Київський національний університет імені Тараса Шевченка
Факультет комп’ютерних наук та кібернетики
Кафедра системного аналізу та теорії прийняття рішень

Звіт
з лабораторної роботи № 1
на тему:

«Взаємодія розподілених процесів через механізм сокетів»

Варіант 12

Студента другого курсу
групи К-23(2)
Міщука Романа Андрійовича
Факультету комп’ютерних наук
та кібернетики

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

Гра "хрестики-нолики" на полі 5x5 клітинок. Один гравець - на сервері, другий - на клієнті. Поле (матрицю символів) до клієнта передає сервер. Сервер робить хід у будь-яку пусту клітинку, стратегія не потрібна, перемогу/програш не відслідковує. Після кожного ходу клієнта сервер передає поле, додавши свій хід. Клієнт в ході гри може її завершити в будь-який момент.

Опис протоколу

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

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

При цьому, існують такі види команд:

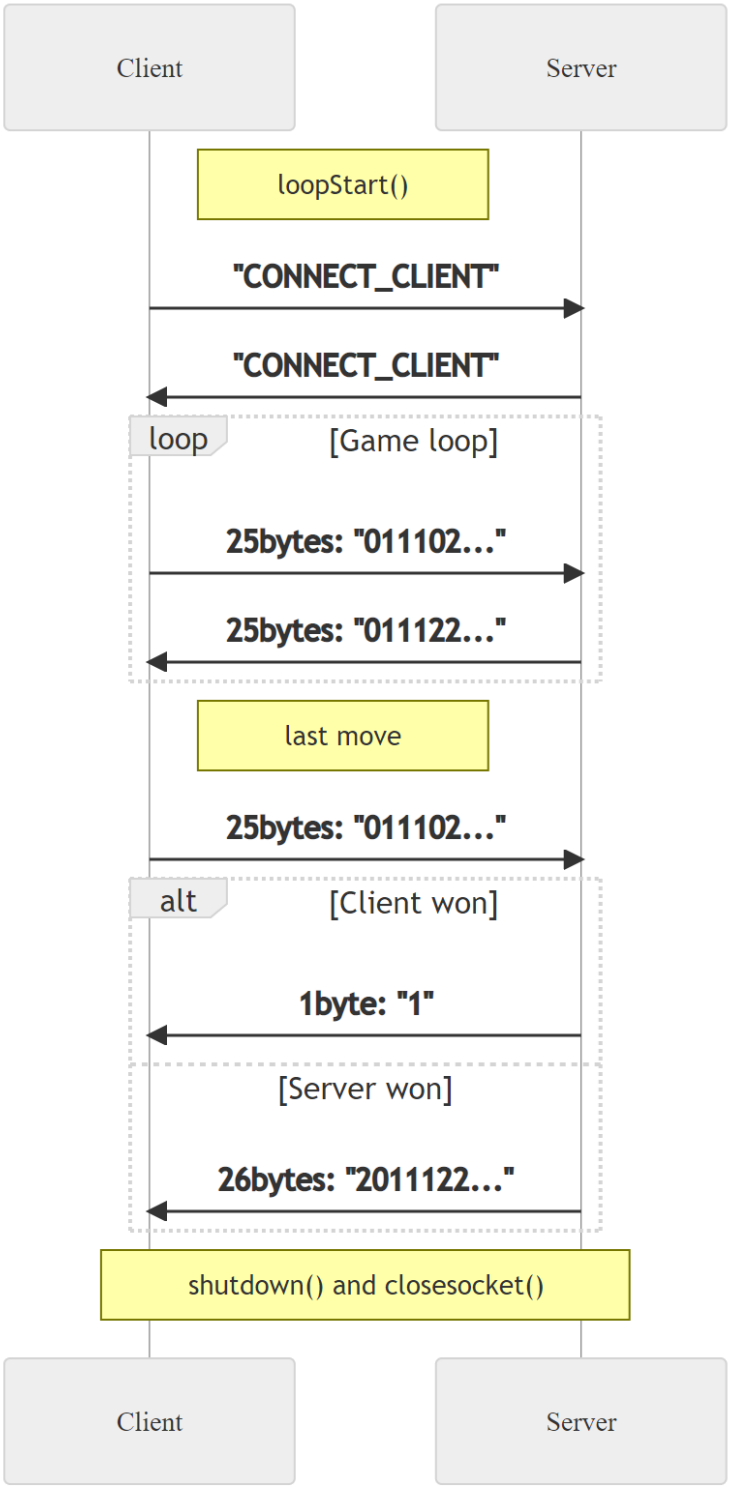
- `"CONNECT_CLIENT"` – команда, за допомогою якої здійснюється запит за з'єднання. Перед початком обміну повідомленнями, клієнт та сервер обмінюються цією командою.
- Команда розміром *25 байт* – команда, що містить інформацію про теперішній стан ігрового поля. Має вигляд послідовності символів *0, 1* та *2*, де 0 – вільна клітинка, 1 – клітинка, зайнята клієнтом, 2 – клітинка, зайнята сервером. Коли хтось робить хід, він змінює значення відповідної комірки, та надсилає оновлений вигляд поля супротивнику.
- Команда розміром *1 байт* – команда, що може бути надіслана лише сервером, та містить у собі інформацію про результат гри. При цьому сервер виступає суддею, і надсилає символ *1* коли переміг клієнт, та *2* коли переміг сервер.
- Команда розміром *26 байт* – команда, що може бути надіслана лише сервером, і містить у собі інформацію про результат гри. Є

