BILKENT UNIVERSITY

ENGINEERING FACULTY

DEPARTMENT OF COMPUTER ENGINEERING

CS 399

SUMMER TRAINING REPORT

Mustafa Göktan Güdükbay

21801740

Performed at

Siyah AR-GE (Araştırma ve Geliştirme)

03.06.2021-02.07.2021

Table of Contents

1. Introduction 3
2. Company Information 3
   2.1. About the company 3
   2.2. About your department 4
   2.3. About the hardware and software systems 4
   2.4. About your supervisor 4
3. Work Done 4
   3.1. First Assignment: Multithreaded Client-Server Application Using Transmission Control Protocol (TCP) 4
   3.2. Second Assignment: Single-Threaded Client-Server Program Using TCP 5
   3.3. Third Assignment: Client-Server Program Using UDP 6
   3.4. Fourth Assignment: Radar Software Project 6
   3.4.1. Transportation of Radar Software Project from VxWorks to Windows Visual Studio 6
   3.4.2. Message Queue Structure on Windows Using Shared Memory 7
4. Performance and Outcomes 8
   4.1. Solving Complex Engineering Problems 8
   4.2. Ethical and Professional Responsibilities 9
   4.3. Making Informed Judgments 9
   4.4. Acquiring New Knowledge by Using Appropriate Learning Strategies 10
   4.5. Applying New Knowledge As Needed 10
5. Conclusions 11
References 12
Appendices 13
   A. First Assignment: Multithreaded Client-Server Application that Uses Transmission Control Protocol (TCP) – Linked List Implementation 13
   B. First Assignment: Multithreaded Client-Server Application that Uses Transmission Control Protocol (TCP) – Standard Library Vector Implementation 21
   C. Second Assignment: Single-Threaded Client-Server Program Using TCP 28
   D. Third Assignment: Client-Server Program Using UDP 35
   E. Message Queue Structure on Windows Using Shared Memory (Non–Blocking Send) 40
   F. Message Queue Structure on Windows Using Shared Memory (Blocking Send) 48

1. Introduction

I performed my internship at the Siyah AR-GE (Araştırma ve Geliştirme) company. The duration of my internship was four weeks. The company works in embedded systems, hardware and software design and development, mobile systems, telecommunication systems, and cloud computing [1]. I did my summer internship at Siyah AR-GE because I wanted to work in system development using operating system concepts.

I worked on four assignments during my internship. In the first assignment, I did a multithreaded TCP (Transmission Control Protocol) server-client application. Then, in the second assignment, I made another TCP server-client application where the server and the client had single threads. Finally, in the third assignment, I made a UDP (User Datagram Protocol) server-client application. In the fourth assignment, I tried to move a radar software project implemented in VxWorks (a real-time embedded operating system) to Windows using Visual Studio. I could not finish the last assignment, but I believe I made significant progress. Besides, in this project, a message queue implementation was required on Windows. So, I implemented a message queue on Windows using a shared memory structure. I could not use the message queue library in Windows because it did not meet my expectations.

2. Company Information

2.1. About the company

Siyah AR-GE was founded in 2007. It is in Gazi University Technology Development Area. They have experience in system development and network communication. They work in:

- Embedded systems hardware and software design & development (Real-time systems, multi-processing / multi-threaded systems, Assembler / C / C++ / ARM / PowerPC / MIPS)
- Mobile systems (Low-level Android customizations and security enhancements, custom Android ROMs, Android Applications)
- Telecommunication systems (Digital Signal Processing, Modem design, Electromagnetic propagation modeling & simulation, powerline communication, protocol design & implementation)
- Cloud computing (establishing simple cloud computing environments for SMEs, developing mobile applications for cloud computing environments)

The managing core team is composed of three people. Gökhan Moral is one of the the founders and he is the chief executive officer (CEO) of the company. Önder Arık is also one of the founders and he is the chief technology officer (CTO) of the company. Prof. Dr. Özgür Barış Akan from Koç University, Department of Electrical and Electronics Engineering, is also the founder of the company [1].

2.2. About your department

There are not any departments in Siyah AR-GE.

2.3. About the hardware and software systems

The company uses languages like C++, C#, Python. They use Unix-based systems, Windows, and VxWorks for system development. The language I used in this internship was C++, and I developed my programs in Unix-based systems and Windows. They use Visual Studio and Eclipse as IDE.

2.4. About your supervisor

My advisor in SRDC was Önder Arık, who is the CTO of the company. He received the B.Sc. degree in electrical and electronics engineering from Bilkent University, Ankara, Turkey, in June 1997. I also worked closely with Gökhan Moral, who is the CEO of the company. He is the General Manager of Siyah R&D LLC. He received the B.Sc. and M.Sc. degrees in electrical and electronics engineering from Bilkent University, Ankara, Turkey, in June 1999 and January 2002, respectively [2].

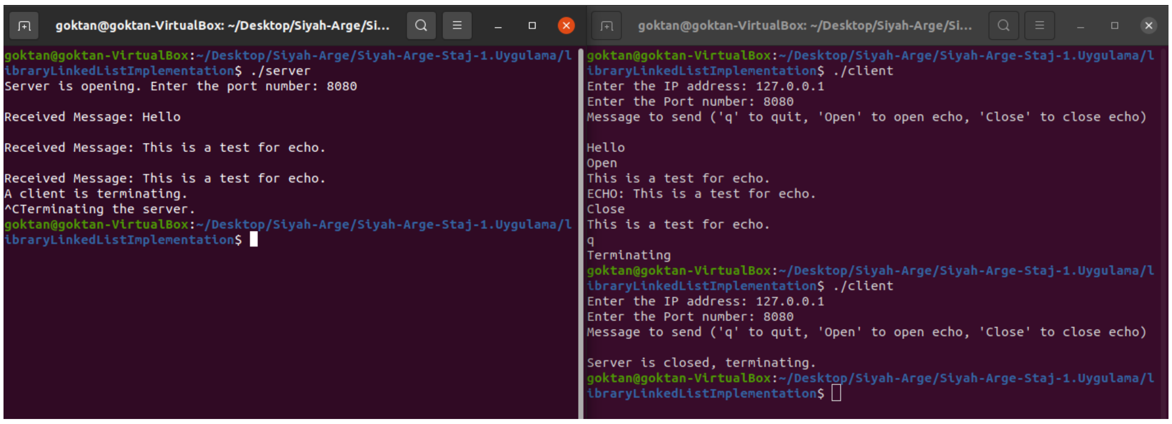
3. Work Done

3.1. First Assignment: Multithreaded Client-Server Application Using Transmission Control Protocol (TCP)

This assignment aims to create a client-server application that uses Transmission Control Protocol (TCP) to communicate. TCP is a communications standard that devices use to transport information over a network. It assures the correctness of the information during transportation. Before it transmits data, TCP establishes a connection between a source and its destination, ensuring it remains alive until communication begins. It then uses small packets to transfer the information while protecting data integrity [3].

The communication has an echo option where the client can open or close the echo during transmission. The echo option is global, so when a client opens or closes the echo, it affects all the servers connected. The server has a main thread that creates a thread for each client connection. The client has two threads, one for listening to the server for echoes and one for getting input from the user and sending messages to the server. Additionally, CTRL-c is handled using signal handling.

A mutex lock is used to access the linked list without producing any race conditions. Both the server and the client use multithreading. The programs support a *graceful shutdown*. A *graceful shutdown* is when a software function turns off a computer, and the operating system is allowed to perform its tasks of safely shutting down processes and closing connections [4]. A *hard shutdown* is when the computer is forcibly shut down [4]. When a client terminates, the server can understand if the connection is closed. And when the server terminates, clients automatically terminate. The select function from the system library allows programs to monitor multiple file descriptors, waiting for file descriptors to become "ready" for some I/O operation [5]. I used the select function for implementing a graceful shutdown. The server must store the thread ids for a graceful shutdown. The server has two different implementations. In the first implementation, the server uses a linked list implemented by myself to store thread ids. In the second implementation, the server uses the standard vector library of C++ to store thread ids.


goktan@goktan-VirtualBox: ~/Desktop/Siyah-Arge/Si...
goktan@goktan-VirtualBox: ~/Desktop/Siyah-Arge/Si...

```
goktan@goktan-VirtualBox:~/Desktop/Siyah-Arge/Siyah-Arge-Staj-1.Uygulama/l
ibraryLinkedListImplementation$ ./server
Server is opening. Enter the port number: 8080

Received Message: Hello

Received Message: This is a test for echo.

Received Message: This is a test for echo.
A client is terminating.
^CTerminating the server.
goktan@goktan-VirtualBox:~/Desktop/Siyah-Arge/Siyah-Arge-Staj-1.Uygulama/l
ibraryLinkedListImplementation$ 
```

```
goktan@goktan-VirtualBox:~/Desktop/Siyah-Arge/Siyah-Arge-Staj-1.Uygulama/l
ibraryLinkedListImplementation$ ./client
Enter the IP address: 127.0.0.1
Enter the Port number: 8080
Message to send ('q' to quit, 'Open' to open echo, 'Close' to close echo)

Hello
Open
This is a test for echo.
ECHO: This is a test for echo.
Close
This is a test for echo.
q
Terminating
goktan@goktan-VirtualBox:~/Desktop/Siyah-Arge/Siyah-Arge-Staj-1.Uygulama/l
ibraryLinkedListImplementation$ ./client
Enter the IP address: 127.0.0.1
Enter the Port number: 8080
Message to send ('q' to quit, 'Open' to open echo, 'Close' to close echo)

Server is closed, terminating.
goktan@goktan-VirtualBox:~/Desktop/Siyah-Arge/Siyah-Arge-Staj-1.Uygulama/l
ibraryLinkedListImplementation$ 
```

Figure 1 - Multithreaded Client-Server Program that Uses TCP

As you can see from Figure 1, when the client terminates, the server prints a message and, when the server terminates, it notifies the clients, and the client automatically terminates.

