Лекции. Рязанова Н.Ю.

Архитектура ОС с монолитными ядрами и микроядрами

Ядро: монолитное и микроядро

Исторически все операционные системы, начиная от самых ранних (или считая даже от самых рудиментарных исполняющих систем, которые с большой натяжкой вообще можно назвать операционной системой) делятся на самом верхнем уровне на два класса, различающихся в принципе:

- монолитное ядро (исторически более ранний класс), называемые ещё: моноядро, макроядро; к этому классу, из числа самых известных, относятся, например (хронологически): OS/360, RSX-11M+, VAX-VMS, MS-DOS, Windows (все модификации), OS/2, Linux, все клоны BSD (FreeBSD, NetBSD, OpenBSD), Solaris — почти все широко звучащие имена операционных систем.
- микроядро (архитектура, появившаяся позже), которое известно также как клиент-серверные операционные системы и системы с обменом сообщениями; к этому классу относятся, например: QNX, MINIX 3, HURD, ядро Darvin MacOS, семейство ядер L4.

В архитектуре с монолитным ядром все услуги для прикладного приложения выполняют отдельные части кода ядра (в адресном пространстве ядра) (рис.1). До некоторого времени в системах с монолитным ядром (так было и в ранних версиях ядра Linux) любое расширение функциональности достигалось пересборкой (перекомпиляцией) ядра. Для системы промышленного уровня это недопустимо. Поэтому, рано или поздно, любая монолитная операционная система начинает включать в себя ту или иную технологию динамической реконфигурации (что сразу же создает проблемы для её безопасности и устойчивости). Для Linux это — технология загружаемых модулей ядра (появившаяся с ядер версий 2.0.x). В Windows это многоуровневые драйверы и службы.

Можно сказать, что монолитное ядро - это один процесс, работающий в одном адресном пространстве в привилегированном режиме. Представляет статический двоичный файл. Все службы ядра существуют и выполняются в адресном пространстве ядра.

Коротко задачи, выполняемые монолитным ядром ОС могут быть предвтавлнены следующим перечислением:

- Модули ядра ОС выполняют следующие базовые функции ОС:
  - управление процессами
  - управление памятью
  - управление устройствами ввода-вывода
- Ядро обеспечивает решение задачи **организации вычислительного процесса**: переключение контекстов, загрузка/выгрузка страниц, обработка прерываний и т.п.
- Другая задача – поддержка приложений, создание для них **прикладной программной среды**. Приложения обращаются к ядру с запросами (**системными вызовами**) для выполнения базовых операций (открытие и чтение файла, вывод информации на дисплей и т.п.)
- Функции выполняемые ядром ОС требуют высокой скорости выполнения и для этого размещаются постоянно в оперативной памяти (**резидентные модули**).

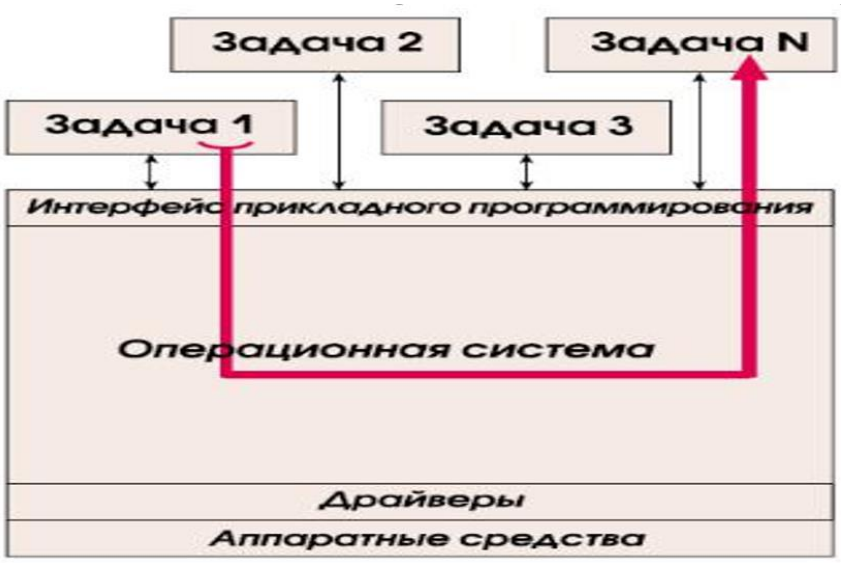


Рис.1

В микроядрах ядро разбивается на отдельные процессы, известные как серверы. Некоторые из серверов работают в пространстве ядра, а некоторые - в пространстве пользователя. Все серверы хранятся отдельно и работают в разных адресных пространствах. В микро-ядерной архитектуре все услуги для прикладных процессов (приложений) система (микроядро) обеспечивает, передавая сообщения-запросы соответствующим сервисам (драйверам, серверам, ...), которые выполняются в пространстве пользователя (в пользовательском кольце защиты) (рис.2). Серверы вызывают могут запрашивать обслуживание у других серверов, отправляя сообщения через IPC (Interprocess Communication). В этом случае не возникает никаких проблем с динамической реконфигурацией системы путем добавления новых функциональностей (например, драйверов проприетарных устройств).