комбінацією команд на *1 байт* та на *25 байтів*. Надсилається у випадку, коли сервер переміг, і в такому випадку клієнт не знає, як виглядає вирішальний хід.

Закриття з’єднання відбувається за допомогою відповідних функцій `shutdown(socket, SD_SEND); closesocket(socket);`, що сповіщає як клієнта, так і сервер.

При надсиланні клієнтом неправильної команди, приймається рішення, що він програв, надсилається відповідна команда, та з’єднання закривається.

Вигляд протоколу



Код pch.h

```
// pch.h: This is a precompiled header file.
// Files listed below are compiled only once, improving build performance for future builds.
// This also affects IntelliSense performance, including code completion and many code
browsing features.
// However, files listed here are ALL re-compiled if any one of them is updated between
builds.
// Do not add files here that you will be updating frequently as this negates the
performance advantage.

#ifndef PCH_H
#define PCH_H

// add headers that you want to pre-compile here
#include "framework.h"

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

#endif //PCH_H
```

Код WinSockWrapper.h

```
#pragma once
#include "pch.h"
#include <string>

class WinSockWrapper
{
private:
	static bool was_initialised;

	static void init();

public:
	static void ensureInit();
	static void close();

	static SOCKET getSocketForAddress(std::string address, std::string port);
	static std::string getIpStringFromAdress(addrinfo* info);
};
```

Код WinSockWrapper.cpp

```
#include "pch.h"
#include "WinSockWrapper.h"

#include <iostream>
#include <stdexcept>

using namespace std;

// Wrapper -----
bool WinSockWrapper::was_initialised = false;

void WinSockWrapper::init()
{
	WSADATA wsaData;
	if (WSAStartup(MAKEWORD(2, 2), &wsaData) != 0) {
		throw runtime_error("WSAStartup failed.");
	}
```

```
}

void WinSockWrapper::ensureInit()
{
       if(was_initialised) return;
       init();
       was_initialised = true;
}

void WinSockWrapper::close()
{
    if (!was_initialised) return;
       WSACleanup();
       was_initialised = false;
}

SOCKET WinSockWrapper::getSocketForAddress(string address, string port)
{
    int ret;

    addrinfo specs;
    ZeroMemory(&specs, sizeof(specs));
    specs.ai_family = AF_UNSPEC;
    specs.ai_socktype = SOCK_STREAM;
    specs.ai_protocol = IPPROTO_TCP;

    // RESOLVING THE HOSTNAME/ADDRESS
    addrinfo* results_list;
    if ((ret = getaddrinfo(address.c_str(), port.c_str(), &specs, &results_list)) != 0)
        return INVALID_SOCKET;

    SOCKET buff = INVALID_SOCKET;
    for (addrinfo* i = results_list; i != NULL; i = i->ai_next)
    {
        // SOCKET
        buff = socket(i->ai_family, i->ai_socktype, i->ai_protocol);

        // CONNECT
        if ((ret = connect(buff, i->ai_addr, (int)i->ai_addrlen)) != 0)
        {
            closesocket(buff);
            buff = INVALID_SOCKET;
            continue;
        }
        break;
    }
    freeaddrinfo(results_list);
    return buff;
}

string WinSockWrapper::getIpStringFromAdress(addrinfo* info)
{
    string ret(INET_ADDRSTRLEN, 0);
    inet_ntop(info->ai_family, &(((sockaddr_in*)info->ai_addr)->sin_addr), (PSTR)
ret.c_str(), ret.size());
    return ret;
}
```

