

Содержимое

Статьи

Гибридное ядро	1
Микроядро	3
Монолитное ядро	5
Наноядро	6
Экзоядро	7
Линейная адресация памяти	8
Страничная память	11
ext2	15
ACL	19
Символьная ссылка	21
Именованный канал	23
Сокет (программный интерфейс)	25
Переносимый набор символов	26
CP437	29

Сноски

Источники и основные авторы	31
Источники, Лицензии и Владельцы(?) изображения.	32

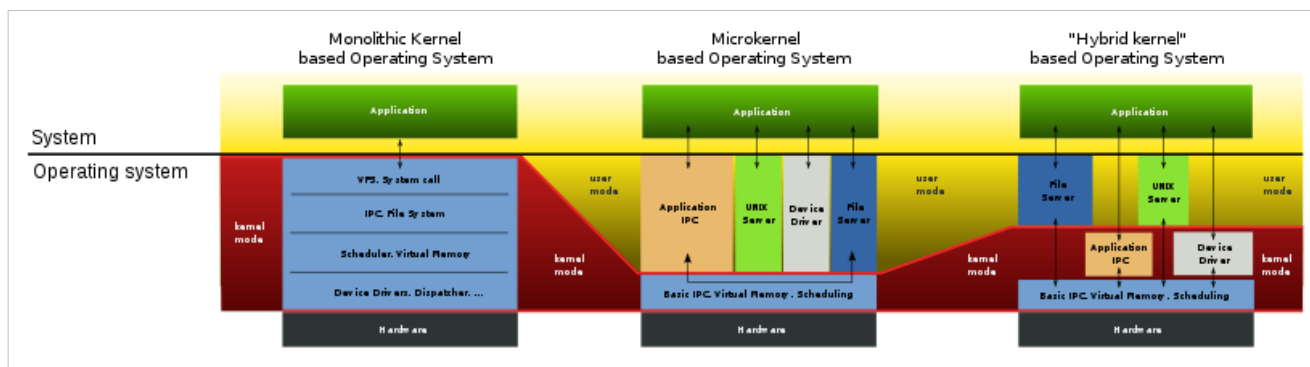
Лицензии статей.

Лицензия	33
----------	----

Гибридное ядро

Гибридное ядро (англ. *Hybrid kernel*) — модифицированные микроядра (минимальная реализация основных функций ядра операционной системы компьютера), позволяющие для ускорения работы запускать «несущественные» части в пространстве ядра.

Имеют «гибридные» достоинства и недостатки.



Все рассмотренные подходы к построению операционных систем имеют свои достоинства и недостатки. В большинстве случаев современные операционные системы используют различные комбинации этих подходов. Так, например сейчас, ядро «Linux» представляет собой монолитную систему с отдельными элементами модульного ядра. При компиляции ядра можно разрешить динамическую загрузку и выгрузку очень многих компонентов ядра — так называемых модулей. В момент загрузки модуля его код загружается на уровне системы и связывается с остальной частью ядра. Внутри модуля могут использоваться любые экспортируемые ядром функции.

Существуют варианты ОС GNU (Debian/GNU Hurd), в которых вместо монолитного ядра применяется ядро Mach (такое же, как в Hurd), а поверх него в пользовательском пространстве работают те же самые процессы, которые при использовании Linux были бы частью ядра. Другим примером смешанного подхода может служить возможность запуска операционной системы с монолитным ядром под управлением микроядра. Так устроены 4.4BSD и MkLinux, основанные на микроядре Mach. Микроядро обеспечивает управление виртуальной памятью и работу низкоуровневых драйверов. Все остальные функции, в том числе взаимодействие с прикладными программами, осуществляется монолитным ядром. Данный подход сформировался в результате попыток использовать преимущества микроядерной архитектуры, сохраняя по возможности хорошо отлаженный код монолитного ядра.

Наиболее тесно элементы микроядерной архитектуры и элементы монолитного ядра переплетены в ядре Windows NT. Хотя Windows NT часто называют микроядерной операционной системой^[1], это не совсем так. Микроядро NT слишком велико (более 1 Мбайт, кроме того, в ядре системы находится, например, ещё и модуль графического интерфейса), чтобы носить приставку «микро». Компоненты ядра Windows NT располагаются в вытесняемой памяти и взаимодействуют друг с другом путем передачи сообщений^{[1] [2]}, как и положено в микроядерных операционных системах. В то же время все компоненты ядра работают в одном адресном пространстве и активно используют общие структуры данных, что свойственно операционным системам с монолитным ядром. По мнению специалистов Microsoft^[источник не указан 323], причина проста: чисто микроядерный дизайн коммерчески невыгоден, поскольку неэффективен.

Таким образом, Windows NT можно с полным правом назвать гибридной операционной системой.

Смешанное ядро, в принципе, должно объединять преимущества монолитного ядра и микроядра: казалось бы, микроядро и монолитное ядро — крайности, а смешанное — золотая середина. В них возможно добавлять драйвера устройств двумя способами: и внутрь ядра, и в пользовательское пространство. Но на практике концепция смешанного ядра часто подчёркивает не только достоинства, но и недостатки обоих типов ядер.

Примеры

- Syllable
- BeOS
 - Haiku
- XNU (на основе Darwin), используется в Mac OS X)
- BSD-основанные
 - DragonFly BSD (первое из семейства BSD микроядро, основанное не на Mach)
- NetWare^[3]
- Plan 9
 - Inferno
- NT kernel (используется Windows NT, 2000, 2003; XP, Vista, Seven)
 - ReactOS

Ссылки

- Linus Torvalds on Real World Tech ^[4]
- Hybrid Kernel category criticised ^[5]
- Sysinternals article about the NT Native API ^[6]

Сноски

- [1] MS Windows NT Kernel-mode User and GDI White Paper (<http://www.microsoft.com/technet/archive/ntwrkstn/evaluate/featfunc/kernelwp.mspx?mfr=true>). Microsoft Corporation (2007). Проверено 1 x12 2007.
- [2] *Abraham Silberschatz* Operating System Concepts; 7th Edition (<http://higheredbcs.wiley.com/legacy/college/silberschatz/0471694665/appendices/appb.pdf>). — Hoboken, New Jersey: John Wiley & Sons Inc. — ISBN 978-0-471-69466-3
- [3] An Overview of the NetWare Operating System ([http://www.usenix.org/publications/library/proceedings/sf94/full_papers/minshall.a\)\(2007-02-07](http://www.usenix.org/publications/library/proceedings/sf94/full_papers/minshall.a)(2007-02-07))
- [4] <http://www.realworldtech.com/forums/index.cfm?action=detail&id=66630&threadid=66595&roomid=11>
- [5] <http://www.realworldtech.com/forums/index.cfm?action=detail&id=66595&threadid=66595&roomid=11>
- [6] <http://www.sysinternals.com/Information/NativeApi.html>

Микроядро

Микроядро — это минимальная реализация функций ядра операционной системы.

Классические микроядра предоставляют лишь очень небольшой набор низкоуровневых примитивов, или системных вызовов, реализующих базовые сервисы операционной системы.

Сюда относятся:

- управление адресным пространством оперативной памяти.
- управление адресным пространством виртуальной памяти.
- управление процессами и потоками (нитеями).
- средства межпроцессной коммуникации.

Все остальные сервисы ОС, в классических монолитных ядрах предоставляемые непосредственно ядром, в микроядерных архитектурах реализуются в адресном пространстве пользователя (Ring3) и называются сервисами. Примерами таких сервисов, выносимых в пространство пользователя в микроядерных архитектурах, являются сетевые сервисы, файловая система, драйверы.

Такая конструкция позволяет улучшить общее быстродействие системы (небольшое микроядро может уместиться в кэше процессора).^[1] Основное достоинство микроядерной архитектуры — высокая степень модульности ядра операционной системы. Это существенно упрощает добавление в него новых компонентов. В микроядерной операционной системе можно, не прерывая ее работы, загружать и выгружать новые драйверы, файловые системы и т. д. Существенно упрощается процесс отладки компонентов ядра, так как новая версия драйвера может загружаться без перезапуска всей операционной системы. Компоненты ядра операционной системы ничем принципиально не отличаются от пользовательских программ, поэтому для их отладки можно применять обычные средства. Микроядерная архитектура повышает надежность системы, поскольку ошибка на уровне непривилегированной программы менее опасна, чем отказ на уровне режима ядра.

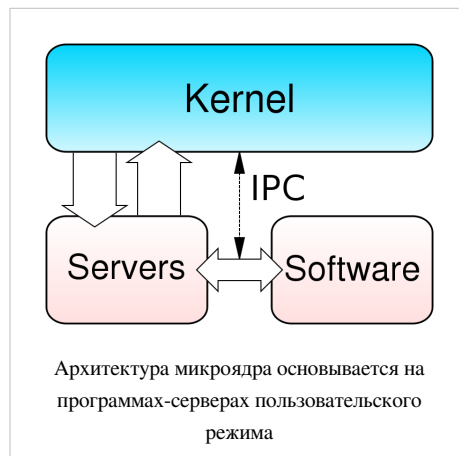
И чтобы добавить в ОС с микроядром драйвер того или иного устройства, не надо перекомпилировать всё ядро, а надо лишь отдельно откомпилировать этот драйвер и запустить его в пользовательском пространстве.

В то же время микроядерная архитектура операционной системы вносит дополнительные накладные расходы, связанные с передачей сообщений, что отрицательно влияет на производительность. Для того чтобы микроядерная операционная система по скорости не уступала операционным системам на базе монолитного ядра, требуется очень аккуратно проектировать разбиение системы на компоненты, стараясь минимизировать взаимодействие между ними. Таким образом, основная сложность при создании микроядерных операционных систем — необходимость очень аккуратного проектирования.

Микроядра типа ядра ОС Minix и GNU Hurd развиваются медленно, гораздо медленнее, чем Linux и ядро систем семейства BSD, но они обладают огромным потенциалом, то есть заделом на будущее, и, возможно, в этом самом будущем, достигнут аналогичного функционала.

Классическим примером микроядерной системы является Symbian OS. Это пример распространенной и отработанной микроядерной (а начиная с версии Symbian OS v8.1, и наноядерной) операционной системы.

В отличие от Windows NT ^[источник не указан 323], создателям Symbian OS удалось совместить эффективность и концептуальную стройность, несмотря на то что современные версии этой системы предоставляют обширные возможности, в том числе средства для работы с потоковыми данными, стеками протоколов, критичными к латентности ядра, графикой и видео высокого разрешения).



Будучи микроядерной операционной системой, Symbian «выносит» практически все прикладные (т.е. выходящие за пределы компетенции ядра) задачи в модули-серверы, функционирующие в пользовательском адресном пространстве.

