Министерство образования Республики Беларусь

Учреждение образования

БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ

ИНФОРМАТИКИ И РАДИОЭЛЕКТРОНИКИ

Факультет компьютерных систем и сетей

Кафедра электронных вычислительных машин

ОТЧЕТ

по лабораторным работам № 1 и 2

на тему

**ПОСЛЕДОВАТЕЛЬНЫЙ ПОРТ**

и

**ПАКЕТНАЯ ПЕРЕДАЧА ДАННЫХ**

Студент: А. В. Гуринович

Руководитель:  В. А. Марцинкевич

Минск 2021

**Цель работы**: разработать модули асинхронной побайтной передачи данных и пакетной передачи данных по последовательному порту.

Модули должен быть оформлен в виде библиотеки функций или класса. Разработанный модули буд использованы в следующих лабораторных работах.

**Задание к лабораторной работе**:

1. Разработать программный модуль реализации процедуры передачи (приема) байта информации через последовательный интерфейс.
2. В программах синхронно изменить скорости передачи и приема байта до минимальной и максимальной. Проверить функционирование звена приемопередачи.
3. Установить различные скорости для приемника и передатчика. Проверить функционирование звена приемопередачи.
4. Разработать программный модуль реализации процедуры пакетной передачи (приема) информации через последовательный интерфейс.

**Код Программы:**

//

// Package.hpp

// TFCN, 2

//

// Created by Andrej Hurynovič on 23.10.21.

//

#ifndef Package\_hpp

#define Package\_hpp

#include <array>

#include <typeinfo>

#include <boost/archive/text\_oarchive.hpp>

#include <boost/archive/text\_iarchive.hpp>

#include <boost/serialization/vector.hpp>

#include <boost/serialization/split\_member.hpp>

**using** **namespace** std;

**const** size\_t packDataMaximalSize = 150;

**struct** Pack {

**unsigned** **long** startPoint;

**unsigned** **long** parent;

size\_t dataSize;

**char** data[packDataMaximalSize];

size\_t packSize;

**template**<**class** Archive>

**void** save(Archive &archive) **const** {

archive & **this**->startPoint;

archive & **this**->parent;

archive & **this**->dataSize;

archive & **this**->packSize;

archive & **this**->data;

}

**template**<**class** Archive>

**void** load(Archive &archive) {

archive & **this**->startPoint;

archive & **this**->parent;

archive & **this**->dataSize;

archive & **this**->packSize;

archive & **this**->data;

}

BOOST\_SERIALIZATION\_SPLIT\_MEMBER()

**friend** **class** boost::serialization::access;

**friend** ostream& **operator** << (ostream &out, **struct** Pack &pack) {

out << "Pack information: " << endl;

out << "Parent: " << pack.parent << endl << endl;

out << "Data: " << pack.data << endl << endl;

out << "Start point: " << pack.startPoint << endl;

out << "Data size: " << pack.dataSize << endl;

out << "Pack size: " << pack.packSize << endl;

**return** out;

}

};

**const** size\_t maxPackSize = **sizeof**(**struct** Pack) \* 2 + 100; // 100 - boost reserved space

**class** Package {

**private**:

**struct** Pack value;

**public**:

**constexpr** Package() : value{} {}

**void** start() **noexcept**;

**char**\* getData() **noexcept**;

**template** <**typename** Type>

**void** changeData(Type &data) {

**this**->value.parent = **typeid**(data).hash\_code();

memset(**this**->value.data, '\0', packDataMaximalSize);

memcpy(**this**->value.data, data, **sizeof**(data) \* **sizeof**(\*data));

**this**->value.dataSize = **sizeof**(data);

}

**template**<**class** Archive>

**void** save(Archive &archive) **const** {

archive & **this**->value;

}

**template**<**class** Archive>

**void** load(Archive &archive) {

archive & **this**->value;

}

BOOST\_SERIALIZATION\_SPLIT\_MEMBER()

**friend** ostream& **operator**<<(ostream &out, Package &P);

**private**:

**void** setFlag() **noexcept**;

**void** set\_sender() **noexcept**;

**void** set\_recipiend() **noexcept**;

**void** set\_other\_flag() **noexcept**;

**void** set\_size\_pack() **noexcept**;

**friend** **class** boost::serialization::access;

**private**:

**const** **unsigned** **long** fstart = 02001;

};

#endif /\* Package\_hpp \*/

//

// Package.cpp

// TFCN, 2

//

// Created by Andrej Hurynovič on 23.10.21.

