Київський національний університет імені Тараса Шевченка
Факультет комп’ютерних наук та кібернетики
Системний аналіз

Звіт
з лабораторної роботи
“Взаємодія розподілених процесів через механізм сокетів”

Виконав
студент(ка) групи К-23 (п/г 2) Флакей Р.Р.

Київ - 2022

Мета

В даній лабораторній роботі необхідно освоїти механізм (технологію) сокетів стеку протоколів TCP/IP, зокрема його реалізацію в MS Windows. Індивідуальний варіант роботи полягає в розробці двох програм (клієнта та сервера, які запускаються на різних станціях мережі), розробці протоколу обміну даними між ними та демонстрації роботи програм.

Зміст індивідуального завдання №18

Гра "вгадування 4-значного числа".

Користувач на клієнті вгадує 4-значне ціле, яке зберігається на сервері. З клієнта передаються 4 цифри, на які сервер дає відповідь із двох цифр: кількість правильних цифр та кількість цифр на своїх місцях. Клієнт в ході гри може її завершити, почати нову гру, завершити сеанс. Клієнт може здатися і тоді сервер розкриває загадане число. Користувач на клієнті може вибирати спосіб задання спроб: в діалозі вводити самому чи автоматична генерація випадкових 4-х значних чисел в заданій кількості.

Опис протоколу:

Будь-яка комунікація між сервером та клієнтом здійснюється у такому вигляді:

Заголовок	Дані

Довжина заголовка (в байтах)	**Команда1 [; Команда2; ...]**
1 байт	до 255 байт

При цьому клієнт і сервер розділяють свої набори команд, для передачі даних, або ж результату:

Server status code:

Сервер для здійснення обробки і відповідного регулювання доступу до певних функцій використовує спеціальні статус коди, які закріплюються за кожним Socket-client.

Нижче наведені відповідні пояснення загальних статус-кодів.

```
0 - Disconnected.
1 - Connect/Finish.
2 - Start/Restart.
```

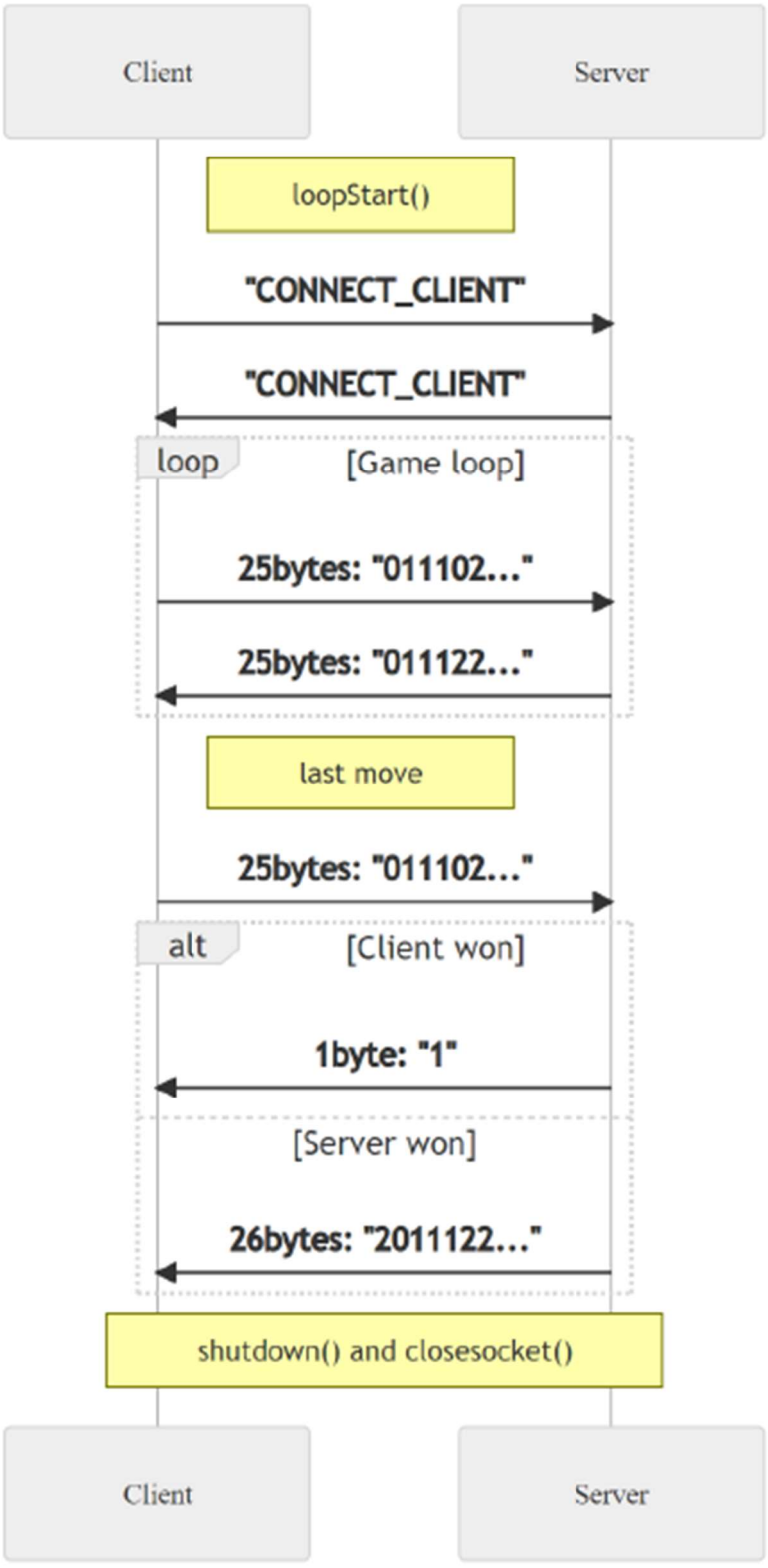
Клієнт (Client) *(команди які подаються від клієнта):*

