# 1 Тема 1. Теоретическая часть

## ОС как базовая часть систем обработки данных (СОД)

Прежде чем изучать ОС, необходимо ответить на вопрос: «*Зачем нужны компью- теры?*».

Одним из возможных ответов: «*Компьютеры нужны для обработки данных*». Известный русский ученый, Ларионов А.М., анализируя возможности обработки данных на ЭВМ, дает определение и классификацию различных систем обработки данных.

***Система обработки данных (СОД)*** — совокупность технических средств и программного обеспечения, предназначенная для информационного обслуживания пользователей и технических объектов.

Общая структура классификации СОД, данная Ларионовым А.М., приведена на рисунке 1.1, которую с успехом можно использовать в качестве ориентира при изу- чении различных дисциплин, связанных с применением вычислительной техники.

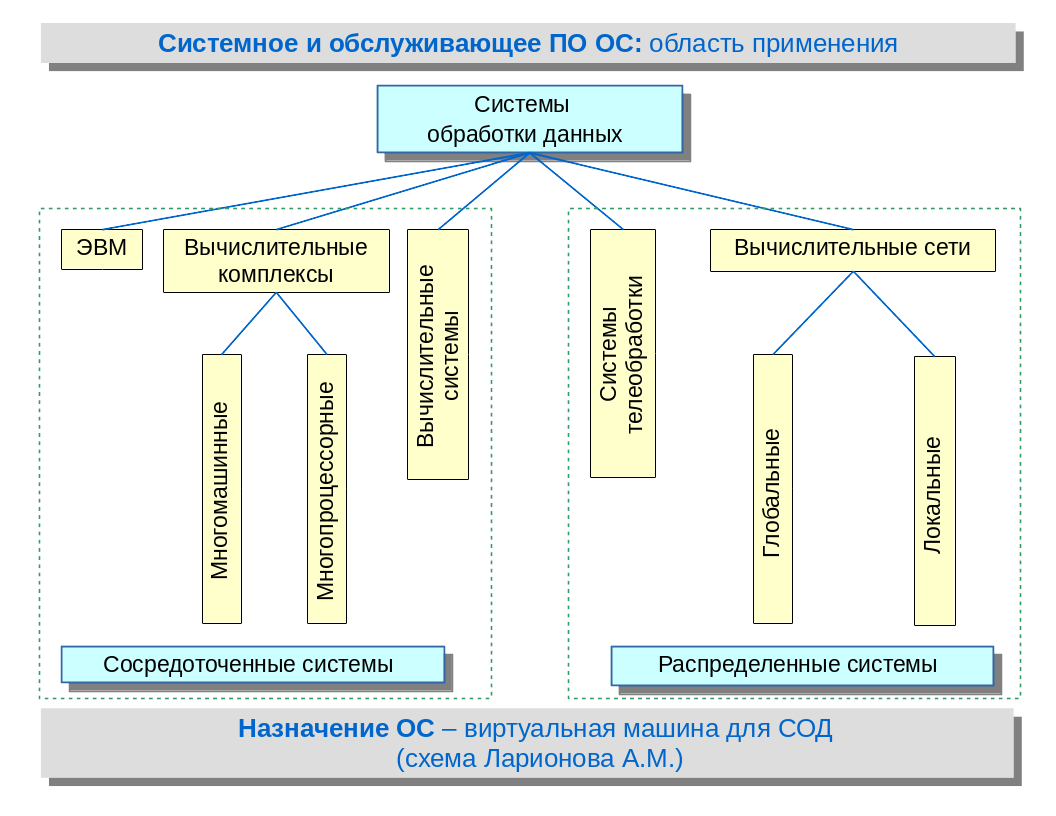


Рисунок 1.1 — Структура классификации СОД (Ларионов А.М.)

Хорошо видно, что вся классификация разделена на два больших класса:

* *сосредоточенные (централизованные) системы*, в которых обработка данных ведется отдельной ЭВМ, вычислительным комплексом или вычислительной системой;
* *распределенные системы*, в которых процессы обработки данных рассредо- точены по многим компонентам: системам телеобработки или вычислитель- ным сетям.

Нетрудно догадаться, что каждый элемент такой классификации имеет свою ОС, хотя их функциональные возможности могут сильно отличаться.

С другой стороны, мы понимаем, что *ОС — это программное обеспечение (ПО)*, которое устанавливается на аппаратную часть вычислительной техники. С этой точки зрения, рассматривая отдельную ЭВМ, мы будем различать:

 *техническую часть* — аппаратное обеспечение компьютера, упрощенная ар- хитектура которого представлена на рисунке 1.2;

 *программную часть* — программное обеспечение компьютера, общая класси- фикация которого показана на рисунке 1.3.

Архитектура современного общедоступного компьютера представляет набор функциональных компонент, во многом работающих независимо друг от друга, но сог- ласованно взаимодействующих через общую системную манистраль.

