ТЕМА 1.3. АРХИТЕКТУРА ОПЕРАЦИОННЫХ СИСТЕМ

В данной теме рассматриваются следующие вопросы:

- Ресурсы операционных систем.
- Структура ядра и его функции.
- Принципы построения ядра.
- Объекты ядра.
- Основные операции над объектами ядра.
- Многослойная структура ОС.
- Средства аппаратной поддержки ОС.
- Машинно-зависимые компоненты ОС.
- Базовые механизмы ядра.
- Утилиты.
- Системные обрабатывающие программы.
- Библиотеки процедур.
- Программы дополнительных услуг.

Лекции – 2 часа, лабораторные занятия – 2 часа, самостоятельная работа – 2 часа.

Экзаменационные вопросы по теме:

- Структура ядра и его функции. Объекты ядра. Основные операции над объектами ядра.
- Утилиты. Системные обрабатывающие программы. Библиотеки процедур. Программы дополнительных услуг.

1.3.1. Ресурсы операционных систем

Термин «ресурс» обычно применяется по отношению к неоднократно используемым, относительно стабильным и «дефицитным» объектам, которые запрашиваются, используются и освобождаются процессами в период их активности. Другими словами, ресурсом является любой объект, который может распределяться внутри системы. Ресурсы могут быть разделяемыми, когда несколько процессов могут их использовать одновременно (в один и тот же момент времени) или параллельно (в течение некоторого интервала времени процессы используют ресурс попеременно), а могут быть и неделимыми (рис. 1.3.1) [1].

Рис. 1.3.1. Классификация ресурсов

В первых системах программирования под понятием «ресурсы» понимали процессорное время, память, каналы ввода/вывода и периферийные устройства. Однако скоро понятие ресурса стало более универсальным и общим.

Различного рода программные и информационные ресурсы также могут быть определены для системы как объекты, которые могут разделяться и распределяться, и доступ к которым необходимо контролировать. В настоящее время понятие ресурса превратилось в абстрактную структуру с целым рядом атрибутов, характеризующих способы доступа к этой структуре и ее физическое представление в системе. К ресурсам стали относиться и такие объекты, как сообщения и синхросигналы, которыми обмениваются задачи.

В первых вычислительных системах любая программа могла выполняться только после полного завершения предыдущей.

Поскольку такие вычислительные системы были построены в соответствии с принципами, изложенными в известной работе фон Неймана, все подсистемы и устройства компьютера функционировали исключительно под управлением центрального процессора. Центральный процессор осуществлял и выполнение вычислений, и управление операциями ввода/вывода данных. Соответственно, пока осуществлялся обмен данными между оперативной памятью и внешними устройствами, процессор не мог выполнять вычисления. Введение в состав ВМ специальных контроллеров позволило совместить во времени (распараллелить) операции вывода полученных данных и последующие вычисления на центральном процессоре. Однако по-прежнему процессор продолжал часто и подолгу простаивать, дожидаясь завершения очередной операции ввода/вывода. Поэтому было предложено организовать так называемый мультипрограммный (мультизадачный) режим работы ВС. Суть его заключается в том, что пока одна программа (один вычислительный процесс или задача) ожидает завершения очередной операции ввода/вывода, другая программа (а точнее, другая задача) может быть поставлена на решение.

ОС поддерживает режим мультипрограммирования и организует процесс эффективного использования ресурсов путем управления к ним очередью запросов, составляемых тем или иным способом. Данный режим достигается поддерживанием в памяти более одного процесса, ожидающего процессор, и более одного процесса, готового использовать другие ресурсы, как только последние станут доступными.

Общая схема выделения ресурсов

При необходимости использовать какой-либо ресурс (оперативную память, устройство ввода/вывода, массив данных и т.п.) задача обращается к супервизору ОС (центральному управляющему модулю, который может состоять из нескольких модулей, например, супервизора ввода/вывода, супервизора прерываний, супервизора программ, диспетчера задач и т.д.) и сообщает о своем требовании. При этом указывается вид ресурса и, если необходимо, его объем (например, количество адресуемых ячеек оперативной памяти, количество дорожек или секторов на системном диске, объем выводимых на устройство печати данных).

Директива обращения задачи к операционной системе передает ей управление, переводя процессор в привилегированный режим работы, если такой существует. Получив запрос, операционная система удовлетворяет его и после выполнения запроса ОС возвращает управление задаче, выдавшей данный запрос, или, если ресурс занят, ставит задачу в очередь, переводя ее в состояние ожидания (блокируя).

Ресурс может быть выделен задаче в следующих случаях:

1) ресурс свободен, и в системе нет запросов от задач более высокого приоритета к запрашиваемому ресурсу;

2) текущий запрос и ранее выданные запросы допускают совместное использование ресурсов;

3) ресурс используется задачей низшего приоритета и может быть временно отобран (разделяемый ресурс).

Схема освобождения ресурса

После окончания работы с ресурсом задача с помощью специального вызова супервизора посредством соответствующей директивы сообщает операционной системе об отказе от ресурса, либо операционная система самостоятельно забирает ресурс, если управление возвращается супервизору после выполнения какой-либо системной функции. Супервизор ОС, получив управление по этому обращению, освобождает ресурс и проверяет, имеется ли очередь к освободившемуся ресурсу. При наличии очереди в соответствии с принятой дисциплиной обслуживания и в зависимости от приоритета заявки он выводит из состояния ожидания ждущую ресурс задачу и переводит ее в состояние готовности к выполнению. После этого управление либо передается данной задаче, либо возвращается той, которая только что освободила ресурс.

Распределением ресурсов между задачами (процессами) пользователей и системными процессами занимается ядро операционной системы.

Классификация ресурсов

По реальности их существования [2]:

- физический – ресурс, который реально существует и при распределении обладает всеми физическими свойствами и характеристиками (диски, принтеры и сетевые устройства);
- виртуальный – мнимый ресурс, модель некоторого ресурса. которая реализуется в программно-аппаратной форме(RAM).

По возможности расширения свойств:

- эластичный – допускает виртуализацию;
- жёсткий – не допускает создание виртуального процесса.

По степени активности:

- активные – при использовании способны выполнять действия по отношению к другим ресурсам или процессам, которые приводят к изменению последних (ЦП);

- пассивные – ресурсы, над которыми можно производить дополнительные действия, которые приводят к их изменению (RAM).

По времени существования:

- постоянные – существуют до зарождения процесса, во время существования процесса и возможно будет существовать после процесса (ПЗУ);
- временные – появляются и уничтожаются в системе динамически, т.е. в течение времени существования процесса. Создание или уничтожение временных ресурсов может производиться самим процессом или другими процессами (виртуальный диск).

По степени важности:

- главный – без выделения этих ресурсов процесс принципиально существовать не может (ЦП, RAM);
- второстепенный – допускает некоторые альтернативные различия (хранение данных на HDD).

По функциональной избыточности:

- дорогие;
- дешевые.

По структуре:

- простые – ресурсы, которые не содержат составных частей;
- составные.

По восстанавливаемости:

- воспроизводимые ресурсы – ресурсы, при распределение которых допускается многократное выполнение следующей последовательности: запрос использование освобождение;
- потребляемые ресурсы – ресурсы, при распределении которых выполняется следующая последовательность: освобождение запрос использование.

По характеру использования:

- последовательно используемые – в отношение, которого допустимо строго последовательное действие: запрос использование освобождение (печатное устройство);
- параллельно используемые – ресурсы, которые одновременно используются более чем одним процессом (массив данных находится в некоторой области RAM);
- критически последовательный ресурс, разделяемый несколькими параллельными процессами (буфер, хранящий принятые данные).

1.3.2. Структура ядра и его функции

Под архитектурой ОС обычно понимают структурную организацию ОС на основе программных модулей [3].

Современные ОС представляют собой хорошо структурированные модульные системы.