3.2. Second Assignment: Single-Threaded Client-Server Program Using TCP

The second assignment had the same purpose as the first assignment. However, this time both the server and client programs are single-threaded. In the server part, rather than using a separate thread for each client, the select function checks for a new message from the connected clients. When a client establishes a connection to the server, a file descriptor is assigned. This time, I used a list to store these file descriptors. The select function is called with a bit array parameter corresponding to the set of file descriptors where each bit in the set tells that if select should monitor that file descriptor [5]. The server also has a file descriptor. The set is marked with the server's file descriptor and the file descriptors for the connected clients. Periodically (0.01 seconds) select function is called. I checked the bit corresponding to a file descriptor with the FD_ISSET function [6]. If the server's file descriptor is set, then a new client is connecting. If a client's file descriptor is set, then it implies that the client sent a message.

In the last part, the client had two threads, one for listening to echo from the server and one for getting input from the user. In the client program, the server's file descriptor and the 0th bit (input file descriptor) were marked. The select function was called periodically (0.01 seconds), the set was checked if there is an echo or if there is input from the user.

In this implementation, I used the C++ standard list library to store the file descriptors. I used the list library because it was better in terms of time complexity than the standard vector library. Multi-threaded servers can have advantages and disadvantages compared to single-threaded servers. Multi-threaded servers can have faster response times if many clients are connected to the server that sends numerous messages. However, if the clients do not send lots of messages using lots of threads, resources may be overhead.

3.3. Third Assignment: Client-Server Program Using UDP

This assignment had the same purpose as the previous assignments. However, this time the server and the client use User Datagram Protocol (UDP) to communicate.

The UDP is a communication protocol used for time-sensitive transmissions such as video playback or DNS lookups. Unlike TCP, it does not establish a connection before transferring data. This increases the speed of data transfer. However, UDP is not reliable compared to TCP because data packets can be lost, and it does not have an error correction mechanism [7].

3.4. Fourth Assignment: Radar Software Project

3.4.1. Transportation of Radar Software Project from VxWorks to Windows Visual Studio

The company has a radar management software project that they have been developing on VxWorks for a long time.

RADAR (Radio Detecting and Ranging) is an electronic system used to determine physical attributes such as distance and angle of objects like vehicles, airplanes, and so on. Radars send out electromagnetic waves and use their reflections to make calculations on the physical attributes of the objects [8]. VxWorks is a real-time operating system (RTOS) to support application development through containers. VxWorks is a priority-based preemptive RTOS. VxWorks can support C++17, Boost, Rust, Python, pandas [9]. My job was to move this radar software project from VxWorks to Visual Studio. However, operating system concepts in VxWorks and Windows have significant differences. The company has a project on operating system abstraction, which had some incomplete parts. The operating system abstraction software combined the general operating system concepts in UNIX, Windows, and VxWorks. I used the operating system abstraction software to move the project from VxWorks to Windows.

At first, I tried to make configurations and arrange the paths in Visual Studio. Then, I used the abstraction software to change the code's parts using threads, semaphores, and mutexes. The first problem I faced was, the radar project was using a message queue structure in VxWorks. However, the abstraction software did not have a message queue structure, and the one in the Windows library does not meet the requirements.

3.4.2. Message Queue Structure on Windows Using Shared Memory

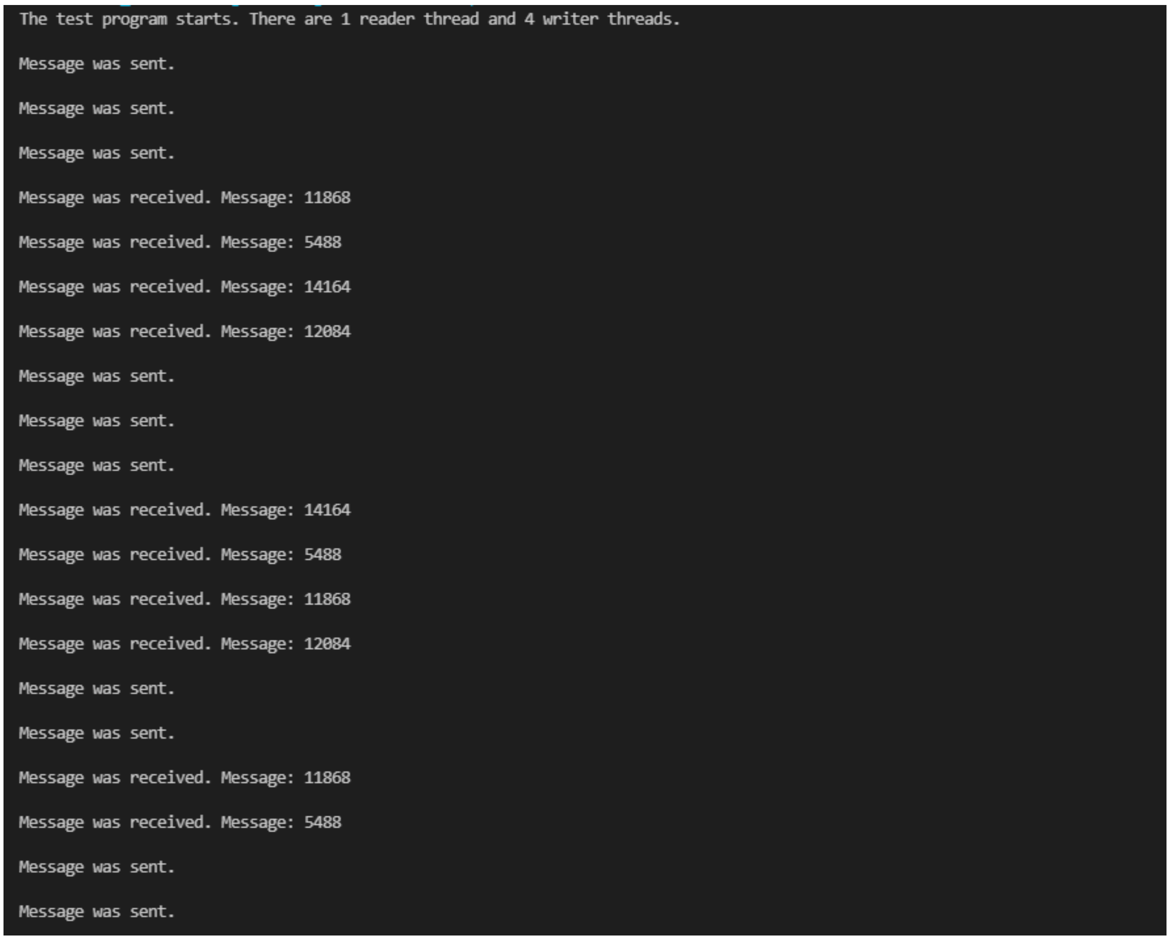
After the need for a message, they asked me to implement a message queue structure on Windows. The message queue requires a shared memory. I used the *CreateFileMappingA* function from the Windows API to create a shared memory. This function creates or opens a named or unnamed file mapping object for a specified file [10]. I also used a tutorial on Creating Named Shared Memory from Microsoft [11].

The message queue I implemented had four operations:

- open the message queue,
- send a message,
- receive a message, and
- close the message queue.

Unlike the message queue implementation on UNIX, there could be only one reader for a message queue. However, there can be multiple writers. The message queue supported four operations: open the message queue, send a message, receive a message, and close the message queue. I tested the message queue implementation by writing a test program with many writer threads writing messages to the message queue and a single reader thread reading from that queue (see Figure 2).

I implemented two different message queues. In the first implementation, I used a semaphore and a mutex. The mutex is used to prevent race conditions when accessing the shared memory. The semaphore is used to make the receive message operation blocking. So, if there are not any messages in the queue, the reader will be blocked until someone sends a new message. However, the send operation is not blocking. If someone tries to write when the queue is full, the send operation terminates unsuccessfully. In the second implementation, I used two semaphores and a mutex. The reason for using an extra semaphore was to make the send operation blocking. So, if the message queue is full, the writer will be blocked until someone receives a message.



```
The test program starts. There are 1 reader thread and 4 writer threads.

Message was sent.

Message was sent.

Message was sent.

Message was received. Message: 11868

Message was received. Message: 5488

Message was received. Message: 14164

Message was received. Message: 12084

Message was sent.

Message was sent.

Message was sent.

Message was received. Message: 14164

Message was received. Message: 5488

Message was received. Message: 11868

Message was received. Message: 12084

Message was sent.

Message was sent.

Message was received. Message: 11868

Message was received. Message: 5488

Message was sent.

Message was sent.
```

Figure 2 - Output of the Test Program

I also created a dynamically linked library of this message queue in Visual Studio using the tutorial [12]:

4. Performance and Outcomes

4.1. Solving Complex Engineering Problems

In the first three assignments, the purpose was to implement a client-server program using different network technologies such as TCP and UDP. The main problem I had during these three assignments was not having prior experience with computer networks. Additionally, debugging client-server architectures can be hard because it is difficult to understand if the problem is on the client or server side. In the first assignment, I did not use any libraries for a linked list. Later, my advisor suggested I use the built-in libraries for fast and clean solutions.

In the last assignment, I faced many problems. The first problem was, I did not have any experience with Visual Studio and system development in Windows. I learned Visual Studio using Microsoft documentation and tutorials from YouTube. There were not many tutorials on system development in Windows. Therefore, I used Microsoft documentation for system development. Also, in the last assignment, I experienced an interesting problem. When I used one mutex and one semaphore and the write operation was not allowed when the queue was full, the system was not assuring bounded waiting on threads. When the reader thread was trying to obtain the unlocked mutex, the system did not give the mutex. This was because the critical section period in the send operation was so fast that the reader thread was not fast enough to obtain the mutex [10].

4.2. Ethical and Professional Responsibilities

My first responsibility was completing the assignments given to me. I showed my code and outputs to my advisor to get feedback. I changed the parts according to the feedback given by my advisor. Although there were no deadlines for any of the assignments given to me, I tried to finish them in the shortest period I could.

4.3. Making Informed Judgments

In all assignments, my first purpose was to write clean and efficient programs. I needed a data structure to hold information like thread ids and file descriptors in some assignments. In the first assignment, I needed to store thread ids to be aware when a client terminates the connection with the server. I first used a linked list that I implemented to store the thread ids. The linked list was advantageous in the insertion and deletion of elements because each operation has $O(1)$ complexity. The search operation is $O(N)$ complexity in the worst case. In the second implementation, I used the standard vector library. The vector structure has amortized $O(1)$ complexity in insertion operation. However, deletion has $O(N)$ complexity in the worst case. The search operation is $O(N)$ complexity in the worst case. Using the vector library is disadvantageous because of the complexity of the deletion operation. However, if there are not many deletions going to take place, we can use the vector library in this context. However, the linked list is better for servers where network traffic is heavy. My primary purpose of using the vector library gain experience on it.

