# Kolibri OS inside

Выходит раз в месяц

## #4_May-June

Тема номера:

Программирование

kosinside

Kolibri OS Inside – что это?

Журнал, рассказывающий о новостях мира Kolibri OS. Выпускается в последнее число каждого месяца.

Над номером работал – Гордон Фримен.

Сайт https://vk.com/kosinside

Журнал не зарегистрирован, бесплатен и доступен для свободного распространения



Полезные ссылки:

http://kolibrios.org

http://kolibri-n.org

https://vk.com/kolibri_os

Требуется: редактор, писатель рубрик. Более подробно на сайте.

## Содержание


**Бывший инженер Intel рассказал, почему Apple переходит на процессоры архитектуры ARM**

Переломный момент наступил в 2015 году, когда компания Intel выпустила процессоры Skylake. «Обеспечение качества Skylake было больше чем проблемой, это было ненормально плохо – считает бывший ведущий инженер Intel Франсуа Пеньоэль. – Мы слишком часто сталкивались с неприятными мелочами внутри Skylake». В сущности говоря, Apple оказалась тестировщиком новой архитектуры, которая, не соответствует стандартам качества.

**Отказ от зарядки и наушников сделает упаковку iPhone … изысканнее?**

Известный ресурс MacRumors подтвердил слухи о том, что новая линейка смартфонов лишится ЗУ и наушников, причём эта политика компании пойдёт на пользу упаковке, которая станет гораздо тоньше. В набор будут входить только кабеля с Lighting и USB-C разъёмами. Зарядку на 20 Вт можно будет докупить отдельно или использовать старую.

**Ремейк Crysis будет отложен из-за раскритикованной графики**

Благодаря недавним утечкам трейлера и скриншотов фанаты смогли сравнить качество оригинала и ремастера. По их мнению, графика ремастера местами выглядит хуже, чем у оригинала. Для полировки картинки Crytek перенесла премьеру на несколько недель, а релиз на неопределённый срок.

**MacOS на iPhone может стать реальностью**

По данным инсайдеров Apple, компания создала прототип смартфона, который может запускать дедсктопную ОС. При подключении к нему монитора он автоматически переключается с iOS на MacOS. Основное отличие от Continuum (Microsoft) и Dex (Samsung) – работать будет не многооконное расширение мобильной ОС, а полноценная MacOS, со всеми фичами и особенностями ARM-платформы.

# «Hello world» в KolibriOS по-Сшному

**Автор: ник, почта/вк**

&

Многие люди задаются вопросом: **«Можно ли написать программу на Си под KolibriOS?»** Ответ: **«Да, можно!»**, и ниже я расскажу, как это сделать.

Для написания программы, нам понадобятся:

- Компьютер или виртуальная машина с **KolibriOS** (если у Вас не установлена KolibriOS, её можно скачать с нашего сайта). Напомню, что KolibriOS для работы требует минимум 8MB RAM и Pentium-совместимый CPU.
- Флешка (если Вы пишете код не в самой Колибри).
- Компилятор TCC (**T**iny **C** **C**ompiler). Сборку для Колибри (**mini_c_dev**) можно скачать на нашем форуме. Тема с обсуждением **ktcc** (Kolibri TCC) находится здесь: board.kolibrios.org/viewtopic.php?f=45&t=565

### Немного теории

В KolibriOS существует два вида программ: оконные и консольные. Программа с GUI пишется при помощи системных функций и библиотеки BoxLib. Консольная программа пишется с использованием библиотеки console.obj. О работе с библиотеками будет рассказано в следующих статьях; сегодня мы напишем оконное приложение на основе системных функций.

Оконное приложение состоит из функции **main()**, в которой есть специальный обработчик сообщений (Листинг 1.1):

**Листинг 1.1**

```
while (!0) {
	switch (_ksys_wait_for_event(10)) {
		case 2:
			return 0;

		case 3:
			if (_ksys_get_button_id() == 1)
				return 0;
			break;

		default:
			//Здесь задаются команды, с помощью которых можно нарисовать окно.
			break;
		}
	}
}
```

В нашей программе мы будем обрабатывать только события клавиатуры и нажатия на кнопки. Их обработка происходит в цикле while при помощи switch — case.

**Разберём эти события:**

2 — нажата клавиша на клавиатуре;

3 — нажата кнопка, определённая ранее кнопка (в данном примере это только крестик закрытия программы, который является кнопкой создаваемой вместе с окном).

Более подробно о системе событий можно прочитать на нашей Wiki или в документации, поставляемой с ОС (программа Docpack на рабочем столе).

Для удобства разработки, мы вынесем команды, которые отвечают за внешний вид окна, в отдельную функцию **draw_window()**.

Теперь пришло время разобраться, как же нарисовать окно.

Рисование окна начинается с функции **_ksys_window_redraw(1)**, и заканчивается **_ksys_window_redraw(2)**. Для рисования окна, нужно воспользоваться функцией **_ksys_draw_window** (координата окна по x, координата окна по y, ширина, высота, цвет, тип окна, цвет заголовка окна, неперемещаемое ли окно, цвет рамки окна).