//

#include "Package.hpp"

**void** Package::set\_size\_pack() **noexcept** {

**this**->value.packSize = **sizeof**(**this**->value);

}

**void** Package::setFlag() **noexcept** {

**this**->value.startPoint = **this**->fstart;

}

**void** Package::set\_other\_flag() **noexcept** {

}

**void** Package::start() **noexcept** {

setFlag();

// set\_other\_flag();

set\_size\_pack();

}

**char**\* Package::getData() **noexcept** {

**return** **this**->value.data;

}

std::ostream& **operator**<<(std::ostream &out, Package &pack) {

out << pack.value;

**return** out;

}

//

// Pseudoports.hpp

// TFCN, 2

//

// Created by Andrej Hurynovič on 23.10.21.

//

#ifndef Pseudoports\_hpp

#define Pseudoports\_hpp

#include <string>

#include <unistd.h>

#include <fcntl.h>

#include <memory>

#include "PseudoPortSettings.hpp"

#include "Package.hpp"

**class** Pseudoports {

**private**:

**int** masterDescriptor;

std::string port;

PseudoPortSettings \*settings;

**public**:

Pseudoports();

~Pseudoports();

std::string create\_port();

**void** closePort();

**bool** isOpen();

std::string getPortName();

**void** changeSpeed(**const** **int** &speed);

std::string readPort(**const** std::size\_t size);

size\_t writePort(**const** std::string str);

**private**:

**void** initPortSettings();

**void** flushPortBuffer();

**void** error(std::string msg);

};

#endif /\* Pseudoports\_hpp \*/

//

// Pseudoports.cpp

// TFCN, 2

//

// Created by Andrej Hurynovič on 23.10.21.

//

#include "Pseudoports.hpp"

Pseudoports::Pseudoports() {

**this**->masterDescriptor = -1;

**this**->settings = **nullptr**;

}

Pseudoports::~Pseudoports() {

**if** (**this**->settings != **nullptr**)

**delete** **this**->settings;

}

std::string Pseudoports::create\_port() {

**if** (**this**->isOpen())

**this**->error("Info: The pseudo terminal has already been created!");

**this**->masterDescriptor = posix\_openpt(O\_RDWR | O\_NOCTTY | O\_NDELAY);

fcntl(**this**->masterDescriptor, F\_SETFL, 0);

**if** (**this**->masterDescriptor < 0) **this**->error("Func: create port\nInfo: open port");

**if** (grantpt(**this**->masterDescriptor) < 0)

**this**->error("Func: create port\nInfo: grantpt");

**if** (unlockpt(**this**->masterDescriptor) < 0)

**this**->error("Func: create port\nInfo: unlockpt");

**this**->port = ptsname(**this**->masterDescriptor);

**if** (**this**->port.empty())

**this**->error("Func: create port\nInfo: ptsname");

**this**->initPortSettings();

**return** **this**->port;

}

**void** Pseudoports::closePort() {

**if** (**this**->isOpen()) {

close(**this**->masterDescriptor);

**this**->masterDescriptor = -1;

**this**->port.clear();

}

}

**bool** Pseudoports::isOpen() {

**return** **this**->masterDescriptor > 0;

}

**void** Pseudoports::error(std::string msg) {

**throw** msg;

}

std::string Pseudoports::getPortName() {

**return** **this**->port;

}

std::string Pseudoports::readPort(**const** size\_t size) {

**if** (!**this**->isOpen())

**this**->error("Func: read port\nInfo: Port no open!");

**long** n = 0;

**char** buffer[size + 1];

**if** ((n = read(**this**->masterDescriptor, &buffer, size)) < 0)

**this**->error("Func: read port.\nInfo: Failed to read");

buffer[n] += '\0';

**return** buffer;

}

size\_t Pseudoports::writePort(**const** std::string str) {

**if** (!**this**->isOpen())

**this**->error("Func: write port\nInfo: Port no open!");

**this**->flushPortBuffer();

**return** write(**this**->masterDescriptor, str.c\_str(), str.size());

}

**void** Pseudoports::initPortSettings() {

**int** settingDescriptor = open(**this**->port.c\_str(), O\_RDWR | O\_NOCTTY | O\_NDELAY);

**this**->settings = **new** PseudoPortSettings(settingDescriptor);

}

**void** Pseudoports::flushPortBuffer() {

**if** (tcflush(**this**->masterDescriptor, TCIFLUSH) < 0) {

**this**->error("Func: flush\_port\_buffer\nError: tcflush");

}

}

**void** Pseudoports::changeSpeed(**const** **int** &speed) {

**if** (!isOpen())

**this**->error("Func: Pseudoports::change\_speed\nInfo: port no open.");

**this**->settings->setInputSpeed(speed);

**this**->settings->setOutputSpeed(speed);

}

//

// PseudoPSettings.hpp

// TFCN, 2

//

// Created by Andrej Hurynovič on 23.10.21.

