МИНОБРНАУКИ РОССИИ

Федеральное государственное автономное образовательное
учреждение высшего образования
«Южный федеральный университет»

Институт математики, механики
и компьютерных наук им. И. И. Воровича

Кафедра информатики и вычислительного эксперимента

Крылов Владислав Вячеславович

РАЗРАБОТКА КРОССПЛАТФОРМЕННОЙ IDE ДЛЯ PASCALABC.NET

ВЫПУСКНАЯ КВАЛИФИКАЦИОННАЯ РАБОТА
по направлению подготовки
02.03.02 – Фундаментальная информатика и информационные технологии

Научный руководитель –
зав. кафедрой, к. ф.-м. н. Михалкович Станислав Станиславович

Допущено к защите:
Руководитель направления 02.03.02 _____ Михалкович С.С.

Ростов-на-Дону – 2025

1. Оглавление

Задание

на выпускную квалификационную работу

студента 4 курса Крылова Владислава Вячеславовича

Разработка кроссплатформенной IDE для PascalABC.NET

Цель работы: создать кроссплатформенную IDE для PascalABC.NET.

Для достижения указанной цели решить следующие задачи

1. Взять за основу свободно распространяемую IDE Red Panda C++ и проанализировать возможность ее модификации и встраивания стороннего компилятора
2. Изменить интерфейс IDE Red Panda C++, приблизив его к IDE PascalABC.NET
3. Разработать и реализовать архитектуру, позволяющую взаимодействовать IDE Red Panda C++ и серверу компилятора.
4. Разработать и реализовать архитектуру взаимодействия IDE Red Panda C++ с сервером Intellisense PascalABC.NET по протоколу LSP
5. Реализовать и описать развертывание проекта под Linux и под Windows

Научный руководитель,

зав. кафедрой информатики

и вычислительного эксперимента Михалкович С.С.

2. Введение

В условиях стремительного развития информационных технологий и их внедрения во все сферы человеческой жизни особую актуальность приобретают удобные и гибкие инструменты обучения программированию систем различной сложности. Такой набор инструментов предоставляет, например, среда PascalABC.NET [1]. Она содержит в себе высокоуровневый язык PascalABC.NET, сочетающий в себе простоту и лаконичность языка Pascal с современными технологиями платформы .NET. Язык PascalABC.NET поддерживает как процедурное, так и объектно-ориентированное и функциональное программирование, его стандартная библиотека содержит множество удобных и гибких методов. Более того, он полностью совместим с платформой .NET, позволяет использовать любые её библиотеки. В поставку среды PascalABC.NET также входит функциональная и легковесная IDE. Среди поддерживаемых ею возможностей сборка и запуск программ на языке PascalABC.NET, подсветка синтаксиса, а также мощная система автодополнения IntelliSense.

Среда PascalABC.NET разрабатывается с использованием технологий .NET Framework и Windows Forms. Это обеспечивает полную совместимость с операционной системой Windows, что стало одной из причин популярности PascalABC.NET в том числе в образовательных учреждениях. Однако в настоящее время всё большую долю рынка операционных систем занимают системы на базе свободно распространяемого ядра Linux. В России такая динамика связана в первую очередь с развитием отечественного ПО. Среди операционных систем можно выделить AstraLinux, AltLinux и РедОС. Отечественное ПО активно поддерживается со стороны государства, интегрируется в работу государственных учреждений [2]. Так, в административных и медицинских учреждениях уже сейчас происходит переход на ОС AstraLinux . Схожие процессы происходят в школах, колледжах

и техникумах, университетах. К сожалению, компания Microsoft не предоставляет поддержки ядра Linux в платформах .NET Framework и Windows Forms. Текущая версия PascalABC.NET для Linux для своей работы использует свободную стороннюю реализацию API .NET Framework под названием Mono Project. Однако Mono не обеспечивает полной совместимости, что приводит к нестабильной работе, ограниченной функциональности и проблемам с запуском на большинстве дистрибутивов Linux. Альтернативы в виде использования технологий виртуализации или Wine, свободной реализации Windows API, технически сложны и непригодны для применения в обучении основам программирования.

Целью данной работы является разработка IDE для PascalABC.NET, отвечающая требованиям кроссплатформенности. Особое внимание уделено запуску и работе на операционных системах, основанных на ядре Linux.

3. Выбор тестовой системы

Основанное на принципах свободного распространения, ядро Linux породило целое семейство операционных систем. В настоящее время поддерживаются и разрабатываются сотни дистрибутивов. Среди них можно выделить наиболее популярных подсемейства [3]:

- Debian (производные Ubuntu, Mint и т.д.)
- RHEL (производные Fedora, Rocky и т.д.)
- Arch Linux (производные Manjaro, EndeavourOS и т.д.)

Для разработки и тестирования кроссплатформенной IDE была выбрана операционная система Debian. Это обусловлено следующими причинами:

- Приоритет стабильности системы перед другими факторами
- Популярность среди сообщества
- Совместимость и переносимость
- Широкий выбор отечественного ПО (современные отечественные ОС в большинстве своём основаны на Debian)

Итоговая спецификация тестируемой системы с точки зрения программного обеспечения:

- Архитектура: amd64
- Дистрибутив: Debian 12 (bookworm)
- Версия ядра Linux: 6.1.0
- Среда рабочего стола: KDE Plasma 5.27.5

Также, для проверки кроссплатформенности разрабатываемой IDE тестирование проводилось также на системе с установленной Windows 11.

4. Среда Red Panda C++

Red Panda C++ – небольшая легковесная IDE, разрабатываемая на фреймворке Qt [4]. Распространяется под лицензией GNU GPLv3. Данная среда разработки содержит минимальное количество кода, поддерживает многоязычность с выбором компилятора (среди встроенных языков C, C++, Assembler, Lua). Исходный код приложения полностью открыт и доступен для модификации.

4.1. Процесс компиляции в Red Panda C++

Блок-схема управляющего потока программы в процессе компиляции приведена ниже на рисунке 2:

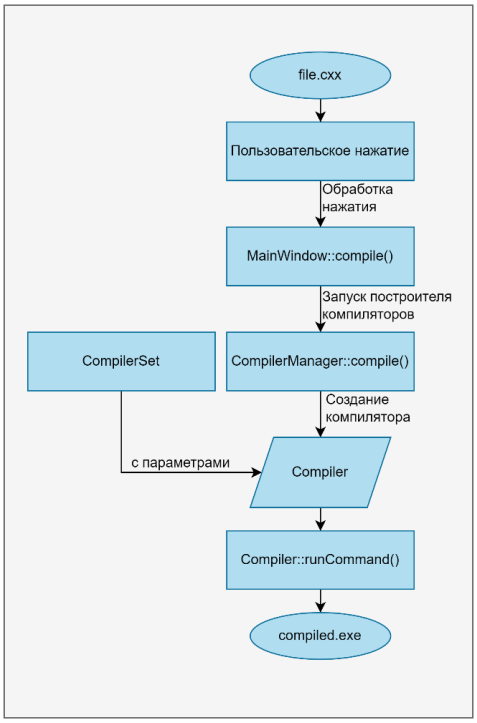

Рисунок 1. Блок-схема процесса компиляции в Red Panda C++

Среда Red Panda C++ не выполняет полную компиляцию программ сама по себе. Для компиляции она использует внешние программы (например, существующие компиляторы GCC, G++ и т.д). Для этого имплементирован класс CompilerSet. Он содержит в себе различные параметры: путь к внешним компиляторам, путь к дополнительным включаемым директориям,

расширение исполняемого файла, дополнительные аргументы программы и так далее. Данные параметры могут настраиваться как программно, так и пользователем динамически во время выполнения работы IDE. Один из таких объектов выбирается в качестве компилятора по умолчанию. Именно он и используется в дальнейшем.

При нажатии пользователем на кнопку Compile класс главного окна MainWindow реагирует, вызывая метод compile(). Данный метод поручает классу CompilerManager создать компилятор класса Compiler с параметрами, извлекаемыми из CompilerSet, выставленного по умолчанию. Compiler является управляющим классом. Он создаёт процесс внешнего компилятора, задаёт ему программу в методе runCommand. Compiler управляет и запуском полученного исполняемого файла.

5. Клиент-серверная архитектура среды PascalABC.NET

Платформа PascalABC.NET состоит из двух составных частей: компилятор языка PascalABC.NET, называемый консольным, и IDE. Консольный компилятор может быть запущен независимо, взаимодействие с пользователем организуется через интерактивное окно эмулятора терминала DOS. В случае запуска вместе с визуальной оболочкой IDE принимает на себя контроль как за пользовательским вводом, так и за процессом консольного компилятора. Компилятор в этом случае запускается в особом режиме /noconsole, который ожидает ввода пользовательских команд в стандартный ввод и выводит результаты в стандартный вывод.

Компилятор и IDE являются независимыми сущностями и находятся в отдельных процессах. IDE при этом является управляющим процессом по отношению к консольному компилятору, может настраивать параметры запуска, посылать сигналы на завершение. Между процессами организуется межпроцессное взаимодействие (IPC) средствами платформы .NET, основанное на связывании друг с другом потоков ввода/вывода. Поток стандартного вывода IDE связывается со стандартным вводом консольного компилятора, а поток стандартного вывода консольного компилятора – со стандартным вводом IDE. На листинге 1 приведен код, реализующий такое связывание:

```
pabcnetcProcess = new Process();
 if (!IsUnix())
 {
     pabcnetcProcess.StartInfo.FileName =
Path.Combine(Tools.GetExecutablePath(), pabcnetcFileName);
     pabcnetcProcess.StartInfo.Arguments = "commandmode";
 }
 else
 {
     pabcnetcProcess.StartInfo.FileName = "mono";
```

```
     pabcnetcProcess.StartInfo.Arguments =
Path.Combine(Tools.GetExecutablePath(), pabcnetcFileName)+"
commandmode";

 }

 pabcnetcProcess.StartInfo.UseShellExecute = false;
 pabcnetcProcess.StartInfo.CreateNoWindow = true;
 pabcnetcProcess.StartInfo.RedirectStandardOutput = true;
 pabcnetcProcess.StartInfo.RedirectStandardInput = true;
 pabcnetcProcess.StartInfo.RedirectStandardError = true;
 pabcnetcProcess.EnableRaisingEvents = true;
 pabcnetcProcess.StartInfo.StandardOutputEncoding =
System.Text.Encoding.UTF8;
 pabcnetcProcess.Exited += pabcnetcProcess_Exited;
 pabcnetcStreamReader.Remove(inputId);
 pabcnetcProcess.Start();
 //pabcnetcProcess.StandardInput.Encoding;
 pabcnetcStreamReader.Add(pabcnetcProcess.StandardOutput,
inputId, inputEncoding);
 compilerReloading = false;
```

Листинг 1. Связывание стандартных потоков ввода/вывода для взаимодействия с консольным компилятором

Далее при взаимодействиях пользователя, связанных с компилятором, IDE посылает в свой стандартный вывод команду в формате некоторого протокола. Этот протокол называется командным режимом консольного компилятора (commandmode). Формат сообщения имеет следующий вид (рис. 1):

Рисунок 2. Формат сообщения в коммандном режиме консольного компилятора PascalABC.NET

1. cmd – Заголовок с номером команды. Формат команды – десятичное число вида 2xx. Примеры некоторых команд консольного компилятора:
    - 200 – завершение работы компилятора

- 215 – установить компилятору имя файла
- 210 – скомпилировать установленный в опциях файл

2. # – Разделяющий символ
3. data – Данные, необходимые команде, по необходимости разделенные символом из пункта 2
4. Символ конца строки LF, который является завершающим

Ответы посылаются компилятором в свободном формате в строковом виде.