Примечание: *Это же свойство обеспечивает и экстремально высокие живучесть и устойчивость микроядерных систем по сравнению с монолитно-ядерными: вышедший из строя драйвер можно перезагрузить не останавливая систему.*

Коротко в микро-ядерной архитектуре можно выделить следующие особенности:

- большинство составляющих ОС программ являются самостоятельными программами
- взаимодействие между ними обеспечивает специальный модуль ядра, называемый микроядром
- микроядро работает в привилегированном режиме и обеспечивает:
  - ✓ взаимодействие между программами,
  - ✓ планирование использования процессора,
  - ✓ первичную обработку прерываний,
  - ✓ операции ввода-вывода
  - ✓ базовое управление памятью
- остальные компоненты ОС взаимодействуют друг с другом путем передачи сообщений через микроядро

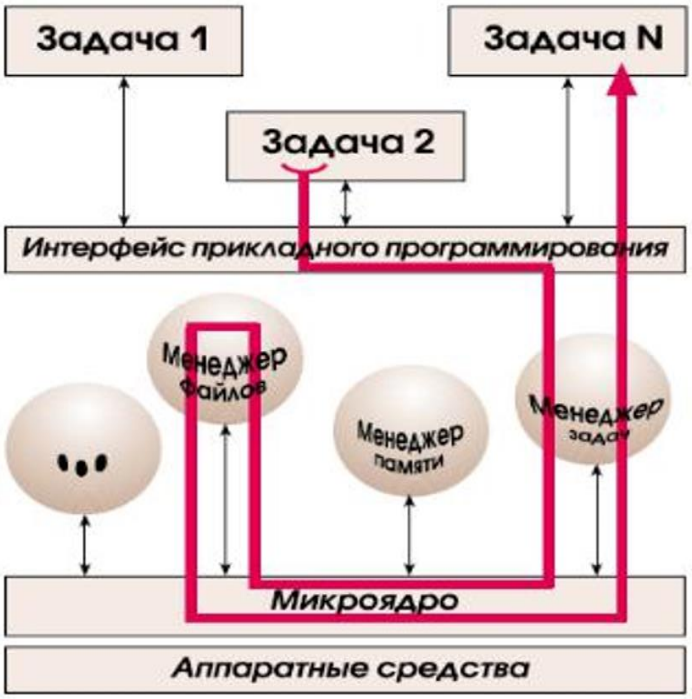
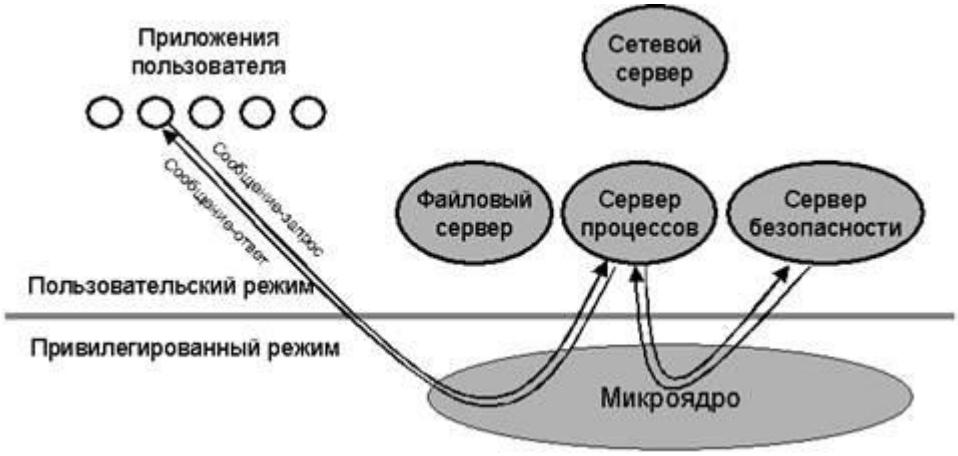


Рис.2

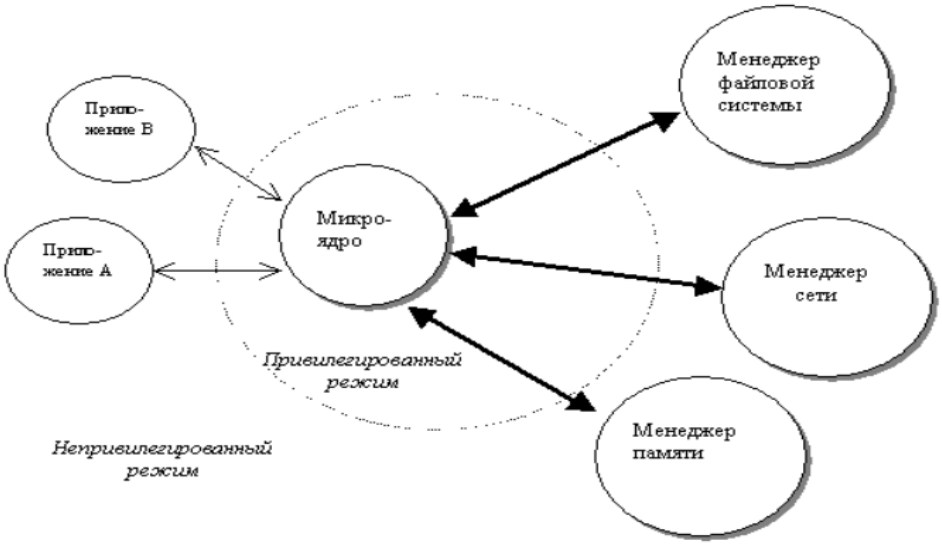
Микроядро

Микроядро (microkernel) – это модуль ядра ОС, обеспечивающий взаимодействие между процессами, планирование, первичную обработку прерываний и базовое управление памятью. Можно сказать, что при разработке микро-ядерных ОС разработчики сосредотачивают в микро-ядре низкоуровневые функции, связанные с непосредственным взаимодействием в аппаратурой. Кроме того, микроядро должно обеспечивать взаимодействие процессов. Это делается с помощью сообщений.

