ОГЛАВЛЕНИЕ

[Объекты (внутренняя организация) 2](#_Toc95479356)

[Объекты исполнительной системы 2](#_Toc95479357)

[Файловая модель 4](#_Toc95479358)

[Объектная модель 4](#_Toc95479359)

[Структура объектов 5](#_Toc95479360)

[Типовой объект 7](#_Toc95479361)

[Управление объектами 8](#_Toc95479362)

[Имена объектов 8](#_Toc95479363)

[Описатели объектов 10](#_Toc95479364)

[Удержание объектов 11](#_Toc95479365)

[Учет использования ресурсов 12](#_Toc95479366)

[Защита объектов 12](#_Toc95479367)

[Маркеры доступа 13](#_Toc95479368)

[Списки контроля доступа 15](#_Toc95479369)

[Как все это работает вместе 16](#_Toc95479370)

[лабораторная работа 3 Объекты (программирование) 18](#_Toc95479371)

[Теория 18](#_Toc95479372)

[Что такое объект 18](#_Toc95479373)

[Таблица описателей объектов 19](#_Toc95479374)

[Создание объекта 19](#_Toc95479375)

[Закрытие объекта 20](#_Toc95479376)

[Учет объектов 21](#_Toc95479377)

[Защита объектов 21](#_Toc95479378)

[Работа с объектами 23](#_Toc95479379)

[Объект WaitableTimer 25](#_Toc95479380)

[Чему нужно научиться 27](#_Toc95479381)

[Задания 27](#_Toc95479382)

[Уровень 1 (A) 27](#_Toc95479383)

[Уровень 2 (A) 27](#_Toc95479384)

[Уровень 3 (A) 27](#_Toc95479385)

[Скелет кода фоновой программы 28](#_Toc95479386)

СПИСОК РИСУНКОВ и ТАБЛИЦ

[Рис. 1.Содержимое заголовка объекта 6](#_Toc95479387)

[Рис. 2 Иерархия имен объектов 9](#_Toc95479388)

[Рис. 3 Объект каталог объектов 10](#_Toc95479389)

[Рис. 4 Структура таблицы объектов 11](#_Toc95479390)

[Рис. 5 Пример маркера доступа 14](#_Toc95479391)

[Рис. 6 Объект маркер доступа 15](#_Toc95479392)

[Рис. 7 Список контроля доступа 16](#_Toc95479393)

[Рис. 8Проверка прав доступа 17](#_Toc95479394)

[Рис. 9 Счетчики описателей и ссылок 24](#_Toc95479395)

[Рис. 10 Описатели и таблица описателей 25](#_Toc95479396)

[Таблица 1 Объекты исполнительной системы 3](#_Toc95479397)

[Таблица 2 Структура таблицы описателей объектов процесса 19](#_Toc95479398)

# Объекты (внутренняя организация)

В конце 80-х объекты стали рекламироваться в качестве панацеи от всех проблем в программировании. Однако впервые они появились в конце 60-х в языках программирования, предназначенных для моделирования. Объектно-ориентированное программирование, которое обеспечивает способ представления и манипулирования как физическими, так и абстрактными объектами, является естественным подходом в этой области.

Операционные системы также работают с объектами. Их объектами являются аппаратные ресурсы (устройства ввода-вывода, память ) или программные ресурсы ( файлы, процессы, семафоры ). При этом можно работать с каждым типом ресурсов по-своему, а можно сосредоточить управление ресурсами в одном месте, используя их сходство.

Рассмотрим типы объектов *исполнительной системы NT*, структуру объектов и то, как ими управляет диспетчер объектов, а также защиту объектов.

## Объекты исполнительной системы

В *исполнительной системе NT* объект – это отдельный экземпляр статически определенного типа объектов, существующий во время выполнения. *Тип объекта* (*object type),* также называемый *классом объектов (object class),* включает набор атрибутов объекта и сервисы, работающие с образцами этого типа. *Атрибут объекта* *(object attribute)* – это поле данных внутри объекта, определяющее его состояние. *Объектные сервисы (object services)* – это способы манипулирования объектами, с их помощью считывают или изменяют атрибуты объекта. Главное различие между объектом и структурой данных состоит в том, что внутренняя структура объекта скрыта и для извлечения данных из объекта или помещения в него информации необходимо использовать объектные сервисы. Вы не можете непосредственно считывать или изменять внутренние данные объекта. Таким образом, внутренняя реализация объекта отделяется от кода, который лишь использует данный объект.

При проектировании исполнительной системы Windows NT было принято решение использовать объекты дла представления системных ресурсов, потомучто они позволяют централизованно выполнить три важные задачи ОС:

* Присвоение системным ресурсам имен;
* Совместное использование системных ресурсов различными процессами;
* Защита системных ресурсов от несанкционированного доступа.

Не все структуры данных в исполнительной системе являются объектами, а только те, которые должны быть совместно используемыми, защищенными, именованными (видимыми) программам пользовательского режима. Структуры, используемые, например, одним компонентом исполнительной системы для реализации внутренних функций, не являются объектами.

Несмотря на то, что объекты используются для представления совместно используемых системных ресурсов, Windows NT не является объектно-ориентированной системой. Большая часть кода написана на C для обеспечения переносимости, а C непосредственно не поддерживает объектно-ориентированные конструкции.

*Диспетчер объектов (object manager)* – это компонент исполнительной системы NT, отвечающий за создание, удаление и защиту объектов. *Диспетчер объектов* централизует операции управления ресурсами и выполняет следующие задачи:

* Обеспечивает общий унифицированный механизм использования системных ресурсов;
* Сосредотачивает защиту объектов в одном месте (C2);
* Позволяет назначить цену использования объектов процессами;
* Задает унифицированные правила удержания объектов (т.е. объект продолжает существовать до тех пор, пока его использует хоть один процесс);
* Поддерживает требования разнообразных сред ОС.

В NT используются два типа объектов: объекты исполнительной системы (executive object) и объекты ядра (kernel object). Объекты исполнительной системы представляются различными компонентами исполнительной системы. Они доступны программам пользовательского режима. Объекты ядра – это более примитивный набор объектов, реализованный ядром. Эти объекты невидимы коду пользовательского режима, а создаются и используются только внутри исполнительной системы. Объекты ядра обеспечивают фундаментальные функции, которые могут выполняться только самым низким уровнем ОС – ядром. Рассмотрим объекты видимые в пользовательском режиме.

|  |  |  |
| --- | --- | --- |
| Тип объекта | Реализующий компонент | Что представляет собой |
| *Процесс (Process)* | Диспетчер процессов | Вызов программы, включая адресное пространство и ресурсы для ее выполнения |
| *Поток (Thread)* | Диспетчер процессов | Исполняемая сущность внутри процесса |
| *Секция (Section)* | Диспетчер памяти | Область совместно используемой памяти |
| *Файл (File)* | Диспетчер ввода/вывода | Образец открытого файла |
| *Маркер доступа (Access token)* | Система защиты | Информация о правах доступа зарегистрировавшегося в системе пользователя |
| *Событие (Event)* | Вспомогательные сервисы исполнительной системы | Объявление о том, что произошло событие |
| *Мутант* | То же | Механизм обеспечения взаимного исключения |
| *Семафор* | То же | Счетчик, регулирующий число потоков, которые могут использовать некоторый ресурс |
| *Таймер* | То же | Счетчик времени |
| *Каталог объектов* | Диспетчер объектов | Хранилище в памяти для имен объектов |

Таблица 1 Объекты исполнительной системы

### Файловая модель

Обычно *объекты исполнительной системы* создаются либо защищенной подсистемой, в качестве непосредственной реакции на некоторое действие пользователя, либо различными компонентами ОС в процессе работы. Например, для того чтобы создать файл, приложение Win32 вызывает функцию Win32 API CreateFile(). В свою очередь подсистема Win32 вызывает базовый сервис NT, создающий файловый объект исполнительной системы. Когда затем приложение читает из файла или записывает в него, подсистема Win32 и исполнительная система используют объект файл для работы с файлом.

