Network Programming

Socket API

Note: **This class lecture will be recorded!**
If you do not consent to this recording, please do not ask questions via your video, audio or public chat; send your question to the instructor using the private chat.

Lisa Frye, Instructor
frye@Kutztown.edu
Kutztown University

Copyright Lisa Frye 2017

Socket API

- UC Berkeley
- BSD UNIX
- de facto standard

Socket

- What is a *stream*?
- How can processes communicate?

- Unrelated process communication
- Bi-directional
- Client / Server
  - How does the client contact the server?
- File descriptor returned

TCP

WHICH ONE?

UDP

Socket Address

- Inter-process communication
- Socket pair
- TCP socket pair → 4-tuple

- **Q:** Why is a 4-tuple required for unique identification of a connection?
- **Q:** What is required to identify one endpoint of a TCP/IP connection?

Socket Address Data Types

- Generic Data type → ***sockaddr***

```
struct sockaddr   {                // generic socket address structure
    sa_family_t   sa_family; // address family (AF_INET, AF_UNIX)
    char          sa_data[];     // endpoint address in that family
};
```

- Internet (TCP, UDP) → ***sockaddr_in***
- UNIX → ***sockaddr_un***
- Include files

Socket Address Structures as Arguments

- Passed by reference

- Function definitions → generic structure (sockaddr)
- Function calls → cast to specific sockaddr structure

- How is this done?

Endian Concepts

- Read left to right or right to left?
- Gulliver's Travels

- Endianness in computer science
  - Big-endian → most significant byte on left
  - Little-endian → most significant byte on right

- Example: 91,329 → Hex value?
  - 00 01 64 C1
  - C1 64 01 00

	Low address							High address
Address	0	1	2	3	4	5	6	7
Little-endian	Byte 0	Byte 1	Byte 2	Byte 3	Byte 4	Byte 5	Byte 6	Byte 7
Big-endian	Byte 7	Byte 6	Byte 5	Byte 4	Byte 3	Byte 2	Byte 1	Byte 0
Memory content	0x11	0x22	0x33	0x44	0x55	0x66	0x77	0x88

64 bit value on Little-endian	64 bit value on Big-endian
0x8877665544332211	0x1122334455667788

Network Byte Order

- Big-endian

- htonl()
- htons()
- ntohl()
- ntohs()