In the second assignment, I needed to store the file descriptors assigned to each client because the server was running on a single thread. I used the standard list library to store the file descriptors because of the advantages of deletion compared to the vector structure. In the third assignment, I used UDP for data transfer. UDP does not require a prior connection like TCP; however, it does not compensate for network errors. Users can favor UDP in situations where lost data is not a problem, like streaming a video. In the last assignment, I needed a message queue structure in Windows. I had to implement it using shared memory. I used a shared memory because I wanted to handle the memory efficiently. At the same time, I wanted to fulfill the needs of the message queue.

4.4. Acquiring New Knowledge by Using Appropriate Learning Strategies

I did not have computer network knowledge before this internship. I used TCP and UDP protocols in three assignments. I read on the internet to obtain background information on network protocols. Additionally, I read these two tutorials:

1. TCP server-client implementation in C. GeeksforGeeks. (2019, October 10). Retrieved August 9, 2021, from https://www.geeksforgeeks.org/tcp-server-client-implementation-in-c/.

2. UDP server-client implementation in C. GeeksforGeeks. (2021, July 26). Retrieved August 9, 2021, from https://www.geeksforgeeks.org/udp-server-client-implementation-c/.

In the last assignment, I needed to learn Visual Studio and Windows system development. I used these official Microsoft documentation and tutorials to obtain new information:

1. J-Martens. (n.d.). *Visual studio ide documentation*. Microsoft Docs. Retrieved August 9, 2021, from https://docs.microsoft.com/en-us/visualstudio/ide/?view=vs-2019.
2. Mcleanbyron. (n.d.). *Creating named shared memory - win32 apps*. Win32 apps | Microsoft Docs. Retrieved August 9, 2021, from https://docs.microsoft.com/en-us/windows/win32/memory/creating-named-shared-memory.
3. Mcleanbyron. (n.d.). *Using semaphore objects - win32 apps*. Win32 apps | Microsoft Docs. Retrieved August 9, 2021, from https://docs.microsoft.com/en-us/windows/win32/sync/using-semaphore-objects.
4. Mcleanbyron. (n.d.). *Build desktop windows apps using the win32 API - win32 apps*. Win32 apps | Microsoft Docs. Retrieved August 9, 2021, from https://docs.microsoft.com/en-us/windows/win32/.
5. Corob-Msft. (n.d.). *Walkthrough: Create and use your own dynamic link library (C++)*. Microsoft Docs. Retrieved August 9, 2021, from https://docs.microsoft.com/en-us/cpp/build/walkthrough-creating-and-using-a-dynamic-link-library-cpp.

4.5. Applying New Knowledge As Needed

I needed to learn and apply network knowledge in various assignments. I learned TCP and UDP network protocols. I gained experience by applying them in C++. I learned the standard C++ libraries. I needed to learn system development in Windows. Additionally, I needed to use the operating system concepts I knew in school to Windows.

5. Conclusions

I did my internship at Siyah AR-GE for four weeks, and my advisor was Önder Arık. I also worked closely with Gökhan Moral. I worked on computer networks and system development, implemented client-server applications using different network protocols like TCP and UDP, data structures and requirements like running the application in different numbers of threads, and gracefully shut down. Additionally, I used Visual Studio to move a project from VxWorks to Windows. I worked on working on large projects which I did not have any prior knowledge of. This was very challenging for me because I did not have any knowledge of radars. Besides, there were many files in the project, making it hard to make the configurations in Visual Studio. Learning a new IDE in such a big project was difficult for me. I also learned system development in Windows. Using Windows libraries for system development was a good experience for me. I believe I learned important concepts and gained experience in this internship.

References

[1] "Siyah R&D". [Online]. Available: http://siyaharge.com/. [Accessed: 09-Aug-2021].

[2] "Gökhan Moral," Profile of Gökhan Moral. [Online]. Available: http://www.siyaharge.com/pages/resume/gokhan-moral.html. [Accessed: 09-Aug-2021].

[3] "What is transmission Control Protocol (TCP)?," Fortinet. [Online]. Available: https://www.fortinet.com/resources/cyberglossary/tcp-ip. [Accessed: 09-Aug-2021].

[4] TechTarget Contributor, "What is graceful shutdown and hard shutdown? - definition from whatis.com," WhatIs.com, 26-Oct-2018. [Online]. Available: https://whatis.techtarget.com/definition/graceful-shutdown-and-hard-shutdown. [Accessed: 09-Aug-2021].

[5] Select(2) - Linux manual page. [Online]. Available: https://man7.org/linux/man-pages/man2/select.2.html. [Accessed: 09-Aug-2021].

[6] "Fd_Isset(3) - linux man page," fd_isset(3): synchronous I/O multiplexing - Linux man page. [Online]. Available: https://linux.die.net/man/3/fd_isset. [Accessed: 09-Aug-2021].

[7] "What is UDP?," Cloudfare. [Online]. Available: https://www.cloudflare.com/learning/ddos/glossary/user-datagram-protocol-udp/. [Accessed: 09-Aug-2021].

[8] Australian Government Bureau of Meteorology, How Radar Works. [Online]. Available: http://www.bom.gov.au/australia/radar/about/what_is_radar.shtml. [Accessed: 09-Aug-2021].

[9] "VxWorks The Leading RTOS for the Intelligent Edge," Wind River. [Online]. Available: https://www.windriver.com/products/vxworks. [Accessed: 09-Aug-2021].

[10] D. Gehriger, Jiri, and Frank, "Windows Critical Section Strange Behaviour," Stack Overflow, 01-Jul-1959. [Online]. Available: https://stackoverflow.com/questions/4733915/windows-critical-section-strange-behaviour. [Accessed: 09-Aug-2021].

[11] Microsoft, Inc., "Creating Named Shared Memory", Microsoft, 31-May-2018. [Online]. Available: https://docs.microsoft.com/en-us/windows/win32/memory/creating-named-shared-memory. [Accessed: 09-Aug-2021].

[12] Corob-Msft, "Walkthrough: Create and use your own dynamic link library (c++)," Microsoft Docs. [Online]. Available: https://docs.microsoft.com/en-us/cpp/build/walkthrough-creating-and-using-a-dynamic-link-library-cpp?view=msvc-160. [Accessed: 09-Aug-2021].