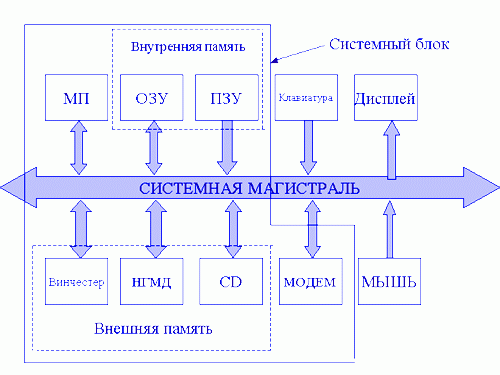


Рисунок 1.2 - Архитектура аппаратной части ЭВМ

Основу архитектуры аппаратной части ЭВМ составляет *системный блок*, в котором размещены:

1. микропроцессор (**МП**);
2. блок оперативного запоминающего устройства (**ОЗУ**);
3. микросхемы постоянного запоминающего устройства (**ПЗУ**);
4. устройства долговременной памяти на «жестком диске» (**Винчестер**);
5. устройства для запуска компакт-дисков (**CD**) и дискет (**НГМД**).

В системном блоке находятся также *интерфейсные платы*: сетевая, видеопамяти, обработки звука, модем (модулятор-демодулятор), платы, обслуживающие устрой- ства ввода-вывода: клавиатуры, дисплея, "мыши", принтера и другие устройства.

Программная часть ЭВМ (рис. 1.3) условно разделяется на три категории:

* *системное ПО* - ОС и программы общего пользования, выполняющие раз- личные вспомогательные функции, например, создание копий используемой

информации, выдачу справочной информации о компьютере, проверку работоспособности устройств компьютера и другие;

* *прикладное ПО* - программы, обеспечивающие выполнение необходимых работ на ЭВМ: редактирование текстовых документов, создание рисунков

или картинок, обработка информационных массивов и другие;

* *инструментальное ПО* - программы, обеспечивающие разработку новых программ для компьютера на различных языках программирования.

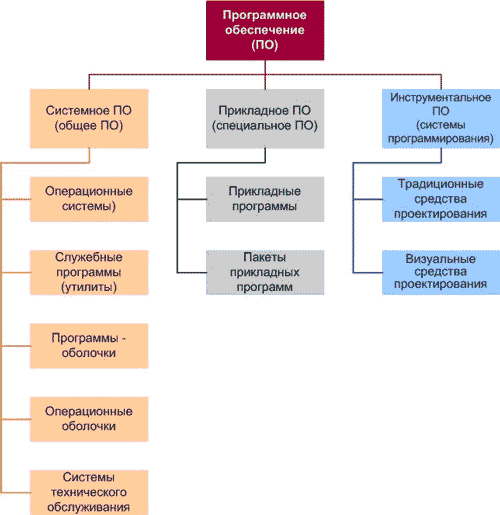


Рисунок 1.3 - Классификация ПО ЭВМ

Хорошо видно, предметом нашей дисциплины является *категория систем- ного ПО*.

***Системное ПО*** - это программы общего пользования, не связаные с конкретным применением ЭВМ и выполняющие традиционные функции: планирование и уп- равление задачами, управление вводом-выводом и другие. К нему относятся:

1. *операционные системы* — программа, которая загружается в ОЗУ ЭВМ при включении компьютера;
2. *программы–оболочки* - обеспечивают более удобный и наглядный способ общения с компьютером, по сравнению с командной строкой DOS, например, Far, Total Commander;
3. *операционные оболочки* – интерфейсные системы, которые используются для создания графических интерфейсов, мультипрограммирования и другие;
4. *драйверы* - программы, предназначенные для управления портами перифе- рийных устройств; обычно загружаются в оперативную память при запуске компьютера;
5. *утилиты* - вспомогательные или служебные программы, которые представля- ют пользователю ряд дополнительных услуг.

**Замечание** Часто бывает сложно разделить ПО, относящееся к утилитам, и прикладное ПО, пос- кольку часть утилит входит в состав ОС, а другая часть поставляется и функционирует автономно. Общепринято считать, что к утилитам относятся:

* *диспетчеры файлов* или *файловые менеджеры*;
* *средства динамического сжатия данных*, которые позволяют уменьшить размеры файлов и увеличить количество информации на диске за счет ее динамического

сжатия;

* *средства просмотра* и *воспроизведения*;
* *средства диагностики* и *контроля*, которые позволяют проверить конфигурацию компьютера и работоспособность его устройств, прежде всего «жестких дисков»;
* *средства коммуникаций* (коммуникационные программы), предназначенные для организации обмена информацией между компьютерами;
* *средства обеспечения компьютерной безопасности*: резервное копирование, ан- тивирусное ПО и другие.