//

#ifndef PseudoPSettings\_hpp

#define PseudoPSettings\_hpp

#include <termios.h>

#include <string>

**class** PseudoPortSettings {

**private**:

**int** descriptor;

**struct** termios oldSettings;

**struct** termios newSettings;

**public**:

PseudoPortSettings();

PseudoPortSettings(**int** descriptor);

~PseudoPortSettings();

**bool** isOpen();

**void** setOutputSpeed(**const** speed\_t speed);

**void** setInputSpeed(**const** speed\_t speed);

**void** setWaitTime(**const** cc\_t min, **const** cc\_t time);

**void** getParametersTerminal();

**void** getOldParametersTerminal();

**void** flushPortBuffer();

**private**:

**void** setAllFlags();

**void** setCFlags();

**void** setLFlags();

**void** setIFlags();

**void** setOFlags();

**void** setParametersTerminal();

**void** setOldParametersTerminal();

**void** clearTermios();

**void** error(**const** std::string msg);

};

#endif /\* PseudoPSettings\_hpp \*/

//

// PseudoPSettings.cpp

// TFCN, 2

//

// Created by Andrej Hurynovič on 23.10.21.

//

#include "PseudoPortSettings.hpp"

PseudoPortSettings::PseudoPortSettings(**const** **int** descriptor) {

**this**->descriptor = descriptor;

**if** (!**this**->isOpen()) **this**->error("Info: Setting port parameters\nError: The file descriptor is not valid!");

**this**->clearTermios();

**this**->getOldParametersTerminal();

**this**->setAllFlags();

**this**->setInputSpeed(B9600);

**this**->setOutputSpeed(B9600);

**this**->setWaitTime(1, 10);

**this**->flushPortBuffer();

}

PseudoPortSettings::~PseudoPortSettings() {

**this**->setOldParametersTerminal();

}

**void** PseudoPortSettings::setAllFlags() {

**this**->getParametersTerminal();

**this**->setIFlags();

**this**->setLFlags();

**this**->setOFlags();

**this**->setCFlags();

**this**->setParametersTerminal();

}

**void** PseudoPortSettings::setIFlags() {

**this**->newSettings.c\_iflag |= (INPCK | ISTRIP);

**this**->newSettings.c\_iflag &= ~(IXON | IXOFF | IXANY);

}

**void** PseudoPortSettings::setLFlags() {

**this**->newSettings.c\_lflag &= ~(ICANON | ECHO | ECHOE | ISIG);

}

**void** PseudoPortSettings::setOFlags() {

**this**->newSettings.c\_oflag &= ~OPOST;

}

**void** PseudoPortSettings::setCFlags() {

**this**->newSettings.c\_cflag |= (CLOCAL | CREAD);

**this**->newSettings.c\_cflag |= CS8;

}

**void** PseudoPortSettings::setInputSpeed(**const** speed\_t speed) {

**this**->getParametersTerminal();

**if** (cfsetispeed(&**this**->newSettings, speed))

**this**->error("Func: set\_speed\_in\_port\nError: cfsetispeed");

**this**->setParametersTerminal();

}

**void** PseudoPortSettings::setOutputSpeed(**const** speed\_t speed) {

**this**->getParametersTerminal();

**if** (cfsetospeed(&**this**->newSettings, speed))

**this**->error("Func: set\_speed\_out\_port\nError: cfsetospeed");

**this**->setParametersTerminal();

}

**void** PseudoPortSettings::setWaitTime(**const** cc\_t min, **const** cc\_t time) {

**this**->getParametersTerminal();

**this**->newSettings.c\_cc[VTIME] = time;

**this**->newSettings.c\_cc[VMIN] = min;

**this**->setParametersTerminal();

}

**void** PseudoPortSettings::setParametersTerminal() {

**if** (tcsetattr(**this**->descriptor, TCSANOW, &**this**->newSettings))

**this**->error("Func: set\_old\_param\_terminal -> oldSettings");

}

**void** PseudoPortSettings::getParametersTerminal() {

**if** (tcgetattr(**this**->descriptor, &**this**->newSettings))

**this**->error("Func: get\_old\_param\_terminal -> tcgetattr");

}