Appendices

A. First Assignment: Multithreaded Client-Server Application that Uses Transmission Control Protocol (TCP) – Linked List Implementation

```
//*****************************************************
//                   MUSTAFA GÖKTAN GÜDÜKBAY
//                        Siyah AR-GE
//*****************************************************

//C++ Implementation of the TCP server that uses linked list.

#include <unistd.h>
#include <stdio.h>
#include <sys/socket.h>
#include <stdlib.h>
#include <netinet/in.h>
#include <string.h>
#include <iostream>
#include <csignal>


using namespace std;

struct ThreadInformation{
  pthread_t thread_id;
  struct ThreadInformation* next;
};

//global variables
int server_fd;
struct sockaddr_in address;
int opt = 1;
int addrlen = sizeof(address);
int echo = 0;//no echo at the beginning
int shuttingDown = 0;
struct ThreadInformation* head = NULL;

//lock variable
pthread_mutex_t lock;

//thread function to listen
void* receiver(void* arg){
  char buffer[1024];
  int socket_id = *((int*) arg);
  int valread;

  fd_set set;
  struct timeval timeout;
  int select_return_value;

  while(!shuttingDown){
    memset(buffer, 0, 1024);
```

```
 //check if there is something to read using select,
 //timeout is 100 microseconds
 timeout.tv_sec = 0;
 timeout.tv_usec = 100;
 FD_ZERO (&set);
 FD_SET (socket_id, &set);

 select_return_value = select(socket_id + 1, &set, NULL,
        NULL, &timeout);

 if(select_return_value > 0)
   valread = recv(socket_id , buffer, 1024, 0);
 else
   continue;

 //client terminating
 if (valread == 0){
   //lock
   pthread_mutex_lock(&lock);
   pthread_t thread_id = pthread_self();

   struct ThreadInformation* cur = head;

   if(cur->thread_id == thread_id){
     head = head->next;
     delete cur;
   }
   else{
     struct ThreadInformation* prev = head;
     while(cur->thread_id != thread_id){
       prev = cur;
       cur = cur->next;
     }

     prev->next = cur->next;
     delete cur;
   }

   printf("A client is terminating.\n");
   //unlock
   pthread_mutex_unlock(&lock);
   break;
 }

 if(strcmp("Open", buffer) == 0){
 //lock
  pthread_mutex_lock(&lock);
  echo = 1;
  //unlock
  pthread_mutex_unlock(&lock);
}
 else if(strcmp("Close", buffer) == 0){
 //lock
  pthread_mutex_lock(&lock);
  echo = 0;
```

```
      //unlock
      pthread_mutex_unlock(&lock);
     }
     else{
       printf("\nReceived Message: %s\n", buffer);
       //send back the message
       if(echo == 1)
         send(socket_id , buffer , valread , 0);
     }
   }

   shutdown(socket_id, SHUT_RDWR);
   close(socket_id);
   delete (int*) arg;
   pthread_exit(NULL);
 }

 void gracefulShutdown (int signum){
   //lock
    pthread_mutex_lock(&lock);
    shuttingDown = 1;
    //unlock
    shutdown(server_fd, SHUT_RDWR);
    close(server_fd);
    pthread_mutex_unlock(&lock);
 }

 void cleaningMethod(){
   //clean
   printf("Terminating the server.\n");
   struct ThreadInformation* temp;
   while(head != NULL){
     temp = head;
     head = head->next;
     pthread_join(temp->thread_id, NULL);
     delete temp;
   }
 }

 int main(int argc, char const *argv[])
 {

     //variables
     int new_socket;//connected socket number of the client
     int* temp_socket;
     pthread_t thread_id;
     int port;

     cout << "Server is opening. Enter the port number: ";
     cin >> port;

     signal(SIGINT, gracefulShutdown);

     // Creating socket file descriptor
     if ((server_fd = socket(AF_INET, SOCK_STREAM, 0)) == 0)
     {
         perror("socket failed");
         return -1;
     }

     // Forcefully attaching socket to the port 8080
     if (setsockopt(server_fd, SOL_SOCKET, SO_REUSEADDR |
            SO_REUSEPORT, &opt, sizeof(opt)))
     {
         cerr << "setsockopt";
         return -1;
     }
     address.sin_family = AF_INET;
     address.sin_addr.s_addr = INADDR_ANY;
     address.sin_port = htons( port );

     // Forcefully attaching socket to the port 8080
     if (bind(server_fd, (struct sockaddr *)&address,
             sizeof(address))<0)
     {
         cerr << "bind failed";
         return -1;
     }


     if (listen(server_fd, 3) < 0)
     {
         cerr << "listen";
         return -1;
     }


     //loop to listen constantly
     while(shuttingDown == 0){
       if ((new_socket = accept(server_fd,
            (struct sockaddr *)&address, (socklen_t*)&addrlen))<0)
       {
         continue;
       }
       else{
         temp_socket = new int;
         *temp_socket = new_socket;

         // Creating a new thread
         pthread_create(&thread_id, NULL, &receiver,
            (void *)temp_socket);

         if(head == NULL){
           head = new struct ThreadInformation;
           head->thread_id = thread_id;
           head->next = NULL;
         }
         else{
           struct ThreadInformation* temp = head;
           while(temp->next != NULL)
             temp = temp->next;
           temp->next = new struct ThreadInformation;
           temp = temp->next;
           temp->thread_id = thread_id;
           temp->next = NULL;
         }
       }
     }


     cleaningMethod();
     return 0;
 }
```

```
//C++ Implementation of the TCP client that uses linked list.


#include <stdio.h>
#include <sys/socket.h>
#include <arpa/inet.h>
#include <unistd.h>
#include <string.h>
#include <iostream>
#include <csignal>


using namespace std;

//global variables
int serverClosed = 0;
int termination = 0;
int sock = 0;

//Handle ctrl-c
void signalHandler (int signum){
  termination = 1;
}

void* readerThread(void* arg){

  int select_return_value1, select_return_value2;
  fd_set set;
  struct timeval timeout;
  int valread;
  char buffer[1024];

  while(!serverClosed & !termination){
    timeout.tv_sec = 0;
    timeout.tv_usec = 100;
    FD_ZERO (&set);
    FD_SET (sock, &set);

    select_return_value1 = select(sock + 1, &set, NULL, NULL,
       &timeout);

    if (select_return_value1 > 0) {
      memset(buffer,0,1024);
      valread = recv(sock, buffer, 1024,0);
      if(valread == 0){
        printf("Server is closed, terminating.\n");
        serverClosed = 1;
        break;
      }

      printf("ECHO: %s\n", buffer);
    }
  }

  pthread_exit(NULL);
}
```

```
int main(int argc, char const *argv[])
{

    //variables
    struct sockaddr_in serv_addr;
    char buffer[1024] = {0};
    pthread_t thread_id;

    fd_set rfds;
    struct timeval tv;
    int retval;
    int len;
    int send_return_value;
    char ip[16];
    int port;


    cout << "Enter the IP address: ";
    cin >> ip;

    cout << "Enter the Port number: ";
    cin >> port;


    //code
    if ((sock = socket(AF_INET, SOCK_STREAM, 0)) < 0)
    {
        printf("\n Socket creation error \n");
        return -1;
    }

    serv_addr.sin_family = AF_INET;
    serv_addr.sin_port = htons(port);

    // Convert IPv4 and IPv6 addresses from text to binary form
    if(inet_pton(AF_INET, ip, &serv_addr.sin_addr)<=0)
    {
        printf("\nInvalid address/ Address not supported \n");
        return -1;
    }

    if (connect(sock, (struct sockaddr *)&serv_addr,
           sizeof(serv_addr)) < 0)
    {
        printf("\nConnection Failed \n");
        return -1;
    }


    signal(SIGINT, signalHandler);

    // Creating a new thread
    pthread_create(&thread_id, NULL, &readerThread, NULL);

    printf("Message to send ('q' to quit, 'Open' to open echo,
      'Close' to close echo)\n\n");
    while(!termination){
      memset(buffer,0,1024);


      while(!serverClosed & !termination){
        FD_ZERO(&rfds);
        FD_SET(0, &rfds);

        /* Wait up to five seconds. */
        tv.tv_sec = 5;
        tv.tv_usec = 0;

        retval = select(1, &rfds, NULL, NULL, &tv);
        if(retval > 0)
          break;
       }
      if(serverClosed | termination)
        break;


      cin.getline(buffer, sizeof(buffer));

      if (strcmp("q", buffer) == 0){
        termination = 1;
        printf("Terminating\n");
        break;
      }

      //send the message
     send_return_value = send(sock, buffer, strlen(buffer), 0);


      if(send_return_value == -1)
        cout << "Could not send the mssage." << endl;
    }

    pthread_join(thread_id, NULL);
    return 0;
}
```

B. First Assignment: Multithreaded Client-Server Application that Uses Transmission Control Protocol (TCP) – Standard Library Vector Implementation

```
//*********************************************************
//                    MUSTAFA GÖKTAN GÜDÜKBAY
//                         Siyah AR-GE
//*********************************************************

//C++ Implementation of the TCP server
//that uses standard library vector.

#include <unistd.h>
#include <stdio.h>
#include <sys/socket.h>
#include <stdlib.h>
#include <netinet/in.h>
#include <string.h>
#include <iostream>
#include <csignal>
#include <vector>


using namespace std;

struct ThreadInformation{
  pthread_t thread_id;
};

//global variables
std::vector<struct ThreadInformation*> myList;
int server_fd;
struct sockaddr_in address;
int opt = 1;
int addrlen = sizeof(address);
int echo = 0;//no echo at the beginning
int shuttingDown = 0;
struct ThreadInformation* head = NULL;

//lock variable
pthread_mutex_t lock;

//thread function to listen
void* receiver(void* arg){
  char buffer[1024];
  int socket_id = *((int*) arg);
  int valread;

  fd_set set;
  struct timeval timeout;
  int select_return_value;

  while(!shuttingDown){
    memset(buffer, 0, 1024);
    //check if there is something to read using select,
    //timeout is 100 microseconds
    timeout.tv_sec = 0;
    timeout.tv_usec = 100;
    FD_ZERO (&set);
    FD_SET (socket_id, &set);

    select_return_value = select(socket_id + 1, &set, NULL,
           NULL, &timeout);

    if(select_return_value > 0)
      valread = recv(socket_id , buffer, 1024, 0);
    else
      continue;

    //client terminating
    if (valread == 0){
      //lock
      pthread_mutex_lock(&lock);
      pthread_t thread_id = pthread_self();

      for (int i = 0; i < myList.size(); i++) {
               if(myList.at(i)->thread_id == thread_id){
                      myList.erase(myList.begin() + i);
                      break;
                }
        }

      printf("A client is terminating.\n");
      //unlock
      pthread_mutex_unlock(&lock);
      break;
    }

    if(strcmp("Open", buffer) == 0){
    //lock
     pthread_mutex_lock(&lock);
     echo = 1;
     //unlock
     pthread_mutex_unlock(&lock);
   }
    else if(strcmp("Close", buffer) == 0){
    //lock
     pthread_mutex_lock(&lock);
     echo = 0;
     //unlock
     pthread_mutex_unlock(&lock);
    }
    else{
      printf("\nReceived Message: %s\n", buffer);
      //send back the message
      if(echo == 1)
        send(socket_id , buffer , valread , 0);
    }
  }
```

```
  shutdown(socket_id, SHUT_RDWR);
  close(socket_id);
  delete (int*) arg;
  pthread_exit(NULL);
}

void gracefulShutdown (int signum){
  //lock
   pthread_mutex_lock(&lock);
   shuttingDown = 1;
   //unlock
   shutdown(server_fd, SHUT_RDWR);
   close(server_fd);
   pthread_mutex_unlock(&lock);
}

void cleaningMethod(){
  //clean
  printf("Terminating the server.\n");

  for (int i = 0; i < myList.size();) {
    pthread_join(myList.at(i)->thread_id, NULL);
    delete myList.at(i);
    myList.erase(myList.begin() + i);
      }
}

int main(int argc, char const *argv[])
{
    //variables
    int new_socket;//connected socket number of the client
    int* temp_socket;
    pthread_t thread_id;
    int port;

    cout << "Server is opening. Enter the port number: ";
    cin >> port;

    signal(SIGINT, gracefulShutdown);

    // Creating socket file descriptor
    if ((server_fd = socket(AF_INET, SOCK_STREAM, 0)) == 0)
    {
        perror("socket failed");
        return -1;
    }

    // Forcefully attaching socket to the port 8080
    if (setsockopt(server_fd, SOL_SOCKET, SO_REUSEADDR |
           SO_REUSEPORT, &opt, sizeof(opt)))
    {
        cerr << "setsockopt";
        return -1;
    }
    address.sin_family = AF_INET;
    address.sin_addr.s_addr = INADDR_ANY;
    address.sin_port = htons( port );

    // Forcefully attaching socket to the port 8080
    if (bind(server_fd, (struct sockaddr *)&address,
           sizeof(address))<0)
    {
        cerr << "bind failed";
        return -1;
    }


    if (listen(server_fd, 3) < 0)
    {
        cerr << "listen";
        return -1;
    }

    //loop to listen constantly
    while(shuttingDown == 0){
      if ((new_socket = accept(server_fd,
           (struct sockaddr *)&address, (socklen_t*)&addrlen))<0)
      {
        continue;
      }
      else{
        temp_socket = new int;
        *temp_socket = new_socket;

        // Creating a new thread
        pthread_create(&thread_id, NULL, &receiver,
            (void *)temp_socket);
        struct ThreadInformation* temp = new ThreadInformation;
        temp->thread_id = thread_id;
        myList.push_back(temp);
      }
    }


    cleaningMethod();
    return 0;
}
```

```
//C++ Implementation of the TCP client
//that uses standard library vector.

#include <stdio.h>
#include <sys/socket.h>
#include <arpa/inet.h>
#include <unistd.h>
#include <string.h>
#include <iostream>
#include <csignal>


using namespace std;

//global variables
int serverClosed = 0;
int termination = 0;
int sock = 0;

//Handle ctrl-c
void signalHandler (int signum){
  termination = 1;
}

void* readerThread(void* arg){

  int select_return_value1, select_return_value2;
  fd_set set;
  struct timeval timeout;
  int valread;
  char buffer[1024];

  while(!serverClosed & !termination){
    timeout.tv_sec = 0;
    timeout.tv_usec = 100;
    FD_ZERO (&set);
    FD_SET (sock, &set);

    select_return_value1 =
           select(sock + 1, &set, NULL, NULL, &timeout);

    if (select_return_value1 > 0) {
      memset(buffer,0,1024);
      valread = recv(sock, buffer, 1024,0);
      if(valread == 0){
        printf("Server is closed, terminating.\n");
        serverClosed = 1;
        break;
      }

      printf("ECHO: %s\n", buffer);
    }
  }

  pthread_exit(NULL);
}
```

```
int main(int argc, char const *argv[])
{

    //variables
    struct sockaddr_in serv_addr;
    char buffer[1024] = {0};
    pthread_t thread_id;

    fd_set rfds;
    struct timeval tv;
    int retval;
    int len;
    int send_return_value;
    char ip[16];
    int port;


    cout << "Enter the IP address: ";
    cin >> ip;

    cout << "Enter the Port number: ";
    cin >> port;


    //code
    if ((sock = socket(AF_INET, SOCK_STREAM, 0)) < 0)
    {
        printf("\n Socket creation error \n");
        return -1;
    }

    serv_addr.sin_family = AF_INET;
    serv_addr.sin_port = htons(port);

    // Convert IPv4 and IPv6 addresses from text to binary form
    if(inet_pton(AF_INET, ip, &serv_addr.sin_addr)<=0)
    {
        printf("\nInvalid address/ Address not supported \n");
        return -1;
    }

    if (connect(sock, (struct sockaddr *)&serv_addr,
           sizeof(serv_addr)) < 0)
    {
        printf("\nConnection Failed \n");
        return -1;
    }


    signal(SIGINT, signalHandler);

    // Creating a new thread
    pthread_create(&thread_id, NULL, &readerThread, NULL);

    printf("Message to send ('q' to quit, 'Open' to open echo,
      'Close' to close echo)\n\n");
    while(!termination){
      memset(buffer,0,1024);


      while(!serverClosed & !termination){
        FD_ZERO(&rfds);
        FD_SET(0, &rfds);

        /* Wait up to five seconds. */
        tv.tv_sec = 5;
        tv.tv_usec = 0;

        retval = select(1, &rfds, NULL, NULL, &tv);
        if(retval > 0)
          break;
       }
      if(serverClosed | termination)
        break;


      cin.getline(buffer, sizeof(buffer));

      if (strcmp("q", buffer) == 0){
        termination = 1;
        printf("Terminating\n");
        break;
      }

      //send the message
     send_return_value = send(sock, buffer, strlen(buffer), 0);

      if(send_return_value == -1)
        cout << "Could not send the mssage." << endl;
    }

    pthread_join(thread_id, NULL);
    return 0;
}
```

