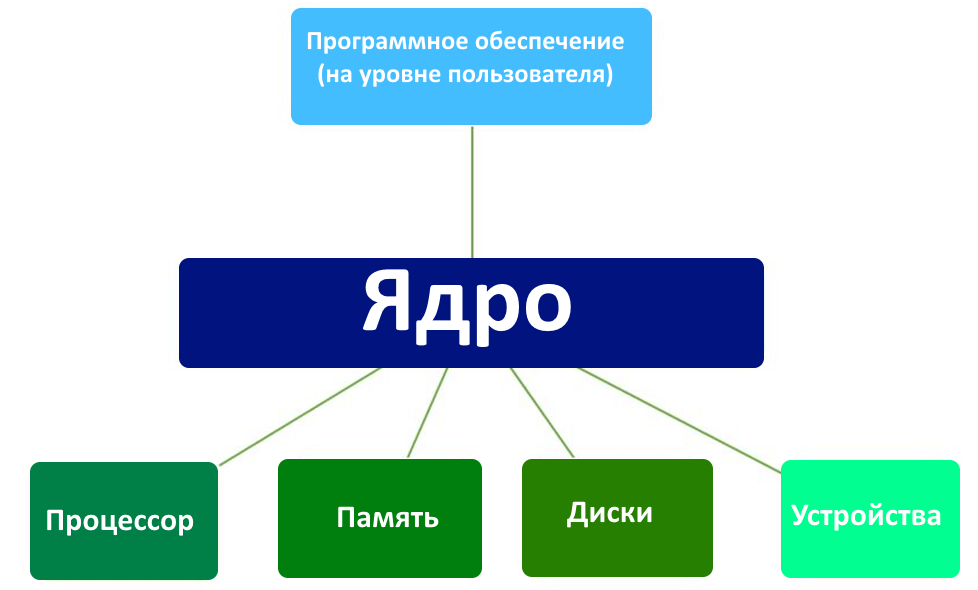
**Ядро. Объекты ядра**

**Ядро операционной системы** – это центральная и наиболее важная часть ОС, которая обеспечивает взаимодействие между аппаратным обеспечением компьютера и прикладными программами. Его главная функция – управление всеми ресурсами компьютера, такими как процессорное время, память и устройства ввода-вывода, а также их координированный доступ для приложений и процессов.



**Основные функции ядра:**

1.Управление ресурсами: Ядро распределяет и управляет такими ресурсами, как процессорное время, оперативная память и доступ к устройствам, между различными программами и процессами, работающими одновременно.

2.Взаимодействие с оборудованием: Ядро выступает в роли "связующего звена", предоставляя приложениям стандартизированный интерфейс для взаимодействия с аппаратным обеспечением компьютера, например, с жестким диском или сетевой картой.

3.Управление процессами: Ядро отвечает за запуск, завершение и координацию работы процессов (исполняющихся программ), чтобы обеспечить их совместную работу.

4.Обеспечение безопасности: В ядро входят модули безопасности, которые контролируют доступ к системным ресурсам и защищают компьютер от несанкционированного доступа или вредоносных программ.

5.Системные вызовы: Ядро предоставляет набор системных вызовов, через которые программы могут запрашивать у операционной системы выполнить те или иные действия, например, открыть файл или создать новый процесс.



Типы ядер:

Существуют разные архитектуры ядер, среди которых наиболее распространены:

**Монолитное ядро:**

Все ключевые функции (управление памятью, процессами, устройствами) работают как единая программа в одном адресном пространстве, обеспечивая высокую скорость.

**Микроядро:**

В ядре реализован только минимальный набор критически важных функций, а остальные сервисы работают как отдельные приложения в пространстве пользователя, что повышает безопасность и надежность.

**Гибридное ядро:**

Комбинирует элементы монолитного и микроядерного подходов.

