Таблица
файлов
исходников
операционной системы
Minix3.1.5
stable

Часть 1
kernel, drivers, servers, etc, tools

Предисловие к первому изданию
(версия 0.1)

Идея создания обзора исходников операционной системы Minix3 возникла при попытке разобраться в том, почему функция mmap в Minix3 не работает так, как она работает в GNU/Linux. Попытка сделать это просто так (используя grep и редакторы программиста) оказалась безуспешной - исходники такого большого проекта, как Minix3, напоминают своего рода лабиринт. Для того, чтобы вторая попытка была успешнее, было решено делать небольшой анонс каждого «коридора этого лабиринта», т.е. файла исходников. Для того, чтобы частично результаты этой работы можно было использовать и для анализа других системных вызовов Minix3, было решено все анонсы заносить в таблицу. Надо сразу сказать, что целью было не только проанализировать причины того, что mmap в Minix3 не работает так, как хотелось бы, но и попробовать изменить ситуацию так, чтобы этот функционал можно было бы использовать для организации функционала dlfcn.h. Поэтому в анонсе сразу планировалось отметить то, насколько будет необходимо изменить тот или иной файл. При этом параллельно происходил поиск учебной задачи по видоизменению микроядра Minix3.1.5 (*вообще-то я не сразу ориентировался именно на эту версию, но к моменту начала обзора данная версия уже была, а когда вопрос встал о формировании обзора для сообщества, то выбор пал на неё, как новую стабильную версию*), а меня не устраивала учебная задача в духе «Hello, World!» (я тогда присматривался к вопросу сохранения расширенных регистров, ибо этого в версии Minix3.1.5 stable нет, а интуиция мне подсказывала, что для этого потребуется изменить довольно мало файлов). Всё это также требовало анализа файлов исходников Minix3.1.5. Всё это привело к идее расположить все краткие описания файлов в таблицу, а также саму аннотацию дополнить индексом, в зависимости от связи данного конкретного файла с остальными. Оформление кратких описаний в виде таблицы позволило бы свободно добавлять такие аннотации в произвольном порядке, а затем сортировать их как по расширенным именам («/kernel/proc.c , вместо proc.c»), так и по индексам. Постепенно бы формировалось описание исходников Minix3.1.5, ориентированных на потенциального разработчика Minix3.

Так как я предположил, что могу быть не одиноким в своих образовательных интересах, то я решил начало таблицы с описанием идеи разместить на форуме сайта www.minix3.ru . Попутно я хотел апробировать описания индексов, в котором я сомневался ибо не имел достаточной квалификации. Увы, но критики самих описаний индексов я не получил (вообще никакой, ни конструктивной, ни деструктивной). Вместо этого я получил предложение развить подобный обзор исходников Minix3.1.5 уже для русской версии Minix3, выход которой был запланирован на 23-е февраля 2010-го года. Это заставила кардинально изменить подход к формированию таблицы:

- требовалось охватить большое количество файлов (желательно – всю систему) в очень сжатые сроки,
- добавление аннотаций в таблицу должно было быть систематическим и планомерным для того, чтобы другой человек мог бы подключиться к заполнению таблицы, а главное – для того, чтобы к назначенному сроку получить полное описание части подсистем, вместо неполного описания всех подсистем,
- пришлось отложить практические задачи, которые были толчком для самой идеи создания обзора (что печально, так как снижает качество аннотаций).

После месяца работы стало возможным определить то количество файлов, которое реально успеть включить в таблицу. Стало ясно, что нереально охватить даже всю собственно систему, однако возможно довольно близко подойти к этому, включив в таблицу микроядро (подсистему /kernel/), серверы (подсистему /servers/), драйверы (подсистему

/drivers/), включаемые файлы (подсистему /include/), системный загрузчик (подсистему /boot/), а также вспомогательные файлы (подсистемы /etc/ и /tools/). Из собственно системы пришло временно отказаться от системной и пользовательской библиотеки (подсистема /lib/). Тесты (подсистема /test/) и системные утилиты (подсистема /commands/) были исключены из приоритетного рассмотрения потому, что не являются частями собственно операционной системы minix3.1.5 (/man/ и /docs/ не содержат файлов исходников).

При более подробном рассмотрении оказалось, что обзор включаемых заголовочных файлов требует совсем иного подхода, нежели чем обзор микроядра, серверов и драйверов. Фактически стало понятно, что для подсистемы /include/ целесообразно иметь отдельную таблицу аннотаций, содержащую большее количество индексов для сортировки. Именно тогда (в контексте времени выхода первой русской версии Minix3 – в самый последний момент) и сформировался полный план обзора:

основные:
Часть 1 (/boot/, /kernel/, /servers/, /drivers/, /etc/, /tools/);
Часть 2 (/include/);
Часть 3 (/lib/);

дополнительные (возможно даже, что их обзор нецелесообразен):
Часть 4 (/commands/);
Часть 5 (/test/).

Вместе с первой русской версией Minix3 выйдут первые две части обзора исходников операционной системы Minix3.1.5 (на которой и будет основана русская версия). Вся эта «история создания» для того, чтобы потенциальный читатель подошёл к версии 0.1 обзора довольно критично, однако с пониманием:
- у меня не было времени согласовывать с сообществом переводы терминов, поэтому очень часто они будут, мягко говоря, довольно спорными, НО при этом я всегда старался в скобках привести исходный английский вариант;
- так как «лобовая» попытка уточнить термины в интернете наткнулась на серьёзные трудности, то я привёл список всех моих терминологических затруднений в «прелюдии» к таблице (хоть это и тема для отдельной статьи, а лучше, если бы в рамках русской wiki был создан соответствующий словарь);
- индексы также имеют пока только ориентировочное значение (их соответствие заявленным описанием требует отдельной проверки, на которую у меня не было времени), следует их рассматривать как попытку разделить все файлы исходников на несколько неравных частей с надеждой, что изучение более малой части существенно облегчит понимание большей;
- «прелюдии» к таблицам не содержат никаких рекомендаций по практическому использованию (на практическую апробацию не было времени);
- текст не выдержан в соответствии с правилами оформления, объявленными для документации русской версии Minix3.

Исправление всех этих недостатков и будет основной целью версии 0.2. Единственной отрадной мыслью является то, что обзора исходников Minix3.1.5 по всей видимости нет вообще – даже на английском языке. А изменения по сравнению с Minix3.1.1 уже довольно значительные: эти и виртуальная память, это и виртуальная файловая система, это и новый сервер ipc. Как говориться: «На безрыбье и рак – рыба!»

В данном обзоре *нет непосредственных цитат*, поэтому в версии 0.1 *отсутствует* и *список используемой литературы*.В целом хочу сказать огромное спасибо русскому Minix-сообществу и особенно его фактическому руководителю - Роману Игнатову!

Введение

Таблица данной части обзора исходников операционной системы Minix3.1.5 создавалась фактически в одиночку и в довольно сжатые сроки. Это не могло не сказаться отрицательно на обзоре, однако все сомнительные части как в плане перевода, так и в плане аннотации отмечены знаком: (?)

Аннотации файлов исходников Minix3.1.5 в большинстве случаев являются переводами комментариев. Во многих случаях приведен также список функций публичного интерфейса. Сама таблица изначально планировалась для электронного использования, однако предполагается, что она будет полезна и при последовательном прочтении. В таблице не приведены make-файлы и depend-файлы.

Таблица предварена «прелюдией», состоящей из 3-х разделов.

В первом разделе изложены основные идеи создания обзора файлов исходников Minix3.1.5, в контексте описаний именно данной части обзора – системного загрузчика, микроядра, серверов, драйверов и вспомогательных файлов(/boot/, /kernel/, /servers/, /drivers/, /etc/, /tools/). Кроме того дана очень краткая аннотация файлов описываемых подсистем в целом.

Во втором разделе даны пояснения значениям столбцов таблицы (за исключением самого столбца описаний – предполагается, что его значение понятно и так; а также столбца расширенного имени файла, который дан в виде абсолютного пути, хотя правильнее было бы убрать начальный «/», превратив абсолютный путь относительный с «префиксом» в файловой иерархии Minix3.1.5 «/usr/src/»).

В третьем разделе очень формально приведен список терминов, перевод которых у меня вызвал затруднения. Предполагается, что это поможет как в дальнейшем улучшении обзора, так и лучшему пониманию текста анонсов файлов исходников.

1. *Основные идеи обзора и описываемые подсистемы в целом*

1.1. *Основная идея обзора*

Главным преимуществом проектов с открытыми исходниками является возможность переиспользования исходного текста как для других проектов, так и путём видоизменения и адаптации данного проекта к нуждам конкретной задачи. Переиспользование кода конкретного проекта подразумевает под собой анализ этого проекта другим программистом или группой программистов (т.е. людьми, не принимавшими участия в создании исходного проекта).

Для проекта операционной системы наибольшее значение имеет как раз задача видоизменения и адаптации к нуждам конкретной задачи: это и портирование на новую архитектуру, это и добавление новых системных вызовов, это и изменение свойств самой системы с сохранением программного интерфейса (например, замена алгоритмов планирования , чтобы обеспечить нужды задач реального времени).

При видоизменении проекта главным вопросом становится: «Как добиться желаемого результата, потратив минимальное время на изучение проекта?» Одним подходом к решению данной проблемы является использование эффективных методов анализа программных проектов (в.т.ч. с применением программных средств, таких как grep, cflow, doxygen, OpenGrok). Другим подходом к решению этой проблемы является стиль и дисциплина программирования самих разработчиков исходного проекта (в.т.ч. с учётом, например, возможностей doxyen). Третьим подходом является написание документации проекта, ориентированной именно на его переиспользование (т.е. документации не для пользователя, а для разработчика).

Создание индексированной таблицы анонсов файлов исходников является одним из видов подобной документации, ориентированной на переиспользование проекта. В данном случае подразумевается только один аспект переиспользования - видоизменение и адаптация проекта. Текст подобной индексированной таблицы будет эффективен, если:
(a) *прочтение анонса будет более быстрым и лёгким процессом, нежели чем прочтение самого файла исходника* (хотя бы потому, что не требуется открывать дополнительный файл; именно поэтому порой целесообразно очень маленькие файлы просто копировать в соответствующую ячейку таблицы - для профессионала код часто говорит сам за себя лучше любого комментария)
(b) *прочтение анонсов позволит существенно сократить число файлов, исходники которых всё равно придётся просмотреть*;
(c) *индексы и анонсы позволят выбрать наиболее эффективный порядок ознакомления* с исходниками проекта для каждой конкретной задачи его видоизменения и адаптации.

В данной работе сделана попытка создать именно такую таблицу анонсов файлов исходников операционной системы Minix3.1.5.

1.2. *Базовая гипотеза.*

Предполагается, что проект состоит из отдельных подсистем, и при этом каждая подсистема
- *понятна сама по себе* (для её понимания не требуется изучение других подсистем),
- состоит из *файлов* (точнее пар: c-файл, h-файл, - специфика практики языка программирования C), *реализующих* отдельные *части* подсистемы, *и файлов, объединяющих эти части* в единое целое – подсистему (при этом может выстраиваться целая иерархия).

Данное предположение основывается на стиле структурированного

программирования, естественно соответствующем основному языку программирования проекта Minix3 – языку программирования C.

В свете данного предположения требуется отделить файлы, реализующие части функций подсистемы, от файлов, объединяющих эти части в единое целое. Для добавление новых функций в подсистему скорее всего придётся добавить дополнительные файлы для реализации новых функций, и видоизменить объединяющие файлы так, чтобы добавленная часть была органично включена в единое целое. Для удаления или (что более вероятно) видоизменения некоторых функций скорее всего потребуется видоизменить соответствующие файлы, реализующие соответствующие части функционала. При этом может потребоваться (а может и не потребоваться) откорректировать объединяющие файлы.

Таким образом сразу можно сказать, что подавляющее большинство файлов, анонсируемых в данной части обзора исходников опреационной системы Minix3.1.5 будет иметь индекс «3». Подавляющая часть интегрирующих файлов будет иметь индекс «5».

1.3. Особенности проекта Minix3 (в яастности - Minix3.1.5)

Любая операционная система реализует некоторые стандарты (как свои – внутренние, так и внешние, например, стандарты UNIX). Вполне естественно, что эти стандарты найдут непосредственное отражение ив файлах исходников проекта. Понятно, что разработчик, вносящий изменения в операционную систему и изучающий по анонсу исходники системы должен быть предупреждён относительно того, что данный файл непосредственно связан с реализацией того или иного внутреннего или внешнего стандарта и его видоизменение может существенно и нежелательно изменить заявленное поведение операционной системы (например, перестанут выполняться ранее выполнявшиеся приложения, или даже прекратить успешно компилироваться).

Операционная система Minix развивалась в течение очень долгого периода времени, поэтому некоторые изначальные идеи существенно изменили своё значение. В частности, конфигурационный файл микроядра, определяющий набор вызовов микроядра, изначально был предназначен для гибкой перенастройки микроядра. На данный момент набор вызовов микроядра является внутренним стандартом Minix3, поэтому очень сомнительно, что подобная «тонкая настройка» микроядра приведёт к работоспособной операционной системе (поэтому и соответствующий конфигурационный файл имеет индекс «2», а не, например, «7»).

Существуют некоторые пары файлов, которые требуется поддерживать в строгом соответствии. В Minix3.1.5 максимально отказываются от динамического размещения памяти, динамической регистрации существенно системной информации и других подобных вещей внутри операционной системы. Поэтому имеются другие не столь явные, но всё же довольно жёсткие связи. Это просто надо иметь в виду при внесении изменений. Иначе скорее всего возникнет паника микроядра с выходом в системный монитор.

Имеются также особенности проекта Minix3.1.5, связанные с микроядерной архитектурой:
- нет строгой грани между пользовательской и системной библиотеками, а LIBC существенно и, я бы сказал – нетривиально взаимодействует с операционной системой (не забываем, что ядру, например, GNU/Linux соответствует набор: микроядро + серверы + драйверы; при этом и драйверы, и серверы имеют очень много общего с обычными пользовательскими процессами);
- к типичным для всех операционных систем библиотечным и ядерным аспектам

взаимодействия подсистем операционной системы между собой и с процессами пользователя, добавляется ещё и аспект внутрисистемного взаимодействия,
- выполнение системных вызовов происходит унифицированным способом – посредством обмена стандартными сообщениями, поэтому подход к анализу функционирования операционной системе в микроядерной мультисерверной архитектуре существенно отличается от анализа функционирования операционной системы с монолитным ядром.

Обзор файлов исходников в виде таблицы отражает в основном место каждого файла в иерархии вызовов функций данной подсистемы, а также отражение внешних и внутренних стандартов, функции взаимодействия подсистемы с остальными. Вышеназванные особенности, связанные с микроядерной архитектурой в таблице отражены слабо. При анализе проекта микроядерной мультисерверной операционной системы информацию из таблицы требуется существенно дополнить анализом передачи, видоизменения и реагирования на сообщения. Требуется провести не только анализ «траектории программных вызовов», но и анализ межпоцессных взаимодействий – анализ «траектории системных сообщений».

2. *Пояснения к таблице*

2.1. *Смысл значений колонны* «pr.»:

1 – этот файл не стоит видоизменять, разве что в исключительном случае; (*файл содержит в себе информацию, связанную со стандартом UNIX, ANSI, соглашениями, принятыми в сообществе разработчиков Minix3; поэтому и цель изменить этот файл может ставиться исключительно только после согласования с сообществом разработчиков Minix3*)

2 – этот файл лучше не видоизменять, а если и видоизменять, то очень и очень осторожно; (*хоть файл и не содержит ничего, непосредственно связанного со стандартами и соглашениями, тем не менее изменения в API и/или структурах данных приведут к значительному количеству изменений в других местах, которые скорее всего следует оценить отрицательно*)

3 – видоизменять имеет смысл только если это непосредственно оправдано (т.е. ставится такая цель в формулировке: «перепишем функционал и структуры данного файла»); (*не стоит сюда добавлять что-то не связанное с уже имеющейся информацией – это будет противоречить принципу модульности; для нового элемента подсистемы лучше создать новый файл;*)

4 - видоизменять имеет смысл только если это непосредственно оправдано, но в файле используются функции, описания которых не ограничены стандартами и/или соглашениями, а также не имеют глобального значения (см. pr. 2); изменения прототипа любой из этих функций потребует внесения изменений и в этот файл; (*не стоит сюда добавлять что-то не связанное с уже имеющейся информацией ; однако может возникнуть необходимость внести изменения, связанные с изменением прототипов используемых функций; этот вид индекса наиболее характерен для «интегрирующих» файлов – содержащих, например, main(), init(), ... подсистемы;*)

[Примечание (1): *Если мы собираемся добавить новый элемент в подсистему, то мы автоматически ставим непосредственную цель изменить интегрирующий файл – иначе просто этот новый элемент так и останется неиспользованным!*]

5 – файл скорее всего также придётся менять, если вносятся изменения в подсистему (обычно это соседние файлы соответствующей папки исходников);(*данный файл содержит глобальные данные для подсистемы; если включение нового элемента также подразумевает свои глобальные данные, то их скорее всего надо размещать в этом файле;этот вид индекса также характерен для «интегрирующих» файлов – содержащих, например, main(), init(), ... подсистемы, однако только в том случае, если в его начале находятся ещё и глобальные данные;*)

6 – файл содержит данные используемые для связи данной подсистемы (какой – указано в аннотации) с остальными подсистемами; (*однако изменения в этот файл вносятся только при изменении данной подсистемы, но они могут повлиять и на другие подсистемы – в случае изменения этого файла скорее всего потребуется что-то поменять и в других подсистемах*)

7 – файл содержит общие данные хотя бы для двух разных подсистем; информация данного файла не может быть непосредственно связана с какой-то одной подсистемой; эти данные обычно не требуется менять при добавлении системных вызовов, новых сообщений,..., разве, что это непосредственно связано с данными подсистемами; (*если ставиться задача добавить системный вызов, новое сообщение, ... , то всё же <u>не возникает автоматически</u> вопрос о необходимых изменениях в данном файле; изменения в файле всё же инициированы внутренними изменениями в подсистемах, а не внешними изменениями в работе системы;*)

8 – файл часто требуется менять при добавлении системных вызовов, новых сообщений, других существенных изменениях в работе OS Minix3, причём вне зависимости от изменений в подсистеме - соседних файлах в папке; (*если ставиться задача добавить системный вызов, новое сообщение, ... , то <u>автоматически возникает</u> вопрос о необходимых изменениях в данном файле;*)

9 – файл придётся менять почти всегда (за редким исключением);

2.2. *Смысл значений колонны* «aut.»:

FW – Борманис Виктор Янович (FireWall)

VS - Valery Solovey

3. *Список терминов и фраз, перевод которых может быть спорным*

(1) Набор битов (bit map) сигнала
/kernel/system/do_getksig.c

(2) точка окончания (endpoint (?))
/kernel/system/do_privctl.c

(3) Идентификатор разрешения (grant id)
/kernel/system/do_safecopy.c

(4) Возвращает индекс опосредованной(?) (remote) памяти.
/kernel/system/do_segctl.c

(5) Выполняет запрос на синхронный сигнал(?) (synchronous alarm), или на отмену синхронного сигнала.
/kernel/system/do_setalarm.c

(6) Адрес таблицы разрешений (?) (grant table)
/kernel/system/do_setgrant.c

(7) Число записей(?) (number of entries)
/kernel/system/do_setgrant.c

(8) Процесс, возвращающийся из (?)(handler).
/kernel/system/do_segreturn.c

(9) Указатель на структуру контекстов сигналов(?) (sigcontext structure).
/kernel/system/do_segreturn.c

(10) Процесс для вызова обеспечения сигнала (?) (signal handler) .
/kernel/system/do_sigsend.c

(11) Частота запрашиваемого образца(?)(sample).
/kernel/system/do_sprofile.c

(12) Конечная точка (?) (endpoint) запрашивающего.
/kernel/system/do_sprofile.c

(13) Устанавливает информацию о времени (?)(times) в сообщение.
/kernel/system/do_times.c

(14) Новое/старое время достижения (?)(expiration time) в тиках.
/kernel/system/do_vtimer.c

(15) Индексами таблицы являются главный номер устройства ((?) major device number).
/servers/vfs/dmap.h

(16) В нём также содержаться некоторые процедуры для динамического добавления, удаления драйверов, а также изменения связей ((?) mappings).
/servers/vfs/dmap.c

(17) заботится (?)(take care)
/servers/vfs/exec.c

(18) Таблица ... (?)***filp***. Прослойка между дескрипторами файлов и ... (?) inodes. Слот свободен , если filp_count == 0.
/servers/vfs/file.h

(19) Этот файл обеспечивает вложенные контр-запросы ((?) counter-request) серверу виртуальной файловой системы (VFS), посылаемые файловыми серверами (FS: MFS, iso9660FS, ...) в ответ на запрос сервера виртуальной файловой системы (VFS).
/servers/vfs/fscall.c

(20) что ему необходимо найти vnode (?) на который партиция этого процесса FS смонтирована.
/servers/mfs/mount.c

(21) Читает одну запись по данному «косвенному» блоку (indirect block)(?).
/servers/mfs/read.c

(22) Поддержка совместимости: обёртка getdents, возвращающая результирующее число байтов в поле m_type ответного сообщения reply message (?).
/servers/mfs/read.c

(23) Таблица суперблоков (содержит записи для базовой (root (?)) файловой системы и точек монтирования).
/servers/mfs/super.h

(24) Эта процедура возвращает время в секундах с 1.1.1970. MINIX является астрофизически наивной системой, которая подразумевает, что земля крутится с постоянной скоростью и таких вещей, как прыгающие секунды ((?) leap seconds), не существует

/servers/mfs/utility.c

(25) Этот файл обеспечивает системные вызовы, связанные с отложенными процессами (системным будильником (?) (alarm clock)), периодически посылая работу функциям в файле «timer.c» и check_sig() в файле «signal.c», для посылки сигнала пробуждения ((?) larm signal) процессу.
/servers/pm/alarm.c

(26) Кроме всего прочего, она определяет сегменты кода, данных и стека, uids and gids (?), а также различные флаги.
/servers/pm/mproc.h

(27) Управление временем «сторожевого пса» ((?) watchdog) сервера управления процессами. Функции этого файла обеспечивают удобный интерфейс к библиотеке таймеров, которая управляет списком таймеров watchdog (?). Все детали планирования и сигналов пробуждения (alarm (?)) задания микроядра CLOCK сокрыты этим интерфейсом.
/servers/pm/timers.c

(28) Этот сервис обеспечивает сохранение небольших (publish/subscribe (?)) данных, критичных для устойчивости при системных сбоях.
/servers/ds/main.c

(29) Эта процедура проходит сквозь все записи для данного клиента и проверяет все элементы данных, если они были обозначены (т.е. созданы или изменены) ((?) matching that subscription.) Возвращает сообщение и копию ключей и значений для каждого.
/servers/ds/store.c

(30) Этот файл содержит интерфейс для сетевого программного обеспечения (with rest of minix (?)).
/servers/inet/inet.c

(31) Обеспечивает запросы на параметры вопросов ((?) queryparams).
/servers/inet/qp.h

(32) Внутреннее определение RS ... ((?) SR internals)
/servers/inet/sr_int.h

(33) Если файлы /usr/adm/wtmp и /etc/utmp существуют и доступны для записи, init (при помощи login) будет сопровождать учётные записи авторизации (?) (login accounting.)
/servers/init/init.c

(34) Этот файл содержит процедуры по выдаче (?) (dump) информации (внутренней информации о системе).
/servers/is/dmp.c

(35) структур данных сервера данных (?) (DS)
/servers/is/dmp_ds.c

(36) (или содержимое proc FS (?))
/servers/is/dmp_fs.c

(37) максимальную длину «спасительного» каталога, а также максимальное число PCI ID, различных классов PCI ID (?), MAX_NR_SYSTEM 2 /* should match RSS_NR_SYSTEM */ (?), максимальное число списка имён целевых процессов для межпроцессного взаимодействия (IPC) , MAX_VM_LIST 256 , ...
/servers/rs/manager.h

(38) Дан запрос на запуск нового системного сервиса. Метит (удаляет - dismember (?)) сообщение запроса, и собирает всю информацию, необходимую для запуска сервиса. Запуск осуществляется при помощи вспомогательных процедур.
/servers/rs/manager.c

(39) Ключевая используемая структура данных – таблица свободных участков ((?) hole table), которая поддерживает список свободных участков памяти.
/servers/vm/alloc.c

(40) Модель выделения памяти MINIX резервирует участки памяти фиксированного размера ((?) a fixed amount of memory) для комбинированных сегментов кода, данных и стека.
/servers/vm/break.c

(41) Абстрактное дерево AVL (?) - общий пакет.
/servers/vm/cavl_if.h

(42) Переводит байтовые сдвиги и размеры в этом списке в клики, нарезанные правильным образом. ((?) to clicks).
/servers/vm/utility.c

(43) Выбирает память сервера из списка свободной памяти. Монитор загрузки ((?) boot monitor) обещает поместить процессы в начале чанков памяти ((?) chunks). Все задания (микроядра) используют одинаковый базовый адрес, таким образом только первое задание изменяет списки памяти ((?) memory lists). Серверы и init имеют свои собственные пространства памяти и их память в дальнейшем будет удалена из этого списка (свободной памяти).
/servers/vm/utility.c

(44) (?) PDE используется для отображения в микроядро, физические адреса микроядра.
/servers/vm/i386/pagetable.c

(45) Выделяет основу для таблицы страниц ((?)root - заготовку?).
/servers/vm/i386/pagetable.c

(46) Указывает процессору об tlb записи ... (?)
/servers/vm/i386/util.s

(47) Начинается с констант, определяющих ID ((?) идентификатор) производителя и самого устройства.
/drivers/audio/es1370/es1370.h

(48) /* Функция, информирующая модуль CODEC о том какой идентификатор производителя AC97 ((?)vendor ID) предполагать. */
/drivers/audio/es1371/codec.h

(49) Начинается с констант определяющих режим синхронизации (SRC (?))
/drivers/audio/es1371/codec.c

(50) (*) Производительность составляет примерно 10% от AT драйвера для чтения и записи на 2:1 перемежающимся диске ((?) 2:1 interleaved disk) , ((?) it will be DMA_BUF_SIZE bytes
/drivers/bios_wini/bios_wini.c

(51) Определения для (MII) Независимого от Среды Интерфейса ((?) Media Independent (Ethernet) Interface)
/drivers/fxp/mii.h

(52) Вычисляет необходимые ключи ((?) round keys). Объём вычислений зависит от keyBits и blockBits.
/drivers/random/aes/rijndael_alg.c

(53) Этот файл содержит драйвер терминала, как ((?) both) для консоли IBM, так и для стандартного ((?) regular) терминала ASCII.
/drivers/tty/tty.c

(54) и несколько вспомогательных ((?) minor) точек входа для использования кодом, зависимым от устройства.
/drivers/tty/tty.c

(55) Вывод на терминал (TTY) посылается на зависимую от устройства часть для обработки вывода и «показа на экране» ((?) "screen" display.).
/drivers/tty/tty.c

(56) Псевдо терминал (PTY) может быть виден как двунаправленный канал ((?) pipe) с терминалом (TTY) для обработки ввода и вывода.
/drivers/tty/pty.c

(57) Выводит бинарный расклад клавиатуры ((?) binary keymap).
/drivers/tty/keymaps/genmap.c

4. *Собственно таблица анонсов файлов дерева исходников подсистем* /kernel/ , /servers/ , /drivers/, /etc/, /boot/, /tools/

pr.	***path***	***an.***	***aut***
3	**/kernel/clock.h**	Определяет функции для инициализации таймера и обработчиков таймера. Этот файл присутствует в SVC но отсутствует в стабильной версии 5.1.5 (!)	FW
4	**/kernel/clock.c**	Содержит функции для инициализации таймера и обработчиков таймера. Содержит как обработчик аппаратного прерывания, так и бесконечный цикл для обработки событий. Важные события, обрабатываемые посредством CLOCK, включают также решения по планированию и перепланированию процессов. CLOCK предлагает непосредственный интерфейс с процессами микроядра. Системные службы могут обращаться (к данному файлу) посредством системных вызовов, таких как sys_setalarm(). CLOCK задание поэтому полностью сокрыты для внешнего мира. При видоизменении данного файла нельзя использовать send() если получатель не готов принять сообщение. Вместо этого желательно использовать notify(). CLOCK – это один из процессов микроядра, не вынесенный в отдельный сервер по соображениям производительности. Эта функция сейчас находится в стадии усовершенствования (!).	FW
2	**/kernel/config.h**	Определяет конфигурацию микроядра. Позволяет установить размеры буферов ядра, включить или исключить отладочный код, функции контроля времени и отдельные вызовы микроядра. (поэтому здесь содержаться краткие их описания, тем не менее настоятельно рекомендуется сохранять все вызовы микроядра включёнными). Этот файл по логике должен быть одним из основных координирующих файлов, однако похоже, что эта функция уже давно стала атавизмом ... *Поэтому вместо должного индекса **8**, стоит **2**.*	FW
3	**/kernel/const.h**	Содержит макросы и константы, используемые в коде микроядра. В частности, содержит макросы для запрещения и разрешения аппаратных прерываний Общие определения и макросы помещаются в этом файле.	FW
3	**/kernel/debug.h**	Определяет все отладочные константы и макросы,а также некоторые (глобальные) переменные. Некоторые отладочные функции требуют переопределения стандартных констант и макросов, поэтому данный заголовочный файл должен находится ПОСЛЕ остальных заголовочных файлов микроядра.	FW
3	**/kernel/debug.c**	Этот файл содержит отладочные функции, не включённые в стандартное микроядро. Доступные функции включают отсчёт времени для блокировок и проверочные функции для очередей управления.	FW
5	**/kernel/glo.h**	*Определяет используемые в микроядре глобальные переменные.* (Сами переменные размещаются в table.o ; что достигается определением _TABLE.) На данный момент определяет переменные режима ядра (исключение, выход из системы); переменные-структуры информации о ядре, машине, диагностические сообщения, генератора случайных чисел, средней загрузке; указатели на текущий выполняющийся процесс, следующий процесс	FW

pr.	*path*	*an.*	*aut*
		после restart(), процесс для подсчёта тиков (/* process to bill for clock ticks */), первые процессы в очередях restart, request, pagefault; переменную учёта тиков, не учтённых в задании CLOCK;	
3	**/kernel/interrupt.c**	Система аппаратных прерываний Minix3. Содержит процедуры для управления контроллером прерываний: - регистрации , а также удаления обработчика прерываний, - PUBLIC void irq_handle(int irq) вызывается системнозависимой частью при возникновении внешнего прерывания; - разрешения , запрещения линии прерываний. Определяет переменную: PUBLIC irq_hook_t* irq_handlers[NR_IRQ_VECTORS] = {0};	FW
3	**/kernel/ipc.h**	Этот файл определяет константы для межпроцессного взаимодействия (MINIX или микроядро?). Определения используются в /kernel/proc.c . Пока это константы NON_BLOCKING, SEND, RECEIVE, SENDREC, NOTIFY, SENDNB, SENDA; а также макрос WILLRECEIVE(target, source_ep) .	FW
3	**/kernel/main.c**	Описывает начальный старт микроядра (оно находится ещё в загрузочном образе и уже имеет набор готовых к исполнению системных процессов – серверов).	FW
3	**/kernel/kernel.h**	Основной заголовочный файл микроядра Minix3. Однако сам по себе он состоит только из заголовков. В частности, он включает почти все заголовочные файлы из /kernel/ .	FW
3	**/kernel/priv.h**	Определяет структуру системы привилегий *struct priv*. Каждый системный процесс имеет собственную структуру привилегий, для всех пользовательских процессов используется одна структура привилегий (#define USER_PRIV_ID 0).	FW
2	**/kernel/proc.h**	Определяет таблицу процессов. Таблица процессов включает - состояние регистров и флагов, - приоритет планировщика, - таблица памяти (физической или логической?), - различные учётные данные (ID , PID ?), - информацию, используемую для передачи сообщений (IPC). Многие ассемблерные процедуры обращаются к полям этой структуры. Смещения определены в ассемблерном включаемом файле kernel/arch/i386/sconst.h . Эти два файла должны соответствовать друг другу! Кроме того определены флаги исполнения (runtime) и макросы, для их проверки, установки, очистки. ... Иными словами - определяет структуру *proc*, флаги и макросы, для работы с ней, а также массивы: EXTERN struct proc proc[NR_TASKS + NR_PROCS]; /* process table */ EXTERN struct proc *rdy_head[NR_SCHED_QUEUES]; /* ptrs to ready list headers */ EXTERN struct proc *rdy_tail[NR_SCHED_QUEUES]; /* ptrs to ready list tails */	FW
3	**/kernel/proc.c**	Вместе с "mpx.s" этот файл содержит наиболее низкоуровневую часть (lowest layer) микроядра. Иммеется одна входящая точка для внешних вызовов: sys_call: системный вызов, когда мы попадаем в микроядро посредством INT. Имеется также несколько (?) точек входа для прерываний и уровня заданий: lock_send: послать сообщение процессу.	FW

