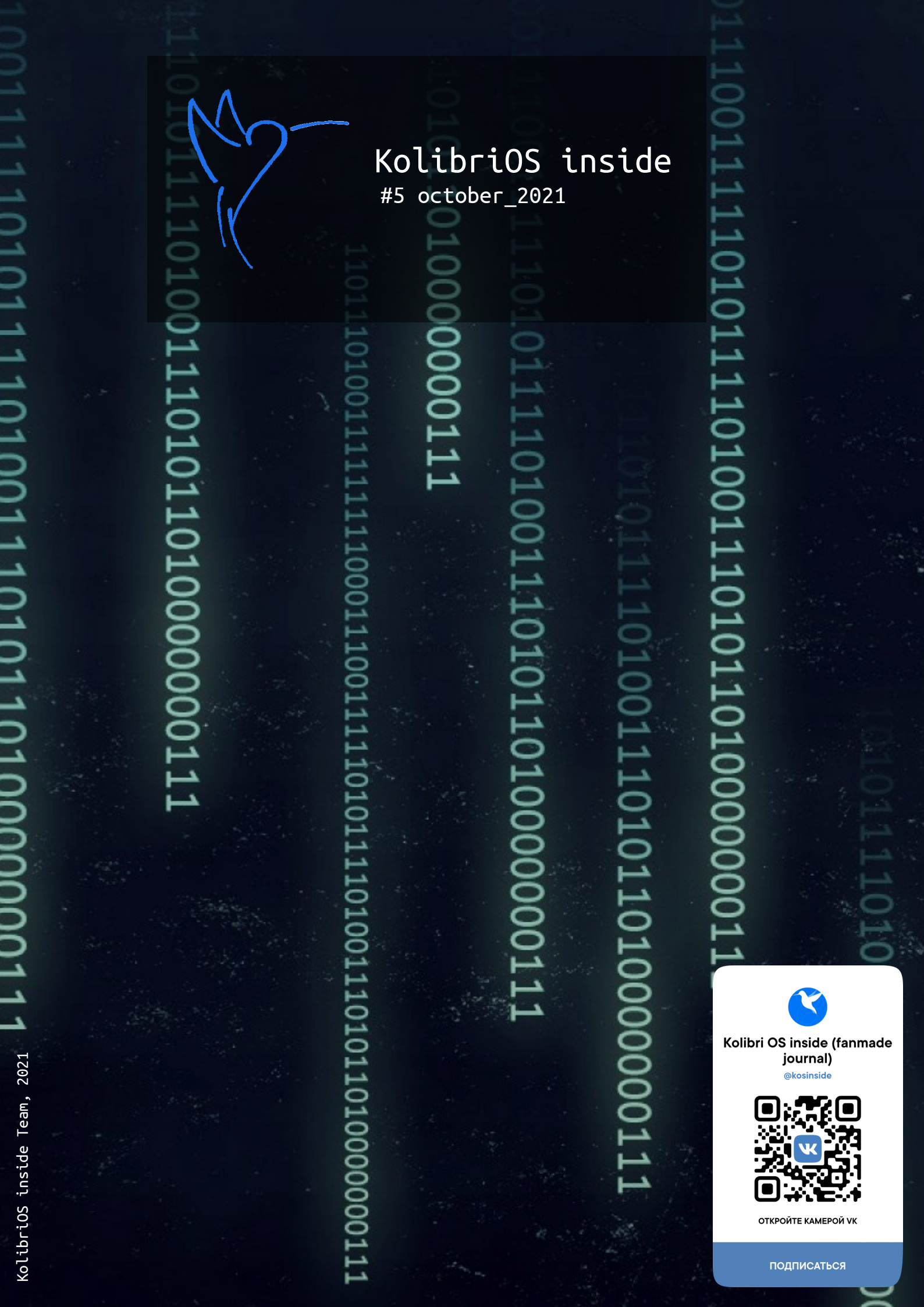


KolibriOS inside

#5 october_2021

Kolibri OS inside (fanmade journal)
@kosinside
ОТКРОЙТЕ КАМЕРОЙ VK
ПОДПИСАТЬСЯ

KolibriOS inside Team, 2021

KolibriOS inside_#5 october_2021

Новости из мира KolibriOS и не только

Выходит первого числа каждого месяца

Автор - virtual_codehack

Вёрстка, дизайн, ведение рубрик и иже с ними - virtual_codehack

Выражаем огромную благодарность rgimad, virtual_codehack за предоставленный для публикации материал.

Полезные ссылочки

kolibrios.org - сайт KolibriOS

kosinsideteam.tilda.ws/ - сайт журнала

Содержание

Стр. 3 - hi-tech - Избранные новости из мира IT

Стр. 4 - codehack - Внутреннее устройство ядра KolibriOS. Введение.

Стр. 6 - app_data - KFM&KFM2 - феникс из прошлого

Стр. 7 - history - Интервью с Serge

Стр. 11 - dev_notes - Новости из мира KolibriOS

От автора

Прости, если не оправдал твоих ожиданий в этом номере — чувствую себя уставшим. Очень уставшим.