## Серверные ОС и рабочие станции

В контексте аппаратного обеспечения ЭВМ:

* *сервер* — существительное от глагола ***to serve*** — служить; специализирован- ный компьютер или оборудование для выполнения на нем сервисного ПО;

ЭВМ с повышенной надежностью исполнения, имеющее сетевые устройства и предназначенное для непрерывной работы в течении длительного без вык- лючения или перезагрузки ОС;

* *рабочая станция* — ***workstation*** — ЭВМ, оборудование которой расширено устройствами мультимедиа и другими системами, предназначенная для реше-

ния определенного круга задач; наличие оборудования для работы в сети яв- ляется необязательным требованием, но требования к возможности интерак- тивного взаимодействия с пользователем являются определяющими.

Фактически, сервер и рабочая станция могут иметь одинаковую аппаратную конфигурацию, что во многом ограничивает возможности данного контекста. Наи- более широко термины сервер и рабочая станция используются в контексте прог- раммного обеспечения или в комбинации с котекстом аппаратного обеспечения ЭВМ, подразумевающие использование парадигмы «*клиент-сервер*»:

* *сервер* — любая запущенная программа, ориентированная в прикладном пла- не на обслуживание запросов других программ — *клиентов*;
* *рабочая станция* — ЭВМ, предназначенная для интерактивной работы с пользователем, на которой установлено *клиентское программное обеспече- ние*.

На ранней стадии развития, многие ОС не поддерживали работу в сети. Нап- ример, MS DOS и MS Windows, первоначально создавались как рабочие станции, предполагающие автономную работу ЭВМ. Со временем, такие ОС стали исполь- зовать сетевое ПО сторонних разработчиков, а те ОС, которые имели собственное сетевое ПО, стали называться ***сетевыми ОС***. В настоящее время, практически все ОС способны работать в сети, поэтому необходимость в дополнительной класси- фикации отпала сама собой. Тем не менее, градация ЭВМ осталась, но перешла в область системного ПО и дистрибутивов ОС:

* *сервер (****server****)* — дистрибутив ОС или ЭВМ, с установленным системным и прикладным ПО, ориентированные на выполнение функций *сервера*;
* *рабочая станция (****desktop****)* — дистрибутив ОС или ЭВМ, предназначенная для интерактивной работы с пользователем, на которой установлено соот- ветствующее *клиентское прикладное программное обеспечение*.

**Замечание** Применительно к ограничениям уровня изучения нашей дисциплины, различия между серверами и рабочими станциями являются не существенными, поскольку определяются ***специализацией системного и прикладного ПО ЭВМ***. ПО ОС УПК АСУ создано на основе дистрибутива типа *desktop*, поэтому для работы в качестве сервера требуется установка дополнительного ПО.

**В целом**, объем нашего курса предполагает изучение ПО ОС, на уровне

***desktop***, *ограниченной отдельной ЭВМ*.

## Многослойная структура ОС

Первоначально, ОС были *монолитными* и не имели *архитектуры*. Например, корпорация IBM ***в 1964 году*** стала разрабатывать первую версию ОС OS/360 и ***за 5 лет***, коллектив из 5000 человек написал ***более 1 млн строк кода***. Постепенно ста- ло ясно, что разработка ОС должна:

* *вестись* на основе модульного программирования;
* *иметь* иерархическую структуру.

**В результате**, были разработаны концептуальные требования к архитектуре ОС.

**Классическая архитектура ОС** основана на:

* концепции иерархической многоуровневой машины;
* привилегированном ядре;
* пользовательском режиме работы транзитных модулей.

**Модули ядра** выполняют базовые функции ОС:

* управление процессами, памятью;
* устройствами ввода-вывода и тому подобное.

**В концепции многоуровневой (***многослойной***) иерархической машины**, структу-

ра ОС представляется рядом слоев, показанных на рисунке 1.4.

**Здесь**, каждый внутренний слой обслуживает вышележащий слой через *межслой- ный интерфейс*.

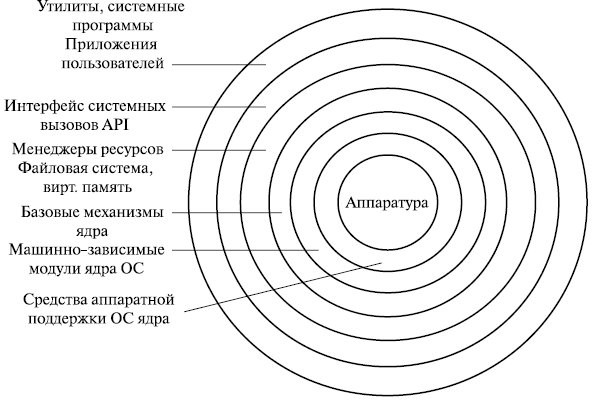


Рисунок 1.4 - Иерархическая архитектура ОС

Такая организация существенно упрощает разработку системы и позволяет:

* сначала, "*сверху вниз*" определить функции слоев и межслойные интерфейсы;
* при детальной реализации, двигаясь "*снизу вверх*", – можно наращивать мощ- ность функций слоев;
* модули каждого слоя можно изменять без необходимости изменений в других слоях, но *не меняя межслойных интерфейсов*!

