#### Донецкий национальный технический университет

#### Кафедра компьютерной инженерии

#### Методические указания

#### к лабораторным работам

#### по курсу Системное программное обеспечение

для студентов специальностей

 Системное программирование и Компьютерные системы и сети

направления подготовки Компьютерная инженерия

Составитель: доцент кафедры КИ

Чередникова Ольга Юрьевна

#### Донецк 2015

Содержание

[Лабораторная работа № 1](#_Лабораторная_работа_№)

[Лабораторная работа №2](#_Лабораторная_работа_№2)

[Лабораторная работа № 3](#_Лабораторная_работа_№_1)

[Лабораторная работа №4](#_Лабораторная_работа_№4)

[Лабораторная работа №5](#_Лабораторная_работа_№5)

##  Лабораторная работа № 1

***Определение ресурсов компьютера***

**Цель работы:** ознакомление с функциями Win API, позволяющими получить характеристики компьютера и сведения о логических дисках.

 **Теоретические сведения**

 Основные функции API для работы с логическими томами.

 DWORD **GetLogicalDrives** (void) определяет логические диски на данном ПК, возвращает 32-битное значение, каждый бит которого указывает, существует ли соответствующий логический диск (например, бит 0 установлен, если в системе виден диск A:\).

UINT **GetDriveType**(LPTSTR lpszRootPathName) возвращает тип диска, идентифицируемого параметром lpszRootPathName – указателем на корневой каталог.

#### Возвращаемое значение

|  |  |
| --- | --- |
| **Value** | **Meaning** |
| DRIVE\_UNKNOWN | Тип неизвестен |
| DRIVE\_NO\_ROOT\_DIR | Неправильный путь корня |
| DRIVE\_REMOVABLE | Съемный диск  |
| DRIVE\_FIXED | Жесткий диск  |
| DRIVE\_REMOTE | Сетевой диск |
| DRIVE\_CDROM | CD-ROM |
| DRIVE\_RAMDISK | Диск виртуальной памяти |

**DWORD Disks;**

**char DriveName[3];**

**Disks = GetLogicalDrives();**

**strcpy(DriveName," :"); i = 0;**

**while (Disks) // Do loop until we have drives**

**{ if (Disks & 0x00000001) // If this drive present**

 **{ DriveName[0] = 'A' + i;**

 **if (GetDriveType(DriveName) == DRIVE\_FIXED) printf(“%s\n”,DriveName);**

 **}**

 **Disks = Disks >> 1; i++;}**

Следующая функция, формирует информацию о конкретной файловой системе и определяет указатели

**GetVolumeInformation** (LPTSTR lpRootPathName, LPTSTR lpVolumeNameBuffer, DWORD nVolumeNameSize, LPDWORD lpVolumeSerialNumber, LPDWORD lpMaximumComponentLength, LPDWORD lpFileSystemFlags, LPTSTR lpFileSystemNameBuffer, DWORD nFileSystemNameSize),

где lpRootPathName Корневой каталог ("C:\").

параметр lpVolumeNameBuffer. Возвращает метку тома,

параметр nVolumeNameSize определяет максимальный размер буфера в символах,

параметр lpVolumeSerialNumber - серийный номер тома,

параметр lpMaximumComponentLength - максимальное количество символов, допустимое в именах каталогов и файлах,

параметр lpFileSystemFlags - флаги файловой системы,

|  |  |
| --- | --- |
| FILE\_NAMED\_STREAMS | Система поддерживает потоки имен |
| FILE\_READ\_ONLY\_VOLUME | Windows XP:Том только для чтения |
| FILE\_SUPPORTS\_OBJECT\_IDS | Система поддерживает описатель объектов |
| FILE\_VOLUME\_QUOTAS | Файловая система поддерживает дисковые квоты |
| FS\_CASE\_SENSITIVE | Чувствительность к регистру букв |
| FS\_FILE\_COMPRESSION | Файловая система поддерживает компрессию |
| FS\_UNICODE\_STORED\_ON\_DISK | Поддержка имен в формате UNICODE |
| FS\_VOL\_IS\_COMPRESSED | Сжатый том |

 параметр lpFileSystemNameBuffer указывает на буфер файловой системы,

параметр nFileSystemNameSize определяет максимальный размер этого буфера.

 BOOL **SetVolumeLabel** (LPTSTR lpRootPathName, LPTSTR lpVolumeName) изменяет метку тома.

 Первый параметр задает корневой каталог, а второй определяет новую метку тома.

Функция **GetDiskFreeSpace** возвращает информацию о размере диска и свободном пространстве. Функция не может обрабатывать тома размером более 2 GB, для этого используется **GetDiskFreeSpaceEx** которая принимает в качестве аргументов указатели на 64-битные беззнаковые числа.

BOOL **GetDiskFreeSpace** (LPTSTR lpRootPathName, LPDWORD lpSectorPerCluster, LPDWORD lpBytesPerSector, LPDWORD lpFreeCluster, LPDWORD lpClusters)

где параметр lpSectorPerCluster возвращает количество секторов в кластере;

 параметр lpBytesPerSector – количество байтов в секторе;

 параметр lpFreeCluster - количество свободных кластеров;

параметр lpClusters - общее количество кластеров.

Размер диска в байтах = **TotalNumberOfClusters\*SectorsPerCluster\*BytesPerSector**

### *Определение типа процессора и ОС*

 Общую информацию об архитектуре и количестве процессоров в системе можно получить, используя функцию **GetSystemInfo,** передав в качестве аргумента указатель на структуру SYSTEM\_INFO:

**typedef struct \_SYSTEM\_INFO {
 union {
 DWORD dwOemId;
 struct {
 WORD wProcessorArchitecture;
 WORD wReserved;
 };
 };
 DWORD dwPageSize;
 LPVOID lpMinimumApplicationAddress;
 LPVOID lpMaximumApplicationAddress;
 DWORD\_PTR dwActiveProcessorMask;
 DWORD dwNumberOfProcessors;
 DWORD dwProcessorType;
 DWORD dwAllocationGranularity;
 WORD wProcessorLevel;
 WORD wProcessorRevision;**

**} SYSTEM\_INFO;**

**dwOemId**

Устаревший элемент, оставлен для совместимости с Wndows NT 3.5 и ранее. Новые приложения должны использовать wProcessorArchitecture.

**wProcessorArchitecture** Архитектура системы:

PROCESSOR\_ARCHITECTURE\_UNKNOWN
PROCESSOR\_ARCHITECTURE\_INTEL
**Windows NT 3.51:** PROCESSOR\_ARCHITECTURE\_MIPS
**Windows NT 4.0 and earlier:** PROCESSOR\_ARCHITECTURE\_ALPHA
**Windows NT 4.0 and earlier:** PROCESSOR\_ARCHITECTURE\_PPC
**64-bit Windows:** PROCESSOR\_ARCHITECTURE\_IA64
**64-bit Windows:** PROCESSOR\_ARCHITECTURE\_IA32\_ON\_WIN64
**64-bit Windows:** PROCESSOR\_ARCHITECTURE\_AMD64

dwPageSize размер страниц памяти, используемый функцией GlobalAlloc

**lpMinimumApplicationAddress и lpMaximumApplicationAddress**

Указатели на минимальный и максимальный адреса памяти, доступные программе

**dwActiveProcessorMask**

Маска, представляющая активный процессор. Бит 0 соответствует 0; бит 31 соответствует процессору 31.

**dwNumberOfProcessors** Количество процессоров в системе.

dwProcessorType

Устаревший элемент, оставленный для совместимости с WindowsNT 3.5 и ранее

dwAllocationGranularity

Гранулярность с которой выделяется виртуальная память. К примеру, если VirtualAlloc запросила для выделения 1 байт, то будет выделено как минимум dwAllocationGranularity байт.

wProcessorLevel системный архитектурно-зависимый уровень процессора. используется только для отображения на экране.

Если **wProcessorArchitecture** соответствует PROCESSOR\_ARCHITECTURE\_INTEL, **wProcessorLevel** содержит CPU Vendor.

Если **wProcessorArchitecture** соответствует PROCESSOR\_ARCHITECTURE\_IA64, **wProcessorLevel** установлен в 1.

Если **wProcessorArchitecture** соответствует PROCESSOR\_ARCHITECTURE\_MIPS, **wProcessorLevel** в форме 00*xx*, где *xx* – это 8-битный номер исполнения. Может принимать следующие значения

|  |  |
| --- | --- |
| **Константа** | **Значение** |
| 0004 | MIPS R4000 |

Если **wProcessorArchitecture** соответствует PROCESSOR\_ARCHITECTURE\_ALPHA, **wProcessorLevel** в форме *xxxx*, где *xxxx* - 16-битный номер версии процессора. Принимает следующие значения

|  |  |
| --- | --- |
| **Константа** | **Значение** |
| 21064 | Alpha 21064 |
| 21066 | Alpha 21066 |
| 21164 | Alpha 21164 |

Если **wProcessorArchitecture** соответствует PROCESSOR\_ARCHITECTURE\_PPC, **wProcessorLevel** в форме *xxxx*, где *xxxx* – это 16-битный номер версии процессора. Может принимать следующие значения

|  |  |
| --- | --- |
| **Константа** | **Значение** |
| 1 | PPC 601 |
| 3 | PPC 603 |
| 4 | PPC 604 |
| 6 | PPC 603+ |
| 9 | PPC 604+ |
| 20 | PPC 620 |

wProcessorRevision

|  |  |
| --- | --- |
|  **Processor** | **Meaning** |
| Intel 80386 or 80486 | Значение типа *xxyz*. Если *xx* соответствует 0xFF, *y* - 0xA номер модели, и *z* определяет степпинг, иначе *xx* + 'A' определяет степпинг |
| Intel Pentium, Cyrix, or NextGen 586 | Значение типа *xxyy*, где *xx* - модель и *yy* - степпинг.  |
| MIPS | Значение типа 00*xx*, где *xx* 8-битный номер ревизии. |
| ALPHA | Значение типа *xxyy*, где *xxyy* младшее слово ревизии процессора. |
| PPC | Значение типа *xxyy*, где *xxyy* младшее слово ревизии процессора. Изображается в виде: *xx*.*yy* |

Одним из вариантов **определения типа процессора и его частоты** является использование инструкций **CPUID** и **RDTSC**. **CPUID** определяет:

* производителя процессора;
* поддержку процессором соответствующих технологий;
* информацию о типе и размере кэш-памяти 1-го, 2-го и 3-го уровня процессора, а также информацию о TLB-буферах процессора;
* название процессора (не во всех) и в некоторых случаях его частоту;
* дату, времени выпуска и партии (Только для процессоров Transmeta).