Микроядерная архитектура (Microkernel architecture) – это такая схема ядра ОС, при которой все её компоненты кроме микроядра являются самостоятельными процессами, выполняющимися возможно в разных адресных пространствах и взаимодействуют друг с другом путем передачи сообщений (рис.3).



а)



б)

Рис.3 а) и б)

Разделение функций

Основным принципом организации микро ядерных ОС является включение в состав микроядра только тех функций, которым абсолютно необходимо выполнять в режиме супервизора и в защищенной памяти. Обычно в микроядро включаются машинно-зависимые программы (включая поддержку мультипроцессорной работы), некоторые функции управления процессами, обработка прерываний и поддержка пересылки сообщений. Во многих случаях в микроядро включается функция планирования процессов, но в реализации Mach компании IBM планировщик процессов размещен вне микроядра, а микроядро используется только для непосредственного управления процессами. Конечно, при этом требуется тесное взаимодействие внешнего планировщика и входящего в состав ядра диспетчера. В некоторых реализациях (например, в реализации OSF) в микроядро помещаются драйверы устройств. В реализациях IBM и Chorus драйверы размещаются вне микроядра, но для регулирования режимов разрешения и запрещения прерываний часть программы драйвера выполняется в пространстве ядра. В NT драйверы устройств и другие функции ввода-вывода выполняются в пространстве ядра, но реально используют ядро только для перехвата и передачи прерываний. Следует заметить, что оба подхода допускают динамическое подключение драйверов к системе и их отключение. Однако имеются другие доводы в пользу выделения драйверов из состава микроядра. Например, поскольку во многих случаях драйверы могут не зависеть от особенностей аппаратуры, такой подход облегчает переносимость системы.

QNX и CTOS - это две полностью законченные микро ядерные операционные системы, поставляемые на протяжении нескольких лет. 8-килобайтное микроядро QNX поддерживает только планирование и диспетчеризацию процессов, взаимодействие процессов, обработку прерываний и сетевые службы нижнего уровня. Это микроядро обеспечивает всего лишь 14 системных вызовов. Микроядро QNX может быть целиком размещено во внутреннем кэше некоторых процессоров, таких как Intel 486. Чтобы построить минимальную систему QNX, требуется добавить к микроядру менеджер процессов, который создает и управляет процессами и памятью процессов. Чтобы ОС QNX была применима не только во встроенных и бездисковых системах, нужно добавить файловую систему и менеджер устройств. Эти менеджеры исполняются вне пространства

ядра, так что ядро остается небольшим. Специалисты компании QNX Software заявляют, что система, основанная на передаче сообщений, может иметь производительность, по меньшей мере сравнимую с производительностью других традиционных операционных систем. CTOS, появившаяся в 1980 году, была написана для рабочих станций фирмы Convergent Technologies - семейства машин на основе процессоров Intel для работы в "кластерных сетях", соединенных по обычным телефонным проводам. Продаваемые в настоящее время фирмой Unisys, эти основанные на CTOS машины продемонстрировали преимущества распределенных вычислений на основе передачи сообщений задолго до того, как этот термин стал модным. Крохотное 4-килобайтное микроядро CTOS взяло на себя только планирование и диспетчеризацию процессов и взаимодействие процессов на основе сообщений. Все другие системные службы взаимодействуют с ядром и друг с другом через четко определенные интерфейсы сообщений. Сетевые средства входят в состав CTOS и являются действительно прозрачными для прикладных программ, которым не требуется знать, будет ли обработан запрос на обслуживание локально или удаленно. В любом случае сообщения передает одна и та же система взаимодействия процессов.

Модель клиент-сервер в микро-ядерной архитектуре

Микро-ядерная архитектура основана на модели клиент-сервер (client-server model). Когда пользовательской программе-клиенту требуется вызвать какую-либо функцию ОС (например, вызов функции read для чтения данных из файла), она посылает сообщение (message) серверу – процессу, реализующую эту функцию. Один и тот же процесс в зависимости от ситуации может быть и/или клиентом и/или сервером. Например, файловая система является сервером для процесса, открывшего файл, и клиентом – для драйвера диска, на котором этот файл располагается.

Основой работы микро-ядерной ОС является обеспечение посылки и приема сообщений, которыми обмениваются процессы и серверы ОС. Такой обмен сообщениями должен быть надежным. Это значит, что процесс, который выполнил запрос на обслуживание, должен быть уверен, что его запрос будет обработан или сразу получить ошибку, если возможны проблемы с обслуживанием. Такое взаимодействие обеспечивается принятым в системе протоколом, в котором должно быть предусмотрено подтверждение приема сообщения.

Рассмотрим диаграмму последовательности событий при обмене сообщениями (рис.4).