- Команда **“s” (start)** розміром *1 байт* – команда, за допомогою якої здійснюється початок нової гри, або ж перезапуск, і відповідно генерація нового випадкового 4-значного числа.
- Команда **“fn” (finish)** розміром *2 байти* – команда, за допомогою якої здійснюється завершення існуючої гри достроково. У випадку якщо гра не почата, повертає сервер повертає результ-відповідь “f”.
- Команда **“gup” (give up)** розміром *3 байти* – команда, за допомогою якої можна достроково завершити гру (здатися), і при цьому дізнатися загаданий сервером код.
- Команда **“t_####”(try_number)** розміром *6 байтів* – команда, за допомогою якої можна спробувати при запущеній грі, вгадати яке число загадано сервером.

Сервер (Server) *(віповіді від серверу):*

- Команда **“gn-s”(generate start)** розміром *4 байти* – команда, яка позначає клієнту, що сервер згенерував нову гру, а й відповідно нові випадкові числа
- Команда **“gn-r”(generate restart)** розміром *4 байти* – команда, яка позначає клієнту, що сервер перегенерував нову гру, а й відповідно нові випадкові числа
- Команда **“f”(fail)** розміром *1 байт* – команда, яка позначає клієнту, що сервер повернув помилку у виконанні якоїсь дії
- Команда **“fn-s”(finish success)** розміром *4 байти* – команда, яка позначає клієнту, що сервер успішно завершив поточну гру
- Команда **“gu-s_####”(give up success)** розміром *9 байтів* – команда, яка позначає клієнту, що сервер успішно завершив поточну гру, і при цьому повернув 4-значний код, який був загаданий. *(# - [0,9])*
- Команда **“ws-fn”(Win success/finished)** розміром *5 байтів* – команда, яка позначає клієнту, що клієнт успішно завершив поточну гру, відгадавши загаданий 4-значний код.
- Команда **“tf_##”(Try failed)** розміром *5 байтів* – команда, яка позначає клієнту, що спроба відгадати була невдала. І повертає у першій # к-ть вірних чисел, а у другій # - к-ть тих, що на своїх місцях
- Команда **“tf-l”(Try failed/Limit)** розміром *4 байти* – команда, яка позначає клієнту, що передане число для спроби виходить за встановлені межі.
- Команда **“unxp_cm”(Unexpected command)** розміром *7 байтів* – команда, яка позначає клієнту, що сервер отримав невідому для нього команду.