Работа с файлами – это нетипичный случай объекта NT, так как файлы являются постоянными ресурсами и расположены не в памяти. Однако модель работы с файлами, используемая в большинстве языков программирования, удобна для работы с объектами. Рассмотрим модель работы с файлами:

* Необходимо открыть файл, прежде чем можно будет выполнить его чтение или запись. Можно открыть существующий файл, либо создать новый, с указанным именем (можно указать полное путевое имя);
* При открытии файла, задается тип операций, которые будут с ним совершатся (чтение, запись, добавление к концу файла и т.д.);
* Файловая система открывает файл и возвращает его описатель. В последующих операциях используется этот описатель, а когда работа с файлом закончена его необходимо закрыть;
* Две программы могут совместно использовать файл, если они обе получили его описатель.

Объектная модель имитирует файловаю модель. Описатели называются *описателями объектов (object handles)*, а сами объекты хранятся в памяти.

### Объектная модель

Как и в большинстве операционных систем, единицей работы в NT является процесс. Каждому процессу выделяется набор ресурсов, позволяющий ему выполнять свою работу: поток, чтобы можно было выполнять программу, и адресное пространство для хранения кода и данных. В процессе работы поток может запросить для своего процесса дополнительные ресурсы путем создания объектов или открывая описатели существующих объектов. Описатели объектов уникальны для процесса и указывают на его доступ к системным ресурсам.

Подсистема Win32 – это процесс NT, который выступает как сервер для приложений Win32. Когда приложение вызывает функцию API Win32, которая создает объект, подсистема Win32 обращается к объектному сервису NT (*Диспетчер объектов - Object Manager)*. И *диспетчер объектов* выполняет следующие функции:

* Выделяет память для объекта;
* Присоединяет к объекту *дескриптор защиты (security descriptor)*, который определяет, кому и как разрешено использовать объект;
* Создает и поддерживает *каталог объектов*, где хранятся имена объектов;
* Создает *описатель объекта* и возвращает его приложению.

Все процессы пользовательского режима, включая подсистемы среды, должны получить описатель объекта, прежде чем их потоки смогут использовать этот объект. Описатели служат косвенными указателями на системные ресурсы; эта косвенность предотвращает непосредственный доступ приложений к системным структурам данных. Кроме того, описатели объектов создают дополнительные преимущества:

* Нет никаких различий между описателем файла, события или процесса за исключением того, на что они ссылаются. Можно работать с различными типами объектов однотипно;
* Только диспетчер объектов создает описатели и производит поиск объекта по его описателю.

Таким образом, любое действие в пользовательском режиме, затрагивающее объект, контролируется диспетчером объектов. Диспетчер объектов:

* Защищает объекты. Всякий раз, когда поток использует описатель объекта, диспетчер объектов проверяет, есть ли у потока право использовать объект, так как он хочет;
* Контролирует, кто использует объект. Не используемые объекты удаляются, но если у какого-либо процесса имеется описатель этого объекта (или у системы есть указатель на него), то он не будет удален;
* Контролирует использование ресурсов. Каждый раз, когда поток открывает описатель объекта, диспетчер объектов списывает с процесса этого потока объем памяти, используемый объектом. Объем использования памяти потоками процесса не может превышать ограничений памяти – квот (quotas), назначенных системным администратором пользователю, который представлен данным процессом.

### Структура объектов

Каждый объект NT принадлежит к определенному типу объектов. Тип объекта определяет, какие данные содержит объект, а также какие сервисы могут к нему применяться. Для универсальности обработки разных объектов диспетчеру объектов необходимо, чтобы каждый объект содержал в заданном месте несколько полей со стандартной информацией. Поэтому объект разделен на две части – заголовок и тело (см. Рис. 1.Содержимое заголовка объекта). Диспетчер объектов работает с заголовком объекта, а другие компоненты *исполнительной системы* – с телами объектов создаваемых ими типов.



Рис. 1.Содержимое заголовка объекта

*Имя объекта* делает его видимым другим процессам для совместного использования.

*Каталог объектов* обеспечивает иерархическую структуру, в которой хранятся имена объектов. *Дескриптор защиты* определяет, кто и каким образом может использовать данный объект.

*Расход квоты* задает квоту на использование ресурсов, которая списывается с процесса, когда тото открывает описатель данного объекта.

*Счетчик открытых описателей* подсчитывает количество открытых описателей данного объекта.

*База данных открытых описателей* содержит список процессов, открывших описатели данного объекта.

*Временный/ постоянный статус* указывает можно ли уничтожить имя и освободить память объекта, если он больше не используется.

*Режим пользовательский/ядра* определяет, в каком режиме доступен объект. *Указатель на типовой объект* ссылается на типовой объект, который содержит атрибуты, общие для набора однотипных объектов.

*Счетчик ссылок –* подсчитывает, сколько раз компоненты режима ядра ссылались на адрес данного объекта.

Диспетчер объектов предоставляет набор сервисов общего назначения, которые работают с атрибутами заголовка объекта. Их можно использовать с объектами любых типов. Часть этих универсальных сервисов подсистема Win32 предоставляет пользовательским приложениям Win 32.

Универсальные объектные сервисы:

* *Закрыть* – закрывает описатель объекта;
* *Дублировать* – обеспечивает совместное использование объекта путем дублирования его описателя и передачи его копии другому процессу;
* *Опросить объект –* получить информацию о стандартных атрибутах объекта.
* *Опросить защиту* – получает дескриптор защиты;
* Установить защиту – *изменяет параметры защиты объекта;*
* *Ждать одного объекта –* синхронизирует выполнение потока с одним объектом;
* *Ждать несколько объектов -* синхронизирует выполнение потока с несколькими объектами.

Кроме заголовка каждый объект имеет тело, формат и содержимое которого определяется типом объекта. Тела всех объектов одного типа имеют одинаковый формат. Например, *диспетчер процессов* определяет тело объекта-процесса и обеспечивает сервисы для работы с хранящимися в нем данными.

