Установка ОС FreeBSD на MacBook выпуска 2008 года

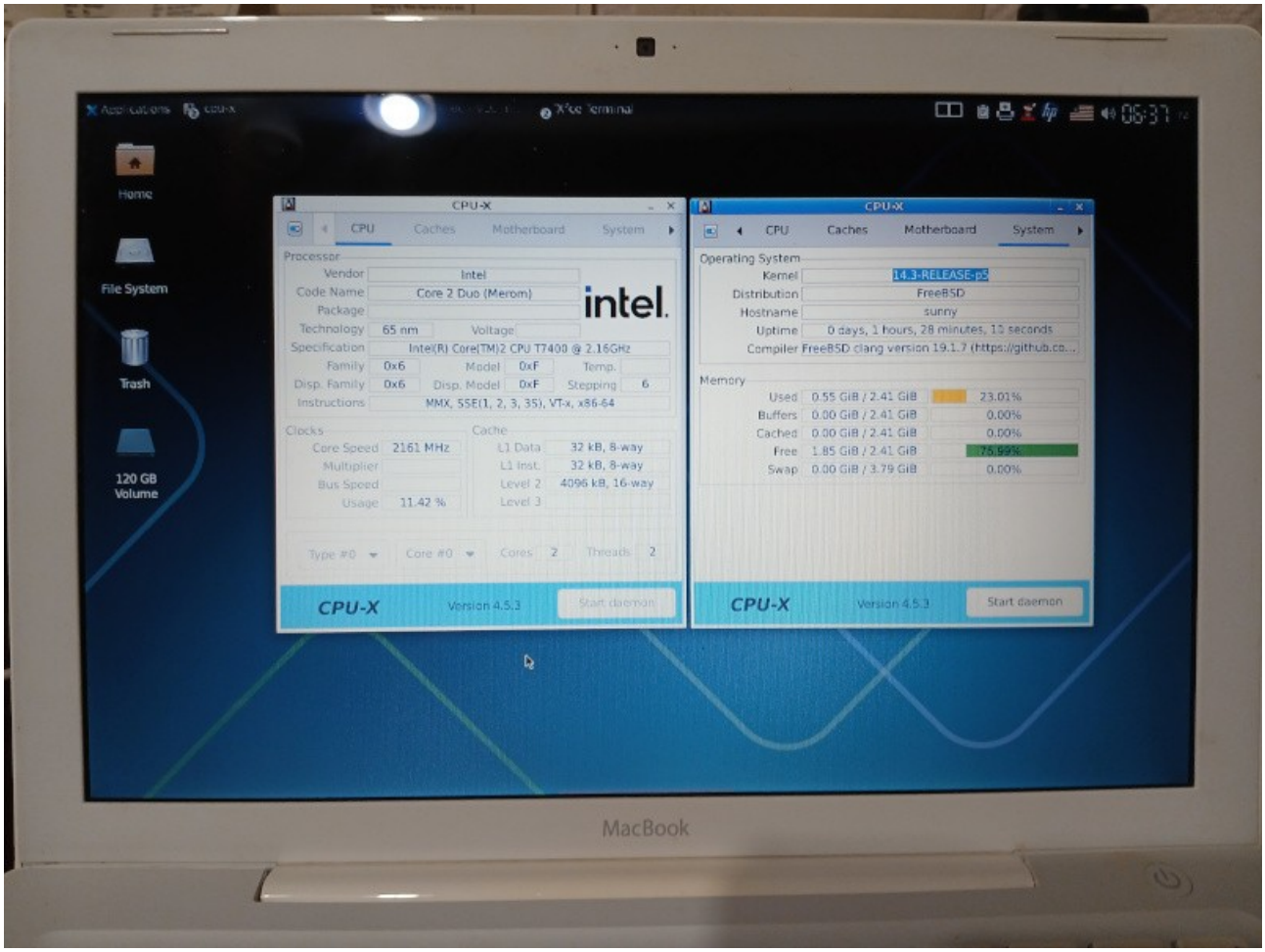
Автор: Залата Руслан Николаевич Опубликовано: 22.01.2026

Ссылка: https://habr.com/ru/articles/981214/

Недавно, перебирая хлам скопившийся на пыльных антресолях, я обнаружил старенький MacBook 2,1 (A1181) образца 2008 года выпуска. Этот MacBook мне на день рождения, который состоялся более 15-ти лет назад, подарила супруга. Большим ценителем яблочной продукции я так и не стал, но некоторое время использовал эту машинку в качестве второй — для чтения почты или просмотра ютуба, брал с собой в туристические поездки. Мне нравился дизайн корпуса и клавиатуры этой машинки, местами даже где-то меня привлекала операционная система MacOS X являющаяся родственником FreeBSD. Но сейчас не об этом. В те времена вычислительная техника персонального применения стремительно устаревала — появлялись процессоры с всё большим числом ядер и большей тактовой частотой, состоялся полный переход от 32-битных архитектур к 64-х битным и т. д. Всё это сильно расслабило разработчиков и, как следствие, требования софта к железу выросли просто непомерно. Таким образом, данная машинка очень быстро «морально устарела». Я пару раз проводил апгрейд — сначала добавил немного SDRAM, потом заменил HDD на SSD, несколько раз апгрейдил MacOS X. Но к 2012 году машинка всё равно стала мало пригодной для работы и отправилась на антресоли.

Две недели назад этот MacBook попался мне на глаза. Я включил его и обнаружил, что MacOS X прекрасно загружается и даже подключается к WiFi, только вот в Safari все сертификаты давно протухли и ни один Web сайт не открывается. Но тут меня посетила совершенно здравая мысль - почему бы не оживить эту машинку установкой на неё современной ОС, такой как например FreeBSD ? К тому же появился повод — одна из моих дочерей доросла до своего компьютера, а покупать ей сейчас новый ноут для посещение «учи.ру» меня, откровенно говоря, «жаба душит». Короче, я быстренько заказал новую АКБ на известном китайском маркетплейсе и принялся изучать вопрос «как пропатчить ~~KDE~~ MacBook для FreeBSD».

В данной статье я расскажу о некоторых особенностях яблочных ноутбуков мало знакомых даже опытным маководам, о тонкостях установки FreeBSD, о проблеме под названием UEFI, о баге в ядре ОС FreeBSD с которым я столкнулся и помог зафиксить, о прекращении поддержки DRM-KMOD драйверов в 32-х битной ветке ОС FreeBSD и о том, как мне все же удалось портировать их для FreeBSD 14.3-RELEASE/i386. И еще о некоторых мелочах, знание о которых необходимы для того, чтобы сделать ОС FreeBSD пригодной для эксплуатации на этом стареньком MacBook-е, и чтобы Ваш кислотно-щелочной баланс всегда оставался в полном порядке™.


Home
File System
Trash
120 GB Volume
CPU-X
CPU
Caches
Motherboard
System
Processor
Vendor Intel
Code Name Core 2 Duo (Merom)
Package
Technology 65 nm
Voltage
Specification Intel(R) Core(TM)2 CPU T7400 @ 2.16GHz
Family 0x6
Model 0xF
Temp.
Disp. Family 0x6
Disp. Model 0xF
Stepping 6
Instructions MMX, SSE(1, 2, 3, 3S), VT-x, x86-64
Clocks
Core Speed 2161 MHz
Multiplier
Bus Speed
Usage 11.42 %
Cache
L1 Data 32 kB, 8-way
L1 Inst. 32 kB, 8-way
Level 2 4096 kB, 16-way
Level 3
Type #0
Core #0
Cores 2
Threads 2
CPU-X
Version 4.5.3
Start daemon
Operating System
Kernel 14.3-RELEASE-p5
Distribution FreeBSD
Hostname sunny
Uptime 0 days, 1 hours, 28 minutes, 11 seconds
Compiler FreeBSD clang version 19.1.7 (https://github.co...
Memory
Used 0.55 GiB / 2.41 GiB 23.01%
Buffers 0.00 GiB / 2.41 GiB 0.00%
Cached 0.00 GiB / 2.41 GiB 0.00%
Free 1.85 GiB / 2.41 GiB 76.99%
Swap 0.00 GiB / 3.79 GiB 0.00%
MacBook

СОДЕРЖАНИЕ

1. Выясняем модель центрального процессора и поддерживаемую систему команд (ISA)

Незадолго до моих изысканий с MacBook-ом состоялся выход ОС FreeBSD 15.0-RELEASE и мне очень хотелось установить на этот MacBook именно её. В этой версии упор был сделан на расширение поддержки WiFi адаптеров, а также на поддержку различного десктопно-ноутбучного железа, о чем не перестают упоминать в мэйл-листах представители FreeBSD Foundation. Собственно хотелось посмотреть как слова сходятся с делом. Но тут есть один серьезный нюанс — с версии 15.0-RELEASE из ОС FreeBSD полностью выносят поддержку архитектуры i386 (32-х битной системы команд x86) и для PC остается только amd64 (x86-64). Я краем сознания начал припоминать, что когда в последний раз мы с коллегой пытались проапгрейдить MacOS X на этом ноуте, то самая максимальная версия которую нам удалось поставить была MacOS X 10.7 «Lion» работавшая в 32-х битном режиме. Следующий релиз MacOS X 10.8 «Mountain Lion» уже не взлетел, так как из него была убрана поддержка 32-х битной системы команд (это, кстати случилось 2012 году, так что FreeBSD еще очень долго продержалась). В общем, возник резонный вопрос — а какой собственно процессор установлен в этом MacBook-е и поддерживает ли он систему команд x86-64.

Немного погуглив я выяснил, что в данной модели MacBook2,1 (A1181) обычно устанавливались процессоры **Intel Core Duo** и **Intel Core2 Duo**, коих модификаций было произведено несколько десятков, в различных т.с. комплектациях (различные частоты, размеры кэшей и наборы команд). Причем, первый является 32-х битным и не поддерживает x86-64 ISA, а вот второй — уже поддерживает!

Загрузив MacOS X и зайдя в меню «**Apple**»->«**System Information**»->«**Hardware**» я обнаружил следующее:

```
Processor Name: Intel Core Duo
Processor Speed: 2.16GHz
```

И тут же слегка приуныл. Но мне захотелось выяснить какая же все таки модель центрального процессора установлена в этом ноутбуке, нигде в «System Information» этой информации не просматривалось. Я запустил в терминале утилиту **dmesg** и, проанализировав её вывод, также не обнаружил модели центрального процессора. Видимо в компании Apple считают, что такие подробности пользователю только во вред.

Пришлось опять задать вопрос поисковой системе и выяснить, что узнать модель процессора в MacOS X можно через системные переменные утилитой **sysctl** (мог бы конечно и сам догадаться ;)), а команда для получения типа и модели процессора выглядит так:

```
$ sysctl machdep.cpu.brand_string

machdep.cpu.brand_string: Intel(R) Core(TM)2 Duo CPU T7400 @ 2.16GHz
```

То есть в моём MacBook-е установлен центральный процессор не «Core Duo», как считает утилита «System Infomation», а «Core2 Duo» который, по идее, должен поддерживать 64-х битные инструкции. На всякий случай я скачал спецификацию на Intel Core2 Duo, где перечислен в том числе и мой T7400, и нашел текст следующего содержания:

> The following list provides some of the key features on this processor:
> • Dual-core processor for mobile with enhanced performance
> • **Intel® 64 architecture**
> • Supports Intel Architecture with Dynamic Execution
> ...

И далее примечание:

> In addition, the Intel Core 2 Duo mobile processor supports **Intel 64 architecture** which is enabled by 64-bit operating systems and consists of 64-bit instructions and registers. Further details on Intel Extended Memory 64 Technology and its programming model can be found in the Intel Extended Memory 64 Technology Software Developer's Guide.

Таким образом я убедился, что запуск 64-х битной ОС на моём MacBook-е всё же возможен. Косвенно это подтверждается некоторыми редкими постами на Linux-овых форумах, а на GitHub-е даже есть коротенькая инструкция описывающая установку 64-х битной Linux Mint на MacBook точно такой же модели.

Но тогда возникает вопрос: какого, простите, лешего 64-х битная MacOS X 10.8 «Mountain Lion» отказалась на него устанавливаться ? И почему «System Information» в MacOS X обманывает пользователя ?

2. Процесс загрузки операционной системы на IBM PC совместимых ЭВМ

Надо заметить, что ноутбуки серии MacBook и MacBook Pro производства компании Apple Inc являются вполне стандартными IBM PC совместимыми машинами, как бы производитель не старался это завуалировать, хотя и с некоторыми особенностями. Поэтому, далее для понимания проблем с которыми придется столкнуться пользователю при установке ОС не от производителя, следует рассмотреть процесс загрузки (и установки) операционных систем принятый в мире IBM PC.

При установке новой операционной системы от производителя на MacBook все достаточно просто — от пользователя требуется вставить фирменный DVD в привод, зажать кнопку «Option» (она же левый ALT) и включить питание коротким нажатием кнопки «Power». Далее компьютер все сделает сам, а пользователю придется только наблюдать за процессом и мило улыбаться. При установке альтернативной операционной системы процесс выглядит не так радужно. Теоретически, установка FreeBSD или Linux на старенький MacBook могла бы выглядеть также легко и непринужденно, если бы не одно «но» - 32-х битный UEFI! Чтобы понять в чем суть этой проблемы, немного погрузимся в детали и историю.

UEFI (или «Unified Extensible Firmware Interface») - это программное обеспечение которое в середине 2000-х годов начало появляться на IBM PC совместимых машинах, с процессорами системы команд Intel x86, в качестве замены старого ПО BIOS («Basic Input/Output System») - придуманного еще Гари Килдейллом, создателем ОС CP/M. Программное обеспечение UEFI, его принято называть «firmware», размещается в ППЗУ на материнской плате и решает ряд важных задач:

1. Производит инициализацию базовой аппаратуры (например, «тренировку» SDRAM, инициализацию PCIe и USB контроллеров).

2. Тестирование базовой аппаратуры вычислительной системы — то, что раньше называлось «Power-On Self Test» (POST).
3. Поиск, считывание и размещение в ОЗУ загрузчика операционной системы, и передача ему управления.
4. Предоставление загрузчику ОС (и самой ОС) программных примитивов для доступа к аппаратуре которая необходима на начальной стадии загрузки. К таким аппаратным средствам относятся порты UART, видеоадаптеры, устройства ввода (USB клавиатуры и мыши), доступ к носителям информации (к дискам).

Надо сказать, что все эти функции прекрасно выполняла и старая BIOS, но тогда зачем потребовалось его заменять на что-то еще ?

Дело в том, что процессоры архитектуры x86 (даже самые современные), в момент снятия сигнала RESET начинают исполнять код в ~~особом~~ обычном 16-ти битном режиме известном как «Real Mode». В этом режиме процессор может адресовать только 1 МБ адресного пространства (2^20 байтов) и не имеет ни защиты памяти, ни её виртуализации. Сделано это, разумеется, исключительно в целях сохранения совместимости со старым ПО. BIOS, как следует полагать, был написан исходя из того, что он, как и загружаемая ОС, будет исполняться в этом «реальном» режиме, а если операционной системе потребуется б**о**льший объем памяти или виртуализация адресного пространства (paging), то она уже самостоятельно переведет процессор в т. н. «защищенный» режим («Protected Mode»), и далее сама займется всем необходимым менеджментом памяти и работой с устройствами без участия BIOS. Вроде бы все логично ? Но не совсем.

К началу 2000-х годов объем кода ядра операционных системы разросся до такого размера, что ядро, как правило представляющее собой монолитный бинарный файл, перестало умещаться в 1 МБ адресного пространства, что создало проблему его начальной загрузки. Масла в огонь подливало и то, что загрузчик, который мог быть считан BIOS-ом, не должен превышать 446 байт, потому как это размер области для размещения загрузчика отведенной в MBR («Master Boot Record»), а MBR, как мы знаем, расположена в первом секторе загрузочного диска и имеет размер всего 512 байт. Разместить в такое ограниченное количество байт код программы который смог бы а) перевести процессор в «защищенный» режим, б) обеспечить драйверы для работы с дисками, UART и видео, и в) считать огромное ядро, распаковать его и запустить — просто нереально. Поэтому разработчики операционных систем начали делить загрузчик на несколько стадий.

В ОС FreeBSD это выглядело, и до сих пор работает, так: загрузчик **boot0**, расположенный в MBR, считывается BIOS-ом и запускается, он подгружает загрузчик промежуточной стадии (**boot1** и **boot2** — они склеиваются в один файл) с этого же диска, но из специально зарезервированной области (первый сектор FreeBSD «слайса»). Далее промежуточный загрузчик переводит процессор в защищенный режим, осуществляет поиск, вытягивание и размещение в памяти более тяжелого загрузчика четвертой стадии под названием **loader_4th**. Этот загрузчик 4-й стадии (не спрашивайте меня почему нет 3-й стадии — я не смог найти ответ на этот вопрос) — это то, что обеспечивает отображение логотипа, меню пользователя при загрузке, конфигурацию переменных ядра и исполнение загрузочных скриптов. В конечном счете задача четвертой стадии сводится к настройкам параметров ядра, к загрузке и размещению его в памяти, и передаче ядру управления. Более подробно, во всех красках и с кусками ассемблерного кода, о процедуре загрузки ОС FreeBSD можно прочитать тут: https://docs.freebsd.org/en/books/arch-handbook/boot/

И вот получается так, что к 2005 году назрела острая необходимость заменить BIOS на что-то более стройное, выверенное и соответствующее современным реалиям, ну чтобы упростить процесс загрузки. Но как всегда, получилось всё наоборот.

Вместе с переходом на UEFI, в дополнение к MBR (устаревший способ ведения таблицы разбивки дисков на разделы), добавили новый способ - GPT («GUID Partition Table») позволяющий идентифицировать разделы не по логическому номеру, а по уникальному GUID идентификатору, что позволило отвязать адресацию дисковых разделов от номера порта/слота на шине и логического расположения на диске. В целом это весьма неплохое достижение прогресса.

