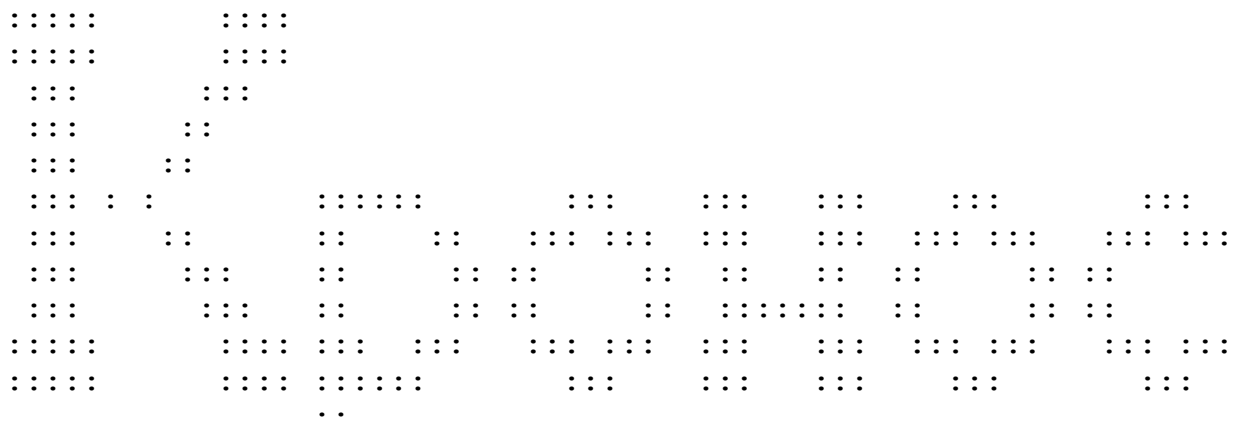
- Сибирское отделение АН СССР -

- Институт информатики -



РУКОВОДСТВО ПО ОС EXCELSIOR

Новосибирск - 90

Наш адрес:

630090, Новосибирск-90, проспект ак. Лаврентьева, 6, ВЦ СОАН СССР, к.503, тел. 35-50-67.
Для сотрудников ВЦ СОАН тел. 8-97.

Последнее изменение 7.08.91.

СОДЕРЖАНИЕ

СТРУКТУРА ЭТОЙ И ДРУГИХ КНИГ О КРОНОСЕ

Структура этого тома

Том, который вы держите в руках, состоит, собственно, из трех книг.

Первую книгу - "Кронос для начинающих" - мы адресуем читателям, не имеющим опыта работы на других машинах и в других системах. В этой книге они найдут самые первые, начальные сведения о Кроносе и программировании. Кроме того, сюда вошли определения некоторых понятий и обозначения, принятые в ОС Excelsior.

Вторая книга - "Кронос для всех" - является фактически руководством по операционной системе Excelsior, и рассчитана на программистов, знакомых с другими системами или с ОС Excelsior по первой книге.

Книга третья - "Кронос не для всех" - содержит информацию о внутреннем устройстве ОС Excelsior, системы программирования Модула-2 и о том, как все это ложится на Кронос-архитектуру. Эта информация предназначается разработчикам компиляторов, отладчиков, потенциальным администраторам системы, а также всем тем, кому это интересно.

В Приложении читатель найдет примеры программ на языке Модула-2. Мы постарались максимально проиллюстрировать с помощью этих примеров работу на Кроносе, поэтому Приложение будет полезно как начинающим, так и выросшим до уровня администратора системы программистам.

Том снабжен глоссарием, который позволяет быстро получить ссылки на разделы по ключевым словам.

Что еще читать о Кроносе

Помимо этой, основной, на наш взгляд, книги, имеются еще четыре. "Архитектура процессоров семейства КРОНОС" - содержит общие сведения об архитектуре и системе команд Кроноса, а также интерпретатор М-кода на языке Модула-2. Кроме того, в этом томе вы найдете фрагменты программ на Модуле-2, иллюстрирующие систему команд семейства.

