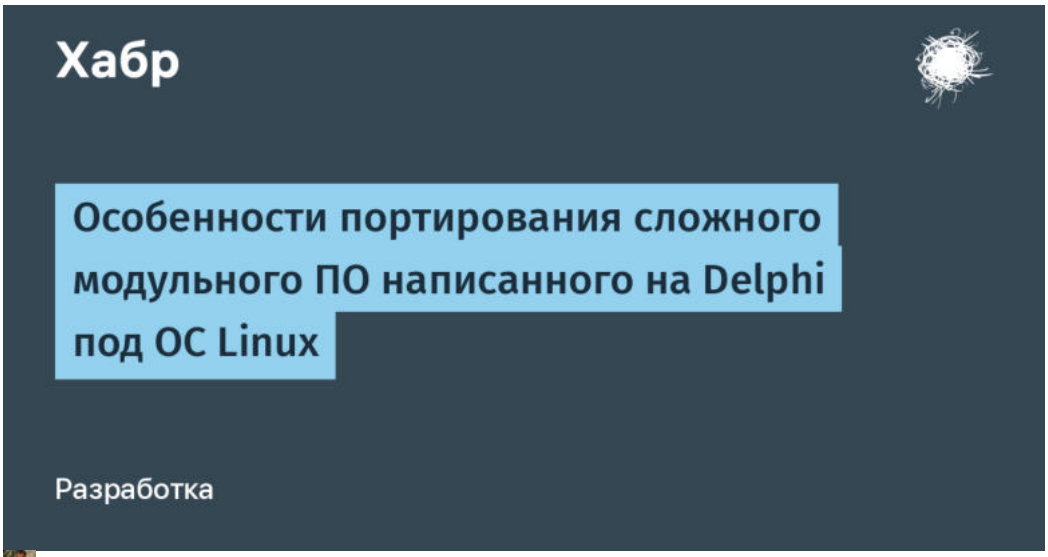
habr.com /ru/articles/534466/

Особенности портирования сложного модульного ПО написанного на Delphi под ОС Linux

Тимофеев Константин ⋮⋮ 22.12.2020



timofeevka 22 дек 2020 в 16:29

34 мин

38K
Из песочницы
Технотекст 2021

Данное повествование не предназначено для разработчиков, которые не знают что такое Delphi и не умеют на нём программировать. Просьба людей, не имеющих дела с данными средствами разработки, не комментировать статью и не травмировать и без того расшатанные нервы дедушек, упорно продолжающих поддерживать многолетние разработки, написанные с применением данных средств разработки. Предложения переписать всё с нуля на что-то более модное не приветствуются.

Введение

Меня зовут Тимофеев Константин, мне 40 лет и я являюсь ведущим программистом компании 3В Сервис в подразделении, занимающемся системами автоматизации динамических расчётов (САДР). Нашим основными программным продуктом является среда моделирования динамических систем SimInTech (в девичестве ПК Моделирование В Технических Устройствах – 4, сокращённо ПК «МВТУ»).

Исторически данный программный продукт начал развиваться на кафедре «Ядерные реакторы и энергетические установки» (Э7) МГТУ им. Баумана. Первые версии ПК «МВТУ» были разработаны на кафедре под руководством к.т.н. Козлова Олега Степановича. Программный комплекс предназначался для структурного моделирования динамики ядерных энергоустановок. Функционал ПК «МВТУ» заключался в составлении модели системы в виде блок-схемы с последующим автоматическим формированием системы дифференциальных уравнений динамики системы и их численном решении и выводе результатов расчёта на графики. В дальнейшем развитии этой темы к разработке в 2003 году подключился я и на базе ПК «МВТУ-3» мы начали создавать промышленную версию программного комплекса – ПК «МВТУ-4», получившую в дальнейшем название SimInTech. Разработка нового варианта программы потребовала очень серьёзной переработки архитектуры программы и позволила создать комплекс, позволяющий строить полномасштабные модели динамики ядерных энергоустановок, а также систем в других отраслях промышленности, и являющийся достойным конкурентом систем Simulink, AmeSim, VisSim.

В качестве среды разработки для ПК «МВТУ-3» мы использовали Delphi, начиная с версий 3 (и заканчивая версией 5). Выбор данного средства разработки был обусловлен простотой создания графического интерфейса программы и большим количеством компонентов, скоростью разработки по сравнению с разработкой системы на других средствах того времени. Применение этой среды позволило создать достаточно большой по объёму пакет в очень сжатые сроки командой включавшей в себя трёх разработчиков (из которых по факту основным был один).

При разработке следующей версии программы в 2003 году, учитывая достаточно серьёзный объём ранее разработанного кода я также использовал в качестве средства разработки Delphi версии 6. В настоящее время мы используем как основное средство разработки под ОС Windows Delphi 10.3.3. Также надо понимать что хоть основная часть кода написана на Delphi, также в составе комплекса присутствуют модули и на C\C++ , а также на FORTRAN. При этом сам наш программный продукт с тех пор значительно усложнился, оброс большим количеством библиотек и модулей и прошёл значительную эволюцию под влиянием его применения на предприятиях атомной отрасли.

Глава 1: Что мы имеем – архитектура и особенности программного комплекса

Главной составной частью ПК SimInTech является графическая оболочка. Графическая оболочка предназначена для того, чтобы пользователи могли визуально составлять блок-схемы технологических систем моделируемых объектов. В её основе лежит система векторной графики собственной разработки, которая включает в себя набор объектов – базовых графических примитивов как то: линии, полигоны, эллипсы, текстовые вставки, блоки, группы примитивов, линии связи, системные контролы (чекбокс, комбо-бокс, однострочный текстовый редактор). Поверх объектов-примитивов лежат групповые объекты – контейнеры графических примитивов (страницы). Контейнер – это то, что обеспечивает отрисовку схемы целиком и её редактирование – т.е. передачу примитивам событий клавиатуры и мыши). Также каждый контейнер включает в себя и объект – скриптовую машину для обеспечения динамизации изображения (изменения цвета положения, текста в зависимости от расчётных параметров схемы). Встроенная скриптовая машина позволяет делать как техническую анимацию, так и проводить расчёты, по сути она представляет собой встроенный язык программирования общего назначения.

Графическая оболочка предоставляет пользователю следующие сервисы:

- составление схемы из блоков, создание видеокадров из примитивов.

- работу с библиотеками блоков.

- вывод параметров расчёта на графики, в текстовые таблицы, в файл, в OPC, в shared memory.

- управление расчётом.

- синхронизация различных моделей и совместный расчёт.

- загрузка плагинов для блоков\схем\слоёв.

- работа со звуком.

- редактирование текстов скриптов с подсветкой кода.

- внешнее управление программой через встроенный COM-сервер (это использовалось для автоматизации расчётов в связке с другими программами).

Оболочка является наиболее сложной частью системы в точки зрения пользовательского интерфейса и объёма кода. Оболочка была написана на Delphi с применением достаточно разнообразных компонентов, из которых кроме стандартных контролов (кнопок полей ввода, комбо и чекбоксов) необходимо выделить следующие:

- для вывода рассчитанных значений на графики – Tee Chart Pro.

- для редактирования текстов с подсветкой – первоначально SynEdit, впоследствии TBCEditor.

- для работы со звуком – DSPack.

- для вывода деревьев и построения редакторов списков сигналов\объектов\свойств блоков\параметров расчёта\дерева библиотеки\дерева структуры проекта – VirtualTreeView.

- для работы с базами данных для отдельных модулей системы – zeosdbo.

- для трехмерной технической анимации – GLScenes.

- для разного – JEDI VCL + JCL -отдельные контролы, некоторые библиотеки например расчёт хешей и контрольных сумм.

Кроме того, в программе как DLL подключается несколько библиотек на C и FORTRAN, например модуль шифрования RSA и расчёта MD5 хеша. Для рендеринга графики схем в данный момент в Windows-версии программы используется Direct2D, который позволяет сделать достаточно плавную и производительную техническую анимацию (в том числе и с большими по размеру растровыми текстурами).

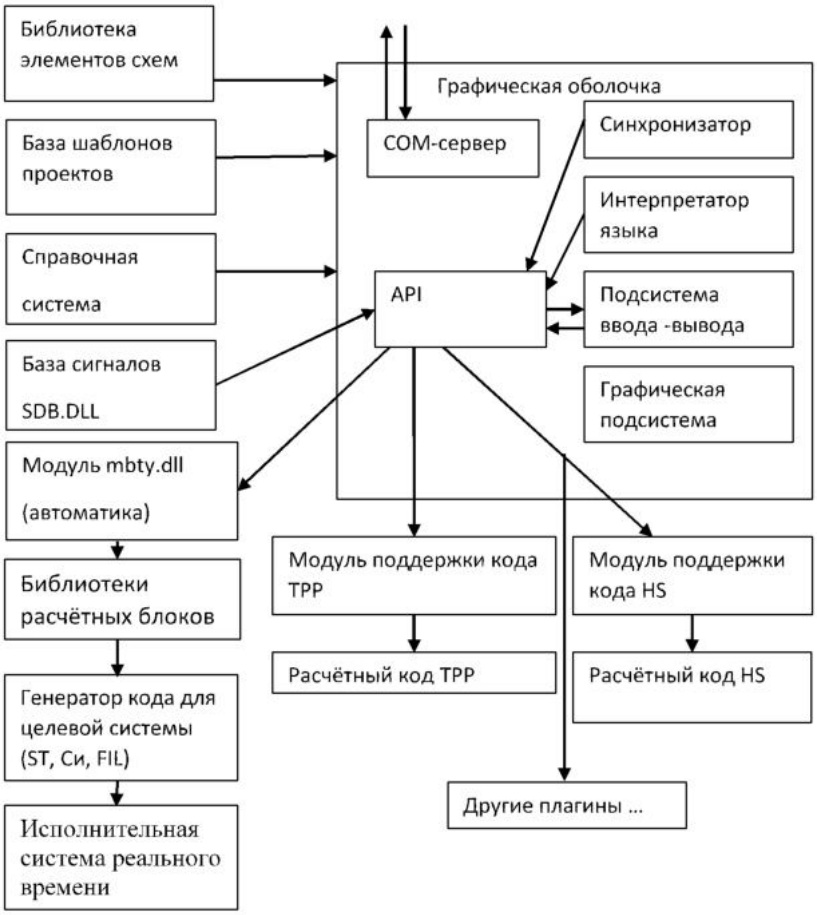
К графической оболочке могут подключаться модули – плагины, которые обеспечивают расширение функций программы на разных уровнях. Например для каждого блока на схеме мы можем указать какой плагин его обслуживает.

Система плагинов построена на основе наследования их от виртуальных базовых классов. То есть есть общий модуль где описываются базовые классы. В оболочке от него ничего не наследуется, а в подгружаемых DLL от базовых классов наследуются те или иные блоки расширения, в которых мы делаем override базовым методам. Для создания плагинов в каждой из библиотек реализована фабрика классов, которая по заданному имени внутри модуля создаёт заданный объект расширения. Такой подход позволяет сделать очень гибкую структуру плагинов и иметь там не только методы (как в классических интерфейсах) но и объекты\ссылки на системные структуры\и так далее. Минусом является то, что ограничивается возможность использования других компиляторов – т.е. при реализации плагина должен использоваться тот же компилятор что и оболочки (ну или совместимый по генерируемому коду). Кроме того, внутри плагинов могут находится и окошки – т.е. вспомогательный интерфейс.

Соответственно, для полноценной реализации расширений графической оболочки в программе строго необходим общий менеджер памяти как для плагинов, так и для библиотек, в качестве какового в данный момент используется fastmm5, прописанный первым модулем как в коде оболочки так и в коде плагинов. Это позволило обеспечить прозрачную для всех модулей работу со строками и выделение памяти для массивов и объектов, критически необходимо это в первую очередь для работы со строками, когда строка возвращается как результат функции из плагина например.

Общий объем исходного кода только графической оболочки составляет приблизительно 215073 строк кода (это чистый код самого проекта, без компонентов и без файлов конфигурации форм dfm). С плагинами объем исходного кода на Delphi в проекте составляет 2858375 строк кода, что уже является весьма серьёзным числом.

Общая архитектура комплекса приведена на рисунке ниже.



Глава 2: Импортозамещение или заказчики захотели извращений

В общем жили мы жили, не тужили и баблос рубили спокойно делая программу под ОС Windows, и тут раз и вторая смена (с). Первые звонки с тем, что недурно бы поддержать и другие операционные системы начались не с отечественного заказчика как ни странно, а с немецкого института GRS для которого мы сделали систему проектирования моделей для их кода на нашей платформе. Немцы предложили нам ~~BDSM~~ – а хорошо бы нам для отчёта перед нашим посконным немецким начальством поддержать отечественную (ну в смысле исконно немецкую) операционную систему Suse Linux, а то непатриотично мол как-то, да и денег платить за неё заставляют. Прельстившись шуршанием не очень большого количества евровалюты, мы начали думать, как таки заставить программу на этом поделии светлого и благородного сумрачного немецко-финского гения.

Первое что пришло при этом в голову – завести программу под Wine и таки действительно СТАРАЯ версия нашего софта под ним пошла, но только 32 битная и без Direct2D рендера, ну и соответственно без красивостей в виде прозрачностей и быстрых градиентов. Ну и кроме того, это было весьма скромно по производительности, особенно в многопоточном режиме. Возможно, после утечек исходного кода от майкрософта этот слой API в линуксе будет доведён до работающего состояния пригодного для полноценной работы нашего софта, но пока это не всегда так. Но захотелось иметь что-то чуть более приличное и по скорости и максимально реализующее функционал Windows-версии.

Java мы отмели сразу, ибо проблемы со стыковкой к ней библиотек на FORTRAN и откровенно отстойная скорость работы с более-менее сложными скриптами и графикой показались недостаточным аргументом для переписывания всего объёма кода на неё. Кроме того, она дико многословная по сравнению с Delphi да и нам нужен был именно нативный код, в первую очередь из-за необходимости стыковки с некоторым количеством стороннего расчётного софта (как немецкого так и нашего). Поэтому Java пошла в топку (на моей прошлой работе часть системы на ней всё-таки переписали, из опыта эксплуатации получилось очень сильно хуже исходной версии), наступать на старые грабли не очень хотелось. Туда же пошёл и C#, из-за плохой поддержки разработки на нём GUI именно в Linux и ненативного кода с проблемами стыковки со сторонними расчётными библиотеками.

Из вариантов сделать кроссплатформенный GUI остались 2:

1. Qt и C++ - это замечательный путь, но переписывание всего кода системы не впечатлило, кроме того под основную систему это на самом деле скорее всего означало было ухудшение отзывчивости и качества пользовательского интерфейса – VCL контролы использует родные системные всё таки. Далее встал вопрос наличия необходимой для формирования достаточно сложного пользовательского интерфейса базы компонентов, которые были перечислены выше. Посмотрев некоторые программы, выполненные при помощи Qt (qucs например как достаточно близкий по функциям) я осознал, что этот путь всё таки будет погибелен скорее всего.

2. Free Pascal + Lazarus. Самый большой плюс этой среды разработки и компилятора был в практически полной (но не совсем) переносимости текста исходной программы, что позволяло дословно сохранить логику работы. Но были большие сомнения в том, что те компоненты и особенности которые были в коде программы корректно переварятся этим компилятором.

Итак в силу ограничений в человеческих ресурсах было решено попробовать максимально использовать имеющуюся кодовую базу, оставив её на Delphi и дополнив код IFDEF-ами для сборки программы под другую ОС в среде разработке Lazarus. Немалую роль также сыграло то, что это позволяло не прерывать работу над основной версией SimInTech для Windows и параллельно вносить изменения сразу в 2 варианта сборки. Ибо срок работы был небольшой, а перенести надо было много и ещё так чтобы оно заработало.

Анализ встроенных в Lazarus штатно компонентов показал, что маловато будет, и было решено поискать что-то, в чём этого добра насыпано поболее, особенно в части отрисовки 2D-графики. Сначала был попробован рендер AggPas в штатном Lazarus-е, но он показался сильно медленным и не устраивающим по некоторым особенностям совместимости кода с Direct2D библиотекой. Поэтому было принято искать решение поинтереснее. Поиск по интернету вывел меня на нестандартную сборку Lazarus-а под названием CodeTyphon Studio и первые эксперименты с этой средой привели к выводу что вот это в принципе то, что позволит собирать программу на Linux с максимальным использованием существующего кода. Свою роль сыграло то что:

1 – там много компонентов по умолчанию, функционально и по использованию совместимых с теми, которые я использовал в Delphi-версии. То есть требовалась фактически минимальная переработка кода. Из всего что мне было нужно отсутствовали BCEditor (но был SynEdit, который у меня использовался в редакторе скриптов в более старой версии), DSPack (но были аналоги, и на самом деле наличие звука пока было не критично), TeeChart Pro (но штатно был аналог TAChart, а вот построение графиков как раз было очень критично).

2 – в наличии оказался неплохой компонент для рендеринга графики – ORCA 2D, структура которого была очень похожа на вызовы Direct2D и который был принципиально пригоден к использованию для отрисовки сложных изображений моделей и видеокадров.

Глава 3: Начинаем портировать

Первое с чего начался процесс портирования – это проверка размера катастрофы, т.к. среда разработки и компилятор, на который это всё должно было натянуться должен был позволять мне реализовать несколько базовых вещей, без которых программу пришлось бы крайне нехорошим образом менять:

1 – необходимо было иметь общий менеджер памяти для головного модуля и плагинов, выполненных в виде динамических shared object – библиотек.

2 – должна была быть возможность загрузки сторонних динамических библиотек на C и FORTRAN.

3 – было необходимо, чтобы некоторые формы находились в динамически загружаемых библиотеках, т.к. за счёт них выполнялось расширение интерфейса.

4 – должны быть аналоги большинства компонентов с идентичным или похожим пользовательским и программным интерфейсом.

5 – файлы форм dfm должны без лишних проблем конвертироваться (дальше как показала практика целесообразно иметь 2 раздельных набора файлов описания форм для Windows и Linux из за некоторой разницы в дизайне системных виджетов.

Самые большие сомнения на этот счёт были касательно пунктов 1 и 3, потому что используемая в Lazarus библиотека LCL хоть и является практически точным клоном VCL, но различается по ряду особенностей.

Первым делом была развернута виртуальная машина под VirtualBox с доступом к нашему репозиторию, где мы ведём разработку. В качестве операционной системы на тот момент была выбрана Open Suse, немножко позднее была сделана виртуальная машина на Alt Linux 8.3 и потом Alt Linux 9. Далее с сайта https://pilotlogic.com/ был выкачан архив с дистрибутивом среды разработки. Развертывание среды разработки было произведено по инструкции к ней – распаковать архив и выполнить su ./install.sh , но в процессе развертывания на Alt Linux встала проблема, что этой ОС в установочном скрипте в списке не было – пришлось добавить самому и написать письмо разработчику чтобы включил в код, сейчас эта разновидность линукса поддерживается. Если что не так – то в дистрибутиве можно дополнить файлик CodeTyphonIns\installbin\ScriptsLin\ln_All_Functions.sh вписав в него особенности установки.

Для Alt Linux туда была добавлен текст, рядом с тем местом, где определяется установка для Ubuntu:

```
#------------ 100 Alt (apt-get compitible)----------
 elif [ -f /etc/altlinux-release ] ; then
   vOSVerNum=100
   vOSDistribution="Alt Linux (apt-get compatible)"
   vMultiArchDirPlan=200
```

В AstraLinux среда разработки ставится без лишних телодвижений на данный момент (с sudo). Бинарные файлы скомпилированные под AltLinux 9 запускаются и под Astra Linux Orel.