Итого протокол можно описать следующим порядком:

1. IDE запускает процесс компилятора
2. По протоколу режима commandmode IDE обменивается с консольным компилятором сообщениями
3. Перед завершением работы IDE завершает работу компилятора

В данный момент в версии для Linux взаимодействие полностью идентично. Единственная разница состоит в том, что и IDE, и консольный компилятор запускаются через прослойку Mono. Такая попытка обеспечить кроссплатформенность возможна, но не обеспечивает ни гладкости интерфейса, ни ожиданий по стабильности использования. IDE требуется переделать с использованием другого, кроссплатформенного фреймворка.

6. Архитектура компиляции в новой IDE

6.1. Место встраивания для запуска программ на PascalABC.NET

Для работы программ на PascalABC.NET наивным решением кажется использование версии компилятора pabcnetcclear.exe. Она принимает в себя путь к файлу программы и параметры компиляции в качестве аргументов командной строки. Данный подход очень близок к компиляторам языков C/C++, например, gcc, что позволяет бесшовно встроить pabcnetcclear в Red Panda C++. Однако такой способ имеет несколько фундаментальных недостатков, главный из которых – низкая производительность. Много данных консольная версия компилятора PascalABC.NET кэширует, что значительно ускоряет компиляцию, а также позволяет запускать компилятор на старых машинах.

Было принято решение в случае компиляции программы на PascalABC.NET прерывать процесс компиляции ещё в главном окне (классе MainWindow). В обработчик события нажатия добавлено ветвление. Если файл программы имеет расширение “.pas”, он компилируется с помощью консольного компилятора PascalABC.NET, иначе следует стандартная процедура компиляции в Red Panda C++. Данное ветвление продемонстрировано на листинге:

```
if (editor->filename().contains(".pas")) {
ExternalCompilerManager::instance().compile(editor->filename());
} else {
mCompilerManager->compile(editor->filename(),editor-
>fileEncoding(),rebuild,compileType);
}
```

Листинг 2. Ветвление при компиляции файла в Red Panda C++

Здесь mCompilerManager – указатель на общий CompilerManager, стандартный для Red Panda C++.

Визуализация новой архитектуры представлена на рисунке:

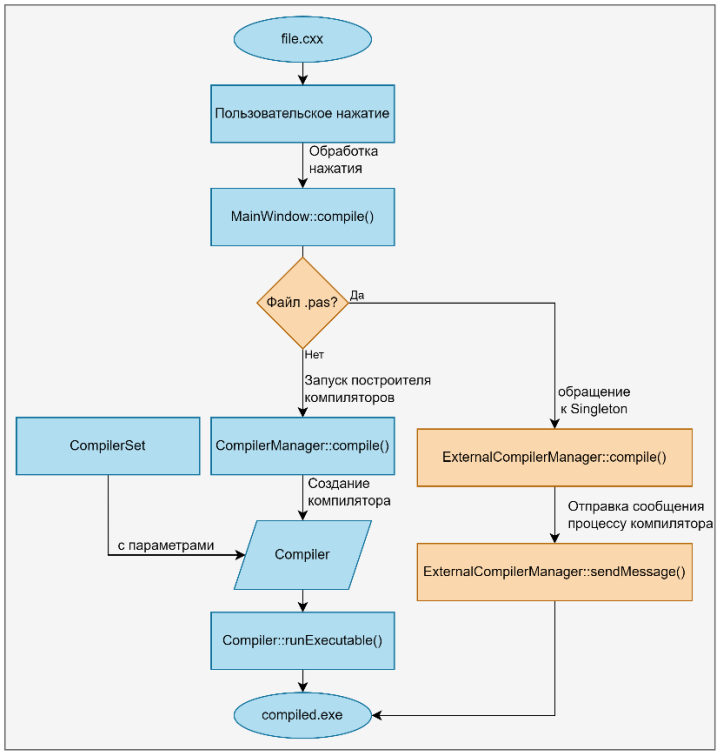

Рисунок 3. Новая архитектура Red Panda C++ с поддержкой PascalABC.NET

О том, что означает ExternalCompilerManager, пойдёт речь далее.

6.1.1. Использование паттерна Singleton для управления процессом компилятора

Одиночка (англ. Singleton) – порождающий паттерн проектирования [5]. Он гарантирует, что в приложении будет единственный экземпляр некоторого класса. Кроме того, на данный класс предоставляется единственная глобальная точка доступа. Данный паттерн позволяет удобно управлять объектом класса, при этом не добавляя излишние объявления в глобальное пространство имён.

Данный паттерн был использован при создании класса ExternalCompilerManager, управляющего процессом компилятора. Именно он запускает консольный компилятор, отвечает за отправку сообщений, предоставляет интерфейс для компиляции. Данный синглтон инстанцируется в функции main – точке входа Qt приложения, незадолго до создания главного

окна. Сразу после инстанцирования запускается и процесс компилятора PascalABC.NET.

6.2. Управление процессами в различных операционных системах

6.2.1. ОС-специфические API

Самым низкоуровневым способом создания и управления процессами является использование API, предоставляемого операционной системой. Для Windows это Win32 API, для ОС на базе Linux это методы, описанные в стандарте POSIX. Чтобы обеспечить кроссплатформенность в данном случае, архитектура должна содержать несколько реализаций, соответствующих разным спецификациям. Выбор нужной осуществляется на этапе компиляции. Это возможно обеспечить с помощью макросов, определяемых в случае использования определенной операционной системы. Это могут быть как определенные вручную макросы, так и макросы, предоставляемые кроссплатформенными библиотеками. В приложении приведён код создания процесса с использованием Win32 API, POSIX и выбором операционной системы на основе определенных библиотекой Qt макросов Q_OS_WINDOWS и Q_OS_UNIX.

Данный метод совершенно не подходит для реализации кроссплатформенности. Любой связанный с процессами программный код необходимо повторять дважды.

6.2.2. QProcess

Библиотека Qt предоставляет кроссплатформенный способ управления процессами [6]. Теперь для создания процесса можно воспользоваться конструктором и методами класса для задания программы и режима запуска. Такой подход гораздо более перспективен. Во-первых, используются методы и классы из библиотеки Qt, что позволяет разрабатывать проект внутри одного

фреймворка. Во-вторых, данные методы не требуют изменений при смене операционной системы. Единственное, что может измениться – это путь к запускаемому приложению и его аргументы. Так как версия консольного компилятора PascalABC.NET для Linux всё ещё требует mono для запуска, именно оно и будет передаваться в качестве запускаемой программы, а путь к PascalABC.NET станет её аргументом. Итоговый код создания процесса приведен ниже на листинге 3:

```
#ifdef Q_OS_WINDOWS
    QString path_to_pas = "D:\\Sci\\pascalabcnet-
zmq\\bin\\pabcnetc.exe";
    //QString path_to_pas = pSettings->dirs().appDir() +
"/../PascalABCNETLinux/pabcnetc.exe"; // Path to the C#
executable
    compilerProcess->setProgram(path_to_pas);
    compilerProcess-
>setProcessChannelMode(QProcess::SeparateChannels);
    compilerProcess->setArguments(QStringList() << "/noconsole"
<< "commandmode");
#else
    QString path_to_mono = "mono";
    compilerProcess->setProgram(path_to_mono);
    compilerProcess-
>setProcessChannelMode(QProcess::SeparateChannels);
    compilerProcess->setArguments(QStringList() << (pSettings-
>dirs().appDir() + "/../PascalABCNETLinux/pabcnetc.exe").c_str()
<< "/noconsole" << "commandmode");
#endif
connect(compilerProcess,
QOverload<QProcess::ProcessError>::of(&QProcess::errorOccurred),
this, handlePascalError);
compilerProcess->start(QProcess::ReadWrite);
```

Листинг 3. Управление процессом консольного компилятора PascalABC.NET

6.3. Способы организации межпроцессного взаимодействия

Для взаимодействия процессов друг с другом существует множество возможных методов. В ходе работы были рассмотрены и протестированы некоторые из них. Методы выбирались с учетом требований к кроссплатформенности полученной архитектуры.

6.3.1. Потоки ввода/вывода процессов

Данный способ используется в архитектуре PascalABC.NET. Потоки ввода управляющего и управляемого процессов привязываются к потокам вывода друг друга. В Qt дочерний процесс по умолчанию связывается с родительским таким образом. Запись в управляемый процесс производится с помощью метода QProcess::write(), а чтение результата – с помощью метода QProcess::readAllStandardOutput(). Со стороны дочернего процесса взаимодействие идентично, как если бы был организован стандартный ввод/вывод. Данный способ удобен, ведь не требует никаких внешних зависимостей. Однако, к сожалению, было обнаружено, что программы, написанные на фреймворке .NET версии после 3.5 не могут взаимодействовать в таком режиме с программой на Qt [7]. Данную проблему решить не удалось. Вместо этого были опробованы другие способы.