Код APeer.h

```
// ReSharper disable CppInconsistentNaming
#pragma once
#include <atomic>
#include <memory>

#include "WinSockWrapper.h"

static constexpr char PORT_NUMBER[5] = "1037";
static constexpr int MAX_COMMAND_SIZE = 255;
static constexpr int RETRY_NUM = 5;

static constexpr char START_MESSAGE[15] = "CONNECT_CLIENT";
static constexpr int START_MESSAGE_SIZE = sizeof(START_MESSAGE);

class APeer  // NOLINT(cppcoreguidelines-special-member-functions)
{
public:
	struct AContext  // NOLINT(cppcoreguidelines-special-member-functions)
	{ virtual ~AContext() = 0; } **context;

protected:
	std::atomic<bool> working;
	SOCKET socket;

	struct ASockResult
	{
		enum Type { OK, SHUTDOWN, ERR } type;
		ASockResult(Type type) : type(type) {}
	};
	struct ErrSockResult : ASockResult
	{
		int code;
		ErrSockResult(int code) : ASockResult(ERR), code(code) {}
	};
	enum ContactType { SEND, RECEIVE };
	std::unique_ptr<ASockResult> contact(
		ContactType type, char* data, int data_len) const;

	friend class APeerManager;
	static std::unique_ptr<ASockResult> contact(
		const SOCKET& socket, ContactType type, char* data, int data_len);

	virtual std::unique_ptr<ASockResult> loopStart(std::string& receiveBuffer,
std::string& sendBuffer) = 0;
	std::unique_ptr<ASockResult> loopIterBody(std::string& receiveBuffer, std::string&
sendBuffer);
	virtual std::unique_ptr<ASockResult> loopEnd(std::unique_ptr<ASockResult> reason);

public:
	APeer();
	virtual ~APeer();

	bool setSocket(SOCKET socket);
	bool setSocket(std::string connectionAddress, std::string connectionPort);

	std::unique_ptr<ASockResult> loop();

	struct HandlerResponse
	{
		std::string message;
		enum Flags
		{
			EMPTY = 0,
			MESSAGE = 1 << 0,
```

```
                SHUTDOWN = 1 << 1
            } flags;

            HandlerResponse(std::string message);
            HandlerResponse(bool is_shutdown, std::string message);
        };
        virtual HandlerResponse messageHandler(AContext** context, std::string message) = 0;
};
```

