Санкт-Петербургский государственный университет
телекоммуникаций им. проф. М.А.Бонч-Бруевича

СПбГУТ)))

Кафедра автоматизации предприятий связи (АПС)

Тема № 1. Первоначальные сведения о системном программировании

Список литературы по курсу Операционные системы

Основная литература:

1. Болтов, Юрий Федорович. Операционные системы : учеб. пособие (спец. 230102, 230105) : в 2 ч. / Ю. Ф. Болтов, В. Ю. Баженова ; рец. М. О. Колбанев ; Федер. агентство связи, Гос. образовательное учреждение высш. проф. образования "С.- Петерб. гос. ун-т телекоммуникаций им. проф. М. А. Бонч-Бруевича". - СПб. : СПбГУТ, 2008. Ч. 1. - 107 с. : ил., табл. - Библиогр.: с. 104-105. - (в обл.) : 111.38 р.
2. Защита информации в ОС UNIX : учебное пособие / А. В. Красов [и др.] ; рец. С. Е. Душин ; Федеральное агентство связи, Федеральное государственное образовательное бюджетное учреждение высшего профессионального образования "Санкт-Петербургский государственный университет телекоммуникаций им. проф. М. А. Бонч-Бруевича". - СПб. : СПбГУТ. Ч. 1. - 2012. - 71 с. - 226.31 р.
3. Защита информации в ОС UNIX : учебное пособие / А. В. Красов [и др.] ; рец. С. Е. Душин ; Федеральное агентство связи, Федеральное государственное образовательное бюджетное учреждение высшего профессионального образования "Санкт-Петербургский государственный университет телекоммуникаций им. проф. М. А. Бонч-Бруевича". - СПб. : СПбГУТ. Ч. 2. - 2012. - 70 с. - 226.31 р.
4. Мартемьянов, Ю. Ф.
5. Операционные системы. Концепции построения и обеспечения безопасности. Учебное пособие для вузов : [Электронный ресурс] / Ю. Ф. Мартемьянов, Ал. В. Яковлев, Ан. В. Яковлев. - М. : Горячая линия–Телеком, 2010. - 332 с. : ил. - URL: http://ibooks.ru/reading.php?productid=334008. - ISBN 978-5-9912-0128-5 : Б. ц.

Дополнительная литература:

1. Олифер, Виктор Григорьевич.Сетевые операционные системы / В. Г. Олифер, Н. А. Олифер. - СПб. : Питер, 2002. - 544 с. : ил. - ISBN 5-272-00120-6 : 102.00 р., 85.00 р., 76.50 р. - Текст :непосредственный.
2. Болтов, Юрий Федорович. Операционные системы : учеб. пособие (спец. 230102, 230105) : в 2 ч. / Ю. Ф. Болтов, В. Ю. Баженова ; рец. М. О. Колбанев ; Федер. агентство связи, Гос. образовательное учреждение высш. проф. образования "С.- Петерб. гос. ун-т телекоммуникаций им. проф. М. А. Бонч-Бруевича". - СПб. : СПбГУТ, 2008. Ч. 2. - 83 с. : ил., табл. - Библиогр.: с. 82-83. - (в обл.) : 86.63 р.
3. 3Красов, Андрей Владимирович.
4. Защита информации в ОС UNIX : метод. указ. к лаб. работам / А. В. Красов, А. Ю. Цветков, И. А. Федянин ; рец. С. Е. Душин ; Федер. агентство связи, Федер. гос. образовательное бюдж. учреждение высш. проф. образования "С.-Петерб. гос. ун-т телекоммуникаций им. проф. М. А. Бонч-Бруевича". - СПб. : СПбГУТ, 2012. - 28 с. : ил. - Библиогр.: с. 28. - (в обл.) : 62.58 р. - Текст : непосредственный.
5. Таненбаум, Э.Современные операционные системы. 3-е изд. : [Электронный ресурс] / Э. Таненбаум. - Санкт-Петербург : Питер, 2013. - 2120 с. : ил. - URL: http://ibooks.ru/reading.php?productid=344100. - ISBN 978-5-496-00301-8 : Б. ц.
6. Зубков, С. В. Assembler. Для DOS, Windows и Unix : [Электронный ресурс] / С. В. Зубков. - Москва : ДМК Пресс, 2004. - 640 с. : ил. - URL: http://ibooks.ru/reading.php?productid=26627. - ISBN 5-94074-259-9 : Б. ц.
7. Операционные системы : учебное пособие. - Москва : ТУСУР. - URL: http://e.lanbook.com/books/element.php?pl1_cid=25&pl1_id=4971. Ч. 2 / Ю. Б. Гриценко. - Москва : ТУСУР, 2009. - 230 с. - Б. ц. Книга из коллекции ТУСУР - Инженерно-технические науки
8. Операционные системы : учебное пособие. - Москва : ТУСУР. - URL: http://e.lanbook.com/books/element.php?pl1_cid=25&pl1_id=4972. Ч. 1 / Ю. Б. Гриценко. - Москва : ТУСУР, 2009. - 187 с. - Б. ц. Книга из коллекции ТУСУР - Инженерно-технические науки

интерфейс прикладного программирования (Application Programming Interface).

Windows API (англ.application programming nterfaces) — общее наименование целого набора базовых функций интерфейсов программирования приложений операционных систем семейств Microsoft Windows корпорации «Майкрософт». Самый прямой способ взаимодействия приложений с Windows.

API – это набор классов, функций, процедур, констант, при помощи которых, одна программа или приложение, описывает способы взаимодействия с другой программой. При помощи API различные программы получают возможность обмениваться своими ресурсами, возможностями, функциями, информацией.

API-интерфейсы экономят время программистов
Именно благодаря API разработчикам программных приложений, не приходится заново создавать то, что уже существует. Создавая новый программный продукт, они могут дополнять его, используя уже существующие разработки.

Что входит в API и как он работает?

Элементами такого программного интерфейса, являются:

- классы защищенных или открытых данных, элементов кодов
- отдельно прописанные процедуры, представляющие собой автономно работоспособный блок программы;
- функций, подразумевающих работу с переменными
- структуры
- констант (наиболее часто используемая форма данных, настроек или образов);*

- конечная цель продукта обеспечить максимальный комфорт (просто, понятно и эффективно) для пользователя;
- прямая и обратная совместимость между всеми элементами API;
- грамотное проектирование (с учетом всех возможностей предоставляемых интерфейсом) обеспечивающее качественную работу конечной программы с различными устройствами.

API не нужно знать, как работает отдельный модуль. Ведь задачей интерфейса является определение функционала программных элементов для выполнения конкретных задач или обработки данных. Причем взаимодействие используемых разноуровневых компонентов посредством API строится по принципу иерархии. Запросы обрабатываются только между сопряженными элементами.

•Сигнатура – это краткое кодированное описание (указываемое в названии) тех задач, которые способна решить данная функция.

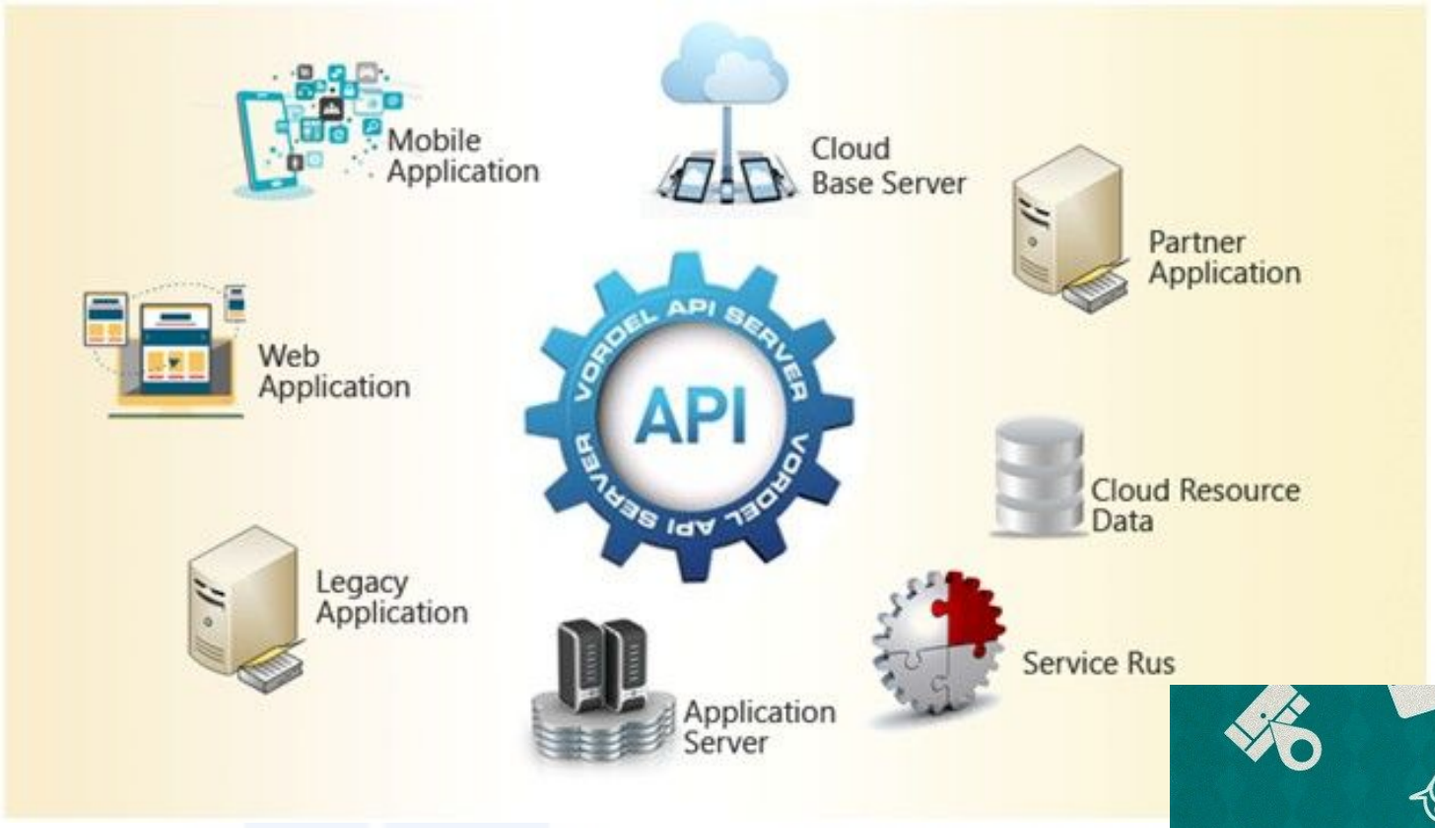
•Семантика – информирует о том, что вы получите, задействовав функцию, и какие данные ей для этого нужно предоставить.