### Типовой объект

В заголовке объекта хранятся данные, формат которых одинаков для всех объектов, а значения могут быть различными. Например, у каждого объекта есть уникальное имя и может быть уникальный дескриптор защиты. Однако, есть данные, которые постоянны для всех объектов данного типа.Например, при открытии описателя объекта данного типа можно выбирать права доступа из некоторого набора прав, специфичных для этого типа. *Исполнительная система NT* поддерживает права доступа, “завершить” и “приостановить”- для объектов-потоков, и права доступа: “чтение”, ”запись” и “удаление”, для файловых объектов.

В целях экономии памяти *диспетчер объектов* задает эти статические, типозависимые атрибуты один раз при создании нового типа объектов. Для хранения этих данных он использует специальный *типовой объект (type object).* Кроме того, *типовой объект* связывает друг с другом все объекты одного типа и при необходимости *диспетчер объектов* может перебрать их все.

Атрибуты тела типового объекта:

* *Имя типа объекта –* название объектов данного типа (“процесс”, “событие” и т.д.);
* *Типы доступа –* типы доступа, которые могут быть запрошены потоком при открытии описателя объекта данного типа (“чтение”, “запись”, “завершить”, “приостановить” и т.д.);
* *Возможности синхронизации –* может ли поток ожидать у объектов данного типа;
* *Резидентный/нерезидентный –* могут ли объекты данного типа выгружаться из памяти.

Некоторые из этих атрибутов видимы посредством базовых сервисов и функций API Win32. *Синхронизация,* один из таких атрибутов и он указывает на то, что поток может синхронизировать свое выполнение, ожидая пока не изменится состояние объекта. Поток синхронизируется со следующими объектами *исполнительной системы*: процесс поток событие и т.д.

Итак, объекты *исполнительной системы NT* состоят из двух частей: заголовка объекта, управляемого *диспетчером объектов,* и тела объекта, которое управляется компонентом *исполнительной системы*, создавшим данный тип. Одним из атрибутов заголовка объекта является указатель на типовой объект – структуру, определяющую статические атрибуты объектов данного типа.

## Управление объектами

*Диспетчер объектов* предоставляет набор универсальных сервисов, применимых к объектам лобого типа. Кроме того, другие компоненты *исполнительной системы NT* обеспечивают типозависимые сервисы для создаваемых ими типов объектов. Эти сервисы вызывают *диспетчер объектов* посредством внутренних интерфейсов. Следовательно, все сервисы, которые работают с объектами, должны пройти через *диспетчер объектов*.

### Имена объектов

При большом количестве объектов необходима эффективная система их отслеживания. Для этой цели *диспетчеру объектов* нужны:

* Способ отличить один объект от другого;
* Метод поиска и выбора заданного объекта.

Первое требование выполняется благодаря тому, что объектам можно назначать имена. Большинство ОС позволяют именовать только файлы, каналы или блоки совместно используемой памяти. В отличие от них, *исполнительная система NT* позволяет назначить имя любому объекту.

Второе требование, возможность поиска объекта, также выполняется благодаря тому, что объекты имеют имена. Так как диспетчер объектов хранит объекты по именам, то он может и провести поиск по имени.

Имена объектов дают еще и возможность процессам совместно использовать объекты. Пространство имен объектов *исполнительной системы NT* является глобальным, доступным всем процессам в системе. Один процесс может создать объект и поместить его имя в глобальное пространство имен, а другой процесс может открытьописатель этого объекта, указав его имя. Если нет необходимости использовать объект совместно, то при создании имя указывате не нужно.

*Диспетчер объектов* не ищет имя объекта всякий раз, когда кто-нибудь использует объект. Поиск по имени осуществляется лишь в двух случаях:

* Во-первых, когда процесс создает новый объект, *диспетчер объектов*, прежде чем поместить имя в глобальное пространство имен, осуществляет поиск по нему, чтобы убедиться, что оно уже не присвоено другому объекту;
* Во-вторых, когда процесс открывает описатель именованного объекта, *диспетчер объектов* осуществляет поиск по имени, находит объект и возвращает его описатель; после этого для ссылок на объект вызывающий процесс использует возвращенный описатель.

Имена объектов являются глобальными для данного компьютера, но не видимы в сети.

*Объект – каталог объектов (object directory object)* – это средство поддержки иерархической структуры имен *диспетчером объектов.* Он является аналогом каталога файловой системы и содержит имена других объектов, возможно и других каталогов объектов (Рис. 2 Иерархия имен объектов).



Рис. 2 Иерархия имен объектов

Сервисы создания и открытия используются для создания каталогов объектов и открытия их описателей. После того, как поток открыл описатель каталога объектов (с доступом по записи), он можем создавать другие объекты и помещать их в этот каталог.

Сервис опроса позволяет просматривать список объектов, хранящихся в каталоге. Объект-каталог объектов содержит информацию для трансляции имен объектов в указатели на них (Рис. 3 Объект каталог объектов). Диспетчер объектов использует эти указатели для создания описателей объектов, которые он возвращает программам пользовательского режима.



Рис. 3 Объект каталог объектов

В NT сервисы создания, открытия и опроса реализованы отдельно для каждого типа объектов.

### Описатели объектов

Имена объектов важны для хранения и совместного использования объектов, но используются они не часто. Процесс указывает имя объекта, когда он создает объект или открывает его описатель. После этого процесс использует описатель объекта. Ссылка на объект при помощи описателя выполняется быстрее, чем по имени, так как диспетчер объектов не производит поиск имени.

*Описатель объекта NT*  - это индекс в *таблице объектов (object table)*процесса. Таблица объектов процесса содержит указатели на все объекты, описатели которых открыты процессом (Рис. 4 Структура таблицы объектов). Процесс может получить описатель объекта:

* создав объект;
* открыв описатель существующего объекта;
* унаследовав описатель от другого процесса;
* получив дубликат описателя из другого процесса.



Рис. 4 Структура таблицы объектов

Каждый вход таблицы процессов содержит предоставленные права доступа для соответствующего описателя и его *режим наследования ( inheritance designation )*. Режим наследования, определяет, получат ли процессы, созданные данным процессом, копию этого описателя в своих таблицах объектов. Хотя термин *описатель (handle)* означает индекс, его используют и для обозначения данных, хранящихся в соответствующем входе таблицы.

Два процесса совместно используют объект, если они оба открыли его описатель. Каждый из этих описателей уникален.

#### Удержание объектов

Так как все процессы пользовательского режима, осуществляющие доступ к некоторому объекту, должны вначале открыть его описатель, то диспетчер объектов может отслеживать, сколько процессов, и какие именно, используют данный объект. Отслеживание открытых описателей – это первый шаг в реализации *удержания объекта (object retention)* , т.е. сохранения объектов только на то время, пока они используются, с последующим удалением.

Удержание объектов включает две фазы. Первая фаза называется *удержанием имени (name retention)*. Всякий раз, когда процесс открывает описатель объекта, диспетчер объектов увеличивает счетчик открытых описателей в заголовке объекта. После того, как процесс закончил работу с объектом и закрыл имеющийся у него описатель объекта, диспетчер объектов уменьшает счетчик открытых описателей в заголовке объекта. Когда счетчик обнуляется, диспетчер объектов удаляет объект из своего пространства имен.