pr.	*path*	*an.*	*aut*
		Следует сразу заметить, что файл имеет 1452 строки! Поэтому даже поверхностное описание его выходит за рамки данной аннотации. Следует только заметить, что PROC является одним из процессов микроядра не вынесенным в отдельный сервер по соображениям производительности.(?)	
3	**/kernel/profile.h**	Определяет переменные для профилирования (GSCo 2009?). Зависит от /include/minix/profile.h Содержит общую опцию #if SPROFILE /* statistical profiling */	FW
3	**/kernel/profile.c**	Этот файл содержит несколько функций и переменных, используемых для системного профилирования. Статистическое профилирование (Statistical Profiling): обработчик прерываний для часов профилирования. Профилирование вызовов (Call Profiling): таблица, используемая для данных профилирования, а также функция для определения её размеров; функция используется процессами микроядра (kernelspace processes) для регистрации их управляющих структур (control struct) и таблицы профилирования.	FW
5	**/kernel/proto.h**	Файл содержит все(?) прототипы функций (публичного интерфейса), определённых в файлах clock.c, main.c, utility.c, proc.c, start.c, system.c, system/do_newmap.c, system/do_vtimer.c, interrupt.c, debug.c, system/do_safecopy.c, system/do_sysctl.c, profile.c подсистемы /kernel , включая части, зависящие от конкретной платформы.	FW
6	**/kernel/start.c**	Первый C-файл, используемый микроядром. (Файл связывает подсистему /kernel с подсистемой /boot (?).) Содержит функции: PUBLIC void cstart(cs, ds, mds, parmoff, parmsize) используется для инициализации системы до вызова main() (функция возможно работает в реальном режиме процессора (?), большинство параметров определяется с помощью переменных окружения, передаваемых загрузчиком Minix3); PRIVATE char *get_value(params, name) определяет значение параметра окружения (environment value) – версия getenv для микроядра, дабы избежать создания обычного массива (переменных) окружения (usual environment array).	FW
8	**/kernel/system.h**	Прототипы функций системной библиотеки (микроядра ?). Сами функции находятся в /kernel/system/ . Если вызов микроядра не включён посредством конфигурации /kernel/config.h , то функция становится синонимом do_unused(). Системная библиотека делает доступным системный сервис (system services) посредством вызовов микроядра. Системные вызовы трансформируются в сообщения-запросы (request messages) к заданию SYS, способному выполнить соответствующий вызов. По соглашению sys_call() преобразуется в сообщение с типом SYS_CALL, которое обрабатывается (handled) функцией do_call(). *[Примечание: call заменяется на конкретное название запроса , например, copy]*	FW
8	**/kernel/system.c**	Обеспечивает интерфейс между микроядром и серверами - интерфейс	FW

pr.	*path*	*an.*	*aut*
		системных вызовов. Для отображения системных вызовов на функции, их обеспечивающие применяется внутренний вектор вызовов (private call vector). Сами эти функции находятся в отдельных файлах (см. /kernel/system/). Вектор вызовов используется в основном цикле системного задания (system task's main loop) для обработки входящих запросов. Кроме основной точки входа (*sys_task()*), в котором и запускается основной цикл, имеется ещё несколько (minior) точек входа: *get_priv*: заполняет структуру привилегий для пользовательского или системного процесса, *set_sendto_bit*: позволяет процессу послать сообщение в новом направлении (расширяет привилегии), *unset_sendto_bit*: запрещает процессу послать сообщение в новом направлении (сужает привилегии), *send_sig:* посылает сигнал прямо системному процессу, *cause_sig:* выполняет действие вызванное сигналом вызывая событие через сервер управления процессами (to cause a signal to occur via PM), *sig_delay_done:* сообщает серверу управления процессами что процесс не посылает (tell PM that a process is not sending), *umap_bios:* отображает виртуальный адрес в BIOS_SEG на физический, *get_randomness:* накапливает случайности в буфер, *clear_endpoint:* лишает (remove) процесс возможности посылать и принимать сообщения. SYS является одним из заданий микроядра.	
8	**/kernel/table.c**	Содержит большинство данных микроядра. Непосредственно в данном файле определены константы препроцессора: - для размеров стеков заданий микроядра, - флаги для различных типов процессов (микроядра?), - списки FS_C и DRV_C (разрешённых высовов микроядра), макросы препроцессора: - для определения масок системных вызовов для различных типов процессов, глобальные переменные: - fs_c[] , pm_c[] , rs_c[] , ds_c[] , vm_c[] , drv_c[] , usr_c[] , tty_c[] , mem_c[] , - PUBLIC struct boot_image image[] EXTERN внутри данного файла принимает значение пустой строки, поэтому глобальные переменные определённые в заголовочных файлах реально привязываются к данному файлу (место им выделяется в table.o).	FW
2	**/kernel/type.h**	Определяет типы, связанные с таблицей процессов и другими свойствами (переменными) системы (микроядра): task_t , proc_nr_t , sys_id_t , sys_map_t ,struct boot_image , irq_policy_t , irq_id_t , struct irq_hook , irq_hook_t , irq_handler_t	FW
3	**/kernel/utility.c**	Этот файл содержит коллекцию различных процедур: *minix_panic:* прерывает MINIX в связи с фатальной ошибкой, *kputc:* буферизированный putc, используемый функцией kprintf, *kprintf:* посредством включения файла lib/sysutil/kprintf.c и замены	FW

pr.	*path*	*an.*	*aut*
		printf на kprintf посредством препроцессора.	
2	**/kernel/vm.h**	Содержит константы препроцессора: VMSUSPEND , EFAULT_SRC , EFAULT_DST , макросы: FIXLINMSG(prp) , PHYS_COPY_CATCH(src, dst, size, a). Файл связан с механизмами виртуальной памяти (?).	FW
3	**/kernel/system/do_abort.c**	Реализуется вызов микроядра : m_type: SYS_ABORT (прерывание работы OS Minix3) Параметры: m1_i1: ABRT_HOW (как выполнить прерывание работы OS Minix3, возможно взяв параметры монитора – системного загрузчика (?)) m1_i2: ABRT_MON_ENDPT (номер процесса параметры монитора которого берутся) m1_i3: ABRT_MON_LEN (длина параметров монитора) m1_p1: ABRT_MON_ADDR (виртуальный адрес параметров)	FW
3	**/kernel/system/do_copy.c**	Реализуется вызовы микроядра: m_type: SYS_VIRCOPY, SYS_PHYSCOPY (копирование областей виртуальной и/или физической памяти) Параметры: m5_c1: CP_SRC_SPACE (виртуальный сегмент копируемых данных) m5_l1: CP_SRC_ADDR (смещение копируемых данных относительно сегмента) m5_i1: CP_SRC_PROC_NR (номер процесса из виртуальной памяти которого происходит копирование) m5_c2: CP_DST_SPACE (виртуальный сегмент, куда копируются данные) m5_l2: CP_DST_ADDR (смещение относительно виртуального сегмента, куда копируются данные) m5_i2: CP_DST_PROC_NR (номер процесса, в виртуальное адресное пространство которого копируются данные) m5_l3: CP_NR_BYTES (размер копируемых данных в байтах)	FW
3	**/kernel/system/do_cprofile.c**	Реализуется вызов микроядра: m_type: SYS_CPROFILE (профилирование вызовов) Параметры: m7_i1: PROF_ACTION (получить/сбросить данные профилирования) m7_i2: PROF_MEM_SIZE (доступная память для данных) m7_i4: PROF_ENDPT (узловая точка (endpoint) вызывающего) m7_p1: PROF_CTL_PTR (расположение информационной структуры) m7_p2: PROF_MEM_PTR	FW

pr.	*path*	*an.*	*aut*
		(местоположение памяти для данных)	
3	**/kernel/system/do_devio.c**	Реализуется вызов микроядра: m_type: SYS_DEVIO (Осуществляет низкоуровневый ввод/вывод в порты ввода/вывода) Параметры: m2_i3: DIO_REQUEST (запрос на ввод или вывод) m2_l1: DIO_PORT (порт для ввода/вывода) m2_l2: DIO_VALUE (значение для записи/ возвращает прочтённое значение)	FW
3	**/kernel/system/do_endksig.c**	Реализуется вызов микроядра: m_type: SYS_ENDKSIG (Вызывается сервером управления процессами (PM) после обработки сигнала процессу SYS_GETKSIG. Обычно это сигнал прерывания процесса.)(?) Параметры: m2_i1: SIG_ENDPT (Процесс, для которого выполнено задание сервера управления процессами (PM))	FW
3	**/kernel/system/do_exec.c**	Реализуется вызов микроядра: m_type: SYS_EXEC (Замена контекста процесса.) Параметры: m1_i1: PR_ENDPT (Процесс, вызвавший exec.) m1_p1: PR_STACK_PTR (Новый указатель на стек: создаётся (?)) m1_p2: PR_NAME_PTR (Указатель на имя программы) m1_p3: PR_IP_PTR (Новый указатель инструкций: создаётся (?))	FW
3	**/kernel/system/do_exit.c**	Реализуется вызов микроядра: m_type: SYS_EXIT (Завершает процесс.) Параметры: m1_i1: PR_ENDPT (Номер слота завершающегося процесса.)	FW
3	**/kernel/system/do_fork.c**	Реализуется вызов микроядра: m_type: SYS_FORK (Создание нового процесса – копии родительского.) Параметры: m1_i1: PR_ENDPT (Родитель, процесс, который «разветвляется».) m1_i2: PR_SLOT (Слот порождаемого процесса-ребёнка в таблице процессов.) m1_p1: PR_MEM_PTR (Новая карта памяти для процесса-ребёнка.) m1_i3: PR_FORK_FLAGS (Флаги – параметры вызова fork.)	FW
3	**/kernel/system/do_getinfo.c**	Реализуется вызов микроядра: m_type: SYS_GETINFO	FW

pr.	*path*	*an.*	*aut*
		(Запрос на системную информацию, которая копируется в адресное пространство запрашивающего процесса. Этот вызов просто копирует соответствующие структуры данных запрашивающему процессу.) Параметры: m1_i3: I_REQUEST (Какую информацию?) m1_p1: I_VAL_PTR (Куда её поместить ?) m1_i1: I_VAL_LEN (Максимальная возможная длина (может не быть).) m1_p2: I_VAL_PTR2 (Второй параметр (может не быть).) m1_i2: I_VAL_LEN2_E (Вторая длина или номер процесса.)	
3	**/kernel/system/do_getksig.c**	Реализуется вызов микроядра: m_type: SYS_GETKSIG (Сервер управления процессами (PM) готов обрабатывать сигналы и периодически делает (данный (?)) вызов микроядра для получения очередного сигнала. ...) Параметры: m2_i1: SIG_ENDPT (Процесс, посылающий сигнал.(?)) m2_l1: SIG_MAP (Набор битов (bit map) сигнала.(?))	FW
3	**/kernel/system/do_irqctl.c**	Реализуется вызов микроядра: m_type: SYS_IRQCTL (Позволяет , в частности, вставить новый обработчик прерываний (?). Возвращает индекс ловушки прерывания, назначенный в микроядре.) Параметры: m5_c1: IRQ_REQUEST (Контрольная операция (control operation), которую надо выполнить.) m5_c2: IRQ_VECTOR (Линия прерываний, которая должна быть проверена.) m5_i1: IRQ_POLICY (Позволяет вновь разрешить прерывания.) m5_l3: IRQ_HOOK_ID (Предоставляет индекс, который будет возвращён при прерывании.)	FW
3	**/kernel/system/do_kill.c**	Реализуется вызов микроядра: m_type: SYS_KILL (Обеспечывает sys_kill(). Вызывает посылку сигнала процессу. Сервер управления процессами (PM) – центральный сервер, где обрабатываются все сигналы и обеспечиваются регистрация порядка (policies) их обработки. Любой запрос, за исключением запросов сервера управления процессами (PM), добавляется в «карту» необработанных сигналов, а сервер управления процессами (PM) информируется о поступлении нового сигнала. Так как системные серверы не могут использовать нормальные POSIX сигналы (ввиду того, что они обычно блокируют процесс на их получении (RECEIVE)), они могут запросить сервер управления процессами (PM) преобразовать сигналы в сообщения. Это выполняется сервером управления процессами (PM) посредством вызова sys_kill().)	FW

pr.	*path*	*an.*	*aut*
		Параметры: m2_i1: SIG_ENDPT (процесс, которому посылается сигнал/ необработанный) m2_i2: SIG_NUMBER (Номер сигнала, который посылается процессу.)	
3	**/kernel/system/do_mapdma.c**	Реализуется вызов микроядра: m_type: SYS_MAPDMA (Выделяет область для выполнения операции устройства с непосредственным доступом к памяти. (?)) Параметры: m5_l1: CP_SRC_ADDR (Виртуальный адрес.) m5_l3: CP_NR_BYTES (Размер структуры данных.)	FW
3	**/kernel/system/do_memset.c**	Реализуется вызов микроядра: m_type: SYS_MEMSET (Записывает образец (данный байт (?)) в определённый участок памяти.) Параметры: m2_p1: MEM_PTR (виртуальный адрес) m2_l1: MEM_COUNT (возвращает физический адрес (точно ? Не число байт? (?))) m2_l2: MEM_PATTERN (байт-образец, которым заполняется область)	FW
3	**/kernel/system/do_newmap.c**	Реализуется вызов микроядра: m_type: SYS_NEWMAP (Создаёт новую карту памяти) Параметры: m1_i1: PR_ENDPT (устанавливает новую карту памяти для этого процесса) m1_p1: PR_MEM_PTR (указатель на новую карту памяти)	FW
3	**/kernel/system/do_nice.c**	Реализуется вызов микроядра: m_type: SYS_NICE (Изменяет приоритет процесса или прекращает выполнение процесса.) Параметры: m1_i1: PR_ENDPT (Номер процесса, приоритет которого изменяется.) m1_i2: PR_PRIORITY (Новый приоритет.)	FW
3	**/kernel/system/do_privctl.c**	Реализуется вызов микроядра: m_type: SYS_PRIVCTL (Обновляет привилегии процесса. Если процесс пока не является системным процессом, выделяет(?) ему его собственную структуру привилегий.) Параметры: m2_i1: CTL_ENDPT (Точка окончания (endpoint (?)) целевого процесса.) m2_i2: CTL_REQUEST (Запрос контроля привилегий.) m2_p1: CTL_ARG_PTR (Указатель на запрашиваемые данные.)	FW
3	**/kernel/system/do_profbuf.c**	Реализуется вызов микроядра: m_type: SYS_PROFBUF (При помощи данного вызова микроядра профилируемые процессы	FW

pr.	*path*	*an.*	*aut*
		информируют микроядро о местоположении их таблицы профилирования и контрольной структуры. Вызов микроядра используется системой профилирования когда установлена опция профилирования вызовов.) Параметры: m7_p1: PROF_CTL_PTR (Местоположение контрольной структуры.) m7_p2: PROF_MEM_PTR (Местоположение таблицы профилирования.)	
3	**/kernel/system/do_runctl.c**	Реализуется вызов микроядра: m_type: SYS_RUNCTL (Контролирует флаги PROC_STOP процесса. Используется для управления процессами. В некоторых случаях устанавливает MF_SIG_DELAY вместо PROC_STOP. Используется сервером управления процессами (PM) для надёжности управления сигналами.) Параметры: m1_i1: RC_ENDPT (Номер контролируемого процесса.) m1_i2: RC_ACTION (Останавливает или восстанавливает исполнение процесса.) m1_i3: RC_FLAGS (Флаги запроса.)	FW
3	**/kernel/system/do_safecopy.c**	Реализуется вызовы микроядра: m_type: SYS_SAFECOPYFROM or SYS_SAFECOPYTO or SYS_VSAFECOPY (Безопасное копирование. Копирование областей памяти с контролем разрешений.) Параметры: SCP_FROM_TO (другая точка окончания (endpoint (?))) SCP_INFO (закодирован: находящийся в собственности вызывающего процесса сегмень из/в который происходит копирование.) SCP_GID (Идентификатор разрешения (grant id) (?).) SCP_OFFSET (Смещение внутри разрешённой области.) SCP_ADDRESS (Адрес в собственном адресном пространстве.) SCP_BYTES (Размер копируемой области в байтах.) (Для векторизированного варианта (do_vsafecopy)): VSCP_VEC_ADDR (Адрес вектора.) VSCP_VEC_SIZE (Число значимых элементов в векторе – размер копируемой области в элементах (?).)	FW
3	**/kernel/system/do_segctl.c**	Реализуется вызов микроядра: m_type: SYS_SEGCTL (Возвращает переключатель сегмента и смещение, которые могут быть использованы для достижения физических адресов, для использования в драйверах выполняющих отображённый на память ввод/вывод в области A0000 – DFFFF.) Параметры: m4_l3: SEG_PHYS (Базовый физический адрес.) m4_l4: SEG_SIZE (Размер сегмента.) m4_l1: SEG_SELECT (Здесь возвращается переключатель сегмента.) m4_l2: SEG_OFFSET (Здесь возвращается смещение внутри сегмента.) m4_l5: SEG_INDEX (Возвращает индекс опосредованной(?)	FW

pr.	*path*	*an.*	*aut*
		(remote) памяти.)	
3	**/kernel/system/do_setalarm.c**	Реализуется вызов микроядра: m_type: SYS_SETALARM (Выполняет запрос на синхронный сигнал(?) (synchronous alarm), или на отмену синхронного сигнала.) m2_l1: ALRM_EXP_TIME (Время (до (?)) подачи сигнала.) m2_i2: ALRM_ABS_TIME (Время (до (?)) подачи сигнала абсолютное?) m2_l1: ALRM_TIME_LEFT (Возвращает секунды, прошедшие от предыдущего (?).)	FW
3	**/kernel/system/do_setgrant.c**	Реализуется вызов микроядра: m_type: SYS_SETGRANT (Устанавливает разрешения.) Параметры: SG_ADDR (Адрес таблицы разрешений (?) (grant table) в собственном адресном пространстве.) SG_SIZE (Число записей(?) (number of entries))	FW
3	**/kernel/system/do_segreturn.c**	Реализуется вызов микроядра: m_type: SYS_SIGRETURN (Запрос в стиле сигналов POSIX требует , чтобы sys_sigreturn упорядочил всё прежде, чем сигнализирующий процесс мог снова выполняться.) Параметры: m2_i1: SIG_ENDPT (Процесс, возвращающийся из (?)(handler).) m2_p1: SIG_CTXT_PTR (Указатель на структуру контекстов сигналов(?) (sigcontext structure).)	FW
3	**/kernel/system/do_sigsend.c**	Реализуется вызов микроядра: m_type: SYS_SIGSEND (Обеспечение сигналов в стиле POSIX.) Параметры: m2_i1: SIG_ENDPT (Процесс для вызова обеспечения сигнала (?) (signal handler) .) m2_p1: SIG_CTXT_PTR (Указатель на структуру контекста сигналов (sigcontext).) m2_i3: SIG_FLAGS (Флаги для вызова S_SIGRETURN.)	FW
3	**/kernel/system/do_sprofile.c**	Реализуется вызов микроядра: m_type: SYS_SPROFILE (Обеспечивает статистическое профилирование.) Параметры: m7_i1: PROF_ACTION (Начинает/прекращает профилирование.) m7_i2: PROF_MEM_SIZE (Доступная память для данных.) m7_i3: PROF_FREQ (Частота запрашиваемого образца(?)(sample).) m7_i4: PROF_ENDPT (Конечная точка (?) (endpoint) запрашивающего.) m7_p1: PROF_CTL_PTR (Местоположение информационной структуры.) m7_p2: PROF_MEM_PTR (Местоположение памяти для данных.)	FW
3	**/kernel/system/do_stime.c**	Реализуется вызов микроядра: m_type: SYS_STIME (Системное время(?).)	FW

pr.	*path*	*an.*	*aut*
		Параметры: m2_l1: T_BOOTITME (Время с момента загрузки системы (?).)	
3	**/kernel/system/do_sysctl.c**	Реализуется вызов микроядра: m_type: SYS_SYSCTL (?) Параметры: SYSCTL_CODE (Запрос.) SYSCTL_ARG1 SYSCTL_ARG2 (Специфические для запроса аргументы.)	FW
3	**/kernel/system/do_times.c**	Реализуется вызов микроядра: m_type: SYS_TIMES (Устанавливает информацию о времени (?)(times) в сообщение. Прерывание часов (таймера(?)) может обновить системное время, что не мешает данному коду.) Параметры: m4_l1: T_ENDPT (Получает информацию для данного процесса.) m4_l1: T_USER_TIME m4_l2: T_SYSTEM_TIME m4_l3: T_BOOTTIME m4_l5: T_BOOT_TICKS	FW
3	**/kernel/system/do_trace.c**	Реализуется вызов микроядра: m_type: SYS_TRACE (Обеспечивает отладочную трассировку.) Параметры: m2_i1: CTL_ENDPT (Трассируемый процесс.) m2_i2: CTL_REQUEST (Запрос трассирования.) m2_l1: CTL_ADDRESS (Адрес в пространстве трассируемого процесса.) m2_l2: CTL_DATA (Данные, которые должны быть записаны, или место для возвращаемых данных.)	FW
3	**/kernel/system/do_umap.c**	Реализуется вызов микроядра: m_type: SYS_UMAP (Создаёт карту отображения виртуальных адресов на физические – для процессов кроме микроядра. (?)) Параметры: m5_i1: CP_SRC_PROC_NR (Номер процесса.) m5_c1: CP_SRC_SPACE (Сегмент, где находится адрес: T (код), D (данные), или S (стек).) m5_l1: CP_SRC_ADDR (Виртуальный адрес.) m5_l2: CP_DST_ADDR (Возвращает физический адрес.) m5_l3: CP_NR_BYTES (Размер структуры данных.)	FW
3	**/kernel/system/do_unused.c**	Этот файл обеспечивает перехват неиспользуемых вызовов микроядра. Вызов микроядра может быть неиспользованным когда он не определён или отключен в конфигурации микроядра.	FW
3	**/kernel/system/do_vcopy.c**	Реализуется вызовы микроядра: m_type: SYS_VIRVCOPY, SYS_PHYSVCOPY (Копирования физической/виртуальной памяти.) Параметры: m1_i3: VCP_VEC_SIZE (Размер вектора, запрашиваемого на копирование.) m1_p1: VCP_VEC_ADDR (Адрес вектора (в адресном пространстве запрашивающего процесса(?)).)	FW

pr.	*path*	*an.*	*aut*
		m1_i2: VCP_NR_OK (Число успешных копий (?) (А может быть скопированных байтов, или элементов вектора?).)	
3	**/kernel/system/do_vdevio.c**	Реализуется вызов микроядра: m_type: SYS_VDEVIO (Выполняет серию операций с устройствами ввода/вывода от имени процесса (а не задания микроядра). Адреса ввода/вывода и значения ввода/вывода получаются или возвращаются в некоторый буфер в адресном пространстве процесса. Реальный ввод/вывод обрамлен lock() и unlock(), во избежание прерываний. Является вызовом, радственным do_devio, выполняющим одиночную операцию с устройством ввода/вывода.) Параметры: m2_i3: DIO_REQUEST (Запрос на ввод или вывод.) m2_p1: DIO_VEC_ADDR (Указатель на пару порт/значение.) m2_i2: DIO_VEC_SIZE (Число портов для ввода/вывода.)	FW
3	**/kernel/system/do_vmctl.c**	Реализуется вызов микроядра: m_type: SYS_VMCTL (Обеспечивает нужды виртуальной памяти (?).) Параметры: SVMCTL_WHO (Какой процесс?) SVMCTL_PARAM (Устанавливает значение (VMCTL_*) ....) SVMCTL_VALUE (равным этому значению.)	FW
3	**/kernel/system/do_vtimer.c**	Реализуется вызов микроядра: m_type: SYS_VTIMER Параметры: m2_i1: VT_WHICH (Таймер: VT_VIRTUAL или VT_PROF.) m2_i2: VT_SET (Установить или просто получить?) m2_l1: VT_VALUE (Новое/старое время достижения (?)(expiration time) в тиках.) m2_l2: VT_ENDPT (Процесс, которому принадлежит таймер.)	FW
2	**/ kernel/arch/i386/include/archconst.h**	Содержит константы для защищённого режима процессора i386.	FW
2	**/ kernel/arch/i386/include/archtypes.h**	Определяет типы и структуры для регистров процессора, сегментного дескриптора защищённого режима процессора, страничных исключений ...	FW
3	**/ kernel/arch/i386/arch_do_vmctl.c**	Реализуется вызов микроядра: m_type: SYS_VMCTL (Обеспечивает нужды виртуальной памяти (?). Архитектурно зависимая часть.) Параметры: SVMCTL_WHO (Какой процесс?) SVMCTL_PARAM (Устанавливает значение (VMCTL_*) ....) SVMCTL_VALUE (равным этому значению.)	FW
2	**/kernel/arch/i386/clock.h**	Прототипы функций инициализации, сброса и чтения значения счётчика системного таймера 8253A. Архитектурно зависимая часть. Отсутствует в стабильной версии 5.1.5. (!).	FW
3	**/kernel/arch/i386/clock.c**	Реализует функции: PUBLIC int init_8253A_timer(unsigned freq) Инициализирует канал 0 таймера 8253A устанавливая частоту 60	FW

pr.	*path*	*an.*	*aut*
		гц, регистрирует обработчик прерываний задания CLOCK для выполнения каждый тик. PUBLIC void stop_8253A_timer(void) Сбрасывает частоту таймера на значение BIOS (Для перезагрузки.) PUBLIC clock_t read_8253A_timer(void) Считывает счётчик по каналу 0 таймера 8253A. Счётчик отсчитывает в обратном порядке с частотой TIMER_FREQ и возвращается в значение TIMER_COUNT-1 , когда достигает нулевого значения. Аппаратное прерывание (тик) возникает в момент, когда счётчик достигает нулевого значения и возобновляет свой цикл. Находится в состоянии видоизменения в данный момент (!).	
3	**/kernel/arch/i386/do_int86.c**	Реализуется вызов микроядра: m_type: SYS_INT86 (.) (?) Параметры: m1_p1: INT86_REG86	FW
3	**/kernel/arch/i386/do_iopenable.c**	Реализуется вызов микроядра: m_type: SYS_IOPENABLE (.) (?) Параметры: m2_i2: IO_ENDPT (Процесс, которому устанавливаются биты уровня защиты ввода/вывода.)	FW
3	**/kernel/arch/i386/do_readbios.c**	Реализуется вызов микроядра: m_type: SYS_READBIOS (Получает данные BIOS.) (?) Параметры: m2_i1: RDB_SIZE (Число байт, которые надо скопировать.) m2_l1: RDB_ADDR (Абсолютный адрес в зоне BIOS.) m2_p1: RDB_BUF (Адрес буфера в запрашивающем процессе.)	FW
3	**/kernel/arch/i386/do_sdevio.c**	Реализуется вызов микроядра: m_type: SYS_SDEVIO (Доступ к портам ввоа/вывода.) (?) Параметры: m2_i3: DIO_REQUEST (Запрос на вывод или ввод.) m2_l1: DIO_PORT (Порт для чтения/записи.) m2_p1: DIO_VEC_ADDR (Виртуальный адрес буфера или ID разрешения (?) (grant ID).) m2_l2: DIO_VEC_SIZE (Число элементов.) m2_i2: DIO_VEC_PROC (Процесс, в котором находится буфер.) m2_i1: DIO_OFFSET (Смещение в разрешении (?) (grant).)	FW
2	**/kernel/arch/i386/exception.c**	Этот файл содержит простой обработчик исключений. Исключения в пользовательских процессах преобразуются в сигналы. Исключения в заданиях микроядра вызывают панику (и прерывание работы Minix3 (?)).	FW

2	**/kernel/arch/i386/hw_intr.h**	(?)
		/* legacy PIC */
		_PROTOTYPE(int irq_8259_unmask,(int irq));
		_PROTOTYPE(int irq_8259_mask,(int irq));
		_PROTOTYPE(void irq_handle,(int irq));
			FW

pr.	*path*	*an.*	*aut*
		#define hw_intr_mask(irq) irq_8259_mask(irq) #define hw_intr_unmask(irq) irq_8259_unmask(irq)	
3	**/kernel/arch/i386/i8259.c**	Этот файл содержит процедуры для инициализации контроллера прерываний 8259: put_irq_handler: регистрирует обработчик прерываний, rm_irq_handler: удаляет обработчик прерываний из регистра, intr_handle: обрабатывает аппаратное прерывание, intr_init: осуществляет инициализацию контроллера (контроллеров) прерываний. (Несмотря на начальный коментарий, что-то я не вижу put_irq_handler, rm_irq_handler, ntr_handle! Насколько я помню – эти функции находятся в аппаратно независимой части ... (?))	FW
2	**/kernel/arch/i386/klib386.S**	Этот файл содержит ассемблерный код процедур, необходимых для микроядра: void monitor() /* выйти из Minix и возвратиться в монитор*/ void int86() /* позволить монитору выполнить прерывание 8086 */ void exit() _exit __exit /* заглушки для библиотечных процедур */ __main /* заглушка для GCC */ void phys_insw(Port_t port, phys_bytes buf, size_t count) /* перемещает данные с (контроллера диска) порта в память */ void phys_insb(Port_t port, phys_bytes buf, size_t count) /* тоже – по байтам */ void phys_outsw(Port_t port, phys_bytes buf, size_t count) /* перемещает данные из памяти в (контроллер диска) порт */ void phys_outsb(Port_t port, phys_bytes buf, size_t count) /* тоже – по байтам */ phys_bytes phys_copy(phys_bytes source, phys_bytes destination, phys_bytes bytecount) /* копирует данные из любой области памяти в любую область памяти */ phys_copy_fault /*точка входа: страничное исключение phys_copy*/ phys_copy_fault_in_kernel /* точка входа: страничное исключение phys_copy в ядре*/ phys_memset(phys_bytes source, unsigned long pattern, phys_bytes bytecount) /* пишет (байт(?)) образец в любую область памяти*/ u16_t mem_rdw(U16_t segment, u16_t *offset) /* копирует одно слово с [segment:offset] */ void reset() /* reset the system – системный сброс*/ idle_task /* точка входа: задание выполняемое когда нет работы */ void level0(void (*func)(void)) /* вызвать функцию на уровне 0 */ unsigned long read_cpu_flags(void) /* прочесть флаги процессора */ unsigned long read_cr0(void) /* read cr0 */ reg_t read_cr2(void) /* read cr2 */ unsigned long getcr3val(void) void write_cr0(unsigned long value) /* write a value in cr0 */ unsigned long read_cr4(void) thecr3 (?) void write_cr4(unsigned long value) catch_pagefaults (?) read_ds (?) read_cs (?) read_ss (?)	FW

pr.	*path*	*an.*	*aut*
		Эти процедуры гарантирую сохранение только тех регистров, сохранение которых требует компилятор C: (ebx, esi, edi, ebp, esp, segment registers, and direction bit in the flags)	
3	**/kernel/arch/i386/memory.c**	Внутренние функции виртуальной памяти (VM). (?)	FW
3	**/kernel/arch/i386/mpx386.S**	Файл mpx386.s включается mpx.s когда Minix компилируется для 32-х битного процессора Intel. Альтернативный mpx88.s включается для 16-битного процессора. Этот файл является наиболее низкоуровневой частью Minix (наряду с /kernel/proc.c). Здесь осуществляется переключение процессов и поддержка обработки сообщений. Кроме того здесь содержаться обработчики 32-х битных прерываний и ассемблерный стартовый код (обеспечивает кооперацию с /kernel/start.c для обеспечения хорошего начального окружения для main()). Каждый переход в микроядро происходит через данный файл. Входы в микроядро могут быть вложенными. Начальный вход может быть вызван системным вызовом (вызовом микроядра (?) - при попытке послать или получить сообщение (?) (i.e., send or receive a message)), исключением или аппаратным прерыванием; вложенный (повторный) вход в микроядро может осуществляться только аппаратными прерываниями. Число вхождений в микроядра (степень вложенности) хранится в переменной "k_reenter". Важными моментами являются: - переключение на стек ядра, - и защита кода передачи сообщений /kernel/proc.c . ... (?)	FW
3	**/kernel/arch/i386/protect.c**	Этот файл содержит код для инициализации защищённого режима процессора, инициализации дескрипторов сегментов кода и данных и для инициализации глобальных дескрипторов для локальных дескрипторов в таблице процессов.	FW
5	**/kernel/arch/i386/proto.h**	Содержит прототипы для: - обработчиков аппаратных прерываний, - обработчиков исключений, - обработчиков программных прерываний, - прототипов функций из файлов **/kernel/arch/i386/memory.c** **/kernel/arch/i386/exception.c** **/kernel/arch/i386/klib386.S** **/kernel/arch/i386/protect.c** - прототипов для работы с таблицей векторов прерываний.	
5	**/kernel/arch/i386/sconst.h**	В данном файле содержаться различные константы, используемые в ассемблерном коде: - размер машинного слова, - смещения в struct proc (должен быть согласован с /kernel/proc.h), - смещение на указатель текущего процесса сразу после прерывания, в предположении, что мы всегда имеем код ошибки в стеке, - макросы (ассемблерного характера) связанные с сохранением контекста (в случае аппаратных прерываний, ...).	FW
5	**/kernel/arch/i386/system.c**	Системно зависимые функции для использования в микроядре в целом. (?)	FW
5	**/servers/vfs/const.h**	Содержит различные константы: - размеры таблиц (filp , file locking , mount , vnode);	FW

pr.	*path*	*an.*	*aut*
		- uid_t суперпользователя MM и INIT, uid_t для которого разрешена FSSIGNON, gid_t MM и INIT; - константы для определения блокировок ; - для аргументов: LOOK_UP, ENTER, DELETE, IS_EMPTY; - DUP_MASK, SYMLOOP, ROOT_INODE; - константы – аргументы для dev_io : VFS_DEV_(READ, WRITE, SCATTER, GATHER, IOCTL, SELECT); Макросы: fp_is_blocked(fp)	
3	**/servers/vfs/device.c**	Когда необходимый блок не находится в кэше, он должен быть получен из диска. Специальные символьные (текстовые(?) character) файлы также нуждаются в вводе/выводе. В данном файле находятся необходимые для этого процедуры. Точки входа данного файла: - для операций с устройствами (dev_open, dev_close, dev_io, dev_statu); - общие операции (gen_opcl, gen_io); - операции для несуществующих устройств (?): (no_dev, no_dev_io); - для tty-устройств (tty_opcl, ctty_opcl, ctty_io,); - для системных вызовов (do_ioctl, do_setsid).	FW
5	**/servers/vfs/dmap.h**	Таблица устройств: <устройство> <-> <таблица драйверов>. Индексами таблицы являются главный номер устройства ((?) major device number). Таблица обеспечивает связь между главным номером устройства и процедурами, которые обеспечивают его функционирование. Таблица может быть дополнена (изменена) динамически. Содержит константы-флаги: DMAP_MUTABLE, DMAP_BUSY, DMAP_BABY.	FW
3	**/servers/vfs/dmap.c**	Этот файл содержит таблицу: <устройство> <-> <таблица драйверов>. В нём также содержаться некоторые процедуры для динамического добавления, удаления драйверов, а также изменения связей ((?) mappings). Интерес представляет начальная инициализация init_dmap[] , содержащая учтройства ((/dev/)mem, fd0, c0, tty00, tty, lp, ip, c1, c2, c3, audio, klog, random).	FW
6	**/servers/vfs/exec.c**	Этот файл обеспечивает системный вызов EXEC (замены контекста процесса). Порядок работы: - проверяет разрешение файла на исполнение, - читает заголовок и определяет размеры, - получает начальные аргументы и переменные окружения из пространства пользователя, - выделяет память для нового процесса, - копирует начальный стек из сервера управления процессами (PM) в данный процесс, - читает сегменты кода и данных и копирует их в данный процесс, - устанавливает (заботится (?)(take care)) биты setuid, setgid, - исправляет (точнее – изменяет) таблицу 'mproc',	FW