Единой архитектуры ОС не существует, но существуют универсальные подходы к структурированию ОС, как показано на рис. 1.3.2.

Рис. 1.3.2. Обобщенная структура операционной системы

Наиболее общим подходом к структуризации ОС является подразделение модулей две группы:

- модули, выполняющие основные функции ОС — **ядро ОС**;
- модули, выполняющие вспомогательные функции ОС.

Модули ядра выполняют базовые функции ОС

- управление процессами;
- управление памятью;
- управление устройствами ввода-вывода.

Функции, входящие в состав ядра, можно разделить на два класса.

1 класс. Функции для решения внутрисистемных задач организации вычислительного процесса (переключение контекстов процессов, загрузка/выгрузка страниц, обработка прерываний). Эти функции недоступны для приложений.

2 класс. Функции для поддержки приложений (доступны приложениям). Эти функции создают для приложений так называемую прикладную программную среду и образуют интерфейс прикладного программирования — API. Приложения обращаются к ядру с запросами — системными вызовами. Функции API обслуживают системные вызовы — предоставляют доступ к ресурсам системы в удобной и компактной форме, без указания деталей их физического расположения.

Функции модулей ядра — наиболее часто используемые функции ОС:

- скорость выполнения этих функций определяет производительность всей системы в целом;
- все (большинство) модули ядра являются резидентными.

Остальные модули ОС выполняют полезные, но менее обязательные функции. Например, к таким вспомогательным модулям могут быть отнесены программы архивирования, дефрагментации диска и т.п. Вспомогательные модули ОС оформляются либо в виде приложений, либо в виде библиотек процедур и функций.

Обычно вспомогательные модули подразделяются на следующие группы:

- **утилиты** — программы, которые решают отдельные задачи управления и сопровождения компьютерной системы (сжатие, дефрагментация …);
- **библиотеки процедур и функций различного назначения** (библиотека математических функций, библиотека функций ввода-вывода и т.д.);
- **программы предоставления пользователю дополнительных услуг** — специальный вариант пользовательского интерфейса, калькулятор, некоторые игры (какие, например, поставляются в составе ОС);
- **системные обрабатывающие программы** — текстовые и графические редакторы, компиляторы, компоновщики, отладчики.

Вспомогательные модули ОС обращаются к функциям ядра, как и обычные приложения, посредством системных вызовов.

Многие модули ОС оформлены как обычные приложения. Решение о том, является ли какая-либо программа частью ОС или нет, принимает производитель ОС.

Главные задачи, решаемые ядром

Ядро операционной системы включает в свой состав следующие программы, решающие соответствующие задачи:

- Программа обработки прерываний.
- Программа для формирования и ликвидации различных процессов.
- Программа переключения состояний процессов.
- Программа, выполняющая диспетчеризацию процессов.
- Программа для приостановки и последующей активации процесса.
- Программа синхронизации выполняемых процессов и организации обменов данными между ними.
- Программа по выполнению ввода и вывода данных.
- Программа управления функциями распределения ресурсов памяти.
- Программа поддержки файловых систем.
- Программа обеспечения вызова и возврата при работе с процедурами.
- Программа поддержки операций по учёту работ ЭВМ.

Подсистемы

Модуль подсистемы управления процессами выполняет следующие функции:

- создание и удаление процессов;
- распределение системных ресурсов между процессами;
- синхронизацию процессов;
- взаимодействие процессов.

Специальная функция ядра, выполняемая **планировщиком процессов** (scheduler), разрешает конфликты между процессами в конкурентной борьбе за системные ресурсы.

Модуль подсистемы управления памятью обеспечивает распределение памяти между процессами. Если для всех процессов недостаточно памяти, ядро перемещает части процесса или несколько процессов (чаще пассивных, ожидающих каких-либо событий в системе) в специальную область диска (область «подкачки»), освобождая ресурсы для выполняющихся (активных) процессов.

Файловая подсистема обеспечивает унифицированный интерфейс доступа к данным, расположенным на дисковых накопителях, и к периферийным устройствам. Одни и те же функции **`open`**, **`read`**, **`write`** могут использоваться как при чтении или записи данных на диск, так и при выводе текста на принтер или терминал [4].

Файловая подсистема контролирует права доступа к файлу, выполняет операции размещения и удаления файла, а также выполняет запись/чтение данных файла. Поскольку большинство прикладных функций выполняется через интерфейс файловой системы (в том числе и доступ к периферийным устройствам), права доступа к файлам определяют привилегии пользователя в системе.

Файловая подсистема обеспечивает перенаправление запросов, адресованных периферийным устройствам, соответствующим модулям подсистемы ввода/вывода.

Подсистема ввода/вывода выполняет запросы файловой подсистемы и подсистемы управления процессами на доступ к периферийным устройствам. Она взаимодействует с драйверами устройств — специальными программами ядра, обслуживающими внешние устройства.

Ядро в сравнении с пользовательскими программами

Ядро операционной системы имеет некоторые специфические особенности в сравнении с программами, которые выполняются в пространстве пользователя. Вот некоторые из них для операционных систем с ядром Linux [5]:

• **Ядро не имеет доступа к стандартным библиотекам языка С**. Причина этого – скорость выполнения и объем кода. Даже самая необходимая часть библиотеки – очень большая и неэффективная для ядра. Часть функций, однако, реализованы в ядре. Например, обычные функции работы со строками описаны в файле `lib/string.c`.

• **Отсутствие защиты памяти**. Если обычная программа предпринимает попытку некорректного обращения к памяти, ядро может аварийно завершить процесс. Если ядро предпримет попытку некорректного обращения к памяти, результаты могут быть менее контролируемыми. К тому же ядро не использует замещение страниц: каждый байт, используемый в ядре – это один байт физической памяти.

• **В ядре нельзя использовать вычисления с плавающей точкой**. Активизация режима вычислений с плавающей точкой требует сохранения и проставления регистров устройства поддержки вычислений с плавающей точкой, помимо других рутинных операций.

• **Фиксированный стек**. Стеком называют область адресного пространства, в которой выделяются локальные переменные. Локальные переменные – это все переменные, объявленные внутри левой открывающей фигурной скобки тела функции (или любой другой левой фигурной скобки) и не имеющие ключевого слова static. Стек в режиме ядра ни большой, ни изменяющийся. Поэтому в коде ядра не рекомендуется использовать рекурсию. Обычно стек равен двум страницам памяти, что соответствует 8 Кбайт для 32-разрядных систем и 16 Кбайт для 64 -разрядных.

• **Переносимость**. Платформо-независимый код, написанный на языке С, должен компилироваться без ошибок на максимально возможном количестве систем.

1.3.3. Принципы построения ядра

Ядро (kernel) — это центральная и главная часть операционной системы, которая обеспечивает архитектуру связи с приложениями, организует и регулирует доступ к ресурсам компьютера. Дополнительно, но не как правило, предоставляет доступ к сетевым протоколам и к файловой системе [6]. На рис. 1.3.3 показано, что ядро операционной системы взаимодействует с приложениями пользователя, с утилитами и с системными обрабатывающими программами. Обратите внимание, что из всех изображенных элементов только библиотеки процедур не взаимодействуют напрямую, а являются «переправой» для взаимодействия пользователя с ядром.

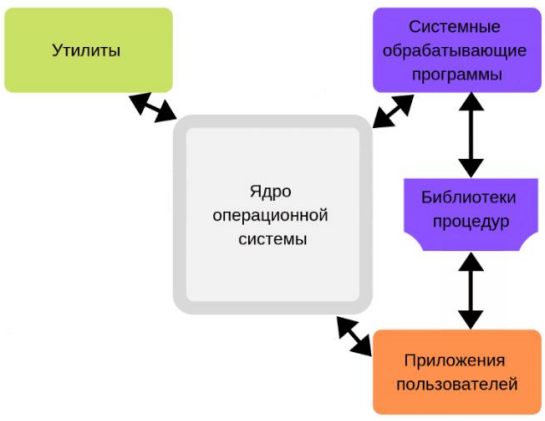


Рис. 1.3.3. Ядро операционной системы

Для того, чтобы многозадачная ОС могла выполнять функции защиты приложений от влияния друг друга и защиты самой себя от приложений, для нее на аппаратном уровне обеспечиваются определенные привилегии. Подразумевается, что ОС или некоторые ее части работают в привилегированном режиме, а приложения — в пользовательском режиме.