Две следующие книги - справочного характера. В книге "Библиотеки ОС Excelsior" вы найдете общие сведения о библиотеках и назначении каждой из них. Собственно справочная часть включает тексты определяющих модулей всех библиотек с подробными комментариями об использовании каждой процедуры. Книга "Утилиты ОС Excelsior" является справочником по утилитам. Описания всех утилит следуют в алфавитном порядке и содержат всю необходимую информацию для того, чтобы даже неподготовленный программист мог воспользоваться утилитой:

порядок запуска, этимологию названия, действие, варианты использования и примеры.

Последняя книга носит специальный характер, поскольку адресована непосредственно администратору системы. Она так и называется: "Кронос для администратора". Это руководство поможет администратору выполнять все действия, понимаемые под выражением "поддержка системы", начиная с запуска тестовых программ в момент, когда Кронос только распакован и включен в сеть, и кончая написанием драйверов специальных внешних устройств, подключаемых по желанию пользователей. Поскольку поддержка системы - дело нешуточное, этот том оформлен в соответствии с нормативными требованиями Единой Системы Программной Документации.

Книга первая
ОС Excelsior ДЛЯ НАЧИНАЮЩИХ

Кронос, в древнегреческой мифологии титан, сын Урана (Неба) и Геи (Земли). Уран, боясь погибнуть от одного из своих детей, возвращал их снова в недра земли. Поэтому Гея, изнемогавшая от бремени, уговорила Кроноса, родившегося последним, оскопить Урана. Кронос стал верховным богом. Испугавшись предсказания матери, что он в свою очередь будет свергнут одним из сыновей, Кронос проглатывал детей, рожденных ему сестрой Реей, пока ей не удалось спрятать от Кроноса и вырастить втайне Зевса. Возмужав, Зевс заставил Кроноса изрыгнуть проглоченных им детей, составивших поколение олимпийских богов,а сам Кронос и другие титаны, побежденные Зевсом, были заключены в Тартар. По более позднему варианту мифа, Кронос впоследствии был переселен на "острова блаженных". Отсюда в представлении древних греков "царство Кроноса" соответствовало сказочному "золотому веку". Считаясь первоначально богом жатвы, Кронос изображался старцем с покрытой головой и серпом в руке.

Позднейшее представление о Кроносе, как о боге времени, возникло вследствие созвучия имени Кронос и греческого слова "хронос" - "время".

Из русских и советских
энциклопедических изданий

Теперь, когда законное любопытство читателя удовлетворено, можно начинать рассказ о Кроносе и об операционной системе Excelsior, под управлением которой работает Кронос.

Эта книжка похожа на слоеный пирог. Каждый может выбрать для себя наиболее подходящий слой. Опытный программист, работавший на разных машинах и в разных системах, скорее всего, выберет слой, бегло знакомящий читателя с соглашениями, принятыми в ОС Excelsior.

Для другого программиста, возможно, ОС Excelsior станет первой системой, которая научит его работать, а эта книжка - первым путеводителем в науку программирования (computer science).

Многие, работая на Кроносе, ощущают его как живое существо. Как хорошо, когда это существо тебя понимает, и как нелегко - понять его. Наша цель - научиться разговаривать с Кроносом.

Часть 1. ДИАЛОГ С КРОНОСОМ

Глава 1.1. Клавиатура и экран

Для общения с Кроносом необходимы две вещи: клавиатура (чтобы вы могли говорить) и экран (чтобы вы могли видеть его ответ).

1.1.1. Курсор

На экране почти всегда можно найти ярко светящийся прямоугольник. Это курсор. Он указывает ту позицию на экране, на которой вы сейчас "находитесь". Если теперь нажать на клавишу "A", на том месте, где был курсор, возникнет изображение символа "A", а сам курсор переместится на одну позицию вправо.

При нажатии некоторых клавиш никакие символы не появляются, а происходит что-нибудь другое - например, перемещается курсор. Это зависит от того, какую клавишу вы нажали - литерную или управляющую.

1.1.2. Литеры

Литерные клавиши - это те, при нажатии которых на экран выдается какой-нибудь печатный символ: ABcd @#$!* пЮЪЙ.