6.3.2. Именованные каналы

Именованные каналы – один из способов взаимодействия между процессами. В системах на базе UNIX они реализованы через файловые дескрипторы и системный вызов mkfifo() [8]. Он создаёт особый объект файловой системы с заданным названием. Данный объект предоставляет дескрипторы на чтение и запись, аналогичные файловым, однако не содержит сам по себе никаких данных. Ядром организуется принцип First-In-First-Out (FIFO) – все записанные в данный объект данные тут же отправляются всем процессам, открывшим объект на чтение. Для корректной работы объект FIFO должен быть открыт как на чтение, так и на запись. Чтение без записи либо блокирует процесс, либо не возвращает ничего (в зависимости от режима открытия канала), запись без чтения посылает записывающему процессу сигнал SIGPIPE, который по умолчанию приводит к завершению записывающего процесса. В системах на базе Windows именованные каналы являются специальными файловыми объектами [9]. Ими управляет система

специальной файловой системой с именем Named Pipe File System (NPFS). Интерфейс взаимодействия с каналами в Windows построен по принципу клиент-сервер, реализуется через функции WinAPI CreateNamedPipe, ConnectNamedPipe, CallNamedPipe. Каналы в Windows можно использовать и для передачи информации по сети.

Для именованных каналов существуют обёртки как в .NET [10], так и в платформе Qt [11]. При их рассмотрении было предпринято решение создать простейший макет для симуляции планируемой архитектуры: клиента на Qt (в будущем IDE), сервера на C# .NET Framework (в будущем компилятор PascalABC.NET). На месте клиента находится простейшее консольное приложение, отправляющее некоторые строковые значения, и печатающее полученные ответы от сервера. На месте сервера приложение, принимающее данные строковые значения, и отправляющее их же в ответ. Реализация макета сервера на C# .NET Framework приведена ниже на листинге 4:

```
const string pipeName = "testpipe";
using (var server = new NamedPipeServerStream(pipeName,
PipeDirection.InOut, 1, PipeTransmissionMode.Byte)) {
server.WaitForConnection();
Console.WriteLine("Client connected!");
while (server.IsConnected) {
     byte[] buffer = new byte[1024];
     int bytesRead = server.Read(buffer, 0, buffer.Length);
     if (bytesRead > 0) {
          string recievedString = Encoding.UTF8.GetString(buffer,
0, bytesRead);
          Console.WriteLine($"Recieved {recievedString}");
          string response = $"Echo {recievedString}";
          byte[] responseBytes =
     Encoding.UTF8.GetBytes(response);
          server.Write(responseBytes, 0, responseBytes.Length);
          server.Flush();
     }}
     Console.WriteLine("Client disconnected!");
}
```

Листинг 4. Реализация сервера в C# с использованием именованных каналов

Использован встроенный класс .NET NamedPipeServerStream, оборачивающий работу с именованными каналами в интерфейс потока языка C#.

Для реализации на Qt были использованы сокеты, подключенные к именованному каналу. Код класса, инкапсулирующего взаимодействие по именованному каналу, приведен ниже на листинге 5:

```
class NamedPipeClient: public QObject {
    //Q_OBJECT
public:
    NamedPipeClient(QObject *parent = nullptr) {
        socket = new QLocalSocket();
        connect(socket, &QLocalSocket::readyRead, this,
&NamedPipeClient::handleReadyRead);
        connect(socket, &QLocalSocket::errorOccurred,
[](QLocalSocket::LocalSocketError error) {
            qDebug() << "Socket error:" << error;
        });
    }
    void connectToServer(const QString& pipeName) {
        QString fullPath = pipeName;
#ifndef Q_OS_UNIX
        fullPath = "\\\\.\\pipe\\" + pipeName;
#else
        fullPath = "CoreFxPipe_" + pipeName;
#endif
        socket->connectToServer(fullPath);
        if (!socket->waitForConnected(5000)) {
            qDebug() << "Connection failed:" << socket-
>errorString();
        } else {
            qDebug() << "Connected to server!";
        }
    }
    void sendMessage(const QString& message) {
        if (socket->state() == QLocalSocket::ConnectedState) {
            socket->write(message.toLocal8Bit().data());
            socket->flush();
        } else {
            qDebug() << "Socket not connected to server!";
        }
    }
private slots:
    void handleReadyRead() {
        QTextStream qout(stdout);
        QByteArray response = socket->readAll();
        qout << response << '\n';
    }
private:
    QLocalSocket *socket;
};
```

Листинг 5. Реализация клиента в C++ Qt с использованием именованных каналов

Оказалось, что из-за особенностей платформы .NET на Windows обращение к именованному каналу требовало задания пути согласно формату, используемый файловой системой NPFS. На GNU/Linux системах .NET Framework запускается через Mono, поэтому путь к именованному каналу также требует особого названия. Запись в поток реализуется с помощью метода socket->write(), чтение происходит асинхронно по сигналу readyRead.

С использованием именованных каналов был достигнут некоторый успех: два процесса смогли связаться друг с другом как на платформе Windows, так и на GNU/Linux. Но у работы с именованными каналами возникали проблемы, связанные с запуском .NET через Mono, а также асинхронным вводом-выводом. К тому же, создание и использование на Qt отдельного класса обёртки над сокетами значительно усложняет отладку и ставит вопрос о масштабируемости в случае взаимодействия с таким сложным процессом, как компилятор PascalABC.NET. Было принято решение искать другие способы, не требующие построения отдельного слоя абстракции.

6.3.3. Библиотека ZeroMQ

ZeroMQ (также **ØMQ**, **ZMQ**) – сетевая библиотека для обмена сообщениями [12]. Действует на основе сокетов, предоставляет абстракции для построения масштабируемых систем. Библиотека отличается от классических систем обмена сообщениями, таких как, например, RabbitMQ, полным отсутствием централизованного брокера. Обмен сообщениями реализуется через очередь. Сокеты ZMQ транспортно-независимы, поддерживают как низкоуровневые протоколы IPC (через Unix Domain Sockets), так и высокоуровневый протокол TCP. Библиотека предоставляет несколько шаблонов взаимодействия:

1. Request-Response (REQ/REP) – клиент-серверная модель

2. Publish-Subscribe (PUB/SUB) – массовая рассылка сообщений
3. Push-Pull (PUSH/PULL) – балансировка нагрузки
4. Router-Dealer (ROUTER/DEALER) – асинхронное расширение REQ/REP
5. Pair (PAIR) – связь «точка-точка» между сокетами

Библиотека ZMQ имеет официальные обёртки под множество языков программирования. Среди них C#, где существует нативная реализация ZMQ для .NET под названием NetMQ, и обёртка на C API для интеграции со стандартной библиотекой C++ cppzmq. Использование сокетов требует минимальных усилий. Так, для реализации шаблона REQ/REP в C++ поверх протокола TCP требуется написать код, приведенный на листинге 6:

```
zmq::context_t context(1);
zmq::socket_t socket(context, zmq::socket_type::req);
socket.connect("tcp://127.0.0.1:5555");
zmq::message_t request("Hello world!", 12);
socket.send(request, zmq::send_flags::none);
zmq::message_t reply;
socket.recv(reply, zmq::recv_flags::none);
```

Листинг 6. Пример REQ клиента ZMQ в C++

По аналогии код получателя запроса в C# представлен на листинге 7:

```
var socket = new ResponseSocket("tcp://127.0.0.1:5555"));
string rec = socket.ReceiveFrameString();
socket.SendFrame("Received: " + reply);
```

Листинг 7. Пример REP сервера ZMQ в C#

Использование данной библиотеки позволяет описывать взаимодействие в простом, компактном, почти декларативном виде, не думая об особенностях реализации протокола общения. Именно этот подход и был выбран для построения архитектуры новой IDE в PascalABC.NET.

6.4. Новая архитектура взаимодействия IDE и компилятора PascalABC.NET

На основе описанных выше технологий была разработана новая архитектура программного взаимодействия с компилятором в новой IDE. При запуске IDE создаёт процесс консольного компилятора, управляет им с помощью инструментов класса QProcess. С определенной периодичностью IDE перезапускает процесс компилятора для освобождения занятой компилятором памяти. Далее IDE ведёт себя как клиент, отправляет запросы к компилятору по протоколу, описанному в главе 5. Консольный компилятор PascalABC.NET ведёт себя как сервер, принимает запросы и отправляет ответы в произвольном строковом виде. Для реализации взаимодействия было принято решение использовать шаблон REQ/REP, продемонстрированный в главе 6.3.3

Схематически архитектура представлена на рисунке 4:

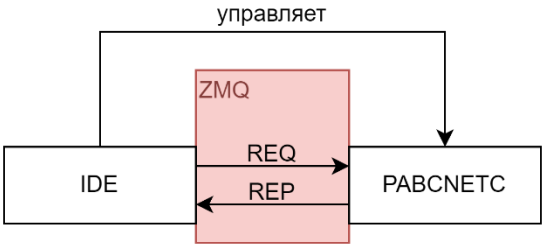

Рисунок 4. Архитектура взаимодействия IDE с компилятором PascalABC.NET

Для реализации подобной архитектуры в командном режиме консольного компилятора любая работа через стандартный ввод-вывод заменена на методы ReceiveFrameString() и SendFrame() объекта ResponseSocket библиотеки NetMQ. Для соответствия шаблону REQ/REP некоторым командам была добавлена отправка ответа в строковом виде. Полный код класса ExternalCompilerManager, отвечающего за взаимодействие с компилятором PascalABC.NET, приведен в приложении 1.