**Типы окон:**

0 — тип I — окно фиксированных размеров (без скина);
1 — только определить область окна, ничего не рисовать;
2 — тип II — окно изменяемых размеров (без скина);
3 — окно со скином (изменяемых размеров);
4 — окно со скином фиксированных размеров.

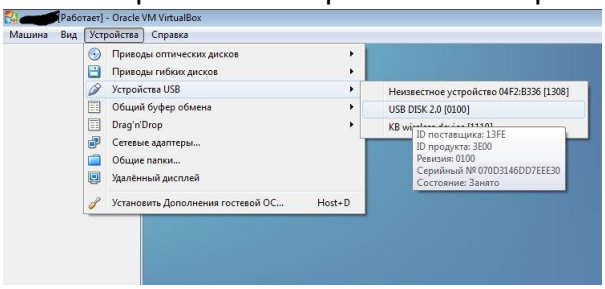
И, наконец, чтобы нарисовать текст, нужно воспользоваться **_ksys_write_text** (координата x, координата y, тип шрифта, строка, длина строки).

Теперь можно приступить к самому коду. Писать программу можно прямо в КолибриОС. Открываем Tinypad, и пишем такой код (Листинг 1.2):

**Листинг 1.2**

```
#include <stdio.h>
#include <string.h>
#include <kolibrisys.h>
#define FONT0          0
#define FONT1          0x10000000

char header[] = {"Hello World!"};

#define BT_NORMAL      0
#define BT_DEL         0x80000000
#define BT_HIDE        0x40000000
#define BT_NOFRAME     0x20000000

void draw_window () {
      _ksys_window_redraw(1);
      _ksys_draw_window(100, 100, 300, 120, 0xaabbcc, 4, 0x5080d0, 0, 0x5080d0);
      _ksys_write_text(50,30,FONT0, header, strlen(header));
      _ksys_window_redraw(2);
}

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

      while (!0) {
            switch (_ksys_wait_for_event(10)) {
            case 2:
                  return 0;

            case 3:
                  if (_ksys_get_button_id() == 1)
                        return 0;
                  break;

            default:
                  draw_window();
                  break;
            }
      }
}
```

Теперь настало время компилировать!



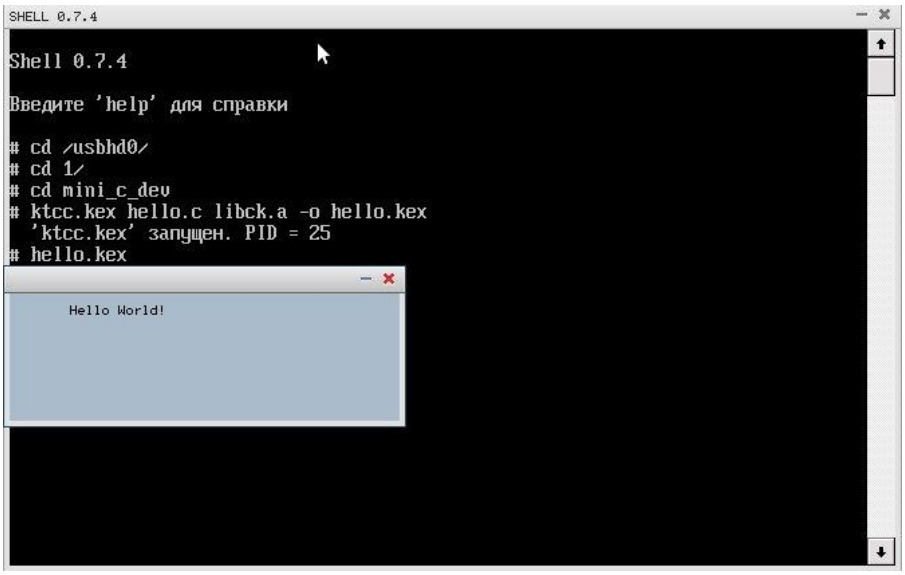

Если вы писали код не в Колибри, сбросьте его на флешку вместе с компилятором. Например, я сохраню файл под именем **hello.c**.

Для людей, которые не могут установить Колибри, можно подключить флешку к виртуальной машине (Рисунок слева).

Для компиляции, откроем в консоли (**Shell**) директорию, где лежит компилятор, и введём **ktcc.kex <имя_исходного_файла> libck.a –o <имя_готовой_программы>.**

Если всё сделано правильно, мы увидим нашу программу в той же директории. Теперь её можно запустить.

**И получим следующую картину:**




Ссылка на код и компилятор.

8 мая

Благодаря усилиям Сергея Грошевого были выявлены и в результате исправлены баги:
- CALENDAR меняет время без предупреждения
- DOCPACK неточности
- TABLE игнорирует точку с запятой
- IRC CLIENT нечитаемые вкладки
- EOLITE невидимые пункты контекстного меню
- PING перехватывает параллельные запросы.