Привилегии ОС обеспечиваются тем, что выполнение некоторых инструкций в пользовательском режиме запрещается. Например, выполнение инструкции доступа к памяти для приложения разрешается, если происходит обращение к области памяти, отведенной данному приложению, и запрещается при обращении к областям памяти, занимаемым ОС или другими приложениями.

Так как ядро выполняет все основные функции ОС, то чаще всего именно ядро — та часть ОС, которая работает в привилегированном режиме.

Термин **ядро** в различных ОС трактуется по-разному. Но чаще всего, именно это свойство — *работа в привилегированном режиме* — служит основным определением понятия **ядра**.

> ***Ядро*** *— низкоуровневая основа любой операционной системы, выполняемая аппаратурой в особом привилегированном режиме. Ядро загружается в память один раз и находится в памяти резидентно (постоянно), по одним и тем же адресам.*

Каждое приложение пользовательского режима работает в своем адресном пространстве и защищено тем самым от вмешательства других приложений. Код ядра имеет доступ к областям памяти всех приложений, но сам полностью от них защищен. Приложения обращаются к ядру с запросами на выполнение системных функций.

Необходимо обратить внимание, на то, что работа системы с привилегированным ядром замедляется за счет замедления выполнения системных вызовов.

Системный вызов привилегированного ядра инициирует переключение процессора из пользовательского режима в привилегированный, а при возврате к приложению — переключение из привилегированного режима в пользовательский.

Виды структуры ядра операционной системы

Возможны следующие типы структуры (архитектуры) ядра операционной системы [7]:

- Монолитная структура ядра.
- Модульная структура ядра.
- Микроструктура ядра.
- Экзо структура ядра.
- Нано структура ядра.
- Гибридная структура ядра.
- Комбинированная структура ядра.

Монолитное ядро формируется из обширного комплекта абстракций оборудования. Все элементы монолитного ядра работают в едином адресном формате. При такой организации операционной системы все составляющие части её ядра выступают как элементы основной программы, применяют одни и те же системы организации данных и работают друг с другом, используя непосредственный вызов процедуры. Определённой структуры данные операционной системы не имеют. По сути, это большой набор сервисных функций. Нет деления на слои и модули.

Это самый старый метод формирования операционной системы. В качестве примера можно привести UNIX и классический MS-DOS. Достоинством является большая скорость выполнения операций и простота конструирования модулей. Недостатком можно считать работу ядра в едином адресном пространстве, так как неисправность в любом элементе способна блокировать работу всей системы.

Модульное ядро является современной и модифицированной версией структуры монолитного ядра операционной системы. Она отличается от классического монолитного ядра тем, что не требует общей реструктуризации ядра при различных вариациях аппаратной оснастки компьютеров. У модульных ядер есть возможность подгружать различные модули (элементы) ядра, которые поддерживают нужное аппаратное оборудование (как пример, загрузка драйвера), причём подзагрузка модуля возможна как в динамическом режиме, то есть без перезагрузки операционной системы, так и в статике, когда выполняется переконфигурирование системы и её перезагрузка. Примеры модулей ядра в Linux: драйверы устройств, файловые системы, сетевые протоколы.

Преимущества:

Производительность — следствие того, что количество переключений из режима контекста пользователя в режим ядра минимально;

Недостатки:

Неустойчивость к сбоям —все базовые компоненты выполняются в режиме ядра, и если хотя бы в одном модуле или блоке ядра произойдет какой-либо сбой, то ему будет подвержена вся операционная система (все ядро), что потребует перезапуска операционной системы.

Микроядро решает лишь самые простые задачи по управлению процессами и имеет небольшой комплект абстракций оборудования. Основная часть функций выполняется специальными процессами пользователя, которые называются сервисами. Главным признаком микроядра можно считать распределение практически всех драйверов и элементов в процессах сервиса. Часто нет возможности загрузки расширительных модулей в такое микроядро.

Операционные системы, основанные на микроядерной архитектуре, удовлетворяют большинству требований, предъявляемым к современным ОС, и их преимуществами являются [3]:

- **обладают переносимостью** (весь машинно-зависимый код изолирован в микроядре, следовательно, нужно мало изменений при переносе системы на новый процессор, к тому же все изменения сгруппированы вместе);
- **высокая степень расширяемости** (для того, чтобы добавить новую подсистему требуется разработать новое приложение, для чего не требуется затрагивать микроядро; с другой стороны, пользователь легко может удалить ненужные подсистемы, удалять из ядра было бы сложнее);
- **достаточная надежность** (каждый сервер ОС выполняется в своем адресном пространстве и таким образом защищен от других серверов, кроме того, если происходит крах одного сервера, он может быть перезапущен без останова или повреждения других серверов).
- **поддержка распределенных вычислений** (серверы ОС могут работать на разных компьютерах - асимметричная ОС; возможен перенос однопроцессорных ОС на распределенные компьютеры)

Основной (и по сути единственный) **недостаток** микроядерной архитектуры — **снижение производительности** из-за накладных расходов.

Экзоядро — это ядро операционной системы, которое предоставляет только возможность взаимного обмена между процессами и надёжного распределения и высвобождения ресурсов.

Экзоядро распределяет ресурсы между виртуальными машинами и отслеживает попыток их использования, чтобы ни одна из машин не пыталась использовать чужие ресурсы [8]. Преимущество схемы экзоядра заключается в том, что она исключает уровень отображения. При других методах работы каждая виртуальная машина считает, что она имеет собственный диск с нумерацией блоков от 0 до некоторого максимума. Поэтому

монитор виртуальных машин должен вести таблицы преобразования адресов на диске (и всех других ресурсов). При использовании экзоядра необходимость в таком переназначении отпадает. Экзоядру нужно лишь отслеживать, какой виртуальной машине какие ресурсы были переданы. Такой подход имеет еще одно преимущество — он отделяет многозадачность (в экзоядре) от пользовательской операционной системы (в пространстве пользователя) с меньшими затратами, так как для этого ему необходимо всего лишь не допускать вмешательства одной виртуальной машины в работу другой.

Наноядро имеет такую структуру, при которой очень простое ядро решает лишь проблему обработки аппаратного прерывания программы, которое генерируют различные блоки компьютера. Когда обработка прерывания, например, при нажатии символа на клавиатуре, завершается, наноядро пересылает результаты программе, которая выше по рангу. При этом пересылка тоже выполняется посредством прерываний.

Гибридное ядро представляет собой модификацию микроядра, которая позволяет ускорить работу системы.

И, наконец, возможен вариант **комбинированного применения** вышеперечисленных вариантов построения структуры ядра операционной системы.

1.3.4. Объекты ядра

Объекты ядра используются системой и приложениями для управления множеством разных ресурсов: процессами, потоками, файлами и т.д. [9]. Приложение не может напрямую обращаться к объектам ядра читать и изменять их содержимое.

Понятие "объект ядра" имеет разный смысл не только в разных операционных системах, но даже у разных авторов, описывающих одну и ту же операционную систему [10]. К тому же, одни объекты существуют во всех операционных системах (процесс, поток), а другие специфичны для конкретной операционной системы (WindowStation). Здесь будет представлена только общая картина и несколько отрывочных иллюстраций. Многие из объектов ядра будут подробно рассматриваться в соответствующих темах.

За рабочее примем определение объектов ядра из документации Microsoft Learn.