**Объекты ядра**

**Объект ядра** — это блок памяти, выделенный ядром операционной системы и доступный только ему.

Этот блок представляет собой структуру данных, в элементах которой содержится информация о некоем объекте. Структурированные блоки данных используются операционной системой для управления компьютерными ресурсами, такими как процессы, потоки, файлы и устройства и тд.

**Примеры объектов ядра:**

**Объекты файлов:** представляют открытые файлы и другую файловую информацию, позволяя системе управлять доступом к ним.

**Объекты событий:** используются для синхронизации между различными частями системы, позволяя им ждать наступления определенного события.

**Объекты потоков:** описывают отдельные пути выполнения внутри процесса, обеспечивая управление их приоритетом и состоянием.

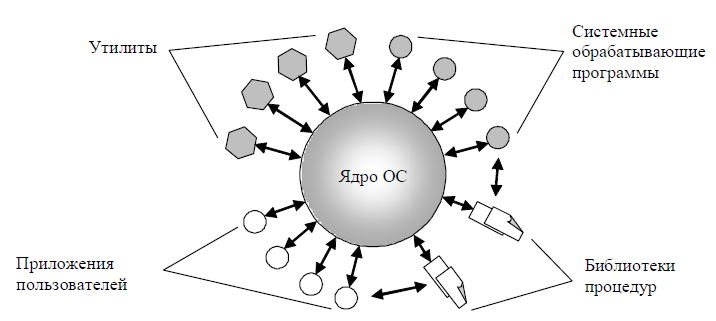
**Объекты процессов:** представляют собой активные программы, запущенные на компьютере.

**Функции объектов ядра:**

* **Управление ресурсами:** объекты ядра предоставляют ядру средства для выделения, управления и освобождения различных системных ресурсов.
* **Абстракция:** они позволяют системе работать на более высоком уровне абстракции, скрывая детали аппаратного и программного обеспечения.
* **Безопасность:** ядро использует объекты для обеспечения безопасного и контролируемого доступа к ресурсам, не позволяя приложениям напрямую взаимодействовать с аппаратурой.

Приложение не может напрямую обращаться к объектам ядра читать и изменять их содержимое.

Объекты ядра используются системой приложениями для управления множеством разных ресурсов: процессами, потоками, файлами и т.д.



Система позволяет создавать и оперировать несколькими типами таких объектов, в том числе:

* маркерами доступа (access token objects),
* файлами (file objects),
* проекциями файлов (file-mapping objects),
* портами завершения ввода вывода (I/O completion port objects),
* заданиями (jobs),
* почтовыми ящиками (mailslot objects),
* мьютексами (mutex objects),
* каналами (pipe objects),
* процессами (thread objects)
* ожидаемыми таймерами (waitable timer objects)

Эти объекты создаются Windows-функциями

Каждый объект ядра описывается своим набором *дескрипторов*. Некоторые элементы (дескриптор защиты, счётчик числа пользователей и др.) присутствуют во всех объектах, но большая их часть специфична для объектов конкретного типа.

**Дескрипторы**

**Дескриптор объекта ядра** («описатель», «handle») – это числовой идентификатор или структура данных, который ядро операционной системы использует для представления и управления различными объектами, такими как файлы, процессы, устройства или память. Иными словами, это некое значение, уникально идентифицирующее объект ядра в пределах данного процесса.

Ядро назначает уникальный дескриптор каждому объекту, когда тот создается, что позволяет процессу обращаться к нему по этому "номеру".

Дескриптор может быть чем угодно: от целочисленного индекса до указателя на ресурс в пространстве ядра. Идея в том, что они обеспечивают абстракцию ресурса, поэтому для его использования вам не нужно много знать о самом ресурсе. Таким образом дескриптор предоставляет процессу уровень абстракции от конкретных деталей реализации объекта, скрывая сложную внутреннюю структуру и предоставляя простой интерфейс для работы.