**В общем случае**, повышение устойчивости работы ОС обеспечивается переходом ядра ОС в *привилегированный режим*.

**Привелигированный режим** — особый режим работы процессора, поддерживае- мый аппаратурой ЭВМ, в котором никакая программа, работающая в других режи- мах, не может прервать работу процессора.

## ОС как базовая часть ПО ЭВМ

Из рисунка 1.4 хорошо видно, что в классической архитектуре ОС приклад- ным программам пользователей отводится верхний, последний слой. Все осталь- ные слои архитектуры составляют *ядро ОС*, которое непосредственно взаимодей- ствует с аппаратным обеспечением ЭВМ.

**Ядро ОС** составляет сердцевину системного ПО, без которого это ПО является полностью неработоспособным и не может выполнить ни одну из своих функций. **В ядре** решаются *внутрисистемные задачи организации вычислительного процес- са*, недоступные для приложений.

**Таким образом**, ядро ОС является *базовым ПО ЭВМ*.

**Особый класс функций ядра** служит для поддержки приложений, создавая для них так называемую *прикладную программную среду*. Все приложения обращаются к ядру со специальными запросами – *системными вызовами*.

**Примеры** системных вызовов:

* *открытие* и чтение файла,
* *получение* системного времени,
* *вывод* информации на дисплей компьютера и другие.

**Функции ядра**, которые могут вызываться приложениями, образуют *интерфейс*

*прикладного программирования* – **API** (*Application Programming Interface*).

Хотя архитектура аппаратной части многих ЭВМ может быть адекватно пред- ставлена рисунком 1.2, конкретные реализации такой архитектуры могут быть раз- личны:

* *различаются* процессора и поддерживаемый набор команд;
* *различаются* шины компьютера и устройства подключения к ним;
* *постоянно идет развитие* и *изменение* конструктивных особенностей всех внешних устройств.

В такой ситуации, *каждое ядро ОС* реализует некоторую абстрактную и более упрощенную архитектуру ЭВМ, доступ которой реализуется через стандарт- ный набор функций, называемый *функциями системных вызовов*.

**В результате**, прикладной программист рассматривает ядро ОС как некото- рую *абстрактную (****виртуальную****) машину*, которая является средой для выпол- нения его программ.

**Систематизируя** различные системные вызовы и развивая идею виртуальной машины, мы с точностью до терминологии можем утверждать, что каждое ядро ОС, абстрагирует три базовых концепции: ***файл***, ***пользователь*** и ***процесс***.

Но прежде чем обосновать это, рассмотрим более подробно взаимодействие прикладных программ пользователя с защищенным ядром ОС, через интерфейс API. Кроме того, следует также учесть, что на основании рисунка 1.4 можно фор- мировать *различные типы ядер ОС*.

## Режимы ядра и пользователя

Чтобы повысить надежность работы ОС, ее ядро работает в специальном привелигированном (***защищенном***) режиме. Соответственно, режим, в котором работают утилиты и остальное прикладное ПО ОС называется *режимом поль- зователя*.

Сначала рассмотрим взаимодействие ПО ЭВМ для ***классической архитек- туры ядра ОС UNIX***, которое показано на рисунке 1.5.