> Объект — это коллекция данных, являющихся частью режима ядра операционной системы, которыми управляет Диспетчер объектов Windows (Windows Object Manager).

Объекты ядра в Windows

Типичные объекты режима ядра включают следующие категории объектов:

- объекты устройств;
- объекты файлов;
- символические ссылки;
- разделы реестра;
- потоки и процессы;
- объекты диспетчера ядра, такие как объекты событий и объекты мьютексов;
- объекты обратного вызова;
- объекты section.

В качестве примера конкретных объектов можно привести: маркеры доступа (access token objects), файлы (file objects), проекции файлов (file-mapping objects), порты завершения ввода вывода (I/O completion port objects), задания (jobs), почтовые ящики (mailslot objects), мьютексы (mutex objects), каналы (pipe objects), процессы (thread objects) и ожидаемые таймеры (waitable timer objects).

Каждый объект ядра на самом деле просто блок памяти, выделенный ядром и доступный только ему. Блок представляет собой структуру данных, в элементах которой содержится информация об объекте. Некоторые элементы (дескриптор защиты, счетчик числа

пользователей и др.) присутствуют во всех объектах, но большая их часть специфична для объектов конкретного типа. Например, у объекта процесс может быть идентификатор, базовый приоритет и код завершения, а у объекта файл - смещение в байтах, режим разделения и режим открытия. [9]

> ***Примечание**. Объекты в режиме ядра позволяют управлять объектами в партнерстве с диспетчером объектов, не повреждая части объектов, необходимые операционной системе. Этот принцип называется **инкапсуляцией** и является одним из основных понятий объектно-ориентированного программирования. (Так как объекты в режиме ядра не предоставляют другие аспекты объектной ориентации, программирование в режиме ядра обычно называется **объектным**.) Объекты в режиме ядра **не** следуют тем же правилам, что и объекты в C++ или Microsoft COM* [11].

Для взаимодействия с объектами ядра у операционной системы предусмотрен набор функций, обрабатывающих структуры объектов ядра по строго определенным правилам. Например, когда мы создаем объект ядра в операционной системе Windows, функция возвращает **описатель**, идентифицирующий созданный объект (HANDLE). Все операции с текущим объектом ядра возможны только при указании этого описателя управляющей функции. Для большей защиты, описатели уникальны только внутри конкретного процесса. Поэтому, передавая по межпроцессорной связи описатель объекта другому процессу, используя описатель в другом процессе, мы получим ошибку [9].

Объекты ядра принадлежат ядру, а не процессу. Это говорит о том, что, завершая работу с процессом, мы не обязательно разрушаем объект ядра. В большинстве случаев объект разрушается, но, если созданный вами объект ядра используется другим процессом, ядро запретит разрушение объекта до тех пор, пока от него не откажется последний пользователь.

Объекты в режиме ядра имеют очень определенный жизненный цикл. В каждом объекте, есть счетчик пользователей объектом. В момент создания объекта счетчику присваивается значение 1. Соответственно, при обращении к нему другого процесса, счетчик увеличивается на 1. Когда пользовательский процесс завершается, счетчик уменьшается. Система удаляет объект ядра, когда счетчик обнуляется.

Среда режима ядра хранит объекты в виртуальной системе каталогов, также называемой пространством имен объектов. Это позволяет получить иерархический доступ к объектам с помощью родительских и дочерних объектов. Это пространство имен похоже на набор каталогов файловой системы, но не соответствует определенной файловой системе на компьютере [11]. В качестве нетерминальной вершины дерева используется объект "каталог объектов". Каталог включает информацию, необходимую для трансляции имен объектов в указатели на сами объекты. Вследствие необходимости выполнения навигации по каталогам ссылка на объект по имени работает существенно дольше, чем по описателю. "Увидеть" пространство имен можно только при помощи специальных инструментальных средств, например, с помощью утилиты WinObj из комплекта SysInternals от Марка Руссиновича, доступной по URI https://learn.microsoft.com/en-us/sysinternals/downloads/winobj (рис. 1.3.4).

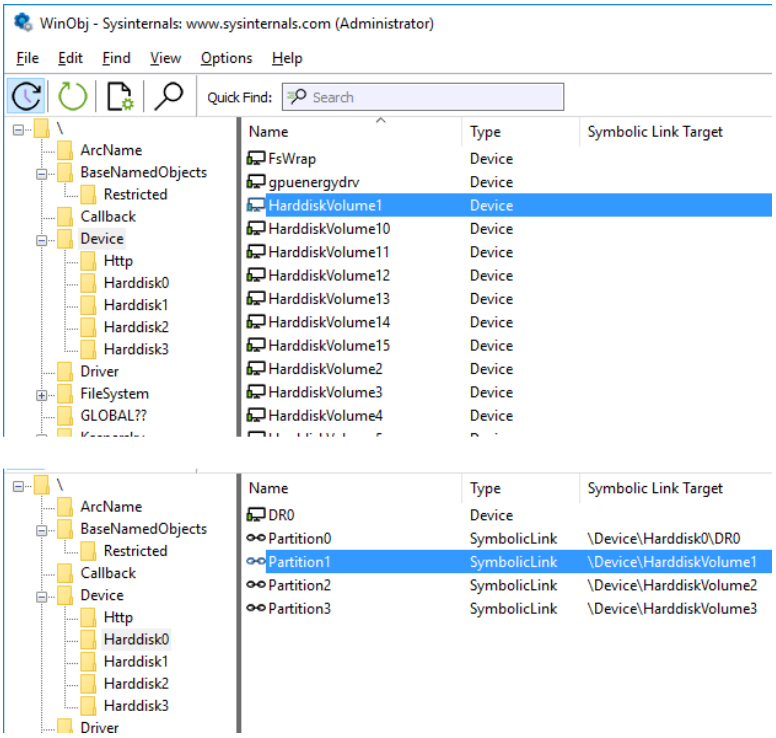


Рис. 1.3.4. Отображение объектов ядра в программе WinObj

1.3.5. Основные операции над объектами ядра

1.3.5.1. Основные операции над объектами ядра в Windows

В состав компонентов объекта типа входит атрибут методы — указатели на внутренние процедуры для выполнения стандартных операций. Методы вызываются диспетчером объектов при создании и уничтожении объекта, открытии и закрытии описателя объекта, изменении параметров защиты.

Первое впечатление об основных операциях над объектами ядра в Windows можно получить, посмотрев на список разрешений дескриптора безопасности объекта ядра (рис. 1.3.5).

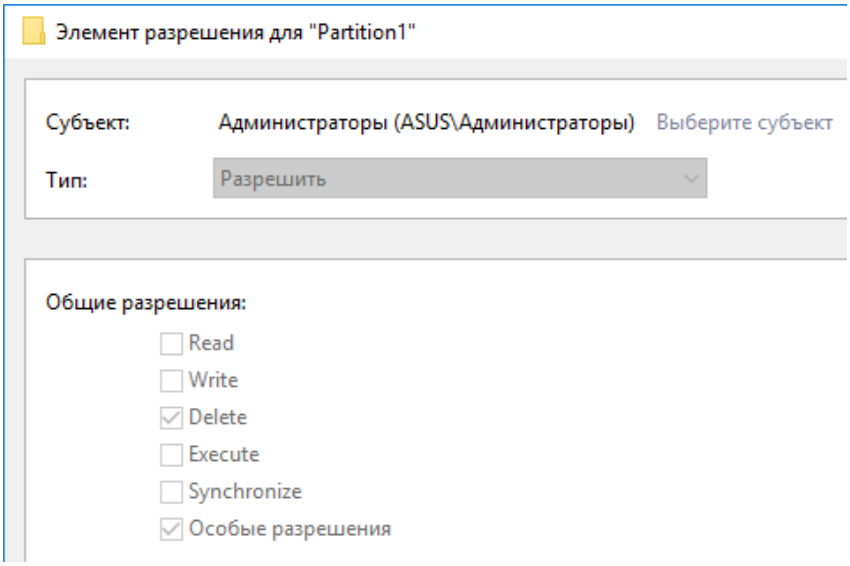


Рис. 1.3.5. Общие разрешения для объекта ядра

Система позволяет динамически создавать новые типы объектов. В этом случае предполагается регистрация его методов у диспетчера объектов. Например, метод **open** вызывается всякий раз, когда создается или открывается объект и создается его новый описатель [10].