Возможно, тебе что-то здесь понравится, что-то нет, но это всё, что я смог.

В принципе, было бы неплохо, если бы мне помогли в следующий раз — надеюсь на это.

virtual_codehack

Уязвимость в процессорах AMD: выяснились новые подробности (16.09.2021)

Кириакос Эконому обнаружил, что уязвимости подвержены чипы серий Ryzen 2000 и 3000, хотя изначально AMD указывала только на процессоры Ryzen 1000. В настоящий момент производитель обновил список процессоров — сейчас в нём значатся практически все современные модели. Проблема касалась драйвера защитной подсистемы Platform Security Processor (PSP). Используя уязвимость, злоумышленник получал доступ к гигабайтам неинициализированных страниц памяти. Это позволяло производить на компьютере жертвы различные действия, включая кражу персональных данных пользователя с правами администратора, а также PtH-атаки (pass-the-hash). Для избавления компьютера от данной уязвимости AMD рекомендует обновить драйвер AMD PSP до версии 5.17.0.0 через Windows Update либо установить AMD Chipset Driver версии 3.08.17.735 и выше.

Samsung представила быстрые и прочные карты памяти microSD EVO/PRO Plus (09.09.2021)

Производитель гарантирует, что они будут устойчивы к воде, экстремальным температурам, ультрафиолетовому излучению, магнитным полям или падениям с большой высоты. Кроме того, на данные карты работает десятилетняя гарантия от Samsung.

Линейка носителей PRO Plus предлагает очень хорошие параметры чтения и записи – скорости, достигаемые картами, составляют соответственно 120 МБ/с и 160 МБ/с. Доступны microSD на 128, 256 и 512 ГБ.

Модели EVO Plus рассчитаны на чуть менее требовательных пользователей – параметры передачи здесь не превышают 130 МБ/с. Емкости у этих карт очень похожи, за одним исключением – в продаже также доступна еще меньшая версия, позволяющая хранить 64 ГБ данных.

Microsoft представит Office 2021 вместе с новой Windows 11 (19.09.2021)

Пятого октября вместе с новой версией Windows будет также выпущен офисный пакет Microsoft Office 2021. Пока не известно ни о редакциях пакета, ни о том, будет ли он продаваться в связке с Windows 11. Цена также неизвестна, и будет наверняка представлена только в день выпуска офисного пакета.

Что нового будет в обновленном «Офисе»? Сообщается о поддержке темного режима (помогает сохранить зрение). Кроме того, интерфейс станет удобнее, и приобретёт стиль ОС. Появится полноценный функционал для пользователей с ограниченными возможностями. Также заявлены более удобный поиск и высокая скорость работы. Разумеется, это не все нововведения — скорее всего, будут и более существенные изменения.

Apple экономит на памяти в iPhone 13 (29.09.2021)

По непонятным причинам, корпорация решила сэкономить на чипах ОЗУ в новой модели смартфона и будет использовать устаревшую LPDRR4X от Hynix, объёмом «всего» в 4-6 гигабайт. Это как минимум странно, поскольку уже есть LPDDR5 с максимальным объёмом в 18 ГБ

Материал взят с сайтов overclockers.net, tech.mail.ru и группы ВК «Мой Компьютер»

Внутреннее устройство ядра KolibriOS. Введение.

Автор: rgimad aka gimrus99

Приветствую, дорогие читатели. Наверное, вы заметили, что большинство статей, так или иначе касающихся KolibriOS, посвящены прикладному софту и его разработке. Материалов же, связанных непосредственно с внутренним устройством ядра Kolibri значительно меньше. Именно поэтому я и решил написать данную статью — наконец-то разобраться и популярно объяснить, что же находится в самом сердце Колибри…

Не так давно, dunkaist нарисовал (а точнее, сгенерировал) замечательную картинку (на следующей странице сверху), предлагаю начать с нее. На ней изображен граф зависимостей ядра. Говоря проще, здесь показана взаимосвязь файлов исходного кода — какой файл от кого зависит, то есть, кто кого include'ит (картинка в качестве: https://board.kolibrios.org/download/file.php?id=9224).

В самом центре стоит kernel32.inc — этот, по сути заголовочный файл просто include'ит в себя почти все подсистемы ядра. Чуть правее и ниже расположен kernel.asm — это самый главный файл ядра. Он include'ит вышеупомянутый kernel32.inc и еще некоторые файлы. Кстати, заметьте, что .asm файл только один (kernel.asm), а остальные .inc . Таким образом, при компиляции kernel.asm, компилятор FASM, выполнив все директивы include, получит один большой-большой ассемблерный исходник, который и будет откомпилирован.