Ваши баги вы можете отправить в тему https://vk.com/topic-48924138_41261639

12 мая

SVN r.7914
При открытии ссылок и файлов браузер WebView теперь не создает новый процесс, а передает аргумент запуска первому запущенному экземпляру, если он есть.
Например, открыл 1.html - открылся WebView с ним. Открыл 2.html - в открытом WebView добавилась вторая вкладка.

20 мая

Новая программа Beat - простой метроном. Для работы использует другую новую программу PlayNote, назначение которой очевидно из ее названия.


24 мая

Создана первая версия Delphi SDK!
Просьба протестировать: ссылка GitHub.


27 мая

Новая версия Eolite 4.35!
- порадует тех, кто запускает Колибри на реальном железе: теперь недоступные диски не отображаются в панели слева
- новая иконка системного диска
- добавлено выделение по shift+клавиши вверх/вниз и ctrl+клик

31 мая

Релиз Image transform - программы для преобразования изображений.
Программу можно использовать для выравнивания искаженных изображений изначально прямоугольной (квадратной) формы, для создания текстур и т.д.
Алгоритм работы программы: открывается изображение и из него делается текстура для библиотеки TinyGL. Пользователь задает 4 контрольные точки, которые станут углами преобразованного изображения, 5-я точка вычисляется как среднее арифметическое от этих 4-х точек. Вычисляются текстурные координаты для точек (в пределах от 0 до 1). Рисуются 4 треугольника по выпрямленным координатам. Если все устраивает можно сохранять полученное изображение.

# HTML? Не вопрос!

**Автор: Alex2003, alex00xr@mail.ru**


Всем привет! Тема этого номера – программирование. И я тоже бы хотел поделиться своим небольшим опытом по языку, который востребован практически везде – HTML.

Итак, что такое HTML? Это кроссплатформенный язык (расшифровывающийся как HyperText Mark-up Language, или на русский язык гипертекстовой разметки), изобретённый Тимом Бёрнсом-Ли в 1990 году и активно применяющийся в создании сайтов. При этом он не требует каких-либо навыков программирования и достаточно просто хотя бы потому, что создать страницу на нём можно в блокноте (КАРЛ!!!).

Для того чтобы создать сайт, нам будет нужно парочку-другую тэгов. Вообще говоря, тэги – это метки для браузера, как он должен показывать наш сайт. Или более популярным языком – это как придорожные знаки для автомобилиста, объясняющие, как он должен себя вести.

Ну что же, начнём писать!

Для начала мы объясним браузеру, на каком языке мы будем с ним общаться. Для этого открываем блокнот и пишем (Листинг 2.1):

**Листинг 2.1**

```
<html> <!-- Открывающий тег  -->

</html><!-- Закрывающий тег -->
```

Теперь ещё небольшое объяснение: тэги бывают парными и непарными. Большинство тэгов, как **<html>**, **<body>** и **<head>** состоят из открывающего и закрывающего тегов. При этом они обязательно пишутся между знаками <>, а к замыкающему перед словом приписывается слеш направо (/).

Теперь, когда мы сказали браузеру, что разговариваем с ним на HTML, пора писать саму страницу. Для этого добавляем парные теги <body> и </body> (Листинг 2.2):

**Листинг 2.1**

```
<html> <!-- Открывающий тег  -->

<body>
<!-- Здесь пишется содержимое страницы -->
</body>

</html><!-- Закрывающий тег -->
```

Внутри этих тегов мы прописываем содержимое страницы. Например, чтобы браузер вывел строчку «Начало положено!» между тегами **<body>** мне нужно написать (Листинг 2.3):

**Листинг 2.3**

```
<text>Начало положено!</text>
```

А вот как это выглядит в браузере (Рис. 1).

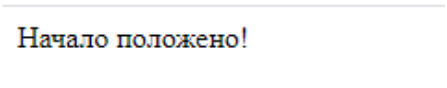



Рисунок 1

Вы скажете: «Эй, а почему текст не такого размера, какой мне нужен?». А вот для этого существуют другие парные теги (Листинг 2.4):

**Листинг 2.4**

```
<h1>Огромный текст!</h1>
<h2>Среднего размера текст</h2>
<h3>Текст нормального размера</h3>
<h4>Мелкий текст</h4>
<small>Ужасно мелкий текст!</small>
<p>Новый параграф</p>
```

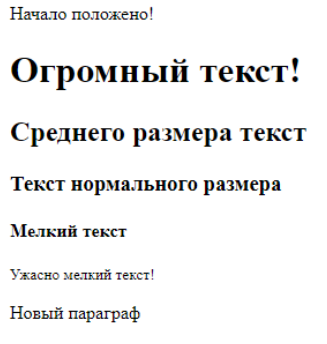



Рисунок 2

В браузере это выглядит так (Рис. 2).

Теперь мы сможем написать простейшую страницу. Но как сделать текст цветным? Или же поменять цвет самой страницы?