Литеры бывают прописные (ABCАБВ) и строчные (abcабв), кириллица (АБВгде) и латиница (ABCdef). ОС Excelsior все их различает и понимает. В системе используются две кодировки символов. Для латиницы кодировка ASCII-8 (American Standard for Information Interchange), для кириллицы - КОИ-8. Кодировку каждой клавиши можно узнать с помощью утилиты ascii (см. в томе "Утилиты ОС Excelsior").

1.1.3. Управляющие клавиши

Управляющие клавиши - клавиши, при нажатии которых терминал отрабатывает какие-либо функции.

Терминалы (и клавиатуры) бывают разные, и на всех работает одна и та же система. Везде в дальнейшем описании мы будем пользоваться некоторыми условными названиями кнопок. Каждой такой кнопке соответствует какое-либо действие. Где они расположены на вашей клавиатуре, вы можете узнать у администратора системы.

1.1.4. Контрольные символы

Этот раздел можно прочитать потом.

Кроме основных функциональных клавиш, используются еще и контрольные символы - символы, полученные одновременным нажатием клавиши CTRL и литеры. Заметим, что при наборе контрольного символа на клавиатуре не происходит его высвечивания на экране. Кроме того, не имеет значения, набирать контрольный символ с прописной буквой или со строчной. Можно набирать его с включенным регистром РУС - результат будет одинаков. В дальнейшем будем обозначать контрольные символы CTRL_<ЛИТЕРА>.

Некоторые контрольные символы имеют особую интерпретацию. Среди них:

CTRL_C - прекращение текущей задачи;
CTRL_X - откреплление текущей задачи;
CTRL_S - останавливает процесс вывода на терминал;
CTRL_Q - возобновляет вывод на терминал;
CTRL_Z - переключает режимы вставки/замены.

Такая интерпретация контрольных символов сохраняется при любой работе с терминалом - в частности, в редакторе текстов 'ex' (см. главу про редактор).

Глава 1.2. Вход в систему

> - Так, значит, тут разыгрывается Настоящая Шахматная Партия?! И целый мир - шахматная доска. Если, конечно, это настоящий мир. Как здорово! Как бы я хотела туда попасть! И... и стать пешкой... если позволят! Хотя БОЛЬШЕ ВСЕГО НА СВЕТЕ я хотела бы стать Королевой.
>
> Л.Кэрролл. Алиса в Зазеркалье

1.2.1. Приглашение к работе

Сядьте поудобней и посмотрите на экран. Если все работает, то вы увидите на экране приглашение. Например, такое:

```
17:55 usr
```

Не огорчайтесь, если приглашение выглядит совсем не так. Просто это не ваше приглашение. Чтобы приглашали именно вас, попросите администратора вашей машины зарегистрировать вас в системе и завести для вас рабочую директорию.

Как правило, вас регистрируют под каким-то (разумной длины) именем или иным милым вашему сердцу буквосочетанием. С этого момента система знает, что вы являетесь одним из пользователей, со всеми вытекающими отсюда последствиями. Вы имеете право войти в систему и получить доступ к определенной информации; в свою очередь, вы можете хранить свою информацию и определять степень ее доступности для остальных пользователей. Кроме того, вы можете создать себе свою, отличную от окружающей, обстановку и работать в ней. В частности, установить любое понравившееся вам приглашение.

1.2.2. Как войти в систему

> - Уважаемый Гладиолус, как жаль, что ты не можешь сказать ни слова.
> - Отчего же, могу, - вымолвил Гладиолус. - Могу, если рядом есть кто-нибудь, кто этого ЗАСЛУЖИВАЕТ.
>
> Л.Кэрролл. Алиса в Зазеркалье

После того, как машину включили и произошла загрузка системы, на экране появляется надпись

username:

Наберите на клавиатуре то имя, под которым вас зарегистрировали в системе. Затем найдите главную кнопку, которая называется ENTER, а может быть, CR (Carriage Return), или ВК (Возврат Каретки) - зависит от клавиатуры, на которой вы работаете. Нажмите ее. После этого на экране появится слово

password:

Это значит, что вы должны набрать ваш пароль. Если имя или пароль набраны неверно (а исправления при наборе запрещены), вам предложат попытаться набрать пароль еще пару раз. Если вы снова ошибетесь (или имя было набрано неверно), то после некоторой задержки опять появится надпись

username:

и все начнется сначала.