Для использования этой инструкции проверяется, поддерживается ли она тестируемым процессором. Это тестируется возможностью программного изменения 21-го бита регистра **EFLAGS**, которое свидетельствует о поддержки процессором инструкции **CPUID**. Делается это следующим образом:

 **pushfd** ; Сохранить EFLAGS в стек

**pop eax** ; Поместить EFLAGS в EAX

**mov ebx, eax** ; Сохранить в EBX для последующего тестирования

**xor eax, 00200000h** ; Переключить бит 21

**push eax** ; Скопировать измененное значение в стек

**popfd** ; Сохранить измененное EAX в EFLAGS

**pushfd** ; EFLAGS на вершину стека

**pop eax** ; Сохранить EFLAGS в EAX

**cmp eax, ebx** ; Проверить 21-й бит на изменение

**jz NO\_CPUID** ; Если нет изменения, то CPUID не поддерживается

 Для получения информации о процессоре с помощью инструкции CPUID необходимо перед выполнением самой инструкции поместить в регистр eax параметр. В зависимости от параметра инструкция CPUID возвращает различные свойства, характеристики и т.п

 **CPUID при eax = 0**

Таблица 1. Значения регистров после выполнения eax = 0

|  |  |
| --- | --- |
| **eax** | Максимально допустимое значение, которое может быть помещено в eax для получения информации о процессоре с помощью CPUID (для стандартного уровня – 0, 1, 2, 3, 4, 5). Все остальные значения параметра стандартного уровня CPUID должны быть не больше этого значения.  |
| **ebx** | Первые четыре символа строки производителя в обратном порядке |
| **edx** | Следующие четыре символа строки производителя в обратном порядке |
| **ecx** | Последние четыре символа строки производителя в обратном порядке |

Варианты строк производителя:

* Intel – ”GenuineIntel”
* AMD – ”AuthenticAMD” (в некоторых предварительных образцах “AMD ISBETTER”)
* Cyrix – ”CyrixInstead”
* UMC – ”UMC UMC UMC ”
* NexGen – “NexGenDriven”
* Centaur – “CentaurHauls”
* Rise Technology – “RiseRiseRise”
* SiS – “SiS SiS SiS ”
* Transmeta – “GenuineTMx86”
* National Semiconductor – “Geode by NSC”

 Эта строчка определяет производителя процессора (некоторые ранние Pentium не возвращали строчку производителя). Обратите внимание, что все производители должны использовать 12 символов, что все они с успехом и делают. Некоторые производители вместо своей строчки использовали строчку компании Intel, что приводит к некоторым сложностям, к счастью это относится к редким процессорам IDT (Centaur).

 **CPUID при eax = 1**

Таблица 2. Значения регистров после выполнения eax = 1

|  |  |
| --- | --- |
| **eax** | **32-x битовый массив, содержащий основную информацию о типе процессора**  |
| Биты 27-20 | Расширенное семейство (00b – Intel P4, AMD K8; 01b – Intel Itanium II (IA-64)) |
| Биты 19-16 | Расширенная модель |
| Биты 13-12 | Тип процессора (00b – основной процессор, 01b – Overdrive, 11b – дополнительный процессор в многопроцессорных системах) |
| Биты 11-8 | Семейство (таблица 2.1) |
| Биты 7-4 | Модель (таблица 2.2) |
| Биты 3-0 | Степпинг (поколение ядра процессора) |
| Биты 31-0 | 95-64 биты 96-битного уникального номера процессора (Intel Pentium III). В общем случае эти биты формируют сигнатуру процессора (совмещают информацию о модели, семействе, степпинге, расш. мод., типе, расш. сем. ) |
| **ebx** | Биты 7-0  | Brand ID – Указывает на тип процессора, если 0, то не поддерживается (для Intel поддерживается, начиная с Pentium III модели 8, а для AMD, начиная с Opteron) |
| Биты 15-8 | Кол-во 8 байтовых порций CLFLUSH (Intel) (только у процессоров с поддержкой CLFLUSH) |
| Биты 23-16 | Число логических процессоров (только у процессоров с поддержкой Hyper-Threading) |
| Биты 31-24 | Физический ID локального программно-доступного контроллера прерываний |
| **ecx** | Информация о возможностях процессора (на данный момент только для Intel таблица 2.4) |
| **edx** | Информация о возможностях процессора (таблица 2.3) |

Таблица 2.1. Таблица семейств процессоров

|  |  |
| --- | --- |
| **Семейство** | **Тип процессора** |
| 4 | все 80486, AMD 5x86, Cyrix 5x86 |
| 5 | Intel P5, P54C, P55C, P24T (Pentium)NexGen Nx586Cyrix M1AMD K5, K6Centaur C6, C2, C3Rise mP6SiS 55xTransmeta Crusoe TM3x00 и TM5x00 |
| 6 | Intel Pentium Pro, PII, PIII, Pentium M (Centrino)AMD K7 (Athlon)Cyrix M2VIA C3 |
| 7 | Intel Itanium (IA-64)  |
| F | Означает, что надо проверить расширенное семейство |

Таблица 2.2. Таблица моделей процессоров

|  |  |  |
| --- | --- | --- |
| **Тип** | **Модель** | **Процессор** |
| все | F | Означает, что надо проверить расширенную модель |
| Intel 80486 | 0 | i80486DX-25/33 |
| 1 | i80486DX-50 |
| 2 | i80486SX |
| 3 | i80486DX2 |
| 4 | i80486SL |
| 5 | i80486SX2 |
| 7 | i80486DX2WB |
| 8 | i80486DX4 |
| 9 | i80486DX4WB |
| UMC 80486 | 1 | U5D |
| 2 | U5S |
| AMD 80486 | 3 | 80486DX2 |
| 7 | 80486DX2WB |
| 8 | 80486DX4 |
| 9 | 80486DX4WB |
| E | 5x86 |
| F | 5x86WB |
| Cyrix 5x86 | 9 | 5x86 |
| Cyrix MediaGX | 4 | GX, GXm |
| Intel P5-ядро(Pentium) | 0 | P5 A-step |
| 1 | P5 (0.80 µm) 60, 66 MHz |
| 2 | P54C |
| 3 | P24T Overdrive |
| 4 | P55C |
| 7 | P54C |
| 8 | P55C (0.25µm) |
| NexGen Nx586 | 0 | Nx586 or Nx586FPU (поздние версии) |
| Cyrix M1 | 2 | 6x86 |
| Cyrix M2 | 0 | 6x86MX |
| AMD K5 | 0 | SSA5 (PR75, PR90, PR100) |
| 1 | 5k86 (PR120, PR133) |
| 2 | 5k86 (PR166) |
| 3 | 5k86 (PR200) |
| AMD K6 | 6 | K6 (0.30 µm) |
| 7 | K6 (0.25 µm) |
| 8 | K6-2 |
| 9 | K6-III |
| D | K6-2+ и K6-III+ (0.18 µm) |
| Centaur | 4 | C6 |
| 8 | C2 |
| 9 | C3 |
| VIA C3 | 5 | Cyrix M2 ядро |
| 6 | WinChip C5A ядро |
| 7 | WinChip ядро C5B (если степпинг = 0..7) |
| 7 | WinChip ядро C5C (если степпинг = 8..F) |
| 8 | WinChip ядро C5N (если степпинг = 0..7) |
| 9 | WinChip ядро C5XL  |
| Rise | 0 | mP6 (0.25 µm) |
| 2 | mP6 (0.18 µm) |
| SiS | 0 | 55x |
| Transmeta | 4 | Crusoe TM3x00 и TM5x00 |
| Intel P6(Pentium Pro, Pentium II, Pentium III)и Pentium M | 0 | Pentium Pro A-step |
| 1 | Pentium Pro |
| 3 | Pentium II (0.28 µm) (Klamath) |
| 5 | Pentium II (0.25 µm) (Deschutes) |
| 6 | Celeron (Medocino) 128 Kb L2 |
| 7 | Pentium III (0.25 µm) (Katmai) |
| 8 | Pentium III (0.18 µm) 256 Kb L2 (Coppermine) |
| 9 | Pentium M (Centrino) (0.13 µm) 1 MB L2 |
| A | Pentium III (0.18 µm) 1 или 2 MB L2 (Cascades) |
| B | Pentium III (0.13 µm) 256 или 512 KB L2 (Tualatin) |
| AMD K7 | 1 | Athlon (0.25 µm) |
| 2 | Athlon (0.18 µm) |
| 3 | Duron (ядро Spitfire) |
| 4 | Athlon (ядро Thunderbird) |
| 6 | Athlon (ядро Palomino) |
| 7 | Duron (ядро Morgan) |
| 8 | Athlon (ядро Thoroughbred) |
| A | Athlon (ядро Barton) |
| AMD K8 | 4 | Athlon 64 (0.13 µm) |
| 5 | Opteron DP (0.13 µm) |
| Intel P4-ядро | 0 | Pentium 4 (0.18 µm) (ядро Willamete) |
| 1 | Pentium 4 (0.18 µm) (ядро Willamete) |
| 2 | Pentium 4 (0.13 µm) (ядро Northwood) |
| 3 | Pentium 4 (0.09 µm) (ядро Prescott) |

Таблица 2.3. Поля регистра edx после выполнения команды CPUID (при eax=1)

|  |  |  |
| --- | --- | --- |
| **Бит** | **Имя** | **Если Бит = 1** |
| 0 | FPU | Присутствует Математический сопроцессор |
| 1 | VME | Поддержка расширенных возможностей обработки прерываний в режиме виртуального i8086 |
| 2 | DE | Поддержка отладки |
| 3 | PSE | Поддержка страниц размером 4 MB |
| 4 | TSC | Счетчик меток реального времени |
| 5 | MSR | Поддержка команд rdmsr и wrmsr |
| 6 | PAE | Поддержка физического адреса более 32 бит |
| 7 | MCE | Поддержка исключений 18 – об аппаратных ошибках |
| 8 | CX8 | Поддержка инструкции cmpxchg8b |
| 9 | APIC | Микропроцессор содержит программно доступный контроллер прерываний |
| 10 | - | Зарезервировано |
| 11 | SEP | Поддержка инструкций быстрых системных вызовов sysenter и sysexit |
| 12 | MTRR | Поддержка регистра mtrr\_cap (относится к MSR-регистрам) |
| 13 | PGE | Поддержка глобальных страниц |
| 14 | MCA | Поддержка архитектуры машинного контроля |
| 15 | CMOV | Поддержка инструкций условной пересылки cmov, fcmovcc, fcomi |
| 16 | PAT | Процессор поддерживает таблицу атрибутов страницы |
| 17 | PSE-36 | Процессор поддерживает 4 MB страницы, которые способны адресовать физическую память до 64 GB |
| 18 | PSN | Уникальный серийный номер процессора |
| 19 | CLFLSH | Поддержка инструкции CLFLUSH |
| 20 | - | Зарезервировано |
| 21 | DS | Поддержка записи отладочной информации |
| 22 | ACPI | Управление охлаждением процессора с помощью пустых циклов в зависимости от температуры |
| 23 | MMX | Поддержка MMX |
| 24 | FXSR | Поддержка инструкций FXSAVE и FXRSTOR |
| 25 | SSE | Поддержка SSE |
| 26 | SSE2 | Поддержка SSE2 |
| 27 | SS | Управление конфликтующими типами памяти |
| 28 | HTT | Поддержка Hyper-Threading |
| 29 | TM | Поддержка автоматического мониторинга температуры |
| 30 | - | Зарезервировано |
| 31 | SBF | Сигнал Останова при FERR |

Таблица 2.4. Поля регистра ecx после выполнения команды CPUID (при eax=1)

|  |  |  |
| --- | --- | --- |
| **Бит** | **Имя** | **Если Бит = 1** |
| 0 | PNI | Поддержка новых инструкций процессора Prescott (SSE3) |
| 3 | MONITOR | MONITOR/MWAIT |
| 4 | DS-CPL | Поддержка расширенной записи отладочной информации |
| 7 | TM2 | Поддержка технологии Thermal Monitor 2 |
| 8 | EST | Улучшенная технология SpeedStep |
| 10 | CNXT-ID | Показывает можно ли переключить способ работы кэш-памяти на адаптивный или распределенный  |

 По результатам побитового сравнения этих полей делается заключение о поддержке соответствующих технологий. Делать заключения о наличии или отсутствии технологий, исходя из принадлежности процессора семейству, модели и т.п. строго говоря, нельзя. Однако из-за того, что некоторые процессоры по-своему трактуют некоторые биты этих полей, приходиться учитывать влияние семейства, модели и т.п. Особый интерес представляют бит 4 регистра EDX после выполнения команды CPUID с EAX=1. Этот бит определяет поддержку процессором внутреннего счётчика тактов. Если данный счётчик поддерживается, то с определённой погрешностью можно определить частоту процессора, используя команду **RDTSC**. RDTSC выполняет чтение 64-битного счётчика тактов и результат возвращает в паре регистров EDX:EAX.