Новая архитектура была успешно протестирована на тестовых системах. Компиляция производилась успешно как на GNU/Linux, так и на Windows. Запуск программ на Windows IDE совершает с помощью консоли cmd.exe, на GNU/Linux же IDE запускает программу через Mono в эмуляторе терминала, указанном в переменной среды XTERM. Пример выполненной программы на Windows приведен на рисунке 5:

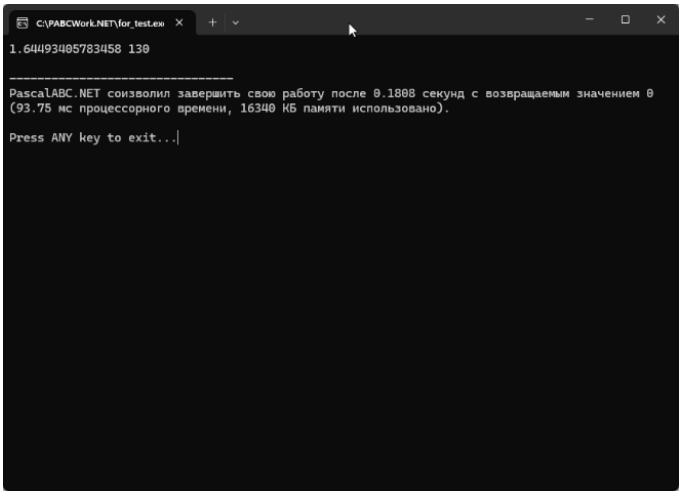

Рисунок 5. Результат выполнения программы на PascalABC.NET в новой IDE

7. Изменение внешнего вида новой IDE

Изменение функционала и предназначения Red Panda C++ требует внесения изменений и во внешний вид для соответствия поддержке другого языка программирования.

7.1. Подсветка синтаксиса для PascalABC.NET

Одной из необходимых компонент современной IDE является возможность подсветки синтаксических конструкций поддерживаемых языков. В существующий на данный момент IDE для PascalABC.NET поддержка синтаксиса реализована с помощью AvalonEdit [13]. Правила подсветки синтаксиса задаются с помощью специальных xml-подобных файлов с расширением .xshd. К сожалению, AvalonEdit доступна только для платформы .NET, в других средах такого удобства реализации нет. В Red Panda поддержка подсветки синтаксиса реализована другим образом. Для каждого поддерживаемого языка разработчиками был написан легковесный парсер. Данный факт очень сильно усложняет добавление новых языков. Тем не менее, разработчики Red Panda имплементировали в свою IDE поддержку подсветки синтаксиса языка Lua. Вместо написания отдельного парсера для PascalABC.NET было принято решение добавить в проект несколько модифицированную версию уже разработанного парсера Lua. Полученная подсветка синтаксиса представлена на рисунке 6:

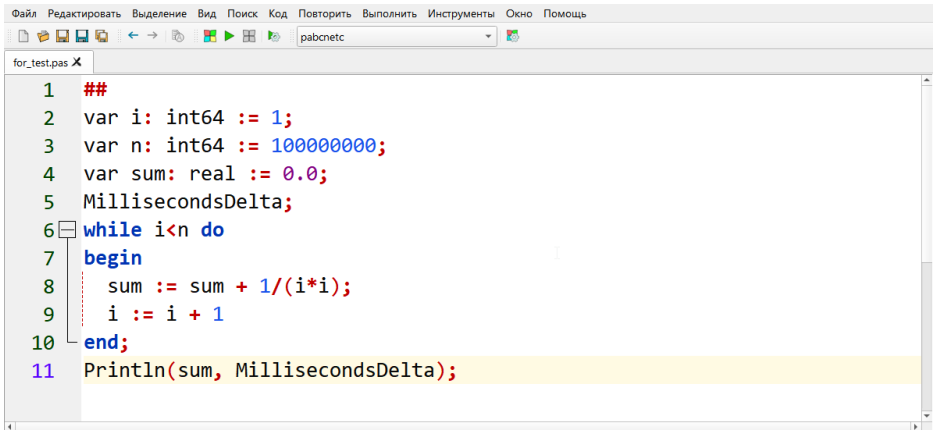

Рисунок 6. Пример работы подсветки синтаксиса

7.2. Перевод главного меню IDE

Основным языком пользователей среды PascalABC.NET в большинстве случаев является русский, поэтому новая оболочка обязана предоставлять возможность отображения элементов меню на русском языке. Библиотека Qt предоставляет широкие возможности интернационализации и локализации приложений. Основным источником переводов являются специальные файлы формата .ts (translation source) [14]. Данный формат является XML-подобным. Базовой единицей в файле является сообщение message, у которого есть поля source и translation – текст на языке разработчика и перевод на целевой язык. Пример сообщения в формате .ts показан ниже на листинге 8:

```
<message>
    <location line="+499"/>
    <location line="+2252"/>
    <source>Files</source>
    <translation>Файлы</translation>
</message>
```

Листинг 8. Пример сообщения с переводом в формате .ts

Файлы .ts глубоко интегрированы в платформу Qt. Так, для них в составе Qt есть отдельная графическая среда редактирования – Qt Linguist. Однако среда разработки Red Panda C++ содержит огромное количество компонент меню. Разрабатывать перевод вручную не эффективно. Для подготовки перевода было решено применить экспериментальный подход.

В среде Red Panda C++ уже подготовлен перевод на португальский язык. Готовый .ts файл, состоящий из сообщений message, по частям отправлялся на вход генеративному ИИ (модели DeepSeek и ChatGPT) с промптом: “Replace Portuguese text in <translation> field with Russian one”. Xml-подобный файл успешно был проанализирован моделью, и был получен перевод, уже отформатированный в .ts. Для добавления перевода необходимо было лишь добавить его в файл сборки проекта qmake. Для этого достаточно было изменить системную переменную TRANSLATIONS, содержащую список

используемых файлов перевода. Среда разработки успешно добавляет полученный перевод в систему автоматически. Теперь его можно выбрать в главном меню. После перезагрузки IDE перевод применяется. Вид нового перевода представлен на рисунке 7:

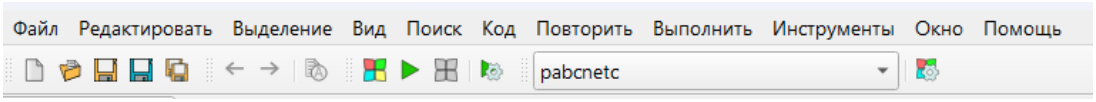

Рисунок 7. Перевод главного меню

7.3. Отображение ошибок компиляции

Консольный компилятор языка PascalABC.NET способен печатать сообщение об ошибке. Сообщение имеет формат, показанный на рисунке 8:

```
[errno][line,col] filepath: msg
```

Рисунок 8. Формат сообщения консольному компилятору PascalABC.NET

Errno здесь – номер ошибки (ошибки нумеруются в порядке положения в файле), line, col – строка и столбец ошибки, filepath – путь к компилируемому файлу, и msg – сообщение ошибки.

Данное сообщение поступает IDE по протоколу ZMQ, а затем разбирается регулярным выражением. Далее создаётся объект класса ошибки, реализованного в Red Panda C++. Он и отображается на экран методами Red Panda C++. Подробный код приведён на листинге 9:

```
    QRegularExpression
regex(R"(\[0\]\[(\d+),(\d+)\]\s+(.:.*?:)\s+(.*?)(?=\s*\[\d+\]|$)
)");
    QRegularExpressionMatchIterator i = regex.globalMatch(msg);
    while (i.hasNext()) {
        QRegularExpressionMatch match = i.next();
        if (match.hasMatch()) {
            PCompileIssue issue =
std::make_shared<CompileIssue>();
            issue->line = match.captured(1).toInt();
            issue->column = match.captured(2).toInt();
            issue->filename = match.captured(3);
            issue->description = match.captured(4);
```

```
            issue->type = CompileIssueType::Error;
            pMainWindow->onCompileIssue(issue);
        }
    }
```

Листинг 9. Парсинг сообщения об ошибке компилятора PascalABC.NET

Пример отображения ошибки приведён на рисунке 9:

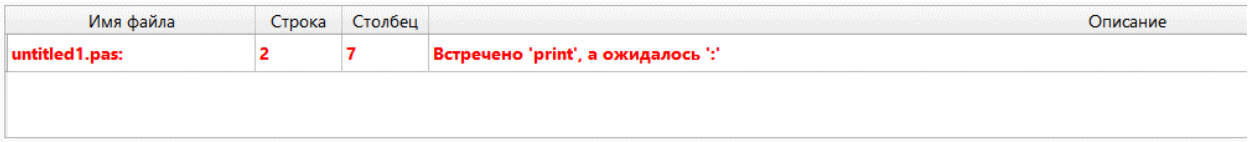

Рисунок 9. Пример сообщения об ошибке

7.4. Изменение внешнего вида главного окна IDE

Во время разработки было принято решение привести главное окно IDE к виду, близкому к PascalABC.NET. Для этого проанализированы виджеты, которые имплементированы в Red Panda C++. Виджеты и пункты меню, отвечающие за несоответствующий языку PascalABC.NET функционалу, были спрятаны. За сокрытие данных виджетов отвечает новая функция hideUIElements. Она вызывается в самом конце конструктора главного окна. Код функции hideUIElements приведен в приложении. Список убранных компонент меню:

- Обзорщик файлов
- Пункты меню, отвечающие за работу отладчика, генерацию ассемблерного кода, профилировщика, рефакторинг кода
- Пункты меню, отвечающие за справочную информацию по языкам C/C++, ассемблеру
- Удалены вкладки меню, отвечающие за консоль отладчика, работы тестов

Внешний вид обновленного главного окна представлен на рисунке 10:

