MSDN: системные сервисы (часть 1)

Мартынов Семён

Санкт-Петербургский государственный политехнический университет

semen.martynov@gmail.com

2 апреля 2015 г.

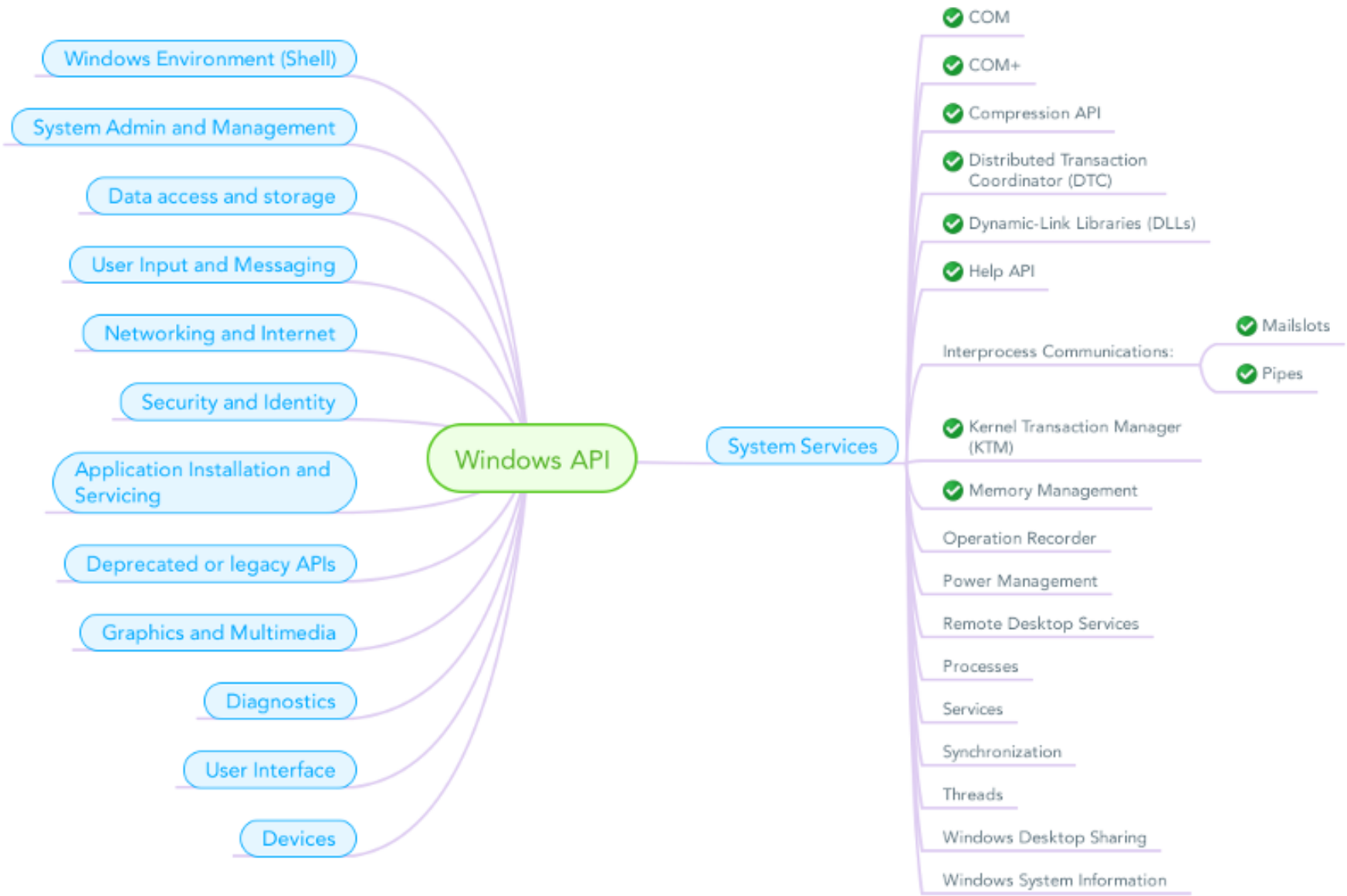
Содержание

Системные сервисы

Системные сервисы предоставляют приложениям доступ к ресурсам компьютера и операционной системы, таким как память, файловые системы, устройства, процессы и потоки. Приложения используют этот функционал для управления и мониторинга ресурсов, необходимых им для выполнения своей работы. Управление процессами и функции синхронизации координируют работу нескольких приложений или нескольких потоков выполнения в рамках одного приложения.

Мы кратко рассмотрим COM/COM+, API сжатия, координацию распределённых транзакций (DTC), работу с DLL, API справки, некоторые средства IPC (почтовые слоты и каналы), менеджер транзакций ядра (KTM) и управление памятью. Рассмотрение подразумевает перечисление некоторых особенностей, за полной справкой стоит обратиться в MSDN.

Рассматриваемые системные средства



COM/COM+

COM

Component Object Model (объектная модель компонентов) - это технологический стандарт, предназначенный для создания программного обеспечения на основе взаимодействующих компонентов, каждый из которых может использоваться во многих программах одновременно.

COM+

Расширение стандартной COM модели, появившиеся в составе Windows 2000. Среди изменений можно отметить:

- автоматический пул потоков (создаваемый стандартным процессом-загрузчиком mtx.exe)
- доступ к контексту, в котором выполняется компонент
- интеграция с транзакциями монитора MS DTC (контекст COM+ может автоматически содержать в себе транзакцию MS DTC)

COM/COM+ сравнение

Одной из наиболее важных черт COM является ее способность предоставлять двоичный стандарт для программных компонентов.

C++-объект (экземпляр класса)	COM-объект
Позволяет использовать только один общий интерфейс, представляющий собой множество C++ - методов	Обычно предоставляет более одного общего интерфейса
