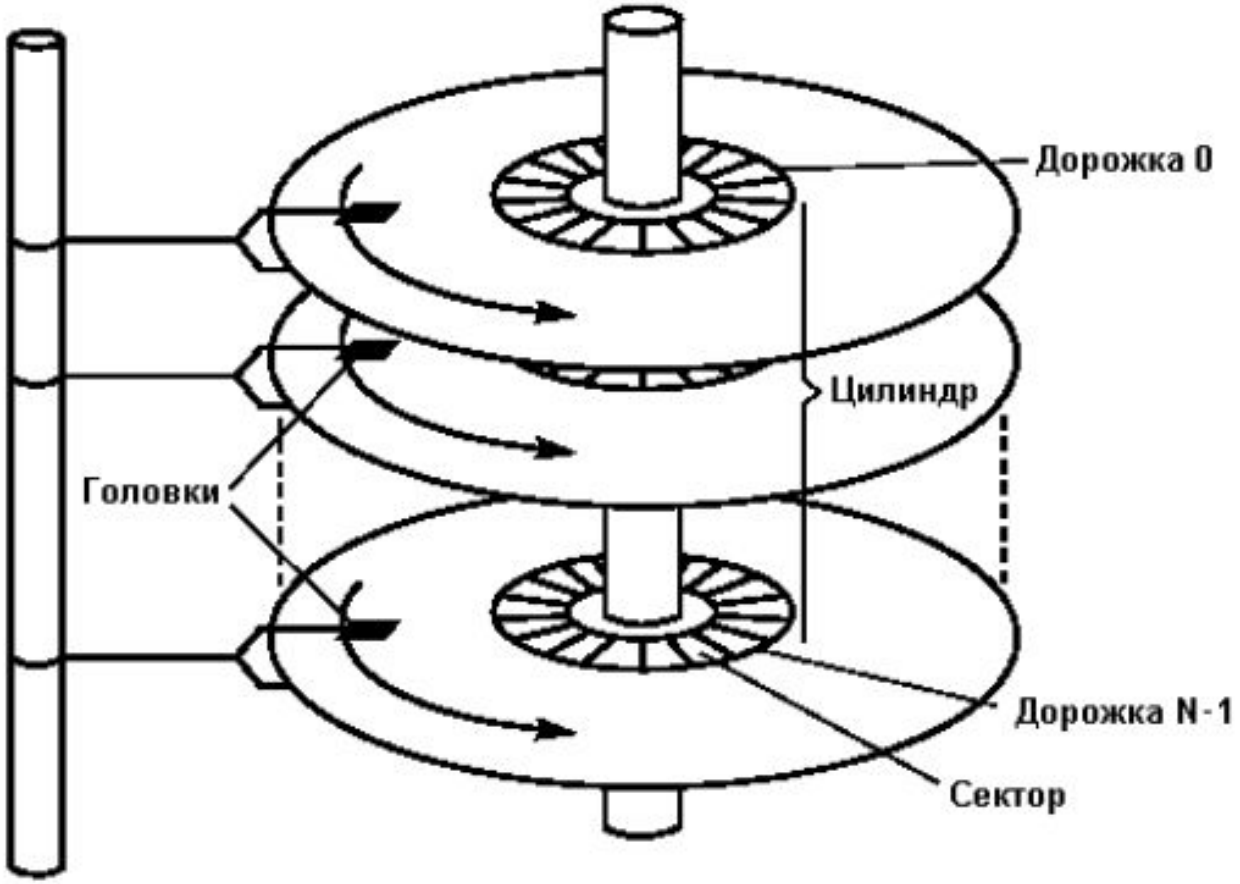
ОРГАНИЗАЦИЯ ВВОДА-ВЫВОДА В СОВРЕМЕННЫХ ОС