Вигляд протоколу



Код до звіту:

Github - https://github.com/MinTins/lab-socket-server-client

Server.cpp

```cpp
/*
Client-server application
Author: Flakey Roman
Type: Server
Var: 18
Short name: Гра "вгадування 4-значного числа".
Task: Користувач на клієнті вгадує 4-значне ціле, яке зберігається на сервері.
    З клієнта передаються 4 цифри, на які сервер дає відповідь із двох цифр:
    кількість правильних цифр та кількість цифр на своїх місцях.
    Клієнт в ході гри може її завершити, почати нову гру, завершити сеанс.
    Клієнт може здатися і тоді сервер розкриває загадане число.
    Користувач на клієнті може вибирати спосіб задання спроб:
    в діалозі вводити самому чи автоматична генерація випадкових
    4-х значних чисел в заданій кількості.
*/

// Thanks to https://github.com/syncopika/winsock-server-client

#define _WIN32_WINNT 0x501 // fix some error

#include <winsock2.h>
#include <ws2tcpip.h>

#include <stdio.h>
#include <stdlib.h>
#include <iostream>
#include <string>
#include <fstream>

#include <vector>
#include <unordered_map>
#include <ctime>
#include <cstdlib>

#pragma comment(lib, "Ws2_32.lib")

#define DEFAULT_PORT 1043
#define DEFAULT_BUFLEN 255

SOCKET constructSocket()
{
```

```
    SOCKET sockRtnVal = socket(AF_INET, SOCK_STREAM, IPPROTO_TCP);
    if (sockRtnVal == INVALID_SOCKET)
    {
        printf("socket() failed\n");
        WSACleanup();
        exit(1);
    }
    return sockRtnVal;
};

void socketErrorCheck(int returnValue, SOCKET socketToClose, const char *action)
{
    const char *actionAttempted = action;
    if (returnValue == SOCKET_ERROR)
    {
        printf("Stop by error [%d] on %s", WSAGetLastError(), actionAttempted);
        closesocket(socketToClose); // WSACleanup(); exit(1);
    };
}

std::string getTimeNow()
{
    std::string result;

    std::time_t unixtime = std::time(nullptr);
    result = std::asctime(std::localtime(&unixtime));
    result.pop_back();

    return result;
};

void checkTryCode(short tryCode, short rightCode, char *outStr)
{
    int rightNumber = 0, rightPlace = 0;
    unsigned short rightNumberCode[4];

    for (short k = 0; k < 4; rightCode = rightCode / 10, k++)
    {
        rightNumberCode[k] = rightCode % 10;
    };

    for (short k = 0; k < 4; tryCode = tryCode / 10, k++)
    {
        short currNumb = tryCode % 10;

        for (short i = 0; i < 4; i++)
        {
            if (rightNumberCode[i] == currNumb)
            {
                rightNumber++;
                if (k == i)
                {
                    rightPlace++;
```

```
                };
                break;
            };
        };
    };

    std::sprintf(outStr, "tf_%d%d", rightNumber, rightPlace);
};

bool is_number(const std::string &s)
{
    std::string::const_iterator it = s.begin();
    while (it != s.end() && std::isdigit(*it))
        ++it;
    return !s.empty() && it == s.end();
};

int sendMessage(SOCKET &s, const char *sendbuf, std::ofstream& logfile)
{
    int sendResult;

    sendResult = send(s, sendbuf, DEFAULT_BUFLEN, 0);
    socketErrorCheck(sendResult, s, "send");
    logfile << getTimeNow() << " | SERVER >> socket-" << s << ":" << sendbuf <<
std::endl;
    logfile.flush();

    return sendResult;
};

void commandController(SOCKET &s, char *clientText, std::unordered_map<SOCKET,
short[2]> &socketToUserMap, std::ofstream &logfile)
{
    /*
    STATUS CODE:
    0 - Disconnected.
    1 - Connect/Finish.
    2 - Start/Restart.

    Send:
        s - start/restart
        fn - finish
        gup - give up
        t_#### - try 4-NUMB

    Return:
        gn-s - success generate start
        gn-r - success generate restart
        f - failed
        fn-s - finish success
        gu-s_#### - give up success _ 4-NUMB
        ws-fn - win success / finished game
        tf_## - try fail / # - number of right / # - in right place