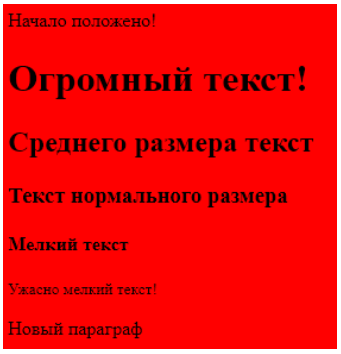



Рисунок 3

Для этого нам понадобятся атрибуты, число которых настолько велико, что понадобится отдельная статья. Сегодня же мы затронем только два: **background-color** и **text-color.**

Первый предназначен для изменения фонового цвета страницы, а второй, соответственно текста. Для того, чтобы применить эти атрибуты мы добавим в нужное место тег **style**. К примеру, если вместо **<body>** написать **<body style="background-color:#ff0000;">** то страница станет красного цвета (Рис. 3).

Точно также можно поступить с текстом, просто написав открывающий тег размера текста, отступить внутри галочек один символ и добавить **style:“color: :#ff0000;”.**

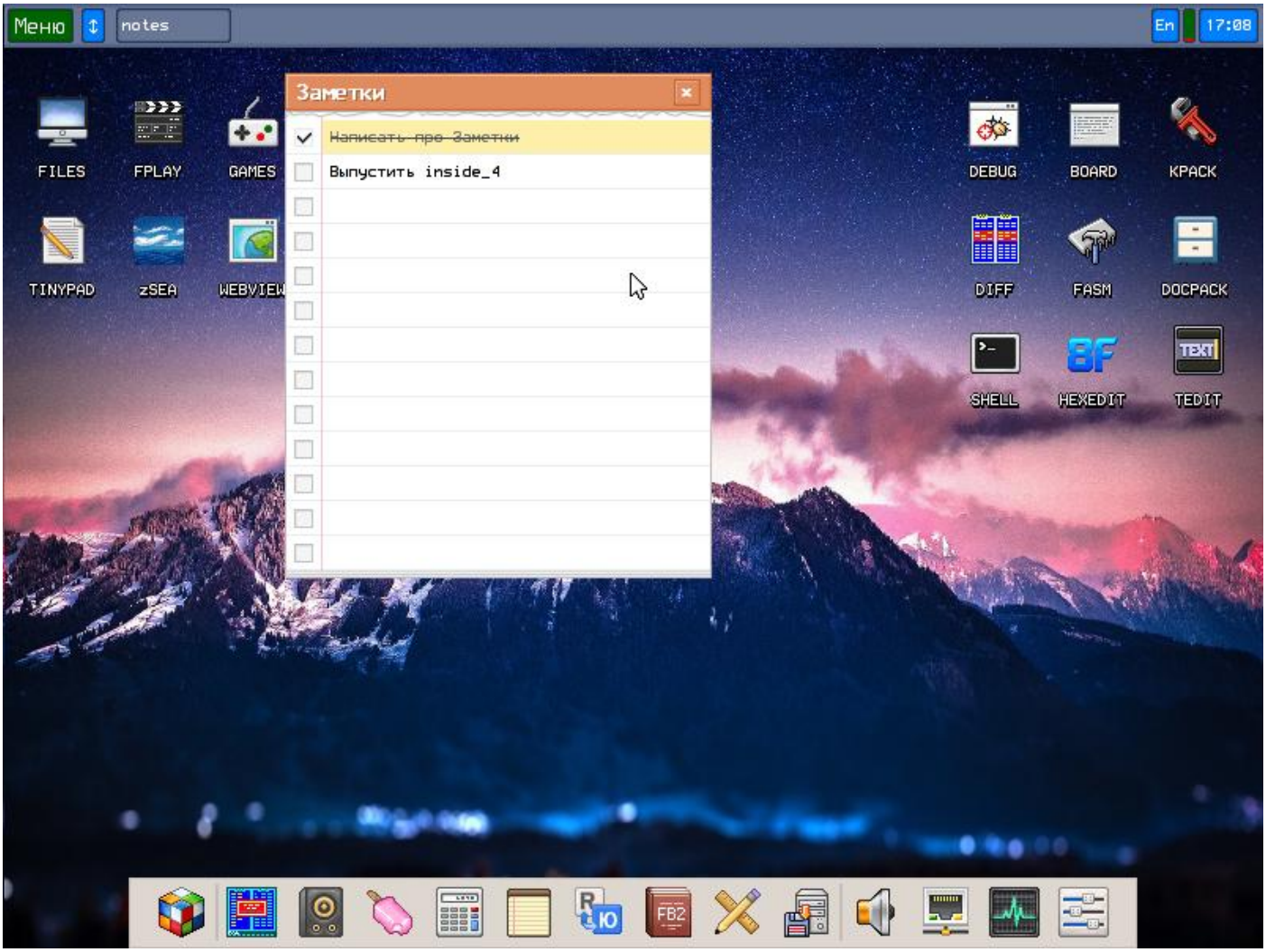
Ещё мы можем поменять начертание текста, заключив его в парные теги **** (полужирный) или **** (курсив).