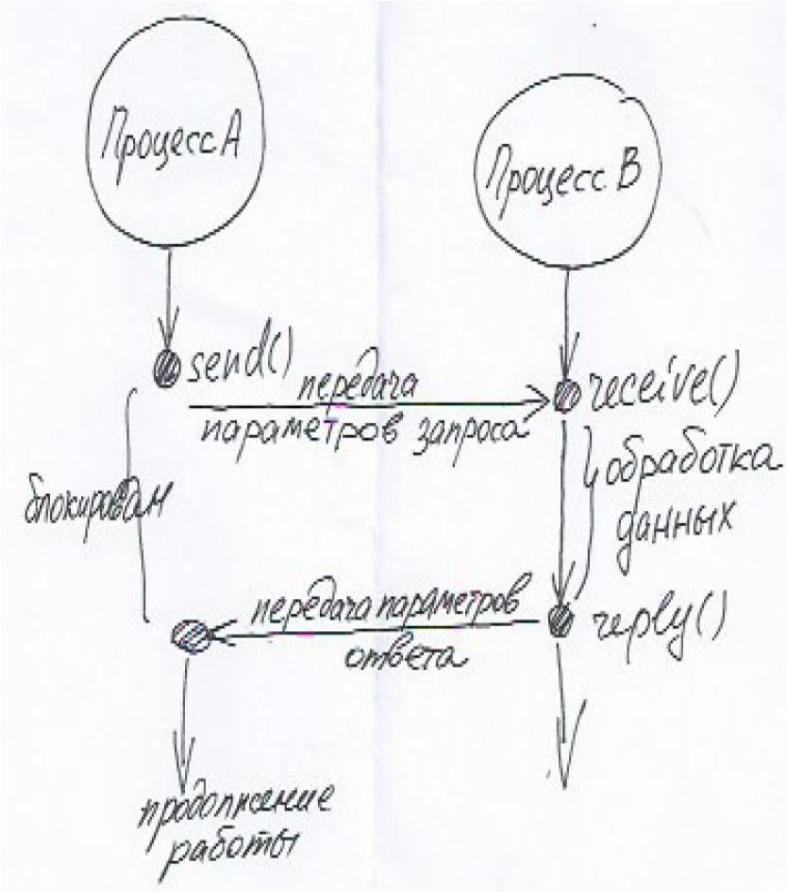


Рис. 4

При таком обмене процессы синхронизируются: процесс, отправивший сообщение-запрос, блокируется в ожидании сообщения-ответа от другого процесса.

При передаче сообщений возможны три состояния блокировки процесса (рис.5).