Создание новых объектов, или открытие по имени уже существующих, приложение может осуществить при помощи Win32-функций, таких, как **CreateFile**, **CreateSemaphore**, **OpenSemaphore** и т.д. Это библиотечные процедуры, за которыми стоят сервисы Windows и методы объектов. В случае успешного выполнения создается 64-битный описатель в таблице описателей процесса в памяти ядра. На эту таблицу есть ссылка из блока управления процессом EPROCESS.

Рассмотрим операции с объектами ядра на примере работы с файлами. Прежде чем прочитать данные из файла, нужно получить описатель. Для этого используется «функция» **CreateFile**. На самом деле это макрос, который раскрывается в одну из функций: **CreateFileA** для имен в кодировке ANSI и **CreateFileW** для имен в кодировке Unicode. Прототипы всех функций для работы в файлами можно найти в заголовочном файле `fileapi.h`.

Ниже приведен синтаксис функции **CreateFileW**.

```
HANDLE CreateFileW(
  [in]           LPCWSTR               lpFileName,
  [in]           DWORD                 dwDesiredAccess,
  [in]           DWORD                 dwShareMode,
  [in, optional] LPSECURITY_ATTRIBUTES lpSecurityAttributes,
  [in]           DWORD                 dwCreationDisposition,
  [in]           DWORD                 dwFlagsAndAttributes,
  [in, optional] HANDLE                hTemplateFile
);
```

Первый параметр – полный или относительный путь к файлу. Чтобы обратиться к пространству имен устройства Win32, а не к пространству имен файлов Win32, нужно использовать префикс `\\.\` перед именем устройства, например, `"\\.\C:"`.

> *Примечание. Функция **CreateFile**, несмотря на первое впечатление от ее имени, используется и при работе с уже существующими файлами, так как слово File здесь обозначает не файл на диске, а объект ядра типа File.*

Полученный описатель затем нужно использовать как первый параметр в различных функциях, например, **ReadFile** и **WriteFile**, для работы с реальным файлом.

Ниже приведен синтаксис функции **WriteFile**.

```
BOOL WriteFile(
    _In_ HANDLE hFile,
    _In_reads_bytes_opt_(nNumberOfBytesToWrite) LPCVOID lpBuffer,
    _In_ DWORD nNumberOfBytesToWrite,
    _Out_opt_ LPDWORD lpNumberOfBytesWritten,
    _Inout_opt_ LPOVERLAPPED lpOverlapped
    );
```

Во избежание утечки памяти всегда рекомендуется закрывать объект, когда в нем отпала надобность. Впрочем, по окончании работы процесса система закрывает все его объекты. Одной из таких функций является функция **CloseHandle**.

```
BOOL CloseHandle(
  [in] HANDLE hObject
);
```

Некоторые дескрипторы требуют своей функции закрытия, например, функция **FindClose** должна применяться для закрытия дескриптора, полученного от функции **FindFirstFile**.

Как показано на рис. 1.3.5, некоторые объекты ядра имеют разрешение Synchronize. Это означает, что их можно использовать как для синхронизации потоков, что будет рассмотрено в теме 2.3.

1.3.6. Многослойная структура ОС

Управление ресурсами включает решение следующих общих, не зависящих от типа ресурса задач [12]:

- планирование ресурса — то есть определение, какому процессу, когда и в каком количестве (если ресурс может выделяться частями) следует выделить данный ресурс;
- удовлетворение запросов на ресурсы;
- отслеживание состояния и учет использования ресурса — то есть поддержание оперативной информации о том, занят или свободен ресурс и какая доля ресурса уже распределена;
- разрешение конфликтов между процессами.

Для решения этих общих задач управления ресурсами разные ОС используют различные алгоритмы. Задача организации эффективного совместного использования ресурсов несколькими процессами является весьма сложной, и сложность эта порождается в основном случайным характером возникновения запросов на потребление ресурсов. Управление ресурсами составляет важную часть функций любой операционной системы.

В пункте 1.3.3 при описании монолитных систем было показано, что простая неорганизованная структура – это плохо. Для улучшения ситуации операционную систему разбивают на ряд уровней (слоев), где каждый уровень базируется на предыдущем, то есть, вводится понятие иерархии уровней (рис. 1.3.6). Самый нижний — аппаратное обеспечение, самый верхний — интерфейс пользователя. Каждый уровень использует только функции (сервисы), предоставляемые нижестоящим уровнем. Все или почти все уровни работают в режиме ядра.

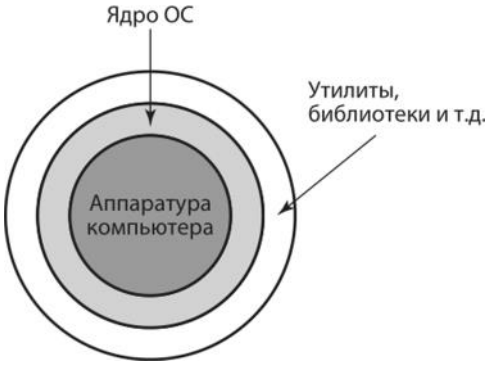
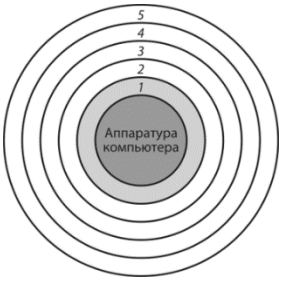


Рис. 1.3.6. Трехслойная структура операционной системы

Многослойные системы хорошо реализуются. При использовании операций нижнего слоя не нужно знать, как они реализованы, нужно лишь понимать, что они делают. Такие системы хорошо тестируются. Отладка начинается с нижнего слоя и проводится послойно. При возникновении ошибки мы можем быть уверены, что она находится в тестируемом слое. Многослойные системы хорошо модифицируются. При необходимости можно заменить лишь один слой, не трогая остальные. Но многослойные системы сложны для разработки: тяжело правильно определить порядок слоев и что к какому слою относится. Многослойные системы менее эффективны, чем монолитные. Так, например, для выполнения операций ввода-вывода программе пользователя придется последовательно проходить все слои от верхнего до нижнего.

Поскольку ядро операционной системы представляет собой сложный многофункциональный комплекс, то многослойный подход обычно распространяется и на структуру ядра [13]. Ядро может состоять из следующих слоев (рис. 1.3.7).



ис. 1.3.7. Многослойная архитектура ядра ОС: 1 — средства аппаратной поддержки ОС; 2 — машинно-зависимые модули; 3 — базовые механизмы ядра; 4 — менеджеры ресурсов; 5 — интерфейс системных вызовов

Средства аппаратной поддержки операционной системы. До сих пор об операционной системе говорилось как о комплексе программ, хотя часть функций операционной системы может выполняться и аппаратными средствами. В этом случае речь идет не о всей аппаратуре компьютера, а только об аппаратуре, непосредственно участвующей в организации вычислительного процесса. Например, средства поддержки привилегированного режима, система прерываний, средства защиты памяти и т.п.

Машинно-зависимые компоненты операционной системы. Этот слой образуют программные модули, в которых отражается специфика аппаратной платформы компьютера. За счет этого слоя происходит полное экранирование вышележащих слоев ядра от особенностей аппаратной реализации, тем самым позволяя разрабатывать вышележащие слои на основе машинно-независимых модулей.