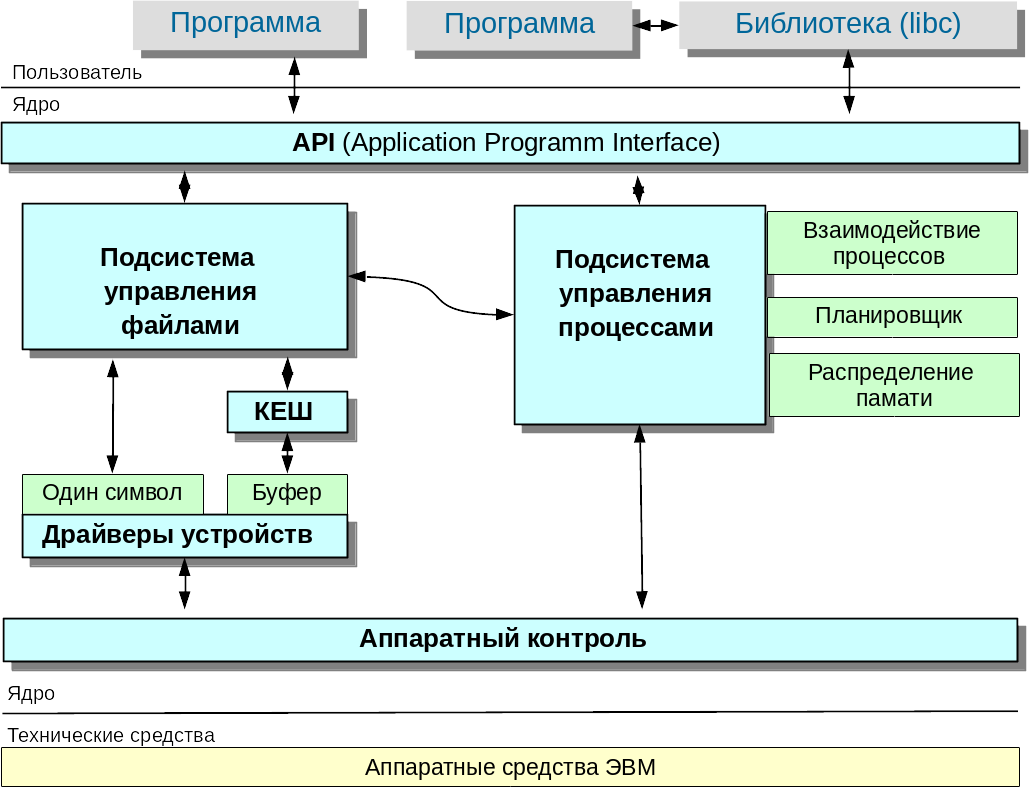


Рисунок 1.5 - Классическая архитектура ПО ОС UNIX

Классическая архитектура ПО ОС UNIX содержит *монолитное ядро ОС*, которое согласно рисунку 1.4 содержит ПО всех уровней, кроме последнего.

**Системный вызовов** такого ядра ОС происходит в два этапа, показаного на рисун- ке 1.6:

* *системный вызов* привилегированного ядра ОС инициирует переключение процессора из пользовательского режима в привилегированный;
* *при возврате к приложению*, происходит обратное переключение.

За счет ***времени переключения 2t*** возникает дополнительная задержка в об- работке системного вызова. Однако такое решение стало классическим и исполь- зуется во многих ОС: ***UNIX***, ***VAX***, ***VMS***, ***IBM OS/390***, ***OS/2*** и других.

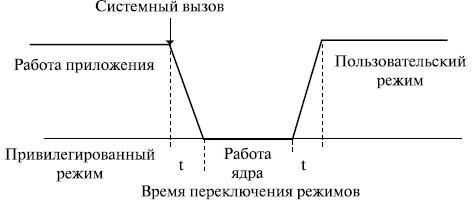


Рисунок 1.6 - Переключение режимов монолитного ядра ОС

### Замечание

Многослойная классическая многоуровневая архитектура ОС не лишена своих проблем:

* *значительные изменения одного из уровней* могут иметь трудно предвидимое влияние на смежные уровни;
* *многочисленные взаимодействия между соседними уровнями* усложняют обеспе- чение безопасности работы ОС.

**Альтернативой** классическому варианту архитектуры ОС является

*микроядерная архитектура ОС*. Суть этой архитектуры состоит в следующем:

* В привилегированном режиме работает только очень небольшая часть ОС, называемая *микроядром*.
* *Микроядро защищено* от остальных частей ОС и приложений.
* В состав микроядра входят *машинно-зависимые модули*, а также *модули, выполняющие базовые механизмы обычного ядра*.
* Все остальные, более высокоуровневые функции ядра, оформляются как *мо- дули, работающие в пользовательском режиме*.

На рисунке 1.7 представлено сравнение классической и микроядерной архи- тектур ОС:

* *менеджеры ресурсов*, являющиеся неотъемлемой частью обычного ядра, ста- новятся "периферийными" модулями, работающими в пользовательском ре-

жиме;

* *внешние*, по отношению к микроядру компоненты ОС, реализуются как обс- луживающие процессы;
* *между собой* эти модули взаимодействуют как равноправные партнеры с помощью обмена сообщениями, которые передаются через микроядро.

