Network Programming - Fall 2023

Sockets:

Socket (local ip + port, remote ip + port)

(Basically, from local machine IP and port -> Remote machine IP and port.)

Socket()

Client server



(host) Data Communication (host2)



(socket) (socket2)



(connect) (accept)



(write) (read)

(read) (write)



(exit)

IP Address:

(example) 198.17.223.4 ([www.cameron.edu](http://www.cameron.edu))

^ Decimal format

Convert to binary format -> 00000100

Machine cannot store decimal value, only binary values.

You store the values as bytes, there are two ways, Forwards and backwards.

80 would be an example of integer short. The IP Address above is considered integer long.

2^32 for ipv4.

API’s:

A set of socket functions.

Hardware Address is 6 bytes long.

AA.12.AC.23.01-213 (with hexadecimal)

On average 2^48 combinations.

Week 1

Chapter 1 Introduction 1.1-1.6

* 1. Introduction
* Client – a machine that uses service
* Server – a machine that provide service
* Client/Server communicate via communication protocols
* Protocols that are agreements between client and server on how to communicate. Such as how to initial communication, how to format data/info, how to transmit, how to interpreter data/information, how to terminate the communication
* Client 🡪 server
* Server 🡪 client (Asynchronous callback)
* Multiple clients connect to one server
* Most common protocol –TCP/IP (Transmission Control Protocol/Internet Protocol)
* The OS Does not care about what data is transmitted, only the applications. The OS’s only job is to translate and transmit the data, regardless of what it is.

**Client and server are on the same LAN**

-------------------------------------------------------------------------------------

Web Client

Web Server

Application Protocol – Data only

matters at this stage

----- TCP protocol

Ethernet hardware has two addresses – IP address and MAC address.

TCP

TCP

----------------------------------------------------------------

IP

IP

IP protocol

-------------------------------------------------------------------------------------

------------------------ Ethernet Protocol – LAN Packet

Ethernet Driver

Ethernet driver

Sniffing – Physical level (hardware)

Typically uses Logical Switch

---------------------------------------------------------

If a message is broadcasted, anyone can see it unless encrypted

Ethernet

Physical switch - A. ----------- B. – Physical connection between two points, a

message will be conveyed two directions, there and back.

Logical switch – A. ------ 0 ------ B. – Can have multiple paths to find a path between two points.

\ / \ /

\ 0 --- 0 /

0/ \0

**Client and server are on different LANs connected through a WAN**

Server Application

Client Application

Host with TCP/IP

Host with TCP/IP

LAN LAN

Router(s)

* 1. A simple Daytime Client

#include "unp.h"

int main (int argc, char \*\*argv) // The only way you can input information from a keyboard to our program

// (ex: > daytimetcpclient 127.0.0.1)

// daytimectcpclient = argument 1, IP address is Argument 2)

// argv is argument vector

// argc = 2 example:

// argv

// 0 -> daytimetcpclient

// 1 -> 127.0.0.1

// 2 -> NULL

// argv[0] = daytimetcpclient (points to spot 0), argv[1] = 127.0.0.1, argv[2] = NULL

{

int sockfd, n;

char recvline[MAXLINE + 1]; //store received data at the end of array in the NULL spot (termination character)

struct sockaddr\_in servaddr;

//structure to store server's socket address ex: #1 ip address,

// 2 port number. You need to indicate either ipv4 or ipv6

if (argc != 2) // if not two arguments, then print an error message and quit

{

printf("a.out <ip address> \n");

exit (1); //exit – best to use because it clears the memory used by the program. (\_exit only clears current process memory)

}

if ((sockfd = socket (AF\_INET, SOCK\_STREAM, 0) <= 0) //if creating a socket failed – This is the socket function that will only print out the integer if one is available. This creates an empty socket, which the later functions will add (ip, server, sizeOf, etc)

{

printf("sockect error\n");

exit (1); //exit – best to use because it clears the memory used by the program. (\_exit only clears current process memory)

}

// Common libs: <Stdio.h>, <Stdlib.h>, <string.h> //

bzero(&servaddr, sizeof(servaddr)); //clean server socket structure – initializes server memory and cleans

servaddr.sin\_family = AF\_INET; //IPv4 family – server socket structure AF\_INET means IPV4

servaddr.sin\_port = htons(13); //time server port is 13 – Socket address is structured for the port number 13

if (inet\_pton(AF\_INET, argv[1] (points to argument vector slot 1, which is IP address), &servaddr.sin\_addr) <= 0)

{

printf("IP address error\n");

exit (1);

}

//connect to the server

if (connect (sockfd, (SA\*) &servaddr, sizeof(servaddr)) <= 0) // SA here is a user-defined structure. Because it is capitalized, it is a const.