**unsigned long redx,reax; // Для временного хранения регистров**

**if (isTSCSupport) // Определение поддержки счётчика TSC**

**{**

 **\_\_asm{**

 **rdtsc // Чтение счётчика**

 **mov dword ptr redx,edx**

 **mov dword ptr reax,eax**

 **}**

 **Sleep(1000); // Организация задержки**

 **\_\_asm{**

 **rdtsc // Чтение счётчика**

 **sub eax,dword ptr reax // Вычесть предыдущее значение счетчика**

 **sbb edx,dword ptr redx**

 **mov dword ptr reax,eax**

 **mov dword ptr redx,edx**

 **}**

 **reax = reax/1000000; // Перевод в MHz, учитывая задержку 1000**

 **printf("\nFrequency : %ld Mhz",reax);**

**}**

 Для использования счётчика тактов вместо ассемблерных вставок можно использовать функции QueryPerformanceFrequency и QuaryPerformanceCounter.

 **QueryPerformanceFrequency** возвращает частоту этого счётчика, если он присутствует. Частота не может изменяться, когда система запущена.

**BOOL QueryPerformanceFrequency( LARGE\_INTEGER\* *lpFrequency*);**

#### lpFrequency Указатель на переменную, которая получает текущую частоту счётчика. Если счётчик не поддерживается аппаратно, этот параметр содержит 0.

#### Return Values Если имеется аппаратная поддержка счётчика, то возвращает ненулевое значение

 **QueryPerformanceCounter** возвращает текущее значение счётчика.

**BOOL QueryPerformanceCounter( LARGE\_INTEGER\* *lpPerformanceCount*);**

lpPerformanceCount указатель на переменную, которая получает текущее значение счётчика.

**Return Values** При удачном выполнении возвращает ненулевое значение.

#### Remarks

На мультипроцессорных системах не имеет значения на каком процессоре вызывается функция. Можно получить разные результаты на разных процессорах из-за ошибки в BIOS или HAL. Для выбора процессора используется **SetThreadAffinityMask**.

**DWORD\_PTR SetThreadAffinityMask(HANDLE *hThread*,**

 **DWORD\_PTR *dwThreadAffinityMask*);**

hThread хэндл потока, которому устанавливается процессорная маска.

**dwThreadAffinityMask** Процессорная маска.

#### Return Values При удачном выполнении возвращает ненулевое значение.

#### Remarks Процессорная маска – это битовый вектор, в котором каждый бит представляет процессоры, на которых разрешён запуск процесса.

Для компиляции примера необходима поддержка компилятором 64 битных чисел

**LARGE\_INTEGER li1,li2 ; // Для временного хранения счётчика**

**If (QueryPerformanceCounter(&li1))**

**{**

 **Sleep(1000); // Организация задержки**

 **QueryPerformanceCounter(&li2);**

 **li2.QuadPart -= li1.QuadPart; // Вычесть li1 из li2 и записать в li2**

 **li2.QuadPart /= 1000000; // Перевод в MHz, учитывая задержку 1000**

 **printf("\nFrequency : %ld Mhz",li2.QuadPart);**

**}**

### Определение мыши

 Различные характеристики системы можно получить из функции GetSystemMetrics, в том числе информацию о мыше.

**int** **GetSystemMetrics(** **int** *nIndex***);**

|  |  |
| --- | --- |
| **Value** | **Meaning** |
| SM\_CMOUSEBUTTONS | Количество кнопок мыши или 0, если мышь не установлена.  |
| SM\_MOUSEPRESENT | Проверка наличия мыши.  |
| SM\_MOUSEWHEELPRESENT | Проверка наличия колёсика.  |
| SM\_SWAPBUTTON | Переназначены ли кнопки мыши. |

**if (GetSystemMetrics(SM\_MOUSEPRESENT))**

**{ printf("Кол-во кнопок : %d",GetSystemMetrics(SM\_CMOUSEBUTTONS));**

 **if (GetSystemMetrics(SM\_MOUSEWHEELPRESENT))**

 **printf("Наличие колёсика : да");**

 **else printf("Наличие колёсика : нет");**

 **if (GetSystemMetrics(SM\_SWAPBUTTON))**

 **printf("Кнопки мыши переназначены");**

**}**

**else printf("Мышь не обнаружена");**

### Определение типа клавиатуры

 Функция **GetKeyboardType** возвращает информацию о текущей клавиатуре.

**int GetKeyboardType(int *nTypeFlag*);**

#### Parameters nTypeFlag Тип запрашиваемой информации

|  |  |
| --- | --- |
| **Value** | **Meaning** |
| 0 | Тип клавиатуры |
| 1 | Подтип клавиатуры |
| 2 | Количество функциональных клавиш |

Тип клавиатуры определяется следующими значениями

|  |  |
| --- | --- |
| **Value** | **Meaning** |
| 1 | IBM PC/XT or compatible (83-key) keyboard |
| 2 | Olivetti "ICO" (102-key) keyboard |
| 3 | IBM PC/AT (84-key) or similar keyboard |
| 4 | IBM enhanced (101- or 102-key) keyboard |
| 5 | Nokia 1050 and similar keyboards |
| 6 | Nokia 9140 and similar keyboards |
| 7 | Japanese keyboard |

Подтип клавиатуры зависит от производителя.

**const char\* cKeybType[] =**

**{**

 **"IBM PC/XT or compatible (83-key)",**

 **"Olivetti (102-key)",**

 **"IBM PC/AT (84-key) or similar",**

 **"IBM enhanced (101- or 102-key)",**

 **"Nokia 1050 similar",**

 **"Nokia 9140 similar",**

 **"Japanese keyboard",**

**};**

**printf(cKeybType[GetKeyboardType(0)]);**

**printf("Код производителя : %d",GetKeyboardType(1));**

**printf("Кол-во функц. клавиш : %d",GetKeyboardType(2));**

### Информация о физической памяти

 Состояние физической памяти определяется функцией **GlobalMemoryStatus,** которой в качестве аргумента передаётся указатель на структуру **MEMORYSTATUS.**

**typedef struct \_MEMORYSTATUS {
 DWORD dwLength;
 DWORD dwMemoryLoad;
 SIZE\_T dwTotalPhys;
 SIZE\_T dwAvailPhys;
 SIZE\_T dwTotalPageFile;
 SIZE\_T dwAvailPageFile;
 SIZE\_T dwTotalVirtual;
 SIZE\_T dwAvailVirtual;**

**} MEMORYSTATUS, \*LPMEMORYSTATUS;**

#### Members

**dwLength** Размер структуры в байтах.

**dwMemoryLoad** Процент используемой физической памяти.

**dwTotalPhys** Всего физической памяти в Kb.

**dwAvailPhys** Доступно физической памяти.

**dwTotalPageFile** Размер файла подкачки в байтах.

**dwAvailPageFile** Размер доступного пространства в файле подкачки в байтах

**dwTotalVirtual** Размер виртуальной памяти для текущего процесса в байтах.

**dwAvailVirtual** Доступно виртуальной памяти для процесса в байтах.

**MEMORYSTATUS stat;**

**GlobalMemoryStatus (&stat);**

**printf("Использование памяти : %d%%",stat.dwMemoryLoad);**

**printf("Доступно физ. памяти : %d Kb",stat.dwTotalPhys/1024);**

**printf("Свободно физ. памяти : %d Kb",stat.dwAvailPhys/1024);**

**printf("Доступно в файле подкачки : %d Kb",stat.dwTotalPageFile/1024);**

**printf("Свободно в файле подкачки : %d Kb",stat.dwAvailPageFile/1024);**

**printf("Доступно виртуальной памяти : %d Kb",stat.dwTotalVirtual/1024);**

**printf("Свободно виртуальной памяти : %d Kb",stat.dwAvailVirtual/1024);**

### Информация о видеокарте

 Общую информацию об устройстве отображения можно получить через функцию **EnumDispalyDevices** или более подробную из реестра Windows.

 Функция **EnumDispalyDevices** позволяет получить название видеокарты

и некоторую информацию о самом устройстве. В качестве аргументов передаётся указатель на название устройства, его индекс в системе и указатель на структуру **DISPLAY\_DEVICE,** поля которой функция и заполняет.

**BOOL EnumDisplayDevices(**

 **LPCTSTR *lpDevice*, // device name**

 **DWORD *iDevNum*, // display device**

 **PDISPLAY\_DEVICE *lpDisplayDevice*, // device information**

 **DWORD *dwFlags* // reserved**

**);**

#### Parameters

**lpDevice** Указатель на название устройства или 0.

iDevNum индекс устройства.

**lpDisplayDevic***e* Указатель на **DISPLAY\_DEVICE** структуру*.*

dwFlags не используется.

 Для получения информации о видеоустройстве необходимо передать в функцию NULL (вместо указателя на название устройства) и указатель на DISPLAY\_DEVICE с заполненным полем cb и StateFlags.

**DISPLAY\_DEVICE dispdev;**

**dispdev.cb = sizeof(DISPLAY\_DEVICE);**

**dispdev.StateFlags = DISPLAY\_DEVICE\_ATTACHED\_TO\_DESKTOP;**

**EnumDisplayDevices(NULL,NULL,&dispdev,NULL);**

**printf(“Видеокарта : %s”,dispdev.DeviceString);**

### Информация о накопителях

 Для взаимодействия с драйвером установленного устройства используется функция DeviceIoControl. Функция позволяет получить любую интересующую информацию о характеристиках устройства.