Вторая фаза удержания объектов – это прекращение удержания (т.е. удаление объектов), когда они больше не используются. Так как ОС осуществляет доступ к объектам посредством указателей, а не описателей, диспетчер объектов должен учитывать количество указателей на объект, которые он передал процессам ОС. Всякий раз при выдаче указателя на объект диспетчер объектов увеличивает *счетчик ссылок (reference count)*. Когда поток ОС заканчивает работу с объектом, он обращается к диспетчеру объектов для уменьшения счетчика ссылок. Таким образом, даже после того, как счетчик описателей достиг нуля, счетчик ссылок может оставаться положительным, указывая, что ОС продолжает использовать объект. Когда счетчик ссылок обнуляется, диспетчер объектов удаляет объект из памяти.

Программистам, разрабатывающим приложения, которые состоят из двух или более взаимодействующих между собой процессов, не нужно беспокоиться о том, что один процесс может удалить объект, прежде чем другой закончит работу с ним. Кроме того, закрытие описателей объекта приложениями не приводит к удалению объекта, если его продолжает использовать ОС. Например, пусть один процесс создал другой процесс для выполнения некоторой программы в фоновом режиме. Сразу же после этого первый процесс закрывает описатель второго. Так как второй процесс нужен ОС для выполнения программы, то она сохраняет ссылку на объект процесс. Только после того, как фоновая программа закончит выполнение, диспетчер объектов уменьшит счетчик ссылок второго процесса и удалит этот процесс.

#### Учет использования ресурсов

Учет использования ресурсов, так же как и удержание объектов, связан с использованием описателей объектов. Если у объекта есть положительный счетчик открытых описателей, это означает, что некоторый процесс использует данный объект. Это также означает, что некоторый процесс “платит” за память, занятую объектом.

Многие ОС используют квоты для ограничения доступа процессов к системным ресурсам. Диспетчер объектов NT централизованный учет использования ресурсов. Каждому пользователю назначается предельный размер квоты – ограничение на объем памяти, который могут использовать его процессы. Заголовок каждого объекта содержит атрибут расход квоты. Это значение диспетчер объектов вычитает из выделенной процессу квоты, когда поток этого процесса открывает описатель этого объекта. Если процессы пользователя открыли слишком много описателей и израсходовали всю его квоту, то некоторые описатели необходимо закрыть, прежде чем удастся открыть новые.

## Защита объектов

Операционная сиситема должна быть защищена от нападения сразу на нескольких фронтах. Многопользовательская система должна защищать память, файла и другие ресурсы одного пользователя от других пользователей. Она должна защищать собственные данные, файлы и память от пользовательских программ. Кроме того, она должна отслеживать попытки обхода защиты. Министерство оборона США определило семь уровней защиты для ОС. Для соответствия уровню C2 в Windows NT должны быть следующие средства:

* Защищенная регистрация в системе. Необходимо, чтобы пользователь идентифицировал себя, введя уникальное имя и пароль, прежде чем ему будет предоставлен доступ к системе;
* Селективный контроль доступа. Владелец ресурса определяет, кто имеет доступ к данному ресурсу и задает тип доступа, назначая права доступа пользователю или группе пользователей;
* Аудит. Существует возможность обнаружения и регистрации важныж событий, имеющих отношение к защите, или любой попытки создания, использования или удаления системных ресурсов. Учет пользователей, выполнивших зарегистрированное действие, ппроизводится используя их идентификаторы;
* Защита памяти. Предотвращается чтение информации, записанной кем-либо в память, после того как этот блок памяти был возвращен ОС. Перед повторным использованием память реинициализируется.

Суть селективного контроля доступа и аудита состоит в защите объектов. Главная идея системы защиты Windows NT – это создание шлюза, через который должен пройти каждый пользователь системных ресурсов. Так как все системные ресурсы, защита которых может быть нарушена, реализованы как объекты, то таким шлюзом является диспетчер объектов.

Рассмотрим защиту объектов с двух точек зрения:

* Идентификация пользователей;
* Управление доступом пользователей к объектам.

### Маркеры доступа

Первая линия защиты Windows NT – это процесс регистрации. *Подсистема защиты (security subsystem)* отвечает за аутентификацию (authenticating) пользователей, т.е. за проверку того, что введенная пользователем информация при регистрации совпадает с информацией, хранящейся в базе данных защиты. После того, как подсистема защиты определила, что регистрация прошла успешно, она создает объект, который остается постоянно связанным с пользовательским процессом. Этот объект называется *маркером доступа (access token)*. Маркер доступа служит официальным удостоверением личности процесса, когда тот пытается использовать какой-либо системный ресурс. Он идентифицирует процесс и его потоки для ОС. Пример маркера доступа показан на Рис. 5Пример маркера доступа.



Рис. 5 Пример маркера доступа

Первый атрибут – это личный пользовательский идентификатор защиты (security ID), соответствующий идентификатору, указываемому пользователем при регистрации. Второй атрибут – это список групп, к которым принадлежит пользователь tvk.

При попытке процесса открыть описатель объекта диспетчер объектов вызывает справочный монитор защиты. Справочный монитор защиты получает маркер доступа, связанный с прцессом, и использует его идентификатор защиты и список групп, чтобы определить имеет ли процесс право доступа к объекту.

Небольшое количество чувствительных к защите системных сервисов (например, создание маркера), также защищены от использования. Атрибут привилегий перечисляет все такие сервисы, к которым имеет право обращаться пользователь. Большинство пользователей привилегий не имеют.

Пользователь, создавший объект, становится его владельцем и может решать, кто еще имеет доступ к объекту. *Список контроля доступа (access control list)*, по умолчанию хранящийся в маркере доступа – это первоначальный список прав доступа, который присоединяется к создаваемым пользователем объектам.

Рассмотрим атрибуты и сервисы объекта маркер доступа (Рис. 6 Объект маркер доступа).



Рис. 6 Объект маркер доступа

### Списки контроля доступа

При создании любого объекта ему присваивается дескриптор защиты. Основной частью дескриптора защиты является список контроля доступа (ACL). Владелец объекта, который обычно является его создателем, обладает правом селективного контроля доступа к объекту и может изменять ACL объекта, чтобы разрешить или запретить другим его использование.

Каждые вход ACL называется *элементом контроля доступа (access control entry)*. ACE содержит идентификатор защиты и набор прав доступа (см. Рис. 7 Список контроля доступа). Пользователю с соответствующим идентификатором защиты перечисленные права могут быть разрешены, запрещены или разрешены с аудитом. Сумма прав доступа, предоставленых отдельными ACE, формирует набор прав доступа, предоставляемых ACL.

Предположим, что вы хотите просмотреть файл. Если ACL фала содержит ACE с вашим идентификатором или с идентификатором одной из ваших групп, и этот ACE содержит право доступа “чтение”, то просмотр файла вам разрешен.