У Astra Linux на данный момент есть одна проблема, не связанная непосредственно с данной разработкой – штатно на нём стоит в настройках стабильный репозиторий в котором git версии 2.11, а для последнего SmartGIT-а нужно не ниже 2.16. Проблема решается путём переключения на экспериментальный репозиторий и обновления пакета git из него.

В итоге среда разработки была успешно установлена на тестовую виртуальную машину с Linux.

Дальше настала череда тестирования указанного выше базового функционала, который был необходим для успешной компиляции всего комплекса. Для начала было сделано минимальное приложение, загружающее DLL внутри которой создавалась форма и которое в свою очередь передавало в вызывающую программу строчку.

Анализ кода, поставляемого с компилятором Free Pascal, показал, что в составе RTL-библиотек имеется модуль cmem, который позволяет использовать менеджер памяти из libc. Он также как и для Delphi ставится первым модулем в dpr-файле и подключает внешний менеджер памяти (в Delphi для этого есть SimpleShareMem ну или fastmm4 \ fastmm5). На этом пункте можно было с уверенностью поставить галочку. Но при дальнейшем тестировании было выяснено что производительность этого менеджера памяти достаточно отстойная, поэтому поиски были продолжены и в результате был найден неплохой менеджер памяти fpcx64mm из пакета Synopse mORMot framework 2. Но он был не распределённый, поэтому для того, чтобы его использовать он был подключен через небольшой промежуточный модуль simmm.pas следующего содержания:

```
unitsimmm;
//  Прокладка для использования общего менеджера памяти
{$IFNDEF DCAD}
{$DEFINE IS_DLL_UNIT}
{$ENDIF}


{$IFDEF FPC}
{$MODE Delphi}{$H+}
{$ENDIF}


interface

{$IFDEF UNIX}
```

```
{$IFDEF IS_DLL_UNIT}
usescthreads, dl;
{$ELSE}
usescthreads, fpcx64mm;
{$ENDIF}


{$ELSE}
usesFastMM5;
{$ENDIF}


implementation

{$IFDEF UNIX}
{$IFDEF IS_DLL_UNIT}
var
NewMM,
OldMM:      TMemoryManager;
MainHandle: Pointer;
GetCommonMemoryManager: procedure(varaMemMgr: TMemoryManager);
{$ENDIF}
{$ENDIF}


initialization


{$IFDEF UNIX}
{$IFDEF IS_DLL_UNIT}


MainHandle:=dlopen(nil, RTLD_LAZY);
GetCommonMemoryManager:=dlsym(MainHandle,'GetMemoryManager');
GetCommonMemoryManager(NewMM);
GetMemoryManager(OldMM);
SetMemoryManager(NewMM);


{$ENDIF}
{$ELSE}


//Расшариваем менеджер памяти
ifIsLibrarythen
FastMM_AttemptToUseSharedMemoryManager
else
FastMM_ShareMemoryManager;
{$ENDIF}

                    {$IFDEF UNIX}
  {$IFDEF IS_DLL_UNIT}
finalization
    SetMemoryManager(OldMM);
  {$ENDIF}
  {$ENDIF}
end.
```

Как видно из данного кода ключ DCAD определяется только в головном модуле, при этом мы в нём определяем как экспортируемую функцию GetMemoryManager. А в dll\so мы получаем адрес этой функции путём вызова dlopen(nil, RTLD_LAZY) – если эта функция вызывается с нулевым первым аргументом, она возвращает хендл на главный исполняемый файл и дальше мы получаем оттуда адрес функции для получения центрального менеджера памяти и используем в плагинах его. Соответственно во всех плагинах первым модулем мы прописываем simmm для обеспечения общего выделения памяти.

Вторая проблема – это вызов форм, находящихся внутри dll\so модулей. В Delphi по факту эта проблем не возникала. Т.е. если мы делаем там TForm.Create, то форма создается нормально. В Lazarus на линуксе оказалось всё несколько интереснее, т.к. для того, чтобы форма создалась, необходимо в so-библиотеке прописать в списке модулей:

```
  …..
  Classes,
  {$IFDEF FPC}
  Interfaces,
  {$ENDIF}
  Forms,
```

и определить инициализацию Application в секции initialization и завершение в finalization:

```
{$IFDEF FPC}
initialization
  Application.Initialize;
finalization
  Application.Terminate;
end.
{$ENDIF}
```

Выполнив такие операции, мы в дальнейшем можем спокойно создавать формы внутри so.

Наше приложение является многопоточным, поэтому сразу возник также вопрос с тем как это будет поддерживаться в Lazarus. В результате чтения документации и экспериментов было выяснено, что на UNIX-системах чтобы внутри программы или so создавать поток, в списке модулей проекта должен быть прописан модуль cthreads:

```
  {$IFDEF UNIX}
  cthreads,
  {$ENDIF}
```

Таким образом все dpr-файлы были модифицированы типовым образом для того, чтобы в них заработал нужный нам функционал, в качестве примера привожу исходник dpr-файла для плагина расчёта систем управления:

```
{$IFDEF FPC}
{$MODE Delphi}//Это определение нужно, чтобы компилятор Free Pascal включил режим
{$ENDIF}//совместимости с Delphi.



librarymbtylib;
uses
simmm,
{$IFDEF UNIX}
cthreads, //сейчас этот модуль включен в simm.pas чтобы он был подключен для всех
библотек сразу
{$ENDIF}
Classes,
{$IFDEF FPC}
Interfaces,
{$ENDIF}
Forms,
MBTYToolsin'MBTYTools.pas',
MBTYObjtsin'MBTYObjts.pas',
uMBTYThreadin'uMBTYThread.pas',
SpecBlocksin'SpecBlocks.pas',
InfoUnitin'InfoUnit.pas'{MBTYInfoForm},
MBTYtranslatein'MBTYtranslate.pas',
uDebugBlockFormin'uDebugBlockForm.pas'{DebugBlockForm};



{$R *.res}

//Тут мы храним картинки, которые у нас пойдут во вспомогательные кнопку на тулбарах
{$R mbtybuttons.res}
```

```
//Эта функция возвращает адрес структуры DllInfo
functionGetEntry:Pointer;
begin
Result:=@DllInfo;
end;



exports
GetEntryname'GetEntry',         //Функция получения адреса структуры DllInfo
CreateObjectname'CreateObject'; //Функция создания объекта

        {$IFDEF FPC}
initialization
  Application.Initialize;
finalization
  Application.Terminate;
end.
{$ENDIF}
begin
end.
```

Таким образом 2 базовые для текущей архитектуры программного комплекса проблемы получили своё решение и на Linux. Дальше оставалось проанализировать насколько корректно будут передаваться в DLL строки и ссылки на объекты – с этим при подключении общего менеджера памяти всё оказалось в порядке. С загрузкой сишных и фотрановских .so также всё оказалось хорошо, но необходимо помнить, что intel fortran и gfortran по умолчанию по разному передают в функции аргументы, поэтому там где это было нужно, пришлось видоизменить декларацию вызова некоторых подпрограмм из сторонних DLL\SO. Как пример можно привести следующий фрагмент кода, подключающего процедуру на FORTRANe:

Вариант для Windows - INTEL FORTRAN, конвенция stdcall:

```
  TSbros = procedure(
    Nuzl,Nu,Ngran,Nmat,Nko,idmdt0,iadiab0: integer;       // +++ 31.03.2014
(iadiab0)
    var El,Iel,Uzel,Iuzel,
      NYZL,G,Nnas,Nzad,Nelu,
      Gran,Granu,Ngu,Ngut,Propm,Nelm,
      Zadv,Pump,Nzadp;
    Dtau,DtauG,Htau,TauE: TPPRealType;
……..
    );stdcall;
```

Вариант для Linux gfortran с конвенцией cdecl:

```
   //Версия для стыковки с версией для под GFortran
  TSbros = procedure(
    var Nuzl,Nu,Ngran,Nmat,Nko,idmdt0,iadiab0: integer;       // +++ 31.03.2014
(iadiab0)
    var El,Iel,Uzel,Iuzel,
      NYZL,G,Nnas,Nzad,Nelu,
      Gran,Granu,Ngu,Ngut,Propm,Nelm,
      Zadv,Pump,Nzadp;
    var Dtau,DtauG,Htau,TauE: TPPRealType;
…….
    );cdecl;
```

Из приведённого кода видно, что в Intel FORTRAN целочисленные параметры функции по умолчанию передаются в стек напрямую, а массивы и вещественные числа передаются через указатели, а в gfortran все параметры передаются через указатели. При этом по умолчанию конвенция вызовов в Intel FORRAN для Windows – stdcall, в gfortran, что для Windows что для Linux – cdecl.

Кроме этого также следует обратить внимание на именование экспортируемых функций при загрузке их из DLL: если в Intel Fortran для Windows имя экспортируемой функции остаётся неизменным и соответствует имени функции, то в **gfortran** добавляется суффикс **_** :

{$IFDEF Windows}**'wwww'**{$ELSE}**'wwww_'**{$ENDIF}

Некоторые части программы были написаны на языке Си и подключены как динамически загружаемые библиотеки. Компиляция этих библиотек производилась компилятором gcc. Пример скрипта компиляции so-файла одной из них приведён ниже:

gcc **-shared -fPIC** -o ../../bin/libcommonclibs.so -s nn.c prime.c r_keygen.c rc4c.c rsa.c md5c.c **-fpack-struct=1** -Wconversion

При компиляции модулей на Си следует обратить внимание на выравнивание структур, для того, чтобы таковое совпадало с кодом на Pascal.

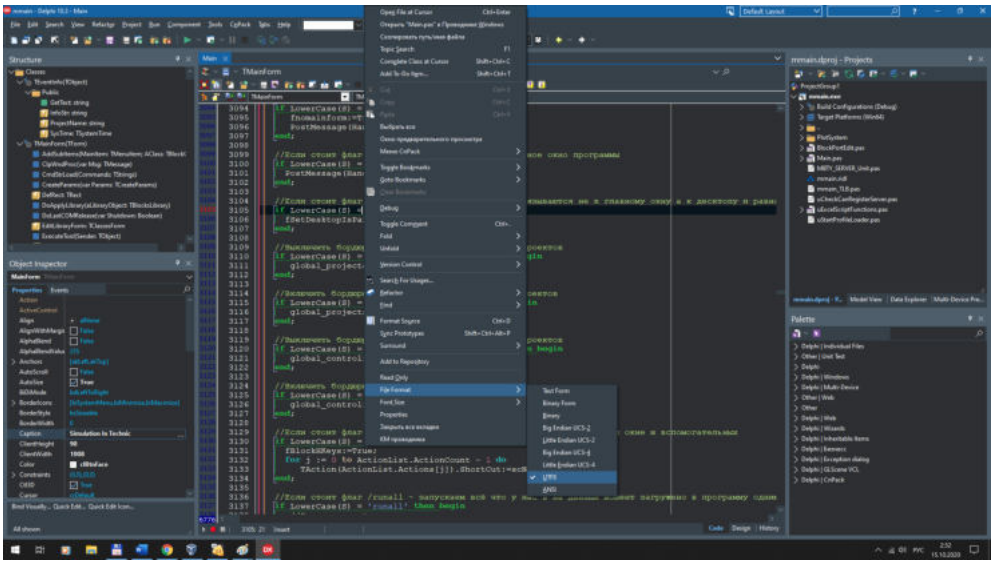
Далее был предпринят анализ используемых компонентов и поиск их аналогов в CodeTyphon Studio, чтобы понять, насколько будет модифицироваться код программы. Ниже приведена таблица, показывающая что где есть.

Компонент	Delphi	CodeTyphon Studio\ Lazarus
TButton	+	+
Tedit	+	+
TjvComponentPanel	+	+
TactionList	+	+
TjvFormMagnet	+	-
TcontrolBar	+	+
TtoolBar	+	+
TjvOfficeColorPanel	+	- есть аналог ThexaColorPicker
TcheckBox	+	+
TcomboBox	+	+
TjvListBox	+	- заменяем на TListBox
TeeChart Pro	+	- штатно нет
TjvSpinEdit	+	- заменяем на TspinEdit
TjvFontComboBox	+	- заменяем на TplFontComboBox
TBCEditor	+	-заменяем на TsynEdit
TvirtualStringTree	+	+
TPageControl	+	+
TTabControl	+	+
TPanel	+	+
TBitBtn	+	+
TJvButton	+	- заменяем на TBitBtn
TScrollBar	+	+
TFrame	+	+
TJvListView	+	- заменяем на TListView
TSpeedButton	+	+
Компоненты системных диалогов	+	+
GLScenes	+	+
DSPack	+	- есть другие компоненты для звука
Indy	+	+
TPngImage	+	+ заменяем на TPortableNetworkGraphic

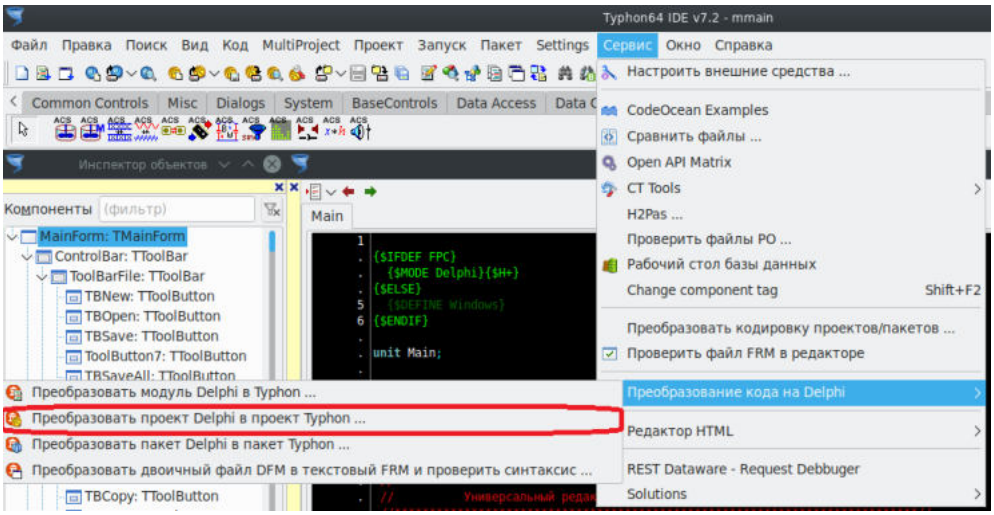
Вообще на текущий момент компоненты JVCL в этой сборке портированы практически все, поэтому часть замен можно и не делать. Анализ показал, что всё что надо программе, чтобы обеспечить сборку на Linux там имеется, кроме TeeChart Pro, о котором будет описано чуть ниже, потому что его удалось в итоге заставить там работать и это сняло сразу большую головную боль.

После успешно проведённой проверки среды CodeTyphon Studio на тестовом приложении был начат непосредственно процесс портирования и формирования единой кодовой базы для Windows и Linux версий программы. Для этого сначала на виртуальной машине был настроен git и установлена для удобства работы программы SmartGit, была выполнена загрузка рабочей копии из репозитория и создана отдельная ветка altlinux.

Первое что необходимо сделать для портирования программы на Lazarus\CodeTyphon это пересохранить ВСЕ файлы исходного текста (pas, dpr и разные include-ы) в формат UTF8 BOM ! Это строго необходимо, **если у вас в сообщениях есть хотя бы одна русская буква** ! Это делается в Delphi или в любом другом текстовом редакторе.



Второе – это с формирования файлов проектов (ctpr) под данную среду разработки. Первоначально это было сделано при помощи функции



В дальнейшем ctpr-файлы, проектов для CodeTyphon Studio просто копировались для разных модулей с изменением внутри этого xml-файла имени проекта и названия самого файла и модулей в проекте. Также можно просто сделать там пустой проект и дальше размножить из него ctpr файлы на все компоненты программы.

Процесс портирования естественно был начат с самого проблемного с этой точки зрения места – то есть графической оболочки программы, где сосредоточен практически весь пользовательский интерфейс. После того, как мы получили ctpr-файл, открываем его в в Code Typhon, прописываем все пути поиска исходников, если проект был разложен по директориям и первым делом вносим модификации в dpr-файл проекта графической оболочки (головного исполняемого модуля программы) – добавляем с условной компиляцией модули:

```
  simmm,
  {$IFDEF UNIX}
  cthreads,
  {$ENDIF }
  {$IFDEF FPC}
  Interfaces,
  {$ENDIF}
```

Делаем экспорт функции получения менеджера памяти головного модуля:

```
{$IFDEF FPC}
exports
  GetMemoryManager;  // это надо чтобы общий менеджер памяти увидели SO-шки
{$ENDIF}
```

и убираем в условную компиляцию модуль подключения COM-сервера и функций для работы с Excel и если есть что-то ещё системно-зависимое.

Дальше была нажата кнопка «Скомпилировать» в среде разработке и пошло исправление выдаваемых компилятором ошибок. Первое что пришлось проделать во всех модулях – это добавить в заголовок юнита вот такое определение, чтобы включался режим совместимости:

```
{$IFDEF FPC}
 {$MODE Delphi}{$H+}
{$ELSE}
 {$DEFINE Windows}
{$ENDIF}
```

Я в курсе что это также включается в настройках проекте в CodeTyphon Studio, но некоторые модули в такой режим не переводились.

Потом были обвязаны условной компиляцией все системно-зависимые модули в uses – в первую очередь Windows и Messages:

```
{$IFDEF FPC} FileUtil,{$ELSE} Windows,{$ENDIF}
```

и введены в базовые модули программы (DataTypes.pas) недостающие элементарные типы:

```
  {$IFDEF FPC}
  PRect = ^TRect;
  {$ENDIF}
TWindowsPointArray = array of {$IFNDEF FPC}Windows.{$ENDIF}TPoint;
```

Дальше оказалось, что для обеспечения бинарной совместимости при сохранении и чтении файлов необходимо учитывать то, что **размеры перечислимых и множественных типов данных в Delphi и Free Pascal разные**:

В Delphi – 1 байт, в **FreePascal – 4**, что касалось перечисление, например, стилей линий или заливок:

```
  //Размеры некоторых типов данных
 SizeOfTColor      = 4;
 SizeOfTPenStyle   = 1;    //SizeOf(TPenStyle);     //В FPC размеры enum 4 байта !
 SizeOfTBrushStyle = 1;   //SizeOf(TBrushStyle);
SizeOfTDataMode   = 1;  //SizeOf(TDataMode);
```

Поэтому везде в коде там где размеры типов в компиляторах различались было закомментировано использование SizeOf и размер задан числом фиксированно !

Также были определены недостающие функции, которые присутствовали в модуле Windows:

```
{$IfDef FPC}
procedure ZeroMemory(const Data: pointer; Size: integer); inline;
begin
  FillChar(Data^,Size,0);
end;
{$EndIf}
```

Ну и соответственно, те места, которые вызывали функции из Windows были заменены эквивалентами из кросс-платформенных модулей:

```
{$IfDef fpc}CreateDir(newdir){$else}CreateDirectory(PChar(newdir),nil) and
(GetLastError <> ERROR_ALREADY_EXISTS){$endif}
```

Также были переделаны функции сохранения и считывания строк из потока для обеспечения возможности загрузки данных из Windows версии, т.к. формат строк в Lazarus\CodeTyphon – UTF8 по умолчанию, а в современном Delphi – UTF16:

```
{$IFDEF FPC}
procedureSaveStr(constS:string;Stream:TStream);
varN,c: cardinal;
UniString: UnicodeString;
begin
UniString:=UTF8Decode(S);
N:=Length(UniString);
```

```
withStreamdobegin
c:=Nor$80000000;
Write(c,SizeOfInt);
ifN>0thenbegin
Write(Pointer(UniString)^,N*2);
end;
end
end;

                                              procedure   LoadStr(var
S:string;Stream:TStream);
  var N: cardinal;
      UniString: UnicodeString;
  procedure DoLoadAsAnsi;
   var tmps: ansistring;
  begin
     //старая версия - ansi
     SetLength(tmps,N);
     if N>0 then Stream.Read(Pointer(tmps)^, N);
     S:=tmps;
  end;
begin
 with Stream do begin
   Read(N,SizeOfInt);
   if (N and $80000000) <> 0 then begin
     //юникод
     N:=N and (not $80000000);
     SetLength(S,N);
     SetLength(UniString,N);
     if N > 0 then begin
        Stream.Read(Pointer(UniString)^, N*2);
        S:=UTF8Encode(UniString);
     end;
   end
   else
     DoLoadAsAnsi;
 end
end;
{$ELSE}
.......
```

При этом надо обратить внимание, что эта функция эволюционировала ещё с не-юникодной версии Delphi, и соответственно, когда программа была переделана на юникод то для идентификации формата строки был задействован старший бит в 4-х байтном числе перед данными строки. Если он 1 то значит строка кодирована 2-байтными символами в UTF16, а если 0 – значит это старая ANSI строка.