Общие принципы организации ввода-вывода

Система ввода-вывода ПК

Механизм управления вводом-выводом

Общие принципы размещения данных на магнитных дисках



Структура хранения информации на жестком диске

Упрощенная структура MBR

Смещение (Off-set)	Размер (Sise) (байт)	Содержимое (Contents)
0	446	Программа анализа Partition Table и загрузки System Bootstrap с активного раздела жесткого диска
+1BEh	16	Partition 1 entry (Описатель раздела)
+1CEh	16	Partition 2 entry
+1DEh	16	Partition 3 entry
+1EEh	16	Partition 3 entry
+1FEh	16	Сигнатура (AA55h)

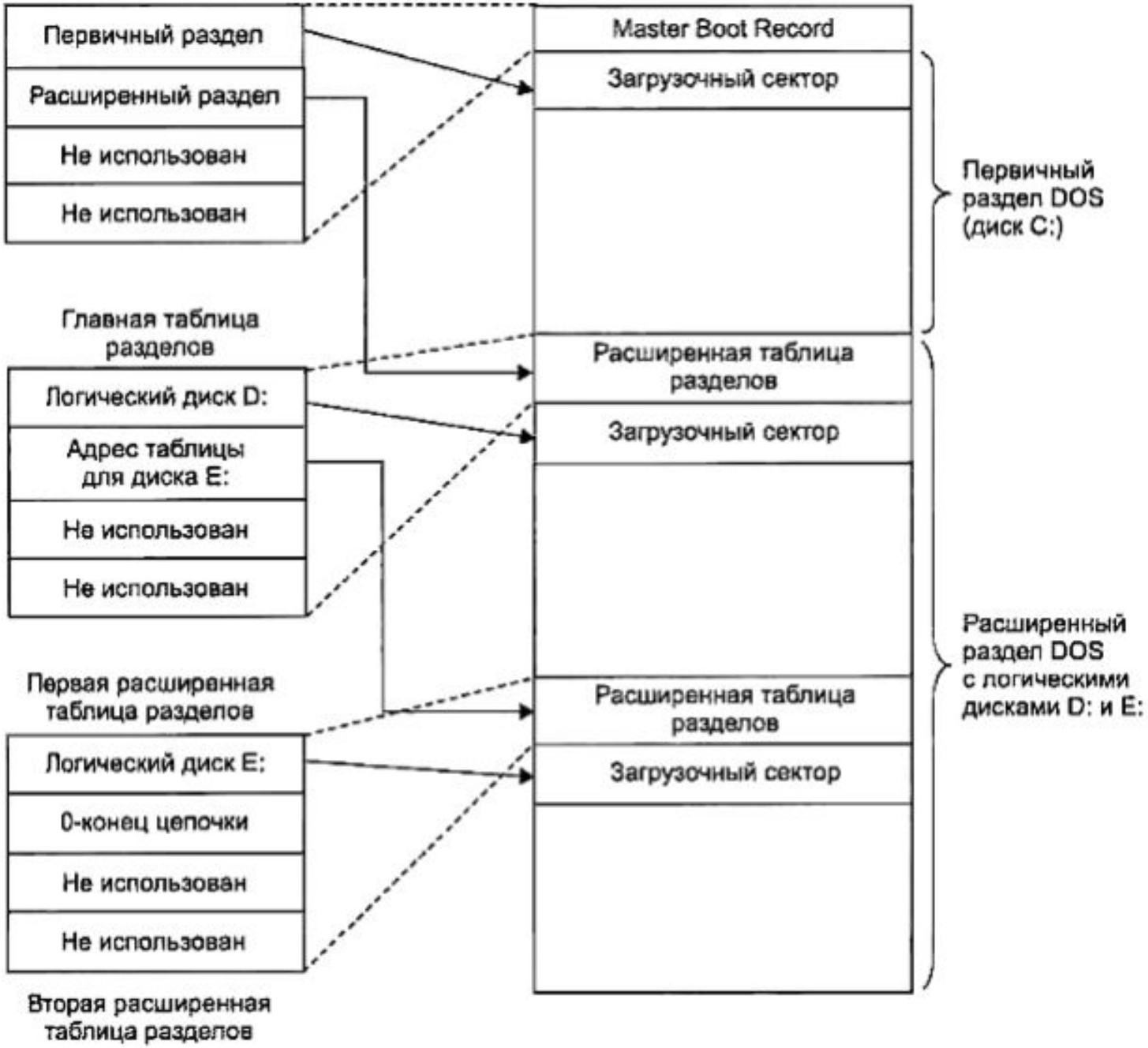
System ID	Тип раздела	System ID	Тип раздела
00	Empty («пустой» раздел)	41	PPC PreP Boot
01	FAT12	42	SFS
02	XENIX root	4D	QNX4.x
03	XENIX usr	4E	QNX 4.x 2nd part
04	FAT16 (<32 Мбайт)	4F	QNX 4.x 3nd part
05	Extended	50	OnTrack DM
06	FAT16	51	OnTrack DM6 Aux
07	HPFS/NTFS	52	CP/M
08	AIX	53	OnTrack DM6
09	AIX bootable	54	OnTrack DM6
0A	OS/2 Boot Manager	55	EZ Drive
0B	Win95 FAT32	56	Golden Bou
0C	Win95 FAT32 LBA	5C	Priam Edisk
0E	Win95 FAT16 LBA	61	Speed Stor
0F	Win95 Extended	64	Novell Netware
10	OPUS	65	Novell Netware
11	Hidden FAT12	75	PC/IX
12	Compaq diagnost	80	Old Minix
14	Hidden FAT16 (<32 Мбайт)	82	Linux swap
16	Hidden FAT16	83	Linux native
17	Hidden HPFS/NTFS	84	OS/2 hidden C:
18	AST Windows swap	85	Linux Extended
1B	Hidden Win95 Fat	86	NTFS volume set
1C	Hidden Win95 Fat	A5	BSD/386
1E	Hidden Win95 Fat	A6	Open BSD
24	NEC DOS	A7	Next Step
3C	Partition Magic	EB	Be OS
40	Venix 80286		