Программное обеспечение UEFI разделено на две части: системная часть UEFI (или «UEFI Firmware») - выполняет функции мини операционной системы предоставляющей UEFI-приложениям программный интерфейс UEFI API состоящий из некоторого набора handler-ов (указателей на процедуры) для взаимодействия с аппаратурой, а также ряда сервисных функций (например, вывод текста на экран). И UEFI приложения. При этом, и системная часть UEFI и UEFI приложения работают в «защищенном» режиме, обе исполняются в нулевом (самом приоритетном) кольце защиты (Ring 0 на x86).

UEFI Firmware, после проведения тестов аппаратуры, производит поиск, загрузку в память и запуск одного из UEFI приложений. Среди UEFI приложений могут быть различные утилиты для дополнительного тестирования и настройки специализированной аппаратуры (например RAID контроллера), оболочки с командной строкой или графическими меню для настройки аппаратуры пользователем (аналог меню BIOS). Одним из важных приложений является «EFI Boot Loader» — приложение предоставляемое разработчиком операционной системы, которое выполняет загрузку ядра этой ОС. Схема процесса загрузки ОС через «EFI Boot Loader» показана на рис. 1 ниже. В целом, видно что процедура действительно несколько подсократилась.

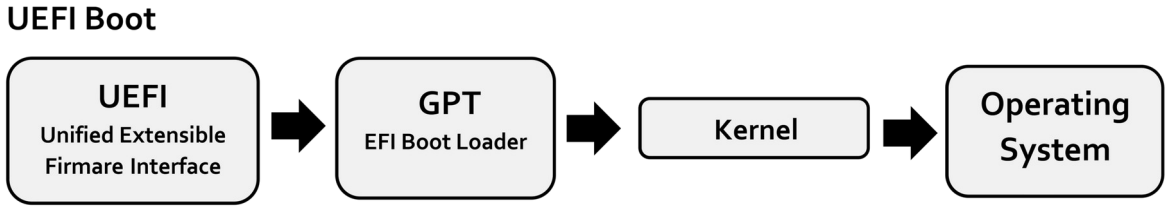


Рис. 1. Процесс загрузки операционной системы на ЭВМ с UEFI.

На этом прелести UEFI заканчиваются и начинаются сложности.

Первая сложность состоит в том, что для того, чтобы UEFI Firmware обнаружило приложение-загрузчик, необходимо на загрузочном диске иметь первый раздел типа **FAT** (FAT12, FAT16 или FAT32) и обязательно поименованный как «**ESP**» (от «EFI System Partition»). В этом разделе в подкаталоге **/EFI/BOOT/** должен содержаться файл с именем **BOOTIA32.EFI** или **BOOTX64.EFI**, в зависимости от типа «битности» UEFI Firmware на данной машине. Этот файл будет загружен и исполнен в качестве приложения-загрузчика.

И тут мы подходим к следующей проблеме. На машинах с x86 процессорами, UEFI Firmware бывают двух видов: **IA32** — работающая в «защищенном» режим с 32-х битной адресацией (также известный как **i386**), и **X64** — работающая в «защищенном» режиме с 64-х битной адресацией и использующая систему команд **x86-64** (**Intel 64** или **amd64** или **«long**

mode»). Проблема состоит в том, что UEFI Firmware работающая в **IA32** режиме может исполнять только UEFI загрузчик также работающий в этом режиме, но не может исполнять загрузчик написанный с использованием **X64** режима и системы команд **amd64**. Это приводит к тому, что пользователи ранних ПК с UEFI, который, как можно догадаться, работал только в **IA32** режиме, не могли устанавливать и загружать на своих ПК операционные системы работающие в 64-х битном режиме, даже если процессор поддерживал набор инструкций **amd64**!

Тут следует заметить, что создатели UEFI предвидели такой сценарий и имплементировали режим совместимости со старым ПО которое загружается по описанному выше классическому методу, через эмуляцию BIOS. Этот способ загрузки называется **CSM** («Compatibility Support Module»). Однако разработчики современных ОС либо игнорировали этот режим, либо считали, что загружаясь через MBR (режим CSM), ОС должна работать только в 32-х битном режиме.

Иными словами, представьте что Вы в 2007-2009 годах приобретаете новенький ПК с 8 ГБ SDRAM, но у Вас нет никакой возможности воспользоваться 64-х битной ОС (даже от Microsoft) и задействовать более 4 ГБ, потому что в Вашем ПК установлена одна из первых редакций UEFI Firmware работающая только в 32-х битном режиме. Не верите ? Вот QA на сайте Microsoft:

> https://learn.microsoft.com/en-us/answers/questions/4014730/i-can-t-boot-64-bit-operating-systems-on-my-laptop

3. 64-х битная ОС на компьютере с 32-х битным UEFI

Как уже догадался уважаемый Читатель, в продукции фирмы Apple Inc того времени тоже присутствовал 32-х битный UEFI Firmware, ровно с этой же проблемой — нет возможности устанавливать и загружать 64-х битные ОС, даже если в машине установлен 64-х битный процессор. И мой **MacBook2,1 (A1181)** оказался как раз из таких!

Но не всё так плохо. Известно, что если в 32-х битном UEFI приложении отказаться от сервисов UEFI, то всё еще можно перевести процессор в 64-х битный режим, но работать с аппаратурой придется уже полагаясь только на свои силы. Собственно, когда ядро операционной системы запущено и готово к работе, оно выполняет именно это действие — отписывается от UEFI сервисов и начинает подгружать и инициализировать свои драйверы, отныне полагаясь только на них. Ниже поясняющая цитата из Wikipedia:

> UEFI only allows executing UEFI applications that match the firmware's bit-width, even if the processor supports smaller or larger bit-widths. For example, a 64-bit UEFI firmware may only execute 64-bit UEFI applications, even if the processor has a 32-bit processor mode.[16]:sections 2.3.2 and 2.3.4 Some low-end computers have been shipped with 32-bit UEFI firmware running on 64-bit CPUs.[21] Once a UEFI application ends the boot services and gets granted full control over the system, it becomes possible to change the processor execution mode.[16]:sections 2.3.2 and 2.3.4 However, calling runtime services requires shortly changing back to the original processor mode,[22] as runtime services may only be called from the same processor mode as the firmware implementation.[16]:sections 2.3.2 and 2.3.4

То есть теоретически, возможен такой фокус: перевести процессор в 64-х битный режим и вести всю основную работу в нем. А когда потребуется вызвать UEFI сервис — временно перейти в 32-х битный режим, выполнить запрос к сервису и вернуться обратно в 64-х битный режим. Даже более того, никто не запрещает загнать UEFI сервис в виртуальную 32-х битную машину и делать к ней вызовы из 64-битного окружения. Теоретически, это позволяет иметь 32-х битный UEFI загрузчик способный стартовать 64-х битную ОС!

Однако, поставлять с операционной системой 32-х битный UEFI загрузчик способный загружать и запускать 64-х битное ядро ОС никто не спешил. Да и особого смысла в этом не было, так как индустрия быстро и полностью перешла на 64-х битные процессоры, и уже совсем скоро все новые ПК начали поставлять с 64-х битным UEFI Firmware. А участь тех бедолаг, которые рискнули приобрести себе технику на изломе технологического перехода, никого не беспокоила. Так большинство ПК, ноутбуков и MacBook-ов с 32-бит UEFI просто ~~вымылось из оборота~~ осело на антресолях.

Тем не менее, где-то в 2010 году появляется проект «The rEFIt Project» (или просто «**rEFIt**») от Кристофа Фистерера (Christoph Pfisterer) — это отдельное 32-х битное UEFI приложение которое является промежуточным загрузчиком или Boot Manager-ом. Оно позволяет выбрать, загрузить и запустить родной загрузчик одной из имеющихся на дисках операционных систем, причем операционная система может быть с MBR, с 32-х битным или с 64-битным UEFI загрузчиком. Как указано на старой страничке проекта **rEFIt**, одна из решаемых задач для этого проекта — найти способ установки 64-х битной версии MacOS X «Snow Leopard» на MacBook с 32-х битной UEFI. Если бы я тогда знал про этот проект, то обязательно воспользовался бы им и поставил 64-х битную MacOS X. :)

Проект «**rEFIt»** с 2013 года не поддерживается, а на основе него создан и развивается другой проект - «The rEFInd Project» (или «**rEFInd**») поддерживаемый Родом Смитом (Roderick W. Smith). Род помог адаптировать код к стандартному GRUB загрузчику в ОС Linux, благодаря чему с 2014 года 64-х битные ядра ОС Linux умеют запускаться на 64-х битных машинах с 32-х битным UEFI Firmware. Что касается ядра ОС Linux, то оно еще с версии 2.6.25 (2008 год) умеет прикидываться UEFI приложением, что позволяет ему загружаться минуя промежуточную стадию загрузчика, но как мы уже выяснили, 64-битное приложение UEFI не может работать в среде 32-х битного UEFI Firmware и эта проблема некоторое время оставалась нерешенной для ОС Linux.

Надо сказать, что утилита «**rEFInd**» очень популярна у хардкорных маководов которым требуется загружать различные версии MacOS X или иные ОС на своих компьютерах, а сам Род Смит — весьма известный перец в кругах линуксоидов и маководов. Род написал более двух десятков технических книг и массу статей на эти темы, так что рекомендую заглянуть на его персональный сайт: https://www.rodsbooks.com/

4. 64-х битная ОС FreeBSD на ПК с 32-х битным UEFI

Но вернемся к ОС FreeBSD. Так получилось, что долгое время запускать 64-х битную (**amd64**) версию этой ОС на машине с 32-х битным UEFI Firmware не представлялось возможным, при том, что 32-х битная версия (**i386**) без проблем устанавливается и работает через режим совместимости CSM (эмуляция BIOS). Объяснялось это тем, что ОС FreeBSD в подавляющем случае использовалась как операционная система для высокопроизводительных серверов, где UEFI либо не существовал, либо появился существенно позже и с первого дня был 64-х битным. Тех, кто использовал FreeBSD в

качестве рабочей (десктопной) ОС на ПК было катастрофически мало и их чаяния долгое время оставались без внимания. Но недавно все поменялось.

Последние несколько лет разработчики ОС FreeBSD стали уделять особое внимание адаптации этой ОС к особенностям десктопного и ноутбучного железа. «**The FreeBSD Foundation»** анонсировал финансовую поддержку всякому, кто возьмется устранять длинный список накопившихся проблем у десктопных пользователей. Отдельно была профинансирована разработка новых и доработка старых драйверов. Так, например, за последние годы были проделаны следующие работы по улучшению работоспособности ОС FreeBSD на десктопах и ноутбуках:

1. В ядре ОС реализован специальный фреймворк «LinuxKPI» (Linux Kernel Programming Interface) позволяющий почти без изменений переносить и исполнять внутри ядра ОС FreeBSD драйверы от ядра ОС Linux. Это решило многие проблемы с недостачей драйверов, вызванные тем, что вендоры просто не желали выпускать драйвера к с своему железу для ОС FreeBSD. Теперь практически любой драйвер можно взять из ядра ОС Linux, поместить в простую обертку и скомпилировать из него загружаемый модуль (Kernel Loadable Module) для ОС FreeBSD.

2. С приходом LinuxKPI, из ОС Linux были позаимствованы все драйверы группы DRM (Direct Rendering Manager), включая видео драйверы для GPU про-ва Intel (**i915kms**), AMD (**amdgpu**) и Nvidia (**nvidia**). Этот большой проект получил название DRM-KMOD.

3. Также не без помощи LinuxKPI, плотно занялись улучшением существующих и переносом новых драйверов для WiFi адаптеров из ОС Linux.

4. Решен ряд проблем с ACPI — с режимами энергосбережения при питании от АКБ и с режимами «сна» (хотя работы там еще предстоит много). Теперь FreeBSD почти с 90% вероятностью беспроблемно устанавливается и работает на современных ноутбуках.

5. В репозитории приложений (в «packages» и в «port collection») поддерживается широкий спектр десктопных графических приложений пользовательского назначения. Фактически, сейчас здесь есть всё, что обычно имеется в репозиториях дистрибутивов Linux, и даже кое-что еще.

Все эти улучшения очень полезны и замечательны, но 64-х битная ОС FreeBSD все еще не могла загружаться на ПК с 32-х битным UEFI. И тут неожиданно, в феврале 2024 года, разработчик **VexedUXR** (Ahmad Khalifa) открывает PR «Add support for 64-bit machines with 32-bit UEFI implementations #1098», который в процессе очень долгого ревью и множества изменений (в основном стилистических и идеологических выверок) принимается и мерджится в основную ветку исходников ядра FreeBSD. Результирующий патч был интегрирован и попал в следующие выпуски ОС FreeBSD: 14.3-RELEASE (июнь 2025) и 15.0-RELEASE (декабрь 2025), то есть совсем совсем недавно (данная статья написана в январе 2026 года). Ниже комментарий к патчу от разработчика:

```
commit ce02470205a1521c75edab1fa466e36abd09cda0
Author:     Ahmad Khalifa <ahmadkhalifa570@gmail.com>
AuthorDate: 2024-05-14 19:40:06 +0000
Commit:     Warner Losh <imp@FreeBSD.org>
CommitDate: 2025-03-26 01:28:38 +0000

    stand: Add support for 64-bit machines with 32-bit UEFI implementations

    Some machines have 64-bit capable cpus but are stuck on 32-bit uefi
    firmware.

    Add support for them by building a new "loader_ia32" with
    LOADER_DEFAULT_INTERP along with the 64-bit one. The loader
    can be disabled using MK_LOADER_IA32.

    Reviewed by: imp
    Pull Request: https://github.com/freebsd/freebsd-src/pull/1098
```

Как отмечает Ахмад, данный патч создает еще один UEFI загрузчик **/boot/loader_ia32.efi** являющийся 32-х битным приложением UEFI. Он содержит код для «трамплина» - процедуры, которая переводит машину в 64-х битный режим («long mode» в терминах Intel) и стартует обычное ELF приложение (ядро FreeBSD или загруженный в память модуль). Если поместить данный загрузчик в раздел «ESP» в файл **/EFI/BOOT/BOOTIA32.EFI** на загрузочном диске, то 32-х битный UEFI Firmware будет использовать его для загрузки 64-х битной ОС FreeBSD.

Мне стало интересно каким образом 64-х битное ядро FreeBSD обращается к сервисам 32-х битного UEFI и как этот вопрос решает данный загрузчик. После недолгого изучения исходников ко мне пришло осознание того факта, что ядро FreeBSD вообще никак не зависит от UEFI и необходимости вызова его сервисов просто нет! Все что требуется ядру — это небольшой набор параметров, сбором которых занимается загрузчик, передаваемых ядру в виде готовых к использованию структур. Среди таких параметров есть например параметры видео фреймбуфера: физический адрес в памяти, его размер, размер строки (stride) и разрешение экрана. Имея эти данные ядро самостоятельно рендерит весь текст напрямую в видео фреймбуфер. Для работы с другой аппаратурой (клавиатура, диски, сетевые адаптеры) на стадии загрузки, ядро использует либо статически влинкованные в него драйверы, либо же загрузчик может загрузить их перед стартом ядра ОС используя UEFI сервисы, для чего внутри загрузчика поддерживается скриптовый язык на базе **Lua**, с помощью которого обеспечивается сбор всех необходимых данных об аппаратуре и предварительная загрузка модулей (драйверов) до старта ядра ОС FreeBSD.

Хочу заметить, что в ядре ОС Linux подобного механизма нет. Если ядру Linux требуются какие-то дополнительные драйверы для запуска, их принято помещать в специальный файл «Initial RAM Disk» (**initrd**), этот файл-образ загружается загрузчиком GRUB или U-Boot в известную область памяти до старта ядра, после чего ядро монтирует этот образ как временную корневую файловую систему и далее загружает и запускает из неё требуемые драйверы. Вот такое фундаментальное различие этих двух ОС я для себя открыл.

Но не обошлось без казуса. Поддержку 32-х битного загрузчика в ядро добавили, но внести изменения в инсталлятор (в утилиту **bsdinstall**), чтобы в процессе установки ОС на диск он копировал правильную версию загрузчика в раздел «ESP», забыли. Точнее, забыли добавить переменную **machdep.efi_arch** в **sysctl**, без которой **bsdinstall** не может определить тип UEFI. Таким образом в августе 2025, через пару недель после выхода 14.3-RELEASE, появилось следующее Errata Notice FreeBSD-EN-25:12.efi:

```
FreeBSD Errata Notice: bsdinstall UEFI Loader Issue
```

```
submited 08 August 2025

The FreeBSD Errata Notice FreeBSD-EN-25:12.efi addresses an issue with bsdinstall(8) on
systems with IA32 UEFI firmware. The problem arises because bsdinstall(8) always copies
loader.efi, regardless of the firmware's architecture, due to a missing sysctl commit. This
results in an unbootable system for those expecting a 32-bit UEFI loader. The errata
provides workarounds and solutions, including updating the system via binary or source code
patches. The issue is corrected in FreeBSD stable and release branches dated after the
correction date. Systems with 64-bit UEFI firmware or non-x86 systems are unaffected.
```

Получается, что установить версию FreeBSD 14.3-RELEASE я смогу, но вот загрузиться с этой новой установки не получится, не проведя предварительные ручные манипуляции с загрузчиком. В целом, тут нет ничего сложного, но я решил начать эксперименты сразу с 15.0-RELEASE, ведь в этой версии с загрузчиком уже всё должно быть «чики-пуки». Ну я так думал.