Итак, ознакомившись с общей структурой, перейдем к самому коду. Наверное, вы могли заметить, что в начале всех файлов ядра располагаются строки вида $Revision: 5363 $. Что же это значит? Сейчас объясню. Для всех файлов кода ядра в SVN репозитории установлено свойство svn:keywords со значением Rev. Это означает, что при коммите, SVN будет подставлять в файлы с таким свойством номер последней ревизии, в которой они были изменены. К примеру, если в файле написано $Revision: 7365 $, это значит что последний раз он был изменен в ревизии 7365. Но, зачем же все это нужно? — спросите вы. Дело в том, что в macros.inc из исходников ядра, подключающемся первым, определён макрос fasm'a $Revision, который вычисляет максимум из всех мест, где он встретился (то есть во всех файлах, прямо или косвенно подключаемым к kernel.asm). Этот самый максимум и является номером самой свежей ревизии ядра (не драйверов). В конце kernel.asm он присваивается переменной __REV__, ну а boot/bootstr.inc включает значение __REV__ в начальную выводимую строку на синем экране загрузки.

В ядре колибри широко используются макросы, благо ассемблер FASM предоставляет достаточно мощный макросный движок. Рассмотрим наиболее часто используемые макросы. В файле proc32.inc содержатся макросы для определения и вызова процедур. Это дает возможность использовать в коде такого вида конструкции (Листинг 1.1):

Листинг 1.1

```
proc enable_irq stdcall, irq_line:dword

…

endp
```

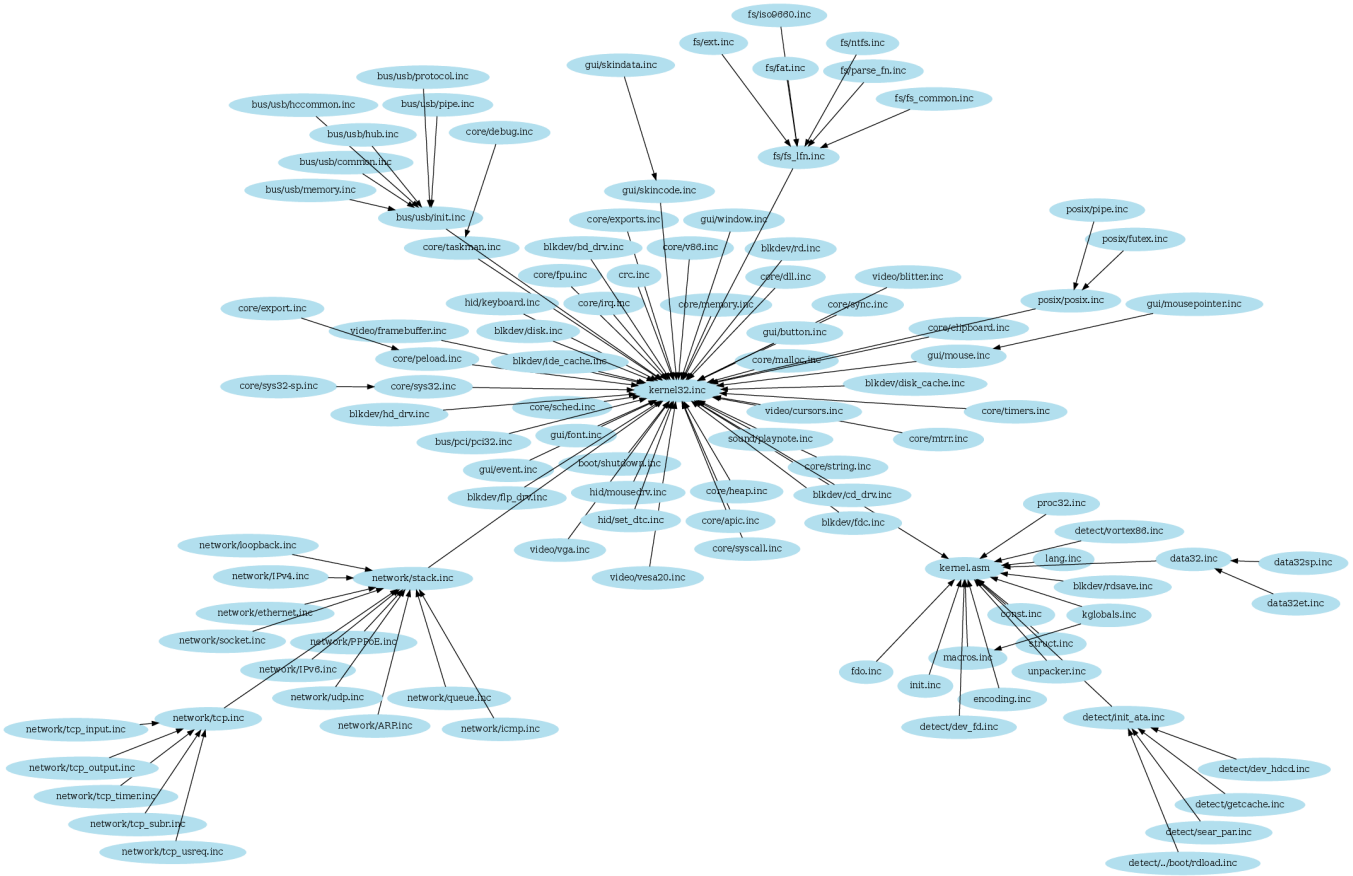
Также, присутствуют макросы для определения структур и работы с ними. Файл struct.inc содержит очень удобный макрос struct, позволяющий писать например так (Листинг 1.2):

Листинг 1.2

```
struct  MUTEX
        wait_list       LHEAD
        count           dd ?
ends
```

Похоже на язык C, не правда ли? Макросы значительно расширяют возможности FASM, приближая его к таким относительно высокоуровневым языкам как C.