наиболее известные идентификаторы

Формат элемента таблицы разделов

Название записи элемента Partition Table	Длина, байт
Флаг активности раздела	1
Номер головки начала раздела	1
Номер сектора и номер цилиндра загрузочного сектора раздела	2
Кодовый идентификатор операционной системы	1
Номер головки конца раздела	1
Номер сектора и цилиндра последнего сектора раздела	2
Младшее и старшее двухбайтовое слово относительного номера начального сектора	4
Младшее и старшее двухбайтовое слово размера раздела в секторах	4

Биты номера цилиндра								Биты номера сектора							
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0


Первичный раздел
Расширенный раздел
Не использован
Не использован
Главная таблица разделов
Логический диск D:
Адрес таблицы для диска E:
Не использован
Не использован
Первая расширенная таблица разделов
Логический диск E:
0-конец цепочки
Не использован
Не использован
Вторая расширенная таблица разделов
Master Boot Record
Загрузочный сектор
Первичный раздел DOS (диск C:)
Расширенная таблица разделов
Загрузочный сектор
Расширенная таблица разделов
Загрузочный сектор
Расширенный раздел DOS с логическими дисками D: и E:

Организация файлового ввода-вывода в ОС Windows
API-функции для организации ввода-вывода

```
HANDLE CreateFile(LPCTSTR lpFileName,
  DWORD dwDesiredAccess, DWORD
  dwShareMode, LPSECURITY_ATTRIBUTES
  lpSecurityAttributes, DWORD
  dwCreationDispostion, DWORD
  dwFlagsAndAttributes, HANDLE
  hTemplateFile);
```