Код APeer.cpp

```
// ReSharper disable CppClangTidyClangDiagnosticCastQual
#include "pch.h"
#include "APeer.h"

#include <iostream>
#include <stdexcept>

using namespace std;

DEFINE_ENUM_FLAG_OPERATORS(APeer::HandlerResponse::Flags)


// https://www.geeksforgeeks.org/pure-virtual-destructor-c/
APeer::AContext::~AContext() = default;

unique_ptr<APeer::ASockResult> APeer::contact(ContactType type, char* data, int data_len)
const
{ return contact(socket, type, data, data_len); }

unique_ptr<APeer::ASockResult> APeer::contact(const SOCKET& socket, ContactType type, char*
data, int data_len)
{
    int result = 0, code, i = 0;
    do
    {
        if (type == RECEIVE) result = recv(socket, data, data_len, 0);
        else if (type == SEND) result = send(socket, data, data_len, 0);
        code = WSAGetLastError();

#ifdef _DEBUG
        if (type == RECEIVE) cout << "r" << result << '\"' << data << '\"' << "\n";
        else if (type == SEND) cout << "s" << result << '\"' << data << '\"' << "\n";
#endif

        if (result == data_len) return make_unique<ASockResult>(ASockResult::OK);

        // https://learn.microsoft.com/uk-ua/windows/win32/winsock/windows-sockets-error-
codes-2?redirectedfrom=MSDN
        if (result == 0 || result == WSAESHUTDOWN || code == WSAECONNRESET)
            return make_unique<ASockResult>(ASockResult::SHUTDOWN);

        i++;
    } while (i < RETRY_NUM);
    return make_unique<ErrSockResult>(code);
}

unique_ptr<APeer::ASockResult> APeer::loopIterBody(string& receiveBuffer, string&
sendBuffer)
{
    unique_ptr<ASockResult> buff_res;

    // Receiving phase
    if ((buff_res = contact(RECEIVE, (char*) receiveBuffer.c_str(), 1)
        )->type != ASockResult::OK || (unsigned char)receiveBuffer[0] < 1)
```

```
        return buff_res;

    receiveBuffer.resize((unsigned char) receiveBuffer[0]);
    if ((buff_res = contact(RECEIVE, (char*) receiveBuffer.c_str(), (unsigned char)
receiveBuffer[0])
        )->type != ASockResult::OK)
        return buff_res;

    // Processing phase
    HandlerResponse response = messageHandler(context, receiveBuffer);
    sendBuffer = move(response.message);
    if (response.flags & HandlerResponse::SHUTDOWN)
    {
        // In case, we need to shut down the connection
        if (response.flags & HandlerResponse::MESSAGE)
        {
            sendBuffer = (char) sendBuffer.size() + sendBuffer;
            if ((buff_res = contact(SEND, (char*) sendBuffer.c_str(),
static_cast<int>(sendBuffer.size()))
                )->type != ASockResult::OK)
                return buff_res;
        }
        return make_unique<ASockResult>(ASockResult::SHUTDOWN);
    }

    // Answering phase
    sendBuffer = (char) sendBuffer.size() + sendBuffer;
    if ((buff_res = contact(SEND, (char*) sendBuffer.c_str(),
static_cast<int>(sendBuffer.size()))
        )->type != ASockResult::OK)
        return buff_res;

    return buff_res;
}

std::unique_ptr<APeer::ASockResult> APeer::loopEnd(std::unique_ptr<ASockResult> reason)
{
    // Shutting down the connection, since we're done
    shutdown(socket, SD_SEND);
    closesocket(socket);
    return reason;
}

APeer::APeer(): socket(INVALID_SOCKET)
{
    context = new AContext * (nullptr);
    working = false;
}

APeer::~APeer()
{
       delete *context;
    delete context;
}

bool APeer::setSocket(SOCKET socket)  // NOLINT(clang-diagnostic-shadow)
{
    this->socket = socket;
    if(socket == INVALID_SOCKET) return false;

    // https://stackoverflow.com/questions/30395258/setting-timeout-to-recv-function
    // setsockopt(socket, SOL_SOCKET, SO_RCVTIMEO, (char*)&TIMEOUT_MS, sizeof(TIMEOUT_MS));

    return true;
}

bool APeer::setSocket(std::string connectionAddress, std::string connectionPort)
{
```

```
    WinSockWrapper::ensureInit();
    const SOCKET buff = WinSockWrapper::getSocketForAddress(
        move(connectionAddress), move(connectionPort));
    return setSocket(buff);
}

std::unique_ptr<APeer::ASockResult> APeer::loop()
{
    if(socket == INVALID_SOCKET)
        return make_unique<ErrSockResult>(INVALID_SOCKET);

    working = true;

    unique_ptr<ASockResult> buffRes;

    string receiveBuffer, sendBuffer;
    receiveBuffer.reserve(MAX_COMMAND_SIZE);
    sendBuffer.reserve(MAX_COMMAND_SIZE);
    receiveBuffer.resize(START_MESSAGE_SIZE);

    // Starting
    if ((buffRes = loopStart(receiveBuffer, sendBuffer)
        )->type != ASockResult::OK)
        return loopEnd(move(buffRes));

    // The Loop
    while(working)
    {
           if((buffRes = loopIterBody(receiveBuffer, sendBuffer)
            )->type != ASockResult::OK)
            break;
    }

    // The ending
    return loopEnd(move(buffRes));
}

APeer::HandlerResponse::HandlerResponse(std::string message):
       message(message), flags(MESSAGE) {}

APeer::HandlerResponse::HandlerResponse(bool is_shutdown, std::string message): flags(EMPTY)
{
    if (!message.empty())
    {
        this->message = message;
        flags |= MESSAGE;
    }
    if (is_shutdown) flags |= SHUTDOWN;
}
```