C. Second Assignment: Single-Threaded Client-Server Program Using TCP

```
//*********************************************************
//                    MUSTAFA GÖKTAN GÜDÜKBAY
//                          Siyah AR-GE
//*********************************************************

//C++ Implementation of the TCP single-threaded server
//that uses standard library list.

#include <unistd.h>
#include <stdio.h>
#include <sys/socket.h>
#include <stdlib.h>
#include <netinet/in.h>
#include <string.h>
#include <iostream>
#include <csignal>
#include <list>


using namespace std;

//global variables
int terminating = 0;

struct ClientInformation{
  int socket_fd;
};

void closeServer (int signum){
  terminating = 1;
}

int main(int argc, char const *argv[]){

  //variables
  int server_fd;
  int port;
  list <struct ClientInformation> clients;
  fd_set set;
  struct timeval timeout;
  int select_return_value;
  int valread;
  int echo = 0;
  char buffer[1024];
  struct sockaddr_in address;
  int opt = 1;
  int addrlen = sizeof(address);
  int new_socket = 0;
```

```
//code
signal(SIGINT, closeServer);

cout << "Server is opening. Enter the port number: ";
cin >> port;

// Creating socket file descriptor
if ((server_fd = socket(AF_INET, SOCK_STREAM, 0)) == 0)
{
    perror("socket failed");
    return -1;
}

// Forcefully attaching socket to the port 8080
if (setsockopt(server_fd, SOL_SOCKET, SO_REUSEADDR |
     SO_REUSEPORT, &opt, sizeof(opt)))
{
    cerr << "setsockopt";
    return -1;
}
address.sin_family = AF_INET;
address.sin_addr.s_addr = INADDR_ANY;
address.sin_port = htons( port );

// Forcefully attaching socket to the port 8080
if (bind(server_fd, (struct sockaddr *)&address,
    sizeof(address))<0)
{
    cerr << "bind failed";
    return -1;
}


if (listen(server_fd, 3) < 0)
{
    cerr << "listen";
    return -1;
}


while(!terminating){

  FD_ZERO (&set);
  FD_SET (server_fd, &set);
  int maxFD = server_fd;
  for (std::list<struct ClientInformation>::iterator
         it=clients.begin(); it != clients.end(); ++it){
    if(it->socket_fd > maxFD)
      maxFD = it->socket_fd;
    FD_SET (it->socket_fd, &set);
  }


  timeout.tv_sec = 0;
  timeout.tv_usec = 10000;
```

```
    select_return_value =
           select(maxFD + 1, &set, NULL, NULL, &timeout);

    if(select_return_value > 0){

      if(FD_ISSET(server_fd, &set)){
        new_socket = accept(server_fd,
           (struct sockaddr *)&address, (socklen_t*)&addrlen);

        if(new_socket <= 0){
          printf("Connection error.\n");
        }
        else{
          printf("%d\n", new_socket);
          struct ClientInformation client;
          client.socket_fd = new_socket;

          clients.push_back(client);
        }
      }

      for (std::list<struct ClientInformation>::iterator
           it=clients.begin(); it != clients.end(); ++it){
        if(FD_ISSET(it->socket_fd, &set)){
          memset(buffer, 0, 1024);

          valread = recv(it->socket_fd , buffer, 1024, 0);

          //client terminating
          if (valread == 0){
            it = clients.erase(it);
            --it;
            printf("A client is terminating.\n");
          }
          else if(strcmp("Open", buffer) == 0){
           echo = 1;
          }
          else if(strcmp("Close", buffer) == 0){
           echo = 0;
          }
          else{
            printf("\nReceived Message: %s\n", buffer);
            //send back the message
            if(echo == 1)
              send(it->socket_fd , buffer , valread , 0);
          }
        }
      }
    }

  }
  //close client sockets and server sockets, graceful shutdown
  for (std::list<struct ClientInformation>::iterator
      it=clients.begin(); it != clients.end(); ++it){
    shutdown(it->socket_fd, SHUT_RDWR);
    close(it->socket_fd);
  }

  shutdown(server_fd, SHUT_RDWR);
  close(server_fd);

  printf("Closing the server.\n");
}
```

```
//C++ Implementation of the TCP single-threaded client that
//uses standard library list.

#include <stdio.h>
#include <sys/socket.h>
#include <arpa/inet.h>
#include <unistd.h>
#include <string.h>
#include <iostream>
#include <csignal>

using namespace std;

//global variables
int terminating = 0;

//Handle ctrl-c
void signalHandler (int signum){
  terminating = 1;
}

int main(int argc, char const *argv[])
{

    //variables
    struct sockaddr_in serv_addr;
    char buffer[1024] = {0};

    fd_set rfds;
    struct timeval tv;
    int retval;
    int len;
    int send_return_value;
    char ip[16];
    int port;
    int inputSelect;
    int sock = 0;
    int valread;

    //code
    cout << "Enter the IP address: ";
    cin >> ip;

    cout << "Enter the Port number: ";
    cin >> port;

    if ((sock = socket(AF_INET, SOCK_STREAM, 0)) < 0)
    {
        printf("\n Socket creation error \n");
        return -1;
    }

    serv_addr.sin_family = AF_INET;
    serv_addr.sin_port = htons(port);
```

```
// Convert IPv4 and IPv6 addresses from text to binary form
if(inet_pton(AF_INET, ip, &serv_addr.sin_addr)<=0)
{
    printf("\nInvalid address/ Address not supported \n");
    return -1;
}

if (connect(sock, (struct sockaddr *)&serv_addr,
       sizeof(serv_addr)) < 0)
{
    printf("\nConnection Failed \n");
    return -1;
}


signal(SIGINT, signalHandler);

printf("Message to send ('q' to quit, 'Open' to open echo,
  'Close' to close echo)\n\n");


while(!terminating){
  memset(buffer,0,1024);

  FD_ZERO(&rfds);
  FD_SET(sock, &rfds);
  FD_SET(0, &rfds);

  tv.tv_sec = 0;
  tv.tv_usec = 100000;

  retval = select(sock + 1, &rfds, NULL, NULL, &tv);

  if(retval > 0){
    if(FD_ISSET(0, &rfds)){
      cin.getline(buffer, sizeof(buffer));

      if (strcmp("q", buffer) == 0){
        printf("Terminating\n");
        break;
      }

      //send the message
      send_return_value =
            send(sock, buffer, strlen(buffer), 0);

      if(send_return_value == -1)
        printf("Could not send the message.\n");

    }
    if(FD_ISSET(sock, &rfds)){
      valread = recv(sock, buffer, 1024,0);
      if(valread == 0){
        printf("Server is closed, terminating.\n");
        break;
      }
```

```
     printf("ECHO: %s\n", buffer);
    }
   }
  }

  if(terminating)
   printf("Terminating with ctrl-c\n");

  return 0;
}
```