В ОС Windows NT версий 3.x микроядерная архитектура с сервисным процессом использовалась для подсистемы графики и пользовательского интерфейса. В частности, драйвер графической аппаратуры загружался в контекст сервисного процесса, а не ядра. Начиная с версии 4, от этого отказались, сервисный процесс сохранился только для управления консольными окнами командной строки, а собственно графическая подсистема вместе с драйвером аппаратуры (в том числе трехмерной графики) переместилась в специально обособленный регион ядра ОС.

ОС Windows CE (и созданные на ее основе сборки, такие, как Windows Mobile), будучи практически полностью совместимой (как подмножество) с Windows NT по вызовам и методам программирования приложений, тем не менее полностью отличается от Windows NT по внутренней архитектуре и является микроядерной ОС с выносом всех драйверов устройств, сетевых стеков и графической подсистемы в сервисные процессы.

Недостаток — плата за принудительное «переключение» процессов в ядре (переключение контекста); этот факт собственно и объясняет трудности в проектировании и написании ядер подобной конструкции. Эти недостатки способны обойти ОС, использующие архитектуру экзоядра, являющуюся дальнейшим развитием микроядерной архитектуры.

См. также

Микроядра

• Mach • L4 • QNX

ОС на основе микроядер

- | | | |
|-----------|------------|-----------|
| • QNX | • Minix3 | • AmigaOS |
| • TUDOS | • GNU/Hurd | • Symbian |
| • Windows | • osFree | OS |
| CE | • Mac OS X | • Jari OS |
| • AIX | | • OpenVMS |

Ссылки

- Интервью с Эндрю Таненбаумом^[2]

Сноски

[1] У ряда современных процессоров кэш процессора достаточно велик, чтобы вместить даже монолитное ядро, поэтому по мере улучшения процессоров данное достоинство все больше нивелируется увеличивающимися размерами кэша процессора.^[источник не указан 323] (Подобное в действительности невозможно из-за того, что все драйверы монолитного ядра и ряд служб работают исключительно в режиме ядра (*kernel-mode*), увеличивая размер исполняемого кода ядра до многих десятков мегабайт. У микроядер все службы и драйверы работают в пространстве пользователя (*user-mode*).

[2] <http://ylsoftware.com/?action=news&na=viewfull&news=222>

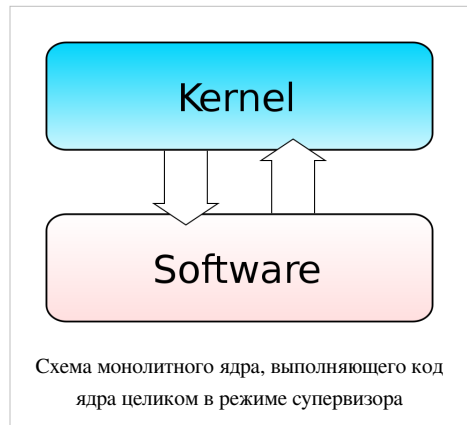
Монолитное ядро

Монолитное ядро́ — классическая и, на сегодняшний день, наиболее распространённая архитектура ядер операционных систем. Монолитные ядра предоставляют богатый набор абстракций оборудования. Все части монолитного ядра работают в одном адресном пространстве.

Монолитные ядра имеют долгую историю развития и усовершенствования и, на данный момент, являются наиболее архитектурно зрелыми и пригодными к эксплуатации. Вместе с тем, монолитность ядер усложняет их отладку, понимание кода ядра, добавление новых функций и возможностей, удаление «мёртвого», ненужного, унаследованного от предыдущих версий кода.

«Разбухание» кода монолитных ядер также повышает требования к объёму оперативной памяти, требуемому для функционирования ядра ОС. Это делает монолитные ядерные архитектуры малоприспособленными к эксплуатации в системах, сильно ограниченных по объёму ОЗУ, например, встраиваемых системах, производственных микроконтроллерах и т. д.

Альтернативой монолитным ядрам считаются архитектуры, основанные на микроядрах.



Подгружаемые модули

Старые монолитные ядра требовали перекомпиляции при любом изменении состава оборудования. Большинство современных ядер, такие как OpenVMS, Linux, FreeBSD, NetBSD и Solaris, позволяют во время работы динамически (по необходимости) подгружать и выгружать *модули*, выполняющие часть функций ядра. Модульность ядра осуществляется на уровне бинарного образа, а не на архитектурном уровне ядра, так как динамически подгружаемые модули загружаются в адресное пространство ядра и в дальнейшем работают как интегральная часть ядра. Модульные монолитные ядра не следует путать с архитектурным уровнем модульности, присущим микроядрам и гибридным ядрам. Практически, динамическая загрузка модулей - это просто более гибкий способ изменения образа ядра во время выполнения — в отличие от перезагрузки с другим ядром. Модули позволяют легко расширить возможности ядра по мере необходимости. Динамическая подгрузка модулей помогает сохранить размер кода, работающего в пространстве ядра, до минимума, например, свести к минимуму отпечаток ядра для встраиваемых устройств с ограниченными аппаратными ресурсами.

Наноядро

Наноядро — архитектура ядра операционной системы компьютеров, в рамках которой крайне упрощённое и минималистичное ядро выполняет лишь одну задачу — обработку аппаратных прерываний, генерируемых устройствами компьютера. После обработки прерываний от аппаратуры наноядро, в свою очередь, посылает информацию о результатах обработки (например, полученные с клавиатуры символы) вышележащему программному обеспечению при помощи того же механизма прерываний.

В некотором смысле концепция наноядра близка к концепции HAL — Hardware Abstraction Layer, предоставляя вышележащему ПО удобные механизмы абстракции от конкретных устройств и способов обработки их прерываний.

Наиболее часто в современных компьютерах наноядра используются для виртуализации аппаратного обеспечения реальных компьютеров или для реализации механизма гипервизора, с целью позволить нескольким или многим различным операционным системам работать одновременно и параллельно на одном и том же компьютере. Например, VMware ESX Server реализует собственное наноядро, не зависящее от ОС и устанавливаемое на «голое железо». Поверх этого наноядра работают пользовательские и административные утилиты VMware и сами операционные системы, виртуализируемые в ESX Server.

Наноядра также могут использоваться для обеспечения переносимости (портability) операционных систем на разное аппаратное обеспечение или для обеспечения возможности запуска «старой» операционной системы на новом, несовместимом аппаратном обеспечении без ее полного переписывания и портирования. Например, фирма Apple Computer использовала наноядро в версии Mac OS Classic для PowerPC для того, чтобы транслировать аппаратные прерывания, генерировавшиеся их компьютерами на базе процессоров PowerPC в форму, которая могла «пониматься» и распознаваться Mac OS для процессоров Motorola 680x0. Таким образом, наноядро эмулировало для Mac OS «старое» 680x0 железо. Альтернативой было бы полное переписывание и портирование кода Mac OS на PowerPC при переходе с 680x0 на них. Позднее, в эпоху Mac OS 8.6, наноядро виртуализировало предоставляемые PowerPC мультипроцессорные возможности и обеспечивало поддержку SMP в Mac OS. Другие удачные примеры использования наноядерных архитектур включают наноядро Adeos, работающее как модуль ядра для Linux и позволяющее выполнять одновременно с Linux какую-либо операционную систему реального времени.

Наноядро может быть настолько маленьким и примитивным, что даже важнейшие устройства, находящиеся непосредственно на материнской плате или на плате контроллера встраиваемого устройства, такие, как таймер или программируемый контроллер прерываний, обслуживаются специальными драйверами устройств, а не непосредственно ядром. Такого рода сверхминималистичные наноядра называют иногда пикоядрами.

Термин «наноядро» иногда неформально используется для описания очень маленьких, упрощённых и лёгких микроядер, таких, как L4.

Экзоядро

Экзоядро — ядро операционной системы компьютеров, предоставляющее лишь функции для взаимодействия между процессами и безопасного выделения и освобождения ресурсов.

Экзо — приставка, обозначающая нечто внешнее, находящееся снаружи.

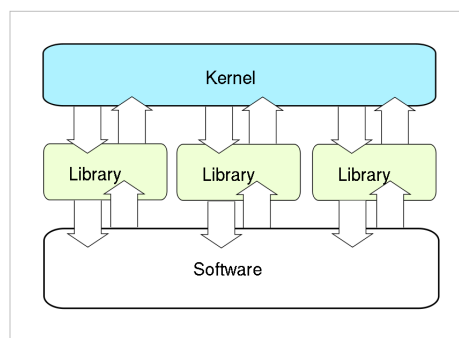
В традиционных операционных системах ядро предоставляет не только минимальный набор сервисов, обеспечивающих выполнение программ, но и большое количество высокоуровневых абстракций для использования разнородных ресурсов компьютера:

оперативной памяти, жестких дисков, сетевых подключений. В отличие от них, ОС на основе экзоядра предоставляет лишь набор сервисов для взаимодействия между приложениями, а также необходимый минимум функций, связанных с защитой: выделение и высвобождение ресурсов, контроль прав доступа, и т. д. Экзоядро не занимается предоставлением абстракций для физических ресурсов — эти функции выносятся в библиотеку пользовательского уровня (так называемую libOS).

Основная идея операционной системы на основе экзоядра состоит в том, что ядро должно выполнять лишь функции координатора для небольших процессов, связанных только одним ограничением — экзоядро должно иметь возможность гарантировать безопасное выделение и освобождение ресурсов оборудования. В отличие от ОС на основе микроядра, ОС, базирующиеся на экзоядре, обеспечивают гораздо большую эффективность за счет отсутствия необходимости в переключении между процессами при каждом обращении к оборудованию.

Архитектуры на основе экзоядер являются дальнейшим развитием и усовершенствованием микроядерных архитектур и одновременно ужесточают требования к минималистичности и простоте кода ядра.

libOS может обеспечивать произвольный набор абстракций, совместимый с той или иной уже существующей операционной системой, например Linux или Windows.



Линейная адресация памяти

Линейная адресация памяти — схема адресации памяти компьютера.

В Intel-совместимых процессорах доступна, начиная с Intel 80386.

Благодаря введению механизма **линейной адресации** можно создавать любое (сколько в память влезет) количество *адресных пространств*. Причём каждая *страница* линейного адресного пространства может находиться по любому (естественно, выравненному по границе 4 КБайт) физическому адресу, а благодаря обработчику #PF и на любом накопителе ^[1].

Страница

Вся физическая память делится на *страницы* фиксированного размера (4 КБайт, 2 МБайт, 4 МБайт). Каждая *страница*, независимо от размера выравнена по границе 4 КБайт.

