Лабораторная работа №4.Память и файлы.

Операции над виртуальной памятью

Порядок выполнения работы:
1. Изучить Функции **GlobalMemoryStatus** и **GetSystemInfo**

2. Написать консольное Win32 приложение, в соответствии с полученным вариантом.

Вариант 1. Определить количество страниц физической памяти.

Вариант 2. Определить количество страниц свободной физической памяти

Вариант 3. Определить количество виртуальных страниц в свободной виртуальной памяти процесса.

Вариант 4. Определить количество виртуальных страниц в файле подкачки.

Вариант 5. Определить количество свободных виртуальных страниц в файле подкачки.

Вариант 6. Определить количество страниц физической памяти, занятых ОС и приложениями.

Вариант 7. Определить количество байтов доступных процессу. Ответ вывести в килобайтах.

Вариант 8. Определить процентное соотношение используемой физической памяти и общего объема физической памяти.

Вариант 9. Определить общее количество доступной физической и дисковой памяти; определить процент полученного объема памяти и максимально возможного объема оперативной памяти.

Вариант 10. Определить процентное соотношение числа виртуальных страниц физической памяти и страниц файла подкачки.

Вариант 11. Определить общий объем виртуальной памяти системы.

GetSystemInfo

The **GetSystemInfo** function returns information about the current system.

```
VOID GetSystemInfo(
  LPSYSTEM_INFO lpSystemInfo   // address of system information
                               // structure
);
```

Parameters

lpSystemInfo
Pointer to a **SYSTEM_INFO** structure to be filled in by this function.

Return Values

This function does not return a value.

Функция GetSystemInfo возвращает информацию о текущей системе.

```
VOID GetSystemInfo(  LPSYSTEM_INFO lpSystemInfo   //
адрес структуры для системной информации  );
```

Подробное описанеи см. В тексте задания к лабораторной работе №1.

Функция GlobalMemoryStatus

Функция GlobalMemoryStatus из библиотеки Kernel32.dll предоставляет информацию об использовании физической и виртуальной памяти компьютера. Вот её прототип:

```
VOID GlobalMemoryStatus(LPMEMORYSTATUS lpBuffer);
// Адрес структуры MemoryStatus
```

Тип данных LPMEMORYSTATUS - это указатель на структуру *MemoryStatus*:

```
typedef struct _MEMORYSTATUS {
    DWORD dwLength;     // Размер структуры
    DWORD dwMemoryLoad; // Процент использования памяти
    DWORD dwTotalPhys;  // Физическая память, байт
    DWORD dwAvailPhys;  // Свободная физическая память, байт
    DWORD dwTotalPageFile;   // Размер файла подкачки, байт
    DWORD dwAvailPageFile;   // Свободных байт в файле подкачки
    DWORD dwTotalVirtual;    // Виртуальная память, используемая
процессом
    DWORD dwAvailVirtual;   // Свободная виртуальная память
    } MEMORYSTATUS, *LPMEMORYSTATUS;
```

Поле	Смысл

dwLength	Длина записи. Поле необходимо инициализировать функцией SizeOf до обращения к функции GlobalMemoryStatus
dwMemoryLoad	Количество использованной памяти в процентах
dwTotalPhys	Число байт установленной на компьютере ОЗУ (физической памяти)
dwAvailPhys	Свободная физическая память в байтах
dwTotalPageFile	Общий объем в байтах, который могут сохранить файлы/файл подкачки (вообще говоря, не совпадает с размером последних)
dwAvailPageFile	Доступный объем из последней величины в байтах
dwTotalVirtual	Общее число байтов виртуальной памяти, используемой в вызывающем процессе
dwAvailVirtual	Объем виртуальной памяти, доступной для вызывающего процесса

Работа с виртуальной памятью (для самостоятельного изучкемя)