Зависит от языка программирования	Обеспечивается независимость от языка - COM-объекты реализуются и используются в различных языках программирования
Отсутствует встроенная возможность проверки версии	Поддерживается встроенный способ проверки версий объектов (независимо от положения в файловой системе)

COM/COM+: список сущностей

COM состоит из следующих сущностей:

- Константы
- Нумераторы
- Функции
- Интерфейсы
- Макросы
- Записи реестра
- Структуры

COM/COM+: Константы

- Флаги доступа - ACTRL_ACCESS_ALLOWED и ACTRL_ACCESS_DENIED;
- Флаги уровня аутентификации - определяют уровень аутентификации для обеспечения защиты и целостности данных;
- Сервисные константы аутентификации - определяют службы проверки подлинности (NTLMSSP, Kerberos, Schannel);
- Константы авторизации - определяют то, что авторизует сервер (к примеру, имя пользователя);
- Коды ошибок - представляют список кодов ошибок, используемых API на базе COM, числовые значения кодов определены в файле Winerror.h;
- Константы уровня представления - определяют уровень полномочий сервера, когда он выполняет роль клиента (анонимный, авторизированный, делегированный).

COM/COM+: Нумераторы

Их достаточно большое количество, но у них простая структура и понятные названия. Пример нумератор BINDSPEED, который показывает, как долго клиент будет ждать захвата объекта.

```
typedef enum tagBINDSPEED {
    BINDSPEED_INDEFINITE  = 1,
    BINDSPEED_MODERATE    = 2,
    BINDSPEED_IMMEDIATE   = 3
} BINDSPEED;
```

COM/COM+: Функции

В распоряжении разработчика более 130 функций с диким количеством параметров! Запомнить их все крайне сложно.

```
HRESULT CoGetInstanceFromFile(
    _In_opt_  COSERVERINFO *pServerInfo,
    _In_opt_  CLSID *pClsid,
    _In_opt_  IUnknown *punkOuter,
    _In_      DWORD dwClsCtx,
    _In_      DWORD grfMode,
    _In_      OLECHAR *pwszName,
    _In_      DWORD dwCount,
    _Inout_   MULTI_QI *pResults
);
```

Они в основном управляет ресурсами, библиотеками, доступом и состоянием.