Когда, наконец, имя и пароль набраны правильно, на экране должно появиться ваше приглашение. Если это произошло, то все в порядке: можно начинать знакомство.

Глава 1.3. Командная строка

Диалог с ОС Excelsior ведется с помощью командных строк. Командная строка - набор символов, завершающийся нажатием клавиши ENTER.

Ведущие пробелы в командной строке игнорируются; следующие за командной строкой пробелы отсекаются. Количество пробелов между словами несущественно.

Командная строка должна содержать только одну команду.

Набранную командную строку можно редактировать.

1.3.1. Редактирование командной строки

Самая важная клавиша - ENTER (Carriage Return - Возврат Каретки). Именно она завершает набор каждой командной строки. Неважно, в какой позиции строки при нажатии ENTER находился курсор - будет введена вся командная строка от начала до конца.

Ввод командной строки с "обрубанием" хвоста (то есть всего того, что находится правее курсора) осуществляет клавиша LF (Line Feed - Перевод Строки - ПС). На клавиатуре рабочей станции это кнопка с букавами SysReq. Возможно, кому-то покажется более удобным пользоваться именно этой клавишей.

Дальше идут стрелки (ARROWS):

LEFT	- переход на символ влево;
RIGHT	- переход на символ вправо.

Клавиши HOME и END осуществляют переходы на начало и конец строки.

Можно возвратить содержимое предыдущих командных строк. В памяти хранятся одновременно 16 команд (15 предыдущих и одна текущая, то есть та, которая набирается в данный момент), которые можно перебирать, двигаясь назад и вперед с помощью стрелок UP и DOWN:

UP	- переход вверх на строку;
DOWN	- переход вниз на строку.

Другие полезные клавиши:

TAB	- табуляция (переход на 8 символов) вправо;
BACK TAB	- табуляция влево;
InsCh	- вставка символа с подвижкой хвоста строки вправо;
DelCh	- удаление символа с подвижкой конца строки влево;
BACKSPACE	- замена символа слева на пробел с переходом на него курсора.

1.3.2. Режимы вставки и замены

Можно вводить символы в так называемом режиме замены, когда слово, набираемое поверх другого слова, заменяет его (как на пишущей машинке).

Можно, наоборот, пользоваться режимом вставки. В этом случае при наборе поверх старого слова вновь набираемые символы пишутся не вместо него, а перед ним, отодвигая хвост строки все дальше вправо.

Заметим, что сами разработчики предпочитают второй режим. Для того, чтобы сменить режим редактирования, нажмите CTRL Z.

1.3.3. Переход в режим кириллицы

Для того, чтобы перейти с латиницы на кириллицу, нажмите последовательно левую, а затем правую кнопку SHIIFT (на клавиатуре она обозначена стрелкой вверх). Для перехода на латиницу эти кнопки нажимаются в обратной последовательности.

Часть 2. ФАЙЛОВАЯ СИСТЕМА ОС Excelsior

Глава 2.1. Корни и ветви дерева

Forest лес # Совокупность деревьев; удаление корневой вершины превращает дерево в лес.

А.Борковский
Англо-русский словарь
по программированию и информатике

Тексты и коды программ, статьи и эта книжка - одним словом, вся информация, с которой работает программист, хранится на магнитных носителях в файлах.

2.1.1. Файл

Файл - это набор информации, имеющий имя; он представляет собой последовательность байтов на носителе (байт = 8 бит; например, код ASCII-символа занимает в точности один байт).

2.1.2. Директория

Директория - специальный вид файла; содержит список имен других файлов и указание на их местоположение в файловой системе.

Файловая система ОС Excelsior имеет древовидную структуру. Корневой директорией называется файл-корень файлового дерева.

2.1.3. Носитель

Носитель - единица совместного хранения файлов (обычно диск или дисковый пакет), устанавливаемая на любое подходящее устройство.

Глава 2.2. Имя файла

- А для чего вообще нужны имена?
- Понятия не имею, - ответил Комар. - Дальше в лесу есть такое место, где ни у кого нет имени.