```

```
        tf_l - try fail / limit
        unxp_cm - unexpected command
    */

    int sendbufsize = DEFAULT_BUFLEN;
    char sendbuf[sendbufsize] = "";

    std::string recvString(clientText);
    if (recvString == "Who")
    {
        strncpy(sendbuf, "Flakey Roman k-23, var 18. \"Guess the 4-number\"",
sendbufsize);
        sendMessage(s, sendbuf, logfile);
    }

    else if (recvString == "s")
    {
        if (socketToUserMap[s][0] > 0)
        {
            int randomNumber = (rand() % 9000) + 1000;
            socketToUserMap[s][1] = randomNumber;

            if (socketToUserMap[s][0] == 1)
            {
                socketToUserMap[s][0] = 2;
                strncpy(sendbuf, "gn-s", sendbufsize);
            }
            else
            {
                strncpy(sendbuf, "gn-r", sendbufsize);
            };
        }
        else
        {
            strncpy(sendbuf, "f", sendbufsize);
        };
        sendMessage(s, sendbuf, logfile);
    }

    else if (recvString == "fn")
    {
        if (socketToUserMap[s][0] == 2)
        {
            socketToUserMap[s][0] = 1;
            socketToUserMap[s][1] = 0;
            strncpy(sendbuf, "fn-s", sendbufsize);
        }
        else
        {
            strncpy(sendbuf, "f", sendbufsize);
        };
        sendMessage(s, sendbuf, logfile);
    }
```

```
    else if (recvString == "gup")
    {
        if (socketToUserMap[s][0] == 2)
        {
            std::string tempMsg("gu-s_" + std::to_string(socketToUserMap[s][1]));
            strncpy(sendbuf, tempMsg.c_str(), sendbufsize);
            socketToUserMap[s][0] = 1;
            socketToUserMap[s][1] = 0;
        }
        else
        {
            strncpy(sendbuf, "f", sendbufsize);
        };
        sendMessage(s, sendbuf, logfile);
    }

    else if (recvString.rfind("t_", 0) == 0)
    {

        if (socketToUserMap[s][0] == 2 && recvString.length() == 6 &&
is_number(recvString.substr(2, 4)))
        {
            int tryCode = stoi(recvString.substr(2, 4));
            if (tryCode >= 1000 && tryCode <= 9999)
            {
                if (tryCode == socketToUserMap[s][1])
                {
                    strncpy(sendbuf, "ws-fn", sendbufsize);
                    sendMessage(s, sendbuf, logfile);

                    socketToUserMap[s][0] = 1;
                    socketToUserMap[s][1] = 0;
                }
                else
                {
                    char resultCheck[6];
                    checkTryCode(tryCode, socketToUserMap[s][1], resultCheck);

                    strncpy(sendbuf, resultCheck, sendbufsize);
                    sendMessage(s, sendbuf, logfile);
                };
            }
            else
            {
                strncpy(sendbuf, "tf_l", sendbufsize);
                sendMessage(s, sendbuf, logfile);
            };
        }
        else
        {
            strncpy(sendbuf, "f", sendbufsize);
            sendMessage(s, sendbuf, logfile);
```

```
        };
    }

    else
    {
        strncpy(sendbuf, "unxp_cm", sendbufsize);
        sendMessage(s, sendbuf, logfile);
    };

    std::string().swap(recvString);
};

int main(void)
{
    WSADATA wsaData;

    std::ofstream logfile;
    logfile.open("server-log.txt", std::ios_base::app);

    int codeResult;
    int sendResult;
    char recvbuf[DEFAULT_BUFLEN] = "";
    int recvbuflen = DEFAULT_BUFLEN;

    SOCKET serverSocket = INVALID_SOCKET;
    SOCKET clientSocket = INVALID_SOCKET;

    struct sockaddr_in servAddr;
    struct sockaddr_in clientAddr;
    socklen_t clientAddrLen;

    fd_set activeFdSet;
    fd_set readFdSet;

    std::vector<SOCKET> socketsArray;
    std::unordered_map<SOCKET, short[2]> socketToUserMap;

    srand(time(NULL));

    codeResult = WSAStartup(MAKEWORD(2, 2), &wsaData);
    if (codeResult != NO_ERROR)
    {