В большинстве модулей где была голая математика не потребовалось вообще никаких изменений в коде, кроме как указать в начале модуля режим совместимости {$MODE Delphi}.

Потом компилятор напоролся на первый файл с формой, и тут уже пришлось видоизменить его по следующей схеме:

В самый верх модуля, как и у всех вставляем:

```
{$IFDEF FPC}
 {$MODE Delphi}{$H+}
{$ENDIF}
```

В uses обвязываем системно-зависимые модули условной компиляцией:

```
{$IFDEF FPC} LCLType, LCLIntf, LMessages, GraphType,{$ELSE}Windows, Messages,
{$ENDIF}
```

Модули:

SysUtils, Classes, Graphics, Controls, Forms, Dialogs, и другие свои

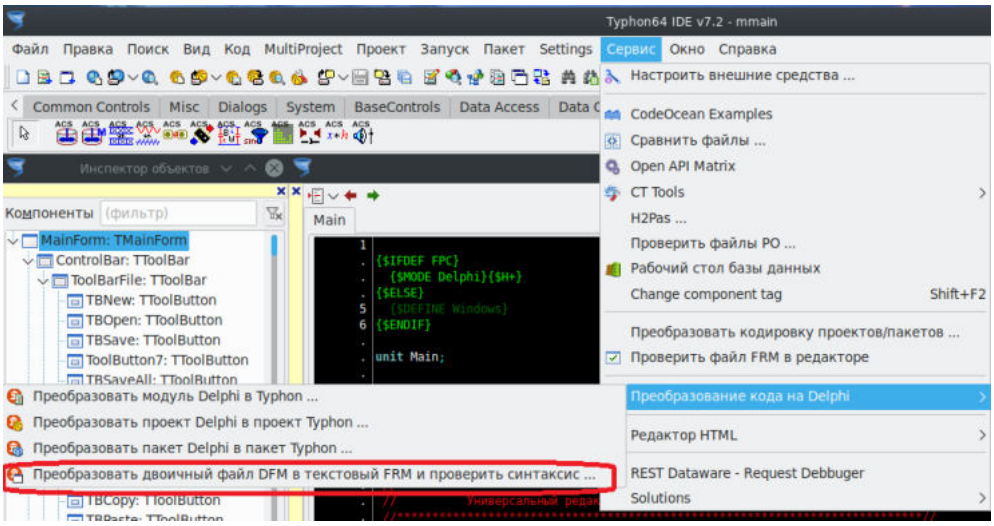
оставляем как есть, но если там префикс VCL. – то его надо убрать !

В секции implementation там где линкуется dfm-файл, вместо {$R *.dfm} вставляем условную компиляцию, чтобы для CodeTyphon у нас грузился файл формы с расширением frm (в данном случае там сделан также выбор и по языку интерфейса):

```
{$IFDEF ENG}
{$IFnDEF FPC}
{$R *.eng.dfm}
{$ELSE}
{$R *.eng.frm}
{$ENDIF}
{$ELSE}
{$IFnDEF FPC}
  {$R *.dfm}
{$ELSE}
  {$R *.frm}
{$ENDIF}
{$ENDIF}
```

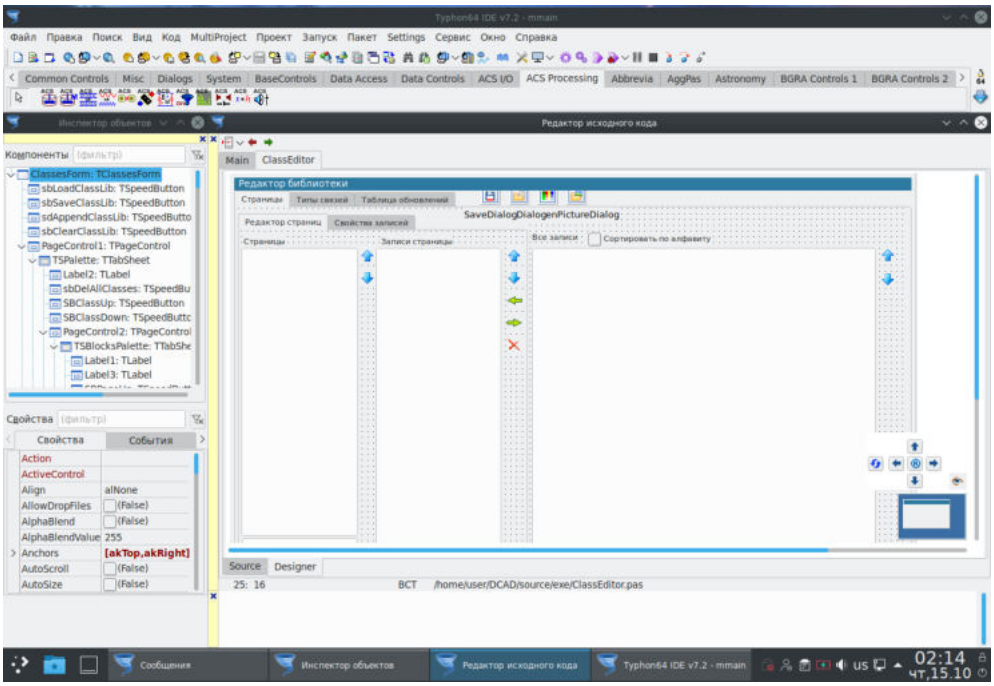
Дальше необходимо перенести файл конфигурации формы из Delphi (dfm) в другой формат CodeTyphon (frm). Для этого в Lazarus\Code Typhon существует инструмент конвертации dfm-файлов, который прекрасно понимает их в текстовом формате (хоть и пишет при этом ошибку).

Поэтому выбираем следующий пункт меню, не обращая никакого внимания на слово двоичный – с текстовым форматом dfm-файлов эта утилита также прекрасно работает:



и выбираем dfm файл конвертируемого модуля. После этого получаем сообщением об «ошибке» и не обращаем на него никакого внимания.

Если в frm -файле никакой экзотики которой нет в данной сборке нет, то мы можем сразу нажать кнопку «Переключить форму\модуль (F12)» которая сделает попытку загрузить frm-файл. Далее возможны 2 варианта: все компоненты есть – тогда мы сообщения о том что каких-то свойств нет просто игнорирует или же каких-то компонентов у нас нет – тогда мы открываем в текстовом редакторе frm-файл и меняем там имена классов компонентов на аналоги имеющиеся под линуксовой средой разработки и пробуем сделать «Переключить форму\модуль (F12)» ещё раз пока всё не станет хорошо и не появится дизайн формы:



В некоторых случаях, чтобы упростить задачу и не искать аналог описание компонента просто вырезалось из frm-файла в текстовом редакторе и компонент создавался уже динамически при FormCreate, где также была для этого сделана условная компиляция. Также надо обратить внимание, что размеры кнопок\полей ввода\чекбоксов в штатной линуксовой GTK2 которая используется там для Lazarus\Code Typhon как системная библиотека виджетов не совпадают с виндовыми и зачастую надо немножко повозить мышкой по формочке, чтобы привести её в более благородный вид.

Там где имена классов компонентов не совпадают – в исходном тексте класса формы делаем условную компиляцию, например так:

```
    cProgressBar: {$IFDEF FPC}TProgressBar{$ELSE}TJvGradientProgressBar{$ENDIF};
    cMsgView:  {$IFDEF FPC}TListBox{$ELSE}TJvListBox{$ENDIF};
```

Сконвертировав подобным образом модуль с формой жмём дальше на компиляцию и смотрим, что интересного нам напишет компилятор ещё. Различия в наименованиях полей компонентов решается тривиально – то есть введением условной компиляции, но как показал процесс портирования таких мест не так много, и всё становится существенно интереснее в тех местах, где начинается использование системного API, в программе таких мест было несколько:

- загрузка модулей через LoadLibrary – вызовы из WinAPI были заменены прокладками из модуля DynLibs с теми же именами. Функции выглядят точно также как в Windows, но единственное что пришлось сделать – это сделать при загрузке плагинов автозамену имени файла *.dll на lib*.so так как имена расширений были прописаны в конфигурациях блоков и проектов созданных в программе.

Касаемо загрузки SO в линуксе надо помнить, что если библиотеки программы находятся не в /bin и пути к ним не сконфигурированы в PATH, то при загрузке библиотеки к имени файла следует обязательно добавить путь к главному исполняемому файлу если модули лежат рядом в ним:

```
fLibHandle:=LoadLibrary({$IFNDEF FPC}PChar{$ENDIF}(ExtractFilePath({$IFDEF FPC}
ParamStr(0) {$ELSE} GetModuleName(HInstance) {$ENDIF})+rsalib_name));
```

Кроме этого если хендл SO\DLL был декларирован как THandle необходимо заменить его на TLibHandle, т.к. они имеют разные типы в разных ОС:

```
Var fLibHandle: TLibHandle; \\ Вместо THandle
```

- критические секции и объекты синхронизации (InitializeCriticalSection, CreateEvent и т.п.) – тоже были заменены прокладками из модуля syncobjs, который присутствует как в Free Pascal так и в Delphi (классы TCriticalSection и TEvent соотвественно):

WinAPI функция	Класс и метод
RTL_CRITICAL_SECTION	TCriticalSection
InitializeCriticalSection	TCriticalSection.Create
DeleteCriticalSection	TCriticalSection.Free
EnterCriticalSection	TCriticalSection.Enter
LeaveCriticalSection	TCriticalSection.Leave

hEvent	TEvent
CreateEvent	TEvent.Create
CloseHandle(hEvent)	TEvent.Free
SetEvent	TEvent.SetEvent
ResetEvent	TEvent.ResetEvent
WaitForSingleObjects	TEvent.WaitFor

Замену функций EnterCriticalSection… на класс TCriticalSection и т.п. категорически **рекомендуется** произвести ! И крайне не желательно при этом пользоваться одноимёнными функциями из LCL, во избежание проблем и путаницы !

- запуск процессов – все функции были выделены в отдельный модуль с условной компиляцией. Запуск процессов был сделан на Linux через вызов system, использовать exec тут нежелательно:

```
uses{$IFDEF Windows}Windows, {$ELSE}LCLType, unix, Baseunix,{$ENDIF}Classes,
SysUtils;
……
//Юниксовые варианты порождения процессов
functionWinExecAndWait32(constFileName:string; Visibility : integer=0;
TimeOut: cardinal={$IFDEF Windows}INFINITE{$ELSE}maxLongInt{$ENDIF}):longword;
varss: AnsiString;
begin
ss :=FileName;
Result:=fpSystem( ss );
end;

                                          function  MyWinExec(const S:
string;aShowMode: cardinal):cardinal;
 var ss : ansistring;
begin
  {$IFDEF Windows}
     ss:=S;
     Result:=Windows.WinExec(PAnsiChar(ss),aShowMode);
  {$ELSE}
     // Делаем выполнение системной команды БЕЗ ожидания - для этого в конце
добавляем &
     // Рекомендация: для порождения внешних вызовов пользовать fpSystem и не
форкать процесс ибо
     // это ведёт к проблемам с отладчиком и последующим закрытием программы.
     ss := S + ' &';
     Result:=fpSystem( ss );
  {$ENDIF}
end;
```

- движок двумерной графики Direct2D.

В старых версиях программы под Windows в качестве рендера использовался GDI и для работы с растрами – GDI+ , в более поздних (с появлением Windows 7) всё было заменено на вызовы Direct2D API, который работал на порядок быстрее и позволял сделать сильно более качественную графику. Но старый код был оставлен для обеспечения совместимости. При портировании на Linux возникло 2 варианта – портировать с обычными вызовами через TCanvas – они идентичны или же попробовать сделать более приличный вариант с использованием более продвинутой библиотеки. В качестве библиотеки ренденринга 2d-графики для Linux была взята ORCA2d а конкретно класс TD2Canvas надо которым была сочинена совместимая с классом TDirect2Canvas надстройка. Для отрисовки графики TD2Canvas использует cairo для Linux. Можно было сделать полностью свой класс для рендера с более прямым вызовом линуксовых API-функций, но это делать было неохота и поэтому я воспользовался более готовым вариантом. Корректировка вызывающего кода заключалась в основном в том, что вместо интерфейсов теперь были просто классы кое-где добавились вызовы Free для объектов. Текст объекта-прокладки для рендера графики приведён в Приложении 1.

Соответственно поскольку изменился рендер, то вместо штатного TPaintBox, для которой определялся контекст Direct2D при конвертации формы в редакторе изображений блоков и редакторе схем были скорректированы компоненты:

В декларации класса формы:

```
ShemePanel: {$IFDEF FPC}TD2Scene{$ELSE}TPanel{$ENDIF};
PaintBox: {$IFDEF FPC}TD2Image{$ELSE}TPaintBox{$ENDIF};                          //
Отрисовочная поверхность
```

При выделении канвы для отрисовки:

```
    {$IFDEF FPC}
    PaintBox.Bitmap.SetSize(ShemePanel.Width,ShemePanel.Height);
    PaintBox.OnPaint:=DoPaintPB;
    Canvas:=TDirect2DCanvas.Create(PaintBox.Bitmap.Canvas);
    Canvas.FBaseImageObject:=PaintBox;
    {$ELSE}
    Canvas:=TMyD2Canvas.Create(ShemePanel.Handle);
    {$ENDIF}
```

Были также переопределены перехватчики событий мыши, поскольку ORCA2D координаты возвращало в single.

Самым интересным оказалось то, что **рисовать под Linux в многопоточном режиме как позволял Direct2D крайне нежелательно**, поэтому поточный таймер был условной компиляцией заменён на обычный, **а вызов отрисовки на канве TD2Image необходимо было делать строго по вызову OnPaint данного компонента**, т.к. иначе могли возникать перемигивания если отрисовку вызывать вне этого вызова. Это обусловило необходимость **сделать вызов отрисовки через вызов ShemePanel.Repaint** вместо прямого вызова рисования как для Direct2D варианта.

Глава 4: Загадки с обработкой исключений в Linux внутри динамически загружаемых библиотек

Всё было хорошо, но однажды мне захотелось добавить в линуксовой виртуалке ядер программы, чтобы она шевелилась пошустрее и тут началось нечто загадочное – стали заваливаться пакеты проектов (т.е. когда внутри программы параллельно на разных потоках считают несколько задач с обменом данными). Т.е. пускаешь, а оно заваливается в абсолютно произвольный момент времени и причём не пишет информации отчего завалилось от слова совсем. Я начал думать. Сделал тест - взял 2 проекта автоматики пустых и пустил их в оболочке паралелльно. Без синхронизации без всего вообще. Результат - если в виртуалке настроено 2 процессора заваливается через секунд 5. Причём при заваливании выводит в произвольные места. Сделал тест - урезал проекты оставил только плагин проекта common в загрузке. Запустил - заваливается через 15 сек. **Начал смотреть что не так и обнаружил что если в DLL присутствует код обработки исключений try except finally то когда в такую функцию входят одновременно 2 потока то происходит капут.** Начал смотреть ассемблерный код обработки исключений в Linux (в Windows он совсем другой) и оказалось, что try except при входе в секцию вызывает функцию записи указателя в связанный список состоящий из стековых переменных но при этом базовая переменная декларирована как threadvar в списке глобальных переменных системного модуля except.inc в RTL компилятора. **И после этого до меня дошло - что threadvar-ы не работают если в SO в головном файле проекта не слинкован модуль cthreads. То есть его надо линковать не только там где ты поток порождаешь в программе а везде в любых SO в которые этот поток может заглянуть и где используется обработка исключений. А если не прописать - то компилятор ничего не скажет, но threadvar-ы будут работать как обычные var - то есть существовать в одном экземпляре внутри процесса. Соотвественно при многопоточном режиме будет оно глючить не пойми где и молча. После того как я во ВСЕХ модулях на паскале подключил первым модулем cthreads всё чудесным образом заработало без сбоев.** Для упрощения задачи я включил cthreads сразу в модуль simm который предоставляет модулям общий менеджер памяти.

Вот этот код виноват был в модуле except.inc в RTL Free Pascal:

```
{$ifdef FPC_HAS_FEATURE_THREADING}
ThreadVar
{$else FPC_HAS_FEATURE_THREADING}
Var
{$endif FPC_HAS_FEATURE_THREADING}
ExceptAddrStack   : PExceptAddr;
ExceptObjectStack : PExceptObject;
ExceptTryLevel    : ObjpasInt;



......

Function fpc_PushExceptAddr (Ft: {$ifdef
CPU16}SmallInt{$else}Longint{$endif};_buf,_newaddr : pointer): PJmp_buf ;
  [Public, Alias : 'FPC_PUSHEXCEPTADDR'];compilerproc;
var
  _ExceptAddrstack : ^PExceptAddr;
begin
```

```
{$ifdef excdebug}
  writeln ('In PushExceptAddr');
{$endif}
  _ExceptAddrstack:=@ExceptAddrstack;
  PExceptAddr(_newaddr)^.Next:=_ExceptAddrstack^;
  _ExceptAddrStack^:=PExceptAddr(_newaddr);
  PExceptAddr(_newaddr)^.Buf:=PJmp_Buf(_buf);
  PExceptAddr(_newaddr)^.FrameType:=ft;
  result:=PJmp_Buf(_buf);
end;
```

Естественно информации о том, что без cthreads под линуксом threadvar-ы не работают, нету нифига нигде вообще, ни в интернете, ни в документации. Поэтому возможно кому-то пригодится этот документ.

Глава 5. Странности при завершении работы программы или загадки со строками

После того, как была прояснена особенность с обработкой исключений в многопоточном режиме в компиляторе FreePascal программа в общем и целом заработала стабильно – то есть тестовый пакет проектов, включающий в себя выполнение нескольких параллельных задач устойчиво считал и ни разу не завалил программу. Но почему-то при завершении работы программы при вызове FreeLibrary для всех плагинов происходила выдача исключения. Этот вопрос меня возбудил и не давал мне спать по ночам, намекая на какую-то нехорошую штуку при выделении памяти. При этом сбой вёл на функцию fpc_ansistr_decr_ref, что намекало на проблемы с работой со строками в плагинах. Сразу надо оговориться – в Delphi под Windows такой проблемы не было. Для прояснения данного момента был проведён эксперимент – в загрузке был оставлен один плагин, дававший гарантированный сбой. У плагинов при загрузке программы вызывается функция Init, которая делает некоторые инициализирующие действия после загрузки библиотеки. В результате эксперимента оказалось, что сбой при выходе программы происходит в случае, если из Init плагина вызывается функция главного интерфейса TMain.AddMenuItem (TMain это структура, содержащая адреса подпрограмм и внутренних данных, которые доступны плагинам, см. модуль InterfaceUnit.pas).