5. Попытка установки 64-х битной FreeBSD 15.0-RELEASE на MacBook

Как и на любой ПК, есть несколько способов установить новую операционную систему на MacBook. UEFI Firmware расположенная в ППЗУ MacBook-а умеет загружать ОС обоими методами: через 32-х битный UEFI загрузчик и старым добрым методом через эмуляцию BIOS (режим CSM). Для того, чтобы начать установку новой ОС, необходимо сначала подготовить загрузочное устройство содержащее либо 32-х битный UEFI загрузчик операционной системы в FAT разделе «ESP», либо с загрузчиком расположенным в MBR. После чего подключить подготовленный носитель к MacBook-у, перевести его в режим выбора источника загрузки («Startup Manager») и произвести загрузку с этого устройства. Далее следовать инструкциям от устанавливаемой ОС.

Выяснилось, что имеющийся у меня MacBook2,1 может вести загрузку операционной системы со следующих носителей:

- с оптического диска (CD/DVD);
- с USB Mass Storage носителя (USB Flash);
- с HDD или SSD подключенного к SATA интерфейсу (к сожалению, в этой модели всего один порт);
- по сети, используя протокол «Apple NetBoot», который мало чем отличается от BOOTP/DHCP.

Из всего этого многообразия мне быстро доступны были только два варианта: с USB Flash носителя и с DVD.

У меня в арсенале давно имеется рабочая машина с относительно древней FreeBSD 9.3-RELEASE предназначенная для всякого рода «археологических» изысканий. В ней, помимо приводов floppy дисков 3.5", установлен рабочий оптический привод Asus BW-16D1HT поддерживающий запись на все виды CD, DVD, Blu-Ray и M-DISC. Вот как он детектируется операционной системой:

```
cd0 at ahcich7 bus 0 scbus5 target 0 lun 0
cd0: <ASUS BW-16D1HT 3.10> Removable CD-ROM SCSI-0 device
cd0: Serial Number KL5L3AG0905
cd0: 150.000MB/s transfers (SATA 1.x, UDMA6, ATAPI 12bytes, PIO 8192bytes)
```

Поэтому план у меня был следующий: сначала попытаться загрузиться и проинсталлировать ОС с USB Flash, а если не получится — то грузиться с DVD. По такому случаю один из коллег поделился со мной завалявшимися у него дома чистыми DVD-R болванками.

5.1. Подготовка загрузочных носителей с ОС FreeBSD

Процедура подготовки загрузочных носителей (USB Flash-ки и DVD диска) для FreeBSD 15.0-RELEASE выглядит так:

1. С помощью утилит **wget, fetch, curl** (или просто Web-браузером) качаем сразу два упакованных образа системы: первый под названием «**memstick**» - это минималистичный загрузочный образ FreeBSD с инсталлятором и «Live System», и второй «**dvd1**» - это полная инсталляшка FreeBSD с коллекцией популярных программ на весь объем диска DVD:

```
$ wget https://download.freebsd.org/releases/ISO-IMAGES/15.0/FreeBSD-15.0-RELEASE-amd64-memstick.img.xz

$ wget https://download.freebsd.org/releases/ISO-IMAGES/15.0/FreeBSD-15.0-RELEASE-amd64-dvd1.iso.xz
```

2. Вставляем USB Flash накопитель (образ потребует около 1,5 ГБ) и записываем на него образ «**memstick**» с помощью утилиты **dd**, минуя предварительную распаковку, следующей командой:

```
# xz -d -c FreeBSD-15.0-RELEASE-amd64-memstick.img.xz | dd of=/dev/da0 bs=8192 conv=sync
```

Здесь **/dev/da0** это блочное устройство USB Flash, так оно обычно детектируется в ОС FreeBSD. В ОС Linux это устройство обычно детектируется как **/dev/sdb**. Выяснить точное имя устройства можно заглянув в вывод команды **dmesg** после подключения USB Flash к рабочему ПК.

3. На всякий случай с помощью утилиты **cdrecord** запишем на DVD-R болванку образ «**dvd1**», также минуя распаковку:

```
# xz -d -c FreeBSD-15.0-RELEASE-amd64-dvd1.iso.xz | cdrecord dev=5,0,0 /dev/fd/0
```

Здесь **dev=5,0,0** задает адрес CD/DVD привода на SCSI шине: **scbus**, **target** и **lun**. Получить его можно из вывода утилиты **dmesg**:

```
cd0 at ahcich7 bus 0 scbus5 target 0 lun 0
```

Команды записи необходимо выполнять от пользователя **root**, то есть предварительно выполнив **su**.

Пользователям ОС Windows для подготовки загрузочного носителя придется воспользоваться сторонней утилитой, например Rufus, предварительно распаковав скачанные образы!

5.2. Перевод MacBook в режим загрузки с альтернативного носителя

Комплект загрузочных носителей у нас есть, настало время разобраться как перевести MacBook в режим выбора загрузочного устройства.

Для тех, что никогда не сталкивался с яблочной продукцией, замечу, что никакого привычного ему меню от AMI BIOS, вызываемого по клавише **F1, F2** или **Del**, тут нет — устройство после включения питания моментально начинает загрузку MacOS X с установленного HDD/SSD. Если этот системный диск отключить физически, то устройство переходит в какой-то «странный режим» — пустой белый экран и не отвечает ни на какие воздействия. Когда я первый раз проделал этот трюк, мне, неопытному маководу, показалось, что MacBook просто напрочь завис. Но это не так. Позже выяснилось, что UEFI фирмарь в течении 60 секунд пытается непрерывно сканировать различные шины в поисках чего-нибудь «загружабельного», при этом на экране ничего не отображается. Если это сканирование закончилось неудачно, то на кране отображается картинка с «вопросительным знаком» как на рис.2. Видимо разработчики MacBook-а в Apple Inc восприняли слишком буквально шутку о создателе ОС Unix Кене Томпсоне, которая звучит так:

> "Ken Thompson has an automobile which he helped design. Unlike most automobiles, it has neither speedometer, nor gas gauge, nor any of the other numerous idiot lights which plague the modern driver. Rather, if the driver makes a mistake, **a giant “?” lights up** in the center of the dashboard. “The experienced driver,” says Thompson, “will usually know what’s wrong.”"

Рис. 2. Состояние MacBook при отсутствии источников загрузки или любой другой проблеме. Опытный пользователь всегда знает что именно пошло не так.

Выяснилось, что UEFI Firmware на MacBook-ах имеет ряд технологических режимов, переход в которые осуществляется зажатием и удержанием в момент включения питания некоторых комбинаций клавиш. В недрах сайта фирмы Apple Inc нашелся краткий перечень таких комбинаций под названием «Mac startup key combinations». На всякий случай приведу его здесь полностью:

> **Key combinations for an Intel-based Mac**
>
> **Command (⌘)-R**: Start up from the built-in macOS Recovery system. Or use Option-Command-R or Shift-Option-Command-R to start up from macOS Recovery over the internet. macOS Recovery installs different versions of macOS, depending on the key combination you use.
>
> **Option (⌥)** or **Alt**: Start up to Startup Manager, which allows you to choose other available startup disks or volumes.

Option-Command-P-R: Reset NVRAM or PRAM.

Shift (⇧): Start up in safe mode.

D: Start up to the Apple Diagnostics utility. Or use Option-D to start up to this utility over the internet.

N: Start up from a NetBoot server, if your Mac supports network startup volumes. To use the default boot image on the server, press and hold Option-N instead.

Command-S: Start up in single-user mode. Disabled in macOS Mojave or later.

T: Start up in target disk mode.

Command-V: Start up in verbose mode.

Eject (⏏) or **F12** or **mouse button** or **trackpad button**: Eject removable media, such as an optical disc.

Нас на данном этапе интересует режим «Startup Manager» для того, чтобы выбрать другой источник загрузки ОС. Для перехода в этот режим необходимо сначала выключить MacBook, потом зажать клавишу **Option** (⌥ или **Alt**) и удерживая её подать питание коротким нажатием клавиши **Power**. Клавишу **Option** требуется удерживать до появления пиктограмм с описанием обнаруженных источников загрузки.

5.3. Загружаем ОС FreeBSD на MacBook с USB Flash носителя

Воспользовавшись этим знанием, я вставил подготовленный USB Flash накопитель с ОС FreeBSD 15.0 в один из USB портов MacBook-у, зажал кнопку **Option** и подал питание. Примерно через 10 секунд появилось меню состоящее из трех пиктограмм (см. рис. 3), предлагающее на выбор аж три варианта загрузки: «Macintosh HD», «Windows» и «EFI Boot».

Рис. 3. MacBook в режиме «Startup Manager», три варианта загрузки.

Так, как последние два варианта были изображены в виде пиктограммы с логотипом USB, то я предположил, что «**EFI Boot**» это старт ОС с USB носителя через UEFI загрузчик, а «**Windows**» - это, надо полагать, то же самое, но через загрузчик в MBR. Я продолжил выбрав третий вариант - «EFI Boot», и через несколько секунд увидел на экране сначала «уши» от FreeBSD **loader**-а, а потом и его стандартное меню, как на рис. 4.

Рис. 4. Традиционное меню загрузчика (loader-а) в ОС FreeBSD.

По привычке я нажал клавишу **Enter** чтобы продолжить загрузку в обычном режиме. Потирая руки и предвкушая быструю прогулку по установщику, я начал наблюдать на экране сообщения о процессе загрузки ядра:

```
Loading kernel...
/boot/kernel/kernel text=...
```

Однако радость была скоротечной. Во-первых, загрузка ядра длилась очень долго, где-то около минуты. «Спиннер», индицирующий процесс загрузки, вращался еле-еле. Учитывая, что размер дефолтного ядра ОС FreeBSD версии 15.0 занимает около 28 МБ, то скорость считывания с USB носителя составила менее 1 МБ/сек. Я предположил, что USB 2.0 порт работает в низкоскоростном режиме совместимости, а это может создать проблему при установке системы.

Во-вторых, после того как ядро ОС было загружено и ему передано управление, всякий вывод на экран прекратился. Последнее сообщение которое наблюдалось на экране выглядело так (см. фото на рис. 5):

```
Start @ 0xffffffff80387000 ...
Loading splash ok
```

По замыслу в этот момент должны были начать появляться сообщения от ядра ОС, однако-ж их нет. Визуально, все говорило о том, что система зависла. Я подождал несколько минут, в надежде, что идет незавершенный процесс чтения с медленного носителя, но ничего не произошло. Я сделал еще несколько попыток загрузки с этого же USB носителя, но всегда эффект был один и тот-же. Я пробовал записывать образ «memstick» на другие USB флешки имеющиеся в моём арсенале, но во всех случаях процесс застревал после сообщения «Loading splash ok».

После нескольких часов «плеваний под капот» я принял решение переходить к плану «Б» и достал подготовленный DVD.

Рис.5. Безуспешная попытка загрузиться установщик ОС FreeBSD 15.0 с USB носителя на старом MacBook.

5.4. Загружаем ОС FreeBSD на MacBook с DVD диска

Чтобы загрузить MacBook с DVD диска необходимо проделать следующие действия:

1. Перевести MacBook в режим «Startup Manager», зажав клавишу **Option** и подав питание. При этом, в меню будет только один пункт «Macintosh HD».

2. Вставить загрузочный DVD диск. Если с диском всё в порядке, т. е. он верно подготовлен и нет ошибок чтения (что на старом железе не всегда справедливо), то «Startup Manager» тут же распознает диск и отобразит дополнительные опции, точно такие же как и с USB флешкой: «Windows» и «EFI Boot».

Важно! Если по какой-то причине на встроенном HDD/SSD отсутствует установленная MacOS X (скажем, её уже успели уничтожить — у меня случилось именно так), то Вы будете наблюдать пустой белый экран! Пусть это не смущает Вас, «Startup Manager» готов и постоянно сканирует устройства на наличие загрузочного носителя, можно смело вставлять подготовленный DVD и он тут же распознается!

В общем, я вставил ранее подготовленный DVD диск в привод, в появившемся меню выбрал «EFI Boot» и… получил точно такой же эффект как и при загрузке с USB накопителя: сообщение «Loading splash ok» и на этом шоу заканчивается. Но я заметил, что загрузка ядра с DVD происходит достаточно быстро, поэтому решил попробовать еще один вариант: в

меню «Startup Manager-а» выбрать вариант «Windows», то есть загрузка через MBR. Этот вариант дал точно такой же результат.

Отчаявшись, я начал нажимать на клавиатуре различные клавиши, в том числе нажал несколько раз клавишу **Enter** и… случилось странное дело — DVD привод начал что-то считывать с диска, хотя на экране MacBook-а новых сообщений не появилось! «Вот те на», подумал я, «система-то работает!!!»

Выполнив еще несколько попыток загрузки с DVD и многократно потыкав в клавиатуру, я пришел к выводу, что ОС FreeBSD все таки загружается с DVD и работает, однако испытывает трудности вывода на видео-консоль (**vidconsole**). При этом клавиатура и все остальные устройства функционируют нормально. Мне даже удалось *вслепую проинсталлировать базовый комплект системы на SSD* (уничтожив при этом MacOS X). Для этого я загрузился с USB носителя на другом компьютере и выписал на лист бумаги последовательность нажатий на клавиатуру в Инсталлере, необходимой для того, чтобы запустить процесс установки. :-) Но работать, на такой машине для меня не представлялось возможным.

На лицо имеем баг! Баг который находится где-то либо в ядре ОС FreeBSD, либо в загрузчике **loader_ia32.efi**. Я собрал еще немного информации про сложившуюся ситуация и оформил баг-репорт на сайте проекта FreeBSD:

https://bugs.freebsd.org/bugzilla/show_bug.cgi?id=291935

Спойлер: Этот баг был достаточно оперативно исправлен. О том, как это происходило, я расскажу в одной из следующих глав.

Ну, а тем временем, не дожидаясь пока баг будет исправлен и выйдет обновление, что может занять месяцы, я принялся вырабатывать план «Вэ». Ход моих мыслей был таков: если в 15.0-RELEASE есть несовместимый с жизнью баг, то надо попробовать предыдущий релиз, а именно 14.3-RELEASE. Но, как мы знаем, в 14.3 есть баг в инсталляторе приводящий к копированию в систему неверного загрузчика. Поэтому я решил попробовать установить 32-х битную версию ОС FreeBSD 14.3-RELEASE/i386.

6. Установка 32-х битной FreeBSD 14.3-RELEASE на MacBook

Надо заметить, что у 32-х битной ОС на ноутбуке с 2,5 ГБ ОЗУ есть определенное преимущество перед 64-х битной. За счет более коротких указателей и выравниваний структур данных (padding) по границе 32 бита (64-х битный код требует выравнивание структур по границе 8 байт или даже по 128 байт), размер программного кода и требовательность программ к объему оперативной памяти несколько ниже, что в свою очередь снижает нагрузку на кэш. Поэтому установка 32-х битной версии ОС на машину с малым объемом ОЗУ была бы предпочтительней, но бесперспективной — как я уже отмечал, 14.х это последняя версия ОС FreeBSD в которой поддерживается архитектура i386. То есть через некоторое время всё равно придется переходить на 64-х битную ОС. Но будем надеяться, что к тому времени выявленный мной баг будет исправлен, а пока займемся установкой ОС FreeBSD 14.3-RELEASE/i386.

Для установки 32-х битной версии FreeBSD выкачиваем соответствующий «**dvd1**» образ с официального сайта проекта FreeBSD и записываем его на чистую DVD-R болванку. Можно конечно попытать счастья и с USB Flash накопителем, но как я уже отмечал, из-за нюансов работы USB портов в этом MacBook-е скорость установки будет просто негуманной и присутствует большая вероятность сбоя.

1. Скачиваем упакованный образ «**dvd1**» 32-х битной (i386) версии ОС:

```
$ wget https://download.freebsd.org/releases/ISO-IMAGES/14.3/FreeBSD-14.3-RELEASE-i386-dvd1.iso.xz
```

2. Записываем образ на DVD болванку минуя стадию распаковки.

```
# xz -d -c FreeBSD-14.3-RELEASE-i386-dvd1.iso.xz | cdrecord dev=5,0,0 /dev/fd/0
```

Здесь всё точно также как и было описано в главе «*5.1. Подготовка загрузочных носителей с ОС FreeBSD*», отличается только имя файла с образом.

Образ «**dvd1**» для i386 версии существенно отличается от аналогичного для amd64. В нём используется старая таблица разделов MBR с нулевым загрузчиком **boot0**, FAT раздел для UEFI загрузчика отсутствует, а весь образ имеет формат ISO 9660. Поэтому загрузка с этого диска будет проводиться в режиме совместимости со старым BIOS (CSM). Для перевода MacBook в этот режим не требуется каких-либо дополнительных действий, достаточно включить питание с зажатой клавишей **Option**. После того как UEFI Firmware перейдет в «Startup Manager», он просканирует содержимое DVD диска и отобразит опцию с **пиктограммой оптического диска и надписью «Windows»** - её и требуется выбрать, см. рис. 6.

Рис.6. «Startup Manager» на MacBook-е в режиме загрузки с оптического диска (CD/DVD).

После выбора этой опции с оптического диска будет считан загрузчик ОС FreeBSD и через несколько секунд на экране появился загрузочное меню подобное тому, что приведено на рис. 4. В нём выбираем первый пункт, то есть просто нажимаем **Enter** и наблюдаем за процессом загрузки и запуска ядра ОС.

На удивление, тут процесс загрузки ядра ОС проходит без запинки, через 20-30 секунд система загружается и запускается утилита инсталлера **bsdinstall** (FreeBSD Installer), общий вид главного меню которой приведен на рис 7.