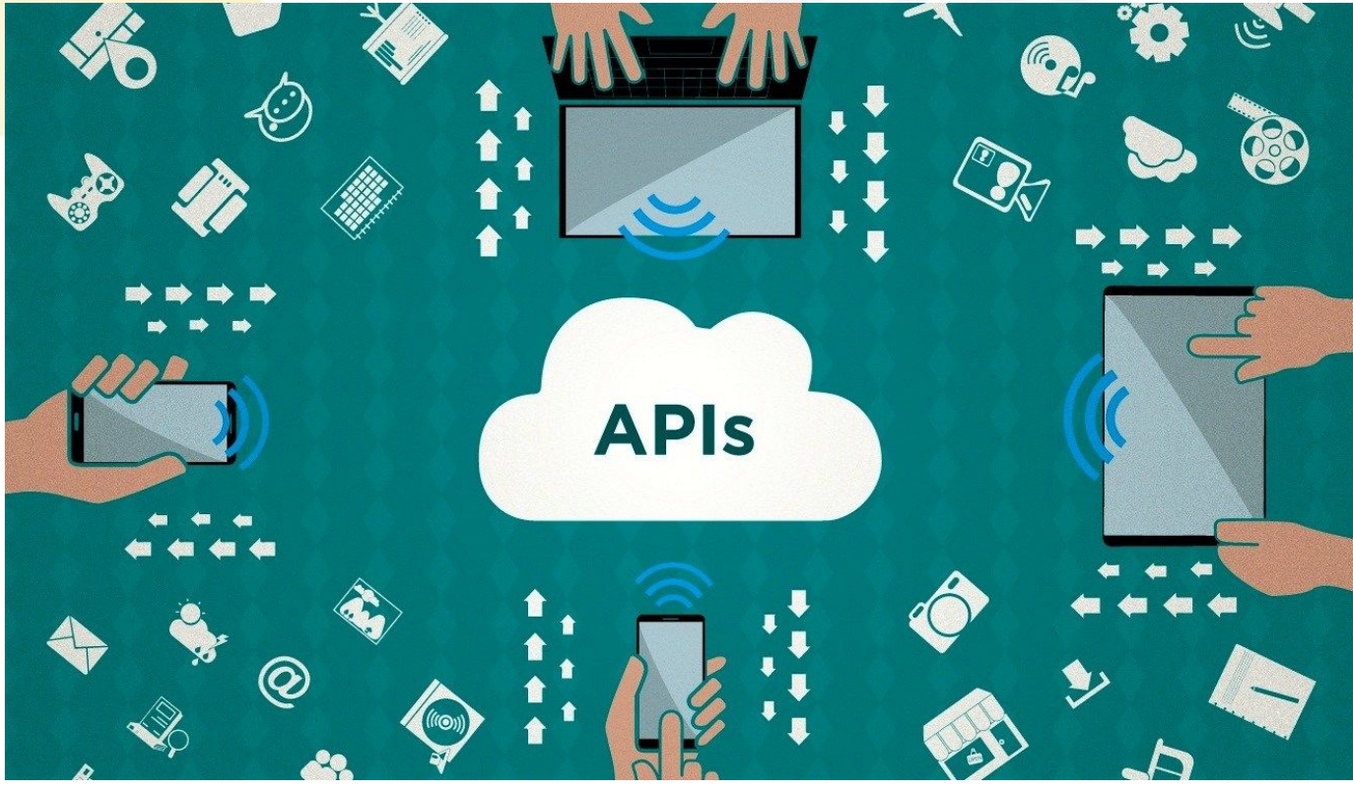
Сферы использования API

Единой API пока не существует, но такие интерфейсы разработаны для отдельных операционных систем:

- Windows API
- Linux Kernel API
- OS/2 API
- Amiga ROM Kernel
- POSIX
- Cocoa
- …….

Примером **API** является **Windows API**, **OpenGL API**, **Direct3D API, DirectX, OpenGL, GDI+, SDL, GTK, Qt** и так далее.


Mobile Application
Cloud Base Server
Partner Application
Web Application
API
VORDEL API SERVER
Cloud Resource Data
Legacy Application
Service Rus
Application Server


APIs

Введение в Windows API

Архитектура ОС Windows

- *Реализация Win32:*
  - kernel32.dll
  - gdi32.dll
  - user32.dll

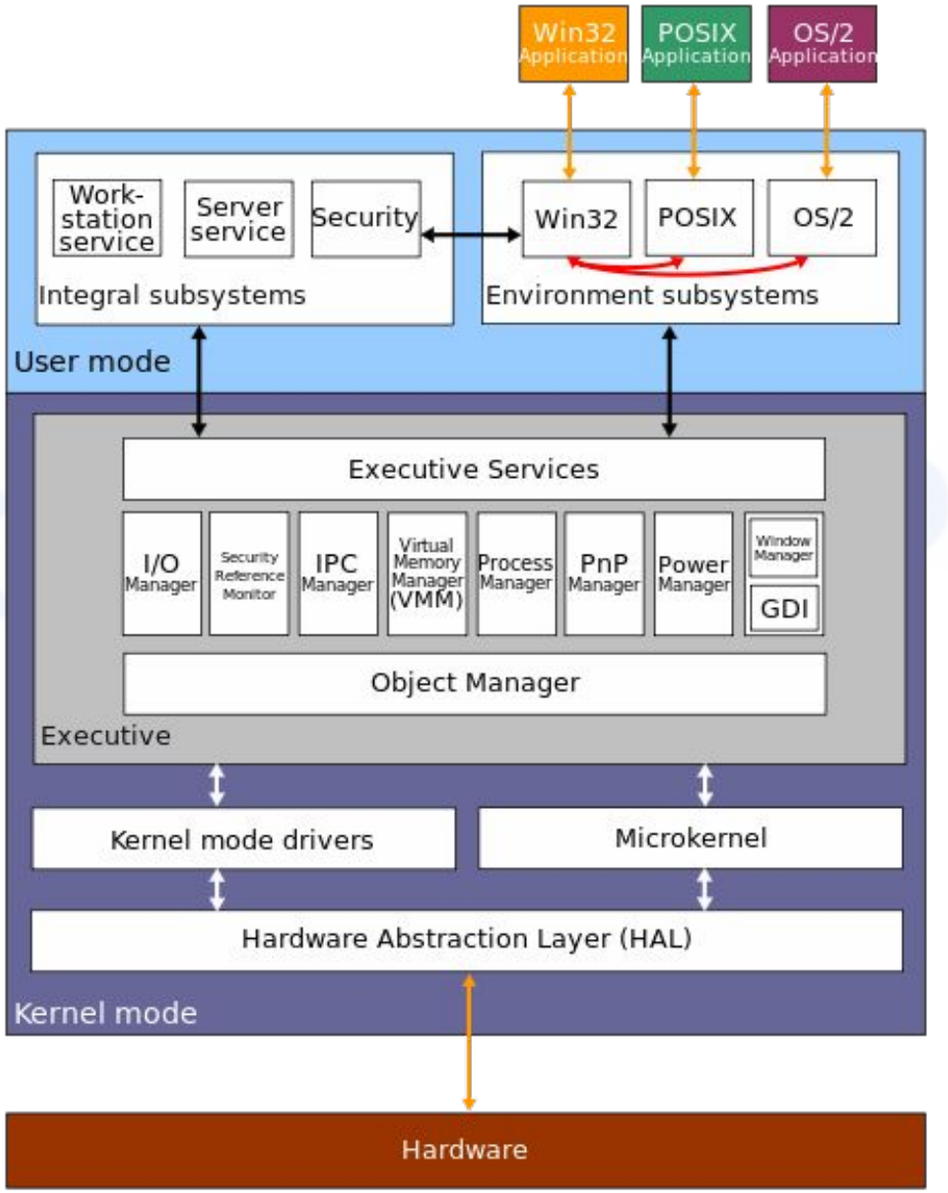
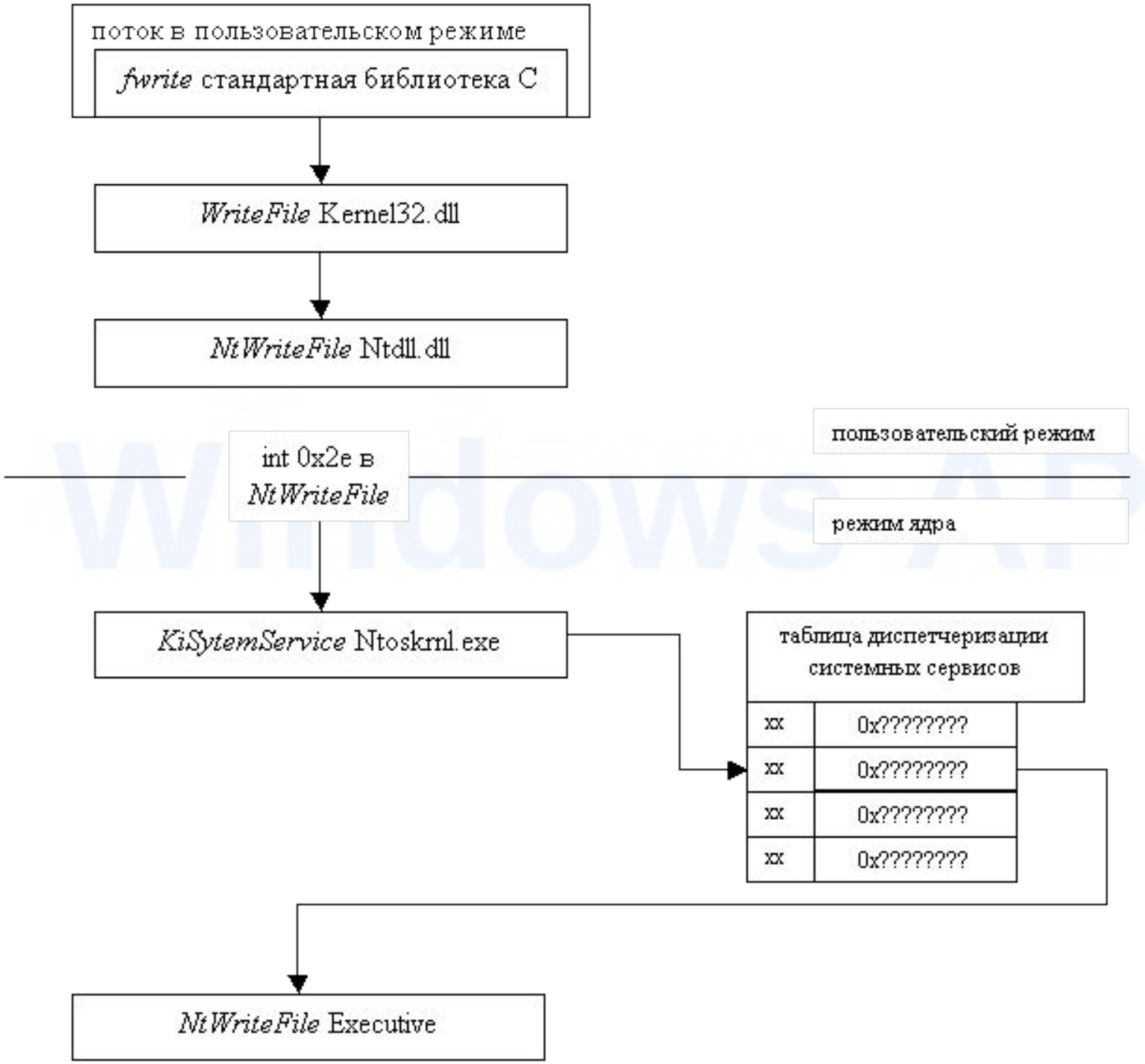


Схема обработки функций ввода/вывода:

Схема обработки системного сервиса:



Раздел MSDN \Platform SDK\ Win32\ Overview of the Win32 API

Win32 API подразделяются на следующие группы.

- **Base Services** - базовые сервисы отвечают за обеспечение доступа к ресурсам компьютера и интерфейс для работы с памятью, файлами, устройствами, процессами и потоками.

- **Common Control Library** - библиотека общих элементов управления для разработки оконных интерфейсов.