COM/COM+: Интерфейсы

Больше 80 интерфейсов, обладающих различным набором функций.

```
typedef enum tagBINDSPEED {
    BINDSPEED_INDEFINITE  = 1,
    BINDSPEED_MODERATE    = 2,
    BINDSPEED_IMMEDIATE   = 3
} BINDSPEED;
```

COM/COM+: Макросы

Больше 80 интерфейсов, обладающих различным набором функций.

Пример - два макроса, проверяющих существование описателя.

```
#define FAILED(hr) (((HRESULT)(hr)) < 0)

#define SUCCEEDED(hr) (((HRESULT)(hr)) >= 0)
```

COM/COM+: Записи реестра

Аспектами функциональности COM управляют значения ключей в следующих ветках реестра:

- `HKEY_LOCAL_MACHINE\SOFTWARE\Classes`
  - Поддерживаемые форматы данных, информация о совместимости, программируемые идентификаторы и средства управления.
- `HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Ole`
  - Параметры настройки права доступа и возможности безопасного вызова COM приложений.
- `HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Windows NT\CurrentVer`
  - Содержат информацию, связанную с функциональностью RPC COM.

COM/COM+: Структуры

Около 30 структур, связанных с авторизаций и аутентификацией.

Пример - структура ACTRL_ACCESS:

```
typedef struct _ACTRL_ACCESS {
    ULONG                   cEntries;
    PACTRL_PROPERTY_ENTRY pPropertyAccessList;
} ACTRL_ACCESS, *PACTRL_ACCESS;
```

Первое поле указывает на количество, а второе на начало массива записей ACTRL_PROPERTY_ENTRY, каждая из которых содержит список записей, определяющих доступ к какому-либо объекту.

Compression API

Сжатие позволяет выиграть пространство на диске за счёт тактов процессора, которые будут потрачены на декомпессию. Windows предоставляет следующие алгоритмы сжатия:

- XPRESS (COMPRESS_ALGORITHM_XPRESS) - очень быстрый и не требовательный к системным ресурсам
- XPRESS с кодированием Хафмена (COMPRESS_ALGORITHM_XPRESS_HUFF) - сжимает чуть лучше предыдущего
- MSZIP (COMPRESS_ALGORITHM_MSZIP) - требует больше ресурсов, но обеспечивает хороший процент компрессии
- LZMS (COMPRESS_ALGORITHM_LZMS) - алгоритм подходит для средних (от 2 МБ) и больших объёмов данных

Compression API: Процедура сжатия

```
CreateCompressor(// Создание компрессора
  // Алгоритм компрессии LZMS, блочный режим
  COMPRESS_ALGORITHM_LZMS|COMPRESS_RAW,
  // Опционально указывается алокатор памяти
  &AllocationRoutines,
  &Compressor); // Описатель компрессора

Compress(         // Процедура сжатия блока.
  Compressor,     // Описатель компрессора
  InputData,      // Входной буфер, не сжатые данные
  BlockSize,      // размер не сжатого блока
  *OutputData,    // Буфер сжатия
  0,              // Размер буфера сжатия
  &CompressedBlockSize); // Размер сжатого блока
```

При сжатии нужно перемещаться по входному (не сжатому) и выходному (сжатому) буферу, а после сохранить сжатый буфер.