pr.	*path*	*an.*	*aut*
		- сообщает микроядру о EXEC, - соханяет смещение на начальные аргументы (?)(for ps)/ Точка входа: pm_exec: выполняет системный вызов EXEC.	
3	**/servers/vfs/file.h**	Таблица ... (?)filp. Прослойка между дескрипторами файлов и ... (?) inodes. Слот свободен , если filp_count == 0.	FW
3	**/servers/vfs/filedes.c**	Содержит процедуры для манипуляций с файловыми дескрипторами. Точки входа: get_fd (ищет свободные fd и filp), get_filp (ищет запись filp для данного fd), find_filp (находит слот filp указывающий на данный ... vnode), inval_filp (делает непригодным flip и ассоциированный с ним fd, только таким образом позволяя выполнить процедуру close() на нём).	FW
6	**/servers/vfs/fproc.h**	Это информация для каждого процесса. Слот резервируется для каждого потенциального процесса. Именно поэтому константа NR_PROCS должна соответствовать конфигурации микроядра.	FW
6	**/servers/vfs/fs.h**	Это базовый заголовочный файл для файловой системы. Он включает несколько иных файлов и определяет принципиальные константы – константы режима компиляции (для библиотечных включаемых файлов).	FW
3	**/servers/vfs/fscall.c**	Этот файл обеспечивает вложенные контр-запросы ((?) counter-request) серверу виртуальной файловой системы (VFS), посылаемые файловыми серверами (FS: MFS, iso9660FS, ...) в ответ на запрос сервера виртуальной файловой системы (VFS). Точка входа: nested_fs_call.	FW
5	**/servers/vfs/glo.h**	В этом файле находятся определения глобальных переменных виртуальной файловой системы (реально выделяются в файле table.o данной подсистемы). Также содержит переменные в которых сохраняются параметры вызова, возвращаемого результата (код ошибки), а также объявлены некоторые данные, которые инициализуются в другом месте (_PROTOTYPE (int (*call_vec[]), (void)); /* sys call table */, char dot1[2]; /* dot1 (&dot1[0]) and dot2 (&dot2[0]) have a special */ char dot2[3]; /* meaning to search_dir: no access permission check. */)	FW
3	**/servers/vfs/link.c**	Этот файл обеспечивает системные вызовы LINK и UNLINK. Здесь также обеспечивается (объявление об(?)) освобождение памяти (ресурсов(?)), когда когда выполняется последний UNLINK и блоки должны быть возвращены в пул свободных блоков. Точки входа: do_link: выполняет системный вызов LINK; do_unlink: выполняет системные вызовы NLINK и RMDIR; do_rename: выполняет системный вызов RENAME; do_truncate: выполняет системный вызов TRUNCATE; do_ftruncate: выполняет системный вызов FTRUNCATE; do_rdlink: выполняет системный вызов RDLNK;	FW
3	**/servers/vfs/lock.h**	Таблица блокировок файлов. Также как и таблица ... (?) (filp), она указывает на таблицу (?) (inode), однако в данном случае для получения информации о блокировках.	FW

pr.	*path*	*an.*	*aut*
		Конкретно определяет только один массив: file_lock[NR_LOCKS];	
3	**/servers/vfs/lock.c**	Обеспечивает предусмотренную стандартом POSIX систему блокировки файлов. Точки входа: lock_op: осуществляет опереции с блокировками для системного вызова FCNTL; lock_revive: просматривает процессы после того, как снята блокировка.	FW
4	**/servers/vfs/main.c**	Стандартный файл для всех серверов и драйверов, в котором находится бесконечный цикл, получающий сообщения о запросах на выполнения операций, обрабатывающий эти запросы соответствующим образом, а также возвращающим необходимые ответы. Точки входа: main: основная функция сервера виртуальной памяти (VFS); reply: посылает ответ (реплику) процессу после выполнения запрошенной работы.	FW
6	**/servers/vfs/misc.c**	Этот файл содержит набор различных процедур. Некоторые из них выполняют простейшие системные вызовы. Другие выполняют небольшую часть работы. Связанной с системными вызовами, основную часть которых выполняет сервер управления памятью (MM и VM). Точки входа: do_dup: выполняет системный вызов DUP do_fcntl: выполняет системный вызов FCNTL do_sync: выполняет системный вызов SYNC do_fsync: выполняет системный вызов FSYNC do_reboot: обновляет диск с буферов в рамках подготовки к завершению работы системы do_fork: выполняет записи в таблицах VFS после выполнения системного вызова FORK do_exec: обеспечивает файлы с установленным флагом FD_CLOEXEC после выполнения системного вызова EXEC do_exit: выполняет записи в такблицах, связанные с завершением процесса do_set: устанавливает uid или gid для некоторого процесса do_revive: пересматривает процессы (с целью некоторых действий), которые ожидают события в файловой системе (вроде TTY) do_svrctl: контроль файловой системы do_getsysinfo: выдаёт копию структур данных файловой системы pm_dumpcore: выполняет дамп памяти	FW
6	**/servers/vfs/mmap.c**	Обеспечение функционала mmap в VFS Точка входа: do_vm_mmap: сервер виртуальной памяти вызывает VM_VFS_MMAP	FW
3	**/servers/vfs/mount.c**	Этот файл обеспечивает системные вызовы MOUNT и UMOUNT. Точки входа:do_mount, do_umount	FW
3	**/servers/vfs/open.c**	Этот файл содержит процедуры для создания, открытия, закрытия и изменения позиции «курсора текущей позиции» файла.	FW

pr.	*path*	*an.*	*aut*
		Точки входа: do_creat, do_open, do_mknod, do_mkdir, do_close, do_lseek.	
5	**/servers/vfs/param.h**	Устанавливает внутренние синонимы с полями в структурах входящих сообщений, а также синонимы в полях исходящих сообщений.	FW
3	**/servers/vfs/path.c**	Lookup() - это основная процедура , контролирующая нахождение имени пути (файла, папки или другого файлового объекта). Она поддерживает точки монтирования и символические ссылки. Настоящий запрос на поиск посылается через функцию-обёртку req_lookup ...	FW
3	**/servers/vfs/pipe.c**	Эта функция обеспечивает приостановку и возобновление исполнения процесса (связанные с опреациями с файловым объектом pipe). ... (Стандартное описание функционирования) Точки входа: do_pipe: pipe_check: suspend: release: revive: unsuspend_by_endpt: do_unpause:	FW
3	**/servers/vfs/protect.c**	Этот файл содержит точки входа, обеспечивающие системные вызовы, связанные с разрешениями использования файла: do_chmod: CHMOD and FCHMOD do_chown: CHOWN and FCHOWN do_umask: UMASK do_access: ACCESS	FW
5	**/servers/vfs/proto.h**	Содержит все прототипы функций подсистемы (с комментариями, указывающими на местонахождение этих фуенкций – файл подсистемы, в котором реализован данный набор функций).	FW
3	**/servers/vfs/read.c**	В этом файле находится сердце механизма, используемого при чиении (и записи) файла. Запросы на чтение и запись разделены (между собой) по чанкам (?) которые не пересекают границы блока (блока диска (?)). Каждый чанк обрабатывается в один ход (?). Чтение специальных файлов также определяется и поддерживается. Точки входа: do_read: выполняет системный вызов READ посредством вызова read_write; do_getdents: читает записи в каталоге (GETDENTS); read_write: как раз и выполняет всю работу, связанную с системными вызовами READ и WRITE.	FW
3	**/servers/vfs/request.h**	Низкоуровневые сообщения-запросы строяться и посылаются посредством функций-обёрток. Этот файл содержит описания структур запросов и ответов, используемых для доступа к этим функциям-обёркам.	FW

pr.	*path*	*an.*	*aut*
3	**/servers/vfs/request.c**	Этот файл содержит функции-обёртки для подачи запросов и получения ответов от процессов файловой системы (FS). Каждая функция строит сообщение запроса в соответствии с запрашиваемыми параметрами, вызывает большинство низкоуровневых fs_sendrec и копирует обратно конечный ответ – результат. Низкоуровневый fs_sendrec обеспечивает (также) механизм восстановления при умершем драйвере и повторяет запрос (не удавшийся в предыдущий раз).	FW
3	**/servers/vfs/select.h**	Содержит только макроопределения SEL_OK, EL_ERROR, SEL_DEFERRED .	FW
3	**/servers/vfs/select.c**	Содержит точки входа, связанные с системным вызовом SELECT: do_select: выполняет системный вызов SELECT; select_callback: сообщает системе SELECT о возможной операции с файловым дескриптором; select_notified: низкоуровневый вход для устройств сообщающих в контексте SELECT; select_unsuspend_by_endpt: отменяет блокировки при завершении процесса-драйвера.	FW
3	**/servers/vfs/stadir.c**	Этот файл содержит код для выполнения 4-х системных вызовов, связанных с состоянием (?)(чего?) и каткогами (файловой системы). Точки входа (однозначно соответствуют довольно стандартным системным вызовам UNIX): do_chdir: do_chroot: do_stat: do_fstat: do_fstatfs: do_lstat:	FW
5	**/servers/vfs/table.c**	Этот файл содержит таблицы, позволяющие отображать номера системных вызовов на процедуры, их обеспечивающие. И больше ничего ...	FW
3	**/servers/vfs/time.c**	Этот файл заботится о системных вызовах, которые связаны со временем (создания, доступа, изменения файловых объектов). Точки входа: do_utime: выполняет системный вызов UTIME; do_stime: сервер управления процессами (PM) информирует файловую систему (FS) о системном вызове STIME.	FW

6	**/servers/vfs/timers.c**	Библиотека таймеров файловой системы (FS).
		
		void fs_set_timer(timer_t *tp, int ticks, tmr_func_t watchdog, int arg);
		void fs_expire_timers(clock_t now);
		void fs_cancel_timer(timer_t *tp);
			FW

5	**/servers/vfs/type.h**	Содержит описание extern struct dmap ... и больше ничего ...	FW
5	**/servers/vfs/utility.c**	Этот файл содержит некоторые процедуры-утилиты общего назначеия. Точки входа: clock_time: запрашивает процесс CLOCK о реальном времени, copy: копирует блок данных, fetch_name: «идёт» получать имя пути из пространства пользователя (связано с сокращениями для домашнего каталога, ... (?)); no_sys: отвергает системный вызов, который файловая система не обеспечивает;	FW

pr.	*path*	*an.*	*aut*
		panic: выполняется тогда, когда происходит событие, исключающее возможность дальнейшего исполнения Minix; conv2: выполняет изменение порядка следования байтов 16-битного int; (?) - точно? conv4: выполняет изменение порядка следования байтов 32-битного long; (?) - точно?	
3	**/servers/vfs/vmnt.h**	Описывает структуру EXTERN struct vmnt и макроопределение #define NIL_VMNT (struct vmnt *) 0 и это всё ...	FW

3	**/servers/vfs/vmnt.c**	Содержит процедуры , связанные с виртуальной таблицей монтирования.
		
		struct vmnt *get_free_vmnt(short *index);
		struct vmnt *find_vmnt(int fs_e);
			FW

3	**/servers/vfs/vnode.h**	Описывает структуру EXTERN struct vnode define NIL_VNODE (struct vnode *) 0 /* indicates absence of vnode slot */ /* Field values. */ #define NO_PIPE 0 /* i_pipe is NO_PIPE if inode is not a pipe */ #define I_PIPE 1 /* i_pipe is I_PIPE if inode is a pipe */	FW
3	**/servers/vfs/vnode.c**	Этот файл содержит точки входа: get_vnode - get_free_vnode - find_vnode - dup_vnode - put_vnode -	FW
3	**/servers/vfs/write.c**	Этот файл является напарником «read.c» данной подсистемы. В нём содержится код для выполнения записи в файл (и файловый объект), который не является частью read_write(). Точка входа: do_write: вызывает read_write для выполнения системного вызова WRITE.	FW
3	**/servers/mfs/buf.h**	Кэш блоков (буфера (?)). Для получения блока вызывается get_block(), с указанием нужного блока. Блок помечается как «in use» («в пользовании») и его поел «b_count» увеличивается на единицу. Все блоки, которые не находятся в пользовании помещены в список LRU, на вершине которого редко используемые блоки, а в конце - часто используемые блоки. Список является двусвязным, обратная явязь обеспечивается полем «b_rev». LRU видоизменяется при вызове «put_block()». Второй параметр этой функции может управлять порядком помещения блока в список LRU, например, поместить этот блок на вершину списка, если он скорее всего в ближайшее время не понадобится. Если блок изменён, то функция, его изменяющая, устанавливает поле «b_dirt» в состояние «DIRTY», что в дальнейшем обеспечивает его перезапись на диск.	FW

pr.	*path*	*an.*	*aut*
3	**/servers/mfs/cache.c**	Кэш файловой системы уменьшает число необходимых обращений к диску. При любой операции read/write сначала проверяется наличие блока в кэше. В данном файле находится код, обеспечивающий управления кэшем. Точки входа: get_block: put_block: alloc_zone: free_zone: invalidate: Внутренние функции: rw_block: осуществляет непосредственное чтение или запись блока с диска.	FW
5	**/servers/mfs/const.h**	Размеры таблиц. Макрос usizeof(t) . Описатель (под-)типов файловой системы Minix. (MFS v1 , v2, v3). ...	FW
3	**/servers/mfs/device.c**	Содержит функции: PUBLIC int fs_clone_opcl(void) возвращает номер нового устройства и создаёт временный файл, для связи с этим устройством. PUBLIC int fs_new_driver(void) Новый endpoint (?) для этого устройства. PUBLIC int block_dev_io(op, dev, proc_e, buf, pos, bytes, flags) int op; /* MFS_DEV_READ, MFS_DEV_WRITE, etc. */ dev_t dev; /* major-minor device number */ int proc_e; /* in whose address space is buf? */ void *buf; /* virtual address of the buffer */ u64_t pos; /* byte position */ int bytes; /* how many bytes to transfer */ int flags; /* special flags, like O_NONBLOCK */ PUBLIC int dev_open(driver_e, dev, proc, flags) endpoint_t driver_e; dev_t dev; /* device to open */ int proc; /* process to open for */ int flags; /* mode bits and flags */ /* Determine the major device number call the device class specific * open/close routine. (This is the only routine that must check the * device number for being in range. All others can trust this check.) */ PUBLIC void dev_close(driver_e, dev) endpoint_t driver_e; dev_t dev; /* device to close */	FW
2	**/servers/mfs/drivers.h**	Endpoint – ы основных устройств (?). Только блочные устройства отражены здесь – это подмножество отображения VFS (сервера виртуальной файловой системы).	FW
4	**/servers/mfs/fs.h**	Это основной заголовочный файл файловой системы. Он подключает некоторые другие файлы и определяет самые основные константы.	FW
4	**/servers/mfs/glo.h**	Содержит объявления глобальных переменных. Реально эти переменные помещаются в table.o .	FW

pr.	*path*	*an.*	*aut*
4	**/servers/mfs/inc.h**	Содержит массу заголовочных файлов из /include .	FW
3	**/servers/mfs/inode.h**	Таблица inode (?). Таблица содержит используемые inodes. В опередёлённом смысле они могут быть открыты посредством системных вызовов open() , creat(), в отдельных случаях файловая система сама по себе нуждается в inodes для определённых целей, таких как поиск пути по каталогам. Первая часть структуры содержит поля , которые присутствуют на диске; вторая – которые не присутствуют на диске. Дисковая часть inode также опередена в «type.h» как «d1_inode» для файловой системы V1, или «d2_inode» для V2.	FW
3	**/servers/mfs/inode.c**	Этот файл содержит процедуры для управления таблицей (используемых) inode (?). В нём имеются процедуры для ...	FW
3	**/servers/mfs/link.c**	Содержит определения функций: ***int fs_link_o()*** : выполняет системный вызов link(name1, name2) ***int fs_link_s()*** : выполняет системный вызов link(name1, name2) (версия для суперпользователя (?)) ***int fs_unlink_o():*** выполняет системные вызовы unlink(name) или rmdir(name) ***int fs_unlink_s():*** выполняет системные вызовы unlink(name) или rmdir(name) (версия для суперпользователя (?)) а также функции публичного интерфейса: ***int fs_rdlink_o(); int fs_rdlink_s(); int fs_rdlink_so(); int fs_rename_o(); int fs_rename_s(); int fs_trunc(); int fs_ftrunc(void); int truncate_inode(rip, newsize); int freesp_inode(rip, start, end);***	FW
4	**/servers/mfs/main.c**	***int main(int argc, char *argv[])***: Основная функция сервера, содержит основной цикл. ***void init_server(void)***: производит инициализацию таблицы indode (?) ***void get_work(m_in); void reply(who, m_out); void cch_check(void);***	FW
3	**/servers/mfs/misc.c**	Содержит функции публичного интерфейса: ***int fs_sync()***: выполняет системный вызов sync(). Приводит в соответствие (?) все таблицы. Порядок привеления важен. Блоки должны приводится в соответствие в последнюю очередь, так как ***rw_inode()*** оставляет свои результаты в кэше блоков. ***int fs_flush()***: Приводит в соответствие (?) блоки или устройство из кэша после записи какого-либо «грязного» блока на диск.	FW
3	**/servers/mfs/mount.c**	Содержит функции публичного интерфейса: ***int fs_readsuper_s()***: читает суперблок партиции, получает его базовый inode (?) и возвращает детали его. Следует иметь в виду, что процесс FS не знает индекс объекта vmnt, указывающий на него (базовый inode партиции(?)), когда поиск имени пути покидает партицию, ошибка ELEAVEMOUNT пересылается (серверу виртуальной файловой системы), чтобы сервер VFS знал, что ему необходимо найти vnode (?) на который партиция этого процесса FS смонтирована.	FW
3	**/servers/mfs/open.c**	Содержит функции публичного интерфейса: int fs_open(); int fs_create_o(); int fs_create_s(); int fs_mknod_o(); int fs_mknod_s(); int fs_mkdir_o(); int fs_mkdir_s(); int fs_slink_o(); int fs_slink_s(); int fs_newnode(); int fs_inhibread();	FW
3	**/servers/mfs/path.c**	Этот файл содержит процедуры, выполняющие поиск имени пути в системе каталогов и определяет номер inode (?) , соответствующий данному имени пути. Точки входа: eat_path: основная функция механизма преобразования пути в inode	FW

pr.	*path*	*an.*	*aut*
		(?) last_dir: находит последний каталог в данном пути advance: обрабатывает одну компоненту пути search_dir: ищет каталог по пути и возвращает номер соответствующего inode (?)	
3	**/servers/mfs/pipe.c**	Содержит только одну функцию: int fs_pipe(void).	FW

3	**/servers/mfs/protect.c**	Содержит функции:
		int fs_chmod();
		Выполняет системный вызов chmod(name, mode).
		int fs_chown();
		int fs_access_o();
		int forbidden(register struct inode *rip, mode_t access_desired);
		Определяет разрешён ли данного типа доступ к inode (?). Если нет -определяет причину. Функция ищет uid вызывающего в таблице fproc. Если доступ разрешён – возвращает OK, если нет – EACCES.
		int read_only(ip);
		Если inode (?) на который указывает ip расположен в файловой системе, смонтированной с доступом только на чтение, то возвращает EROFS, если нет – то OK.
			FW

5	**/servers/mfs/proto.c**	Содержит все прототипы функций (только публичного интерфейса (?)) подсистемы, предварённые используемыми в прототипах структурами.	FW

3	**/servers/mfs/read.c**	Содержит функции публичного интерфейса:
		int fs_readwrite_o(void);
		int fs_readwrite_s(void);
		int fs_breadwrite_o(void);
		int fs_breadwrite_s(void);
		block_t read_map(rip, position);
		По данным inode (?) и позиции в соответствующем файле, находит номер блока (не зоны!), в котором эта «позиция» находится.
		zone_t rd_indir(bp, index);
		Читает одну запись по данному «косвенному» блоку (indirect block) (?). Смысл в выделении отдельной функции для данной операции: V1 (IBM and 68000), and V2 (IBM and 68000).
		void read_ahead();
		Считывает блок в кэш (до того, как он понадобится (?)).
		struct buf *rahead(rip, baseblock, position, bytes_ahead);
		Считывает блок из кэша или устройства. Если требуется чтение из физического устройства, то считывает оперделённое соглашением (конфигурацией(?)) число блоков в кэш (обычно в зависимости от bytes_ahead и по крайней мере BLOCKS_MINIMUM).
		Драйвер устройства может решить, что ему лучше известно, и остановить чтение на границе цилиндра (или после ошибки). На подобное поведение влияет Rw_scattered() - устанавливая соответствующий флаг.
		int fs_getdents(void);
		int fs_getdents_o(void);
		Поддержка совместимости: обёртка getdents, возвращающая результирующее число байтов в поле m_type ответного сообщения reply message (?).
			FW

3	**/servers/mfs/stadir.c**	Содержит функции:
		int stat_inode(rip, pipe_pos, who_e, gid);
		Общий код для системных вызовов stat и fstat.
		int fs_fstatfs();
		int fs_stat();
			FW

pr.	*path*	*an.*	*aut*
2	**/servers/mfs/super.h**	Таблица суперблоков (содержит записи для базовой (root (?)) файловой системы и точек монтирования). Запись содержит информацию о размерах битовых карт (bit maps (?)) и inode (?). Поле s_ninodes даёт число inode (?), доступных для файлов и каталогов, включая корневой каталог. 0-вой inode (?) есть на диске но не используется ... Схема диска: * Item # blocks * boot block 1 * super block 1 (offset 1kB) * inode map s_imap_blocks * zone map s_zmap_blocks * inodes (s_ninodes + 'inodes per block' - 1)/'inodes per block' * unused whatever is needed to fill out the current zone * data zones (s_zones - s_firstdatazone) << s_log_zone_size * * A super_block slot is free if s_dev == NO_DEV.	FW
2	**/servers/mfs/super.c**	Этот файл содержит код, управляющий таблицей суперблоков и структурами, связанными с таблицей суперблоков. Точки входа: alloc_bit: если требуется выделить (свободную) зону или inode (?); находит её free_bit: указывает, что зона или inode (?) доступен для выделения (свободен); get_super: ищет таблицу суперблоков для устройства; mounted: сообщает если indoe (?) файла находится на смонтированной (или корневой) файловой системе; read_super: читает суперблое;	FW
6	**/servers/mfs/table.c**	Этот файл содержит таблицу, используемую для отображения системных вызовов на соответствующие процедуры, которые выполняют их.	FW
3	**/servers/mfs/time.c**	Содержит функцию PUBLIC int fs_utime()	FW
2	**/servers/mfs/type.h**	d1_inode ; d2_inode; Содержит определения i-узла ((?) inode) версии 1 и 2, так как он расположен на диске (не в памяти). Также определена структура struct buf;	FW
3	**/servers/mfs/utility.c**	Содержит определения процедур: PUBLIC **int no_sys()**; /* кто-то использовал неправильный номер системного вызова*/ PUBLIC **unsigned conv2(norm, w)**; int norm; /* TRUE если не менять, FALSE для обмена байтов */ int w; /* 16-и битное слово, аргумент для преобразования*/ PUBLIC **long conv4(norm, x)**; int norm; /* TRUE если не менять, FALSE для обмена байтов */ long x; /* 32-bit long, аргумент для обмена */ - эта функция меняет порядок байтов между 8086 и 68000. **PUBLIC time_t clock_time()**; /* Эта процедура возвращает время в секундах с 1.1.1970. MINIX является астрофизически наивной системой, которая подразумевает, что земля крутится с постоянной скоростью и таких вещей, как прыгающие секунды ((?) leap seconds), не существует */	FW
3	**/servers/mfs/write.c**	Этот файл является напарником «read.c». В нём содержится код для записи (файла на диск) в той мере, в какой он не содержится в	FW

pr.	*path*	*an.*	*aut*
		read_write(). Точки входа в этот файл: **do_write**: вызывает read_write для выполнения системного вызова WRITE **clear_zone**: стирает зону посередине файла. **new_block**: получает новый блок.	
2	**/servers/iso9660fs/buf.h**	Определяет: переменную buf[NR_BUFS]; NIL_BUF , INODE_BLOCK 0 , DIRECTORY_BLOCK 1; А также некотрые определения, позволяющие использовать bp->b_data вместо bp->b.b__data.	FW
3	**/servers/iso9660fs/cache.c**	Файловая система имеет кэш буферов для уменьшения числа обращений к диску. Всякий раз, когда выполняется чтение или запись на диск, сначала выполняется проверка на наличие блока в кэше. Точки входа: get_block: запрос на нахождение блока для чтени или записи из кэша. put_block: возвращает блок, предварительно запрошенный посредством get_block. Внутренние функции: read_block: читает блок физически.	FW
4	**/servers/iso9660fs/const.h**	В этом файле определены все константы, используемые сервером.	FW
3	**/servers/iso9660fs/device.c**	Этот файл обеспечивает непосредственную коммуникацию с устройством.	FW
2	**/servers/iso9660fs/drivers.h**	Конечные точки ((?) endpoints) драйверов основных устройств. Только блочные устройства отображаются здесь, является подмножеством отображения в сервере виртуальной файловой системе (VFS). Содержит только: EXTERN struct driver_endpoints { endpoint_t driver_e; } driver_endpoints[NR_DEVICES];	FW
5	**/servers/iso9660fs/glo.h**	Содержит глобальные переменные (в основном переменные для возврата результата запрашивающему процессу), которые реально выделяются в table.o .	FW
2	**/servers/iso9660fs/inc.h**	Содержит глобальные заголовочные файлы, предварённые настройками заголовков из «/include» , характерных для сервера.	FW
3	**/servers/iso9660fs/inode.h**	Определения переменных: dir_records[NR_DIR_RECORDS]; ext_attr_recs[NR_ATTR_RECS]; а также: #define D_DIRECTORY 0x2 #define D_TYPE 0x8E #define ID_DIR_RECORD(dir) dir->d_ino_nr	FW
3	**/servers/iso9660fs/inode.c**	Этот файл содержит все функции, которые поддерживают записи каталогов (i-узлов ((?) inodes)) для файловой системы ISO9660.	FW
5	**/servers/iso9660fs/main.c**	Этот файл содержит основную функцию сервера. Основная функция	FW

pr.	*path*	*an.*	*aut*
		ожидает запрос и (по получению) посылает ответ (на запрос). Содержаться также стандартные функции: **void init_server(void); void get_work(message *m_in);** а также функцию **void reply(who, m_out);**	
3	**/servers/iso9660fs/misc.c**	Содержит только функцию: int fs_sync(); (которая ничего не делает, так как iso9660fs используется только для чтения).	FW
3	**/servers/iso9660fs/mount.c**	Функции для монтирования и размонтирования ISO9660. Содержит функции: **int fs_readsuper(); int fs_readsuper_s();** Читает супер i-узел. Эта функция запрашивается в момент монтирования файловой системы. **int fs_mountpoint_s();** Эта функция ищет точку монтирования, также проверяет условия пригодности, когда партиция может быть смонтирована в i-узел (или не может). **int fs_unmount(void);**	FW
3	**/servers/iso9660fs/path.c**	Содержит функции: **int lookup();** - функция поиска пути (вызывается очень часто). **int fs_lookup_s();** **int search_dir(ldir_ptr,string,numb);** - функция выполняет операции поиска компоненты «string» в ldir_ptr. Она возвращает реплику (успешности операции (?)) и номер i-узла в numb. **struct dir_record *parse_path(path, string, action);** - функция обрабатывает путь и возвращает запись финального каталога. Конечная компонента пути будет возвращена в строке. Она работает двумя путями: PATH_PENULTIMATE – возвращает последний каталог, PATH_NON_SYMBOLIC – возвращает последнюю компоненту пути. **int parse_path_s(dir_ino, root_ino, flags, res_inop, offsetp);** - обрабатывает путь в user_path, начиная с dir_ino. (Описаны также особенности обработки пустого пути, а также путей , состоящих только из символов '/'). **struct dir_record *advance(pdirp, string);** - функция возвращает компоненту в «string» выполняя поиск в каталоге pdirp ... **int advance_s(dirp, string, resp);** - даны каталог и компонента пути; ищет компоненту в каталоге, находит i-узел, открывает его, и возвращает указатель на его слот i-узлов. **char *get_name(old_name, string);** - 'get_name' = 'old_name' – 'string'. /usr/ast, /usr//ast, //usr///ast и /usr/ast/ являются эквивалентными. **char *get_name_s(path_name, string);** - напарница предыдущей.	FW
3	**/servers/iso9660fs/protect.c**	Содержит функцию, используемые для определения прав доступа (работы с правами доступа): int fs_access(); iso9660fs не поддерживает контроля за правами доступа.	FW
5	**/servers/iso9660fs/proto.h**	Прототипы функций (публичного интерфейса) файловой системы iso9660 (распределённые по файлам).	FW
3	**/servers/iso9660fs/read.c**	Функции для чтения файла.	FW

pr.	*path*	*an.*	*aut*
3	**/servers/iso9660fs/stadir.c**	Содержит функции: **int stat_dir_record(dir, pipe_pos, who_e, gid);** - эта функция возвращает всю информацию о конкретном i-узле (она пропускает дату записи по причине бага в стандартной функции stdtime; как только баг будет исправлен, ...). **int fs_stat();** - эта функция-обёртка предыдущей (именно она и вызывается по запросу). **int fs_fstatfs();**	FW
3	**/servers/iso9660fs/super.h**	Этот файл содержит определения структур файлововй системы ISO9660.	FW
3	**/servers/iso9660fs/super.c**	Функции для управления суперблоком файловой системы. Эти функции вызываются в начале и в конце работы сервера. **int release_v_pri(v_pri);** **int create_v_pri(v_pri,buf,address);** **int read_vds(v_pri,dev);**	FW
6	**/servers/iso9660fs/table.c**	Этот файл содержит таблицу, используемую для отображения номеров системных вызовов на процедуры, их выполняющие: PUBLIC _PROTOTYPE (int (*fs_call_vec[]), (void)) = ...	FW

3	**/servers/iso9660fs/utility.c**	Содержит функции:
		int no_sys();
		- кто-то использовал несуществующий номер системного вызова.
		void panic(who, mess, num);
		- паника возникает, когда обнаруживается внутреннее несоответствие (ошибка программы, или недопустимое значение константы).
			FW

3	**/servers/pm/alarm.c**	Этот файл обеспечивает системные вызовы, связанные с отложенными процессами (системным будильником (?) (alarm clock)), периодически посылая работу функциям в файле «timer.c» и check_sig() в файле «signal.c», для посылки сигнала пробуждения ((?) larm signal) процессу. Точки входа: do_itimer: выполняет системный вызов ITIMER; do_alarm: выполняет системный вызов ALARM; set_alarm: сообщает интерфейсу таймера, чтобы запустить или остановить таймер процесса; check_vtimer: проверяет необходимость перезапуска виртуальных таймеров.	FW
3	**/servers/pm/break.c**	Содержит функцию: int do_brk(); Точка входа для системного вызова brk(addr).	FW
5	**/servers/pm/const.h**	Константы, используемые сервером управления процессами (PM). NR_PIDS, PM_PID, INIT_PID, NO_TRACER, DUMPED, MAX_SECS, NR_ITIMERS	FW