h – host

n – network

s – short

l - long

Socket Connection using TCP

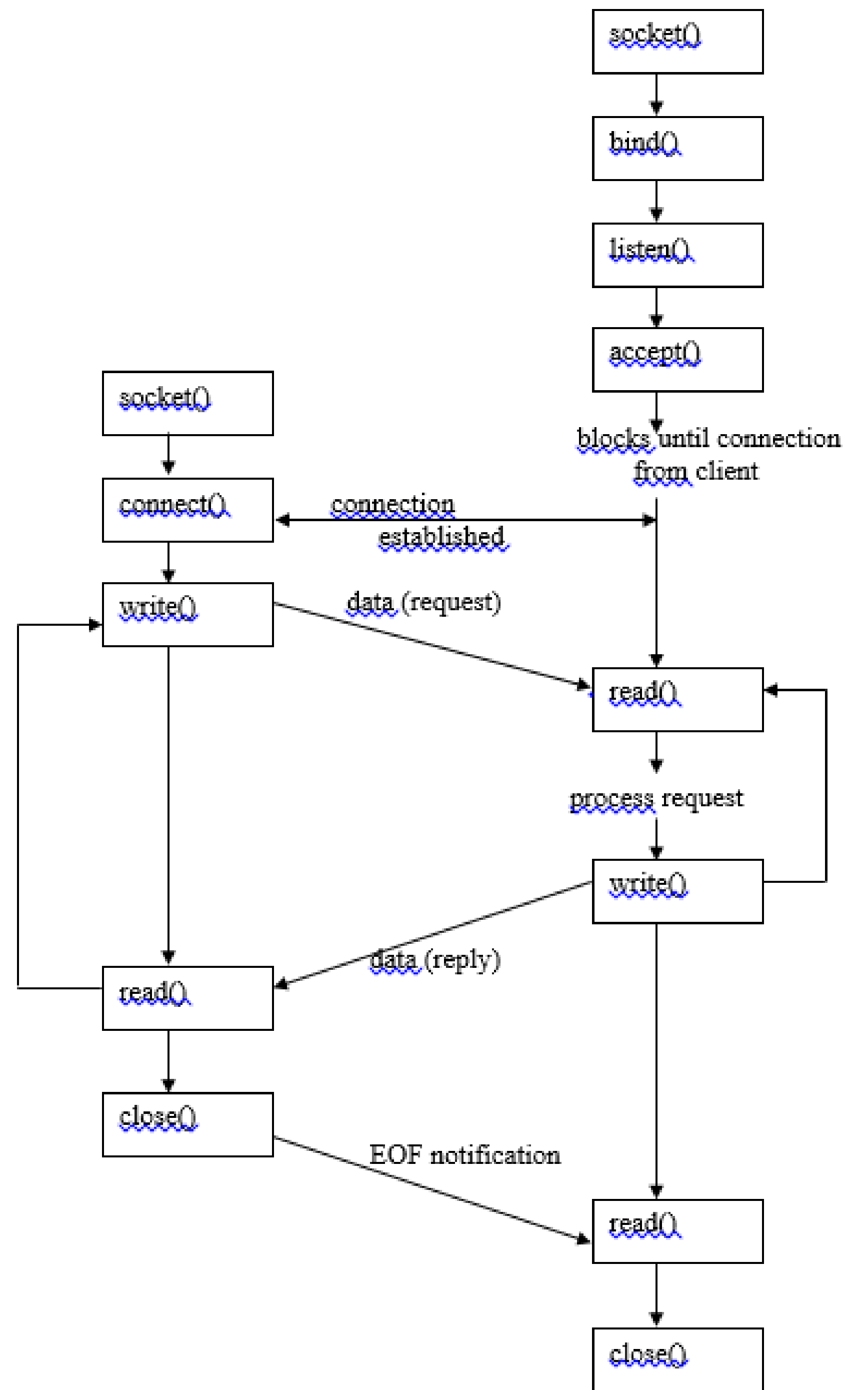
- Passive open
- Active open

- Passive close – receive FIN packet
- Active close – close socket

- UDP Connection

Socket Libraries

C / C++	nsl and socket (-lnsl –lsocket when link)
Java	java.net
Python	import socket No extra run-time libraries necessary


socket()
bind()
listen()
accept()
blocks until connection from client
socket()
connect()
connection established
write()
data (request)
read()
process request
write()
data (reply)
read()
close()
EOF notification
read()
close()

Server Actions

- Create a socket
- Assign an address to the socket
- Make socket a passive socket and listen
- Accept incoming connection

Server Creates a Socket

C / C++	socket(), bind(), listen()
Java	ServerSocket, accept()
Python	socket(), bind(), listen()

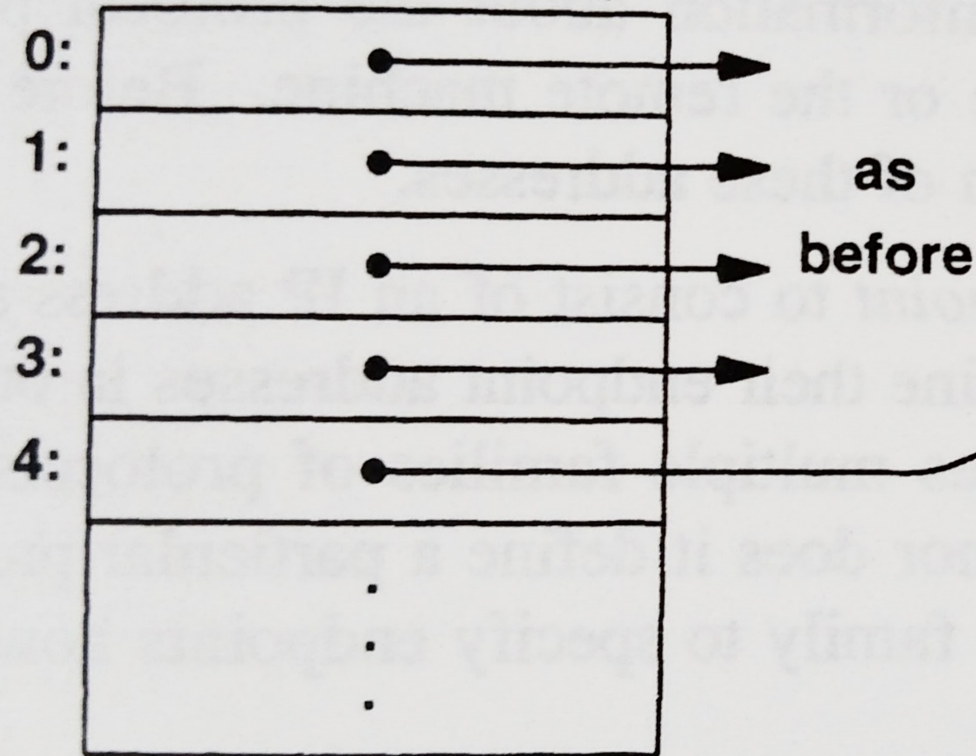
Client Creates a Socket

C/ C++	socket(), connect()
Java	Socket
Python	socket(), connect()

