_name;
        socklen_t msg_namelen;
        ...
}
```

- Flags have special purposes which include sending out-of-bound data, preventing packet to route out of network, etc
- Flags are specifies by the constants: MSG_DONTROUTE, MSG_DONTWAIT, MSG_OOB, MSG_EOR

Using a Socket

Receiving Data

- For connection-oriented service read() can also be used, provided a connection is established
- Three specific socket data transfer functions are available if we want to specify options, or receive packets from multiple clients,...etc
  - ssize_t recv(int *sock_fd*,const void **buff*, size_t *nbytes,* int *flags*);
  - ssize_t recvfrom(int *sock_fd*,const void **buff*, size_t *nbytes,* int *flags*,const struct sockaddr *des_*addr*, socklen_t *deslen*);
  - ssize_t recvmsg(int *sock_fd*,const struct msghdr **msg*, int *flags*);
- Returns number of bytes sent if successful or -1 if error occurs
- All arguments have the same or analogous meaning as those of the reading functions.

Using a Socket

Other Operations

1) Shutdown:
   - We can "disable" I/O on a socket with the shutdown() function.

     #include <sys/socket.h>

     int shutdown (int *sockfd, int how);*

   - Returns 0 if successful or -1 if an error occurs
   - Values for *how:*
     - SHUT_RD – Reading is disabled
     - SHUT_WR –Writing is disabled
     - SHUT_RDWR – Reading and writing is disabled

Using a Socket

Other Operations

2) Close:
   - Deallocates the socket (similar to closing file descriptors).

     int close (int *sockfd);*
   - Returns 0 if successful or -1 if an error occurs
   - Closing occurs only last active referenced is closed (in case dup is used)

3) Socket Options:
   - Control the behaviour of sockets or allow special behaviour
   - Set and get option a particular socket

     #include <sys/socket.h>

     int setsockopt(int *sockfd, int level, int option,* const void **val,* socklen_t *len);*

Using a Socket

Other Operations

int getsockopt(int *sockfd, int level, int option,* void *restrict *val,* socklen_t *restrict *lenp);*

- *level* specifies the protocol to which the option applies
- *val* points to a data structure or an integer (depending on the option)
- *len* specifies the size of the object specified by *val*
- We can set and get three kinds of options:
  - Generic options which work with all sockets
  - Options managed at socket level but depends on the protocol
  - Protocol-specific options
- Single UNIX Specification specifies only socket-layer options.

Using a Socket

Other Operations

4) Other:

- Other operations include some function having a file descriptor argument.
- Examples are:
  - Duplication through the use of dup and dup2
  - lseek

UNIX Socket Domain

- UNIX domain sockets are used for IPC within a machine.
- Internet Domain sockets can be used for above purpose as well as for inter-machine communication
- UNIX domain sockets are more efficient for processes running in the same machine.
- Used only for copying data but have no acknowledgements to send, no n\w headers to add/remove, no checksums to calculate and so on.
- Provide stream as well as datagram interface.
- Datagram sockets here are however, much reliable and ordered than in internet domain
- It is like cross b/w sockets and pipes.

UNIX Socket Domain

- UNIX domain sockets are specified with the sockaddr_un structure and it differs from one implementation to another.
- Sockets can not be used with out binding.
- Socket without a name is same as a file descriptor with out a file name or address in file system.

The End