Ещё интересен файл kglobals.inc . На первый взгляд вовсе не очевидно, что же делают определенные в нем макросы iglobal и uglobal. Их используют при объявления данных в некоторых участках кода ядра. Но зачем же они нужны? Задача этих макросов собрать данные, объявленные с помощью них, в конец компилируемого файла, чтобы добиться уменьшения размера файла (за счет отсутствия лишних выравниваний на границу align 4) и возможно лучшего сжатия, процедурой упаковки ядра. Данные, обозначенные макросом iglobal являются инициализированными и уже зара-



нее содержат нужное значение. Также есть макрос uglobal, который также собирает данные в конец файла, но помещает их за пределами получаемого бинарника - это не инициализированные данные и содержимое их не гарантируется, хотя обычно содержит ноль. Цель макросов iglobal и uglobal — не столько уменьшение размера файла, хотя и это тоже присутствует, сколько ускорение, процессору существенно лучше, когда код и данные разделены. Данные, размещённые в uglobal, сами по себе действительно не инициализированы, но конкретно в случае ядра Колибри при загрузке выполняется код, который область uglobal явным образом обнуляет.

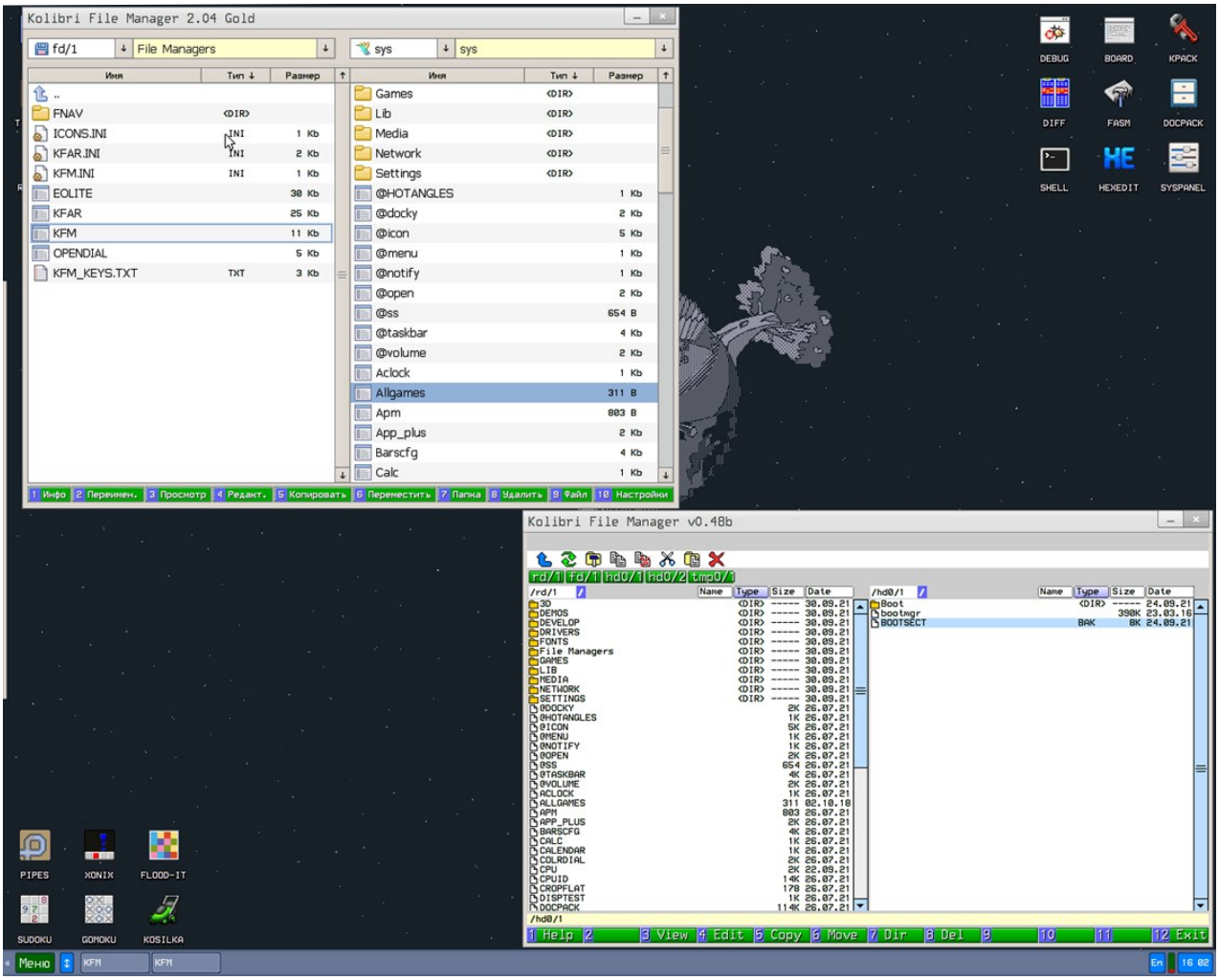
В файле fdo.inc содержится набор макросов для повышения удобства отладки ядра. Кстати, fdo это сокращение от Formatted Debug Output (Форматированный отладочный вывод). Работает он подобно функции printf(), выводя на доску отладки форматированные сообщения. Доступны спецификаторы формата %s, %d, %u, %x . Использовать обычно нужно только макрос DEBUGF, хотя доступны также и DEBUGS, DEBUGD, DEBUGH. Например так:

Листинг 1.3

```
DEBUGF 1, "%s - %d (%x)", eax, 123, ch
```

Что ж, думаю, на этом можно завершить наше вводное ознакомление с устройством ядра KolibriOS. На мой взгляд, ядру Колибри можно было бы посвятить целую книгу — его код насчитывает десятки тысяч строк ассемблерного кода, а вместе с тем и богатую историю длинной более чем в 15 лет. В будущем, планирую написать еще несколько статей по данной теме, а пока — спасибо за внимание.

KFM&KFM2 — феникс из далёкого прошлого



Сегодняшний обзор будет не совсем обычным — ведь мы будем обозревать сразу две версии одной и той же программы, а именно файлового менеджера, который почти настолько же стар, как и KFAR.

KFM (Kolibri File Manager) – это файловый менеджер, по дизайну напоминающий KFAR. Он имеет такие приятные и полезные фичи как горячие клавиши для быстрого выполнения действий, две панели, которые работают как два отдельных окна и бар для переключения дисков. По сравнению со своей старой версией KFM2 приобрёл более приятный глазу вид, исправление множества багов и возможность применения системных скинов.

Давайте сравним две версии программы — 2.04 Gold и 0.48b. В глаза бросается освежённый интерфейс новой версии - по сравнению с более старой KFM 2.04 выглядит более приятным глазу благодаря использованию библиотек от eolite и новому шрифту. Впрочем, так было не всегда – до версии 2.0 программа выглядела совсем иначе и написана на FASM. Кстати, тогда она весила 13 КБ, а сегодняшняя версия бьёт рекорды по компактности и весит всего лишь 307 байт.

Точкой отсчёта создания KFM можно считать 18 октября 2006 года – именно тогда Марат Закиянов (он же Mario/Mario79/Mario_r4) создал файл HISTORY с незатейливой фразой «Начало проекта». Впоследствии, с уходом Марата в 2015 году из коллектива программа долго оставалась без разработчиков (соответственно и без багфиксов). В 2016 году началась война за место в образе дискеты, и KFM почти пошёл под нож. Он висел на волоске целых пять лет, и только в июне этого года старый Kolibri File Manager окончательно ушёл со сцены, уступив место KFM2. Новую версию данной программы сделал достаточно известный в кругу разработчиков дизайнер Leency, который поддерживает данную программу и по сей день.

Автор: Alex2003 aka virtual_codehack

Интервью с Serge

Автор: Сергей Кузьмин aka Wildwest

Предлагаем Вашему вниманию интервью с **Сергеем Семеновым** aka **Serge** (на Хабре ion2), одним из самых продуктивных разработчиков в проекте KolibriOS.

Интервью было проведено **Сергеем Кузьминым** aka **Wildwest** (на Хабре W__W)

У тебя есть хобби?

Когда-то я клеил ~~танчики~~ корабли и самолёты. Теперь хобби стало программирование железа, чтение документации и изучение чужого кода.

Как и когда ты заинтересовался компьютерами и программированием? Какие книги были полезными для тебя?

Сперва были программируемые калькуляторы и первые статьи в журнале «Наука и жизнь». Это был 1985 год. До этого домашний компьютер казался мне таким же нереальным, как полёт на Луну. Потом появилась Искра 1080, на которой я изучал ассемблер 8080.
Из книг «Просто и ясно о Borland C++», отличная «освой самостоятельно Программирование в Windows за 21 день» Чарльза Калверта, наши «Язык программирования Си для персонального компьютера» и «Использование Turbo Assembler при разработке программ». Конечно «Программирование на аппаратном уровне» Кулакова и ещё два десятка других книг, Страуструп в их числе. Но почётное место на книжной полке занимают три тома «IA-32 Intel Architecture Software Developer's Manual» и руководство по оптимизации. Было время, когда Интел рассылал эти кирпичи по всему свету (Рис. 1-2).

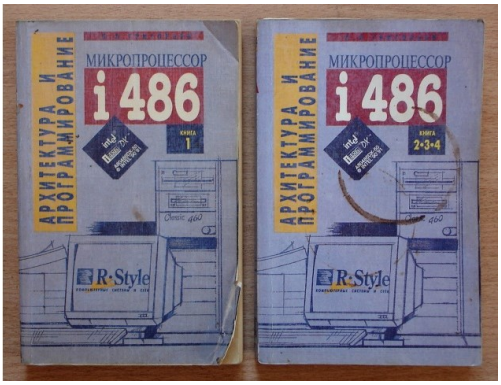


Рис. 1 Тёплые ламповые мануалы до обновления...

