МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ

«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ»

НАВЧАЛЬНО-НАУКОВИЙ КОМПЛЕКС

«ІНСТИТУТ ПРИКЛАДНОГО СИСТЕМНОГО АНАЛІЗУ»

Лабораторна робота № 1-4

з курсу «Кросс-платформене програмування»

Виконав:

студент IV курсу

групи ДА-32

Колінько Анжела

Київ – 2017

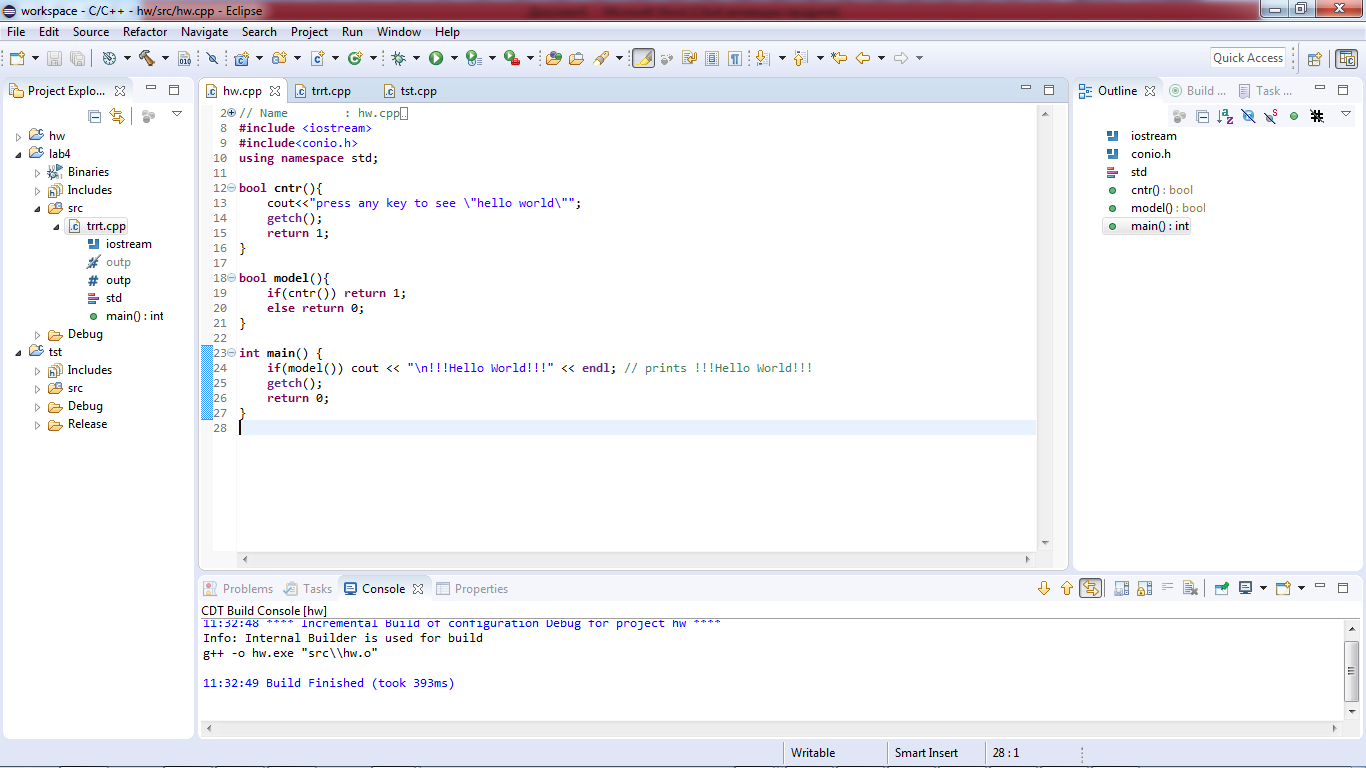
Варіант 11

**Лаб. 1. Встановлення середовища розробки кросс-платформеного програмування**

Мета роботи: ознайомитися з середовищем Integrated Development Environment (IDE)

Задача: встановити Integrated Development Environment, ознайомитися з конфігурацією середовища розробки. На прикладі системи Eclipse вивчити систему

Оформлення результатів роботи.



Висновки: eclipse – середовище написане на java, тому є крос-платформенним. Також серед переваг eclipse є можливість гнучкого налаштування та наявність віртуальної машини.

