Lab 1

Last update: 12/10/2021 09:08:00

This lab aims to teach you how to encode data for over the network exchange. This is a vital aspect in network communication as different data types require different types of encodings when exchanged over a TCP/IP network. These data representations are not required by the functioning of the network itself, but merely by the heterogeneous nature of the end systems. Depending on their architecture, end nodes might have different *in memory* data representation schemas.

The network stack implements data exchange as a stream/array/packet of bytes. Practically this means that the network is unaware of any data typing.��

When sending data, the network stack reads it from a memory address unaware of its representation on the host computer. It reads a number of bytes from the memory address and it sends those bytes over the network and lays them on the destination computer at the provided address in the same order. This could be a problem on systems with different endianness, etc as data type layouts in memory are different on different platforms.

Solutions to address the above issue are handled in multiple ways:

�         Using Host to Network and Network to Host functions (htonl, ntohl, etc)

�         Transforming the data representation into a common format (ex: a string of chars).Depending on� the case this might not be optimal.

Do you know about any other issues you might encounter when exchanging data over the network ? (When answering consider the network to be reliable: whatever is transmitted on one end successfully arrives in a finite amount of time on the other side).���

�

Implement a pair of client/server apps communicating trough TCP/IP (TCP sockets) for solving the following problems (try to handle all errors and communication issues) :

1.   A client sends to the server an array of numbers. Server returns the sum of the numbers.

2.   A client sends to the server a string. The server returns the count of spaces in the string.

3.   A client sends to the server a string. The server returns the reversed string to the client (characters from the end to begging)

4.   The client send to the server two sorted array of chars. The server will merge sort the two arrays and return the result to the client.

5.   The client sends to the server an integer. The server returns the list of divisors for the specified number.

6.   The client sends to the server a string and a character. The server returns to the client a list of all positions in the string where specified character is found.

7.   The client sends to the server a string and two numbers (s, I, l). The sever returns to the client the substring of *s* starting at index *I*, of length *l*.

8.   The client sends to the server two arrays of integers. The server returns an arrays containing the common numbers found in both arrays.

9.   The client sends to the server two arrays of numbers. The server returns to the client a list of numbers that are present in the first arrays but not in the second.

10.The client sends to the server two strings. The server sends back the character with the largest number of occurrences on the same positions in both strings together with its count.

Students should practice when solving these problems: *strace* and *ltrace* system commands.

|  |  |
| --- | --- |
| Client.c � sum of numbers |  |
| #include <sys/types.h>  #include <sys/socket.h>  #include <stdio.h>  #include <netinet/in.h>  #include <netinet/ip.h>  #include <string.h>  #include <unistd.h>  #include <arpa/inet.h>    int main() {  ������ int c;  ������ struct sockaddr\_in server;  ������ uint16\_t a, b, suma;    ������ c = socket(AF\_INET, SOCK\_STREAM, 0);  ������ if (c < 0) {  ������������� printf("Eroare la crearea socketului client\n");  ������������� return 1;  ������ }    ������ memset(&server, 0, sizeof(server));  ������ server.sin\_port = htons(1234);  ������ server.sin\_family = AF\_INET;  ������ server.sin\_addr.s\_addr = inet\_addr("127.0.0.1");    ������ if (connect(c, (struct sockaddr \*) &server, sizeof(server)) < 0) {  ������������� printf("Eroare la conectarea la server\n");  ������������� return 1;  ������ }    ������ printf("a = ");  ������ scanf("%hu", &a);  ������ printf("b = ");  ������ scanf("%hu", &b);  ������ a = htons(a);  ������ b = htons(b);  ������ send(c, &a, sizeof(a), 0);  ������ send(c, &b, sizeof(b), 0);  ������ recv(c, &suma, sizeof(suma), 0);  ������ suma = ntohs(suma);  ������ printf("Suma este %hu\n", suma);  ������ close(c);  } |  |

Server.c � iterative - sum of numbers

// ServerSuma-Iterativ.cpp : Defines the entry point for the console application.