Compression API: Процедура декомпрессии

```
CreateDecompressor( // Создание декомпрессора
  // Алгоритм компрессии LZMS, блочный режим
  COMPRESS_ALGORITHM_LZMS|COMPRESS_RAW,
  // Опционально указывается алокатор памяти
  &AllocationRoutines,
  &Compressor); // Описатель декомпрессора

Decompress(     // Процедура сжатия блока.
  Decompressor, // Описатель декомпрессора
  InputData,    // Входные данные, сжатые
  CompressedBlockSize,   // Размер сжатого блока
  *OutputData,  // Выходные данные, не сжатые
  UncompressedBlockSize, // Размер не сжатого блока
  NULL);        // Количество разархивированных данных
```

Как и раньше, нужно обеспечить смещение по каждому буферу и сохранить результат.

Distributed Transaction Coordinator (DTC)

Координатор распределённых транзакций (DTC) - компонент Microsoft Windows, предназначенный для координации изменения данных на двух или более сетевых компьютерных системах.

Координатор распределённых транзакций основан на технологии COM+ и включает в себя:

- менеджер транзакций;
- журнал транзакций;
- прокси (реализует интерфейсы DTC);
- утилиты администрирования;
- заголовочные файлы API.

Distributed Transaction Coordinator (DTC)

При запросе фиксации или отката транзакции, менеджером транзакций выполняется двухфазный протокол фиксации. Во время первой фазы менеджеру ресурсов посылается запрос на подготовку к завершению, во время второй — на фиксацию или откат транзакции. По дереву, образованному вышестоящими и подчинёнными системами, рассылаются сообщения для подготовки к завершению, фиксации или отката. Любой узел дерева может прервать транзакцию до подтверждения подготовки к завершению.

После того, как узел подтвердил подготовку, он остаётся в этом состоянии до фиксации или отката транзакции вышестоящим узлом. В случае сбоя и перезагрузки компьютера, менеджер транзакций запрашивает вышестоящий узел о дальнейшей судьбе подготовленных к завершению транзакций.

Dynamic-Link Libraries (DLLs)

Ниже перечислены функции, которые используются в динамическом связывании:

- DisableThreadLibraryCalls - отключает уведомления DLL_THREAD_ATTACH и DLL_THREAD_DETACH для указанной динамически подключаемой библиотеки (DLL).
- DllMain - дополнительная точка входа в динамически подключаемую библиотеку (DLL).
- FreeLibrary - уменьшает итоговое число ссылок на загруженные динамически подключаемые библиотеки (DLL). Когда итоговое число ссылок достигает нуля, модуль отменяет отображение в адресном пространстве вызывающего процесса.
- FreeLibraryAndExitThread - уменьшает итоговое число ссылок загруженной динамически подключаемой библиотеки (DLL) до единицы, также, как это делает FreeLibrary , затем вызывает ExitThread, чтобы завершить работу вызывающего потока.

Dynamic-Link Libraries (DLLs)

Продолжение:

- GetDllDirectory - извлекает конкретную для приложения часть пути поиска, используемого, чтобы определить местонахождение DLLs для прикладной программы.
- GetModuleFileName - извлекает полный путь доступа к файлу, содержащему указанный модуль, которым владеет текущий процесс.
- GetModuleFileNameEx - извлекает полный путь доступа к файлу, содержащему заданный модуль.
- GetModuleHandle - извлекает дескриптор указанного модуля, если файл был отображён в адресном пространстве вызывающего процесса.
- GetModuleHandleEx - извлекает дескриптор указанного модуля, если файл был отображён в адресное пространство вызывающего процесса.

Dynamic-Link Libraries (DLLs)

Продолжение:

- GetProcAddress - извлекает адрес экспортируемой функции или переменной от заданной динамически подключаемой библиотеки (DLL).
- LoadLibrary - отображает заданный исполняемый модуль в адресное пространство вызывающего процесса.
- LoadLibraryEx - отображает указанный исполняемый модуль в адресное пространство вызывающего процесса.
- SetDllDirectory - добавляет каталог к пути поиска, используемый, чтобы определить местонахождение DLL для прикладной программы.