- **Graphics Device Interface** - вывод графики на дисплей и другие устройства.

- **Network Services** - сетевые сервисы.

- **User Interface** - интерфейс пользователя.

- **Windows Shell** - функции для работы с оболочкой.

- **Windows System Information** - информация о конфигурации системы Windows.

- *Принципы, лежащие в основе Windows API:*
  - Системные ресурсы представляются в виде объектов:
    - Объект Windows – структура данных, представляющая системный ресурс
  - Основные классы объектов Windows:
    - Объекты ядра (файлы, процессы, потоки, сокеты…)
    - Объекты интерфейса пользователя (окна, курсоры, меню,…)
    - Объекты графического интерфейса (перья, кисти,…)
  - Манипуляции с объектами Windows – только через Windows API
  - Для идентификации объектов используются дескрипторы – специальные структуры, указывающие на объекты ОС и хранящие информацию о них
    - Дескриптор – тип `HANDLE`
    - Создание объектов: `Create<имя>` (например, `CreateFile`)
    - Закрытие дескриптора: `CloseHandle(<дескриптор>)`

- *Принципы, лежащие в основе Windows API:*
  - Имеется собственный набор типов данных:
    - Типы пишутся заглавными буквами (для Си)
    - Примеры типов: `HANDLE`, `BOOL`, `DWORD`, `LPTSTR`
  - Типы Windows API – «переобозначенные» базовые типы с учетом параметров компиляции:
    - `typedef unsigned long DWORD`
  - В именах типов Windows API «*» не используется:
    - `LPTSTR` – это `TCHAR *`
    - `LPCTSTR` – это `const TCHAR *`
    - `LPDWORD` – это `DWORD *`
  - Типы данных, представляющие собой указатели могут записываться в двух вариантах:
    - `LPDWORD` = `PDWORD`
    - `LPVOID` = `PVOID`

- *Принципы, лежащие в основе Windows API:*
  - Для возможности использовать Windows API нужно подключать библиотеки (модули)
    - Библиотека <windows.h> – для C/C++
    - Модуль Windows – для Delphi
  - В библиотеках (модулях) содержатся внешние определения функций («мостик» для обращения к соответствующим DLL-библиотекам)

- *Основные типы данных в Windows API:*
  - Типы данных объявлены в:
    - `<WinDef.h>`, `<WinNT.h>`, `<BaseTsd.h>` и некоторых других
  - Константы:
    - `#define CONST const`
  - Пустой (любой) тип:
    - `#define VOID void`
  - Целочисленные типы:
    - `typedef unsigned char BYTE;`
    - `typedef unsigned short WORD;`
    - `typedef unsigned long DWORD;`
    - `typedef short SHORT;`
      `typedef unsigned short USHORT;`
    - `typedef int INT; typedef unsigned int UINT;`
    - `typedef long LONG;  typedef unsigned long ULONG;`
  - Вещественные типы:
    - `typedef float FLOAT;`

- *Основные типы данных в Windows API:*
  - Логические типы:
    - `typedef int` **`BOOL;`**
    - `typedef BYTE` **`BOOLEAN;`**
  - Символьные типы:
    - `typedef char` **`CHAR;`** `typedef unsigned char` **`UCHAR;`**
    - `typedef wchar_t` **`WCHAR;`**
    - `#ifdef UNICODE typedef WCHAR` **`TCHAR;`**
      `#else typedef char` **`TCHAR;`**
  - Указатели:
    - `typedef BOOL *`**`PBOOL,*LPBOOL;`**
    - `typedef BYTE *`**`PBYTE,*LPBYTE;`**
    - `typedef int *`**`PINT,*LPINT;`**
    - `typedef WORD *`**`PWORD,*LPWORD;`**
    - `typedef DWORD *`**`PDWORD,*LPDWORD;`**
    - `typedef long *`**`PLONG,*LPLONG;`**
    - `typedef FLOAT *`**`PFLOAT;`**
    - `typedef UINT *`**`PUINT,*LPUINT;`** и др.

- *Основные типы данных в Windows API:*
  - Указатели:
    - `typedef void *PVOID,*LPVOID;`
    - `typedef CONST void *PCVOID,*LPCVOID;`
    - `typedef CHAR *PCHAR;`
    - `typedef CHAR *PSTR,*LPSTR;`
    - `typedef WCHAR *PWSTR,*LPWSTR;`
    - `typedef CONST CHAR *PCSTR,*LPCSTR;`
    - `typedef CONST WCHAR *PCWSTR,*LPCWSTR;`
    - `#ifdef UNICODE typedef LPWSTR PTSTR,LPTSTR;`
      `#else typedef LPSTR PTSTR,LPTSTR;`
    - `#ifdef UNICODE typedef LPCWSTR PCTSTR,LPCTSTR;`
      `#else typedef LPCSTR PCTSTR,LPCTSTR;`
  - Дескриптор объектов:
    - `typedef PVOID HANDLE;`
  - Win32/Win64 (пара примеров):
    - `typedef unsigned int DWORD32;`
    - `typedef unsigned __int64 DWORD64;`

- *Символы ASCII и Unicode (UTF-16):*
  - 8-битовые символы (ASCII): `char` = `CHAR`
  - 16-битовые символы (UTF-16): `wchar_t` = `WCHAR`

- Для написания обобщенных приложений нужно:

1. Определить все символы и строки с использованием обобщенных типов: `TCHAR`, `LPTSTR`, `LPCTSTR`
2. Включить в самом начале во все модули (для UTF-16):
   - `#define UNICODE` – для управления компиляцией библиотек Windows
   - `#define _UNICODE` – для управления компиляцией стандартных библиотек C
   - **Примечание**: Лучше управлять выбором через тип проекта
3. Размеры буферов в операциях ввода/вывода и других определять с использованием операции `sizeof(TCHAR)`

- Для написания обобщенных приложений нужно:
  4. Включить библиотеку `<tchar.h>` перед `<Windows.h>`
  5. Для ввода/вывода и преобразования строк использовать функции библиотеки `<tchar.h>`:
     - `_tprintf` вместо `printf`
     - `_tscanf` вместо `scanf`
     - `_totupper` вместо `toupper`
     - `_totlower` вместо `tolower`
     - `_ttoi` вместо `atoi`
     - и т.д.
     - **Примечание**: в библиотеке `<tchar.h>` определен тип `_TCHAR` – это аналог `TCHAR` Windows API
  6. Использовать макрос `_T(<строка>)` для строковых констант:
     - Пример: `_T("Hello world")`
     - **Примечание**: 16-битовую строковую константу можно описать явно: `L"Hello world"`

- Для написания обобщенных приложений нужно:
  7. Использовать обобщенную главную функцию:
     - `_tmain` вместо `main` и `wmain` – для консольных
     - `_tWinMain` вместо `WinMain` и `wWinMain` – для Win32
- Windows API предоставляет свои функции для работы с обобщенными строками и символами:
  - `CharUpper`, `CharLower`, `IsCharAlphaNumeric` и др.
  - Учитываются региональные особенности
- Функции Windows API автоматически являются обобщенными:
  - Например, для функции `CreateFile`:
    - `CreateFileA` – вариант с использованием ASCII-строк
    - `CreateFileW` – вариант с использованием UNICODE-строк
- Функции стандартных библиотек как правило обобщенными не являются!

Работа с файлами в Windows API

Работа с логическими дисками и томами

Получение списка томов

- Для выяснения того, какие логические диски существуют в системе, используется функция
  DWORD GetLogicalDrives(void)
  Каждый установленный бит возвращаемого значения соответствует существующему в системе логическому устройству. Например, если в системе существуют диски A:, C: и D:, то возвращаемое функцией значение равно 13(10).

- Функция
  DWORD GetLogicalDrivesStrings(DWORD cchBuffer, LPTSTR lpszBuffer)
  заполняет lpszBuffer информацией о корневом каталоге каждого логического диска в системе. В приведенном выше примере буфер будет заполнен символами

  A:\<null>C:\<null>D:\<null><null>

  параметр cchBuffer определяет длину буфера. Функция возвращает реальную длину буфера, необходимую для размещения всей информации.

Определение типа устройства

- Для определения типа диска предназначена функция
  UINT GetDriveType(LPTSTR lpszRootPathName)
  В качестве параметра ей передается символическое имя корневого каталога (напр. **A:**), а возвращаемое значение может быть одно из следующих:

Идентификатор	Описание
0	Тип устройства определить нельзя
1	Корневой каталог не существует
DRIVE_REMOVABLE	Гибкий диск
DRIVE_FIXED	Жесткий диск
DRIVE_REMOTE	Сетевой диск
DRIVE_CDROM	Компакт диск
DRIVE_RAMDISK	RAM диск

- Для получения подробной информации о носителе используется функция **GetVolumeInformation**. Она заполняет параметры информацией об имени тома, названии файловой структуры, максимальной длине имени файла, дополнительных атрибутах тома, специфических для файловой структуры.
- Функция **GetDiskFreeSpace** сообщает информацию о размерах сектора и кластера и о наличии свободных кластеров.

Работа с каталогами и файлами

Функция	**Выполняемое действие**
GetCurrentDirectory	Получение текущего каталога
SetCurrentDirectory	Смена текущего каталога
GetSystemDirectory	Получение системного каталога
GetWindowsDirectory	Получение основного каталога системы
CreateDirectory	Создание каталога
RemoveDirectory	Удаление каталога
CopyFile	Копирование файла
MoveFile MoveFileEx	Перемещение или переименование файла
DeleteFile	Удаление файла

Создание и открытие файла

```
HANDLE CreateFile (
    LPCTSTR lpFileName,          // pointer to name of the file
    DWORD dwDesiredAccess,  // access (read-write) mode
    DWORD dwShareMode,       // share mode
    LPSECURITY_ATTRIBUTES lpSecurityAttributes,
                    // pointer to security descriptor
    DWORD dwCreationDistribution, // how to create
    DWORD dwFlagsAndAttributes,  // file attributes
    HANDLE hTemplateFile         // handle to file with attributes to copy
);
```

- В случае удачи функция *CreateFile ()* возвращает дескриптор открытого файла.
- В случае ошибки функция возвращает не NULL, а значение INVALID_HANDLE_VALUE.

Параметры dwDesiredAccess и dwShareMode

- Параметр **dwDesiredAccess** задает тип доступа к файлу. Можно определить флаги GENERIC_READ и GENERIC_WRITE а так же их комбинацию для разрешения чтения или записи в файл.

- Параметр **dwShareMode** определяет режим совместного использования файла различными процессами. Если этот параметр равен нулю, то никакой другой поток не сможет открыть этот же файл. Флаги FILE_SHARE_READ и FILE_SHARE_WRITE а так же их комбинация разрешают другим потокам осуществлять доступ к файлу для чтения или записи.

Параметр dwCreationDistribution

- Параметр **dwCreationDistribution** определяет действия функции в зависимости от того, существует ли уже файл с указанным именем:
  - CREATE_NEW – создает файл, если файл существует, то ошибка;
  - CREATE_ALWAYS – создает файл , если файл существует, то старый файл удаляется и новый создается;
  - OPEN_EXISTING – открывает существующий файл;
  - OPEN_ALWAYS – создает файл, если файл не существует, то создается новый файл;
  - TRUNCATE_EXISTING – открывает файл и урезает его до нулевой длины.