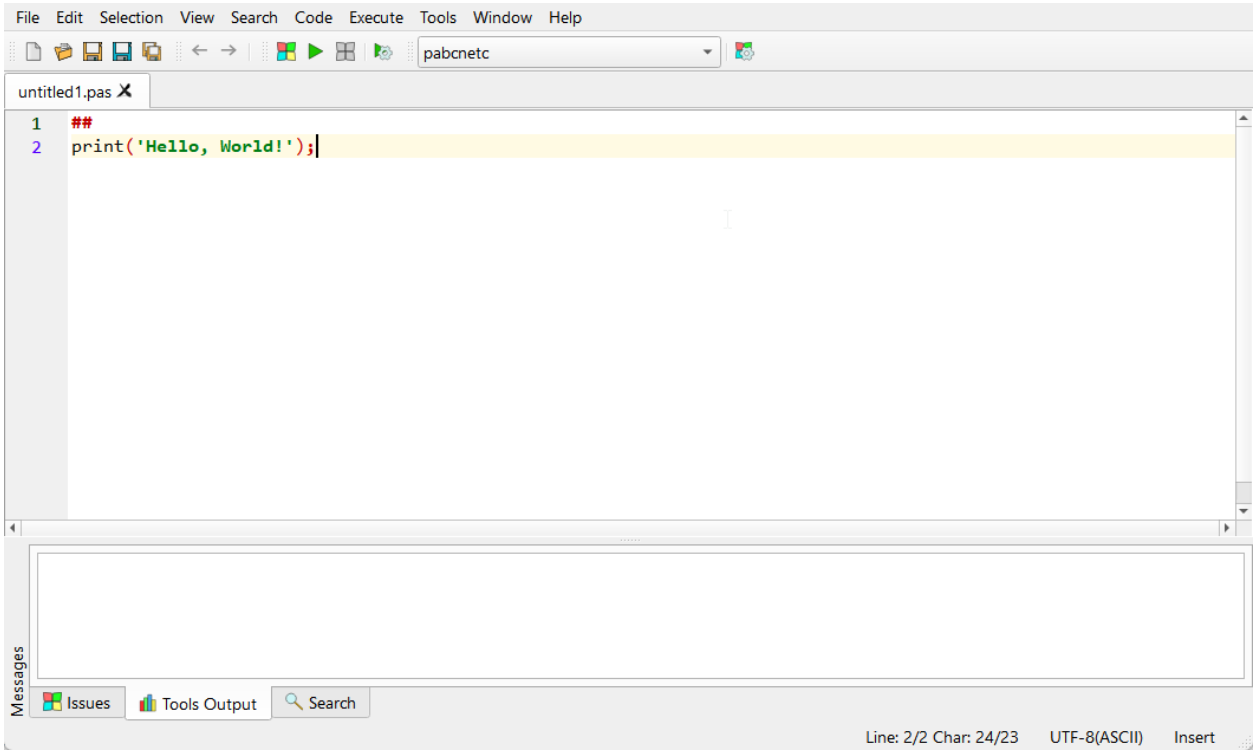

Рисунок 10. Главное окно новой IDE

Исходный код функции hideUIElements приведен в приложении 2.

8. Реализация системы IntelliSense в новой IDE

Одной из важнейших частей нынешней IDE PascalABC.NET является система автодополнения IntelliSense. Она значительно ускоряет написание кода, помогает в изучении языка.

Внутри платформы PascalABC.NET система IntelliSense была реализована внутренними средствами. Данный подход неприменим в новой IDE на основе Red Panda C++, где уже имплементировано автодополнение с помощью легковесного парсера языка C++. Вместо подстраивания под данные протоколы, было принято решение разработать новое, масштабируемое решение на основе существующего протокола LSP.

8.1. Протокол LSP

LSP (Language Server Protocol) – протокол, разработанный компанией Microsoft для обеспечения взаимодействия сред разработки и текстовых редакторов со специальными LSP серверами, предоставляющими возможности анализа кода. Основан на протоколе JSON-RPC (Remote Procedure Call). Архитектурно LSP следует клиент-серверной модели. LSP клиент – это компонент, интегрированный в редактор кода. Он отвечает за отправку запросов, обработку результата и визуализацию результата. LSP сервер – независимый процесс, реализующий различные возможности анализа программ на выбранном языке программирования. Взаимодействие между клиентом и сервером организуется через прикладной сетевой протокол JSON-RPC. Посылаемый объект запроса имеет структуру JSON документа со следующими полями:

- jsonrpc – версия протокола (на протяжении всей работы использована версия 2.0)
- method – название вызываемого метода
- params – массив данных, которые должны быть переданы методу, как параметры

- id – значение произвольного типа, используемое для синхронизации запроса и ответа.

Сервер должен отослать ответ на каждый запрос. Ответ содержит следующие поля:

- result – данные, которые вернул метод. Если во время выполнения метода произошла ошибка, поле должно быть установлено в null
- error – код и описание ошибки, если произошла ошибка во время выполнения метода, иначе null
- id – то же значение, что и в запросе, для синхронизации

В протоколе LSP сообщения также имеют заголовок Content-Length – длину посылаемого json-файла в байтах.

Главной единицей работы является текстовый документ, однозначно идентифицированный по URI. Поддерживаемые методы для документа включают в себя:

- Синхронизацию текста – методы открытия textDocument/didOpen, изменения файла textDocument/didChange, сохранения файла textDocument/didSave, закрытия textDocument/didClose. Работа с документом может осуществляться в двух режимах – полном (Full) и инкрементальном (Incremental). Полный режим подразумевает передачу всего текста программы при любом изменении, инкрементальная предполагает обмен точечными изменениями документа с указанием позиции.
- Автодополнение – метод textDocument/completion
- Справку по наведению – метод textDocument/hover
- Навигацию по документу – метод поиска определений textDocument/definition, объявлений textDocument/declaration, типов textDocument/typeDefinition, реализаций textDocument/implementation, ссылок textDocument/references
- Поиск символов – методы textDocument/documentSymbol

- Работа с кодом – методы переименования файла textDocument/rename, быстрого исправления textDocument/codeAction, форматирования textDocument/formatting
- Семантического анализа кода – методы поиска ошибок, предупреждений textDocument/publishDiagnostics, подсветки семантических токенов textDocument/semanticTokens
- Выделения ссылок – метод textDocument/documentHighlight
- Поиска информации о параметрах функции – метод textDocument/signatureHelp

Полная спецификация протокола хранится в открытом доступе и разрабатывается в том числе сообществом [15]. Протокол LSP стал де-факто стандартом в области инструментов разработки. Почти все современные IDE поддерживают работу с данным протоколом (это и множество плагинов Visual Studio Code, и инструменты JetBrains).

8.2. Существующая реализация протокола LSP в PascalABC.NET

В среде PascalABC.NET уже создаётся система IntelliSense на основе протокола LSP. В рамках её реализации была предоставлена обёртка LSP сервера в виде независимо запускаемого процесса. Для своей работы сервер использует библиотеку OmniSharp. Она позволяет удобно в высокоуровневом стиле создавать LSP сервера и модифицировать поведение в случае получения запросов на определённые методы. Для коммуникации сервер использует стандартные потоки ввода-вывода. Синхронизация текста документа производится в режиме Full.

8.3. Архитектура взаимодействия LSP сервера и кроссплатформенной IDE

Для реализации системы IntelliSense в новой кроссплатформенной IDE разработана новая архитектура. По аналогии с консольным компилятором LSP сервер является отдельным управляемым со стороны IDE процессом. Общение должно производиться с помощью библиотеки ZMQ по шаблону REQ/REP. Исходный код полученного класса IntelliSenseManager приведён в приложении 3.

Во время реализации возникли некоторые трудности, которые потребовали переработки изначального замысла. Так, LSP сервер на основе OmniSharp для ввода данных и вывода результата требует использования объектов, соответствующих интерфейсу Stream в C#. Библиотека NetMQ не предоставляет обёрток, наследуемых от Stream, над своими сокетами.

8.3.1. Использование паттерна Adapter для соединения LSP сервера с IDE

Адаптер (Adapter) – структурный паттерн проектирования [5]. Его предназначение – обеспечение взаимодействия объектов с несовместимыми интерфейсами путём создания объекта-посредника, выполняющего трансляцию вызовов и данных. Данный паттерн эффективен при необходимости интеграции независимо разработанных компонент, имеющих несоответствующие интерфейсы. С точки зрения целевых объектов взаимодействие никак не меняется, они считают, что общаются друг с другом напрямую.

Было принято добавить в архитектуру IntelliSense использование паттерна Adapter. Визуализация архитектуры приведена на рисунке 11:

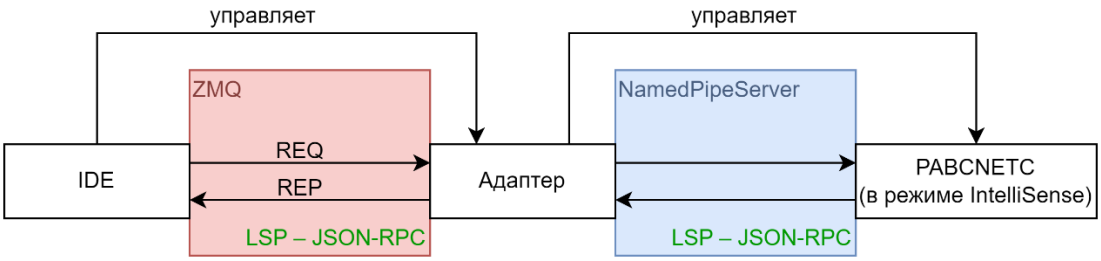

Рисунок 11. Архитектура взаимодействия IDE и компилятора по протоколу LSP

Теперь IDE взаимодействует не непосредственно с LSP сервером, а с объектом-посредником. При этом механизм посылки данных никак не меняется, с точки зрения IDE общение производится напрямую по протоколу LSP. Но управляет LSP сервером теперь объект посредник. Он принимает запрос от IDE с использованием библиотеки ZMQ, добавляет к json файлу заголовок Content-Length, отправляет полученные байты LSP серверу с помощью именованных каналов .NET. Именованные каналы .NET соответствуют интерфейсу C# Stream, поэтому данные по ним могут передаваться напрямую в LSP сервер, созданный с помощью OmniSharp. Результат сервер возвращает по этому же каналу. Объект-посредник получает ответ сервера, и передаёт его с помощью ZMQ в ответ на запрос IDE. Исходный код посредника приведен в приложении 4.

9. Упаковка новой IDE для различных операционных систем

Важнейшим этапом после разработки приложения является его сборка и распространение. Учитывая специфику платформы Qt, особое внимание требуется уделить сборке зависимостей и обеспечению запуска приложения на целевых системах без необходимости установки библиотек Qt вручную. Переносимая сборка также должна содержать версию консольного компилятора PascalABC.NET, систему IntelliSense.

9.1. Windows. Инструмент windeployqt

Для платформы Windows, файл формата PE (portable executable) является универсальным и уже сам по себе подходит для распространения вместе с папками ресурсов проекта. Однако для успешного запуска в случае динамической сборки требуются файлы DLL (dynamic link library), связанные с проектом. Платформа Qt состоит из огромного количества библиотек. Для того, чтобы определить и приложить к .exe файлу только необходимые библиотеки, в наборе инструментов Qt существует утилита windeployqt [16]. Он предназначен для автоматического копирования всех необходимых библиотек и ресурсов в целевую директорию рядом с исполняемым файлом. Инструмент анализирует заданный исполняемый файл и определяет его зависимости, включая динамически подключаемые библиотеки (.dll), необходимые модули Qt (такие как QtWidgets, QtGui, QtCore и другие), а также ресурсы, плагины и переводческие файлы, если они используются. Результатом работы является папка, содержащая исполняемый файл и все необходимые библиотеки. Данная папка готова к распространению, её можно запускать на всех платформах с соответствующей архитектурой (например, x64) без установки библиотек Qt.

