Архитектура ОС MS Windows 2000+

Реестр

Реестр

- Операционная система управляет большим объемом информации, необходимой для ее загрузки и конфигурирования.
- В ранних версиях Windows эта информация содержалась в различных текстовых файлах с расширением .ini (Win.ini, System.ini и т.д.).
- Начиная с Windows 95, эта информация хранится в централизованной общесистемной базе данных, называемой реестром (registry). Для просмотра и модификации данных реестра имеется штатный утилиты редактор реестра regedit.

Структура реестра

- Данные реестра хранятся в виде иерархической древовидной структуры. Каждый узел или каталог называется разделом или ключом (keys), а названия каталогов верхнего уровня начинаются со строки HKEY. Каждый раздел может содержать подраздел (subkey).
- Реестр содержит шесть корневых разделов:
  - HKEY_CURRENT_USER,
  - HKEY_USERS,
  - HKEY_CLASSES_ROOT,
  - HKEY_LOCAL_MACHINE,
  - HKEY_PERFORMANCE_DATA,
  - HKEY_CURRENT_CONFIG.
- Наиболее важным, вероятно, является раздел HKEY_LOCAL_MACHINE. В нем содержится вся информация о локальной системе.

Хранение реестра

- Реестр хранится на диске в виде набора файлов, называемых «кустами» или «ульями» (hives). Большинство из них находится в каталоге \Systemroot\System32\Config. Большое значение уделяется повышению надежности хранения.
- В частности, система ведет протоколы модификации кустов (при помощи так называемых регистрационных кустов, log hives), которые обеспечивают гарантированную возможность восстановления постоянных кустов реестра. Для еще большей защиты целостности на диске поддерживаются зеркальные копии критически важных кустов.

Управление реестром с использованием Win32 API

- Функция *RegCreateKeyEx ()* создает указанный ключ. Если ключ уже существует в реестре, то функция открывает его.
- Функция *RegOpenKeyEx ()* открывает указанный ключ.
- Функция RegCloseKey освобождает дескриптор указанного ключа.
- Функция *RegDeleteKey ()* удаляет указанный ключ. Эта функция не может удалить ключ, который является подключем.
- Функция *RegSetValueEx ()* сохраняет данные в поле значения открытого ключа реестра.
- Функция *RegQueryValueEx ()* возвращает тип и данные указанного значения по имени, ассоциирующимся с открытым ключом реестра.

Подробнее

- Структура кустов подробно описана в (Руссинович М., Соломон Д. Внутреннее устройство Microsoft Windows: Windows Server 2003, Windows XP и Windows 2000), а их дескрипторы можно просмотреть с помощью утилиты **Handleex.exe** с сайта www.sysinternals.com.

Самостоятельная работа

- Исходники http://jmhartsoftware.com/WSP4_Examples.zip
- Глава 3, программа LsREG

Архитектура ОС MS Windows 2000+

Подсистема защиты Windows

Базовые принципы уровня C2

- Выделяемая процессам память защищена таким образом, что прочитать информацию оттуда невозможно даже после того, как она уже освобождена процессом.
- Система должна быть защищена от вмешательства, например, от модификации системного кода в памяти или системных файлов на диске.
- Только системный администратор имеет физическую возможность управлять безопасностью системы и уровнем доступа отдельных лиц и групп лиц.
- Должно осуществляться управление доступом к ресурсам. Должно быть возможным разрешать или запрещать доступ к указанным ресурсам как отдельным пользователям, так и группам пользователей.
- Пользователи должны регистрировать себя в системе и иметь уникальные идентификаторы. Все действия пользователей, контролируемые системой, должны быть персонифицированы.

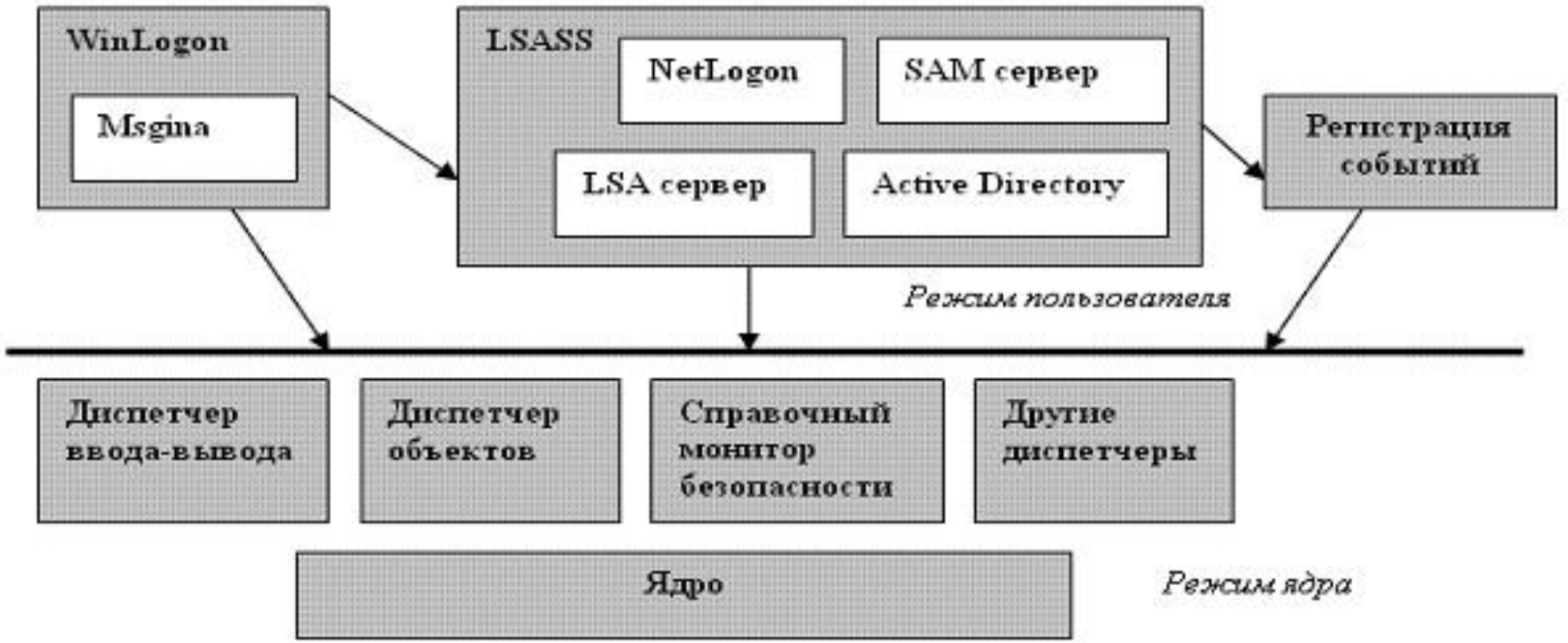
Идентификатор безопасности

- Идентификатор безопасности (*Security Identifier – SID*) – структура данных переменной длины, которая идентифицирует учетную запись пользователя, группы, домена или компьютера.
- SID ставится в соответствие каждой учетной записи в момент её создания.
- Существует список стандартных (хорошо известных) SID в операционных системах Windows:
  - http://support.microsoft.com/kb/243330
- Подробнее о структуре SID:
  - http://www.samag.ru/archive/article/1666

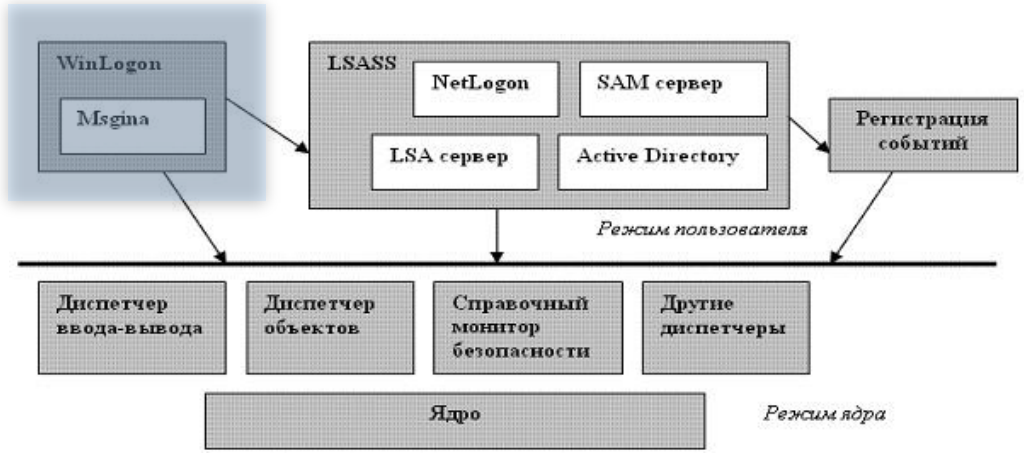
Маркер доступа

- При регистрации пользователя в системе после успешной проверки имени и пароля создается маркер доступа, содержащий информацию о пользователе. Для каждого процесса, выполняемого далее в контексте этого пользователя, создается копия маркера доступа.
- Маркер доступ содержит информацию безопасности сеанса входа, определяющую пользователя, группы пользователей и список привилегий.
- Операционная система использует маркер доступа для контроля доступа к защищаемым объектам и контролирует возможность выполнения пользователем различных связанных с системой операций на локальном компьютере.

Основные компоненты подсистемы защиты



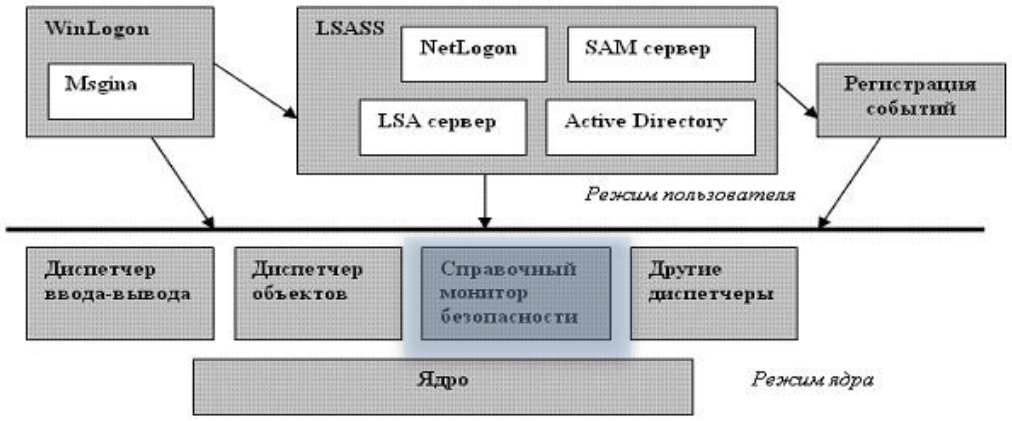
Процесс входа (Winlogon)



- Процесс пользовательского режима *\Windows\System32\ Winlogon.exe*.