Параметр dwFlagsAndAttributes

- Параметр **dwFlagsAndAttributes** определяет атрибуты создания файла:
  - FILE_ATTRIBUTE_ARCHIVE, FILE_ATTRIBUTE_HIDDEN,
  - FILE_ATTRIBUTE_NORMAL, FILE_ATTRIBUTE_READONLY,
  - FILE_ATTRIBUTE_SYSTEM, FILE_ATTRIBUTE_TEMPORARY
- Атрибуты файла могут комбинироваться за исключением FILE_ATTRIBUTE_NORMAL, который всегда используется один.
- Вместе с атрибутами могут комбинироваться и флаги, задающие режим работы с файлом:
  - FILE_FLAG_NO_BUFFERING – не осуществлять кэширование и опережающее чтение;
  - FILE_FLAG_RANDOM_ACCESS – кэшировать как файл произвольного доступа;
  - FILE_FLAG_SEQUENTIAL_SCAN – кэшировать как файл последовательного доступа;
  - FILE_FLAG_WRITE_TROUGH – не буферизовать операцию записи, производить запись на диск немедленно;
  - FILE_FLAG_DELETE_ON_CLOSE – уничтожить файл при закрытии, полезно комбинировать с атрибутом FILE_ATTRIBUTE_TEMPORARY;
  - FILE_FLAG_OVERLAPPED – работа с файлом будет осуществляться асинхронно.

Получение размера файла

```
DWORD GetFileSize(
    HANDLE hFile, // дескриптор файла
    LPDWORD lpFileSizeHigh // указатель на старшую часть
                           // 64-разрядного размера файла
);
```

- **Возвращаемое значение:** младшие 32 бита размера файла.

Получение типа файла

```
DWORD GetFileType (
    HANDLE hFile, // дескриптор файла
);
```

- **FILE_TYPE_CHAR** – символьный файл, обычно устройства LPT или консоли;
- **FILE_TYPE_DISK** – файл на диске;
- **FILE_TYPE_PIPE** – файл является именованным или анонимным каналом;
- **FILE_TYPE_UNKNOWN** – тип указанного файла неизвестен, или функция завершилась ошибкой.

Сброс изменений файла из буфера на диск

- Так как ввод и вывод данных на диск в операционной системе Windows буферизуется, запись данных на диск может быть отложена до тех пор, пока система не освободится от выполнения текущей работы.
- С помощью функции *FlushFileBuffers ()* вы можете принудительно заставить операционную систему записать на диск все изменения для файла, дескриптор которого передается этой функции через единственный параметр:

```
BOOL FlushFileBuffers (HANDLE hFile);
```

- В случае успешного завершения функция возвращает значение TRUE, при ошибке – FALSE.

Сброс изменений всех файлов тома

- Если Вы хотите сбросить на диск изменения не одного файла, а всех файлов тома, то необходимо с помощью функции *CreateFile ()* открыть том как **\\.\<x>:**
.
- Для выполнения сброса изменений тома программа должна иметь права доступа администратора.

Работа с атрибутами файла

```
BOOL SetFileAttributes(
    LPCTSTR lpFileName, // имя файла
    DWORD dwFileAttributes // атрибуты
);

DWORD GetFileAttributes(
   LPCTSTR lpFileName // имя файла или каталога
);
```

Ограничения функции SetFileAttributes

- **FILE_ATTRIBUTE_COMPRESSED** – чтобы установить сжатое состояние файла, используйте функцию *DeviceIoControl ()* с операцией **FSCTL_SET_COMPRESSION**;
- **FILE_ATTRIBUTE_DIRECTORY** – файл не может быть преобразован к каталог;
- **FILE_ATTRIBUTE_ENCRYPTED** – чтобы создать зашифрованный файл, используйте функцию *CreateFile ()* этим с атрибутом. Для конвертации существующего файла в зашифрованный, используйте функцию *EncryptFile ()*.
- **FILE_ATTRIBUTE_REPARSE_POINT** – чтобы связать точку монтирования с файлом или каталогом, используйте функцию *DeviceIoControl ()* с операцией **FSCTL_SET_REPARSE_POINT**;
- **FILE_ATTRIBUTE_SPARSE_FILE** – чтобы установить атрибут разреженности файла, используйте функцию *DeviceIoControl ()* с операцией **FSCTL_SET_SPARSE**.

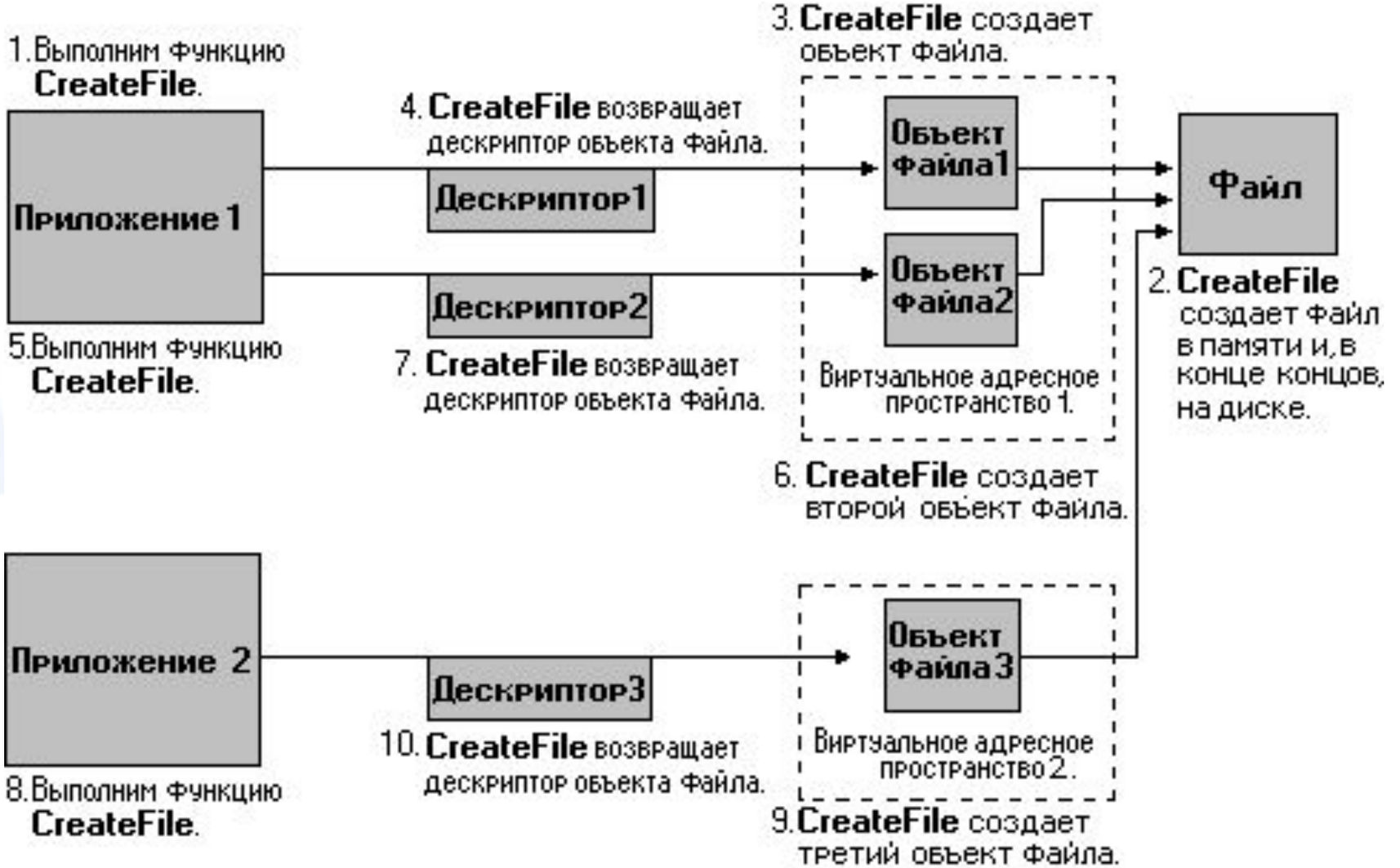
Вопрос

- С помощью какой функции следует закрыть файл, который был открыт/создан с помощью *CreateFile ()* ?

Совместная работа с файлами

Совместный доступ к файлу

- Управление объектами типа «файл» отличается от управления другими объектам ядра.
- Вы не можете открыть ранее созданный объект типа «файл», Вы можете только создать новый объект типа «файл» и далее выполнять совместный доступ к файлу на диске.



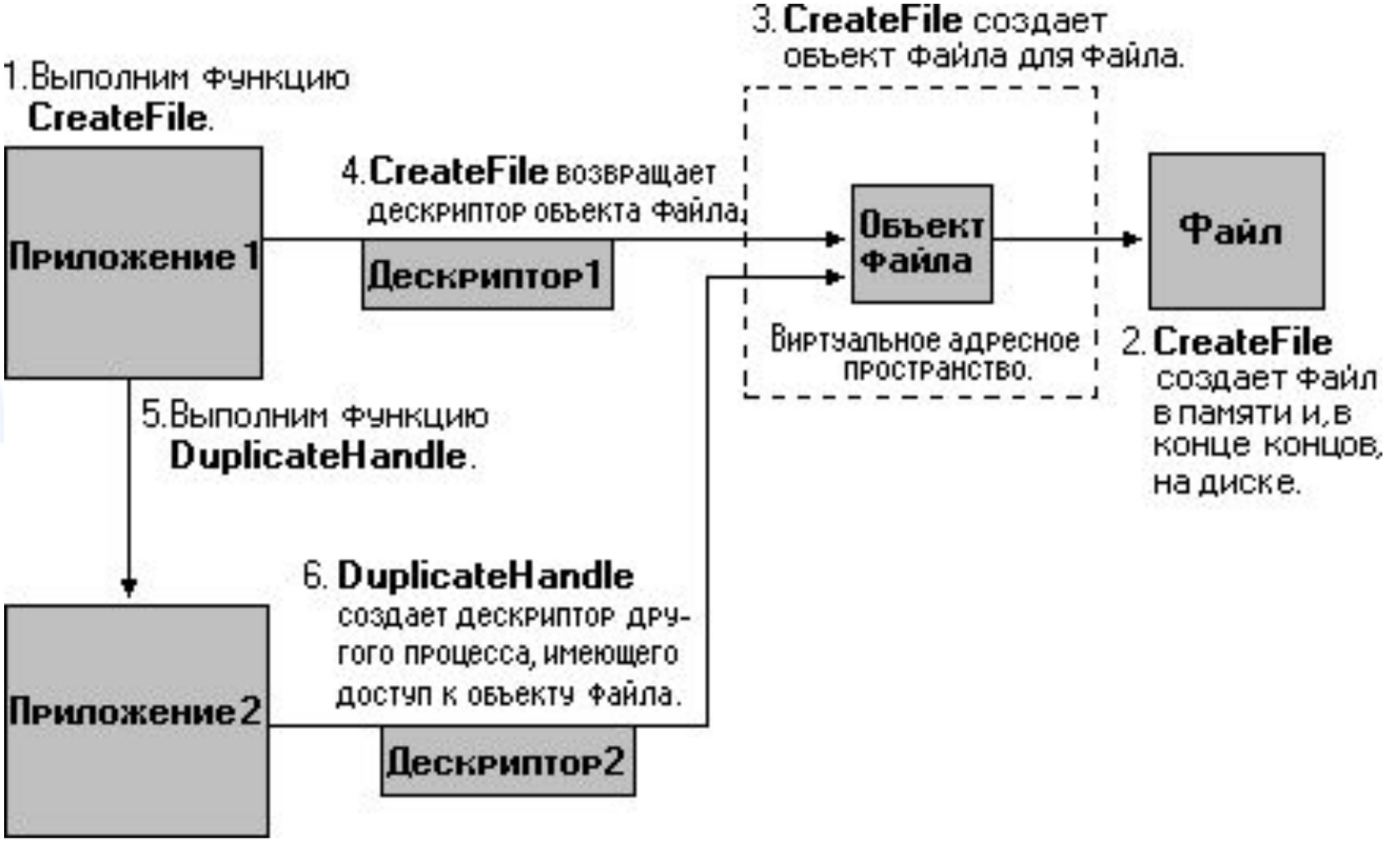
Функция повторного открытия файла

```
HANDLE ReOpenFile(
    HANDLE hOriginalFile, // дескриптор уже открытого файла
    DWORD dwDesiredAccess,  //  права доступа
    DWORD dwShareMode,  // режим совместного доступа
    DWORD dwFlags // флаги
);
```

- **Примечание:** Параметр *dwFlags* не может содержать какой-либо из атрибутов файла (**FILE_ATTRIBUTE_ ***). Эти атрибуты могут задаваться только тогда, когда файл создается.