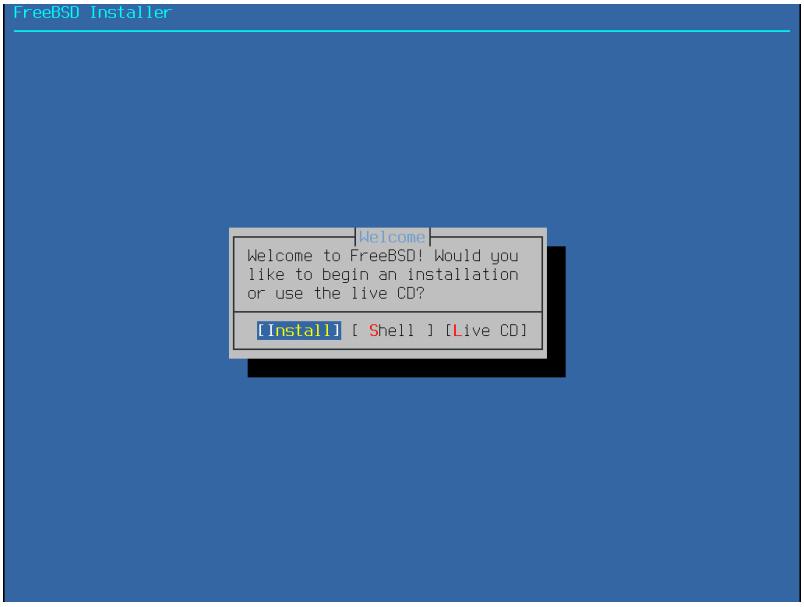


Рис. 7. Главное меню утилиты установщика «FreeBSD Installer».

Для установки системы необходимо выбрать пунт «**[Install]**», после чего утилита установщика задаст ряд вопросов касаемых локализации, попросит указать имя хоста, задать настройки сетевого интерфейса и т.д.

Подробную инструкцию о начальной установки ОС FreeBSD я уже описывал с статьях «*Операционная система FreeBSD на ноутбуке Lenovo*» и «*Чёрт в табакерке: инструмент для диагностики сети на базе ОС FreeBSD*». Также во FreeBSD Handbook имеется целая глава «*Глава 2. Установка FreeBSD*» на русском языке, посвященная процессу начальной установки этой ОС. Можно смело воспользоваться приведенной там информацией, но с некоторыми поправками:

1. При установке на MacBook 32-х битной версии ОС FreeBSD следует использовать формат таблицы разделов MBR, так как загружаться система будет в режиме совместимости с BIOS. Таблица GPT в данном режиме не поддерживается!

2. Использовать формат файловой системы UFS (**freebsd-ufs**). Файловая система ZFS, предлагаемая по умолчанию, слишком требовательна к ресурсам (прежде всего к ОЗУ) и мало пригодна для таких стареньких машин.

Замечу, что утилита **bsdinstall** это исключительно консольное (текстовое) приложение, она проведет установку *базового комплекта операционной системы*: загрузчик, ядро ОС, драйверы и небольшой набор стандартных текстово-командных утилит как в любой UNIX-подобной ОС. Не ожидайте, что Вы сразу получите графический оконный интерфейс, всё на что можно рассчитывать после перезагрузки системы — это приглашение к входу в систему в виде строки «**login:**».

Про настройку X11 и и графического оконного менеджера мы поговорим в следующей главе, а на данном этапе обходимо:

1. Установить локаль: **Russia**.
2. Установить раскладку клавиатуры: **ru.kbd**.

3. Назначить имя хоста для данной машины (любой набор латинских символов).
4. Выбрать и установить дополнительные системные компоненты: **ports** и **src**.
5. Выбрать разбивку диска «**Auto (UFS)**» на весь диск (**Entire Disk**).
6. Выбрать тип раздела **MBR (DOS Partitions)**.
7. Настроить сетевой интерфейс: WiFi (**wlan0**) или GigabitEthernet (**msk0**), назначить IP адрес (или включить DHCP).
8. Дождаться завершения установки системных компонентов.
9. Задать пароль пользователя **root** и не забыть его!
10. Выбрать часовой пояс. Часовые пояса для России находится как в **Europe**, так и в **Asia**.
11. Включить **ntpd, powerd** и **sshd** — инсталлятор предложит это сделать на одном из завершающих этапов.
12. Никакой «усиленной безопасности» (**System Hardening**) включать не стоит!
13. Создать нового пользователя (например **user**) и не забыть пароль от него. При создании пользователя добавить его группу **wheel** — это группа системных администраторов. Если этого не сделать, то пользователь **user** не сможет выполнять команду **su**.
14. Выйти из Инсталлера (меню **[Exit]**) и отправить систему на **[Reboot]**.

В моей версии MacBook-а установлен следующий адаптер проводной сети GigabitEthernet:

```
mskc0: <Marvell Yukon 88E8053 Gigabit Ethernet> port 0x3000-0x30ff mem
0xb0200000-0xb0203fff irq 16 at device 0.0 on pci1
msk0: <Marvell Technology Group Ltd. Yukon EC Id 0xb6 Rev 0x02> on mskc0
msk0: Using defaults for TSO: 65518/35/2048
msk0: Ethernet address: 00:1b:63:34:0b:a8
```

А также адаптер беспроводной сети WiFi для диапазонов 2.4/5 GHz:

```
ath0: <Atheros 5418> mem 0xb0100000-0xb010ffff irq 17 at device 0.0 on
pci2
ath0: [HT] enabling HT modes
ath0: [HT] RTS aggregates limited to 8 KiB
ath0: [HT] 2 RX streams; 2 TX streams
ath0: AR5418 mac 12.10 RF5133 phy 8.1
ath0: 2GHz radio: 0x0000; 5GHz radio: 0x00c0
```

Оба сетевых адаптера работают в ОС FreeBSD 14.3/i386 что называется «из коробки», никаких дополнительных драйверов устанавливать от меня не потребовалось.

После того, как система отправлена на перезагрузку, необходимо вынуть из MacBook-а оптический диск с которого была выполнена установка системы. Для этого необходимо выключить компьютер, зажать клавишу **Eject** и подать питание. После чего загрузить компьютер обычным способом без зажимания каких либо клавиш.

И тут мы попадаем на еще один неприятный нюанс! У нас на диске теперь установлена «устаревшая» (legacy), с точки зрения UEFI Firmware, операционная система, которая требует выполнение загрузки через эмуляцию BIOS (режим CSM). UEFI Firmware на MacBook-е, как мы уже знаем, осуществляет поиск загрузочного носителя по наличию FAT раздела с подкаталогом **/EFI/BOOT/**. В нашем случае ничего подобного на диске нет. UEFI Firmware на MacBook-е будет в течении 30 секунд выполнять непрерывное сканирование источников загрузки в надежде на то, что появится UEFI совместимый загрузочный

носитель. Если через 30 сек этого не происходит, то UEFI Firmware автоматически переходит в режим CSM и начинает считывать загрузчик из MBR. Всё это означает, что перед началом загрузки 32-х битной версии FreeBSD на MacBook-е мы будем в течении 30 секунд наблюдать пустой белый кран. Это весьма неприятная особенность UEFI Firmware в MacBook-ах. О том, можно ли как-то переключить настройки «фирмвари» в CSM режим по умолчанию, мне не известно. Однако, есть способ обойти этот нюанс путем установки промежуточного менеджера загрузки «**rEFInd**», о котором я рассказывал в одной из предыдущих глав.

7. Донастройка системы и установка полезных пакетов программ

После того, как недавно установленная ОС FreeBSD загрузится с диска, пользователю представляется приглашение для входа в систему в виде промпта «**login:**» на текстовой консоли. Набрав имя пользователя **root** и введя пароль, который был указан при установке, пользователь попадает в оболочку командной строки — шелл (промпт # означает что у вас права суперпользователя). С этого момента пользователь волен делать с системой всё, что ему захочется. Однако есть некоторый список действий который следует предпринять пользователю в первую очередь.

7.1. Настройка демона `ntpd’ для синхронизации реального времени

Так как мы имеем дело с достаточно старым ноутбуком, то с очень большой вероятностью у него есть проблемы с часами реального времени (Real-Time Clock). RTC — это такая микросхема, расположенная на материнской плате, которая постоянно ведет отсчет времени, даже когда компьютер полностью обесточен. Чтобы часы RTC работали во время отсутствия внешнего питания компьютера, у них имеется своя маленькая батарейка (аккумулятор или супер-конденсатор). Если RTC часы испытывают проблемы с питанием, то они будут показывать неправильное время — скорее всего каждый раз после включения питания на компьютере будет устанавливаться одна и та же дата (в моем случае это 1 января 2001 года). За 15 лет источник питания RTC приходит в полную негодность и требует физической замены. На просторах Ютуба есть немало видео-инструкций как выполнить замену батареи часов RTC самостоятельно, достаточно загуглить по фразе «MacBook changing CMOS battery», но сейчас мы этот момент рассматривать не будем. Нам достаточно сделать так, чтобы демон **ntpd**, следящий за синхронизацией времени в системе, правильно выполнил синхронизацию после загрузки ОС.

Напомню, что при базовой установки системы мы включили поддержку **ntpd**, как раз для того, чтобы он при запуске системы обращался по сети к источникам точного времени (их называют «Clock Stratum») и устанавливал на машине корректное время. Но этого оказывается не достаточно. Дело в том, что у **ntpd** есть защитный механизм предупреждающий некорректную синхронизацию времени в случае, если расхождение между тем, что есть на машине и тем что он получил по сети от стратума составляет **более 1000 сек**. Так, как RTC при включении машины всегда сбрасываются на какую-то дату в глубоком прошлом, то при загрузке ОС всегда образуется гигантское расхождение во времени, что воспринимается демоном **ntpd** как ошибку и он умышленно не корректирует данные в RTC и не изменяет системные часы. Отсутствие правильного времени (и даты) в системы, в свою очередь приводит к тому, что все SSL сертификаты становятся автоматически невалидными, а это означает, что мы получим массу проблем при работе с

сайтами в сети Интернет — ведь все современные сервисы в интернете работают только по SSL/TLS.

Чтобы обойти это недоразумение, у **ntpd** есть специальная опция (флаг командной строки) **-g** , она отключает вот эту вот защиту в 1000 секунд. Ниже небольшая выдержка из руководстве по **ntpd**:

```
root@sunny:~ # man ntpd

NTPD(8)             FreeBSD System Manager's Manual (user)             NTPD(8)

NAME
     ntpd – set clock via Network Time Protocol daemon

SYNOPSIS
     ntpd [-flags] [-flag [value]] [--option-name[[=| ]value]] [ <server1> ...
          <serverN> ]

...skipping...
     -g, --panicgate
             Allow the first adjustment to be Big.  This option may appear an
             unlimited number of times.

             Normally, ntpd exits with a message to the system log if the
             offset exceeds the panic threshold, which is 1000 s by default.
             This option allows the time to be set to any value without
             restriction; however, this can happen only once. If the threshold
             is exceeded after that, ntpd will exit with a message to the
             system log. This option can be used with the -q and -x options.
             See the tinker configuration file directive for other options.
```

Таким образом нам требуется установит флаг **-g** в переменной **ntpd_flags** в файле **/etc/rc.conf**. Сделать это можно следующей командой:

```
root@sunny:~ # sysrc ntpd_flags="-g" && sysrc ntpd_enable="YES"
ntpd_flags:  -> -g
ntpd_enable: NO -> YES
```

После чего можно перезапустить **ntpd**, чтобы он тут же синхронизировал время:

```
root@sunny:~ # date
Mon Jan  1 00:01:17 +05 2001

root@sunny:~ # service ntpd restart
ntpd not running? (check /var/db/ntp/ntpd.pid).
Starting ntpd.

root@sunny:~ # date
Thu Jan 15 01:03:11 +05 2026
```

7.2. Настройка пакетного менеджера `pkg’

Установка программного обеспечения на ОС FreeBSD лучше всего выполнять из репозитория используя встроенный пакетный менеджер **pkg**. Это системная утилита с помощью которой можно найти (**pkg search**), установить (**pkg install**), запросить информацию (**pkg info**) или удалить (**pkg delete**) любой из более чем 36000 пакетов программ, расположенным в репозитории проекта FreeBSD.

Утилита **pkg** имеет свой набор конфигурационных файлов, главный из которых является **/etc/pkg/FreeBSD.conf**. Важным моментом является здесь то, что репозиториев на самом деле много, как минимум по два для каждой версии релиза ОС. Первый (по

умолчанию) называется **quarterly** — он обновляется раз в три месяца, и второй — **latest**, с обновлением в несколько дней. Смысл тут в том, что не все обновления программного обеспечения одинаково полезны. Часто бывает так, что последние обновления (которые сразу попадают в **latest**) могут оперативно исправить какую-нибудь уязвимость в безопасности, но принесут с собой ряд других ошибок или неприятных артефактов. Поэтому, чтобы у пользователя была возможность а) откатиться на предыдущий вариант пакета, и б) не испытывать на себе все багфиксы, существует репозиторий **quarterly**.

Но бывает и так, что необходимо срочно установить самую последнюю версию пакета, не дожидаясь пока он отлежится в «отстойнике» и попадет в **quarterly**. Для этого можно временно переключиться на репозиторий **latest** подправив указанный конфигурационный файл. Но не стоит часто смешивать пакеты из различных репозиториев, это может привести к нарушению цепочки зависимостей. Об этом моменте необходимо помнить каждому пользователю ОС FreeBSD.

Посмотрим как выглядит конфиг для утилиты **pkg**:

```
root@sunny:~ # cat /etc/pkg/FreeBSD.conf

FreeBSD: {
  url: "pkg+https://pkg.FreeBSD.org/${ABI}/quarterly",
  mirror_type: "srv",
  signature_type: "fingerprints",
  fingerprints: "/usr/share/keys/pkg",
  enabled: yes
}
FreeBSD-kmods: {
  url: "pkg+https://pkg.FreeBSD.org/${ABI}/kmods_quarterly_${VERSION_MINOR}",
  mirror_type: "srv",
  signature_type: "fingerprints",
  fingerprints: "/usr/share/keys/pkg",
  enabled: yes
}
```

Здесь мы видим описание двух репозиториев с именами «**FreeBSD»** и «**FreeBSD-kmod»**, оба ссылаются на вариант **quarterly**. Репозиторий с именем «**FreeBSD**» содержит пакеты программ, а «**FreeBSD-kmod**» — драйвера (kernel modules) импортированные из ядра ОС Linux. Этот еще один репозиторий, кстати, появился недавно, с версии 14.0-RELEASE.

После того как в конфиге проделаны изменения, а также перед первым использованием утилиты **pkg** после установки новой системы, необходимо выполнить синхронизацию каталогов командой **pkg update**:

```
root@sunny:~ # pkg update

Updating FreeBSD repository catalogue...
Fetching data.pkg: 100%   10 MiB   1.8MB/s    00:06
Processing entries: 100%
FreeBSD repository update completed. 36472 packages processed.
Updating FreeBSD-kmods repository catalogue...
Fetching data.pkg: 100%   36 KiB  36.7kB/s    00:01
Processing entries: 100%
FreeBSD-kmods repository update completed. 245 packages processed.
All repositories are up to date.
```

Теперь можно смело пользоваться утилитой **pkg** и устанавливать пакеты требуемых нам программ и утилит.

7.3. Установка и настройка демона управления питанием `powerd++’

В ОС FreeBSD существует минимум два демона для управления питанием и частотами работы центрального процессора. По умолчанию в системе присутствует демон **powerd**. Он неплохо справляется с задачей, но на мой взгляд, для ноутбуков, особенно с «уставшей» батареей, лучшим вариантом будет установить и использовать демона **powerdxx**. Сделать можно так:

Установить **powerdxx**:

```
root@sunny:~ # pkg install powerdxx
```

После чего следует включить его автоматический запуск при загрузке системы:

```
root@sunny:~ # sysrc powerdxx_enable="YES" && sysrc powerdxx_flags="-a maximum -b adaptive"
```

Отключить демон **powerd**, так как он был ранее включен при установке:

```
root@sunny:~ # sysrc powerd_enable="NO"
```

Остановить **powerd** и запустить **powerdxx**:

```
root@sunny:~ # service powerd stop

root@sunny:~ # service powerdxx start
```

7.4. Настройка утилиты `fsck’ на полную проверку файловой системы

Опять же, так как мы имеем дело со старым ноутбуком с изрядно «уставшей» батареей, то имеется большая вероятность произвольного пропадания электропитания, что может вызвать сбои в файловой системе. Поэтому имеет смысл указать системной утилите **fsck**, чтобы она выполняла полную проверку файловой системы каждый раз при загрузке ОС и устраняла неполадки до того, как файловая система будет смонтирована в режиме read/write. Это немного замедлит процесс загрузки, но зато избавить от массы неприятностей вызванных сбойной файловой системой. Для этого выполним следующие команды:

```
# Force a full file system check
root@sunny:~ # sysrc fsck_flags="-f"

# Set to YES to do fsck -y if the initial preen fails.
root@sunny:~ # fsck_y_enable="YES"

# Additional flags for fsck -y to reduce to 10 restarts on failure
root@sunny:~ # fsck_y_flags="-T ffs:-R -T ufs:-R"
```

Теперь можно смело дергать электропитание компьютера. Вероятность того, что произойдет что-то очень страшное с файловой системы, крайне низка. Но не стоит зря испытывать судьбу. :-)

7.5. Настройка встроенного тач-пада на MacBook

В компьютерах производства Apple Inc используется свой вариант тач-пада, который нам может понадобиться как для работы в консоли, так и для графической подсистемы.

Чтобы его задействовать необходимо во-первых, подгрузить ряд драйверов, и во-вторых, указать утилите **devmatch**, которая запускается при загрузке системы, на то, что **не** следует загружать дайвер **atp.ko**. Иначе мышь (тач-пад) не будет работать в графической подсистеме X11, так как этот драйвер перехватит управление на себя.