Рис. 7 Список контроля доступа

Система защиты назначает ACL новому объекту, применяя три взаимоисключающих правила, в приведенном ниже порядке:

* Если ACL явно задан при создании объекта, то система защиты присваивает объекту данный ACL;
* Если ACL не задан, а у объекта есть имя, то система защиты ищет ACL каталога объектов, в котором будет сохранено имя нового объекта, определяет те ACE, которые заданы наследуемыми и из них составляет ACL, который получит новый объект;
* Если не задано ни то ни другое, то система защиты присваивает новому объекту ACL по умолчанию из маркера доступа вызывающего процесса.

Кроме ACL, дескриптор защиты объектовсодержит поле, управляющее аудитом объекта. Аудит (auditing) – способность системы защиты шпионить за выбранными объектами и их пользователями и генерировать сообщения или оповещения, если кто-либо пытается выполнить операцию, использование которой с данным объектом запрещено. Например, аудит попыток чтения или изменения системного файла. Если кто-то пытается модифицировать заданный файл, то система защиты помещает сообщение в журнал аудита с SID пользователя.

### Как все это работает вместе

Маркер доступа идентифицирует процесс и его потоки для ОС, а дескриптор защиты содержит информацию о том, какие процессы имеют доступ к объекту. Когда поток открывает описатель объекта, диспетчер объектов и подсистема защиты сопоставляют эту информацию, чтобы определить, следует ли предоставить вызывающему потоку запрашиваемый им описатель (см. Рис. 8Проверка прав доступа).



Рис. 8 Проверка прав доступа

Проверяя ACL, подсистема защиты просматривает список, начиная с первого ACE. Когда она находит идентификатор защиты вызывающего или его группы, она останавливает поиск и проверяет, разрешает ли данный ACE доступ запрашиваемого типа. Если ACL, разрешающий доступ, найден, то поиск прекращается и вызывающему возвращается описатель. Если подсистема достигает конца списка, не найдя идентификатор защиты вызывающего или его группы, то доступ запрещается.

Делать проверку при всяком использовании описателя процессом неэффективно, так как ACL может содержать много записей, а процесс может за время работы осуществлять доступ ко многим объектам. Кроме того, одновременно могут быть активными несколько процессов. Поэтому проверка осуществляется только при открытии описателя, а не при всяком его использовании. Когда объекты использует сама ОС используя указатели проверка прав доступа не производится.

После того, как процесс успешно открыл описатель, предоставленные ему права доступа не могут быть отозваны подсистемой защиты, даже если изменился ACL объекта. Последнее потребовало бы проверки прав доступа при всяком использовании описателя, а это могло бы значительно снизить производительность, особенно для объектов с длинным ACL.

## лабораторная работа 3 Объекты (программирование)

### Теория

Без четкого понимания, что такое объекты не стать профессионалом в области разработки Windows программ. Объекты используются ОС (исполнительной системой и ядром) и приложениями для управления различными ресурсами: процессами, потоками, файлами и т.д.

Объектами управляет **диспетчером объектов** (**Object Manager**) исполнительной системы. Исполнительная система создает объект, когда она выделяет ресурсы процессу. Приложение не может получить прямой доступ к объекту, а только через соответствующий интерфейс. Кроме того, объект размещается в системной области и пользовательские программы не могут напрямую обращаться к памяти, в которой объект расположен.

#### Что такое объект

Как разработчик Windows приложений вы будите постоянно выполнять различные операции с объектами. Типы используемых объектов частично уже упоминались, но приведем некоторые из них еще раз:

* **Object Directory** – тип объекта, который включает набор других объектов. Исполнительная система использует этот тип для организации набора объектов.
* **Symbolic link** – ссылка на другой объект по его символическому имени. Этот тип используется для поддержки такой возможности.
* **Process** **–** тип объекта, который представляет процесс**.**
* **Thread** – тип объекта, который представляет поток.
* **Section** – тип объекта, который используется для реализации разделяемой памяти, используемой в адресных пространствах процессов.
* **File Port** – тип объекта, который представляет описатель файла, когда файл открыт для использования.
* **Access Token** – тип объекта, который использует Security Reference, Monitor Local Security Authority (LSA) для аутентификации пользователя и другие части системы защиты.
* **Event** – класс объектов, которые могут перехватывать определенное событие (event) в системе, таким образом, что другие части системы смогут выполнить определенные действия, когда они уверены, что событие произошло. Объект событие это основа для многих операций синхронизации.
* **Semaphore** – классические семафоры (Dijkstra).
* **Mutex** – тип объекта для синхронизации, используемый для взаимно исключающего доступа к критическим секциям.
* **Timer** – тип объекта, который используется для уведомления потока о том, что заданное время истекло.

Каждый объект содержит данные, определяемые типом, и стандартный *заголовок* (*title*) с которым работает **диспетчер объектов** (**Object Manager)**. В заголовке содержится имя объекта, атрибуты безопасности и т.д.

Каждый объект – это просто блок памяти, выделенный в системной области памяти. Этот блок представляет структуру данных, в элементах которой содержится информация об объекте. Некоторые элементы (дескриптор защиты, счетчик числа открытых описателей и д.р.) присутствует во всех объектах (заголовок), но большая часть специфична для объектов конкретного типа.

Приложение не может найти и модифицировать содержимое объектов. Для этого в Windows предусмотрен набор функций. Мы получаем доступ к объектам только через эти функции. Когда вы вызываете функцию, создающую объект, она возвращает описатель, идентифицирующий созданный объект. Далее этот описатель может использовать любой поток вашего процесса, чтобы сообщить системе какой объект вас интересует.

Для надежности ОС Microsoft сделала так, чтобы значения описателей зависели от конкретного процесса.

#### Таблица описателей объектов

При инициализации процесса система создает в нем таблицу описателей объектов. Сведения о структуре этой таблицы плохо документированы, но квалифицированный программист должен понимать, как устроена таблица описателей объектов в процессе. Как видно из Таблица 2 Структура таблицы описателей объектов процесса,это просто массив структур данных. Каждая структура содержит указатель на какой-нибудь объект, маску доступа и некоторые флаги.

|  |  |  |  |
| --- | --- | --- | --- |
| Индекс | Указатель на блок памяти объекта ядра | Маска доступа | Флаги |
| 1 | 0x???????? | 0x???????? | 0x???????? |
| 2 | 0x???????? | 0x???????? | 0x???????? |
| *...* | *...* | *...* | *...* |

Таблица 2 Структура таблицы описателей объектов процесса

#### Создание объекта

Когда процесс создается, таблица описателей еще пуста. Но стоит одному из его потоков вызвать функцию, создающую объект, как диспетчер объектов выделяет для этого объекта блок памяти и инициализирует его. Дальше диспетчер объектов просматривает таблицу описателей процесса и ищет в ней первую свободную запись. Поскольку таблица еще пуста, то будет инициализирована структура с индексом 1. Указатель будет установлен на внутренний адрес структуры объекта, маска доступа – на доступ без ограничений.