Код AServer.h

```
#pragma once

#include <mutex>
#include <queue>
#include <set>
#include <thread>

#include "APeer.h"
#include "WinSockWrapper.h"

class AServer
{
protected:
	std::atomic<bool> working;
	SOCKET listenSocket = INVALID_SOCKET;

	class Peer : public APeer
	{
	private:
		AServer* server;
		std::thread* thread;

		std::unique_ptr<ASockResult> loopEnd(std::unique_ptr<ASockResult> reason)
override;
		std::unique_ptr<ASockResult> loopStart(std::string& receiveBuffer,
std::string& sendBuffer) override;

	public:
		Peer(AServer* server, SOCKET socket);
		virtual ~Peer();

		HandlerResponse messageHandler(AContext** context, std::string message)
override;
	};

	std::set<Peer*> peers;
	std::mutex finishedPeersMutex;
	std::queue<Peer*> finishedPeers;

	static void resolveAddress(const std::string& address, const std::string& port,
addrinfo*& result);

public:
	AServer(std::string address, std::string port);

	void loop();

	virtual APeer::HandlerResponse messageHandler(APeer::AContext** context, std::string
message) = 0;
};
```

Код AServer.cpp

```
// ReSharper disable CppCStyleCast
// ReSharper disable CppClangTidyClangDiagnosticCastQual
#include "pch.h"
#include "AServer.h"

using namespace std;
```

```cpp
std::unique_ptr<APeer::ASockResult> AServer::Peer::loopEnd(std::unique_ptr<ASockResult>
reason)
{
    reason = APeer::loopEnd(move(reason));

    // Signalling work end
    server->finishedPeersMutex.lock();
    server->finishedPeers.push(this);
    server->finishedPeersMutex.unlock();

    return reason;
}

std::unique_ptr<APeer::ASockResult> AServer::Peer::loopStart(std::string& receiveBuffer,
std::string& sendBuffer)
{
    unique_ptr<ASockResult> buffRes;

    // Receiving start message from the client
    if ((buffRes = contact(RECEIVE, (char*)receiveBuffer.c_str(), 1)
        )->type != ASockResult::OK || (unsigned char)receiveBuffer[0] != START_MESSAGE_SIZE)
    {
        if (buffRes->type != ASockResult::OK) return buffRes;
        return make_unique<ASockResult>(ASockResult::SHUTDOWN);
    }
    if ((buffRes = contact(RECEIVE, (char*)receiveBuffer.c_str(), START_MESSAGE_SIZE)
        )->type != ASockResult::OK || strcmp(receiveBuffer.c_str(), START_MESSAGE) != 0)
    {
        if (buffRes->type != ASockResult::OK) return buffRes;
        return make_unique<ASockResult>(ASockResult::SHUTDOWN);
    }

    // Responding him with the confirmation message
    if ((buffRes = contact(SEND, (char*)&START_MESSAGE_SIZE, 1)
        )->type != ASockResult::OK)
        return buffRes;
    if ((buffRes = contact(SEND, (char*)START_MESSAGE, START_MESSAGE_SIZE)
        )->type != ASockResult::OK)
        return buffRes;

    return buffRes;
}

AServer::Peer::Peer(AServer* server, SOCKET socket) :  // NOLINT(clang-diagnostic-shadow-
field)
    APeer(), server(server)
{
    // https://stackoverflow.com/questions/10998780/stdthread-calling-method-of-class
    thread = new std::thread(&Peer::loop, this);
    setSocket(socket);
}

AServer::Peer::~Peer()
{
    thread->join();
    delete thread;
}

APeer::HandlerResponse AServer::Peer::messageHandler(AContext** context, std::string
message)  // NOLINT(clang-diagnostic-shadow-field)
{
    return server->messageHandler(context, move(message));
}

void AServer::resolveAddress(const std::string& address, const std::string& port, addrinfo*&
result)
{
    addrinfo specs;  // NOLINT(cppcoreguidelines-pro-type-member-init)
```

```
    ZeroMemory(&specs, sizeof(specs));
    specs.ai_family = AF_INET;
    specs.ai_socktype = SOCK_STREAM;
    specs.ai_protocol = IPPROTO_TCP;
    specs.ai_flags = AI_PASSIVE;

    // https://docs.microsoft.com/en-us/windows/win32/api/ws2tcpip/nf-ws2tcpip-
getaddrinfo#:~:text=If%20the-,pNodeName,-
parameter%20points%20to%20a%20string%20equal%20to%20%22localhost
    const int ret = getaddrinfo(
           address.empty() ? NULL : address.c_str(),
        port.c_str(), &specs, &result
    );
}

AServer::AServer(std::string address, std::string port) : working(false)
{
    int ret;

    WinSockWrapper::ensureInit();

    // Resolve the server address and port
    addrinfo* result = nullptr;
    resolveAddress(address, port, result);

    // Create a SOCKET for the server to listen connections
    listenSocket = socket(result->ai_family, result->ai_socktype, result->ai_protocol);
    if (listenSocket == INVALID_SOCKET) {
        freeaddrinfo(result);
        throw runtime_error("listenSocket failed with error: " +
to_string(WSAGetLastError()));
    }

    // Setup the TCP listening listenSocket
    ret = bind(listenSocket, result->ai_addr, (int)result->ai_addrlen);
    freeaddrinfo(result);

    if (ret == SOCKET_ERROR) {
        closesocket(listenSocket);
        throw runtime_error("bind failed with error: " + to_string(ret));
    }
}

void AServer::loop()
{
    if (listen(listenSocket, SOMAXCONN) == SOCKET_ERROR) {
        closesocket(listenSocket);
        throw runtime_error("listenSocket failed with error: " +
to_string(WSAGetLastError()));
    }

    working = true;

    while (working)
    {
        const SOCKET buffSocket = accept(listenSocket, nullptr, nullptr);
        if (buffSocket == INVALID_SOCKET) continue;

        peers.insert(new Peer(this, buffSocket));

        // Cleaning up finished connections
        Peer* buff;
        finishedPeersMutex.lock();
        while (!finishedPeers.empty())
        {
            buff = finishedPeers.front();
            finishedPeers.pop();
```

```
            delete buff;
            peers.erase(buff);
        }
        finishedPeersMutex.unlock();
    }

    closesocket(listenSocket);
}
```