Для этого последовательно выполним следующие команды:

```
root@sunny:~ # sysrc moused_enable="YES"
root@sunny:~ # sysrc moused_port="/dev/ums1"
root@sunny:~ # devmatch_blocklist="atp.ko"
root@sunny:~ # sysrc devmatch_blocklist="atp.ko"
root@sunny:~ # sysrc kld_list+="ig4 iichid"
```

После этого можно перезагрузить систему командой **reboot** и проверить как работает тач-пад в консоли.

7.6. Настройка автоматического монтирования USB носителей

В UNIX-подобных операционных системах принято монтировать файловые системы, перед их использованием, вручную. Это создает некоторые неудобства для новых пользователей привыкших вставлять USB носители в свой компьютер и тут же копировать файлы методом «drag-n-drop» из распахнувшейся папки. Чтобы приблизить опыт пользователя FreeBSD к такому стилю, в репозитории имеется утилита под названием **automount**. После её установки, система (демон **devd**) будет детектировать подключение к компьютеру новых устройство и автоматически запускать процедуру монтирования файловой системы с установленного носителя.

Утилита **automount** зависит от пакета **fusefs** — представляющий собой набор «драйверов» для различных файловых систем.

Для установки **automount** выполним следующий набор команд:

```
root@sunny:~ # pkg install -y fusefs automount
root@sunny:~ # sysrc kld_list+="fusefs"
root@sunny:~ # sysrc -f /usr/local/etc/automount.conf FAT_CODEPAGE="cp1251"
root@sunny:~ # sysrc -f /usr/local/etc/automount.conf FAT_ENCODING="ru_RU.UTF-8"
root@sunny:~ # service devd restart
```

После этого можно смело вставлять USB носитель. Файловые системы с них будут доступны в подкаталоге **/media**. А после того, как мы установим **X11** и оконный менеджер **xfce4**, на «Рабочем Столе» будет автоматически появляться папка с содержимым «флэшки».

7.7. Установка средств разработки

Для решения ряда проблем, описываемых в последующих главах, нам придется прибегнуть к компиляции драйверов из исходных кодов, а для этого потребуются компиляторы GCC и LLVM, а также некоторые утилиты разработчика. Установим их все следующей командой:

```
root@sunny:~ # pkg install -y llvm gcc git gmake portsnap sudo vim
```

Теперь можно приступить к установке того, что больше всего хочется иметь пользователю UNIX-подобной ОС на своём ноутбуке.

8. Устанавливаем и настраиваем графический интерфейс на базе X11

А хочется обычному рядовому пользователю, как правило, графического интерфейса и красивых окошек с Web-браузером **Firefox**, или, на худой конец, с **Chromium**. Поэтому сейчас мы займемся установкой и настройкой графической подсистемы «X Window System» (её еще принято называть **X11,** по одноименному протоколу).

8.1. Установка пакета драйверов DRM-KMOD для графических адаптеров в 32-х битной FreeBSD 14.3.

В MacBook-ах той эпохи использовались встроенные в процессор видео-адаптеры серии **Intel i9xx**. В ОС FeeBSD имеется отдельный модуль (драйвер) называемый **i915kms.ko** предоставляющий прямой доступ к аппаратуре рендеринга изображения (Direct Rendering Manager) для большого набора совместимых адаптеров. В моём макбуке оказался именно такой видео-адаптер:

```
root@sunny:~ # pciconf -lv | grep -A 4 vgapci0

vgapci0@pci0:0:2:0:     class=0x030000 rev=0x03 hdr=0x00 vendor=0x8086 device=0x27a2
subvendor=0x8086 subdevice=0x7270
    vendor     = 'Intel Corporation'
    device     = 'Mobile 945GM/GMS, 943/940GML Express Integrated Graphics Controller'
    class      = display
    subclass   = VGA
```

Модуль **i915kms.ko** входит в состав пакета **drm-kmod** — того самого набора драйверов заимствованных из ОС Linux и обернутых с помощью LinuxKPI. Установить пакет **drm-kmod** всегда можно было командой **pkg install drm-kmod**, но тут мы сталкиваемся с еще одним нюансом — в версии FreeBSD 14.3/i386 больше не поддерживается сборка этого пакета! Если попытаться собрать этот пакет в портах (port collection) из исходников, то сборочный скрип выдаст сообщение о том, что в i386 версии FreeBSD поддержка данного пакета прекращена с версии 14.2-RELEASE, и предлагается откатиться либо на 14.1-RELEASE, либо перейти на 64-х битную версию ОС. Встают два извечных вопроса: «кто виноват ?» и «что делать ?».

Озадаченный этой проблемой я написал пост на форуме и задал прямой вопрос: «*Will we have drm-kmod driver (i915kms.ko) for 14.3-RELEASE/i386 ?*», на что получил весьма уклончивый ответ от одного из мантейнеров видео-драйверов, привожу его полностью:

```
Ask about it to Linux guys. Not to FreeBSD guys.

These are ported from Linux, and strongly (mandatorily) tied to LinuxKPI on base.
Even on same series of Linux kernel versions (5.10.*, 5.15.*, 6.1.*, ...), its KBI / KPI
easily changes (means, "moving goals") and once DRM / KMS drivers start depending on
modified newer version, LinuxKPI on base need to chase it, but only for not-yet-released *-
RELEASEs (as it breaks KPI / KBI).

Not sure symbol versioning works fine on Linux(KPI) world, but even if it's possible, as far
```

```
as I know, graphics team on FreeBSD is tooooooooooooooooooooo small to maintain every
versions.

And you'll see a bunch of Linux distros start dropping i386 (aka x86 without _64).
```

Краткий перевод: «*Все вопросы по линуксовым драйверам во FreeBSD задавай на линуксоидных форумах, и вообще i386 давно мертв!*».

Разумеется меня такой ответ не удовлетворил и я решил слегка исследовать вопрос самостоятельно. Ход моих мыслей был следующий. С одной стороны разницы в LinuxKPI между версией 14.1 и последующих релизов 14.x быть **не может.** Таковы правила игры — не менять API в пределах одной major ветки. С другой, пакет **drm-kmod** в стабильных ветках FreeBSD собирается из драйверов вынутых из ядра Linux **5.10-tls**. Эти драйверы не изменялись уже много лет! Я это знаю потому, что в 2021 году собирал драйвер **amdgpu.ko** для версии FreeBSD-13.0 которую я тогда установил на свой новенький ноутбук Lenovo. Мне пришлось слегка пропатчить его добавив PID/VID для «**Cezanne Radeon Vega Series / Radeon Vega Mobile Series**», и у меня всё получилось — драйвер заработал со встроенной GPU и работает по сей день, при том, что система уже итеративно проапгрейдилась до 13.5. Иными словами, драйверы из пакета **drm-kmod** прекрасно собираются и есть полная совместимость с новыми версиями ОС в пределах одного major номера!

Что-ж, надо проверить, вдруг всё это справедливо и для версии 14.x ? Короче, клонируем исходник **drm-kmod** с репозитория на Github-е, выбираем ветку **5.10-lts** и собираем:

```
root@sunny:~ # pkg insall -y git

root@sunny:~ # git clone https://github.com/freebsd/drm-kmod.git
Cloning into 'drm-kmod'...

root@sunny:~ # cd drm-kmod

root@sunny:~/drm-kmod # git checkout 5.10-lts
Updating files: 100% (3152/3152), done.
branch '5.10-lts' set up to track 'origin/5.10-lts'.
Switched to a new branch '5.10-lts'

root@sunny:~/drm-kmod # make -j2
```

И наблюдаем как через несколько минут процесс сборки успешно завершается с сообщением:

```
...
awk -f /usr/src/sys/conf/kmod_syms.awk i915kms.ko  export_syms | xargs -J% objcopy %
i915kms.ko
objcopy --strip-debug i915kms.ko
```

а собранные драйверы готовы к установке в систему. Установим их командой:

```
root@sunny:~/drm-kmod # make install
```

И проверим, что файл **i915kms.ko** действительно установился и находится там где ему положено, а именно — в каталоге **/boot**:

```
root@sunny:~/drm-kmod # find /boot | grep i915kms
/boot/modules/i915kms.ko
```

Драйвер на месте! Попробуем его тут же подгрузить, пока он еще горяченький:

```
root@sunny:~/drm-kmod # kldload i915kms.ko
```

Драйвер загружается и на консоле мы видим сообщения:

```
[drm] Got Intel graphics stolen memory base 0x9f000000, size 0x1000000
drmn0: <drmn> on vgapci0
vgapci0: child drmn0 requested pci_enable_io
vgapci0: child drmn0 requested pci_enable_io
drmn0: [drm] Failed to find VBIOS tables (VBT)
drmn0: [drm] applying backlight present quirk
drmn0: [drm] Initialized overlay support.
[drm] Initialized i915 1.6.0 20201103 for drmn0 on minor 0
```

Судя по последней строке, а также по тому, что в консоле изменился шрифт, драйвер успешно запустился, обнаружил и проинициализировал видео-адаптер. Давайте попробуем установить и запустить X-сервер и проверить что у нас получилось.

Но перед этим добавим драйвер **i915kms.ko** в список **kld_list** в файле **/etc/rc.conf**, чтобы он подгружался автоматически при загрузке ОС:

```
root@sunny:~/drm-kmod # sysrc kld_list+="i915kms"
kld_list: ig4 iichid fusefs -> ig4 iichid fusefs i915kms
```

Ну и на всякий случай перезагрузим систему, чтобы убедиться в том, что драйвер запускается при загрузке:

```
root@sunny:~/drm-kmod # reboot
```

А после перезагрузки снова потребуется залогиниться в систему под пользователем **root**.

8.2. Установка и первый запуск X-сервера

X-сервер это приложение (программа) которая обеспечивает отрисовку изображения на экране компьютера по получаемым от X-клиентов потоку команд. X-сервер это главный компонент в графической подсистеме «X Window System»: с одной стороны он взаимодействует с X-клиентами которые подключаются к нему через TCP, UDP или Unix-сокеты, с другой — с драйвером видео-адаптера. X-клиентами являются все остальные привычные пользователю приложения осуществляющие вывод в графическую подсистему (пример, web-браузер Firefox или приложение для векторной графики Inkscape). Взаимодействуют X-клиенты с X-сервером посредством протокола X11.

Самым распространенным опенсорсным X-сервером на данный момент является проект «X.Org Server» (или **Xorg**). Существует еще один опенсорсный X-сервер набирающий популярность — Xlibre, но в репозитории FreeBSD он появился совсем недавно, и, на мой взгляд, еще несколько сыроват. Так, что устанавливать будем Xorg, что можно сделать одной командой:

```
root@sunny:~ # pkg install -y xorg
```

К X-серверу **Xorg** нам потребуется его внутренний драйвер **xf86-video-intel** выступающий своего рода «прокладкой» между **Xorg** и драйвером видео-адаптера **i915kms.ko**:

```
root@sunny:~ # pkg install -y xf86-video-intel
```

На само деле **Xorg** уже давно умеет взаимодействовать с DRM драйверами напрямую, без драйверов-прокладок. Но опыт показал, что конкретно на этой модели MacBook-а с видео-адаптером «Intel Mobile 945GM/GMS, 943/940GML» у такой схемы имеется ряд не очень приятных артефактов: во-первых, не рендерятся многие шрифты, во-вторых, не рендерится аппаратный курсор (указатель мыши). При работе через **xf86-video-intel** таких проблем не возникает.

Чтобы запустить X-сервер, то есть начать X-сессию и проверить таким образом работоспособность видео-драйвера, необходимо выполнить простую команду:

```
root@sunny:~ # startx
```

Если с драйвером всё в порядке, то консоль переключиться в графический режим и на экране будут отображены несколько окон с приглашения от шелла. Окна будут выглядеть очень примитивно, так как по умолчанию используется аскетичный менеджер окон под названием **twm**.

Для того, чтобы убедится в том, что **Xorg** сейчас работает с драйвером **i915kms.ko**, а не с каким-то другим (**Xorg** может работать с эмуляцией VESA — это самый медленный, но универсальный, из способов доступа к видео-адаптеру), необходимо в любом из открывшихся графических окон терминала ввести команду:

```
root@sunny:~ # xrandr --listproviders
Providers: number : 1
Provider 0: id: 0x43 cap: 0xb, Source Output, Sink Output, Sink Offload crtcs: 2 outputs: 3
associated providers: 0 name:Intel
```

Если в качестве **name** выдается **Intel**, значит всё в порядке.

Чтобы закончить X-сессию необходимо либо закрыть все X-окна, либо переключившись обратно в текстовую консоль одновременным нажатием клавиш **Ctrl+Option (Alt)+F1**, завершить процесс нажатием **Ctrl-C**.

Если с видео-драйвером возникли проблемы (выбран неверный провайдер или **Xorg** отказался запускаться), необходимо заглянуть в файл **/var/log/Xorg.0.log**, попытаться выяснить в чем причина и постараться собрать как можно больше информации. На форуме https://forums.freebsd.org/ всегда помогут советом.

Последним штрихом в настройке X-сервера является необходимость добавить пользователей системы, которым будет разрешен доступ к графической подсистеме «X Window», в группу **video**. Если этого не сделать, то для таких пользователей **Xorg** всё еще будет запускаться, но только в режиме VESA и не будет иметь прямого доступа к DRM, как результат — графика будет отрисовываться очень медленно! Особенно OpenGL.

Напомню, что при базовой установке системы, в утилите **bsdinstall** мы создавали пользователя **user.** Добавим этого пользователя в группу **video** следующей командой:

```
root@sunny:~ # pw groupmod video -m user
```

Эту команду необходимо повторять для каждого нового пользователя системы!

Таким образом мы убедились, в том, что драйвер **i915kms.ko** из пакета **drm-kmod** собирается без ошибок, а главное - замечательно работает в **i386** версии FreeBSD 14.3-RELEASE. В чем была истинная причина того, что поддержку **drm-kmod** убрали из 14.2-RELEASE и старше — мне непонятно, но полагаю причина была достаточно весомой.

8.3. Установка менеджера логинов `sddm’

Для того, чтобы вход в систему можно было осуществлять через графический интерфейс, необходимо установить приложение (X-клиента), обеспечивающее авторизацию пользователей и создание X-сессий. Такие приложения принято называть Display или Login Manager. Существует несколько таких приложений, самое популярное из них это **sddm —** «Simple Desktop and Display Manager». Установить его можно командой:

```
root@sunny:~ # pkg install -y sddm
```

Чтобы sddm автоматически запускался при загрузке системы необходимо добавить строку **sddm_enable="YES"** в файл **/etc/rc.conf**:

```
root@sunny:~ # sysrc sddm_enable="YES"
```

Можно сразу же запустить **sddm** командой:

```
root@sunny:~ # service sddm start
```

В этот момент консоль переключится в графический режим где **sddm** предложит войти в систему, выбрать пользователя из списка и ввести пароль. Если сейчас произвести вход в систему под пользователем **user**, то отобразятся всё те же три минималистичных окна с шеллом как и при запуске X-сессии командой **startx**. Закрытие всех окон приведет к завершению сессии, но не завершению **sddm**, он просто вернется к окну ввода пароля.

Вернуться обратно в текстовую консоль можно, как и прежде, нажав одновременно клавиши: **Ctrl+Option (Alt)+F1**. Переключиться из текстовой консоли в графический режим можно нажатием: **Option (Alt)+F9.**

Чтобы временно остановить работу графической подсистемы, можно ввести следующую команду:

```
root@sunny:~ # service sddm stop
```

8.4. Обходим баг в `sddm-greeter-qt6’

В одном из компонентов **sddm**, а именно в **sddm-greeter-qt6**, имеется пренеприятнейший артефакт: когда **sddm** находится в пассивном состоянии, а на экрана отображается окно приветствия с приглашением ввести пароль (т. н. «greeter»), то процесс

sddm-greeter-qt6 потребляет очень много процессорного времени, в утилите **top** это выглядит так:

```
root@sunny:~ # top -b 4

last pid:  1643;  load averages:  0.40,  1.13,  0.70; battery: -1%
up 0+00:09:10  01:56:53
37 processes:  1 running, 36 sleeping
CPU: 15.6% user,  0.0% nice,  0.5% system,  0.0% interrupt, 84.0% idle
Mem: 103M Active, 58M Inact, 161M Wired, 85M Buf, 2140M Free
Swap: 3881M Total, 3881M Free

  PID USERNAME    THR PRI NICE   SIZE    RES STATE    C   TIME    WCPU COMMAND
 1637 sddm         10  29    0   328M   149M select   1   0:05  31.67% sddm-greeter-qt6
 1630 root          4  26    0   158M    73M select   0   0:01   0.97% Xorg
 1643 root          1  20    0  6736K  3096K CPU1     1   0:00   0.24% top
 1408 root          1  20    0  6032K  2908K nanslp   1   0:01   0.11% powerd++
```

Вызвано такое поведение **sddm** тем, что при отрисовке поля ввода пароля происходит непрерывный рендеринг мигающего курсора с большой частотой кадров, что сильно нагружает процессор большим потоком обращений к аппаратуре. Этот баг известен уже десяток лет и никто из разработчиков не хочет его исправлять. Вот, что об этом говорят пользователи на сайте *askubuntu.com*:

```
sddm-greeter uses this much CPU for cursor blinking in the password field. Use [Tab] button
on keyboard to move away from the password field. Cursor will disappear from the password
field and CPU load become less then 1% of 1 CPU thread.

answered Apr 24, 2023 at 3:17 Mikhail M

To anyone thinking this answer was a belated April Fools, it was not. It does sound like a
joke, but it's true: sddm-greeter really grinds on full core to blink the cursor, and the
bug remains ignored by devs and unfixed for years now. –

Szczepan Hołyszewski Commented Dec 15, 2024 at 9:48
```

https://askubuntu.com/questions/1403355/sddm-greeter-cpu-usage-lxqt

Выяснилось, что имеется достаточно простой способ избавить **sddm** от такого поведения: необходимо запретить ему использовать аппаратный рендерер. Делается это добавлением одной строки в конфигурационный файл **/usr/local/etc/sddm.conf**:

```
root@sunny:~ # echo GreeterEnvironment=QT_QUICK_BACKEND=software >> /usr/local/etc/sddm.conf

root@sunny:~ # cat /usr/local/etc/sddm.conf

# Disable virtual keyboard by default
InputMethod=""
GreeterEnvironment=QT_QUICK_BACKEND=software
```

После чего необходимо перезапустить **sddm** командой:

```
root@sunny:~ # service sddm restart
```

8.5. Устанавливаем оконный менеджер `xfce4’

Настало время установить оконный менеджер поприличнее. Я, как и многие пользователи FreeBSD, предпочитаю использовать оконный менеджер **xfce4**. Он достаточно

прост, удобен, гибок в настройках и не так требователен до ресурсов как KDE или GNOME. Но самое главное — **xfce4** стабилен и не претерпевает серьезных изменений с каждой новой версией.