- Winlogon отвечает за поддержку аутентификации и управление сеансами интерактивного входа в систему. Например, при регистрации пользователя Winlogon создает пользовательский интерфейс.
- Стандартная библиотека аутентификации Gina реализована в файле Msgina.dll.

Монитор безопасности



- Монитор безопасности – компонент исполнительной системы *\Windows\System32\Ntoskrnl.exe*.

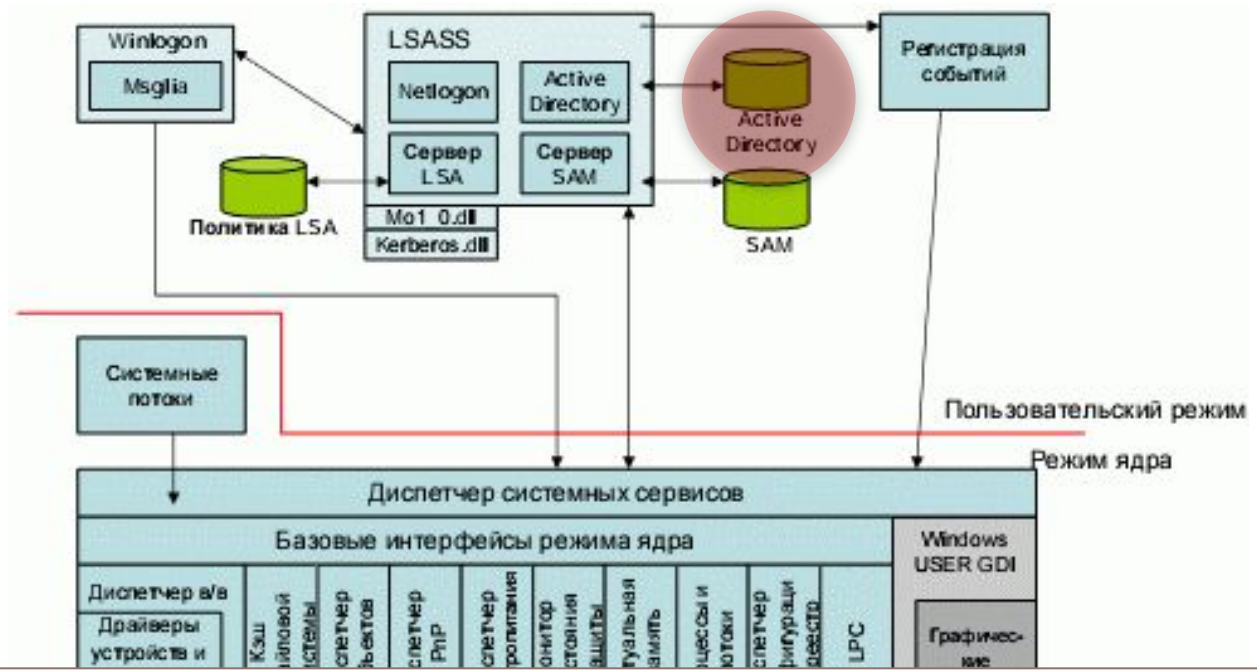
- Монитор отвечает за определение структуры данных маркера доступа для представления контекста защиты, за проверку прав доступа к объектам, манипулирование привилегиями (правами пользователей) и генерацию сообщений аудита безопасности.

Подсистема локальной аутентификации

- Процесс пользовательского режима, выполняющий образ *\Windows\System32\l_sass.exe*.

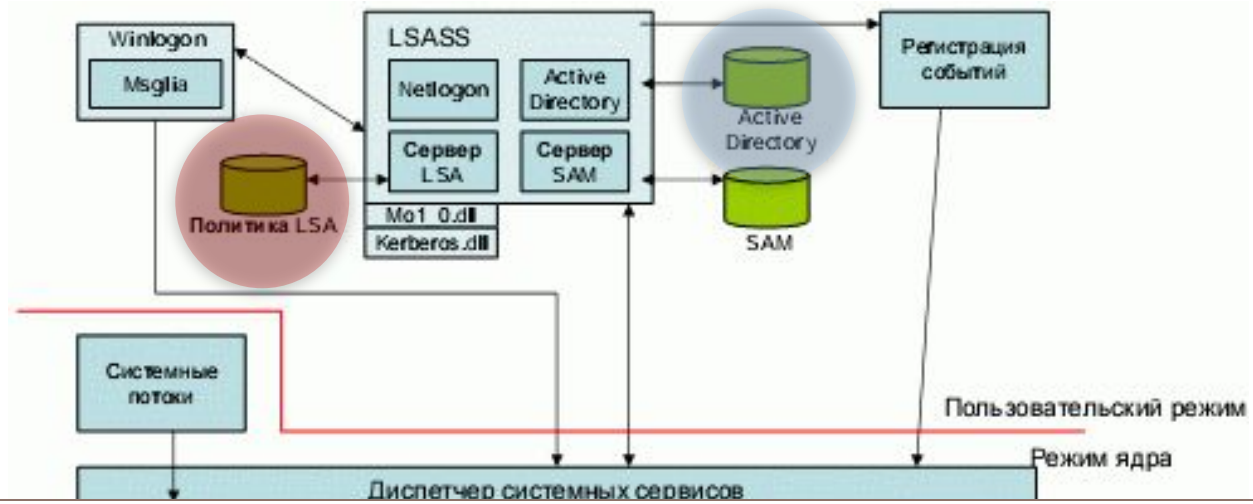
- LSASS отвечает за политику безопасности в локальной системе (например, круг пользователей, имеющих право на вход в систему, правила, связанные с паролями, привилегии, выдаваемые пользователям и их группам, параметры аудита безопасности системы), а также за аутентификацию пользователей и передачу сообщений аудита безопасности в журнал событий.

Подробная схема системы защиты



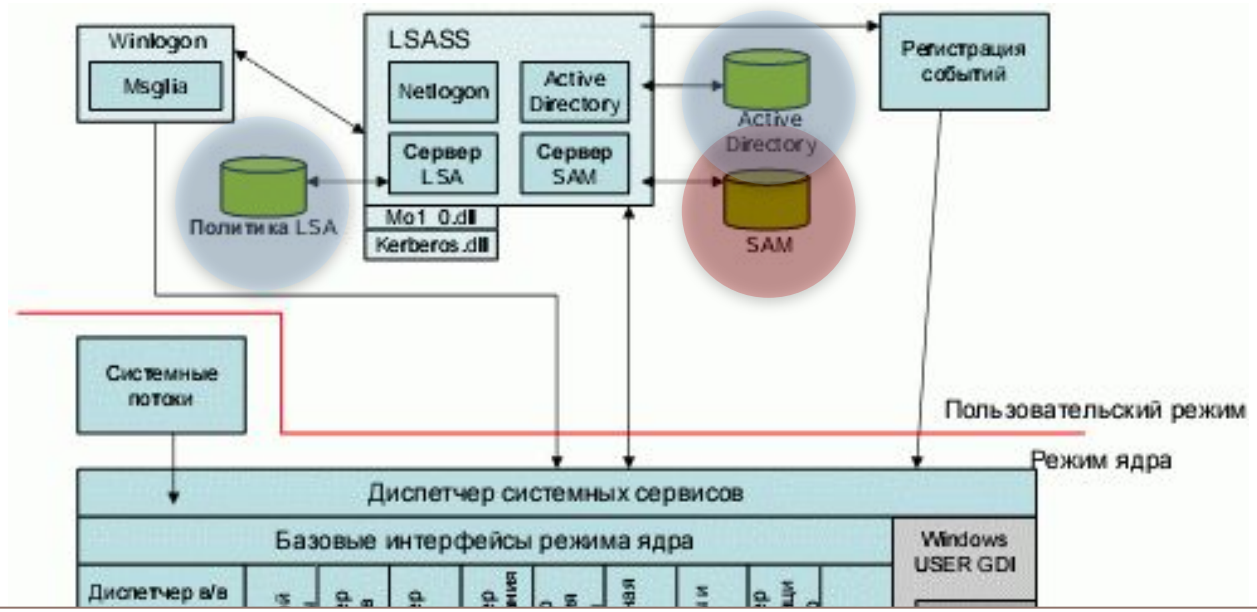
- Active Directory – служба каталогов, содержащая базу данных со сведениями об объектах домена, в том числе о пользователях, группах и компьютерах. Сведения о паролях и привилегиях пользователей домена и их групп содержатся в AD.

Подробная схема системы защиты



- База данных LSASS хранится в разделе реестра HKEY_LOCAL_MACHINE\SECURITY и содержит параметры политики безопасности локальной системы: каким доменам доверена аутентификация попыток входа в систему, кто имеет права на доступ к системе и каким образом, кому предоставлены те или иные привилегии и какие виды аудита следует выполнять.

Подробная схема системы защиты



- Диспетчер учетных записей безопасности SAM (Security Accounts Manager) – набор подпрограмм, отвечающих за поддержку базы данных, которая содержит имена пользователей и группы, определенные на локальной машине.

Архитектура ОС MS Windows 2000+

Объекты Windows

Объекты Windows

Общие сведения

Объекты Windows

- В ОС Windows объект – это отдельный экземпляр периода выполнения (runtime instance) статически определенного типа объекта.
- Не все структуры данных в Windows являются объектами. В объекты помещаются лишь те данные, которые нужно разделять, защищать, именовать или сделать доступными программам пользовательского режима (через системные сервисы).
- Структуры, используемые только одним из компонентов операционной системы для поддержки каких-то внутренних функций, к объектам не относятся.
- Управление объектами возложено на менеджер объектов.

Вопрос

- Где расположен менеджер объектов в архитектуре MS Windows ?