9.2. Unix-системы. Формат AppImage. Инструмент linuxdeployqt

Для Unix-систем на основе ядра Linux задача построения переносимой сборки является более сложной. Это связано в первую очередь с разнообразием дистрибутивов ввиду открытости ядра. Наиболее распространёнными форматами бинарных пакетов являются .deb и .rpm, специфические для дистрибутивов на основе Debian и RHEL соответственно. Их использование более оптимизировано для данных семейств дистрибутивов, однако в условиях стремительного роста количества Linux-систем, необходимо полностью кроссплатформенное решение. Для этого сообществом Linux разработан универсальный формат AppImage [17]. Он не подразумевает под собой установку в привычном смысле этого слова. Вместо этого файл AppImage представляет собой один сжатый файл со всеми необходимыми для выполнения программы данными и всеми исполняемыми файлами формата ELF. При запуске файл монтируется как отдельная файловая система внутри модуля FUSE. Это позволяет абстрагироваться от конкретных реализаций бинарных пакетов внутри дистрибутивов. Запуск также осуществляется без прав администратора, что тоже является преимуществом (установка .deb или .rpm пакетов обычно требует прав суперпользователя или администратора). Формат AppImage поддерживается почти всеми дистрибутивами Linux, и был одобрен в том числе Линусом Торвальдсом [18].

Для сборки приложения в формат AppImage оно должно соответствовать некоторым требованиям. Все файлы должны находиться внутри папки с названием usr. Далее должна быть соблюдена следующая иерархия:

- Исполняемые файлы хранятся в usr/bin
- Qt, а также сторонние библиотеки хранятся в usr/lib
- Все необходимые ресурсы приложения хранятся в usr/share

- В папке usr находится файл AppRun, указывающий точку входа для приложения

Подобную структуру можно организовать с помощью qmake. Динамические библиотеки Qt можно загрузить в usr/lib с помощью утилиты linuxdeployqt [19]. Её функционал аналогичен windeployqt за исключением того, что linuxdeployqt способен на автоматическую генерацию AppImage файла после копирования библиотек. Следует отметить, что, как и в случае с Windows, сборка является динамической, поэтому использование подобных утилит необходимо.

Полученный образ успешно был запущен на тестовой машине. Было решено провести дополнительное тестирование на других дистрибутивов. Для запуска AppImage файла на некоторых дистрибутивах GNU/Linux (например, Ubuntu версии 24.02) потребовалось установить дополнительно библиотеку libfuse2 для поддержки технологии FUSE.

10. Заключение

В работе рассмотрена проблемы отсутствия кроссплатформенной версии IDE для PascalABC.NET. Проведен краткий анализ семейства операционных систем на базе ядра Linux, актуальности таких систем. Рассмотрены примеры современных IDE, ориентированные как на Windows, так и на GNU/Linux. Изучен кроссплатформенный фреймворк Qt. В результате исследования взята за основу IDE Red Panda C++. Изучены принципы её устройства.

На основе Red Panda C++ разработана кроссплатформенная версия оболочки PascalABC.NET. Разработана новая архитектура взаимодействия с консольным компилятором на основе библиотеки ZMQ. Для соответствия данной архитектуре изменен командный режим компилятора PascalABC.NET. Кроме того, изменен интерфейс IDE Red Panda C++. Меню переведено на русский язык, добавлена подсветка синтаксиса для PascalABC.NET, внешний вид приближен к существующей оболочке PascalABC.NET.

Кроме того, на основе существующей системы автодополнения в Red Panda C++ и системы IntelliSense языка PascalABC.NET была разработана новая архитектура взаимодействия автодополнения с IDE на основе протокола LSP. Для обеспечения взаимодействия применен паттерн Adapter.

Для полученного программного продукта подготовлены переносимые версии. Данные версии были протестированы на ОС Windows, а также на наиболее распространённом и поддерживаемом дистрибутиве GNU/Linux.

Таким образом, можно сказать, что все поставленные задачи были успешно выполнены. Новая IDE реализует весь основной функционал оболочки PascalABC.NET на основе Windows Forms. Среда разработки готова к применению, позволяет использовать все современные возможности языка PascalABC.NET в условиях стремительного распространения GNU/Linux систем.

11. Использование ИИ в рамках работы

Инструменты генеративного искусственного интеллекта занимают прочное положение в современном мире. Они предоставляют возможности удобного поиска, форматирования, структуризации информации. В рамках настоящей работы использование искусственного интеллекта уже было частично затронуто в главе 7.2. Инструмент ChatGPT удачно себя показал при выполнении таких массовых операций, как перевод XML-файла.

Кроме того, при работе над ВКР инструменты DeepSeek и ChatGPT были задействованы в различных областях. Так, генеративный ИИ помогал осуществлять поиск информации, существующих решений, библиотек. Важно учитывать, что не всё, сгенерированное ИИ, оказывалось верным, поэтому все факты были дополнительно уточнены и проверены. ИИ оказывал помощь при написании кода, генерировал черновики, демонстрирующие принципы работы некоторых компонент библиотеки Qt. ИИ также осуществлял контролируемое форматирование кода – расстановку пробелов, переносов. Был полезен ИИ и при написании настоящей работы. Он был использован для создания черновиков описаний различных протоколов и технологий: ZeroMQ, LSP, библиотек Qt. Данные черновики были полезны как каркас для создания лаконичного, академического текста. При этом такой сгенерированный текст не попал в настоящую работу.

Были и сферы, где технологии генеративного ИИ показали себя не с лучшей стороны. Так, ChatGPT и DeepSeek показали отрицательные результаты при попытках анализа кода Red Panda C++ и PascalABC.NET. В настоящий момент открытым чат-ботам недоступен крупный контекст кода: групп файлов, репозитория. Это делает его использование бесполезным, т. к. ИИ попросту придумывает факты. Кроме того, ИИ зачастую предлагается использовать для проверки грамматики и стиля, однако такое применение не нашло успеха в настоящей работе.

12. Список литературы

1. Бондарев И. В., Михалкович С. С. PascalABC.NET URL: https://pascalabc.net/
2. Указ Президента Российской Федерации от 30.03.2022 № 166 "О мерах по обеспечению технологической независимости и безопасности критической информационной инфраструктуры Российской Федерации" // Официальное опубликование правовых актов. 2022. URL: http://publication.pravo.gov.ru/Document/View/0001202203300001?index=1 (дата обращения: 08.06.2025).
3. Atea Ataroa Limited. DistroWatch URL: https://distrowatch.com/?language=RU (дата обращения: 05.06.2025).
4. Qu R. RedPanda C++ 2025. URL: https://github.com/royqh1979/RedPanda-CPP (дата обращения: 05.06.2025).
5. Э. Гамма, Р. Хелм, Р. Джонсон, Дж. Влиссидес. Паттерны объектно-ориентированного проектирования. СПб.: Питер, 2021. 448 с., ISBN 978-5-4461-1595-2
6. The Qt Company Ltd. QProccess Class // The Qt Company Ltd. Documentation. 2025. URL: https://doc.qt.io/qt-6/qprocess.html (дата обращения: 05.06.2025).
7. SimpleIsBetter. QProcess not communicating with.net framework > 3.5 // Qt Centre. 2015. URL: https://www.qtcentre.org/threads/62415-QProcess-not-communicating-with-net-framework-gt-3-5 (дата обращения: 05.06.2025).
8. Kerrisk M. mkfifo(3) - Linux manual page // man7.org. 2025. URL: https://man7.org/linux/man-pages/man3/mkfifo.3.html (дата обращения: 05.06.2025).

9. Microsoft. Именованные каналы - Win32 apps // Microsoft Learn. 2025. URL: https://learn.microsoft.com/ru-ru/windows/win32/ipc/named-pipes (дата обращения: 05.06.2025).
10. Microsoft. NamedPipeServerStream Класс // Microsoft Learn. 2025. URL: https://learn.microsoft.com/ru-ru/dotnet/api/system.io.pipes.namedpipeserverstream?view=net-8.0 (дата обращения: 05.06.2025).
11. The Qt Company Ltd. QLocalServer Class // The Qt Company Ltd. Documentation. 2025. URL: https://doc.qt.io/qt-6/qlocalserver.html (дата обращения: 05.06.2025).
12. The ZeroMQ authors. ZeroMQ // zeromq.org. 2025. URL: https://zeromq.org/ (дата обращения: 05.06.2025).
13. AvalonEdit Contributors. AvalonEdit URL: https://github.com/icsharpcode/AvalonEdit (дата обращения: 05.06.2025).
14. The Qt Company Ltd. TS file format // The Qt Company Ltd. Documentation. 2025. URL: https://doc.qt.io/qt-6/linguist-ts-file-format.html (дата обращения: 05.06.2025).
15. Microsoft. Official page for Language Server Protocol 2025. URL: https://microsoft.github.io/language-server-protocol/ (дата обращения: 05.06.2025).
16. The Qt Company Ltd.. Qt for Windows - Deployment // The Qt Company Ltd. Documentation. 2025. URL: https://doc.qt.io/qt-6/windows-deployment.html (дата обращения: 05.06.2025).
17. Peter S. AppImage 2015-19. URL: https://appimage.org/ (дата обращения: 05.06.2025).
18. Торвальдс Л. This is just very cool. // Google Plus. 2016. URL: https://archive.ph/PSqlB (дата обращения: 05.06.2025).

19. probonopd. linuxdeployqt URL: https://github.com/probonopd/linuxdeployqt?tab=readme-ov-file (дата обращения: 05.06.2025).