Деление линейного адреса

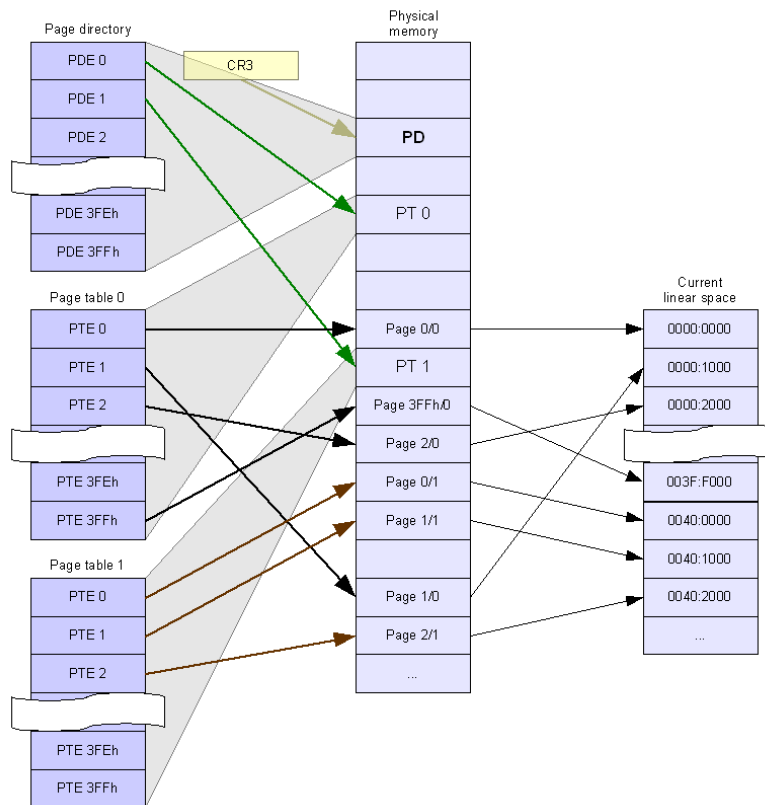
При использовании *линейной адресации* 32-битный адрес делится на три части:

- Номер записи в *каталоге страниц* (номер таблицы страниц, см. ниже) — биты 31-22 (10). Одна запись из каталога страниц определяет 4 МБайт адресного пространства.
- Номер записи в *таблице страниц* (номер страницы в таблице страниц, см. ниже) — биты 21-12 (10). Одна запись из таблицы страниц определяет 4 КБайт адресного пространства.
- Смещение в странице — биты 11-0 (12).

При использовании страниц по 4 МБайт вторая часть отсутствует. Смещение же в странице определяют биты 21-0 (22).

Принцип работы

Линейная адресация доступна только в защищённом режиме. Для её включения необходимо установить бит PG в регистре CR0. Предварительно необходимо создать в памяти *каталог страниц* (англ. *Page Directory*, PD) и *таблицы страниц* (англ. *Page Table*, PT), после чего в регистр CR3 загрузить физический адрес *каталога*

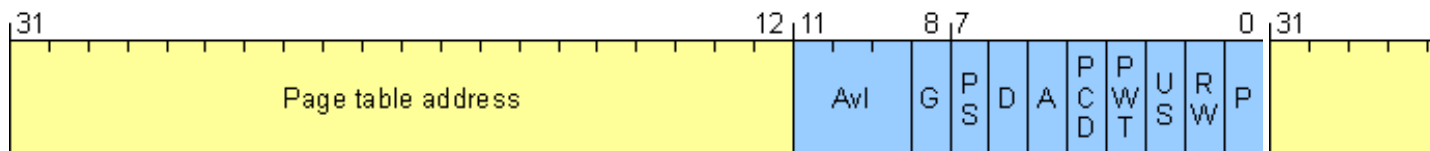


страниц.

Каталог и таблицы страниц

Обе эти структуры представляют собой таблицы элементов каталога и таблицы страниц (англ. *Page Directory Entry*, PDE и англ. *Page Table Entry*, PTE) страницы памяти по 4 КБайт.

Оба элемента занимают по 4 байта (32 бита) и имеют похожую структуру:



В жёлтых полях (Page table address, Page address) записаны старшие 20 бит адреса *таблицы страниц* и *страницы* соответственно (младшие 12 бит физического адреса всегда равны нулю — не забывайте о выравнивании).

Три бита Avl — это биты, отданные системе. В них можно записать всё что угодно.

Описание флагов:

- Бит P (англ. *Present*) определяет наличие данной страницы или таблицы страниц в физической памяти. Если он сброшен, то процессор записывает *линейный адрес* отсутствующей страницы ^[2] в регистр CR2 и передаёт управление обработчику #PF, который должен загрузить страницу в память (или создать её) и установить этот бит.
- Бит RW (англ. *Read/Write*) определяет, можно ли в эту страницу что-то писать (1 — можно, 0 — нельзя).
- Бит US (англ. *User/Supervisor*) разрешает коду с CPL=3 (код пользователя) обращаться к этой странице (при US=1).
- Бит PWT (англ. *Page write through*) — запрещение кэша записи (немедленная запись). Используется для управления кэшированием данной страницы. Если он установлен, то запись происходит непосредственно в оперативную память. Актуально обычно в многоядерных машинах.

- Бит PCD (англ. *Page cache disable*) – запрещение кэширования этой страницы. При обращении к такой странице, она *не заносится* в кэш.
- Бит A (англ. *Accessed*). Аналогично биту A в дескрипторе сегмента, этот бит никак не влияет на работу со страницей. Он просто устанавливается процессором при первом же обращении к этой странице (чтение, запись, выполнение).
- Бит D (англ. *Dirty*, букв. *грязный*) используется только в элементах *таблицы страниц* (PTE) для отслеживания изменений страницы. Аналогично биту A, устанавливается процессором, но только *при записи* на эту страницу.
- Бит PS (англ. *Page size*, *только каталог страниц*) определяет размер страницы. Если сброшен, то этот элемент указывает на таблицу страниц с размером страниц 4 КБайт. Если установлен, то элемент указывает на страницу размером 4 МБайт при 32-битной физической адресации или 2 МБайт при 36-битной. **Важно!** Работает только при установленном бите PSE в регистре CR4.
- Бит PAT (англ. *Page attribute table*, *только таблица страниц*). **Нет информации**
- Бит G (англ. *Global*). Если этот бит установлен, то адрес страницы (или таблицы страниц) никогда не удаляется из TLB кэша. ^[3]

Ссылки

- <http://www.intel.com/products/processor/manuals/>

Сноски

- [1] В этом случае обработчик #PF лишь *загружает* страницу с накопителя в физическую память. Процессор всё равно обращается только к физической памяти.
- [2] Конкретнее, в CR2 записывается полный адрес (32 бита). Напр. если программа обратилась по адресу 00001543h (то есть ко второй странице (№1) при страницах по 4 КБайт), то в CR2 запишется именно это число
- [3] Удалить из TLB кэша можно любую страницу привилегированной командой INVLPG

Страничная память

Страничная память — способ организации виртуальной памяти, при котором единицей отображения виртуальных адресов на физические является регион постоянного размера (т. н. *страница*).

Поддержка такого режима присутствует в большинстве 32битных и 64битных процессоров. Такой режим является классическим для почти всех современных ОС, в том числе Windows и семейства UNIX. Широкое использование такого режима началось с процессора VAX и ОС VMS с конца 70х годов (по некоторым сведениям, первая реализация). В семействе x86 поддержка появилась с поколения 386, оно же первое 32битное поколение.

Решаемые задачи

- поддержка изоляции процессов и защиты памяти путём создания своего собственного виртуального адресного пространства для каждого процесса
- поддержка изоляции области ядра от кода пользовательского режима
- поддержка памяти «только для чтения» и неисполняемой памяти
- поддержка отгрузки давно не используемых страниц в область подкачки на диске
- поддержка отображённых в память файлов, в том числе загрузочных модулей
- поддержка разделяемой между процессами памяти, в том числе с копированием-по-записи для экономии физических страниц
- поддержка системного вызова `fork()` в ОС семейства UNIX

Концепции

Адрес, используемый в машинном коде, то есть значение указателя, называется «виртуальный адрес».

Адрес, выставяемый процессором на шину, называется «физический адрес».

Процессор содержит в себе небольшой объём сверхбыстрой ассоциативной памяти, т. н. TLB (Translation Lookaside Buffer), в котором содержится преобразование нескольких (часто 64) виртуальных адресов страниц в физические. Все обращения процессора к памяти подлежат трансляции адресов через TLB.

Так как 64 строк таблицы явно недостаточно для реальных задач, в архитектуре используются **таблицы страниц**, размещённые в основной памяти. Каждая таблица страниц сама является страницей с теми же требованиями по выравниваю и тем же размером, и содержит в себе массив входов таблицы страниц (*page table entries* — PTE). Широко используется и отображение самой таблицы страниц как одной из страниц данных для внесения изменений во входы.

Вход таблицы страниц обычно содержит в себе следующую информацию:

- флаг «страница отображена»
- физический адрес
- флаг «страница доступна из режима пользователя». При неустановке данного флага страница доступна только из режима ядра.
- флаг «страница доступна только на чтение». В некоторых случаях используется только для режима пользователя, то есть в режиме ядра все страницы всегда доступны на запись.
- флаг «страница недоступна на исполнение».
- режим использования кэша для страницы. Влияет на тип шинных транзакций, инициируемых процессором при обращении через данный вход. Особенно часто используется для видеопамати (комбинированная запись) и для отображенных в память регистров устройств (полное отсутствие кэширования).

Так как число входов в одной таблице ограничено размером входа и размером страницы, используется многоуровневая организация таблиц, часто 2 или 3 уровня, иногда 4 уровня (для 64-х разрядных архитектур). В случае 2 уровней используется «директория» страниц, имеющая в себе входы, указывающие на физические адреса таблиц страниц. Таблицы содержат в себе входы, указывающие уже на страницы данных. В случае 3 уровней возникает еще и супер-директория, содержащая в себе входы, указывающие на несколько директорий.

Старшие биты виртуального адреса понимаются как номер входа в директорию, средние — как номер входа в таблицу, младшие (адрес внутри страницы) попадают в физический адрес без трансляции.

Формат входов таблиц, их размер, размер страницы и организация таблиц зависит от типа процессора, а иногда и от режима его работы.

Например, x86 использует 32битные PTE, 32битные виртуальные адреса, 4КВ страницы, 1024 входа в таблицу, двухуровневые таблицы, старшие 10 бит виртуального адреса — номер входа в директорию, следующие 10 — номер входа в таблицу, младшие 12 — адрес внутри страницы.

Процессор x86 в режиме PAE (Physical Address Extension) и в режиме x86_64 (long mode), использует 64битные PTE (из них реально задействованы не все биты физического адреса, от 36 в PAE до 48 в некоторых x86_64), 32битные виртуальные адреса, 4КВ страницы, 512 входов в таблицу, трехуровневые таблицы с 4 директориями и 4 входами в супер-директорию, старшие 2 бита виртуального адреса — номер входа в супер-директорию, следующие 9 — в директорию, следующие 9 — в таблицу.