Данная функция была задекларирована следующим образом:

AddMenuItem:function(Parent: Pointer;**const Caption:string**;OnClick: TNotifyEvent;Tag:NativeInt):Pointer;

И вызывалась в коде плагина как

DllInfo.Main.AddMenuItem(DllInfo.Main.Menu,**txtTppTools**,nil,0);

где txtTppTools – это строковая константа

Внутри реализации интерфейсной функции AddMenuItem в головном модуле программы Caption присваивался соответственно полю Caption нового экземпляра TMenuItem. Для интереса присвоение строки было закомментировано и случилось чудо – ошибка при выходе исчезла !

Дальнейшие раздумья позволили прояснить детали проблемы:

FreePascal при присвоении строк может делать это без копирования данных – т.е. если строка-источник является константой то он берёт и присваивает указатель строки приёмника = адресу строки источника без копирования собственно данных строки на новой место при этом для строки-источника увеличивается на 1 счётчик ссылок, когда строка-приёмник деаллоцируется или меняется то счётчик ссылок строки-источника наоборот уменьшается на 1. И всё это хорошо и правильно пока мы не выгружаем shared object из памяти. А после этого случается страшное –менеджер памяти при завершении программы пытается декрементировать счётчик ссылок на строчку в том месте, где её нету уже и в помине (т.к. константа выделяется не в общей куче). Выходом из данной ситуации является замена типа string для функций которые могут быть вызваны с аргументом-строковой константой на PChar. **В идеале для всех интерфейсных функций лучше как раз PChar и использовать, т.е. он гарантирует, что внутри реализации интерфейсной функции будут скопированы данные строки, а не просто указатель.**

Соответственно, интерфейсные функции со строковыми аргументами были переделаны на использование PChar в качестве входного параметра:

```
AddMenuItem:function(Parent: Pointer;Caption:PChar;OnClick:
TNotifyEvent;Tag:NativeInt):Pointer;

InsertMenuItem:function(Parent: Pointer;Caption:PChar;OnClick:
TNotifyEvent;Tag,AIndex:integer):Pointer;

InsertMenuItem:function(Parent: Pointer;Caption:PChar;OnClick:
TNotifyEvent;Tag,AIndex:integer):Pointer;
```

```
SetAlias: function(aAlias,aAliasValue: PChar;fReplace: boolean):boolean;
```

Также была добавлена интерфейсная функция для безопасного присвоения имени компонента внутри головного модуля:

```
SetComponentName:  procedure(aComponentPtr: Pointer;aName: PChar);
```

На замену конструкции TComponent(aComponentPtr).Name := aName

После замены в плагинах типа строкового аргумента с const … : string на PChar проблемы связанные с выгрузкой плагинов были устранены.

Глава 6. Особенности некоторых компонентов под Linux в библиотеке LCL

У стандартных компонентов LCL существует разница в реализации по сравнению с компонентами VCL, связанная с применением некоторых особенностей.

При использовании компонента TTabControl в LCL следует помнить, что **использовать TTabControl.Tabs.Objects** под свои нужды в LCL категорически нельзя в отличие от VCL ! Поля Objects в поле Tabs в классе TTabControl в LCL используются для хранения ссылок на внутренние данные этого компонента ! Поэтому для хранения ссылок на объекты ассоциированные с табами следует ввести дополнительный список, фрагмент кода как надо делать для FPC и как было для Delphi приведён ниже:

```
         {$IFDEF FPC}
         TCComps.Tabs.Add( TranslationEngine.TranslateWordFunc(
MainLibrary.Tabs[i], TID_PANELBUTTONS) );
         TShowedTabList.Add(Tab);
         {$ELSE}
         TCComps.Tabs.AddObject(
TranslationEngine.TranslateWordFunc(MainLibrary.Tabs[i],TID_PANELBUTTONS ),Tab);
         {$ENDIF}
```

От TControlBar лучше отказаться и заменить его на TToolBar – это может быть сделано условной компиляцией и заменой имени класса в frm-файле. TControlBar работает в GTK2 плохо.

При динамической смене стиля бордюра окна на безрамочный режим (используется при переключении с режима редактора в режим видеокадра) следует пересоздать окно:

```


{$IFDEF FPC}
ActiveEditor.DefRect:=ActiveEditor.BoundsRect;
ActiveEditor.PaintTimer.Enabled:=False;
{$ENDIF}



ActiveEditor.BorderStyle:=bsNone;

        {$IFDEF FPC}
        RecreateWnd(ActiveEditor);   //Вот без этого в линуксе бордюры динамически
не скрываются
        {$ENDIF}
```

Динамическая смена бордюра окон на безрамочные может вызывать сбои LCL (в случае когда было окно bsSizeable а делаем мы его bsNone), поэтому если штатно дизайн окна в frm-файле выполнен с рамкой, то лучше сделать глобальную переменную для режима запуска для всех окон программы в безрамочном стиле. Переопределить стиль окна можно путём override для функции CreateParams оконного класса:

```
procedure TEditForm.CreateParams(var Params :TCreateParams);
begin
  // Тут мы проверяем и если надо применяем флаг для того, чтобы инициализировать
окно редактора без бордюров !
  if global_projects_window_no_border then begin
     {$IFDEF FPC}
     FFormBorderStyle:=bsNone;
     {$ELSE}
     //Такой фокус позволяем присвоить FBorderStyle напрямую без RecreateWindow на
этапе создания окна
```

```
    TFormBorderStyle((@BorderStyle)^):=bsNone;
    {$ENDIF}
  end;
```

Перерисовку после изменения размеров компонента\окна лучше выполнять через отдельное оконное сообщение, т.к. при инициировании прямой перерисовки из обработчика события она может не пройти.

```
  {$IFDEF FPC}
  //В Linux перерисовку через сообщение, т.к. иначе не инициализируется контекст
отрисовки
  PostMessage(Handle,WM_PaintSheme,0,0) ;
  {$ELSE}
  RepaintFormAll;    //Здесь делаем полную перерисовку, т.к. иначе изображение может
не обновиться !!!
  {$ENDIF}
```

Если внутри SO (DLL) есть форма, которая должна самоуничтожаться, то есть особенность – присвоение Action:=caFree внутри таких форм работать не будет ! Поэтому необходимо в обработчике OnClose напрямую вызвать Free для формы – это работает.

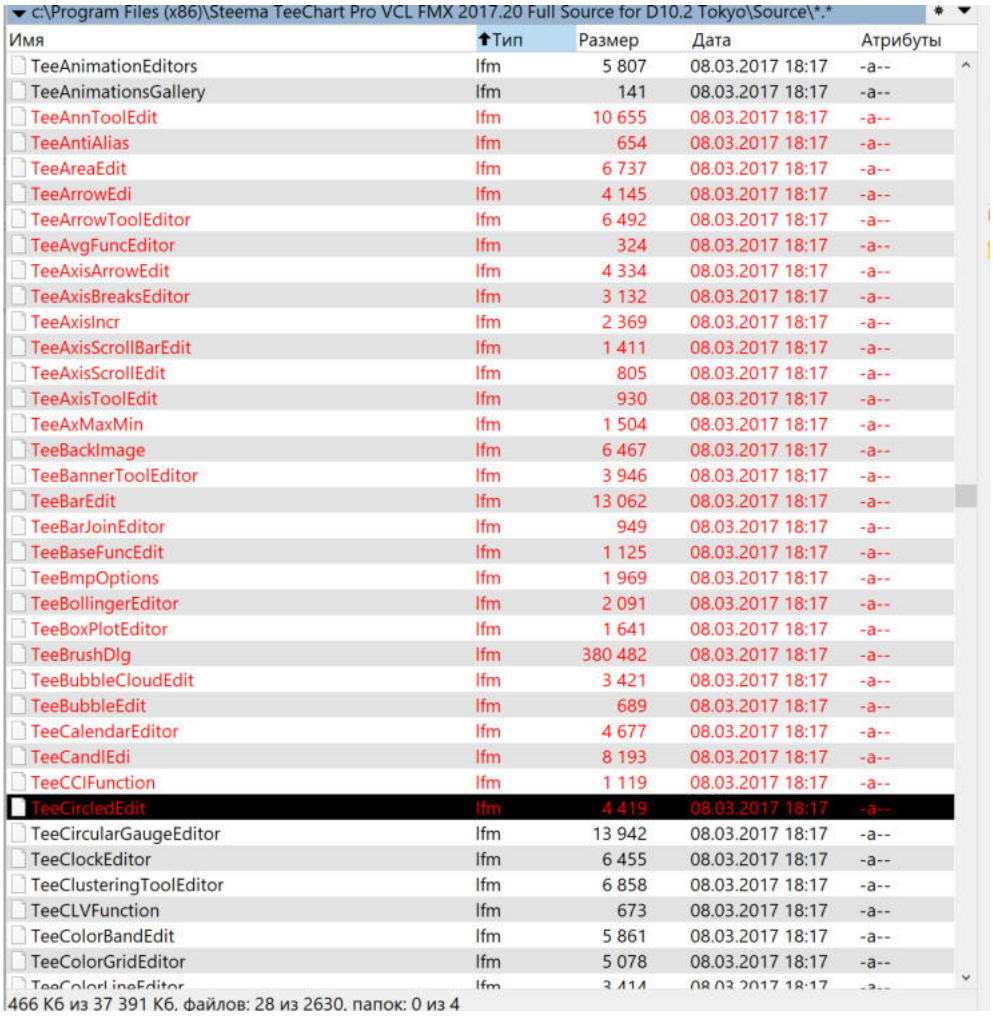
```
procedure TDebugBlockForm.FormClose(Sender: TObject; var Action: TCloseAction);
 var i: integer;
begin

  if aBlockPtr <> nil then begin
    StopShowData;
    TMBTYBlock(aBlockPtr).fBlockDebugForm:=nil;
    for I := 0 to Length(aInputsDataList) - 1 do
      aInputsDataList[i].fReplaceArray.Free;
    for I := 0 to Length(aOutputsDataList) - 1 do
      aOutputsDataList[i].fReplaceArray.Free;
  end;

  {$IFDEF FPC}
  Free;
  {$ELSE}
  Action:=caFree;
  {$ENDIF}
end;
```

Глава 7. А что с графиками под Linux?

Одной из основных проблем при портировании программы оказалось наличие компонента для построения графиков (2-х и 3-х мерных). Это обусловлено тем, что программа предназначена для расчёта динамики систем и результаты её работы именно в виде графиков в основном и представляются пользователю. Под Delphi в Windows мы использовали известный и достаточно качественный компонент TeeChart Pro от Steema software надо которым была сделана надстройка, которая этим компонентом управляла. Фирма даже его купила за 100 баксов официально и мы при разработке под Windows используем версию от 2017 года которая нас абсолютно полностью устраивает. Данная подсистема представляет одну из самых важных и сложных алгоритмически частей программного комплекса.

При портировании на Linux оказалось что в составе Lazarus штатно компонента этого нет, что меня сильно огорчило. Первоначально была сделана попытка использовать имеющийся в нём аналог - TAChart, который имеет схожий функционал. Соответственно в коде управляющей надстройки были сделаны IFDEF-ы и построение графика было переделано на этот компонент, но оказалось что он по визуальному представлению и некоторым очень необходимым функциям уступает TChart-у в Delphi (в частности не позволяет расположить вертикальные шкалы в одну линейку на разных уровнях).

При внимательном рассмотрении исходников компонента TeeChart Pro обнаружились забавные артефакты:



А в коде обнаружилось следующее:

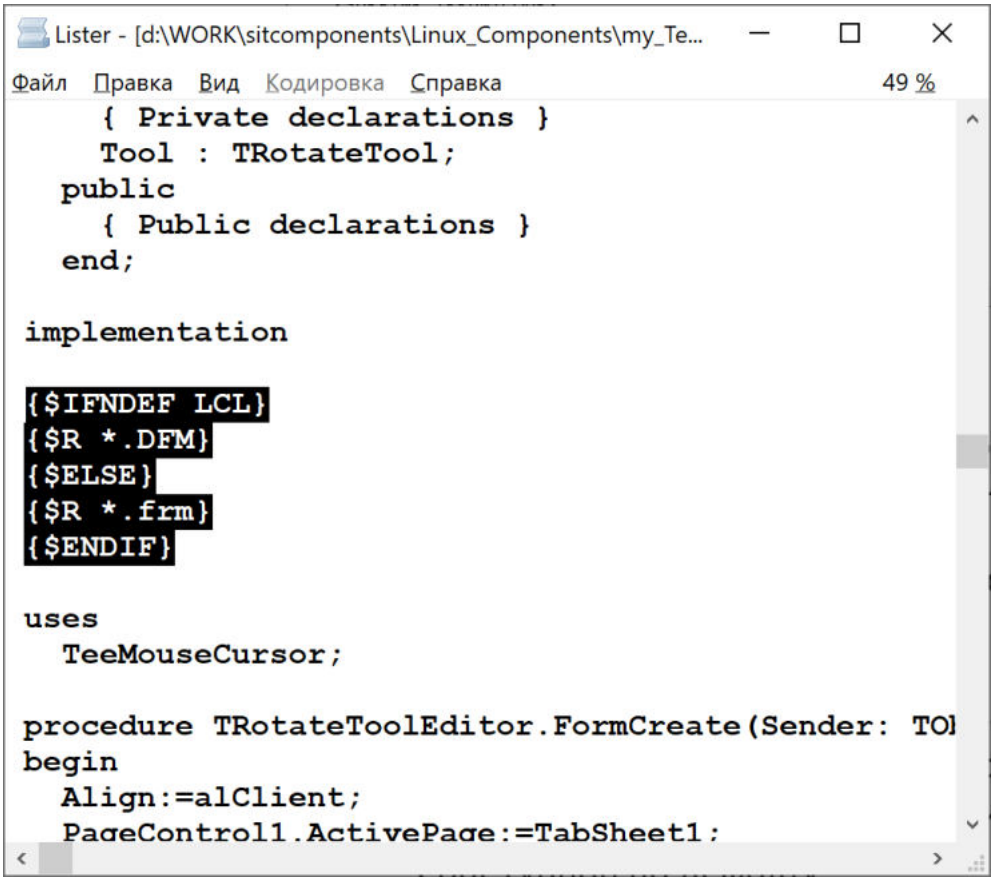
Ой что же это такое ! Это вжжж неспроста подумал я почесав опилки в голове ! Оказывается кто-то этот компонент начинал портировать под LCL и даже что-то умудрился сделать !

Далее была проделана некоторая подготовительная работа для того, чтобы попробовать всё на Code Typhon:

- Все найденные в Source .pas, .lpr, .lfm файлы были переконвертированы в формат UTF8+BOM для корректного восприятия их компилятором в Linux.

- У файлов .lfm расширение было изменено на .frm которое принято для форм в среде CodeTyphon по дефолту.

- Имя файла teechart.lpk было изменено на teechart.ctpkg
- Во всех файлах где были ссылки на lfm-файлы тоже был автоматом заменён IFDEF:

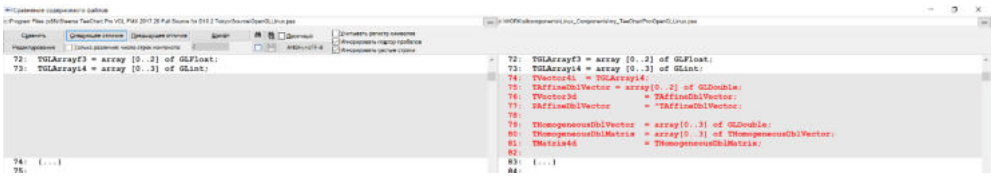


В результате проведённых корректировок был получен пакет который был успешно откомпилирован в Windows-версии среды разработки Code Typhon. И этот компонент даже вполне откомпилировался и заработал!

Этот результат воодушевил, поскольку это означало, что удастся без переписывания кода сохранить имеющийся функционал отображения графика.

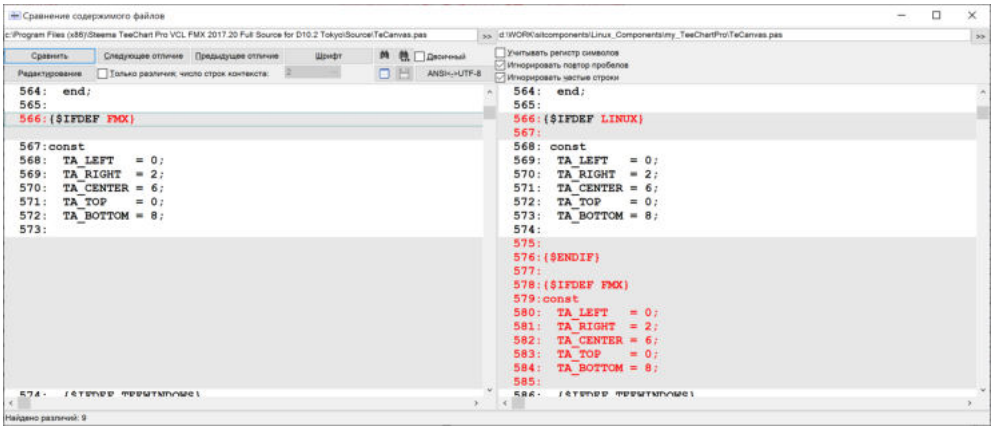
Решено было попробовать проделать тоже самое в Linux – компонент был переписан в usr\local\codetyphon\typhon\components в директорию my_TeeChartPro.

Файл teechart.ctpkg был скопирован в teechartlinux.ctpkg. Далее штатными средствами среды разработки файл был открыт нажата кнопка «Собрать». Но далее компилятор начал выдавать ошибки компиляции. Первое что оказалось, это то, что модуль OpenGLLinux был неполным и его пришлось дополнить определениями типов:

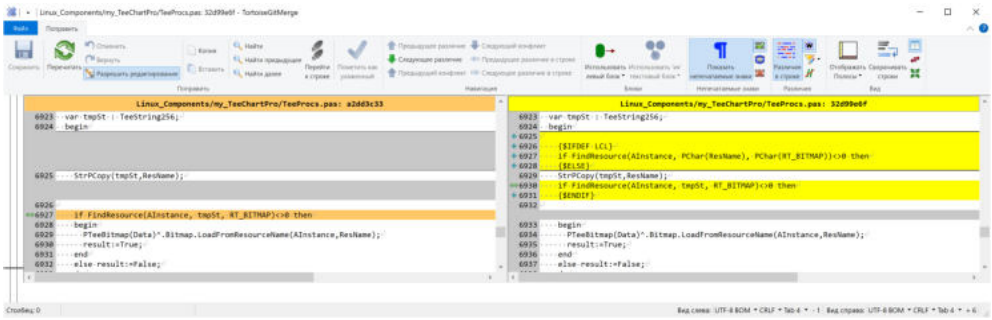


и некоторых функций, которых в этом файле не было (но в библиотеке OpenGL они точно есть). В принципе этот заголовочный файл есть в других компонентах и его можно было взять оттуда. Т.к. OpenGL рендер тут не планировалось пока использовать то цель была просто заставить компонент компилироваться в максимальной степени.

Далее были устранены ошибки компиляции в TeCanvas.pas путём добавления недостающих определений, которые были в модуле Windows:



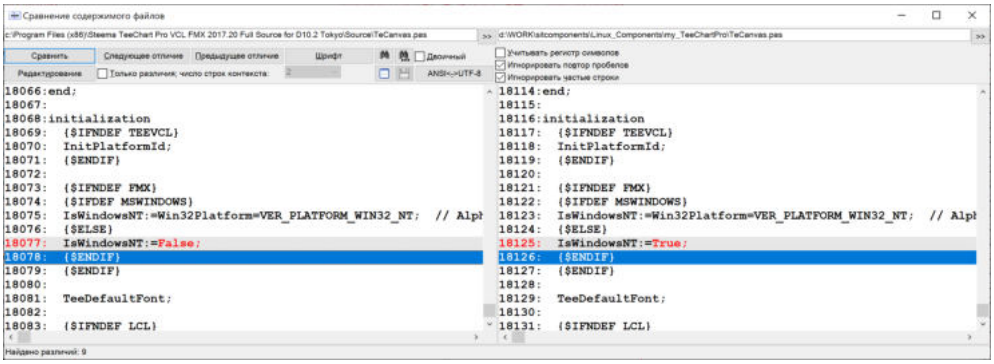
Также была сделана небольшое исправление при загрузке ресурсов.