Вот некоторые функции, создающие объекты ядра:

HANDLE ***CreateThread***(

PSECURITY\_ATTRIBUTES psa,

DWORD dwStackSize,

PTHREAD\_START\_ROUTINE pfnStartAddr,

PVOID pvParam,

DWORD dwCreationFlags,

PDWORD pdwThreadId);

HANDLE ***CreateFile***(

PCTSTR pszFileName,

DWORD dwDesiredAccess,

DWORD dwShareMade,

PSECURITY\_ATTRIBUTES psa,

DWORD dwCreationDistribution,

DWORD dwFlagsAndAttributes,

HANDLE hTemplateFile);

HANDLE ***CreateFileMapping***(

HANDLE hFile,

PSECURITY\_ATTRIBUTES psa,

DWORD flProtect,

DWORD dwMaximumSizeHigh,

DWORD dwMaximumSizeLow,

PCTSTR pszFileName);

HANDLE ***CreateSemaphore***(

PSECURITY\_ATTRIBUTES psa,

LONG lInitialCount,

LONG lMaximumCount,

PCTSTR pszFileName);

Все функции, создающие объекты, возвращают описатели, которые привязаны к конкретному процессу и могут быть использованы в любом потоке данного процесса. Значение описателя представляет собой индекс в таблице описателей процесса, и таким образом идентифицирует место, где хранится объект. В Windows 2000 это значение определяет не индекс, а байтовое смещение нужной записи от начала таблицы.

Всякий раз, при вызове функции, которая в качестве аргумента использует описатель объекта, вы передаете ей значение, возвращаемое одной из ***Create<Class>*** функций. При этом функция смотрит в таблицу описателей объектов вашего процесса и считывает адрес нужного объекта.

Если вы передаете неверный описатель, то функция завершается с ошибкой и ***GetLastError*** возвращает ERROR\_INVALID\_HANDLE. Это может быть связано с тем, что описатели представляют собой индексы в таблице объектов процесса и недействительны в других процессах.

Если объект создать не удалось, то обычно возвращается NULL (0). Это может произойти при нехватке памяти или проблем с защитой. Но некоторые функции возвращают в таком случае не NULL, а INVALID\_HANDLE\_VALUE (-1). Например, если ***CreateFile*** не сможет открыть указанный файл, то она вернет INVALID\_HANDLE\_VALUE. **Поэтому будьте очень внимательны при проверке значений, возвращаемых функциями, создающими объекты.**

HANDLE hMutex = ***CreateMutex***(…);

If ( hMutex == INVALID\_HANDLE\_VALUE )

{

// этот код не будет выполнен никогда

// при ошибке CreateMutex возвращает NULL

}

HANDLE hFile = ***CreateFile***(…);

If (hFile == NULL)

{

// этот код не будет выполнен никогда

// при ошибке CreateFile возвращает INVALID\_HANDLE\_VALUE

}

#### Закрытие объекта

Независимо от того, как именно вы создали объект, по окончании работы с ним его нужно закрыть вызвав ***CloseHandle***:

BOOL ***CloseHandle***(HANDLE hObj);

Эта функция сначала проверяет таблицу описателей вызывающего процесса, чтобы убедиться, идентифицирует ли переданный ей описатель объект, к которому этот процесс действительно имеет доступ. Если передан правильный индекс, то система получает адрес структуры данных объекта и уменьшает в этой структуре счетчик открытых описателей; как только счетчик будет равен 0, объект будет удален из памяти.

Если описатель неверен, то происходит одно из двух:

* в нормальном режиме ***CloseHandle*** возвращает FALSE, а GetLastError – ERROR\_INVALID\_HANDLE;
* в режиме отладки система просто уведомляет об ошибке.

Перед самым возвратом управления ***CloseHandle*** удаляет соответствующую запись из таблицы описателей. После этого данный описатель недействителен в вашем процессе, и использовать его нельзя. После вызова ***CloseHandle*** вы больше не получите доступ к этому объекту ядра; но если счетчик числа открытых описателей не обнулен, то объект останется в памяти. Это означает, что объект используется другим процессом или процессами. Когда и они завершат с ним работу, вызвав ***CloseHandle***, он будет уничтожен.

#### Учет объектов

Объекты принадлежат системе, а не процессу. Например, если ваш процесс создает объект, а затем завершается, объект может быть не уничтожен. Если созданный вами объект используется другим процессом, то система запретит его разрушение до тех пор, пока от него не откажется и тот процесс.

Системе известно, сколько процессов использует конкретный объект, так как в заголовке каждого объекта имеется *счетчик открытых описателей*. В момент создания объекта счетчику присваивается 1. Когда к существующему объекту обращается другой процесс, счетчик увеличивается на 1. А когда какой-то процесс завершается, все его счетчики автоматически уменьшаются на 1. Как только счетчик какого-либо объекта обнуляется, этот объект уничтожает.

#### Защита объектов

Вызовы функций ***Create<Class>*** или ***Open<Class>*** использовались для запроса системных ресурсов, для того, чтобы объект был размещен, а описатель возвращен вызывающему потоку. Механизм защиты **диспетчера объектов** требует, чтобы при вызове этих функций были заданы *права доступ*. Существует набор общих прав доступа для всех объектов (чтение и запись), также есть типозависимые права (например, отложить доступ к вновь созданному процессу). При вызове ***Open<Class>*** **диспетчер объектов** проверяет права доступа, используя SECURITY\_DESCRIPTOR и если возможно, предоставляет желаемый доступ.

Механизм безопасности основан на возможности аутентифицировать пользователя, который запустил программу. Процесс WINLOGIN аутентифицирует пользователей, когда они регистрируются в системе. Когда **диспетчера объектов** создает объект, он создает дескриптор безопасности, который включает идентификатор владельца, дискретный и системный списки контроля доступа. Дискретный список контроля доступа это список процессов и их прав для объекта, а системный список контроля доступа это список процессов и их прав, которые должны быть записаны в журнал безопасности при использовании для последующего аудита.

Если доступ разрешен, то Security Reference Monitor возвращает точный набор прав доступа, которые гарантируются вызывающему процессу и **диспетчера объектов** сохраняет эти права как часть описателя объекта в таблице описателей объектов процесса. Когда поток процесса использует описатель, его права доступа к данному объекту сравниваются с гарантированным доступом, перед тем как к объекту будет предоставлен запрашиваемый доступ.

В лабораторной работе 2 мы уже использовали ***CreateProcess*** для создания нового процесса.

BOOL ***CreateProcess***(

…

LPSECURITY\_ATTRIBUTES lpProcessAttributes,

// атрибуты безопасности процесса ( указатель )

LPSECURITY\_ATTRIBUTES lpThreadAttributes,

// атрибуты безопасности потока ( указатель )

…

);

lpProcessAttributes и lpThreadAttributes это указатели на структуру данных SECURITY\_ATTRIBUTES.

typedef struct\_SECURITY\_ATRIBUTES { //sa

DWORD nLength;

LPVOID lpSecurityDescriptor;

BOOL bInheritHandle;

} SECURITY\_ATTRIBUTES;

lpSecurityDescriptor это указатель на структуру SECURITY\_DESCRIPTOR, она не документирована, но ее поля можно прочитать или установить используя Win32 API (см. в MSDN SECURITY\_DESCRIPTOR). Функции Win32 API ***GetSecurityDescriptorControl***, ***SetSecurityDescriptorDacl*** и другие позволяют установить значения полей в SECURITY\_DESCRIPTOR. Таким образом, вызывающий поток может специфицировать желаемые права доступа, которые будут назначены объекту или использовать аутентифицированный доступ для существующего объекта.