Например, HWND в Win32 API — это дескриптор окна. Сам по себе он бесполезен: из него нельзя извлечь никакой информации. Но если передать его нужным функциям API, с его помощью можно будет проделать множество различных действий. Внутренне HWND можно рассматривать как индекс в таблице окон графического интерфейса (возможно, это не совсем так реализовано, но это объясняет магию).

Дескриптор может быть полезен для сохранения состояний (среди прочего). Если у вас есть данные в такой структуре, как std::vector , то объект может находиться в разных местах памяти в разное время во время выполнения программы, а это значит, что указатель на эту память будет менять значения. Дескриптор никогда не меняется, он всегда ссылается на тот объект.

Дескрипторы представлены непрозрачным типом данных HANDLE.

В зависимости от операционной системы, существует большое количество типов дескрипторов:

* Дескрипторы процессов: идентифицируют запущенные программы.
* Дескрипторы памяти: определяют области памяти.
* Дескрипторы окон (в Windows): служат для идентификации графических элементов пользовательского интерфейса
* И тд.

**Дескрипторные таблицы** - это массивы памяти переменной длины, содержащие дескрипторы. Поскольку эти таблицы обслуживает операционная система, то команды загрузки таблицы дескрипторов являются привилегированными командами.

Существуют две обязательных дескрипторных таблицы - глобальная дескрипторная таблица (Global Descriptor Table - GDT) и дескрипторная таблица прерывания (Interrupt Descriptor Table - IDT),- а также множество (до 8191) локальных дескрипторных таблиц (Local Descriptor Table - LDT), из которых в один момент времени процессору доступна только одна.

**Диспетчер объектов Windows**

Диспетчер объектов Windows (Object Manager) управляет объектами, которые являются частью операционной системы в режиме ядра. Соответственно, *объект* — это коллекция данных, которым управляет операционная система.

Создание и уничтожение этих объектов управляются Диспетчером объектов, который также поддерживает базу данных объектов и следит за их использованием, обеспечивая координацию доступа приложений к ресурсам компьютера.

Для взаимодействия с объектами ядра у Windows предусмотрен набор функций, обрабатывающих структуры объектов ядра по строго определенным правилам. Когда мы создаем объект ядра, функция возвращает описатель идентифицирующий созданный объект (HANDLE). Все операции с текущим объектом ядра возможны только при указании этого описателя управляющей функции.

 Доступ к общесистемным структурам данных и пулам памяти, используемым системными компонентами режима ядра, возможен лишь из режима ядра — у потоков пользовательского режима нет доступа к соответствующим страницам.

 У каждого процесса имеется индивидуальное закрытое адресное пространство, защищенное от доступа потоков других процессов.

 Все процессоры, поддерживаемые Windows, предоставляют ту или иную форму аппаратной защиты памяти.

 Совместно используемые объекты имеют стандартные для Windows списки контроля доступа, проверяемые при попытках процессов открыть эти объекты.

**Создание объектов ядра**

Создание объектов ядра происходит с помощью специальных функций. Функция, которая создаёт объект ядра, возвращает дескриптор, который идентифицирует объект ядра. Передавая этот дескриптор различным функциям Windows, система точно знает, с каким объектом ядра необходимо взаимодействовать.

**Процесс создания объекта:**

1.Когда процесс инициализируется, таблица дескрипторов пуста.

2. Но стоит одному из его потоков вызвать функцию, создающую объект ядра (например, *CreateFileMapping),* как ядро выделяет для этого объекта блок памяти и инициализирует его

3. далее ядро просматривает таблицу дескрипторов, принадлежащую данному процессу, и отыскивает свободную строку и инициализирует ее ссылкой на созданный системный объект.

4. В указатель записывается адрес структуры данных объекта ядра (в области ОС), маска доступа — на доступ без ограничений, и, наконец, определяется последний компонент —флаги. Приложению возвращается дескриптор (значение типа HANDLE) который ссылается на эту строку.

Если вызов функции, создающей объект ядра, неудачен, то обычно возвращается 0(NULL). Такая ситуация возможна только при острой нехватке памяти или при наличии проблем с защитой.