**Лаб. 2. Налаштування проекту в середовищі IDE**

Мета роботи: створити та налаштувати проект під IDE Eclipse для крос-платформеного програмного забезпечення

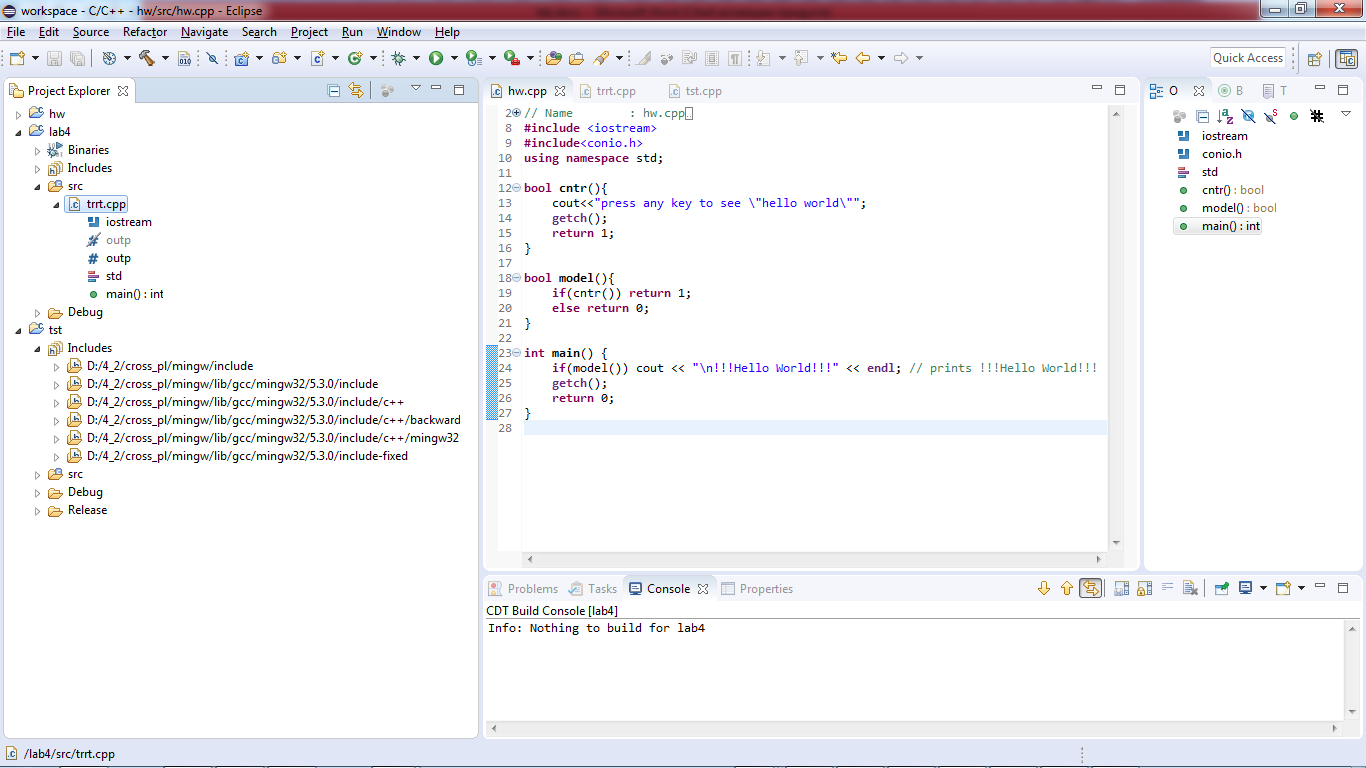
Задача: ознайомитися з параметрами та налаштуваннями проекту в крос-платформених системах.

Опис основних налаштувань проекту:

1. Встановлення компілятору MinGW.

2. Внесення шляху до папки bin компілятору в змінну середовища PATH.

3. Додавання шляхів до includes



Висновки: використання крос-платформених бібліотек дозволяє компілювати і запускати один source code для різних операційних систем.

**Лаб. 3. Компіляція та відладка крос-платформеного тестового прикладу**

Мета роботи: розробити тестовий приклад програми Hello World, запустити його у 2-3 різноманітних операційних системах.

Задача: вивчити основні функції бібліотеки крос-платформеного програмування, відлагодження на різноманітних операційних системах. Вивчити застосування патернів, наприклад MVC для розділення інтерфейсу користувача від бізнес-логіки інформаційної системи.

Лістинг:

#include <iostream>

//#include<conio.h>

#include<stdio.h>

using namespace std;

unsigned char ch;

bool cntr(){

cout<<"press enter to see \"hello world\"";

//getch();

getchar();

return 1;

}

bool model(){

if(cntr()) return 1;

else return 0;

}

bool view(){

if(model()){

cout << "\n!!!Hello World!!!" << endl; // prints !!!Hello World!!!

return 1;

}

else return 0;

}

int main() {

if(view()){

//getch();

cout<<"\npress enter to exit";

getchar();

return 0;

}

}

conio.h – заготовочний файл, що використовувався здебільшого компіляторами MS-DOS. Проте цей заготовочний файл не є ні частиною стандартної бібліотеки C, ні частиною ISO C, ні визначений в POSIX. Тому виклик getch() був замінений викликом getchar(); (із stdio.h).