Перед установкой **xfce4** необходимо установить и настроить подсистему обмена сообщений **dbus**:

```
root@sunny:~ # pkg install -y dbus
root@sunny:~ # dbus_enable="YES"
root@sunny:~ # service dbus start
```

Теперь можно установить **xfce4** и различные полезные плагины для него:

```
root@sunny:~ # pkg install -y xfce
root@sunny:~ # pkg install -y xfce4-screensaver xfce4-xkb-plugin xfce4-pulseaudio-plugin
root@sunny:~ # pkg install -y xfce4-power-manager xfce4-clipman-plugin
```

Установим немного полезных программ:

```
root@sunny:~ # pkg install -y firefox-esr far2l libreoffice inkscape
```

Установим немного бесполезных, но занимательных игр:

```
root@sunny:~ # pkg install -y kgoldrunner supertux2 dreamchess chromium-bsu
```

И перезапустить **sddm** командой:

```
root@sunny:~ # service sddm restart
```

Теперь, при входе в систему через **sddm**, в верхнем левом гулу экрана появится возможность выбрать тип сессии: "**xfce4-session**".

8.6. Настройка переключателя раскладок клавиатуры в `xfce4’

Для того, чтобы в **xfce4** была возможность переключаться между кириллицей и латинским алфавитом, необходимо в файл конфигурации **~/.config/xfce4/xfconf/xfce-perchannel-xml/keyboard-layout.xml**, находящийся в домашнем каталоге пользователя **user**, добавить следующие строки:

```
<?xml version="1.1" encoding="UTF-8"?>

<channel name="keyboard-layout" version="1.0">
  <property name="Default" type="empty">
    <property name="XkbDisable" type="bool" value="false"/>
    <property name="XkbModel" type="string" value="asus_laptop"/>
    <property name="XkbOptions" type="empty">
      <property name="Group" type="string" value="grp:caps_toggle"/>
    </property>
    <property name="XkbLayout" type="string" value="us,ru"/>
    <property name="XkbVariant" type="string" value=",legacy"/>
  </property>
```

```
</channel>
```

И перезапустить сессию (выйти и войти, или перезапустить **sddm**). После чего переключение раскладок клавиатуры будет доступно по клавише **CapsLock**.

9. Установка системы печати и настройка принтера HP LaserJet

Так как данный MacBook планируется использовать в учебных целях для школьницы, то без вывода на печать он становится практически бесполезен. У меня в хозяйстве находятся несколько сетевых принтеров производства HP. Один из них, «HP LaserJet P1102w», мы подключим и настроем печать на него из любых приложений.

Организация печати в ОС FreeBSD, как и во многих других современных UNIX-подобных операционный системах реализована через программную подсистему «Common UNIX Printing System» (CUPS). Если кратенько, тот CUPS состоит из демона **cupsd**, который принимает данные для печати от приложений, складывает их в очередь, откуда последовательно изымает и отправляет на принтер через «Internet Printing Protocol» (IPP), при необходимости применяя программные «фильтры» для преобразования форматов данных. Описание свойств принтеров задаются в CUPS с помощью специальных PPD-файлов («PostScript Printer Description») которые обычно, вместе с фильтрами, поставляются производителем печатающего оборудования. Общий вид пайплайна который проходит документ при печати через CUPS изображен рис. 8.

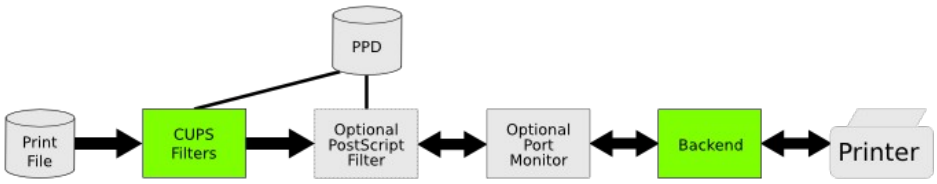


Рис. 8. Пайплайн в системе печати CUPS.

Настройка печати через CUPS состоит из следующих шагов: 1) установка и запуск CUPS демона, 2) установка драйвера для конкретного принтера (PPD и фильтра) и 3) добавление в систему принтера и настройка параметров печати. Рассмотрим все эти шаги поподробнее.

9.1. Установка и запуск CUPS

Установить CUPS можно из репозитория следующей командой:

```
root@sunny:~ # pkg install -y cups
```

Добавить демона **cupsd** в автозапуск:

```
root@sunny:~ # sysrc cupsd_enable="YES"
```

Запустить демона **cupsd** без перезагрузки системы:

```
root@sunny:~ # service cupsd start
```

9.2. Установка драйверов для принтеров производства `HP’

Подавляющее большинство современных сетевых устройств печати поддерживают IPP протокол и не требуют наличие специального драйвера, так как протокол IPP предусматривает выгрузку (запрос) PPD файла из принтера в систему CUPS. Тем не менее, в репозитории FreeBSD имеется пакет с названием **hplip**, он содержит большую коллекцию «драйверов» (PPD и фильтров) для печатающих устройство производства HP. Данный пакет регулярно обновляется, в том числе актуализируются и PPD файлы в нем. Если ПО на Вашем принтере не обновлялось много лет (а это мой случай), то имеет смысл установить этот пакет и получить свежий набор драйверов.

```
root@sunny:~ # pkg search hplip
hplip-3.24.4_3                 Drivers and utilities for HP printers and All-in-One devices

root@sunny:~ # pkg install -y hplip


root@sunny:~ # pkg info hplip | tail
       ports_top_git_hash: a3de54ab220
       repo_type      : binary
       repository     : FreeBSD
Flat size      : 31.4MiB
Description    :
HPLIP (HP Linux Imaging & Printing) is an HP-developed solution for printing,
scanning, and faxing with HP inkjet and laser printers in Linux. The HPLIP
project provides printing support for 3,171 printer and scanner models,
including Deskjet, Officejet, Photosmart, PSC (Print Scan Copy), Business
Inkjet, LaserJet, Edgeline Multi-function Printers,LaserJet MFPs and ScanJets.
```

Как видно из описания, пакет содержит поддержку для **3171** устройства производства HP.

9.3. Подключение сетевого принтера к CUPS

Имеется множество способов администрирования (управления принтерами и очередями) CUPS, в том числе:

- из командной строки с помощью утилиты **lpadmin** входящей в состав CUPS;
- через Web интерфейс подключившись браузером к на локальный хост: http://127.0.0.1:631
- через GUI утилиту **system-config-printer** которую можно установить из репозитория.

Кратко рассмотрим первые два.

9.3.1. С помощью утилиты `lpadmin’

Надо сказать, что для подключения в CUPS принтера поддерживающего IPP достаточно только знать его IP адрес (ну и логин/пароль, если на принтере включена авторизация), при этом знать производителя и модель принтера не обязательно, так как можно использовать специальную модель «**everywhere»** и система печати сама определит как работать с таким принтером.

Самый простой и быстрый способ добавить новый сетевой принтер в CUPS это с помощью утилиты **lpadmin**, для этого нужно выполнить всего одну команду:

```
root@sunny:~ # lpadmin -p MyBestPrinter -v "ipp://192.168.168.13/ipp/print" -m everywhere -o
PageSize=A4
```

Где **192.168.168.13** — IP адрес сетевого принтера, а **MyBestPrinter** — это т. н. «destination», задает то, как данный принтер будет идентифицирован в системе и виден для других программ. Фактически, в параметр **-p** можно указать произвольную строку текста. Параметр **-o PageSize A4** задает тип используемой по умолчанию бумаги (A4). Для некоторых принтеров тип бумаги является важной опцией.

Просмотреть список принтеров в системе можно командой:

```
root@sunny:~ # lpstat -p
printer HP-LaserJet-500-color-MFP-M570 is idle.  enabled since Wed Jan 14 19:31:48 2026
printer HP-LaserJet-Pro-P1102w is idle.  enabled since Fri Jan 16 04:22:38 2026
printer MyBestPrinter is idle.  enabled since Fri Jan 16 05:35:56 2026
printer Roland-VersaWorks-Dual is idle.  enabled since Sun Jan 28 20:25:01 2024
printer TSC-TTP-2410MT is idle.  enabled since Fri Sep  6 14:46:50 2024
```

В данном случае команда **lpstat** отображает состояние всех принтеров имеющихся в моём распоряжении, в том числе **MyBestPrinter** который мы только что создали.

9.3.2. С помощью Web-браузера

Подсистема CUPS работает поверх протокола HTTP/1.1, поэтому открываем в Web-браузере страницу с адресом http://127.0.0.1:631/admin и попадаем на страницу с администрированием принтеров (см. рис. 9), при этом будет запрошена авторизация где нужно указать пользователя **root** и его пароль.

Чтобы добавить новый принтер, необходимо в разделе «**Printers**» нажать кнопку «**Add Priner**». После чего указать каким способом CUPS будет добираться до принтера (в моем случает этот IPP). Далее — указать URL в формате: **ipp://192.168.169.13:631/ipp/print**, точно так же как и в утилите **lpadmin**. Система CUPS попросит выбрать производителя и модель принтера из огромного списка, но как я уже упоминал, достаточно указать «IPP Everywhere».

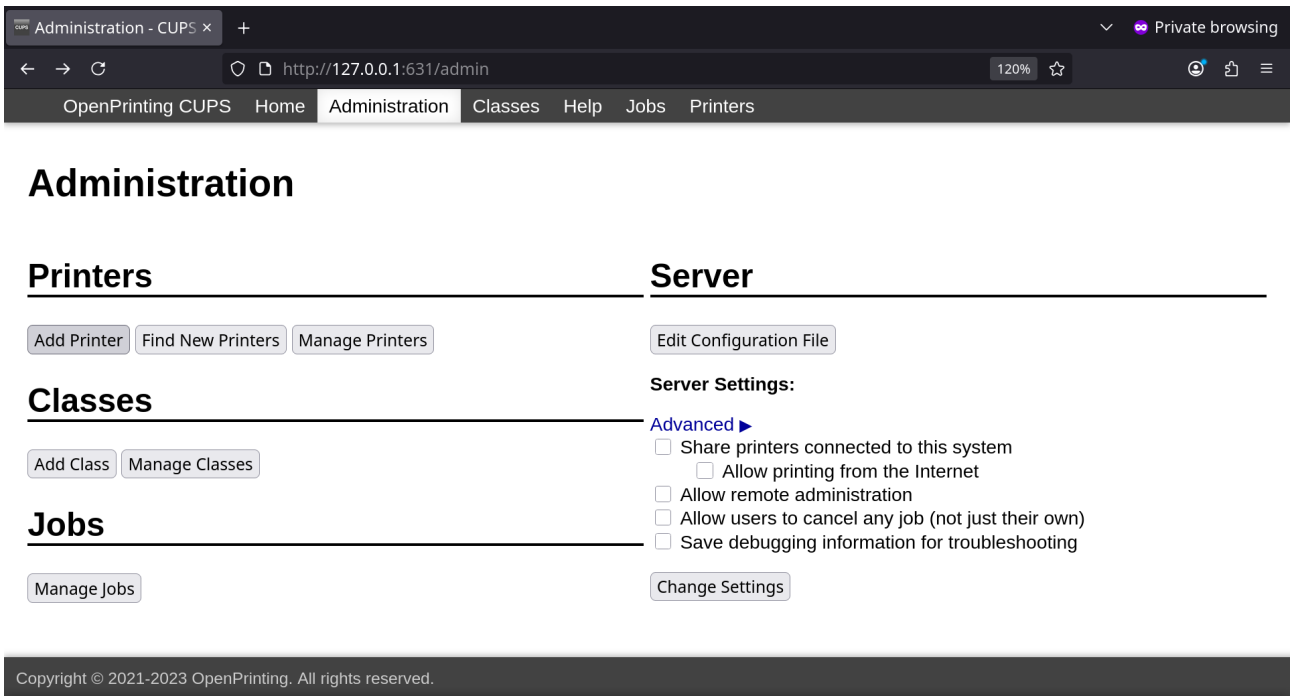


Рис. 9. Страница администрирования принтеров CUPS в Web-браузере.

Чтобы проверить работоспособность принтера, необходимо в верхнем навигационном меню выбрать раздел «**Printers**». После чего в появившемся списке выбрать требуемый принтер, затем в ниспадающем меню «**Maintenance**» выбрать опцию «**Print test Page**».

Аналогичным образом мы можем подключиться с помощью Web-браузера к любому принтеру поддерживающему IPP на порт **631** и получить страницу с информацией о его состоянии. На рис. 10 показана страница статуса на моём офисном МФУ полученная из Web-браузера. Очень удобно!

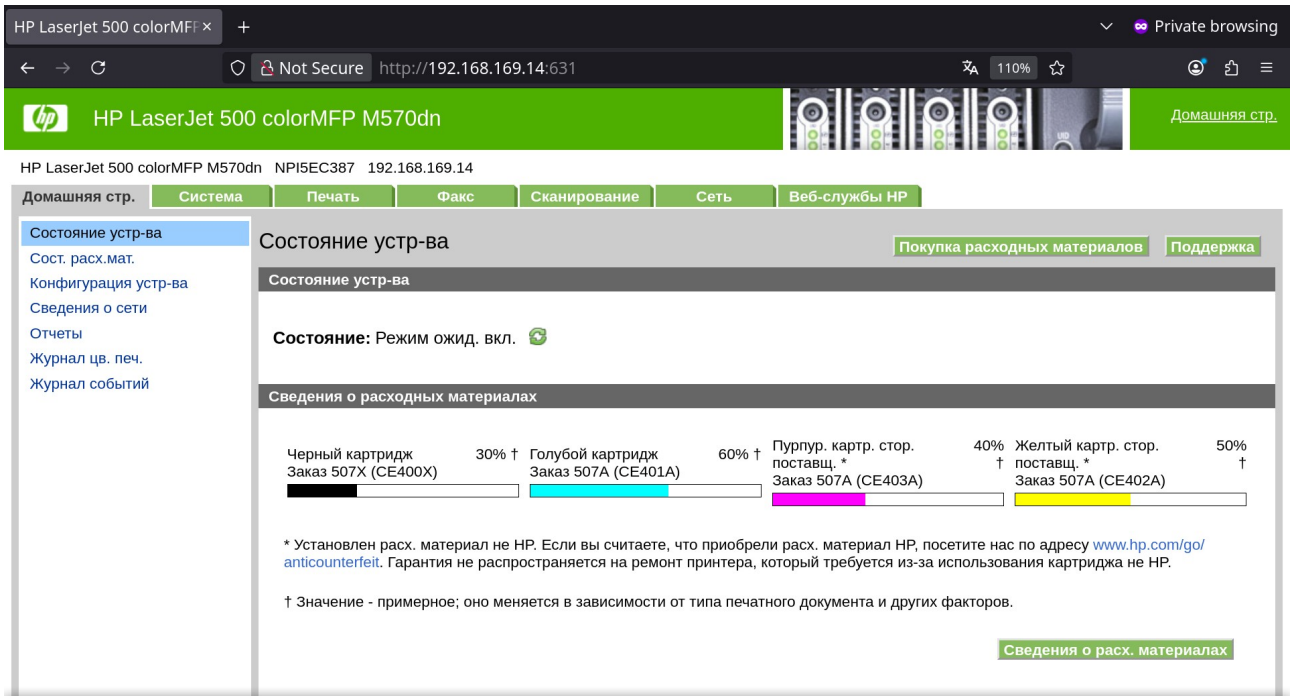


Рис 10. Страница статуса МФУ в Web-браузере.

CUPS позволяет работать и с USB принтерами, и даже с принтерами имеющими параллельный (LPT) или последовательный (COM) интерфейс. Для добавления USB принтера с помощью утилиты **lpadmin** необходимо указать URI в специальном формате, например:

```
root@sunny:~ # lpadmin -p USB_Printer -v "usb://HP/LaserJet%20MFP?serial=42"
```

Узнать URI можно с помощью команды **lpinfo -v.**

В составе CUPS имеется утилита **/usr/local/libexec/cups/backend/usb** которая является backend-ом к USB принтерам (через неё **cupsd** отправляет данные на USB принтер). Она подгружает текстовый файл **/usr/local/share/cups/usb/org.cups.usb-quirks**, в котором перечислены VID/PID печатающих устройств, и импортирует обнаруженные на шине USB устройства с совпадающими VIP/PID в CUPS. Если Ваш принтер отсутствует в этом списке (не отображается в **lpinfo**), то можно попытаться добавить пару VIP/PID близкой модели в этот файл и, перезапустив **cupsd**, попробовать еще раз.

Более подробно о CUPS можно ознакомиться в системном руководстве по команде **man cups**. А также на Web странице **http://127.0.0.1:631/**

Ну, а теперь можно открывать **LibreOffice** и набивать тексты рефератов.