Основные функции объектов

- присвоения понятных имен системным ресурсам;
- разделение ресурсов и данных между процессами;
- защита ресурсов от несанкционированного доступа;
- учет квот ресурсов.

Многообразие объектов

MS Windows 2000 –
27 объектов

MS Windows XP-2003 –
29 объектов

Типы объектов

- **Объекты исполнительной системы** (executive object) представляются различными компонентами исполнительной системы. Они доступны программам пользовательского режима (защищенным подсистемам) посредством базовых сервисов и могут создаваться и использоваться как подсистемами, так и исполнительной системой.
- **Объекты ядра** (kernel object) – это более примитивный набор объектов, реализованный ядром. Большинство этих объектов создаются и используются только внутри исполнительной системы.

Примеры объектов

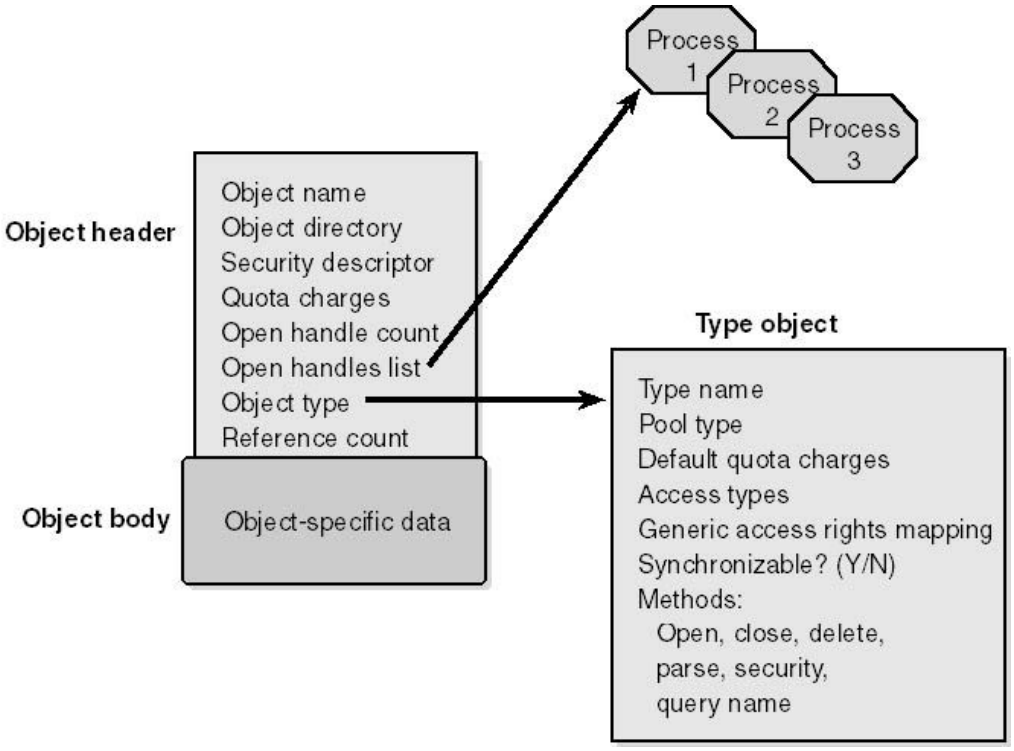
- Файл (File)
- Символическая связь (SymbolicLink)
- Коммуникационное устройство (Communications device)
- Рабочий стол (Desktop)
- Окно (Window)
- Меню (Menu)
- Палитра (Palette)
- Процесс (Process)
- Поток (Thread)
- Маркер доступа (Access token)
- Куча (Heap)
- Событие (Event)
- Семафор (Semaphore)
- Таймер (Timer)
- Мьютекс (Mutex)
- Сокет (Socket)

Объекты Windows

Структура объектов

Структура объекта

- У каждого объекта есть заголовок и тело.
- Диспетчер объектов управляет заголовками объектов, а телами объектов управляют владеющие ими компоненты исполнительной системы.
- Каждый заголовок объекта содержит список стандартных атрибутов.



Стандартные атрибуты объектов

Имя объекта	Делает объект видимым другим процессам для совместного использования.
Каталог объектов	Обеспечивает иерархическую структуру, в которой хранятся имена объектов.
Дескриптор безопасности	Определяет, кто и каким образом может использовать данный объект.
Расход квоты	Задает квоту на использование ресурсов, которая списывается с процесса при открытии дескриптора данного объекта.
Счетчик открытых дескрипторов	Подсчитывает количество открытых дескрипторов данного объекта.
Список открытых дескрипторов	Содержит список процессов, открывших дескрипторы данного объекта.
Временный/ постоянный статус	Указывает, можно ли уничтожить имя и освободить память объекта, если он более не используется.
Режим: пользовательский/ ядра	Определяет доступность объекта в пользовательском режиме.
Указатель на типовой объект	Ссылается на типовой объект, который содержит атрибуты, общие для набора однотипных объектов.

Базовые сервисы объектов

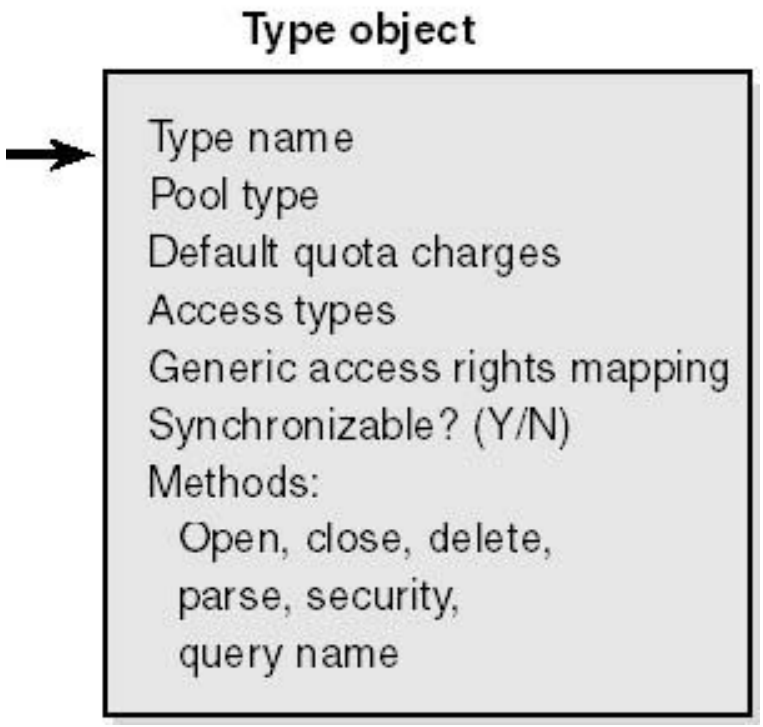
- Диспетчер объектов предоставляет небольшой набор базовых сервисов, которые работают с атрибутами заголовка объекта и применимы к объектам любого типа (хотя некоторые базовые сервисы не имеют смысла для отдельных объектов).
- Базовые сервисы поддерживаются для всех типов объектов, но у каждого объекта есть свои сервисы для создания, открытия и запроса. Так, подсистема ввода-вывода реализует сервис создания файлов для объектов «файл», а диспетчер процессов – сервис создания процессов для объектов «процесс».

Перечень базовых сервисов объектов

Сервис	Описание
Закрытие (close)	Закрывает описатель объекта
Дублирование (duplicate)	Позволяет совместно использовать объект за счет дублирования его описателя и передачи его другому процессу
Запрос объекта (query object)	Сообщает информацию о стандартных атрибутах объекта
Запрос защиты (query security)	Сообщает дескриптор защиты объекта
Установка защиты (set security)	Изменяет защиту объекта
Ожидание одного объекта (wait for a single object)	Синхронизирует выполнение потока с одним объектом
Ожидание нескольких объектов (wait for multiple objects)	Синхронизирует выполнение потока с несколькими объектами

Атрибуты типового объекта

- имя объекта
- тип пула – тип памяти выделяемой для объекта (подкачиваемая, неподкачиваемая)
- квота по умолчанию
- виды доступа
- базовые права доступа
- поддержка объектом синхронизации
- методы объекта



Объекты Windows

Управление объектами

Отслеживание открытых дескрипторов

- Так как все процессы пользовательского режима, осуществляющие доступ к некоторому объекту, должны вначале открыть его дескриптор, то диспетчер объектов может отслеживать, сколько процессов, и какие именно, используют данный объект.
- Отслеживание открытых дескрипторов – 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

- Первая фаза называется **удержанием имени** (name retention) и управляется количеством открытых дескрипторов данного объекта. Всякий раз, когда процесс открывает дескриптор объекта, диспетчер объектов увеличивает счетчик открытых дескрипторов в заголовке объекта. После того, как процесс закончил работу с объектом и закрыл имеющиеся у него дескрипторы данного объекта, диспетчер объектов уменьшает счетчик.
- Вторая фаза удержания объектов – это **прекращение удержания** (т.е. удаление объектов), когда они более не используются. Когда счетчик ссылок обнуляется, диспетчер объектов удаляет объект из памяти.

Учет использования ресурсов

- Каждому пользователю назначаются **предельные размеры квот**, ограничивающие суммарный объем системной памяти, который может быть использован его процессами. Соответственно, заголовок каждого объекта содержит атрибут, называемый «расход квоты» и содержащий значение, которое диспетчер объектов вычитает из выделенной процессу квоты, когда поток этого процесса открывает дескриптор данного объекта.

Объекты Windows

Защита объектов

Виды контроля доступа к объектам

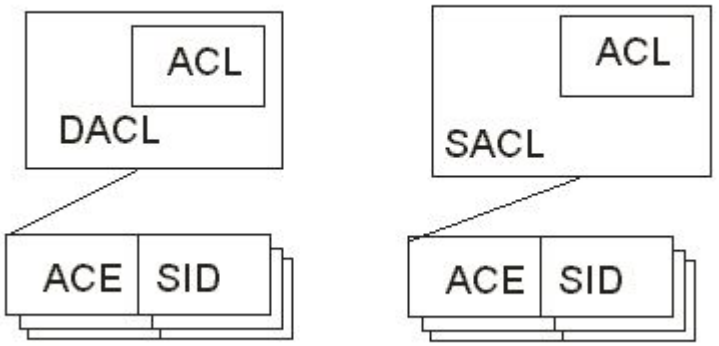
- **управление избирательным доступом** (discretionary access control) – основной механизм контроля доступа, при котором владельцы объектов разрешают или запрещают доступ к ним для других пользователей (процессов). При первой попытке доступа процесса (приложения) к общему (разделяемому) объекту подсистема Windows проверяет, имеет ли это приложение соответствующие права. Если проверка завершается успешно, подсистема Windows разрешает приложению доступ.
- **управление привилегированным доступом** (privileged access control) – необходим в тех случаях, когда управления избирательным доступом недостаточно. Данный метод гарантирует, что пользователь сможет обратиться к защищенным объектам, даже если их владелец недоступен.