        printf("WSAStartup failed with code %d\n", codeResult);
        return 1;
    }

    serverSocket = constructSocket();

    ZeroMemory(&servAddr, sizeof(servAddr));
    servAddr.sin_family = AF_INET;
    servAddr.sin_addr.s_addr = inet_addr("0.0.0.0");
    servAddr.sin_port = htons(DEFAULT_PORT);
```

```
    codeResult = bind(serverSocket, (struct sockaddr *)&servAddr,
sizeof(servAddr));
    socketErrorCheck(codeResult, serverSocket, "bind");

    codeResult = listen(serverSocket, SOMAXCONN);
    socketErrorCheck(codeResult, serverSocket, "listen");

    FD_ZERO(&activeFdSet);
    FD_SET(serverSocket, &activeFdSet);
    socketsArray.push_back(serverSocket);

    std::vector<SOCKET> newSocketArray(socketsArray);

    printf("waiting for connections...\n");

    while (true)
    {
        readFdSet = activeFdSet;

        codeResult = select(FD_SETSIZE, &readFdSet, NULL, NULL, NULL);
        socketErrorCheck(codeResult, serverSocket, "select");

        for (int i = 0; i < (int)socketsArray.size(); i++)
        {
            SOCKET currSocketFd = socketsArray.at(i);
            if (FD_ISSET(currSocketFd, &readFdSet))
            {
                if (currSocketFd == serverSocket)
                {
                    clientAddrLen = sizeof(clientAddr);
                    clientSocket = accept(serverSocket, (struct sockaddr
*)&clientAddr, &clientAddrLen);
                    if (clientSocket == INVALID_SOCKET)
                    {
                        printf("Stop by error [%d] on accept", WSAGetLastError());
                        closesocket(serverSocket);
                        WSACleanup();
                        exit(1);
                    }
                    else
                    {
                        printf("Got a connection socket-%d!\n", clientSocket);
                        logfile << getTimeNow() << " | SERVER-CONNECT [socket-" <<
clientSocket << "]" << std::endl;
                        logfile.flush();
                    };
                    socketToUserMap[clientSocket][0] = 1;

                    FD_SET(clientSocket, &activeFdSet);
                    newSocketArray.push_back(clientSocket);
                }
                else
                {
```

```
                    ZeroMemory(recvbuf, recvbuflen);
                    codeResult = recv(currSocketFd, recvbuf, recvbuflen, 0);

                    if (codeResult > 0)
                    {
                        logfile << getTimeNow() << " | SERVER << socket-" <<
currSocketFd << ":" << recvbuf << std::endl;
                        logfile.flush();

                        commandController(currSocketFd, recvbuf, socketToUserMap,
logfile);
                    }

                    else if (codeResult == 0 || codeResult == -1)
                    {
                        printf("Closed connection with socket-%d\n", currSocketFd);

                        logfile << getTimeNow() << " | SERVER-DISCONNECT [socket-"
<< currSocketFd << "]" << std::endl;
                        logfile.flush();

                        shutdown(currSocketFd, SD_SEND);
                        closesocket(currSocketFd);
                        FD_CLR(currSocketFd, &activeFdSet);

                        socketToUserMap.erase(currSocketFd);
                        std::vector<SOCKET> tempArr;
                        tempArr.push_back(serverSocket);
                        newSocketArray.assign(tempArr.begin(), tempArr.end());
                    }

                    else
                    {
                        printf("hmmm.. idk this result code %d\n", codeResult);
                    };
                    ZeroMemory(recvbuf, recvbuflen);
                };
            }
        };
        socketsArray.assign(newSocketArray.begin(), newSocketArray.end());
    };
    WSACleanup();
    logfile.close();
    return 0;
};
```