**BOOL DeviceIoControl(**

 **HANDLE *hDevice*,**

 **DWORD *dwIoControlCode*,**

 **LPVOID *lpInBuffer*,**

 **DWORD *nInBufferSize*,**

 **LPVOID *lpOutBuffer*,**

 **DWORD *nOutBufferSize*,**

 **LPDWORD *lpBytesReturned*,**

 **LPOVERLAPPED *lpOverlapped***

**);**

#### Parameters

**hDevice** Хэндл устройства.

**dwIoControlCode**  Код производимой операции.

**lpInBuffer** Указатель на буфер с данными, которые необходимы для выполнения заданной операции

**nInBufferSize** Размер входного буфера в байтах.

**lpOutBuffer** Буфер для выходных данных

**nOutBufferSize**  Размер буфера для выходных данных (заполнит DeviceIoControl)

**lpBytesReturned** Количество байт данных, возвращённых в выходном буфере.

**lpOverlapped** Указатель на структуру перекрытия.

#### Замечание

 Для получения хэндла устройства необходимо использовать **CreateFile** с названием устройства или драйвера, ассоциированного с данным устройством. Для указания устройства необходимо использовать формат:\\.\DeviceName

Тип устройств, которые могут быть открыты, зависит от платформы, на которой запускается приложение.

* **Windows NT/2000/XP:  DeviceIoControl** может принимать хэндл устройства. Для примера, для получения хэндла для дисковода гибких дисков A: вызываем **CreateFile**, указывая \\.\a:. Альтернативой является использование имён \\.\PhysicalDrive0, \\.\PhysicalDrive1, и т.д. для открытия физических устройств.
* **Windows 95/98/Me:  DeviceIoControl** может принимать хэндл на драйвер виртуального устройства. Например, для получения хэндла на системный VxD с **CreateFile** указывают \\.\vwin32.

Необходимо указать FILE\_SHARE\_READ и FILE\_SHARE\_WRITE права доступа при вызове **CreateFile** для получения хэндла драйвера устройства. Используй **CreateFile** параметры, описанные ниже, для открытия устройства:

* *fdwCreate* параметр OPEN\_EXISTING.
* *hTemplateFile* должен быть NULL.
* *fdwAttrsAndFlags* может быть FILE\_FLAG\_OVERLAPPED для определения того, что хэндл может использоваться для выполнения асинхронных операций I/O.

**!!! Для доступа к физическим дискам и томам при использовании функции CreateFile необходимо иметь привилегии администратора**

**DISK\_GEOMETRY dg;**

**HANDLE hDevice;**

**DWORD ReturnSize;**

**DWORD DiskSize;**

**hDevice = СreateFile("\\\\.\\PhysicalDrive0",0,FILE\_SHARE\_WRITE|FILE\_SHARE\_READ,**

**NULL,OPEN\_EXISTING,0,NULL);**

**if (hDevice == INVALID\_HANDLE\_VALUE)**

 **printf(“Невозможно получить хэндл 1-ого жесткого диска.”);**

**else**

**{ dg.MediaType = FixedMedia;**

 **DeviceIoControl(hDevice,IOCTL\_DISK\_GET\_DRIVE\_GEOMETRY,NULL,0,**

 **&dg,sizeof(DISK\_GEOMETRY),&ReturnSize,(LPOVERLAPPED) NULL);**

 **CloseHandle(hDevice);**

 **printf("Цилиндров : %d",dg.Cylinders);**

 **printf("Дорожек на цилиндр : %d",dg.TracksPerCylinder);**

 **printf("Секторов на дорожку : %d",dg.SectorsPerTrack);**

 **printf("Размер сектора : %d байт",dg.BytesPerSector);**

 **DiskSize = (dg.Cylinders.LowPart \* dg.TracksPerCylinder \***

 **dg.SectorsPerTrack)/1024 \* dg.BytesPerSector;**

 **printf("Размер HDD : %d Mb",DiskSize/1024);**

**}**

**Варианты заданий**

1. Определить логические диски и для каждого диска его размер. Вывести информацию о частоте процессора и фирму производитель процессора.

2. Определить логические диски и для каждого диска общее количество кластеров. Вывести информацию о видеокарте и модель процессора.

3. Определить логические диски и тип каждого диска, информацию о накопителях и архитектуру процессора.

4. Определить логические диски и для каждого диска количество свободных кластеров. Определить информацию о физической памяти и степпинг процессора

5. Определить логические диски и для каждого диска его серийный номер. Вывести информацию о видеокарте и возможностях процессора (с помощью CPUID eax=1 return ecx).

6. Определить логические диски и для каждого диска его метку тома. Вывести информацию о накопителях и возможностях процессора (с помощью CPUID eax=1 return edx).

7. Определить логические диски и для каждого диска его количество свободных кластеров. Вывести информацию о модели и расширенной модели процессора, также гранулярность выделения памяти.

8. Определить логические диски и для каждого диска флаги файловой системы. Вывести информацию о мыши и семейство, расширенное семейство процессора.

9. Определить логические диски и для каждого диска размер сектора. Вывести информацию о типе клавиатуры и фирму-производитель процессора.

10. Определить логические диски и для каждого диска количество секторов в кластере. Вывести информацию о видеокарте и возможностях процессора (с помощью CPUID eax=1 return edx).

11. Определить логические диски и для каждого диска количество свободных кластеров. Вывести информацию о модели процессора и количестве процессоров в системе.

12. Определить логические диски и для каждого диска его размер. Вывести информацию о архитектуре и фирме-производителе процессора.

13. Определить логические диски и для каждого диска флаги файловой системы. Вывести информацию о возможностях процессора (с помощью CPUID eax=1 return ecx) и активных процессорах.

14. Определить логические диски и для каждого диска максимальное количество символов, допустимое в именах каталогов и файлах. Вывести информацию о гранулярности виртуальной памяти и семействе, расширенном семействе процессора.

15. Определить логические диски и для каждого диска его размер. Вывести информацию о системном архитектурно-зависимом уровне процессора и степпинг процессора.

## Лабораторная работа №2

**Командные файлы**

***Цель работы:*** Получение навыков написания .bat файлов.

***Теоретические сведения***

ПАКЕТНЫЕ КОМАНДНЫЕ ФАЙЛЫ

 Довольно часто в процессе работы с компьютером обнаруживается, что необходимо повторять одни и те же команды ОС (может быть с небольшими изменениями) для того, чтобы осуществить некоторые периодически выполняемые действия. Операционная система позволяет записать нужную для этого последовательность команд в специальный файл, называемый командным файлом. Командный файл должен иметь расширение .BAT. Последовательность команд, записанную в командном файле, можно выполнить, запустив командный файл на выполнение.

 Командный файл представляет собой простой текстовый документ.

 **1. Выполнение командных файлов**

 1) Выполнение командного файла можно прервать, нажав <Ctrl>/C или <Ctrl>/<Break>. После этого на экране появится запрос:

 Terminate batch job (Y/N) ?

 (Прервать выполнение командного файла (Да/Нет) ?)

 2) Если требуется из одного командного файла вызвать выполнение другого командного файла, а затем продолжить выполнение исходного командного файла, то это можно сделать командой

 command /c имя\_командного\_файла [параметры]

 Из одного командного файла можно вызвать другой командный файл с помощью команды call :

 call имя\_командного\_файла [параметры]

Допускается рекурсивный вызов командного файла из самого себя.

 3)По умолчанию команды пакетного файла выводятся на экран перед выполнением. Если в пакетный файл вставить команду echo off, то выполняемые за ней команды не будут выводиться на экран.Команда echo on включает режим вывода выполняемых команд на экран. Очень часто в качестве первой строки командного файла используется команда echo off. В этом случае в начале выполнения командного файла на экран выводится строка echo off, а затем выполняемые команды на экран не выводятся.

 Чтобы избежать вывода на экран и самой строки echo off, перед этой командой следует поставить символ @:

 @echo off

 **2. Командные файлы с заменяемыми параметрами**

 Часто приходится выполнять одни и те же команды или последовательности команд только с небольшими отличиями.

Например, для просмотра содержимого файла надо набрать следующую команду:

 type AUTOEXEC.BAT

 Чтобы упростить вызов этой команды, можно создать файл T.BAT следующего содержания:

 type %1.BAT

 При выполнении этого командного файла символ %1 будет заменен на значение первого параметра при вызове командного файла. Например, если ввести команду

 T AUTOEXEC

то вместо %1 будет подставлено AUTOEXEC и тем самым будет выполнена команда

 type AUTOEXEC.BAT

 Всего может быть использовано до девяти параметров, обозначаемых символами %1 - %9. Если в командной строке при вызове командного файла задано меньше девяти параметров, то "лишние" символы из %1 - %9 замещаются пустыми строками.

 В командном файле можно использовать также символ %0, значение которого - имя выполняемого командного файла.

 Пусть имеется командный файл MY.BAT, содержащий следующие команды:

 @echo off

 type %1.bat

 type %2.sys

 type %0

 Тогда после ввода команды

 MY AUTOEXEC CONFIG

на экран дисплея будет последовательно выводиться содержимое файлов AUTOEXEC.BAT, CONFIG.SYS и MY.BAT.

 Если в командном файле знак процента используется не для обозначения параметров, а для других целей (например, как часть имени файла), то его надо набрать дважды. Так, чтобы в командном файле указать файл XY%.COM, надо написать в строке командного файла XY%%.COM.

 Командный файл может содержать любые команды ОС, а также специальные внутренние команды командных файлов, описанные ниже.

 **3. Комментарии в командном файле**

 Команда **rem** позволяет включать в командный файл комментарии, которые не будут интерпретироваться как команды во время исполнения этого файла. Они могут использоваться для улучшения удобочитаемости командного файла. Если включен режим вывода исполняемых команд на экран (echo on), то комментарии выводятся на экран, а если этот режим выключен командой echo off, то комментарии в командном файле на экран не выводятся.

 Формат команды:

 rem любые\_символы

 Например:

 @echo off

 rem Вывод содержимого трех файлов

 type %1.bat

 type %2.sys

 type %0

 **4. Сообщения при выполнении командного файла**

 Команда echo позволяет выдавать сообщения из командного файла на экран.

 Формат команды:

 **echo** сообщение

 Сообщение, выдаваемое с помощью команды echo, выводится на экран даже тогда, когда режим вывода исполняемых команд на экран выключен командой echo off. Сообщение не может быть пустым или равным on или off, так как команды echo on и echo off управляют режимом вывода исполняемых команд на экран, а команда echo без параметров сообщает включен или выключен режим echo .

 С помощью команды echo можно получить звуковой сигнал компьютера. Для этого следует в сообщение, выводимое командой echo, включить символ с кодом 7. Это можно сделать, нажав клавишу Alt и, не отпуская ее, клавишу 7 в правой части клавиатуры (т.е. клавишу Home).

 Для того, чтобы вывести на экран пустую строку, можно использовать команду echo с сообщением, состоящим из символа с кодом 255. Этому символу не соответствует никакое изображение. Для ввода символа с кодом 255 надо нажать клавишу Alt и, не отпуская ее, последовательно нажать клавиши 2, 5 и 5 в правой части клавиатуры.

