

**Secure Data Transfer System Simulator**

**PROJECT ON DATA STRUCTURES**

MAY 2020

****

**Introduction & Background**

This Project is a simulation of safe and optimized transfer of data from one system to another through parallel network links.

The main aim of this project is to optimize the time required to transfer data which involves getting the message from the user, encrypting it, sending it through a network link to a server, receiving it from the server from another device through network links, decrypting the message and displaying it to the recipient. If the user wishes to see the original message, then the message will be decrypted and displayed to him/her.

End-to-end encryption (E2EE) is a method of secure communication that prevents third-parties from accessing data while it's transferred from one end system or device to another. In E2EE, the data is encrypted on the sender's system or device and only the recipient is able to decrypt it.

The Diffie-Hellman key exchange algorithm is used in the encryption process.

**Thus, obtaining the key to the encryption is really tough, even with a supercomputer.**

****

**Data structures used**

We have used the queue data structure for this project. The queue data structure is preferred in this project because of its FIFO ( First In First Out ) feature which is most appropriate to implement Network links which we have used in this project. A network link is something that is used to transmit data packets from one device to another device. There are a number of network links through which data can be transferred. The data links may not have the same bandwidth so that the transmission speed differs while transferring data.

Using the transmission links in an efficient way is necessary for communicating with another device with minimal lag. Queue ADT is contained in the header file queue.h. Each node of the queue contains a message packet, packet number, and the address of the next node. The message packet is the data that is to be transmitted. The packet number is used to overcome a setback and organize data in the correct order in the receiver end. The next node stores the address of the next node in the transmission link.

****

**Files and Functions**

* basiclib.h
* main.c
* resource.h
* queue.h
* transmit.h
* encryption.h
* Keyexchange.h

**1) BASICLIB.H**

This file holds all the required inbuilt header files.

Stdio.h - standard input and output library

Stdlib.h - contains the random and dynamic allocation functions

Limits.h - contains MAX\_INT and MIN\_INT

String.h - contains all the string manipulation functions

Time.h  - contains all the function involving time

Math.h - contains all the mathematical functions

**2) MAIN.C**

This file holds the main function and the file to be executed, the main calls all the functions and orchestrates the program providing the interface for the user to work with.

***2.1) void initDC()***

This function initializes all the messages and key values to null and zero.

***2.2) void printdevice()***

Displays the information present in all the devices so that the simulation

 is transparent to the user.

***2.3) int input()***

To get the integral input from the user.

**3) RESOURCE.H**

This header file holds all the assets ( constants and structures ) required to run this project

***3.1)struct datacontainer***

This structure is the representation of a physical device, it has two properties, a string to hold the messages, and then an array of integers to hold 4 keys.

***3.2) #define ps***

Size of the data packets which are passed through the network links

**4) QUEUE.H**

The queue is the main Data structure ADT we have used for this project. A queue uses first in first out (FIFO) concept, enqueue and dequeue are the main functions used in queues. The message is divided into packets and is sent via transmission links (queue) takes minimal time to transfer the data.

***4.1) struct node***

Structure to represent a packet, it contains a string to store the message packet, an integer to store the packet number ( to identify a packet ), and a next pointer.

***4.2) struct Queue***

Structure to represent the queue, it has a front and rear pointer.

***4.3) struct node \*newNode(char k[ps], int p)***

This function creates and returns a new node.

***4.4) struct Queue \*createqueue()***

This function creates and initializes the Queue.

***4.5) void enQueue(struct Queue \*q, char k[ps], int pacnum)***

This function is used to append a data packet to the queue.

***4.6) void deQueue(struct Queue \*q)***

This function is used to receive/pop out a packet from the queue.

***4.7) void display(struct Queue \*q)***

This function is used to display the elements present in the queue.

**5) TRANSMIT.H**

Transmission of data is one of the crucial parts of any data transfer system, we are basically splitting the message into packets so that data can be sent through multiple network links with distinct bandwidths thus saving time. Packets are processed and pushed into a link in a way to achieve overall optimal minimal time required to deliver the whole message from one device to another.

The number of physical links between the sender and receiver is an important consideration because it is a major factor for determining the time required to process a message. Here, in our project we consider the number of such links(of max size 5) and the bandwidth size to be user-defined.

We have used dynamically created queues as network links which have their respective bandwidths as given by the user.

The message sent by the user is encrypted and divided into packets of sizes defined in the code (default 5) and is sent to the server through the network links. We have created a deterministic greedy algorithm that always finds the most optimal path for transferring each packet of data so the end message can be received by the receiver in minimal time.

***5.1) float push(struct datacontainer data, struct Queue \*q[5], int link, int bw[])***

This function splits the message from the datacontainer into packets and optimally chooses the link the packet will be appended to using a greedy algorithm.

***5.2) char \*pop(struct datacontainer data, struct Queue \*q[5], int link, char \*tempmsg)***

This function dequeues the packets appended into the links and combines the packet to form the message by using the packet number which is added in the push function and store this message in the appropriate destination ( datacontainers ).

***5.3) void emptylists(struct Queue \*q[5])***

This function is used to dequeue and empty all lists so that the queues can be used again. This function is called primarily when a new message is sent from the device1.