Л.Кэрролл. Алиса в Зазеркалье

У каждого файла есть имя, которое дается файлу при его создании. Имя файла - последовательность символов, за исключением символов " " (пробел), "/" (косая черта) и символа с кодировкой 0. Все остальные символы можно использовать, хотя использование некоторых символов приводит к неудобствам. Для начала рекомендуем пользоваться только печатными символами. Смысл этих ограничений станет ясен позже. Имя файла должно быть не длиннее 32 символов. Например:

my_file @@@ 2*2=4 и так далее.

2.2.1. Расширители имен файлов

Часто имена файла снабжаются расширителем. Расширитель имени файла - последовательность литер, отделяемая от имени символом "." (точка). Например: util.doc - расширитель - doc. Существуют также стандартные расширители для некоторых специальных файлов, принятые в ОС Excelsior.

.d - расширитель для файла, содержащего определяющий модуль (DEFINITION MODULE);

.m - расширитель для файла, содержащего реализующий модуль (IMPLEMENTATION MODULE);

.cod - расширитель для файла, содержащего код модуля;

.sym - расширитель для симфайла;

.ref - расширитель для реффайла.

Заметим, что директориям не принято давать имена с расширителями.

2.2.2. Полное имя файла

Местоположение каждого файла в файловом дереве определяется последовательностью директорий, которые требуется пройти системе до этого файла. Точкой отсчета при этом может быть как текущая директория, так и корневая директория. Это позволяет ввести понятие полного имени файла. Полное имя - имя файла с указанием маршрута к этому файлу от корня файлового дерева:

<полное имя файла> ::= ["/"]{<директория>"/"}<имя файла>

Например:

```
/doc/util1.doc
/users/andy/ccScan.m
/bin/StdIO.cod
```

Кроме того, местоположение файла можно определить, указав маршрут до него от текущей директории. Итак, определим маршрут:

<маршрут> ::= ["/" | "."] { ".."/ } { <директория>"/"}

Замечание. Возможно, читатель удивится, увидев слово "маршрут" вместо привычного слова "путь". Leopold предложил использовать именно этот термин во избежание путаницы (что не удалось переводчикам документации по ОС UNIX). А что такое путь в ОС Excelsior - читайте дальше.

2.2.3. Соглашения об именах

В ОС Excelsior приняты следующие соглашения для имен файлов:

- имена директорий в полном имени разделяются символом "/" (косая черта);
- именем корневой директории является строка "/";
- имя текущей директории обозначается символом "." (точка);
- имя родительской директории обозначается символом ".." (две точки), который можно повторить несколько раз, двигаясь при этом к корню файлового дерева.

Глава 2.3. Прогулки по файловому дереву

> Сунул я руку в карман, вытащил горсть гаек. Показал их Кириллу на ладони и говорю:
> - Мальчика с пальчик помнишь? Проходили в школе? Так вот сейчас будет все наоборот. Смотри! - и бросил первую гаечку. Недалеко бросил, как положено. Метров на десять. Гаечка прошла нормально.
> - Видел?
> - Ну? - говорит.
> - Не "ну", а видел, я спрашиваю?
> - Видел.
> - Теперь самым малым веди "галошу" к этой гаечке и в двух метрах до нее не доходя остановись, понял?
>
> А.Стругацкий, Б.Стругацкий
> Пикник на обочине

Для перехода на другую директорию служит команда 'cd' (cHANGE dIRECTORY). Команде необходимо указать маршрут до новой директории.

cd ["/"|"."] {"../"} {<директория>"/"} <имя_директории>

Например:

cd
- показывает маршрут до текущей директории от корня файлового дерева;

cd ..
- переход на директорию, содержащую текущую;

cd /
- переход на корневую директорию;

cd ../../users/my
- переход на директорию my, расположенную на директории users.

Заметим, что 'cd' является командой пользовательской оболочки shell, о которой можно почерпнуть подробные сведения в разделе "SHELL: ПОЛЬЗОВАТЕЛЬСКАЯ ОБОЛОЧКА ОС".

Глава 2.4. Создание и редактирование файла

2.4.1. Запуск редактора

Чтобы создать новый текст, требуется создать новый файл. Новые файлы создаются с помощью редактора текстов 'ex'. Редактор запускается так:

ex <имя_файла> [-ключ]

Ключ у редактора только один - 'w'. Он означает "запретить запись в файл", то есть файл открывается только для чтения (о ключах см. раздел 8.1).