Аргумент	Описание
lpFileName	Указатель на имя файла или устройства
DesiredAccess	Устанавливает вид доступа к объекту. Используются флаги GENERIC_READ (чтение), GENERIC_WRITE (запись) или оба при помощи оператора логического сложения. Значение 0 указывает на необходимость проверки возможности доступа.
dwShareMode	Набор битовых флагов, указывающий на режим совместного доступа к объекту. Если значение dwShareMode равно 0, совместный доступ к объекту запрещен. Флаги FILE_SHARE_DELETE (совместное удаление), FILE_SHARE_READ (совместное чтение) и FILE_SHARE_WRITE (совместная запись).
lpSecurityAttributes	Указатель на структуру SECURITY_ATTRIBUTES, которая определяет, будет ли создаваемый дескриптор наследоваться дочерними процессами. Если аргумент lpSecurityAttributes имеет значение NULL, дескриптор не будет наследоваться.
dwCreationDisposition	Указывает, каким образом следует создать (или открыть) файл. Допускается использовать следующие значения: CREATE_NEW (создать новый файл; если файл существует, функция не срабатывает), CREATE_ALWAYS (создать новый файл; если файл существует, он перезаписывается), OPEN_EXISTING (открыть файл; если файл не существует, функция не срабатывает), OPEN_ALWAYS (открыть файл; если файл не существует, он создается) или TRUNCATE_EXISTING (открыть файл и сделать его равным нулю; если файл не существует- не срабатывает).
dwFlagsAndAttributes	Набор атрибутов и флагов, которыми должен обладать файл. Например, если требуется, чтобы новый файл был скрытым, используйте значение FILE_ATTRIBUTE_HIDDEN. Другие возможные значения флагов перечислены в табл. 5.5.
hTemplateFile	Содержит дескриптор с доступом GENERIC_READ. Это дескриптор шаблонного файла, атрибуты которого (включая расширенные) будут присвоены создаваемому файлу.

Комбинации флагов

Флаг	Значение
1	2
FILE_FLAG_WRITE_THROUGH	Приказывает Windows осуществлять немедленную запись данных на диск. При этом возможно использование КЭШа.
FILE_FLAG_NO_BUFFERING	Приказывает системе открыть файл без использования кэширования или буферизации, что дает максимальную производительность.
FILE_FLAG_RANDOM_ACCESS	Оповещает систему о том, что доступ к файлу осуществляется случайным образом. Система может использовать это обстоятельство для оптимизации кэширования файла.
FILE_FLAG_SEQUENTIAL_SCAN	Оповещает систему о том, что доступ к файлу осуществляется последовательно от начала к концу файла. Система может использовать это обстоятельство для оптимизации кэширования файла.
FILE_FLAG_OVERLAPPED	Приказывает системе инициализировать объект для перекрывающегося ввода/вывода.
FILE_FLAG_DELETE_ON_CLOSE	Приказывает системе уничтожить файл сразу же после того, как все его дескрипторы будут закрыты.
FILE_FLAG_BACKUP_SEMANTICS	Указывает на то, что файл предназначен для операций резервного копирования или восстановления из резервной копии. Операционная система разрешает вызывающему процессу любой доступ к файлу при условии, что вызывающий процесс обладает привилегиями SE_BACKUP_NAME и SE_RESTORE_NAME.
FILE_FLAG_POSIX_SEMANTICS	Доступ к файлу осуществляется в соответствии с правилами POSIX. При этом разрешается использовать несколько различных файлов, имена которых отличаются только регистром букв. Такие файлы разрешается создавать только в системах, поддерживающих подобную схему именования файлов.
FILE_FLAG_OPEN_REPARSE_POINT	Подавляет поведение, свойственное для точек грамматического разбора (reparse points) файловой системы NTFS. Когда файл открывается, вызывающему процессу возвращается его дескриптор вне зависимости от того, работоспособен ли фильтр, контролирующий точку грамматического разбора, или нет. Этот флаг не может использоваться совместно с флагом CREATE_ALWAYS.
FILE_FLAG_OPEN_NO_RECALL	Информирует систему о том, что вызывающее приложение запрашивает данные, хранящиеся в файле, однако файл может продолжать оставаться на удаленном носителе данных. Этот флаг используется удаленными системами хранения данных или совместно с системой Hierarchical Storage Management.

```
 BOOL ReadFile( HANDLE hFile;                    // дескриптор файла
 LPVOID lpBuffer;                  // буфер для временного хранения
                                          // прочитанных данных
 DWORD dwBytesToRead;              // количество байтов, которые
                                  // должны быть прочитаны LPDWORD
lpdwBytesRead;                          // возвращает количество
                                           //    прочитанных байтов
 LPOVERLAPPED lpOverlapped ); // поддержка асинхронного
                                           // ввода/вывода
 BOOL WriteFile( HANDLE hFile,      // дескриптор файла
 CONST VOID *lpBuffer,                    //указывает данные, которые
                                          // должны быть записаны в файл
 DWORD dwBytesToWrite,             // количество записываемых байтов
LPDWORD lpdwBytesWritten,          // возвращает количество
                                          // записанных байтов
 LPOVERLAPPED lpOverlapped ); // задает поддержку
                                          // асинхронного ввода/вывода
```