Базовые механизмы ядра. Этот слой выполняет наиболее примитивные операции ядра. Модули данного слоя не принимают решений о распределении ресурсов — они только отрабатывают принятые «наверху» решения, что и дает повод называть их исполнительными механизмами для модулей верхних слоев.

Менеджеры ресурсов. Этот слой состоит из мощных функциональных модулей, реализующих стратегические задачи по управлению основными ресурсами компьютера. Обычно на данном слое работают менеджеры (диспетчеры) процессов, ввода-вывода, файловой системы и оперативной памяти.

Интерфейс системных вызовов. Этот слой является самым верхним слоем ядра и взаимодействует непосредственно с приложениями и системными утилитами, образуя прикладной программный интерфейс операционной системы.

Приведенное разбиение ядра операционной системы на слои является достаточно условным. В реальной системе количество слоев и распределение функций между ними могут быть и иными.

1.3.7. Средства аппаратной поддержки ОС

Аппаратная зависимость и переносимость ОС

Многие операционные системы успешно работают на различных аппаратных платформах без существенных изменений в своем составе. Во многом это объясняется тем, что, несмотря на различия в деталях, средства аппаратной поддержки ОС большинства компьютеров приобрели сегодня много типовых черт, а именно эти средства в первую очередь влияют на работу компонентов операционной системы. В результате в ОС можно выделить достаточно компактный слой машинно-зависимых компонентов ядра и сделать остальные слои ОС общими для разных аппаратных платформ.

Типовые средства аппаратной поддержки ОС

Четкой границы между программной и аппаратной реализацией функций ОС не существует — решение о том, какие функции ОС будут выполняться программно, а какие аппаратно, принимается разработчиками аппаратного и программного обеспечения

компьютера. Тем не менее практически все современные аппаратные платформы имеют некоторый типичный набор средств аппаратной поддержки ОС, в который входят следующие компоненты:

- средства поддержки привилегированного режима;
- средства трансляции адресов;
- средства переключения процессов;
- система прерываний;
- системный таймер;
- средства защиты областей памяти.

Средства поддержки привилегированного режима обычно основаны на системном регистре процессора, часто называемом «словом состояния» машины или процессора. Этот регистр содержит некоторые признаки, определяющие режимы работы процессора, в том числе и признак текущего режима привилегий. Смена режима привилегий выполняется за счет изменения слова состояния машины в результате прерывания или выполнения привилегированной команды. Число градаций привилегированности может быть разным у разных типов процессоров, наиболее часто используются два уровня (ядро-пользователь) или четыре (например, ядро- супервизор- выполнение-пользователь у платформы VAX или 0-1-2-3 у процессоров Intel x86/Pentium). В обязанности средств поддержки привилегированного режима входит выполнение проверки допустимости выполнения активной программой инструкций процессора при текущем уровне привилегированности.

Средства трансляции адресов выполняют операции преобразования виртуальных адресов, которые содержатся в кодах процесса, в адреса физической памяти. Таблицы, предназначенные при трансляции адресов, обычно имеют большой объем, поэтому для их хранения используются области оперативной памяти, а аппаратура процессора содержит только указатели на эти области. Средства трансляции адресов используют данные указатели для доступа к элементам таблиц и аппаратного выполнения алгоритма преобразования адреса, что значительно ускоряет процедуру трансляции по сравнению с ее чисто программной реализацией.

Средства переключения процессов предназначены для быстрого сохранения контекста приостанавливаемого процесса и восстановления контекста процесса, который становится активным. Содержимое контекста обычно включает содержимое всех регистров общего назначения процессора, регистра флагов операций (то есть флагов нуля, переноса, переполнения и т. п.), а также тех системных регистров и указателей, которые связаны с отдельным процессом, а не операционной системой, например, указателя на таблицу трансляции адресов процесса. Для хранения контекстов приостановленных процессов обычно используются области оперативной памяти, которые поддерживаются указателями процессора.

Переключение контекста выполняется по определенным командам процессора, например, по команде перехода на новую задачу. Такая команда вызывает автоматическую загрузку данных из сохраненного контекста в регистры процессора, после чего процесс продолжается с прерванного ранее места.

Система прерываний позволяет компьютеру реагировать на внешние события, синхронизировать выполнение процессов и работу устройств ввода-вывода, быстро переходить с одной программы на другую. Механизм прерываний нужен для того, чтобы оповестить процессор о возникновении в вычислительной системе некоторого непредсказуемого события или события, которое не синхронизировано с циклом работы процессора. Примерами таких событий могут служить завершение операции ввода-вывода внешним устройством (например, запись блока данных контроллером диска), некорректное завершение арифметической операции (например, переполнение

регистра), истечение интервала астрономического времени. При возникновении условий прерывания его источник (контроллер внешнего устройства, таймер, арифметический блок процессора и т. п.) выставляет определенный электрический сигнал. Этот сигнал прерывает выполнение процессором последовательности команд, задаваемой исполняемым кодом, и вызывает автоматический переход на заранее определенную процедуру, называемую процедурой обработки прерываний. В большинстве моделей процессоров отрабатываемый аппаратурой переход на процедуру обработки прерываний сопровождается заменой слова состояния машины (или даже всего контекста процесса), что позволяет одновременно с переходом по нужному адресу выполнить переход в привилегированный режим. После завершения обработки прерывания обычно происходит возврат к исполнению прерванного кода.

Прерывания играют важнейшую роль в работе любой операционной системы, являясь ее движущей силой. Действительно, большая часть действий ОС инициируется прерываниями различного типа. Даже системные вызовы от приложений выполняются на многих аппаратных платформах с помощью специальной инструкции прерывания, вызывающей переход к выполнению соответствующих процедур ядра (например, инструкция **`int`** в процессорах Intel или **`svc`** в мэйнфреймах IBM).

Системный таймер, часто реализуемый в виде быстродействующего регистра-счетчика, необходим операционной системе для выдержки интервалов времени. Для этого в регистр таймера программно загружается значение требуемого интервала в условных единицах, из которого затем автоматически с определенной частотой начинает вычитаться по единице. Частота «тиков» таймера, как правило, тесно связана с частотой тактового генератора процессора. (Не следует путать таймер ни с тактовым генератором, который вырабатывает сигналы, синхронизирующие все операции в компьютере, ни с системными часами — работающей на батареях электронной схеме, — которые ведут независимый отсчет времени и календарной даты.) При достижении нулевого значения счетчика таймер инициирует прерывание, которое обрабатывается процедурой операционной системы. Прерывания от системного таймера используются ОС в первую очередь для слежения за тем, как отдельные процессы расходуют время процессора. Например, в системе разделения времени при обработке очередного прерывания от таймера планировщик процессов может принудительно передать управление другому процессу, если данный процесс исчерпал выделенный ему квант времени.

Средства защиты областей памяти обеспечивают на аппаратном уровне проверку возможности программного кода осуществлять с данными определенной области памяти такие операции, как чтение, запись или выполнение (при передачах управления). Если аппаратура компьютера поддерживает механизм трансляции адресов, то средства защиты областей памяти встраиваются в этот механизм. Функции аппаратуры по защите памяти обычно состоят в сравнении уровней привилегий текущего кода процессора и сегмента памяти, к которому производится обращение.

1.3.8. Машинно-зависимые компоненты ОС

Одна и та же операционная система не может без каких-либо изменений устанавливаться на компьютерах, отличающихся типом процессора или/и способом организации всей аппаратуры. Однако опыт разработки операционных систем показывает: ядро можно спроектировать таким образом, что только часть модулей будут машинно-зависимыми, а остальные не будут зависеть от особенностей аппаратной платформы [14].

