TA: Ishaan Lagwankar

Help Hours: Mon/Fri 11am-1pm

EB3105

Socket - an abstraction to send and receive data to another process. Data can be sent or received b between two processes on the same machine or two processes on different machines.

Transport Services - TCP(transmission control protocol) or UDP(user datagram protocol)

* Code socket. Give port number. Tell what to do.
* Considerations: Data loss, timing, throughout(minimum by app or elastic), security.

TCP vs UDP

TCP: - no data loss

* Reliable
* • Flow Control: Sender will not overwhelm receiver
* • Congestion Control: Throttle sender when network is
* congested
* • Connection Oriented: Setup between client and server
* required (3-way Handshake\*\*)
* • Does NOT provide:
* • Timing
* • Minimum throughput guarantees
* • Securit

UDP: -data loss but faster

• Unreliable

• Does NOT provide:

• Reliability

• Flow Control

• Congestion Control

• Timing

• Throughput Guarantees

• Security

• Connection setup

Kernel Data Structure : Uses kernel data to build a socket layer.

TCP is the typical client program use.

* UDP doesn’t guarantee connection to the server.
* Create socket -> determine server address and port number -> initiate connection to the server using TCP - > exchange data with server(write/receive) - > CLOSE socket.
* Close socket to prevent extra data transfer bogging down client and server.

SOCKET() system call:

• int socket ( int domain, int type, int protocol )

• Domain: protocol family (AF\_INET for the internet)

• Type: Semantics of the communication (SOCK\_STREAM (TCP), SOCK\_DGRAM (UDP)

SOCK\_RAW (RAW))

• Protocol: Usually unspecified, implied

• Returns a file descriptor

• Created without a name

• Includes protocol family, IP address and a port

• Name is bound to a socket later, using bind()

SOCKET PROGRAMMING WITH UDP

• No “connection” between client and server

• No handshaking

• Sender explicitly attaches IP address and port to each packet

• Receiver extracts sender IP and port for each packet

• Out-of-order packets allowed

• Unreliable transfer

SOCKET PROGRAMMING WITH TCP

• Client must contact server to establish connection

• Create TCP socket, specify IP address and port

• Establish connection to server TCP socket

• Server TCP creates new socket for server to connect with client

• Reliable, in-order byte-stream transfer (pipe) between client and server

MULTIPROCESSING

• To handle multiple clients, we use fork()\*\*

• Fork() creates a child process by duplicating the parent

• The child has pid = 0

• Both parent and child execute the next instruction

• Both parent and child have the same code segment but are independent of each other

• Child process has its own address space and own copy of variables

• We can create multiprocess programs that

• Parent listens for incoming connections

• For each connection, fork a child process to handle that connection

HTONS and NTONS

* Data organization from little endian to big endian