Отдельные функции возвращают в таких случаях не 0, а -1 (INVALID\_HANDLE\_VAULE – недействительное значение дескриптора). Например, если *CreateFile*не сможет открыть указанный файл, она вернет именно INVALID\_HANDLE\_VALUE.Так что следует быть осторожны при проверке значения, возвращаемого функцией.

Для создания объекта ядра необходимо использовать одну из специальных функций (специфичных для каждого типа объекта).

Создание объекта файл

HANDLE CreateFile(

LPCTSTR lpFileName,// имя файла

DWORD dwDesiredAccess,// права доступа к файлу

DWORD dwShareMode,// режим совместного использования

LPSECURITY\_ATTRIBUTES lpSecurityAttributes,// права доступа к объекту ядра

DWORD dwCreationDisposition,// режим создания

DWORD dwFlagsAndAttributes,// атрибуты файла

HANDLE hTemplateFile// дескриптор шаблона

);

LPCTSTR *Long + Pointer + Constant + TCHAR (тип символа, который может быть либо 8-битнымANSI, либо 16-битным Unicode) + STR* – это псевдоним для указателя на константную строку (LPCTSTR), которая может быть представлена как обычными 8-битными символами ANSI, так и широкими символами Unicode, в зависимости от того, определен ли макрос UNICODE во время компиляции. Этот тип данных широко используется в Windows API, чтобы обеспечить совместимость с разными кодировками строк.

Создание семафора

Семафор - это примитив синхронизации, который контролирует доступ к ограниченному набору ресурсов, позволяя не более чем заданному числу потоков одновременно выполнять операцию. Семафор создается с начальным значением, указывающим максимальное количество потоков, которые могут одновременно получить доступ к ресурсу.

HANDLE CreateSemaphore(

LPSECURITY\_ATTRIBUTES lpSemaphoreAttributes,// права доступа к объекту ядра

LONG lInitialCount// начальное значение

LONG lMaximumCount// максимальное значение

LPCTSTR lpName// имя объекта

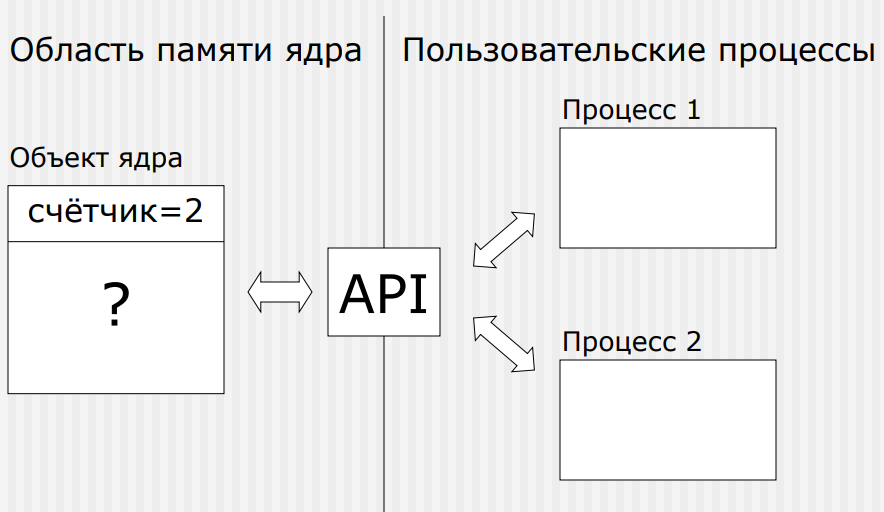
);

**Уничтожение объектов ядра**

Уничтожение объектов ядра происходит следующим образом: когда объект ядра создаётся впервые, его счётчик использования устанавливается на 1. Затем, когда другой процесс получает доступ к существующему объекту ядра, счётчик использования увеличивается на единицу. Когда процесс завершается, ядро ОС автоматически уменьшает счётчик использования для всех объектов ядра на единицу. Если счётчик использования объекта становится равным 0, ядро ОС уничтожает объект

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.