Приложение 1. Исходный код класса ExternalCompilerManager

externalcompilermanager.h:

```
#ifndef EXTERNALCOMPILERMANAGER_H
#define EXTERNALCOMPILERMANAGER_H
#include <QObject>
#include <QProcess>
#include <QTimer>
#include <zmq.hpp>
class ExternalCompilerManager : public QObject
{
    Q_OBJECT
public:
    static ExternalCompilerManager& instance();
    ExternalCompilerManager(const ExternalCompilerManager&) =
delete;
    ExternalCompilerManager& operator=(const
ExternalCompilerManager&) = delete;
    void startCompiler();
    void killCompiler();
    void restartCompiler();
    void compile(const QString& filepath);
    QString findPascalABCNET(const QString& exename);
    void scheduleRestart(int msecs);
private:
    explicit ExternalCompilerManager(QObject *parent = nullptr);
    ~ExternalCompilerManager();
    void error(const QString& msg);
    void sendMessage(const std::string& message);

    QProcess* compilerProcess;
    QTimer timer;
    zmq::context_t context;
    zmq::socket_t requester;
};
#endif // EXTERNALCOMPILERMANAGER_H
```

externalcompilermanager.cpp

```
#include "externalcompilermanager.h"
#include <QDebug>
#include <QApplication>
#include <QMessageBox>
#include <QRegularExpression>
#include <QStandardPaths>
#include "mainwindow.h"
#include "settings.h"
```

```
ExternalCompilerManager& ExternalCompilerManager::instance()
{
    static ExternalCompilerManager instance(nullptr);
    return instance;
}
ExternalCompilerManager::ExternalCompilerManager(QObject*
parent)
    : QObject(parent),
    context(1),
    requester(context, zmq::socket_type::req)
{
    compilerProcess = new QProcess(this);
    requester.connect("tcp://127.0.0.1:5555");
}
ExternalCompilerManager::~ExternalCompilerManager()
{
    compilerProcess->disconnect();
    compilerProcess->kill();
    compilerProcess->waitForFinished();
    requester.close();
    context.close();
}
QString ExternalCompilerManager::findPascalABCNET(const QString&
exename)
{
    QStringList possiblePaths = {
QCoreApplication::applicationDirPath() + "/../../" +
QString("%1/share/%2/PascalABCNETLinux").arg(PREFIX).arg(APP_NAM
E), QCoreApplication::applicationDirPath() +
"/../../../PascalABCNETLinux"
    };
    for (const QString& path : possiblePaths) {
        QFile file(path + "/" + exename);
        if (file.exists()) {
            return file.fileName();
        }
    }
    return QString();
}
void ExternalCompilerManager::startCompiler()
{
#ifdef Q_OS_WINDOWS
    QString path_to_pas = "D:\\Sci\\pascalabcnet-
zmq\\bin\\pabcnetc.exe";
    compilerProcess->setProgram(path_to_pas);
    compilerProcess-
>setProcessChannelMode(QProcess::SeparateChannels);
    compilerProcess->setArguments(QStringList() << "/noconsole"
<< "commandmode");
#else
```

```
    QString path_to_mono = "mono";
    QString path_to_pas = findPascalABCNET("pabcnetc.exe");

    compilerProcess->setProgram(path_to_mono);
    compilerProcess-
>setProcessChannelMode(QProcess::SeparateChannels);
    compilerProcess->setArguments(QStringList()
                                  << path_to_pas
                                  << "/noconsole"
                                  << "commandmode");
#endif
    compilerProcess->start(QProcess::ReadWrite);
    if (!compilerProcess->waitForStarted()) {
        qDebug() << "Failed to start process!";
    } else {
        qDebug() << "PABCNETC Process started successfully.";
    }
}
void ExternalCompilerManager::killCompiler(){
    compilerProcess->kill();
}
void ExternalCompilerManager::restartCompiler(){
    killCompiler();
    compilerProcess->waitForFinished();
    startCompiler();
}
void ExternalCompilerManager::sendMessage(const std::string&
message){
    try {
        zmq::message_t msg(message.size());
        memcpy(msg.data(), message.c_str(), message.size());
        requester.send(msg, zmq::send_flags::none);

        zmq::pollitem_t items[] =
{{static_cast<void*>(requester), 0, ZMQ_POLLIN, 0}};
        zmq::poll(items, 1, 15000);

        if (items[0].revents & ZMQ_POLLIN) {
            zmq::message_t reply;
            if (requester.recv(reply)) {
                QString replyMessage = QString::fromStdString(

std::string(static_cast<char*>(reply.data()), reply.size()));

                pMainWindow->logToolsOutput(replyMessage);
                if (replyMessage.startsWith("100")) {
                    pMainWindow->logToolsOutput("COMPILED
SUCCESSFULLY!");
                } else {
                    error(replyMessage);
                }
```

```
            } else {
                QMessageBox::warning(pMainWindow, "Error",
                                     "Failed to receive
response.");
            }
        } else {
            QMessageBox::warning(pMainWindow, "Error",
                                 "No response received within
the timeout period.");
        }
    } catch (const std::exception& e) {
        QMessageBox::critical(pMainWindow, "Error",
                              QString("Communication
error: %1").arg(e.what()));
    }
}
void ExternalCompilerManager::error(const QString& msg)
{
    QRegularExpression
regex(R"(\[0\]\[(\d+),(\d+)\]\s+(.:.*?:)\s+(.*?)(?=\s*\[\d+\]|$)
)");
    QRegularExpressionMatchIterator i = regex.globalMatch(msg);
    while (i.hasNext()) {
        QRegularExpressionMatch match = i.next();
        if (match.hasMatch()) {
            PCompileIssue issue =
std::make_shared<CompileIssue>();
            issue->line = match.captured(1).toInt();
            issue->column = match.captured(2).toInt();
            issue->filename = match.captured(3);
            issue->description = match.captured(4);
            issue->type = CompileIssueType::Error;
            pMainWindow->onCompileIssue(issue);
        }
    }
}
void ExternalCompilerManager::compile(const QString& filepath){
    std::string message = "215#5#" + filepath.toStdString();
    sendMessage(message);
    sendMessage("210");
    pMainWindow->onCompileFinished(filepath, true);
}
void ExternalCompilerManager::scheduleRestart(int msecs) {
    timer.setInterval(msecs);
    connect(&timer, &QTimer::timeout, this,
&ExternalCompilerManager::restartCompiler);
    timer.start();
}
```

Приложение 2. Исходный код функции hideUIElements

```
void MainWindow::hideUIElements() {
    // remove file explorer dock
    ui->dockExplorer->setDisabled(true);
    ui->dockExplorer->setVisible(false);

    // remove debug buttons near compile
    ui->toolbarDebug->setDisabled(true);
    ui->toolbarDebug->setVisible(false);

    // remove debug and generate assembly from execute menu
    for (auto& act: ui->toolbarDebug->actions()) {
        ui->menuExecute->removeAction(act);
    }
    ui->menuExecute->removeAction(ui->actionGenerate_Assembly);
    ui->menuExecute->removeAction(ui->actionView_CPU_Window);
    ui->menuExecute->removeAction(ui->actionAdd_Watch);

    // remove "very useful" C/C++/raylib/assembly references
help menus
    for (auto& act: ui->menuHelp->actions()) {
        if (act != ui->actionAbout) {
            ui->menuHelp->removeAction(act);
        }
    }

    // remove refactor and project menus
    ui->menubar->removeAction(ui->menuRefactor->menuAction());
    ui->menubar->removeAction(ui->menuProject->menuAction());
    ui->toolbarCode->removeAction(ui->actionReformat_Code);

    // remove tabs in the bottom
    ui->tabMessages->removeTab(ui->tabMessages->indexOf(ui-
>tabDebug));
    ui->tabMessages->removeTab(ui->tabMessages->indexOf(ui-
>tabTODO));
    ui->tabMessages->removeTab(ui->tabMessages->indexOf(ui-
>tabBookmark));
    ui->tabMessages->removeTab(ui->tabMessages->indexOf(ui-
>tabProblem));

    // set appropriate settings so tabs in the bottom wont show
ever
    pSettings->ui().setShowProblem(false);
    pSettings->ui().setShowBookmark(false);
    pSettings->ui().setShowDebug(false);
    pSettings->ui().setShowTODO(false);}
```

Приложение 3. Исходный код класса IntelliSenseManager

intellisensemanager.h

```
#ifndef INTELLISENSEMANAGER_H
#define INTELLISENSEMANAGER_H
#include <QObject>
#include <QProcess>
#include <QUrl>
#include <QHash>

#include <zmq.hpp>
#include "qsynedit/types.h"
#include "editor.h"
class IntelliSenseManager : public QObject
{
    Q_OBJECT
public:
    static IntelliSenseManager& instance();
    IntelliSenseManager(const IntelliSenseManager&) = delete;
    IntelliSenseManager& operator=(const IntelliSenseManager&) =
delete;
    void startIntelli();
    void killIntelli();
    void restartIntelli();
    void connectEvents();
    void didOpen(const QString& filename, Editor* editor);
    void didClose(const QString& filename);
    void didChange(const QString& filename, const QString&
fulltext);
    void didSave(const QString& filename);
    QStringList callIntelli(const QSynedit::BufferCoord& pos,
                            const QString& filename,
                            const QString& request_type);

private:
    explicit IntelliSenseManager(QObject* parent = nullptr);
    void initializeLSP(const QString& filename);
    QString sendMessage(const std::string& message);
    QHash<QString, int> documentVersions;
    int requestId = 0;
    QProcess* intelliProcess = nullptr;
    zmq::context_t context;
    zmq::socket_t requester;
};
#endif // INTELLISENSEMANAGER_H
```