Физический адрес директории или же супер-директории загружен в один из управляющих регистров процессора.

Если обращение к памяти не может быть оттранслировано через TLB, то микрокод процессора обращается к таблицам страниц и пытается загрузить PTE оттуда в TLB. Если и после такой попытки сохранились проблемы, то процессор исполняет специальное прерывание, называемое «отказ страницы» (page fault). Обработчик этого прерывания находится в подсистеме виртуальной памяти ядра ОС.

Некоторые процессоры (MIPS) не имеют обращающегося к таблице микрокода, и генерируют отказ страницы сразу после отказа в TLB, обращение к таблице и ее интерпретация возлагаются уже на обработчик отказа страницы. Это лишает таблицы страниц требования соответствовать жёстко заданному на уровне аппаратуры формату.

Причины отказа страницы (*page fault*):

- не существует таблицы, отображающей данный регион
- PTE не имеет взведённого флага «страница отображена».
- попытка обратиться из пользовательского режима к странице «только для ядра».
- попытка записи в страницу «только для чтения».
- попытка исполнения кода из страницы «исполнение запрещено».

Обработчик отказов в ядре может загрузить нужную страницу из файла или же из области подкачки, может создать доступную на запись копию страницы «только для чтения», а может и возбудить исключительную ситуацию (в терминах UNIX — сигнал SIGSEGV) в данном процессе.

Каждый процесс имеет свой собственный набор таблиц страниц. Регистр «директория страниц» перегружается при каждом переключении контекста процесса. Также необходимо сбросить ту часть TLB, которая относится к данному процессу.

В большинстве случаев ядро ОС помещается в то же адресное пространство, что и процессы, для него резервируются верхние 1-2 гигабайта 32битного адресного пространства каждого процесса. Это делается с целью избежать переключения таблиц страниц при входе в ядро и выходе из него. Страницы ядра помечаются как недоступные для кода режима пользователя.

Память региона ядра часто совершенно одинакова для всех процессов, однако — например, регион Windows, где находится подсистема графики и драйвер видео — некоторые подрегионы региона ядра могут быть

различными для разных групп процессов (сессий).

Так как память ядра одинакова у всех процессов, соответствующие ей TLB не нужно перегружать после переключения процесса. Для этой оптимизации x86 поддерживает флаг «глобальный» у PTE.

Так как память ядра есть одинаково отображенный регион строго определенного большого размера, есть смысл сэкономить на таблицах страниц путём создания напрямую отображённого входа в директорию страниц, разом отображающего сразу крупный регион без использования небольших страниц. Такие входы, называемые 4 мегабайтными страницами и отображающие сразу 4М, появились в Pentium.

При использовании PAE возможно создавать 2 мегабайтные страницы. См. также PSE.

В архитектуре x86_64 возможно использовать страницы размером 4 килобайта (4096 байт), 2 мегабайта (2,097,152 байт), и (в самых современных процессорах AMD64) 1 гигабайт (1,073,741,824 байт).

Отображаемые в память файлы

Обработчик отказа страницы в ядре способен прочитывать данную страницу из файла.

Это приводит к возможности легкой реализации отображенных в память файлов. Концептуально это то же, что выделение памяти и чтение в нее отрезка файла, с той разницей, что чтение осуществляется неявно «по требованию», выраженному отказом страницы при попытке обращения к ней.

Вторым преимуществом такого подхода является — в случае отображения «только для чтения» — разделение одной и той же физической памяти между всеми процессами, отображающими данный файл (выделенная же память своя у каждого процесса).

Третьим преимуществом является возможность «забывания» (discard) некоторых отображенных страниц без выгрузки их в область подкачки, обязательной для выделенной памяти. В случае повторной потребности в странице она может быть быстро загружена из файла снова.

Четвертым преимуществом является не-использование дискового кэша в этом режиме, что означает экономию на копировании данных из кэша в запрошенный регион. Преимущества дискового кэша, оптимизирующего операции небольшого размера, а также повторное чтение одних и тех же данных, полностью исчезают при чтениях целых страниц и тем более их групп, недостаток же в виде обязательного лишнего копирования — сохраняется.

Отображаемые в память файлы используются в ОС Windows, а также ОС семейства UNIX, для загрузки исполняемых модулей и динамических библиотек. Они же используются утилитой GNU grep для чтения входящего файла, а также для загрузки шрифтов в ряде графических подсистем.

Страничная и сегментная виртуальная память

Огромным достоинством страничной виртуальной памяти по сравнению с сегментной является отсутствие «ближних» и «дальних» указателей.

Наличие таких концепций в программировании уменьшает применимость арифметики указателей, и приводит к огромным проблемам с переносимостью кода с/на такие архитектуры. Так, например, значительная часть ПО с открытым кодом изначально разрабатывалась для бессегментных 32битных платформ со страничной памятью и не может быть перенесена на сегментные архитектуры без серьезной переработки.

Кроме того, сегментные архитектуры имеют тяжелейшую проблему $SS \neq DS$, широко известную в начале 90х годов в программировании под 16битные версии Windows. Эта проблема приводит к затруднениям в реализации динамических библиотек, ибо они имеют свой собственный DS, и SS текущего процесса, что приводит к невозможности использования «ближних» указателей в них. Также наличие своего собственного DS в библиотеках требует устанавливающих правильное значение DS заплаток (MakeProcInstance) для обратных вызовов из библиотеки в вызвавшее приложение.

Виртуальная память и дисковый кэш

Поддержка файлов, отображенных в память, требует поддержки ядром ОС структуры «совокупность физических страниц, содержащих в себе отрезки данного файла». Отображение файла в память делается путем заполнения входов таблиц ссылками на страницы данной структуры.

Совершенно очевидно, что данная структура является уже готовым дисковым кэшем. Использование ее в качестве кэша также решает проблему когерентности файла, доступного через read/write, и его же, отображенного в память.

Таким образом, пути кэшированного ввода/вывода в дисковый файл (FsRtlCopyRead в Windows и аналогичная ей generic_file_read() в Linux) реализуются как копирования данных в физические страницы, отображенные на файл.

Такая организация кэша является единственной в Windows, эта ОС вообще не имеет классического блочного кэша диска. Метаданные файловых систем кэшируются путем создания лже-файлов (IoCreateStreamFileObject) и создания страничного кэша для них.

Соображения безопасности

Первоначально архитектура x86 не имела флага «страница недоступна на исполнение» (NX).

Поддержка данного флага появилась в архитектуре x86 как часть режима PAE (Physical Address Extension) в поколении Pentium 4, под большим давлением со стороны специалистов по безопасности (см. архивы NTBugTraq). Установка данного флага на страницах стека и кучи (heap) позволяет реализовать аппаратно защиту от исполнения данных, что делает невозможной работу многих разновидностей вредоносного ПО, в том числе, злонамеренную эксплуатацию многих брешей в Internet Explorer (брешь декабря 2008 года, см. MS knowledge base, не может быть задействована в случае включенной DEP).

Поддержка PAE в Windows, дающая возможность включения защиты от исполнения данных, появилась в Windows 2000, она включена по умолчанию в серверных версиях Windows и отключена в клиентских.

Поддержка памяти свыше 4Гб в Windows

Устройства PCI, в том числе память видеокарты, обычно поддерживают только 32битные адреса. Следовательно, им должны быть выданы физические адреса ниже отметки 4Гб. Эта «апертура» уменьшает объем видимой физической памяти ниже отметки 4Гб до примерно 3.2Гб. Остальная часть физической памяти переотображается контроллером выше отметки 4Гб.

Для любого обращения к памяти свыше отметки 4Гб (то есть более, чем примерно 3.2Гб) требуется поддержка контроллером (то есть северным мостом чипсета) такой конфигурации. Современные чипсеты (например Intel G33) такую поддержку имеют.

Также требуется настройка BIOS под названием *memory remapping*, отображающая регион [3.2...4] на [4..4.8].

Процессор x86 вне режима PAE использует 32битные PTE и физические адреса, то есть ему недоступно ничто, находящееся выше отметки 4Гб (см. также PSE-36 об одном из вариантов обхода данного ограничения). Таким образом, для использования памяти более, чем примерно 3.2Гб в ОС она должна поддерживать PAE. Для Windows это опция загрузки, для Linux — опция построения ядра.

Кроме того, Microsoft принудительно отключила поддержку физических адресов выше 4Гб по политико-маркетинговым соображениям в следующих ОС:

- 32битная Windows XP
- 32битный Windows Server 2003 Web Edition
- 32битная Windows Vista (подключение поддержки требует набора команд в командной строке: «BCDEdit /set PAE forceenable», «BCDEdit /set nolowmem on» - не работает, ничего не изменяя, ни в Windows Vista,

ни в Windows 7, возможно, что это миф)

- 32битная Windows 7

Поддержка физических адресов выше 4Гб имеется в следующих версиях:

- все 64битные версии
- 32битная Windows Vista SP1 (поддержка выключена по умолчанию, её подключение требует набора команд в командной строке)
- 32битный Windows Server 2003, отличный от Web Edition
- 32битный Windows Server 2008

Таким образом, для того, чтобы использовать память выше 3.2Гб в Windows, нужны а) поддержка чипсетом б) правильные настройки BIOS в) правильная версия Windows г) правильная опция загрузки (с поддержкой PAE).

Тем не менее даже в «урезанной» версии Windows, не поддерживающей адреса выше 4Гб, имеет смысл всегда использовать PAE, ибо (см. выше) защита от исполнения данных (DEP) тоже требует PAE. При включении PAE может перестать работать небольшая часть ПО, например, эмулятор windows mobile.

См. также

- Линейная адресация памяти в процессорах семейства x86
- Защищённый режим

ext2

ext2 или **2я расширенная файловая система** — файловая система для ядра Linux. Она была разработана Rémy Card'om в качестве замены для extended file system. Она достаточно быстра для того, чтобы служить эталоном в тестах производительности файловых систем. Она не является журналируемой файловой системой и это её главный недостаток. Развитием ext2 стала журналируемая файловая система ext3, полностью совместимая с ext2.

История

На заре развития Linux использовала файловую систему ОС Minix. Эта файловая система была довольно стабильна, но была 16-разрядной и как следствие имела жёсткое ограничение в 64 Мегабайта на раздел. Также присутствовало ограничение имени файла: оно составляло 14 символов. Эти и не только ограничения повлекли появление в апреле 1992 года «расширенной файловой системы» (extended file system), решавшей 2 главные проблемы Minix. Новая файловая система расширила ограничения на размер файла до 2 гигабайт и установила предельную длину имени файла в 255 символов. Но она всё равно имела проблемы: не было поддержки раздельного доступа, временных меток модификации данных.