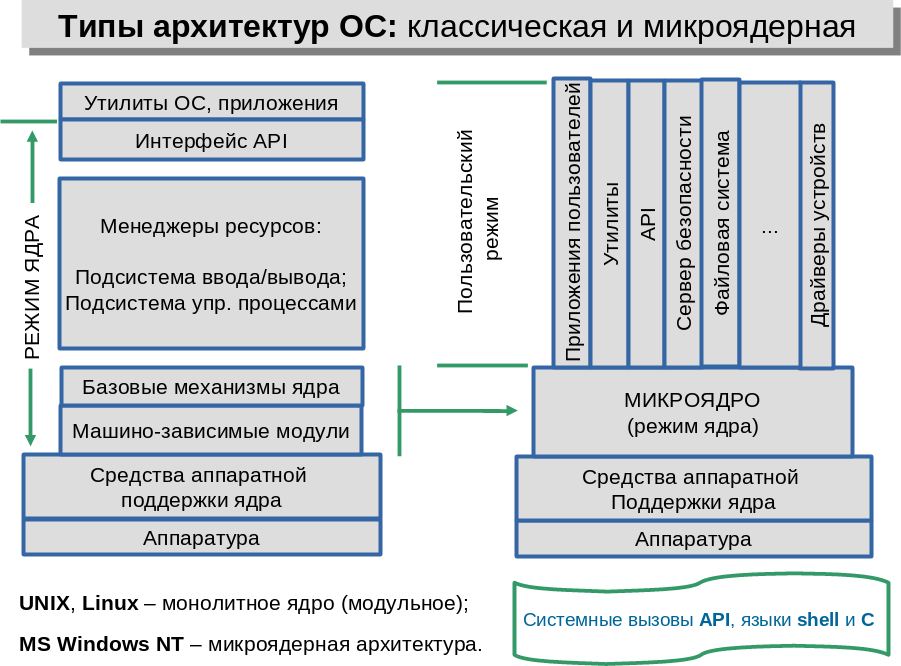


Рисунок 1.7 - Сравнение классической и микроядерной архитектур ОС

**Менеджеры ресурсов**, вынесенные в пользовательский режим, называются ***серве- рами ОС***. Схематично механизм обращений к функциям ОС, оформленным в виде серверов, выглядит, как показано на рисунке 1.8.

Для микроядра, схема смены режимов, при выполнении системного вызова в ОС, показана на рисунке 1.9. Хорошо видно, что выполнение системного вызова сопровождается *четырьмя переключениями режимов (4t)*, а в классической архитектуре – *двумя (2t)*. Следовательно, производительность ОС с микроядерной архитектурой, при прочих равных условиях, будет ниже, чем у ОС с классическим ядром.

**Замечание** По многим литературным источникам, вопрос масштабов потери производительности в микроядерных ОС является спорным. *Большинство ядер дистрибутивов ОС Linux являются монолитными*.



Рисунок 1.8 - Системные вызовы через микроядро ОС

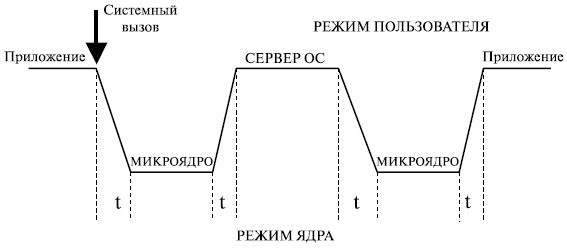


Рисунок 1.9 - Схема переключения режимов микроядерной архитектуры ОС

С **1990 года**, в рамках проекта ***GNU***, ведется разработка микроядерная архитектура ***Hurd ОС Linux***, основанная на микроядре ***GNU Mach***. **Ричард Столлман**, руководитель проекта GNU, ***в 2002 году*** заявил о скором выходе произ- водственной версии ***Hurd***, однако его обещания не оправдались.

**Микроядерными** являются ядра ***ОС Minix*** и ядро ***систем семейства BSD***.

**Windows NT** часто также называют микроядерной ОС, однако многие считают, что:

* *микроядро NT слишком велико* (более 1 Мбайт), чтобы носить приставку "микро";
* *все компоненты ядра работают в одном адресном пространстве* и активно используют общие структуры данных, что свойственно операционным сис- темам с монолитным ядром.

## Ядро и модули ОС

Как было отмечено выше, модули ядра выполняют основные базовые функ- ции ОС:

* *управление* процессами и памятью;
* *управление* устройствами ввода-вывода и другими элементами.

Хотя модульная организация ОС характерна для микроядерной архитектуры, монолитные ядра также используют модули. Вызвано это тем, что аппаратная часть ЭВМ настолько многообразна, что практически неэффективно создавать ядро на все возможные варианты конфигурации модулей.

**Традиционно**, взаимодействие ядра ОС с аппаратной частью ЭВМ осуществляется через специальное ПО, которое называется ***драйверами***.

**Любое ядро ОС** является одной большой программной, которая:

* *выполняется* в ***защищенном режиме*** (режиме ядра);
* *выполняется* в своем ***собственном адресном пространстве***.

Поэтому, перед компиляцией ядра запускается ***программа-конфигуратор***, в кото-

рой можно указать какие драйвера включаются в ядро ***статически***, а какие будут присутствовать как ***модули***.

Когда ядро ОС загружено в память ЭВМ и начинает работать, запускается первый пользовательский процесс ***init*** (или скрипт ***init***), имеющий ***PID=1***, который обеспечивает дальнейшую загрузку необходимых модулей.

**Для работы** с модулями ОС Linux имеет специальные утилиты:

* **lsmod** — просмотр списка модулей;
* **insmod** — инсталляция модулей;
* **rmmod** — удаление модуля;
* **modprobe** — может выполнять функции ***insmod*** и ***rmmod***;
* **modinfo** — получение информации о модуле.