**void** PseudoPortSettings::setOldParametersTerminal() {

**if** (tcsetattr(**this**->descriptor, TCSANOW, &**this**->oldSettings))

**this**->error("Func: set\_old\_param\_terminal -> oldSettings");

}

**void** PseudoPortSettings::getOldParametersTerminal() {

**if** (tcgetattr(**this**->descriptor, &**this**->oldSettings))

**this**->error("Func: get\_old\_param\_terminal -> tcgetattr");

}

**void** PseudoPortSettings::flushPortBuffer() {

**if** (tcflush(**this**->descriptor, TCIFLUSH) < 0) {

**this**->error("Func: flush\_port\_buffer\nError: tcflush");

}

}

**void** PseudoPortSettings::clearTermios() {

bzero(&**this**->newSettings, **sizeof**(**this**->newSettings));

bzero(&**this**->oldSettings, **sizeof**(**this**->oldSettings));

}

**bool** PseudoPortSettings::isOpen() {

**return** descriptor > 0;

}

**void** PseudoPortSettings::error(**const** std::string msg) {

**throw** msg;

}

//

// SerialPorts.hpp

// TFCN, 2

//

// Created by Andrej Hurynovič on 23.10.21.

//

#ifndef SerialPorts\_hpp

#define SerialPorts\_hpp

#include <string>

#include <termios.h>

#include <unistd.h>

#include <fcntl.h>

#include "PseudoPortSettings.hpp"

#include "Package.hpp"

**class** SerialPorts

{

**private**:

**int** descriptor;

std::string port;

PseudoPortSettings \*settings;

**public**:

SerialPorts();

SerialPorts(std::string port);

~SerialPorts();

**void** startPort();

**void** openPort();

**void** openPort(**const** std::string port);

**void** closePort();

**bool** isOpen();

size\_t writePort(**const** std::string str);

std::string readPort(**const** size\_t size);

**void** changeSpeed(**const** **int** &speed);

**private**:

**void** flushPortBuffer();

**void** initPortSettings();

// bool change\_law\_port(const fs::path path, const mode\_t mode);

**void** error(**const** std::string msg);

};

#endif

//

// SerialPorts.cpp

// TFCN, 2

//

// Created by Andrej Hurynovič on 23.10.21.

//

#include "SerialPorts.hpp"

#include <iostream>

SerialPorts::SerialPorts() {

**this**->descriptor = -1;

**this**->settings = **nullptr**;

}

SerialPorts::SerialPorts(std::string port) : SerialPorts() {

**this**->port = port;

openPort();

}

SerialPorts::~SerialPorts() {

**if** (**this**->settings != **nullptr**)

**delete** **this**->settings;

}

**void** SerialPorts::openPort() {

**if** (**this**->port.empty())

**this**->error("Func: open port\nPort name not specified.");

**this**->descriptor = open(**this**->port.c\_str(), O\_RDWR | O\_NOCTTY | O\_NDELAY);

**if** (!isOpen())

**this**->error("Func: open\_port\nInfo: Unable to open file.");

fcntl(**this**->descriptor, F\_SETFL, 0);

**this**->initPortSettings();

}

**void** SerialPorts::openPort(**const** std::string port) {

**this**->port = port;

openPort();

}

**void** SerialPorts::closePort() {

**if** (**this**->isOpen()) {

close(**this**->descriptor);

**this**->descriptor = -1;

}

}

**bool** SerialPorts::isOpen() {

**return** **this**->descriptor > 0;

}

**void** SerialPorts::error(**const** std::string msg) {

**throw** msg;

}

size\_t SerialPorts::writePort(**const** std::string str) {

**if** (!**this**->isOpen())

**this**->error("Func: read\_port\nInfo: Port no open!");

**this**->flushPortBuffer();

**return** write(**this**->descriptor, str.c\_str(), str.size());

}

std::string SerialPorts::readPort(**const** size\_t size) {

**if** (!**this**->isOpen())

**this**->error("Func: write\_port\nInfo: Port no open!");

**long** n = 0;

**char** buffer[size + 1];

**if** ((n = read(**this**->descriptor, &buffer, size)) < 0)

**throw** "Func: read port.\nInfo: Failed to read";

buffer[size] += '\0';