Для того чтобы зарезервировать или получить в свое распоряжение некоторое количество страниц виртуальной памяти, приложение должно воспользоваться функцией VirtualAlloc, прототип которой представлен ниже:

```
LPVOID VirtualAlloc(
  LPVOID lpvAddress,        // адрес области
  DWORD  cbSize,            // размер области
  DWORD  fdwAllocationType, // способ получения памяти
  DWORD  fdwProtect);       // тип доступа
```

Параметры `lpvAddress` и `cbSize` задают, соответственно, начальный адрес и размер резервируемой либо получаемой в пользование области памяти. При резервировании адрес округляется до ближайшей границы блока размером 64 Кбайт. В остальных случаях адрес округляется до границы ближайшей страницы памяти.

Заметим, что параметр `lpvAddress` можно указать как NULL. При этом операционная система выберет начальный адрес самостоятельно.

Что же касается параметра cbSize, то он округляется до целого числа страниц. Поэтому если вы пытаетесь с помощью функции VirtualAlloc получить область памяти размером в один байт, вам будет выделена страница размером 4096 байт. Аналогично, при попытке получить блок памяти размером 4097 байт вы получите две страницы памяти общим размером 8192 байта. Как мы уже говорили, программный интерфейс системы управления виртуальной памятью не предназначен для работы с областями малого размера.

Для параметра `fdwAllocationType` вы можете использовать одно из следующих значений:

Значение	*Описание*
MEM_RESERVE	Функция VirtualAlloc выполняет резервирование диапазона адресов в адресном пространстве приложения
MEM_COMMIT	Выполняется выделение страниц памяти для непосредственной работы с ними. Выделенные страницы заполняются нулями
MEM_TOP_DOWN	Память выделяется в области верхних адресов адресного пространства приложения

С помощью параметра fdwProtect приложение может установить желаемй тип доступа для заказанных страниц. Можно использвать одно из следующих значений:

Значение	*Разрешенный доступ*

PAGE_READWRITE	Чтение и запись
PAGE_READONLY	Только чтение
PAGE_EXECUTE	Только исполнение программного кода
PAGE_EXECUTE_READ	Исполнение и чтение
PAGE_EXECUTE_READ WRITE	Исполнение, чтение и запись
PAGE_NOACCESS	Запрещен любой вид доступа
PAGE_GUARD	Сигнализация доступа к старнице. Это значение можно использовать вместе с любыми другими, кроме PAGE_NOACCESS
PAGE_NOCACHE	Отмена кэширования для страницы памяти. Используется драйверами устройств. Это значение можно использовать вместе с любыми другими, кроме PAGE_NOACCESS

Если страница отмечена как PAGE_READONLY, при попытке записи в нее возникает аппаратное прерывание защиты доступа (access violation). Эта страница также не может содержать исполнимый код. Попытка выполнения такого кода приведет к возникновению прерывания.
С другой стороны, у вас есть возможность получения страниц, предназначенных только для хранения исполнимого кода. Если такие страницы отмечены как PAGE_EXECUTE, для них не разрешаются операции чтения и записи.
При необходимости зафиксировать обращение к той или иной странице приложение может отметить ее как PAGE_GUARD. Если произойдет попытка обращения к такой странице, возникнет исключение с кодом STATUS_GUARD_PAGE, после чего признак PAGE_GUARD будет автоматически сброшен.
В случае успешного завершения функция VirtualAlloc возвратит адрес зарезервированной или полученной области страниц. При ошибке будет возвращено значение NULL.
Приложение может вначале зарезервировать страницы, вызвав функцию VirtualAlloc с параметром MEM_RESERVE, а затем получить их в пользование, вызвав эту же функцию еще раз для полученной области памяти, но уже с параметром MEM_COMMIT
После использования вы должны освободить полученную ранее виртуальную память, вызвав функцию VirtualFree:

```
BOOL VirtualFree(
  LPVOID lpvAddress,   // адрес области
  DWORD  cbSize,       // размер области
  DWORD  fdwFreeType); // выполняемая операция
```