D. Third Assignment: Client-Server Program Using UDP

```
//*********************************************************
//                    MUSTAFA GÖKTAN GÜDÜKBAY
//                         Siyah AR-GE
//*********************************************************

//C++ Implementation of the UDP server.


#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <string.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <arpa/inet.h>
#include <netinet/in.h>
#include <csignal>
#include <iostream>

using namespace std;

#define PORT     8080

//global variables
int shuttingDown = 0;

//graceful shutdown
void gracefulShutdown (int signum){
   shuttingDown = 1;
}

int main() {
  //variables
     int sockfd;
     char buffer[1024];
  pthread_t thread_id;
  int port;
  struct sockaddr_in servaddr, cliaddr;
  int n;
  socklen_t len;
  int echo;
  fd_set set;
  struct timeval timeout;
  int select_return_value;

  //code
  signal(SIGINT, gracefulShutdown);
  //Handling ctrl-c for graceful shutdown

  echo = 0;

  cout << "Server is opening. Enter the port number: ";
  cin >> port;
```

```
    // Creating socket file descriptor
    if ( (sockfd = socket(AF_INET, SOCK_DGRAM, 0)) < 0 ) {
         perror("socket creation failed");
         exit(EXIT_FAILURE);
    }

    memset(&servaddr, 0, sizeof(servaddr));
    memset(&cliaddr, 0, sizeof(cliaddr));

    // Filling server information
    servaddr.sin_family = AF_INET; // IPv4
    servaddr.sin_port = htons(port);
    servaddr.sin_addr.s_addr = inet_addr("127.0.0.1");

    // Bind the socket with the server address
    if ( bind(sockfd, (const struct sockaddr *)&servaddr,
              sizeof(servaddr)) < 0 )
    {
         perror("bind failed");
         exit(EXIT_FAILURE);
    }

    len = sizeof(cliaddr); //len is value/result

while(!shuttingDown){
  memset(buffer, 0, 1024);

  //check if there is something to read using select,
  //timeout is 100 microseconds
  timeout.tv_sec = 0;
  timeout.tv_usec = 100;
  FD_ZERO (&set);
  FD_SET (sockfd, &set);

  select_return_value =
         select(sockfd + 1, &set, NULL, NULL, &timeout);

  if(select_return_value <= 0)
    continue;

  n = recvfrom(sockfd, (char *)buffer, 1024,
        MSG_WAITALL, ( struct sockaddr *) &cliaddr,
        &len);
  buffer[n] = '\0';

  //client terminating
  if (n == 0)
    printf("A client is terminating.\n");

  else if(strcmp("Open", buffer) == 0)
   echo = 1;

  else if(strcmp("Close", buffer) == 0)
   echo = 0;
```

```
    else{
      printf("\nReceived Message: %s\n", buffer);
      //send back the message
      if(echo == 1)
        sendto(sockfd, buffer, strlen(buffer),
          MSG_CONFIRM, (const struct sockaddr *) &cliaddr,
          len);
    }
  }

  printf("Server is shutting down.\n");

  close(sockfd);
     return 0;
}
```

```
//C++ Implementation of the UDP server.


#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <string.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <arpa/inet.h>
#include <netinet/in.h>
#include <iostream>

using namespace std;

#define PORT     8080

//global variables
int terminating = 0;

//Handle ctrl-c
void signalHandler (int signum){
  terminating = 1;
}

int main() {
  //variables
      int sockfd;
      char buffer[1024];
      struct sockaddr_in servaddr;
  char ip[16];
  int port;
  int n;
  socklen_t len;
  fd_set set;
  struct timeval timeout;
  int select_return_value;
  int send_return_value;

  //code
  cout << "Enter the IP address: ";
  cin >> ip;

  cout << "Enter the Port number: ";
  cin >> port;

      // Creating socket file descriptor
      if ( (sockfd = socket(AF_INET, SOCK_DGRAM, 0)) < 0 ) {
            perror("socket creation failed");
            exit(EXIT_FAILURE);
      }

      memset(&servaddr, 0, sizeof(servaddr));

      // Filling server information
      servaddr.sin_family = AF_INET;
```

```
        servaddr.sin_port = htons(port);
        servaddr.sin_addr.s_addr = inet_addr(ip);;

    printf("Message to send ('q' to quit, 'Open' to open echo,
        'Close' to close echo)\n\n");

    while(!terminating){
      memset(buffer,0,1024);

      //check if there is echo
      timeout.tv_sec = 0;
      timeout.tv_usec = 100;
      FD_ZERO (&set);
      FD_SET (sockfd, &set);
      FD_SET (0, &set);

      select_return_value =
             select(sockfd + 1, &set, NULL, NULL, &timeout);

      if(select_return_value){

        if(FD_ISSET(0, &set)){
            cin.getline(buffer, sizeof(buffer));

            if (strcmp("q", buffer) == 0){
              printf("Terminating\n");
              break;
            }

            send_return_value = sendto(sockfd, buffer, strlen(buffer),
              MSG_CONFIRM, (const struct sockaddr *) &servaddr,
                sizeof(servaddr));

            printf("send_return_value: %d\n", send_return_value);

            if(send_return_value == -1)
              printf("Could not send the message.\n");
          }

          if(FD_ISSET(sockfd, &set)){
            memset(buffer,0,1024);

            n = recvfrom(sockfd, (char *)buffer, 1024,
                MSG_WAITALL, (struct sockaddr *) &servaddr,
                &len);

            buffer[n] = '\0';

            printf("ECHO: %s\n", buffer);
          }
        }
    }

        close(sockfd);
        return 0;
}
```