Висновки: для розділення інтерфейсу користувача від бізнес-логіки інформаційної системи був використаний паттерн MVC. Тестовий приклад має запускатися на ОС Windows та Unix. Оскільки при написанні коду використовувалися крос-платформенні бібліотеки відмінності роботи на різних ОС повинні бути мінімальними.

**Лаб. 4. Розробка коду крос-платформеного програмного забезпечення**

Мета роботи: розробити крос-платформений додаток та відлагодити його роботу в 2-3 різноманітних операційних системах.

Задача: опанувати методами крос-платформеного програмування на прикладі розробки тестового завдання. За досвідом з 3-ї лабораторної роботи, застосувати необхідний паттерн проектування (наприклад MVC) та відділити бізнес-логіку додатку від інтерфейсу користувача.

Лістинг:

#include <iostream>

#include <ctime>

#include <chrono>

#include<string>

#ifdef \_WIN32

#pragma comment(lib, "ws2\_32.lib")

#include <winsock2.h>

#elif \_\_linux\_\_

#include <sys/types.h>

#include <sys/socket.h>

#include <netinet/in.h>

#include <arpa/inet.h>

#include <netdb.h>

#include <cstring>

#endif

using namespace std;

typedef unsigned short int usi;

struct ICMPheader{

unsigned char byType;

unsigned char byCode;

unsigned short nChecksum;

unsigned short nId;

unsigned short nSequence;

};

struct IPheader{

unsigned char byVerLen;

unsigned char byTos;

unsigned short nTotalLength;

unsigned short nId;

unsigned short nOffset;

unsigned char byTtl;

unsigned char byProtocol;

unsigned short nChecksum;

unsigned int nSrcAddr;

unsigned int nDestAddr;

};

class target{

public:

int nSequence = 0;

int nTimeOut = 5000; //Request time out for echo request (in milliseconds)

int nMessageSize = 32; //The message size that the ICMP echo request should carry with it

int nCount = 4; //Number of times the request has to be send

string szRemoteIP, szRemoteHost;

char\* SendBuffer=NULL;

target(){}

target(int argc, char\*\* argv){

if(argc>=2) szRemoteHost=string(argv [1]); else szRemoteHost=string("www.google.com");

//get param;

switch(argc){

case 5:

nTimeOut = atoi (argv [4]);

cout<<"\nargc="<<argc<<" nTimeOut="<<nTimeOut;

case 4:

nMessageSize = atoi (argv [3]);

cout<<"\nargc="<<argc<<" nMessageSize="<<nMessageSize;

case 3:

nCount = atoi (argv [2]);

cout<<"\nargc="<<argc<<" nCount="<<nCount;

}

}

int virtual png(){}

bool virtual Initialize(){}

bool virtual UnInitialize(){}

bool ResolveIP (string& szRemoteHost, string& szIPAddress){

hostent \*pHostent = gethostbyname (szRemoteHost.c\_str());

if (pHostent == NULL){

cerr << endl << "An error occured in gethostbyname operation: WSAGetLastError () = ";

#ifdef \_WIN32

cout<< WSAGetLastError () << endl;

#endif

return false;

}

szIPAddress=string(pHostent->h\_addr\_list[0]);

return true;

}

unsigned short CalcChecksum (unsigned char \*pBuffer, int nLen){

//Checksum for ICMP is calculated in the same way as for IP header

//This code was taken from: http://www.netfor2.com/ipsum.htm

unsigned short nWord;

unsigned int nSum = 0;

int i;

//Make 16 bit words out of every two adjacent 8 bit words in the packet and add them up

for (i = 0; i < nLen; i = i + 2) {

nWord =((pBuffer [i] << 8)& 0xFF00) + (pBuffer [i + 1] & 0xFF);

nSum = nSum + (unsigned int)nWord;

}

//Take only 16 bits out of the 32 bit sum and add up the carries

while (nSum >> 16) {

nSum = (nSum & 0xFFFF) + (nSum >> 16);

}

//One's complement the result

nSum = ~nSum;

return ((unsigned short) nSum);

}

bool ValidateChecksum (unsigned char \*pBuffer, int nLen){

unsigned short nWord;

unsigned int nSum = 0;

int i;

//Make 16 bit words out of every two adjacent 8 bit words in the packet and add them up

for (i = 0; i < nLen; i = i + 2) {

nWord =((pBuffer [i] << 8)& 0xFF00) + (pBuffer [i + 1] & 0xFF);

nSum = nSum + (unsigned int)nWord;