Избирательный доступ

- Избирательный доступ основан на списках контроля доступа (access control list, ACL), которые описывают каким пользователям какие операции можно выполнять. При отсутствии ACL объект является незащищенным.
- ACL представляет собой список элементов контроля доступа (access control element, ACE).
- Когда процесс пытается использовать какой-либо объект, система проводит просмотр **всех** ACE в ACL. В ходе просмотра выполняется сравнение маски доступа с проверяемыми правами.
- В списке ACL есть записи ACE двух типов – разрешающие и запрещающие доступ. Для запрещающих ACE даже при частичном совпадении прав доступ немедленно отклоняется. Для успешной проверки разрешающих элементов необходимо совпадение всех прав. Поэтому для ускорения рекомендуется размещать запрещающие элементы перед разрешающими.

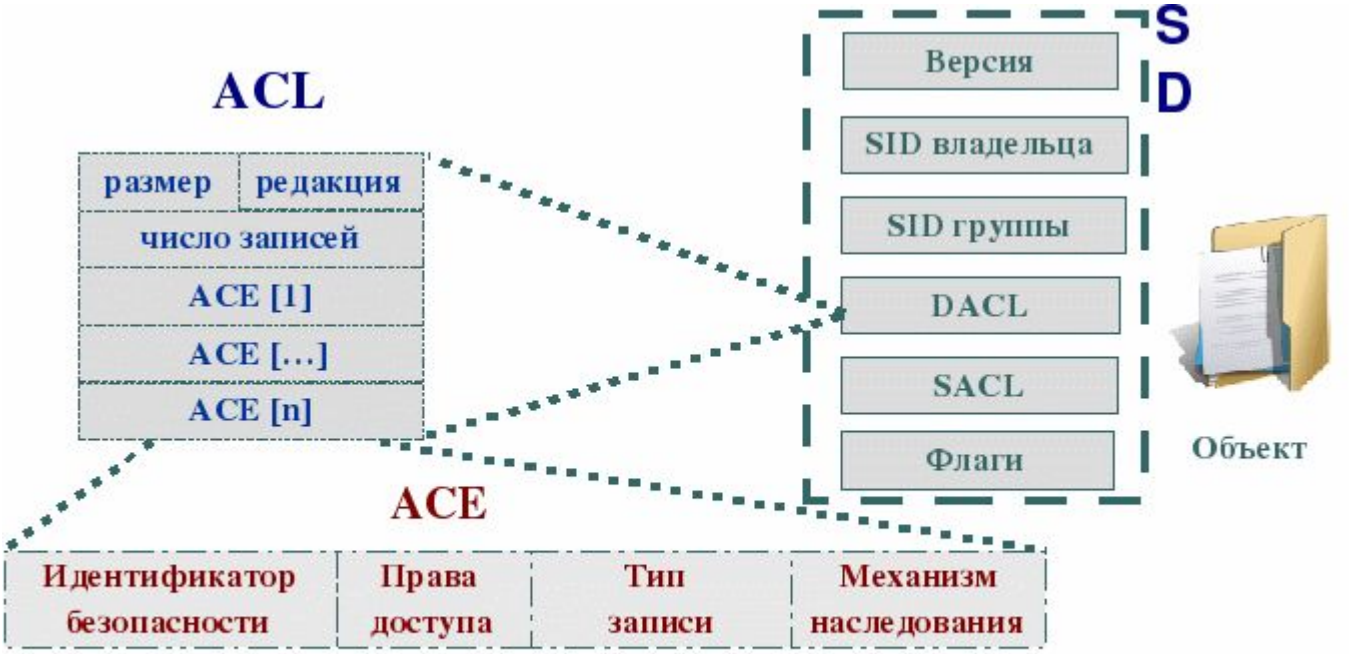
Списки SACL и DACL

- Для каждого защищаемого объекта есть два списка ACL:
  - **DACL** – указывает пользователей или группы к которым разрешен доступ к объектам;
  - **SACL** – список управления доступом к объектам Windows, используемый для аудита доступа к объекту, указывает какие операции и каких пользователей должны регистрироваться в журнале аудита безопасности

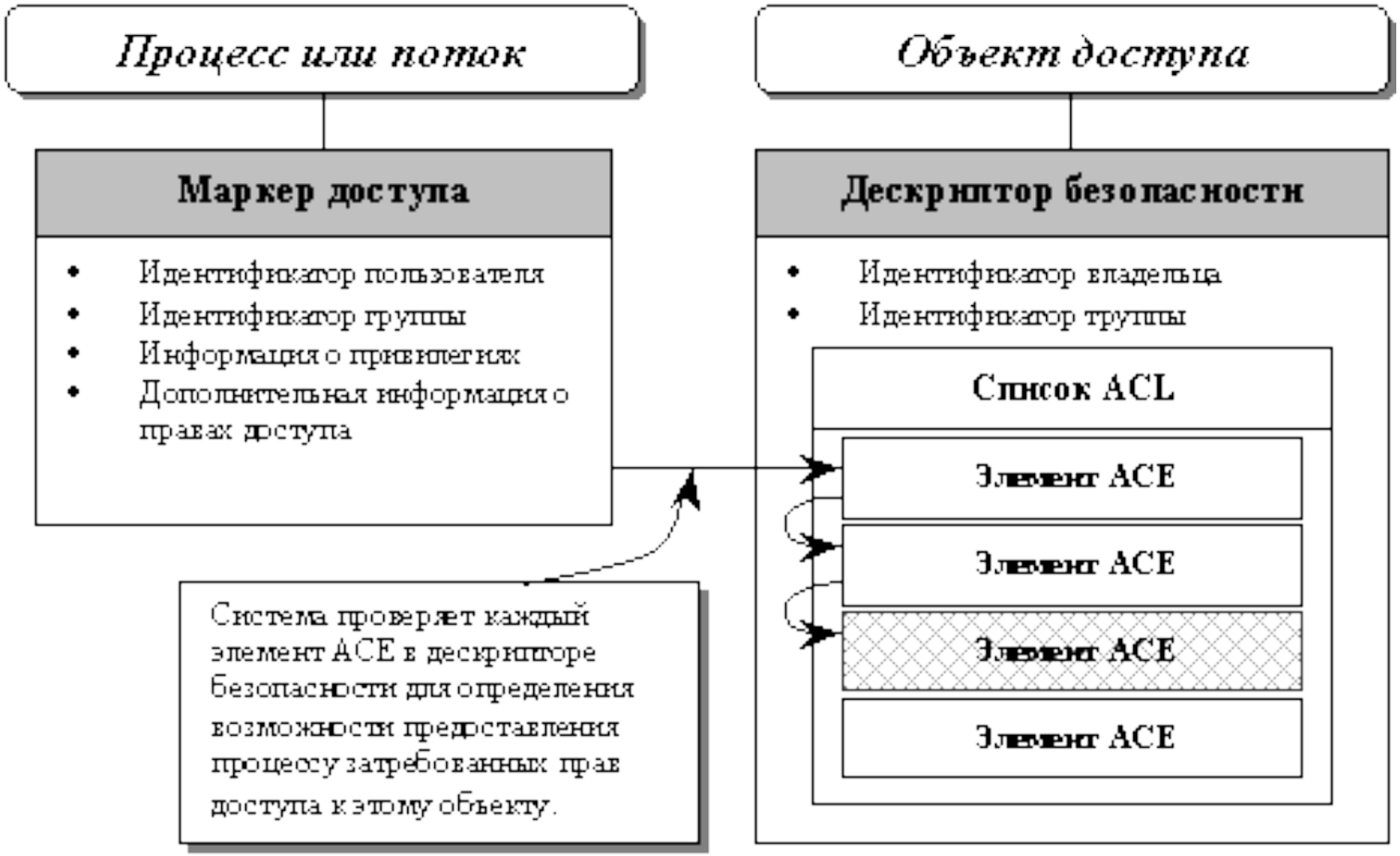


Дескриптор безопасности

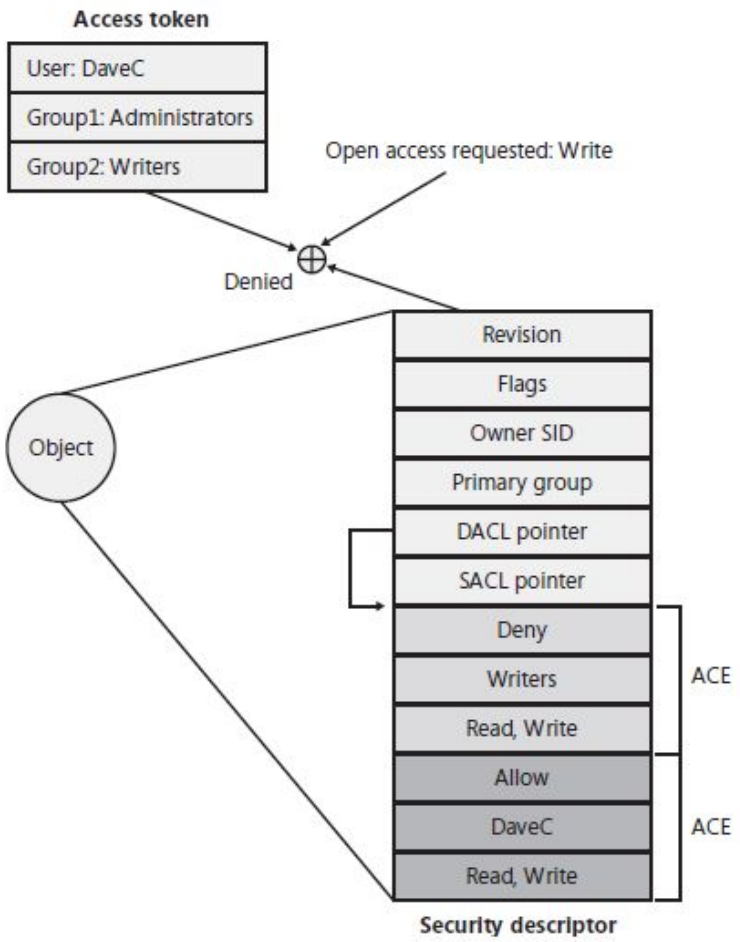
- Дескриптор безопасности – это специальная структура, которая хранит информацию о безопасности объекта.