E. Message Queue Structure on Windows Using Shared Memory (Non–Blocking Send)

```
//**********************************************************
//                    MUSTAFA GÖKTAN GÜDÜKBAY
//                          Siyah AR-GE
//**********************************************************

//C++ Header for Message Queue Class


#ifndef __MESSAGE_QUEUE_H
#define __MESSAGE_QUEUE_H

#define MESSAGE_SIZE 256


#include <windows.h>
#include <string.h>
#include <tchar.h>
#include <cstdio>


struct Message {
  char buffer[MESSAGE_SIZE];
};

enum PermissionType {read, write, read_write};

class MessageQueue{
public:
    MessageQueue();
    ~MessageQueue();
    bool mq_open(TCHAR mappingObjectName[], DWORD
     maxNoOfMessages, PermissionType permission);
    bool mq_send(struct Message* message);
    bool mq_receive(struct Message* message);
    bool mq_close();

private:
    bool notOpened;
    HANDLE hMapFile;
    LPCTSTR pBuf;
    PermissionType permission;
    HANDLE mutex;
    HANDLE full;
};
#endif
```

```
//C++ Implementation of the Message Queue Class.


#include "MessageQueue.h"

MessageQueue::MessageQueue(){
    notOpened = true;
}

MessageQueue::~MessageQueue(){}

bool MessageQueue::mq_open(TCHAR mappingObjectName[], DWORD
maxNoOfMessages, PermissionType permission){

    this->permission = permission;

    int maxSize = maxNoOfMessages * sizeof(struct Message) +
           4*sizeof(DWORD);//rear and front of the queue

    TCHAR szName[200] = TEXT("Global\\");


     _tcscat_s(szName, 200, mappingObjectName);


    hMapFile = CreateFileMapping(
                 INVALID_HANDLE_VALUE, // use paging file
                 NULL,                 // default security
                 PAGE_READWRITE,       // read/write access
                 0,                    // maximum object size
                                       // (high-order DWORD)
                 maxSize,              // maximum object size
                                       //(low-order DWORD)
                 szName);              // name of mapping object

   if (hMapFile == NULL)
   {
      _tprintf(TEXT("Could not create file mapping object (%d).\n"),
             GetLastError());
      return false;
   }

    DWORD errorCode = GetLastError();

   if((permission == read) && (errorCode == ERROR_ALREADY_EXISTS)){
     //already a reader created the message queue.
      CloseHandle(hMapFile);
      return false;
   }

   if((permission == write) && (errorCode == ERROR_SUCCESS)){
     //a writer cannot create the message queue.
      CloseHandle(hMapFile);
      return false;
   }
```

```
pBuf = (LPTSTR) MapViewOfFile(hMapFile,
             FILE_MAP_ALL_ACCESS, // read/write permission
             0,
             0,
             maxSize);

if (pBuf == NULL)
{
   _tprintf(TEXT("Could not map view of file (%d).\n"),
          GetLastError());
    CloseHandle(hMapFile);
   return false;
}

 TCHAR emptyName[200] = TEXT("Global\\Empty");

  _tcscat_s(emptyName, 200, mappingObjectName);

 empty = CreateSemaphore(
     NULL,           // default security attributes
     maxNoOfMessages,  // initial count
     maxNoOfMessages,  // maximum count
     emptyName);

 if (empty == NULL){
    CloseHandle(hMapFile);
     _tprintf(TEXT("Semaphore Error (%d).\n"),
          GetLastError());
     CloseHandle(empty);
     return false;
 }

 TCHAR fullName[200] = TEXT("Global\\Full");

  _tcscat_s(fullName, 200, mappingObjectName);

 full = CreateSemaphore(
     NULL,           // default security attributes
     0,  // initial count
     maxNoOfMessages,  // maximum count
     fullName);


 if (full == NULL){
    CloseHandle(hMapFile);
     _tprintf(TEXT("Semaphore Error (%d).\n"),
          GetLastError());
     CloseHandle(empty);
     CloseHandle(full);
     return false;
 }

 TCHAR mutexName[200] = TEXT("Global\\Mutex");

  _tcscat_s(mutexName, 200, mappingObjectName);
```

```
    mutex = CreateMutex(
        NULL,                   // default security attributes
        FALSE,                  // initially not owned
        mutexName);

    if (mutex == NULL){
       CloseHandle(hMapFile);
        _tprintf(TEXT("Mutex Error (%d).\n"),
             GetLastError());
        CloseHandle(mutex);
        CloseHandle(empty);
        CloseHandle(full);
        return false;
    }

    if(permission == write)
        return true;

    CopyMemory((PVOID)pBuf, &maxNoOfMessages,
     sizeof(DWORD));//max number of messages

    DWORD size = 0;
    CopyMemory((PVOID)((PCHAR)pBuf + 1*sizeof(DWORD)), &size,
     sizeof(DWORD));//size

    DWORD front = 0;
    CopyMemory((PVOID)((PCHAR)pBuf + 2*sizeof(DWORD)), &front,
     sizeof(DWORD));//front

    int rear = -1;
    CopyMemory((PVOID)((PCHAR)pBuf + 3*sizeof(DWORD)), &rear,
     sizeof(int));//rear

    notOpened = false;

    return true;
}

bool MessageQueue::mq_send(struct Message* message){

    if(this->permission == read)
        return false;

    DWORD front, size, maxNoOfMessages;
    int rear;

    WaitForSingleObject(empty, INFINITE);
    WaitForSingleObject(mutex, INFINITE);

    CopyMemory((PVOID)(&maxNoOfMessages), (PVOID)pBuf,
     sizeof(DWORD));//maxNoOfMessages
    CopyMemory((PVOID)(&size), (PVOID)((PCHAR)pBuf +
     1*sizeof(DWORD)), sizeof(DWORD));//size
    CopyMemory((PVOID)(&front), (PVOID)((PCHAR)pBuf +
     2*sizeof(DWORD)), sizeof(DWORD));//front
    CopyMemory((PVOID)(&rear), (PVOID)((PCHAR)pBuf +
      3*sizeof(DWORD)), sizeof(int));//rear

    int indexToInsert = (rear + 1) % maxNoOfMessages;
    CopyMemory((PVOID)((PCHAR)pBuf + 4*sizeof(DWORD) +
      indexToInsert*MESSAGE_SIZE), message, sizeof(*message));

    //write rear and new size into the shared memory
    size++;

    rear++;
    CopyMemory((PVOID)((PCHAR)pBuf + 1*sizeof(DWORD)), &size,
      sizeof(DWORD));//size
    CopyMemory((PVOID)((PCHAR)pBuf + 3*sizeof(DWORD)), &rear,
      sizeof(int));//rear

    ReleaseMutex(mutex);
    ReleaseSemaphore(full, 1, NULL);

    return true;
}

bool MessageQueue::mq_receive(struct Message* message){

    if(this->permission == write)
        return false;

    DWORD size;

    WaitForSingleObject(full, INFINITE);
    WaitForSingleObject(mutex, INFINITE);

    CopyMemory((PVOID)(&size), (PVOID)((PCHAR)pBuf +
      1*sizeof(DWORD)), sizeof(DWORD));//get size

    DWORD front, maxNoOfMessages;
    CopyMemory((PVOID)(&front), (PVOID)((PCHAR)pBuf +
      2*sizeof(DWORD)), sizeof(DWORD));//get front
    CopyMemory((PVOID)(&maxNoOfMessages), (PVOID)pBuf,
      sizeof(DWORD));//get maxNoOfMessages

    int indexToDelete = front % maxNoOfMessages;

    CopyMemory((PVOID)(message->buffer), (PVOID)((PCHAR)pBuf +
      4*sizeof(DWORD) + indexToDelete*MESSAGE_SIZE),
      sizeof(*message));//front

   //write rear and new size into the shared memory

    size--;

    front++;
    CopyMemory((PVOID)((PCHAR)pBuf + 1*sizeof(DWORD)), &size,
      sizeof(DWORD));//size
    CopyMemory((PVOID)((PCHAR)pBuf + 2*sizeof(DWORD)), &front,
      sizeof(DWORD));//front
    ReleaseMutex(mutex);
    ReleaseSemaphore(empty, 1, NULL);

    return true;
}

bool MessageQueue::mq_close(){
    if(notOpened)
        return false;

    if(permission == write){
        return false;
    }

    CloseHandle(hMapFile);
    CloseHandle(empty);
    CloseHandle(hMapFile);
    CloseHandle(full);

    return true;
}
```

```
//Driver Code


#include <windows.h>
#include <tchar.h>
#include <strsafe.h>
#include "MessageQueue.h"
#include <time.h>

#define MAX_THREADS 5
TCHAR mappingObjectName[]=TEXT("MyFileMappingObject");
DWORD maxNoOfMessages = 5;

void rand_string(char *str)
{
    sprintf(str,"%d", GetCurrentThreadId());
}

DWORD WINAPI writerThread( LPVOID lpParam )
{
    Sleep(10000);
    MessageQueue mq;
    if(!(mq.mq_open(mappingObjectName, maxNoOfMessages, write))){
        printf("Message queue could not be opened.");
        return 1;
    }

    struct Message* m = new Message;
    memset(m->buffer, 0, MESSAGE_SIZE);

    while(TRUE){
        rand_string(m->buffer);
        if(!mq.mq_send(m)){
            printf("\nMessage was not sent.\n");
        }
        else{
            printf("\nMessage was sent.\n");
        }
        Sleep(5000);
    }
    mq.mq_close();
    return 0;
}

DWORD WINAPI readerThread( LPVOID lpParam )
{
    MessageQueue mq;
    if(!(mq.mq_open(mappingObjectName, maxNoOfMessages, read))){
        printf("Message queue could not be opened.");
        return 1;
    }

    struct Message* m = new Message;

    while(TRUE){
        if(mq.mq_receive(m)){
            printf("\nMessage was received. Message: %s \n",
                 m->buffer);
        }
        else{
            printf("Message was not received.");
        }
    }
    mq.mq_close();
    return 0;
}

int _tmain()
{

    HANDLE  hThreadArray[MAX_THREADS];
    DWORD   dwThreadIdArray[MAX_THREADS];

    hThreadArray[0] = CreateThread(NULL, 0, readerThread, NULL,
           0, &dwThreadIdArray[0]);

    if (hThreadArray[0] == NULL)
    {
        printf("Create thread failed.\n");
        ExitProcess(3);
    }


    printf("The test program starts. There are 1 reader thread
and %d writer threads.\n", MAX_THREADS - 1);

    for(int i = 1; i < MAX_THREADS; i++){
        hThreadArray[i] = CreateThread(NULL, 0, writerThread,
           NULL, 0, &dwThreadIdArray[i]);

        if (hThreadArray[i] == NULL)
        {
           printf("Create thread failed.\n");
           ExitProcess(3);
        }
    }

    WaitForMultipleObjects(MAX_THREADS, hThreadArray, TRUE,
           INFINITE);


    for(int i=0; i < MAX_THREADS; i++)
        CloseHandle(hThreadArray[i]);

    return 0;
}
```