{

printf("connection error\n");

exit (1);

}

while ((n = read (sockfd, recvline, MAXLINE)) >= 0) // Will read through the array until the last spot

{

recvline[n] = 0; //null terminator

if(fputs(recvline, stdout) == EOF) // fputs is the print statement. Recvline is the array, it reads form the source and outputs in fputs. The stdout here basically stands that it will print it on the screen. If you reach the EOF, then it quits. \*\* fputs adds the termination character automatically, making it safer to use\*\*

{

printf("print error.\n");

exit (1);

}

if (n < 0 ) //read error

{

printf("read error\n");

exit (1);

}

exit (0); //complete the task and exit

}

* 1. Protocol Independency

The above version for IP v4 only. If changing IP family to IP v6, socket family needs to be changed to AF\_INET6. Changes to code may look like the followings:

struct sockaddr\_in6 seraddr;

socket (AF\_INET6, SOCK\_STREAM, 0);

servaddr. sin6\_family = AF\_INET6;

servaddr. sin6\_port = hton(13);

* 1. Error Handling: wrap Functions

Using wrap functions to handling errors to locate an error quickly. Calling a real function from a wrap function. A wrap function has the same name with the corresponding real function, but the first letter of the function is upper case letter. For instance:

// Wrapper function used to do error checking (error handling statement) \*\* to use wrap functions you must #include <wrap.h> //

Int Socket(int family, int type, int protocol) // System interface is all lowercase. It’s technically not a real function here (wrapper), here you call the Socket function, then the function inside, which is another socket function. It checks to see if it can be called successfully, and if so then it works, If not, it prints an error.

{

int n;

if( n = socket(family, type, protocol) < 0 )

{

printf(“Socket error!\n”);

}

return n;

}

* 1. A Simple Daytime Server

#include "unp.h"

#include <time.h> // Included so you can fetch the system information. You use the system time as a random number seed.

int main (int argc, char \*\*argv)

{

int listenfd, connfd; // Listen and connector file descriptor

struct sockaddr\_in servaddr; // Address structure

char buff [MAXLINE];

time\_t ticks; // In UNIX, no days or anything, it just counts the amount of “ticks”

listenfd = socket (AF\_INET, SOCK\_STREAM, 0); // Creates the socket (empty)

bzero(&servaddr, sizeof(servaddr)); // Cleans up this piece of memory

servaddr.sin\_family = AF\_INET; // Initializes ipv4

servaddr.sin\_addr.s\_addr = htonl(INADDR\_ANY); //INADDR\_ANY = take ANY ip address

servaddr.sin\_port = htons(13); // declares the port as 13 (client/server must match)

bind (listenfd, (SA\*) &servaddr, sizeof(servaddr)); //binding the server address (SA is a constant) – Since you now have the structure, you “bind” the structure to your socket.

listen (listenfd, LISTENQ); //make the socket ready to receive connect requests

for (; ;) //server infinite loop

{

\*\* THIS IS A SERVICE LOOP \*\*

connfd = accept (listenfd, (SA\*) NULL, NULL); //accepting any requests

ticks = time (NULL); //fetch system time

snprintf(buff, sizeof(buff), "%.24s\r\n", ctime(&ticks)); //print function that prints nothing to your screen, but it prints to your buffer. source(ticks) to the 24 is the string, how many characters(sizeof) in this place(buff).

converting ticks to time and save the time into the buff

write (connfd, sizeof(buff), strlen(buff)); //write time into the socket

close (connfd);//close the connected socket after completing the task.

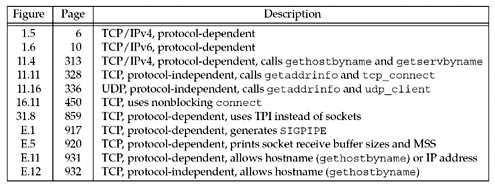
}

}

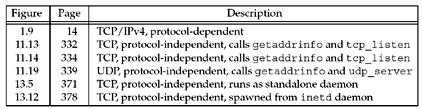
* 1. Roadmap to Client/Server Examples in the Text

On page 16-18. You may look at the tables

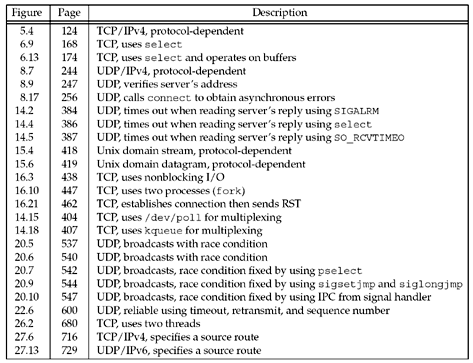
**Figure 1.10. Different versions of the daytime client developed in the text.**



**Figure 1.11. Different versions of the daytime server developed in the text.**



**Figure 1.12. Different versions of the echo client developed in the text.**



Ethernet standards:

802.3 – wired – Telephone wires/cables

802.11 – Wireless – radio waves (wifi)

Example: int main (int argc, char \*\*argv) – the argc means the number of arguments, the \*\* points to the multiple arguments stored in the initialized array.

Fd – file descriptor – Ex: stdin = 0; stdout = 1; stderr 2, so fd would be 3 since there were three processes.

Stdin – keyboard or file

Stdout – second file

Stderr – save to file

Standard File descriptors {0,1,2,3}  
 0-1023 ports (1024 usable ports) 2^16 is largest number for ports, or 65535 (65536)

Cout:

printf(), - control string

fprint, -%s can be many

snprintf() <- determines data safety,

sprintf() <- may not guaruntee data flow,

fputs(),

fputc() <- character string