Чтобы закрыть файл, используется функция CloseHandle(). Эту функцию можно использовать не только для закрытия дескрипторов файлов. С ее помощью можно закрыть любой другой дескриптор. BOOL

CloseHandle(HANDLE hObject);

```
#include <windows.h>
void MB(char *s) // Для удобства использования MessageBox
{ MessageBox(NULL, s, NULL, MB_OK | MB_ICONSTOP);
}
void docat(char *fname)     // основная подпрограмма
{
HANDLE f=CreateFile ( fname, GENERIC_READ, 0, NULL,
OPEN_EXISTING, 0, NULL);
HANDLE out=GetStdHandle(STD_OUTPUT_HANDLE);
if (f==INVALID_HANDLE_VALUE)
{
    MB("Не могу открыть файл");
    exit(1);
}
char buf[4096];
unsigned long n;
```

```
  do { unsigned long wct;
  if (!ReadFi1e(f, buf, sizeof(buf), &n, NULL))
  break;
  if (n)
  WriteFile(out, buf, n, &wct, NULL);
  }
  while (n==sizeof(buf)); // Если EOF, это условие не выполняется CloseHandle(f);
  }
  void main(int argc, char *argv[])
  {
  if (argc==1)
  {         // утилита cat - Ошибки фактически не обрабатываются
            // Любая ошибка вызывает аварийное завершение программы MB("Usage:
cat FILENAME [FILENAME ....]"); exit(9);
  }
  // Обработать все указанные файлы
  while (--argc) docat(*++argv);
  exit(0);
  }
```

Механизмы асинхронного ввода-вывода

BOOL GetOverlappedResult(HANDLE hFile, // дескриптор
// файла или устройства

LPOVERLAPPED lpOverlapped, // поддержка асинхронного
// ввода/вывода

LPDWORD lpNumberOfBytesTransferred, // количество
// переданных байт

BOOL bWait); // флаг ожидания

Чтобы прервать выполнение операции ввода/вывода, следует использовать функцию **Cansello**.

BOOL Cancello(HANDLE hFile); // дескриптор файла

```
BOOL ReadFileEx( HANDLE hFile,
                 LPVOID lpBuffer,
                 DWORD nNumberOfBytesToRead,
                 LPOVERLAPPED lpOverlapped,
                 POVERLAPPED_COMPLETION_ROUTINE
                 lpCompletionRoutine );

BOOL WriteFileEx( HANDLE hFile,
                  LPCVOID lpBuffer,
                  DWORD nNumberOfBytesToWrite,
                  LPOVERLAPPED lpOverlapped,
                  LPOVERLAPPED_COMPLETION_ROUTINE
                  lpCompletionRoutine );
```

