g***Министерство образования Республики Беларусь***

***Учреждение образования***

***«Брестский государственный технический университет»***

***Кафедра ИИТ***

**Лабораторная работа №5**

**По дисциплине КМЗИ за IV семестр**

**Тема: «Контроль целостности»**

**Выполнил:**

Студент группы ИИ-15 (1)

2-го курса

Волк И. А.

**Проверил:**

Хацкевич М. В.

Брест 2019

Цель работы: изучить методы контроля целостности.

Ход работы

В данной лабораторной работе реализована проверка целостности на основе битов четности.

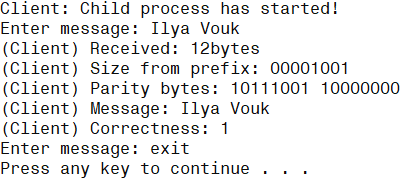
Описание метода решения:

1. Со стороны сервера запрашивается введение данных с клавиатуры.
2. К данным находится префикс, который содержит следующую информацию:
   1. Первый байт обозначает количество байтов в сообщении
   2. Далее идут байты, содержащие в себе биты чётности. Размеры последовательности данных байтов определяются длиной сообщения. Каждый байт содержит 8 бит четности, которые заполняются справа налево.

Данная конструкция позволяет корректно определить необходимое количество бит четности для изъятия.

1. Префикс соединяется с сообщением и отсылается клиенту
2. Клиент разделяет пришедшие данные на префикс и сообщение
3. Для данных в сообщении заново находится префикс, который затем сравнивается с пришедшим со стороны сервера

Демонстрация работы системы:



Листинги:

1. **Сервер**

#include "pch.h"

#include <iostream>

#include <SFML/Network.hpp>

#include <windows.h>

#include <string>

const TCHAR szClientApp[100] = TEXT("S:\\Labs\\bstu.labs.2.2\\КМЗИ\\lab.5\\parity-bits\\Debug\\client.exe");

const TCHAR szMutexName[100] = TEXT("std\_out\_mutex");

bool OpenClientApp();

void ErrorMessage(const wchar\_t\*);

void GetClient(sf::TcpSocket &client);

void SendDataToClient(sf::TcpSocket& client, const char\* szData);

std::string GetPrefix(std::string sData);

int main()

{

// creating mutex for standard output

SECURITY\_ATTRIBUTES sa;

sa.bInheritHandle = TRUE;

sa.lpSecurityDescriptor = NULL;

sa.nLength = sizeof(sa);

HANDLE hStdMutex = CreateMutex(&sa, NULL, szMutexName);

// getting accepted client

// client is in another process

sf::TcpSocket client;

GetClient(client);

// sending data to client in processing loop

std::string sMessage;

do

{

Sleep(100);

WaitForSingleObject(hStdMutex, INFINITE);

std::cout << "Enter message: ";

std::getline(std::cin, sMessage);

ReleaseMutex(hStdMutex);

std::string sPrefix = GetPrefix(sMessage);

std::string sDataToSend = sPrefix + sMessage;

SendDataToClient(client, sDataToSend.c\_str());

} while (sMessage != "exit");

system("pause");

}

std::string GetPrefix(std::string sData)

{

// max length of one package - 255

if (sData.length() > 255) return "";

// the first character is number of bytes in message

// it will define number of parity bits

char cNumberOfBytes = sData.length();

std::string sParityBits;

// filling of parity bits

// format: \*\*\*\*\*000 where \* - parity bit, 0 - redundant byte

char cCurrent = 0; int nCurrentPos = 0;

for (char ch : sData)

{

char cParity = 0;

for (int i = 0; i < 8; i++)

{

cParity += ch & 0b1;

cParity %= 2;

ch >>= 1;

}

cParity <<= 7 - nCurrentPos;

cCurrent |= cParity;

nCurrentPos++;

if (nCurrentPos == 8)

{

sParityBits += cCurrent;

cCurrent = 0;

nCurrentPos = 0;

}

}

if (nCurrentPos != 0)

sParityBits.push\_back(cCurrent);

sParityBits.insert(sParityBits.begin(), cNumberOfBytes);

return sParityBits;

}

void GetClient(sf::TcpSocket& client)

{

sf::TcpListener listener;

if (listener.listen(53000) != sf::Socket::Done)

ErrorMessage(TEXT("While listening"));

if (!OpenClientApp())

ErrorMessage(TEXT("Cannot create client application"));

if (listener.accept(client) != sf::Socket::Done)

ErrorMessage(TEXT("While client accepting"));

}

void SendDataToClient(sf::TcpSocket& client, const char\* szData)

{

if (client.send(szData, strlen(szData)) != sf::Socket::Done)

ErrorMessage(TEXT("Error occurred while data sending"));

}

bool OpenClientApp()

{

PROCESS\_INFORMATION piProcInfo;

ZeroMemory(&piProcInfo, sizeof(PROCESS\_INFORMATION));

STARTUPINFO siStartInfo;

ZeroMemory(&siStartInfo, sizeof(STARTUPINFO));

siStartInfo.cb = sizeof(STARTUPINFO);

siStartInfo.hStdError = GetStdHandle(STD\_OUTPUT\_HANDLE);

siStartInfo.hStdOutput = GetStdHandle(STD\_ERROR\_HANDLE);