**return** buffer;

}

**void** SerialPorts::flushPortBuffer() {

**if** (tcflush(**this**->descriptor, TCIFLUSH) < 0) {

**this**->error("Func: flush\_port\_buffer\nError: tcflush");

}

}

**void** SerialPorts::initPortSettings() {

**int** settingDescriptor = open(**this**->port.c\_str(), O\_RDWR | O\_NOCTTY | O\_NDELAY);

**this**->settings = **new** PseudoPortSettings(settingDescriptor);

}

**void** SerialPorts::changeSpeed(**const** **int** &speed) {

**if** (!isOpen())

**this**->error("Func: Pseudoports::change\_speed\nInfo: port no open.");

**this**->settings->setInputSpeed(speed);

**this**->settings->setInputSpeed(speed);

}

//

// cmain.cpp

// TFCN, 2

//

// Created by Andrej Hurynovič on 23.10.21.

//

//

#include <iostream>

#include "Package.hpp"

#include "Pseudoports.hpp"

#include <array>

#include <sstream>

**using** **namespace**::std;

**static** **int** portMode = 1;

**void** clearTerm() {

system("clear");

}

**void** confirmation() {

cout << "\nconfirm to continue...";

cin.ignore(numeric\_limits<streamsize>::max(), '\n');

cin.ios\_base::clear();

getchar();

clearTerm();

}

**void** menuOut(Pseudoports &port) {

cout << "Port name: " << port.getPortName();

cout << endl << "Port mode: " << portMode;

cout << endl;

cout << "1. Send message." << endl;

cout << "2. Accept message." << endl;

cout << endl;

cout << "3. Send package." << endl ;

cout << "4. Accept package." << endl;

cout << endl;

cout << "5. Change speed." << endl;

cout << "6. Change mode." << endl;

cout << endl;

cout << "7. Exit." << endl;

}

**void** changeSpeed(Pseudoports &port) {

std::string speed;

cout << "Type speed: ";

cin >> speed;

port.changeSpeed(stoi(speed));

confirmation();

}

**void** changeMode(Pseudoports &P) {

cout << "\tPort mode\n";

cout << "1. Read. " << "2. Write";

cin >> portMode;

confirmation();

}

**void** sendPackage(Pseudoports &P) {

**if** (portMode != 2) {

cout << "Port mode: Read.\n";

cout << "In this mode, you cannot send a pack.";

confirmation();

**return**;

}

**char** message[packDataMaximalSize];

cout << "Set message pack: ";

cin >> message;

Package pack;

pack.changeData(message);

pack.start();

std::stringstream ss;

boost::archive::text\_oarchive wr(ss);

wr & pack;

cout << "\ndispatch...." << endl;

P.writePort(ss.str());

confirmation();

**return**;

}

**void** sendMessage(Pseudoports &P) {

**if** (portMode != 2) {

cout << "Port mode: Read.\n";

cout << "In this mode, you cannot send a message.";

confirmation();

**return**;

}

std::string message;

cout << "Set message: ";

cin >> message;

P.writePort(message);

confirmation();

}

**void** closePort(Pseudoports &P) {

P.closePort();

portMode = 0;

confirmation();

}

**void** acceptMessage(Pseudoports &P) {

**if** (portMode != 1) {

cout << "Port mode: Write.\n";

cout << "It is impossible to receive a message in this mode.";

confirmation();

**return**;

}

size\_t size;

cout << "Set size: ";

cin >> size;

cout << "Waiting for a message....";

string message = P.readPort(size);

clearTerm();

cout << "Message: " << message;

confirmation();

}

**void** acceptPack(Pseudoports &P) {

**if** (portMode != 1) {

cout << "Port mode: Write.\n";

cout << "It is impossible to receive a pack in this mode.";

confirmation();

**return**;

}

Package pack;

stringstream ss(P.readPort(maxPackSize));

boost::archive::text\_iarchive rd(ss);

rd & pack;

cout << pack;

confirmation();

}

**int** main(**int** argc, **const** **char** \* argv[]) {

**try** {

Pseudoports P;

P.create\_port();

array<function<**void**(Pseudoports &)>, 7> menu = {

sendMessage,

acceptMessage,

sendPackage,

acceptPack,

changeSpeed,

changeMode

};

**unsigned** **int** answer = 0;

**while**(**true**) {

menuOut(P);

cin >> answer;

answer--;

**if** (answer > 6)

**break**;

clearTerm();

menu[answer](P);

}

} **catch** (std::string **const** &message) {

std::cout << "\t-Error-\n" << std::endl;

std::cout << message << std::endl;

}

**return** 0;

}

//

// smain.cpp

// TFCN, 2

//

// Created by Andrej Hurynovič on 23.10.21.