Порты завершения ввода/вывода

```
HANDLE CreateIoCompletionPort
(HANDLE FileHandle,                 // дескриптор файла HANDLE
ExistingCompletionPort,                 // дескриптор создаваемого
                                    // (или открываемого) порта
завершения
ULONG_PTR CompletionKey,                // ключ завершения,
                                    // вставляемый в каждый пакет
DWORD NumberOfConcurrentThreads ); //количество
                                        // подключаемых потоков
```

```
  BOOL GetQueuedCompletionStatus(
  HANDLE CompletionPort,                //дескриптор порта
LPDWORD lpNumberOfBytes,     // количество переданных байт
PULONG_PTR lpCompletionKey,      //указатель на ключ
                                  // завершения, (если он объявлен
ранее)
  LPOVERLAPPED *lpOverlapped, //указатель на Overlapped
DWORD dwMilliseconds ); // время ожидания пакета
```

```
PostQueuedCompletionStatus( HANDLE
   CompletionPort, DWORD
   dwNumberOfBytesTransferred, ULONG_PTR
   dwCompletionKey, LPOVERLAPPED
   lpOverlapped );
```

Информация об ошибках системной функции Windows

Наиболее экзотической является получение информации об ошибках в MS Windows. Во первых, отсутствует какое-либо подобие систематичности в системных функциях. Возвращаемые значения системных функций могут быть описаны как VOID, BOOL, HANDLE, PVOID, LONG или DWORD.

Функция GetLastError() возвращает последнюю ошибку, возникшую в ходе выполнения программы (точнее нити программы). Именно это 32-битное значение дает код ошибки. Собственно коды ошибок, общие для всех системных функций, содержатся в заголовочном файле WinError.h.

Текстовая информация об ошибке в Windows

Числовые коды ошибок, возвращаемые функцией GetLastError(), достаточно сложно для разработчика соотнести с наименованием ошибки. Если требуется распознавание вида ошибки при автоматическом выполнении программы, то разработчики этой ОС предлагают для использования специальную функцию FormatMessage.

FormatMessage(DWORD dwFlags, LPCVOID lpSource,

DWORD dwMessageId, DWORD dwLanguageId,

LPTSTR lpBuffer, DWORD nSize, va_list *Arguments).

Простейшее использование FormatMessage

len=FormatMessage(FORMAT_MESSAGE_FROM_SYSTEM,
NULL, k, // k - номер ошибки от GetLastError()
MAKELANGID(LANG_NEUTRAL, SUBLANG_DEFAULT),
txtmess, sizeof(txtmess), NULL);

где имя *txtmess* определено предварительно в описании вида char txtmess[300]. Возвращаемое функцией значение дает число символов в тексте, сформированном ею в заданном буфере.

Возвращает порядка тысячи различных наименований ошибок на языке текущей локализации.

Техническая проблема использования для русского языка

Существенной особенностью рассматриваемой функции оказывается использование возвращаемых текстов сообщений об ошибках, представленных в кодировке для графического режима. Исторически сложилось так, что тексты, записанные не с помощью латинского алфавита, имеют различное представления в графическом и текстовом режиме.

Решение возникающих при этом проблем обеспечивается парой функций преобразования из одной формы представления в другую.

BOOL CharToOem(char *textsource, char *textresult),

BOOL OemToChar(char *textsource, char *textresult).

Особенности наименования функции перекодировки

Разработчики условным буквосочетанием Char в названии функций обозначают кодировку графического режима, а обозначением Oem в названии – кодировку текстового режима. Таким образом функция CharToOem задает преобразование текста из кодировки графического режима Windows в текстовый режим, а функция OemToChar – преобразование текста из кодировки текстового режима в графический режим.

МНОГОФУНКЦИОНАЛЬНЫЙ КОНСОЛЬНЫЙ ВЫВОД

Классические языки высокого уровня не содержат средств управления позицией вывода на экране и цветом символов текста. Когда создавались эти языки, подобные средства были недоступны по аппаратным причинам. Глубокая связь упомянутых возможностей с конкретной аппаратурой привела к тому, что подобные средства оказались зависимыми от операционной системы. Сейчас мониторы поддерживают как позиционирование курсора, так и многоцветные изображения, но особенности управления этими возможностями по-прежнему оказываются зависимыми от операционных систем.

Управление курсором

Большинство программных средств для вывода текста выводят этот текст не просто на экран или в консольное окно, а в место, указываемое *текстовым курсором* (специальным символом указания на текущее место для вывода). Поэтому для вывода в место экрана, желаемого на текущем момент программистом, требует предварительного перемещения курсора в это место. Дополнительно могут использоваться функции вывода с указанием в них же места вывода (редко).