  **5. Приостановка выполнения командного файла**

 Во время выполнения командного файла может возникнуть необходимость его приостановки, например, для того, чтобы можно было поставить нужную дискету. Команда pause позволяет приостановить выполнение командного файла и ждать, пока не будет нажата какая-либо клавиша (алфавитно-цифровая клавиша, пробел,<Enter> или <Ctrl>/C).

 Формат команды:

 **pause**

 При выполнении этой команды на экран выводится сообщение:

 strike a key when ready ...

 (нажмите любую клавишу, когда будете готовы)

и выполнение командного файла приостанавливается. Если нажать любую алфавитно-цифровую клавишу, пробел или <Enter>, выполнение командного файла будет продолжено. Если нажать <Ctrl>/C или <Ctrl>/<Break>, то будет выведено сообщение

 Terminate batch job (Y/N) ?

 (прервать выполнение командного файла (Y - да, N - нет) ? )

Если ответить Y, то выполнение командного файла будет окончено,а оставшиеся команды из командного файла будут проигнорированы. Если ответить N, то выполнение командного файла будет продолжено со следующей команды.

 Ниже в качестве примера приводится текст командного файла для проверки файловой структуры на флеш-диске:

 @echo off

 echo Проверка диска . Вставьте флеш-диск

 pause

 c:\dos\chkdsk Е:

 **6. Переходы в командном файле**

 Командный файл может содержать метки и команды перехода. Это позволяет управлять порядком выполнения команд в файле. Любая строка командного файла, начинающаяся с двоеточия (":"), воспринимается при обработке командного файла как метка. Имя метки определяется набором символов, следующих за двоеточием. Оно может содержать от одного до восьми символов. Для того, чтобы указать командному процессору, что требуется продолжить выполнение команд из командного файла, начиная со строки, которая следует сразу после некоторой метки, надо воспользоваться командой

***GOTO*** *метка*

 Если метка в команде GOTO не указана, процесс пакетной обработки завершается. Аналогично при попытке перехода на несуществующую метку выводится аварийное сообщение и выполнение командного файла прекращается.

 П р и м е р

 goto xxx

 .

 .

 .

 :xxx

 rem Продолжение выполнения командного файла

 OS игнорирует любую строку командного файла, начинающуюся с двоеточия. Поэтому не следует на одной строке с меткой помещать какие-либо команды.

 **7. Проверка условий в командном файле**

 Команда if позволяет в зависимости от выполнения условий выполнять или не выполнять команды в командном файле.

 Формат команды:

 **if** условие команда

где команда - это любая допустимая команда (в том числе goto).

Эта команда выполняется, если условие в команде if истинно, в противном случае команда игнорируется; условие - это одно из приведенных ниже выражений:

 **ERRORLEVEL** число - условие истинно тогда, когда код завершения предыдущей выполненной программы или команды больше заданного числа или равен ему (код завершения устанавливается программами или командами OS при окончании их работы, по умолчанию этот код равен нулю);

 строка1 **==** строка2 - условие истинно тогда, когда строка1 и строка2 полностью совпадают. Если в этих строках имеются символы %0 - %9, то вместо этих символов подставляются параметры командного файла;

 **EXIST** имя\_файла - условие истинно, когда указанный файл существует;

 **NOT** условие - истинно тогда, когда указанное условие ложно.

 Ниже в качестве примера приводится текст командного файла, позволяющего удалить из текущего каталога все файлы типа .BAK, если они существуют:

 @echo off

 rem Удаление всех файлов типа BAK

 if exist \*.bak del \*.bak

 Рассмотрим более сложный пример. Пусть необходимо создать командный файл TYP.BAT, который выводит некоторый файл на экран. Формат вызова будет:

 TYP имя\_файла

Необходимо предусмотреть проверку, что при вызове задан параметр и что указанный файл существует. Командный файл TYP.BAT может быть таким:

 @echo off

 rem Вывод содеpжимого файла, если он существует

 if -%1 ==- goto no\_parm

 if not exist %1 goto not\_exist

 type %1

 goto exit

 :no\_parm

 echo Должен быть задан параметр

 goto exit

 :not\_exist

 echo Файл %1 не найден

 :exit

 Следует подчеркнуть, что, если сравниваемые символьные строки содержат формальные параметры %0 - %9, то ни одна из строк не может быть пустой, так как нельзя сравнивать пустые строки. Для того, чтобы обойти этот запрет, следует включить в состав сравниваемых строк какой-либо символ. Этот прием иллюстрируется командой

 if -%1 ==- goto no\_parm

из показанного выше примера.

  **8. Циклическое выполнение команд в командном файле**

 Для организации циклов в командном файле используется команда

 **FOR** %%переменная **IN** (набор) **DO** команда

где переменная - переменная из одной буквы, последовательно принимающая значения слов или имен файлов, входящих в параметр набор;

 набор - одно или несколько символьных слов или спецификаций файлов. Спецификация файла имеет вид

 [дисковод:][путь\] имя\_файла.

Допускаются шаблоны \* и ? . Слова и спецификации файлов разделяются пробелами или запятыми. В случае применения шаблонов обрабатывается только одно выражение из набора, а остальные игнорируются.

 Команда выполняется для каждого слова или файла из параметра набор.

 Ниже в качестве примера приводится текст командного файла CP.BAT, предназначенного для копирования всех файлов типов .C, .PAS и .BAK из текущего каталога в другой, определяемый значением первого параметра %1:

 @echo off

 rem Копирование файлов типов .C, .PAS, .BAK

 rem из текущего каталога в другой

 for %%E in (C PAS BAK) do copy \*.%%E %1

 Вызов данного командного файла осуществляется командой, аналогичной следующей:

 CP С:\

 **9. Сдвиг параметров при обработке командного файла**

 Команда **SHIFT** предоставляет возможность обращения более, чем к 9 заменяемым параметрам при обработке командного файла. Это достигается путем изменения нумерации параметров командной строки.

Например, для

 %0 = с.bat

 %1 = bar

 %2 = name

%3... %9 не заняты.

ействие команды SHIFT приведет к следующим изменениям:

 %0 = bar

 %1 = name

 %2 ... %9 не заняты.

 Если в командной строке было введено больше 9 параметров, то для использования в командном файле тех параметров, которые появляются после девятого (%9), необходимо поочередно их сдвигать по команде SHIFT в направлении %9.

 **10. Установка значения переменной окружения**

 Команда **SET** позволяет присвоить некоторое значение переменной окружения. Формат команды:

 SET переменная=значение

 Здесь переменная - любая строка, не содержащая знаков равенства и пробелов. При этом в переменной большие и малые латинские буквы считаются одинаковыми.

 Команда SET записывает строку "переменная=значение" в специальную область памяти, зарезервированную для хранения переменных окружения. Если данной переменной уже было присвоено какое-то значение, то оно заменяется новым.

 Если в командном файле употребить имя переменной окружения, заключенное с обеих сторон в знаки процента, то оно будет замещено на значение этой переменной. Например, после ввода команды

 SET CMD=%1

строка %CMD% в командном файле будет замещена на %1.

 При использовании команды SET без параметров на экране показываются действующие в настоящее время соответствия SET.

 Для удаления переменной окружения (и соответственно ее значения) после команды SET следует ввести имя переменной и только знак равенства, например:

 SET CMD=

 Ниже в качестве примера приводится текст командного файла REPEAT.BAT, который вызывает повторение указанной команды всякий раз с новым параметром:

 @echo off

 rem Многокpатное выполнение команды с pазличными паpаметpами

 if -%1 ==- goto inctruct

 if -%2 ==- goto inctruct

 set cmd=%1

 :docmd

 echo %cmd% %2

 %cmd% %2

 shift

 if not -%2 ==- goto docmd

 goto end

 :inctruct

 echo Ошибка: пpопущены паpаметpы

 echo Используйте:

 echo REPEAT команда паpаметp [паpаметр ...]

 :end

 set cmd=

 Для запуска этого файла следует набрать его имя REPEAT, а затем команду и список параметров, например:

 REPEAT DIR \*.EXE \*.COM

 По этой команде сначала будет выведен список файлов с расширением.EXE, а затем список файлов с расширением.COM.

 Файл REPEAT начинает свою работу с запоминания имени команды во временной переменной CMD. Затем управление передается организованному в файле циклу, в результате выполнения которого имя команды и первое значение параметра отображается на экране, после чего сформированная командная строка поступает в систему для выполнения. Далее при помощи оператора SHIFT осуществляется сдвиг значений параметров влево на одну позицию. При следующем выполнении тела цикла значение второго параметра команды замещает значение первого. Командный файл прекращает свою работу, достигнув конца списка значений параметров. Если набрать слово REPEAT без списка параметров, то на экран будет выдана краткая инструкция для пользователя.

 **11. Операции перенаправления**

 Многие программы (как системные, так и прикладные), обращаются при выполнении к стандартному устройству ввода для получения исходной информации и выводят диагностические сообщения или иные результаты своей работы на стандартное устройство вывода. Обычно стандартным устройством ввода является клавиатура, а стандартным устройством вывода - экран. Например, команда DIR формирует листинг заданного каталога и посылает его на экран, т.е. основной результат работы команды DIR выдается на стандартное устройство вывода. В других случаях намэкран поступают только вспомогательные, диагностические сообщения. Так, назначение команды COPY - копирование файлов, однако, выполнив (или не выполнив) копирование, команда COPY выводит на экран краткое диагностическое сообщение; так же работают команды

FORMAT, CHKDSK и многие другие.

 Стандартный ввод или стандартный вывод могут быть перенаправлены на другое внешнее устройство или в файл. Для обозначения перенаправления вывода служит символ **>** . Например, команда

 DIR > PRN

выведет листинг текущего каталога не на экран, а на принтер.

Пробелы вокруг символа > можно опустить: DIR>PRN

Команда

 COPY FILE1.TXT FILE1.DOC > LOG.LST

выполнит свою основную задачу, как обычно, т.е. скопирует содержимое файла FILE1.TXT в файл FILE1.DOC, а диагностическое сообщение о результатах копирования выведет не на экран, а в файл LOG.LST (который будет ею создан). Если файл уже существует, он стирается, и создается новый файл с тем же именем. Причем, в спецификации файла недопустимы шаблоны \* и ? .

 Предположим, что файл LOG.LST из предыдущего примера представляет собой журнал копирования, т.е. мы хотим накапливать в нем результаты последовательных операций копирования. В этом случае символ > следует заменить на символ >> :

 COPY FILE1.TXT FILE1.DOC >> LOG.LST

Тогда каждая команда COPY будет добавлять свое сообщение к уже имеющимся в файле .

 Аналогично можно перенаправить и стандартный ввод. Если, например, программа EDITOR выполняет форматирование текста, вводимого с клавиатуры, и выводит отформатированный текст на экран, то команда

 EDITOR < MYTEXT.TXT

заставит программу EDITOR вводить исходный текст из файла MYTEXT.TXT, а команда

 EDITOR < AUX

приведет к вводу в EDITOR информации, поступающей в компьютер по линии связи через последовательный порт (за которым обычно закрепляется имя AUX). С помощью команды

 EDITOR < NOFORM.TXT > FORM.DOC

можно перенаправить как ввод, так и вывод: программа EDITOR получит исходный текст из файла NOFORM.TXT, а результат отправит в файл FORM.DOC. Строго говоря, сама программа ничего не знает об этих перенаправлениях - она во всех случаях обращается к стандартному устройству ввода для ввода информации и к стандартному устройству вывода для ее вывода. Однако OS как бы подставляет ей на входе и выходе другие устройства.