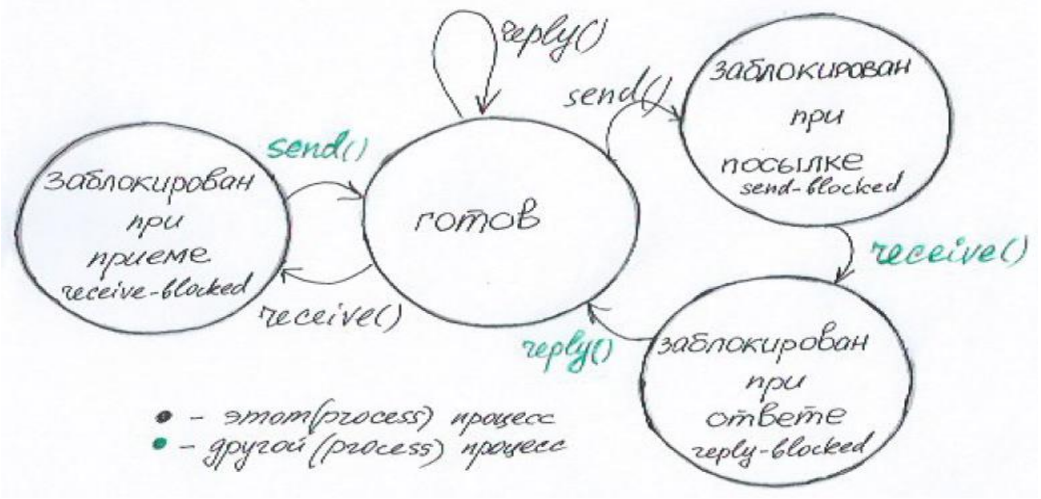


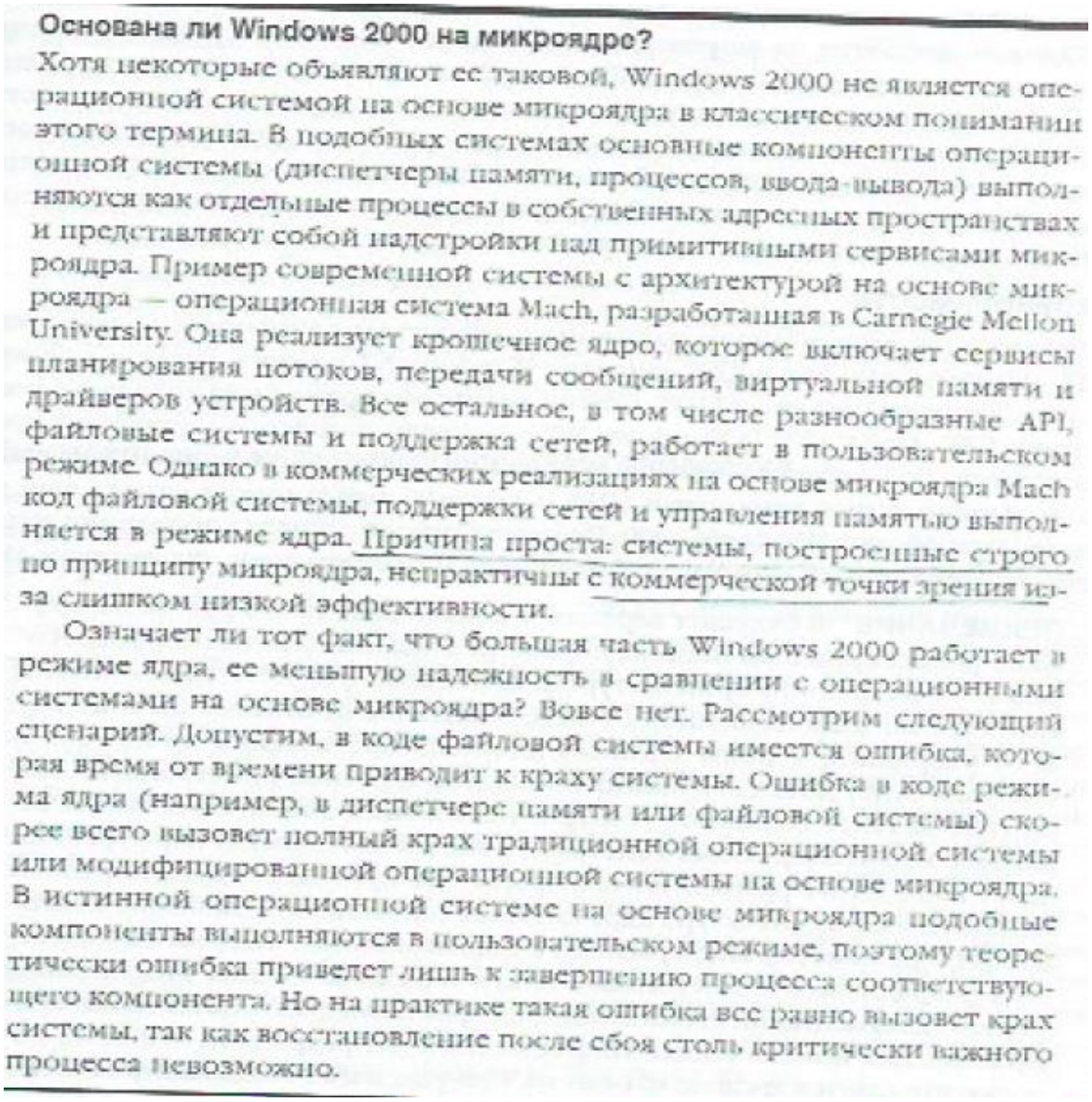
Рис. 5

Процесс блокируется при посылке, если он вызвал send(), а процесс, которому сообщение адресовано, не готов к его приему. Как только адресат вызовет receive(), процесс, пославший сообщение будет, процесс, пославший сообщение будет заблокирован при ответе и будет в этом состоянии до тех пор пока запрос не будет обработан и на стороне другого процессе не будет вызван reply(). Если процесс вызвал

receive(), а сообщение ему еще не послано, то процесс окажется заблокированным при приеме. Аналогичная ситуация будет при вызове reply().

Основные концепции ОС с микро-ядром Mach

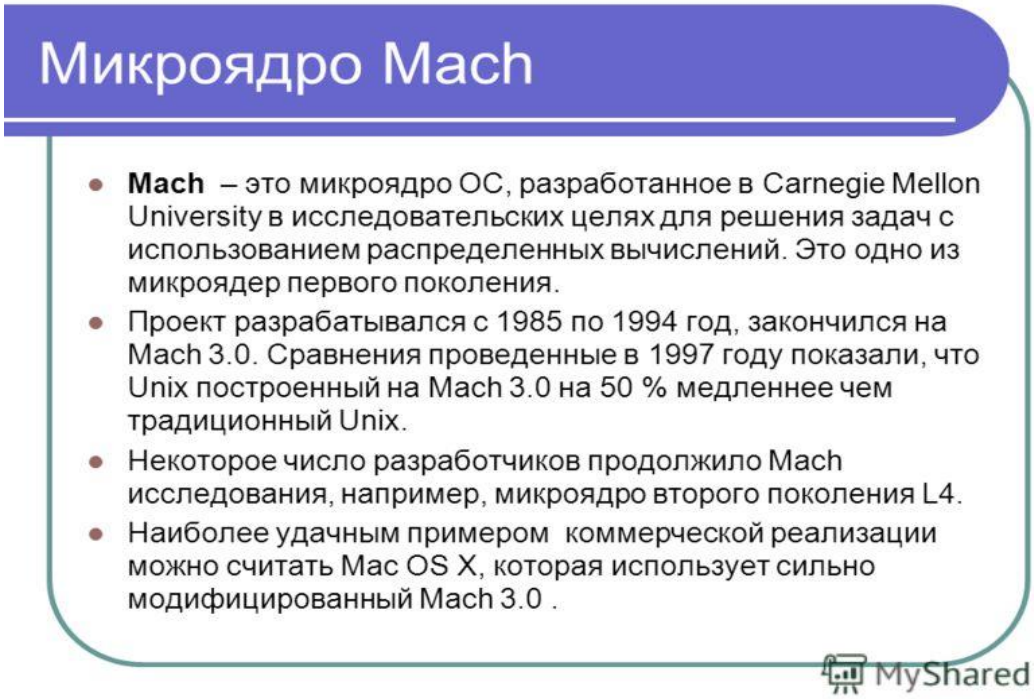
Рассмотрим, что известные авторы Microsoft Соломон и русинович пишут о Windows 2000 с точки зрения микро-ядерной архитектуры. В приведенной вставке перечислены очень важные составляющие микроядерной архитектуры. Данный текст взят из книги Соломона Д., Русиновича М. Внутреннее устройство Microsoft Windows 2000. Мастер класс. 2001 г.



Основана ли Windows 2000 на микроядре?