## Три базовых концепции ОС: файл, пользователь, процесс

Подведем итог изученного ранее учебного материала. Нами были расмотрены различные концепции и представления:

* *модель СОД* раскрывает различные архитектурные возможности применения средств вычислительной техники; на уровне отдельной ЭВМ можно выде-

лить аппаратную и программную части; в программной части выделяется системное ПО, которое содержит объект нашего изучения — ОС;

* *концепции серверных ОС и ОС рабочих станций* в большей степени отража- ют особенности применения прикладного ПО, что находит свое отражение в

дистрибутивах ОС и их использования в пределах СОД;

* *концепция многослойной структуры ОС* отражает современную парадигму построения сложных программных систем; стремление повысить надежность

работы системного ПО приводит к выделению *ядра ОС*, работающего в при- велигированном режиме; стремление повысить эффективность работы сис- темного ПО приводит к созданию различных моделей ядер ОС, среди кото- рых были выделены: *монолитное ядро ОС* и *микроядро ОС*;

* *концепция базового ПО ЭВМ* является попыткой формализовать и стандарти- зировать наиболее важные понятия ОС, к которым в первую очередь относит-

ся ее ядро; формализуется *интерфейс прикладного программирования* (API) и *идея абстрактной (виртуальной) машины*;

* *модели взаимодействия режимов ядра и пользователя* наглядно показывают архитектурные возможности построения ОС; демонстрируются архитектуры

*классического (монолитного) ядра ОС* и *альтернативного — микроядра ОС*;

* *идея модульности ОС* является техническим решением, обеспечивающим устранение основного недостатка монолитных ядер — привязка к аппаратной

конфигурации конкретной ЭВМ; в частности, *ядро ОС УПК АСУ является монолитным и модульным*.

**Таким образом**, нами изучены основные архитектурные особенности ОС, *кроме представлений прикладного уровня*, которые характерны для ПО, работающего в режиме пользователя. Данный подраздел посвящен именно этому вопросу.

Современный пользователь воспринимает ОС с внешней стороны — на уров- не графической оболочки ОС. Эта оболочка представлена в виде *рабочего стола —* ***Desktop***, на котором имеются: окна, панели, меню, курсор мыши... **Некоторые ОС**, например, *MS Windows и MacOS*, не могут запуститься без графической оболочки. В других, например, *UNIX и Linux*, графическая оболочка является пользова- тельским приложением: *X Window (X-сервер)*.

Если рассматирвать работу ПО ЭВМ на профессиональном уровне, то можно утверждать:

* *в привигелированном* режиме процессора (*режим ядра*) работает ядро ОС;
* *в не привигелированном* режиме процессора (*режим пользователя*) работают утилиты, инструментальное и прикладное ПО ЭВМ;
* *для прикладного ПО* — режим пользователя представляет некоторую *среду*

*исполнения ОС*, которой ядро ОС управляет;

* *прикладное ПО реализует некоторую модель СОД* и обращается к ядру ОС за дополнительными функциями, в основном связанными с доступом к аппарат- ным средствам ЭВМ;
* *обращение прикладного ПО к ядру ОС* осуществляется посредством *систем- ных вызовов*;
* *классификация и абстрагирование* системных вызовов функций ядра ОС на более высоком уровне приводит к трем базовыми концепциям ОС: *файл, пользователь и процесс*.

**Таким образом**, мы пришли к понятию ***среды исполнения ОС***, которая опирается на три базовых концепции ***файл***, ***пользователь*** и ***процесс***, образующие иерархию отношений, показанную на рисунке 1.10.



Рисунок 1.10 - Базовые концепции пользовательского режима ОС

Все концепции, структурно представленные на рисунке 1.10, будут подробно изучаться в последующих темах. Здесь мы рассмотрим лишь основные понятия и идеи, которые уже известны из теории других дисциплин или без которых невоз- можно обойтись при выполнении лабораторных работ.

Первое понятие, которое следует обсудить, яаляется ***концепция файла***.

**ОС UNIX** (***Linux***) прямо декларируют парадигму: **«***Все есть файл***»**.

**К любому файлу** потенциально применимы три операции: **r - *чтения***, **w** - ***записи***

и **x** - ***запуска***.

**Конкретизация** понятия файл, в аспекте хранилища данных, формализуется в по- нятие *файловой системы*, которая представляет собой *поименованную совокуп- ность обычных и специальных файлов*.

**Обычный файл** — именованная упорядоченная последовательность байт.

**Специальный файл** имеет *имя* и *специализацию* по назначению:

* *устройства* — отображение аппаратных средств компьютера в файловую систему ОС;
* *директории* — файлы, представляющие ***список имен*** файлов и директорий; обеспечивают иерархическую структуру файловой системы ОС. Операция запуск, применительно к директории означает возможность «***войти в нее***»

пользователю;

* *ссылки* — именованные *указатели* на другие файлы, позволяющие работать с ними *по имени ссылки*.
* *именованные каналы* — точки доступа, через которые можно передавать данные;
* *сокеты* — точки доступа, через которые работает сетевое обеспечение ЭВМ.

Все операции с файлами интерпретируются через концепцию *пользователя*.