Help API

Справочное API позволяет открывать справочные каталоги, и получать оттуда материалы справки, такие как:

- Индексируемые справочные кончпекты (XHTML, HTML)
- Не индексируемые изображения
- Файлы CSS
- Файлы JavaScript
- Аудио/видео файл

Help API

Работа Help API осуществляется через ряд специфичных интерфейсов для объектов:

- ICatalog - интерфейс для объекта, хранящего состояние, т.е. хранящего описатель открытого каталога и всю информацию о нём
- ICatalogRead - интерфейс для объекта, не хранящего своё состояние, т.е. обработка каталога осуществляется на основе полученных во время работы параметров
- ICatalogReadWriteLock - интерфейс для объекта, блокирующего хранимый каталог, т.е. в процессе работы устанавливается замок на дескриптор каталога и всю информацию о нем
- IHelpFilter - коллекция критериев-фильтров, образованных парами ключ-значение
- IHelpKeyValuePair - пара ключ-значение объектов IHelpFilter

Help API

Продолжение:

- Ikeyword - интерфейс поиска по ключевым словам, содержит методы поиска информации по ключевым словам
- IKeywordCollection – коллекция интерфейсов IKeyword
- ITopic - интерфейс тематического поиска, содержит методы поиска по заданной теме
- ITopicCollection - коллекция интерфейсов Itopics
- IndexException - интерфейс который получает свои характеристики от интерфейса IDispatch (общий интерфейс COM)

Interprocess Communications

Операционная система Microsoft ® Windows ® предусматривает механизмы, которые облегчают обмен совместно использующейся информацией и данными между приложениями. В собирательном значении - это действия, включающие в работу механизмы, называемые межпроцессными взаимодействиями (interprocess communications) (IPC).

Некоторые формы межпроцессного взаимодействия (IPC) облегчают разделение задания между несколькими специализированными процессами. Другие формы межпроцессорного взаимодействия (IPC) облегчают разделение задания между компьютерами в сети.

Mailslots

Почтовый слот в ядре Windows предоставляет разностороннюю связь. Любой процесс, который создает почтовый слот – это сервер почтового слота (mailslot server). Другие процессы, называемые клиентами почтового слота (mailslot clients), отправляют сообщения серверу почтового слота, записывая сообщение в его почтовом ящике.

Входящие сообщения всегда добавляются в конец почтового слота. Почтовый слот в ядре Windows сохраняет сообщения до тех пор, пока сервер слота не прочтёт их.

Mailslots

Клиент почтового слота может отправлять сообщение почтовому слоту на своем локальном компьютере, почтовому слоту на другом компьютере или всем почтовым слотам в ядре Windows с тем же самым именем на всех компьютерах в заданном сетевом домене. Сообщения, транслируемые всем почтовым слотам ядра Windows в домене, не могут быть длиннее, чем 400 байтов, принимая во внимание то, что сообщения, передаваемые в отдельно взятый почтовый слот, ограничиваются только максимальным размером сообщения, определяемым сервером почтового слота, когда он создавал этот слот в ядре Windows.

Mailslots

```
HANDLE CreateMailslot(    // Создание Mailslot
  LPCTSTR lpName,         // имя
  DWORD nMaxMessageSize,  // максимальный размер
  DWORD lReadTimeout,     // интервал тайм-аута чтения
  LPSECURITY_ATTRIBUTES lpSecurityAttributes //безопасность
);

BOOL GetMailslotInfo(       // Проверка данных
  HANDLE hMailslot,         // указатель на слот
  LPDWORD lpMaxMessageSize, // максимальный размер
  LPDWORD lpNextSize,       // размер следующего
  LPDWORD lpMessageCount,   // количество сообщений
  LPDWORD lpReadTimeout);   // тайм-аут.
```

Pipes

Имеются два типа каналов (абстрактных файлов) для двусторонней связи: анонимные (неименованные) и именованные каналы. Анонимные (или без имени) каналы (Anonymous pipes) дают возможность родственным процессам передавать информацию друг другу. Как правило, неименованный канал используется для переназначения стандартного ввода или вывода данных дочернего процесса так, чтобы он мог обмениваться данными со своим родительским процессом. Чтобы обмениваться данными в обоих направлениях (дуплексная работа), нужно создать два анонимных канала.