}

//Take only 16 bits out of the 32 bit sum and add up the carries

while (nSum >> 16) {

nSum = (nSum & 0xFFFF) + (nSum >> 16);

}

//To validate the checksum on the received message we don't complement the sum of one's complement

//One's complement the result nSum = ~nSum;

//The sum of one's complement should be 0xFFFF

return ((unsigned short)nSum == 0xFFFF);

}

};

class intf{

public:

target tgp;

bool ret=0;

intf(int argc, char\*\* argv){

if (argc > 5){

PrintUsage (); ret=1;

}

else tgp=target(argc,argv);

}

void PrintUsage (){

cout << "Usage: ping r n b t" << endl << endl;

cout << " r - Remote host" << endl;

cout << " n - Number of echo requests to send" << endl;

cout << " b - Bytes to send" << endl;

cout << " t - Timeout after these many milliseconds" << endl << endl;

cout << "\rping microsoft.com 4 32 4000" << endl << endl;

}

};

class wadaptee{

public:

void wr(){

cout<<"\n\nit is \_WIN32\n";

}

};

class ladaptee{

public:

void wr(){

cout<<"\n\nit is \_\_linux\_\_\n";

}

};

#ifdef \_WIN32

class cw: public target{

public:

wadaptee adt;

cw(target tg){

nSequence = tg.nSequence;

nTimeOut = tg.nTimeOut; //Request time out for echo request (in milliseconds)

nMessageSize = tg.nMessageSize; //The message size that the ICMP echo request should carry with it

nCount = tg.nCount; //Number of times the request has to be send

szRemoteIP=tg.szRemoteIP;

szRemoteHost=tg.szRemoteHost;

SendBuffer=tg.SendBuffer;

}

bool Initialize (){

WSADATA wsaData;//Initialize WinSock

if (WSAStartup (MAKEWORD (2, 2), &wsaData) == SOCKET\_ERROR) {

cerr << endl << "An error occured in WSAStartup operation: " << "WSAGetLastError () = " << WSAGetLastError () << endl;

return false;

}

srand(time(NULL)); //Seed the random number generator

return true;

}

bool UnInitialize (){

if (WSACleanup () == SOCKET\_ERROR) {//Cleanup

cerr << endl << "An error occured in WSACleanup operation: WSAGetLastError () = " << WSAGetLastError () << endl;

return false;

}

return true;

}

int png(){

if (Initialize () == false){

return -1;

}

if (ResolveIP (szRemoteHost, szRemoteIP) == false){

cerr << endl << "Unable to resolve hostname" << endl;

return -1;

}

in\_addr in; in.s\_addr = \*(u\_long \*) szRemoteIP.c\_str();

cout << "\nPinging " << szRemoteHost << " [";

//#ifdef \_WIN32

cout<<(usi)in.S\_un.S\_un\_b.s\_b1<<'.'<<(usi)in.S\_un.S\_un\_b.s\_b2<<'.'<<(usi)in.S\_un.S\_un\_b.s\_b3<<'.'<<(usi)in.S\_un.S\_un\_b.s\_b4;

/\*#elif \_\_linux\_\_

cout<<inet\_ntoa(in);

#endif\*/

cout<< "] with " << nMessageSize << " bytes of data\n";

//create socet;

ICMPheader sendHdr; sendHdr.nId = htons (rand ()); //Set the transaction Id

//#ifdef \_WIN32

SOCKET sock;

/\*#elif \_\_linux\_\_

int sock;

#endif\*/

sock= socket (AF\_INET, SOCK\_RAW, IPPROTO\_ICMP); //Create a raw socket which will use ICMP

sockaddr\_in dest; memset(&dest, 0, sizeof(dest));//Dest address to send the ICMP request

memcpy(&(dest.sin\_addr), szRemoteIP.c\_str(),sizeof(szRemoteIP.c\_str()));//dest.sin\_addr.S\_un.S\_addr = inet\_addr (szRemoteIP.c\_str());

dest.sin\_family = AF\_INET;

int nResult = 0; chrono::time\_point<std::chrono::system\_clock> tsend, trecv;

//fd\_set fdRead; SYSTEMTIME timeSend, timeRecv;//TODO;

int nTotalRoundTripTime = 0, nMaxRoundTripTime = 0, nMinRoundTripTime = -1;

int nPacketsSent = 0, nPacketsReceived = 0;

//timeval timeInterval = {0, 0}; timeInterval.tv\_usec = nTimeOut \* 1000;

while (nPacketsSent < nCount){

//create ICMP message;

SendBuffer = new char [sizeof (ICMPheader) + nMessageSize];

sendHdr.byType = 8; //Eight for ICMP echo message

sendHdr.byCode = 0; //Zero for ICMP echo and reply messages

sendHdr.nChecksum = 0; //Checksum is calculated later on

sendHdr.nSequence = htons (nSequence++);

memcpy(SendBuffer,&sendHdr,sizeof(ICMPheader));

memset(SendBuffer+sizeof (ICMPheader),0,nMessageSize);