3	**/servers/pm/dma.c**	Содержит функции:
		int do_adddma();
		int do_deldma();
		int do_getdma();
			FW

pr.	*path*	*an.*	*aut*
3	**/servers/pm/exec.c**	Этот файл обеспечивает системный вызов EXEC. Порядок выполняемой работы: - проверяет, что файл может быть исполнен (по разрешениям UNIX); - читает заголовок и получает размеры; - получает начальные аргументы и переменные окружения из пространства пользователя; - выделяет память для нового процесса; - копирует начальный стек из PM в проесс; - читает сегменты кода и данных и копирует в процесс; - контролирует биты setuid, setgid; - исправляет (изменяет) таблицу «mproc»; - сообщает микроядру о EXEC; - сохраняет смещение для начального argc (для PS). Точки входа этого файла: do_exec: выполняет системный вызов EXEC; exec_newmem: выделяет новую карту памяти для процесса, который пытается выполнить EXEC; do_execrestart: заканчивает специальный вызов exec для сервера реинкарнации (RS); exec_restart: заканчивает обычный вызов exec; find_share: находит процесс, сегмент кода которого может быть совместно использова.	FW
3	**/servers/pm/forkexit.c**	Этот файл обеспечивает создание процесса (через системный вызов FORK) и уничтожение (стирание) процесса (через EXIT/WAIT). Когда процесс раздваивается (forks), для него (точнее - процесса-ребёнка) выделяется новый слот в таблице «mproc», а также копия образа родительского процесса (для процесса-ребёнка). Затем информируется ядро и файловая система. Процесс удаляется из таблицы «mproc» когда происходят два события: (1) процесс завершил исполнение или был убит посредством сигнала, и (2) процесс-родитель выполнил WAIT. Если процесс завершил исполнение прежде (как его родитель вызвал WAIT), то слот продолжает существовать до тех пор, пока процесс родитель не выполнит соответствующий WAIT. Точки входа: do_fork: выполняет системный вызов FORK; do_fork_nb: неблокирующая версия FORK для сервера реинкарнации (RS); do_exit: выполняет системный вызов EXIT (посредством вызова exit_proc()); exit_proc: собственно и выполняет завершение процесса, а также сообщает файловой системе об этом; exit_restart: продолжает завершение процесса после того, как получен ответ от файловой системы (FS); do_waitpid: выполняет системные вызовы WAITPID или WAIT; wait_test:	FW

pr.	*path*	*an.*	*aut*
		проверяет: ждёт ли процесс-родитель завершения данного процесса.	
3	**/servers/pm/getset.c**	Этот файл обеспечивает 6 системных вызовов, которые получают и устанавливают значения uid, gid. Он также обеспечивает getpid(), setsid(), и getpgrp(). Реально содержит функции: **int do_get();** обеспечивает системные вызовы GETUID, GETGID, GETPID, GETPGRP. **int do_set();** обеспечивает системные вызовы ETUID, SETEUID, SETGID, SETEGID, SETSID (эти вызовы связаны также с VFS).	FW
5	**/servers/pm/glo.h**	Глобальные переменные сервера (управления процессами). Они реально выделяются в table.o .	FW
5	**/servers/pm/main.c**	Этот файл содержит функцию main для сервера управления процессами и некоторые связанные с main процедуры. Когда MINIX запускается, в самом начале микроядро инициализует себя свои задания, а затем оно передаёт управление серверам управления процессами (PM) и файловой системы (FS). Оба: PM и FS инициализуют себя настолько, насколько они могут. Сервер управления процессами (PM) запрашивает микроядро для всей свободной памяти и начинает обслуживать запросы. Точки входа: main: запускает сервер управления процессами; setreply: устанавливает ответ, который отсылается процессу, выполняющему системный вызов к серверу управления процессами (PM).	FW
3	**/servers/pm/misc.c**	Различные системные вызовы: do_reboot: «уничтожает» все процессы, затем перезагружает систему; do_procstat: запрашивает статус процесса (Jorrit N. Herder) do_getsysinfo: запрашивает копию структуры данных PM (Jorrit N. Herder); do_getprocnr: ищет номер слота процесса (Jorrit N. Herder) do_getpuid: по данной конечной точке ((?) endpoint) находит uid/euid процесса; do_allocmem: выделяет (процессу) чанк памяти (Jorrit N. Herder); do_freemem: удаляет (из адресного пространства процесса) чанк памяти (Jorrit N. Herder); do_getsetpriority: получает/устанавливает приоритет процесса; do_svrctl: контроль управляющего процессами (планировщика (?)).	FW
6	**/servers/pm/mproc.h**	Эта таблица содержит по одному слоту для каждого процесса. Она содержит всю информацию по управлению процессами для каждого процесса. Кроме всего прочего, она определяет сегменты кода, данных и стека, uids and gids (?), а также различные флаги. Микроядро и файловая система также имеют таблицы, индексированные по процессам, которые содержат соответствующие слоты, относящиеся к одному процессу во всех трёх таблицах.	FW
6	**/servers/pm/param.h**	Соответствующие имена являются синонимами для переменных во входящих сообщениях: exec_name, exec_len, func, grp_id, namelen, pid, seconds, which_timer, new_val, old_val, sig, stack_bytes, stack_ptr, status, usr_id, request, data, sig_nr, sig_nsa, sig_osa, sig_ret, stat_nr, sig_set, sig_how, sig_context, info_what, info_where, reboot_flag, reboot_code, reboot_strlen, svrctl_req, svrctl_argp, stime, memsize, membase,	FW

pr.	*path*	*an.*	*aut*
		sysuname_req, sysuname_field, sysuname_len, sysuname_value. Соответствующие имена являются синонимами для переменных в исходящих (посылаемых) сообщениях: reply_res, reply_res2, reply_res3, reply_ptr, reply_mask, reply_trace, reply_time, reply_utime, reply_t1, reply_t2, reply_t3, reply_t4, reply_t5.	
5	**/servers/pm/pm.h**	Это основной заголовочный файл сервера управления процессами. Он включает некторые другие файлы и определяет некоторые важнейшие константы: _POSIX_SOURCE, _MINIX, _SYSTEM. (В общем – это стандартный заголовочный файл подсистемы...)	FW
3	**/servers/pm/profile.c**	Этот файл содержит точки входа для системного профилирования: do_sprofile: начать/закончить статистическое профилирование; do_cprofile: получить/сбросить таблицы профилирования вызовов (функций); Содержит также функцию: int check_addrs(info_size). (Rogier Meurs)	FW
5	**/servers/pm/proto.h**	Прототипы функций, приведенные по файлам. (В общем это также некий стандартный файл для всех подсистем...)	FW
3	**/servers/pm/signal.c**	Этот файл поддерживает сигналы, которые являются асинхронными событиями, представляющими собой «грязную и неблагодарную» работу. Сигналы могут быть сгенерированы посредством системного вызова KILL, или от клавиатуры (SIGINT) или от часов (SIGALRM). Во всех случаях контроль в дальнейшем передаётся к функции check_sig() для определения того, какому процессу посылается сигнал ((?) нуждается в проверке реальный смысл текста!). Реально сигнал выполняется (т.е. изменяется состояние соответствующего процесса) посредством функции sig_proc(). Точки входа: do_sigaction: выполняет системный вызов SIGACTION; do_sigpending: выполняет системный вызов SIGPENDING; do_sigprocmask: выполняет системный вызов SIGPROCMASK; do_sigreturn: выполняет системный вызов SIGRETURN; do_sigsuspend: выполняет системный вызов SIGSUSPEND; do_kill: выполняет системный вызов KILL; do_pause: выполняет системный вызов PAUSE; ksig_pending: микроядро уведомляется о необработанном сигнале; sig_proc: прерывает или завершает сигнализирующий (или сигнализируемый (?)) процесс; check_sig: проверяет, какие процессы должны сигнализироваться посредством sig_proc(); check_pending: проверяет, может ли необработанный (ранее) сигнал сейчас быть обработан; restart_sigs: возобновляет работу с сигналами после вызова сервера (серверов) файловой системы.	FW
5	**/servers/pm/table.c**	Этот файл содержит таблицу, которая используется для отображения номеров системных вызовов на процедуры, их обрабатывающие. (В общем то также стандартный файл для подсистем, связанных с обработкой системных вызовов ...)	FW
3	**/servers/pm/time.c**	Этот файл «заботится» о системных вызовах, связанных со временем.	FW

pr.	*path*	*an.*	*aut*
		Точки входа: do_time: выполняет системный вызов TIME; do_stime: выполняет системный вызов STIME; do_times: выполняет системный вызов TIMES;	
3	**/servers/pm/timers.c**	Управление временем «сторожевого пса» ((?) watchdog) сервера управления процессами. Функции этого файла обеспечивают удобный интерфейс к библиотеке таймеров, которая управляет списком таймеров watchdog (?). Все детали планирования и сигналов пробуждения (alarm (?)) задания микроядра CLOCK сокрыты этим интерфейсом. Тобко системные процессы имеют возможность установить сигнал пробуждения (alarm (?)) в микроядре. Поэтому сервер управления процессами (PM) поддерживает локальный список пользовательских процессов (непривилегированных процессов (?)), которые запросили сигнал пробуждения (alarm (?)). Точки входа: pm_set_timer: запускает заново существующий, или создаёт новый watchdog (?) таймер; pm_expire_timers: проверяет на истечение времени таймеров и запускает соответствующие watchdog (?) функции. pm_cancel_timer: удаляет время из списка таймеров (?).	FW
3	**/servers/pm/trace.c**	Этот файл обеспечивает часть сервера управления процессами (PM), ответственную за отладочные функции, использующие системный вызов ptrace. Большинство команд в дальнейшем посылаются в системное задание микроядра (SYSTEM) для завершения. Доступные отладочные команды: T_STOP останавливают процесс, T_OK включает трассировку родителем этого процесса, T_GETINS возвращает значение из пространства инструкций, T_GETDATA возвращает значение из пространства данных, T_GETUSER возвращает значение из таблицы пользовательского процесса, T_SETINS устанавливает значение в пространстве инструкций, T_SETDATA устанавливает значение в пространстве данных, T_SETUSER устанавливает значение в таблице пользовательского процесса, T_RESUME восстановить исполнение, T_EXIT выход, T_STEP устанавливает бит трассирования, T_SYSCALL системный вызов трассирования, T_ATTACH подключить к существующему процессу, T_DETACH отключить от существующего процесса, T_SETOPT устанавливает опции трассирования. T_OK, T_ATTACH, T_EXIT, и T_SETOPT обеспечиваются здесь; T_RESUME, T_STEP, T_SYSCALL, и T_DETACH частично обеспечиваются здесь, но завершение осуществляется в задании SYSTEM микроядра. Остальные в основном обеспечиваются в задании SYSTEM микроядра.	FW
5	**/servers/pm/type.h**	Если и имеются какие либо локальные для сервера управления процессами определения типов, то они должны быть в этом файле. Этот файл включён только для «симметрии» с микроядром и сервером файловой системы, которые имеют некоторые локальные определения типов.	FW

pr.	*path*	*an.*	*aut*
		(Тоже похож на стандартный файл подсистемы ...)	
3	**/servers/pm/utility.c**	Этот файл содержит некоторые обслуживающие функции скрвкра управления процессами (PM). Точки входа: get_free_pid: get a free process or group id no_sys: called for invalid system call numbers find_param: look up a boot monitor parameter find_proc: return process pointer from pid number pm_isokendpt: check the validity of an endpoint tell_fs: send a request to FS on behalf of a process (Тоже похож на стандартный файл подсистемы ...)	FW
5	**/servers/ds/glo.h**	Глобальные переменные: int who; номер вызывающего процесса, int callnr; номер системного вызова, int dont_reply; нормально 0; установка в 1 подавляет ответ.	FW
5	**/servers/ds/inc.h**	Заголовочный файл, включающий все необходимые системные заголовки (стандартный для серверов (?)).	FW
5	**/servers/ds/main.c**	Сервер сохранения данных. Этот сервис обеспечивает сохранение небольших (publish/subscribe (?)) данных, критичных для устойчивости при системных сбоях. Компоненты, которые могут иметь различные состояния, могут их сохранять здесь для получения позже: после падения и последующего перезапуска посредством сервера реинкарнации (RS).	FW
5	**/servers/ds/proto.h**	Содержит прототипы функций (публичного интерфейса файлов) сервера сохранения данных. Стандартный файл для подсистем.	FW
3	**/servers/ds/store.h**	Определения типов и констант для сервера сохранения данных (DS).	FW

3	**/servers/ds/store.c**	Основной файл, в котором определено функционирование сервера сохранения данных.
		
		Функции:
		void ds_init(void);
		int set_owner(dsp, auth); /* всегда возвращает TRUE !*/
		int is_authorized(dsp, ap); /* всегда возвращает TRUE !*/
		int find_key(key_name, dsp, type);
		int do_publish(m_ptr);
		Сохраняет пару (ключ , значение). Сначала проверяет наличие ключа. Если ключ имеется. То проверяется наличие полномочий к вызывающего процесса переписать значение данного ключа. Если ключ отсутствует, то находится новый слот для нового значения.
		int do_retrieve(m_ptr);
		int do_check(m_ptr);
		Эта процедура проходит сквозь все записи для данного клиента и проверяет все элементы данных, если они были обозначены (т.е. созданы или изменены) ((?) matching that subscription.) Возвращает сообщение и копию ключей и значений для каждого.
		int do_subscribe(m_ptr);
			FW

pr.	*path*	*an.*	*aut*

		Выполняет выписку интересующего ключа. Все обновления ключа вызывают сообщение, посылаемое абоненту. При успехе, прямо возвращает копию данных для данного ключа.
		int do_getsysinfo(m_ptr);
		void check_subscribers(struct data_store *dsp);
		Посылает абонентам, котрые соответствуют этому (новому или обновлённому) элементу данных уведомление, и помечает данную запись, как обновлённую.	

3	**/servers/inet/buf.c**	Этот файл содержит процедуры для правления буферами. (Philip Homburg)	FW

3	**/servers/inet/clock.c**	Содержит функции:
		void clck_init();
		time_t get_time();
		void set_time (tim);
		void reset_time();
		void clck_timer(timer, timeout, func, fd);
		void clck_tick (mess);
		void clck_fast_release (timer);
		void set_timer();
		void clck_untimer (timer);
		void clck_expire_timers();
			FW

5	**/servers/inet/const.h**	Содержит определения некоторых констант.... (Стандартный файл для подсистем ...)	FW
5	**/servers/inet/inet.h**	Основной заголовочный файл сетевого сервера (INET). (Стандартный файл для подсистемы ...)	FW
5	**/servers/inet/inet.c**	Этот файл содержит интерфейс для сетевого программного обеспечения (with rest of minix (?)). Кроме того, он содержит основной цикл сетевого задания. (Сетевой сервер (INET) не содержит файла main.c – данный файл выполняет его функцию!) В начале файла содержится описание возможных сообщений и их параметров от сервера файловой системы (FS).	FW
5	**/servers/inet/inet_config.h**	Определяет значения для конфигурационных параметров сетевого сервера (INET). Также содержит определения структур для конфигурационной информации.	FW
3	**/servers/inet/inet_config.c**	Читает конфигурационный файл и заполняет массивы xx_conf[]	FW

3	**/servers/inet/mnx_eth.c**	Содержаться функции:
		void osdep_eth_init();
		Вначале инициализует обычный сетевой интерфейса, а затем и VLAN.
		void eth_write_port(eth_port, pack);
		void eth_rec(m);
		void eth_check_drivers(m);
		int eth_get_stat(eth_port, eth_stat);
		void eth_set_rec_conf (eth_port, flags);
		void eth_issue_send(eth_port);
		void write_int(eth_port);
		void read_int(eth_port, count);
		void setup_read(eth_port);
		void eth_recvev(ev, ev_arg);
		void eth_sendev(ev, ev_arg);
		eth_port_t *find_port(m);
		static void eth_restart(eth_port, tasknr);
			FW

pr.	*path*	*an.*	*aut*
		void send_getstat(eth_port);	
3	**/servers/inet/mq.h**	Определения структуры mq, и прототипов функций, связанных с её обслуживанием.	FW

3	**/servers/inet/mq.c**	Содержит функции:
		void mq_init();
		mq_t *mq_get();
		void mq_free(mq);
			FW

4	**/servers/inet/osdep_eth.h**	Определения структуры osdep_eth_port_t; а также констант: OEPS_INIT, OEPS_CONF_SENT, OEPS_IDLE, OEPS_RECV_SENT, OEPS_SEND_SENT, OEPS_GETSTAT_SENT; OEPF_EMPTY, OEPF_NEED_RECV, OEPF_NEED_SEND, OEPF_NEED_CONF, OEPF_NEED_STAT	FW
5	**/servers/inet/proto.h**	Содержит прототипы функций (публичного интерфейса файлов) сетевого сервера (INET). Стандартный файл подсистем...	FW
3	**/servers/inet/qp.h**	Обеспечивает запросы на параметры вопросов ((?) queryparams).	FW
3	**/servers/inet/qp.c**	Параметры вопросов. ((?) queryparams)	FW
2	**/servers/inet/sha2.h**	Порт соответствующего способа аутентификации целостности файлов.	FW
2	**/servers/inet/sha2.c**	Порт соответствующего способа аутентификации целостности файлов.	FW
3	**/servers/inet/sr.c**	Этот файл содержит интерфейс сетевого программного обеспечения с файловой системой. В начале файла дана таблицы формата запросов и ответов.	FW
3	**/servers/inet/sr_int.h**	Внутреннее определение RS ... ((?) SR internals)	FW
5	**/servers/inet/version.c**	Содержит строку, определяющую версию сервера.	FW
3	**/servers/inet/generic/arp.h**	#define ARP_ETHERNET 1 #define ARP_REQUEST 1 #define ARP_REPLY 2 + Прототипы функций ...	FW

3	**/servers/inet/generic/arp.c**	Содержит в том числе функции:
		void arp_prep();
		void arp_init();
		void arp_main(arp_port);
		void arp_set_ipaddr (eth_port, ipaddr);
		int arp_set_cb(eth_port, ip_port, arp_func);
		int arp_ioctl (eth_port, fd, req, get_userdata, put_userdata);
			FW

3	**/servers/inet/generic/assert.h**	Определяет макросы: assert(x), compare(a,t,b)	FW
3	**/servers/inet/generic/buf.h**	Макросы и прототипы ...	FW
3	**/servers/inet/generic/clock.h**	Определения макросов, структур и пототипов	FW
3	**/servers/inet/generic/eth.h**	#define NWEO_DEFAULT (NWEO_EN_LOC | NWEO_DI_BROAD | NWEO_DI_MULTI | \ NWEO_DI_PROMISC | NWEO_REMANY | NWEO_RWDATALL) #define eth_addrcmp(a,b) (memcmp((_VOIDSTAR)&a,	FW

pr.	*path*	*an.*	*aut*
		(_VOIDSTAR)&b, \ sizeof(a))) А также множество прототипов ...	

3	**/servers/inet/generic/eth.c**	Содержит в том числе функции:
		void eth_prep();
		void eth_init();
		int eth_open(port, srfd, get_userdata, put_userdata, put_pkt, select_res);
		int eth_ioctl(fd, req);
		int eth_write(fd, count);
		int eth_send(fd, data, data_len);
		int eth_read (fd, count);
		int eth_cancel(fd, which_operation);
		int eth_select(fd, operations);
		void eth_close(fd);
		void eth_loop_ev(ev, ev_arg);
			FW

3	**/servers/inet/generic/eth_int.h**	Определения: структура eth_port_t; А также константы и макросы , в.т.ч. : #define ETH_TYPE_HASH_NR 16 #define ETH_VLAN_HASH_NR 16	FW
3	**/servers/inet/generic/event.h**	Заголовочный файл для механизма событий.	FW
3	**/servers/inet/generic/event.c**	Реализация очереди событий.	FW
3	**/servers/inet/generic/icmp.h**	Определения констант и макросов.	FW

3	**/servers/inet/generic/icmp.c**	Содержит функции, в.т.ч. :
		void icmp_prep();
		void icmp_init();
		void icmp_snd_time_exceeded(port_nr, pack, code);
		void icmp_snd_redirect(port_nr, pack, code, gw);
		void icmp_snd_unreachable(port_nr, pack, code);
		void icmp_snd_mtu(port_nr, pack, mtu);
			FW

3	**/servers/inet/generic/icmp_lib.h**	Прототипы соответствующих функций.	FW
3	**/servers/inet/generic/io.h**	Прототипы функций ...	FW

3	**/servers/inet/generic/io.c**	Содержит функции:
		void writeIpAddr(addr);
		void writeEtherAddr(addr);
			FW

3	**/servers/inet/generic/ip.h**	Прототипы функций ...	FW

3	**/servers/inet/generic/ip.c**	Содержит массивы:
		ip_fd_t ip_fd_table[IP_FD_NR];
		ip_ass_t ip_ass_table[IP_ASS_NR];
		
		Содержит в.т.ч. функции:
		void ip_prep();
		void ip_init();
		int ip_open (port, srfd, get_userdata, put_userdata, put_pkt, select_res);
			FW

3	**/servers/inet/generic/ip_eth.c**	Специфическая для Ethernet часть реализации IP в.т.ч. : int ipeth_init(ip_port);	FW
3	**/servers/inet/generic/ip_int.h**	Определения типов и констант ... Например:	FW

pr.	*path*	*an.*	*aut*
		typedef enum nettype { IPNT_ZERO, /* 0.xx.xx.xx */ IPNT_CLASS_A, /* 1.xx.xx.xx .. 126.xx.xx.xx */ IPNT_LOCAL, /* 127.xx.xx.xx */ IPNT_CLASS_B, /* 128.xx.xx.xx .. 191.xx.xx.xx */ IPNT_CLASS_C, /* 192.xx.xx.xx .. 223.xx.xx.xx */ IPNT_CLASS_D, /* 224.xx.xx.xx .. 239.xx.xx.xx */ IPNT_CLASS_E, /* 240.xx.xx.xx .. 247.xx.xx.xx */ IPNT_MARTIAN, /* 248.xx.xx.xx .. 254.xx.xx.xx + others */ IPNT_BROADCAST /* 255.255.255.255 */ } nettype_t;	

3	**/servers/inet/generic/ip_ioctl.c**	Содержит в.т.ч. функции:
		int ip_ioctl (fd, req);
		void ip_hash_proto(ip_fd);
		void ip_unhash_proto(ip_fd);
		int ip_setconf(ip_port_nr, ipconf);
			FW

3	**/servers/inet/generic/ip_lib.c**	Содержит в.т.ч. функции:
		ipaddr_t ip_get_netmask (hostaddr);
		int ip_chk_hdropt (opt, optlen);
		void ip_print_frags(acc);
		ipaddr_t ip_get_ifaddr(port_nr);
		nettype_t ip_nettype(ipaddr);
		ipaddr_t ip_netmask(nettype);
		char *ip_nettoa(nettype);
			FW

3	**/servers/inet/generic/ip_ps.c**	Псевдо IP специфическая часть реализации IP адресов.	FW

3	**/servers/inet/generic/ip_read.c**	Содержит в.т.ч. функции:
		int ip_read (fd, count);
		void ip_packet2user (ip_fd, pack, exp_time, data_len);
		void ip_port_arrive (ip_port, pack, ip_hdr);
		void ip_arrived(ip_port, pack);
		void ip_arrived_broadcast(ip_port, pack);
			FW

3	**/servers/inet/generic/ip_write.c**	Содержит в.т.ч. функции:
		int ip_write (fd, count);
		int ip_send(fd, data, data_len);
		void ip_hdr_chksum(ip_hdr, ip_hdr_len);
		acc_t *ip_split_pack (ip_port, ref_last, mtu);
			FW

3	**/servers/inet/generic/ipr.h**	Определяет структуры : oroute_t, iroute_t; а также с ними связанные константы и прототипы.	FW
3	**/servers/inet/generic/ipr.c**	Содержит определения флаговых констант: OROUTE_NR, OROUTE_STATIC_NR, OROUTE_HASH_ASS_NR, OROUTE_HASH_NR, макросов: OROUTE_HASH_MASK как (OROUTE_HASH_NR-1); hash_oroute(port_nr, ipaddr, hash_tmp); структуры: oroute_hash_t Аналогично для iroute ... Файл связан с маршрутизацией.	FW

pr.	*path*	*an.*	*aut*

		Содержит в.т.ч. функции:
		void ipr_init();
		iroute_t *iroute_frag(port_nr, dest);
		int oroute_frag(port_nr, dest, ttl, msgsize, nexthop);
		int ipr_add_oroute(port_nr, dest, subnetmask, gateway, timeout, dist, mtu, static_route, preference, oroute_p);
		int ipr_del_oroute(port_nr, dest, subnetmask, gateway, static_route);
		void ipr_chk_otab(port_nr, addr, mask);
		void ipr_gateway_down(port_nr, gateway, timeout);
		void ipr_destunrch(port_nr, dest, netmask, timeout);
		void ipr_redirect(port_nr, dest, netmask, old_gateway, new_gateway, timeout);
		void ipr_ttl_exc(port_nr, dest, netmask, timeout);
		void ipr_mtu(port_nr, dest, mtu, timeout);
		int ipr_get_oroute(ent_no, route_ent);
		
		int ipr_get_iroute(ent_no, route_ent);
		int ipr_add_iroute(port_nr, dest, subnetmask, gateway, dist, mtu, static_route, iroute_p);
		int ipr_del_iroute(port_nr, dest, subnetmask, gateway, static_route);
		void ipr_chk_itab(port_nr, addr, mask);	

3	**/servers/inet/generic/psip.h**	Публичный интерфейс для модуля псевдо IP.
		
		Содержит прототипы функций:
		void psip_prep ARGS((void));
		void psip_init ARGS((void));
		int psip_enable ARGS((int port_nr, int ip_port_nr));
		int psip_send ARGS((int port_nr, ipaddr_t dest, acc_t *pack));
			FW

3	**/servers/inet/generic/psip.c**	Реализация устройства псевдо IP. Определяет структуры: psip_port_t; psip_fd_t; и связанные с ними флаговые константы. А также содержит определения функций, прототипы которых находятся в заголовочном файле, а также необходимых для них дополнительных функций.	FW

3	**/servers/inet/generic/rand256.h**	Обеспечивает 256-битные (пчевдо-)случайные числа.
		
		Содержит только:
		#define RAND256_BUFSIZE 32
		
		void init_rand256 ARGS((u8_t bits[RAND256_BUFSIZE]));
		void rand256 ARGS((u8_t bits[RAND256_BUFSIZE]));
			FW

3	**/servers/inet/generic/rand256.c**	Генерирует 256-битные (псевдо-) случайные числа. Использует функции из SHA256 ...	FW
4	**/servers/inet/generic/sr.h**	Содержит константы, определения типов, в.т.ч. типов – ссылок на функции ...	FW

6	**/servers/inet/generic/tcp.h**	Определения констант (в основном связанным со временем).
		
		Существенно зависит от тиков, поэтому при изменении частоты тиков, требуется изменить и данный файл...
		
		Содержит также прототипы функций:
		void tcp_prep ARGS((void));
		void tcp_init ARGS((void));
		int tcp_open ARGS((int port, int srfd, get_userdata_t get_userdata,
			FW

pr.	*path*	*an.*	*aut*

		put_userdata_t put_userdata, put_pkt_t put_pkt, select_res_t select_res));
		int tcp_read ARGS((int fd, size_t count));
		int tcp_write ARGS((int fd, size_t count));
		int tcp_ioctl ARGS((int fd, ioreq_t req));
		int tcp_cancel ARGS((int fd, int which_operation));
		void tcp_close ARGS((int fd));	

3	**/servers/inet/generic/tcp.c**	(Определяет соответствующие функции, объявленные в tcp_int.h ... (?)) Использует глобальные константы: PUBLIC tcp_port_t *tcp_port_table; PUBLIC tcp_fd_t tcp_fd_table[TCP_FD_NR]; PUBLIC tcp_conn_t tcp_conn_table[TCP_CONN_NR]; PUBLIC sr_cancel_t tcp_cancel_f; Определяет функции, объявленные в заголовке, кроме того интерес представляют: void tcp_main(tcp_port);	FW
4	**/servers/inet/generic/tcp_int.h**	Определяет тпы tcp_port_t; tcp_fd_t; tcp_conn_t; связанные с ними константы, а также прототипы функций, сгруппированные по файлам. /* tcp_recv.c */ /* tcp_send.c */ /* tcp_lib.c */ /* tcp.c */ Является объединяющим файлом данной группы файлов.	FW
3	**/servers/inet/generic/tcp_lib.c**	Определяет соответствующие функции, объявленные в tcp_int.h ... Следует заметить наличие записей вида: #undef tcp_LEmod4G PUBLIC int tcp_LEmod4G(n1, n2) ... Перед некоторыми функциями.	FW
3	**/servers/inet/generic/tcp_recv.c**	Определяет соответствующие функции, объявленные в tcp_int.h ... В комментариях находятся формализованные алгоритмы (?) ?	FW
3	**/servers/inet/generic/tcp_send.c**	Определяет соответствующие функции, объявленные в tcp_int.h ... (?)	FW
5	**/servers/inet/generic/type.h**	Определения некоторых глобальных функциональных типов (?) .	FW

3	**/servers/inet/generic/udp.h**	Определения констант, и прототипы функций:
		
		void udp_prep ARGS((void));
		void udp_init ARGS((void));
		int udp_open ARGS((int port, int srfd, get_userdata_t get_userdata,
		put_userdata_t put_userdata, put_pkt_t put_pkt, select_res_t select_res));
		int udp_ioctl ARGS((int fd, ioreq_t req));
		int udp_read ARGS((int fd, size_t count));
			FW

pr.	*path*	*an.*	*aut*

		int udp_write ARGS((int fd, size_t count));
		void udp_close ARGS((int fd));
		int udp_cancel ARGS((int fd, int which_operation));	

3	**/servers/inet/generic/udp.c**	Содержит определения функций, объявленных в заголовочном файле.	FW
4	**/servers/inet/generic/udp_int.h**	Некоторые «внутренности» модуля UDP. Определения типов: udp_port_t; udp_fd_t; связанных с ними флаговых констант, а также объявления глобальных переменных: EXTERN udp_port_t *udp_port_table; EXTERN udp_fd_t udp_fd_table[UDP_FD_NR];	FW
3	**/servers/inet/minix3/queryparam.h**	Параметры программ очереди.	FW
3	**/servers/inet/minix3/queryparam.c**	queryparam() - позволяет располагаться параметрам программ в очереди (?).	FW
3	**/servers/init/init.c**	Этот процесс является родителем всех пользовательских процессов Minix3. При старте Minix3, этот процесс с номером 2, и имеет pid 1. Он выполняет скрипт (командного интерпретатора) /etc/rc , а затем читает файл /etc/ttytab для того, чтобы определить какие терминалы требуют (login process) процесса авторизации. Если файлы /usr/adm/wtmp и /etc/utmp существуют и доступны для записи, init (при помощи login) будет сопровождать учётные записи авторизации (?) (login accounting.) Сигнал 1 (SIGHUP) посланный init заставит его прочесть заново /etc/ttytab и запустить новые процессы (командной оболочки), если это необходимо. Однако не будут уничтожены те процессы авторизации, для которых линии в скрипте выключены; выполните это вручную. Сигнал 15 (SIGTERM) выключит создание новых процессов, он используется для shutdown и его «друзей», когда они готовят остановить операционную систему.	FW
5	**/servers/ipc/inc.h**	Содержит стандартные константы и включения для подобного типа подсистем (привилегированных процессов), а также определения внутренних констант и прототипов функций. Данный сервер является довольно новым (включён только в Minix3.1.5) и довольно малым по объёму. Поэтому данный файл выполняет функции сразу нескольких файлов более объёмных подсистем (?).	FW
5	**/servers/ipc/main.c**	Содержит определения структур и глобальных переменных, в.т.ч.: ipc_calls[] = { { IPC_SHMGET, do_shmget, 0 }, { IPC_SHMAT, do_shmat, 0 }, { IPC_SHMDT, do_shmdt, 0 }, { IPC_SHMCTL, do_shmctl, 0 }, { IPC_SEMGET, do_semget, 0 }, { IPC_SEMCTL, do_semctl, 0 },	FW

pr.	*path*	*an.*	*aut*

		{ IPC_SEMOP, do_semop, 1 },
		};
		
		Из функций содержит только
		int main(int argc, char *argv[]);	

3	**/servers/ipc/sem.c**	Содержит функции (связанные с семаформаи) в.т.ч.:
		int do_semget(message *m);
		int do_semctl(message *m);
		int do_semop(message *m);
		int is_sem_nil(void);
			FW

3	**/servers/ipc/shm.c**	Максимальное число областей разделяемой памяти (?):
		#define MAX_SHM_NR 1024
		
		Содержит функции (связанные с разделяемой памятью) в.т.ч.:
		int do_shmget(message *m);
		int do_shmat(message *m);
		void update_refcount_and_destroy(void);
		int do_shmdt(message *m);
		int do_shmctl(message *m);
		void list_shm_ds(void);
		int is_shm_nil(void);
			FW

3	**/servers/ipc/utility.c**	Содержит только функцию int check_perm(struct ipc_perm *req, endpoint_t who, int mode);	FW
4	**/servers/is/dmp.c**	Этот файл содержит процедуры по выдаче (?) (dump) информации (внутренней информации о системе). В процессе инициализации сервера системной информации (IS) 'известные' функциональные клавиши регистрируются в сервере TTY ((?) терминальном сервере – вообще-то это драйвер ...) в порядке получения извещения если одна из них нажата. Здесь вызывается соответствующая (зарегистрированная) процедура. Точка входа: handle_fkey: обеспечивает функцию извещения о нажатой функциональной клавише. Содержит в.т.ч. список значений функциональных клавиш.	FW
3	**/servers/is/dmp_ds.c**	Этот файл содержит процедуры по выдаче (?)(dump) структур данных сервера данных (?) (DS). Точка входа: data_store_dmp: показывает содержимое хранилища данных сервера данных (DS).	FW
3	**/servers/is/dmp_fs.c**	Этот файл содержит процедуры по выдаче данных структур данных файловой системы (FS). Точки входа: dtab_dump: показывает отображение утройство <-> драйвер fproc_dump: показывает таблицу процессов файловой системы (FS) (или содержимое proc FS (?))	FW
3	**/servers/is/dmp_kernel.c**	Отладочные процедуры выдачи ((?) dump) для микроядра.	FW
3	**/servers/is/dmp_pm.c**	Этот файл содержит процедуры выдачи ((?) dump) структур данных сервера управления процессами (PM).	FW

pr.	*path*	*an.*	*aut*
		Точка входа: mproc_dmp: показывает таблицу процессов сервера управления процессами.	
3	**/servers/is/dmp_rs.c**	Этот файл содержит процедуры выдачи ((?) dump) структур данных сервера реинкарнации (RS). Точка входа: rproc_dump: показывает таблицу системных процессов сервера реинкарнации (RS)	
5	**/servers/is/glo.h**	Содержит глобальные переменные сервера системной информации (IS).	FW
4	**/servers/is/inc.h**	Заголовочный файл для сервера системной информации (IS). Типичный файл для подсистем подобного рода (привелигированных процессов и заданий микроядра).	FW
4	**/servers/is/is.h**	Заголовочный файл для сервера системной информации (IS). В чём-то очень похож на **/servers/is/is.h** (?)	FW

5	**/servers/is/main.c**	Сервер системной информации (IS).
		
		Эльль сервис обеспечивает различные отладочные выдачи информации (?)(dump), такие как таблица процессов, хотя это больше не затрагивает память микроядра напрямую. Вместо этого системное задание (SYSTEM) вызывается для копирования некоторых таблиц в локальной памяти.
		