Значение по умолчанию для атрибутов безопасности (в вызове ***CreateProcess***) NULL, это означает, что структура SECURITY\_ATTRIBUTES при вызове не передается. **Диспетчер процессов** (**Process Manager**) и **диспетчер объектов** (**Object Manager**) интерпретируют NULL следующим образом: процесс ребенок (или поток) будет использовать SECURITY\_DESCRIPTOR для существующего объекта или системное значение по умолчанию, если создается новый объект.

Если вы хотите, то можете ограничить доступ ребенка к его собственному объекту-процессу. Для этого необходимо создать SECURITY\_ATTRIBUTE и выполнить вызов, чтобы список контроля доступа не давал возможности ребенку ссылаться на объект. Это необычно когда вы ограничиваете доступ ребенка к его собственному объекту процессу. Один и тот же механизм атрибутов безопасности используется для всех объектов исполнительной системы файлов, секций и т.д.

Итак, объекты можно защищать, используя дескриптор защиты (security descriptor), который описывает, кто создал объект и кто имеет права доступа к нему.

Почти все функции, создающие объекты, принимают указатель на структуру SECURITY\_ATTRIBUTES. Рассмотрим еще один пример.

HANDLE ***CreateFileMapping***(

HANDLE hFile,

PSECURITY\_ATTRIBUTES psa,

DWORD flProtect,

DWORD dwMaximumSizeHigh,

DWORD dwMaximumSizeLow,

PCTSTR pszFileName);

Чтобы ограничить доступ к созданному объекту, создадим дескриптор защиты и инициализируем структуру SECURITY\_ATTRIBUTES следующим образом:

SECURITY\_ATTRIBUTES sa;

sa.nLength = sizeof(sa);

sa.lpSecurityDescriptor = pSD; // адрес инициализированной SD

sa.bInheritHandle = FALSE;

HANDLE hFileMapping = ***CreateFileMapping***(INVALID\_HANDLE\_VALUE, &sa,

PAGE\_READWRITE, 0, 1024, “MyFileMapping”);

Для получения доступа к существующему объекту, необходимо указать какие операции вы собираетесь проводить с ним. Например, если мы хотим считать данные из существующей проекции файла, то надо вызвать функцию ***OpenFileMapping*** так:

HANDLE hFileMapping=***OpenFileMapping***(FILE\_MAP\_READ, FALSE, “MyFileMapping”);

Передавая первым параметром FILE\_MAP\_READ, мы просим предоставить доступ к проекции файла для чтения. Функция ***OpenFileMapping*** прежде чем вернуть описатель, проверит тип защиты объекта. Если доступ разрешен, то функция возвратит действительный описатель, а если в доступе отказано, то NULL, а вызов ***GetLastError*** вернет код ошибки ERROR\_ACCESS\_DENIED (5).

#### Работа с объектами

Когда в лабораторной работе 2 ваш поток использовал ***CreateProcess*** и ***CreateThread*** для создания другого процесса или потока, он передавал имя **диспетчеру объектов** (Object Manager). При этом возвращались описатель (HANDLE) и идентификатор. В случае ***CreateProcess*** описатели нового процесса и нового потока, а также их идентификаторы заносятся в структуру данных PROCESS\_INFORMATION. Функция ***CreateThread*** также возвращает описатель созданного потока и его идентификатор. После того как пользовательский поток получил описатель, он может передавать его исполнительной системе как параметр для других запросов. Например, после того как один поток создал другой поток, он передает описатель обратно исполнительной системе, закрывая описатель потока.

ChildThreadHandle = ***CreateThread***(…);

…

***CloseHandle***(ChildThreadHandle);

Когда **диспетчеру объектов** обрабатывает первый вызов, который ссылается на объект, он:

* добавляет имя объекта к набору используемых имен объектов;
* создает объект исполнительной системы с телом и заголовком;
* инициализирует поля заголовка;
* передает объект другим компонентам исполнительной системы, для заполнения тела типозависимой информацией.

Исполнительная система может определить, что такой объект уже существует, так как другой процесс (или другой поток) его уже создал. В этом случае *счетчик открытых описателей* в заголовке объекта увеличивается на 1, указывая, что теперь два описателя ссылаются на этот объект. Операции “открыть” приводят к увеличению значения счетчика на 1, а “закрыть” к уменьшению значения счетчика на 1. Когда значение счетчика становится равным 0, это означает, что больше не существует пользовательских описателей, которые ссылаются на этот объект. В этом случае **диспетчеру объектов** удаляет имя объекта из пространства имен.

Кроме того, компоненты исполнительной системы могут также ссылаться на объект, но для этого им нет необходимости использовать описатель (они могут использовать адрес объекта, так как они функционируют в режиме ядра). Поэтому заголовок объекта содержит также *счетчик ссылок* для хранения числа всех ссылок на объект ( см. Рис. 9 Счетчики описателей и ссылок).



Рис. 9 Счетчики описателей и ссылок

Операции “открыть” в пользовательском режиме или в режиме ядра приводят к увеличению значения счетчика на 1, а “закрыть” к уменьшению значения счетчика на 1. Когда значение счетчика становится равным 0, это означает, что объект не используется никакими программными компонентами (пользовательскими или ядра) и можно освободить память, которую он занимал.

Какие отношения существуют между объектом ядра, описателем объекта и его идентификатором? Как показано на Рис. 10 Описатели и таблица описателей создается исполнительной системой и ядром и хранится в области памяти ядра, поэтому пользовательский поток не может получить к нему прямой доступ. Описатель это 32 битная ссылка на объект, определенная для процесса. Это смещение в таблице описателей объектов процесса. Когда исполнительной системе нужно создать описатель объекта, она сначала ищет в таблице описателей процесса свободную строку. Затем заносит в первое поле адрес объекта в области ядра, во второе разрешения на доступ и т.д. Таким образом, возвращаемый описатель это индекс в таблице описателей объектов процесса, расположенной в области ядра. Описатель доступен только потокам, которые существуют в том же адресном пространстве.



Рис. Описатели и таблица описателей

Если необходимо идентифицировать объект в разных адресных пространствах, то вместо описателя можно использовать идентификатор.

#### Объект WaitableTimer

В этой лабораторной работе надо будет использовать *объект ожидаемый таймер* (*WaitableTimer*) для управления поведением потока. В лабораторной работе 2 предлагалось использовать ***GetSystemTime*** и разделяемую переменную runFlag для определения, когда остановить набор потоков. Один поток читал системное время и определял, не пора ли завершить работу, и если нет, то он блокировал себя на определенное время, а затем просыпался и снова проверял. Попробуем сделать лучше, используя *ожидаемый таймер*. Этот тип объекта умеет периодически сообщать вашему процессу, что заданный интервал времени истек. Точность 100 ns.