И вот вы уже написали собственную страницу, запустили её в браузере, и заметили, что страница имеет какое-то странное название. Но это легко исправить: для этого пишем между **<head>** и **</head> <title>**Название страницы**</title>**.

На этом я заканчиваю.

P.S. Простите за такой слог…

# Notes by Leency – сделайте заметку!




Можно сказать, простейший органайзер, куда вы можете записать всё что угодно, чтобы не забыть в самый важный момент. К сожалению, таймера не поддерживает – было бы намного удобнее. Но зато он поддерживает скроллинг, если заметок у вас очень много. Ну а если вы поставили галочку, задача будет зачёркнута, но останется в списке.

Автор: Гордон Фримен

# Несколько мыслей о результатах WWDC 2020

**Автор: Сергей Ефременков aka TheOnlyMirage**

Приветствую всех! 22 июня 2020 года компания Apple заявила о переходе на процессоры собственной архитектуры. Теперь вся линейка продуктов от часов и роутеров до ноутбуков и десктопных ПК компании перейдёт на единую платформу. Пообещали, что произойдёт это в ближайшие два года. Такое заявление было сделано в рамках проводимого онлайн WWDC 2020. На мероприятии нам представили только один такой процессор, это был Apple A12Z. Для пользователей новых систем и их плавного перехода проделана достаточно большая работа. Программное обеспечение крупных компаний (Microsoft, Adobe и прочих), в том числе самой Apple, уже адаптировано для работы на новом железе. Также показали блок решений, который покрывает большую часть пользовательских задач, но об этом стоит поговорить в рамках отдельной статьи.

Единственное, что точно стоит упомянуть, это технология Rosetta 2 для запуска старых приложений на новой архитектуре. Остальное программное обеспечение должно подготовить сообщество разработчиков, которым дано время на адаптацию своих программ. Для разработчиков компания Apple подготовила специальный Developer Transition Kit. В его основу взят Mac mini, в который был внесён ряд изменений. Здесь используется тот самый процессор Apple A12Z Bionic, 16 Гбайт оперативной памяти и SSD-накопитель объёмом 512 Гбайт. На борту работает бета-версия системы macOS Big Sur и среда разработки Xcode 12. В Apple отметили, что в ближайшее время всё ещё продолжат сотрудничество с Intel. В чём это будет выражаться, не пояснено. Возможно, на единую платформу перейдут только потребительские устройства, а ряд линейки Pro и серверов всё же останется на процессорах Intel. Или может быть, что массовым производством новых процессоров для компании займётся Intel, так как собственных мощностей и мощностей подрядчиков недостаточно для производства необходимого числа чипов. Так ли это или есть иной подвох, мы узнаем в ближайший года.

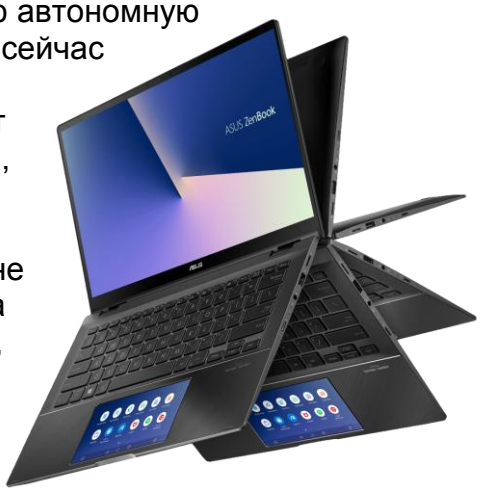

Тем не менее, с этой новостью будущее Intel пошатнулось. Можно смело сказать, что компанию ждут тёмные времена. Только недавно Core i9 с архитектурой Skylake подвёл компанию тротлингом из-за плохой реализации теплоотвода, затем прогремело множество найденных уязвимостей в архитектуре. Исправление таких ошибок обходится дорого. Где-то требуется кардинально переработать архитектуру, а где-то уже обошлись софтовым исправлением, которое существенно влияет на производительность. Многие компании пытались и продолжают делать попытки перехода на ARM архитектуру. Если у Apple получится, то мы увидим, как остальные медленно последуют в таком же направлении. Уже очевидно, что из-за стагнации в десктопном сегменте, мобильные процессоры сравнялись с ними по мощности, но при этом показывают меньшее тепловыделение и меньшее потребление электроэнергии.

Не поймите неправильно, прогресс есть и был, но он был неуверенным (как в случае APU от AMD) и недостаточным, чтобы твёрдо держать лидерство. Здесь стоит напомнить, что десктопные процессоры зачастую могут использовать те же ARM ядра внутри своих ядер. Исторически это были собственные блоки RISC архитектуры для выполнения ряда команд, но позднее компаниям стало проще приобретать готовый кусок чипа, чем поддерживать собственные разработки. Можно сказать, что произошло нагромождение слоёв в спешке, которое привело к формированию других нерешённых проблем. Из-за требований совместимости, из-за большого числа неиспользуемых команд (по статистике компиляторы и большая часть популярного ПО задействуют только треть всех команд процессора, не говоря о уже забытых технологиях и наборах команд как 3DNow!) и из-за конфликтов технологий и плохих архитектурных решений накопился большой пласт проблем. С серверами всё проще и легче, но Intel и AMD не стоит расслабляться. Если Центры Обработки Данных (ЦОДы) озаботятся оптимизацией (а такое произойдёт, если софта под

ARM-десктопы будет становиться всё больше) и выберут меньшее электропотребление, тогда удар по Intel будет ещё сильнее. Ждёт ли индустрию переход? Честно говоря, никто не уверен, но все воодушевлены такой возможностью. Возможно, кроме Intel.