Анонимные каналы не могут быть использованы в сети, и при этом они не могут быть использованы между несвязанными процессами.

Pipes

Именованные каналы (Named pipes) используются для передачи данных между процессами, которые являются не связанными процессами и между процессами на разных компьютерах. Как правило, процесс сервера именованного канала (named-pipe server) создаёт именованный канал с известным именем или именем, которое должно быть сообщено его клиентам.

Процесс-клиент именованного канала (named-pipe client), который знает имя канала (абстрактного файла), может открыть его с другого конца, подчиняясь ограничениям доступа, которые определяются процессом сервера именованного канала. После этого, и сервер, и клиент присоединяются к каналу, они могут обмениваться данными, выполняя операции чтения и записи в канале.

Pipes

```
BOOL CreatePipe (    // Создание анонимного канала
  PHANDLE phRead,    // дескриптор файла для чтения
  PHANDLE phWrite,   // дескриптор файла для записи
  LPSECURITY_ATTRIBUTES lpsa,// безопасность
  DWORD nsize)       // опциональный размер канала

BOOL WINAPI ReadFile(// Чтение из канала
  HANDLE hFile,       // файл, из которого читаем
  LPVOID lpBuffer,    // буфер-приемник данных
  DWORD nNumberOfBytesToRead, // размер буфера
  LPDWORD lpNumberOfBytesRead,// сколько прочитано
  LPOVERLAPPED lpOverlapped); // асинхронный обмен
```

Kernel Transaction Manager (KTM)

Транзакционная работа с файлами и реестром обеспечивается компонентом Kernel Transaction Manager. Не смотря на название, его можно использовать и из пользовательского режима. Более того, KTM поддерживает Microsoft Distributed Transaction Coordinator (MSDTC), таким образом, транзакции могут быть распределёнными двухфазными и затрагивать сразу несколько машин.

Kernel Transaction Manager основывается на Common Log File System (CLFS) и не ограничивает использование транзакций конкретными типами ресурсов. Предоставляются стандартизированные реализации транзакционной работы с файловой системой и реестром. Вполне возможно добавить поддержку новых типов ресурсов самостоятельно. Таким образом, базовые функции KTM не специализированы, и их использование одинаково вне зависимости от типа ресурсов, над которыми производятся транзакции. Более того, в рамках одной транзакции допустимо обращаться к разным типам ресурсов.

Kernel Transaction Manager (KTM)

В рамках транзакции можно создавать и удалять файлы, менять их содержимое и атрибуты. Все эти изменения будут отменены в случае отката транзакции. Поддерживается также транзакционная работа с жёсткими и символическими ссылками.

Следует отметить особую возможность — миниверсии. В рамках транзакции файл можно повторно открыть на чтение в состоянии до начала транзакции. Данная возможность позволяет выполнять сложные обновления файла на основе его текущего содержимого, без создания временных файлов или угрозы нарушения согласованности данных.

```
HANDLE hTrans = CreateTransaction( // создание транзакции
     NULL,0, 0, 0, 0, NULL, _T("My NTFS Transaction"));

USHORT view = 0xFFFE; // Открытие на запись (изменение)
HANDLE hFile = CreateFileTransacted(
     _T("test.file"), GENERIC_WRITE, 0, NULL, CREATE_ALWAYS,
     0, NULL, hTrans, &view, NULL); // запуск транзакции

DWORD dwWritten = 0; // Сколько реально данных записано
string str = "Test String"; // данные файла
WriteFile(hFile,str.c_str(),str.length(), &dwWritten, NULL);
CloseHandle(hFile); // остановка транзакции

CommitTransaction(hTrans);// или RollbackTransaction(hTrans);
CloseHandle(hTrans);
```

Memory Management

Управление динамической памятью в той или иной форме требуется в большинстве программ. Необходимость в этом возникает всякий раз, когда требуется создавать структуры данных, размер которых не может быть определен заранее на стадии создания программы. Типичными примерами динамических структур данных могут служить деревья поиска, таблицы имен и связанные списки.