Установка курсора в текстовом режиме

Такое действие должно задаваться чем-то в виде ***функций_установки*(X-позиция, Y-позиция)**.

В Windows функция задания места курсора внутри текстового окна:

SetConsoleCursorPosition(HANDLE hConsOut,
COORD pos).

Координаты места должны быть предварительно помещены в экземпляр структуры типа COORD, в которой два поля с именами X и Y. Другой аргумент функции — действующий хэндл стандартного вывода (обычно). Возвращает значение типа BOOL. Координаты задают *знакоместа* (фиксированные позиции для символов), а не позицию отдельных пикселов.

Получение координат курсора

GetConsoleScreenBufferInfo(HANDLE hConsOut, CONSOLE_SCREEN_BUFFER_INFO* pInfo), где последний аргумент возвращаемый, должен быть подготовлен как экземпляр структуры и содержит ряд полей. Главные из них описываются как

COORD dwSize; COORD dwCursorPosition;

Они возвращают пару значений из ширины и высоты окна и координаты курсора в нем.

Управляющие последовательности как средство управления выводом

В операционных системах Unix и Linux для управления курсором и некоторых других действий с экраном и текстовым окном предназначены *управляющие последовательности*. Идея их использования расширяет управляющие символы, которые в языке Си и Unix служат основным средством управления выводом на экран. Управляющие последовательности определяются стандартом ANSI и называются также

ANSI-последовательностями.

Запись управляющих последовательностей на языке С

Управляющие последовательности начинаются со специального символа с десятичным эквивалентом 27. Этот код на языке Си в составе текстовых констант записывают в виде '\033'. Здесь использована универсальная форма записи произвольных (в том числе явно не изображаемых) символов в виде восьмеричных констант. Второй символ управляющих последовательностей - обязательный символ ‘[‘ (открывающаяся квадратная скобка), последним символом – латинская буква, детализирующая операцию. Иногда для такой детализации используется и предпоследний символ.

Управляющая последовательность позиционирования курсора

Для установки курсора служит управляющая последовательность, записываемая на языке Си как текстовая константа

"\033[*строка*;*столбец*H"

Здесь компоненты *строка* и *столбец* должны быть обязательно заданы десятичными числами и обязательно без дополнительных пробелов. Нумерация позиции считается от 1, так что установка в верхний левый угол экрана требует последовательности *\033*[1;1H .

Перемещение курсора в разные стороны

Последовательность

\033[*строка*A

приказывает переместить курсор на заданное в ней число строк вверх, последовательность

\033[*строка*B

– на заданное число строк вниз,

\033с[*столбец*C

– на заданное в ней число столбцов вправо, а

\033[*столбец*D

– на заданное число столбцов влево. Если при заданных значениях параметров курсор должен выйти за пределы экрана, то действие управляющей последовательности игнорируется.

Получение позиции курсора

Управляющая последовательность

\033[6n

выдает информацию о текущей позиции курсора в в виде текста **\033[строка;столбецR**

Программа должна считывать этот текст, задающий позицию курсора, *со стандартного устройства ввода сразу же после записи этой управляющей последовательности*. Что оказывается не очень удобно для современного стиля и привычек программирования.

Очистка части консольного окна и вывод повторяемых символов

В Windows для очистки окна или его части может быть использован вывод повторяющихся символов, в частности пробелов.

Для этого служат функции

FillConsoleOutputCharacter(HANDLE hConsOut,

CHAR char, WORD len, COORD pos, DWORD* actlen);

Очистка части консольного окна в Linux

Управляющая последовательность

\033[2J

очищает экран и перемещает курсор в исходное положение (строка 0, столбец 0)

Управляющая последовательность

\033[K

удаляет все символы, начиная с позиции курсора до конца строки (включая символ в позиции курсора).