Совместное использование объекта типа «файл»

- Только используя механизм дублирования дескрипторов объектов, Вы можете получить более одного дескриптора для одного и того же объекта типа «файл».



Совместный доступ и блокировка файлов

- Если функции *CreateFile ()* указать режимы совместного использования файла FILE_SHARE_READ или FILE_SHARE_WRITE, то несколько процессов смогут одновременно открыть файлы и выполнять операции чтения и записи. Если же эти режимы не указаны, совместное использование файлов будет невозможно – первый процесс, который откроет файл, заблокирует возможность работы с этим файлом для других процессов.
- Для организации совместного доступа нескольких процессов к разным участкам файла участку файла требуется использовать пару функций *LockFile () / UnlockFile ()* или *LockFileEx () / UnlockFileEx ()*.
- Блокирование файлов является ограниченной разновидностью синхронизации параллельно выполняющихся процессов и потоков, обсуждение вопросов синхронизации будет рассмотрено в соответствующей лекции.

Функции блокировки

- Блокировка участка файла для монопольного доступа выполняется функцией *LockFile ()*, после использования заблокированного участка, а также перед завершением своей работы процессы должны разблокировать все заблокированные ранее участки, вызвав для этого функцию *UnlockFile ()*.
- Функции *LockFileEx ()* и *UnlockFileEx ()* позволяют блокировать/разблокировать участок открытого файла либо для разделяемого доступа (разрешающего доступ одновременно нескольким приложениям в режиме чтения), либо для монопольного доступа (разрешающего доступ только одному приложению в режиме чтения/записи).

Функции *LockFile* и *UnlockFile*

```
BOOL LockFile(
    HANDLE hFile, // дескриптор файла
    DWORD  dwFileOffsetLow,  // младшее слово смещения участка
    DWORD  dwFileOffsetHigh, // старшее слово смещения участка
    DWORD  nNumberOfBytesToLockLow, // младшее слово длины участка
    DWORD  nNumberOfBytesToLockHigh // старшее слово длины участка
);

BOOL UnlockFile(
    HANDLE hFile, //  дескриптор  файла
    DWORD  dwFileOffsetLow,  // младшее слово смещения участка
    DWORD  dwFileOffsetHigh, // старшее слово смещения участка
    DWORD  nNumberOfBytesToUnlockLow, // млад. слово длины участка
    DWORD  nNumberOfBytesToUnlockHigh // старш. слово длины участка
);
```

Функции *LockFileEx* и *UnlockFileEx*

```
BOOL LockFileEx(
    HANDLE hFile, //  дескриптор  файла
    DWORD dwFlags, // вид блокировки и режим ожидания
    DWORD dwReserved, // зарезервировано
    DWORD nNumberOfBytesToLockLow, // млад. слово длины участка
    DWORD nNumberOfBytesToLockHigh, // старш. слово длины участка
    LPOVERLAPPED lpOverlapped // указание начала участка
) ;

BOOL UnlockFileEx(
    HANDLE hFile, //  дескриптор  файла
    DWORD dwFlags, // вид блокировки и режим ожидания
    DWORD dwReserved, // зарезервировано
    DWORD nNumberOfBytesToUnlockLow, // млад. слово длины участка
    DWORD nNumberOfBytesToUnlockHigh, // старш. слово длины участка
    LPOVERLAPPED lpOverlapped // указание начала участка
) ;
```

Параметры *LockFileEx* и *UnlockFileEx*

- *dwFlags* – определяет вид блокировки файла, а также режим ожидания доступности затребованной блокировки:
  - LOCKFILE_EXCLUSIVE_LOCK – запрос монопольной блокировки в режиме чтения/записи. Если это значение не задано, запрашивается разделяемая блокировка (только чтение).
  - LOCKFILE_FAIL_IMMEDIATELY – задает режим немедленного возврата функции с возвращаемым значением равным FALSE, если приобрести блокировку не удается. Если это значение не задано, функция переходит в режим ожидания.
- *lpOverlapped* – используется для указания 64-битовой позиции начала участка файла, подлежащего блокированию;
- *dwReserved* – значение этого параметра должно быть равным 0.

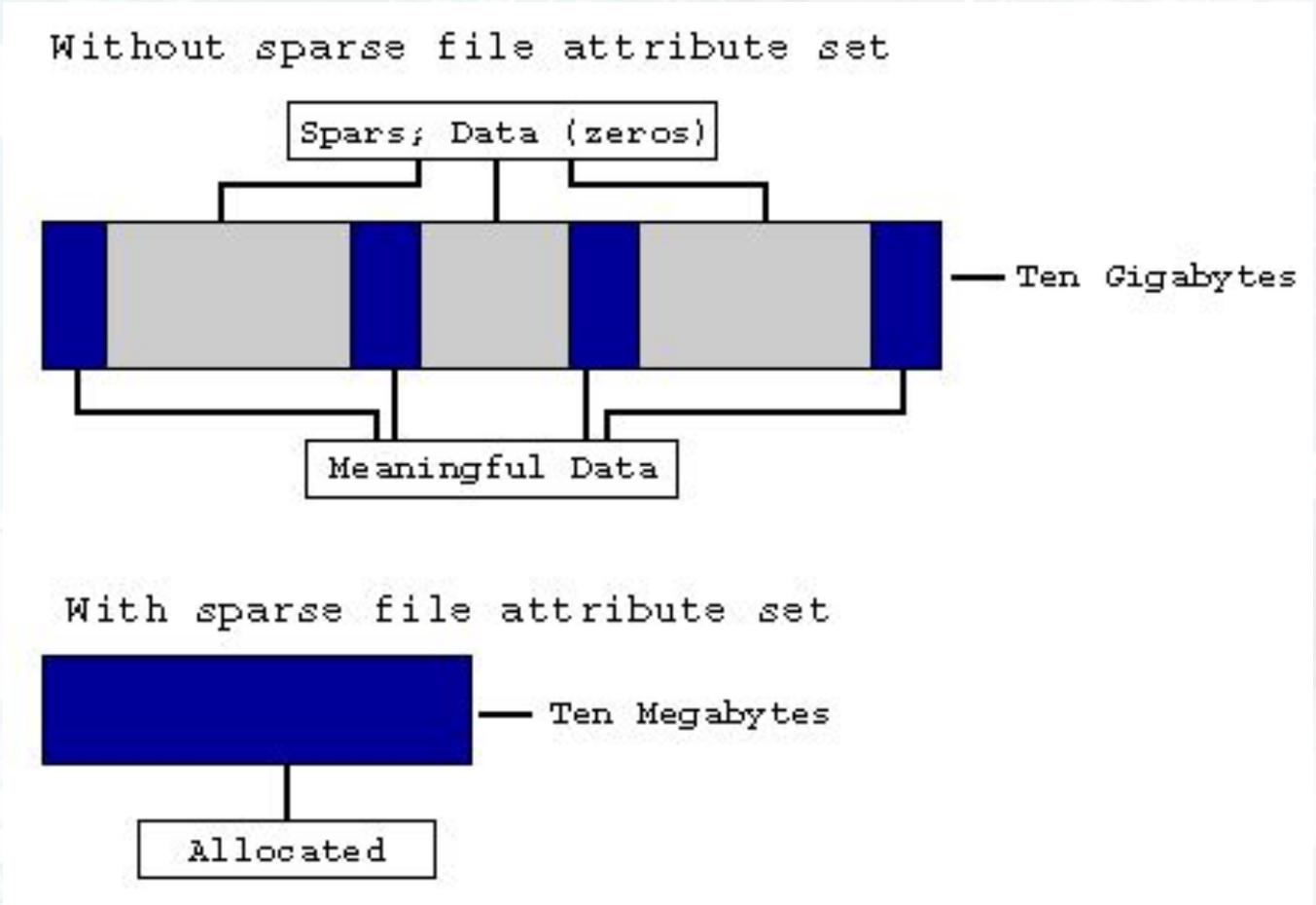
Особенности блокирования участков файла

- Вновь создаваемая и существующие области блокирования в файле не могут перекрываться.
- Возможно блокирование участка, границы которого выходят за пределы файла. Такая операция может оказаться полезной в случае расширения файла процессом или потоком.
- Блокировки не наследуются вновь создаваемыми процессами.
- Границы участка разблокирования должны в точности совпадать с границами ранее заблокированной области. Любая попытка разблокирования участка, не совпадающего в точности с одной из существующих заблокированных областей, будет неудачной (функция вернет FALSE).

Работа с файлами в Windows API

Работа с «разреженными» файлами

Разреженные файлы (sparse files)



Создание разреженных файлов

- Прежде чем создать разреженный файл необходимо проверить поддержку **FILE_SUPPORTS_SPARSE_FILES** со стороны файловой системы.
- Для создания разреженного файла необходимо установить этому файлу атрибут **FILE_ATTRIBUTE_SPARSE_FILE**.
- С помощью функции *CreateFile ()* установить атрибут разреженного файла невозможно, для установки атрибута разреженного файла необходимо воспользоваться специальной функцией управления устройствами ввода-вывода *DeviceIoControl ()* с управляющим кодом **FSCTL_SET_SPARSE.**
- Единственный способ сбросить бит этого атрибута состоит в том, чтобы переписать файл, например, при помощи вызова функция *CreateFile ()* с флагом **CREATE_ALWAYS**.

Пример проверки поддержки разреженных файлов

```
char   szVolName[MAX_PATH],  szFSName[MAX_PATH];
DWORD  dwSN, dwMaxLen, dwVolFlags;

GetVolumeInformation("C:\\", szVolName, MAX_PATH, &dwSN,
      &dwMaxLen, &dwVolFlags, szFSName, MAX_PATH);

if (dwVolFlags & FILE_SUPPORTS_SPARSE_FILES)
{
      // File system supports sparse streams
} else {
      // Sparse streams are not supported
}
```

Функция *DeviceIoControl*

```
BOOL DeviceIoControl (
    HANDLE hDevice, // дескриптор устройства
    DWORD dwIoControlCode, // код операции
    LPVOID lpInBuffer, // буфер входных данных
    DWORD nInBufferSize, // размер буфера входных
    данных
    LPVOID lpOutBuffer, // буфер данных результата
    DWORD nOutBufferSize, // размер буфера результата
    LPDWORD lpBytesReturned, // адрес данных для вывода
    LPOVERLAPPED lpOverlapped  // адрес структуры
                // OVERLAPPED
);
```

Пример создания разреженного файла

```
HANDLE hFile = CreateFile("C:\\Sparse.dat", GENERIC_WRITE,
0,    NULL, CREATE_NEW,  FILE_ATTRIBUTE_NORMAL, NULL);

DWORD dwTemp;

DeviceIoControl(hFile, FSCTL_SET_SPARSE, NULL, 0, NULL, 0,
&dwTemp, NULL);}
```

Особенности вызова *DeviceIoControl*

- Если параметр *hDevice* открывался без установки флажка FILE_FLAG_OVERLAPPED, параметр *lpOverlapped* игнорируется.
- Если параметр *hDevice* открывался с флагом FILE_FLAG_OVERLAPPED, операция выполняется как перекрывающая (асинхронная). В этом случае, параметр *lpOverlapped* должен указать на допустимую структуру OVERLAPPED, которая содержит дескриптор объекта события. Иначе, функция завершается ошибкой непредсказуемыми способами.
- Если параметр *lpOverlapped* – NULL, то *lpBytesReturned* не может быть NULL.