Решением всех проблем стала новая файловая система, разработанная в январе 1993 года. В ext2 были сразу реализованы соответствующие стандарту POSIX списки контроля доступа ACL и расширенные атрибуты файлов.

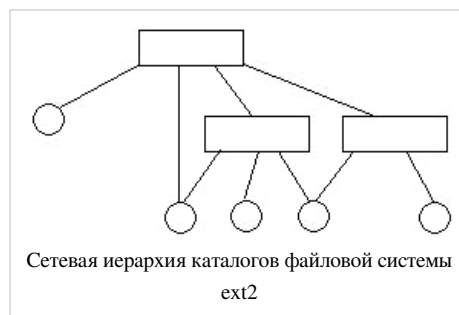
Логическая организация файловой системы ext2

Граф, описывающий иерархию каталогов файловой системы ext2, представляет собой сеть, это достигается тем, что один файл может входить сразу в несколько каталогов.

Все типы файлов имеют символьные имена. В иерархически организованных файловых системах обычно используются три типа имен — файлов: простые, составные и относительные. Не является исключением и «вторая расширенная файловая система».

Ограничения на простое имя состоят в том что, его длина не должна превышать 255 символов, а также в имени не должны

присутствовать символ NUL и '/'. Ограничения на символ NUL связаны с представлением строк на языке Си, а на символ '/' с тем, что он используется как разделительный символ между каталогами. Полное имя представляет собой цепочку простых символьных имен всех каталогов, через которые проходит путь от корня до данного файла. В файловой системе ext2 файл может входить в несколько каталогов, а значит, иметь несколько полных имен; здесь справедливо соответствие «один файл — много полных имен». В любом случае полное имя однозначно определяет файл.



Атрибутами файловой системы ext2 являются:

- Тип и права доступа к файлу;
- Владелец, группа;
- Информация о разрешённых операциях доступа к файлу;
- Времена создания, последнего доступа, последнего изменения и время последнего удаления;
- текущий размер файла;
- тип файла;
 - обычный файл;
 - каталог;
 - файл байт-ориентированного устройства;
 - файл блочно-ориентированного устройства;
 - Сокет;
 - именованный канал;
 - символическая ссылка;
- число блоков, занимаемых файлом;
- ACL
- другие

Атрибуты файлов хранятся не в каталогах, как это сделано в ряде простых файловых систем, а в специальных таблицах. В результате каталог имеет очень простую структуру, состоящую всего из двух частей: номера индексного дескриптора и имени файла.

Физическая организация файловой системы ext2

Структура дискового раздела

Как и в любой файловой системе UNIX, в составе ext2 можно выделить следующие составляющие:

- блоки и группы блоков;
- индексный дескриптор;
- суперблок;

Всё пространство раздела диска разбивается на блоки фиксированного размера, кратные размеру сектора — 1024, 2048, 4096 или 8120 байт. Размер блока указывается при создании файловой системы на разделе диска. Меньший размер блока позволяет экономить место на жёстком диске, но также ограничивает максимальный размер файловой системы. Все блоки имеют порядковые номера. С целью уменьшения фрагментации и количества перемещений головок жёсткого диска при чтении больших массивов данных блоки объединяются в группы блоков.

Базовым понятием файловой системы является индексный дескриптор (информационный узел), information node, или inode. Это специальная структура, которая содержит информацию об атрибутах и физическом расположении файла. Inode объединены в таблицу индексных дескрипторов, которая содержится в начале каждой группы блоков

Каждая группа блоков имеет одинаковое строение. Суперблок — основной элемент файловой системы ext2. Он содержит общую информацию о файловой системе:

- общее число блоков и индексных дескрипторов в файловой системе;
- число свободных блоков и индексных дескрипторов в файловой системе;
- размер блока файловой системы;
- количество блоков и индексных дескрипторов в группе;
- размер индексного дескриптора;
- идентификатор файловой системы.



Обобщенная структурная схема ФС ext2

Суперблок находится в 1024 байтах от начала раздела. От целостности суперблока напрямую зависит работоспособность файловой системы. Операционная система создает несколько резервных копий суперблока для возможности его восстановления в случае повреждения. В следующем блоке после суперблока располагается глобальная дескрипторная таблица - описание групп блоков, представляющее собой массив, содержащий общую информацию обо всех группах блоков раздела.

Группа блоков

Все блоки раздел ext2 разделяется на группы блоков. Для каждой группы блоков создается отдельная запись в глобальной дескрипторной таблице, в этой записи хранятся основные параметры:

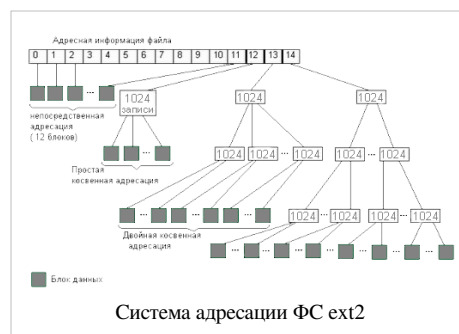
- номер блока битовой карты блоков
- номер блока битовой карты inode
- номер блока таблицы inode
- число свободных блоков в группе
- число inode содержащих каталоги

Битовая карта блоков — это структура, каждый бит которой показывает, отведён ли соответствующий ему блок какому-либо файлу. Если бит равен 1, то блок занят. Аналогичную функцию выполняет битовая карта индексных дескрипторов, показывая какие именно индексные дескрипторы заняты, а какие нет. Ядро Linux,

используя число inode, содержащих каталоги, пытается равномерно распределить inode каталогов по группам, а inode файлов помещает, если это возможно, в группу с родительским каталогом. Все оставшееся место, обозначенное в таблице как данные, отводится для хранения файлов.

Система адресации данных

Система адресации данных — это одна из самых существенных составных частей файловой системы. Именно система адресации позволяет находить нужный файл среди множества как пустых, так и занятых блоков на диске. Файловая система ext2 использует следующую схему адресации блоков файла. Для хранения адреса файла выделено 15 полей, каждое из которых состоит из 4 байт. Если размер файла меньше или равен 12 блокам, то номера этих кластеров непосредственно перечисляются в первых двенадцати полях адреса. Если размер файла превышает 12 блоков, то следующее 13-е поле содержит адрес кластера, в котором могут быть расположены номера следующих блоков файла. Таким образом, 13-й элемент адреса используется для косвенной адресации. При максимальном размере блока равном 4096 байт, 13-й элемент, может содержать до 1024 номеров следующих кластеров данных файла. Если размер файла превышает 12+1024 блоков, то используется 14-е поле, в котором находится номер блока, содержащего 1024 номеров блоков, каждый из которых хранят 1024 номеров блоков данных файла. Здесь применяется уже двойная косвенная адресация. И наконец, если файл включает более $12+1024+1048576 = 1049612$ блоков, то используется последнее 15-е поле для тройной косвенной адресации.



Таким образом, описанная выше система адресации, позволяет при максимальном размере блока 4 Кб иметь файлы размера до 2 терабайт. или больше

См. также

- Список файловых систем
- Сравнение файловых систем
- файловая система ext3

Ссылки

- инструменты пользователя для ext2 ^[1]
- драйвер для MS Windows NT/2000/XP ^[2]
- драйвер для Mac OS X ^[3]

Сноски

- [1] <http://e2fsgprogs.sourceforge.net/ext2.html>
 [2] <http://ext2fsd.sourceforge.net/>
 [3] <http://sourceforge.net/projects/ext2fsx/>

ACL

ACL (англ. *Access Control List* — список контроля доступа, по-английски произносится «экл») — определяет, кто или что может получать доступ к конкретному объекту, и какие именно операции разрешено или запрещено этому субъекту проводить над объектом.

Списки контроля доступа являются основой систем с избирательным управлением доступом.

Введение

В типичных ACL каждая запись определяет субъект воздействия и операцию: например, запись (*Vasya, delete*) в ACL для файла *XYZ* даёт пользователю *Vasya* удалить файл *XYZ*.

В системе с моделью безопасности, основанной на ACL, когда субъект запрашивает выполнение операции над объектом, система сначала проверяет список разрешённых для этого субъекта операций, и только после этого даёт (или не даёт) доступ к запрошенному действию.

При централизованном хранении списков контроля доступа можно говорить о *матрице доступа*, в которой по осям размещены объекты и субъекты, а в ячейках — соответствующие права. Однако в большом количестве систем списки контроля доступа к объектам хранятся отдельно для каждого объекта, зачастую непосредственно с самим объектом.

Традиционные ACL системы назначают права индивидуальным пользователям, и со временем и ростом числа пользователей в системе списки доступа могут стать громоздкими. Вариантом решения этой проблемы является назначения прав группам пользователей, а не персонально. Другим вариантом решения этой проблемы является «Управление доступом на основе ролей», где функциональные подмножества прав к ряду объектов объединяются в «роли», и эти роли назначаются пользователям. Однако, в первом варианте группы пользователей также часто называются ролями.

Файловые системы с ACL

В файловых системах для реализации ACL используется идентификатор пользователя процесса (UID в терминах POSIX).

Список доступа представляет собой структуру данных (обычно таблицу), содержащую записи, определяющие права индивидуального пользователя или группы на специальные системные объекты, такие как программы, процессы или файлы. Эти записи также известны как ACE (англ. *Access Control Entries*) в операционных системах Microsoft Windows и OpenVMS. В операционной системе Linux и Mac OS X большинство файловых систем имеют расширенные атрибуты, выполняющие роль ACL. Каждый объект в системе содержит указатель на свой ACL. Привилегии (или полномочия) определяют специальные права доступа, разрешающие пользователю **читать** из (англ. *read*), **писать** в (англ. *write*), или **исполнять** (англ. *execute*) объект. В некоторых реализациях ACE могут определять право пользователя или группы на изменение ACL объекта.

Концепции ACL в разных операционных системах различаются, несмотря на существующий «стандарт» POSIX. (Проекты безопасности POSIX, .1e и .2c, были отозваны, когда стало ясно что они затрагивают слишком обширную область и работа не может быть завершена, но хорошо проработанные части, определяющие ACL, были широко реализованы и известны как «POSIX ACLs».)

Сетевые ACL

В сетях **ACL** представляют список правил, определяющих порты служб или имена доменов, доступных на узле или другом устройстве третьего уровня OSI, каждый со списком узлов и/или сетей, которым разрешен доступ к сервису. Сетевые ACL могут быть настроены как на обычном сервере, так и на маршрутизаторе и могут управлять как входящим, так и исходящим трафиком, в качестве межсетевого экрана.