//Calculate checksum over ICMP header and message data

sendHdr.nChecksum = htons (CalcChecksum ((unsigned char\*)SendBuffer, sizeof (ICMPheader) + nMessageSize));

memcpy(SendBuffer,&sendHdr,sizeof(ICMPheader));

//send;

tsend=chrono::system\_clock::now();

nResult = sendto (sock, SendBuffer, sizeof (ICMPheader) + nMessageSize, 0, (struct sockaddr\*)&dest, sizeof (sockaddr\_in));

nPacketsSent++;

auto sckerr=SOCKET\_ERROR;//gserrv();

if (nResult == sckerr){

cerr << endl << "An error occured in sendto operation: " << "WSAGetLastError () = ";

//#ifdef \_WIN32

cout<< WSAGetLastError () << endl; UnInitialize ();

//#endif

delete []SendBuffer;

return -1;

}

//recieve;

chrono::duration<double,milli> ds=chrono::system\_clock::now()-tsend;

if (nResult>0 && ds.count()<=((double)nTimeOut)){

unsigned char \*RecvBuffer = new unsigned char [1500];//Allocate a large buffer to store the response

if ((nResult = recvfrom (sock, (char\*)RecvBuffer, 1500, 0, 0, 0))== sckerr){

cerr<<"\nAn error occured in recvfrom operation: " << "WSAGetLastError () = ";

//#ifdef \_WIN32

cout<< WSAGetLastError () << endl; UnInitialize ();

//#endif

delete []SendBuffer;

delete []RecvBuffer;

return -1;

}

trecv=chrono::system\_clock::now();//::GetSystemTime (&timeRecv);//TODO;//Get the time at which response is received

ICMPheader recvHdr;//We got a response so we construct the ICMP header and message out of it

IPheader ipHdr;

unsigned char \*ICMPbuffer= RecvBuffer + sizeof(IPheader);//The response includes the IP header as well,

//so we move 20 bytes ahead to read the ICMP header

int nICMPmsgLen = nResult - sizeof(IPheader); //ICMP message length is calculated by subtracting the IP header size

//from the total bytes received

memcpy(&recvHdr,ICMPbuffer,sizeof(ICMPheader));//Construct the ICMP header

memcpy(&ipHdr,RecvBuffer,sizeof(IPheader));//Construct the IP header from the response

recvHdr.nChecksum = ntohs (recvHdr.nChecksum);

//check

if (recvHdr.byType == 0 &&//Check if the response is an echo reply, transaction ID and sequence number are same

recvHdr.nId == sendHdr.nId && recvHdr.nSequence == sendHdr.nSequence &&

ValidateChecksum (ICMPbuffer, nICMPmsgLen) && //as for the request, and that the checksum is correct

memcmp(SendBuffer+sizeof(ICMPheader),RecvBuffer+sizeof(ICMPheader)+sizeof(IPheader),

nResult-sizeof(ICMPheader)-sizeof(IPheader))==0){//All's OK

/\*int nSec = timeRecv.wSecond - timeSend.wSecond;//TODO;

if (nSec < 0){

nSec = nSec + 60;

}\*/

//int nMilliSec = abs (timeRecv.wMilliseconds - timeSend.wMilliseconds);

chrono::duration<double,milli> rtt=trecv-tsend;

//int nRoundTripTime=abs (nSec \* 1000 - nMilliSec);

cout << "Reply " << nPacketsReceived << ": bytes = " << nResult - sizeof (ICMPheader) - sizeof (IPheader)

<< ", time = " << rtt.count() << "ms, TTL = " << (int)ipHdr.byTtl << endl;

nTotalRoundTripTime = nTotalRoundTripTime + rtt.count();

if (nMinRoundTripTime == -1){

nMinRoundTripTime = rtt.count();;

nMaxRoundTripTime = rtt.count();;

}

else if (rtt.count()< nMinRoundTripTime){

nMinRoundTripTime = rtt.count();;

}

else if (rtt.count()> nMaxRoundTripTime){

nMaxRoundTripTime = rtt.count();;

}

++nPacketsReceived;

}

else{

cout << "The echo reply is not correct!" << endl;

}

delete []RecvBuffer;

}

else{

cout << "Request timed out." << endl;

}

delete []SendBuffer;

}

adt.wr();

}

};

#endif

class cl: public target{

public:

ladaptee adt;

cl(target tg){

nSequence = tg.nSequence;

nTimeOut = tg.nTimeOut; //Request time out for echo request (in milliseconds)

nMessageSize = tg.nMessageSize; //The message size that the ICMP echo request should carry with it

nCount = tg.nCount; //Number of times the request has to be send

szRemoteIP=tg.szRemoteIP;

szRemoteHost=tg.szRemoteHost;