//

#include <iostream>

#include "SerialPorts.hpp"

#include <array>

#include <sstream>

**using** **namespace**::std;

**int** portMode = 2;

string namePort;

**void** clearTerm() {

system("clear");

}

**void** confirmation() {

cout << "\nconfirm to continue...";

cin.ignore(numeric\_limits<streamsize>::max(), '\n');

cin.ios\_base::clear();

getchar();

clearTerm();

}

**void** menuOut(SerialPorts &S) {

cout << "Port name: " << namePort;

cout << endl;

cout << "1. Send message." << endl;

cout << "2. Accept message." << endl;

cout << endl;

cout << "3. Send package." << endl ;

cout << "4. Accept package." << endl;

cout << endl;

cout << "5. Change speed." << endl;

cout << "6. Change mode." << endl;

cout << endl;

cout << "7. Exit." << endl;

}

**void** changeSpeed(SerialPorts &S) {

std::string speed;

cout << "Set speed: ";

cin >> speed;

S.changeSpeed(stoi(speed));

confirmation();

}

**void** changeMode(SerialPorts &S) {

cout << "\tPort mode\n";

cout << "1. Read. " << "2. Write";

cin >> portMode;

confirmation();

}

**void** sendPackage(SerialPorts &S) {

**if** (portMode != 2) {

cout << "Port mode: Read.\n";

cout << "In this mode, you cannot send a pack.";

confirmation();

**return**;

}

**char** message[packDataMaximalSize];

cout << "Set message pack: ";

cin >> message;

Package pack;

pack.changeData(message);

pack.start();

std::stringstream ss;

boost::archive::text\_oarchive wr(ss);

wr & pack;

cout << "\ndispatch...." << endl;

S.writePort(ss.str());

confirmation();

**return**;

}

**void** sendMessage(SerialPorts &S) {

**if** (portMode != 2) {

cout << "Port mode: Read.\n";

cout << "In this mode, you cannot send a message.";

confirmation();

**return**;

}

std::string message;

cout << "Set message: ";

cin >> message;

S.writePort(message);

confirmation();

}

**void** closePort(SerialPorts &S) {

S.closePort();

portMode = 0;

confirmation();

}

**void** acceptMessage(SerialPorts &S) {

**if** (portMode != 1) {

cout << "Port mode: Write.\n";

cout << "It is impossible to receive a message in this mode.";

confirmation();

**return**;

}

size\_t size;

cout << "Set size: ";

cin >> size;

cout << "Waiting for a message....";

string message = S.readPort(size);

clearTerm();

cout << "Message: " << message;

confirmation();

}

**void** acceptPackage(SerialPorts &S) {

**if** (portMode != 1) {

cout << "Port mode: Write.\n";

cout << "It is impossible to receive a pack in this mode.";

confirmation();

**return**;

}

Package pack;

stringstream ss(S.readPort(maxPackSize));

boost::archive::text\_iarchive rd(ss);

rd & pack;

cout << pack;

confirmation();

}

**int** main(**int** argc, **const** **char** \* argv[]) {

**try** {

SerialPorts S;

cout << "Set name port: ";

cin >> namePort;

S.openPort(namePort);

array<function<**void**(SerialPorts &)>, 7> menu = {

sendMessage,

acceptMessage,

sendPackage,

acceptPackage,

changeSpeed,

changeMode

};

**unsigned** **int** answer = 0;

**while**(**true**) {

menuOut(S);

cin >> answer;

answer--;

**if** (answer > 6)

**break**;

clearTerm();

menu[answer](S);

}

} **catch** (std::string **const** &message) {

std::cout << "\t-Error-\n" << std::endl;

std::cout << message << std::endl;

}

**return** 0;

}

**Вывод:** разработаны простые и удобные модули для эмуляции последовательного порта. В будущем модули можно использовать для последующих лабораторных работ.