10. Исправление бага в UEFI загрузчике

Как я уже отметил в главе 5.4, при установке 64-х битной версии ОС FreeBSD 15.0-RELEASE на макбук с 32-х битный UEFI Firmware, возникло непредвиденное обстоятельство: сразу после того, как загрузчик передает управление ядру ОС, прекращается любой вывод на видео-консоль, хотя сама система продолжает функционировать вполне нормально. Если бы я устанавливал ОС на «безголовый» сервер, то меня бы это не смутило — можно было бы воспользоваться UART консолью, но тут обстоятельства совершенно иные — без функционирующей видео-консоли с ноутбуком работать достаточно тяжело. Причина проблемы, как мне показалось, находится в ядре ОС, поэтому я собрал немного информации и оформил на на сайте баг-трекера проекта FreeBSD новый тикет с номером 291935.

Отчет я оформил 25-го декабря 2025г. Не смотря на рождественские и новогодние праздники, уже 29-го декабря в отчет отписался один из разработчиков UEFI загрузчика - Ахмад Халифа. Он предположил, что проблема находится не в ядре ОС, а в загрузчике **loader_ia32.efi**, как раз в том самом, который Ахмад совсем недавно портировал на 32-х битный UEFI. Его первое предположение состояло в том, что в UEFI Firmware на данной машине отсутствует поддержка GOP (Graphics Output Protocol) — набора сервисов для отображения текста и графики на экран. Он также отметил, что если GOP не поддерживается, то загрузчик должен переключиться в режим UGA (Universal Graphics Adapter — обязательная поддержка в UEFI 1.0) и работать с консолью уже самостоятельно, занимаясь рендерингом текста «вручную», а также передать указатель на видео фреймбуфер и его параметры в ядро ОС чтобы там драйвер консоли **vt** тоже мог рендерить текстовый вывод.

После анализа кода загрузчика, Ахмад заметил, что в нём логика работы не предусматривает отход на UGA в том случае если в UEFI отсутствует поддержка GOP, хотя сам код работы с UGA присутствует. Видимо когда-то это работало, но было сломано и не проверено должным образом. Для решения этой проблемы Ахмад предложил первый патч и попросил меня протестировать данное решение.

Чтобы приложить патч к какой-либо части ОС FreeBSD и собрать новое ядро или загрузчик, необходимо иметь на отдельной машине (физической или виртуальной) установленную FreeBSD этой версии с развернутым исходным кодом ОС и средствами разработки. Так удачно сложилось, что совсем недавно, для экспериментов, я приобрел еще один мини-ПК Horizon N5 на базе Intel Celeron N5105 (об установке FreeBSD на такую же машину см. мою статью «Чёрт в табакерке» на Хабре) и уже успел развернуть на ней ОС FreeBSD 14.3/amd64. Я развернул на ней исходный код системы и проанализировал исходники загрузчика, и даже попытался автоматически применить предложенный Ахмадом патч с помощью утилиты **patch**. Однако патч, выпущенный для 15.0-RELEASE, отказался прикладываться к загрузчику версии 14.3-RELEASE, поэтому, не долго думая, я занялся патчингом вручную, благо патч оказался достаточно простой.

Пропатченный загрузчик собрался без проблем. Я развернул образ «memstick» с версией FreeBSD 14.3-RELEASE/amd64 на еще один USB носитель и перенес пропатченный загрузчик **loader_ia32.efi** в файл **/EFI/BOOT/BOOTIA32.EFI** в разделе «ESP». После чего вставил USB носитель в макбук, перевел его в режим «Startup Manager» и попытался загрузиться с USB.

Эффект был весьма интересный. Посредине черного экрана появилось белое сообщение, которое в ту же секунду исчезло (я даже не успел его прочитать), и компьютер остался с пустым экраном. Это состояние продлилось секунд 10, после чего начали одна за другой отображаться строки от ядра ОС, как при обычно загрузке, система нормально загрузилась и запустился Инсталлер. Я догадался что, патч таки сработал! Но сломалось что-

то другое — теперь на экран не выводятся сообщения от самого загрузчика и я не вижу меню и красивой заставки с логотипом. То, что такое состояние продлилось около 10 секунд объясняется тем, что загрузчик автоматически начинает загружать систему через 10 секунд неактивности пользователя. Иными словами, загрузчик не обнаружив поддержки GOP, откатился в режим UGA, собрал данные для ядра и запустил его. При этом ядро замечательно рендерит текст, а вот сам загрузчик уже не может ничего отобразить. Меня заинтересовало, что за текстовое сообщение выдается загрузчиком в самом начале. Я еще раз выполнил загрузку с USB носителя и записал процесс на видео. В тексте сообщения содержалось следующее:

```
No change detected in framebuffer -- error 0???Unable to reliably detect frame
buffer.???Unable to reliably detect the stride.???
```

Я нашел эти строки в исходном коде загрузчика, они относились к работе с видео-фрейм буфером. Точнее, к коду детектирования смещения физического адреса фреймбуфера и вычисления размера строки пикселей на экране (stride). Проанализировав этот код я слегка прифигел от примененного там решения. Попытаюсь объяснить его в четырех строках.

1. Загрузчик запрашивает GOP сервисы у UEFI, но их нет (не поддерживаются).

2. Тогда загрузчик откатывается в режим UGA и пытается запросить у UGA размеры экрана, адрес (смещение) начала видимой области видео фреймбуфера и длину строки. Тут его тоже поджидает частичный облом — часть возвращенных данных содержат нули.

3. Но загрузчик не теряет надежду. Он перебирает BAR регистры PCI контроллеров и находит BAR с самым длинным блоком данных предполагая, что это и есть видео фреймбуфер!

4. Найдя таким образом базовый адрес, загрузчик применяет хитрый алгоритм для вычисления смещения и длины строки: используя UGA сервис для отображения одного пикселя на экране, он последовательно рисует на на экран точки, при этом каждый раз ищет изменения в физической памяти. Таким образом он пытается понять где начинается видимая область экрана и какова длина строки!

Вот ссылка на код этого воистину феноменального алгоритма:

1. Отгадывание базового адреса фреймбуфера перебором регистров PCI BAR, функция efifb_uga_locate_framebuffer().

2. Поиск отрисованного пикселя в видео-памяти, функция efifb_uga_find_pixel().

В коде обнаружились жестко закодированные параметры для двух макбуков: "MacBook3,1" и "MacBookPro3,1" — это чтобы не перебирать и не искать точки. Я добавил в это же место данные для своего макбука "MacBook2,1". К тому времени я уже выяснил, что базовый адрес видео фреймбуфера на моей машине начинается с **0xA0000000**, с нулевым смещением и длиной строки равной **2048** точки при основном разрешении экрана 1280x800. Но это не сработало! Собрав таким образом пропатченный загрузчик и загрузившись с него я увидел все то же странное сообщение. Складывалось такое ощущение, что UGA режим на этом макбуке тоже не работает!

Но я выяснил, что у загрузчика есть еще и чисто текстовый режим в котором он использует сервис UEFI Firmware для вывода текста. Я нашел нужное место и добавил строку: **gfx_state.tg_fb_type = FB_TEXT**. Это сработало. Загрузчик отобразил меню с помощью текстовых сервисов UEFI, отрисовав ASCII лого FreeBSD и запустил ядро ОС, которое тоже без проблем смогло отображать свой текстовый вывод, но уже через свой драйвер.

Я изложил результаты своих изысканий в тикет баг-трекера и на следующий день получил интересный ответ от Ахмада с еще одним патчем. Он выяснил, что в загрузчике присутствует еще несколько багов, причем один из них весьма серьезный — неверное использование указателя (указатель на структуру одного типа данных, а возвращается и применяется к совсем другой структуре). Он, также как и я предположил, что UGA на этой старенькой машине не работает, поэтому использовать его нельзя, вместо UGA надо использовать свой встроенный метод (**gfxfb**). Второй его патч решает обе проблемы. И этот патч полностью сработал — после его применения выводятся сообщения и от загрузчика и от ядра ОС, и графический логотип FreeBSD тоже отображается. :-)

Я отписался в тикет о результатах успешного применения патча, оформил все изменения в один объединенный патч для 14.3-RELEASE и приложил его к тикету, вдруг кому-то еще потребуется срочно установить FreeBSD на MacBook. Ведь финальный патч еще должен пройти ревью от других разработчиков ядра FreeBSD, а значит появится в релизе не скоро. Так, что если кому-то требуется, забирайте прямо сейчас: https://bz-attachments.freebsd.org/attachment.cgi?id=266664

Проблема с графикой и выводом текста в загрузчике **loader_ia32.efi** была устранена к 30-му декабря, всего за пять дней с момента обнаружения. Ахмад пообещал побыстрее протащить этот патч через систему ревью и уже 7-го января я получил уведомление том, что патч принят и смерджен в 13.x, 14.x и 15.x ветки ОС FreeBSD. Так, что в следующем релизе 14.4, намеченном на 6 февраля 2026 года, будут присутствовать эти изменения. Ну, а пока можно воспользоваться моим объединенным патчем или скачать уже пропатченный и собранный мной бинарник загрузчика: **/EFI/BOOT/BOOTIA32.EFI**.

11. Установка 64-х битной ОС FreeBSD 14.3-RELEASE

Имея исправленный загрузчик **loader_ia32.efi**, предназначенный для загрузки 64-х битной ОС FreeBSD на старых ПК (макбуках) с 32-х битной UEFI Firmware, установить 64-х битную версию ОС FreeBSD не представляет большого труда — требуется только скопировать бинарный файл загрузчика на «ESP» раздел в каталог **/EFI/BOOT** на инсталляционном USB носителе. Единственная незначительная проблема — низкая скорость считывания данных с USB устройства при использовании UEFI сервисов. Но обнаружилось, что как только ядро ОС запустится и система перейдет на использования своего «родного» драйвера USB, эта проблема самоустраняется. Ниже приведена краткая пошаговая инструкция по установке ОС FreeBSD 14.3-RELEASE/amd64 на MacBook2,1.

1. На рабочем ПК скачиваем образ «memstick» версии FreeBSD 14.3-RELEASE-amd64:

```
$ wget https://download.freebsd.org/releases/ISO-IMAGES/14.3/FreeBSD-14.3-RELEASE-amd64-memstick.img.xz
```

2. Вставляем USB Flash накопитель и записываем на него образ «**memstick**» с помощью утилиты **dd**, минуя предварительную распаковку, следующей командой:

```
# xz -d -c FreeBSD-14.3-RELEASE-amd64-memstick.img.xz | dd of=/dev/da0 bs=8192 conv=sync
```

Здесь **/dev/da0** это блочное устройство USB Flash, так оно обычно детектируется в ОС FreeBSD. В ОС Linux это устройство обычно детектируется как **/dev/sdb**. Выяснить точное имя устройства можно заглянув в вывод команды **dmesg** после подключения USB Flash к рабочему ПК.

Пользователям ОС Windows для подготовки носителя придется воспользоваться сторонней утилитой, например Rufus, предварительно распаковав скачанный образ!

3. Монтируем раздел «ESP» с USB носителя:

для FreeBSD:

```
# mount_msdosfs /dev/da0s1 /mnt
```

на Linux:

```
# mount -t vfat /dev/sdb1 /mnt
```

Пользователям Windows ничего монтировать не надо, раздел «ESP» появится в системе как отдельный диск.

4. Скачиваем и кладем в него патченный загрузчик:

```
# wget https://www.fabmicro.ru/pub/FreeBSD_on_MacBook/EFI/BOOT/bootia32.efi
# cp bootia32.efi /mnt/EFI/BOOT/
```

5. Вежливо размонтируем «ESP» раздел:

```
# sync && sync && umount /mnt
```

6. Вынимаем USB носитель из рабочего ПК и вставляем в MacBook, зажимаем клавишу **Option** (**⌥ или Alt**) и подаем питание однократным нажатием кнопки **Power**.

7. Через 10 секунд запустится «Startup Manager» и на экране отобразятся источники загрузки, см. рис. 3. Выбираем пиктограмму с USB носителем обозначенную как «**EFI Boot**».

8. В появившемся меню загрузчика выбираем первый пункт однократным нажатием клавиши **Enter**.

9. В меню запустившегося Инсталлера (FreeBSD Installer) выбираем пунк «**[Install]**» и следуем общим инструкциям по установке FreeBSD приведенным в главе «***6. Установка 32-х битной FreeBSD 14.3-RELEASE на MacBook***», а также в последующих главах. Донастройка для 64-х битной системы идентична, за исключением ряда моментов приведенных ниже.

11.1. Особенности установки 64-х битной ОС FreeBSD 14.3-RELEASE

1. В Инталлере при разбивке диска следует выбрать тип раздела **GPT**. Это избавит от 30-ти секундной задержки возникающей каждый раз при загрузке MacBook-а, вызванной переходом UEFI Firmware в режим совместимости с старым BIOS. При использовании таблицы разделов GPT на диске будет создан «ESP» раздел, а значит будет использован современный метод загрузки ОС — через UEFI loader.

2. Сразу после завершения процесса установки, перед первой перезагрузкой, необходимо войти в меню Инсталлера «**[Live System]»** и скопировать пропатченный загрузчик с USB носителя на системный диск в раздел «ESP». Если этого не сделать, то UEFI Firmware на MacBook-е откажется загружать ОС, так как в разделе «ESP» отсутствует 32-х битная версия загрузчика. Напомню, что это происходит из-за описанного в главе «***4. 64-х битная ОС FreeBSD на ПК с 32-х битным UEFI***» бага — в версии 14.3-RELEASE Инсталлер «забывает» скопировать 32-х битный загрузчик. Сделаем это за него вручную:

Монтируем раздел «ESP» с USB носителя:

```
# mount_msdosfs /dev/da0s1 /boot/efi
```

Монтируем раздел «ESP» с системного диска:

```
# mount_msdosfs /dev/ada0p1 /mnt
```

Копируем патченный загрузчик с USB на системный диск:

```
# cp /boot/efi/efi/boot/bootia32.efi /mnt/efi/boot/
```

Вежливо размонтируем оба «ESP» раздела:

```
# sync && sync && umount /mnt
```

```
# sync && sync && umount /boot/efi
```

И отправляемся на перезагрузку:

```
# reboot
```

И на этом с установкой, пожалуй, закончим. Далее мы рассмотрим как провести несколько тестов производительности двух вариантов ОС FreeBSD — 32-х и 64-х битной, и сравним результаты.

12. Тесты производительности под управлением ОС FreeBSD

Когда я сталкиваюсь с какой-либо ранее не опробованной мной вычислительной системой, меня всегда интересует три простых теста: тест **CoreMark** — для измерения вычислительной мощности ЦПУ широкого круга, от архаичных 8-ми битных до современный 64-х битных суперскаляров; тест **STREAM** — замер пропускной способности ОЗУ; и тест чтения/записи на системный диск выполняемый утилитой **dd**. Проведем все три теста для каждого из вариантов системы — 32-х битной (i386) и 64-х битной (amd64).

12.1. Тест производительности ЦПУ `CoreMark’

Для выполнения теста CoreMark необходимо сначала клонировать его исходный код из репозитория с Github и выполнить сборку исполняемого файла. Тест CoreMark принято выполнять в двух вариантах: в однопоточном — замер одного вычислительного ядра, и в многопоточном — замер пиковой производительности всех ядер сразу. Я приведу замеры только для многопоточного теста.

Выкачиваем исходник CoreMark и собираем его:

```
root@sunny:~ # git clone https://github.com/eembc/coremark.git
Cloning into 'coremark'...

root@sunny:~ # cd coremark
```

У процессора Intel Core2 Duo имеется два вычислительных ядра, что позволяет выполнять два одновременных потока инструкций. Укажем на этот факт тесту CoreMark при его сборке задав макро **MULTITHREAD=2**. Для запуска двух потоков будем использовать системный вызов **fork()**, для этого объявим макро **USE_FORK**. Все эти параметры передадим в переменной окружения **XCFLAGS** при запуске команды **make**:

```
root@sunny:~/coremark # gmake XCFLAGS="-DMULTITHREAD=2 -DUSE_FORK"

gmake XCFLAGS="-DMULTITHREAD=2 -DUSE_FORK -DPERFORMANCE_RUN=1" load run1.log
./coremark.exe  0x0 0x0 0x66 0 7 1 2000 > ./run1.log

gmake XCFLAGS="-DMULTITHREAD=2 -DUSE_FORK -DVALIDATION_RUN=1" load run2.log
./coremark.exe  0x3415 0x3415 0x66 0 7 1 2000  > ./run2.log
```

В результате сборки получим исполняемый файл **coremark.exe**.

Сборочный сприпт (Makefile) для теста CoreMark устроен так, что он сразу после сборки запускает два измерения, а их результаты записывает в файлы **run1.log** и **run2.log**. Первый будет содержать результаты теста максимальной производительности, второй — результаты проверочного теста (валидация) для отправки данных на сайт EEBMC. Как видно,

разница состоит в некоторых входных параметрах задающих «затравочные значения» (seed) для используемых алгоритмов.

12.2. Тест производительности ОЗУ `STREAM'

Исходный код теста STREAM также доступен на Github-е. В репозитории содержится две реализации данного теста: на языке «Си» и на языке «FORTRAN». Для измерения можно использовать любой из них, оба варианта собираются одним сборочным скриптом.