После исправления этих ошибок компонент собрался. В коде моей программы были выключены старые IFDEF-ы и для Linux также был подключен TeeChart Pro для Linux который был мною модифицирован.

После того как программа была запущена оказалось, что на графике ничего не рисуется ☹ Какой облом однако. Был сделан вывод что хоть Бернеда и начал под FPC компонент портировать, но таки не доделал это нихрена.

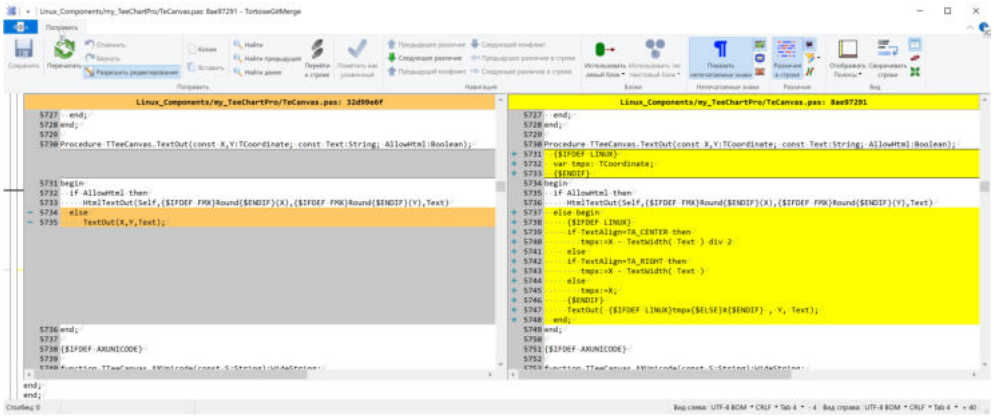
Разбирательство с кодом привело к тому, что был поправлен один небольшой фрагмент кода:

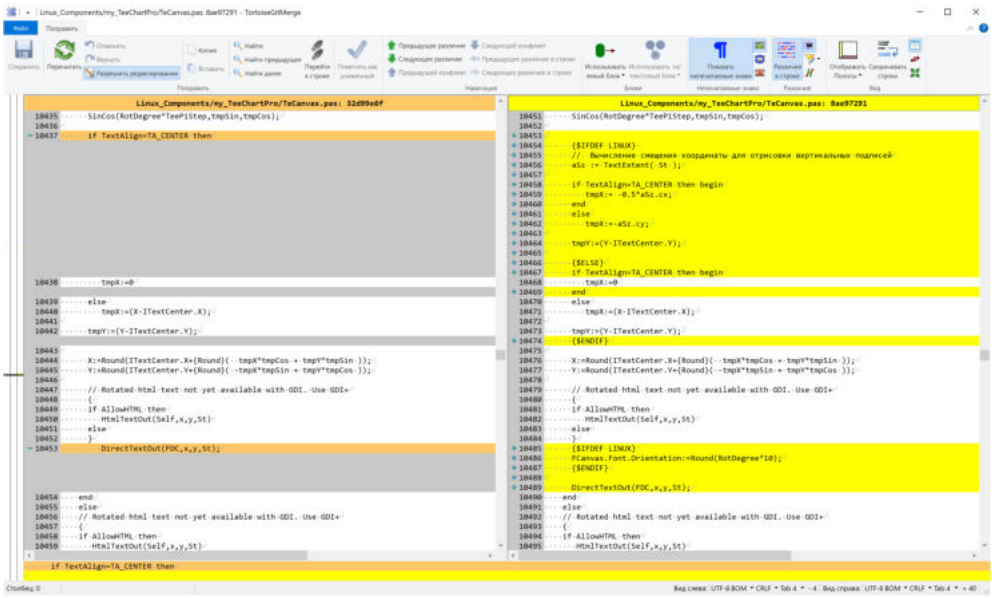


То есть флаг **IsWindowsNT был сделан True** по умолчанию. **После включения этого флага компонент начал рисовывать графики на Linux** причём логика его работы оставалась такой же как и в Windows, за исключением 2-х косяков:

1. Перестал работать разворот шрифта для вертикальных надписей.
2. Изменилось выравнивание подписей шкал, легенды и подписи графика относительно центра.

Эти проблемы были обусловлены тем, что в коде компонента не были доделана обработка этих особенностей для Linux. Для того, чтобы это всё заработало пришлось внести очень небольшие исправления в исходный код модуля TeCanvas.pas (желтым выделены доработки).





После внесения данных исправлений графики с применением TeeChart Pro завелись под LCL в том объёме, который требовался для функционирования нашего программного комплекса. Это позволило сильно сэкономить на переделке подсистемы построения графиков и сохранить логику её работы.

Выводы

1. Перенести сложный код на Delphi под Linux можно и вполне успешно. При этом вид и функционал может быть сохранён практически полностью. Придётся пожертвовать вызовами Excel через COM-объекты и COM-сервером внутри программы если таковой имеется.

2. При переносе надо учитывать подводные камни как библиотек и компилятора, так и самой операционной системы, что требует определённой квалификации. Я надеюсь, что данный документ прояснит некоторые вопросы с этим.

3. Переписывать проект на Java (или C++) **существенно дольше** и сложнее отлаживать из-за изменения логики исходного текста программы. Если считать непрерывную работу над этим вопросом, то для нашего программного комплекса это заняло чистого времени примерно месяца 3. Тут надо учитывать, что время это было несколько размазано, т.к. портировалась не замороженная версия и не отдельным специалистом, а всё это шло параллельно текущей разработке основной Windows-версии.

4. Компоненты в Code Typhon есть практически все, что и в Delphi. Но в принципе ничто не мешает взять и другую сборку Lazarus и настроить её под себя.

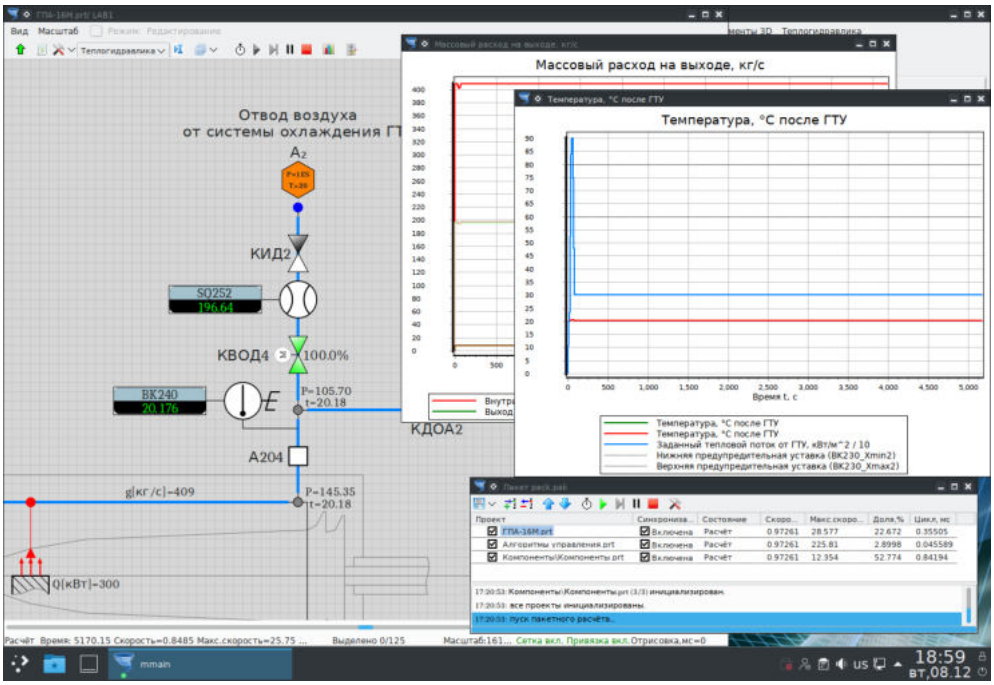
5. Писать код в Code Typhon\Lazarus вполне нормально, глюков не так много и в основном они связаны с дизайнером форм, если дизайн переносится с Delphi это не так уже критично. 90 % форм конвертируются полностью автоматом. У меня основное кодирование проводится сейчас на Delphi в Windows как привычной для меня среде, но дописывание кусков кода под линуксом проблем не составило.

6. С русскими символами проблем не возникает, если все файлы исходного текста сохранены в формате UTF8+BOM.

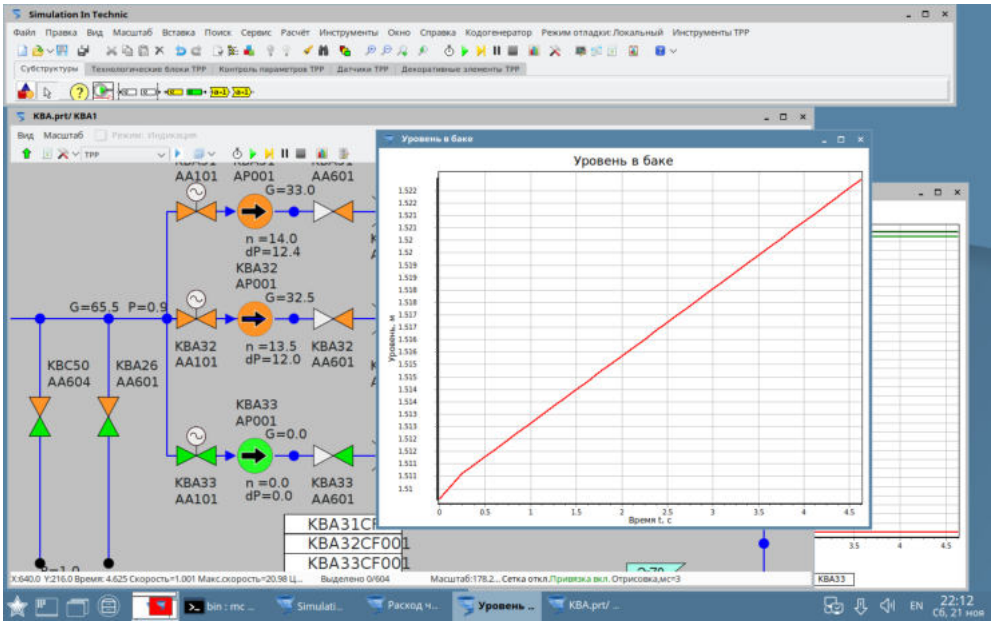
7. Сложное приложение на VCL быстрее перенести на LCL, нежели на FMX, так как между ними меньше различий.

8. Delphi закапывать рановато по крайней мере при разработке десктопного программного обеспечения, т.к. оно под Win32 в принципе до сих пор является одним из самых удобных средств разработки с компиляцией **в нативный код**, а под Linux имеются очень хорошо совместимые с ним аналоги. При этом ПО которое компилировалось на Free Pascal очень хорошо запускается на разных версиях Linux, чего нельзя сказать о некоторых модулях на FORTRAN – модуль скомпилированный с одной версией RTL может не работать на другой.

Скриншоты линуксовой версии прилагаются:



Фрагмент динамической модели газотурбинной установки под Alt Linux 9



Фрагмент модели промежуточного контура охлаждения АЭС с реактором типа ВВЭР под Astra Linux Orel

+90

165

Редакторский дайджест

Присылаем лучшие статьи раз в месяц

Тимофеев Константин @timofeevka

Программист

Комментарии 165

404

Вижу много боли по делу. Вопрос: вы рассматривали вариант delphi+wine? При кажущейся чужеродности, это вполне могла быть опция. Хотя у вашего подхода явным плюсом является нативный интерфейс получившегося.

НЛО прилетело и опубликовало эту надпись здесь

Тестировали. Медленнее и не держало direct2d версию. Старая с gdi рендером работала

а не пробовали наоборот, вести разработку под WSL (Windows Subsystem for Linux)?

Нет. Надо было проверять на конкретном дистрибутиве линукс. Но можно попробовать.

1. это позволит сразу компилировать под Linux, т.е. получить рабочие бинарники (там много дистрибутивов, вам явно хватит)
2. а так же есть возможность и запустить. Нужно лишь установить один из X-серверов под Windows и прописать его в переменные окружения
в файл .bashrc достаточно добавить
export DISPLAY=localhost:0

Мы пробовали свой проект запустить под Астру. Вроде шевелится кое-как, но файловые операции АДСКИ медленны.

НЛО прилетело и опубликовало эту надпись здесь

Линуксовая версия живёт в виртуалка, большие проекты тяжело сравнить. Небольшие примерно процентов на 20 медленнее но не все. В виндовой версии используется обсчет графики в отдельных потоках, поэтому если ядер много то лучше работает. Gtk так не умеет. Под винду используем delphi 10.3.3 так удобнее сейчас. Лазарус тоже пробовал. Заказчиков на линукс версию не особо много, но точто им надо (видеокадры системы регулирования для аэс) сейчас завелось.

Прям то что надо. Хотел искать подобную статью, а она как раз на хабре.
А вы пробовали последние версии Delphi которые в Architect редакции умеют в линукс из коробки, а в комплекте с CrossVcl могут собирать простые программы вообще без изменений?

Долго они её модерировали только. Сейчас сидим на 10.3.3. Возможно обновимся. Программа не очень простая. Fmx не охота было пользовать, написано было много. А так вроде заработало все с не особо большими корректировками.

Ммм… а почему Fmx? Vcl же https://www.crossvcl.com.

Эта штука не все сторонние GUI-компоненты "умеет".

К сожалению, не под любой Linux так можно компилить.
Там поддерживается совсем небольшой набор различных дистрибутивов: Supported Target Platforms

У вас не совсем верная информация. При использовании Delphi и FmxLinux подтверждена поддержка 172 различных дистрибутивов Linux


OCTAGRAM 8 янв 2021 в 03:35

Недавно поставил Delphi 10.3.3 Community и был удивлён, что в этой редакции стали доступны все цели: Win32&Win64, Linux, macOS, Android, iOS. В 10.2 был только Win32 на шару.

НЛО прилетело и опубликовало эту надпись здесь

Легко. Делаете модуль с формой и в винде он работает. Чтобы окна были в той же группе что и главное application.handle := тоже самое в головном модуле

> необходимо в so-библиотеке… определить инициализацию Application в секции initialization и завершение в finalization
> {$IFDEF FPC}
> initialization
> Application.Initialize;
> finalization
> Application.Terminate;
> end.
> {$ENDIF}

Удивил момент что надо инициировать Application в каждой библиотеке. В Delphi/VCL, насколько я помню, Application глобальный singleton объект и повторная инициализация смысла не имеет.

nicky7

> Вы нашли способ использования форм из dll (не bpl)?

bpl это расширенная версия dll с точки зрения интерфейса — к стандартным функциям добавлены дополнительные под нужды среды/vcl

PS: Очень впечатлён! Мне казалось что портирование сколько-то сложного проекта задача просто непосильная.

Application глобальный в пределах одного исполняемого файла (не процесса), т.е. он у каждой DLL загруженной вообще-то свой будет. В Delphi всё обходится без такой инициализации, в Free Pascal в Linux — надо в каждой DLL так делать, если там внутри формы есть.
Портировать задача посильная, но вообще всё сильно зависит от используемых библиотек.

Посмотрел исходники

```
procedure TApplication.Initialize;
begin
  if InitProc <> nil then TProcedure(InitProc);
end;
```

В простом проекте это пустышка — InitProc nil.
Только всякие платформозависимые модули добавляют обработчик — ComObj, ComServ, OleAuto и почему-то DBTables. Хотя и в нём всё сводится к тому-же CoInitialize. Оно и понятно — кто раньше застолбил потоковую модель использования COM, того и тапки.
С последним обстоятельством был связан очень неприятный баг — у нас клиент-серверное приложение работающее по DCOM с потоковой моделью FreeThread. На некоторых компах клиент (точнее винда) при попытке коннекта выдавал ошибку о несовпадении потоковой модели COM клиента и сервера. Оказалось драйверы принтера некоторой корпорации (не буду показывать пальцем, та у которой статьи почему чернила дороже чем кровь) при загрузке переинициализировали потоковую модель в SingleThread. Да MS явно в доке пишет что такое недопустимо, но что делать если хочется… Пришлось алаверды в dpr дописывать код принудительной переинициализации COM


OCTAGRAM 8 янв 2021 в 03:46

> bpl это расширенная версия dll с точки зрения интерфейса — к стандартным функциям добавлены дополнительные под нужды среды/vcl

Самые интересные расширения в ней — это то, что глобальные переменные типа Application начинают искаться в одном для всех связанных bpl модуле. Кроме Application ещё System.Classes.ApplicationHandleException и System.Classes.ApplicationShowException. Также в некотором смысле переменными являются ссылки на классы. То есть, если ловить исключение между dll, которые не bpl, как

```
on Occurrence: Exception do
```

То оно не поймается, ведь оно не тот Exception. И даже не тот TObject.

Респект Вам за работу и статью.
Сейчас на нашем гос предприятии сильно принуждают на русские линуксы переходить. Проблем не меренно. Так что общая тенденция ипортозамещения ПО будет только усиливаться.

Собственно вопрос (возможно детский):
А как через границу с dll ссылки на объекты вообще передаются? Может статью напишите по правильному и современному подключению плагинов с формами, объектами, колбеками…

Если компилятор одинаковый и для головного модуля и плагинов то при одинаковой настройке выравнивания структур выделяемые им в памяти области тоже одинаковые. Поэтому я делаю базовый абстрактный класс в модуле общем для exe и dll, а потом в dll его наследую. У dll наружу торчит функция createobject(name: pchar) которая внутри создаёт класс и возвращает наружу его указатель. А дальше в exe я обращаюсь к функциям абстрактного класса, по этому указателю. Это один из использованных способов. Это надо отдельно будет написать.

А если компиляторы разные — надо использовать COM.

Не обязательно. Можно использовать и структуры обычные. Можно интерфейсы без всякого COM. Для стыковки с сишными частями я делал интерфейсную структуру, как оказалось был прав в отношении переносимости.

В линуксе что надо сделать описано.

Линуксоид-вопрос. А найденные изменения послали в апстрим, разработчикам лазаруса-fpc?
«Идущие следом» были бы очень благода.

Ну статью могу послать. Если подскажете куда именно. Изменений то в компонентах только в tee chart pro за который фирма 500 баксов заплатила. А разработчикам лазарус эти особенности недурно было бы в справку внести.

> Фирма даже его купила за **100** баксов официально и мы при разработке под Windows используем версию от 2017 года которая нас абсолютно полностью устраивает.

Так 500 или 100?

Я на тот момент не могу сказать точной суммы, сейчас у них на сайте full source code 599$ стоит. Не осталось у меня тех старых документов.

НЛО прилетело и опубликовало эту надпись здесь

После прочтения заголовка мой внутренний голос взвопил «Зачем!??!», и я открыл почитать…

После прочтения я пожимаю автору руку и говорю «Браво!».

Но вот только «Delphi закапывать рановато» слишком пафосно. К сожалению уже поздно пить боржоми… Закапывать то, что уже закопано — действительно не надо. А закопан Delphi потому, что сейчас его уже никто в здравом умен не выберет для нового проекта. И то, что он уже закопан лично у меня вызывает грусть и сожаления.