F. Message Queue Structure on Windows Using Shared Memory (Blocking Send)

```
//*************************************************************
//                     MUSTAFA GÖKTAN GÜDÜKBAY
//                           Siyah AR-GE
//*************************************************************

//C++ Header for Message Queue Class

#ifndef __MESSAGE_QUEUE_H
#define __MESSAGE_QUEUE_H

#define MESSAGE_SIZE 256

#include <windows.h>
#include <string.h>
#include <tchar.h>
#include <cstdio>

struct Message {
  char buffer[MESSAGE_SIZE];
};

enum PermissionType {read, write, read_write};

class MessageQueue{
public:
    MessageQueue();
    ~MessageQueue();
    bool mq_open(TCHAR mappingObjectName[], DWORD maxNoOfMessages,
      PermissionType permission);
    bool mq_send(struct Message* message);
    bool mq_receive(struct Message* message);
    bool mq_close();

private:
    bool notOpened;
    HANDLE hMapFile;
    LPCTSTR pBuf;
    PermissionType permission;
    HANDLE empty;
    HANDLE mutex;
    HANDLE full;
};
#endif
```

```
//C++ Implementation of the Message Queue Class.

#include "MessageQueue.h"

MessageQueue::MessageQueue(){
    notOpened = true;
}

MessageQueue::~MessageQueue(){}

bool MessageQueue::mq_open(TCHAR mappingObjectName[], DWORD
maxNoOfMessages, PermissionType permission){

    this->permission = permission;

    int maxSize = maxNoOfMessages * sizeof(struct Message) +
4*sizeof(DWORD);//rear and front of the queue

    TCHAR szName[200] = TEXT("Global\\");


     _tcscat_s(szName, 200, mappingObjectName);

    hMapFile = CreateFileMapping(
                 INVALID_HANDLE_VALUE, // use paging file
                 NULL,                 // default security
                 PAGE_READWRITE,       // read/write access
                 0,                    // maximum object size
                                       // (high-order DWORD)
                 maxSize,              // maximum object size
                                       // (low-order DWORD)
                 szName);              // name of mapping object

    if (hMapFile == NULL)
   {
      _tprintf(TEXT("Could not create file mapping object (%d).\n"),
             GetLastError());
      return false;
   }

    DWORD errorCode = GetLastError();

   if((permission == read) && (errorCode == ERROR_ALREADY_EXISTS)){
      //already a reader created the message queue.
       CloseHandle(hMapFile);
       return false;
   }

   if((permission == write) && (errorCode == ERROR_SUCCESS)){
      //a writer cannot create the message queue.
       CloseHandle(hMapFile);
       return false;
   }

   pBuf = (LPTSTR) MapViewOfFile(hMapFile,
                  FILE_MAP_ALL_ACCESS, // read/write permission
                  0,
                  0,
                  maxSize);
```

```
if (pBuf == NULL)
{
   _tprintf(TEXT("Could not map view of file (%d).\n"),
          GetLastError());
    CloseHandle(hMapFile);
   return false;
}

 TCHAR emptyName[200] = TEXT("Global\\Empty");

  _tcscat_s(emptyName, 200, mappingObjectName);

 empty = CreateSemaphore(
     NULL,           // default security attributes
     maxNoOfMessages,  // initial count
     maxNoOfMessages,  // maximum count
     emptyName);

 if (empty == NULL){
    CloseHandle(hMapFile);
     _tprintf(TEXT("Semaphore Error (%d).\n"),
          GetLastError());
     CloseHandle(empty);
     return false;
 }

 TCHAR fullName[200] = TEXT("Global\\Full");

  _tcscat_s(fullName, 200, mappingObjectName);

 full = CreateSemaphore(
     NULL,           // default security attributes
     0,  // initial count
     maxNoOfMessages,  // maximum count
     fullName);


 if (full == NULL){
    CloseHandle(hMapFile);
     _tprintf(TEXT("Semaphore Error (%d).\n"),
          GetLastError());
     CloseHandle(empty);
     CloseHandle(full);
     return false;
 }

 TCHAR mutexName[200] = TEXT("Global\\Mutex");

  _tcscat_s(mutexName, 200, mappingObjectName);

 mutex = CreateMutex(
     NULL,                    // default security attributes
     FALSE,                   // initially not owned
     mutexName);

 if (mutex == NULL){
    CloseHandle(hMapFile);
     _tprintf(TEXT("Mutex Error (%d).\n"),
          GetLastError());
     CloseHandle(mutex);
     CloseHandle(empty);
```

```
            CloseHandle(full);
            return false;
        }

        if(permission == write)
            return true;

        CopyMemory((PVOID)pBuf, &maxNoOfMessages, sizeof(DWORD));
          //max number of messages

        DWORD size = 0;
        CopyMemory((PVOID)((PCHAR)pBuf + 1*sizeof(DWORD)), &size,
          sizeof(DWORD));//size

        DWORD front = 0;
        CopyMemory((PVOID)((PCHAR)pBuf + 2*sizeof(DWORD)), &front,
          sizeof(DWORD));//front

        int rear = -1;
        CopyMemory((PVOID)((PCHAR)pBuf + 3*sizeof(DWORD)), &rear,
          sizeof(int));//rear

        notOpened = false;

        return true;
    }

    bool MessageQueue::mq_send(struct Message* message){

        if(this->permission == read)
            return false;

        DWORD front, size, maxNoOfMessages;
        int rear;

        WaitForSingleObject(empty, INFINITE);
        WaitForSingleObject(mutex, INFINITE);

        CopyMemory((PVOID)(&maxNoOfMessages), (PVOID)pBuf,
          sizeof(DWORD));//maxNoOfMessages
        CopyMemory((PVOID)(&size), (PVOID)((PCHAR)pBuf + 1*sizeof(DWORD)),
          sizeof(DWORD));//size
        CopyMemory((PVOID)(&front), (PVOID)((PCHAR)pBuf + 2*sizeof(DWORD)),
          sizeof(DWORD));//front
        CopyMemory((PVOID)(&rear), (PVOID)((PCHAR)pBuf + 3*sizeof(DWORD)),
          sizeof(int));//rear

        int indexToInsert = (rear + 1) % maxNoOfMessages;
        CopyMemory((PVOID)((PCHAR)pBuf + 4*sizeof(DWORD) +
          indexToInsert*MESSAGE_SIZE), message, sizeof(*message));

        //write rear and new size into the shared memory
        size++;

        rear++;
        CopyMemory((PVOID)((PCHAR)pBuf + 1*sizeof(DWORD)), &size,
          sizeof(DWORD));//size
        CopyMemory((PVOID)((PCHAR)pBuf + 3*sizeof(DWORD)), &rear,
          sizeof(int));//rear
        ReleaseMutex(mutex);
        ReleaseSemaphore(full, 1, NULL);

        return true;
    }

    bool MessageQueue::mq_receive(struct Message* message){

        if(this->permission == write)
            return false;

        DWORD size;

        WaitForSingleObject(full, INFINITE);
        WaitForSingleObject(mutex, INFINITE);

        CopyMemory((PVOID)(&size), (PVOID)((PCHAR)pBuf + 1*sizeof(DWORD)),
          sizeof(DWORD));//get size

        DWORD front, maxNoOfMessages;
        CopyMemory((PVOID)(&front), (PVOID)((PCHAR)pBuf + 2*sizeof(DWORD)),
          sizeof(DWORD));//get front
        CopyMemory((PVOID)(&maxNoOfMessages), (PVOID)pBuf,
          sizeof(DWORD));//get maxNoOfMessages

        int indexToDelete = front % maxNoOfMessages;

        CopyMemory((PVOID)(message->buffer), (PVOID)((PCHAR)pBuf +
          4*sizeof(DWORD) + indexToDelete*MESSAGE_SIZE), sizeof(*message));
          //front

       //write rear and new size to the shared memory

        size--;

        front++;
        CopyMemory((PVOID)((PCHAR)pBuf + 1*sizeof(DWORD)), &size,
          sizeof(DWORD));//size
        CopyMemory((PVOID)((PCHAR)pBuf + 2*sizeof(DWORD)), &front,
          sizeof(DWORD));//front

        ReleaseMutex(mutex);
        ReleaseSemaphore(empty, 1, NULL);

        return true;
    }

    bool MessageQueue::mq_close(){
        if(notOpened)
            return false;

        if(permission == write){
            return false;
        }

        CloseHandle(hMapFile);
        CloseHandle(empty);
        CloseHandle(hMapFile);
        CloseHandle(full);

        return true;
    }
```

```
#include <windows.h>
#include <tchar.h>
#include <strsafe.h>
#include "MessageQueue.h"
#include <time.h>

#define MAX_THREADS 5
TCHAR mappingObjectName[]=TEXT("MyFileMappingObject");
DWORD maxNoOfMessages = 5;

void rand_string(char *str)
{
    /*srand(time(0));

    const char charset[] = "abcdefghijklmnopqrstuvwxyz0123456789";
    for (int n = 0; n < MESSAGE_SIZE; n++) {
        int key = rand() % (int) (sizeof charset - 1);
        str[n] = charset[key];
    }
    str[MESSAGE_SIZE] = '\0';*/

    sprintf(str,"%d", GetCurrentThreadId());
}

DWORD WINAPI writerThread( LPVOID lpParam )
{
    Sleep(10000);
    MessageQueue mq;
    if(!(mq.mq_open(mappingObjectName, maxNoOfMessages, write))){
        printf("Message queue could not be opened.");
        return 1;
    }

    struct Message* m = new Message;
    memset(m->buffer, 0, MESSAGE_SIZE);

    while(TRUE){
        rand_string(m->buffer);
        if(!mq.mq_send(m)){
            printf("\nMessage was not sent.\n");
        }
        Sleep(5000);
    }
    mq.mq_close();
    return 0;
}

DWORD WINAPI readerThread( LPVOID lpParam )
{
    MessageQueue mq;
    if(!(mq.mq_open(mappingObjectName, maxNoOfMessages, read))){
        printf("Message queue could not be opened.");
        return 1;
    }

    struct Message* m = new Message;

    while(TRUE){
        if(mq.mq_receive(m)){
            printf("\nMessage was received.\n");
            printf("%s\n\n", m->buffer);
        }
        else{
            printf("Message was not received.");
        }
        Sleep(5000);

    }
    mq.mq_close();
    return 0;
}

int _tmain()
{

    HANDLE  hThreadArray[MAX_THREADS];
    DWORD   dwThreadIdArray[MAX_THREADS];

    hThreadArray[0] = CreateThread(NULL, 0, readerThread, NULL, 0,
            &dwThreadIdArray[0]);

    if (hThreadArray[0] == NULL)
    {
        printf("Create thread failed.\n");
        ExitProcess(3);
    }

    for(int i = 1; i < MAX_THREADS; i++){
        hThreadArray[i] = CreateThread(NULL, 0, writerThread, NULL, 0,
            &dwThreadIdArray[i]);

        if (hThreadArray[i] == NULL)
        {
           printf("Create thread failed.\n");
           ExitProcess(3);
        }
    }

    WaitForMultipleObjects(MAX_THREADS, hThreadArray, TRUE, INFINITE);


    for(int i=0; i < MAX_THREADS; i++)
        CloseHandle(hThreadArray[i]);

    return 0;
}
```

```
#include <windows.h>
#include <tchar.h>
#include <strsafe.h>
#include "MessageQueue.h"
#include <time.h>

#define MAX_THREADS 5
TCHAR mappingObjectName[]=TEXT("MyFileMappingObject");
DWORD maxNoOfMessages = 5;

void rand_string(char *str)
{
    /*srand(time(0));

    const char charset[] = "abcdefghijklmnopqrstuvwxyz0123456789";
    for (int n = 0; n < MESSAGE_SIZE; n++) {
        int key = rand() % (int) (sizeof charset - 1);
        str[n] = charset[key];
    }
    str[MESSAGE_SIZE] = '\0';*/

    sprintf(str,"%d", GetCurrentThreadId());
}

DWORD WINAPI writerThread( LPVOID lpParam )
{
    Sleep(10000);
    MessageQueue mq;
    if(!(mq.mq_open(mappingObjectName, maxNoOfMessages, write))){
        printf("Message queue could not be opened.");
        return 1;
    }

    struct Message* m = new Message;
    memset(m->buffer, 0, MESSAGE_SIZE);

    while(TRUE){
        rand_string(m->buffer);
        if(!mq.mq_send(m)){
            printf("\nMessage was not sent.\n");
        }
        Sleep(5000);
    }
    mq.mq_close();
    return 0;
}

DWORD WINAPI readerThread( LPVOID lpParam )
{
    MessageQueue mq;
    if(!(mq.mq_open(mappingObjectName, maxNoOfMessages, read))){
        printf("Message queue could not be opened.");
        return 1;
    }

    struct Message* m = new Message;

    while(TRUE){
        if(mq.mq_receive(m)){
            printf("\nMessage was received.\n");
            printf("%s\n\n", m->buffer);
        }
        else{
            printf("Message was not received.");
        }
        Sleep(5000);

    }
    mq.mq_close();
    return 0;
}

int _tmain()
{

    HANDLE  hThreadArray[MAX_THREADS];
    DWORD   dwThreadIdArray[MAX_THREADS];

    hThreadArray[0] = CreateThread(NULL, 0, readerThread, NULL, 0,
&dwThreadIdArray[0]);

    if (hThreadArray[0] == NULL)
    {
        printf("Create thread failed.\n");
        ExitProcess(3);
    }

    for(int i = 1; i < MAX_THREADS; i++){
        hThreadArray[i] = CreateThread(NULL, 0, writerThread, NULL, 0,
            &dwThreadIdArray[i]);

        if (hThreadArray[i] == NULL)
        {
           printf("Create thread failed.\n");
           ExitProcess(3);
        }
    }

    WaitForMultipleObjects(MAX_THREADS, hThreadArray, TRUE, INFINITE);


    for(int i=0; i < MAX_THREADS; i++)
        CloseHandle(hThreadArray[i]);

    return 0;
}
```

Self-Checklist for Your Report

Please check the items here before submitting your report. This signed checklist should be the final page of your report.

- ✓ Did you provide detailed information about the work you did?
- ✓ Is supervisor information included?
- ✓ Did you use the Report Template to prepare your report, so that it has a cover page, has all sections and subsections specified in the Table of Contents, and uses the required section names?
- ✓ Did you follow the style guidelines?
- ✓ Does you report look professionally written?
- ✓ Does your report include all necessary References, and proper citations to them in the body?
- ✓ Did you remove all explanations from the Report Template, which are marked with yellow color? Did you modify all text marked with green according to your case?

Signature: MUSTAFA GÖKTAN GÜDÜKBAY

[illegible]