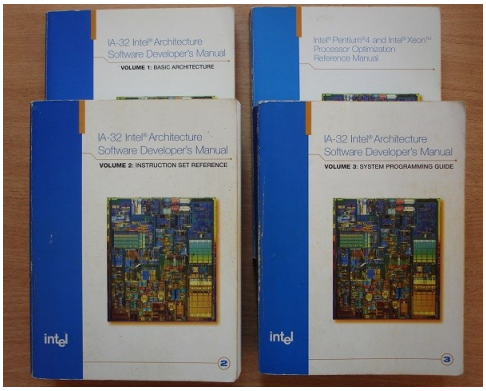


Рис. 2 …и после него

Какие ОС пробовал? Какие твоя домашняя, рабочая и самая любимая ОС?

DOS 3.0, 4.0, 5.0 и почти все Windows: 3.1, 95, 95OSR2, 98, NT4.0, XP, Win7.

Сейчас стоит 8.1 Она меня вполне устраивает, но перегружаться в Колибри из предыдущих было удобней. Ещё пользовался разными Linux и остановился на Debian Jessie и KDE desktop.

Какие языки программирования ты знаешь и почему учил их? Какие из них ты любишь больше других и почему?

На курсах в универе знакомили с PL/M, в школе мучили ершолом. В институте был обязательный fortran, на работе Clarion 2.0. Некоторое время пользовался Borland Pascal 7.0, но потом надоел этот жуткий синтаксис и перешёл на Си. С тех пор прошло 20 лет.

Что можешь сказать относительно своих проектов (предыдущих и текущих), которые не связаны с KolibriOS?

Сейчас таких не осталось.

Как и когда ты обнаружил KolibriOS? Сразу ли начал разрабатывать для неё? Почему ты программируешь для неё?

В 2001 году в Компьютерре была маленькая заметка о MenuetOS. Наверное там писали о какой-то другой Menuet, потому что свежескачанная 0.38 жутко тормозила и выглядела страшновато.

Время от времени я проверял новые версии, пока в 2006 не наткнулся на Колибри. Новая системка была намного симпатичнее на вид и работала шустрее. Потом я попал на форум и влез в дискуссию об обработке IRQ. Кто-то опять был неправ. Так и засосало. Мне всегда было интересно «играться» с железом. Колибри удобна тем, что в ней нет сложных HAL прослоек между ядром и железом. Всё очень просто, наглядно и ядро компилируется и загружается мгновенно. Это немаловажно, когда делаешь это по сотне раз.

Чем занимался и занимаешься в проекте KolibriOS? Опиши текущее состояние своих проектов и их историю.

Первым проектом была звуковая подсистема. В Колибри был плеер ac97wav **Ивана Поддубного**, работавший напрямую с контроллером и кодеком. Проблема была в том, что программа не могла обрабатывать прерывания от контроллера и звук шёл со щелчками, во-вторых это был монопольный доступ к железу. Требовалось сделать загружаемый драйвер и системный микшер, чтобы несколько программ могли одновременно выводить звук. С драйвером контроллера для AC97 не было особых проблем, хотя по глупости спалил наушники, но тестировать код приходилось в unreal mode DOS, потому что в ядре была нулевая поддержка загружаемых драйверов. Нельзя было выделить виртуальную память в ядре, замапить физическую страницу на требуемый адрес, установить обработчик IRQ и т.п. Даже не было внутренней функции для загрузки файла. Приходилось делать это через вызов ядра 0x40. В результате пришлось написать управление страничной и виртуальной памятью, загрузчик драйверов, переделать обработку IRQ, прежде чем в Колибри появился звук.

Второй большой работой, все ещё незавершённой, было преодоление наследства «тяжелого финского прошлого». Во-первых, в коде Менует не использовались символьные имена переменных. В ранних ревизиях Колибри приходилось продираться через такое (Листинг 2.1).

Листинг 2.1

```
mov eax,[0xFE00]
shr eax,1
mov [0xFB0A],ax
mov eax,[0xFE04]
shr eax,1
mov [0xFB0C],ax
```

Или же такое (Листинг 2.2).

Листинг 2.2

```
mov edi, [0x3004]; the last process (number)
movzx esi, word [0xC400 + edi * 2]
shl esi, 5
add esi, window_data
```

Во-вторых, «маленькое ассемблерное ядро» жрало память в совершенно диких количествах. Для байтовой переменной было типичным выделение килобайта, для буфера десятков килобайт — адреса абсолютные, поэтому выравнивали на красивые круглые значения. Всё это было от бедности. Если ядро не умеет динамически выделять память, остаётся резервировать статически и с запасом. Постепенно все абсолютные адреса были заменены символьными метками а аппетит ядра ужат примерно в три раза.