**6) ENCRYPTION.H**

When it comes to the data transfer system, encryption is required to safeguard the privacy of the users. Encryption is a process that encodes a message or file so that it can only be read by certain people. Encryption uses an algorithm to scramble, or encrypt, data, and then uses a key for the receiving party to unscramble, or decrypt, to obtain the information.

***6.1) char shuffle(char a[], char ch1, int l)***

This function is to swap the values. It is specifically designed in synchronization with the scrambling algorithm.

***6.2) void encrypt(char ch[], int k[4])***

This function is used to encrypt the message in such a way that it can not be read by any outsider. The encryption is done in two steps.

Step 1: Scrambling the string.

* For scrambling, we are using a key number that is picked on random which ranges from 1 to 7.  This number n is the common key.
* The entire string is divided into sets of n characters.
* Scrabble each set in a pattern such that the first character goes to the last index and the last character goes to the second index, the second character goes to the second-last index, and so on. Finally, the leftover middle character takes over the first index.

Step 2: Converting the characters of the string to special characters.

* We devised a formula using another key value to map each and every character of the printable character list to a special character.

This makes the code unrecognizable and safe for transmission.

***6.3) void decrypt(char ch[], int k[4])***

When it comes to data, decryption is used to switch unreadable ciphertext to readable information.

The decryption is done by reversing the encryption process with the help of the key obtained. The two steps of decryption are:

Step 1: Converting back the special characters to normal characters.

* By reversing the equation used in the encryption using the key, we can remap the special characters into printable characters.

Step 2:  Unscrambling the string.

* With the help of the common key, we would get to know how the characters are scrambled and hence we can unscramble the string and thereby get back the original order of the characters.

After decrypting, we get back the message sent from the other device without any loss of information.

**7) KEYEXCHANGE.H**

The Key is an important part of encryption. The key is used to encrypt and decrypt the message so if anyone gets access to the key he can crack the encryption, so the key must be protected from hackers at all causes.

To achieve better protection, we have used end-to-end encryption (E2EE) which allows only the sender ( i.e *device1* ) and the receiver ( i.e *device2* ) to access the encrypted message even the server ( the intermediate ) would have no access to the encrypted message, so to achieve this E2EE key exchange is a huge challenge since only the device1 and device2 should be able to access the key.

To tackle this problem we have implemented the Diffie-Hellman key exchange method which is not actually an exchange of keys but both the device agreeing on a common key such that no one else can access it.

The Diffie-Hellman key exchange involves 3 keys

private key: a (or) b

Public key: g^(a (or) b) mod n

common key: g^(a.b) mod n

Here constants g and n are produced by a common interface which will be accessible to everyone ( the server ), the devices use these constant to form the public key these public keys are shared with one another then this is combined with the private key thus creating the common key.

where,

* g is stored in key[0]
* Public and common key in key[1]
* n in key[2]
* a (or) b in key[3]

in their respective devices

***7.1) int randnum(int lower, int upper);***

To randomly choose a number between the given bounds. This function is mostly used to get the key as random key values to increase the security making it really hard to crack it even using brute force.

***7.2) void generateGN(struct datacontainer \*data);***

This function is used to assign a value to constants ‘g’ and ‘n’.

***7.3) void privatekeygen(struct datacontainer \*data);***

This function is called by device1 and device2 to get their own private keys. Only the respective devices have access to this key and the server has no access to this key value.

***7.4) void publickeygen(struct datacontainer \*data);***

This function creates a public key  ( g^(a) mod n ) in device1 and ( g^(b) mod n ) in device2.

***7.5) void commonkeygen(struct datacontainer \*data);***

This function is used to create the common key ( g^(a.b) mod n ) which is generated using the public and private keys of the respective devices.

***7.6) void keyexchange(struct datacontainer \*d1, struct datacontainer \*d2, struct datacontainer \*serv);***

This is the main function which calls other functions and conducts the key exchange.

****

**Features**

**Safe data transmission**

We have used end to end encryption ( E2EE ) to safely transfer data from device1 to device 2.

**Optimal data transmission**

Data is transferred in an optimum minimal time by splitting the message into packets through different links with respective bandwidths using a Greedy algorithm.

**Data abstraction**

We are using multiple files to hold the function in appropriate files only showing the necessary prototypes.

****

**Application**

1. Used to transfer data in minimal optimal time.
2. Used for safely sending, processing, and receiving a message by encrypting it.
3. Used for parallel processing of data with the help of several links, which eventually helps to optimize the time required in transmitting data.
4. Highly realistic model of data transmission with user-defined links and bandwidth size that can be used in real-time.

****

**Future  Scope**

* With little modification to the code, this project can be used to prepare a job schedule for an organization with the data inputs as the number of workers and productivity of each worker to do a job (measured in job/hour).
* It can also be used in queue management, with little modification to the code. It can be used in tracking the queue length and directing the work of the new entry to the queue, where the optimal time to process the work will be minimal. For example, customer call service, in banks, etc.

****

**Pros:**

* Better use of processor cache (keeps the message object local to all handlers which will use it).
* Small handlers don't impose as much overhead (as long as there are other handlers also to be run).
* More messages are expected than there are handlers, so the potential for parallelism is greater.

**Cons:**

* Unpredictable ordering - if message A is sent before message B, they may both be processed at the same time, or B may finish processing before all of A's handlers are finished (order is non-deterministic)

.