Так о чём это говорит и к чему стоит готовиться? Однозначно ответить нельзя, но мы можем немного поразмышлять на данную тему. С точки зрения потребителя стоит готовиться к прохладным, но производительным, устройствам с долгой автономной работой. Мощные планшеты/телефоны и долгоживущие ноутбуки.

В остальном не всё так однозначно. Любые мощности и долгую автономную работу быстро компенсируют сложность кода ПО. К тому же, уже сейчас мощности современных устройств используются в основном для рекламы и смайликов. Смотря правде в глаза, современный софт пишется отвратительно и задорого. Что будет со старыми играми, когда переход завершится через 10-15 лет? Сейчас таким приложениям ничего не грозит, по крайней мере, пока компиляторы, среды разработки, игровые движки и фреймворки не будут адаптированы к новому железу, и пока самого этого железа нет в массовом наличии. Потом их ждёт эмуляция. Как говорится, костёр может потухнуть раньше, или же наоборот, разгореться быстрее.




Запустить другие существующие ОС на железе новых устройств будет всё сложнее. Производители всячески внедряют DRM защиту и свои пропиетарные прошивки. Тем самым свободы в мире ПО станет куда поменьше. Это будет предшествовать появлению новых ОС, учитывающих новую архитектуру. Словосочетание «новая архитектура» звучит слишком оптимистично, ведь никаких подвижек и стандартов для того не видно. Значит, всех ждёт особенность ARM-архитектуры. Разработчикам программного обеспечения стоит обратить внимание на следующее поколение ARM и особенности графических чипов в этих архитектурах. Не помешает вовремя сформировать новые стандарты, там где их очень не хватает, чтобы в будущем не получить плохой совместимости и дополнительных сложностей в разработке ПО. Даже если Apple станет инициатором начала эры, где проприетарный софт прибит намертво к железу, то нельзя отрицать, что оптимизация софта под железо это важно, но так же важно не лишать всех былой простоты и возможности выбора.

P.S. Благодарю за прочтение! Мысли бывают неправдой, а предсказанный прогноз погоды имеет возможности сбыться.

## ХРАНИЛИЩА SUBVERSION

Всего в репозиторий было внесено 8045 поправок!

Для KolibriOS существует 22 библиотеки общим весом в 19,4 МБ

# C--. Первое знакомство

**Автор: Punk_Joker**

Процесс портирования и создания средств разработки программ для KolibriOS продолжается. По наиболее активно используемым языкам программирования мы публикуем статьи. Сегодня мы начинаем рассказывать о языке C--, вокруг которого сложилось активное сообщество в 2000-е годы. Подробности под катом.

Кратко о языке с официального сайта:

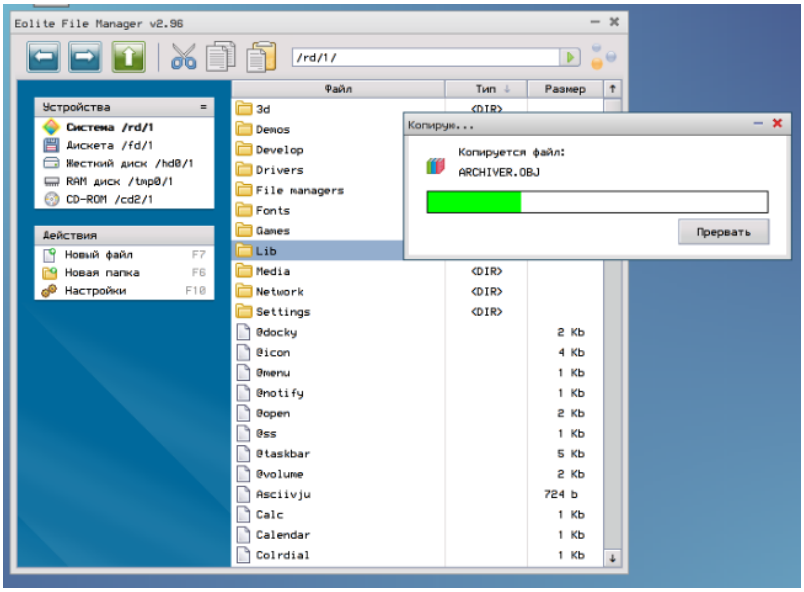
C-- — это язык программирования, занимающий промежуточное положение между ассемблером и Си. Идеально подходит для написания маленьких программ, резидентов (TSR), драйверов, обработчиков прерываний. Для работы с языком C-- необходимо знакомство с ассемблером и Си. Сейчас C-- умеет генерировать 32-битный код под ДОС и Windows *(прим. ред. а также под MenuetOS, KolibriOS)*.