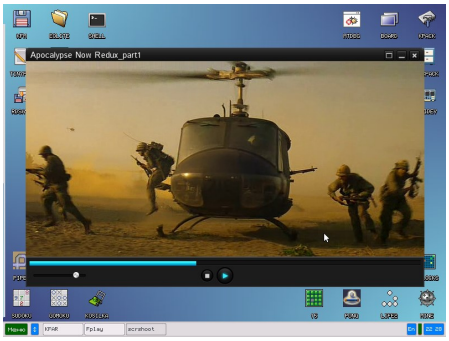


Рис. 3 Видеоплеер FPlay

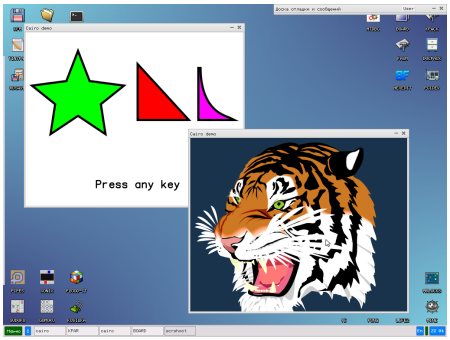


Рис. 4 Демки Cairo

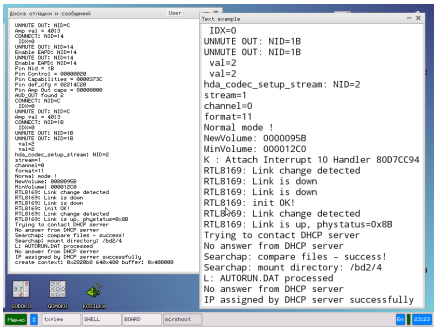


Рис. 5 Встроенный шрифт Колибри (слева) vs Droid Sans Mono и FreeType 2.5

Ещё было множество мелких и не очень изменений в ядре и бесконечная эпопея с видеокартами. Сперва был ассемблерный

драйвер для ATI R100-R500, включавший аппаратные курсоры. Потом AMD представило radeonhd и в Колибри появилась возможность менять видеорежим. Через несколько месяцев ядро Линукс вслед за Колибри :-) перешло на KMS и новый драйвер был опять портирован. В конце 11 года появился драйвер для i915, а через два года Mesa драйвер i965 и закрутились первые «железные» шестерёнки. Параллельно был портирован ffmpeg и сделан видеоплеер на его базе. + портированы несколько популярных библиотек: cairo, freetype, libpng, zlib, mesa (Результаты на рисунках 3-5). Последним проектом были binutils: ar as ld objcopy strip, всё что нужно для линковки непосредственно в Колибри.

Расскажи про бранчи ядра, разрабатываемые тобой.

kolibri-acpi правильнее назвать kolibri-next. Здесь тестируется код перед включением в основное ядро.

Kolibri-process — улучшалась поддержка многопоточных приложений. Изменения залиты в основное ядро, ветка удалена за ненадобностью.

kolibri_pe – историю и мотивы лучше почитать на форуме. Начальная часть работы была успешно завершена. Код ядра на Си и ассемблере линковался при помощи ld в PE DLL с фиксированным базовым адресом. Ядро загружалось GRUB, прыгало из защищённого в real mode, показывало наш любимый синий экран загрузки с выбором опций, включало графический режим и возвращалось в защищённый режим. Первый гвоздь в крышку проекта забила АМД. Появился драйвер radeonhd, на который я сразу переключился. Второй гвоздь был от CCP Games.

Результаты представлены на FTP-сервере проекта, в ночной сборке. Какой из этих проектов был самым трудным?

Портирование Doom. Уже была версия для Менуэт, так что работа казалась не слишком сложной, главной задачей было добавить звук. Но чёртова программа вылетала после загрузки и было совсем непонятно в чём дело. Пришлось неделю возиться с отладкой, прежде чем выяснилось, что id Software и Open Watcom по разному определяют sizeof(bool). Дальше всё пошло как по маслу (Рис. 6).

Рис. 6 Портированный DOOM

Ты программировал для устройств разных фирм, читал документацию. Были ли сложные моменты, когда было ограничение со стороны железа или со стороны ОС?

В Колибри нет поддержки ACPI и диспетчера устройств. Это создаёт множество проблем. Например ядро до сих пор обрабатывает прерывания в режиме доисторического 8259A, а BIOS часто назначает разным PCI устройствам общие линии IRQ. В теории такое разделение линий должно работать, но в реальной жизни это не так. Ядро умеет работать и в режиме APIC, но это требует некоторых подготовительных действий со стороны пользователя и, к сожалению, положительный результат не гарантирован.

Перечисление устройств на шине PCI даёт информацию об используемых ресурсах, но без ACPI она неполна. Иногда устройствам (i915 Gen3 gpu) требуется незанятый диапазон физических адресов, а у ядра недостаточно данных для его резервирования. А ещё есть древний SoundBlaster, способный работать только в первых 16Мб, и для него приходится статически резервировать память.

Драйверы для звуковых и видеокарт, портирование библиотек, видеоплеер… Что дальше?