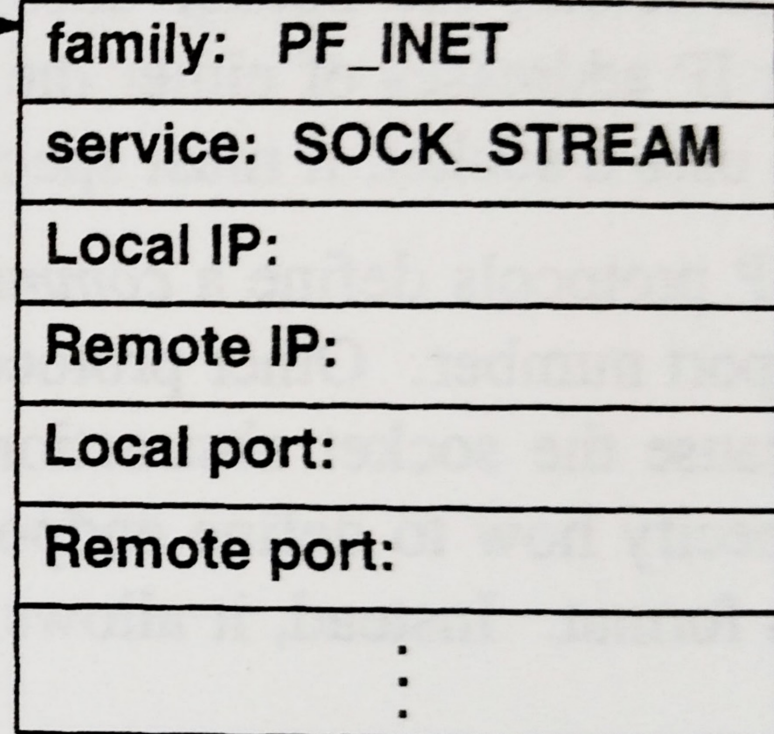
Socket Descriptor

Operating System

descriptor table
(one per process)



data structure for a socket



Socket system call

- **int socket(int domain, int type, int protocol);**

- Domain – address family
- Type – SOCK_STREAM or SOCK_DGRAM
- Protocol – typically 0 (system determines protocol)
- Returns
  - Socket descriptor
  - -1

Assign Address to Socket

- **int bind(int s, const struct sockaddr *name, int addrlen);**
- s – socket descriptor
- Name – address for socket
  - TCP – IP address and port number
- Addrlen – length of address in name
- Returns
  - 0 or -1

Listen for Incoming Connections

- Convert socket to a *passive socket*
- **int listen(int s, int backlog);**

- S – socket descriptor
- Backlog – number of connection requests in queue
- Returns
  - 0 or -1

Accept Connection

- **int accept(int s, struct sockaddr *name, int *addrlen);**

- S – socket descriptor
- Name – address of client (if not NULL)
- Addrlen – maximum length and actual length of address in name
  - Value-result argument (Input-output argument)
- Returns
  - Connected socket descriptor
  - -1

Address and Host System Calls

- gethostname()
- gethostbyname()
- inet_addr()
- inet_aton()
- inet_ntoa()

Client Actions

- Creates socket
- Connects socket to server's socket
- **int connect(int s, struct sockaddr *name, int addrlen);**

- S – socket descriptor
- Name – server's address (sockaddr_in)
- Addrlen – length of address in name
- Returns
  - 0 or -1

Transfer Data

- read()
- write()

- recv()
- send()

Discussion

- What are some ways that the client and server could synchronize data transfer?

Develop Functions

- ssize_t readn(int filedes, void *buff, size_t nbytes);
- ssize_t written(int filedes, const void *buff, size_t nbytes);
- ssize_t readline(int filedes, void *buff, size_t maxlen);

Understanding Buffering

- TCP and UDP buffer data
- Successful call to write() or send()
  - Where is the data written?
- Receive buffer
  - read() or recv()
- Program exit or crash??

Close Socket

C / C++	close(), shutdown()
Java	Automatically close if use a try-with-resources statement
Python	close(), shutdown()

Connectionless vs. Connection-Oriented

Similarities	Connection	Connectionless
Create socket		
Bind local address to socket		
	Server must listen for connections	
	Client just connects to server	Client must create a socket and bind its local address to that socket
	SOCK_STREAM	SOCK_DGRAM
	TCP protocol	UDP protocol

UDP Sockets

- How is socket programming over UDP different from socket programming over TCP?
- From the program perspective, how is UDP socket programming different from TCP?
- What should the client do if the packet does not reach the server?
- **Q:** What if the client needs to receive a response from the server?

Transferring Data over UDP Sockets

- sendto()
  - int sendto(int s, const char *buf, int len, int flags, struct sockaddr *to, int tolen);
- recvfrom()
  - int recvfrom(int s, char *buf, int len, int flags, struct sockaddr *from, int fromlen);
- sendmsg()
- recvmsg()

UDP Sockets

- Server
  - socket()
  - bind()
  - recvfrom() / sendto()
  - close()
- Client
  - socket()
  - sendto() / recvfrom()
  - close()