Через параметры `lpvAddress` и `cbSize` передаются, соответственно, адрес и размер освобождаемой области.
Если вы зарезервировали область виртуальной памяти функцией `VirtualAlloc` с параметром MEM_RESERVE для последующего получения страниц в пользование и затем вызвали эту функцию еще раз с параметром MEM_COMMIT, вы можете либо совсем освободить область памяти, обозначив соответствующие страницы как свободные, либо оставить их зарезервированными, но не используемыми.
В первом случае вы должны вызвать функцию VirtualFree с параметром fdwFreeType, равным MEM_RELEASE, во втором - с параметром MEM_DECOMMIT.

Файловая система. Работа с деревом каталогов. Атрибуты файлов и каталогов.

Операционные системы семейства Windows для каждого логического диска формируют корневой каталог, хранящий данные о дереве каталогов данного диска.
Для логического диска C: корневым является каталог C:\, а следующая строка обозначает «все файлы и каталоги из корневого» – C:* или C:*.*. (последняя точка соответствует концу предложения)
Информацию об именах логических дисков можно получить, используя функции **GetLogicalDrives** или **GetLogicalDriveStrings** (см. данные к первой лабораторной работе).
Обычно, для рекурсивного обхода дерева каталогов последовательно используются функции **FindFirstFile** и **FindNextFile**.

Функция **FindFirstFile** ищет в каталоге файл или каталог в соответствии с указанным именем.

```
HANDLE FindFirstFile(
     LPCTSTR lpFileName,
     LPWIN32_FIND_DATA lpFindFileData);
```

Параметры:

lpFileName [in]	Указатель на нуль-завершенную строку, содержащую имя файла или каталога. Строка может содержать символы * и(или) ? Возможны следующее варианты содержимого этой строки: • "C:\\WINDOWS" – получить информацию о каталоге C:\WINDOWS • "C:*" – получить информацию о первом файле или каталоге в корневом каталоге диска C:. • "D:\\MYDIR*.DOC" – получить информацию о первом файле с расширением DOC в каталоге D:\MYDIR. • "D:\\MYDIR\\??*.DOC" – получить информацию о первом файле с расширением DOC имя, которого состоит не менее чем из двух символов, хранящемся в каталоге D:\MYDIR. • "C:\\" – Неверное значение. Количество символов в строке ограничено константой **MAX_PATH**. Значение этой константы можно узнать в файле **windef.h**
lpFindFileData [out]	Адрес структуры WIN32_FIND_DATA, в которую будет помещена информация о найденном файле или каталоге.

Возвращаемые значения:
Если функция успешно завершила работу, она возвращает дескриптор поиска файла или каталога, указанного в первом параметре. В дальнейшем это значение можно использовать для последующего поиска файлов или каталогов с помощью функции **FindNextFile**.

При ошибке (отсутствует заданный каталог или файл) возвращается значение **INVALID_HANDLE_VALUE**. Для получения дополнительной информации об ошибке, необходимо вызывать функцию **GetLastError**.

Замечания:
Функция FindFirstFile возвращает дескриптор поиска и заполняет структуру **WIN32_FIND_DATA** данными о первом файле или каталоге, имя которого соответствует указанному образцу. Кроме того, поиск производится только по имени файла, а не по любым его атрибутам, таким как время или тип. Поиск допускает длинные и короткие имена файла.
Если необходимо получить информацию о других файлах, отвечающим заданному условию (поиск файлов с одинаковыми расширениями или именами) используйте функцию **FindNextFile**. После того, как все необходимые манипуляции с найденным файлом или каталогом произведены, вызовите функцию **FindClose**.