Вопрос мегаспорный. Это скорее выбор инструмента под задачу и под человека. По сути если писать подобный нашему софт с требованиями скорости работы, кроссплатформенности и нативной интеграции с некоторыми внешними библиотека и на фортрана то выбор невелик — c++ и qt и как не странно Free Pascal с реализацией некоторых критических мест на си. На одной из прошлых моих работ мою систему 3 года переписывали на java и на редкость криво это сделали, хотя я им qt тогда советовал, на что мне ответили что сложно типа на qt. С другой стороны я знаю программы на qt которые разработчики не могут на линукс перетащить почему-то таким же образом. По поводу выбора инструментов под задачи разработки сапр вообще можно подискутировать.

А если бы не фортран, то были бы еще приемлемые варианты?

Вообще очень впечатлен вашим кругозором, который и позволил выбрать столь эффективный способ решения проблемы. Своим вопросом пытаюсь вашим кругозором воспользоваться. Если бы начинали свой проект сегодня, то выбрали бы C++ и QT?

Три месяца для такого проекта это просто ничто. Когда спрашивали Петухова (если не ошибаюсь, в комментах к вводной статье про SimInTech) когда будет линукс-версия, то он ничего толком не отвечал, работаем мол. Готовился уже годами ждать и надеяться что поспеете хоть к 16-му Эльбрусу. А оно вон как обернулось! Осталась только одна печаль — стоимость. Для АЭС конечно нормально, а для мелочи всякой и домашнего использования мда… SciLab+Xcos, а то и Jigrein наш удел :)

Для домашнего использования мы бесплатно ключ выписывает. Для мелких задач тоже — делаете заявку и мне в почту кидаете или на официальную. Если бы с нуля делал именно сейчас для gui наверное был бы qt. Но не факт, потому что них коммерческая политика лицензирования не особо дешёвая, да и говорят переход с версии на версию небезударный. Я знаю точно что я бы НЕ использовал — java. На прошлой работе часть моей системы на неё переписали и вышло сильно хуже исходной версии на delphi, да ещё и jni вызовы не быстрые…

Если с нуля делать, то, судя по описанному функционалу, я б серьёзно в сторону LabVIEW посмотрел. Там почти всё заточено под вашу задачу, как мне кажется. И линукс версия есть. У меня есть опыт разработки похожих продуктов и на Delphi и на LabVIEW, так вот на LabVIEW скорость разработки была заметно выше. Там лишь надо изначально грамотно всё спроектировать, чтоб потом монструозные спагетти-диаграммы не плодить. Но есть и минусы, конечно.

У LabView своя ниша, у нас своя которая с ними пересекается лишь частично. Основное назначение нашего пакета — создание комплексных моделей динамики энергетических установок, включая также и разработку моделей теплогидравлических сетей.

Во первых, хочу выразить благодарность за статью и «снимаю шляпу» перед вашим профессионализмом и мастерством.

Во-вторых, что касается выбора инструментальной среды. Мы сейчас тоже реализуем крупный проект, перед началом которого проводили выбор среды разработки. Поскольку изначально одним из условий ставилось обеспечение кроссплатформенности, то многие из популярных сред отпали «на старте». Java в корне не понравился своей производительностью и прожорливостью к ресурсам. У C# проблемы с кроссплатформенностью, что бы там не говорили его адепты. В итоге осталось два реальных варианта. Qt и C++ или Lazarus + FreePascal.
Так вот, в итоге мы выбрали именно Lazarus + FreePascal, поскольку оказалось, что вести большой проект

небольшой командой разработчиков (у нас тоже 3 человека) на Lazarus + FreePascal намного проще и быстрее, чем на связке Qt и C++. Причём мы даже сделали два средних проекта на Qt и C++, которые только убедили нас в этом. Сборка проекта происходит намного дольше, управление файлами существенно сложнее, «плясок с бубном» при сборке под разные платформы заметно больше.

В конечном итоге основной проект был запущен на Lazarus + FreePascal и имеющиеся на сегодня результаты показывают, что это был правильный выбор. Правда, мы не используем параллельно сборку на Delphi и Lazarus, что заметно упрощает нам жизнь. Часть когда когда-то была написана именно на Delphi, потом был небольшой период совместного использования кода, но сейчас уже практически всё отвязали от поддержки Delphi. В основном потому, что нам оно не требуется, а без проверки сборки в Delphi смысла прописывать условную компиляцию особого нет, только лишняя трата времени.

Спасибо за комментарий! У нас исходно проект был на Delphi и в принципе под Windows там нас устраивает данный инструмент разработки. У меня примерно те же самые мысли насчёт названных вами инструментов. На Java был опыт частичного переписывания — получилось медленнее и рендеринг двумерной графики медленно работал, короче вышло хуже исходного варианта на Delphi, а ещё она многословная очень. C# не рассматривали, потому что на Windows смысла менять инструмент не было, а на Linux с ним не всё хорошо. Qt и C++ — по сути единственная альтернатива для такого типа приложений, но тоже кода по объёму будет больше изрядно, в поддержке будет скорее всего сложнее, а будет ли лучше — это вопрос. Как показал мой опыт дельфовый код вполне переносим на FPC + Lazarus, особенно когда разобрался с приколами LCL и компилятора.


OCTAGRAM 8 янв 2021 в 04:06

Почему никто не смотрит в сторону AdaQt?


MylnikovDm 8 янв 2021 в 05:20

Именно потому, что в основе язык Ada, который морально устарел и не имеет тех удобств в программировании, которые есть у полноценных объектно-ориентированных языков.
У языка Ada, несомненно, есть определённые достоинства, но они не перевешивают имеющихся в нём минусов.


OCTAGRAM 8 янв 2021 в 08:41

В чём моральное устаревание, какая версия стандарта бралась для оценки и какие удобства имеются в виду?


MylnikovDm 8 янв 2021 в 14:37

Изначально язык Ada создавался Пентагоном как монолитная конструкция с достаточно чёткой концепцией его использования. И в этом плане на момент своего создания язык Ada был на самом деле очень хорош. Но, когда появилась и начала активно развиваться концепция объектно-ориентированного программирования, которую органично вписали во многие существовавшие на тот момент языка, например в C или Pascal, либо создав новые, изначально объектно ориентированные языки, такие как Java, разработчики Ada пошли своим путём. Да, в языке Ada изначально было понятие пакетов, которое позволяло реализовывать часть механизмов из концепции ООП. Но делалось это более громоздкими способами с точки зрения синтаксиса и удобства написания и восприятия программ. Например, та же точечная нотация при работе с объектами, насколько я знаю, появилась только в стандарте Ada 2005. А до этого момента использовалась так называемая «процедурная нотация».
Или то же нововведение в синтаксис, которое должно помогать компилятору ловить ошибки с перекрываемыми функциями и процедурами, когда для перекрываемых функция необходимо в начале указывать ключевое слово overriding, а для всех новых процедур и функций not overriding. В итоге получился механизм, который либо бесполезен, поскольку данные ключевые слова не являются обязательными, либо создаёт дополнительные заморочки разработчику, если он решает активировать в компиляторе эту возможность и вынужден будет многократно писать not overriding, поскольку новые функции в большой объектной иерархии добавляются постоянно и их часто больше, чем перекрываемых.

Кроме того, удобство использования того или иного языка определяются не только самим языком, но и наличием удобных и эффективных интегрированных сред разработки, которые включают в себя как инструментальные средства, так и развитые программные библиотеки. А и с тем, и с другим, у Ады, увы, всегда были проблемы, которые изначально во многом были обусловлены той сферой применения, под которую она разрабатывалась.

Кстати, появление проекта Qt Ada, о котором вы упомянули в самом начале, это как раз попытка решить данную проблему путём скрещивания библиотеки и среды разработки Qt с компилятором Ada. Но одним

только GUI разработка больших прикладных систем не ограничивается. Нужна ещё масса других библиотек для работы с СУБД, графикой, документами в стандартных форматах, тот же PDF, например.

Кстати, тот же Object Pascal, к сожалению, постепенно умирает как раз по причине того, что под него всё сложнее найти качественные библиотеки, соответствующие новым стандартам или поддерживающие последние редакции форматов тех или иных файлов данных. Особенно когда речь заходить о Free Pascal, а не о платной среде Delphi от Embacadero. Под последнюю ещё можно что-то найти из платных решений, но многие из этих решений не работают с Free Pascal или реализуют там не полный функционал, либо реализуют его менее качественно (много ошибок из-за менее тщательной отладки). Что в общем-то понятно и объяснимо. Если люди на этом зарабатывают, то основные усилия будут вкладываться туда, где больше платят. А сообщество Free Pascal деньги платить не любит.

> В итоге получился механизм, который либо бесполезен, поскольку данные ключевые слова не являются обязательными, либо создаёт дополнительные заморочки разработчику, если он решает активировать в компиляторе эту возможность и вынужден будет многократно писать not overriding, поскольку новые функции в большой объектной иерархии добавляются постоянно и их часто больше, чем перекрываемых.

Этот механизм активируется вместе с указанием версии стандарта, то есть, условно всегда. И нет, разработчик не становится обязан. Может быть, pragma Restriction есть, которая бы обязала, или ключ принуждения к стилю, но я такого не видел.

> та же точечная нотация при работе с объектами, насколько я знаю, появилась только в стандарте Ada 2005

И это было 15 лет назад.

Для некоторых проектов мне нужна супер переносимость, и я получаю её, используя AdaMagic, сертифицированный транслятор в Си. У него стандарт 95. Какой-то большой трагедии не усмотрел. Чисто инерция после Delphi.

> Нужна ещё масса других библиотек для работы с СУБД, графикой, документами в стандартных форматах, тот же PDF, например.

То есть, не хватает некоего COM, который позволял хорошо жить на Windows самым разным языкам программирования. Или обновления p2ada, чтоб апгрейдить больше паскалевских библиотек до адских. Ясно.

Я в своей практике, бывало, садился и полные привязки делал. Но, бывало, муторно, и привязывал только те внешние функции, которыми собрался пользоваться. Например, чтоб из AWS порулить ipset в ядре, привязал самый минимум. И сверх этого так больше ничего и не понадобилось.

timofeevka 8 янв 2021 в 16:39

Портирование на Ada не рассматривалось в данной работе. Про сам данный язык я ничего плохого не хочу сказать, но поскольку код системы был на писан на Delphi, то и для портирования применялся наиболее близкий по функционалу аналог. Цель была — иметь единую кодовую базу для Windows и Linux вариантов программы с сохранением используемого инструментария.

Я вообще из другой области, так что извините за возможную наивность. Вы не рассматривали модель frontend-backend — где собственно рендеринг отделен от вычислений каким-нибудь кросс-платформенным API? Тогда UI можно бы было хоть на HTML5 написать — с WebGL и монашками…

Рассматривали, но есть ограничения: 1. Программа инженерная и заточена на работу на локальной рабочей станции. 2. Переделывать придётся ВСЁ. 3. Совместимость со всеми старыми наработками критична.

Paskin 24 дек 2020 в 08:37

Так HTML5 != Web, есть фреймворки вроде JxBrowser/CEF или часто упоминаемого на Хабре ElectronJS, позволяющие писать десктопные аппликации путем оборачивания Chromium и добавления в него новых API.

timofeevka 24 дек 2020 в 09:52

Надо объем работ подробно оценить, сложность и результат на выходе. Для этого надо как минимум сделать минимально функциональный прототип. Посмотрим, пощупаем, понюхаем а там видно будет.

OCTAGRAM 8 янв 2021 в 03:56

Ещё это делают на Аде, с AdaQt или GtkAda. Нативно, кроссплатформенно, вменяемый синтаксис, система модулей и типов. И аналог bpl есть и работает.

Жаль, что путь миграции с Delphi на Ada не проторен. Приходится увиливать на гораздо менее продвинутый Free Pascal.

RaJa 2 фев 2021 в 15:00

Delphi — не просто компилятор. Это целая экосистема с огромным количеством библиотек, компонентов + возможность легко реализовать свои собственные компоненты как визуальные так и невизуальные с нативным кодом проекта и высокой скоростью разработки и функционирования приложения.
Ничего этого не предлагают ни Visual C++ ни C# ни Qt C++ ни тем более AdaQt.

OCTAGRAM 2 фев 2021 в 16:34

> Ничего этого не предлагают

Qt в сочетании с Адой или C++ — это как минимум нативный код и высокая скорость функционирования приложения.

RaJa 2 фев 2021 в 16:45

Которая на поверку оказывается не такой уж высокой из-за ненативных контролов Qt, весьма приличным размером и вечно несовпадающими Qt dll — ками. Свои компоненты невозможно интегрировать в панель редактора форм, из-за чего он практически бесполезен кроме как для примитивных форм. Переход между major версиями qt — боль и страдания, переносил с 4.x на 5.x несколько приложений, что-то удалось, что-то пришлось забросить, т.к. в 5.x вообще аналога нужных модулей нет и они несовместимы с идеологией 5.x

OCTAGRAM 2 фев 2021 в 17:15

> ненативных контролов Qt

Столбовая дорога современного Delphi, FireMonkey, в конечном итоге тоже такого типа получилась.

> Свои компоненты невозможно интегрировать в панель редактора форм

Да, в этом плане Delphi IDE блистает.

RaJa 2 фев 2021 в 17:20

> Столбовая дорога современного Delphi, FireMonkey, в конечном итоге тоже такого типа получилась.

Поэтому и не использую активно FireMonkey несмотря на ее некоторые плюшки.

tumaso 2 фев 2021 в 19:23

В FireMonkey есть возможность использования некоторых контролов не только как стилевых, но и как нативных (если уж очень сильно хочется именно нативные).

А вот насчет плюшек — самая главная плюшка это кроссплатформенность, за это можно смириться с некоторой ограниченностью относительно VCL. Хотя на мой взгляд, текущая версия fmx в 10.4.1 достигла коммерческой зрелости.

У меня рабочая версия 10.3.3 — 10.4.1 пока не видел.
Да и кросс-платформенность меня интересует только в ключе Linux. Android не особо нравится как компилируется из под Delphi. А вот FMXLinux только в дорогих редакциях вроде, а в 10.3.3 его просто нет

Gordon01 2 фев 2021 в 16:33

Есть еще Dear ImGui. На нем удобно делать приложения вроде вашего.


OCTAGRAM 8 янв 2021 в 03:51

> сейчас его уже никто в здравом уме не выберет для нового проекта

Запятые неправильно поставлены.

> сейчас его уже не выбирают для нового проекта

> сейчас уже никто не в здравом уме


HemulGM 25 июл 2022 в 12:39

С вами не согласятся многие, в их числе и я. Прошло много времени с момента пика популярности Delphi, утекло много воды, однако до сих пор он является одним из самых удобных инструментов для создания быстрых нативных десктопных приложений. Не говоря уже о том, что он развивается сейчас достаточно сильно. Можете зайти на офф. сайт Эмбаркадеро и посмотреть новости и блог.


Sly_tom_cat 29 июл 2022 в 22:04

> десктопных приложений

И где сейчас рынок этих приложений?
Мой вариант ответа: примерно там же где и Delphi....


timofeevka 29 июл 2022 в 23:05

CAD, CAE, спец софт всякий, PDM ы, офисные пакеты и т.п.


HemulGM 29 июл 2022 в 23:12

Вы забываете, что мир не только вебом полон. И всё через него, а тем более удобно и производительно не сделаешь.
Я, мы и многие другие занимаются написанием десктопного софта:
CRM, работа с графикой, и прочее.
Не говоря уже о том, что Delphi позволяет создавать и сайты (TMS, uniGUI) и речь тут именно про сайты целиком, а не только бэкенд, создавать мобильные приложения (FMX, FGX) и создавать софт под линукс или полностью кроссплатформенный софт. А в ближайшее время ожидается сборка под WebASM, что ещё сильнее продвинет Делфи в веб-разработке.


У Delphi есть проблема с большими проектами, IDE постоянно вылетает, жутко тормозит и постоянно проблемы с переходами к определению функции. У Lazarus такие же проблемы?


timofeevka 22 дек 2020 в 20:44

У меня delphi 10.3.3 не вылетает вроде. Code typhon может но редко крайне и это чаще происходило из за заваливания вообще всей операционки сначала.


swame 23 дек 2020 в 10:51

Проблемы со скоростью среды обычно связаны с плохим дизайном своих модулей (циклические ссылки между модулями), и с использованием внешних «помощников» типа CnPack. Еще с сильно загруженными компонентами формами, лучше делить на более мелкие и динамически компоновать в рантайме. У нас проекты более 1M+ строк, 2K+ модулей в проекте, проект билдится менее минуты, компилится в пределах 10-15 сек. Некоторые помощники Delphi типа AutoComplition отключаем. Среда падает не так часто, может раз в несколько дней.


andreyiq 23 дек 2020 в 17:04

У нас просто проект немного больше, полная сборка комплекса занимает несколько минут. Если правильно помню, то когда последний раз смотрел было 15M+ строк

swame 23 дек 2020 в 17:46

Точно 15M строк а не 15 Mb? 15M строк это что-то колоссальное.
А небольшое проект 15 Mb может билдиться несколько минут, если плохо структурирован.
У нас был период, время компиляции проекта порядка 400 тыс строк резко выросло с 1 минуты до 3. После того, как из него вычистили несколько десятков циклических зависимостей, время упало до 30 сек.


kryvichh 23 дек 2020 в 18:03

Мой самый большой проект 260K строк. Зато писан полностью собственноручно, и билдится за считанные секунды.


alan008 24 дек 2020 в 15:16

У нас текстовый редактор а-ля Ворд — 1.5 миллиона строк, система электронного документооборота — 1.5 миллиона строк. Так что 15 M строк вовсе не особо колоссальное, я думаю любая банковская система или ERP — что-то в этом районе и будет (плюс минус 5 миллионов :))


MylnikovDm 25 дек 2020 в 00:51

Если в проекте 15 млн. строк, и это именно одно монолитно приложение, то на 99% это плохо структурированный «клубок спагетти», содержащий большое количество старого кода, который тащится либо в целях «совместимости», либо просто потому, что некому навести порядок и провести нормальный реинжиниринг.
Также из практики могу предположить, что имеет место быть слабое объектно-ориентированное программирование или не грамотно построенная иерархия классов, из-за чего приходится писать много похожего или вообще дублирующегося кода.
Мне приходилось работать с подобным «большим проектом», в котором было сразу четыре фреймворка для работы с классами, которые были написаны в разное время и с разным пониманием того, что там должно быть. При этом на 90% код этих фреймворков, а также иерархий классов, которые от них строились, повторялись, но каждый для своей ветки.

Кстати, если попробовать собрать проект их 15 млн. строк на чём нибудь типа Qt в паре практически с любым компилятором C++, то такая сборка будет занимать не несколько минут, а несколько часов.


d_ilyich 25 дек 2020 в 11:00

«Моя душа беззвучно слёзы льёт» :_(


Raidar 22 дек 2020 в 19:39

Спасибо за статью.
Не пробовали создавать свои компоненты-обёртки?
Два пакета — под Delphi и CodeTyphon Studio — и меньше директив условной компиляции.


timofeevka 22 дек 2020 в 20:47

А их мало которые различаются. Честно говоря тупо лень было и так быстрее получилось. Проблемы основные были с выбором библиотеки 2д графики под линукс и забавными приколами с памятью и многопоточностью.