Код AClient.h

```
#pragma once
#include "APeer.h"
#include "WinSockWrapper.h"

class AClient : public APeer
{
protected:
	std::unique_ptr<ASockResult> loopStart(std::string& receiveBuffer, std::string&
sendBuffer) override;

public:
	AClient(std::string connectionAddress = "localhost", std::string connectionPort =
PORT_NUMBER);
};
```

Код AClient.cpp

```
// ReSharper disable CppCStyleCast
#include "pch.h"
#include "AClient.h"

#include <iostream>
#include <stdexcept>

using namespace std;


std::unique_ptr<APeer::ASockResult> AClient::loopStart(std::string& receiveBuffer,
std::string& sendBuffer)
{
    unique_ptr<ASockResult> buffRes;

    // Exchanging START_MESSAGEs
    if ((buffRes = contact(SEND, (char*) &START_MESSAGE_SIZE, 1)
        )->type != ASockResult::OK)
        return buffRes;
    if ((buffRes = contact(SEND, (char*) START_MESSAGE, START_MESSAGE_SIZE)
        )->type != ASockResult::OK)
        return buffRes;

    return buffRes;
}

AClient::AClient(std::string connectionAddress, std::string connectionPort): APeer()
{ setSocket(move(connectionAddress), move(connectionPort)); }
```