Объем машинно-зависимых компонентов операционной системы зависит от того, насколько велики отличия в аппаратных платформах, для которых разрабатывается операционная система. Одно из наиболее очевидных отличий — несовпадение системы команд процессоров — преодолевается достаточно просто. Операционная система программируется на языке высокого уровня, а затем соответствующим компилятором

вырабатывается код для конкретного типа процессора. Однако во многих случаях различия в организации аппаратуры компьютера лежат гораздо глубже и преодолеть их таким образом не удается. Например, однопроцессорный и двухпроцессорный компьютеры требуют применения в операционных системах совершенно разных алгоритмов распределения процессорного времени. Аналогично отсутствие аппаратной поддержки виртуальной памяти приводит к принципиальному различию в реализации подсистемы управления памятью. В таких случаях в код операционной системы придется вносить специфику аппаратной платформы, для которой эта операционная система разрабатывается.

Для уменьшения количества машинно-зависимых модулей производители операционных систем ограничивают распространение своей операционной системы только несколькими типами процессоров и созданными на их базе аппаратными платформами.

Для компьютеров на основе процессоров Intel x86/Pentium разработка экранирующего машинно-зависимого слоя операционной системы несколько упрощается за счет встроенной в постоянную память компьютера базовой системы ввода-вывода — BIOS. BIOS содержит драйверы для всех устройств, входящих в базовую конфигурацию компьютера: жестких дисков, клавиатуры, дисплея и т.д. Эти драйверы выполняют примитивные операции по управлению устройствами компьютера, но за счет этих операций экранируются различия аппаратных платформ, созданных на процессорах компании Intel или совместимых с ними процессорах разных производителей. Разработчики операционной системы могут пользоваться слоем драйверов BIOS как частью машинно-зависимого слоя операционной системы, а могут и заменить все или часть драйверов BIOS компонентами операционной системы.

Новые компьютеры используют прошивку UEFI вместо традиционного BIOS. Обе эти программы – примеры ПО низкого уровня, запускающегося при старте компьютера перед тем, как загрузится операционная система. UEFI – более новое решение, он поддерживает жёсткие диски большего объёма, быстрее грузится, более безопасен – и, что очень удобно, обладает графическим интерфейсом и поддерживает мышь.

Некоторые новые компьютеры, поставляемые с UEFI, по-прежнему называют его «BIOS», чтобы не запутать пользователя, привычного к традиционным PC BIOS. Но, даже встретив его упоминание, знайте, что ваш новый компьютер, скорее всего, будет оснащён UEFI, а не BIOS.

Об отличиях BIOS и UEFI можно почитать по ссылке https://habr.com/ru/articles/404511/.

Современные версии Windows используют HAL (Hardware Abstraction Layer) — слой абстрагирования, реализованный в программном обеспечении, находящийся между физическим уровнем аппаратного обеспечения и программным обеспечением, запускаемом на этом компьютере. HAL предназначен для скрытия различий в аппаратном обеспечении от основной части ядра операционной системы, таким образом, чтобы большая часть кода, работающая в режиме ядра, не нуждалась в изменении при её запуске на системах с различным аппаратным обеспечением.

В операционных системах семейства Windows NT HAL является неотъемлемой частью кода, исполняемого в режиме ядра, находится в отдельном загрузочном модуле, загружаемом совместно с ядром. Это обеспечивает возможность использования одного и того же загрузочного модуля собственно ядра ОС Windows NT на ряде систем с различными архитектурами шин ввода-вывода, управления прерываниями и таймерами.

1.3.9. Утилиты

Вспомогательное (или служебное) программное обеспечение для краткости принято называть утилитами.

Утилиты — это системное программное обеспечение, которое помогает поддерживать правильное и бесперебойное функционирование компьютерной системы. Они помогают операционной системе управлять, организовывать, поддерживать и оптимизировать функционирование компьютерной системы.

Утилиты выполняют определенные задачи, такие как обнаружение, установка и удаление вирусов, резервное копирование данных, удаление ненужных файлов и т. д. Некоторыми примерами являются антивирусное программное обеспечение, инструменты управления файлами, инструменты сжатия, инструменты управления дисками и т. д. [15].

Утилиты предоставляют доступ к возможностям (параметрам, настройкам, установкам), недоступным без их применения, либо делают процесс изменения некоторых параметров проще (автоматизируют его) [16].

Утилиты могут входить в состав операционных систем, идти в комплекте со специализированным оборудованием или распространяться отдельно.

Интегрированные пакеты утилит — набор нескольких программных продуктов объединенных в единый удобный инструмент.

Типы утилит

1. Антивирусы

Вирус — это вредоносное программное обеспечение, которое попадает в систему вместе с хост-программой. Более того, со временем он размножается и создает несколько копий, что, в свою очередь, замедляет и повреждает систему.

Антивирус — это служебное программное обеспечение, которое помогает защитить компьютер от вирусов. Более того, он уведомляет об обнаружении любого вредоносного файла и удаляет такие файлы. Кроме того, он сканирует любое новое устройство, подключенное к компьютеру, и удаляет любые вирусы, если они есть. Более того, он также время от времени сканирует систему на наличие угроз и уничтожает их. Примерами антивирусов являются McAfee Antivirus, Quickheal Antivirus, Windows Defender и т. д.

2. Системы управления файлами

Эти служебные программы используются для управления файлами компьютерной системы. Поскольку файлы являются важной частью системы, поскольку все данные хранятся в файлах. Таким образом, это служебное программное обеспечение помогает просматривать, искать, упорядочивать, находить информацию и быстро просматривать файлы системы.

Проводник Windows — это инструмент управления файлами по умолчанию, присутствующий в системе. Другими примерами инструментов управления файлами являются Google Desktop, Double Commander, Directory Opus и т. д.

3. Инструменты сжатия

Важной частью компьютера является место для хранения данных, очень важно поддерживать это хранилище. Поэтому мы используем определенные служебные программы для сжатия больших файлов и уменьшения их размера. Это инструменты сжатия. Формат файлов изменяется во время сжатия, и мы не можем получить к ним доступ или редактировать их напрямую. Кроме того, мы можем легко распаковать файл и вернуть исходный файл. Примерами инструментов сжатия являются WinZip, WinRAR, WinAce, PeaZip, 7-Zip и т. д.

4. Инструменты управления дисками

Эти служебные программы используются для управления данными на дисках. Более того, они выполняют такие функции, как разбиение устройств на разделы, управление дисками и т. д. Примерами инструментов управления дисками являются MiniTool Partition Wizard, Paragon Partition Manager и т. д.

5. Инструменты очистки диска

Эта служебная программа помогает освободить дисковое пространство. Кроме того, с диска удаляются файлы, которые больше не используются. Примеры: Razer Cortex, Piriform CCleaner и т. д.

6. Дефрагментация диска

Эта служебная программа помогает уменьшить фрагментацию и, следовательно, снижает скорость доступа. Дефрагментация означает переупорядочение файлов и сохранение их в смежных местах памяти. Более того, экономится время при чтении файлов и записи файлов на диск. Примерами дефрагментаторов дисков являются Perfect disk, Deflaggler и т. д.

7. Утилита резервного копирования

Эта служебная программа помогает создавать резервные копии файлов, папок, баз данных или целых дисков. Более того, под резервным копированием подразумевается дублирование информации на диске, чтобы данные можно было восстановить в случае потери данных.

1.3.10. Системные обрабатывающие программы

Обрабатывающие системные программы отличаются от управляющих программ как по своим функциям, так и по способу их инициирования (запуска) [17]. Основные функции обрабатывающих программ:

1) **перенос информации**. Перенос может выполняться между различными устройствами или в пределах одного устройства. При этом под устройствами понимаются: ОП, устройства ВП, устройства ввода-вывода;

2) **преобразование информации**. То есть после считывания информации с устройства обрабатывающая программа преобразует эту информацию, а только затем записывает ее на это же или на другое устройство.