		Определяет глобальные переменные:
		message m_in; /* the input message itself */
		message m_out; /* the output message used for reply */
		int who_e; /* caller's proc number */
		int callnr; /* system call number */
		
		а также стандартные процедуры для данного файла подобных подсистем:
		int main(int argc, char **argv);
		void init_server(int argc, char **argv);
		void get_work();
		void reply(who, result);
			FW

5	**/servers/is/proto.h**	Содержит прототипы функций, сгруппированные по файлам.	FW

3	**/servers/rs/exec.c**	Содержит функции:
		int dev_execve(int proc_e, char *exec, size_t exec_len, char **argv, char **Xenvp);
		tatic void do_exec(int proc_e, char *exec, size_t exec_len, char *progname, char *frame, int frame_len);
		int exec_newmem(proc_e, text_bytes, data_bytes, bss_bytes, tot_bytes, frame_len, sep_id, st_dev, st_ino, st_ctime, progname, new_uid, new_gid, stack_topp, load_textp, allow_setuidp);
		int exec_restart(proc_e, result);
		int read_header(exec, exec_len, sep_id, text_bytes, data_bytes, bss_bytes, tot_bytes, pc, hdrlenp);
		void patch_ptr(stack, base);
		int read_seg(exec, exec_len, off, proc_e, seg, seg_bytes);
			FW

5	**/servers/rs/inc.h**	Заголовочный файл для сервера управления системными процессами (сервера реинкарнации (RS)(?)).	FW

pr.	*path*	*an.*	*aut*
		Содержит стандартные константы _SYSTEM, _MINIX, а в остальном – только набор заголовочных файлов.	
5	**/servers/rs/main.c**	Сервер реинкарнации (RS). Этот сервер запускает новые системные сервисы и определяет, что они вышли. В случае ошибок, системный сервис может быть перезапущен. Сервер реинкарнации (RS) периодически проверяет статус всех зарегистрированных сервисов для того, чтобы убедиться в том, что они всё ещё работоспособны. От системных сервисов ожидается периодический посыл сообщений «сердцебиения» (heartbeat message). Содержит (стандартные для привилегированного процесса) функции: int main(void); void init_server(void); void sig_handler(); void get_work(m_in); void reply(who, m_out);	FW
3	**/servers/rs/manager.h**	Эта таблица содержит по одному слоту для каждого системного процесса. Она содержит информацию для серверов и драйверов, необходимую серверу реинкарнации (RS) для того, чтобы прослеживать статус каждого такого процесса. Содержит определения констант препроцессора, определяющих максимальные длину строки аргументов, уникального имени экземпляра драйвера, имени скрипта перезапуска, максимальное число аргументов (4), максимальную длину «спасительного» каталога, а также максимальное число PCI ID, различных классов PCI ID (?), MAX_NR_SYSTEM 2 /* should match RSS_NR_SYSTEM */ (?), максимальное число списка имён целевых процессов для межпроцессного взаимодействия (IPC) , MAX_VM_LIST 256 , ... Опрделяет глобальную переменную rproc[NR_SYS_PROCS]; - таблицу системных процессов (Эта таблица содержит только записи для серверов и драйверов, однако она не является непосредственно индексированной по номерам слотов). А также некоторые флаговые константы, константы определяющие период сервера реинкарнации (RS) (связан с тиками!) , ...	FW
3	**/servers/rs/manager.c**	Содержит функции: **int do_up(m_ptr, do_copy, flags);** Дан запрос на запуск нового системного сервиса. Метит (удаляет - dismember (?)) сообщение запроса, и собирает всю информацию, необходимую для запуска сервиса. Запуск осуществляется при помощи вспомогательных процедур. **int do_start(m_ptr);** Дан запрос на запуск нового системного сервиса. **int do_down(message *m_ptr);** **int do_restart(message *m_ptr);** **int do_refresh(message *m_ptr);** **int do_shutdown(message *m_ptr);** Устанавливает флаги так, чтобы сервер реинкарнации знал, что данный сервис не должен быть перезапущен. **void do_exit(message *m_ptr);** Проверяет, какой ребёнок выходит (заканчивает выполнение) и каков его статус окончания (?). Это делается в цикле потому, что множество детей может закончить исполнение, и о всех будет сообщаться в одном SIGCHLD. Опция WNOHANG используется для	FW

pr.	*path*	*an.*	*aut*
		того, чтобы запретить блокировку, если каким-то образом обнаружен ребёнок, не завершивший исполнение. **void do_period(m_ptr);** **int start_service(rp, flags, endpoint);** Пробует исполнить данный системный сервис. Получает путём разветвления новый процесс. Исполнение процесса ребёнка предотвращается путём установки флага NO_PRIV. Его исполнение будет разрешено только после того, как процесс родитель установит его привилегии. **int stop_service(rp,how);** Пробует остановить системный сервис. Сначала посылает сигнал SIGTERM , который «просит» системному сервису остановиться. Если сервис не установил поддержку сигналов, он будет внешне уничтожен. Тот факт, что процесс игнорирует сигналы будет установлен потому, что фиксируется время посылки сигнала и посылается SIGKILL. **int do_getsysinfo(m_ptr);**	
5	**/servers/rs/proto.h**	Прототипы функций, сгруппированные по файлам. Это стандартный заголовочный файл проекта подобных подсистем.	FW
3	**/servers/rs/service.c**	Утилиты для запуска и остановки системного сервиса. Запрос посылается серверу реинкарнации (RS), который и выполняет реальную работу. В частности, одержит всех известных (серверу реинкарнации (RS)) запросов, скрипт по умолчанию для запуска, ...	FW
3	**/servers/vm/addravl.h**	Содержит определения констант и макросов препроцессора.	FW
3	**/servers/vm/addravl.c**	Содержит только включения заголовков (!) (?)	FW
3	**/servers/vm/alloc.c**	Этот файл связан с выделением и освобождением блоков физической памяти произвольной величины в контексте системных вызовов FORK и EXEC. Ключевая используемая структура данных – таблица свободных участков ((?) hole table), которая поддерживает список свободных участков памяти. Они поддерживаются в порядке возрастания адреса памяти. Эти адреса содержат ссылки на физическую память, начиная с абсолютного адреса 0 (т.е. они не являются относительными от начала (адресного пространства) сервера управления процессами (PM)). Во время инициализации системы, часть памяти, содержащая вектора прерываний, микроядро и сервер управления процессами (PM) «размещены» в смысле пометки их (областей памяти) как недоступными для выделения и удаления их из списка свободных мест (?). Точки входа: alloc_mem: выделяет чанк памяти данного размера; free_mem: освобождает прежде выделенный чанк памяти; mem_init: инициализирует таблицы когда сервер управления процессами запускается.	FW
3	**/servers/vm/break.c**	Модель выделения памяти MINIX резервирует участки памяти фиксированного размера ((?) a fixed amount of memory) для комбинированных сегментов кода, данных и стека. Эти размеры используются для процесса-ребёнка созданного посредством FORK те же, что и для процесса-родителя. Если процесс-ребёнок позже выполняет EXEC , новые размеры из заголовка запускаемого	FW

pr.	*path*	*an.*	*aut*
		бинарного исполняемого файла. Расположение памяти состоит из сегмента текста, после которого следует сегмент данных, за которым следует «куча» (неиспользуемая память), после которой следует сегмент стека. Сегмент данных растёт вверх, а сегмент стека – вниз, таким образом каждый из них может заимствовать память из «кучи». Если они встречаются процесс должен быть уничтожен. Процедуры данного файла заботятся о росте сегментов данных и стека. Точки входа: do_brk: системные вызовы BRK/SBRK для роста или сокращения сегмента данных; adjust: смотрит разрешено ли запрашиваемое изменение сегментов ((?) в смысле полномочий и привилегий ?).	
2	**/servers/vm/cavl_if.h**	Абстрактное дерево AVL (?) - общий пакет. Заголовочный файл интерфейса создания. Портирован: см**. cavl_tree.html**	FW
2	**/servers/vm/cavl_impl.h**	Реализация: Абстрактное дерево AVL (?) - общий пакет. Портирован: см**. cavl_tree.html**	FW

3	**/servers/vm/exec.c**	Содержит функции:
		struct vmproc *find_share(vmp_ign, ino, dev, ctime);
		Ищет процесс, который исполняет файл <ino, dev, ctime>. Не обязательно найдёт vmp_ign, так как это процесс, от лица которого этот вызов выполняется.
		int do_exec_newmem(message *msg);
		
		int new_mem(rmp, sh_mp, text_bytes, data_bytes, bss_bytes, stk_bytes, tot_bytes, stack_top);
		Выделяет новую память и освобождает старую память. Изменяет карту памяти и сообщает новую микроядру. Обнуляет bss, «кучу» и стек нового образа (памяти).
		phys_bytes find_kernel_top(void);
		Определяет где находится микроядро, чтобы таким образом знать начало отображения пользовательских процессов.
		
		i**nt proc_new(struct vmproc *vmp**,
		phys_bytes vstart, /* где начать процесс в таблице страниц*/
		phys_bytes text_bytes, /* как много кода, в байтах но выравненного по страницам*/
		phys_bytes data_bytes, /* как много данных + bss, в байтах но ...*/
		phys_bytes stack_bytes, /* резервируемый размер стека, в байтах но ... */
		phys_bytes gap_bytes, /* «куча» , в байтах но ...*/
		phys_bytes text_start, /* код начинается здесь, если предварительно выделен, иначе 0 */
		phys_bytes data_start, /*участок данных начинается здесь, если предварительно выделен, иначе 0 */
		phys_bytes stacktop
		);
			FW

3	**/servers/vm/exit.c**	Содержит функции:	FW

pr.	*path*	*an.*	*aut*

		void free_proc(struct vmproc *vmp);
		void clear_proc(struct vmproc *vmp);
		int do_exit(message *msg);
		int do_willexit(message *msg);
		void _exit(int code);
		void __exit(int code);	

3	**/servers/vm/fork.c**	Содержит функцию: int do_fork(message *msg);	FW
2	**/servers/vm/glo.h**	Определения некоторых констант препроцессора.	FW

5	**/servers/vm/main.c**	Стандартный файл для подсистем подобного рода. Содержит традиционные функции:
		
		int main(void);
		void vm_init(void);
			FW

3	**/servers/vm/mmap.c**	Содержит функции:
		
		int do_mmap(message *m);
		int do_map_phys(message *m);
		int do_unmap_phys(message *m);
		int do_remap(message *m);
		int do_shared_unmap(message *m);
		int do_get_phys(message *m);
		int do_get_refcount(message *m);
		int do_munmap(message *m);
		int munmap_lin(vir_bytes addr, size_t len); *(override for VM)*
		int munmap(void *addr, size_t len); *(override for VM)*
		int munmap_text(void *addr, size_t len);*(override for VM)*
			FW

3	**/servers/vm/pagefaults.c**	Содержит функции:
		
		*pf_errstr(u32_t err);
		void do_pagefaults(void);
		void do_memory(void);
		int handle_memory(struct vmproc *vmp, vir_bytes mem, vir_bytes len, int wrflag);
			FW

3	**/servers/vm/pagerange.h**	Определяет структуру: pagerange_t; (?) /* AVL fields */ /* AVL balance factor */	FW
3	**/servers/vm/physravl.h**	Определяет константы и макросы.	FW
3	**/servers/vm/physravl.c**	Состоит только из включений заголовочных файлов.	FW
5	**/servers/vm/proto.h**	Содержит прототипы функций сгруппированные по файлам... Стандартный файл подобного вида подсистем.	FW

3	**/servers/vm/queryexit.c**	Содержит функции:
		
		int do_query_exit(message *m);
		int do_notify_sig(message *m);
		void init_query_exit(void);
			FW

3	**/servers/vm/region.h**	Определяет типы: struct phys_block	FW

pr.	*path*	*an.*	*aut*
		phys_region_t; struct vir_region Также определяет флаги отображения памяти.	

3	**/servers/vm/region.c**	Определяет макрос
		WRITABLE(r, pb)
		
		Содержит функции:
		
		char *map_name(struct vir_region *vr);
		void map_printregion(struct vmproc *vmp, struct vir_region *vr);
		void map_printmap(vmp);
		int map_ph_writept(struct vmproc *vmp, struct vir_region *vr, struct phys_region *pr);
		vir_bytes region_find_slot(struct vmproc *vmp, vir_bytes minv, vir_bytes maxv, vir_bytes length, struct vir_region **prev);
		struct vir_region *map_page_region(vmp, minv, maxv, length, what, flags, mapflags);
		void pb_unreferenced(struct vir_region *region, struct phys_region *pr);
		int map_subfree(struct vmproc *vmp, struct vir_region *region, vir_bytes len);
		int map_free(struct vmproc *vmp, struct vir_region *region);
		int map_free_proc(vmp);
		struct vir_region *map_lookup(vmp, offset);
		struct phys_region *map_new_physblock(vmp, region, offset, length, what_mem);
		int map_copy_ph_block(vmp, region, ph);
		int map_pf(vmp, region, offset, write);
		int map_handle_memory(vmp, region, offset, length, write);
		struct vir_region *map_copy_region(struct vmproc *vmp, struct vir_region *vr);
		int map_writept(struct vmproc *vmp);
		int map_proc_copy(dst, src);
		struct vir_region *map_proc_kernel(struct vmproc *vmp);
		int map_region_extend(struct vmproc *vmp, struct vir_region *vr, vir_bytes delta);
		int map_region_shrink(struct vir_region *vr, vir_bytes delta);
		struct vir_region *map_region_lookup_tag(vmp, tag);
		void map_region_set_tag(struct vir_region *vr, u32_t tag);
		u32_t map_region_get_tag(struct vir_region *vr);
		int map_unmap_region(struct vmproc *vmp, struct vir_region *region, vir_bytes len);
		int map_remap(struct vmproc *dvmp, vir_bytes da, size_t size, struct vir_region *region, vir_bytes *r);
		int map_get_phys(struct vmproc *vmp, vir_bytes addr, phys_bytes *r);
		int map_get_ref(struct vmproc *vmp, vir_bytes addr, u8_t *cnt);
		void printregionstats(struct vmproc *vmp);
			FW

6	**/servers/vm/rs.c**	Содержит функцию: int do_rs_set_priv(message *m);	FW
3	**/servers/vm/sanitycheck.h**	Определяет множество макросов для дополнительных проверок корректности работы сервера.	FW
3	**/servers/vm/signal.c**	Определяет константы #define DATA_CHANGED 1 #define STACK_CHANGED 2 Содержит функцию: int do_push_sig(message *msg);	FW

pr.	*path*	*an.*	*aut*

3	**/servers/vm/slaballoc.c**	Содержит огромное множество определений макросов,
		
		а также функции:
		
		struct slabdata *newslabdata(int list);
		void *slaballoc(int bytes);
		int objstats(void *mem, int bytes, struct slabheader **sp, struct slabdata **fp, int *ip);
		void slabfree(void *mem, int bytes);
		void slablock(void *mem, int bytes);
		void slabunlock(void *mem, int bytes);
			FW

3	**/servers/vm/util.h**	Содержит определения макросов ...	FW
3	**/servers/vm/utility.c**	Этот файл содержит некоторые служебные функции для сервера виртуальной памяти (VM): **int get_mem_map(proc_nr, mem_map);** **void get_mem_chunks(mem_chunks);** Инициализирует свободную память из переменной загрузки 'boot'. Переводит байтовые сдвиги и размеры в этом списке в клики, нарезанные правильным образом. ((?) to clicks). **void reserve_proc_mem(mem_chunks, map_ptr);** Выбирает память сервера из списка свободной памяти. Монитор загрузки ((?) boot monitor) обещает поместить процессы в начале чанков памяти ((?) chunks). Все задания (микроядра) используют одинаковый базовый адрес, таким образом только первое задание изменяет списки памяти ((?) memory lists). Серверы и init имеют свои собственные пространства памяти и их память в дальнейшем будет удалена из этого списка (свободной памяти). **int vm_isokendpt(endpoint_t endpoint, int *proc);** **int get_stack_ptr(proc_nr_e, sp);** **int brk(brk_addr);** **int do_ctl(message *m);**	FW

6	**/servers/vm/vfs.c**	Содержит функции:
		
		void register_callback(struct vmproc *for_who, callback_t callback, int callback_type);
		int vfs_open(struct vmproc *for_who, callback_t callback, cp_grant_id_t filename_gid, int filename_len, int flags, int mode);
		int vfs_close(struct vmproc *for_who, callback_t callback, int fd);
		int do_vfs_reply(message *m);
		Ответ на запрос был получен из сервера виртуальной памяти (VFS). Обеспечивает его (Выполняет его). Сначала проверяет и ищет с каким процессом (идентифицируемым по точке окончания ((?) endpoint)), связан данный запрос. Затем вызывает функцию обратного вызова, которая была регистрирована когда был выполнен запрос. Возвращает результат серверу виртуальной памяти (VFS).
			FW

5	**/servers/vm/vm.h**	Определяет некоторые общие константы и макросы.	FW
3	**/servers/vm/vmproc.h**	Определяет типы: typedef void (*callback_t)(struct vmproc *who, message *m); struct vmproc А также флаговые константы.	FW
3	**/ servers/vm/i386/arch_pagefaults.c**	Содержит функцию: int arch_get_pagefault(who, addr, err);	FW

pr.	*path*	*an.*	*aut*
3	**/servers/vm/i386/arch_vmproc.h**	Определяет struct vm_arch vm_data_top указывает на вершину адресного пространства, как оно видно из пространства пользователя, в байтах. Для сегментных процессов это тоже, что и вершина vm_seg[S] сегмента (сегмента стека процесса). Для страничных процессов – это может быть намного выше (таким образом больше памяти доступно под стеком).	FW
3	**/servers/vm/i386/memory.h**	Определяет константы: #define VM_STACKTOP 0x80000000 Вершина стека с точки зрения пользовательского адресного пространства. #define VM_DATATOP 0xFFFFF000 Определяет вершину адресуемого пространтсва. А также некоторые другие подобного рода ...	FW
3	**/servers/vm/i386/pagefaults.h**	Определяет макросы #define PFERR_NOPAGE(e) (!((e) & I386_VM_PFE_P)) #define PFERR_PROT(e) (((e) & I386_VM_PFE_P)) #define PFERR_WRITE(e) ((e) & I386_VM_PFE_W) #define PFERR_READ(e) (!((e) & I386_VM_PFE_W))	FW
3	**/servers/vm/i386/pagetable.h**	Определяет типы: pt_t; Таблица страниц i386. А также архитектурно специфические флаговые константы.	FW
3	**/servers/vm/i386/pagetable.c**	(?) PDE используется для отображения в микроядро, физические адреса микроядра. Определяет понятие кликов памяти (clicks) Определяет функции: **void *aalloc(size_t bytes);** странично выровненный malloc(). Используется только в случае, когда не может быть использован vm_allocpage. **u32_t findhole(pt_t *pt, u32_t vmin, u32_t vmax);** Находит пространство в виртуальном адресном пространстве таблицы страниц 'pt', между странично выровненными байт-смещениями vmin и vmax, чтобы вместить внутри страницу. Возвращает смещение в байтах. **void *vm_allocpage(phys_bytes *phys, int reason);** Выделяет страницу для собственных нужд сервера виртуальной памяти (VM). **void vm_pagelock(void *vir, int lockflag);** Помечает страницу, выделенную посредством vm_allocpage() как недоступную для записи, т.е. только для сервера виртуальной памяти (VM).	FW

pr.	*path*	*an.*	*aut*

		nt pt_writemap(pt_t *pt, vir_bytes v, phys_bytes physaddr, size_t bytes, u32_t flags, u32_t writemapflags);
		Записывает отображение в таблицу страниц. Выделяет новую таблицу страниц, если это необходимо. Каталог страниц и записи таблицы для этих виртуальных адресов.
		
		int pt_checkrange(pt_t *pt, vir_bytes v, size_t bytes, int write);
		
		int pt_new(pt_t *pt);
		Выделяет основу для таблицы страниц ((?)root - заготовку?). Выделяет выровненный по страницам каталог страниц, и устанавливает их на 0 (что указывает – нет выделенных таблиц страниц). Ищет их физические адреса, так как они понадобятся в будещем. Проверяет что они выровнены по страницам.
		
		void pt_init(void);
		По умолчанию микроядро даёт нам сегмент данных с предварительно выделенной памятью, которая не способна расти. Однако мы хотим быть способными выделять память динамически. Здесь мы для себя копируем часть таблицы страниц, таким образом мы получаем частную таблицу страниц. Затем мы увеличиваем размеры аппаратных сегментов – таким образом мы можем выделить память выше нашего стека.
		
		int pt_bind(pt_t *pt, struct vmproc *who);
		
		void pt_free(pt_t *pt);
		Освобождает память, асоциированную с этой таблицей страниц.
		
		int pt_mapkernel(pt_t *pt);
		Каждая страница i386 нуждает в отображении в адресное пространство ядра (в частности – микроядра).
		
		void pt_check(struct vmproc *vmp);
		
		void pt_cycle(void);	

3	**/servers/vm/i386/vm.c**	Содержит функции:
		
		vir_bytes arch_map2vir(struct vmproc *vmp, vir_bytes addr);
		char *arch_map2str(struct vmproc *vmp, vir_bytes addr);
		vir_bytes arch_addrok(struct vmproc *vmp, vir_bytes addr);
		vir_bytes arch_vir2map(struct vmproc *vmp, vir_bytes addr);
		vir_bytes arch_vir2map_text(struct vmproc *vmp, vir_bytes addr);
			FW

3	**/servers/vm/i386/util.s**	Определяет ассемблерную функцию: void i386_invlpg(u32_t addr); Указывает процессору об tlb записи ... (?)	FW
3	**/drivers/amddev/amddev.c**	Драйвер для Вектора Исключения Устройств (DEV) AMD. Начинается с #define _SYSTEM #define _MINIX	FW

pr.	*path*	*an.*	*aut*

		Содержит функцию
		
		int main(void);
		
		а также
		
		static int init(void);
		static int find_dev(int *devindp, u8_t *capaddrp);
		static u32_t read_reg(int function, int index);
		static void write_reg(int function, int index, u32_t value);
		static void init_domain(int index);
		static void init_map(int index);
		static int do_add_phys(message *m);
		static int do_del_phys(message *m);
		static int do_add4pci(message *m);
		static void add_range(u32_t busaddr, u32_t size);
		static void del_range(u32_t busaddr, u32_t size);
		static int do_pm_notify(message *m);
		static void report_exceptions(void);	

3	**/drivers/at_wini/at_wini.h**	_PROTOTYPE(int main, (int argc, char *argv[])); #define VERBOSE 0 /* display identify messages during boot */ #define ENABLE_ATAPI 1 /* add ATAPI cd-rom support to driver */	FW
3	**/drivers/at_wini/at_wini.c**	Этот файл содержит зависящую от устройства часть драйвера для контроллера винчестера IBM-AT. (Written by Adri Koppes.) Точка входа: at_winchester_task:	FW
1	**/drivers/audio/AC97.h**	Это копия основного кэша памяти конфигурационных регистров кодека ac97. См. Intel's Audio Codec 97 standard (rev2.3) для дополнительной информации.	FW
3	**/drivers/audio/es1370/ak4531.h**	Содержит прототипы функций.	FW

3	**/drivers/audio/es1370/ak4531.c**	Содержит функции:
		
		int ak4531_finished(void);
		int ak4531_write (u8_t address, u8_t data) ;
		int ak4531_init(u16_t base, u16_t status_reg, u16_t bit, u16_t poll);
		int ak4531_get_set_volume(struct volume_level *level, int flag) ;
		int set_volume(struct volume_level *level, int cmd_left, int cmd_right, int max_level);
			FW

3	**/drivers/audio/es1370/es1370.h**	Начинается с констант, определяющих ID ((?) идентификатор) производителя и самого устройства. Далее константы – индексы каналов и подустройств, команды PCI и определения регистров. Далее определяется тип aud_sub_dev_conf_t; и константы для него. А также тип DEV_STRUCT;	FW

pr.	*path*	*an.*	*aut*
3	**/drivers/audio/es1370/es1370.c**	Это основной фал звукового драйвера ES1370. Здесь нет функции main() - эта функция находится в общем драйвере DMA (непосредственного доступа к памяти). Весь этот драйвер обеспечивает интерфейс, определённый в файле audio/audio_fw.h (?). Все функции с префиксом 'drv_' декларированы в audio/audio_fw.h (?). Ниже декларированные прототипы функций представляют собой множество вспомогательных функций. Контроль над кодеком AK4531 обеспечивается в файле ak4531.c (*/drivers/audio/es1370/ak4531.c*)	FW
3	**/ drivers/audio/es1370/pci_helper.h**	Определяет только прототипы функций.	FW
3	**/ drivers/audio/es1370/pci_helper.c**	Содержит реальные тела функций, прототипы которых находятся в заголовочном файле. Функции определяют специфический для Minix3 способ работы с портами ввода-вывода устройств PCI. (Через функции sys_inb ; sys_inw; pci_inl; pci_outb; ...)	FW
2	**/drivers/audio/es1371/AC97.h**	Это копия основного кэша пмяти конфигурационных регистров кодека ac97. См. Intel's Audio Codec 97 standard (rev2.3) для дополнительной информации. Файл практически совпадает с **/drivers/audio/AC97.h**	FW

2	**/drivers/audio/es1371/AC97.c**	Файл начинается с прототипов функций аудио – миксера и управления режимами .
		
		Содержит функции:
		
		void set_src_sync_state (int state);
		int AC97_write (DEV_STRUCT * pCC, u16_t wAddr, u16_t wData);
		int AC97_read (DEV_STRUCT * pCC, u16_t wAddr, u16_t *data);
		int AC97_write_unsynced (DEV_STRUCT * pCC, u16_t wAddr, u16_t wData);
		int AC97_read_unsynced (DEV_STRUCT * pCC, u16_t wAddr, u16_t *data);
		AC97_init(DEV_STRUCT * pCC);
		void set_nice_volume(void);
		«глупый» код для того, чтобы установить канал DAC1на слышимый уровень, чтобы была возможность проверить его без использования миксера.
		int get_volume(u8_t *left, u8_t *right, int cmd);
		int set_volume(int left, int right, int cmd);
		convert(int left_in, int right_in, int max_in, int *left_out, int *right_out, int max_out, int swaplr);
		AC97_get_set_volume(struct volume_level *level, int flag);
		int AC97_get_volume(struct volume_level *level);
		int AC97_set_volume(struct volume_level *level) ;	

3	**/drivers/audio/es1371/codec.h**	Определения констант и декларации функций: /* Функция, информирующая модуль CODEC о том какой идентификатор производителя AC97 ((?)vendor ID) предполагать. */ void CodecSetVendorId (char *tbuf); /*Прототипы функций аудио – миксера и управления режимами CODEC */ int CodecRead (DEV_STRUCT * pCC, u16_t wAddr, u16_t *data);	FW

pr.	*path*	*an.*	*aut*
		int CodecWrite (DEV_STRUCT * pCC, u16_t wAddr, u16_t wData); void CodecSetSrcSyncState (int state); int CodecWriteUnsynced (DEV_STRUCT * pCC, u16_t wAddr, u16_t wData); int CodecReadUnsynced (DEV_STRUCT * pCC, u16_t wAddr, u16_t *data); /* Эта функция инициализации CODEC в значение по умолчанию. */ int CODECInit(DEV_STRUCT * pCC);	
3	**/drivers/audio/es1371/codec.c**	Начинается с констант определяющих режим синхронизации (SRC (?)) и время для timeout. Содержит реальные определения функций, прототипы которых даны в заголовочном файле.	FW
3	**/drivers/audio/es1371/es1371.h**	Начинается с констант, определяющих ID ((?) идентификатор) производителя и самого устройства. Далее константы – индексы каналов и подустройств, команды PCI и определения регистров. Далее определяется тип aud_sub_dev_conf_t; и константы для него. А также тип DEV_STRUCT;	FW
3	**/drivers/audio/es1371/es1371.c**	Драйвер для звуковой карты Ensoniq ES1371. Это основной фал звукового драйвера ES1370. Здесь нет функции main() - эта функция находится в общем драйвере DMA (непосредственного доступа к памяти). Весь этот драйвер обеспечивает интерфейс, определённый в файле audio/audio_fw.h (?). Все функции с префиксом 'drv_' декларированы в audio/audio_fw.h (?). Ниже декларированные прототипы функций представляют собой множество вспомогательных функций. Контроль над кодеком AC97 обеспечивается в файле AC97.c (*/drivers/audio/es1371/AC97.c*) Основан на драйвере ES1370 (?) который основан на драйвере ES1371	FW
2	**/drivers/audio/es1371/pci_helper.h**	Определяет только прототипы функций.	FW
2	**/drivers/audio/es1371/pci_helper.c**	Содержит реальные тела функций, прототипы которых находятся в заголовочном файле. Функции определяют специфический для Minix3 способ работы с портами ввода-вывода устройств PCI. (Через функции sys_inb ; sys_inw; pci_inl; pci_outb; ...)	FW
	/drivers/audio/es1371/sample_rate_converter.h	Содержит прототипы двух функций (src_init ; src_set_rate ;) и константы (препроцессора).	FW
	/drivers/audio/es1371/sample_rate_converter.c	Работает с устройством SRC (аппаратная реализация преобразования частоты семплирования чипом звуковой карты Ensoniq ES1371 (?))	FW

pr.	*path*	*an.*	*aut*

		Содержит функции:
		
		int src_init (DEV_STRUCT * DSP);
		int src_reg_read(DEV_STRUCT * DSP, u16_t reg, u16_t *data);
		int src_reg_write(DEV_STRUCT * DSP, u16_t reg, u16_t val);
		src_set_rate(DEV_STRUCT * DSP, char base, u16_t rate);	

	/drivers/audio/es1371/SRC.h	Содержит прототипы 4-х функций (SRCInit; SRCRegRead; SRCRegWrite; SRCSetRate;) и константы (препроцессора).	FW

	/drivers/audio/es1371/SRC.c	Начинается с констант:
		
		#define SRC_RATE 48000U
		#define reg(n) DSP->base + n
		
		Содержит функции:
		int SRCInit (DEV_STRUCT * DSP);
		int SRCRegRead(DEV_STRUCT * DSP, u16_t reg, u16_t *data);
		int SRCRegWrite(DEV_STRUCT * DSP, u16_t reg, u16_t val);
		void SRCSetRate(DEV_STRUCT * DSP, char base, u16_t rate);
			FW

	/drivers/audio/es1371/wait.h	Процедуры ожидания общего назначения. int WaitBitb (int paddr, int bitno, int state, long tmout); int WaitBitw (int paddr, int bitno, int state, long tmout); int WaitBitd (int paddr, int bitno, int state, long tmout); int MemWaitw (unsigned int volatile *gaddr, int bitno, int state, long tmout);	FW
	/drivers/audio/es1371/wait.c	Содержит реализации функций, прототипы которых приведены в заголовочном файле.	FW
6	**/drivers/audio/framework/audio_fw.h**	Содержит прототипы функций, выражающие некий стандартный интерфейс аудио-драйвера операционной системы Minix3 (?). Даны прототипы функций и определения типов данных.	FW
6	**/drivers/audio/framework/audio_fw.c**	Этот файл содержит стандартный драйвер для аудио устройств (Он используется для драйверов конкретных звуковых карт, расположенных в соседних каталогах (ES1370, ES1371, SB16, ...)). Он поддерживает двойной буфер непосредственного доступа к памяти (DMA) и может быть сконфигурирован так, чтобы использовать дополнительное пространство буферов не считая буферов непосредственного доступа к памяти. Этот драйвер также поддерживает подустройства, которые могут быть открыты и закрыты независимо. Драйвер поддерживает следующие операции: m_type DEVICE IO_ENDPT COUNT POSITION ADRRESS * ---------- * | DEV_OPEN | device | proc nr | | | | * |----------+---------+---------+---------+---------+---------| * | DEV_CLOSE | device | proc nr | | | | * |----------+---------+---------+---------+---------+---------| * | DEV_READ_S | device | proc nr | bytes | | buf ptr | * |----------+---------+---------+---------+---------+---------| * | DEV_WRITE_S | device | proc nr | bytes | | buf ptr | * |----------+---------+---------+---------+---------+---------| * | DEV_IOCTL_S | device | proc nr |func code| | buf ptr | * |----------+---------+---------+---------+---------+---------| * | DEV_STATUS | | | | | | * |----------+---------+---------+---------+---------+---------| * | HARD_INT | | | | | | * |----------+---------+---------+---------+---------+---------| * | SIG_STOP | | | | | | * ----------	FW

pr.	*path*	*an.*	*aut*
		Файл содержит одну точку входа: main: основная точка входа когда драйвер запускается... (?)	
3	**/drivers/audio/sb16/mixer.h**	Содержит прототипы 4-х функций (mixer_init; mixer_ioctl; mixer_set; mixer_get;).	FW
3	**/drivers/audio/sb16/mixer.c**	Начинается с прототипов 3-х внутренних функций (get_set_volume, get_set_input, get_set_output). Далее – реализация функций, прототипы которых находятся в заголовочном файле и вышеназванных 3-х .	FW

3	**/drivers/audio/sb16/sb16.h**	Содержит множество определений препроцессора (в основном – это различные константы).
		
		В конце прототипы 3-х функций:
		
		_PROTOTYPE(int dsp_command, (int value));
		_PROTOTYPE(int sb16_inb, (int port));
		_PROTOTYPE(void sb16_outb, (int port, int value));
			FW

5	**/drivers/audio/sb16/sb16.c**	Драйвер для звуковой карты SB16 ISA. Реализует интерфейс audio/audio_fw.h Содержит также множество функций, с префиксом 'dsp_'	FW
3	**/drivers/bios_wini/bios_wini.c**	Этот файл содержит «независимую от устройств» часть длайвера жёсткого диска, который использует ROM BIOS. Он выполняет вызов и ждёт, пока произойдёт пересылка (данных). Он не управляется пррываниями и, поэтому, будет (*) иметь слабую производительность. Преимущество его в том, что он будет работать практически на любом PC, XT, 386, PS/2 и их клоне (также при помощи этого драйвера можно обеспечить запуск и работу Minix3 с usb диска, если загрузка с usb диска поддерживается BIOS компьютера). Демонстрационный диск использует этот драйвер (скорее всего это было давно и уже неправда (?)). Предполагается, что все пользователи MINIX сначала опробуют другие драйверы, и будут использовать данный лишь как последний шанс, когда все остальные окажутся неработоспособными (на конкретном компьютере). (*) Производительность составляет примерно 10% от AT драйвера для чтения и записи на 2:1 перемежающимся диске ((?) 2:1 interleaved disk) , ((?) it will be DMA_BUF_SIZE bytes * per revolution for a minimum of 60 kb/s for writes to 1:1 disks.) Файл содержит одну точку входа: bios_winchester_task: основной вход при запуске драйвера. (На самом деле – это функция main() !!!) PUBLIC int main()	FW
3	**/drivers/dp8390/3c503.h**	Драйвер разделяемой памяти для сетевой карты Etherlink II. Содержит константы и макросы препроцессора.	FW
3	**/drivers/dp8390/3c503.c**	Драйвер разделяемой памяти для сетевой карты Etherlink II.	FW

pr.	*path*	*an.*	*aut*

		Использованы :
		пакет TNET (by M. Ostrowski);
		драйвер для Linux (D. Becker);
		драйвер Crynwr 3c503;
		драйвер Amoeba.
		