Код Сервера

```
#include <iostream>

#include "AServer.h"

using namespace std;

class Server: public AServer
{
	int cur_id = 0;

public:
	Server(std::string address = "", std::string port = PORT_NUMBER) :
		AServer(move(address), move(port)){}

	struct Context: APeer::AContext
	{
		int id;
		string prev_grid;

		Context(int id, string grid): AContext(), id(id), prev_grid(grid) {}

		int compare_grid(const string& grid)
		{
			if (grid.size() != prev_grid.size())
				return -1;
			int ret = 0;
			for (int i = 0; i < grid.size(); i++)
				if(grid[i] != prev_grid[i])
				{
					ret++;
					ret += prev_grid[i] != '0';
				}
			return ret;
		}

		virtual ~Context()
		{ cout << "Removed finished client!\n"; }
	};

	int checkWinLoose(const string& grid)
	{
		char vert[5], horis[5], d1 = grid[0], d2 = grid[4];
		for(int i = 0; i < 5; i++)
		{
			if (grid[i] == '1' || grid[i] == '2') vert[i] = grid[i];
			else vert[i] = 0;
			if (grid[5*i] == '1' || grid[5*i] == '2') horis[i] = grid[5 *i];
			else horis[i] = 0;
		}

		for(int y = 0; y < 5; y++)
			for(int x = 0; x < 5; x++)
			{
				if (x == y && d1 != grid[x + 5 * y]) d1 = 0;
				if (4 - x == y && d2 != grid[x + 5 * y]) d2 = 0;
				if (grid[x + 5 * y] != vert[x]) vert[x] = 0;
				if (grid[x + 5 * y] != horis[y]) horis[y] = 0;
			}

		int was = 0;
		if (d1 == '2' || d2 == '2') was = 2;
		if (d1 == '1' || d2 == '1') was = 1;
		for (int i = 0; i < 5 && was == 0; i++)
		{
```

```
                if (vert[i] == '2' || horis[i] == '2') was = 2;
                if (vert[i] == '1' || horis[i] == '1') was = 1;
            }
            return was;
        }

        APeer::HandlerResponse messageHandler(APeer::AContext** _context, std::string
message) override
        {
            if(*_context == nullptr)
            {
                cout << "New client!\n";

                if (message.size() != 25) {
                    return APeer::HandlerResponse(true, "2");
                }

                *_context = new Context(cur_id++, message);
            }
            else if(((Context*)*_context)->compare_grid(message) != 1)
                return APeer::HandlerResponse(true, "2");

            Context* context = (Context*)*_context;

            int status = checkWinLoose(message);
            if(status != 0)
            {
                if (status == 1) return APeer::HandlerResponse(true, "1");
                if (status == 2) return APeer::HandlerResponse(true, "2");
            }

            int ind = -1;
            for (int i = 0; i < 25 && ind == -1; i++)
                if (message[i] == '0') ind = i;

            if(ind == -1) return APeer::HandlerResponse(true, "0");
            message[ind] = '2';

            status = checkWinLoose(message);
            if (status != 0)
            {
                if (status == 1) return APeer::HandlerResponse(true, "1" + message);
                if (status == 2) return APeer::HandlerResponse(true, "2" + message);
            }

            context->prev_grid = message;
            return APeer::HandlerResponse(move(message));
        }
};

int main()
{
    Server server;
        server.loop();

    WinSockWrapper::close();

    return 0;
}
```

Код Клієнта

```
#include <iostream>
#include <string>

#include "AClient.h"

using namespace std;

class Client: public AClient
{
public:
    Client(std::string connectionAddress = "localhost", std::string connectionPort =
PORT_NUMBER):
                AClient(move(connectionAddress), move(connectionPort)) {}

    void print_grid(const string& grid)
    {
        cout << "\tGrid:\n\t___________\n";
        for (int y = 0; y < 5; y++) {
            cout << "\t|";
            for (int x = 0; x < 5; x++)
            {
                if (grid[y * 5 + x] == '1') cout << "@|";
                else if (grid[y * 5 + x] == '2') cout << "#|";
                else cout << (char)('a' + y * 5 + x) << "|";
            }
            cout << "\n\t-----------\n";
        }
    }

    bool make_move(string& grid)
    {
        cout << "Now enter a character, you want to place on a grid\n(that must not be @ or
#, or ! to exit): ";
        char buff;
        cin >> buff;
        while(buff != '!' && ('A' > buff || ('Y' < buff && buff < 'a') || buff > 'y'))
        {
            cout << "Reenter please: ";
            cin >> buff;
        }

        if(buff == '!') return false;

        int i;
        if (buff < 'Y') i = buff - 'A';
        else i = buff - 'a';
        grid[i] = '1';

        return true;
    }

    HandlerResponse messageHandler(AContext** context, std::string message) override
    {
        if(strcmp(message.c_str(), START_MESSAGE) == 0)
        {
            cout << "Successfully connected to the server!\nMake your first move:\n";
            string start_grid(25, '0');
            print_grid(start_grid);
            bool res = make_move(start_grid);
            if (res) return HandlerResponse(start_grid);
            return HandlerResponse(true, "");
        }
        if(message.size() == 25)
        {
```

```
            cout << "Received new move from a server. Now its time for yours:\n";
            print_grid(message);
            bool res = make_move(message);
            if (res) return HandlerResponse(message);
            return HandlerResponse(true, "");
        }
        if(message.size() == 1 || message.size() == 26)
        {
            if (message[0] == '2') cout << "!!! Server won\n";
            else if (message[0] == '1') cout << "!!! You won!\n";
            else if (message[0] == '0') cout << "!!! No one win(!\n";

            if(message.size() == 26)
            {
                cout << "It responded with such a grid:\n";
                print_grid(message.substr(1));
            }

            return HandlerResponse(true, "");
        }

        return HandlerResponse(true, "");
    }

    std::unique_ptr<ASockResult> loopEnd(std::unique_ptr<ASockResult> reason) override
    {
        if (reason->type == ASockResult::SHUTDOWN)
            cout << "Connection was shutdown\n";
        else if (reason->type == ASockResult::ERR)
            cout << "Connection closed because of an error: " <<
((ErrSockResult*)reason.get())->code;

        return AClient::loopEnd(move(reason));
    }
};

int main()
{
    Client cl;
    cl.loop();
    WinSockWrapper::close();
    cin.get(); cin.get();
}
```