MylnikovDm 25 дек 2020 в 01:07

Я правильно понял, что библиотека, которую вы в конечном итоге использовали, из состава CodeTyphon, является просто фреймворком над OpenGL?
Просто мы тоже достаточно долго выбирали библиотеку для графической части, но пока так ни на чём и не остановились. Либо не устраивает функционал, либо скорость работы, либо и то, и другое одновременно. :(
В конечном итоге 3D сделали через OpenGL, частично использовав GLScene, частично написав свой код, поскольку другого кроссплатформенного варианта для этих целей просто нет. А вот с реализацией 2D возникли некоторые проблемы, особенно в части использования прозрачности. Поэтому пока вывод 2D через OpenGL отложили и сделали первый вариант просто через Canvas, но отказавшись от использования прозрачности.
Проблема была в том, что на платах AMD и NVidia реализация OpenGL при работе с 2D графикой отличаются.
В итоге при построении сложных изображений приходится уходить на уровень самых базовых примитивов: отрезок и треугольник. Но когда начинаешь строить ломанную линию из отдельных сегментов, используя при

этом прозрачность, то при толщине в несколько пикселов концы отрезков перекрываются и в этом месте прозрачность отображается не так, как требуется.


timofeevka 25 дек 2020 в 09:06

Под Windows используется direct2d. Под Linux — надстройка над cairo (gdk). Просто они функционально похожи оказались сильно и оказалось просто сделать прокладку. Речь о 2д графике — отрисовке схем и видеокадров.


timofeevka 25 дек 2020 в 09:08

Вопрос с прозрачностью у меня тоже возник, поэтому и делал на линуксе через orca2d а не через обычный canvas.


arrakisfremen 22 дек 2020 в 22:18

Я понимаю, что вопрос про деньги задавать не очень принято, но не могу не поинтересоваться, ваша зарплата окупает необходимость долботни с устаревшими технологиями и попытки портировать непортируемое?


timofeevka 22 дек 2020 в 23:11

Зарплата устраивает. Переписывать на другом стеке это минимум 5 лет работы и долботни будет больше намного, да и текущие средства разработки под наши задачи устраивают. Необходимо обеспечивать совместимость проектов между версиями по и непрерывную поддержку. Программа достаточно сложная и писалась не один год, а главное она вполне боевая и продаваемая.


0xf0a00 23 дек 2020 в 11:46

> устаревшими технологиями

Такое сказать может только тот у кого «Delphi» это Delphi 7. Хоть чуточку поинтересуйтесь на досуге, что из себя представляет современная версия.
Тем более что автор не раз указал, что использует версию 10.3.3 которая вышла в конце 2019.


arrakisfremen 23 дек 2020 в 16:56

Мало того, что я в курсе, у меня вот прям сейчас стоит на компе свежая версия. И я пытался перенести свою старую программу с седьмой версии на текущую, но потом забил, потому что это боль и всё не так работает. Хотя, в моей программе самые банальные контролы VCL и разные окошки. И дело совсем не в том, что нужно переписывать участки кода под юникод и всякие такие нюансы, а в том, что интерфейс кособочит и как это исправить, чёрт знает. Ну и сама иде — скорее пародия на вижуал студию, уж простите.


0xf0a00 23 дек 2020 в 18:29

Извините конечно, но я склонен вам не верить, потому что по работе я всякую старую мелочевку переводил с 7/2006 на последние версии и даже со сторонними компонентами в основном это заключилось в просто открытии проекта, а среда конвертила сама. По коду в основном надо было править типы и FormatSettings.

> иде — скорее пародия на вижуал студию

Oxx… насколько я помню то был человек который вложил тонну работы в делфи и его редактор, а потом он оказался в MC и работал над визуал студио и повторил свой редактор. Так что все наоборот, это не делфи похож на визуал, а визуал херовый клон делфи. Кстати как там визуал? Фризить и тупить он уже перестал?


Kealon 24 дек 2020 в 09:42

> И дело совсем не в том, что нужно переписывать участки кода под юникод и всякие такие нюансы,
> а в том, что интерфейс кособочит и как это исправить, чёрт знает.

так вам программист нужен

pfemidi 23 дек 2020 в 15:39

В сторону: А в Oreans на Delphi пишут… И не жужжат..."

megahertz 22 дек 2020 в 22:49

Тоже занимался таким, но все было куда проще — нужно было только выдрать бизнес логику и дать к ней доступ через вебсокеты. Кодовая база чуть больше мегабайта, но большая часть модуль вообще не требовали изменений, а обратная совместимость с Delphi не требовалась. Все необходимые флаги были выставлены в lpr файле, либо передавались fpc.

timofeevka 22 дек 2020 в 23:14

У меня все несколько сложнее — требуется полная совместимость, ну и объем исходников большой.

timofeevka 22 дек 2020 в 23:15

Вообще опыт разработки и внедрения системы на протяжении 17 лет сам по себе интересен.

kryvichh 23 дек 2020 в 00:47

Мощная статья, я бы так не смог. Скорее смотрел бы в сторону https://www.crossvcl.com/ .

hiddenman 23 дек 2020 в 04:36

Эх, какая ностальгия! Прочитал и вернулся куда-то в 1998 год, когда писал ночами на Borland Pascal-e, потом Virtual Pascal-e, Delphi, Free Pascal-e. Портировал сначала из DOS в Win32 и OS/2, потом в Linux.
А потом закончилось время just for fun и начались серые и скучные рабочие будни и семейные дела, которые уже не позволяют сидеть всю ночь над любимым проектом. Так иногда хочется вернуться во времена, когда я был жаден до любой информации и полон энтузиазма.
Спасибо!

arrakisfremen 23 дек 2020 в 17:09

> начались серые и скучные рабочие будни и семейные дел

Простите за непопулярное мнение, но… может стоит сменить работу и развестись? Жизнь одна.

hiddenman 23 дек 2020 в 17:11

К сожалению, за just for fun уже никто не платит. Это тогда можно было на кофе и сигаретах жить и лабать что-то интересное. А сейчас всё, что приносит деньги, обычно совсем неинтересное, тяжелое и выматывающее. Да и технологии уже далеко вперед ушли, никому эти все старые знания и умения не нужны.

fraks 23 дек 2020 в 06:39

Очень интересная тема.
Насколько я понял, несмотря на все IFDEF, исходники для Lazarus и исходники в Delphi — это теперь разные исходники, а DFM вообще пришлось допилить вручную после переноса, и делать это придется каждый раз при исправлениях в DFM.

Не возникло мысли полностью перейти на Lazarus, в т.ч. и на винде?
Или в этом случае есть какие-то существенные минусы?

timofeevka 23 дек 2020 в 08:16

Нет. Исходники у меня сейчас одни на всё. 2 файла форм пришлось откорректировать руками после конвертации (замена компонентов), но они редко меняются и это не трудоемко.

KongEnGe 23 дек 2020 в 12:29

Lazarus и близко не стоит к Delphi по удобству среды разработки, к сожалению.

timofeevka 23 дек 2020 в 12:38

Объективно да — в Delphi работать мне лично удобнее. Поэтому и делал общую кодовую базу — чтобы при изменении в под Delphi автоматом они были в Linux-версии. То есть сейчас если я исправляю что-то в коде — сначала делаю в Delphi, потом после изменений проверяю собираемость и работу под Linux — жму кнопку пересобрать проект там и проверяю так ли оно работает как и в Windows.

Hazactam 23 дек 2020 в 23:13

В Лазаре есть одна старая боль — отладчик. Его, к слову, сейчас весьма активно пилят. А так то некоторые фишки есть в Лазаре, которых в Делфи не хватает. Выделение дефанов хотя бы. Что бы далеко не ходить.

timofeevka 24 дек 2020 в 09:54

Вот то что он ifdef ы неактивные выделает это удобно. Кстати отладчик в Linux работал у меня лучше чем в виндовой версии lazarus

Hazactam 25 дек 2020 в 23:54

Лазарь вообще весь в Линуксе почти всегда лучше чем в Винде работает. А причина проста: авторы под Линуксом сами сидят :)

MrShoor 26 дек 2020 в 11:30

Это не так. Писать и рефакторить код в Lazarus давно намного удобнее. Давно перебрался на него, и пишу код почти только в нём. А вот в чем лазарус проигрывает делфи — так это в отладке. И проигрывает сильно к сожалению.

timofeevka 26 дек 2020 в 14:25

Везде свои косяки есть. Лазарус иногда может криво переколбасить код если новый обработчик события через редактор компонента делать.

MrShoor 27 дек 2020 в 05:42

У меня эта проблема как раз встречается в делфи, а не в лазарусе. Делфи почему-то иногда вставляет метод не в новое место, а разрывает уже существующее слово procedure после кажется четвертого символа. Выходит procprocedure и само собой оно не компилируется потом.

timofeevka 27 дек 2020 в 10:13

Было такое в какой то из версий. В 10.3.3 не сталкивался.

HemulGM 25 июл 2022 в 12:45

Можно подробнее. Чем именно Лазарус лучше при написании и рефакторинге?

alt78 23 дек 2020 в 10:16

Спасибо, крайне интересно было читать, особенно про решения по архитектуре. Мысленно аплодирую и снимаю шляпу!
А вот где можно линукс-версию найти? :)
приходится держать virtualbox+win7 только из-за SiT…

timofeevka 23 дек 2020 в 10:21

Если вы хотите её помучать пишите в личку или почту.

ianzag 23 дек 2020 в 10:36

Отличная статья! Буду к ней периодически возвращаться. В моменты, когда кажется что у нас в проекте наконец то настал ну совсем полный трындец и адово бессмыслие это как глоток свежего воздуха. Спасибо!

timofeevka 23 дек 2020 в 10:49

Пожалуйста! Пользуйтесь. Возможно я ещё выложу некоторые материалы и исходники наработанные по этому вопросу позже. Из этой статьи приложение было убрано, т.к. кода много.

linux_art 23 дек 2020 в 11:03

Спасибо за шикарный лонгрид. Прочитал на одном дыхании. Лет 20+ назад начинал писать на Delphi 2. Такая ностальгия пробрала.

swame 23 дек 2020 в 11:36

Тоже стоим перед вопросом портирования Delphi комплекса под Linux.
Тоже VCL, тоже своя схемная графика с выводом через GDI+ и Direct 2D.
Но у нас более 1млн строк, более 2 тыс модулей в одном проекте, а во всех проектах порядка 4 млн,
около 20 пакетов компонентов сторонних разработчиков.
плюс использование новых возможностей языка последних 12 лет.
Плюс ActixeX, с ним, конечно, придется распрощаться.

Поэтому возможность использования Лазаря под сомнением.
Смотрели скорее на возможность перевода на FMX, crossvcl + пакеты типа
www.tmssoftware.com/site/tmsfmxpack.asp
Сделали пробное портирование на FMX. Конечно, очень трудозатратно.

Свой абстрактный слой для вывода графики в разные графические движки уже есть.

Такой подход как у вас, с дефайнами, по моему опыту, годится только для кода, который больше не планируется развивать.
После обвязки дефайнами трудоемкость модификации увеличивается в разы, и так же падает мотивация что-то менять в этом коде.
Либо выносить все вызовы в абстрации, а платформо-зависимую часть в отдельные модули. А это уже не месяц работы.
Так что для себя я рассматриваю только перевод на всего проекта на другую платформу.
Но трудоемкость этого, если оптимистично, то 10 человеко — лет.

Наши заказчики электроэнергетики, году так в 18 многие спрашивали про Linux версию, в связи с правительственными приказами,
но оплачивать разработку и внедрение желанием не горят.
По разговорам с коллегами, у которых есть Linux-версия серверной части их SCADA, использует ее только 1 их заказчик, а заказчиков у них сотни.

timofeevka 23 дек 2020 в 12:12

При портировании дефайны в основном идут в заголовках и системно-зависимом кода (например при работе с COM-объектами, для некоторых вещей аналогов в линуксе просто нет, например OPC DA). В невизуальном коде расчётных библиотек их мало. Заказчиков на линукс-версию действительно не особо много, но они есть и хотят попробовать. При таком варианте портирования как я делал фактически VCL код старый неизменен и программа дописывается в текущем режиме без разрывов. TMX компоненты мы не использовали, использовали некоторые из JVCL, так что в чём-то было проще. Я за лазарь не агитирую, но по крайней мере что-то сделать работающее получилось. Я не считал подробно сколько у меня строк именно моего кода, но модулей более 900. Трудоёмкость например нашего проекта при частичном переписывании на Java составила 3 года * 3 человека и результат по качеству так себе на выходе был — работало хуже, это без меня делали другие люди. Вообще тут в принципе основное это то, что при портировании оказалось, что можно архитектуру сохранить и использованные технические приёмы разделения программы на части.

swame 23 дек 2020 в 14:56

Вызовы чего -то типа OPC DA в зрелой архитектуре должны быть скрыты в транспортном уровне данных телеметрии, а реализация подключаться через IoC, и в Linux версии соответствующие модули просто не подключаются к проекту, и должна быть замена на что-то другое. Тут дефайнов не должно быть вообще, разве что в dpr. TMS компоненты появились из-за компонентов деревьев, если в Лазаре вроде бы можно использовать VirtualTreeView, то в чистом FMX подобного нет

Вызовы opc da и так в отдельной библиотеке сделаны. Я использовал и в delphi и lazarus компонент virtualtreeview для редакторов свойств, сигналов. Удобная штука.

e_v_genius 23 дек 2020 в 12:54

Автору огромное спасибо!
Интересный опыт и полезный разбор проблем.

timofeevka 23 дек 2020 в 14:22

Пожалуйста!

mSnus 23 дек 2020 в 16:21

Огромное спасибо за статью, добавил в закладки! Предстоит перенос проекта из D7 во что-то более современное, даже не представлял, с какого конца взяться. Начну с UTF8 BOM))

Самое страшное в вашем случае (как и в моем) — переход с AnsiChar на WideChar. Это зачастую может быть даже сложнее — с однобайтной кодировки string в D7 перепрыгнуть в двухбайтную современную.
PChar тоже не подарок — повсеместно раньше использовался как аналог PByte (возможно его и не было те времена). А это всё низкоуровневое общение с памятью и куча проблем со старыми/сторонними компонентами

Я делал такое в дельфи. Сначала был переход на widechar вместе новой версией делтфи — там строки стали двухбайтные по умолчанию.

Самому Delphi (новому) без разницы в каком формате модули, UTF или не UTF. Данное замечание касалось только Лазаруса.

Спасибо! Я с Дельфи не работал лет 20 и наивно надеялся, что нынче проект тех времён довольно легко откроется и скомпилируется, или хотя бы есть автоматические инструменты миграции… конечно же, и RAD Studio, и Lazarus обругали меня нехорошими словами, а уж BDE+MS SQL выдали вообще нечего совсем нецензурное. Так что очень рад любым советам)

BDE в новых Delphi давно отстутствует.
Но некие энтузиасты еще извращаются и ставят его:
https://it-blackcat.blogspot.com/2019/12/walking-dead-bde-in-delphi-10-3-3-rio.html

Честно говоря, если вы не глубоко разбираетесь в тонкостях языка и всех компонентах вашего проекта, то лучше и не соваться в эти миграции. Т.к. проблемы конечно будут и нужно уметь их решать и переписывать те части проекта, которые того требуют. Иногда это довольно существенные изменения, поэтому, конечно, лучше мигрировать проект чаще (например, наш путь был Delphi 7 — CodeGear Delphi 2007 — Delphi XE — Delphi XE 3 — Delphi XE 7 — Delphi 10.1 — Delphi 10.3 — Delphi 10.4.1), т.е. мы мигрировали все проекты (с общей кодовой базой) на новые версии примерно через год после выпуска каждой из этих версий, иногда чуть раньше, пропуская некоторые версии, которые казались нам нестабильными или совсем уж бесполезными с точки зрения конкретно наших проектов).

Кстати как сейчас версия delphi 10.4? Я тоже не сразу перехожу на обновление, только после проверки.

Hazactam 27 дек 2020 в 00:19

Мелкие шероховатости, конечно, есть. Но так все отлично работает. Юзаю 10.4.1, правда

Последняя официальная — 10.4.1 (вышла в сентябре 2020). Готовится к закрытому тестированию 10.4.2 beta


OCTAGRAM 8 янв 2021 в 04:24

А почему вы думаете, что таких инструментов нет?

У этой конторы есть инструмент Duster: gdksoftware.com/services/delphi-upgrades-and-updates

Ещё вроде бы Oxygene пытается расти за счёт того, что переваривает код на Delphi. Oxygene наиболее известен как Delphi Prism для .NET, а самая популярная цель у него JVM, но и натив тоже есть. Правда, с трассирующей сборкой мусора, кроме макос. На макосе родной ARC, остальные платформы, к сожалению, обделены. Беда какая-то с ARC в Паскалях.


mSnus 8 янв 2021 в 18:12

Спасибо большое!!! Буду смотреть


mSnus 9 янв 2021 в 13:00

Насколько я понял, это компания, которая предлагает сервис по переносу проектов в новую версию D, за деньги. А открытых инструментов у них нет.

Или я что-то не так понял?


OCTAGRAM 9 янв 2021 в 13:02

Да, и их инструмент Duster стоит денег


В общем случае да, но есть нюансы. В Windows 10 insider build 17035 появилась возможность установить UTF-8 как 1-байтовую кодировку по умолчанию (Beta: Use Unicode UTF-8 for worldwide language support). В этом случае обязательно нужно перекодировать исходники *.pas в UTF-8, иначе словите глюков. Вот тут обсуждали: https://en.delphipraxis.net/topic/2556-test-your-product-with-utf-8-beta-setting/


Ужас какой. Я вообще не слежу ни за какими Insider Build'ами, т.к. глюков и в официальных Windows-релизах хватает.
Я надеюсь, что никто в здравом уме (по крайней мере среди наших пользователей) ни за что и никогда не будет пользоваться данной опцией.


OCTAGRAM 8 янв 2021 в 04:29

Я даже специально себе поставил полтора года назад, именно, чтоб глюки ловить и пристреливать на месте.


Исходя из вашего опыта использования Code Typhoon, как там обстоят дела с лицензиями у различных компонентов? Их там очень много, но все ли идут с лицензией, например, на статическую линковку и использование в коммерческом ПО без ограничений?


Mpl там в основном и все с исходниками. Линкуются статический они все. Это просто сборка, в которую вставили наиболее употребительные компоненты. Из того, что мы использовали коммерческого и купили ещё давно был tee chart pro. Про него написано.


Большое спасибо автору и за информацию, и за программу!
Учились мы на ней в универе, сначала МВТУ 3.5, затем 3.7. На мой взгляд, достаточно простая, легкая (в смысле размера на диске) и ничего лишнего. А чего не хватало — решалось блоками кодов.


Приятно слышать что ту версию тоже знают. Я над ней работал тоже. Вставлял туда свой новый язык программирования и систему рисования видеокадров, из которых потом вырос SimInTech.

Многия «открытия» автора описаны в ВИКИ на лазаревском сайте. Но статья интересна именно этими «открытиями», за что автору всяческие респекты :)

ИМХО, если начинать новый кроссплаформенный проект, то лучше сразу в Лазаре с использованием родных компонентов. Это поможет избежать кучи директив условной компиляции. Старый проект, конечно, да — придется пилить с кучей тестов

А вы в проекте использовали дженерики? Если да, то как сейчас с ними обстоят дела во FreePascsl? Раньше у него была своя реализация дженериков. Сейчас же я смотрю, что появился вариант дженериков совместимый с Делфи.

Использовали. Во FreePascal они поддерживаются. Но есть небольшая разница с Delphi: нельзя делать вот так: function(): TArray<ну например Double> — надо тип указать сначала с дженериком, а потом уже его указывать в декларации функции.

В FPC более продвинутые дженерики, чем в Делфи. Впрочем, режим Делфи совместимости присутствует тоже.
Можно тут обсуждение посмотреть:
www.sql.ru/forum/1263410-a/lazarus-dzheneriki
www.sql.ru/forum/1332221-a/generics-a-sobstvenno-kak-sravnit-elementy-t


OCTAGRAM 8 янв 2021 в 04:42

Размышления на эту тему привели к тому, что использовать штуки типа Wine надо, но без WinAPI. То есть, всё тот же Win-Delphi с рабочим механизмом bpl и внешними dll. Но без тормозов файловой системы из-за перегонки всего трафика через wineserver. И нормальный GUI.

Вроде бы это надо многим, и если этих многих собрать в одном месте, можно вместе начать что-то решать в этом направлении. Но как же их собрать?