		Он работает в режиме разделяемой памяти и (должен быть) использован вместе с драйвером устройств, основанных на NS 8390 карт для Minix. Программируемый ввод/вывод также может быть использован, но он имеет низкую производительность.
		
		Содержит функции:
		static void el2_init(dep);
		Инициализирует аппаратное обеспечение и структуры данных.
		static void el2_stop(dep);
		Останавливает карты посредством запрещения прерываний.
		int el2_probe(dep);
		Проверяет наличие карты EtherLink II. Инициализирует адресацию памяти если карта обнаружена.
		static void milli_delay(unsigned long millis);	

3	**/drivers/dp8390/dp8390.h**	National Semiconductor DP8390 Network Interface Controller. Содержит константы препроцессора, макросы, определения типов, прототипы функций и снова определения типов. Назначения констант и структур довольно хорошо прокомментированы.	FW
3	**/drivers/dp8390/dp8390.c**	Этот файл содержит драйвер сетевого утройства для сетевых карт, основанных на NS dp8390. Допустимые сообщения и их параметры: * m_type DL_PORT DL_PROC DL_COUNT DL_MODE DL_ADDR DL_GRANT * |-----------+---------+---------+---------+---------+---------+---------| * | HARDINT | | | | | | | * |-----------|---------|---------|---------|---------|---------|---------| * | DL_WRITE | port nr | proc nr | count | mode | address | | * |-----------|---------|---------|---------|---------|---------|---------| * | DL_WRITEV | port nr | proc nr | count | mode | address | | * |-----------|---------|---------|---------|---------|---------|---------| * | DL_WRITEV_S| port nr | proc nr | count | mode | | grant | * |-----------|---------|---------|---------|---------|---------|---------| * | DL_READ | port nr | proc nr | count | | address | | * |-----------|---------|---------|---------|---------|---------|---------| * | DL_READV | port nr | proc nr | count | | address | | * |-----------|---------|---------|---------|---------|---------|---------| * | DL_READV_S | port nr | proc nr | count | | | grant | * |-----------|---------|---------|---------|---------|---------|---------| * | DL_CONF | port nr | proc nr | | mode | address | | * |-----------|---------|---------|---------|---------|---------|---------| * | DL_GETSTAT | port nr | proc nr | | | address | | * |-----------|---------|---------|---------|---------|---------|---------| * |DL_GETSTAT_S| port nr | proc nr | | | | grant | * |-----------|---------|---------|---------|---------|---------|---------| * | DL_STOP | port_nr | | | | | | * |-----------|---------|---------|---------|---------|---------|---------| * Посылаемые сообщения: * * m-type DL_PORT DL_PROC DL_COUNT DL_STAT DL_CLCK * |------------+----------+---------+----------+---------+---------| * |DL_TASK_REPLY| port nr | proc nr | rd-count | err|stat| clock | * |------------+----------+---------+----------+---------+---------|	FW

pr.	*path*	*an.*	*aut*
		* * m_type m3_i1 m3_i2 m3_ca1 * |-------------+---------+-----------+---------------| * |DL_CONF_REPLY| port nr | last port | ethernet addr | * |-------------+---------+-----------+---------------| *	
5	**/drivers/dp8390/local.h**	Содержит прототипы важнейших функций, сгруппированные по файлам.	FW
3	**/drivers/dp8390/ne2000.h**	Содержит небольшое количество констант и макросов препроцессора.	FW

3	**/drivers/dp8390/ne2000.c**	Драйвер для сетевой карты ne2000. Этот файл содержит только специфический для ne2000 код, остальное находится в dp8390.c .
		
		Содержит функции:
		int ne_probe(dep);
		void ne_init(dep);
		static int test_8(dep, pos, pat);
		static int test_16(dep, pos, pat);
		static void ne_stop(dep);
		static void milli_delay(unsigned long millis);
			FW

3	**/drivers/dp8390/rtl8029.h**	Содержит всего одну константу #define CR_PS_P3 0xC0 и два макроса: #define inb_reg3(dep, reg) (inb (dep->de_dp8390_port+reg)) #define outb_reg3(dep, reg, data) (outb(dep->de_dp8390_port+reg, data))	FW
3	**/drivers/dp8390/rtl8029.c**	Инициализация сетевой карты, основанной на PCI DP8390 .	FW
3	**/drivers/dp8390/wdeth.h**	Western Digital Ethercard Plus, or WD8003E Содержит константы препроцессора, макросы. #define inb_we(dep, reg) (inb(dep->de_base_port+reg)) #define outb_we(dep, reg, data) (outb(dep->de_base_port+reg, data))	FW

3	**/drivers/dp8390/wdeth.c**	Содержит в.т.ч. функции:
		
		_PROTOTYPE(static void we_init, (dpeth_t *dep));
		_PROTOTYPE(static void we_stop, (dpeth_t *dep));
		_PROTOTYPE(static int we_aliasing, (dpeth_t *dep));
		_PROTOTYPE(static int we_interface_chip, (dpeth_t *dep));
		_PROTOTYPE(static int we_16bitboard, (dpeth_t *dep));
		_PROTOTYPE(static int we_16bitslot, (dpeth_t *dep));
		_PROTOTYPE(static int we_ultra, (dpeth_t *dep));
			FW

5	**/drivers/dpeth/README**	Аннотация к подсистеме. Примерный перевод (в сокращении): « Это мой вариант реализации сетевого задания для Minix. Первоначальной моей целью была поддержка сетевой карты 3c501 (Etherlink) однако эти карты были настолько нестабильны, что не было смысла их использовать, разве что для обучения тому, как писать драйвер. Когда у меня появилась сетевая карта 3c509b (Etherlink III) уже было проще написать код для их поддержки. Код Minix 'dp8390.c' был слишком специфическим для множества карт National, таким образом я максимально убрал специфический для чипа код, для того, чтобы создать общее задание, которое, как я надеюсь, применимо для значительно большего числа сетевых карт.	FW

pr.	*path*	*an.*	*aut*
		»	
3	**/drivers/dpeth/3c501.h**	Описание интерфейса для сетевых карт 3Com Etherlink. Содержит константы и макросы препроцессора. Макросы: #define inb_el1(dep,reg) (inb(dep->de_base_port+(reg))) #define inw_el1(dep,reg) (inw(dep->de_base_port+(reg))) #define outb_el1(dep,reg,data) (outb(dep->de_base_port+(reg),data)) #define outw_el1(dep,reg,data) (outw(dep->de_base_port+(reg),data))	FW

3	**/drivers/dpeth/3c501.c**	Этот файл содержит специфическую поддержку драйвера сетевого устройства для сетевой карты 3Com Etherlink (3c501). Это очень старая карта и её производительность очень низкая для современного сетевого окружения.
		
		Содержит функции:
		static void el1_getstats(dpeth_t * dep);
		static void el1_reset(dpeth_t * dep);
		static void el1_dumpstats(dpeth_t * dep);
		static void el1_mode_init(dpeth_t * dep);
		static void el1_recv(dpeth_t * dep, int from, int size);
		static void el1_send(dpeth_t * dep, int from_int, int pktsize);
		static void el1_stop(dpeth_t * dep);
		static void el1_interrupt(dpeth_t * dep);
		static void el1_init(dpeth_t * dep);
		PUBLIC int el1_probe(dpeth_t * dep);
			FW

3	**/drivers/dpeth/3c503.h**	Описание интерфейса для сетевых карт 3Com Etherlink. Содержит константы и макросы препроцессора. Макросы: #define inb_el2(dep,reg) (inb((dep)->de_base_port+(reg))) #define outb_el2(dep,reg,data) (outb((dep)->de_base_port+(reg),(data)))	FW

3	**/drivers/dpeth/3c503.c**	Драйвер для сетевой карты Etherlink II. Работает в режиме разделяемой памяти. Программируемый ввод/вывод также может быть использован, однако даёт низкую производительность. Этот файл содержит только специфический для карты код, остальная часть находится в 8390.c . Код специфичен только для ISA bus.
		
		Содержит функции:
		static void el2_init(dpeth_t * dep);
		static void el2_stop(dpeth_t * dep);
		int el2_probe(dpeth_t * dep);
			FW

3	**/drivers/dpeth/3c509.h**	Описание интерфейса для сетевых карт 3Com Etherlink III. Содержит константы и макросы препроцессора. Макросы: /* EL3 access macros */ #define inb_el3(dep,reg) (inb((dep)->de_base_port+(reg))) #define inw_el3(dep,reg) (inw((dep)->de_base_port+(reg))) #define outb_el3(dep,reg,data) (outb((dep)->de_base_port+(reg),(data))) #define outw_el3(dep,reg,data) (outw((dep)->de_base_port+(reg),(data))) #define SetWindow(win) \ outw(dep->de_base_port+REG_CmdStatus,CMD_SelectWindow|(win))	FW
3	**/drivers/dpeth/3c509.c**	Этот файл содержит специфическую поддержку драйвера сетевого	FW

pr.	*path*	*an.*	*aut*

		устройства для сетевой карты 3Com Etherlink III (3c509).
		
		ВНИМАНИЕ: Эта карта должна быть установлена с отключенным PnP и с установленной базой ввода/вывода и вектором прерываний (IRQ). Драйвер только для ISA bus.
		
		Содержит функции:
		static void el3_update_stats(dpeth_t * dep);
		static void el3_getstats(dpeth_t * dep);
		static void el3_dodump(dpeth_t * dep);
		static void el3_rx_mode(dpeth_t * dep);
		static void el3_reset(dpeth_t * dep);
		static void el3_write_fifo(dpeth_t * dep, int pktsize);
		static void el3_recv(dpeth_t *dep, int fromint, int size);
		static void el3_rx_complete(dpeth_t * dep);
		static void el3_send(dpeth_t * dep, int from_int, int count);
		static void el3_close(dpeth_t * dep);
		static void el3_interrupt(dpeth_t * dep);
		static unsigned el3_read_eeprom(port_t port, unsigned address);
		static void el3_read_StationAddress(dpeth_t * dep);
		static void el3_open(dpeth_t * dep);
		static unsigned short el3_checksum(port_t port);
		static void el3_write_id(port_t port);
		PUBLIC int el3_probe(dpeth_t * dep);	

3	**/drivers/dpeth/8390.h**	Контроллер сетевого интерфейса National Semiconductor NS 8390. Содержит также описание типа: dp_rcvhdr_t; А также прототип функции: void ns_init(dpeth_t *);	FW

3	**/drivers/dpeth/8390.c**	Этот файл содержит драйвер сетевого устройства для сетевых карт, оснащённых чипом National Semiconductor NS 8390. Он (этот файл) ассоциирован со специфическим для карты кодом драйвера. Переписан заново из драйвера dp8390.c для Minix 2.0.0 для того, чтобы выделить общие функции для NS 8390.
		
		Содержит функции:
		static void ns_rw_setup(dpeth_t *dep, int mode, int size, u16_t offset);
		static void ns_start_xmit(dpeth_t * dep, int size, int pageno);
		static void mem_getblock(dpeth_t *dep, u16_t offset, int size, void *dst);
		static void mem_nic2user(dpeth_t * dep, int pageno, int pktsize);
		static void mem_user2nic(dpeth_t *dep, int pageno, int pktsize);
		static void pio_getblock(dpeth_t *dep, u16_t offset, int size, void *dst);
		static void pio_nic2user(dpeth_t *dep, int pageno, int pktsize);
		static void pio_user2nic(dpeth_t *dep, int pageno, int pktsize);
		static void ns_stats(dpeth_t * dep);
		static void ns_dodump(dpeth_t * dep);
		static void ns_reinit(dpeth_t * dep);
		static void ns_send(dpeth_t * dep, int from_int, int size);
		static void ns_reset(dpeth_t * dep);
		static void ns_recv(dpeth_t *dep, int fromint, int size);
		static void ns_interrupt(dpeth_t * dep);
		void ns_init(dpeth_t * dep);
		static void dp_pio16_user2nic(dpeth_t *dep, int pageno, int pktsize);
		static void dp_pio16_nic2user(dpeth_t * dep, int nic_addr, int count);
			FW

3	**/drivers/dpeth/devio.c**	Этот файл содержит процедуры чтения/записи из/в регистров устройства.	FW

pr.	*path*	*an.*	*aut*
		Функции: static void warning(const char *type, int err); PUBLIC unsigned int inb(unsigned short port); PUBLIC unsigned int inw(unsigned short port); PUBLIC void insb(unsigned short int port, int proc_nr, void *buffer, int count); PUBLIC void insw(unsigned short int port, int proc_nr, void *buffer, int count); PUBLIC void outb(unsigned short port, unsigned long value); PUBLIC void outw(unsigned short port, unsigned long value); PUBLIC void outsb(unsigned short port, int proc_nr, void *buffer, int count); PUBLIC void outsw(unsigned short port, int proc_nr, void *buffer, int count);	
5	**/drivers/dpeth/dp.h**	Описание интерфейса для драйвера сетевого устройства. Содержит константы для включения поддержки конкретных сетевых карт. Содержит определения типов: m_hdr_t; /* Заголовок для поддержки буеров*/ buff_t; /* Получить/послать заголовок буфера*/ iovec_dat_s_t; dpeth_t; /* Основная структура сетевого устройства */ Определения функций, сгруппированные по файлам (содержащим их реализации). Стандартный интегрирующий файл подсистемы подобного рода.	FW
4	**/drivers/dpeth/dp.c**	Этот файл содержит основное задание драйвера сетевого устройства. Он должен быть интегрирован со специфическим для карты драйвером. Переписан заново, но основан на драйвере сетевого устройства для Minix 2.0.0 (dp8390.c), в котором удалён специфический для карты код. Он должен работать (Я надеюсь) с драйвером любой карты. Допустимые сообщения: ** m_type DL_PORT DL_PROC DL_COUNT DL_MODE DL_ADDR ** +------------+---------+---------+--------+-------+---------+ ** | NOTIFY from HARDWARE, CLOCK, TTY, RS, PM, SYSTEM | ** +------------+---------+---------+--------+-------+---------+ ** | HARD_STOP | | | | | | ** +------------+---------+---------+--------+-------+---------+ ** | DL_WRITE | port nr | proc nr | count | mode | address | (3) ** +------------+---------+---------+--------+-------+---------+ ** | DL_WRITEV | port nr | proc nr | count | mode | address | (4) ** +------------+---------+---------+--------+-------+---------+ ** | DL_READ | port nr | proc nr | count | | address | (5) ** +------------+---------+---------+--------+-------+---------+ ** | DL_READV | port nr | proc nr | count | | address | (6) ** +------------+---------+---------+--------+-------+---------+ ** | DL_CONF | port nr | proc nr | | mode | address | (7) ** +------------+---------+---------+--------+-------+---------+ ** | DL_STOP | port_nr | | | | | (8) ** +------------+---------+---------+--------+-------+---------+ ** | DL_GETSTAT | port nr | proc nr | | | address | (9) ** +------------+---------+---------+--------+-------+---------+ ** ** Посылаемые сообщения: **	FW

pr.	*path*	*an.*	*aut*
		** m-type DL_PORT DL_PROC DL_COUNT DL_STAT DL_CLCK ** +------------+---------+---------+--------+---------+---------+ ** |DL_TASK_REPL| port nr | proc nr |rd-count| err|stat| clock | (21) ** +------------+---------+---------+--------+---------+---------+ ** ** m_type m3_i1 m3_i2 m3_ca1 ** +------------+---------+---------+--------------+ ** |DL_CONF_REPL| port nr |last port| ethernet addr | (20) ** +------------+---------+---------+--------------+ **	
3	**/drivers/dpeth/ne.h**	Содержит небольшое число констант препроцессора и макросы: #define inb_ne(dep, reg) (inb(dep->de_base_port+reg)) #define outb_ne(dep, reg, data) (outb(dep->de_base_port+reg, data)) #define inw_ne(dep, reg) (inw(dep->de_base_port+reg)) #define outw_ne(dep, reg, data) (outw(dep->de_base_port+reg, data)) По всей видимости связан с сетевой картой NE2000.	FW

3	**/drivers/dpeth/ne.c**	Драйвер для сетевых карт серии NE*000 и их производных. Этот файл содержит только специфический для данной карты код.
		
		Код специфичен только для ISA bus.
		
		Содержит функции:
		static int ne_reset(dpeth_t * dep);
		static void ne_close(dpeth_t * dep);
		static void ne_init(dpeth_t * dep);
		PUBLIC int ne_probe(dpeth_t * dep);
			FW

3	**/drivers/dpeth/netbuff.c**	Этот файл содержит специфическую поддержку для буферизации сетевых пакетов. Содержит функции: PUBLIC void *alloc_buff(dpeth_t *dep, int size); PUBLIC void free_buff(dpeth_t *dep, void *blk); PUBLIC void init_buff(dpeth_t *dep, buff_t **tx_buff); PUBLIC void mem2user(dpeth_t *dep, buff_t *rxbuff); PUBLIC void user2mem(dpeth_t *dep, buff_t *txbuff);	FW
3	**/drivers/dpeth/wd.h**	Содержит множество констант препроцессора и макросы: #define inb_we(dep, reg) (inb(dep->de_base_port+reg)) #define outb_we(dep, reg, data) (outb(dep->de_base_port+reg, data)) Скорее всего связан с сетевой картой WDETH.	FW

3	**/drivers/dpeth/wd.c**	Драйвер для сетевых карт серии Ethercard (WD80x3) и их производных. Этот файл содержит только специфический для карты wd80x3 код.
		
		Содержит функции:
		int wdeth_probe(dep);
		static void we_init(dep);
		static void we_stop(dep);
		static int we_aliasing(dep);
		static int we_interface_chip(dep);
		static int we_16bitboard(dep);
		static int we_16bitslot(dep);
		static int we_ultra(dep);
			FW

3	**/drivers/floppy/floppy.h**	Всего-то: #include "../drivers.h" #include "../libdriver/driver.h" #include "../libdriver/drvlib.h"	FW

pr.	*path*	*an.*	*aut*
		_PROTOTYPE(void main, (void));	
3	**/drivers/floppy/floppy.c**	Этот файл содержит специфичную для устройства часть драйвера контроллера дискеты (FDC), использующего чип NEC PD765. Файл содержит две точки входа: floppy_task: функция : PUBLIC void main(); основной вход, когда запускается система.	FW
4	**/drivers/fxp/fxp.h**	Регистры и структуры данных контроллеров быстрой сети Intel 82557, 82558, 82559, 82550 и 82562	FW
4	**/drivers/fxp/fxp.c**	Этот файл содержит драйвер сетевого устройства для сетевых контроллеров (повышенного быстродействия) Intel 82557, 82558, 82559, 82550 и 82562. Допустимые сообщения: * m_type DL_PORT DL_PROC DL_COUNT DL_MODE DL_ADDR DL_GRANT * |-----------+---------+---------+---------+---------+---------+---------| * | HARDINT | | | | | | | * |-----------|---------|---------|---------|---------|---------| * | DL_WRITE | port nr | proc nr | count | mode | address | | * |-----------|---------|---------|---------|---------|---------| * | DL_WRITEV | port nr | proc nr | count | mode | address | | * |-----------|---------|---------|---------|---------|---------| * | DL_WRITEV_S| port nr | proc nr | count | mode | | grant | * |-----------|---------|---------|---------|---------|---------| * | DL_READ | port nr | proc nr | count | | address | | * |-----------|---------|---------|---------|---------|---------| * | DL_READV | port nr | proc nr | count | | address | | * |-----------|---------|---------|---------|---------|---------| * | DL_READV_S | port nr | proc nr | count | | | grant | * |-----------|---------|---------|---------|---------|---------| * | DL_CONF | port nr | proc nr | | mode | address | | * |-----------|---------|---------|---------|---------|---------| * | DL_GETSTAT | port nr | proc nr | | | address | | * |-----------|---------|---------|---------|---------|---------| * |DL_GETSTAT_S| port nr | proc nr | | | | grant | * |-----------|---------|---------|---------|---------|---------| * | DL_STOP | port_nr | | | | | | * |-----------+---------+---------+---------+---------+---------+---------| * * Посылаемые сообщения: * * m-type DL_PORT DL_PROC DL_COUNT DL_STAT DL_CLCK * |-------------+---------+---------+---------+---------+---------| * |DL_TASK_REPLY| port nr | proc nr | rd-count | err|stat| clock | * |-------------+---------+---------+---------+---------+---------| * * m_type m3_i1 m3_i2 m3_ca1 * |-------------+---------+-----------+---------------| * |DL_CONF_REPLY| port nr | last port | ethernet addr | * |-------------+---------+-----------+---------------| * * m_type DL_PORT DL_STAT * |------------|---------|-----------| * |DL_STAT_REPL| port nr | err | * |------------|---------|-----------|	FW

3	**/drivers/fxp/mii.h**	Определения для (MII) Независимого от Среды Интерфейса ((?) Media Independent (Ethernet) Interface).
		
		Содержит множество констант и два прототипа функций:
		_PROTOTYPE(void mii_print_stat_speed, (U16_t stat, U16_t extstat));
		_PROTOTYPE(void mii_print_techab, (U16_t techab));
			FW

3	**/drivers/fxp/mii.c**	Содержит функции (объявленные в заголовочном файле):	FW

pr.	*path*	*an.*	*aut*
		PUBLIC void mii_print_stat_speed(stat, extstat); PUBLIC void mii_print_techab(techab);	
3	**/drivers/lance/lance.h**	Содержит множество констант, определения типов: iovec_dat_t; ether_card_t; Всё довольно хорошо задукоментировано (на английском языке).	FW
3	**/drivers/lance/lance.c**	Этот файл содержит драйвер сетевого устройства для сетевых карт, основанных на AMD LANCE. Допустимые сообщения: * m_type DL_PORT DL_PROC DL_COUNT DL_MODE DL_ADDR DL_GRANT * |----------+---------+---------+---------+---------+---------+---------| * | HARDINT | | | | | | | * |----------|---------|---------|---------|---------|---------|---------| * | DL_WRITEV_S| port nr | proc nr | count | mode | | grant | * |----------|---------|---------|---------|---------|---------|---------| * | DL_READV_S | port nr | proc nr | count | | | grant | * |----------|---------|---------|---------|---------|---------|---------| * | DL_CONF | port nr | proc nr | | mode | address | | * |----------|---------|---------|---------|---------|---------|---------| * |DL_GETSTAT_S| port nr | proc nr | | | | grant | * |----------|---------|---------|---------|---------|---------|---------| * | DL_STOP | port_nr | | | | | | | * |----------|---------|---------|---------|---------|---------|---------| * *Посылаемые сообщения: * * m-type DL_POR T DL_PROC DL_COUNT DL_STAT DL_CLCK * |----------|---------|---------|---------|---------|---------| * |DL_TASK_REPL| port nr | proc nr | rd-count | err|stat| clock | * |----------|---------|---------|---------|---------|---------| * * m_type m3_i1 m3_i2 m3_ca1 * |----------+---------+----------+---------------| * |DL_CONF_REPL| port nr | last port | ethernet addr | * |----------|---------|----------|---------------|	FW
2	**/drivers/libdriver/driver.h**	Типы и константы, разделяемые общей и специфической для устройства частью. Типичное начало: #define _POSIX_SOURCE 1 #define _MINIX 1 #define _SYSTEM 1 И далее множество заголовочных файлов, прототипов функций, а также некоторые определения типов, констант и объявления переменных.	FW
2	**/drivers/libdriver/driver.c**	Этот файл содержит независимый от устройства интерфейс драйвера (устройства). Драйверы поддерживают следующие операции (используя формат сообщения m2): * m_type DEVICE IO_ENDPT COUNT POSITION ADRRESS * ---------- * | DEV_OPEN | device | proc nr | | | | * |----------+---------+---------+---------+---------+---------| * | DEV_CLOSE | device | proc nr | | | | * |----------+---------+---------+---------+---------+---------| * | DEV_READ | device | proc nr | bytes | offset | buf ptr | * |----------+---------+---------+---------+---------+---------| * | DEV_WRITE | device | proc nr | bytes | offset | buf ptr |	FW

pr.	*path*	*an.*	*aut*

		* |------------+---------+---------+---------+---------+---------|
		* | DEV_GATHER | device | proc nr | iov len | offset | iov ptr |
		* |------------+---------+---------+---------+---------+---------|
		* | DEV_SCATTER| device | proc nr | iov len | offset | iov ptr |
		* |------------+---------+---------+---------+---------+---------|
		* | DEV_IOCTL | device | proc nr |func code| | buf ptr |
		* |------------+---------+---------+---------+---------+---------|
		* | CANCEL | device | proc nr | r/w | | |
		* |------------+---------+---------+---------+---------+---------|
		* | HARD_STOP | | | | | |
		* |------------+---------+---------+---------+---------+---------|
		* | DEV_*_S | variants using safecopies of above |
		* ---------- 	
		*
		
		Файл содержит одну точку входа:
		
		driver_task:
		вызываемую зависимой от устройства точкой входа (заданием устройства).
		
		Содержит функции:
		PUBLIC void driver_task(dp);
		PUBLIC void init_buffer(void);
		int do_rdwt(dp, mp, safe);
		int do_vrdwt(dp, mp, safe);
		PUBLIC char *no_name();
		PUBLIC int do_nop(dp, mp);
		PUBLIC int nop_ioctl(dp, mp, safe);
		PUBLIC void nop_signal(dp, set);
		PUBLIC void nop_alarm(dp, mp);
		PUBLIC struct device *nop_prepare(device);
		PUBLIC void nop_cleanup();
		PUBLIC int nop_cancel(struct driver *dr, message *m);
		PUBLIC int nop_select(struct driver *dr, message *m);
		PUBLIC int do_diocntl(dp, mp, safe);
		PUBLIC int mq_queue(message *m);	

2	**/drivers/libdriver/drvlib.h**	Определения драйвера устройства IBM. Начало: #include <ibm/partition.h> _PROTOTYPE(void partition, (struct driver *dr, int device, int style, int atapi)); /* BIOS parameter table layout. */ и.т.д.	FW
2	**/drivers/libdriver/drvlib.c**	Вспомогательные функции драйвера устройства IBM. Точка входа: partition: размечает диск в соответствии с записью в таблице партиций на нём. (?) /* Extended partition? */ #define ext_part(s) ((s) == 0x05 || (s) == 0x0F) Содержит функции:	FW

pr.	*path*	*an.*	*aut*

		PUBLIC void partition(dp, device, style, atapi);
		void extpartition(dp, extdev, extbase);
		int get_part_table(dp, device, offset, table);
		void sort(table);	

2	**/drivers/libdriver/mq.c**	Содержит
		
		#define MQ_SIZE 128
		
		PRIVATE mq_t mq_list[MQ_SIZE];
		PRIVATE mq_t *mq_freelist;
		
		А также функции:
		void mq_init();
		mq_t *mq_get();
		void mq_free(mq);
			FW

2	**/drivers/libdriver_asyn/driver.h**	Типы и константы, разделяемые общей и специфической для устройства частью. Типичное начало: #define _POSIX_SOURCE 1 #define _MINIX 1 #define _SYSTEM 1 И далее множество заголовочных файлов, прототипов функций, а также некоторые определения типов, констант и объявления переменных.	FW
2	**/drivers/libdriver_asyn/driver.c**	Этот файл содержит независимый от устройства интерфейс драйвера (устройства). Драйверы поддерживают следующие операции (используя формат сообщения m2): * m_type DEVICE IO_ENDPT COUNT POSITION ADRRESS * ---------- * | DEV_OPEN | device | proc nr | | | | * |----------+---------+---------+---------+---------+---------| * | DEV_CLOSE | device | proc nr | | | | * |----------+---------+---------+---------+---------+---------| * | DEV_READ | device | proc nr | bytes | offset | buf ptr | * |----------+---------+---------+---------+---------+---------| * | DEV_WRITE | device | proc nr | bytes | offset | buf ptr | * |----------+---------+---------+---------+---------+---------| * | DEV_GATHER | device | proc nr | iov len | offset | iov ptr | * |----------+---------+---------+---------+---------+---------| * | DEV_SCATTER| device | proc nr | iov len | offset | iov ptr | * |----------+---------+---------+---------+---------+---------| * | DEV_IOCTL | device | proc nr |func code| | buf ptr | * |----------+---------+---------+---------+---------+---------| * | CANCEL | device | proc nr | r/w | | | * |----------+---------+---------+---------+---------+---------| * | HARD_STOP | | | | | | * |----------+---------+---------+---------+---------+---------| * | DEV_*_S | variants using safecopies of above | * ---------- * Файл содержит одну точку входа: driver_task: вызываемую зависимой от устройства точкой входа (заданием устройства).	FW

pr.	*path*	*an.*	*aut*

		Содержит функции:
		PUBLIC void driver_task(dp);
		void init_buffer(void);
		int do_rdwt(dp, mp, safe);
		int do_vrdwt(dp, mp, safe);
		PUBLIC char *no_name();
		PUBLIC int do_nop(dp, mp);
		PUBLIC int nop_ioctl(dp, mp, safe);
		PUBLIC void nop_signal(dp, set);
		PUBLIC void nop_alarm(dp, mp);
		PUBLIC struct device *nop_prepare(device);
		PUBLIC void nop_cleanup();
		PUBLIC int nop_cancel(struct driver *dr, message *m);
		PUBLIC int nop_select(struct driver *dr, message *m);
		PUBLIC int do_diocntl(dp, mp, safe);
		PUBLIC int mq_queue(message *m);
		
		+
		
		PUBLIC int asynsend(dst, mp);	

2	/ **drivers/libdriver_asyn/drvlib.h**	Определения драйвера устройства IBM. Начало: #include <ibm/partition.h> _PROTOTYPE(void partition, (struct driver *dr, int device, int style, int atapi)); /* BIOS parameter table layout. */ и.т.д.	FW

2	**/drivers/libdriver_asyn/drvlib.c**	Вспомогательные функции драйвера устройства IBM.
		
		Точка входа:
		
		partition:
		размечает диск в соответствии с записью в таблице партиций на нём. (?)
		
		/* Extended partition? */
		#define ext_part(s) ((s) == 0x05 || (s) == 0x0F)
		
		Содержит функции:
		
		PUBLIC void partition(dp, device, style, atapi);
		void extpartition(dp, extdev, extbase);
		int get_part_table(dp, device, offset, table);
		void sort(table);
			FW

2	**/drivers/libdriver_asyn/mq.c**	Содержит #define MQ_SIZE 128 PRIVATE mq_t mq_list[MQ_SIZE]; PRIVATE mq_t *mq_freelist;	FW

pr.	*path*	*an.*	*aut*

		А также функции:
		void mq_init();
		mq_t *mq_get();
		void mq_free(mq);	

3	**/drivers/log/log.h**	Константы: #define LOG_SIZE (50*1024) #define SUSPENDABLE 1 Структура: struct logdevice; Прототипы функций: do_new_kmess do_diagnostics log_append	FW

3	**/drivers/log/log.c**	Этот файл содержит драйвер для :
		/dev/klog - system log device
		(системного протоколирования, системных сообщений)
		
		Содержит функции:
		PUBLIC int main(void);
		char *log_name();
		struct device *log_prepare(device);
		int subwrite(struct logdevice *log, int count, int proc_nr, vir_bytes user_vir, size_t offset, int safe);
		PUBLIC void log_append(char *buf, int count);
		int subread(struct logdevice *log, int count, int proc_nr, vir_bytes user_vir, size_t offset, int safe);
		int log_transfer(proc_nr, opcode, position, iov, nr_req, safe);
		int log_do_open(dp, m_ptr);
		void log_geometry(entry);
		int log_cancel(dp, m_ptr);
		void log_signal(dp, set);
		int log_other(dp, m_ptr, safe);
		int log_select(dp, m_ptr);
			FW

3	**/drivers/log/diag.c**	Этот файл обеспечивает диагностический вывод, который напрямую направляется в драйвер LOG. Этот вывод (эти сообщения) могут быть либо сообщениями микроядра (объявляемые через SYS_EVENT с SIGKMESS во множестве сигналов), либо вывод (сообщений) другого системного процесса (объявляемые через сообщение DIAGNOSTICS). Этот файл содержит функции: **PUBLIC int do_new_kmess(from);** **PUBLIC int do_diagnostics(message *m, int safe);** Сервер (драйвер (?)) обеспечивает все диагностические сообщения от серверов и драйверов устройств. Он направляет сообщения драйверу TTY для показа их пользователю. Он также сохраняет копию в локальном буфере, чтобы таким образом эти сообщения могли бы просмотрены позже.	FW
3	**/drivers/memory/allocmem.c**	Содержит функцию: PUBLIC int allocmem(size, base);	FW
3	**/drivers/memory/imgrd.c**	Виртуальный диск, являющийся частью образа (загружаемого образа (?)). Всего то: #include <stddef.h>	FW

pr.	*path*	*an.*	*aut*

		#include "local.h"
		
		unsigned char imgrd[]=
		{
		#include "ramdisk/image.c"
		};
		
		size_t imgrd_size= sizeof(imgrd);	

3	**/drivers/memory/imgrd_s.s**	Содержит переменные: _imgrd: _imgrd_size: Первая определена довольно хитро: _imgrd: 0: #include "ramdisk/image.s" 1:	FW
4	**/drivers/memory/local.h**	Локальные определения и прототипы. Всего-то: extern unsigned char imgrd[]; extern size_t imgrd_size;	FW

3	**/drivers/memory/memory.c**	Этот файл содержит зависящую от устройства часть драйверов для специальных файлов:
		/dev/ram - виртуальный диск, расположенный в виртуальной памяти
		/dev/mem - абсолютная память
		/dev/kmem - виртуальная память микроядра
		/dev/null -
		/dev/boot - загрузочное устройство, загруженное из загрузочного образа
		/dev/zero - генератор потока нулевых байтов
		
		Содержит функции:
		PUBLIC int main(void);
		Основная программа. Инициализирует драйвер памяти и начинает основной цикл (стандартный файл системного процесса).
		char *m_name();
		struct device *m_prepare(device);
		int m_transfer(proc_nr, opcode, pos64, iov, nr_req, safe);
		int m_do_open(dp, m_ptr);
		int m_do_close(dp, m_ptr);
		void m_init();
		int m_ioctl(dp, m_ptr, safe);
		void m_geometry(entry);
			FW

2	**/drivers/memory/ramdisk/bintoc.c**	Преобразует (бинарный) файл в разделённый запятыми набор шестнадцатеричных значений, пригодных для инициализации символьного массива в C.
		