 **12. Некоторые команды выполняемые Windows**

Для получения сведений об определенной команде наберите HELP <имя команды>

ASSOC Вывод либо изменение сопоставлений по расширениям имен файлов.

AT Выполнение команд и запуск программ по расписанию.

ATTRIB Отображение и изменение атрибутов файлов.

BREAK Включение/выключение режима обработки комбинации клавиш CTRL+C.

CACLS Отображение/редактирование списков управления доступом (ACL) к файлам.

CALL Вызов одного пакетного файла из другого.

CD Вывод имени либо смена текущей папки.

CHCP Вывод либо установка активной кодовой страницы.

CHDIR Вывод имени либо смена текущей папки.

CHKDSK Проверка диска и вывод статистики.

CHKNTFS Отображение или изменение выполнения проверки диска во время загрузки.

CLS Очистка экрана.

CMD Запуск еще одного интерпретатора командных строк Windows.

COLOR Установка цвета текста и фона, используемых по умолчанию.

COMP Сравнение содержимого двух файлов или двух наборов файлов.

COMPACT Отображение/изменение сжатия файлов в разделах NTFS.

CONVERT Преобразование дисковых томов FAT в NTFS. Нельзя выполнить

 преобразование текущего активного диска.

COPY Копирование одного или нескольких файлов в другое место.

DATE Вывод либо установка текущей даты.

DEL Удаление одного или нескольких файлов.

DIR Вывод списка файлов и подпапок из указанной папки.

DISKCOMP Сравнение содержимого двух гибких дисков.

DISKCOPY Копирование содержимого одного гибкого диска на другой.

DOSKEY Редактирование и повторный вызов командных строк; создание макросов.

ECHO Вывод сообщений и переключение режима отображения команд на экране.

ENDLOCAL Конец локальных изменений среды для пакетного файла.

ERASE Удаление одного или нескольких файлов.

EXIT Завершение работы программы CMD.EXE (интерпретатора командных строк).

FC Сравнение двух файлов или двух наборов файлов и вывод различий между ними.

FIND Поиск текстовой строки в одном или нескольких файлах.

FINDSTR Поиск строк в файлах.

FOR Запуск указанной команды для каждого из файлов в наборе.

FORMAT Форматирование диска для работы с Windows.

FTYPE Вывод либо изменение типов файлов, используемых при сопоставлении по расширениям имен файлов.

GOTO Передача управления в отмеченную строку пакетного файла.

GRAFTABL Позволяет Windows отображать расширенный набор символов в графическом режиме.

HELP Выводит справочную информацию о командах Windows.

IF Оператор условного выполнения команд в пакетном файле.

LABEL Создание, изменение и удаление меток тома для дисков.

MD | MKDIR Создание папки.

MODE Конфигурирование системных устройств.

MORE Последовательный вывод данных по частям размером в один экран.

MOVE Перемещение одного или нескольких файлов из одной папки в другую.

PATH Вывод либо установка пути поиска исполняемых файлов.

PAUSE Приостановка выполнения пакетного файла и вывод сообщения.

POPD Восстановление предыдущего значения текущей активной папки,

 сохраненного с помощью команды PUSHD.

PRINT Вывод на печать содержимого текстовых файлов.

PROMPT Изменение приглашения в командной строке Windows.

PUSHD Сохранение значения текущей активной папки и переход к другой папке.

RD Удаление папки.

RECOVER Восстановление читаемой информации с плохого или поврежденного диска.

REM Помещение комментариев в пакетные файлы и файл CONFIG.SYS.

REN | RENAME Переименование файлов и папок.

REPLACE Замещение файлов.

RMDIR Удаление папки.

SET Вывод, установка и удаление переменных среды Windows.

SETLOCAL Начало локальных изменений среды для пакетного файла.

SHIFT Cдвиг содержимого подставляемых параметров для пакетного файла.

SORT Сортировка ввода.

START Запуск программы или команды в отдельном окне.

SUBST Сопоставляет заданному пути имя диска.

TIME Вывод и установка системного времени.

TITLE Назначение заголовка окна для текущего сеанса интерпретатора командных строк CMD.EXE.

TREE Графическое отображение структуры папок заданного диска или папки.

TYPE Вывод на экран содержимого текстовых файлов.

VER Вывод сведений о версии Windows.

VERIFY Установка режима проверки правильности записи файлов на диск.

VOL Вывод метки и серийного номера тома для диска.

XCOPY Копирование файлов и дерева папок.

Варианты заданий

1. Если задано 2 параметра, то выполнить команду, заданную первым параметром, и перенаправить ее результаты во второй файл-параметр. Если задан один параметр, то удалить этот файл. Если параметры не заданы, то вывести на экран содержимое этого командного файла.
2. Записать все файлы-параметры в один файл 1. txt, если эти файлы существуют.
3. Все файлы с расширениями .txt, .asm, .cpp скопировать в папку, заданную параметром командного файла.
4. Если файл-параметр существует, то удалить его, а если нет, то создать его и записать с него дату и время системы. Если параметр не задан, то вывести на экран содержимое этого командного файла.
5. Если задан один параметр, считая, что это путь , сделать этот путь текущим. Если задано больше параметров, то, считая что это файлы, удалить их.
6. Если первый параметр командного файла – это существующий файл, то скопировать его во второй параметр-путь, иначе перенаправить в файл с этим именем выполнение второго параметра –команды.
7. Записать в файл х%. txt информацию о файлах, находящихся в параметрах-каталогах.
8. Удалить все заданные файлы-параметры, если они существуют.
9. Если задан один параметр, то, считая что это файл, вывести его содержимое на экран. Если задано два параметра, то содержимое этих двух файлов скопировать в один файл х%. txt.
10. Выполнить копирование файлов, начинающихся с букв а, в, с в подкаталог , заданный параметром.
11. Выдать на экран содержимое всех папок-параметров с приостановкой работы после вывода каждой.
12. Скопировать все файлы-параметры, начиная со второго, по пути, заданному первым параметром пути. Если параметры не заданы, вывести на экран содержимое самого командного файла.
13. Если параметры не заданы, вывести на экран содержимое самого командного файла. Если задан один параметр, то перенаправить в него текущее время системы. Если задано два параметра, то скопировать первый параметр во второй параметр-путь.
14. Добавить в переменную окружения PATH директории, заданные в параметрах, если параметры не заданы, вывести на экран содержимое командного файла.
15. Если задано 2 параметра, то сравнить файлы-параметры (если они существуют). Если задан 1 параметр, то считая, что это путь, сделать его текущим.

## Лабораторная работа № 3

**Главная загрузочная запись MBR**

***Цель работы:*** Научиться определять наличие, тип логических дисков, исходя из информации, полученной чтением MBR.

***Теоретические сведения***

Вся информация, необходимая для первоначальной загрузки ОС, находится в первом секторе жесткого диска. Эта информация называется главной загрузочной записью, MBR(Master Boot Record). Структура MBR приведена в таблице 3.1.

Таблица 3. 1. Структура MBR

|  |  |  |
| --- | --- | --- |
| смещение | длина | Описание |
| 000h | 446 | Код загрузчика |
| 1BEh | 64 | Таблица разделов |
| 1BEh | 16 | Раздел 1 |
| 1CEh | 16 | Раздел 2 |
| 1DEh | 16 | Раздел 3 |
| 1EEh | 16 | Раздел 4 |
| 1FEh | 2 | Сигнатура 55ААh |

MBR находится в начале жесткого диска по физическому адресу 0-головка, 0-дорожка, 1 - сектор. При использовании абсолютного адреса номер сектора, начиная с которого расположен MBR, равна нулю. Код чтения MBR представлен ниже

Структура MBR содержит четыре раздела Partition Table (PT), следовательно на одном жестком диске может быть расположено до четырех основных разделов, один из которых может быть загрузочным. Если раздел является загрузочным, то размещенный на нем логический диск должен содержать готовую к работе копию операционной системы, которой будет передано управление при загрузке компьютера с жесткого диска. Структура элемента (партиции) Partition Table приведена в таблице 2.

Таблица 3.2 Структура элемента Partition Table

|  |  |  |
| --- | --- | --- |
| смещение | длина(байт) | Описание |
| 00h | 1 | Признак активности раздела |
| 01h | 1 | Начало раздела – головка |
| 02h | 1 | Начало раздела - сектор (биты 0-5), дорожка (биты 6,7) |
| 03h | 1 | Начало раздела - дорожка  |
| 04h | 1 | Код типа раздела |
| 05h | 1 | Конец раздела – головка |
| 06h | 1 | Конец раздела - сектор (биты 0-5), дорожка (биты 6,7) |
| 07h | 1 | Конец раздела - дорожка  |
| 08h | 4 | Относительный номер первого сектора раздела |
| 0Ch | 4 | Количество секторов раздела |

Поле, находящееся по 4 смещению в партиции определяет тип файловой системы и имеет следующие значения:

Тип Описание

0x00 Пусто

0x01 FAT12, CHS

0x04 FAT16, 16-32 Мбайт, CHS

0x05 Расширенный раздел Microsoft, CHS

0x06 FAT16, 32 Мбайт-2 Гбайт, CHS

0x07 NTFS

0x0b FAT32, CHS

0х0с FAT32, LBA

0х0е FAT16, 32 Мбайт-2 Гбайт, LBA

0x0f Расширенный раздел Microsoft, LBA

0x82 Solaris x86

0x82 Раздел подкачки Linux

0x84 Данные спящего режима

0x85 Расширенный раздел Linux

0x86 Набор томов NTFS

0x87 Набор томов NTFS

0ха8 Mac OSX

0xab Mac OSX

0xfb Файловая система Vmware

Если четырех разделов не достаточно для конфигурирования системы, используется расширенный раздел. Расширенный раздел может содержать несколько логических дисков. Запись РТ расширенного раздела содержит адрес вторичной MBR. Устанавливаем адрес первой вторичной MBR равной адресу начала расширенного раздела (ADR\_BEG\_EXP), и по данному адресу читаем сектор. Первый элемент PT во вторичной MBR характеризует логический диск в расширенном разделе. Если второй элемент PT равен нулю, следовательно информация в данном разделе жесткого диска исчерпана, иначе проверяем код системы на 05h или 0Fh, и вычисляем адрес следующей вторичной MBR следующим образом:

ADR\_BEG\_EXP + OTN\_ADR,

где ADR\_BEG\_EXP - адрес начала расширенного раздела

 OTN\_ADR - относительный номер первого сектора раздела

По этому адресу считываем очередную вторичную MBR и повторяем действия выполняемые с первой вторичной MBR.