WIN32_FIND_DATA
Структура описывает файл, найденный с помощью функций **FindFirstFile**, **FindFirstFileEx**, или **FindNextFile**.

```
typedef struct _WIN32_FIND_DATA {
    DWORD dwFileAttributes;
    FILETIME ftCreationTime;
    FILETIME ftLastAccessTime;
    FILETIME ftLastWriteTime;
    DWORD    nFileSizeHigh;
    DWORD    nFileSizeLow;
    DWORD    dwReserved0;
    DWORD    dwReserved1;
    TCHAR    cFileName[ MAX_PATH ];
    TCHAR    cAlternateFileName[ 14 ];
} WIN32_FIND_DATA;
```

Поля структуры:

dwFileAttributes
Описывает атрибуты найденного файла или каталога, может принимать одно или несколько из следующих значений:

Атрибут	**Значение**
FILE_ATTRIBUTE_ARCHIVE	Файл или каталог – архивный файл. Приложения используют этот атрибут, чтобы отметить файлы для резервного копирования или перемещения.
FILE_ATTRIBUTE_COMPRESSED	Файл или каталог – сжатые. Для файла, это означает, что все данные в файле сжимаются. Для каталога, это означает, что по умолчанию все создаваемые в нем новые файлы и каталоги сжимаются.
FILE_ATTRIBUTE_DIRECTORY	Дескриптор идентифицирует каталог
FILE_ATTRIBUTE_ENCRYPTED	Файл или каталог – зашифрованные. Для файла, это означает, что все данные в файле зашифрованы. Для каталога, это означает, что по умолчанию все создаваемые в нем файлы или подкаталоги шифруются.
FILE_ATTRIBUTE_HIDDEN	Файл или каталог – скрытый. Они не включаются в обычный перечень

	файлов каталога.
FILE_ATTRIBUTE_NORMAL	Файл или каталог не имеют других установленных атрибутов. Этот атрибут допустим только в том случае, если используется как единственный.
FILE_ATTRIBUTE_OFFLINE	К данным файла нельзя обратиться немедленно. Показывает, что содержимое файла было физически перемещено на недоступное (offline) устройство.
FILE_ATTRIBUTE_READONLY	Файл или каталог только для чтения. Приложения могут читать из этого файла, но не могут записать в него или удалить его. Если это каталог, то приложения не могут удалить его.
FILE_ATTRIBUTE_REPARSE_POINT	Файл имеет связанную точку монтирования.
FILE_ATTRIBUTE_SPARSE_FILE	Файл является разреженным. В файловой системе NTFS разреженные файлы позволяют программам создавать очень большие файлы, но хранить их на минимуме дискового пространства.
FILE_ATTRIBUTE_SYSTEM	Файл или каталог - часть операционной системы или используются исключительно операционной системой.
FILE_ATTRIBUTE_TEMPORARY	Файл для временного хранения данных. Файловая система предпримет попытку хранения данных не на дисках, а в памяти для ускорения доступа к ним. Временные файлы должны удаляться программами, создавшими их, как только в них больше нет необходимости

ftCreationTime

Структура **FILETIME**. Хранит время создания файла или каталога Поля структуры будут содержать нули, если файловая система не поддерживает **FILETIME** получение соответствующей информации.

ftLastAccessTime

Структура **FILETIME**. Хранит время последнего доступа к файлу или каталогу. Поля структуры будут содержать нули, если файловая система не поддерживает **FILETIME** получение соответствующей информации.

ftLastWriteTime

Структура **FILETIME**. Хранит время последней записи в файл. Поля структуры будут содержать нули, если файловая система не поддерживает **FILETIME** получение соответствующей информации.

nFileSizeHigh
Старшая часть размера файла.
nFileSizeLow
Младшая часть размера файла.
Таким образом общая длина размера файла – 64 двоичных разряда. Размер файла может быть определен по следующей формуле:
(**nFileSizeHigh * MAXDWORD**) + **nFileSizeLow**, где **MAXDWORD** – константа, заданная в файле `winnt.h`.

dwReserved0
Если один из атрибутов файла FILE_ATTRIBUTE_REPARSE_POINT, то данное поле определяет признак монтирования. В противном случае значение не определено, и не может быть использовано.

dwReserved1
Зарезервировано для использования в будущем.

cFileName
Нуль-завершенная строка, содержащая имя файла.

cAlternateFileName
Нуль-завершенная строка, содержащая альтернативное имя файла в классическом формате 8.3, например filename.ext.