Отношения между маркером доступа и атрибутами безопасности объекта



Пример проверки ACE



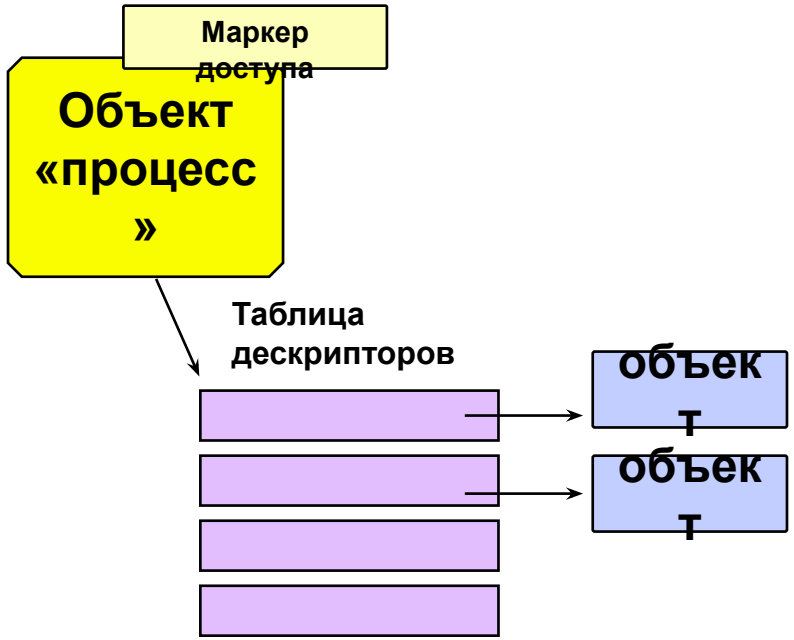
- Пример проверки прав доступа, демонстрирующий, насколько важен порядок ACE.
- В этом примере пользователю отказано в доступе к файлу, хотя ACE в DACL объекта предоставляет такое право.

Объекты Windows

Таблицы дескрипторов объектов

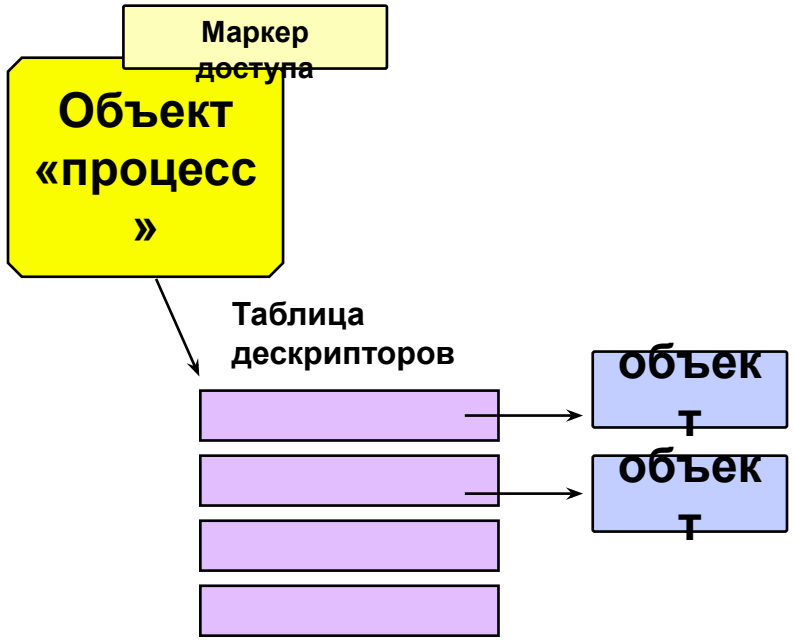
Процессы и таблицы дескрипторов объектов

- Каждый процесс имеет таблицу объектов доступных ему для использования.
- Для получения доступа к объекту процесс должен использовать дескриптор этого объекта (индекс элемента таблицы).
- Дескриптор объекта предоставляется Windows после создания нового (или открытия уже существующего) объекта.



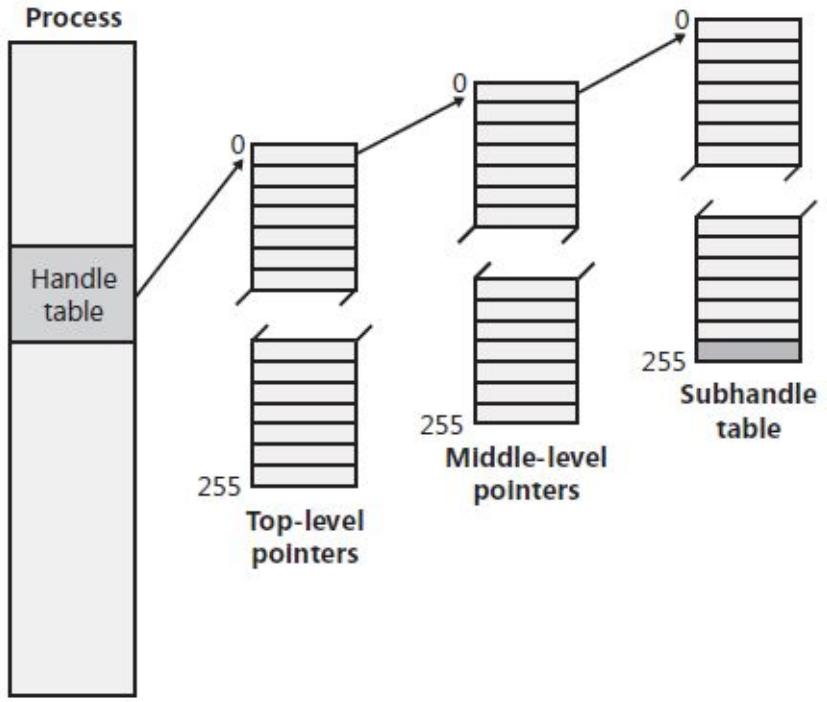
Ограничения на количество дескрипторов

- Некоторые объекты одновременно поддерживают только один дескриптор. Другие объекты поддерживают многочисленные дескрипторы для единственного объекта.
- Общее количество открытых дескрипторов в системе ограничено только доступным объемом памяти.