Задание разреженных областей файла

- Для задания в разреженном файле «нулевых» областей необходимо использовать функцию *DeviceIoControl ()* с управляющим кодом **FSCTL_SET_ZERO_DATA** .
- Начало и конец создаваемой «нулевой» области задаются через параметр *lpInBuffer* функции *DeviceIoControl ().*
- Если используете функцию *DeviceIoControl ()* с кодом **FSCTL_SET_ZERO_DATA** в отношении неразреженного файла, то в файл будут записаны «физические» нули.

Пример создания разреженной области файла

```
FILE_ZERO_DATA_INFORMATION fzdi;
DWORD dwTemp;

fzdi.FileOffset.QuadPart = uAddress;
fzdi.BeyondFinalZero.QuadPart =   uAddress + uSize;

DeviceIoControl(hFile, FSCTL_SET_ZERO_DATA, &fzdi,
sizeof(fzdi),      NULL, 0, &dwTemp, NULL);
```

Задание разреженной области в конце файла

- Для задания разреженной области в конце файла нет необходимо использовать функцию *DeviceIoControl ()* с управляющим кодом **FSCTL_SET_ZERO_DATA** .
- Достаточно установить указатель файла в новую позицию за «пределами» файла и затем вызвать функцию *SetEndOfFile ()*.

Получение информации о разреженных областях файла

- Для получения информации о разреженных областях файла необходимо использовать функцию *DeviceIoControl ()* с управляющим кодом **FSCTL_QUERY_ALLOCATED_RANGES**.
- Пример кода программы, определяющей расположение разреженных областей файла приведен в статье http://www.flexhex.com/docs/articles/sparse-files.phtml

Синхронный и асинхронный ввод/вывод

- При синхронной работе приложение, запустив операцию ввода вывода, переходит в состояние блокировки до ее окончания (т.е. ожидает завершения операции ввода вывода).

- При асинхронной работе прикладная программа, запустив операцию ввода вывода, не ожидает ее завершения, а продолжает исполняться.

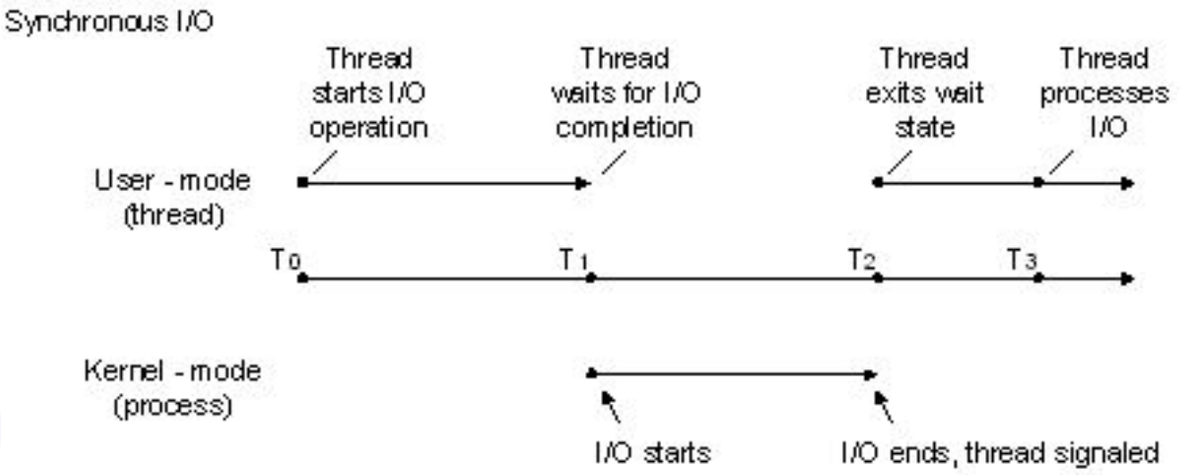
•Выполнение

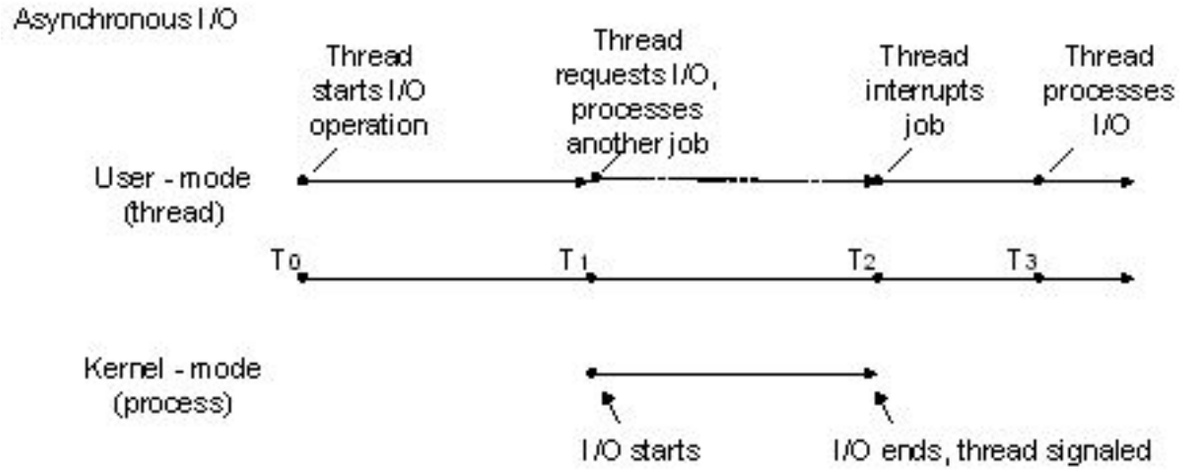
•Ожидание

•Готовность

Синхронный и асинхронный ввод/вывод

- При синхронной работе приложение, запустив операцию ввода вывода, переходит в состояние блокировки до ее окончания (т.е. ожидает завершения операции ввода вывода).
- При асинхронной работе прикладная программа, запустив операцию ввода вывода, не ожидает ее завершения, а продолжает исполняться.





Асинхронный ввод-вывод

- Для организации асинхронной работы с файлами необходимо при вызове функции *CreateFile ()* установить флаг FILE_FLAG_OVERLAPPED в параметре dwFlagsAndAttributes.

- После этого функции *ReadFile ()* и *WriteFile ()* будут работать асинхронно, т. е. только запускать операции ввода вывода и не ожидать их завершения.

- Структура данных OVERLAPPED, на которую указывает параметр *lpOverlapped,* должна оставаться допустимой для длительной операции чтения. Она не должна быть переменной, которая может выйти из области действия, пока происходит операция чтения файла.

Перекрывающийся асинхронный ввод-вывод

- Когда над одним файлом (или другим объектом ввода-вывода) одновременно выполняют несколько асинхронных операций ввода-вывода, то говорят, что это перекрывающийся ввод-вывод (Overlapped I/O).

- Использование перекрывающегося ввод-вывода позволяет увеличить производительность приложений.

Функции файлового ввода-вывода

```
BOOL ReadFile(
      HANDLE    hFile, // дескриптор файла
      LPVOID     lpBuffer, // адрес буфера
      DWORD     nNumberOfBytesToRead, // кол-во байт
      LPDWORD      lpNumberOfBytesRead, // адрес кол-во байт
      LPOVERLAPPED lpOverlapped // адрес OVERLAPPED
);

BOOL WriteFile(
      HANDLE    hFile, // дескриптор файла
      LPCVOID   lpBuffer, // адрес буфера
      DWORD     nNumberOfBytesToWrite, // кол-во байт
      LPDWORD      lpNumberOfBytesToWrite, // адрес кол-во байт
      LPOVERLAPPED lpOverlapped // адрес OVERLAPPED
);
```

Параметры функций файлового ввода-вывода

- hFile – дескриптор файла;
- lpBuffer – адрес буфера, в который будет производиться чтение/запись;
- nNumberOfBytes… – количество байт, которые необходимо прочитать/записать;
- lpNumberOfBytes… – адрес переменной, в которой будет размещено количество реально прочитанных/записанных байт;
- lpOverlapped – указатель на структуру OVERLAPPED, управляющую асинхронным вводом выводом.

Пример синхронного копирования файла

```
/* Open files for input and output. */
inhandle = CreateFile("data", GENERIC_READ, 0, NULL, OPEN_EXISTING, 0, NULL);
outhandle = CreateFile ("newf", GENERIC_WRITE, 0, NULL, CREATE_ALWAYS,
     FILE_ATTRIBUTE_NORMAL, NULL);

/* Copy the file. */
do {
       s = ReadFile(inhandle, buffer, BUF_SIZE, &count, NULL);
       if (s && count > 0) WriteFile(outhandle, buffer, count, Socnt, NULL);
} while (s>0 && count>0);

/* Close the files. */
CloseHandle (inhandle):
CloseHandle (outhandle);
```

Позиционирование указателя синхронного ввода-вывода

```
DWORD SetFilePointer(
    HANDLE hFile, // дескриптор файла
    LONG lDistanceToMove, // смещение указателя
    PLONG lpDistanceToMoveHigh, // указатель на старшую часть
    // 64-разрядного  смещения
    DWORD dwMoveMethod // точка отсчета
);
```

- **FILE_BEGIN** – отсчет от начала файла;
- **FILE_CURRENT** – отсчет от текущей позиции файла;
- **FILE_END** – отсчет от конца файла.

Установка конца файла

```
BOOL SetEndOfFile(
    HANDLE hFile, // дескриптор файла
);
```

- Функция перемещает позицию метки конца файла (EOF) для заданного файла к текущую позицию его указателя.

Структура перекрывающегося асинхронного ввода-вывода

```
typedef struct _OVERLAPPED {
    DWORD Internal;    //Используется операционной системой.
      //Хранит статус завершения операции.
    DWORD InternalHigh;  //Используется ОС. Хранит
      //количество переданных байт.
    DWORD Offset;      //Позиция в файле, начиная с которой
      //необходимо производить операцию
                //чтения (записи).
    DWORD OffsetHigh;//Количество байт для передачи.
    HANDLE hEvent;    //Описатель события, которое произойдет
      //при завершении операции чтения              //(записи).
} OVERLAPPED;
```

Вариант 1 организации асинхронного ввода-вывода

- Перед запуском операции асинхронного ввода-вывода необходимо создать объект «событие» и затем передать его дескриптор в функцию *ReadFile ()* или *WriteFile ()* в качестве элемента *hEvent* структуры OVERLAPPED.
- Позиция файла, начиная с которой производится операция ввода-вывода, должна быть установлена в членах *Offset* и *OffsetHigh* структуры OVERLAPPED.
- Программа, выполнив необходимые действия одновременно с операцией передачи данных, вызывает одну из функций ожидания, например, *WaitForSingleObject ()*, передавая ей в качестве параметра дескриптор события.
- Выполнение программы при этом приостанавливается до завершения операции ввода-вывода.