FILETIME
Структура хранит 64-х разрядное значение, соответствующее количеству 100-наносекундных интервалов, прошедших с нуля часов 1 января 1601 года.

```
typedef struct _FILETIME {
    DWORD dwLowDateTime;
    DWORD dwHighDateTime;
} FILETIME;
```

Поля структуры:

dwLowDateTime
Младшие 32 разряда.
dwHighDateTime
Старшие 32 разряда.
Функция **FileTimeToSystemTime** используется для преобразования количества интервалов к естественному виду (часы, минуты, день, месяц, год).

Функция **FindNextFile** продолжает поиск файлов и каталогов, начатый вызовом функции **FindFirstFile**.

```
BOOL FindNextFile(
  HANDLE hFindFile,               // дескриптор поиска
  LPWIN32_FIND_DATA lpFindFileData
// указатель на структуру для записи данных о найденном файле (каталоге)
);
```

Параметры:

hFindFile	Дескриптор поиска, возвращенный предыдущим вызовом функции

[in]	**FindFirstFile**.
lpFindFileData [out]	Адрес структуры WIN32_FIND_DATA, в которую будет помещена информация о найденном файле или каталоге.

Пример использования функций **FindFirstFile** и **FindNextFile**.
Требуется определить число подкаталогов в заданном каталоге (d:\mydir\).

```
HANDLE              fh;
WIN32_FIND_DATA     fd;

fh = FindFirstFile("D:\\MYDIR\\*",&fd);
int count = 0;
if(fh != INVALID_HANDLE_VALUE)
{
        while(FindNextFile(fh,&fd))
                if(fd.dwFileAttributes & FILE_ATTRIBUTE_DIRECTORY)
                        count++;

        cout << count << endl;

        FindClose(fh);
}
```

Функция **FileTimeToSystemTime** преобразует 64 разрядное значение из структуры **FILETIME** в формат системного времени.

```
BOOL FileTimeToSystemTime(
  CONST FILETIME *lpFileTime,
  LPSYSTEMTIME lpSystemTime
);
```

Параметры:

lpFileTime [in]	Указатель на структуру **FILETIME**, содержащую время создания файла, последнего доступа к файлу и пр., для преобразования к системному форматы даты и времени.
lpSystemTime [out]	Адрес структуры **SYSTEMTIME** в которую будут помещены время и дата, соответствующие содержимому структуры **FILETIME**.

Замечание: функция **FileTimeToSystemTime** работает только со значениями меньшими, чем 0x8000000000000000.

SYSTEMTIME

Структура хранит время и дату в следующем формате:

```
typedef struct _SYSTEMTIME {
    WORD wYear;         //год
    WORD wMonth;        //месяц; январь = 1, февраль = 2, и так далее
    WORD wDayOfWeek;    //день недели; воскресенье = 0, понедельник = 1 и т.д.
    WORD wDay;          //число (номер дня в месяце)
    WORD wHour;         //часы
    WORD wMinute;       //минуты
    WORD wSecond;       //секунды
    WORD wMilliseconds;//миллисекунды
} SYSTEMTIME;
```

Пример: для файлов с расширением doc из каталога d:\mydir\ вывести все данные о времени последнего доступа к ним с точностью до секунды.

```
FILETIME           ft;
HANDLE             fh;
SYSTEMTIME         st;
WIN32_FIND_DATA    fd;

char rbuf[256];
fh = FindFirstFile("D:\\MYDIR\\*.DOC",&fd);
if(fh != INVALID_HANDLE_VALUE)
{
       while(FindNextFile(fh,&fd))
       {
              ft = fd.ftLastAccessTime;

              CharToOem(fd.cFileName,rbuf);
              printf(" %-40s",rbuf,rbuf14);

              FileTimeToSystemTime(&ft, &st);
              printf("%2d.%2d.%4d %2d:%2d:%2d\n",
                           st.wDay,st.wMonth,st.wYear,
                           st.wHour,st.wMinute,st.wSecond);
       }
       if(FindClose(fh)) printf("oK Find File\n");
}
```

Для получения информации об атрибутах файла, или их изменения, можно воспользоваться функциями **GetFileAttributes, SetFileAttributes**.