//

#define \_WINSOCK\_DEPRECATED\_NO\_WARNINGS 1

// exists on all platforms

#include <stdio.h>

// this section will only be compiled on NON Windows platforms

#ifndef \_WIN32

#include <sys/types.h>

#include <sys/socket.h>

#include <netinet/in.h>

#include <netinet/ip.h>

#include <string.h>

#include <arpa/inet.h>

#include <unistd.h>

#include <errno.h>

#define closesocket close

typedef int SOCKET;

#else

// on Windows include and link these things

#include<WinSock2.h>

// for uint16\_t an so

#include<cstdint>

// this is how we can link a library directly from the source code with the VC++ compiler � otherwise got o project settings and link to it explicitly

//#pragma comment(lib,"Ws2\_32.lib")

#endif

int main() {

������ SOCKET s;

������ struct sockaddr\_in server, client;

������ int c, l, err;

// initialize the Windows Sockets LIbrary only when compiled on Windows

#ifdef \_WIN32

������ WSADATA wsaData;

������ if (WSAStartup(MAKEWORD(2, 2), &wsaData) < 0) {

������������� printf("Error initializing the Windows Sockets LIbrary");

������������� return -1;

������ }

#endif

������ s = socket(AF\_INET, SOCK\_STREAM, 0);

������ if (s < 0) {

������������� printf("Eroare la crearea socketului server\n");

������������� return 1;

������ }

������ memset(&server, 0, sizeof(server));

������ server.sin\_port = htons(1234);

������ server.sin\_family = AF\_INET;

������ server.sin\_addr.s\_addr = INADDR\_ANY;

������ if (bind(s, (struct sockaddr \*) &server, sizeof(server)) < 0) {

������������� perror("Bind error:");

������������� return 1;

������ }

������ listen(s, 5);

������ l = sizeof(client);

������ memset(&client, 0, sizeof(client));

������ while (1) {

������������� uint16\_t a, b, suma;

������������� printf("Listening for incomming connections\n");

������������� c = accept(s, (struct sockaddr \*) &client, &l);

������������� err = errno;

#ifdef \_WIN32

������������� err = WSAGetLastError();

#endif

������������� if (c < 0) {

�������������������� printf("accept error: %d", err);

�������������������� continue;

������������� }

������������� printf("Incomming connected client from: %s:%d", inet\_ntoa(client.sin\_addr), ntohs(client.sin\_port));

������������� // serving the connected client

������������� int res = recv(c, (char\*)&a, sizeof(a), 0);

������������� //check we got an unsigned short value

������������� if (res != sizeof(a)) printf("Error receiving operand\n");

������������� res = recv(c, (char\*)&b, sizeof(b), 0);

������������� if (res != sizeof(b)) printf("Error receiving operand\n");

������������� //decode the value to the local representation

������������� a = ntohs(a);

������������� b = ntohs(b);

������������� suma = a + b;

������������� suma = htons(suma);

������������� res = send(c, (char\*)&suma, sizeof(suma), 0);

������������� if (res != sizeof(suma)) printf("Error sending result\n");

������������� //on Linux closesocket does not exist but was defined above as a define to close

������������� closesocket(c);

������ }

������ // never reached

������ // Release the Windows Sockets Library

#ifdef \_WIN32

������ WSACleanup();

#endif

}

Python Client for Sum Server

\_\_author\_\_ = **'dadi'***#import socket for all socket related primitives***import**socket  
*# we need struct in order to be able to pack data in  
# a stream of bytes so that we can actually send  
# an integer as a binary four byte sequence - instead  
# of a string***import**struct  
  
s = socket.socket(socket.AF\_INET, socket.SOCK\_STREAM)  
  
*# input return actually a string and we need an int*a=int(input(**'a='**))  
b=int(input(**'b='**))  
  
*# The obscure struct\_addr is elegantly replaced by  
# a simple pair - very convenient. Replace the IP Address with  
# the one of your server*s.connect( (**"192.168.56.130"**,1234) )  
  
*# pack the value of a as a short int (16 bits) in network representation*res = s.send(struct.pack(**"!H"**, a))  
res = s.send( struct.pack(**'!H'**, b))  
c = s.recv(2)  
  
*# unpack the content read from the network into a short int  
# and convert from network representation back to host*c = struct.unpack(**'!H'**,c)  
print(**'a+b='**+ c[0].\_\_format\_\_(**'d'**))  
  
s.close()