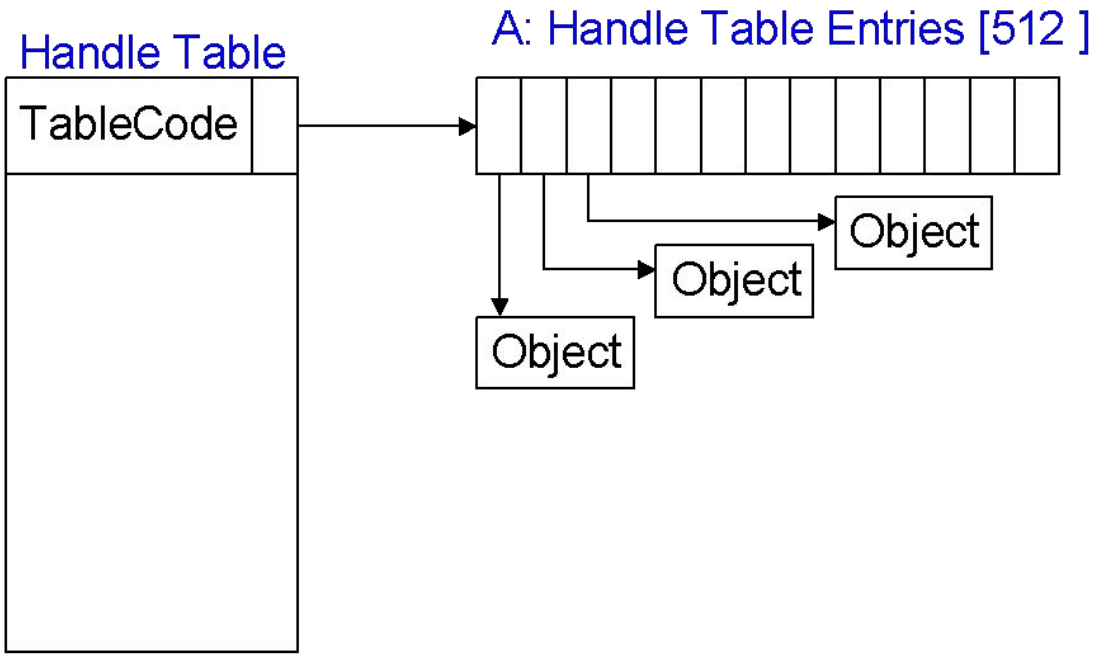


Размер таблицы дескрипторов

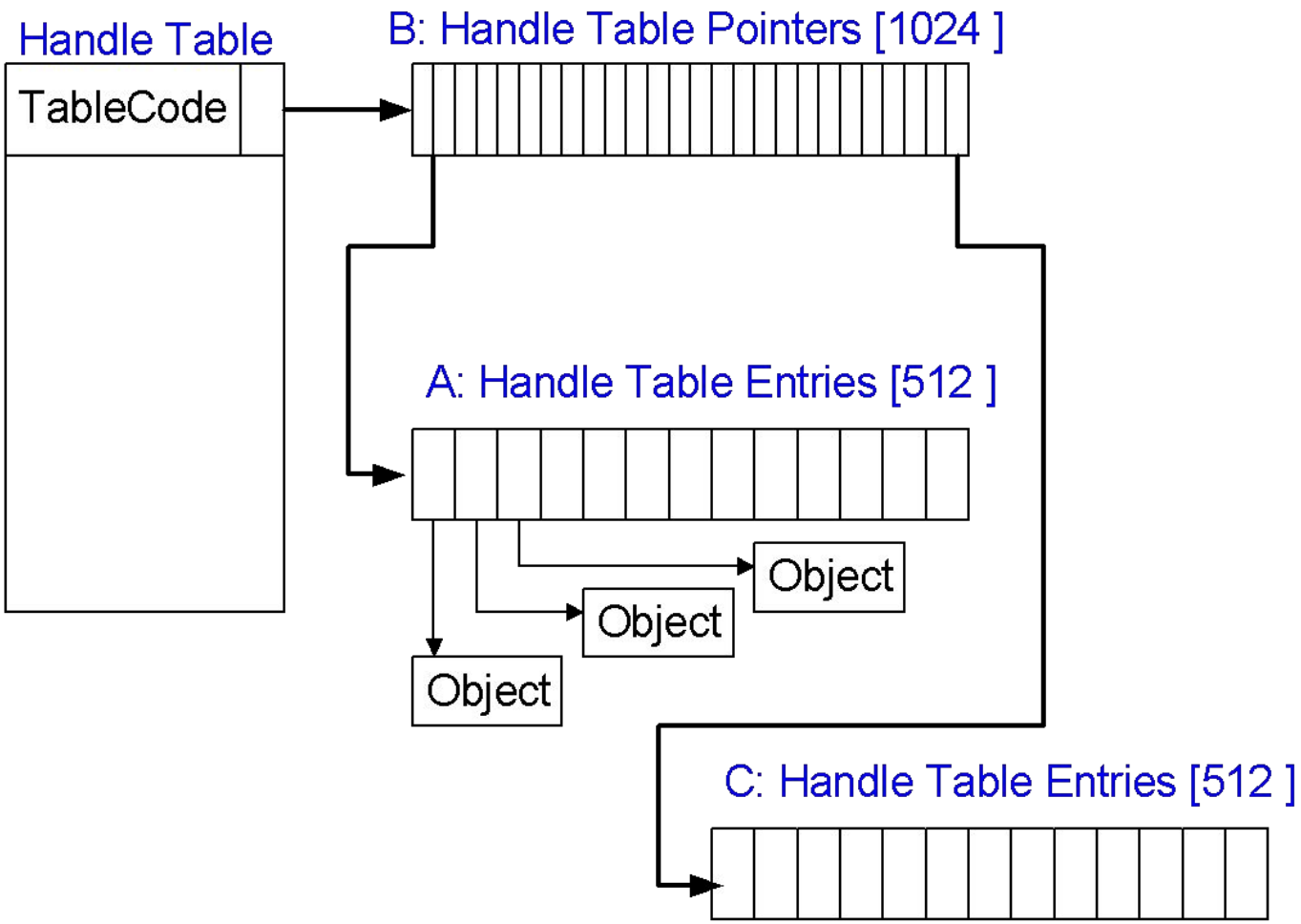
- Изначально процессу выделяет таблица на 255 дескрипторов.
- Если процессу не хватает размера таблицы, то реализуется двухуровневая схема, которая обеспечивает хранение до 512К дескрипторов.
- Если процессу не хватает двухуровневой таблицы, то реализуется трехуровневая схема, поддерживающая до 16М дескрипторов.



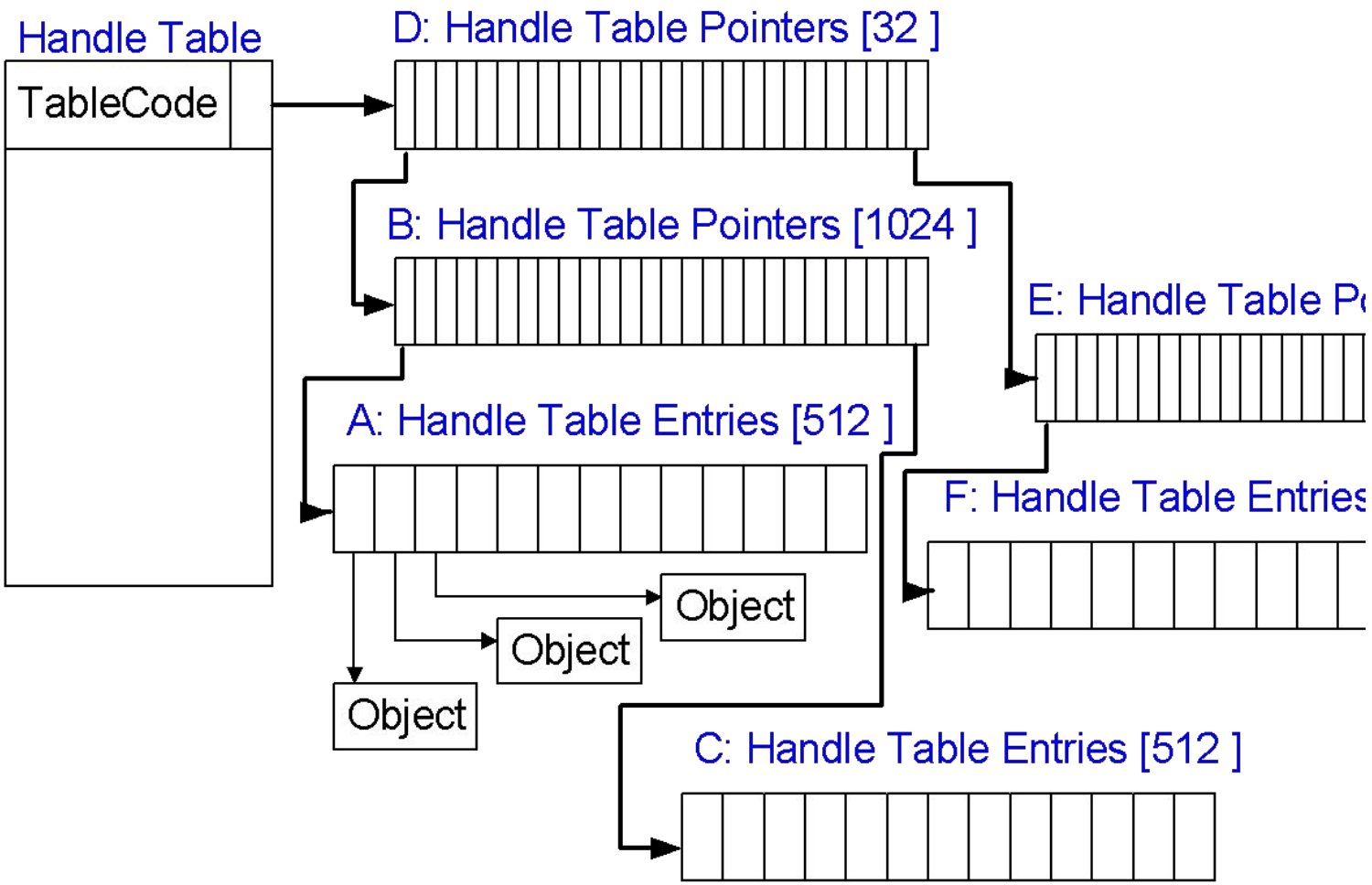
Одноуровневая таблица (до 512 дескрипторов)



Двухуровневая таблица (до 512К дескрипторов)

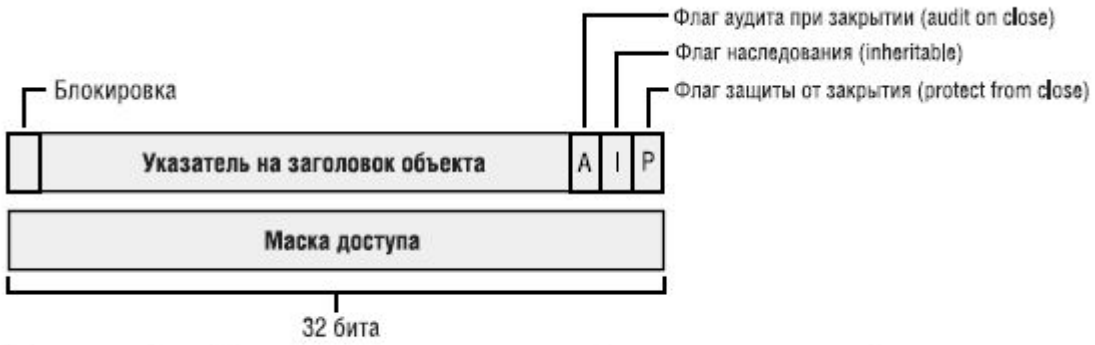


Трехуровневая таблица (до 16M дескрипторов)



Элементы таблицы дескрипторов

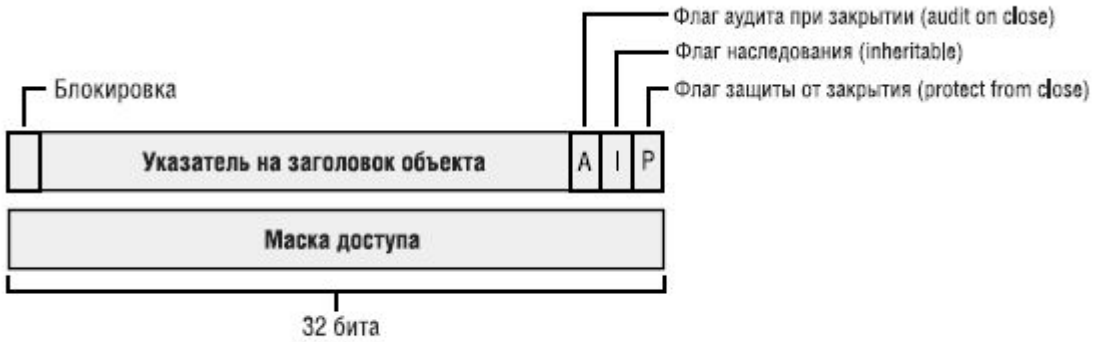
- В x86-системах каждый элемент таблицы дескрипторов состоит из структуры с двумя 32-битными элементами: указателем на объект (с флагами) и маской доступа.



- В 64-разрядных системах каждый элемент имеет размер 12 байтов и состоит из 64-битного указателя на заголовок объекта и 32-битной маски доступа.