См. также

- Информационная безопасность

Ссылки

- POSIX Access Control Lists on Linux ^[1]
- RSBAC Access Control Lists on Linux ^[2]
- Generic Access Control Lists for PHP/Perl ^[3]
- C2-Wiki Discussion and Relational Implementation ^[4]
- Information Security Definitions: ACL ^[5]
- Easy and detailed ACL howto for linux ^[6]
- Security Briefs: Access Control List Editing in .NET ^[7]
- MS Windows .NET ACL Technology ^[8]
- What could have been IEEE 1003.1e/2c ^[9]
- Apple Mac OS X Server version 10.4+ *File Services Administration Manual* (see pages 16-26) ^[10]
- ACLbit — ACL Backup and Inspect Tool for Linux ^[11]

Сноски

- [1] <http://www.suse.de/~agruen/acl/linux-acls/online/>
- [2] http://www.rsba.org/documentation/rsbac_handbook/security_models#access_control_lists_acl
- [3] <http://phpgacl.sourceforge.net>
- [4] <http://www.c2.com/cgi/wiki?AccessControllist>
- [5] http://searchsecurity.techtarget.com/sDefinition/0,,sid14_gci213757,00.html
- [6] <http://wiki.kaspersandberg.com/doku.php?id=howtos:acl>
- [7] <http://msdn.microsoft.com/msdnmag/issues/05/03/SecurityBriefs/default.aspx>
- [8] <http://msdn2.microsoft.com/en-us/library/ms229742.aspx>
- [9] <http://wt.xpilot.org/publications/posix.1e/download.html>
- [10] http://manuals.info.apple.com/en/File_Services_v10.4.pdf
- [11] <http://aclbit.sourceforge.net/>

Символьная ссылка

Символьная ссылка (также **симлинк** от англ. *Symbolic link*, **символическая ссылка**) — специальный файл в UNIX-подобных операционных системах, для которого в файловой системе не хранится никакой информации, кроме одной текстовой строки. Эта строка трактуется как путь к файлу, который должен быть открыт при попытке обратиться к данной ссылке. Символьная ссылка занимает ровно столько места на файловой системе, сколько требуется для записи её содержимого (нормальный файл занимает как минимум один блок раздела).

Целью ссылки может быть любой объект — например, другая ссылка, или даже несуществующий файл (в последнем случае при попытке открыть его должно выдаваться сообщение об отсутствии файла). Ссылка, указывающая на несуществующий файл, называется *висячей*.

Практически символьные ссылки используются для более удобной организации структуры файлов на компьютере, так как позволяют одному файлу или каталогу иметь несколько имён и свободны от некоторых ограничений, присущих жёстким ссылкам (последние действуют только в пределах одного раздела и не могут ссылаться на каталоги).

Windows

В отличие от жестких ссылок, могут указывать на файлы и директории в других томах.

Виды символьных ссылок в Windows:

- Символьные связи (junction points) — доступна с Windows 2000 (файловая система NTFS 5). Может указывать только на директории.

Команда — `linkd` (Microsoft Windows Resource Kit)

- Символическая ссылка (symbolic links) — доступна с Windows Vista. Может указывать и на файлы, и на директории.

Команда — `mklink`

Создание символической ссылки.

`MKLINK [[/D] | [/H] | [/J]] Ссылка Назначение`

/D	Создание символической ссылки на каталог. По умолчанию создается символическая ссылка на файл.
/H	Создание жесткой связи вместо символической ссылки.
/J	Создание соединения для каталога.
Ссылка	Имя новой символической ссылки.
Назначение	Путь (относительный или абсолютный), на который ссылается новая ссылка.

Unix

```
ln -s файл1 файл2
# создаётся «символьная» ссылка (symbolic link)
```

Если вы создаёте символическую ссылку (используя ключ «-s»), то при этом файла «файл1» может и не существовать. Символическая ссылка с именем «файл2» всё равно будет создана.

Подробнее: [ln](#) ^[1]

или же:

```
man ln
```

См. также

- Жёсткая ссылка
- ln
- Ярлык

Ссылки

- Ссылки в Windows, символические и не только ^[2]
- NTFS-Link ^[3] — дополнение к оболочке Windows, позволяющее создавать символьные и жёсткие ссылки на томах NTFS. Существующие символьные ссылки помечаются в Проводнике маленькой дополнительной иконкой.
- NTFS Links ^[4] — плагин для файлового менеджера Total Commander, позволяющий создавать жёсткие и символьные ссылки на NTFS-разделах системы Windows из Total Commander.

Сноски

[1] http://ru.wikipedia.org/wiki/Ln_%28UNIX%29

[2] <http://habrahabr.ru/blogs/windows/50878/>

[3] <http://www.elsdoerfer.info/ntfslink/>

[4] <http://wincmd.ru/plugring/ntfslinks.html>

Именованный канал

В программировании **именованный канал** или **именованный конвейер** (англ. *named pipe*) — расширение понятия конвейера в Unix и подобных ОС, один из методов межпроцессного взаимодействия. Это понятие также существует и в Microsoft Windows, хотя там его семантика существенно отличается. Традиционный канал — «безымянный», потому что существует анонимно и только во время выполнения процесса. Именованный канал — существует в системе и после завершения процесса. Он должен быть «отсоединён» или удалён когда уже не используется. Процессы обычно подсоединяются к каналу для осуществления взаимодействия между процессами.

Именованные каналы в Unix

Вместо традиционного, безымянного конвейера оболочки (англ. *shell pipeline*), именованный канал создаётся явно с помощью `mknod`^[1] или `mkfifo`^[2], и два различных процесса могут обратиться к нему по имени.

Например, можно создать канал и настроить `gzip` на сжатие того, что туда попадает:

```
mkfifo pipe
gzip -9 -c < pipe > out
```

Параллельно, в другом процессе можно выполнить:

```
cat file > pipe
```

что приведёт к сжатию передаваемых данных `gzip`-ом.

Именованные каналы в Windows

В Windows дизайн именованных каналов смещён к взаимодействию «клиент-сервер», и они работают во многом как сокеты: помимо обычных операций чтения и записи, именованные каналы в Windows поддерживают явный «пассивный» режим для серверных приложений (для сравнения: Unix domain socket). Windows 95 поддерживает клиенты именованных каналов, а системы ветви Windows NT могут служить также и серверами.

К именованному каналу можно обращаться в значительной степени как к файлу. Можно использовать функции Windows API `CreateFile`, `CloseHandle`, `ReadFile`, `WriteFile`, чтобы открывать и закрывать канал, выполнять чтение и запись. Функции стандартной библиотеки Си такие как `fopen`, `fread`, `fwrite` и `fclose`, тоже можно использовать, в отличие от сокетов Windows (англ.), которые не реализуют использование стандартных файловых операций в сети. Интерфейс командной строки (как в Unix) отсутствует.

Именованные каналы — не существуют постоянно и не могут, в отличие от Unix, быть созданы как специальные файлы в произвольной доступной для записи файловой системе, но имеют временные имена (освобождаемые после закрытия последней ссылки на них), которые выделяются в корне файловой системы именованных каналов (англ. *named pipe filesystem*, *NPFS*) и монтируются по специальному пути «`\\.\pipe\`» (т. е. у канала под названием «foo» полное имя будет «`\\.\pipe\foo`»). Анонимные каналы, использующиеся в конвейерах — это на самом деле именованные каналы со случайным именем.

Именованные каналы обычно не доступны непосредственно пользователю, но есть существенные исключения. Например, средство виртуализации рабочих станций VMWare может открывать эмулируемый последовательный порт для главной системы как именованный канал, а отладчик уровня ядра `kd` от Microsoft поддерживает именованные каналы в качестве средства сообщения при отладке (фактически, так как `kd` обычно требует подключения к целевому компьютеру по последовательному порту, VMWare и `kd` можно соединить вместе для отладки драйверов устройств на одном компьютере). Обе программы требуют от

пользователя указания имён канала в виде «\\.\pipe\имя».

Именованные каналы в Windows NT могут наследовать контекст безопасности.

Именованные каналы в сетях Windows

Именованные каналы — это также сетевой протокол в SMB, основанный на использовании особой части межпроцессного взаимодействия (IPC). IPC в SMB может бесшовно и прозрачно передавать контекст аутентификации пользователя на другую сторону именованного канала. Наследование аутентификации для именованных каналов Windows NT для пользователя и разработчика настолько прозрачно, что почти незаметно, в связи с чем его часто неправильно понимают.

См. также

- Анонимный канал

Внешние ссылки

- The Linux Programmer's Guide: Named Pipes ^[3]
- Linux Journal: Introduction to Named Pipes ^[4]
- MSDN Library: Named Pipes ^[5]
- Programing with named pipes (from Sun and for Solaris, but a general enough intro for anyone) ^[6]

Сноски

[1] <http://www.die.net/doc/linux/man/man1/mknod.1.html>

[2] <http://www.die.net/doc/linux/man/man1/mkfifo.1.html>

[3] <http://www.tldp.org/LDP/lpg/node15.html>

[4] <http://www2.linuxjournal.com/article/2156>

[5] <http://msdn2.microsoft.com/en-us/library/aa365590.aspx>

[6] http://developers.sun.com/solaris/articles/named_pipes.html

Сокет (программный интерфейс)

Сокеты (англ. *socket* — углубление, гнездо, разъём) — название программного интерфейса для обеспечения обмена данными между процессами. Процессы при таком обмене могут исполняться как на одной ЭВМ, так и на различных ЭВМ, связанных между собой сетью. Сокет — абстрактный объект, представляющий конечную точку соединения.

Следует различать **клиентские** и **серверные сокеты**. Клиентские сокеты грубо можно сравнить с оконечными аппаратами телефонной сети, а серверные — с коммутаторами. Клиентское приложение (например, браузер) использует только клиентские сокеты, а серверное (например, веб-сервер, которому браузер посылает запросы) — как клиентские, так и серверные сокеты.

Интерфейс сокетов впервые появился в BSD Unix. Программный интерфейс сокетов описан в стандарте POSIX.1 и в той или иной мере поддерживается *всеми* современными операционными системами.

Сокет на сленге системных администраторов означает комбинацию IP-адреса и номера порта, например 10.10.10.10:80.

Принципы сокетов

Каждый процесс может создать *слушающий* сокет (серверный сокет) и *привязать* его к какому-нибудь порту операционной системы (тем не менее, в UNIX непривилегированные процессы не могут использовать порты меньше 1024). Слушающий процесс обычно находится в цикле ожидания, то есть просыпается при появлении нового соединения. При этом сохраняется возможность просто проверить наличие соединений на данный момент, установить тайм-аут для операции и так далее.

Каждый сокет имеет свой адрес. ОС семейства UNIX могут поддерживать много типов адресов, но обязательными являются INET-адрес и UNIX-адрес. Если привязать сокет к UNIX-адресу, то просто будет создан специальный файл (*файл сокета*) по заданному пути, через который смогут общаться любые локальные процессы путём простого чтения/записи из него. Сокеты типа INET доступны из сети и требуют выделения номера порта.