intellisensemanager.cpp

```
#include "intellisensemanager.h"
#include <QJsonDocument>
#include <QJsonObject>
#include <QCoreApplication>
#include <QJsonArray>
#include <zmq.hpp>
#include "qsynedit/qsynedit.h"
#include "editor.h"
#include "compiler/externalcompilermanager.h"
IntelliSenseManager::IntelliSenseManager(QObject *parent)
    : QObject{parent}, requestId(0), context(1),
    requester(context, zmq::socket_type::req)
{
    intelliProcess = new QProcess(this);
    requester.connect("tcp://127.0.0.1:5557");
}
IntelliSenseManager& IntelliSenseManager::instance()
{
    static IntelliSenseManager instance(nullptr);
    return instance;
}
void IntelliSenseManager::connectEvents() {
    initializeLSP("");
}
void IntelliSenseManager::initializeLSP(const QString& filename)
{
    QJsonObject lspParams;
    lspParams["processId"] =
static_cast<int>(QCoreApplication::applicationPid());

    QUrl rootUrl =
QUrl::fromLocalFile(QFileInfo(filename).canonicalFilePath());
    lspParams["rootUri"] = rootUrl.toString();
    QJsonObject synchronization{
        {"didSave", true},
        {"willSave", true},
        {"willSaveWaitUntil", false}
    };
    QJsonObject completionItem{
        {"snippetSupport", true}
    };
    QJsonObject textDocument{
        {"synchronization", synchronization},
        {"completion", QJsonObject{{"completionItem",
completionItem}}}};
    QJsonObject capabilities{
        {"workspace", QJsonObject{{"applyEdit", true}}},
        {"textDocument", textDocument}};

    lspParams["capabilities"] = capabilities;
```

```
    QJsonObject resultJSON{
        {"jsonrpc", "2.0"},
        {"id", requestId++},
        {"method", "initialize"},
        {"params", lspParams}
    };
sendMessage(QJsonDocument(resultJSON).toJson(QJsonDocument::Comp
act).toStdString());
}
QStringList IntelliSenseManager::callIntelli(const
QSynedit::BufferCoord& pos, const QString& filename, const
QString& request_type) {
    QJsonObject lspPosition{
        {"line", pos.line - 1},
        {"character", pos.ch - 1}};
    QJsonObject textDocument{
        {"uri",
QUrl::fromLocalFile(QFileInfo(filename).canonicalFilePath()).toS
tring()}};
    QJsonObject resultJSON{
        {"jsonrpc", "2.0"},
        {"id", requestId++},
        {"method", "textDocument/" + request_type},
        {"params", QJsonObject{
                        {"textDocument", textDocument},
                        {"position", lspPosition}
                    }}};
    QString data =
QJsonDocument(resultJSON).toJson(QJsonDocument::Compact);
    QString result = sendMessage(data.toStdString());
    if (request_type == "hover") {
        return QStringList() <<
QJsonDocument::fromJson(result.toUtf8()).object()["result"].toOb
ject()["contents"].toObject()["value"].toString();
    } else if (request_type == "completion") {
        QStringList res;
        QJsonArray resultArray =
QJsonDocument::fromJson(result.toUtf8()).object()["result"].toAr
ray();
        for (const QJsonValue& item : resultArray) {
            if (item.isObject() &&
item.toObject().contains("label")) {
                res << item.toObject()["label"].toString();
            }
        }
        return res;
    }
    return QStringList() << "Nothing really happened.";
}
void IntelliSenseManager::didChange(const QString& filename,
const QString& fulltext) {
```

```
    if (!documentVersions.contains(filename)) {
        documentVersions[filename] = 0;
    }
    QJsonObject message{
        {"jsonrpc", "2.0"},
        {"method", "textDocument/didChange"},
        {"params", QJsonObject{
                        {"textDocument", QJsonObject{
                                                {"uri",
QUrl::fromLocalFile(QFileInfo(filename).canonicalFilePath()).toS
tring()},
                                                {"version",
++documentVersions[filename]}
                                            }},
                        {"contentChanges", QJsonArray{

QJsonObject{{"text", fulltext}}
                                             }}
                    }}
    };
sendMessage(QJsonDocument(message).toJson(QJsonDocument::Compact
).toStdString());}
void IntelliSenseManager::startIntelli() {
#ifdef Q_OS_WINDOWS
    QString path_to_pas = QCoreApplication::applicationDirPath()
+ "\\..\\PascalABCNETLinux\\LSPProxy\\TestIntelli.exe";
    intelliProcess->setProgram(path_to_pas);
    intelliProcess-
>setProcessChannelMode(QProcess::SeparateChannels);
    intelliProcess->setArguments(QStringList() << "/noconsole"
<< "commandmode");
#else
    QString path_to_mono = "mono";
    intelliProcess->setProgram(path_to_mono);
    QString path_to_pas =
ExternalCompilerManager::instance().findPascalABCNET("LSPProxy/T
estIntelli.exe");
    intelliProcess->setArguments(QStringList() << path_to_pas);
#endif

    intelliProcess->start();
    QObject::connect(QCoreApplication::instance(),
&QCoreApplication::aboutToQuit, [this]() {
        if (intelliProcess->state() == QProcess::Running) {
            intelliProcess->terminate();
        }});
    if (!intelliProcess->waitForStarted()) {
        qDebug() << "Failed to start process!";
    } else {
        qDebug() << "INTELLISENSE Process started
successfully.";
```

```
    }
}
void IntelliSenseManager::didOpen(const QString& filename,
Editor* editor) {
    QUrl fileUrl =
QUrl::fromLocalFile(QFileInfo(filename).canonicalFilePath());
    editor-
>setFilename(QFileInfo(filename).canonicalFilePath());

    QJsonObject textDocument{
        {"uri", fileUrl.toString(QUrl::FullyEncoded)},
        {"languageId", "pas"},
        {"version", 0},
        {"text", editor->text()}
    };

    QJsonObject resultJSON{
        {"jsonrpc", "2.0"},
        {"method", "textDocument/didOpen"},
        {"params", QJsonObject{{"textDocument", textDocument}}}
    };
sendMessage(QJsonDocument(resultJSON).toJson(QJsonDocument::Comp
act).toStdString());

    connect(editor, &QSynedit::QSynEdit::changeForIntelli, this,
[this, filename](const QString& text) {
        didChange(filename, text);
    });
}
QString IntelliSenseManager::sendMessage(const std::string&
message) {
    try {
        zmq::message_t msg(message.size());
        memcpy(msg.data(), message.c_str(), message.size());
        requester.send(msg, zmq::send_flags::none);
        zmq::pollitem_t items[] =
{{static_cast<void*>(requester), 0, ZMQ_POLLIN, 0}};
        zmq::poll(items, 1, 10000);

        if (items[0].revents & ZMQ_POLLIN) {
            zmq::message_t reply;
            if (requester.recv(reply)) {
                std::string
replyMessage(static_cast<char*>(reply.data()), reply.size());
                return QString::fromStdString(replyMessage);
            }
        }
    } catch (...) {
        qDebug() << "Unknown error occured!";
    }
    return "";}
```

Приложение 4. Исходный код адаптера для взаимодействия по протоколу LSP

```
using System;
using System.IO;
using System.IO.Pipes;
using System.Text;
using System.Threading.Tasks;
using NetMQ;
using System.Linq;
using System.Diagnostics;
using System.Runtime.InteropServices;

class NamedPipeLanguageClient
{
    static void Main(string[] args)
    {
        var server = new Process();
        server.StartInfo.UseShellExecute = false;
        server.StartInfo.CreateNoWindow = true;
        if (Environment.OSVersion.Platform == PlatformID.Unix)
        {
            server.StartInfo.FileName = "mono";
            server.StartInfo.Arguments =
AppDomain.CurrentDomain.BaseDirectory +
"../IntelliSense/LanguageServerEngine.exe";
        }
        else
        {
            server.StartInfo.FileName =
AppDomain.CurrentDomain.BaseDirectory +
"..\\IntelliSense\\LanguageServerEngine.exe";
        }
        server.StartInfo.EnvironmentVariables.Add("parentId",
Process.GetCurrentProcess().Id.ToString());

        server.Start();
        RunClient().Wait();
    }
    static async Task RunClient()
    {
        var pipeName = "language-pipe";
        var socket = new NetMQ.Sockets.ResponseSocket();
        socket.Bind("tcp://127.0.0.1:5557");

        using (var pipeClient = new NamedPipeClientStream(".",
pipeName, PipeDirection.InOut, PipeOptions.Asynchronous))
        {
            Console.WriteLine("Connecting to server...");
```

```
            await pipeClient.ConnectAsync();
            Console.WriteLine("Connected!");
            using (var writer = new StreamWriter(pipeClient, new
UTF8Encoding(false)) { AutoFlush = true })
            using (var reader = new StreamReader(pipeClient, new
UTF8Encoding(false)))
            {
                while (true)
                {
                    var json = socket.ReceiveFrameString();
                    Console.WriteLine(json);
                    // didOpen and didChange do not return
answer, so this branching is neccesary
                    if (json.Contains("didOpen") ||
json.Contains("didChange"))
                    {
                        socket.SendFrame("code GREEN!");
                        await SendMessage(writer, json);
                        continue;
                    }
                    await SendMessage(writer, json);
                    Console.WriteLine("Sent initialize
request.");
                    var response = await
ReadMessage(reader.BaseStream);
                    Console.WriteLine("Received:\n" + response);

                    socket.SendFrame(response);
                }
            }
        }
    }
    static async Task SendMessage(StreamWriter writer, string
json)
    {
        var contentBytes = Encoding.UTF8.GetBytes(json);
        var header = $"Content-Length:
{contentBytes.Length}\r\n\r\n";
        await writer.WriteAsync(header + json);
    }
    static async Task<string> ReadMessage(Stream stream)
    {
        var headerBuilder = new MemoryStream();
        byte[] buffer = new byte[1];

        // read until \r\n\r\n
        int state = 0; // 0 = nothing, 1 = \r, 2 = \r\n, 3 =
\r\n\r, 4 = \r\n\r\n
        while (state < 4)
        {
            int read = await stream.ReadAsync(buffer, 0, 1);
```

```
            if (read == 0) break;

            headerBuilder.WriteByte(buffer[0]);

            switch (state)
            {
                case 0: state = buffer[0] == '\r' ? 1 : 0;
break;
                case 1: state = buffer[0] == '\n' ? 2 : 0;
break;
                case 2: state = buffer[0] == '\r' ? 3 : 0;
break;
                case 3: state = buffer[0] == '\n' ? 4 : 0;
break;
            }
        }
        string headers =
Encoding.ASCII.GetString(headerBuilder.ToArray());
        int contentLength = 0;
        foreach (var line in headers.Split(new[] { "\r\n" },
StringSplitOptions.RemoveEmptyEntries))
        {
            if (line.StartsWith("Content-Length:"))
            {
                contentLength =
int.Parse(line.Substring("Content-Length:".Length).Trim());
            }
        }
        if (contentLength == 0) return null;
        byte[] contentBuffer = new byte[contentLength];
        int bytesRead = 0;
        while (bytesRead < contentLength)
        {
            int n = await stream.ReadAsync(contentBuffer,
bytesRead, contentLength - bytesRead);
            if (n == 0) break;
            bytesRead += n;
        }

        return Encoding.UTF8.GetString(contentBuffer, 0,
bytesRead);
    }

}
```