Функция **GetFileAttributes** возвращает атрибуты указанного файла или каталога.

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

Параметры:

lpFileName [in]	Указатель на нуль-завершенную строку, которая определяет имя файла или каталога.

В случае удачного завершения работы функции возвращаемое значение, содержащее значения атрибутов файла или каталога см. описание поля ***dwFileAttributes*** структуры **WIN32_FIND_DATA**. В противном случае будет возвращено значение 0xFFFFFFFF, т.е. -1.

Функция ***SetFileAttributes*** устанавливает атрибуты файла.

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

Параметры:

lpFileName [in]	Указатель на нуль-завершенную строку, которая определяет имя файла или каталога.
dwFileAttributes [in]	Атрибуты файла. См. описание поля ***dwFileAttributes*** структуры **WIN32_FIND_DATA.**

Возвращаемые значения

Внимание: **BOOL** – не является встроенным типом языка C++ `bool`! В файле `windef.h` **BOOL** определено, как **int**. Если функция завершается успешно, возвращаемое значение - не нуль.

Если функция завершается ошибкой, величина возвращаемого значения - нуль. Для получения дополнительной информации об ошибке, необходимо вызывать функцию **GetLastError.**

Задания к лабораторным работам:

Вариант 1.

Написать программу, которая для заданного каталога определяет объем файлов, входящий в данный каталог и все его подкаталоги.

Вариант 2.

Написать программу, которая для заданного каталога определяет объем файлов, с расширением txt, являющимися архивными.

Вариант 3.

Написать программу, которая для заданного каталога ищет и выводит имя подкаталога, который содержит максимальное число файлов с расширением cpp.

Вариант 4.

Написать программу, которая для заданного каталога задает для всех файлов с расширением cpp атрибут «только для чтения», и выводит количество файлов, для которых произошло изменение атрибута.

Вариант 5.

Написать программу, которая для заданного каталога задает для всех файлов с расширением cpp атрибут «системный файл, и выводит общий объем файлов, для которых произошло изменение атрибута.

Вариант 6.

Написать программу, которая для заданного каталога задает для всех файлов, кроме файлов с расширением txt устанавливает атрибуты «только для чтения» и «скрытый файл». Программа должна вывести количество файлов атрибуты, которых были изменены.

Вариант 7.

Написать программу, которая для заданного каталога ищет все вложенные в него подкаталоги и изменяет их атрибуты на «скрытый». Программа должна вывести имена всех подкаталогов.

Вариант 8.

Написать программу, которая для заданного каталога задает для всех файлов с расширением cpp, созданных позже указанной преподавателем даты атрибут «только для чтения», и выводит количество файлов, для которых произошло изменение атрибута.

Вариант 9.
Написать программу, которая для заданного каталога задает для всех файлов с расширением cpp атрибут «только для чтения», и выводит имена и даты создания, с точностью до минут, для которых произошло изменение атрибута.

Вариант 10.
Написать программу, которая для заданного каталога ищет подкаталог, созданный раньше всех, и выводит список файлов из этого каталога с расширениями cpp и txt.

Вариант 11.
Написать программу, которая для заданного каталога ищет подкаталог, созданный позже всех, и выводит список файлов из этого каталога в «классическом формате», а также даты времен создания и последней модификации с точностью до минут.

Действия по заданиям можно объединить в одной консольной программе.