В зависимости от того, какая из этих двух функций является основной, обрабатывающие системные программы делятся на **утилиты** и **лингвистические процессоры**. Основной функцией утилиты является перенос информации, а основная функция лингвистического процессора – перевод описания алгоритма с одного языка на другой. Сущность алгоритма при этом сохраняется, но форма его представления, ориентированная на программиста, преобразуется в форму, ориентированную на ЦП.

Утилиты были рассмотрены ранее в пункте 1.3.9.

Лингвистические процессоры делятся на **трансляторы** и **интерпретаторы**.

В результате работы **транслятора** алгоритм, записанный на языке программирования (исходная виртуальная программа), преобразуется в алгоритм, записанный на машинном языке. (На самом деле, машинная программа является результатом совместной работы нескольких лингвистических процессоров.)

Трансляторы делятся на компиляторы и ассемблеры. Исходная программа для транслятора–ассемблера записывается на языке-ассемблере. Один оператор данного языка транслируется в одну машинную команду. Исходная программа для компилятора записывается на языке программирования высокого уровня. Каждый оператор такого языка транслируется в несколько машинных команд. Примерами языков высокого уровня являются Паскаль и Си.

Интерпретатор в отличие от транслятора не выдаёт машинную программу целиком. Выполнив перевод очередного оператора исходной программы в соответствующую совокупность машинных команд, интерпретатор обеспечивает их выполнение. Затем преобразуется тот исходный оператор, который должен выполняться следующим по логике алгоритма, и так далее.

Примером интерпретатора является интерпретатор командного языка (ИК). Другие названия этого модуля: командный процессор, командная оболочка. ИК представляет собой лишь "видимую", сравнительно небольшую часть операционной системы. На самом деле, чтобы интерпретировать, т.е. выполнить (преобразовав в совокупность машинных команд), очередную команду пользователя, ИК инициирует многие другие модули ОС. Одни из них ищут текст требуемой прикладной программы на диске, другие выделяют необходимые аппаратные ресурсы, третьи загружают программу в ОП и инициируют её. Попутно заметим, что, так как одна команда пользователя задаёт выполнение целой программы (прикладной, утилиты или системы программирования), то язык управления ОС может рассматриваться как язык программирования очень высокого уровня.

Для Windows это командный процессор **`cmd.exe`** и **`Powershell`**.

Для Linux – **`sh`**, **`bash`**, **`zsh`** и другие **`*sh.`**

Кроссплатформенный PowerShell — **`pwsh.`**

Современная реализация командного процессора CMD кроме классического применения, позволяет также использовать [18]:

- возможности Инструментария управления Windows (WMI –Windows Management Instrumentation),
- сценарии Windows Script Host (WSH),
- оболочки (shell) операционных систем Linux.

Последнее возможно без установки стороннего программного обеспечения, поскольку в современных Windows, начиная с Windows 10, в качестве стандартного компонента системы может применяться Подсистема Windows для Linux (WSL – Windows Subsystem for Linux), что позволяет объединять в командных файлах исходные тексты CMD и, например, - bash .

Командный файл – это обычный текстовый файл с набором команд, которые последовательно выполняются командным процессором CMD Windows (cmd.exe). Командный процессор последовательно считывает и выполняет команды, которые представляют собой своеобразную программу, реализующую определенный алгоритм. Естественно, как и в любой другой среде программирования, программирование в командной строке подчиняется определенным правилам и командные файлы должны им полностью соответствовать.

Наиболее часто используемые системные обрабатывающие программы — текстовые и графические редакторы, компиляторы, компоновщики, отладчики.

1.3.11. Библиотеки процедур

Библиотеки процедур и функций различного назначения включены в категорию вспомогательных модулей операционной системы. К ним можно отнести библиотеки математических функций, библиотеки функций ввода-вывода и т.д.

Библиотеки — это набор функций, которые могут использоваться в различных программах. Библиотеки могут быть статические (библиотека привязывается к определенной программе или софт содержит данную библиотеку в своем теле.) и динамическими (библиотеки грузятся в оперативную память и используются).

В операционных системах Windows системные библиотеки хранятся в папках **System32** и **SysWOW64**. Первая папка для 64-хразрядных библиотек, вторая для 32-хразрядных (а не наоборот).

В операционных системах на основе ядра Linux по умолчанию библиотеки находятся в двух папках. Это корневая папка **/lib** (в ней находятся библиотеки, которые используют программы, расположенные в корневой папке **/bin**) и вторая папка **/usr/lib** (в ней находятся библиотеки, которые используют программы расположенные **/usr/bin**). Пути к библиотекам указаны в файле **/etc/ld.so.conf**.

1.3.12. Программы дополнительных услуг

В эту категорию входит большое количество разнообразных программ: специальный вариант пользовательского интерфейса, калькулятор, некоторые игры (какие, например, поставляются в составе ОС).

Список использованных источников

1. Объясните понятие ресурса. Почему понятие ресурса является одним из фундаментальных при рассмотрении ОС? Какие виды и типы ресурсов вы знаете? [Электронный ресурс]. – Режим доступа. – https://msbro.ru/index.php/archives/392

2. Ресурсы. Свойства ресурсов. Классификация ресурсов [Электронный ресурс]. – Режим доступа. – https://studfile.net/preview/7580687/page:3/

3. АРХИТЕКТУРА (СТРУКТУРА) ОПЕРАЦИОННОЙ СИСТЕМЫ [Электронный ресурс]. – Режим доступа. – http://mf.grsu.by/UchProc/livak/po/lections/lec_arhit.htm

4. Файловая подсистема [Электронный ресурс]. – Режим доступа. – https://it.wikireading.ru/6446

5. Лав, Р. Разработка ядра Linux / Р. Лав. – М.: ООО "И.Д. Вильямс", 2006. – 448 с.

6. Ядро операционной системы [Электронный ресурс]. – Режим доступа. – https://best-exam.ru/yadro_operacionnoy_systemi

7. Ядро операционной системы [Электронный ресурс]. – Режим доступа. – https://spravochnick.ru/informacionnye_tehnologii/yadro_operacionnoy_sistemy/

8. Таненбаум, Э. Современные операционные системы. / Э. Таненбаум, Х. Бос. — 4-е изд. — СПб.: Питер, 2015. — 1120 с.

9. Объекты ядра. (часть 1) [Электронный ресурс]. – Режим доступа. – https://club.shelek.ru/viewart.php?id=78

10. Академия Microsoft: Основы организации операционных систем Microsoft Windows [Электронный ресурс]. – Режим доступа. – https://intuit.ru/studies/courses/1089/217/lecture/5591

11. Управление объектами ядра [Электронный ресурс]. – Режим доступа. –
https://learn.microsoft.com/ru-ru/windows-hardware/drivers/kernel/managing-kernel-objects

12. Основные виды ресурсов операционной системы. [Электронный ресурс]. – Режим доступа. –
https://cyberpedia.su/12x55b9.html

13. Многослойная структура операционной системы [Электронный ресурс]. – Режим доступа. –
https://studref.com/389074/informatika/mnogosloynaya_struktura_operatsionnoy_sistemy

14. Машинно зависимые компоненты операционной системы [Электронный ресурс]. – Режим доступа. –
https://studref.com/389076/informatika/mashinno_zavisimye_komponenty_operatsionnoy_sistemy

15 Utility Software [Электронный ресурс]. – Режим доступа. –
https://www.toppr.com/guides/computer-science/computer-fundamentals/software-concepts/utility-software/

16. Леонтьев В. П. Самые полезные программы: утилиты. — ОЛМА-ПРЕСС Образование, 2004. — 48 с. — (Компьютер. Карманный справочник пользователя).

17. ЧАСТЬ 2. АССЕМБЛЕРНЫЕ ПРОГРАММЫ В СРЕДЕ DOS [Электронный ресурс]. – Режим доступа. –
https://asm.kcup.tusur.ru/Library/chapter%202/9-1.html

18. Командные файлы Windows [Электронный ресурс]. – Режим доступа. –
https://ab57.ru/cmd.html