***Алгоритм именования дисков***

Алгоритм именования учитывает следующие действия:

1. Поиск всех логических дисков согласно ранее приведенным алгоритмам;

2. Для каждого логического диска в соответствии с файловой системой с загрузочного сектора определить серийный номер;

3. Определить количество и имена логических дисков, созданных операционной системой, воспользовавшись для этого функцией GetLogicalDrives.

4. Для каждого логического диска через функцию GetVolumeInformation получить серийный номер;

5. Выполнить сравнение серийных номеров найденных в п.5 и п.2, совпадение их позволит назначить логическим дискам, найденным в п.1 соответствующие имена.

***Функции API для чтения информации с диска***

Для чтения с диска прежде всего диск необходимо открыть – т.е. получить доступ к нему. Это выполняет функция **CreateFile.**

**HANDLE CreateFile( LPCTSTR lpFileName**

 **DWORD dwDesiredAccess**

 **DWORD dwShareMode**

 **LPSECURITY\_ATTRIBUTES lpSecurityAttributes**

 **DWORD dwCreationDisposition**

 **DWORD dwFlagsAndAttributes**

 **HANDLE hTemplateFile**

**);**

Устройство (жесткий диск) открывается как файл с особым именем (задается параметром **lpFileName)**:

* \\.\PhysicalDrive0 нулевое устройство
* \\.\PhysicalDrive1 первое устройство
* …

Изменяя номера устройств, получаем доступ к следующим жестким дискам. Условием окончания цикла по жестким дискам является неуспешное открытие очередного диска. В этом случае функция CreateFile возвращает значение INVALID\_HANDLE\_VALUE, а функция GetLastError - ERROR\_FILE\_NOT\_FOUND.

 HANDLE hDrive = CreateFile("\\\\.\\PhysicalDrive0", GENERIC\_READ

 |GENERIC\_WRITE, FILE\_SHARE\_READ | FILE\_SHARE\_WRITE, 0,

 OPEN\_EXISTING, FILE\_ATTRIBUTE\_NORMAL, 0);

 if (INVALID\_HANDLE\_VALUE == hDrive)

 if (GetLastError()==ERROR\_FILE\_NOT\_FOUND)

 printf( " Все HD просканированы\n ");

После успешного открытия устройства, можно читать данные. Однако чтение будет производиться с текущей позиции в файле (устройстве). Первичная MBR находится в самом начале диска, поэтому перед ее чтением позиционировать указатель не нужно, однако для чтения вторичной MBR это делать уже необходимо.

Для позиционирования указателя файла используется функция:

DWORD **SetFilePointer**( HANDLE hFile

 LONG lDistanceToMove

 PLONG lpDistanceToMoveHigh

 DWORD dwMoveMethod );

Параметры:

hFile - хэндл на файл;

lDistanceToMove - младшие 32 бита знаковой величины в байтах, на которую нужно сместить указатель файла;

lpDistanceToMoveHigh - указатель на старших 32 бита знаковой величины в байтах, на которую нужно сместить указатель файла;

dwMoveMethod - указывает позицию, в отношении которой указатель надо двигать. Может принимать одно из значений:

FILE\_BEGIN - отсчет от начала файла

FILE\_CURRENT - отсчет от текущей позиции указателя файла

FILE\_END - отсчет от конца файла

Возращаемое значение.

Если операция завершилась успешно и lpDistanceToMoveHigh равное NULL, то возращаемое значение равное lDistanceToMove. Иначе возвращается 0xFFFFFFFF.

Функцией **ReadFile** можно прочитать MBR:

**BOOL ReadFile( HANDLE hFile**

 **LPVOID lpBuffer**

 **DWORD nNumberOfBytesToRead**

 **LPDWORD lpNumberOfBytesRead**

 **LPOVERLAPPED lpOverlapped**

 **);**

hFile – хэндл на файл;

lpBuffer – указатель на буфер, в который читаются данные;

nNumberOfBytesToRead – количество запрошенных для чтения байт;

lpNumberOfBytesToRead – указатель на число прочитанных байт.

Возвращаемое значение - не ноль, если функция выполнена успешно.

Общий алгоритм чтения MBR может иметь следующий вид:

ReadMBR (Adr)

Позиционирование указателя диска на Adr

Чтение MBR

(ReadFile)

i=0;i<4;i++

Тип раздела=0

Информация о разделе

расширенный раздел

Определение буквы диска

ReadMBR (Adr++ OTN\_ADR)

выход

 ***+ +***

 ***+ -***

Параметром функции ReadMBR является адрес, с которого нужно читать MBR. После чтения MBR производится цикл по четырем партициям (разделам) Partition Table. Если код типа раздела (смещение 4) равен ноль, значит раздел не используется и после него используемых разделов нет, т.е. является признаком конца Partition Table. Если код типа раздела указывает на расширенный раздел, то производится рекурсивный вызов функции ReadMBR. Значение, хранящееся в партиции по смещению 8 (OTN\_ADR на блок-схеме), для не расширенных разделов является адресом начала раздела (адрес ВООТ-сектора). Для расширенных разделов это поле содержит адрес вторичной MBR относительно предыдущего расширенного раздела (для первой вторичной MBR это абсолютный адрес).

***Требования к программе:***

1. Необходимо определить список всех логических дисков в системе.
2. Необходимо учесть наличие нескольких жестких дисков на компьютере.
3. Имена логических дисков должны совпадать с именами дисков в системе.
4. Для каждого логического диска определить: тип файловой системы, начало диска, его размер.

***Варианты заданий***

1. Определить активные разделы жесткого диска.
2. Вывести неиспользуемые разделы в каждой MBR.
3. Определить количество MBR в системе.
4. Для каждого логического диска вывести его серийный номер (из ВООТ-сектора).
5. Для каждого логического диска определить размер сектора (из ВООТ-сектора).
6. Для каждого логического диска определить размер кластера (из ВООТ-сектора).
7. Для каждого расширенного раздела вывести его относительный номер сектора.
8. Для каждого расширенного раздела вывести его размер.
9. Подсчитать количество используемых разделов в каждой MBR.
10. Для логических дисков с файловой системой FAT вывести размер FAT.
11. Для логических дисков с файловой системой FAT вывести метку тома.
12. Для логических дисков с файловой системой NTFS вывести размер записи MFT.
13. Для каждого расширенного раздела вывести его абсолютный номер сектора.
14. Для активных разделов вывести CHS-адрес начала раздела.
15. Для активных разделов вывести CHS-адрес конца раздела.

***Вопросы к защите***

1. Назначение MBR, его структура и расположение на диске
2. Расширенный раздел. Назначение, расположение на диске
3. Структура партиции Partition Table
4. Алгоритм именования дисков
5. Просмотр MBR нескольких жестких дисков

## Лабораторная работа №4

**Организация просмотра корневого каталога FAT-раздела**

***Цель работы:*** Получить навыки использования функций Win API для просмотра корневого каталога FAT-раздела

***Теоретические сведения***

**Структура логического диска**

Физическая модель файловой системы FAT32 включает три физические области (рис 4.1). Первая область называется зарезервированной; ее размер определяется в загрузочном секторе. Вторая область - область FAT - содержит основные и резервные структуры FAT. Она начинается в секторе, который расположен за зарезервированной областью, а ее размер определяется количеством и размером структур FAT. Третья - корневой каталог в FAT 32 имеет произвольное положение в области данных. Область данных - содержит кластеры, выделенные для хранения файлов и директорий.

|  |  |  |
| --- | --- | --- |
| Зарезервированная область | Область FAT | Область данных |
|  | Корневойкаталог |  |

Рис 4.1 Физическая модель файловой системы FAT32

*Загрузочный сектор*

Загрузочный сектор располагается в первом секторе логического диска, поэтому и является частью зарезервированной области файловой системы. Microsoft называет некоторые данные первого сектора блоком параметров BIOS, или ВРВ (BIOS Parameter Block). Структура загрузочного сектора FAT32 приведена в табл. 4.1.

Таблица 4.1 Структура загрузочного сектора FAT32

|  |  |  |
| --- | --- | --- |
| Смещение в секторе | Размер | Содержание |
| 00 | 0 | 3 | Инструкция перехода на программу загрузки |
| 03 | 3 | 8 | Аббревиатура операционной системы |
| 0Bh | 11 | 2 | Число байтов в секторе (чаще всего 512) |
| 0Dh | 13 | 1 | Число секторов в кластере |
| 0Eh | 14 | 2 | Размер зарезервированной области (включая этот сектор) |
| 10h | 16 | 1 | Число таблиц FAT (чаще всего 2) |
| 11h | 17 | 2 | Число описателей файлов в корневом каталоге (в FAT32 - 0) |
| 13h | 19 | 2 | Общее число секторов на диске (если 0, то размер - в поле со смещением 20h) |
| 15h | 21 | 1 | Тип устройства |
| 16h | 22 | 2 | Размер одной FAT в секторах (0 в FAT32) |
| 18h | 24 | 2 | Число секторов на дорожке |
| 1Ah | 26 | 2 | Число головок |
| 1Ch | 28 | 4 | Количество скрытых секторов |
| 20h | 32 | 4 | Размер диска в секторах |
| 24h | 36 | 4 | Размер одной FAT в секторах |
| 28h | 40 | 2 | Номер главной таблицы FAT |
| 2Ah | 42 | 2 | Версия FAT32 (обычно 0) |
| 2Ch | 44 | 4 | Первый кластер корневого каталога (обычно 2) |
| 30h | 48 | 2 | Номер сектора структуры FSINFO (обычно 1) |
| 32h | 50 | 2 | Номер сектора - копии загрузочного (обычно 6) |
| 34h | 52 | 12 | Зарезервировано |
| 40h | 64 | 1 | Номер дисковода для функций BIOS |
| 41h | 65 | 1 | Зарезервировано |
| 42h | 66 | 1 | Сигнатура - 29h |
| 43h | 67 | 4 | Дата/время создания диска |
| 47h | 71 | 11 | Метка диска |
| 52h | 82 | 8 | Аббревиатура файловой системы |

Область FAT содержит одну или несколько структур FAT и начинается в секторе, находящемся за зарезервированной областью. Ее размер исчисляется умножением количества структур FAT на размер одной структуры; оба значения хранятся в загрузочном секторе. Ниже представлены формулы, по которым вычисляется начало и размер области FAT.

beginFAT=begin\_disk+BOOT.reservsect

size\_obl\_FAT=k\_copy\_FAT\*size\_FAT

где, beginFAT - начало области FAT (абсолютный адрес); size\_obl\_FAT - размер области FAT; begin\_disk - начало логического диска (абсолютный адрес); BOOT.reservsect - размер резервной области; k\_copy\_FAT - количество копий FAT; size\_FAT - размер одной копии FAT

В области данных находятся кластеры с содержимым файлов и каталогов; она начинается для FAT32 - за областью FAT . Размер области данных вычисляется как разница между размером логического диска и размером служебной области. Область данных разделена на кластеры, а количество секторов в кластере указывается в загрузочном секторе.