Обычно клиент явно *подсоединяется* к слушателю, после чего любое чтение или запись через его файловый дескриптор будут на самом деле передавать данные между ним и сервером.

См. также

- Интерфейс транспортного уровня

Переносимый набор символов

Переносимый набор символов (англ. *Portable Character Set*) — набор из 103 символов, которые, согласно стандарту POSIX, должны присутствовать в любой используемой кодировке. Включает в себя все печатные символы US-ASCII и часть управляющих. Является базовым алфавитом для практически всех современных языков программирования.

На коды символов из переносимого набора накладываются следующие ограничения:

- NUL должен быть символом, где все биты установлены в 0.
- Коды десятичных цифр 0—9 должны идти в возрастающем порядке, причём коды двух соседних цифр должны отличаться на единицу.
- Коды всех символов из этого набора должны быть представимы одним байтом^[источник не указан 35] (т. е. не должны превышать 255^[источник не указан 35]).
- Если для представления символов в языке C используется тип **char**, то коды символов из этого набора обязательно должны быть неотрицательными (это значит, что при использовании EBCDIC тип **char** должен быть эквивалентен **unsigned char**).

название	глиф	строка Си	Юникод	название в Юникоде
NUL		\0	U+0000	NULL (NUL)
alert		\a	U+0007	BELL (BEL)
backspace		\b	U+0008	BACKSPACE (BS)
tab		\t	U+0009	CHARACTER TABULATION (HT)
carriage-return		\r	U+000D	CARRIAGE RETURN (CR)
newline		\n	U+000A	LINE FEED (LF)
vertical-tab		\v	U+000B	LINE TABULATION (VT)
form-feed		\f	U+000C	FORM FEED (FF)
space			U+0020	SPACE
exclamation-mark	!	!	U+0021	EXCLAMATION MARK
quotation-mark	"	\"	U+0022	QUOTATION MARK
number-sign	#	#	U+0023	NUMBER SIGN
dollar-sign	\$	\$	U+0024	DOLLAR SIGN
percent-sign	%	%	U+0025	PERCENT SIGN
ampersand	&	&	U+0026	AMPERSAND
apostrophe	'	\'	U+0027	APOSTROPHE
left-parenthesis	((U+0028	LEFT PARENTHESIS
right-parenthesis))	U+0029	RIGHT PARENTHESIS
asterisk	*	*	U+002A	ASTERISK
plus-sign	+	+	U+002B	PLUS SIGN
comma	,	,	U+002C	COMMA
hyphen	-	-	U+002D	HYPHEN-MINUS
period	.	.	U+002E	FULL STOP

slash	/	/	U+002F	SOLIDUS
zero	0	0	U+0030	DIGIT ZERO
one	1	1	U+0031	DIGIT ONE
two	2	2	U+0032	DIGIT TWO
three	3	3	U+0033	DIGIT THREE
four	4	4	U+0034	DIGIT FOUR
five	5	5	U+0035	DIGIT FIVE
six	6	6	U+0036	DIGIT SIX
seven	7	7	U+0037	DIGIT SEVEN
eight	8	8	U+0038	DIGIT EIGHT
nine	9	9	U+0039	DIGIT NINE
colon	:	:	U+003A	COLON
semicolon	;	;	U+003B	SEMICOLON
less-than-sign	<	<	U+003C	LESS-THAN SIGN
equals-sign	=	=	U+003D	EQUALS SIGN
greater-than-sign	>	>	U+003E	GREATER-THAN SIGN
question-mark	?	?	U+003F	QUESTION MARK
commercial-at	@	@	U+0040	COMMERCIAL AT
A	A	A	U+0041	LATIN CAPITAL LETTER A
B	B	B	U+0042	LATIN CAPITAL LETTER B
C	C	C	U+0043	LATIN CAPITAL LETTER C
D	D	D	U+0044	LATIN CAPITAL LETTER D
E	E	E	U+0045	LATIN CAPITAL LETTER E
F	F	F	U+0046	LATIN CAPITAL LETTER F
G	G	G	U+0047	LATIN CAPITAL LETTER G
H	H	H	U+0048	LATIN CAPITAL LETTER H
I	I	I	U+0049	LATIN CAPITAL LETTER I
J	J	J	U+004A	LATIN CAPITAL LETTER J
K	K	K	U+004B	LATIN CAPITAL LETTER K
L	L	L	U+004C	LATIN CAPITAL LETTER L
M	M	M	U+004D	LATIN CAPITAL LETTER M
N	N	N	U+004E	LATIN CAPITAL LETTER N
O	O	O	U+004F	LATIN CAPITAL LETTER O
P	P	P	U+0050	LATIN CAPITAL LETTER P
Q	Q	Q	U+0051	LATIN CAPITAL LETTER Q
R	R	R	U+0052	LATIN CAPITAL LETTER R
S	S	S	U+0053	LATIN CAPITAL LETTER S
T	T	T	U+0054	LATIN CAPITAL LETTER T
U	U	U	U+0055	LATIN CAPITAL LETTER U

V	V	V	U+0056	LATIN CAPITAL LETTER V
W	W	W	U+0057	LATIN CAPITAL LETTER W
X	X	X	U+0058	LATIN CAPITAL LETTER X
Y	Y	Y	U+0059	LATIN CAPITAL LETTER Y
Z	Z	Z	U+005A	LATIN CAPITAL LETTER Z
left-square-bracket	[[U+005B	LEFT SQUARE BRACKET
backslash	\	\	U+005C	REVERSE SOLIDUS
right-square-bracket]]	U+005D	RIGHT SQUARE BRACKET
circumflex	^	^	U+005E	CIRCUMFLEX ACCENT
underscore	_	_	U+005F	LOW LINE
grave-accent	`	`	U+0060	GRAVE ACCENT
a	a	a	U+0061	LATIN SMALL LETTER A
b	b	b	U+0062	LATIN SMALL LETTER B
c	c	c	U+0063	LATIN SMALL LETTER C
d	d	d	U+0064	LATIN SMALL LETTER D
e	e	e	U+0065	LATIN SMALL LETTER E
f	f	f	U+0066	LATIN SMALL LETTER F
g	g	g	U+0067	LATIN SMALL LETTER G
h	h	h	U+0068	LATIN SMALL LETTER H
i	i	i	U+0069	LATIN SMALL LETTER I
j	j	j	U+006A	LATIN SMALL LETTER J
k	k	k	U+006B	LATIN SMALL LETTER K
l	l	l	U+006C	LATIN SMALL LETTER L
m	m	m	U+006D	LATIN SMALL LETTER M
n	n	n	U+006E	LATIN SMALL LETTER N
o	o	o	U+006F	LATIN SMALL LETTER O
p	p	p	U+0070	LATIN SMALL LETTER P
q	q	q	U+0071	LATIN SMALL LETTER Q
r	r	r	U+0072	LATIN SMALL LETTER R
s	s	s	U+0073	LATIN SMALL LETTER S
t	t	t	U+0074	LATIN SMALL LETTER T
u	u	u	U+0075	LATIN SMALL LETTER U
v	v	v	U+0076	LATIN SMALL LETTER V
w	w	w	U+0077	LATIN SMALL LETTER W
x	x	x	U+0078	LATIN SMALL LETTER X
y	y	y	U+0079	LATIN SMALL LETTER Y
z	z	z	U+007A	LATIN SMALL LETTER Z
left-brace	{	{	U+007B	LEFT CURLY BRACKET
vertical-line			U+007C	VERTICAL LINE

right-brace	}	}	U+007D	RIGHT CURLY BRACKET
tilde	~	~	U+007E	TILDE

Кодировки символов



Основы →	алфавит • текст (файл • данные) • набор символов • конверсия
Исторические кодировки →	Докомп.: семафорная (Макарова) • Морзе • Бодо • MTK-2 Комп.: 6 бит • УПП • RADIX-50 • EBCDIC (ДКОИ-8) • КОИ-7 • ISO 646
совре- менное 8-битноепредстав- ление	символы → ASCII (управляющие • печатные) не-ASCII (псевдографика) Разные → Кириллица: КОИ-8 • ГОСТ 19768-87 • MacCyrillic ISO 8859 → 1(лат.) 2 3 4 5(кир.) 6 7 8 9 10 11 12 13 14 15(€) 16 Windows → 1250 1251(кир.) 1252 1253 1254 1255 1256 1257 1258 WGL4 IBM&DOS → 437 • 850 • 852 • 855 • 866 «альт.» (МИК)
Много-байтные	Традиционные → DBCS (GB2312) • HTML Unicode → UTF-16 • UTF-8 • список символов (кириллица)
Связанные темы →	интерфейс пользователя • раскладки клавиатур • локаль • перевод строки • шрифт • крокозябры • транслит • нестандартные шрифты • текст как изображение Утилиты: iconv • recode

CP437

CP437 (Codepage 437, DOSLatinUS) — кодовая страница, использовавшаяся в первоначальной версии IBM PC с 1981 года. CP437 послужила образцом для множества других кодировок, в том числе альтернативной кодировки для русского языка.

В ПЗУ видеоадаптера IBM PC был зашит шрифт для следующего набора символов:

	.0	.1	.2	.3	.4	.5	.6	.7	.8	.9	.A	.B	.C	.D	.E	.F
0.	NULL 0	☐ 263A	☐ 263B	♥ 2665	♦ 2666	♣ 2663	♠ 2660	• 2022	☐ 25D8	◯ 25CB	☐ 25D9	☐ 2642	☐ 2640	♪ 266A	♫ 266B	☐ 263C
1.	☐ 25BA	☐ 25C4	↕ 2195	!! 203C	¶ B6	§ A7	■ 25AC	☐ 21A8	↑ 2191	↓ 2193	→ 2192	← 2190	☐ 221F	↔ 2194	☐ 25B2	☐ 25BC
2.	20 21	! 22	" 23	# 24	\$ 25	% 26	& 27	' 28	(29	*) 2A	++ 2B	, 2C	- 2D	. 2E	/ 2F	
3.	0 30	1 31	2 32	3 33	4 34	5 35	6 36	7 37	8 38	9 39	: 3A	; 3B	< 3C	= 3D	> 3E	? 3F
4.	@ 40	A 41	B 42	C 43	D 44	E 45	F 46	G 47	H 48	I 49	J 4A	K 4B	L 4C	M 4D	N 4E	O 4F
5.	P 50	Q 51	R 52	S 53	T 54	U 55	V 56	W 57	X 58	Y 59	Z 5A	[5B	\ 5C] 5D	^ 5E	_ 5F
6.	` 60	a 61	b 62	c 63	d 64	e 65	f 66	g 67	h 68	i 69	j 6A	k 6B	l 6C	m 6D	n 6E	o 6F
7.	p 70	q 71	r 72	s 73	t 74	u 75	v 76	w 77	x 78	y 79	z 7A	{ 7B	 7C	} 7D	~ 7E	☐ 2302