HANDLE ***CreateWaitableTimer***(

LPSECURITY\_ATTRIBUTES lpTimerAttributes,

// указатель на атрибуты безопасности

BOOL bManualReset, // флаг ручного перезапуска

LPCTSTR lpTimerName // указатель на имя объекта таймера

);

Атрибуты безопасности используются такие же, как в ***CreateProcess*** и ***CreateThread***. Задавая lpTimerName можно создать именованный объект или получить описатель уже существующего объекта (открыть). Если объект с таким именем уже существует, то ***GetLastError*** вернет ERROR\_ALREADY\_EXISTS. Параметр bManualReset контролирует число потоков, которые получат сигнал от *ожидаемого таймера*, когда он его пошлет. Если он TRUE, то все потоки, которые ждут таймер, получат уведомление, в противном случае только один. Функция возвращает описатель таймера.

Поток может получить описатель существующего таймера, используя ***OpenWaitableTimer***.

HANDLE ***OpenWaitableTimer***(

DWORD dwDesiredAccesss, // флаг доступа

BOOL bInheritFlag, // флаг наследования

LPCTSTR lpTimerName // указатель на имя объекта таймера

);

Используя dwDesiredAccess, процесс может задать то, как он собирается использовать описатель таймера. Вы можете только получать уведомление от таймера, устанавливать период между посылаемыми уведомлениями или иметь полный доступ ко всем функциям таймера.

После того, как таймер создан, он должен быть установлен, перед тем как начнет выдавать уведомления. Для этого используется функция ***SetWaitableTimer***.

BOOL ***SetWaitableTimer***(

HANDLE hTimer, // описатель таймера

Const LARGE\_INTEGER \*pDueTime,

// когда таймер перейдет в состояние signaled

LONG lPeriod, // периодический временной интервал

PTIMERAPCROUTINE pfnCompletionRoutine,

// указатель на процедуру (APC)

LPVOID lpArgToCompletionRoutine,

// данные, передаваемые в процедуру (APC)

BOOL fResume // флаг состояния

);

htimer это описатель, который возвращают функции ***CreateWaitableTimer*** и ***OpenWaitableTimer*** (при его создании или открытии).

pdueTime - это число в 64 битовом формате FILETIME.

typedef struct\_FILETIME { //ft

DWORD dwLowDateTime;

// это младшие 32 бита времени в системном формате

DWORD dwHighDateTime;

// это старшие 32 бита

} FILETIME;

Вы можете задать pDueTime как относительное или абсолютное время, отличить относительное время от абсолютного можно установив 64 битовое время отрицательным. lperiod задает время в милисекундах между уведомлениями. Если задано 0 значение, то таймер пошлет уведомление только один раз.

Следующие два параметра pfnCompletionRoutine и lpArgToCompletionRoutine здесь не рассматриваются. Флаг fResume необходим для особого режима использования компьютера, в случае если компьютер завис, то если значение флага TRUE, при получении уведомления он будет перезагружен.

Теперь рассмотрим использование функции ***WaitForSingleObject***. Когда поток вызывает эту функцию, он будет заблокирован, пока не придет уведомление от указанного объекта. Поток может создать таймер, установить его и затем ждать от него уведомления, вызвав ***WaitForSingleObject***.

DWORD ***WaitForSignleObject***(

HANDLE hHandle; // описатель ожидаемого объекта

DWORD dwMilliseconds; // тайм-аут в милисекудах

);

У нас hHandle это описатель таймера. dwMilliseconds определяет максимальный промежуток времени, который поток желает ждать пока истечет время. Вы можете использовать ***GetLastError***, чтобы посмотреть, что возвращает функция:

* WAIT\_OBJECT\_O – если послано уведомление;
* или WAIT\_TIMEOUT – истек тайм-аут.

Вы можете также задать INFINITE для ожидания уведомления без тайм-аута.

Скелет кода с использованием таймера.

#define \_WIN32\_WINNT 0x0400

#include <windows.h>

#include <stdio.h>

#define \_SECOND 10000000

void main( void )

{

HANDLE wTimer;

\_\_int64 endTime;

LARGE\_INTEGER quitTime;

SYSTEMTIME now;

wTimer = CreateWaitableTimer(NULL,FALSE,NULL);

endTime = -5 \* \_SECOND;

quitTime.LowPart = (DWORD) (endTime & 0xFFFFFFFF);

quitTime.HighPart = (LONG) (endTime >> 32);

SetWaitableTimer(wTimer,&quitTime,0,NULL,NULL,FALSE);

GetSystemTime(&now);

printf("System Time %d %d %d\n",now.wHour,now.wMinute,now.wSecond);

WaitForSingleObject(wTimer,INFINITE);

printf("Waitable Timer sent signal\n");

GetSystemTime(&now);

printf("System Time %d %d %d\n",now.wHour,now.wMinute,now.wSecond);

CloseHandle( wTimer );

}

### Чему нужно научиться

Необходимо научиться работать с объектами.

### Задания

#### Уровень 1 (A)

Написать программу, которая использует ожидаемый таймер (WaitableTimer), чтобы остановить себя через К секунд после старта, где К параметр командной строки.

#### Уровень 2 (A)

Модифицировать программу первого уровня, чтобы она создавала N фоновых процессов, каждый запускал бы программу, которая заканчивалась бы в случайное время по своему собственному усмотрению (N – параметр командной строки). А после того, как истекут К секунд (К – параметр командной строки), она уничтожала процессы, которые не закончились сами. Если все фоновые процессы закончились, то главная программа сама выполняется К секунд, а потом заканчивается.

#### Уровень 3 (A)

Поработайте с правами доступа. Убедитесь в том, что проверка прав доступа осуществляется только при открытии описателя, а не при всяком его использовании. После того, как процесс успешно открыл описатель, предоставленные ему права доступа не отзываются подсистемой защиты, даже если изменился ACL объекта.

Используйте функции: ***GetSecurityInfo, SetSecurityInfo, GetNamedSecurityInfo, SetNamedSecurityInfo, LookupAccountSid, GetEffectiveRightsFromAcl,*** ***AddAccessAllowedAce, AddAccessDeniedAce*** и т.д.



#### Скелет кода фоновой программы

Хотим создать процесс, который может закончиться или не закончиться по своему собственному усмотрению.

#include <windows.h>

#include <stdio.h>

int main(int argc, char \*argv[])

{

printf(“Client %s beginning to run\n”,argv[1]);

while(TRUE)

{

if(getc(stdin)==’y’) break;

getc(stdin);

}

return 0;

}

Для выполнения задания вам может понадобиться ***TerminateProcess***:

BOOL ***TerminateProcess***(

HANDLE hProcess, // описатель процесса

UINT uExitCode // код выхода для процесса

);

Один процесс может закончить другой процесс, но он должен иметь на это соответствующий допуск (PROCESS\_TERMINATE) для этого описателя.