**Понятие пользователя** интерпретируется через свои элементы - ***владелец***, ***группа***

и ***другие***:

* *владелец* — именованный и индексированный объект ОС, права которого интерпретируются на операции с файлами; индекс владельца или ***UID* — User Identification** — целое число (от нуля и выше), которое присутствует в

каждом файле и интерпретируется как «хозяин файла»; чем меньше значение ***UID***, тем более «важным» является пользователь; например, пользователь ***root*** имеет ***UID=0*** и наивысшие права на файлы ОС;

* *группа* — именованный и индексированный объект ОС, предназначенный для объединения владельцев ОС; каждый владелец должен входить хотя бы в

одну группу; индекс группы или **GID — Group Identification**, аналогичен ***UID*** и выполняет те же функции для групп; владелец ***root*** имеет собственную группу с именем ***root*** и идентификатом ***GID=0***; все администраторы ОС обычно входят в *группу root*;

* *другие* — это пользователи без имени и идентификатора, которые дополняют концепцию пользователя и права которых также отображены в каждом файле.

Третья концепция - **процесс**, который обычно интерпретируется как *запущен- ная программа* или *задача*.

**Процесс** — это элементарный управляемый объект ОС, имеющий целочисленный идентификатор ***PID — Process Identification***, обеспечивающий функциональное преобразование файлов (данных) с правами, которые определяются объектами *пользователь*.

Значения **PID** начинаются с 1 (обычно - это процесс ***init***) — *главный роди-*

*тельский процесс*, и увеличиваются по мере *порождения дочерних процессов*.

**Новому процессу** присваивается номер на 1 больше, чем максимальный номер существующего или существовавшего с момента запуска ОС процесса.

## Системные вызовы fork(...) и exec(…)

**Ядро ОС** самостоятельно запускает только один процесс ***init***. Все остальные *про- цессы режима пользователя* являются дочерними относительно процесса ***init***.

Запуск любой программы в режиме пользователя осуществляется с помощью двух системных вызовов: *fork(...)* и ***exec(...)***.

**Вызов** функции *fork(...)* из программы на языке С, имеет вид:

**#include <unistd.h> pid\_t fork(void);**

Функция *fork(...)* полностью дублирует существующий процесс, вместе со всеми открытыми файлами, порождая новый (дочерний) процесс с новым ***PID***. Программист различает *родительский и дочерний процессы* только по целочислен- ному значению, которое возвращает функция *fork(...)*:

**-1 —** *ошибка*, дочерний процесс не создан**; 0** - дочерний процесс;

**> 0 —** *родительский процесс*, которому передано значение ***PID*** дочернего процесса.

**Замечание**

Родительский процесс обязан дождаться завершения дочернего процесса, иначе контроль передается по иерархии выше. Процесс ***init*** является родителем для всех остальных процессов.

Если дочерний процесс создан для запуска некоторой программы, то исполь- зуется одна из разновидностей системной функции *exec(...)*:

**#include <unistd.h> extern char \*\*environ;**

**int execl(const char \*path, const char \*arg, ...); int execp(const char \*file, const char \*arg, ...);**

**int execle(const char \*path, const char \*arg, …, char \* const envp[]); int execv(const char \*path, const \*char argv[]);**

**int execvp(const char \*file, const \*char argv[]);**

**int execvpe(const char \*file, const \*char argv[], char \* const envp[]);**

Хорошо видно, что любой вызов имеет ссылку на файл, в качестве аргумента:

* **После всех проверок** на права запуска, указанный файл загружается в прост-

ранство дочернего процесса.

* **Загруженной программе** передаются все ресурсы дочернего процесса, включая открытые и созданные файлы.

**Замечание**

Обратите внимание, что родительский процесс продолжает контролировать работу уже новой программы и *имеет право принудительно завершить ее работу*.

## Дистрибутивы ОС

Когда говорят, что на ЭВМ установлена некоторая ОС, то обычно подразу- мевают некоторый ее *дистрибутив*, включающий *конкретное ядро ОС и другое системное ПО*, а также прикладное ПО и системы разработки.

**Выбор** конкретного дистрибутива предполагает учет многих факторов:

* производитель дистрибутива;
* тип процессора, на который рассчитан дистрибутив;
* лицензия дистрибутива и ценовая политика дистрибьютора;
* поддержка национальных языков;
* типы носителей, на которых распространяется дистрибутив;
* особенности инсталляции;
* сопровождение дистрибутива;
* наличие документации.

**Учет** всех перечисленных факторов может оказаться довольно сложной задачей и выходит за рамки нашей дисциплины. Более того, каждому дистрибутиву и его версиям посвящены отдельные сайты, а многие дистрибутивы постоянно обсужда- ются на форумах в Интернете.

**Для целей обучения** выбран 64-битный базовый дистрибутив ***Arch Linux***. На его основе создан набор ПО, организованный как учебный программный комплекс кафедры АСУ (УПК АСУ).

**Замечание**

Структура ***ОС УПК АСУ*** ориентирована не только на задачи курса «***Операционные системы***», но имеет все необходимое ПО для его изучения и организации проведения лабораторных работ.