SendBuffer=tg.SendBuffer;

}

int png(){

if (ResolveIP (szRemoteHost, szRemoteIP) == false){

cerr << endl << "Unable to resolve hostname" << endl;

return -1;

}

in\_addr in; in.s\_addr = \*(u\_long \*) szRemoteIP.c\_str();

cout << "\nPinging " << szRemoteHost << " [";

/\*#ifdef \_WIN32

cout<<(usi)in.S\_un.S\_un\_b.s\_b1<<'.'<<(usi)in.S\_un.S\_un\_b.s\_b2<<'.'<<(usi)in.S\_un.S\_un\_b.s\_b3<<'.'<<(usi)in.S\_un.S\_un\_b.s\_b4;

#elif \_\_linux\_\_\*/

cout<<inet\_ntoa(in);

//#endif

cout<< "] with " << nMessageSize << " bytes of data\n";

//create socet;

ICMPheader sendHdr; sendHdr.nId = htons (rand ()); //Set the transaction Id

/\*#ifdef \_WIN32

SOCKET sock;

#elif \_\_linux\_\_\*/

int sock;

//#endif

sock= socket (AF\_INET, SOCK\_RAW, IPPROTO\_ICMP); //Create a raw socket which will use ICMP

sockaddr\_in dest; memset(&dest, 0, sizeof(dest));//Dest address to send the ICMP request

memcpy(&(dest.sin\_addr), szRemoteIP.c\_str(),sizeof(szRemoteIP.c\_str()));//dest.sin\_addr.S\_un.S\_addr = inet\_addr (szRemoteIP.c\_str());

dest.sin\_family = AF\_INET;

int nResult = 0; chrono::time\_point<std::chrono::system\_clock> tsend, trecv;

//fd\_set fdRead; SYSTEMTIME timeSend, timeRecv;//TODO;

int nTotalRoundTripTime = 0, nMaxRoundTripTime = 0, nMinRoundTripTime = -1;

int nPacketsSent = 0, nPacketsReceived = 0;

//timeval timeInterval = {0, 0}; timeInterval.tv\_usec = nTimeOut \* 1000;

while (nPacketsSent < nCount){

//create ICMP message;

SendBuffer = new char [sizeof (ICMPheader) + nMessageSize];

sendHdr.byType = 8; //Eight for ICMP echo message

sendHdr.byCode = 0; //Zero for ICMP echo and reply messages

sendHdr.nChecksum = 0; //Checksum is calculated later on

sendHdr.nSequence = htons (nSequence++);

memcpy(SendBuffer,&sendHdr,sizeof(ICMPheader));

memset(SendBuffer+sizeof (ICMPheader),0,nMessageSize);

//Calculate checksum over ICMP header and message data

sendHdr.nChecksum = htons (CalcChecksum ((unsigned char\*)SendBuffer, sizeof (ICMPheader) + nMessageSize));

memcpy(SendBuffer,&sendHdr,sizeof(ICMPheader));

//send;

tsend=chrono::system\_clock::now();

nResult = sendto (sock, SendBuffer, sizeof (ICMPheader) + nMessageSize, 0, (struct sockaddr\*)&dest, sizeof (sockaddr\_in));

nPacketsSent++;

auto sckerr=-1;//gserrv();

if (nResult == sckerr){

cerr << endl << "An error occured in sendto operation: " << "WSAGetLastError () = ";

/\*#ifdef \_WIN32

cout<< WSAGetLastError () << endl; UnInitialize ();

#endif\*/

delete []SendBuffer;

return -1;

}

//recieve;

chrono::duration<double,milli> ds=chrono::system\_clock::now()-tsend;

if (nResult>0 && ds.count()<=((double)nTimeOut)){

unsigned char \*RecvBuffer = new unsigned char [1500];//Allocate a large buffer to store the response

if ((nResult = recvfrom (sock, (char\*)RecvBuffer, 1500, 0, 0, 0))== sckerr){

cerr<<"\nAn error occured in recvfrom operation: " << "WSAGetLastError () = ";

/\*#ifdef \_WIN32

cout<< WSAGetLastError () << endl; UnInitialize ();

#endif\*/

delete []SendBuffer;

delete []RecvBuffer;

return -1;

}

trecv=chrono::system\_clock::now();//::GetSystemTime (&timeRecv);//TODO;//Get the time at which response is received

ICMPheader recvHdr;//We got a response so we construct the ICMP header and message out of it

IPheader ipHdr;

unsigned char \*ICMPbuffer= RecvBuffer + sizeof(IPheader);//The response includes the IP header as well,