Client.cpp

```
/*
Client-server application
Author: Flakey Roman
Type: Client
Var: 18
Short name: Гра "вгадування 4-значного числа".
Task: Користувач на клієнті вгадує 4-значне ціле, яке зберігається на сервері.
    З клієнта передаються 4 цифри, на які сервер дає відповідь із двох цифр:
    кількість правильних цифр та кількість цифр на своїх місцях.
    Клієнт в ході гри може її завершити, почати нову гру, завершити сеанс.
    Клієнт може здатися і тоді сервер розкриває загадане число.
    Користувач на клієнті може вибирати спосіб задання спроб:
    в діалозі вводити самому чи автоматична генерація випадкових
    4-х значних чисел в заданій кількості.
*/

#define _WIN32_WINNT 0x501 // fix some error

#include <winsock2.h>

#include <iostream>
#include <fstream>
#include <string>
#include <stdio.h>

#include <ctime>
#include <cstdlib>

#define DEFAULT_PORT 1043
#define DEFAULT_BUFLEN 255


SOCKET constructSocket()
{
  SOCKET sock;
  sock = socket(AF_INET, SOCK_STREAM, IPPROTO_TCP);
  if (sock == INVALID_SOCKET)
  {
    printf("Error at socket() in constructSocket: %ld\n", WSAGetLastError());
    WSACleanup();
    return 0;
  };
  return sock;
}


void socketErrorCheck(int returnValue, SOCKET socketToClose, const char* action)
{
  const char *actionAttempted = action;
  if(returnValue == SOCKET_ERROR){
```

```
      if(WSAGetLastError() == WSAECONNREFUSED){
        printf("Server not found/refused the connection.\n");
      }
      else if(WSAGetLastError() == WSAECONNRESET){
        printf("Server closed the connection.\n");
      }
      else{
        printf("Stop by error [%d] on %s\n", WSAGetLastError(), actionAttempted);
      };

      closesocket(socketToClose); WSACleanup(); exit(1);
    };
};


std::string getTimeNow()
{
    std::string result;

    std::time_t unixtime = std::time(nullptr);
    result = std::asctime(std::localtime(&unixtime));
    result.pop_back();

    return result;
};


bool is_number(const std::string& s)
{
    std::string::const_iterator it = s.begin();
    while (it != s.end() && std::isdigit(*it)) ++it;
    return !s.empty() && it == s.end();
};


int sendMessage(SOCKET& s, std::string sendbuf, int& codeResult, std::ofstream&
logfile)
{
    codeResult = send(s, sendbuf.c_str(), sendbuf.size(), 0);
    socketErrorCheck(codeResult, s, "send");
    logfile << getTimeNow() << " | CLIENT -> SERVER" << ":" << sendbuf <<
std::endl;
    logfile.flush();

    return codeResult;
};


int recvMessage(SOCKET& s, char* recvbuf, int& recvbuflen, int& bytesRecv,
std::ofstream& logfile)
{
  bytesRecv = recv(s, recvbuf, recvbuflen, 0);
  if (bytesRecv == 0 || bytesRecv == WSAECONNRESET)
```

```
  {
     printf("Connection Closed.\n");
      return -1;
  };
  logfile << getTimeNow() << " | SERVER -> CLIENT" << ":" << recvbuf << std::endl;
  logfile.flush();

  std::string recvString(recvbuf);

  if (recvString == "gn-s") printf("<- SERVER: [GAME-STARTED] Code generated
success.\n");
  else if (recvString == "gn-r") printf("<- SERVER: [GAME-RESTARTED] Code generated
success.\n");
  else if (recvString == "f") printf("<- SERVER: [FAILED] Check your input
data/state.\n");
  else if (recvString == "fn-s") printf("<- SERVER: [FINISH] Game break by
command.\n");
  else if (recvString.rfind("gu-s_", 0) == 0) printf("<- SERVER: [GIVE-UP] Right
code was: %s\n", recvString.substr(5, 4).c_str());
  else if (recvString == "ws-fn") printf("<- SERVER: [WIN] Game win.\nInput: Start
- to start a new game.\n");
  else if (recvString.rfind("tf_", 0) == 0) printf("<- SERVER: [TRY-FAIL] Right
number - %c, In Right Place - %c\n", recvString[3], recvString[4]);
  else if (recvString == "f") printf("<- SERVER: [ERROR] Unexpected commmand.\n");
  else printf("<- SERVER: %s\n", recvString.c_str());

  return bytesRecv;
};



int main()
{
  WSADATA wsaData;

  std::ofstream logfile;
  logfile.open("client-log.txt", std::ios_base::app);

  int bytesRecv;
  int codeResult;

  char sendbuf[DEFAULT_BUFLEN] = "";
