#include <iostream>

#include <Windows.h>

#include <stdio.h>

#include <stdlib.h>

#include <string>

#define BUFSIZE 255

using namespace std;

HANDLE COMport, handle, reader, writer;

//unsigned char\* bufrd = new unsigned char[BUFSIZE];

unsigned char bufrd[BUFSIZE];

unsigned char bufwr[];

int counter;

DWORD bytesWritten, bytesRidden;

OVERLAPPED overlapped; //будем использовать для операций чтения (см. поток ReadThread)

OVERLAPPED overlappedwr; //будем использовать для операций записи (см. поток WriteThread)

void COMOpen(); //открыть порт

void COMClose(); //закрыть порт

DWORD WINAPI ReadThread(LPVOID);

void ReadPrinting(void);

DWORD WINAPI WriteThread(LPVOID);

static DWORD WINAPI WriteThread(LPVOID)

{

DWORD temp, signal;

overlappedwr.hEvent = CreateEvent(NULL, true, true, NULL);

while (1)

{

//cout << "Введите кол-во оборотов: ";

string data;

cin >> data;

char\* charArray = new char[data.size() + 1];

copy(data.begin(), data.end(), charArray);

charArray[data.size()] = '\n';

WriteFile(COMport, charArray, data.size() + 1, &temp, &overlappedwr);

signal = WaitForSingleObject(overlappedwr.hEvent, INFINITE);

if ((signal == WAIT\_OBJECT\_0) && (GetOverlappedResult(COMport, &overlappedwr, &temp, true)))

{

cout << "Передача прошла успешно" << endl;

}

else

{

cout << "Ошибка передачи!" << endl;

}

SuspendThread(writer);

}

}

static DWORD WINAPI ReadThread(LPVOID)

{

COMSTAT comstat;

DWORD btr, mask, signal;

//DWORD temp;

overlapped.hEvent = CreateEvent(NULL, true, true, NULL);

SetCommMask(COMport, EV\_RXCHAR);

while (1)

{

WaitCommEvent(COMport, &mask, &overlapped);

signal = WaitForSingleObject(overlapped.hEvent, INFINITE);

if (signal == WAIT\_OBJECT\_0)

{

if (GetOverlappedResult(COMport, &overlapped, &bytesRidden, true)) // было GetOverlappedResult(COMport, &overlapped, &temp, true))

{

if ((mask & EV\_RXCHAR) != 0)

{

ClearCommError(COMport, &bytesRidden, &comstat); // было ClearCommError(COMport, &temp, &comstat)

btr = comstat.cbInQue;

if (btr)

{

ReadFile(COMport, bufrd, btr, &bytesRidden, &overlapped); // было ReadFile(COMport, bufrd, btr, &temp, &overlapped)

ReadPrinting();

}

}

}

}

}

}

void ReadPrinting(void)

{

cout << bufrd;

WriteFile(handle, bufrd, bytesRidden, &bytesWritten, 0); // было WriteFile(handle, bufrd, sizeof(bufrd), &bytesWritten, 0)

memset(bufrd, 0, BUFSIZE);

}

void COMOpen()

{

string portname = "COM4";

DCB dcb;

COMMTIMEOUTS timeouts;

COMport = CreateFileA(portname.c\_str(), GENERIC\_READ | GENERIC\_WRITE, 0, NULL, OPEN\_EXISTING, FILE\_FLAG\_OVERLAPPED, NULL);

if (COMport == INVALID\_HANDLE\_VALUE)

{

if (GetLastError() != 0)

{

cout << "Ошибка №" << GetLastError() << " ";

cout << "Порт не доступен" << endl;

CloseHandle(COMport);

}

else

{

cout << "Неизвестная ошибка!" << endl;

CloseHandle(COMport);

}

}

dcb.DCBlength = sizeof(DCB);

//считать структуру DCB из порта

if (!GetCommState(COMport, &dcb))

{

COMClose();

cout << "Получить настройки не удалось или порт не настроен!" << endl;

CloseHandle(COMport);

}

else

{

//инициализация структуры DCB

dcb.BaudRate = CBR\_9600;

dcb.fBinary = TRUE; //включаем двоичный режим обмена

dcb.fOutxCtsFlow = FALSE; //выключаем режим слежения за сигналом CTS

dcb.fOutxDsrFlow = FALSE; //выключаем режим слежения за сигналом DSR

dcb.fDtrControl = DTR\_CONTROL\_DISABLE; //отключаем использование линии DTR

dcb.fDsrSensitivity = FALSE; //отключаем восприимчивость драйвера к состоянию линии DSR

dcb.fNull = FALSE; //разрешить приём нулевых байтов

dcb.fRtsControl = RTS\_CONTROL\_DISABLE; //отключаем использование линии RTS

dcb.fAbortOnError = FALSE; //отключаем остановку всех операций чтения/записи при ошибке

dcb.ByteSize = 8; //задаём 8 бит в байте

dcb.Parity = 0; //отключаем проверку чётности

dcb.StopBits = 0; //задаём один стоп-бит

//загрузить структуру DCB в порт

if (SetCommState(COMport, &dcb) == 0) // Функция повторно инициализирует все аппаратные и управляющие настройки, но не очищает очереди вывода или ввода данных (отправляем параметры DCB в порт)

{

cout << "Нельзя настроить порт (DCB)" << endl;

CloseHandle(COMport);

}

//установить таймауты

timeouts.ReadIntervalTimeout = 0; //таймаут между двумя символами

timeouts.ReadTotalTimeoutMultiplier = 0; //общий таймаут операции чтения

timeouts.ReadTotalTimeoutConstant = 0; //константа для общего таймаута операции чтения

timeouts.WriteTotalTimeoutMultiplier = 0; //общий таймаут операции записи

timeouts.WriteTotalTimeoutConstant = 0; //константа для общего таймаута операции записи

//записать структуру таймаутов в порт

if (!SetCommTimeouts(COMport, &timeouts))

{

CloseHandle(COMport);

cout << "Не удалось установить тайм-ауты" << endl;

}

//установить размеры очередей приёма и передачи

SetupComm(COMport, 2000, 2000);

//создать или открыть существующий файл для записи принимаемых данных

handle = CreateFileA("test.txt", GENERIC\_WRITE | GENERIC\_READ, 0, NULL, OPEN\_ALWAYS, 0, NULL);

if (!handle)

{

cout << "Ошибка создания файла" << endl;

CloseHandle(handle);

}

PurgeComm(COMport, PURGE\_RXCLEAR | PURGE\_TXCLEAR);

reader = CreateThread(NULL, 0, ReadThread, NULL, 0, NULL);

writer = CreateThread(NULL, 0, WriteThread, NULL, CREATE\_SUSPENDED, NULL);

}

}

void COMClose()

{

if (writer)

{

TerminateThread(writer, 0);

CloseHandle(overlappedwr.hEvent);

CloseHandle(writer);

}

if (reader)

{

TerminateThread(reader, 0);

CloseHandle(overlapped.hEvent);

CloseHandle(reader);

}

CloseHandle(COMport);

COMport = 0;

CloseHandle(handle);

handle = 0;

}

int main()

{

setlocale(LC\_ALL, "Rus");

COMOpen();

WriteThread(0);

}