В Windows предусмотрены гибкие механизмы управления динамической памятью программы. На уровне WinAPI есть ряд функций, которые позволяют сообщать ядру о том, как нужно управлять памятью. Данные о том, какое количество памяти является реальным, а какое виртуальным может потребоваться системным твикерам или высокопроизводительным приложениям.

Memory Management

Список функций для управления памятью:

- AddSecureMemoryCacheCallback - регистрирует функцию обратного вызова, чтоб обратиться к ней, когда освободится надёжный объем памяти или изменятся параметры надёжности
- CopyMemory - копирует блок памяти из одного места в другое
- CreateMemoryResourceNotification – создаёт объект оповещения о ресурсах памяти
- FillMemory – заполняет блок памяти присвоенным значением
- GetLargePageMinimum – находит минимальный большой размер страницы (если процессор поддерживает большие страницы)

Memory Management

Продолжение:

- GetPhysicallyInstalledSystemMemory — находит объем оперативной памяти, установленной на компьютере
- GetSystemFileCacheSize - находит текущие пределы размера для рабочих настроек системного кэша
- GetWriteWatch - находит адреса страниц, к которым было произведено обращение на запись (в области виртуальной памяти)
- GlobalMemoryStatusEx - обладает информацией о текущем использовании физической и виртуальной памяти
- MoveMemory - перемещает блок памяти из одного места в другое
- QueryMemoryResourceNotification - определяет состояние некого объекта в памяти

Memory Management

Продолжение:

- RemoveSecureMemoryCacheCallback — отменяет регистрацию функции обратного вызова, ранее зарегистрированную с помощью функции AddSecureMemoryCacheCallback
- ResetWriteWatch — отключает отслеживание состояния раздела виртуальной памяти
- SecureMemoryCacheCallback — функция приложения, которая вызывается, когда освобождается надёжный объем памяти или изменятся параметры надёжности
- SecureZeroMemory — заполняет блок памяти нолями
- SetSystemFileCacheSize — ограничивает размер рабочих настроек для кэша файловой системы
- ZeroMemory - заполняет блок памяти нолями

Заключение

Далее следует рассматривать следующие разделы MSDN:

- Operation Recorder (Контроллер записи)
- Power Management (Управление энергопотреблением)
- Remote Desktop Services (удалённый рабочий стол)
- Processes (Процессы)
- Services (Службы)
- Synchronization (Синхронизация)
- Threads (Потоки)
- Windows Desktop Sharing (совместное использование рабочего стола)
- Windows System Information (Информация о системе)
  - Handle and Objects (Описатели и объекты):
  - Registry (Реестр)
  - Time (Время)
  - Time Provider (Служба времени)

Ссылки

- COM - https://msdn.microsoft.com/ms693341
- COM+ - https://msdn.microsoft.com/ms687816
- Compression API - https://msdn.microsoft.com/hh437596
- Distributed Transaction Coordinator (DTC) - https://msdn.microsoft.com/ms686108
- Dynamic-Link Libraries (DLLs) - https://msdn.microsoft.com/ms682599
- Help API - https://msdn.microsoft.com/hh447319
- Interprocess Communications: - https://msdn.microsoft.com/aa365574
  - Mailslots - https://msdn.microsoft.com/aa365580
  - Pipes - https://msdn.microsoft.com/aa365781
- Kernel Transaction Manager (KTM) - https://msdn.microsoft.com/aa366288
- Memory Management - https://msdn.microsoft.com/aa366782

Вопросы?