Управление цветом текста в Windows

Задание цвета вывода на «ближайшее будущее»

(установки цвета для вывода на экран консоли)

SetConsoleTextAttribute(HANDLE houtput, WORD attrib).

Задание цвета использует идею *атрибутов*. Понятие атрибутов включает как цвет собственно символов (в более точных терминах – цвет переднего плана – foreground), так и цвет фона знакового места для символа — background. Цвет здесь относиться не собственно к символу, а к знакоместу. Поэтому может быть установлен предварительно, одновременно с символом или изменен в дальнейшем.

Кодирование цвета в Windows

Для задания атрибутов в Windows можно использовать символические константы, которые заданы в файле wincon.h. Они имеют названия FOREGROUND_BLUE, FOREGROUND_GREEN, FOREGROUND_RED, FOREGROUND_INTENSITY, BACKGROUND_BLUE, BACKGROUND_GREEN, BACKGROUND_RED, BACKGROUND_INTENSITY. Для получения комбинированного цвета с их помощью нужно несколько из них соединить символами побитовой операции ИЛИ. Так для задания белового цвета символа - указать операнд в виде FOREGROUND_RED | FOREGROUND_GREEN | FOREGROUND_BLUE. (Неплохо представлять, что значения перечисленных выше констант есть соответственно 1, 2, 4, 8, 16, 32, 64 и 128.)

Предварительное или последующее задание атрибутов

FillConsoleOutputAttribute(HANDLE hConsOut,

WORD attr, DWORD len, COORD pos, DWORD* actlen);

Как бы «раскрашивает» заданными в атрибуте цветами len знаковых ячейки, начиная с позиции, задаваемой аргументом pos. Возвращаемое значение дает информацию, сколько ячеек удалось раскрасить. Может отличаться от заданного числа, если указанные ячейки выходят за пределы консольного окна.

Дополнительная возможность: вывод текста с одновременным раскрашиванием

Альтернативой использования **SetConsoleTextAttribute** является последовательное выполнение функций

FillConsoleOutputAttribute и

WriteConsoleOutputCharacter

WriteConsoleOutputCharacter(HANDLE hConsOut,

CSTR* text, DWORD len, COORD pos,

DWORD* actlen);

Установка цвета для последующего вывода в Linux

Используются управляющие последовательности

\033[*цвет***m**

где компонент *цвет* задается одним или несколькими десятичными числами, разделяемыми символами «точка с запятой» (без пробелов).

Для кодирования цвета служат значения из следующей таблицы

Таблица атрибутов цвета для управляющей последовательности

Цвета изображения		**Цвета фона**	
30	Черный	40	Черный
31	Красный	41	Красный
32	Зеленый	42	Зеленый
33	Желтый	43	Желтый
34	Голубой	44	Голубой
35	Малиновый	45	Малиновый
36	Бирюзовый	46	Бирюзовый
37	Белый	47	Белый

Первая цифра для фона — 4, для символа — 3 , вторая цифра соответствует десятичному значению трех битов, младший из которых дает красный цвет, средний — зеленый, а старший — синий. (Сокращенно BGR — по буквам цветов)

Дополнительный атрибут текста

0	Отменить все атрибуты
1	**Повышенная яркость**
2	Пониженная яркость
4	Подчеркивание
7	Негативное изображение

В современных версиях, в частности на Linux, не работают ранее использовавшиеся атрибуты «Курсив», «Мерцание», «Скрытое изображение»

Пример управляющей последовательности цвета

Задание ярко-желтого символа на синем фоне, можно получить управляющей последовательностью

\033[1;33;44m

Например

printf("**\033[1;33;44m**Privet**\033[0m**Vsem")

выведет слово Privet желтыми буквами на синим фоне, а слово Vsem будет выводиться далее белыми буквами на черном фоне.

В Windows управляющие последовательности в настоящее время не используются!!!

Ввод в Windows данных , размещенных предварительно на экране

Основная функция

ReadConsoleOutputCharacter(HANDLE hConsOut,

STR* buffer, DWORD len, COORD dwReadCoord,

DWORD* actlen);