Клонируем репозиторий и выполним сборку теста STREAM:

```
root@sunny:~ # git clone https://github.com/jeffhammond/STREAM.git
Cloning into 'STREAM'...

root@sunny:~ # cd STREAM

root@sunny:~/STREAM # gmake
gcc -O2 -fopenmp   -c -o mysecond.o mysecond.c
gcc -O2 -fopenmp -c mysecond.c
gfortran -O2 -fopenmp -c stream.f
gfortran -O2 -fopenmp stream.o mysecond.o -o stream_f.exe
gcc -O2 -fopenmp stream.c -o stream_c.exe
```

В результате получим два исполняемых файла: **stream_f.exe** и **stream_c.exe** . Запуск теста STREAM осуществляется командой:

```
root@sunny:~/STREAM # ./stream_c.exe
```

или

```
root@sunny:~/STREAM # ./stream_f.exe
```

В результате исполнения на консоле будут отображаться замеры по четырем видам операций с данными в ОЗУ (подробности см. в описание тесте STREAM на сайте Wikipedia). Пример замера с моего рабочего ноутбука Lenovo Ideapad выглядит так:

```
-------------------------------------------------------------
Function    Best Rate MB/s  Avg time     Min time     Max time
Copy:           20629.2     0.008022     0.007756     0.009219
Scale:          19231.1     0.008791     0.008320     0.010929
Add:            21895.7     0.011473     0.010961     0.012910
Triad:          22271.6     0.011063     0.010776     0.011662
-------------------------------------------------------------
Solution Validates: avg error less than 1.000000e-13 on all three arrays
-------------------------------------------------------------
```

Последняя строчка также показывает вероятность возникновения ошибки при работе с памятью. Значение **1.0e-13** говорит в том, что ошибок не выявлено.

12.3. Тест производительности дисковой подсистемы (dd)

Самый простой и достаточно репрезентативный тест дисковой подсистемы состоит в поблочном копирования большого объема данных с помощью системной утилиты **dd**. У этой утилиты имеется опции для замера и отображения пропускной способности.

Запустить тест на чтение с диска можно следующей командой:

```
root@butterfly:~ # dd if=/dev/nvme0ns1 of=/dev/null bs=1M iflag=direct status=progress
13116637184 bytes (13 GB, 12 GiB) transferred 33.002s, 397 MB/s
^C
```

На консоли будет отображаться статистика чтения с диска и усредненное значение пропускной способности. Данный замер выполнен на моём рабочем ноутбуке с SSD диском имеющим интерфейс NVME (2xPCIe) скорость чтения с которого составляет **397 мегабайт/сек**.

Тест записи на SSD с установленной ОС выполнить сложнее, так как потребует выделить на диске неиспользуемые файловой системой блоки, что потребуется переразбивки диска. Поэтому данный тест выполнять не будем.

12.4. Загрузка ОС FreeBSD в «однопользовательском» режиме и прогон тестов

Для того, чтобы тесты более-менее адекватно отражали реальное положение дел, их необходимо проводить на «голой» системе, то есть с отключенным менеджером питания и без лишних тяжеловесных процессов, как, например, **xorg**, **sddm** (и всем что, связано с графической подсистемой X11). Для этого мы будем загружать ОС FreeBSD в т. н. «однопользовательский» режим (**Single User Mode**), где кроме процесса **/sbin/init** и оболочки **/bin/sh** не будет никаких других процессов и нагрузки. Также нам потребуется вручную вывести ЦПУ на максимальную тактовую частоту с помощью установки специальной переменной в **sysctl**.

Чтобы загрузить ОС FreeBSD в «однопользовательский режим» необходимо в самом начале загрузки операционной системы, в меню загрузчика, выбрать второй пункт «**2. Boot Single user**» (см. рис. 4), делается это путем нажатия клавиши **S**. После того, как загрузится и запустится ядро ОС, на консоли вместо привычного приглашения «login:» появится приглашение ввести путь к оболочке, здесь необходимо просто нажать клавишу **Enter**, тогда будет использована оболочка **/bin/sh**. Далее необходимо выполнить следующие операции:

Перемонтировать корневую файловую систему в режим «read/write», так как сразу после загрузки ядра она находится в режиме «read-only»:

```
root@sunny:~ # mount -a
```

Установить максимальную тактовую частоту процессора, предварительно запросив список доступных частот:

```
root@sunny:~ # sysctl dev.cpu.0.freq_levels
dev.cpu.0.freq_levels: 2167/31000 2000/28442 1833/25885 1667/23328 1500/20771 1333/18214
1000/13100

root@sunny:~ # sysctl dev.cpu.0.freq=2167
dev.cpu.0.freq: 1000 -> 2167
```

Частота процессора установлена в **2167** МГц. Все готово к запуску тестов.

12.5. Прогон тестов производительности для MacBook2,1

12.5.1. Тест CoreMark на FreeBSD 14.3-RELEASE/i386

Тест CoreMark на 32-х битной ОС FreeBSD выдает следующий результат:

```
root@sunny:~ # uname -a
FreeBSD  14.3-RELEASE FreeBSD 14.3-RELEASE releng/14.3-n271432-8c9ce319fef7 GENERIC i386

root@sunny:~ # cd coremark

root@sunny:~/coremark # ./coremark.exe

2K performance run parameters for coremark.
CoreMark Size    : 666
Total ticks      : 13556
Total time (secs): 13.556000
Iterations/Sec   : 16228.976099
Iterations       : 220000
Compiler version : GCCFreeBSD Clang 19.1.7 (https://github.com/llvm/llvm-project.git
llvmorg-19.1.7-0-gcd708029e0b2)
Compiler flags   : -O2 -DMULTITHREAD=2 -DUSE_FORK -DPERFORMANCE_RUN=1  -lrt
Parallel Fork : 2
Memory location  : Please put data memory location here
                        (e.g. code in flash, data on heap etc)
seedcrc          : 0xe9f5
[0]crclist       : 0xe714
[1]crclist       : 0xe714
[0]crcmatrix     : 0x1fd7
[1]crcmatrix     : 0x1fd7
[0]crcstate      : 0x8e3a
[1]crcstate      : 0x8e3a
[0]crcfinal      : 0x33ff
[1]crcfinal      : 0x33ff
Correct operation validated. See README.md for run and reporting rules.
CoreMark 1.0 : 16228.976099 / GCCFreeBSD Clang 19.1.7 (https://github.com/llvm/llvm-
project.git llvmorg-19.1.7-0-gcd708029e0b2) -O2 -DMULTITHREAD=2 -DUSE_FORK
-DPERFORMANCE_RUN=1  -lrt / Heap / 2:Fork
```

12.5.2. Тест CoreMark на FreeBSD 14.3-RELEASE/amd64

Тест CoreMark на 64-х битной ОС FreeBSD выдает следующий результат:

```
root@sunny:~ # uname -a
FreeBSD sunny 14.3-RELEASE FreeBSD 14.3-RELEASE releng/14.3-n271432-8c9ce319fef7 GENERIC
amd64

root@sunny:~ # cd coremark

root@sunny:~/coremark # ./coremark.exe
2K performance run parameters for coremark.
CoreMark Size    : 666
Total ticks      : 11433
Total time (secs): 11.433000
Iterations/Sec   : 19242.543514
Iterations       : 220000
Compiler version : GCC13.3.0
Compiler flags   : -O2 -DMULTITHREAD=2 -DUSE_FORK -DPERFORMANCE_RUN=1  -lrt
Parallel Fork : 2
Memory location  : Please put data memory location here
                        (e.g. code in flash, data on heap etc)
seedcrc          : 0xe9f5
[0]crclist       : 0xe714
[1]crclist       : 0xe714
[0]crcmatrix     : 0x1fd7
```

```
[1]crcmatrix     : 0x1fd7
[0]crcstate      : 0x8e3a
[1]crcstate      : 0x8e3a
[0]crcfinal      : 0x33ff
[1]crcfinal      : 0x33ff
Correct operation validated. See README.md for run and reporting rules.
CoreMark 1.0 : 19242.543514 / GCC13.3.0 -O2 -DMULTITHREAD=2 -DUSE_FORK -DPERFORMANCE_RUN=1
-lrt / Heap / 2:Fork
```

12.5.3. Тест STREAM на FreeBSD 14.3-RELEASE/i386

Тест STREAM «Си» на 32-х битной ОС FreeBSD выдает следующий результат:

```
root@sunny:~ # uname -a
FreeBSD  14.3-RELEASE FreeBSD 14.3-RELEASE releng/14.3-n271432-8c9ce319fef7 GENERIC i386

root@sunny:~ # cd STREAM

root@sunny:~/STREAM # ./stream_c.exe
-------------------------------------------------------------
STREAM version $Revision: 5.10 $
-------------------------------------------------------------
This system uses 8 bytes per array element.
-------------------------------------------------------------
Array size = 10000000 (elements), Offset = 0 (elements)
Memory per array = 76.3 MiB (= 0.1 GiB).
Total memory required = 228.9 MiB (= 0.2 GiB).
Each kernel will be executed 10 times.
 The *best* time for each kernel (excluding the first iteration)
 will be used to compute the reported bandwidth.
-------------------------------------------------------------
Number of Threads requested = 2
Number of Threads counted = 2
-------------------------------------------------------------
Your clock granularity/precision appears to be 11 microseconds.
Each test below will take on the order of 65090 microseconds.
   (= 5917 clock ticks)
Increase the size of the arrays if this shows that
you are not getting at least 20 clock ticks per test.
-------------------------------------------------------------
WARNING -- The above is only a rough guideline.
For best results, please be sure you know the
precision of your system timer.
-------------------------------------------------------------
Function    Best Rate MB/s  Avg time     Min time     Max time
Copy:            2210.6     0.073282     0.072380     0.079542
Scale:           2128.5     0.076196     0.075169     0.083009
Add:             2450.3     0.102436     0.097948     0.123862
Triad:           2290.1     0.110585     0.104797     0.130175
-------------------------------------------------------------
Solution Validates: avg error less than 1.000000e-13 on all three arrays
-------------------------------------------------------------
```

12.5.4. Тест STREAM на FreeBSD 14.3-RELEASE/amd64

Тест STREAM «Си» на 64-х битной ОС FreeBSD выдает следующий результат:

```
root@sunny:~ # uname -a
FreeBSD sunny 14.3-RELEASE FreeBSD 14.3-RELEASE releng/14.3-n271432-8c9ce319fef7 GENERIC amd6

root@sunny:~ # cd STREAM

root@sunny:~/STREAM # ./stream_c.exe
-------------------------------------------------------------
```

```
STREAM version $Revision: 5.10 $
-------------------------------------------------------------
This system uses 8 bytes per array element.
-------------------------------------------------------------
Array size = 10000000 (elements), Offset = 0 (elements)
Memory per array = 76.3 MiB (= 0.1 GiB).
Total memory required = 228.9 MiB (= 0.2 GiB).
Each kernel will be executed 10 times.
 The *best* time for each kernel (excluding the first iteration)
 will be used to compute the reported bandwidth.
-------------------------------------------------------------
Number of Threads requested = 2
Number of Threads counted = 2
-------------------------------------------------------------
Your clock granularity/precision appears to be 2 microseconds.
Each test below will take on the order of 65402 microseconds.
   (= 32701 clock ticks)
Increase the size of the arrays if this shows that
you are not getting at least 20 clock ticks per test.
-------------------------------------------------------------
WARNING -- The above is only a rough guideline.
For best results, please be sure you know the
precision of your system timer.
-------------------------------------------------------------
Function    Best Rate MB/s  Avg time     Min time     Max time
Copy:            2214.4     0.072455     0.072254     0.072691
Scale:           2109.6     0.076156     0.075844     0.076645
Add:             2418.9     0.101655     0.099217     0.120725
Triad:           2420.7     0.101867     0.099144     0.122635
-------------------------------------------------------------
Solution Validates: avg error less than 1.000000e-13 on all three arrays
-------------------------------------------------------------
```

12.5.5. Тест скорости чтения с SSD на FreeBSD 14.3-RELEASE/i386

Результат выполнения блочного чтения командой **dd** с SSD диска на 32-х битной ОС FreeBSD приведен ниже:

```
root@sunny:~ # uname -a
FreeBSD  14.3-RELEASE FreeBSD 14.3-RELEASE releng/14.3-n271432-8c9ce319fef7 GENERIC i386

rz@sunny:~ # dd if=/dev/ada0 of=/dev/null bs=1M iflag=direct status=progress
  4065329152 bytes (4065 MB, 3877 MiB) transferred 32.005s, 127 MB/s
```

12.5.6. Тест скорости чтения с SSD на FreeBSD 14.3-RELEASE/amd64

Результат выполнения блочного чтения командой **dd** с SSD диска на 64-х битной ОС FreeBSD приведен ниже:

```
root@sunny:~ # uname -a
FreeBSD sunny 14.3-RELEASE FreeBSD 14.3-RELEASE releng/14.3-n271432-8c9ce319fef7 GENERIC
amd64

rz@sunny:~ # dd if=/dev/ada0 of=/dev/null bs=1M iflag=direct status=progress
  3483369472 bytes (3483 MB, 3322 MiB) transferred 25.006s, 139 MB/s
```

Видно, что для установленного в этом макбуке SSD накопителя «KINGSTON SV300S37A120G» с шиной SATA1 пропускная способность составляет **139 Мегабайт/сек**, что близко к теоретическому значению:

```
root@sunny:~ # dmesg | grep ada0:
ada0: <KINGSTON SV300S37A120G 60AABBF0> ATA8-ACS SATA 3.x device
ada0: Serial Number 50026B76670208CF
ada0: 150.000MB/s transfers (SATA 1.x, UDMA6, PIO 512bytes)
```

```
ada0: Command Queueing enabled
ada0: 114473MB (234441648 512 byte sectors)
```

Также видно, что скорость блочного чтения с SSD на 32-х битной версии ОС FreeBSD почти на 10% ниже чем на 64-х битной.

12.5.7. Тест `glmark2' для видео подсистемы на FreeBSD 14.3-RELEASE/i386

Оценку скорости рендеринга 3D изображения с использованием встроенного в процессор видеоадаптера, можно выполнить с помощью тестовой утилиты **glmark2**. Данная утилита является X11 клиентом, поэтому запускать её нужно из окна терминала в графической подсистеме:

```
rz@sunny:~ % uname -a
FreeBSD  14.3-RELEASE FreeBSD 14.3-RELEASE releng/14.3-n271432-8c9ce319fef7 GENERIC i386

rz@sunny:~ % glmark2 --fullscreen
=======================================================
    glmark2 2023.01
=======================================================
    OpenGL Information
    GL_VENDOR:      Mesa Project
    GL_RENDERER:    i915 (chipset: 945GM)
    GL_VERSION:     2.1 Mesa 24.1.7
    Surface Config: buf=32 r=8 g=8 b=8 a=8 depth=24 stencil=0 samples=0
    Surface Size:   1280x800 fullscreen
=======================================================
[build] use-vbo=false:'-avx512er' is not a recognized feature for this target (ignoring
feature)
'-avx512pf' is not a recognized feature for this target (ignoring feature)
'-avx512er' is not a recognized feature for this target (ignoring feature)
'-avx512pf' is not a recognized feature for this target (ignoring feature)
'-avx512er' is not a recognized feature for this target (ignoring feature)
'-avx512pf' is not a recognized feature for this target (ignoring feature)
 FPS: 57 FrameTime: 17.733 ms
=======================================================
                                  glmark2 Score: 56
=======================================================
```

12.5.8. Тест `glmark2' для видео подсистемы на FreeBSD 14.3-RELEASE/amd64

Результаты аналогичного теста для 64-х битной версии ОС FreeBSD приведены ниже:

```
rz@sunny:~ % uname -a
FreeBSD sunny 14.3-RELEASE FreeBSD 14.3-RELEASE releng/14.3-n271432-8c9ce319fef7 GENERIC
amd64

rz@sunny:~ % glmark2 --fullscreen
=======================================================
    glmark2 2023.01
=======================================================
    OpenGL Information
    GL_VENDOR:      Mesa Project
    GL_RENDERER:    i915 (chipset: 945GM)
    GL_VERSION:     2.1 Mesa 24.1.7
    Surface Config: buf=32 r=8 g=8 b=8 a=8 depth=24 stencil=0 samples=0
    Surface Size:   1280x800 fullscreen
=======================================================
[build] use-vbo=false:'-avx512er' is not a recognized feature for this target (ignoring
feature)
'-avx512pf' is not a recognized feature for this target (ignoring feature)
'-avx512er' is not a recognized feature for this target (ignoring feature)
```

```
'-avx512pf' is not a recognized feature for this target (ignoring feature)
'-avx512er' is not a recognized feature for this target (ignoring feature)
'-avx512pf' is not a recognized feature for this target (ignoring feature)
 FPS: 78 FrameTime: 12.897 ms
=======================================================
                                  glmark2 Score: 77
=======================================================
```

Производительность 64-х битной версии ОС FreeBSD в части рендеринга 3D изображения на данному ноутбуке почти на 38% выше чем для 32-х битного версии!

12.6. Сводная таблица результатов тестирования

Для удобства я собрал все данные в одну таблицу, приведенную ниже. По ней четко видно, что на одной и той же машине 64-х битная ОС существенно обгоняет в производительности 32-х битную версию. Для кого-то это может показаться банальным, но для меня это было не очевидно, и, честно говоря я, ожидал несколько другого результата. Предлагаю в комментариях к статье обсудить возможные причины такого ускорения, может быть даже с примерами ассемблерного кода. Для затравки предлагаю посмотреть на способы организации системных вызовов в ОС FreeBSD, они сильно отличаются для 32-х битного и 64-х битного режимов. Также отличается метод передачи параметров при вызове библиотечных функций (calling convention). Но это уже тема следующей большой статьи.

Тип бенчмарка	**FreeBSD 14.3/i386**	**FreeBSD 14.3/amd64**	**Разница в %%**
CoreMark 1.0 многопоточный	16228	19242	18,57
STREAM Copy	2210	2214	0,18
STREAM Scale	2128	2109	-0,89
STREAM Add	2450	2418	-1,31
STREAM Triad	2290	2420	5,68
SSD read block, MB/sec	127	139	9,45
Gmark2 fullscreen, FPS	56	77	37,5