Такое в норме, в понимании, делает государство в лице своих научных институтов. А ещё у нас ведь импортозамещение. Вот только в науку финансирование импортозамещения не идёт. Там ИИ и блокчейны, инновации, а подмести в своём дворе — это же не инновационно. Поддержка импортозамещения идёт только частникам. А чтоб частник в науку вкладывался, он должен быть монополистом, и тогда у него есть средства, и его наукоёмкая деятельность критична для сохранения монополии. Но монополисты все на Западе. Так и остаётся проект на уровне идеи.


timofeevka 8 янв 2021 в 17:19

Текст статьи по ссылке не доступен к сожалению. У нас как бы капитализм сейчас. Есть требование заказчика — заставить программу работать на Astra Linux например. Дальше от этого требования мы выбираем наименее трудозатратный способ проведения работы, описанный в статье. Мы вот те самые частники и никто нам государственных денег напрямую не дает. Мы получаем финансирование от наших заказчиков — отраслевых НИИ (и частных фирм также) за техподдержку и доработку модулей к ПО, причём в общей стоимости работ то, что перепадает нам это очень небольшая часть.
Вообще денег в государстве хватает, только их надо давать по адресу.

Все статьи доступны. Для первой ссылка справа рабочая.

> У нас как бы капитализм сейчас.

При капитализме придумали вкладываться в инфраструктуру, а потом Кейнс обосновал, почему государство должно так делать, апеллируя к понятию общественного блага.

Но если государство забыло о своём существовании, то много частников могли бы собраться.


timofeevka 9 янв 2021 в 13:12

Ну я просто в курсе как именно люди работают над например программами cfd расчётов во ВНИИЭФ. Они деньги получают не за лицензии, а за НИР и продавать от себя разработку другим фирмам не могут. Поэтому у нас то, что было внутри НИИ сделано за их пределы как правило не выходит. А чтобы продать кому то лицензию или исходник надо организовывать независимое юридическое лицо. Такие конторки есть как

правило при Вузах, мы сами некоторые работы так проводили, но там половину забирает себе вуз и мало остаётся, поэтому для нормальной коммерциализации надо или свою или дружественную фирму иметь. Это весьма большая проблема нашей экономики, из за которой собственно мозги утекают и разработки загибаются.

OCTAGRAM 9 янв 2021 в 13:28

Конкретно эту разработку я отношу к общественному благу, и проблемы коммерциализации к ней не относятся. Она лучше всего сработает, если, когда вам как разработчику понадобится компонент, он был сделан на этой платформе и поэтому переносился на линукс, а чтобы максимизировать шансы, что чужой компонент на этой платформе, она должна быть всем доступна. Коммерциализировать можно что-то с одним сливом в раковине. А тут благо получают производители операционных систем, процессоров, программного обеспечения, и основной вклад этого блага в том, что и все остальные его тоже получили.

Задача государства — идентифицировать дефицитные общественные блага и восполнять их. С этим в IT некоторые затруднения, и не только у нашего государства.

> Задача государства — идентифицировать дефицитные общественные блага и восполнять их. С этим в IT некоторые затруднения, и не только у нашего государства.

Согласен с вами на все сто процентов. Меня не покидают похожие мысли.
Возможно, в деталях я вижу другие блага, но в общем, они очень похожи.
Например, я все время думаю, почему в нашем государстве не уделяют внимание разработке собственных средств производства в ИТ?

Ведь **любая среда разработки (IDE) + язык программирования — это средства производства в ИТ.** Они первичны, а уже на основе IDE+ЯП пишутся платформы, на основе платформ пишется ПО или конфигурация. Это как в машиностроении, без станков вы не произведете ни одного механизма, а производители станков, они же владельцы средств производства, все на Западе.
Почему наше государство не работает над созданием собственных средств производства в ИТ? Я бы с удовольствием в этом поучаствовал.

Вы можете поучаствовать например в добавлении поддержки процессорных архитектур в GCC или FPC, но скорее всего будете делать это за свой счёт. У нас выделяют деньги на разработку компиляторов для собственных процессоров, но это вещи нишевые и за это получают зарплату в профильных НИИ (НИИСИ РАН например). Из примеров успешной отечественной разработки в области языков программирования и сред разработки можно упомянуть наверное Kotlin + среда разработки IntelliJ IDEA, который был выкуплен Google-м и от которого они зарплату и получают собственно — наше государство им не платит ни копейки и живут они за счёт своего популярного и качественного продукта.
Та система (ПК SimInTech) которой мы занимается и особенности портирования которой были описаны в статье развивается давно, является платформообразующей в области расчётов динамики систем и активно используется организациями атомной отрасли и ВПК и деньги мы получаем от них за лицензии, доработки и техническую поддержку программного комплекса.
Было бы конечно просто замечательно если бы на разработки подобного рода выделялось прямое постоянное государственное финансирование, так как это позволило бы вести разработку с полностью открытым исходным кодом, но это во всём мире не так — над ядром Linux в основном работают сотрудники крупных корпораций, которые пишут доработки под свои нужды для своих изделий.
В общем короче: кто хочет делать — сначала делает и внедряет, а потом если это оказывается нужным, то это рано или поздно заметят и купят в той или иной форме.

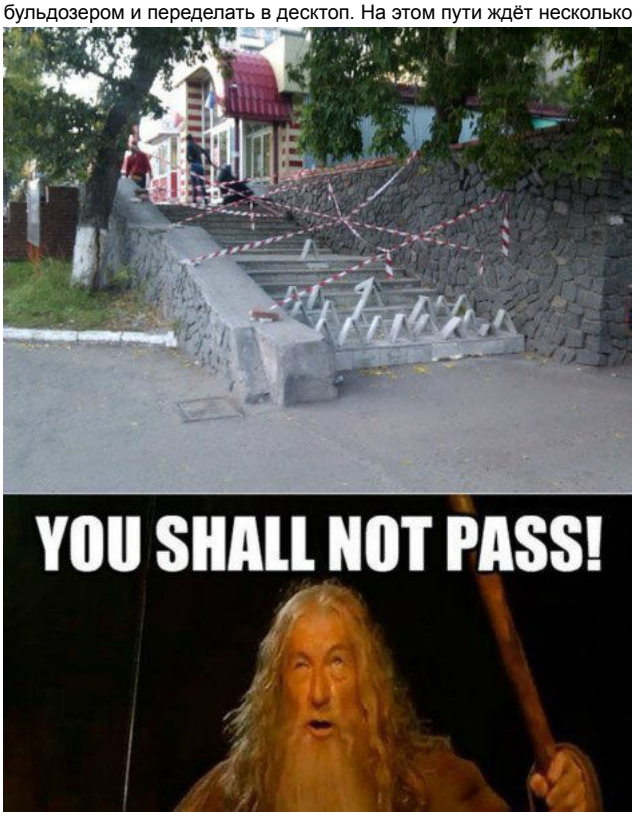
> Из примеров успешной отечественной разработки в области языков программирования и сред разработки можно упомянуть наверное Kotlin + среда разработки IntelliJ IDEA, который был выкуплен Google-м и от которого они зарплату и получают собственно

Так вот и обидно, что зарплату получают от Гугля, а не например от Ростех или Роснано. Нам (России), что не нужны собственные среды разработки и языки программирования?
Когда мы поймем, наконец, что без собственных ИТ-средств производитва мы так и будем на задворках этой индустрии находиться, а уровень нашей цифровизации будет на 100% зависеть от западного дяди Сэма. Или может государство думает, что мы такие богатые, что частным образом поднимем отечественный ИТ с колен? Так не бывает. Государство должно понять, что нам просто жизненно необходимы собственные средства разработки для Эльбрусов, АстраЛинукс и т.д… и государство должно вкладывать в это деньги, а не частный капитал.

Спасибо Вам огромное за статью!
Вы затронули очень важную для меня тему разработки кроссплатформенного нативного софта. Ну не могу я без боли смотреть на потуги web-разработчиков и старания джавщиков, которые пишут на третьем-четвертом уровне надстройки над нативом. У меня просто не поднимаются руки писать софт для интерпрайз на web-платфоме.
Может конечно это мои «тараканы», но я все равно считаю, что более мене серьезный софт для бизнеса (B2B) должен быть нативным, не зависимо от ОСи или архитектуры CPU.

Во многом согласен, делать программу расчёта динамики с тесной интеграцией с Си и ФОРТРАНом и требованиями к скорости рендеринга видеокадров лучше на чём-то, что компилируется в нативный код. А так инструмент определяется задачей.

На счёт веб я размышлял, и мой анализ свёлся к тому, что в веб гонит потребность кроссплатформенности, для которой раньше было достаточно LCL, а теперь в веб никаких LCL нет, а только сам веб кроссплатформенным и получился. Если хотим развернуть эти потоки вспять, надо веб разровнять бульдозером и переделать в десктоп. На этом пути ждёт несколько препятствий:



- Service Workers во множестве запускать из одной вкладки — моветон, так что нужны зелёные потоки, работающие в одном реальном потоке браузера.
- Обработка исключений в WebAssembly сделана сейчас из рук вон плохо. Исключение сдирает весь стек Wasm, так что ничего ценного (читай: RAII) там хранить нельзя, и это ещё один повод делать зелёные потоки.
- Текущие трансляторы в Wasm в гораздо большей степени следуют спецификации Wasm, чем пытаются быть конгруэнтными desktop. И вообще с ними больше шансов потерять в производительности даже по сравнению с JS, чем выиграть. Например, ВКонтакте и другие сайты грузят только модули, которые нужны, по мере необходимости, и кешируемые независимо, а EmScripten заточен под большие статически собранные модули, обновляемые, загружаемые и компилируемые целиком. С SDL2 под EmScripten экспериментировал, так там нет ожидания события, а есть только опрос. Если вкрутить опрос в быстрый таймер, нагрузка 100% CPU на перерисовку вне зависимости от того, делаю ли я что-то в UI. Если вкрутить опрос в небыстрый таймер, получается слайд-шоу. SDL2 по-любому где-то ставит свои обработчики на клавиатуру и мышь, но не даёт подкопаться к ним, делать работу только когда она появляется. Да любой Ангуляр лучше работает, чем SDL2 под EmScripten.
- Asincify в EmScripten не поддерживается, а и раньше был как-то криво сделан, что вызывало раздутие кода до 10 раз. Предположительно, кривизна связана с тем, что EmScripten следует спецификации Wasm, то есть, принимает аргументы и размещает локальные переменные средствами Wasm. Такой контекст тяжело сохранять и воссоздавать, а ещё продолжать с любой точки останова.

- Emterpreter же способен реализовать потоки, переживающие возврат управления браузеру, но супер тормозной. Я убеждён, что можно реализовать промежуточный вариант, не супер тормозной, но и не раздувающий код в 10 раз. В идеале можно было бы просто взять Continuation Passing C, но…
- AdaMagic реализует адские исключения либо через setjmp/longjmp, либо через исключения C++, и оба способа не поддерживаются в CPC.
- CPC преобразует только код, размеченный особым соглашением о вызове, о котором AdaMagic не знает, а надо, наоборот, преобразовать всё, кроме редких исключений.
- CPC реализует асинхронизацию корректно, но очень не оптимально. Одна функция разбивается на несколько по всем возможным линиям разреза. В прологе преобразованных кусочков загрузка локальных переменных и освобождение памяти, в эпилоге выделение памяти и выгрузка локальных переменных. Это невероятное давление на менеджер памяти.
- Кроме того, в Wasm косвенные вызовы расходуют место в таблице косвенно вызываемых функций, а CPC в силу своего принципа работы производит их в изобилии. Представляется возможным сделать преобразование вида while case, чтоб одну исходную функцию преобразовывать в одну преобразованную.
- Динамически загружаемый модули в EmScripten есть, но с учётом всех особенностей ведут они себя совершенно неконгруэнтно desktop. Загрузка динамическая, которой на desktop нет, и надо ещё смотреть, как, например, таблицы косвенно вызываемых функций распределяются при подгрузке модулей, там наверняка ещё проблемы вскроются.

В общем, все проблемы, какие есть, представляются решаемыми. И если поверх веб сделать среду исполнения, похожую на desktop, то можно будет писать программы так, чтобы на desktop они могли развернуться по-полной, а в веб просто работали в более стеснённых условиях.

Подготовить веб — первое, что я хочу сделать для Objective PE.

Если честно, я мало что понял из написанного, наверное квалификации не хватает, не заныривал я так глубоко. Но детали для меня сейчас не так важны. Я не могу понять другое.

> И если поверх веб сделать среду исполнения, похожую на desktop, то можно будет писать программы так, чтобы на desktop они могли развернуться по-полной, а в веб просто работали в более стеснённых условиях

Зачем надо переделывать кривой web и ставить на нем новую надстройку?
Почему бы просто не отказаться от web, а сделать новую платформу вместо web, которая будет поддерживать как кривые стандарты weba для совместимости, так и новые нативные стандарты для разработки desktop приложений нового поколения?

> новую надстройку

Эта новая надстройка через перекомпиляцию переносима на уровне исходников и, можно ещё сделать, RPC, с

> новые нативные стандарты для разработки desktop приложений нового поколения

Вопросы к тем, кто реализовывал кросс-платформу с помощью Delphi или Lazarus.

На Astra Linux SE (Smolensk) запускали своё ПО? Работает ли ПО, созданное на Delphi, в этой ОС? Пробовали вести разработку на Lazarus в самой Astra Linux SE?

У нас сейчас стоит схожая задача — сделать наше ПО, разрабатываемое на Delphi, кросс-платформенным. Причём, скажем так, приоритетная не-Windows ОС — именно Astra Linux **SE**. Я начал читать их «РУКОВОДЯЩИЕ УКАЗАНИЯ ПО КОНСТРУИРОВАНИЮ» и приуныл. С первого взгляда похоже, что необходимое требование — разработка с использованием Qt. Lazarus, вроде, поддерживает Qt. Но как оно всё работает в реальности на разных ОС? Можете что-то прояснить?

НЛО прилетело и опубликовало эту надпись здесь

Можно и под qt сделать, но там надо прокладку между qt и Паскалем собирать под строго определенную версию qt и вот с этим как раз бывают проблемы. А Gtk2 есть штатно и в астра линукс и в альт и вроде как и работает нормально.

На Astra Linux я ставил и code typhon и компилировал программу, причём на версии smolensk у меня заводился код который я собирал на alt Linux 9. Также проверяли и на более новой версии Orel. Библиотека gui была использована штатная, то есть. Gtk2 с которой Лазарус работает по умолчанию. Все там завелось. Были проблемы с фортрановскими so, т.к. там требовалась определённая версия фортрана 77 для компиляции, чтобы ошибка не возникала.

А почему бы не взять Delphi для Linux? И CrossVcl или FmxLinux

Так мы сейчас и выбираем инструменты разработки. На данный момент используется Delphi Professional, а поддержка Linux заявлена в Enterprise.
На странице CrossVcl написано:

> CrossVCL doesn't work properly on some Linux distros. Here is known list:
> — AstroLinux Smolensk 1.6 because of non-standard version of GTK+ (Works fine on AstroLinux CE)

LSB туда поставить можно со стандартным GTK+?

1. Не уверен, что правильно понимаю аббревиатуру LSB
2. Со своей тестовой машиной можно, вероятно, много чего вытворять, а вот с клиентской — не факт. И учитывая, что речь идёт об Astra **SE**, лично я бы рассчитывал на минимальную возможность маневрирования.
В любом случае, саму Астру нам ещё не приобрели, поэтому ничего конкретного утверждать не могу. Вот и собираю информацию пока.

Linux Standard Base определяет двоичный стандарт на пользовательское окружение (библиотеки). Gtk+ там есть. Коммерческий софт компилируется под LSB, а потом запускается в самых разных дистрибутивах, используя пакеты для поддержки LSB.

У меня учетная программа, написанная на Delphi7.
Однажды встал вопрос использования и под Linux. Пробы показали что вполне приемлемо работает через wine, за исключением чисто виндовых зависимостей, которые были не критичны и от которых я просто отказался, и сейчас даже и не помню что это было, ибо сейчас и на windows эти функции не нужны.

Единственно, не помню как там у меня с печатью обстоит, вроде не было нужды печатать из Linux.

В качестве Linux использовались различные разновидности Ubuntu, Lubuntu 12.4, 16.4, какие-то разновидности линукса с Asus EEE PC и тому подобное.

Так же был написан на том же Delphi7 информационный киоск, работает как на винде так и на линухе. Т.к. работает в режиме фулл-скрин — операционка вообще без разницы.

Был еще один затык на киосках с линуксом — т.к. программа стартует так сказать из автозагрузки, то получается что гуй и программа уже запустились, а сеть — еще нет. Можно было позадуряться с зависимостями запуска, но я просто влепил паузу, в конфиг, и если это линух — то паузу включаю, а если винда — то пауза не нужна.


kisskin 8 июл 2021 в 23:58

есть вопрос (в рамках того же импортзамещения) — в программе под Win используется TWebBrowser для отображения html и Word/Excel для отображения(100%-е соответствие не нужно) и конвертации doc/xls в html/txt. Как-то это решается под linux относительно простыми способами? В принципе, есть ли какие-то аналоги (CreateOLEObject('Word.Application')) программно открыть и исправить документ типа doc?


OCTAGRAM 9 июл 2021 в 05:45

OpenOffice и LibreOffice реализуют UNO, в котором есть и посредственное внутрипроцессное взаимодействие (в отличие от COM, всё через аналог IDispatch, как в OLE Automation), и через него же — удалённое.

Публикации

- 
  ru_vds 23 часа назад

  ### Анализ задачи с собеседования в Google: конь и телефонные кнопки

  Средний

  13 мин

  8K

  Обзор

  Перевод

  +42

  20

  +20

- 
  sheknitrtch 21 час назад

  ### Jujutsu — новая система контроля версий

  Простой

  2 мин

  12K

  Обзор

  +31

  23

  +23

- 
  dlinyj 4 часа назад

  ### Апгрейд компьютера паяльником. Часть II

  Средний

  5 мин

  2.1K

  Роадмэп

+29

14

+14

- 
ayushindin 2 часа назад

Препарируем менеджмент гигантов Кремниевой долины. Четыре причины прочитать «Transformed»

5 мин

368

+28

0

- 
Realife 3 часа назад

Всё о цифровых методах восстановлении аудио у себя дома. От ручных методов до ИИ

Простой

7 мин

800

Туториал

+23

6

+6

- 
ta2024 5 часов назад

8 лет одиночества. Как небольшой отель один боролся с локдауном, мошенниками и агрегаторами и вырос в 8 раз

Простой

9 мин

887

Кейс

+20

13

+13

- 
Byurrer 1 час назад

CI/CD для начинающих: деплой React-приложения без боли

6 мин

746

+17

0

- 
pa-kond 5 часов назад

To bind or not to bind: как мы управляем identity корпоративных «Маков»

12 мин

948

+17

2

+2

- 
Spacelight 4 часа назад

Не можешь победить — автоматизируй. Упрощаем рутину в аналитических задачах

Простой

6 мин

802

Кейс

+16

0

- 
ivntv 5 часов назад

Как я прошел через замогильный холод и адское пекло — и вернулся с дарами. Байки монтажника

4 мин

1.1K

+16

3

+3

Вакансии

- Программист Delphi

  от 30 000 ₽Базис-ЦентрКоломна

- Middle+ / Senior UI/UX designer

  от 1 500 $groМоскваМожно удаленно

- UX/UI-дизайнер

  до 150 000 ₽Wanted.Санкт-Петербург

- Преподаватель в онлайн-школу Трехмерное моделирование в 3Ds Max

  от 20 000 до 100 000 ₽CODDYМожно удаленно

- UI/UX Designer

  от 2 000 $FreudersМожно удаленно

Больше вакансий на Хабр Карьере