  char recvbuf[DEFAULT_BUFLEN] = "";
  int recvbuflen = DEFAULT_BUFLEN;

  std::string inputbuf;

  SOCKET ConnectSocket;

  srand(time(NULL));

  sockaddr_in clientService;
  clientService.sin_family = AF_INET;
```

```
  clientService.sin_addr.s_addr = inet_addr("127.0.0.1");
  clientService.sin_port = htons(DEFAULT_PORT);

  codeResult = WSAStartup(MAKEWORD(2, 2), &wsaData);
  if (codeResult != NO_ERROR)
  {
    printf("WSAStartup failed with code %d\n", codeResult);
    return 1;
  };

  ConnectSocket = constructSocket();

  codeResult = connect(ConnectSocket, (SOCKADDR *)&clientService,
sizeof(clientService));
  socketErrorCheck(codeResult, ConnectSocket, "connect");
  printf("[Connected to server.]\n");

  logfile << getTimeNow() << " | CLIENT-CONNECT]" << std::endl;
  logfile.flush();

  while (true)
  {
    std::string().swap(inputbuf);

    std::cout << "CLIENT ->: ";
    std::cin >> inputbuf;

    if (inputbuf == "Start")
    {
      sendMessage(ConnectSocket, "s", codeResult, logfile);
      bytesRecv = recvMessage(ConnectSocket, recvbuf, recvbuflen, bytesRecv,
logfile);
      if (bytesRecv == -1) break;
    }

    else if (inputbuf == "Who")
    {
      sendMessage(ConnectSocket, "Who", codeResult, logfile);
      bytesRecv = recvMessage(ConnectSocket, recvbuf, recvbuflen, bytesRecv,
logfile);
      if (bytesRecv == -1) break;
    }

    else if (inputbuf == "Restart")
    {
      sendMessage(ConnectSocket, "s", codeResult, logfile);
      bytesRecv = recvMessage(ConnectSocket, recvbuf, recvbuflen, bytesRecv,
logfile);
      if (bytesRecv == -1) break;
    }

    else if (inputbuf == "GiveUp")
    {
```

```
      sendMessage(ConnectSocket, "gup", codeResult, logfile);
      bytesRecv = recvMessage(ConnectSocket, recvbuf, recvbuflen, bytesRecv,
logfile);
      if (bytesRecv == -1) break;
    }

    else if (inputbuf == "Finish")
    {
      sendMessage(ConnectSocket, "fn", codeResult, logfile);
      bytesRecv = recvMessage(ConnectSocket, recvbuf, recvbuflen, bytesRecv,
logfile);
      if (bytesRecv == -1) break;
    }

    else if (inputbuf.rfind("TryCode_", 0) == 0 && inputbuf.length() == 12
    && is_number(inputbuf.substr(8, 4)))
    {
      sendMessage(ConnectSocket, "t_" + inputbuf.substr(8, 4), codeResult,
logfile);
      bytesRecv = recvMessage(ConnectSocket, recvbuf, recvbuflen, bytesRecv,
logfile);
      if (bytesRecv == -1) break;
    }

    else if (inputbuf.rfind("TryRand_", 0) == 0
    && is_number(inputbuf.substr(8, 5)))
    {
      unsigned int tryCount = stoi(inputbuf.substr(8, 5));

      for (int i = 0; i < tryCount; i++)
      {
        int randomNumber = (rand() % 9000) + 1000;

        std::string tempMsg("t_" + std::to_string(randomNumber));
        sendMessage(ConnectSocket, tempMsg.c_str(), codeResult, logfile);
        std::string().swap(tempMsg);

        bytesRecv = recvMessage(ConnectSocket, recvbuf, recvbuflen, bytesRecv,
logfile);
        if (bytesRecv == -1) break;

        if (recvbuf == "ws-fn")
        {
          printf("[RandomCode-check *SUCCESS*] Code was: %d\n", randomNumber);
          break;
        }
        else
        {
          printf("[Check-code] %d - FAIL\n", randomNumber);
        };

      };
    }
```

```
    else if (inputbuf == "Exit")
    {
      printf("Closing the connection...\n");
      break;
    }

    else
    {
      printf("  Availible command:\n"
      "    Start/Restart - start game-session\n"
      "    Finish - stop game session\n"
      "    GiveUp - stop game and tell hidden number\n"
      "    TryCode_#### - try check own code | # - [0-9]\n"
      "    TryRand_# - try check random code | # - count of random code\n"
      "    Exit - close connection.\n");
    };
    std::string().swap(inputbuf);
  };
  closesocket(ConnectSocket);
  WSACleanup();
  printf("[Connection closed.]\n");
  logfile << getTimeNow() << " | CLIENT-DISCONNECT]" << std::endl;
  logfile.flush();
  return 0;
}
```