Согласно структуры логического диска корневой каталог для FAT32 размещен в области данных, адрес начала корневого каталога зафиксирован в BOOT. Динамический размер и местонахождение корневого каталога позволяют FAT32 адаптироваться к появлению поврежденных секторов в начале области данных и увеличивать размер каталога до необходимого размера, который определяется структурой FAT. Ниже приведены формулы, по которым вычисляется начало корневого каталога для FAT32:

beginROOT=begin\_disk+BOOT.reservsect+k\_copy\_FAT\*size\_FAT+(N\_klast-2) \* size\_klast

N\_klast - первый номер кластера корневого каталога

size\_klast - количество секторов в кластере

 *Дескрипторы файлов (записи каталога)*

В дескрипторах объектов FAT хранятся имя и данные файла или директории. Эти записи создаются для каждого объекта файловой системы. Структура записей поддерживает короткие имена по схеме 8 + 3 (8 символов имени, 3 символа расширения). Если файл имеет более сложное имя, в дополнение к обычной записи каталога (SFN) создаются записи LFN (Long File Name).В табл. 4.2 приведены поля, входящие в базовую структуру дескриптора каталога.

Таблица 4.2 Структура дескриптора с коротким именем

|  |  |
| --- | --- |
| **Диапазон****(байты)** | **Описание** |
| 0 | первый символ имени файла |
| 1-10 | символы 2-11 имени файла |
| 11 | атрибут файла |
| 12 | Зарезервировано |
| 13 | Время создания (десятые доли секунды) |
| 14-15 | Время создания (часы, минуты, секунды) |
| 16-17 | Дата создания |
| 18-19 | дата последнего обращения |
| 20-21 | старшие 2 байта адреса первого кластера файла, 0 для FAT12/16 |
| 22-23 | Время последней модификации |
| 24-25 | дата последней записи |
| 26-27 | младшие 2 байта адреса первого кластера |
| 28-31 | размер файла |

Первый байт структуры данных используется как признак занятого дескриптора. Если этот байт равен 0хЕ5 (5) или 0, - запись каталога свободна. В противном случае байт содержит первый символ имени файла. Конечно имена файлов задаются в кодировке ASCII, но они также могут содержать символы национальных алфавитов, для чего используются кодовые страницы Microsoft. Если имя файла короче 8 символов, неиспользуемые байты справа заполняются ASCII-кодом пробела (0x20). Поле размера файла занимает 4 байта, следовательно, максимальный размер файла равен 4 Гбайт. У каталогов поле размера равно 0, и для определения количества выделенных им кластеров необходимо использовать структуру FAT. Поле атрибутов содержит один или несколько флагов, перечисленных в табл. 4.3.

Таблица 4.3 Атрибуты файлов

|  |  |
| --- | --- |
| **Флаг** | **Описание** |
| 0х01 | Доступ только для чтения |
| 0х02 | скрытый файл |
| 0х04 | системный файл |
| 0х08 | метка тома |
| 0х0f | длинное имя файла |
| 0x10 | Каталог |
| 0x20 | архивный файл |

*Таблица FAT*

Таблица FAT занимает центральное место в файловой системе FAT и выполняет две функции. Она используется для определения состояния выделенных кластеров и для поиска следующего выделенного кластера файла или каталога. Обычно в файловой системе FAT хранятся две копии FAT, но точное их количество указывается в загрузочном секторе. Первая копия FAT начинается после зарезервированных секторов, размер которых указывается в загрузочном секторе. Размер одной копии FAT также хранится в загрузочном секторе. Вторая копия FAT начинается в следующем секторе за первой копией. Таблица состоит из записей одинакового размера (для FAT32 этот размер 4 байта (32 бита)). Адресация записей начинается с 0, и каждая запись соответствует кластеру с тем же адресом. В первых 2 элементах таблицы расположена служебная область, т.к. нумерация кластеров начинается с 2. .Если кластер свободен, его запись равна 0. Записи выделенных кластеров отличные от нуля и содержат адреса следующих кластеров в файле или каталоге. Если кластер завершает цепочку кластеров файла или каталога, его запись содержит маркер конца файла; в FAT32 - 0x0ffffff8-0x0fffffff. Если запись содержит 0x0ffffff7 в FAT32, кластер обозначен как поврежденный и не должен выделяться системой. Записи зарезервированных кластеров содержат значения 0x0ffffff0-0x0ffffff6 в FAT32. Общий диапазон кластеров доступных для выделения в файловой системе FAT32 0x00000002-0x0fffffef.

Алгоритм просмотра корневого каталога показан на рис.4.2

вход

-

+

-

+

+

-

+

выход

Buf[0]=0

Конец каталога

Определение из FAT следующего кластера ROOT

Buf[11]=15

LFN

Buf[0]=0хЕ5

Файл удален

Buf+=32

Имя файла…

Конец кластера

Чтение кластера ROOT в buf

Цикл по кластерам ROOT

Определение количества кластеров ROOT

Чтение BOOT

Открыть доступ к логическому диску

Для открытия логического диска, также как и для открытия жесткого диска, используется функция CreateFile. Однако, первый параметр функции задается в формате “\\.\\*:”, где \* - буква логического диска. Например,

hDrive = CreateFile(TEXT("\\\\.\\E:"),GENERIC\_READ | GENERIC\_WRITE,

 FILE\_SHARE\_READ | FILE\_SHARE\_WRITE,NULL,OPEN\_EXISTING,0,

 NULL);

Функция, которая подсчитывает количество кластеров, занятых объектом, приведена ниже. Входными параметрами являются хэндл диска, начальный сектор таблицы FAT и начальный номер кластера объекта.

unsigned int GetCountKlast(HANDLE hDrive, unsigned int BeginFAT, unsigned int Nklast)

{

 unsigned int flag = 1;

 unsigned int n = 1;

 int KlastRead = Nklast;

 while (flag)

 {

 KlastRead = ReadNextKlastFAT(hDrive, BeginFAT, KlastRead);

 if (KlastRead < 0x0FFFFFF8)

 n++;

 else

 flag = 0;

 }

 return n;

}

Функция ReadNextKlastFAT читает из FAT содержимое элемента, соответствующего заданному кластеру KlastRead. Этим содержимым является или номер следующего кластера в цепочке или значение 0x0FFFFFF8 в качестве признака конца цепочки кластеров.

unsigned int ReadNextKlastFAT (HANDLE hDrive, unsigned int BeginFAT, unsigned int NKlast)

{

 unsigned char \*buff = new unsigned char[512];

 unsigned int FAT\_Entry;

 int NSec;//Номер сектора для чтения

 int NEnt;//Номер записи в секторе

 int lDistance;

 NSec = NKlast\*KLAST\_SIZE;//KLAST\_SIZE–количество секторов в кластере

 NEnt = NSec%SEC\_SIZE;// SEC\_SIZE – количество байт в секторе;

 NSec = NSec / SEC\_SIZE;

 lDistance = (BeginFAT +NSec)\* SEC\_SIZE;

 SetFilePointer(hDrive, lDistance, NULL,FILE\_BEGIN);

 ReadFile(hDrive, buff, SEC\_SIZE, &dwByte, NULL);

 FAT\_Entry = \*((unsigned int \*)(buff + NEnt));

 delete[] buff;

 return FAT\_Entry;

}

***Варианты заданий***

Разработать алгоритм и программу, которая в корневом каталоге заданного логического диска выполняет следующие действия:

1. Поиск скрытых объектов
2. Сортировка по расширению
3. Поиск удаленных файлов
4. Определение последовательности кластеров для выбранного файла
5. Поиск файлов, длина которых более заданного количества кластеров
6. Сортировка по дате модификации
7. Поиск объектов с максимальным количеством символов в имени.
8. Сортировка по размеру дискового пространства
9. Поиск не фрагментированных файлов
10. Определение количества кластеров директорий
11. Поиск объектов, длина имени которых не превышает заданной величины.
12. Поиск объектов, созданных (модифицированных) в текущем году
13. Поиск файлов, имена которых удовлетворяют шаблону «\*.срр»
14. Поиск скрытых и доступных только для чтения файлов.
15. Поиск объектов, созданных после заданной даты.

***Вопросы к защите***

1. Определение адреса корневого каталога
2. Признак окончания корневого каталога
3. Переход к следующему элементу корневого каталога
4. Атрибуты файлов
5. Назначение и структура таблицы FAT
6. Просмотр цепочки кластеров объекта файловой системы
7. Логическая структура диска
8. Кластер. Размер кластера

## Лабораторная работа №5

**Организация просмотра состава директорий. Работа с длинными именами в FAT32**

***Цель работы:*** Получить навыки программирования с помощью API функций задач полного просмотра состава логического диска.

***Теоретические сведения***

*Длинные имена*

Стандартные записи каталога позволяют представить файл, имя которого состоит из 8 символов, а расширение - с 3. Если имя имеет большую длину или содержит специальные символы, для него нужны особые записи каталогов - записи LFN (Long File Name). Кроме всех записей LFN для файла создается обычная запись, причем записи LFN должны предшествовать обычной записи. Поля LFN записи каталога перечислены в табл. 3.5.

Таблица 5.1. Структура дескриптора с длинным именем

|  |  |
| --- | --- |
| **Диапазон** | **Описание** |
| 0 | порядковый номер |
| 1-10 | символы имени файла 1-5 (Unicode) |
| 11 | атрибут файла |
| 12 | Зарезервирован (0) |
| 13 | контрольная сумма |
| 14-25 |  символы имени файла 6-11 (Unicode) |
| 26-27 | Зарезервирован (0) |
| 28-31 | символы имени файла 13-13 (Unicode) |

Поле порядкового номера содержит счетчик записей, необходимых для хранения имени файла. Первой записи соответствует порядковый номер 1. Порядковый номер увеличивается на 1 для каждой записи LFN вплоть до последней, признаком которой является единица в шестом бите.

Записи LFN располагаются в обратном порядке перед записью короткого имени файла. Итак, первая запись, находящаяся в каталоге, является последней записью LFN для файла и имеет наибольший порядковый номер. Неиспользованные символы в имени дополняются кодами 0xff, а имя завершается символом NULL при наличии свободного места. Поле атрибутов записи LFN должно равняться 0x0F.

*Просмотр содержимого директории*

При обнаружении каталога (по атрибуту – 0x10) необходимо перейти по указанному номеру кластера, где будут находиться такие (табл. 4.2, 5.1) 32 байтные записи каталога. Поле «размер» в записи каталога не используется и всегда должно быть равно 0. Единственный способ определить размер каталога — использовать начальный кластер из записи каталога, перебирать элементы каталога, пока не встретится 0 в первом байте или не будет достигнут конец кластера. Если обнаружили конец кластера, то в FAT необходимо прочитать номер следующего кластера и следовать по цепочке кластеров структуры FAT до обнаружения маркера конца файла. Первые две записи каталога предназначены для элементов «.» и «..». Эти обозначения хорошо знакомы пользователям, работающим в режиме командной строки. Имя «.» обозначает текущий каталог, а имя «..» — родительский каталог.

Обычно функцию чтения каталога делают рекурсивной.

***Задания к лабораторной работе***

Выполнить задания лабораторной работы №4 с учетом длинных имен файлов и просмотром всех подкаталогов.