Если редактор не находит файла с таким именем, то сообщает об этом и после вашего подтверждения нажатием клавиши ENTER создает новый файл. Если вы передумали, нажмите CTRL C, и файл не будет создан.

2.4.2. Как выйти из редактора

Если вы закончили набирать или исправлять текст, то для того, чтобы выйти из режима редактирования и сохранить все ваши исправления, нажмите одновременно CTRL и e (^E).

Если вы хотите, чтобы прежний текст остался неисправленным, или вам не требуется сохранить набранный файл, выходите из редактора без записи, набрав ^C. После этого у вас запросят подтверждение, и после Y или y (yes) произойдет выход из редактора. Если вы вдруг передумали, наберите N или n, чтобы остаться в редакторе.

2.4.2.1. Полезный совет

> - Мне никогда, никогда не забыть, - заявил Король, - это кошмарное мгновенье!
>
> - Ты непременно о нем забудешь, - сказала Королева, - если не запишешь его в записную книжку.
>
> Л.Кэрролл. Алиса в Зазеркалье

Во избежание потерь при неожиданном сбое полезно время от времени записывать набираемый текст. Это можно сделать, не выходя из редактора. Нажмите последовательно кнопки F10 и w - и ваш текст спасен на диск.

2.4.3. Экран

Вы находитесь в редакторе. Перед вами чистая страница -

экран, который вы можете заполнять по своему усмотрению.

В нашей системе экранное редактирование построено по принципу "что на витрине, то и в магазине", иначе говоря, программист имеет ровно то, что видит на экране. Для примера: редактор не делает различия между пробелами, набранными клавишей SPACE, и пробелами, получившимися с помощью TAB. Как на экране, так и в файле это будут просто пробелы. То же относится и ко всяческим контрольным символам - если символ не изображен на экране, программист может быть уверен, что его нет и в редактируемом файле.

Самая нижняя строка экрана - информационная. Она выглядит приблизительно так:

```
<<<SCREEN 10:14 INS off  BELL on  3:59 p.m.  0071 FILE x.m >>>
```

Первые два числа в информационной строке - положение курсора на экране (номер строки, считая сверху, и номер знакоместа, считая слева, разумеется, с нуля). Далее идет сообщение, включен ли режим вставки (INS on - включен, INS off - выключен). Затем - сообщение, включен ли звуковой сигнал, затем - текущее время (post meridiam или ante meridiam, в зависимости от времени суток), далее - номер строки в файле, на которой находится курсор, и, наконец, имя редактируемого файла.

В процессе редактирования могут выполняться разные действия - например, запись редактируемого файла на диск, или форматирование текста, и всякий раз сообщение о выполняемом действии появляется в информационной строке.

2.4.4. Клавиатура

Литерные клавиши в текстовом редакторе работают так же, как и в однострочном командном редакторе.

Редактору приходится работать с самыми разными типами терминалов (Фрящик, Видеотон, Mera, Labtam, Elorg и т.д.), с разными клавиатурами и разными названиями клавиш на них. Поэтому, как мы уже условились раньше, рассказ будет вестись в терминах функциональных кнопок.

2.4.4.1. Драгоценные кнопочки

Кроме обычных управляющих кнопок, в редакторе выделены три кнопки с условными названиями GOLD, SILVER, BRONZE. Эти кнопки меняют смысл кнопки, которая нажата после них. На клавиатуре рабочей станции это кнопки F1,F10 и F2 соответственно. С кнопкой SILVER вы уже частично знакомы. (F10 'w' - запись вашего текста на диск без выхода из редактора).

Кнопка GOLD (F1) предназначена для "усиления" действия следущей кнопки, то есть она является точкой опоры для переворачивания мира. Например:

```
F1 "стрелка вправо"   - переход на конец строки;
F1 "стрелка влево"    - переход на начало строки;
F1 PageUp             - переход на начало файла;
F1 PageDn             - переход на конец файла.
```

Кроме этого, кнопка F1 поможет вам быстро набирать тексты программы и выполнять поиск и замену кусков текста (когда вы доберетесь до соответствующих разделов книги "ОС Excelsior для всех").