Формат записи элемента таблицы дескрипторов

- старший бит является флагом блокировки (lock), когда диспетчер объектов транслирует описатель в указатель на объект, он блокирует соответствующую запись на время трансляции;



- флаг **P** указывает, имеет ли право вызывающая программа закрывать данный дескриптор;
- флаг **I** определяет, смогут ли процессы, созданные данным процессом, получить копию этого дескриптора;
- флаг **A** задает, будет ли генерироваться сообщение аудита при закрытии объекта.

Объекты Windows

Работа с объектами в Win32 API

Получение дескриптора объекта

- Создание объекта
- Открытие дескриптора существующего объекта (по имени объекта)
- Наследование дескриптора от родительского процесса
- Дублирование дескриптора процесса

Создание объекта

- Для каждого типа объекта предусмотрена собственная функция создания, например:
  - *CreateProcess ()*
  - *CreateEvent ()*
  - *CreateFileMapping ()*
- Обязательным параметром любой функции создания объекта является адрес на структуру SECURITY_ATTRIBUTES, описывающую атрибуты безопасности нового объекта.
- Большинство создаваемых объектов Windows являются именованными.

Пример функции создания объекта «событие»

```
HANDLE CreateEvent (
    LPSECURITY_ATTRIBUTES lpEventAttributes, // атрибуты защиты
    BOOL bManualReset, // тип сброса TRUE – ручной
    BOOL bInitialState, // начальное состояние TRUE – сигнальное
    LPCTSTR lpName // имя обьекта
);
```

Создание объекта «событие»

- Функция *CreateEvent ()* создает объект типа «событие» для вызвавшего это функцию процесса и возвращает дескриптор объекта.

- Если с таким именем событие уже создано, то вернется дескриптор уже созданного события, а *GetLastError()* вернет код **ERROR_ALREADY_EXISTS**.
- Если объект не удалось создать, функция вернет NULL.

Структура SECURITY_ATTRIBUTES

```
typedef struct _SECURITY_ATTRIBUTES { // sa
    DWORD nLength;          // размер структуры в байтах
    LPVOID lpSecurityDescriptor;      // адрес структуры дескриптора
                                      // безопасности
    BOOL bInheritHandle;          // флаг наследования
} SECURITY_ATTRIBUTES;
```

- Если *lpSecurityDescriptor* = NULL, то для объекта назначается дескриптор безопасности по умолчанию для текущего процесса.
- *bInheritHandle* – флаг, разрешающий наследование объекта дочерним процессом. Если равен TRUE, новый процесс наследует дескриптор.

Пример инициализации структуры SECURITY_ATTRIBUTES

```
SECURITY_ATTRIBUTES sa =
   { sizeof (SECURITY_ATTRIBUTES), NULL, TRUE };
```

- sa.nLength = sizeof (SECURITY_ATTRIBUTES);
- sa.lpSecurityDescriptor = NULL;
- sa.bInheritHandle = TRUE;

Открытие именованного объекта

- Практически для каждого именованного объекта есть функция, которая позволяет открыть уже созданный объект:
  - *OpenProcess ()*
  - *OpenEvent ()*
  - *OpenFileMapping ()*
- Открыть уже созданный объект может «другой» процесс, чтобы совместно использовать этот объект с процессом-владельцем, либо сам процесс-владелец, чтобы иметь несколько дескрипторов с разными правами доступа.
- Для разрешения доступа «другого» процесса к объекту процесса-владельца выполняется проверка отношений между дескриптором безопасности объекта и маркером доступа «другого» процесс.

Функция открытия объекта «событие»

```
HANDLE OpenEvent (
    DWORD dwDesiredAccess, // режим доступа к объекту
    BOOL bInheritHandle, // флаг наследования
    LPCTSTR lpName // имя обьекта
);
```

- **EVENT_ALL_ACCESS** – устанавливает все возможные флаги доступа к объекту события;
- **EVENT_MODIFY_STATE** – обеспечивает возможность использования дескриптора в функциях, изменяющих состояние объекта события;
- **SYNCHRONIZE** – позволяет использовать дескриптор в любой из функций ожидания для задания состояния объекта.

Описание функции открытия объекта «событие»

- Функция *OpenEvent ()* открывает ранее созданный объект типа «событие» с установленными параметрами доступа и возвращает дескриптор объекта.

- Если объект не удалось открыть, функция вернет **NULL**.

Переустановка свойств объекта

```
BOOL SetHandleInformation (
    HANDLE hObject, // дескриптор объекта
    DWORD dwMask, // определяет изменяемые флаги
    DWORD dwFlags  // задает новые значения флагов
);
```

- флаг **HANDLE_FLAG_INHERIT** – разрешение наследования дескриптора дочерним процессом;
- флаг **HANDLE_FLAG_PROTECT_FROM_CLOSE** – запрет на закрытие дескриптора, если установлен этот флаг, то вызов функции *CloseHandle()* не закрывает дескриптор объекта.

Наследование дескриптора

- Дочерний процесс может наследовать дескрипторы объектов своего родительского процесса. Унаследованный дескриптор правилен только применительно к дочернему процессу.
- Для того, что дать возможность дочернему процессу наследовать открытые дескрипторы его родительского процесса, выполните:
  - установите у наследуемого объекта флаг разрешения наследования в TRUE;
  - создайте дочерний процесс, используя функцию *CreateProcess ()* с параметром *bInheritHandles* = TRUE.
- Передайте дескриптор объекта дочернему процессу (будет рассмотрено в следующих разделах).

Функция получения дубликата дескриптора

```
BOOL DuplicateHandle(
    HANDLE hSourceProcessHandle,
    HANDLE hSourceHandle,
    HANDLE hTargetProcessHandle,
    LPHANDLE lphTargetHandle,
    DWORD dwDesiredAccess,
    BOOL bInheritHandle,
    DWORD dwOptions
);
```

Параметры функции *DuplicateHandle*

- *hSourceProcessHandle* – дескриптор процесс-владельца объекта;
- *hSourceHandle* – дескриптор-оригинал объекта;
- *hTargetProcessHandle* – дескриптор процесс-получателя объекта;
- *lphTargetHandle* – адрес переменной типа HANDLE, по которому возвращается дескриптор-дубликат объекта;
- *dwDesiredAccess и bInheritHandle* – маска доступа и флаг разрешения наследования, устанавливаемые для данного дескриптора в процессе-получателе;
- *dwOptions* может быть 0 или любой комбинацией двух флагов:
  - DUPLICATE_SAME_ACCESS – у дескриптора-дубликата должна быть та же маска доступа, что и у дескриптора-оригинала, этот флаг заставляет *DuplicateHandle ()* игнорировать параметр *dwDesiredAccess*;
  - DUPLICATE_CLOSE_SOURCE – закрытие дескриптора в процессе-владельце объета (при этом счетчик объекта не меняется).

Инициатор дублирования

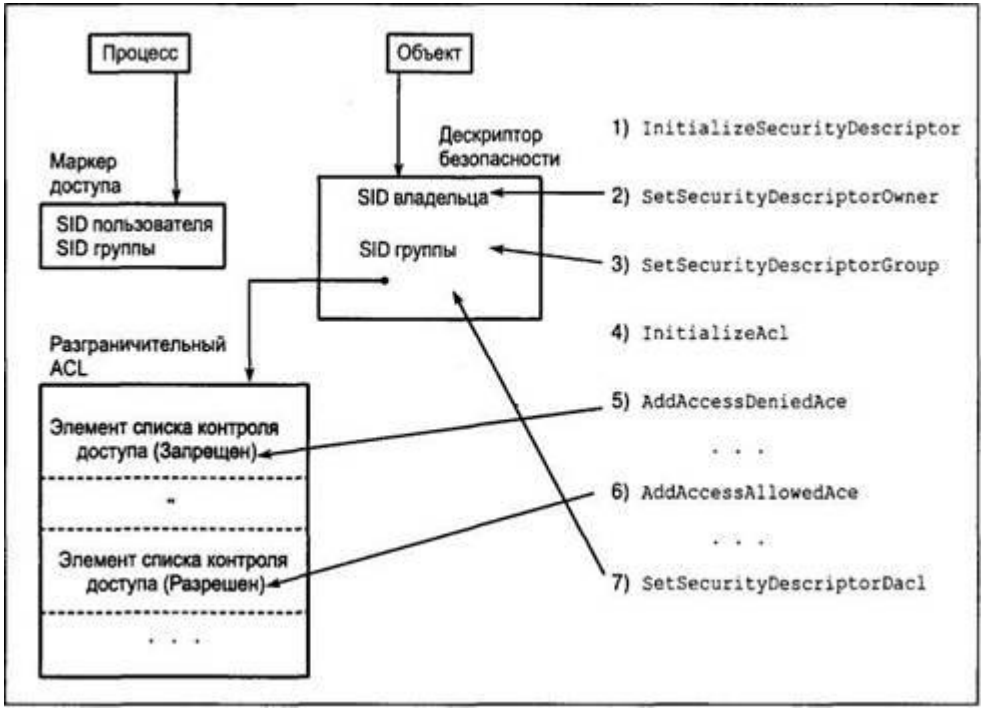
- **Вариант 1:** процесс-владелец передает дескриптор-оригинал другому процессу, процесс-получатель вызывает *DuplicateHandle()* и использует дескриптор-дубликат.
- **Вариант 2:** процесс-владелец вызывает *DuplicateHandle()*, получает дескриптор-дубликат и передает его процессу-получателю.

Совместное использование наследования и дублирования

- Очень часто наследование дескрипторов используют совместно с функцией дублирования:
  - родительский процесс создает объект с ненаследуемым дескриптором;
  - родительский процесс делает наследуемый дескриптор-дубликат для самого себя, но с ограниченными правами доступа;
  - создает дочерний процесс и через параметры создания дочернего процесса (например, командную строку) передает дескриптор-дубликат;
  - дескриптор-оригинал родительский процесс использует и/или закрывает.