Хотя некоторые объявляют ее таковой, Windows 2000 не является операционной системой на основе микроядра в классическом понимании этого термина. В подобных системах основные компоненты операционной системы (диспетчеры памяти, процессов, ввода-вывода) выполняются как отдельные процессы в собственных адресных пространствах и представляют собой надстройки над примитивными сервисами микроядра. Пример современной системы с архитектурой на основе микроядра — операционная система Mach, разработанная в Carnegie Mellon University. Она реализует крошечное ядро, которое включает сервисы планирования потоков, передачи сообщений, виртуальной памяти и драйверов устройств. Все остальное, в том числе разнообразные API, файловые системы и поддержка сетей, работает в пользовательском режиме. Однако в коммерческих реализациях на основе микроядра Mach код файловой системы, поддержки сетей и управления памятью выполняется в режиме ядра. Причина проста: системы, построенные строго по принципу микроядра, непрактичны с коммерческой точки зрения из-за слишком низкой эффективности.

Означает ли тот факт, что большая часть Windows 2000 работает в режиме ядра, ее меньшую надежность в сравнении с операционными системами на основе микроядра? Вовсе нет. Рассмотрим следующий сценарий. Допустим, в коде файловой системы имеется ошибка, которая время от времени приводит к краху системы. Ошибка в коде режима ядра (например, в диспетчере памяти или файловой системы) скорее всего вызовет полный крах традиционной операционной системы или модифицированной операционной системы на основе микроядра. В истинной операционной системе на основе микроядра подобные компоненты выполняются в пользовательском режиме, поэтому теоретически ошибка приведет лишь к завершению процесса соответствующего компонента. Но на практике такая ошибка все равно вызовет крах системы, так как восстановление после сбоя столь критически важного процесса невозможно.



Любая ОС оперирует определенными абстракциями. Известная ОС Mach не является

Ядро ОС Mach оперирует пятью основными абстракциями:

• Task

– Базовая единица распределения ресурсов. Классическое представление о процессе как о владельце ресурсов

– Виртуальное адресное пространство Virtual address space, communication capabilities

• Thread

– Базовая единица диспетчеризации

• Port

– Коммуникационный канал для IPC (Inter Process Comnucations – Межпроцессное взаимодействие). Безопасные симплексные каналы связи, доступные только через функции отправки и получения (известные как права порта).

• Message

– Перемещается между программами через порт. Может содержать данные, определяемые возможностями порта, указатели.

• Memory Object

- Внутренние блоки управления памятью. Объекты памяти включают именованные записи и регионы; они представляют собой потенциально персистентные данные, которые могут отображаться в адресные пространства.

Поток или нить (Thread) является единицей выполнения. Она имеет счетчик команд и набор регистров. Каждая нить является частью точно одного процесса. Процесс, состоящий из одной нити, подобен традиционному процессу, например, процессу Unix.

Отличительной особенностью ОС Mach является понятие «объект памяти» (memory object), который представляет структуру данных, отображаемую в адресное пространство процесса. Объекты памяти занимают одну или несколько страниц и образуют основу для системы управления виртуальной памятью. Когда процесс ссылается на объект памяти, который не находится в физической памяти, это вызывает страничное прерывание. Как и в других ОС, ядро перехватывает страничное прерывание. Однако, в отличие от других ОС, ядро Mach для загрузки отсутствующей страницы посылает сообщение серверу пользовательского режима, а не самостоятельно выполняет эту операцию.

Межпроцессное взаимодействие в Mach основано на передаче сообщений.

Для того, чтобы получить сообщение, пользовательский процесс запрашивает ядро, чтобы оно создало защищенный почтовый ящик, который называется порт.

Порт создается в адресном пространстве ядра и способен поддерживать очередь – упорядоченный список сообщений. Очереди не имеют фиксированной длины, но в целях управления потоком для каждого порта устанавливается пороговое значение в n сообщений. Любой процесс, пытающийся послать еще одно сообщение в очередь длины n приостанавливается для того, чтобы дать порту возможность очистки.

Процесс может предоставить другому процессу возможность посылать или получать сообщения в один из принадлежащих ему портов. Такая возможность реализуется в виде мандата (capability), который включает не только указатель на порт, но и список прав, которыми обладает другой процесс по отношению к ланному порту, например, право выполнить операцию send().

Все коммуникации Mach используют этот механизм.

Проблемы Mach с производительностью

Проблемы с производительностью Mach очевидно связаны с необходимостью использование IPC, что для большинства задач снижает производительность[3]. Сравнения, проведенные в 1997 году, показали, что Unix, построенный на Mach 3.0, на 50 % медленнее, чем традиционный Unix[4].

Исследования показали, что производительность падает из-за IPC, и достичь ускорения за счет раздробления ОС на маленькие сервера невозможно. Было сделано множество попыток улучшить производительность Mach, но в середине 1990 интерес пропал.

Фактически, исследование природы проблем производительности выявило несколько интересных фактов: один — IPC сам по себе не является проблемой, проблема в необходимости отображения памяти для его поддержки, что добавляет небольшие временные затраты. Большинство времени (около 80 %) тратится на дополнительные задачи в ядре — на обработку сообщения, в первую очередь - проверку прав порта и целостность сообщения. В тестах на Intel 80486DX-50 стандартный Unix-вызов занимает около 21 микросекунды, соответствующий вызов в Mach занимает 114 микросекунд, из них 18 микросекунд относятся к аппаратному обеспечению, остальное относится к различным функциям ядра Mach.

Когда Mach впервые была использована в серьёзных разработках (2я версия), производительность была ниже, чем в традиционных ядрах, примерно на 25 %. Эта цена не вызывала беспокойства, потому что система хорошо портировалась и работала на множестве процессоров. Фактически, система скрыла серьёзные проблемы с производительностью до выхода Mach 3, когда множество разработчиков попыталось создать системы, запускаемые в непривилегированном режиме.