		Содержит функции:
		int main(int argc, char *argv[]);
		static void fatal(char *fmt, ...);
		static void usage(void);
		
		Этот файл является самостоятельной утилитой и вряд ли включается в драйвер (?).
			FW

3	**/drivers/memory/ramdisk/mtab**	/dev/imgrd / 3 rw	FW

pr.	*path*	*an.*	*aut*
3	**/drivers/memory/ramdisk/proto**	boot 175 400 d--755 0 0 bin d--755 0 0 at_wini ---755 0 0 at_wini bios_wini ---755 0 0 bios_wini cdprobe ---755 0 0 cdprobe dev2name ---755 0 0 dev2name floppy ---755 0 0 floppy loadramdisk ---755 0 0 loadramdisk newroot ---755 0 0 newroot pci ---755 0 0 pci sh ---755 0 0 sh service ---755 0 0 service sysenv ---755 0 0 sysenv $ sbin d--755 0 0 mfs ---755 0 0 mfs $ dev d--755 0 0 @DEV@ $ etc d--755 0 0 drivers.conf ---644 0 0 drivers.conf mtab ---644 0 0 mtab passwd ---644 0 0 passwd rc ---755 0 0 rc $ $	FW
3	**/drivers/memory/ramdisk/proto.sh**	#!/bin/sh PATH=/bin:/sbin:/usr/bin:/usr/sbin sed -n '1,/@DEV/p' <proto | grep -v @DEV@ (cd /dev ls -aln | grep '^[bc]' | egrep -v ' (fd1|fd0p|tcp|eth|ip|udp|tty[pq]|pty)' | grep -v 13, | \ sed -e 's/^[bc]/& /' -e 's/rw-/6/g' -e 's/r--/4/g' \ -e 's/-w-/2/g' -e 's/---/0/g' | \ awk '{ printf "\t\t%s %s--%s %d %d %d %d \n", $11, $1, $2, $4, $5, $6, $7; }') sed -n '/@DEV/,$p' <proto | grep -v @DEV@	FW
3	**/drivers/memory/ramdisk/rc**	#!/bin/sh set -e exec >/dev/log exec 2>/dev/log exec </dev/null /bin/service up /bin/pci -config /etc/drivers.conf /bin/service -c up /bin/floppy -config /etc/drivers.conf -dev /dev/fd0 if [X`/bin/sysenv bios_wini` = Xyes] then echo Using bios_wini. /bin/service -c up /bin/bios_wini -dev /dev/c0d0 else /bin/service -c up /bin/at_wini -dev /dev/c0d0 -config /etc/drivers.conf -label at_wini_0 /bin/service -c up /bin/at_wini -dev /dev/c1d0 -config /etc/drivers.conf -label at_wini_1 -args ata_instance=1 fi rootdev=`sysenv rootdev` || echo 'No rootdev?' rootdevname=`/bin/dev2name "$rootdev"` || { echo 'No device name for root device'; exit 1; } if ["`sysenv bin_img`" = 1] then bin_img="-i " fi if sysenv cdproberoot >/dev/null then echo echo 'Looking for boot CD. This may take a minute.' echo 'Please ignore any error messages.'	FW

pr.	*path*	*an.*	*aut*
		echo cddev=`cdprobe` || { echo 'No CD found'; exit 1; } export cddev echo "Loading ramdisk from ${cddev}p1" loadramdisk "$cddev"p1 elif ["$rootdevname" = "/dev/ram"] then ramimagedev=`sysenv ramimagedev` \|\| { echo 'ramimagedev not found'; exit 1; } ramimagename=`/bin/dev2name "$ramimagedev"` || { echo 'No device name for ramimagedev'; exit 1; } echo "Loading ramdisk from $ramimagename" loadramdisk "$ramimagename" fi echo "Root device name is $rootdevname" /bin/newroot $bin_img"$rootdevname" exec /bin/sh /etc/rc "$@"	
3	**/drivers/orinoco/INSTALL.txt**	Драйвер проверялся на Minix 3.1.3,(svn revision 2928: http://derelict.cs.vu.nl/images/minix3_1_3_ide_r2928.iso.bz2). Для установки драйвера наберите: #sh install_script Скрипт изменит, скомпилирует и установит в Minix 2 файла исходников: /usr/src/services/rs/service.c /usr/src/commands/dhcpd/devices.c (см. каталог docs для документации) Кроме того скрипт изменит 1 конфигурационный файл: /etc/drivers.conf (добавит запись для 'orinoco') и 1 файл скрипта (сценрия): /usr/etc/rc (добавит запись для цикла, чтобы запустить orinoco) Наконец скрипт скопирует драйверы (orinoco) в каталог драйверов, скомпилирует и установит их. Когда скрипт успешно завершит свою работу, файл inet.conf должен быть дополнен таким образом, чтобы Minix использовал драйвер orinoco. Для этого может быть добавлена прмерно такая запись: eth0 orinoco 0 {default;}; Последний шаг - это установка ключей essid и WEP для сети в загрузочном мониторе. Для этого, находясь в загрузочном мониторе, надо напечатать: essid=<essid> wep=<WEP key> save Ключ essid – это имя беспроводной сети. Ключ WEP – это строка из 13 ASCII символов, являющаяся ключом для беспроводной сети. Эта переменная не должна устанавливаться если беспроводная сеть не защищена посредством WEP. Если ключ essid не установлен, то сетевая карта будет пытаться присоединиться к случайной сети (just try connect a network quite at random). [N.B.: WPA не поддерживается] [N.B.: WEP похоже содержит ошибки. На моём рабочем месте, на VU не работал. Таким образом, если не работает, попробуйте без WEP.]	FW
3	**/drivers/orinoco/hermes.c**	Этот файл содержит низкоуровневые функции доступа к картам беспроводной сети, основанным на Prism. Портирован из Linux (?). Содержит функции: void milli_delay(unsigned int msecs); static int hermes_issue_cmd (hermes_t * hw, u16_t cmd, u16_t param0); void hermes_struct_init (hermes_t * hw, u32_t address, int io_space, int reg_spacing);	FW

pr.	*path*	*an.*	*aut*
		int hermes_cor_reset (hermes_t *hw); int hermes_init (hermes_t * hw); int hermes_docmd_wait (hermes_t * hw, u16_t cmd, u16_t parm0, hermes_response_t * resp); int hermes_allocate (hermes_t * hw, u16_t size, u16_t * fid); static int hermes_bap_seek (hermes_t * hw, int bap, u16_t id, u16_t offset); int hermes_bap_pread (hermes_t * hw, int bap, void *buf, unsigned len, u16_t id, u16_t offset); int hermes_bap_pwrite (hermes_t * hw, int bap, const void *buf, unsigned len, u16_t id, u16_t offset); int hermes_present (hermes_t * hw); int hermes_set_irqmask (hermes_t * hw, u16_t events); u16_t hermes_get_irqmask (hermes_t * hw); int hermes_read_ltv (hermes_t * hw, int bap, u16_t rid, unsigned bufsize, u16_t * length, void *buf); int hermes_write_ltv (hermes_t * hw, int bap, u16_t rid, u16_t length, const void *value); int hermes_write_wordrec (hermes_t * hw, int bap, u16_t rid, u16_t word); int hermes_read_wordrec (hermes_t * hw, int bap, u16_t rid, u16_t * word); void hermes_read_words (hermes_t * hw, int off, void *buf, unsigned count); void hermes_write_words (hermes_t * hw, int off, const void *buf, unsigned count); u16_t hermes_read_reg (hermes_t * hw, u16_t off); void hermes_write_reg (hermes_t * hw, u16_t off, u16_t val);	
3	**/drivers/orinoco/hermes.h**	Константы, структуры и прототипы, необходимые для низкоуровневого доступа к картам Prism. Файл hermes.h взят за основу для этого файла (имеется в виду информация о портировании? (?))	FW
2	**/drivers/orinoco/hermes_rid.h**	Этот файл содержит идентификаторы ресурсов (RID). Это элементы данных (блоки данных (?)), которые могут быть сконфигурированы в сетевой карте. Большинство из них не используются в этом драйвере на данный момент, и как вы увидите – здесь действительно много такого, что может быть сконфигирировано. Реально имеем константы препроцессора...	FW
3	**/drivers/orinoco/orinoco.h**	Этот файл содержит наиболее важную структуру для драйвера: **t_or** и некоторые конфигурационные определения (препроцессора).	FW
3	**/drivers/orinoco/orinoco.c**	Этот файл содержит драйвер беспроводного сетевого устройства для карт беспроводной сети, основанных на Prism. Допустимые сообщения и их параметры: * m_type DL_PORT DL_PROC DL_COUNT DL_MODE DL_ADDR DL_GRANT * |------------+----------+---------+----------+---------+---------+---------| * | HARDINT | | | | | | | * |------------|----------|---------|----------|---------|---------|---------| * | DL_WRITE | port nr | proc nr | count | mode | address | | * |------------|----------|---------|----------|---------|---------|---------| * | DL_WRITEV | port nr | proc nr | count | mode | address | | * |------------|----------|---------|----------|---------|---------|---------| * | DL_WRITEV_S| port nr | proc nr | count | mode | | grant | * |------------|----------|---------|----------|---------|---------|---------| * | DL_READ | port nr | proc nr | count | | address | | * |------------|----------|---------|----------|---------|---------|---------|	FW

pr.	*path*	*an.*	*aut*
		* | DL_READV | port nr | proc nr | count | | address | | * |---|---|---|---|---|---|---| * | DL_READV_S| port nr | proc nr | count | | | grant | * |---|---|---|---|---|---|---| * | DL_CONF | port nr | proc nr | | mode | address | | * |---|---|---|---|---|---|---| * | DL_GETSTAT | port nr | proc nr | | | address | | * |---|---|---|---|---|---|---| * |DL_GETSTAT_S| port nr | proc nr | | | | grant | * |---|---|---|---|---|---|---| * | DL_STOP | port_nr | | | | | | * |---|---|---|---|---|---|---| * Посылаемые сообщения: * * m_type DL_PORT DL_PROC DL_COUNT DL_STAT DL_CLCK * |---|---|---|---|---|---| * |DL_TASK_REPL| port nr | proc nr | rd-count | err|stat| clock | * |---|---|---|---|---|---| * * m_type m3_i1 m3_i2 m3_ca1 * |---|---|---|---| * |DL_CONF_REPL| port nr | last port | ethernet addr | * |---|---|---|---| * * m_type DL_PORT DL_STAT * |---|---|---| * |DL_STAT_REPL| port nr | err | * |---|---|---|	
5	**/drivers/pci/main.c**	Содержит функции: do_init, do_first_dev, do_next_dev, do_find_dev, do_ids, do_dev_name, do_dev_name_s, do_slot_name_s, do_set_acl, do_del_acl, do_reserve, do_attr_r8, do_attr_r16, do_attr_r32, do_attr_w8, do_attr_w16, do_attr_w32, do_rescan_bus, reply, find_acl а также: int main(void);	FW

3	**/drivers/pci/pci.h**	Содержит
		
		прототипы временных функций (из затем заменяет pci_intel.h):
		_PROTOTYPE(unsigned pci_inb, (U16_t port));
		_PROTOTYPE(unsigned pci_inw, (U16_t port));
		_PROTOTYPE(unsigned pci_inl, (U16_t port));
			FW

pr.	*path*	*an.*	*aut*

		_PROTOTYPE(void pci_outb, (U16_t port, U8_t value));
		_PROTOTYPE(void pci_outw, (U16_t port, U16_t value));
		_PROTOTYPE(void pci_outl, (U16_t port, U32_t value));
		
		Определения структур:
		
		struct pci_vendor
		struct pci_device
		struct pci_baseclass
		struct pci_subclass
		struct pci_intel_ctrl
		struct pci_isabridge
		struct pci_pcibridge
		
		Константы препроцессора:
		
		#define PCI_IB_PIIX 1 /* Intel PIIX compatible ISA bridge */
		#define PCI_IB_VIA 2 /* VIA compatible ISA bridge */
		#define PCI_IB_AMD 3 /* AMD compatible ISA bridge */
		#define PCI_IB_SIS 4 /* SIS compatible ISA bridge */
		
		#define PCI_PPB_STD 1 /* Standard PCI-to-PCI bridge */
		#define PCI_PPB_CB 2 /* Cardbus bridge */
		/* Still needed? */
		#define PCI_AGPB_VIA 3 /* VIA compatible AGP bridge */
		
		глобальные переменные,
		
		и прототипы функций – утилит (вспомогательных функций):
		pci_reserve2
		pci_release
		pci_first_dev_a
		pci_next_dev_a
		pci_attr_r8_s
		pci_attr_r32_s
		pci_slot_name_s
		pci_ids_s	

3	**/drivers/pci/pci.c**	Конфигурирует устройства для PCI bus Интересно начало: #define USER_SPACE 1 Определяет типы и объявляет глобальные переменные: pcibus[NR_PCIBUS]; pcidev[NR_PCIDEV]; Содержит множество определений функций.	FW
3	**/drivers/pci/pci_amd.h**	#define AMD_ISABR_FUNC 3 /* Registers are in function 3 */ #define AMD_ISABR_PCIIRQ_LEV 0x54 #define AMD_PCILEV_INTA 0x1 #define AMD_PCILEV_INTB 0x2 #define AMD_PCILEV_INTC 4x2 #define AMD_PCILEV_INTD 4x8 #define AMD_ISABR_PCIIRQ_ROUTE 0x56 #define AMD_PCIIRQ_INTA_MASK 0x000F #define AMD_PCIIRQ_INTB_MASK 0x00F0 #define AMD_PCIIRQ_INTC_MASK 0x0F00 #define AMD_PCIIRQ_INTD_MASK 0xF000	FW
3	**/drivers/pci/pci_intel.h**	#define PCII_CONFADD 0xCF8 #define PCIIC_CODE 0x80000000 #define PCIIC_BUSNUM_MASK 0xff0000	FW

pr.	*path*	*an.*	*aut*
		#define PCIIC_BUSNUM_SHIFT 16 #define PCIIC_DEVNUM_MASK 0xf800 #define PCIIC_DEVNUM_SHIFT 11 #define PCIIC_FUNCNUM_MASK 0x700 #define PCIIC_FUNCNUM_SHIFT 8 #define PCIIC_REGNUM_MASK 0xfc #define PCIIC_REGNUM_SHIFT 2 #define PCII_CONFDATA 0xCFC #define PCII_SELREG_(bus, dev, func, reg) \ (PCIIC_CODE | \ (((bus) << PCIIC_BUSNUM_SHIFT) & PCIIC_BUSNUM_MASK) | \ (((dev) << PCIIC_DEVNUM_SHIFT) & PCIIC_DEVNUM_MASK) | \ (((func) << PCIIC_FUNCNUM_SHIFT) & PCIIC_FUNCNUM_MASK) | \ ((((reg)/4) << PCIIC_REGNUM_SHIFT) & PCIIC_REGNUM_MASK)) #define PCII_UNSEL (0) #define PCII_RREG8_(bus, dev, func, reg) \ (pci_outl(PCII_CONFADD, PCII_SELREG_(bus, dev, func, reg)), \ pci_inb(PCII_CONFDATA+((reg)&3))) #define PCII_RREG16_(bus, dev, func, reg) \ (PCII_RREG8_(bus, dev, func, reg) | \ (PCII_RREG8_(bus, dev, func, reg+1) << 8)) #define PCII_RREG32_(bus, dev, func, reg) \ (PCII_RREG16_(bus, dev, func, reg) | \ (PCII_RREG16_(bus, dev, func, reg+2) << 16)) #define PCII_WREG8_(bus, dev, func, reg, val) \ (pci_outl(PCII_CONFADD, PCII_SELREG_(bus, dev, func, reg)), \ pci_outb(PCII_CONFDATA+((reg)&3), (val))) #define PCII_WREG16_(bus, dev, func, reg, val) \ (PCII_WREG8_(bus, dev, func, reg, (val)), \ (PCII_WREG8_(bus, dev, func, reg+1, (val) >> 8))) #define PCII_WREG32_(bus, dev, func, reg, val) \ (PCII_WREG16_(bus, dev, func, reg, (val)), \ (PCII_WREG16_(bus, dev, func, reg+2, (val) >> 16))) /* PIIX configuration registers */ #define PIIX_PIRQRCA 0x60 #define PIIX_IRQ_DI 0x80 #define PIIX_IRQ_MASK 0x0F /* PIIX extensions to the PIC */ #define PIIX_ELCR1 0x4D0 #define PIIX_ELCR2 0x4D1	
3	**/drivers/pci/pci_sys.h**	#define SIS_ISABR_IRQ_A 0x41 /* IRQA routing */ #define SIS_ISABR_IRQ_B 0x42 /* IRQB routing */ #define SIS_ISABR_IRQ_C 0x43 /* IRQC routing */ #define SIS_ISABR_IRQ_D 0x44 /* IRQD routing */ #define SIS_IRQ_DISABLED 0x80 #define SIS_IRQ_MASK 0x0F	FW
3	**/drivers/pci/pci_table.c**	Таблицы с распространителями PCI (vendor) и индетификаторами устройств. *Эта таблица при необходимости может быть дополнена.*	FW
3	**/drivers/pci/pci_via.h**	#define VIA_ISABR_EL 0x54 /* Edge or level triggered */ #define VIA_ISABR_EL_INTA 0x08 /* Edge (1) or level (0) */ #define VIA_ISABR_EL_INTB 0x04 #define VIA_ISABR_EL_INTC 0x02 #define VIA_ISABR_EL_INTD 0x01	FW

pr.	*path*	*an.*	*aut*
		#define VIA_ISABR_IRQ_R1 0x55 /* IRQ routing 1 */ #define VIA_ISABR_IRQ_INTD 0xf0 /* routing for INTD */ #define VIA_ISABR_IRQ_INT0 0x0f /* routing for INT0 */ #define VIA_ISABR_IRQ_R2 0x56 /* IRQ routing 2 */ #define VIA_ISABR_IRQ_INTA 0xf0 /* routing for INTA */ #define VIA_ISABR_IRQ_INTB 0x0f /* routing for INTB */ #define VIA_ISABR_IRQ_R3 0x57 /* IRQ routing 3 */ #define VIA_ISABR_IRQ_INTC 0xf0 /* routing for INTC */ #define VIA_ISABR_IRQ_INT1 0x0f /* routing for INT1 */ #define VIA_ISABR_IRQ_R4 0x58 /* IRQ routing 4 */ #define VIA_ISABR_IRQ_INT2 0x0f /* routing for INT2 */	
3	**/drivers/printer/printer.c**	Этот файл содержит драйвер принтера. Это совсем простой драйвер, поддерживающий только один принтер. Символы, которые посылаются в драйвер, распечатываются принтером сосем без каких-либо изменений. Допустимые сообщения и их параметры: * DEV_OPEN: initializes the printer * DEV_CLOSE: does nothing * HARD_INT: interrupt handler has finished current chunk of output * DEV_WRITE: a process wants to write on a terminal * CANCEL: terminate a previous incomplete system call immediately * * m_type TTY_LINE IO_ENDPT COUNT ADDRESS * |----------+---------+---------+---------+---------| * | DEV_OPEN | | | | | * |----------+---------+---------+---------+---------| * | DEV_CLOSE | | proc nr | | | * ---------- * | HARD_INT | | | | | * |----------+---------+---------+---------+---------| * | SYS_EVENT | | | | | * |----------+---------+---------+---------+---------| * | DEV_WRITE |minor dev| proc nr | count | buf ptr | * |----------+---------+---------+---------+---------| * | CANCEL |minor dev| proc nr | | | * ----------	FW
5	**/drivers/random/main.c**	Этот файл содержит зависящую от устройства часть драйвера для специального файла: /dev/random генератор случайных чисел. Определяет структуру, представляющую собой точку входа в этот драйвер (?): *PRIVATE struct driver r_dtab = {* *r_name, /* current device's name */* *r_do_open, /* open or mount */* *do_nop, /* nothing on a close */* *r_ioctl, /* specify ram disk geometry */* *r_prepare, /* prepare for I/O on a given minor device */* *r_transfer, /* do the I/O */* *nop_cleanup, /* no need to clean up */* *r_geometry, /* device "geometry" */* *nop_signal, /* system signals */* *r_random, /* get randomness from kernel (alarm) */* *nop_cancel,* *nop_select,* *NULL,* *NULL* *};* Содержит функции:	FW

pr.	*path*	*an.*	*aut*

		PUBLIC int main(void);
		char *r_name();
		struct device *r_prepare(device);
		int r_transfer(proc_nr, opcode, position, iov, nr_req, safe);
		int r_do_open(dp, m_ptr);
		void r_init();
		int r_ioctl(dp, m_ptr, safe);
		
		#define UPDATE(binnumber, bp, startitem, elems)
		
		void r_updatebin(int source, struct k_randomness_bin *rb);
		void r_random(dp, m_ptr);
		void r_geometry(entry);	

3	**/drivers/random/random.h**	Публичный интерфейс для генератора случайных чисел. *#define RND_TIMING 0* *#define RANDOM_SOURCES_INTERNAL 1* *#define TOTAL_SOURCES* *(RANDOM_SOURCES+RANDOM_SOURCES_INTERNAL)* *_PROTOTYPE(void random_init, (void)* *);* *_PROTOTYPE(int random_isseeded, (void)* *);* *_PROTOTYPE(void random_update, (int source, rand_t *buf, int count));* *_PROTOTYPE(void random_getbytes, (void *buf, size_t size));* *_PROTOTYPE(void random_putbytes, (void *buf, size_t size));*	FW
3	**/drivers/random/random.c**	Генератор случайных чисел. Этот генератор случайных чисел собирает данные из микроядра и «сжимает» эти данные в заготовку для генератора псевдо-случайных чисел. Реализует (в частности) функции, объявленные в заголовочном файле.	FW
3	**/drivers/random/sha2.h**	Портирован из $FreeBSD: src/sys/crypto/sha2/sha2.h,v 1.1.2.1 2001/07/03 Содержит константы, определения типов, а также прототипы функций вроде: ***void SHA256_Init*** ***void SHA256_Update*** ***void SHA256_Final*** ***char* SHA256_End*** ***char* SHA256_Data*** а также для SHA384_* , SHA512_* .	FW
3	**/drivers/random/sha2.c**	Портирован из $FreeBSD: src/sys/crypto/sha2/sha2.c,v 1.2.2.2 2002/03/05 В том числе даёт реализацию функций, объявленных в заголовочном файле.	FW
3	**/drivers/random/aes/rijndael.h**	rijndael-api.h - Rijndael encryption programming interface.	FW
3	**/drivers/random/aes/rijndael_alg.c**	Содержит функции: ***int rijndael_KeySched(word8 k[MAXKC][4], word8 W[MAXROUNDS+1][4][4], int ROUNDS);***	FW

pr.	*path*	*an.*	*aut*

		Вычисляет необходимые ключи ((?) round keys). Объём вычислений зависит от keyBits и blockBits.
		
		int rijndael_KeyEncToDec(word8 W[MAXROUNDS+1][4][4], int ROUNDS);
		
		int rijndael_Encrypt(const void *va, void *vb, word8 rk[MAXROUNDS+1][4][4], int ROUNDS);
		Кодирует (зашифровывает) один блок.
		
		int rijndaelEncryptRound(word8 a[4][4], word8 rk[MAXROUNDS+1][4][4], int ROUNDS, int rounds);
		
		int rijndael_Decrypt(const void *va, void *vb, word8 rk[MAXROUNDS+1][4][4], int ROUNDS);
		Разшифровывает (декодирует) один блок.
		
		int rijndaelDecryptRound(word8 a[4][4], word8 rk[MAXROUNDS+1][4][4], int ROUNDS, int rounds);	

3	**/ drivers/random/aes/rijndael_api.c**	/* rijndael-api.c - Rijndael encryption programming interface.
		
		Содержит функции:
		
		static void blockcpy(void *dst, const void *src); /* в двух экземплярах! */
		int rijndael_makekey(rd_keyinstance *key, size_t keylen, const void *keymaterial);
		ssize_t rijndael_ecb_encrypt(rd_keyinstance *key, const void *input, void *output, size_t length, void *dummyIV);
		ssize_t rijndael_ecb_decrypt(rd_keyinstance *key, const void *input, void *output, size_t length, void *dummyIV);
		ssize_t rijndael_cbc_encrypt(rd_keyinstance *key, const void *input, void *output, size_t length, void *IV);
		ssize_t rijndael_cbc_decrypt(rd_keyinstance *key, const void *input, void *output, size_t length, void *IV);
		ssize_t rijndael_cfb1_encrypt(rd_keyinstance *key, const void *input, void *output, size_t length, void *IV);
		ssize_t rijndael_cfb1_decrypt(rd_keyinstance *key, const void *input, void *output, size_t length, void *IV);
		ssize_t rijndael_cfb8_encrypt(rd_keyinstance *key, const void *input, void *output, size_t length, void *IV);
		ssize_t rijndael_cfb8_decrypt(rd_keyinstance *key, const void *input, void *output, size_t length, void *IV);
		ssize_t rijndael_pad(void *input, size_t length);
		ssize_t rijndael_unpad(const void *input, size_t length);
		void cipherEncryptUpdateRounds(rd_keyinstance *key, const void *input, void *output, int rounds);
		void cipherDecryptUpdateRounds(rd_keyinstance *key, const void *input, void *output, int rounds);
			FW

3	**/drivers/random/aes/rijndael-alg.h**	Содержит в основном прототипы функций.	FW
3	**/drivers/random/aes/rijndael-api.h**	rijndael-api.h - Rijndael encryption programming interface. Содержит в основном прототипы функций.	FW
2	**/ drivers/random/aes/word_i386.h**	Определения типов (в соответствии с архитектурой i386). ***typedef unsigned char byte;*** ***typedef unsigned char word8;*** ***typedef unsigned short word16;*** ***typedef unsigned word32;*** ***#define STRICT_ALIGN 1***	FW
3	**/drivers/random/aes/boxes.dat**	Содержит таблицы чисел, используемые в шифровании.	FW
3	**/drivers/readclock/readclock.c**	Читает часы реального времени. Изменён, чтобы стал драйвером пользовательского пространства.	FW

pr.	*path*	*an.*	*aut*

		Читает значение часов из 64-х байтного (?) участка памяти CMOS RAM, затем устанавливает системное время.
		
		Если байт идентификатора компьютера (machine ID) 0xFC или 0xF8, устройство /dev/mem существует и может быть открыто для чтения, и RTC не выдаёт сообщений об ошибках в CMOS RAM, то время читается из области памяти часов (clock RAM), указанной RTC.
		
		Значения памяти часов (clock RAM) расшифровываются и отправляются в структуру mktime, чтобы создать значение time_t, затем вызывается stime(2).
		
		...
		
		((?) *что-то больно сомнительный текст перевода получился* ...)
		
		Содержит функции:
		
		int main(int argc, char **argv);
		
		а также:
		
		void errmsg(char *s);
		void get_time(struct tm *t);
		int read_register(int reg_addr);
		void set_time(struct tm *t);
		void write_register(int reg_addr, int value);
		int bcd_to_dec(int n);
		int dec_to_bcd(int n);
		void usage(void);	

3	**/drivers/rtl8139/rtl8139.h**	Содержит только огромное количество констант препроцессора.	FW
3	**/drivers/rtl8139/rtl8139.c**	Этот файл содержит драйвер сетевого устройства для сетевых карт, основанных на rtl8139. Допустимые сообщения и их параметры: * m_type DL_PORT DL_PROC DL_COUNT DL_MODE DL_ADDR DL_GRANT * |----------+----------+---------+----------+---------+---------+---------| * | HARDINT | | | | | | | * |----------|----------|---------|----------|---------|---------|---------| * | DL_WRITE | port nr | proc nr | count | mode | address | | * |----------|----------|---------|----------|---------|---------|---------| * | DL_WRITEV | port nr | proc nr | count | mode | address | | * |----------|----------|---------|----------|---------|---------|---------| * | DL_WRITEV_S| port nr | proc nr | count | mode | | grant | * |----------|----------|---------|----------|---------|---------|---------| * | DL_READ | port nr | proc nr | count | | address | | * |----------|----------|---------|----------|---------|---------|---------| * | DL_READV | port nr | proc nr | count | | address | | * |----------|----------|---------|----------|---------|---------|---------| * | DL_READV_S| port nr | proc nr | count | | | grant | * |----------|----------|---------|----------|---------|---------|---------| * | DL_CONF | port nr | proc nr | | mode | address | | * |----------|----------|---------|----------|---------|---------|---------| * | DL_GETSTAT | port nr | proc nr | | | address | | * |----------|----------|---------|----------|---------|---------|---------| * |DL_GETSTAT_S| port nr | proc nr | | | | grant | * |----------|----------|---------|----------|---------|---------|---------| * | DL_STOP | port_nr | | | | | | | * |----------|----------|---------|----------|---------|---------|---------|	FW

pr.	*path*	*an.*	*aut*
		* Посылаемые сообщения: * * m_type DL_PORT DL_PROC DL_COUNT DL_STAT DL_CLCK * |------------|----------|---------|----------|---------|---------| * |DL_TASK_REPL| port nr | proc nr | rd-count | err|stat| clock | * |------------|----------|---------|----------|---------|---------| * * m_type m3_i1 m3_i2 m3_ca1 * |------------|---------|-----------|---------------| * |DL_CONF_REPL| port nr | last port | ethernet addr | * |------------|---------|-----------|---------------| * * m_type DL_PORT DL_STAT * |------------|---------|-----------| * |DL_STAT_REPL| port nr | err | * |------------|---------|-----------| * Содержит функции: static unsigned my_inb(U16_t port) ; static unsigned my_inw(U16_t port); static unsigned my_inl(U16_t port); static void my_outw(U16_t port, U16_t value); static void my_outl(U16_t port, U32_t value); int main(int argc, char *argv[]); а также: ***_PROTOTYPE(static void rl_init, (message *mp));*** ***_PROTOTYPE(static void rl_pci_conf, (void));*** ***_PROTOTYPE(static int rl_probe, (re_t *rep));*** ***_PROTOTYPE(static void rl_conf_hw, (re_t *rep));*** ***_PROTOTYPE(static void rl_init_buf, (re_t *rep));*** ***_PROTOTYPE(static void rl_init_hw, (re_t *rep));*** ***_PROTOTYPE(static void rl_reset_hw, (re_t *rep));*** ***_PROTOTYPE(static void rl_confaddr, (re_t *rep));*** ***_PROTOTYPE(static void rl_rec_mode, (re_t *rep));*** ***_PROTOTYPE(static void rl_readv, (message *mp, int from_int, int vectored));*** ***_PROTOTYPE(static void rl_readv_s, (message *mp, int from_int));*** ***_PROTOTYPE(static void rl_writev, (message *mp, int from_int, int vectored));*** ***_PROTOTYPE(static void rl_writev_s, (message *mp, int from_int));*** ***_PROTOTYPE(static void rl_check_ints, (re_t *rep));*** ***_PROTOTYPE(static void rl_report_link, (re_t *rep));*** ***_PROTOTYPE(static void mii_print_techab, (U16_t techab));*** ***_PROTOTYPE(static void mii_print_stat_speed, (U16_t stat, U16_t extstat));*** ***_PROTOTYPE(static void rl_clear_rx, (re_t *rep));*** ***_PROTOTYPE(static void rl_do_reset, (re_t *rep));*** ***_PROTOTYPE(static void rl_getstat, (message *mp));*** ***_PROTOTYPE(static void rl_getstat_s, (message *mp));*** ***_PROTOTYPE(static void rl_getname, (message *mp));*** ***_PROTOTYPE(static void reply, (re_t *rep, int err, int may_block));*** ***_PROTOTYPE(static void mess_reply, (message *req, message *reply));*** ***_PROTOTYPE(static void rtl8139_stop, (void));***	

pr.	*path*	*an.*	*aut*
		_PROTOTYPE(static void check_int_events, (void)); ***_PROTOTYPE(static int do_hard_int, (void));*** ***_PROTOTYPE(static void rtl8139_dump, (message *m));*** ***#if 0*** ***_PROTOTYPE(static void dump_phy, (re_t *rep));*** ***#endif*** ***_PROTOTYPE(static int rl_handler, (re_t *rep));*** ***_PROTOTYPE(static void rl_watchdog_f, (timer_t *tp));*** ***_PROTOTYPE(static void tell_dev, (vir_bytes start, size_t size, int pci_bus, int pci_dev, int pci_func));***	
3	**/drivers/sb16/sb16.h**	Содержит множество констант (определений препроцессора). Кроме того, макрос: ***/* Number of bytes you can DMA before hitting a 64K boundary: */*** ***#define dma_bytes_left(phys) *** ***((unsigned) (sizeof(int) == 2 ? 0 : 0x10000) - (unsigned) ((phys) & 0xFFFF))*** и прототипы функций: ***_PROTOTYPE(int mixer_set, (int reg, int data));*** ***_PROTOTYPE(int sb16_inb, (int port));*** ***_PROTOTYPE(void sb16_outb, (int port, int value));***	FW
3	**/drivers/sb16/sb16.c**	Содержит реализации функций, объявленных в заголовочном файле. В целом три простые и небольшие функции.	FW
5	**/drivers/sb16/sb16_dsp.c**	Этот файл содержит драйвер для процессора цифрового звука (DSP) звуковых карт SoundBlaster 16. Драйвер поддерживает следующие операции (используя формат сообщений m2): * m_type DEVICE IO_ENDPT COUNT POSITION ADRRESS * ---------- * | DEV_OPEN | device | proc nr | | | | * |----------+---------+---------+---------+---------+---------| * | DEV_CLOSE | device | proc nr | | | | * |----------+---------+---------+---------+---------+---------| * | DEV_READ | device | proc nr | bytes | | buf ptr | * |----------+---------+---------+---------+---------+---------| * | DEV_WRITE | device | proc nr | bytes | | buf ptr | * |----------+---------+---------+---------+---------+---------| * | DEV_IOCTL | device | proc nr |func code| | buf ptr | * ---------- Файл содержит одну точку входа: main: основная точка входа при запуске драйвера. Содержит также функции: main, dsp_open, dsp_close, dsp_ioctl, dsp_write, dsp_hardware, dsp_status, reply, init_buffer, dsp_init, dsp_reset, dsp_command, dsp_set_size, dsp_set_speed, dsp_set_stereo, dsp_set_bits, dsp_set_sign, dsp_dma_setup, dsp_setup. Интерес представляет также глобальная переменная:	FW