//so we move 20 bytes ahead to read the ICMP header

int nICMPmsgLen = nResult - sizeof(IPheader); //ICMP message length is calculated by subtracting the IP header size

//from the total bytes received

memcpy(&recvHdr,ICMPbuffer,sizeof(ICMPheader));//Construct the ICMP header

memcpy(&ipHdr,RecvBuffer,sizeof(IPheader));//Construct the IP header from the response

recvHdr.nChecksum = ntohs (recvHdr.nChecksum);

//check

if (recvHdr.byType == 0 &&//Check if the response is an echo reply, transaction ID and sequence number are same

recvHdr.nId == sendHdr.nId && recvHdr.nSequence == sendHdr.nSequence &&

ValidateChecksum (ICMPbuffer, nICMPmsgLen) && //as for the request, and that the checksum is correct

memcmp(SendBuffer+sizeof(ICMPheader),RecvBuffer+sizeof(ICMPheader)+sizeof(IPheader),

nResult-sizeof(ICMPheader)-sizeof(IPheader))==0){//All's OK

/\*int nSec = timeRecv.wSecond - timeSend.wSecond;//TODO;

if (nSec < 0){

nSec = nSec + 60;

}\*/

//int nMilliSec = abs (timeRecv.wMilliseconds - timeSend.wMilliseconds);

chrono::duration<double,milli> rtt=trecv-tsend;

//int nRoundTripTime=abs (nSec \* 1000 - nMilliSec);

cout << "Reply " << nPacketsReceived << ": bytes = " << nResult - sizeof (ICMPheader) - sizeof (IPheader)

<< ", time = " << rtt.count() << "ms, TTL = " << (int)ipHdr.byTtl << endl;

nTotalRoundTripTime = nTotalRoundTripTime + rtt.count();

if (nMinRoundTripTime == -1){

nMinRoundTripTime = rtt.count();;

nMaxRoundTripTime = rtt.count();;

}

else if (rtt.count()< nMinRoundTripTime){

nMinRoundTripTime = rtt.count();;

}

else if (rtt.count()> nMaxRoundTripTime){

nMaxRoundTripTime = rtt.count();;

}

++nPacketsReceived;

}

else{

cout << "The echo reply is not correct!" << endl;

}

delete []RecvBuffer;

}

else{

cout << "Request timed out." << endl;

}

delete []SendBuffer;

}

adt.wr();

}

};

auto getpng(intf& intrfc){

#ifdef \_WIN32

cw ifptr=intrfc.tgp;

#elif \_\_linux\_\_

cl ifptr=intrfc.tgp;

#endif

return ifptr;

}

int main(int argc, char\*\* argv){

intf intrfc(argc,argv);

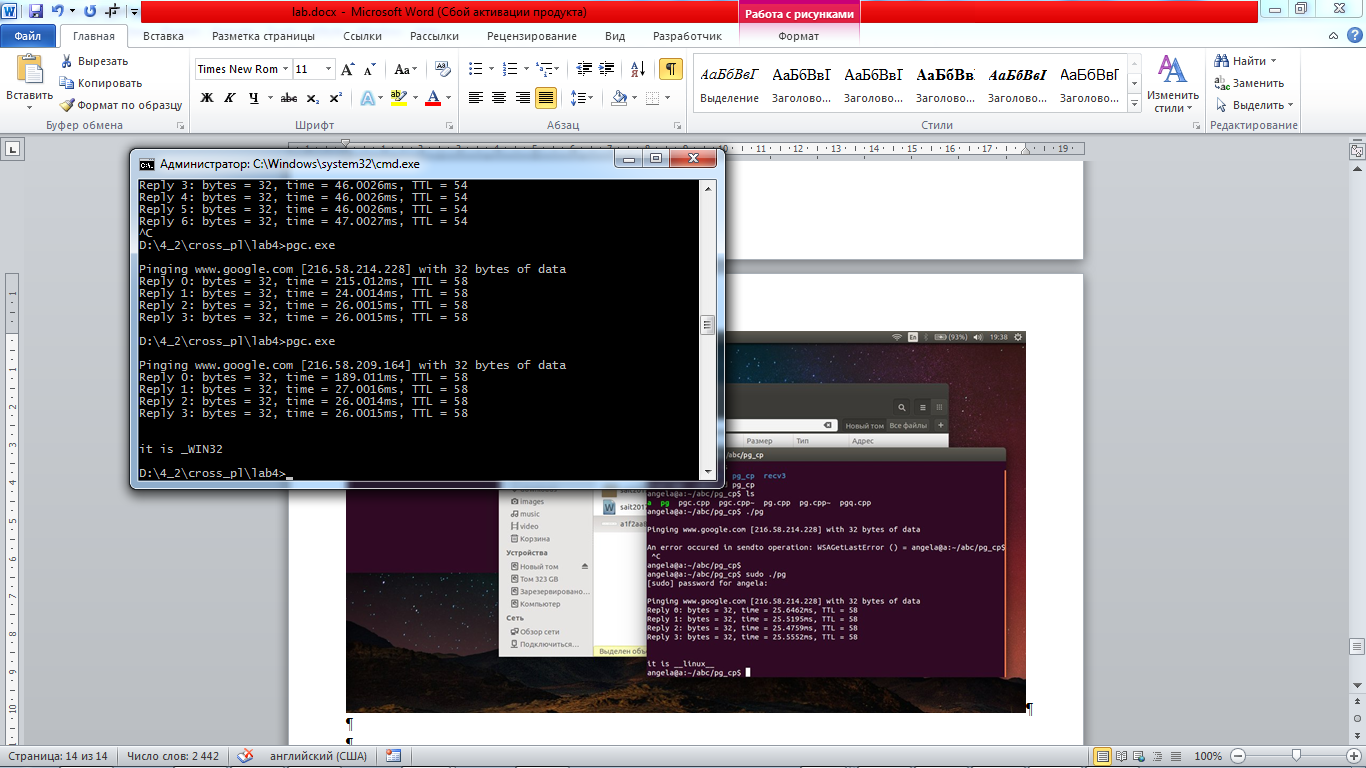
if(intrfc.ret==1) return 0;