Автором языка SPHINX C-- является Peter Cellik (CANADA). Последняя авторская версия SPHINX C-- v0.203 от 28.Oct.96. К сожалению, автор отказался от дальнейшего развития языка. Сам язык вместе с исходными текстами был объявлен сиротой и отдан никому в никуда. Т.е. делайте что хотите. А почему бы и не сделать? На этой страничке Вы найдете самую последнюю (но уже не авторскую) версию самого языка, библиотеки, примеры программ, описание самого языка и библиотек в формате Notron Guide и много другой полезной информации по языку C--.

После Питера Селлика поддержкой занимался Михаил Шекер, но он прекратил разработку несколько лет назад. В прошлом году исходники были выложены на Github https://github.com/jossk/c--sphinx, но мы узнали об этом только в марте 2016 года.

Было решено рассказать о данном языке по двум причинам:

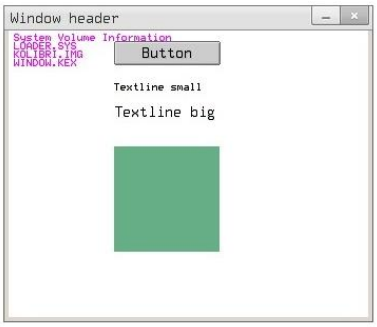
1. На нём написаны несколько популярных программ KolibriOS, среди которых и Eolite — пожалуй, самый используемый файловый менеджер (скриншот справа).
2. Относительно недавно компилятор данного языка был портирован для KolibriOS (ответ на вопрос о лицензии можно найти тут).




Немного о процессе портирования от **GerdtR**:

Итак. Сначала я переделал код под gcc. Исходники просто были написаны для Watcom C, ну и пусть не значительная, но несовместимость была. Исправил пару названий переменных(extern в ваткоме можно, видимо), и обозначение 64-битных чисел по другому. Потом уже Leency выявил, что gcc-версия совсем не ищет файлы в текущей папке, только в папке с самим экзешником. Когда и это исправил, осталось самое сложное: порт для Колибри. Ну так как под C я в КОС не писал, то была стандартная проблема — с кем линковать, да и какие инклудники брать. Выбор не широкий, menuetlibc отпал сразу, как я, покопавшись в исходниках, увидел ещё старые функции обращения к файлам (функция 56 вроде). Короче menuetlibc устарел сильно, остался newlib. Долго разбирался, как всё-таки на выходе получить kex, а не PE, помог один Makefile, где писалась дополнительно обработка objcopy. Ну а дальше уже дописывание функций, которых нет в newlib (например, stat не было, в port.c лежит его реализация). Ну и пара функций преобразования кодировок ANSI->ASCII, а точнее заглушек, потому как русских букв всё равно в строках компилятора нет. Ну и последней проблемой было, да и частично осталось вот это: из-за newlib(скорее всего, больше некому) текущий каталог устанавливается как путь до самого cmm(а обычно /rd/1), потому оказывается, что узнать, где лежит исходник компилятор не может, хоть откуда его запускай. Выход на данный момент — копировать компилятор в папку с исходниками, либо указывать абсолютный путь. Кстати, раньше символ '/' считался началом параметра, и абсолютный путь указать было не возможно, сейчас можно. Но с абсолютным путём пока не уверен, что

инклудники находить будет. Пока что-то не хочется с этим возиться, да и можно просто написать -IP="/hd1/1/my_source". В теории должно работать, на практике никто не спрашивал. Кстати… указать в форуме что ли, что параметры через '-' указывать теперь…. И при запуске без параметров, в той таблице поправить, а то некоторая путаница получается. Ладно, потом как-нибудь. Ну, вот в общем, вся история. Самое сложное было — это разобраться, откуда брать либы. Сам код довольно несложный, не в супер-порядке, но особо ковыряться в коде не пришлось.




Руководство по языку находится по ссылке http://www.c--sphinx.narod.ru/c--doc.htm. Рассмотрим пример простого приложения для KolibriOS (Листинг 3.1, скриншот слева):