pr.	*path*	*an.*	*aut*
		PRIVATE int irq_hook_id; /* id of irq hook at the kernel */ - идентификатор ловушки прерывания (irq) в микроядре.	
5	**/drivers/sb16/sb16_mixer.c**	Этот файл содержит драйвер для миксера (аудио микшера) звуковой карты SoundBlaster 16. Драйвер поддерживает следующие операции (используя формат сообщений m2): * m_type DEVICE IO_ENDPT COUNT POSITION ADRRESS * ---------- * | DEV_OPEN | device | proc nr | | | | * |----------+---------+---------+---------+---------+---------| * | DEV_CLOSE | device | proc nr | | | | * |----------+---------+---------+---------+---------+---------| * | DEV_IOCTL | device | proc nr |func code| | buf_ptr | * ---------- Файл содержит одну точку входа: sb16mixer_task (main): основная точка входа при запуске драйвера. Содержит также функции: main, mixer_open, mixer_close, mixer_ioctl, mixer_get, get_set_volume, get_set_input, get_set_output.	FW
3	**/drivers/sb16/README**	Драйвер для Minix звуковой карты Sound Blaster 16 ISA . Инструкции по установке драйвера SB16 (Minix >= 3.0.7): - установите IRQ и адрес ввода/вывода (I/O) в файле sb16.h (по умолчанию: 7 и 220) - make install - MAKEDEV /dev/audio (if /dev/audio doesn't allready exist) - service up /usr/sbin/sb16_dsp -dev /dev/audio - service up /usr/sbin/sb16_mixer -dev /dev/mixer это всё... (вы можете включить последние 2 линии в файл /usr/etc/rc)	FW
3	**/drivers/ti1225/ti1225.h**	Содержит определение типа: struct csr /* CardBus Socket Registers */ А также множество констант (определений препроцессора): /* PCI attribute space registers */ /* csr_mask */ /* csr_present */ /* csr_control */	FW
3	**/drivers/ti1225/ti1225.c**	Определяет типы данных и глобальные переменные: struct port ... ports[NR_PORTS]; struct pcitab struct pcitab pcitab_ti[] PRIVATE char *progname; PRIVATE int debug; Содержит также функции: init, hw_init, map_regs, do_int, read_exca, do_outb, do_inb, ...	FW

pr.	*path*	*an.*	*aut*
		Точка входа: int main(int argc, char *argv[]);	
3	**/drivers/ti1225/i82365.h**	Содержит множество констант (определений препроцессора), связанных с i82365.	FW
5	**/drivers/tty/tty.h**	Терминалы. Интересно начало: ***#include <timers.h>*** ***#undef lock*** ***#undef unlock*** ***#define TTY_REVIVE 6767*** Содержит определения и макросы препроцессора, определения типов, в.т.ч. центрального типа: tty_t; Содержит прототипы функций, сгруппированные по файлам.	FW
5	**/drivers/tty/tty.c**	Этот файл содержит драйвер терминала, как ((?) both) для консоли IBM, так и для стандартного ((?) regular) терминала ASCII. Этот файл обеспечивает только независимую от устройства часть драйвера терминала (TTY), зависящие от устройства части находятся в файлах console.c, rs232.c, и.т.д. . Этот файл содержит две основные точки входа: tty_task(), PUBLIC int main(void); tty_wakeup(), и несколько вспомогательных ((?) minor) точек входа для использования кодом, зависимым от устройства. Не зависящая от устройства часть понимает «клавиатурный» ввод из зависимой от устройства части, производит обработку этого ввода (интерпретация специальных клавиш), и посылает ввод (в уже обработанном виде) в процесс, читающий из терминала (связанный в данный момент с данным терминалом) (TTY). Вывод на терминал (TTY) посылается на зависимую от устройства часть для обработки вывода и «показа на экране» ((?) "screen" display.). Обработка ввода производится устройством посредством вызова 'in_process' на вводимых символах (каждом (?)), обработка вывода может быть выполнена самим устройством посредством вызова 'out_process'. Терминал (TTY) заботится об очереди ввода,устройство – об очереди вывода. Если устройство получает внешний сигнал, вроде прерывания, тогда это вызывает tty_wakeup(). ((?) If a device receives an external signal, * like an interrupt, then it causes tty_wakeup() to be run by the CLOCK task * to, you guessed it, wake up the TTY to check if input or output can * continue. Допустимые сообщения и их параметры: * notify from HARDWARE: output has been completed or input has arrived * notify from SYSTEM : e.g., MINIX wants to shutdown; run code to * cleanly stop * DEV_READ: a process wants to read from a terminal	FW

pr.	*path*	*an.*	*aut*
		* DEV_WRITE: a process wants to write on a terminal * DEV_IOCTL: a process wants to change a terminal's parameters * DEV_OPEN: a tty line has been opened * DEV_CLOSE: a tty line has been closed * DEV_SELECT: start select notification request * DEV_STATUS: FS wants to know status for SELECT or REVIVE * CANCEL: terminate a previous incomplete system call immediately * * m_type TTY_LINE IO_ENDPT COUNT TTY_SPEKS ADDRESS * ---------- * | HARD_INT | | | | | | * |----------+---------+---------+---------+---------+---------| * | SYS_SIG | sig set | | | | | * |----------+---------+---------+---------+---------+---------| * | DEV_READ |minor dev| proc nr | count | | buf ptr | * |----------+---------+---------+---------+---------+---------| * | DEV_WRITE |minor dev| proc nr | count | | buf ptr | * |----------+---------+---------+---------+---------+---------| * | DEV_IOCTL |minor dev| proc nr |func code|erase etc| | * |----------+---------+---------+---------+---------+---------| * | DEV_OPEN |minor dev| proc nr | O_NOCTTY| | | * |----------+---------+---------+---------+---------+---------| * | DEV_CLOSE |minor dev| proc nr | | | | * |----------+---------+---------+---------+---------+---------| * | DEV_STATUS | | | | | | * |----------+---------+---------+---------+---------+---------| * | CANCEL |minor dev| proc nr | | | | * ---------- Содержит в.т.ч. функции: tty_timed_out, settimer, do_cancel, do_ioctl, do_open, do_close, do_read, do_write, do_select, do_status, in_transfer, tty_echo, rawecho, back_over, reprint, dev_ioctl, setattr, tty_icancel, tty_init, ...	
3	**/drivers/tty/console.c**	Код и данные для драйвера консоли IBM. Видео-контроллер 6845, используемый персональными компьютерами IBM PC разделяет собственную память с памятью центрального процессора (CPU) где-то в банке памяти 0xB0000. Для 6845 эта память заполнена 16-битными словами. Каждое слово содержит код клавиши в младшем байте (low byte) и так называемый байт атрибута в старшем байте (high byte) этого слова. Центральный процессор (CPU) непосредственно изменяет видео-память для воспроизведения символов, а также устанавливает два регистра видео контроллера 6845, которые определяют видео страницу (video origin) и позицию курсора. Видео страница (The video origin) – это место в видео памяти, где находится первый символ (верхний левый угол). Менять видео страницу (video origin) – это самый быстрый способ для прокрутки страницы. Некоторые видео адаптеры позволяют непрерывно изменять расположение видео страницы (video origin), однако для некоторых это невозможно, и поэтому видео память должна перемещаться. ((?) Some * video adapters wrap around the top of video memory, so the origin can * move without bounds. For other adapters screen memory must sometimes be * moved to reset the origin.) Все вычисления на видео памяти используют адресацию символов (вышеописанных 16-битных слов <атрибут , символ>), предполагающую (для простоты) отсутствие границ. Ассемблерные функции переводят адресацию слов в адресацию байтов, а также	FW

pr.	*path*	*an.*	*aut*
		функции прокрутки, учитывающие наличие/отсутствие границ у данного устройства. [(?) Ассемблерные функции по всей видимости расположены в libc !] Содержит также функции: cons_write, cons_echo, out_char, cons_putk, beep, do_escape, flush, parse_escape, scroll_screen, set_6845, stop_beep, cons_org0, disable_console, reenable_console, ga_program, cons_ioctl, mem_vid_copy, vid_vid_copy, get_6845, ...	
3	**/drivers/tty/keyboard.c**	Драйвер клавиатуры для персональных компьютеров PC и AT. Содержит в том числе функции: handle_req, handle_status, kbc_cmd0, kbc_cmd1, kbc_read, kbd_send, kb_ack, kb_wait, func_key, scan_keyboard, make_break, set_leds, show_key_mappings, kb_read, map_key, kbd_watchdog, int micro_delay(u32_t usecs); PUBLIC void do_kbd(message *m); PUBLIC int kbd_status(message *m); PUBLIC void do_kbdaux(message *m);	FW
3	**/drivers/tty/pty.c**	Драйвер псевдо терминала. Псевдо терминал (PTY) может быть виден как двунаправленный канал ((?) pipe) с терминалом (TTY) для обработки ввода и вывода. Для примера простая сессия авторизации (rlogin session): keyboard -> rlogin -> in.rld -> /dev/ptypX -> /dev/ttypX -> shell shell -> /dev/ttypX -> /dev/ptypX -> in.rld -> rlogin -> screen Этот файл заботится о копировании данных между парами устройств tty/pty и вызовами open/read/write/close на специальных файлах устройств псевдо терминалов (pty devices). Задание терминала (TTY) заботится об обработке ввода и вывода (прерывания, клавиши забоя (backspace) , непосредственный ввод/вывод, и.т.п.), используя функции pty_read() и pty_write() как функции «клавиатуры» и «экрана» устройств ttypX. Будьте осторожны, читая данный код, ибо термины "reading" и "writing" («чтение» и «запись») используются как для терминалов (tty), так и для псевдо терминалов (pty) со стороны псевдо терминалов. При этом чтение для одних означает запись для других и наоборот. Содержит также функции: pty_write, pty_echo, pty_start, pty_finish, pty_read, pty_close, pty_icancel, pty_ocancel, pty_select, PUBLIC void do_pty(tp, m_ptr);	FW
3	**/drivers/tty/rs232.c**	Драйвер серийного порта для 8250 и 16450 UART. Добавлена также поддержка Atari ST M68901 и YM-2149. Содержит множество констант (определений препроцессора), макросы, определения типов, в.т.ч.: rs232_t;	FW

pr.	*path*	*an.*	*aut*

		/* 8250 base addresses. */
		PRIVATE port_t addr_8250[] = {
		0x3F8, /* COM1 */
		0x2F8, /* COM2 */
		0x3E8, /* COM3 */
		0x2E8, /* COM4 */
		};
		
		Содержит в том числе функции:
		in_int, line_int, modem_int, rs_write, rs_echo, rs_ioctl, rs_config, rs_read, rs_icancel, rs_ocancel, rs_ostart, rs_break, rs_close, out_int, rs232_handler,
		
		PRIVATE void lock(void) {}
		PRIVATE void unlock(void) {}
		
		PRIVATE int my_inb(port_t port);	

2	**/ drivers/tty/keymaps/dvorak.src**	Исходный файл для расклада клавиатуры (Keymap)	FW
2	**/drivers/tty/keymaps/french.src**	Исходный файл для расклада клавиатуры (Keymap)	FW
2	**/ drivers/tty/keymaps/german.src**	Исходный файл для расклада клавиатуры (Keymap)	FW
2	**/drivers/tty/keymaps/italian.src**	Исходный файл для расклада клавиатуры (Keymap)	FW
2	**/ drivers/tty/keymaps/japanese.src**	Исходный файл для расклада клавиатуры (Keymap)	FW
2	**/drivers/tty/keymaps/latin-america.src**	Исходный файл для расклада клавиатуры (Keymap)	FW
2	**/ drivers/tty/keymaps/olivetti.src**	Исходный файл для расклада клавиатуры (Keymap)	FW
2	**/drivers/tty/keymaps/polish.src**	Исходный файл для расклада клавиатуры (Keymap)	FW
2	**/ drivers/tty/keymaps/russian.src**	Исходный файл для расклада клавиатуры (Keymap)	FW
2	**/drivers/tty/keymaps/russian-cp866.src**	Исходный файл для расклада клавиатуры (Keymap)	FW
2	**/ drivers/tty/keymaps/scandinavian.src**	Исходный файл для расклада клавиатуры (Keymap)	FW
2	**/ drivers/tty/keymaps/spanish.src**	Исходный файл для расклада клавиатуры (Keymap)	FW
2	**/drivers/tty/keymaps/uk.src**	Исходный файл для расклада клавиатуры (Keymap)	FW
2	**/drivers/tty/keymaps/us-std.src**	Исходный файл для расклада клавиатуры (Keymap)	FW
2	**/drivers/tty/keymaps/us-std-esc.src**	Исходный файл для расклада клавиатуры (Keymap)	FW
2	**/drivers/tty/keymaps/us-swap.src**	Исходный файл для расклада клавиатуры (Keymap)	FW
3	**/drivers/tty/keymaps/genmap.c**	Выводит бинарный расклад клавиатуры ((?) binary keymap). (использует в качестве исходных файлов, приведенные в папке	FW

pr.	*path*	*an.*	*aut*
		/drivers/tty/keymaps/ файлы с расширением .src).	
3	**/boot/bootblock.s**	Интересный способ проверки вариантов диска, а также пример загрузочного сектора.	FW
1	**/boot/a.out2com**	Скрипт, превращающий Minix a.out файл в com-файл. У меня возникает такое подозрение, что a.out в Minix3 также реализован не полностью и представляет собой дополненный заголовком 32-х битный raw-файл.	FW
3	**/boot/boothead.s**	Поддержка BIOS для boot.c . Файл содержит начальный и низкоуровневый код для вторичного загрузчика. Содержит функции для диска, tty (консоли) - ввода с клавиатуры, копирования памяти в произвольную точку.	FW
3	**/boot/rawfs.h**	В этих файлах соуществляется поддержка файловой системы Minix v1 v2 * One function needs to be provided by the outside world: * * void readblock(off_t blockno, char *buf, int block_size); * Read a block into the buffer. Outside world handles * errors.	FW
3	**/boot/rawfs.c**	В этих файлах соуществляется поддержка файловой системы Minix v1 v2 Based on readfs by Paul Polderman	FW
3	**/boot/addaout.c**	Маленькая утилита для добавления заголовка minix a.out к произвольному файлу. Это позволяет использовать произвольные данные в загрузочном образе так, что эти даннуе становятся образом участка оперативной памяти.	FW
3	**/boot/boot.c**	Загружает и запускает Minix.	FW
3	**/boot/boot.h**	Информация между различными частями процесса загрузки.	FW
3	**/boot/bootimage.c**	Загружает образ и запускает его.	FW
3	**/boot/doshead.s**	Файл содержит стартовую и низкоуровневою поддержку вторичной программы загрузчика. Данный вариант загрузчика запускается как com-файл из-под DOS-а.	FW
3	**/boot/image.h**	Информация между инсталяцией загрузчика и загручиком.	FW
3	**/boot/installboot.c**	Делает устройство загрузочным. Либо делает загрузочное устройство, либо делает загрузочный образ из микроялря , mm, fs, ...	FW
3	**/boot/jumpboot.s**	Этот код может быть помещён в любой свободный загрузочный сектор, подобный первому сектору расширенной партиции, партицию файловой системы, отличной от базовой (root), или в основной загрузочный сектор (?). Этот код загружает новый загрузчик, диск партиция и слайс (субпартиция) помещается в данный код утилитой installboot. Если нажата клавиша ALT, то диск, партиция и субпартиция вводятся вручную. Ручной интерфейс используется (по умолчанию) также в том случае, если installboot (по какой-то причине) не занёс свои данные для загрузки.	FW
3	**/boot/masterboot.s**	Код первичного загрузочного сектора.	FW
3	**/boot/mkfhead.s**	Содержит начальный код и код низкоуровневой поддержки MKFILE.COM.	FW
3	**/boot/mkfile.c**	Код MKFILE.COM. Работает в DOS, создаёт файл, который может быть использован ОС Minix как «диск».	FW

pr.	*path*	*an.*	*aut*
1	**/boot/updateboot.sh**	Скрипт устанавливает загрузочный сектор.	FW
3	**commands/ibm/postmort.c**	Этот файл похоже относится к процессору 8086 и осуществляет dump состояния ядра (после краха).	FW
3	**commands/ibm/sdump.c**	Этот файл выполняет dump памяти процесса. Он относится к i386.	FW
3	**/etc/binary_sizes**	/usr/lib/cpp.ansi 216004 /usr/lib/cv 1238570 /usr/lib/em_cemcom.ansi 624004 /usr/lib/em_led 538122 /usr/lib/em_opt 208000 /usr/lib/i386/as 55000 /usr/lib/i386/cg 50655 /bin/sh 142400 /usr/bin/make 380000	FW
3	**/etc/binary_sizes.big**	/usr/lib/cpp.ansi 1416004 /usr/lib/cv 3738570 /usr/lib/em_cemcom.ansi 10683676 /usr/lib/em_led 4078088 /usr/lib/em_opt 1370162 /usr/lib/i386/as 1239922 /usr/lib/i386/cg 1285513 /bin/sh 150000 /usr/bin/make 337920	FW
3	**/etc/binary_sizes.xxl**	/usr/lib/cpp.ansi 1416004 /usr/lib/cv 3738570 /usr/lib/em_cemcom.ansi 10683676 /usr/lib/em_led 4078088 /usr/lib/em_opt 1370162 /usr/lib/i386/as 1239922 /usr/lib/i386/cg 1285513 /usr/bin/make 2000000 /bin/sh 1000000	FW
3	**/etc/crontab**	? 6 * * * /usr/etc/daily cron	FW
3	**/etc/drivers.conf**	Содержит список конфигурационных параметров для всех драйверов (а скорее даже подгружаемых привилегированных процессов), например: ***driver mfs*** ***{*** ***system*** ***TIMES # 25*** ***SAFECOPYFROM # 31*** ***SAFECOPYTO # 32*** ***GETINFO*** ***SETGRANT # 34*** ***UMAP # 14*** ***PROFBUF # 38*** ***SYSCTL*** ***;*** ***uid 0;*** ***};***	FW
3	**/etc/fstab**	# Poor man's File System Table. root=/dev/ROOT usr=/dev/USR	FW

pr.	*path*	*an.*	*aut*
3	**/etc/group**	operator:*:0: daemon:*:1: bin:*:2: other:*:3: tty:*:4: uucp:*:5: news:*:6: ftp:*:7: kmem:*:8: www:*:9: driver:*:10: server:*:11: smtpd:*:26: nogroup:*:99:	FW
2	**/etc/hostname.file**	minix	FW
3	**/etc/inet.conf**	psip0 { default; };	FW

3	**/etc/make.conf**	# which architecture to compile for
		ARCH=i386
			FW

3	**/etc/motd**	To install the X Window System, run 'packman' with the install CD still in the drive. To start X Windows after you have installed it, login as root and type: 'xdm'. For more information about configuring X Windows, see www.minix3.org. If you do not have sufficient memory to run X, standard MINIX 3 supports multiple virtual terminals. Just use ALT+F1, F2, F3 and F4 to navigate among them. To get rid of this message, edit /etc/motd.	FW
2	**/etc/motd.install**	To install X Windows, run 'packman' with the install CD still in the drive. To start X Windows after you have installed it, login as root and type: 'xdm'. For more information about configuring X Windows, see www.minix3.org. If you do not have sufficient memory to run X Windows, standard MINIX 3 supports multiple virtual terminals. Just use ALT+F1, F2, F3 and F4 to navigate among them. To get rid of this message, edit /etc/motd.	FW
1	**/etc/mtab**	Содержит список реально смонтированных файловых систем. Изначально этот файл пуст ... Но заполняется при инициализации системы, а также дополняется при монтировании дополнительных файловых систем. После запуска команды packman , на виртуальной машине запуск команды cat /etc/mtab Даёт результат: /dev/c0d0p0s0 / 3 rw /dev/c0d0p0s2 /usr 3 rw /dev/c0d0p0s1 /home 3 rw /dev/c0d2p2 /mnt 3 ro (По сравнению с результатом этой команды до запуска packman, добавилась последняя строчка! После выхода из packman, последняя строчка сохраняется – так как файловая система остаётся смонтированной. Однако она пропадает после команды umount	FW

pr.	*path*	*an.*	*aut*
		/dev/c0d2p2)	
2	**/etc/mtree.sh**	#!/bin/sh cat $1 | while read line do echo $line | awk 'NF==4 { print "mkdir -p "$4" || exit 1; chmod "$1" "$4" || exit 1; chown "$2" "$4" || exit 1; chgrp "$3" "$4" || exit 1" } NF==3 { print "rm "$1" ; ln -s "$3" "$1" || exit 1" } ' | sh || exit 1 done Однако этот файл отсутствует в /etc/ реально установленной ОС Minix	FW
3	**/etc/passwd**	root:##root:0:0:Big Brother:/root: daemon:*:1:1:The Deuce:/etc: bin:##root:2:0:Binaries:/home/bin: uucp:*:5:5:UNIX to UNIX copy:/usr/spool/uucp:/usr/bin/uucico news:*:6:6:Usenet news:/usr/spool/news: ftp:*:7:7:Anonymous FTP:/usr/ftp: ast:*:8:3:Andrew S. Tanenbaum:/home/ast: www:*:9:9:World Wide Web:/usr/www: driver:*:10:10:Device Drivers:/: server:*:11:11:OS Servers:/: sshd:*:22:22:sshd:/: smtpd:*:25:25:smtpd:/: games:*:9998:98::/: nobody:*:9999:99::/tmp:	FW
3	**/etc/profile**	RC_TZ=/etc/rc.timezone export TZ=GMT0 if [-f "$RC_TZ"] then . "$RC_TZ" fi export MANPATH=/usr/man:/usr/local/man:/usr/gnu/man:/usr/X11R6/man	FW
2	**/etc/protocols**	# # Internet (IP) protocols # # @(#)protocols 8.1 (Berkeley) 6/9/93 # ip 0 IP # internet protocol, pseudo protocol number icmp 1 ICMP # internet control message protocol igmp 2 IGMP # internet group management protocol ggp 3 GGP # gateway-gateway protocol tcp 6 TCP # transmission control protocol egp 8 EGP # exterior gateway protocol pup 12 PUP # PARC universal packet protocol udp 17 UDP # user datagram protocol hmp 20 HMP # host monitoring protocol xns-idp 22 XNS-IDP # Xerox NS IDP rdp 27 RDP # reliable data protocol iso-tp4 29 ISO-TP4 # ISO Transport Protocol Class 4 iso-ip 80 ISO-IP # ISO Internet Protocol encap 98 ENCAP # RFC1241 encapsulation	FW
3	**/etc/rc**	Скрипт начального запуска системы, выполняемый init до выхода в многопользовательский режим.	FW
2	**/etc/rc.cd**	#!/bin/sh # CD boottime initializations.	FW
2	**/etc/rc.daemons.dist**	daemonize talkd daemonize tcpd shell in.rshd daemonize tcpd login in.rlogind daemonize tcpd telnet in.telnetd daemonize tcpd ftp in.ftpd	FW
2	**/etc/rc.rescue**	#!/bin/sh DRIVERS=/sbin RESCUE=/boot/rescue if ["$1" != start] then exit fi	FW

pr.	*path*	*an.*	*aut*
		set -e service up $DRIVERS/rescue -dev /dev/rescue -args 128 -period 4HZ mkfs /dev/rescue mount /dev/rescue $RESCUE cd $DRIVERS cp -p at_wini floppy bios_wini $RESCUE service rescue $RESCUE	
2	**/etc/rs.inet**	#!/bin/sh # 'Recovery' script that doesn't. This script is to be used for drivers that # should not be restarted. Instead, the scripts configures the driver 'down'. kill_by_name() { label="$1" pid=`ps ax \| grep "$label" | grep -v grep | sed 's,[ ]*\([0-9]*\).*,\1,` if [X"$pid" = X] then return 1 # No such process fi echo "killing pid $pid for $label" kill -9 $pid } daemonize() { # Function to start a daemon, if it exists. local IFS=':' local name="$1" test "$1" = tcpd && name="$2" for dir in $PATH do if [-f "$dir/$1"] then # check if this service is disabled at the boot monitor. if disabled $name; then return; fi echo -n " $name" "$@" & return fi done } disabled() { ifs="$IFS"; IFS=, for skip in `sysenv disable` do if ["$skip" = "$1"] then IFS="$ifs"; unset ifs return 0 fi done IFS="$ifs"; unset ifs return 1 } exec > /dev/console echo "Arguments: $@" kill_by_name dhcpd kill_by_name nonamed kill_by_name syslogd sleep 1 service restart "$1" sleep 1 daemonize dhcpd daemonize nonamed -L daemonize syslogd	FW

pr.	*path*	*an.*	*aut*
2	**/etc/rs.single**	#!/bin/sh # 'Recovery' script that doesn't. This script is to be used for drivers that # should not be restarted. Instead, the scripts configures the driver 'down'. #echo "Arguments: $@" >/dev/console service down "$1"	FW
3	**/etc/services**	Сетевые сервисы, стиль интернета.	FW
2	**/etc/shadow**	Сетевые сервисы, стиль интернета. (ещё один вариант)	FW
3	**/etc/syslog.conf**	## ## @(#)syslog.conf 1.0 Jan. 18, 2000 ## ## Use kill -HUP `cat /usr/run/syslogd.pid` to restart ## the server, forcing it to reread this file. ## Emergency messages (system may be unusable) *.emerg * *.alert /dev/log ## High severity errors *.alert;*.crit /var/log/syslog ## Every other message (errors/warning and informational) *.info;*.notice;*.warning;*.err /var/log/messages ## Debug informations (tracing programs) #*.debug /var/log/debug ## $Id: syslog.conf 2063 2006-04-05 14:25:21Z beng $ ## end syslog.conf	FW
2	**/etc/termcap**	(?) Доаольно большой файл, как то связан с терминалами ...	FW
2	**/etc/termcap.big**	# PURPOSE OF THIS FILE: # # This file describes the capabilities of various character-cell terminals, # as needed by software such as screen-oriented editors. ... (?)	FW
3	**/etc/ttytab**	# ttytab - terminals # # Device Type Program Init console minix getty ttyc1 minix getty ttyc2 minix getty ttyc3 minix getty tty00 unknown tty01 unknown ttyp0 network ttyp1 network ttyp2 network ttyp3 network ttyp4 network ttyp5 network ttyp6 network ttyp7 network ttyp8 network ttyp9 network ttypa network ttypb network ttypc network ttypd network ttype network ttypf network ttyq0 network ttyq1 network	FW

pr.	*path*	*an.*	*aut*
		ttyq2 network ttyq3 network ttyq4 network ttyq5 network ttyq6 network ttyq7 network ttyq8 network ttyq9 network ttyqa network ttyqb network ttyqc network ttyqd network ttyqe network ttyqf network	
1	**/etc/utmp**	Сам по себе пуст, однако также заполняется при инициализации системы ... (?)	FW
3	**/etc/ast/.ashrc**	# Ash initialization. test -z "$EDITOR" && { # Don't repeat in subshells. umask 022 # Favourite editor and pager, search path for binaries, etc. export EDITOR=vi export PAGER=more export PATH=$HOME/bin:/usr/local/bin:/bin:/usr/bin:/usr/X11R6/bin } # End of no-repeat. # Let cd display the current directory on the status line. if ["$-" : '.*i.*' -a -t 0 -a -f /usr/bin/tget] && tget -flag hs then hostname=$(expr $(uname -n) : '\([^.]*\)') eval "cd() { chdir \"\$@\" && echo -n '$(tget -str ts \ "$USER@$hostname:\"\`pwd\`\"" \ -str fs)' }" unset hostname cd . fi	FW
1	**/etc/ast/.ellepro.b1**	Какой-то бинарный файл (?)	FW
2	**/etc/ast/.ellepro.e**	; This is the default user profile to emulate "mined" (MINIX editor), ; as per Andy Tanenbaum.	FW
2	**/etc/ast/.exrc**	set autoindent autowrite report=2 showmatch	FW
3	**/etc/ast/.profile**	# Login shell profile. # Erase character and erase line interrupt keys stty sane erase '^H' kill '^U' # Check terminal type. case $TERM in dialup\|unknown\|network) echo -n "Terminal type? ($TERM) "; read term TERM="${term:-$TERM}" unset term esac # Shell configuration. unset EDITOR; . $HOME/.ashrc	FW
1	**/etc/fonts/cp437**	Бинарный файл фонта – отображений символов:(?)	FW
1	**/etc/fonts/cp850**	Бинарный файл фонта – отображений символов:(?)	FW

pr.	*path*	*an.*	*aut*
1	**/etc/fonts/cp865**	Бинарный файл фонта – отображений символов:(?)	FW
1	**/etc/fonts/cp866**	Бинарный файл фонта – отображений символов:(?)	FW
1	**/etc/fonts/iso1**	Бинарный файл фонта – отображений символов:(?)	FW
1	**/etc/fonts/koi8-r**	Бинарный файл фонта – отображений символов:(?)	FW
1	**/etc/fonts/polish**	Бинарный файл фонта – отображений символов:(?)	FW
2	**/etc/mtree/minix.tree**	Файл вида (?) (скорее всего генерируется автоматически): ***755 root operator /*** ***755 bin operator /bin*** ***755 bin operator /sbin*** ***755 root operator /dev*** ***/dev/mouse -> /dev/kbdaux*** ***755 root operator /etc*** ***755 root operator /lib*** ***755 root operator /lib/i386*** ***755 root operator /boot*** ***755 root operator /boot/image*** ...	FW
2	**/etc/usr/daily**	#!/bin/sh # # daily - daily cleanup of the system. # Doesn't make sense when running from CD ...	FW
2	**/etc/usr/dhcptags.conf**	# A list of all tags mentioned in RFC-1533. tag 1 netmask ip 1 1; tag 2 zoneoffset number 4 1; tag 3 gateway ip 1 0; tag 4 timeserver ip 1 0; tag 5 nameserver ip 1 0; tag 6 DNSserver ip 1 0; tag 7 logserver ip 1 0; tag 8 cookieserver ip 1 0; tag 9 LPR ip 1 0; ... (?)	FW
3	**/etc/usr/rc**	Продолжение инициализации системы (?).	FW
3	**/tools/chrootmake.sh**	#!/bin/sh set -e export SHELL=/bin/sh cd /usr/src make etcfiles su bin -c 'make world install' cd tools rm revision rm /boot/image/* make install cp /boot/image/* /boot/image_big # Make big image accessible by this name cp ../boot/boot /boot/boot cd /usr/src make clean # Let man find the manpages su bin -c 'makewhatis /usr/man' su bin -c 'makewhatis /usr/local/man' binsizes normal	FW

pr.	*path*	*an.*	*aut*
1	**/tools/fdbootparams**	rootdev=c0d0p1s0; ramimagedev=c0d0p1s0; save	FW
3	**/tools/issue.install**	Welcome to MINIX 3.1.5. This snapshot is an interim release, not final release quality. It is intended as a prerelease for developers. The system is now running and many commands work normally. To use MINIX in a serious way, you need to install it to your hard disk, which you can do by typing 'setup' while logged in as root. Then just follow the on-screen directions. After setup is complete, type 'shutdown' and when the boot monitor starts, boot your new system by following the instructions at the end of setup. Keep the CD-ROM in the drive, login as root and type 'packman' to begin installing the many software packages available. After you have installed the packages, type 'xdm' to start X Windows if you have installed it. Before you begin the installation process, it is strongly recommended that you print and carefully read the installation instructions available on the MINIX 3 website: http://www.minix3.org.	FW
3	**/tools/Makefile**	Основной файл, управляющий перекомпиляцией операционной системы Minix3. Фрагмент: usage: @echo " " >&2 @echo "Master Makefile to create new MINIX configuration." >& 2 @echo "Root privileges are required." >&2 @echo " " >&2 @echo "Usage:" >&2 @echo " make includes # Install include files" >&2 @echo " make depend # Generate dependency files" >&2 @echo " make libraries # Make system libraries" >&2 @echo " make services # Compile and install all services" >&2 @echo " make fresh # Make clean, libraries, and services" >&2 @echo " make image # Make needed services and create boot image" >&2 @echo " make install # Make image, and install to hard disk" >&2 @echo " make hdboot # Make image, and install to hard disk" >&2 @echo " make fdboot # Make image, and install to floppy disk" >&2 @echo " make bootable # Make hard disk bootable" >&2 @echo " make clean # Remove all compiler results, except libs" >&2 @echo " " >&2 @echo "To create a fresh MINIX configuration, try:" >&2 @echo " make clean install # new boot image" >&2 @echo " make fresh install # new everything" >&2 @echo " " >&2	FW
2	**/tools/mkboot**	Создаёт загрузочную дискету (скрипт командного интерпретатора) #!/bin/sh # # mkboot 2.0 - make boot floppy, make root device bootable, etc. #	FW
3	**/tools/package_sources.install**	Список пакетов, устанавливаемых на диск (?) вместе с исходниками.	FW
3	**/tools/packages.install**	Список пакетов, устанавливаемых на диск (?).	FW
2	**/tools/release.sh**	Начало: #!/bin/sh set -e PATH=$PATH:/usr/local/bin XBIN=usr/xbin SRC=src # size of /tmp during build	FW

pr.	*path*	*an.*	*aut*

		TMPKB=32000
		
		PACKAGEDIR=/usr/bigports/Packages
		PACKAGESOURCEDIR=/usr/bigports/Sources
		# List of packages included on installation media
		PACKAGELIST=packages.install
		# List of package source included on installation media
		PACKAGESOURCELIST=package_sources.install
		secs=`expr 32 '*' 64`
		export SHELL=/bin/sh
		
		# SVN trunk repo
		TRUNK=https://gforge.cs.vu.nl/svn/minix/trunk
		
		make_hdimage()
		{	

2	**/tools/revision**	0	FW
2	**/tools/tell_config**	#!/bin/sh # # tellconfig - Tell the value of a <minix/config.h> parameter # Author: Kees J. Bot echo " #include <minix/config.h> $* " >/tmp/tell.$$ exec </tmp/tell.$$ rm /tmp/tell.$$ exec cc -P -E -	FW
3	**/tools/release/cd/README.txt**	Краткое описание дистрибутива на английском языке.	FW