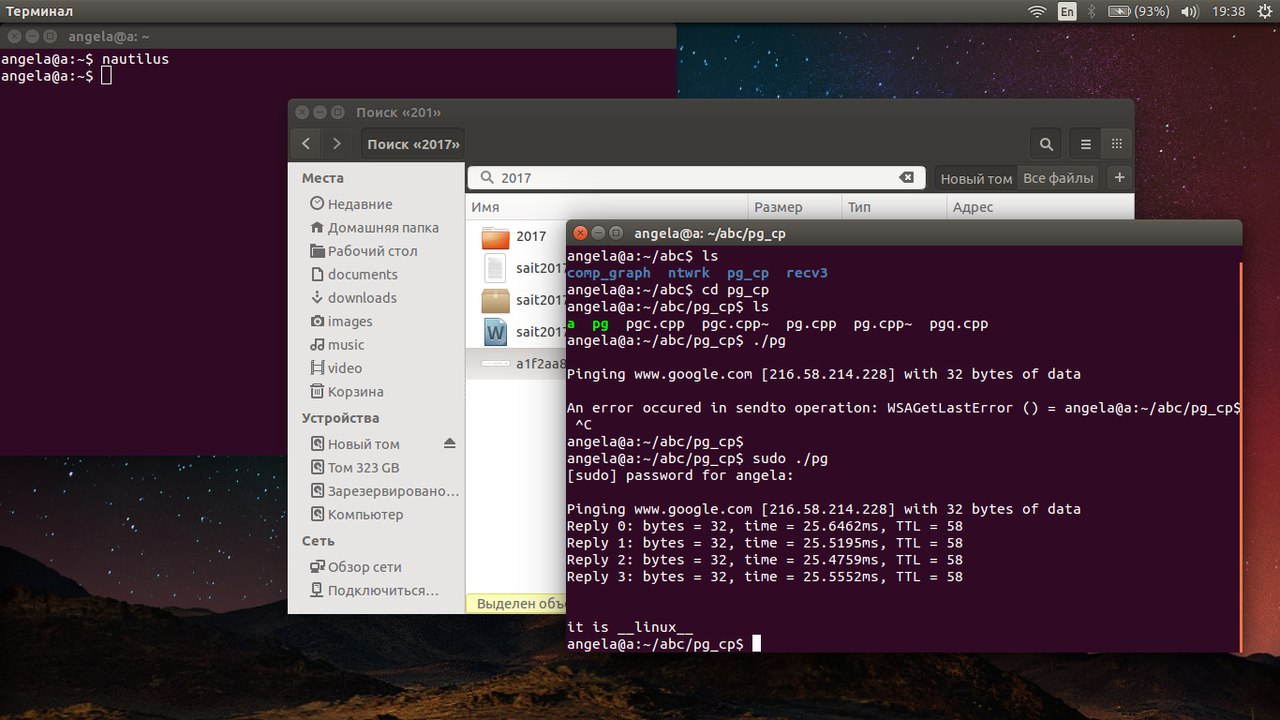
auto finpng=getpng(intrfc);

finpng.png();

}

Результат роботи:





Висновки: було розроблено крос-платформенну реалізацію команди ping. Використовувався паттерн Adapter. За необхідності додати модуль для інших операційних систем,необхідно додати новий subclass, котрий наслідує клас target, відповідний клас adaptee та доповнити необхідним кодом функцію getpng().