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.

Example – iterative server using TCP sockets - Sum Server

Example Python – Client for Sum Server

General Sockets Theory and References