Кнопка F2 предназначена в основном для ваших собственных расширений возможностей редактора.

Подробности обо всем этом можно узнать из документации по редактору или с помощью утилиты "help". А пока перечислим действия всех управляющих кнопок вашей клавиатуры.

2.4.4.2. Перемещения по тексту

переход вверх на строку	UP
переход вниз на строку	DOWN
переход на символ влево	LEFT
переход на символ вправо	RIGHT
табуляция вправо	TAB
табуляция влево	BACK TAB
переход на 16 строк вверх	PageUp
переход на 16 строк вниз	PageDw
переход на начало текущей строки	Enter
со стиранием хвоста текущей строки	SysReq

2.4.4.3. Вставка и удаление

вставка символа с подвижкой хвоста строки вправо	InsCh
удаление символа с подвижкой хвоста строки влево	DelCh
замена символа слева на пробел с переходом на него курсора	BACKSPACE
удаление хвоста строки	F8
удаление хвоста строки с приклеиванием хвоста следующей	F4
вставка строки с отламыванием хвоста на следующую строку	F3
отмена последнего DelLn	F1 F2

2.4.4.4. Операции над строками

дублирование хвоста строки	F7
обмен хвостов вверх от курсора	F5
обмен хвостов вниз от курсора	F6
склеивание строк	F1 F4
разрыв строки	F1 F3
дублирование хвоста поверх следующей строки	F1 F7

2.4.4.5. Операции над единицами текста

переход к началу слова	F1 UP (F2 LEFT)
переход к концу слова	F1 DOWN (F2 RIGHT)
удаление слова	F1 DelCh
переход к началу строки	F1 LEFT
а также	Home
переход за конец строки	F1 RIGHT
а также	End
переход на конец текста	F1 PageDw
переход к началу текста	F1 PageUp
форматирование текущего абзаца	F10 1
центрирование заголовка	F10 2
возврат к первоначальному содержимому строки	BRONZE F1

2.4.4.6. Выход в другие режимы редактора

командный режим	F1 F1
установка параметров (SETUP-монитор)	F1 F10
вход в shell редактора (выход - ESC)	F10 PgDn

2.4.5. Возможности редактора

После того, как вы освоитесь с клавиатурой, переходите к к ознакомлению с возможностями редактора.

Редактор располагает следующими возможностями:

- позиционирование в файле;
- поиск в файле;
- контекстная замена;
- работа с файлами (переименование и пр.);
- работа с областью текста;
- установка макросов;
- установка параметров редактирования;
- форматирование абзаца и области текста;
- запуск задач из shell редактора.

Подробное описание всего этого дается в книге "Утилиты ОС Excelsior", глава 'ex'.

Редактор - достаточно сложный инструмент. Чтобы овладеть им в совершенстве, нужно не только понять, как он работает, изучить его функции, но и привыкнуть к нему настолько, чтобы пользоваться его возможностями не раздумывая, автоматически. На это потребуется некоторое время, как требуется время, чтобы научиться игре на музыкальном инструменте.

Часть 3. ЗАДАЧИ ДЛЯ КРОНОСА

Глава 3.1. Запуск задач

Для запуска любой задачи в ОС Excelsior достаточно просто набрать на экране ее имя (с соответствующими параметрами, если они есть). Именем задачи служит имя кодового файла (файла с расширителем '.cod') программного модуля задачи. Вот как будет выглядеть запуск задачи "my_task", находящейся на вашей рабочей директории, скажем, "wrk":

```
/users/vasya/wrk/my_task
```

Приходится набирать довольно длинную строчку, чтобы указать системе, с какой именно директории она должна взять коды задачи. Но полное имя указывать не обязательно, если у вас установлен путь поиска кодофайлов.

3.1.1. Путь поиска кодофайлов

Этот пункт можно пропустить при первом чтении.

Умея определять местоположение файла, можно установить очередность маршрутов при поиске файлов для запуска программ. Это позволит вместо полного имени файла указывать лишь его имя, даже если файл лежит на удаленной директории или на другом носителе.