Функция *WaitForSingleObject*

DWORD WaitForSingleObject
(HANDLE hObject, DWORD dwMilliseconds);

- *hObject* – идентифицирует объект ядра, относительно которого будет выполняться синхронизация, в случае асинхронного ввода-вывода файла это обычно дескриптор объекта «событие»;
- *dwMilliseconds* – указывает, сколько времени (в миллисекундах) поток готов ждать синхронизации:
  - значение «0» – функция просто проверит состояние объекта синхронизации;
  - значение INFINITE (-1) – ожидание будет «вечным», пока объект синхронизации не сработает.

Функция *WaitForSingleObject*

- Функция *WaitForSingleObject ()* возвращает одно из следующих значений:
  - WAIT_OBJECT_0 – синхронизация была выполнена;
  - WAIT_TIMEOUT – функция была завершена по тайм-ауту, синхронизация не выполнена;
  - WAIT_ABANDONED – для объекта типа «событие» не используется (используется для «семафоров» и «мьютексов»);
  - WAIT_FAILED – функция завершилась с ошибкой.

Проверка завершения асинхронного ввода-вывода

- Проверить статус незавершенной операции асинхронного ввода-вывода можно используя макрос

```
BOOL HasOverlappedIoCompleted
(LPOVERLAPPED lpOverlapped);
```

- #define HasOverlappedIoCompleted (lpOverlapped)
  ((lpOverlapped)->Internal != STATUS_PENDING)

Вариант 2 организации асинхронного ввода-вывода

- Событие не создается. В качестве ожидаемого объекта выступает сам файл. Его описатель передается в функцию *WaitForSingleObject ()*.

- Этот метод прост и корректен, но не позволяет производить параллельно несколько операций ввода-вывода с одним и тем же файлом, т.е. не поддерживает перекрывающийся ввод-вывод.

Вариант 3 организации асинхронного ввода-вывода

- «Тревожный» (allertable) асинхронный ввод-вывод предполагает использование функций *ReadFileEx ()* и *WriteFileEx ()*. В качестве дополнительного параметра в эти функции передается адрес функции завершения (APC –asynchronous procedure call), которая будет вызываться всякий раз при завершении операции ввода-вывода.

- Существенно, что эти функции завершения выполняются в том же самом потоке что и функции файлового ввода/вывода. Это значит, что поток, запустивший операции чтения/записи должен приостановить себя, например, с помощью функций *Sleep ()* и *SleepEx ()*, и предоставить возможность выполнения функции завершения.

Функции *ReadFileEx* и *WriteFileEx*

- *BOOL ReadFileEx(*
  HANDLE hFile, LPVOID lpBuffer,
  DWORD nNumberOfBytesToRead,
  LPOVERLAPPED lpOverlapped,
  LPOVERLAPPED_COMPLETION_ROUTINE lpcr)

- *BOOL WriteFileEx(*
  HANDLE hFile, LPVOID lpBuffer,
  DWORD nNumberOfBytesToWrite,
  LPOVERLAPPED lpOverlapped,
  LPOVERLAPPED_COMPLETION_ROUTINE lpcr)

Особенности тревожного асинхронного ввода-вывода

- Структура данных OVERLAPPED, на которую, указывает параметр *lpOverlapped* должна оставаться допустимой для длительной операции чтения. Она не должна быть переменной, которая может выйти из области действия, пока происходит операция чтения файла.

- Функции *ReadFileEx ()* и *WriteFileEx ()* игнорируют поле *hEvent* структуры OVERLAPPED.

Функция завершения

VOID CALLBACK FileIOCompletionRoutine
(DWORD dwErrorCode,
DWORD dwNumberOfBytesTransfered,
LPOVERLAPPED *lpOverlapped*);

- *dwErrorCode* – состояние завершения ввода-вывода, значения параметра ограничены 0 (успешное завершение) и ERROR_HANDLE_EOF (при попытке выполнить чтение с выходом за пределы файла);
- *dwNumberOfBytesTransfered* – переданное число байтов;
- *lpOverlapped* – структура, которая использовалась завершившимся вызовом *ReadFileEx ()* или *WriteFileEx ()*.

Функция *Sleep*

VOID Sleep (DWORD dwMilliseconds);

- Функция приостанавливает поток на *dwMilliseconds* миллисекунд.
- Особенности выполнения функции *Sleep ()*:
  - поток добровольно отказывается от остатка кванта времени;
  - система приостанавливает поток на период, ***примерно*** равный заданному;
  - Вы можете вообще запретить планировать поток, передав в качестве *dwMilliseconds* значение INFINITE (-1);
  - Вы можете вызвать *Sleep* и передать в качестве *dwMilliseconds* ноль. В этом случае поток будет вытеснен с процессора и помещен в очередь ожидания. Однако поток снова будет запущен, если нет других готовых потоков с тем же приоритетом.

Функция *SleepEx*

DWORD SleepEx
(DWORD dwMilliseconds, BOOL bAlertable);

- Функция приостанавливает выполнения потока до наступления события ввода/вывода или на время.
- Отличия выполнения от функции *Sleep ()*:
  - если параметр ***bAlertable = FALSE***, то функция ведет себя аналогично *Sleep ()*;
  - если параметр ***bAlertable = TRUE***, и этот поток переходит в ожидание оповещения и может продолжить выполнение после срабатывания вызова APC или истечение времени блокировки;
  - функция возвращает значение **WAIT_IO_COMPLETION**, если завершение произошло в результате срабатывания вызова APC.

Асинхронные вызовы процедур

- Главный поток указывает APC-функцию данной целевого потока путем помещения объекта APC в очередь APC данного потока (функция *QueueUserAPC()*). В очередь могут быть помещены несколько APC.
- Целевой поток переходит в состояние дежурного ожидания (alertable wait state), обеспечивающее возможность безопасного выполнения потоком APC.
- Целевой поток, находящийся в состоянии ожидания, выполняет все APC, находящиеся в очереди.

- **Примечание:** Порядок первых двух шагов не важен, поэтому о возникновении «гонок» можно не беспокоиться.

Функция *QueueUserAPC*

- Текущий поток помещает APC в очередь целевого потока с помощью функции:

```
DWORD QueueUserAPC(
    PAPCFUNC pfnAPC, //  указатель на APC-функцию
    HANDLE hThread, // дескриптор целевого потока
    ULONG_PTR dwData // передаваемое APC-функции значение
);
```

- В случае успешного завершения функция возвращает – ненулевое значение, иначе – ноль.

Усовершенствованные средства для работы с файлами и каталогами и знакомство с реестром

Файловые системы обеспечивают не только простую последовательную обработку файлов; кроме этого, они должны предоставлять возможности прямого доступа к файлам и блокирования файлов, а также предлагать средства для управления каталогами и атрибутами файлов. В данной главе, которая начинается с обсуждения прямого доступа к файлам, требуемого для обслуживания баз данных, обработки файлов и решения целого ряда других задач, демонстрируются методы непосредственного доступа к данным, находящимся в произвольном месте файла, которые обеспечиваются файловыми указателями. Для этого, в частности, нам надо будет обсудить использование 64-битовых указателей Windows, поскольку файловая система NTFS способна поддерживать файлы гигантских размеров.

Далее будут рассмотрены методы просмотра каталогов, рассказано о том, что такое атрибуты файлов, такие, например, как метки времени, атрибуты прав доступа или размер файла, и показано, как управлять атрибутами и интерпретировать их. Наконец, вы ознакомитесь с тем, как использовать блокирование файлов с целью предотвращения попыток изменения их содержимого одновременно несколькими процессами.

Завершает данную главу рассмотрение реестра Windows — централизованной базы данных, хранящей информация о конфигурации системы, которой могут пользоваться как приложения, так и сама операционная система. Приведенный в конце главы пример программы показывает, что функции, с помощью которых осуществляется доступ к реестру, и структура соответствующих программ напоминают те, которые применяются для управления файлами и каталогами, что и послужило причиной включения этой темы в данную главу.

Реестр Windows

Реестр Windows был введён для упорядочения информации, хранившейся до этого во множестве INI-файлов, которые использовались для хранения настроек до того, как появился реестр.

- или системный реестр (англ. Windows Registry) — иерархически построенная база данных параметров и настроек в большинстве операционных систем Microsoft Windows.

- Реестр содержит информацию и настройки для аппаратного обеспечения, программного обеспечения, профилей пользователей. Большинство изменений в Панели управления, ассоциации файлов, системные политики, список установленного ПО и т.д. фиксируются в реестре.

Операционная система управляет большим объемом информации, необходимой для ее загрузки и конфигурирования.

Состав Реестра

Windows 95/98

- User.dat
- System.dat

Windows ME

- Classes.dat
- User.dat
- System.dat

Windows XP

- system
- software
- sam
- security
- default

В System32\Config\

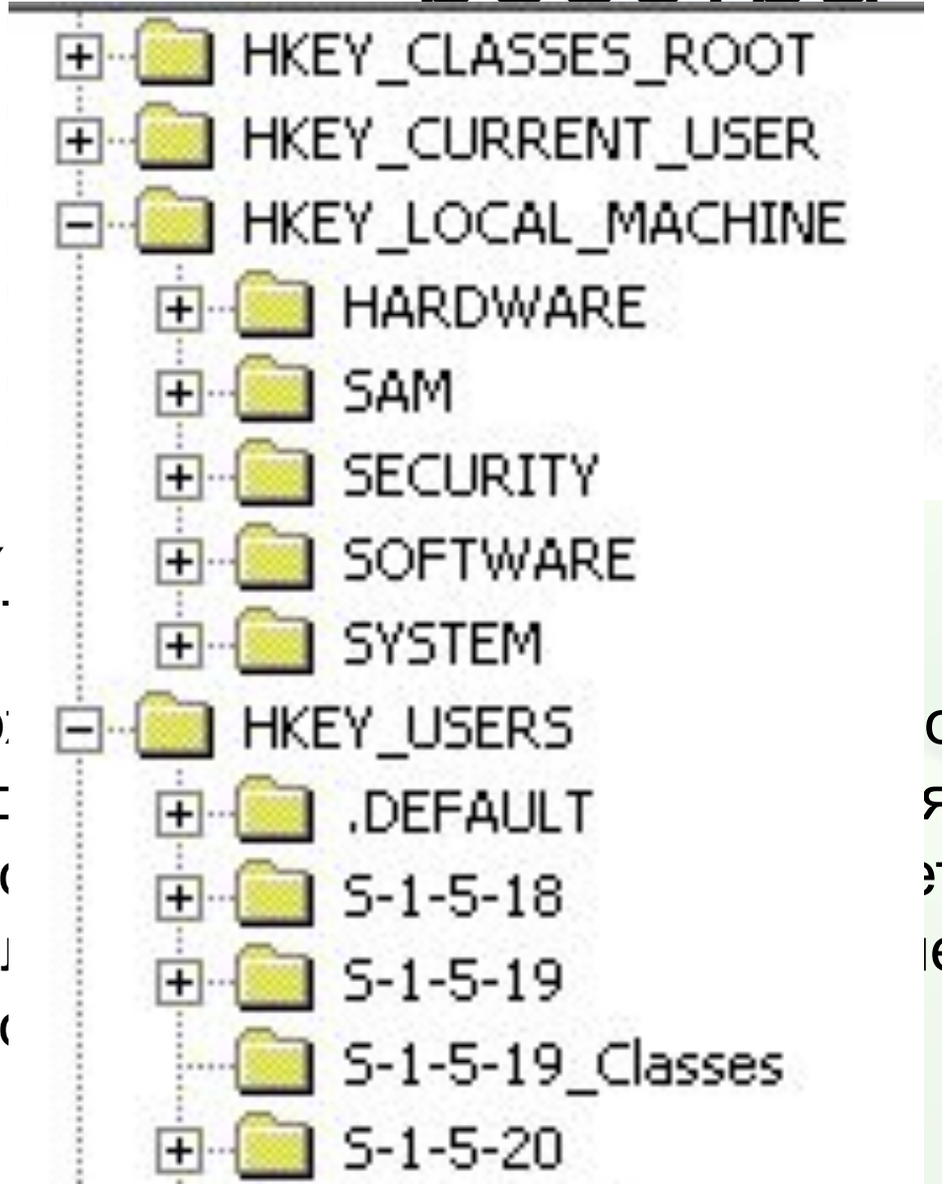
Дополнительно могут создаваться файлы:

Documents and Settings\<Username>\ —**Ntuser.dat**

Documents and Settings\<Username>\Local Settings\Application Data\Microsoft\Windows\ —**UsrClass.dat**

- Можно провести некое примерное соответствие файлов и веток реестра, но оно не такое простое, полное и однозначное. Однако примерно можно сказать следующее:
- Ветка реестра «HKEY_LOCAL_MACHINE\Software\» формируется из файла «software».
- Ветка реестра «HKEY_LOCAL_MACHINE\System\» формируется из файла «system».
- Ветка реестра «HKEY_USERS\» формируется из файлов «default» и других.

Логическая структура реестра



Hive –
улей (куст)

HKEY

содер ого пользователя,
вошед я папки
польз етры панели
управ ены с профилем
польз

Логическая структура реестра

HKEY_USERS (HKU)

- содержит все активные загруженные профили пользователей компьютера.

HKEY_LOCAL_MACHINE (HKLM)

- Раздел содержит параметры конфигурации, относящиеся к данному компьютеру (для всех пользователей).

HKEY_CLASSES_ROOT (HKCR)

- В основном, содержит информацию о зарегистрированных типах файлов и объектах COM и ActiveX. Кроме того, раздел HKEY_CLASSES_ROOT предоставляет объединённые данные программам, написанным под ранние версии Windows.

HKEY_CURRENT_CONFIG

- содержит сведения о профиле оборудования, используемом при запуске системы.

HKEY_DYN_DATA (только в реестре ОС Windows 9x/ME)

- Содержит динамически изменяемые данные о компьютере (загрузка процессора, размер файла подкачки и т. п.).

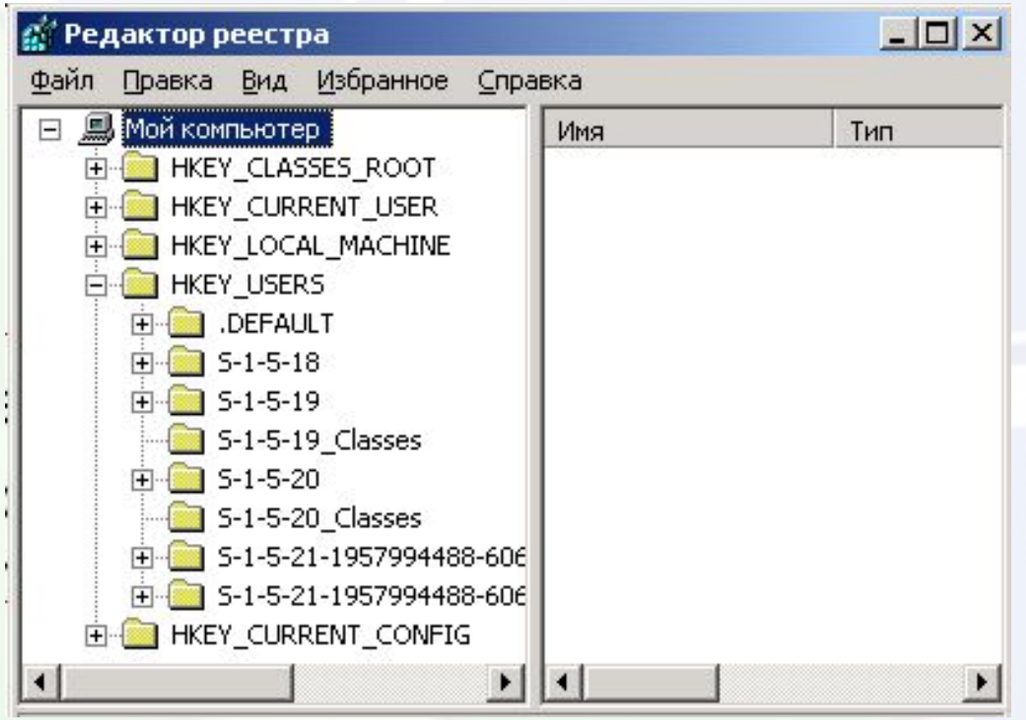
Недостатки реестра

- Реестр подвержен фрагментации, из-за чего доступ к реестру постепенно замедляется
- В связи с тем, что помимо настроек в реестре хранится различная информация системы и приложений (например многие приложения хранят в реестре список недавно открытых файлов), размер реестра значительно увеличивается по мере использования операционной системы. Эта проблема частично решается при помощи специальных утилит
- Не все настройки системы хранятся в реестре, соответственно перенос настроек системы путём копирования реестра невозможен.

Критики приводят в пример UNIX-подобные операционные системы, где нет реестра, выполняемые им задачи решаются другими средствами.

Программы для работы с реестром

- regedit.exe
- regedt32.exe
- Утилиты сторонних производителей (Regcleaner, Norton Utilites)



ОКНО РЕДАКТОРА РЕЕСТРА

Для просмотра и модификации данных реестра имеется штатный утилиты редактор реестра regedit.

Редактирование РЕЕСТРА

- Изменение значений параметров реестра
- Сохранение копии реестра
- Экспорт и импорт настроек
- REG – файлы (файлы с расширением reg)

запустите REG-файл как обычную программу, будет выдан запрос о необходимости провести изменения в реестре, и после ответа информация из файла будет импортирована.

```
Windows Registry Editor Version 5.00

[HKEY_CLASSES_ROOT\Software\CLASSES\PROGDATA.4]
@="581907209372"

[HKEY_CLASSES_ROOT\Software\CLASSES\PROGDATA.5]
@="01533"
```

Структура REG файла

Текстовый файл, в первой строке которого написано:

- **REGEDIT4**

или Windows Registry Editor Version 5.00

Кроме этого в первой строке ничего быть не должно!

После этого текста **ОБЯЗАТЕЛЬНО** должна быть пустая строка. Затем, указывается раздел реестра, в котором надо прописать или изменить какие-то параметры. Название раздела должно быть заключено в квадратные скобки
REGEDIT4

[Razdel1]
"param1"="znachenie1"
"param2"="znachenei2"
"param3"="znachenie3"

[Razdel2]
"param_1"="znachenie_1"

Если надо провести изменения в нескольких разделах, то вы должны оставлять одну пустую строку между последним параметром предыдущего раздела и названием следующего раздела.

Последняя строка в файле должна быть пустой.

Параметры REG - файла

- Какие параметры можно можно добавлять?
- **Строковые:**
  “param1”="znachenie".
  Двоичные
  "param"=hex:XX,XX,XX,.... – вместо X
- **Dword**
  "param"=dword:XXXXXXXX после двоеточия следует значение из восьми цифр в шестнадцатеричном (!) формате. Однако большинство параметров dword имеют значение 0, либо 1. **Пробелы в строке не допускаются.**

- **По умолчанию** ("Default")
  @="znachenie"

  значок @ НЕ ЗАКЛЮЧАЕТСЯ в кавычки.

"FIXEDFON.FON"="vgafixr.fon"

"param"=hex:be,00,00,00

"Start Page" = "http://windowos.info"

Примеры REG-файлов

```
REGEDIT4

[HKEY_CURRENT_USER\Software\ Microsoft\Internet Explorer\Main]
"Start Page" = "http://windowos.info"
```

Удаление раздела

надо перед его именем в квадратных ско

```
REGEDIT4

[-HKEY_LOCAL_MACHINE\Software\ QuickSoft\QuickStart]
```

Благодаря этой записи, подраздел "QuickStart" из раздела "QuickSoft" будет удален со всем содержимым.

Примеры REG-файлов

Для удаления отдельных параметров используется следующий синтаксис:

```
REGEDIT4

[HKEY_CURRENT_USER\Software]
"xxx"=-
```

```
Windows Registry Editor Version 5.00

[HKEY_CURRENT_CONFIG\Software\Fonts]
"FIXEDFON.FON"="vgafixr.fon"
"FONTS.FON"="vgasysr.fon"
"OEMFONT.FON"="vga866.fon"
"LogPixels"=dword:00000060
```

Параметры командной строки

- Редактор реестра regedit можно запускать с некоторыми ключами:

  /s (импортирует значения из reg-файла без вывода диалогового окна)

  /e (экспортирует параметры в reg-файл.
- Пример:

  regedit /e myfile.reg

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.

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.

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.

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.

Windows API

Соответствие Win32 API и UNIX

Win32 API	UNIX	Описание
CreateFile	open	создать новый файл или открыть существующий
DeleteFile	unlink	удалить существующий файл
CloseHandle	close	закрыть файл
ReadFile	read	прочитать данные из файла
WriteFile	write	записать данные в файл
SetFilePointer	lseek	установить указатель работы с файлом
GetFileAttributes	stat	вернуть аттритбуты файла
LockFile	fcntl	заблокировать регион файла для решения взаимного исключения
UnlockFile	fcntl	разблокировать регион файла

Соответствие Win32 API и UNIX

Win32 API	UNIX	Описание
CreateDirectory	mkdir	создать новый каталог
RemoveDirectory	rmdir	удалить пустой каталог
FindFirstFile	opendir	выполнить инициализацию для начала чтения записей каталога
FindNextFile	readdir	прочитать следующую запись каталога
MoveFile	rename	перенести файл из одного каталога в другой
SetCurrentDirectory	chdir	изменить текущий каталог

Работа с каталогами и файлами

Функция	Выполняемое действие
GetCurrentDirectory	Получение текущего каталога
SetCurrentDirectory	Смена текущего каталога
GetSystemDirectory	Получение системного каталога
GetWindowsDirectory	Получение основного каталога системы
CreateDirectory	Создание каталога
RemoveDirectory	Удаление каталога
CopyFile	Копирование файла
MoveFile MoveFileEx	Перемещение или переименование файла
DeleteFile	Удаление файла

Вопросы, жалобы, предложения?

Подготовил
к.т.н. Павлович А.А.

Задание для самостоятельной работы

1. Windows API – для чего нужен, где используется….
2. Win32 API группы
3. Принципы, лежащие в основе Windows API
4. типы данных в Windows API
5. функции библиотек
6. Работа с файлами, дисками в Windows API
7. Ввод/вывод
8. Реестр
9. Установить требуемое ПО для лабораторных, пз