Когда Mach 3 попытался переместить ОС в непривилегированный режим, потеря производительности стала заметной. Рассмотрим простой пример: задача узнаёт текущее время. Посылается сообщение ядру Mach, это вызывает переключение контекста, отображение памяти, затем ядро проверяет сообщение и права, если все хорошо, то вызывается переключение контекста на сервер, затем сервер выполняет действия и посылает сообщение назад, ядро выделяет ещё памяти и переключает контекст дважды.

Но здесь есть проблема. Она заключается в системе подкачки виртуальной памяти. Традиционные монолитные ядра знают, где ядро и его модули, а где память, которую можно выгружать, в то время как Mach не имеет ни малейшего представления о том, из чего состоит система. Вместо этого он использует единое решение, добавляющее проблем с производительностью. Mach 3 дает простой менеджер памяти, который обращается к другим менеджерам, выполняющимся в непривилегированном режиме, что заставляет систему делать дорогие IPC-вызовы.

Многие из этих проблем существуют в любой системе, которой необходимо работать на многопроцессорной машине, и в середине 1980-х разработчикам ОС казалось, что будущий рынок будет наполнен ими. Фактически эволюция не работает, как ожидается. Мультипроцессорные машины использовались в серверных приложениях в начале 1990-х, но затем исчезли. Тем временем производительность CPU возрастала на 60 % в год, умножая неэффективность кода. Плохо, что скорость доступа к памяти растет только на 7 % в год, это значит, что цена обращения к памяти не уменьшилась, и вызовы IPC Mach'а, которые не сохраняются в кэше, работают очень медленно.

Несмотря на возможности Mach, такие потери производительности в реальном мире неприемлемы.

Дополнительно

Иерархическая структура ОС Медника и Донована

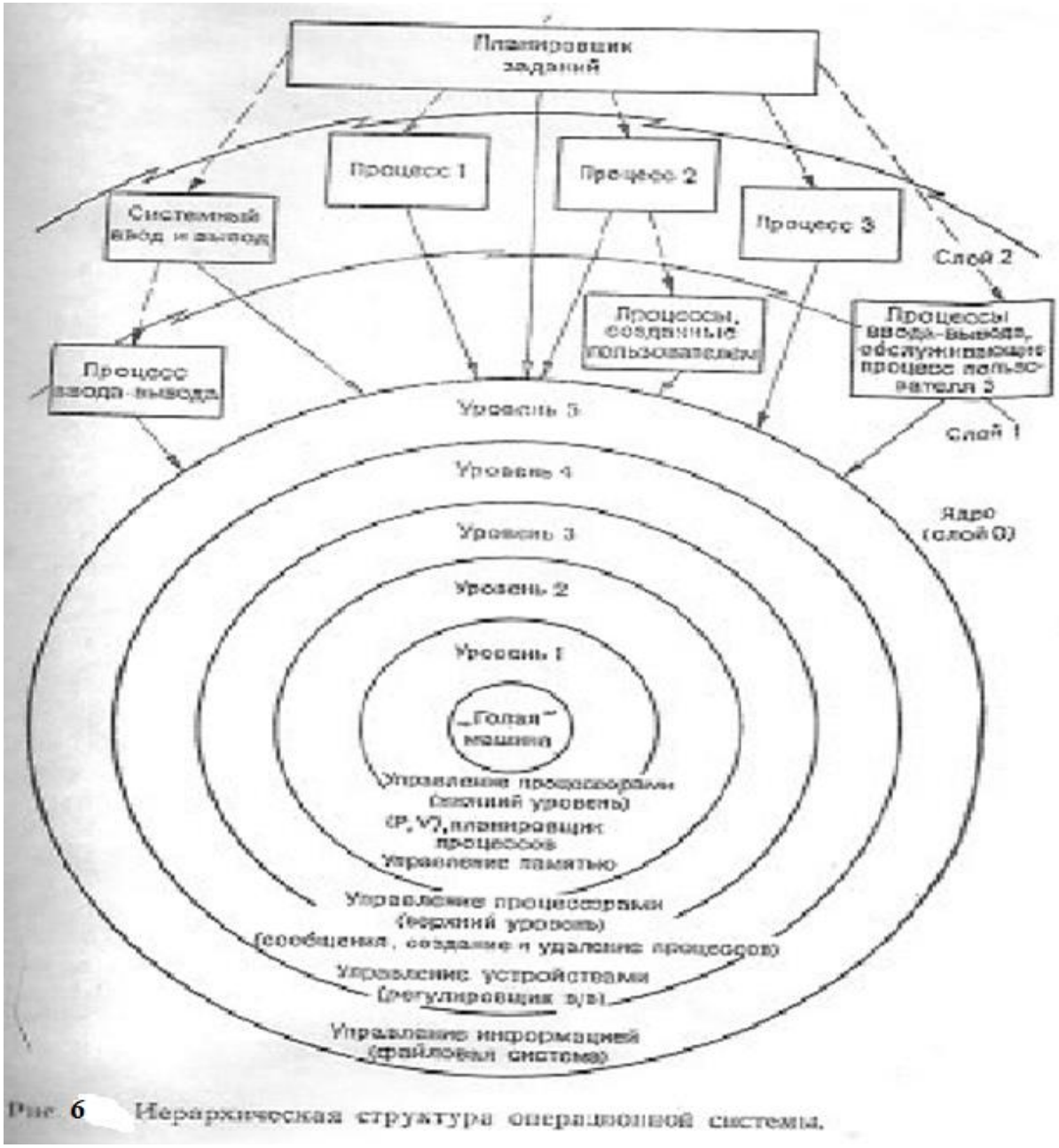


Рис 6 Иерархическая структура операционной системы.