TCHAR szCommandLine[100] = TEXT("Hello world my name is Ilya!");

bool bSuccess = CreateProcess(

szClientApp,

szCommandLine,

NULL,

NULL,

TRUE,

0,

NULL,

NULL,

&siStartInfo,

&piProcInfo

);

return bSuccess;

}

void ErrorMessage(const wchar\_t\* szMsg)

{

MessageBox(NULL, szMsg, L"Error", MB\_OK);

PostQuitMessage(0);

}

1. **Клиент**

#include "pch.h"

#include <iostream>

#include <SFML/Network.hpp>

#include <Windows.h>

#include <utility>

const TCHAR szMutexName[100] = TEXT("std\_out\_mutex");

void ClientPrint(const char\* msg);

void ErrorMessage(const wchar\_t\*);

bool CheckCorrectness(std::string sMessage);

// return <prefix, message>

std::pair <std::string, std::string> split(std::string sMessage);

std::vector<std::string> ConvertBytesToString(std::string sBytes);

int main(int argv, char \*args[])

{

ClientPrint("Child process has started!");

// connection to server

sf::TcpSocket socket;

sf::IpAddress ipAddress("127.0.0.1");

sf::Socket::Status status = socket.connect(ipAddress, 53000);

if (status != sf::Socket::Done)

{

std::cout << "Error occurred!";

return -1;

}

// mutex for standard output

HANDLE hStdOutMutex = OpenMutex(SYNCHRONIZE, FALSE, szMutexName);

while (true)

{

// receiving data

char data[255] = "";

std::size\_t received;

if (socket.receive(data, 100, received) != sf::Socket::Done)

{

ClientPrint("Error while data receiving");

return -1;

}

std::pair<std::string, std::string> sSplited = split(data);

// converting prefix to readable form

std::vector<std::string> sBytes = ConvertBytesToString(sSplited.first);

// printing results

WaitForSingleObject(hStdOutMutex, INFINITE);

std::cout << "(Client) Received: " << received << "bytes" << std::endl;

std::cout << "(Client) Size from prefix: " << sBytes[0] << std::endl;

std::cout << "(Client) Parity bytes: ";

for (auto i = sBytes.begin() + 1; i != sBytes.end(); i++)

std::cout << \*i << " ";

std::endl(std::cout);

std::cout << "(Client) Message: " << sSplited.second << std::endl;

std::cout << "(Client) Correctness: " << CheckCorrectness(data) << std::endl;

ReleaseMutex(hStdOutMutex);

}

ClientPrint("Ending process");

}

void ClientPrint(const char\* msg)

{

std::cout << "Client: " << msg << std::endl;

}

std::vector<std::string> ConvertBytesToString(std::string sBytes)

{

std::vector<std::string> vecBytes;

for (char byte : sBytes)

{

std::string sByte;

for (int i = 0; i < 8; i++)

{

sByte.insert(sByte.begin(), (byte & 0b1) ? '1' : '0');

byte >>= 1;

}

vecBytes.push\_back(sByte);

}

return vecBytes;

}

// to split into <prefix, message>

std::pair <std::string, std::string> split(std::string sMessage)

{

char cNumberOfBytes = sMessage[0];

int nNumberOfBytes = cNumberOfBytes;

int nNumberOfParityBytes = (cNumberOfBytes % 8) ? (cNumberOfBytes / 8 + 1) : (cNumberOfBytes / 8);

std::string sParityBytes = { sMessage.begin() + 1, sMessage.begin() + 1 + nNumberOfParityBytes };

sParityBytes.insert(sParityBytes.begin(), cNumberOfBytes);

sMessage = { sMessage.begin() + 1 + nNumberOfParityBytes, sMessage.end() };

return { sParityBytes, sMessage };

}

std::string GetPrefix(std::string sData)

{

// max length of one package - 255

if (sData.length() > 255) return "";

// the first character is number of bytes in message

// it will define number of parity bits

char cNumberOfBytes = sData.length();

std::string sParityBits;

// filling of parity bits

// format: \*\*\*\*\*000 where \* - parity bit, 0 - redundant byte

char cCurrent = 0; int nCurrentPos = 0;

for (char ch : sData)

{

char cParity = 0;

for (int i = 0; i < 8; i++)

{

cParity += ch & 0b1;

cParity %= 2;

ch >>= 1;

}

cParity <<= 7 - nCurrentPos;

cCurrent |= cParity;

nCurrentPos++;

if (nCurrentPos == 8)

{

sParityBits += cCurrent;

cCurrent = 0;

nCurrentPos = 0;

}

}

if (nCurrentPos != 0)

sParityBits.push\_back(cCurrent);

sParityBits.insert(sParityBits.begin(), cNumberOfBytes);

return sParityBits;

}

bool CheckCorrectness(std::string sMessage)

{

// <prefix, message>

std::pair <std::string, std::string> sSplited = split(sMessage);

std::string sActualPrefix = GetPrefix(sSplited.second);

if (sSplited.first == sActualPrefix)

return true;

return false;

}

void ErrorMessage(const wchar\_t\* szMsg)

{

MessageBox(NULL, szMsg, L"Error", MB\_OK);

PostQuitMessage(0);

}

Вывод: по ходу лабораторной работы научился производить проверку числа на простоту, находить простые делители.