8.	Ç	ü	é	â	ä	à	å	ç	ê	ë	è	ï	î	ì	Ä	Å
	C7	FC	E9	E2	E4	E0	E5	E7	EA	EB	E8	EF	EE	EC	C4	C5
9.	É	æ	Æ	ô	ö	ò	û	ù	ÿ	Ö	Ü	ç	£	¥	Pts	f
	C9	E6	C6	F4	F6	F2	FB	F9	FF	D6	DC	A2	A3	A5	20A7	192
A.	á	í	ó	ú	ñ	Ñ	ª	º	¿	□	¬	½	¼	¡	«	»
	E1	ED	F3	FA	F1	D1	AA	BA	BF	2310	AC	BD	BC	A1	AB	BB
B.																
	2591	2592	2593	2502	2524	2561	2562	2556	2555	2563	2551	2557	255D	255C	255B	2510
C.	L	┐	└	┌	—	┴	┴	┴	┴	┴	┴	┴	┴	┴	┴	┴
	2514	2534	252C	251C	2500	253C	255E	255F	255A	2554	2569	2566	2560	2550	256C	2567
D.	┐	┐	┐	┐	┐	┐	┐	┐	┐	┐	┐	■	■	■	■	■
	2568	2564	2565	2559	2558	2552	2553	256B	256A	2518	250C	2588	2584	258C	2590	2580
E.	α	β	Γ	π	Σ	σ	μ	τ	Φ	Θ	Ω	δ	∞	φ	ε	∩
	3B1	DF	393	3C0	3A3	3C3	B5	3C4	3A6	398	3A9	3B4	221E	3C6	3B5	2229
F.	≡	±	≥	≤	∫	∫	÷	≈	°	•	.	√	n	2	■	A0
	2261	B1	2265	2264	2320	2321	F7	2248	B0	2219	B7	221A	207F	B2	25A0	

В CP437, однако, кодовые позиции 0x00—0x1F и 0x7F заняты управляющими символами, т. е. не имеют графического представления.

Большинство современных видеоадаптеров ПК также содержат этот шрифт в ПЗУ. Хотя и были попытки «аппаратной русификации» компьютеров (например, на ЕС ПЭВМ), более популярной оказалась загрузка национальных шрифтов средствами операционной системы.

Интересный факт

В шрифтах видеоадаптеров MDA и VGA все символы имеют ширину 8 пикселей, но на экране отображаются шириной в 9 пикселей. Дополнительная пустая колонка нужна для визуального отделения букв друг от друга. Однако при отображении символов псевдографики дополнительная 9-я колонка не пуста, а повторяет 8-ю, чтобы горизонтальные линии были неразрывны. Оборудование видеокарт определяет графический символ по его коду. Поэтому, создавая другой шрифт, нельзя размещать псевдографику в другом диапазоне, иначе линии будут разорванные.

Внешние ссылки

- Таблица CP437 на сайте www.unicode.org ^[1]
- Таблица дополнительных графических символов, там же ^[2]

Сноски

[1] <http://www.unicode.org/Public/MAPPINGS/VENDORS/MICSFT/PC/CP437.TXT>

[2] <http://www.unicode.org/Public/MAPPINGS/VENDORS/MISC/IBMGRAPH.TXT>

Источники и основные авторы

Гибридное ядро *Источник:* <http://ru.wikipedia.org/w/index.php?oldid=21256293> *Основные авторы:* AVB, Alvelcom, Animist, BooBSD, Carn, Cheops, DEBoshirr, Dm, Indigio, Lego1as, Lucas Novokuznetsk, Silentium, Softy, Stassats, TarzanASG, Tmii, Vald, Victor-435, Zenixan, Кондратьев, Мартин Радю, He A, 15 анонимных правок

Микроядро *Источник:* <http://ru.wikipedia.org/w/index.php?oldid=23254886> *Основные авторы:* AVB, Alvelcom, Amoses, Ann sablina, Aquarius51, Archi fsb, Dsc, Edward Chernenko, Gdn, Goryachev, Gribozavr, Jego.ruS, Lostdj, Lucas Novokuznetsk, Nzeemin, Oxyd, Rebus, Roxis, TarzanASG, Toyota prius 2, User№101, Victor-435, Vort, Xchgall, Yms, Вик1114, Роман Беккер, 25 анонимных правок

Монолитное ядро *Источник:* <http://ru.wikipedia.org/w/index.php?oldid=23520397> *Основные авторы:* Alvelcom, Deineka, Exlex, GermanX, Incnis Mrsi, Loveless, Lucas Novokuznetsk, Rbuj, Redmond Barry, Roxis, Samant, TarzanASG, Victor-435, YuryKirienko, Роман Беккер, 5 анонимных правок

Наноядро *Источник:* <http://ru.wikipedia.org/w/index.php?oldid=21293074> *Основные авторы:* AVRS, Avenger911, Korynd, Lucas Novokuznetsk, Roxis, TarzanASG, Trgl, Victor-435, Роман Беккер, 2 анонимных правок

Экзоядро *Источник:* <http://ru.wikipedia.org/w/index.php?oldid=22816208> *Основные авторы:* Alvelcom, Lucas Novokuznetsk, Rebus, Saø, Stassats, Victor-435, Роман Беккер, С. Л., 8 анонимных правок

Линейная адресация памяти *Источник:* <http://ru.wikipedia.org/w/index.php?oldid=21100497> *Основные авторы:* AVB, Jego.ruS, Vort, 1 анонимных правок

Страничная память *Источник:* <http://ru.wikipedia.org/w/index.php?oldid=22239081> *Основные авторы:* A5b, Jego.ruS, Vort, Четыре тильды, 26 анонимных правок

ext2 *Источник:* <http://ru.wikipedia.org/w/index.php?oldid=23207017> *Основные авторы:* AVRS, Alex Spade, AlexeyKouznetsov, BLISTHRV, Deyteriy, Frantony, Gribozavr, Himself, KamaZee, Kirrun, Rokur, Roxis, Shkutkov Michael, Sorib, Speakus, Stassats, Turbanoff, Vmarvin, Winterheart, 15 анонимных правок

ACL *Источник:* <http://ru.wikipedia.org/w/index.php?oldid=16786725> *Основные авторы:* Alex Kapranoff, Claymore, Deyteriy, DnAp, Dodonov, Loveless, Mashiah Davidson, Maximamax, Michaello, Murtasa, Nashev, Roxis, Ruslanpisarev, Rusnino, Ujo, Urod, VPliousnine, Wind, Yms, Yurik, Yuriy Gavrilov, 13 анонимных правок

Символьная ссылка *Источник:* <http://ru.wikipedia.org/w/index.php?oldid=23404368> *Основные авторы:* ASTAPP, AlexeyKouznetsov, Devilseye, DonRumata, KR, M17, Melancholic, Roxis, Softy, Вадим Мартянов, Гекльберри Финн, 15 анонимных правок

Именованный канал *Источник:* <http://ru.wikipedia.org/w/index.php?oldid=20109603> *Основные авторы:* D V S, DenisKrivosheev, Dionys, Junky, Филатов Алексей, 3 анонимных правок

Сокет (программный интерфейс) *Источник:* <http://ru.wikipedia.org/w/index.php?oldid=22820380> *Основные авторы:* A5b, AGGreSSor, Anonymous1, Atr2006, Batareikin, Beat, Cysneavox, DenisKrivosheev, Denver, Edward Chernenko, Grenadine, Korvin2050, Lankier, Libach, Peni, Roxis, Sgolubev, Веон, Филатов Алексей, 23 анонимных правок

Переносимый набор символов *Источник:* <http://ru.wikipedia.org/w/index.php?oldid=22613298> *Основные авторы:* Incnis Mrsi, Monedula, Stassats, №231-567, 8 анонимных правок

CP437 *Источник:* <http://ru.wikipedia.org/w/index.php?oldid=21473259> *Основные авторы:* Ashik, Dstary, GranD, Maximamax, Monedula, Александр Сигаёв, Ботильда, 7 анонимных правок

Источники, Лицензии и Владельцы(?) изображения.

Файл:OS-structure2.svg *Источник:* <http://ru.wikipedia.org/w/index.php?title=Файл:OS-structure2.svg> *Лицензия:* Public Domain *Основные авторы:* Golftheman

Файл:Kernel-microkernel.svg *Источник:* <http://ru.wikipedia.org/w/index.php?title=Файл:Kernel-microkernel.svg> *Лицензия:* Public Domain *Основные авторы:* Mattia Gentilini

Файл:Kernel-monolithic.svg *Источник:* <http://ru.wikipedia.org/w/index.php?title=Файл:Kernel-monolithic.svg> *Лицензия:* Public Domain *Основные авторы:* Mattia Gentilini

Файл:Kernel-exo.svg *Источник:* <http://ru.wikipedia.org/w/index.php?title=Файл:Kernel-exo.svg> *Лицензия:* Creative Commons Attribution-Sharealike 2.5 *Основные авторы:* User:Surachit

Изображение:Linear_addressing.png *Источник:* http://ru.wikipedia.org/w/index.php?title=Файл:Linear_addressing.png *Лицензия:* Public Domain *Основные авторы:* User:Jego.ruS

Изображение:PDE.png *Источник:* <http://ru.wikipedia.org/w/index.php?title=Файл:PDE.png> *Лицензия:* Public Domain *Основные авторы:* User:Jego.ruS

Изображение:PTE.png *Источник:* <http://ru.wikipedia.org/w/index.php?title=Файл:PTE.png> *Лицензия:* Public Domain *Основные авторы:* User:Jego.ruS

Файл:Сетевая_иерархия_каталогов.png *Источник:* http://ru.wikipedia.org/w/index.php?title=Файл:Сетевая_иерархия_каталогов.png *Лицензия:* неизвестно *Основные авторы:* Obersachse, Panther, Shkutkov Michael, Ботильда

Файл:Структурная_схема_файловой_системы_ext2.png *Источник:* http://ru.wikipedia.org/w/index.php?title=Файл:Структурная_схема_файловой_системы_ext2.png *Лицензия:* Public Domain *Основные авторы:* Obersachse, Panther, Shkutkov Michael

Файл:Система_адресации_файловой_системы_ext2.png *Источник:* http://ru.wikipedia.org/w/index.php?title=Файл:Система_адресации_файловой_системы_ext2.png *Лицензия:* неизвестно *Основные авторы:* Alex Spade, Obersachse, Panther, Shkutkov Michael

Лицензия

Creative Commons Attribution-Share Alike 3.0 Unported
<http://creativecommons.org/licenses/by-sa/3.0/>