Виртуальные машины

Исходная версия OS/360 была системой исключительно пакетной обработки. Однако множество пользователей OS/360 желали работать в системе с разделением времени, поэтому различные группы программистов как в самой корпорации IBM, так и вне ее решили написать для этой машины системы с разделением времени. Официальная система с разделением времени от IBM, которая называлась TSS/360, поздно вышла в свет и оказалась настолько громоздкой и медленной, что на нее перешли немногие. В конечном счете от нее отказались, но уже после того, как ее разработка потребовала около 50 млн. долларов. Группа из научного центра IBM в Кембридже, штат Массачусетс, разработала в корне отличающуюся от нее систему, которую IBM в результате приняла как законченный продукт. Сейчас она широко используется на еще оставшихся мэйнфреймах. Эта система, в оригинале называвшаяся CP/CMS, а позже переименованная в VM/370, была основана на следующем проницательном наблюдении: система с разделением времени обеспечивает (1) многозадачность и (2) расширенную машину с более удобным интерфейсом, чем тот, что предоставляется оборудованием напрямую. VM/370 основана на полном разделении этих двух функций.

Сердце системы, называемое монитором виртуальной машины, работает с оборудованием и обеспечивает многозадачность, предоставляя верхнему слою не одну, а несколько виртуальных машин, как показано на рис. 7.

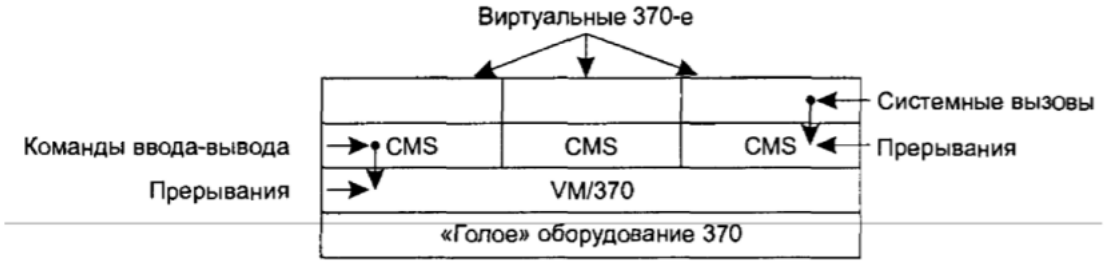


Рис. 7 . Структура VM/370 с системой CMS

Но, в отличие от всех других ОС, эти виртуальные машины не являются расширенными. Они не поддерживают файлы и прочие удобства, а представляют собой точные копии голой аппаратуры, включая режимы ядра и пользователя, ввод-вывод данных, прерывания и все остальное, присутствующее на реальном компьютере. Поскольку каждая виртуальная машина идентична настоящему оборудованию, на каждой из них может работать любая ОС, которая запускается прямо на аппаратуре. На разных виртуальных машинах могут (а зачастую так и происходит) функционировать различные ОС. На некоторых из них для обработки пакетов и транзакций работают потомки OS/360, а на других структура VM/370 с системой CMS для интерактивного разделения времени пользователей работает однопользовательская интерактивная система CMS (Conversational Monitor System — система диалоговой обработки). Рис. 7. Структура VM/370 с системой CMS Когда программа ОС CMS выполняет системный вызов, он прерывает ОС на своей собственной виртуальной машине, а не на VM/370, как произошло бы, если бы он работал на реальной машине, вместо виртуальной. Затем CMS выдает обычные команды ввода-вывода для чтения своего виртуального диска или другие команды, которые ей могут понадобиться для выполнения Вызова. Эти Команды ввода-вывода перехватываются VM/370, которая выполняет их в рамках моделирования реального оборудования. При полном разделении функций многозадачности и предоставления расширенной машины каждая часть может быть намного проще, гибче и удобней для обслуживания. Идея виртуальной машины очень часто используется в наши дни, но в несколько другом контексте: для работы старых программ, написанных для системы MS-DOS на Pentium (или на других 32-разрядных процессорах Intel). При разработке компьютера Pentium и его программного обеспечения обе компании, Intel и Microsoft, понимали, что возникнет острая потребность в работе старых программ на новом оборудовании. Поэтому корпорация Intel создала на процессоре Pentium режим виртуального процессора 8086. В этом режиме машина действует как 8086 (которая с точки зрения программного обеспечения идентична 8088), включая 16-разрядную адресацию памяти с ограничением объема памяти в 1 Мбайт. Такой режим используется системой Windows и другими ОС для запуска программ MSDOS. Программы запускаются в режиме виртуального процессора 8086. Пока они выполняют обычные команды, они работают напрямую с оборудованием. Но когда программа пытается обратиться по прерыванию к ОС, чтобы сделать системный вызов, или пытается напрямую осуществить ввод-вывод данных, происходит прерывание с переключением на монитор виртуальной машины.

Структура ядра ОС UNIX 4.4 BSD


Системные вызовы
Аппаратные и эмулированные прерывания
Управление терминалом
Сокеты
Именование файла
Отображение адресов
Страничные прерывания
Обработка сигналов
Создание и завершение процессов
Необработанный телетайп
Обработанный телетайп
Сетевые протоколы
Файловые системы
Виртуальная память
Дисциплины линии связи
Маршрутизация
Буферный кэш
Страничный кэш
Планирование процесса
Символьные устройства
Драйверы сетевых устройств
Драйверы дисковых устройств
Диспетчеризация процессов
Аппаратура

Рис.8