Ядру известно, сколько процессов использует конкретный объект ядра, поскольку в каждом объекте есть счетчик числа его пользователей. В момент создания объекта счетчику присваивается единица. Когда к существующему объекту ядра обращается другой процесс, счетчик увеличивается на единицу. А когда процесс завершается, счетчики всех, еще используемых им объектом ядра автоматически уменьшаются на единицу. Как только счетчик какого-либо объекта обнуляется, ядро уничтожает этот объект.

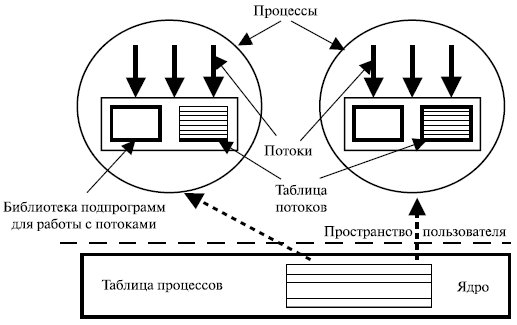


По окончанию работы с объектом ядра его нужно закрыть вызовом ***CloseHandle.***

**/kernel (создание двоичного файла режима ядра)**

Создает двоичный файл, который можно выполнить в ядре Windows. Код в текущем проекте компилируется и связывается с помощью упрощенного набора функций языка C++, относящихся к коду, который выполняется в режиме ядра.

**Процесс выполнения программ: создание, завершение процессов и потоков**



**Процесс** – это выполняющаяся программа с её ресурсами (память, файлы). Создание процесса включает выделение операционной системой ресурсов и присвоение уникального идентификатора, а завершение – их освобождение.

* Создание: При запуске программы операционная система выделяет ей пространство памяти, системные ресурсы и присваивает уникальный идентификатор (ID процесса).
* Выполнение: Процесс – это динамический объект, представляющий собой активную программу, которая получает процессорное время.
* Завершение: По окончании работы или при возникновении ошибки процесс завершается, а операционная система освобождает все связанные с ним ресурсы.

**Поток** – это наименьшая единица выполнения кода внутри процесса, позволяющая параллельно выполнять несколько задач. Потоки создаются и уничтожаются в процессе выполнения кода и могут параллельно работать в одном процессе, используя его ресурсы - потоки в одном процессе разделяют его ресурсы, такие как адресное пространство и открытые файлы, но имеют свои собственные стек и регистры

**Взаимосвязь процессов и потоков**

* Процесс служит контейнером для одного или нескольких потоков.
* Потоки являются единицами выполнения, которые планируются операционной системой для выполнения на процессоре.
* Многопоточность внутри процесса позволяет повысить производительность и отзывчивость приложений, выполняя несколько операций одновременно.

Три основных события, приводящие к созданию процессов (вызов **fork** или **CreateProcess**):

* Загрузка системы
* Работающий процесс подает системный вызов на создание процесса
* Запрос пользователя на создание процесса

Четыре события, приводящие к остановке процесса (вызов **exit** или **ExitProcess**):

* Плановое завершение (окончание выполнения)
* Плановый выход по известной ошибке (например, отсутствие файла)
* Выход по неисправимой ошибке (ошибка в программе)
* Уничтожение другим процессом

Три состояния процесса:

* Выполнение (занимает процессор)
* Готовность (процесс временно приостановлен, чтобы позволить выполняться другому процессу)
* Ожидание (процесс не может быть запущен по своим внутренним причинам, например, ожидая операции ввода/вывода)

При создании процесса, он переходит в готовность.Из готовности он может перейти в выполнение(3). С выполнения он может сменить состояние на ожидание(1) или готовность (2) Из ожидания он может перейти только в готовность(4).