Текст системного журналу Сервера

```
r1""
r15"CONNECT_CLIENT"
s1""
s15"CONNECT_CLIENT"
r1"↓ONNECT_CLIENT"
r25"10000000000000000000000"
New client!
s26"↓12000000000000000000000"
r1"↓00000000000000000000000"
r25"12000010000000000000000"
s26"↓12200010000000000000000"
r1"↓20000100000000000000000"
r25"12200010000010000000000"
s26"↓12220010000010000000000"
r1"↓22000100000100000000000"
r25"12220010000010000001000000"
s26"↓122220100000100000010000000"
r1"↓222001000001000001000000"
r25"122220100000100000100000l"
s2"☺1"
Removed finished client!
```

Текст системного журналу Клієнта

```
s1""
s15"CONNECT_CLIENT"
r1""
r15"CONNECT_CLIENT"
Successfully connected to the server!
Make your first move:
        Grid:

        ___________
        |a|b|c|d|e|
        -----------
        |f|g|h|i|j|
        -----------
        |k|l|m|n|o|
        -----------
        |p|q|r|s|t|
        -----------
        |u|v|w|x|y|
        -----------
Now enter a character, you want to place on a grid
(that must not be @ or #, or ! to exit): a
s26"↓10000000000000000000000"
r1"↓ONNECT_CLIENT"
r25"12000000000000000000000"
Received new move from a server. Now its time for yours:
        Grid:

        ___________
        |@|#|c|d|e|
        -----------
        |f|g|h|i|j|
        -----------
        |k|l|m|n|o|
        -----------
        |p|q|r|s|t|
        -----------
        |u|v|w|x|y|
        -----------
Now enter a character, you want to place on a grid
(that must not be @ or #, or ! to exit): g
s26"↓12000010000000000000000"
r1"↓20000000000000000000000"
r25"12200010000000000000000"
Received new move from a server. Now its time for yours:
        Grid:

        ___________
```

```
        |@|#|#|d|e|
        -----------
        |f|@|h|i|j|
        -----------
        |k|l|m|n|o|
        -----------
        |p|q|r|s|t|
        -----------
        |u|v|w|x|y|
        -----------
Now enter a character, you want to place on a grid
(that must not be @ or #, or ! to exit): m
s26"↓12200010000010000000000"
r1"↓22000100000000000000000"
r25"12220010000010000000000"
Received new move from a server. Now its time for yours:
        Grid:

        _____________
        |@|#|#|#|e|
        -----------
        |f|@|h|i|j|
        -----------
        |k|l|@|n|o|
        -----------
        |p|q|r|s|t|
        -----------
        |u|v|w|x|y|
        -----------
Now enter a character, you want to place on a grid
(that must not be @ or #, or ! to exit): s
s26"↓122200100000100000001000000"
r1"↓222001000001000000000000"
r25"122220100000100000001000000"
Received new move from a server. Now its time for yours:
        Grid:

        _____________
        |@|#|#|#|#|
        -----------
        |f|@|h|i|j|
        -----------
        |k|l|@|n|o|
        -----------
        |p|q|r|@|t|
        -----------
        |u|v|w|x|y|
        -----------
```

```
Now enter a character, you want to place on a grid
(that must not be @ or #, or ! to exit): y
s26"↓12222010000010000001000001"
r1"☺22220100000100000001000000"
r1"1"
!!! You won!
Connection was shutdown
```