Порядок задания дескриптора безопасности

1. Инициализация дескриптора безопасности
2. Связывание SID пользователя (группы) с дескриптором безопасности
3. Инициализация списка прав доступа
4. Добавление правил ACE к ACL
5. Связывание ACL с дескриптором безопасности



1. Инициализация дескриптора безопасности

```
BOOL InitializeSecurityDescriptor(
    LPSECURITY_DESCRIPTOR lpSecurityDescriptor, // адрес
      // структуры дескриптора безопасности
    DWORD dwRevision // версия дескриптора безопасности
      // должна быть SECURITY_DESCRIPTOR_REVISION
);
```

2. Связывание SID пользователя с дескриптором безопасности

```
BOOL SetSecurityDescriptorOwner(
    LPSECURITY_DESCRIPTOR pSecurityDescriptor, // адрес
      // структуры дескриптора безопасности
    PSID pOwner, // указатель на SID пользователя, которого мы
      // хотим установить в качестве владельца
    BOOL bOwnerDefaulted // назначение владельца по
      // умолчанию (создатель становится  владельцем)
);
```

3. Связывание SID группы с дескриптором безопасности

```
BOOL SetSecurityDescriptorGroup(
    LPSECURITY_DESCRIPTOR pSecurityDescriptor, // адрес
      // структуры дескриптора безопасности
    PSID pGroup, // указатель на SID группы, которую мы
      // хотим установить в качестве владельца
    BOOL bOwnerDefaulted // назначение группы по умолчанию
);
```

4. Инициализация списка прав доступа

```
BOOL InitializeAcl(
    LPACL lpAcl, // адрес буфера для хранения ACL
    DWORD cbAcl, // размер буфера для хранения ACL
    DWORD dwAclRevision // версия ACL, должно быть ACL_REVISION
) ;
```

- Для хранения ACL обычно вполне достаточно буфера размером 1Кбайт.

5-6. Добавление правил ACE к ACL

```
BOOL AddAccessAllowedAce(
    LPACL lpAcl,
    DWORD dwAclRevision,
    DWORD dwAccessMask,
    PSID lpSid)
);
```

```
BOOL AddAccessDeniedAce(
    LPACL lpAcl,
    DWORD dwAclRevision,
    DWORD dwAccessMask,
    PSID lpSid)
);
```

- Параметр *lpAcl* указывает на структуру ACL, которая была инициализирована функцией *InitializeACL ()*.
- Параметр *dwAclRevision* должен быть **ACL_REVISION**.
- Параметр *lpSid* указывает на SID пользователя или группы.
- Параметр *dwAccessMask* определяет права, которые предоставляются или в которых отказывается указанному SID.

7. Связывание ACL с дескриптором безопасности

```
BOOL SetSecurityDescriptorDacl(
    LPSECURITY_DESCRIPTOR lpSecurityDescriptor, // адрес
      // структуры дескриптора безопасности
    BOOL bDaclPresent, // флаг присутствия ACL
    LPACL lpAcl, // указатель на структуру, хранящую ACL
    BOOL fDaclDefaulted // ACL был получен с помощью
      // механизма по умолчанию
);
```

Параметры *SetSecurityDescriptorDacl*

- Значение параметра *bDaclPresent*, равное TRUE, указывает на то, что в структуре *lpAcl* имеется ACL. Если этот параметр равен FALSE, то следующие два параметра, *lpAcl* и *fDaclDefaulted*, игнорируются. Флаг SE_DACL_PRESENT структуры SECURITY_DESCRIPTOR_CONTROL также устанавливается равным значению этого параметра.
- Значение FALSE параметра *fDaclDefaulted* указывает на то, что ACL был сгенерирован программистом. В противном случае ACL был получен с использованием механизма, принятого по умолчанию.

Получение SID для учетной записи

```
BOOL WINAPI LookupAccountName(
    LPCTSTR lpSystemName, // имя системы
    LPCTSTR lpAccountName, // имя пользователя
    PSID Sid, // указатель на память, куда будет записан SID
    LPDWORD cbSid, // длина буфера для записи SID
    LPTSTR ReferencedDomainName, // имя домена
    LPDWORD cchReferencedDomainName, // длина буфера
      // ReferencedDomainName
    PSID_NAME_USE peUse // переменная типа enum,
      // которая определяет тип учетной записи
);
```

Тип данных SID_NAME_USE

- SidTypeUser – пользовательский SID;
- SidTypeGroup – SID группы;
- SidTypeDomain – SID доменной учетной записи;
- SidTypeAlias – псевдоним;
- SidTypeDeletedAccount – удаленная учетная запись;
- SidTypeInvalid – некорректный тип;
- SidTypeUnknown – тип неизвестен;
- SidTypeComputer – идентификатор компьютера.

Получение учетного имени пользователя процесса

```
BOOL GetUserName(
    LPTSTR lpBuffer, // адрес буфера
    LPDWORD nSize // размер буфера
);
```

Пример настройки атрибутов безопасности

```
SECURITY_ATTRIBUTES sa;
SECURITY_DESCRIPTOR sd;

InitializeSecurityDescriptor(&sd,SECURITY_DESCRIPTOR_REVISION);
SetSecurityDescriptorDacl(&sd, true, NULL, false);
sa.lpSecurityDescriptor = &sd;
sa.nLength = sizeof(SECURITY_ATTRIBUTES);
sa.bInheritHandle = true; //разрешаем наследование дескрипторов
```

Закрытие дескриптора объекта

```
BOOL CloseHandle(
    HANDLE hObject // дескриптор объекта
);
```

- В случае успешного выполнения функции возвращается TRUE, иначе – FALSE.

Вопрос

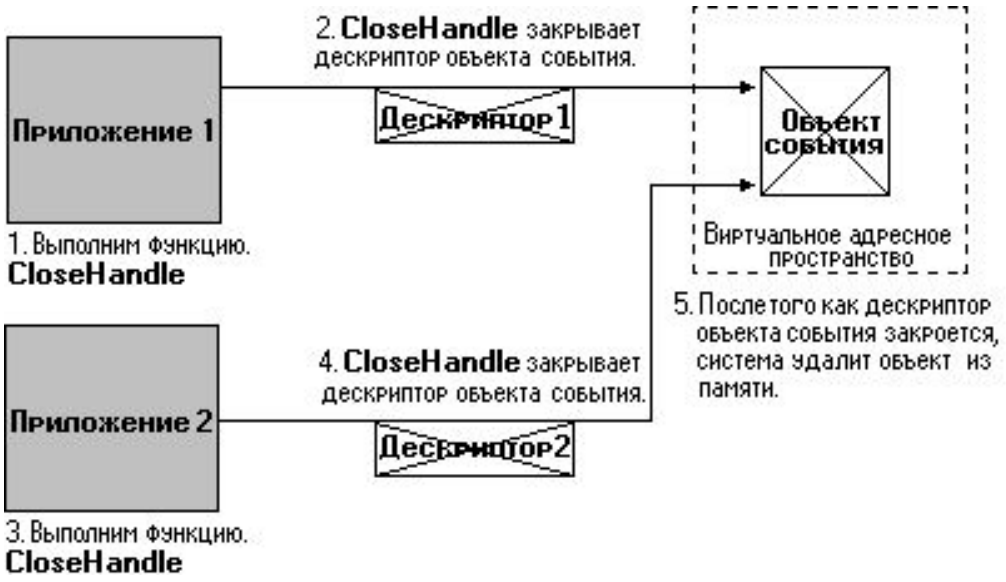
- Что происходит после закрытия дескриптора объекта?

Закрытие дескриптора объекта

```
BOOL CloseHandle(
    HANDLE hObject // дескриптор объекта
);
```

- В случае успешного выполнения функции возвращается TRUE, иначе – FALSE.
- При закрытии дескриптора объекта у объекта уменьшается счетчик числа ссылок.
- После закрытия последнего дескриптора ссылающегося на объект (число ссылок равно 0), объект удаляется из памяти (если только объект не имеет постоянный статус).

Удаление объекта из памяти

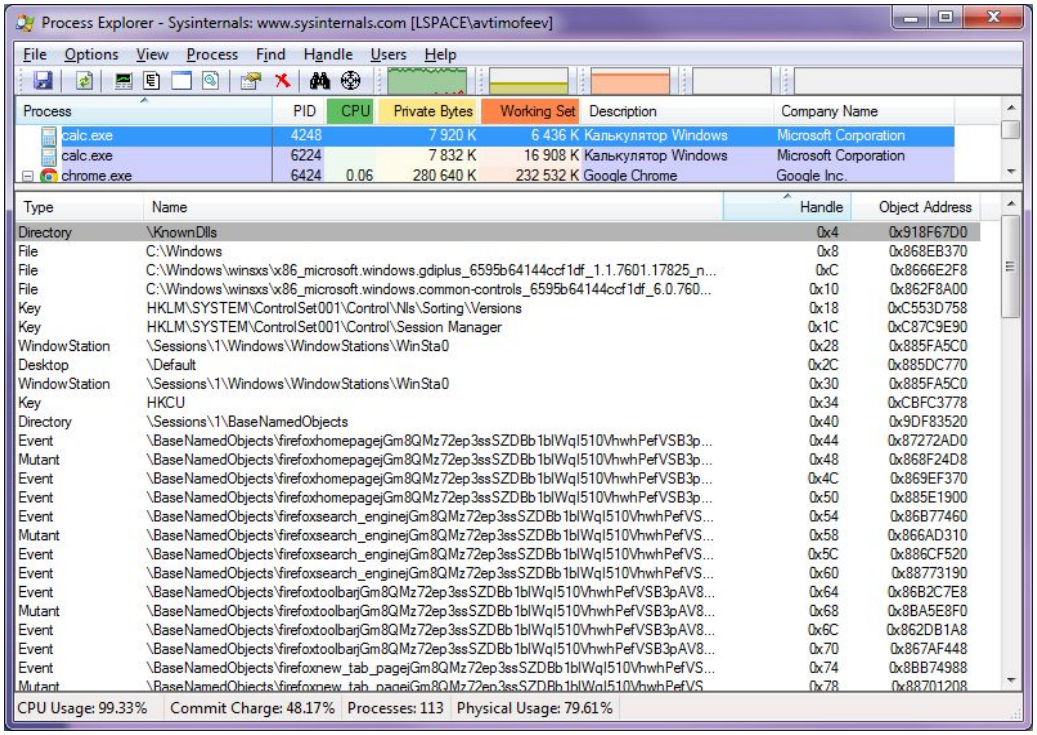


Объекты Windows

Утилиты для работы с объектами Windows

Утилита Process Explorer

- Утилита *Process Explorer* (www.sysinternals.com) позволяет ознакомиться со списком объектов выбранного процесса.



Утилита Handle

- Утилита *Handle* (www.sysinternals.com) позволяет получить сводный список с количеством объектов разного типа, принадлежащих выбранному процессу.