В планах 3D драйверы для AMD Radeon и аппаратное декодирование видео в Fplay. К сожалению, на этом пути есть несколько препятствий. Fplay использует ffmpeg, а его разработчики не очень дружны с разработчиками VAAPI и libVA, наверное потому, что у последних есть свой gstreamer. С другой стороны у ffmpeg всё прекрасно с vdpau, так что на радеонах декодер заработает, как только будут готовы r600 и radeonsi для Galluim3D. Но они зависят от llvm, а портированная llvm вместо ожидаемого кода генерирует… Неизвестно точно, что и как генерирует llvm, потому что трассировка двадцатимегабайтного бинарника немного затруднительна без отладчика с поддержкой исходного кода.

Часто приходят новички и спрашивают про переписывание отдельных частей ОС, портирование на другие аппаратные платформы. Время от времени даже появляются форки разработчиков-одиночек, например от *mike.dld*, *art_zh* и т.п. Как ты к этому относишься?

Спокойно отношусь. Я сам запустил форк, а потом бросил.

Важна ли совместимость с аппаратными платформами или другими ОС для KolibriOS?

Колибри ассемблерная система для x86. Вряд ли есть смысл говорить о других аппаратных платформах.

Ты в основном пишешь и портируешь программы на C++. Какие инструменты ты используешь в разработке ?

У нас есть тулчейн на основе gcc-4.8.2 и binutils-2.24 в версиях для Windows и Linux и небольшой набор необходимых библиотек, в том числе libstdc++.

Часто ли приходится править стандартные библиотеки компилятора, добавляя недостающие функции? Насколько полно поддерживается POSIX?

C/C++ программы используют newlib в качестве стандартной библиотеки. Всё начиналось с видеоплеера на базе ffpmeg. В первой версии было минимально необходимое количество функций. С каждым новым проектом я добавляю требуемые функции в newlib. Это конечно неправильно, но портирование стандартной библиотеки довольно нудное занятие. В menuetlibc POSIX поддерживается в большем объёме. В целом все трудности с POSIX преодолимы, требуется немного упорства и терпения.

Можешь дать какой-нибудь совет стремящимся стать программистами? Какие языки надо попробовать? Что посоветуешь разработчикам новых ОС? Как начать? Какие учебники читать?

Раньше я бы посоветовал изучить ассемблер для целевой платформы, чтобы понимать во что в итоге превратится твой код. А теперь всё кроссплатформенное, интерпретируемое и неизвестно на чём оно будет исполняться. Но для разработчиков ОС есть несколько советов:

Первое: присоединиться к существующему проекту или найти единомышленников. Три головы в четыре раза лучше одной.

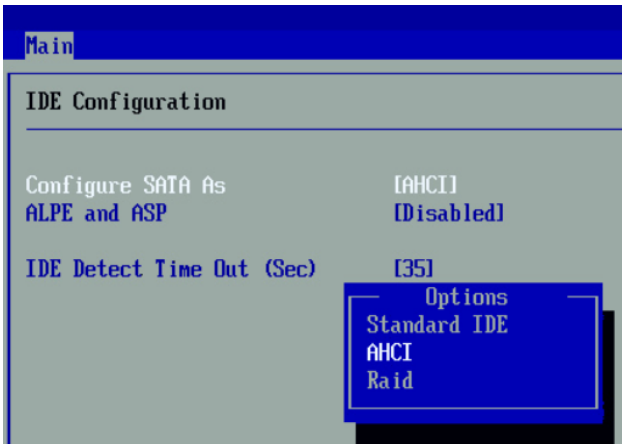
Второе: не стоит начинать разработку с микроядра, особенно в одиночку. Вам будет сложнее разрабатывать драйверы и приложения.

И третье: когда будете придумывать свой «уникальный, не имеющий аналогов API» вспомните о POSIX. Рано или поздно вам понадобится совместимость.

Спасибо за интервью!

Данная статья была взята с блога KolibriOS на HabraHabr. Ссылка на оригинал: **https://habr.com/ru/company/kolibrios/blog/256405/**

9 сентября 2021



Драйвер AHCI успешно написан и протестирован в Колибри! Теперь SATA диски смогут работать в родном режиме AHCI без использования эмуляции BIOS или устаревшего драйвера на Си (подключался вручную). Разработку оплачивал участник проекта Липатов Кирилл (leency), заявку на выполнение мог подать любой желающий. По итогу работу выполнил другой участник нашего проекта Рустем Гимадутдинов (rgimad).
Преимущество нового драйвера AHCI над старым:
1. Поддержка записи
2. Открытые исходники против закрытых
3. Ассемблер против Си
4. Примерно 4 Кб бинарного кода против 19 Кб
5. Встроен в ядро, что даёт возможность ранней инициализации дисков

У Рустема это первый подобный по сложности проект. Код получился достойного качества и доступен для ознакомления по ссылке http://websvn.kolibrios.org/filedetails.php?repname=K..
Осталось небольшое TODO: добавить поддержку CD/DVD приводов.
Согласно договорённости, 500$ отправляются Рустему в Казань. Спасибо ему за отличную работу!
Славы и процветания Колибри!

KolibriOS inside Team, 2021

Журнал разрешается цитировать, свободно распространять и упоминать в Интернете.