Системний журнал дій

Client.cpp

Fri Sep 30 11:49:21 2022 | CLIENT-CONNECT]

Fri Sep 30 11:49:23 2022 | CLIENT -> SERVER:Who

Fri Sep 30 11:49:23 2022 | SERVER -> CLIENT:Flakey Roman k-23, var 18. "Guess the 4-number"

Fri Sep 30 11:49:24 2022 | CLIENT -> SERVER:s

Fri Sep 30 11:49:24 2022 | SERVER -> CLIENT:gn-s

Fri Sep 30 11:49:27 2022 | CLIENT -> SERVER:t_8936

Fri Sep 30 11:49:27 2022 | SERVER -> CLIENT:tf_33

Fri Sep 30 11:49:27 2022 | CLIENT -> SERVER:t_8938

Fri Sep 30 11:49:27 2022 | SERVER -> CLIENT:tf_43

Fri Sep 30 11:49:27 2022 | CLIENT -> SERVER:t_2750

Fri Sep 30 11:49:27 2022 | SERVER -> CLIENT:tf_10

Fri Sep 30 11:49:27 2022 | CLIENT -> SERVER:t_6246

Fri Sep 30 11:49:27 2022 | SERVER -> CLIENT:tf_10

Fri Sep 30 11:49:27 2022 | CLIENT -> SERVER:t_9098

Fri Sep 30 11:49:27 2022 | SERVER -> CLIENT:tf_30

Fri Sep 30 11:49:27 2022 | CLIENT -> SERVER:t_3287

Fri Sep 30 11:49:27 2022 | SERVER -> CLIENT:tf_30

Fri Sep 30 11:49:27 2022 | CLIENT -> SERVER:t_3515

Fri Sep 30 11:49:27 2022 | SERVER -> CLIENT:tf_10

Fri Sep 30 11:49:27 2022 | CLIENT -> SERVER:t_7486

Fri Sep 30 11:49:27 2022 | SERVER -> CLIENT:tf_10

Fri Sep 30 11:49:27 2022 | CLIENT -> SERVER:t_1339

Fri Sep 30 11:49:27 2022 | SERVER -> CLIENT:tf_31

Fri Sep 30 11:49:27 2022 | CLIENT -> SERVER:t_3532

Fri Sep 30 11:49:27 2022 | SERVER -> CLIENT:tf_32

Fri Sep 30 11:49:29 2022 | CLIENT-DISCONNECT]

Server.cpp

Fri Sep 30 11:49:21 2022 | SERVER-CONNECT [socket-356]

Fri Sep 30 11:49:23 2022 | SERVER << socket-356:Who

Fri Sep 30 11:49:23 2022 | SERVER >> socket-356:Flakey Roman k-23, var 18. "Guess the 4-number"

Fri Sep 30 11:49:24 2022 | SERVER << socket-356:s

Fri Sep 30 11:49:24 2022 | SERVER >> socket-356:gn-s

Fri Sep 30 11:49:27 2022 | SERVER << socket-356:t_8936

Fri Sep 30 11:49:27 2022 | SERVER >> socket-356:tf_33

Fri Sep 30 11:49:27 2022 | SERVER << socket-356:t_8938

Fri Sep 30 11:49:27 2022 | SERVER >> socket-356:tf_43

Fri Sep 30 11:49:27 2022 | SERVER << socket-356:t_2750

Fri Sep 30 11:49:27 2022 | SERVER >> socket-356:tf_10

Fri Sep 30 11:49:27 2022 | SERVER << socket-356:t_6246

Fri Sep 30 11:49:27 2022 | SERVER >> socket-356:tf_10

Fri Sep 30 11:49:27 2022 | SERVER << socket-356:t_9098

Fri Sep 30 11:49:27 2022 | SERVER >> socket-356:tf_30

Fri Sep 30 11:49:27 2022 | SERVER << socket-356:t_3287

Fri Sep 30 11:49:27 2022 | SERVER >> socket-356:tf_30

Fri Sep 30 11:49:27 2022 | SERVER << socket-356:t_3515

Fri Sep 30 11:49:27 2022 | SERVER >> socket-356:tf_10

Fri Sep 30 11:49:27 2022 | SERVER << socket-356:t_7486

Fri Sep 30 11:49:27 2022 | SERVER >> socket-356:tf_10

Fri Sep 30 11:49:27 2022 | SERVER << socket-356:t_1339

Fri Sep 30 11:49:27 2022 | SERVER >> socket-356:tf_31

Fri Sep 30 11:49:27 2022 | SERVER << socket-356:t_3532

Fri Sep 30 11:49:27 2022 | SERVER >> socket-356:tf_32

Fri Sep 30 11:49:29 2022 | SERVER-DISCONNECT [socket-356]