**Листинг 3.1**

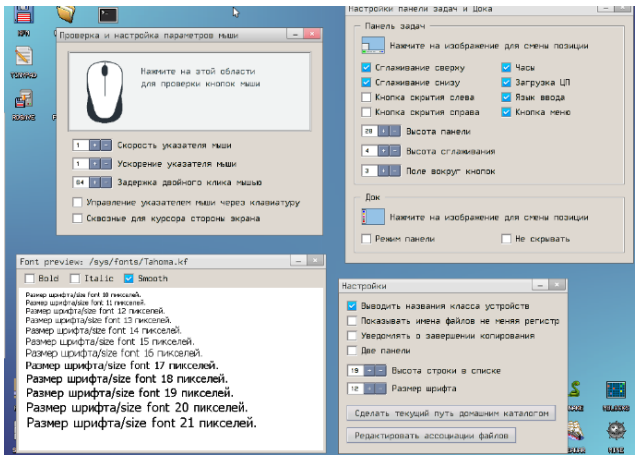
```
#define MEMSIZE 4096*10

#include "../lib/io.h"
#include "../lib/gui.h"

void main()
{
	word id;
	dword file;
	io.dir.load(0,DIR_ONLYREAL);
	loop() switch(WaitEvent())
	{
		case evButton:
			id=GetButtonID();
			if (id==1) ExitProcess();
			break;

		case evKey:
			GetKeys();
			if (key_scancode == SCAN_CODE_ESC ) ExitProcess();
			break;

		case evReDraw:
			draw_window();
			break;
	}
}
void draw_window()
{
	proc_info Form;
	int i;
	DefineAndDrawWindow(215,100,350,300,0x34,0xFFFFFF,"Window header");
	GetProcessInfo(#Form, SelfInfo);
	for (i=0; i<io.dir.count; i++)
	{
		WriteText(5,i*8+3,0x80,0xFF00FF,io.dir.position(i));
	}
	DrawCaptButton(100, 10, 100, 22, 22, 0xCCCccc, 0x000000, "Button");
	WriteText(100,50,0x80,0,"Textline small");
	WriteText(100,70,0x90,0,"Textline big");
	DrawBar(100, 110, 100, 100, 0x66AF86);
}
```

Как видите, данный код почти не отличается от кода на Си, но при этом есть возможность писать спокойно в почти ассемблерном стиле, аналогично написанию оберток для системных функций. Например, функция создания окна (Листинг 3.2):

**Листинг 3.2**

```
void DefineAndDrawWindow(dword x, y, size_w, size_h, byte WindowType,dword WindowAreaColor, EDI, ESI)
{
   EAX = 12;
   EBX = 1;
   $int 0x40

   $xor EAX,EAX
   EBX = x << 16 + size_w;
   ECX = y << 16 + size_h;

   EDX = WindowType << 24 | WindowAreaColor;
   $int 0x40

   EAX = 12;
   EBX = 2;
   $int 0x40
}
```

Чем же он привлекателен для разработчиков ПО для KolibriOS? Во-первых тем, что начать писать на нём очень легко. Это был единственный язык, не считая FASM, на котором можно было просто начать писать, не заморачиваясь с настройкой кросс-компиляции (сейчас в этом деле Си уже набирает обороты). Во-вторых, для C-- было написано множество библиотек и различных оберток над системными функциями. Среди них собственный набор элементов интерфейса и даже неплохие шрифты (скриншот справа).




Вы наверняка зададитесь вопросом: если всё так прекрасно, то в чем же проблема? Почему он не применяется повсеместно, хотя бы в рамках проекта KolibriOS?
Ответ следующий. Несмотря на преимущество в простоте освоения, при его использовании всплывают и недостатки:

- плохая оптимизация результирующего кода;
- практически отсутствуют приоритеты операций (C-- всё считает слева направо, т.е. 2+2*2=8, а не 6);
- самое главное, хоть он и похож на Си, но это другой язык программирования со своей сферой применения. В рамках KolibriOS он наиболее подходит для создания программ с графическим интерфейсом и почти не применяется в системном программировании.

Ссылка на код.

## Послесловие от автора

В общем, хочу попросить у моих читателей прощения за то, что не выпустил номер 30 мая: сбежал на время всем известной бациллы в деревню к родителям и по уши погрузился в сад и огород. Ну, зато как следует, разминаюсь и гуляю по улице без маски. Вообще, приятно, когда у тебя есть возможность куда-то свалить из душного города на месяц-другой от всего этого коронавирусного безумия.

За эти два месяца много чего произошло. Например, вы уже слышали, что Apple собирается переходить на свои, фирменные ARM-процессоры Bionic. Что же, я давно ждал такого хода, но не от яблочного гиганта точно, хотя слухов об этом до официального заявления было много. Ну, по крайней мере, теперь MacBook'и будут ещё тише (если не абсолютно бесшумны), легче и быстрее.

А теперь о главном: о журнале. Я несколько раз начинал писать и бросал свои статьи. Ну вот не выходит у меня, и всё тут! Вот сейчас я выпустил журнал, и большинство статей скопировано с HabraHabr. Вот такой вот плагиат! Вынужденный, из-за отсутствия статей.

Я вообще не понимаю общественную спячку: я предложил комьюнити разработчиков написать статьи на любую тему – программирование, дизайн, документация… В ответ – тишина. Хотя я даже создал почту для статей, мнений, вопросов моих подписчиков (kosinsideteam@gmail.com).

Ещё: один человек спрашивал, почему я делаю журнал? Ну, хотя бы потому что just for fun, да и вообще, мои действия одобрили голосованием. Или вам не нужен PR вашего проекта?

Простите, за сумбур,

ваш Гордон Фримен

P.S. Я подумываю завербовать кого-то ещё, занят по горло и не успеваю делать всё, что я планирую.