Возможные переходы между состояниями:

1. Процесс блокируется, ожидая входных данных

2. Планировщик выбирает другой процесс

3. Планировщик выбирает этот процесс

4. Поступили входные данные

**Синхронизация:**

Синхронизация процессов осуществляется через взаимодействие и согласование работы между ними, чтобы избежать ошибок и обеспечить корректное выполнение задач в многозадачных системах. Для этого используются различные механизмы синхронизации, например, операционной системой, через ожидание одними процессами событий, связанных с работой других процессов, или через обмен данными.

Типы отношений между процессами при синхронизации:

* Старт-старт: Оба процесса начинают работу одновременно или в определенной последовательности.
* Финиш-старт: Один процесс завершает свою работу, после чего начинается работа другого процесса.
* Старт-финиш: Один процесс начинает свою работу, а другой процесс завершает свою.
* Финиш-финиш: Оба процесса завершают свою работу одновременно.

На основе синхронизации процессы подразделяются на два типа: Независимый процесс: выполнение одного процесса не влияет на выполнение других процессов. Кооперативный процесс : процесс, который может влиять на другие процессы, выполняемые в системе, или подвергаться их влиянию.

Сложность проблемы синхронизации состоит в нерегулярности возникающих ситуаций при взаимодействии процессов. Пренебрежение вопросами синхронизации может привести к неправильной работе процессов или даже к краху системы.

**Задача об обедающих философах** — классический пример, используемый в информатике для иллюстрации проблем синхронизации при разработке параллельных алгоритмов и техник решения этих проблем. Задача сформулирована таким образом, чтобы иллюстрировать проблему избежания взаимной блокировки (англ. deadlock) — состояния системы, при котором прогресс невозможен.



Пять философов сидят вокруг круглого стола, перед каждым философом стоит тарелка еды. На столе между каждой парой ближайших философов лежит по одной вилке.

Каждый философ может либо есть, либо размышлять. Приём пищи не ограничен количеством еды – у них бесконечный запас. Тем не менее, философ может есть только тогда, когда держит две вилки — взятую справа и слева. Закончив еду, философ кладет вилки слева и справа от своей тарелки и опять начинает размышлять до тех пор, пока снова не проголодается.

Каждый философ может взять ближайшую вилку (если она доступна) или положить — если он уже держит её. Взятие каждой вилки и возвращение её на стол являются раздельными действиями, которые должны выполняться одно за другим.

Вопрос задачи заключается в том, чтобы разработать модель поведения (параллельный алгоритм), при котором ни один из философов не будет голодать, то есть будет вечно чередовать приём пищи и размышления.

Решение задачи

* **«Официант»** [**Семафоры**](https://www.google.com/search?cs=0&sca_esv=7550878c098e0420&q=%D0%A1%D0%B5%D0%BC%D0%B0%D1%84%D0%BE%D1%80%D1%8B&sa=X&ved=2ahUKEwjjltHL-r-PAxWHEhAIHQ8FN5sQxccNegQIMhAB&mstk=AUtExfA_BgR4l9movMMiHO9okGdTXJlja2ntUFRRo45UXZlGYA63VNwaYTfL5aNo-pTYq76o8XwkMwpET22TqN7YneBZ-qkxehGJK5d5gPWdAqgYRwN7ToJ9XWwKHs4KcmK-RaA&csui=3)

Каждую вилку можно представить как семафор. Философ запрашивает доступ к семафору (берет вилку). Семафоры обеспечивают взаимоисключение, гарантируя, что только один философ может держать вилку в каждый момент времени.

* **Ограничение числа философов:**

Можно разрешить одновременно за столом находиться только n-1 философам. Последний философ должен ждать, прежде чем сесть и начать попытки взять вилки.

* **Иерархия ресурсов:**

Каждой вилке присваивается номер, и философ всегда сначала берет вилку с меньшим номером, а затем с большим. Это гарантирует, что не будет циклических зависимостей, и тупики исключены.