Путь, на которых будут разыскиваться коды запускаемой задачи, устанавливается так:

```
BIN='"'[.] [..] {{'/'<имя_директории>}' '} '"'
```

Теперь задана последовательность полных имен директорий, на которых надо искать коды загружаемой программы, например:

```
BIN=" . .. /ii/моя_дир"
```

- файл будет разыскиваться сначала на текущей директории, затем - на директории, содержащей текущую, затем на директории '/ii/моя_дир'.

Можно узнать ранее установленный путь, набрав команду

```
set
```

Кроме всего прочего, в строчке "BIN" команда set показывает установленный путь.

3.1.2. Как запустить задачу

Итак, для запуска задачи достаточно указать имя кодофайла ее головного модуля:

```
my_task
```

и, если таковой будет найден на какой-нибудь из указанных пуем директорий, задача начнет исполняться.

Заметим, что если на текущей директории имеются файлы name.cod (кодофайл) и name.@ (командный файл с тем же именем, главу о командных файлах), то команда

```
name
```

приведет к исполнению командного файла name.@, а не задачи с кодофайлом name.cod. Как обойти это препятствие? Очень просто: запуская задачу, укажите имя ее кодофайла с расширителем:

```
name.cod
```

3.1.3. Как остановить задачу

Исполнение текущей задачи можно прервать нажатием CTRL_C. Разумеется, есть задачи, которые так просто не прервешь (например, пользовательская оболочка 'sh', которую запускает администратор системы). Случаям, когда CTRL_C "не действует", посвящена специальная глава в книге "ОС Excelsior для всех".

3.1.4. Стандартный ввод-вывод

Запуская какую-либо задачу (утилиту, пользовательскую программу), вы ожидаете вывода ее результатов. Некоторые задачи требуют ввода дополнительной информации. Многие задачи пользуются стандартным вводом-выводом, который по умолчанию открыт на терминал, с которого была запущена задача.

Вывод может быть направлен в файл. Для этого при запуске требуется указать имя файла, куда будут записаны результаты, предварив его символом '>':

```
<имя_задачи> '>'<файл_результатов>
```

- результат записывается в указанный файл.

Если в файле лежала какая-то информация, она будет удалена. Чтобы этого не произошло, можно дописать результаты в файл с помощью значка '>>':

<имя_задачи> '>>'<файл_результатов>

- результат дописывается в указанный файл.

Ввод тоже может осуществляться из файла с помощью '<':

<имя_задачи> '<'<файл_ввода>

3.1.5. Параллельное исполнение задач

В ОС Excelsior есть возможность запустить несколько (число зависит от потребностей задачи в ресурсах) задач одновременно. С точки зрения операционной системы они совершенно равноправны с задачей, запускаемой "явно". После завершения "фоновая" задача освобождает выделенные ей ресурсы. Для обозначения таких задач существуют разные термины - параллельная, открепленная, независимая и т.д.. Мы будем использовать, пожалуй, последний.

3.1.5.1. Запуск независимой задачи

Запуск выглядит так:

<имя_задачи> &

После этого вы можете запускать следующую задачу в "явном" виде, ее исполнение не будет мешать исполнению ранее запущенной задачи.

Вам следует лишь позаботиться о выводе результатов, если результатом является, например, печать на экран, чтобы вывод не мешал вашей работе с "явной" задачей.

Открепить уже исполняющуюся задачу можно нажатием CTRL_X.

3.1.5.2. Перечень задач

Можно ознакомиться с перечнем всех задач, исполняемых компьютером в данный момент. Это делается с помощью команды shell'a ps:

ps

На экран будет выдано приблизительно следующее:

```
0066    run  ex            shell.doc
0043    run  shell
```

В первой колонке указан номер задачи, в третьей - ее имя. Полное описание команды ps дается в разделе, посвященном пользовательской оболочке ОС, в книге "Кронос для всех".

3.1.5.2. Прекращение исполнения независимой задачи

Прекратить исполнение можно командой shell'а kill :

kill [<номер задачи>]

, например:

kill 1031

или

kill

Если номер задачи не указывать, то вам предложат на удаление все ваши (запущенные на текущем терминале) задачи.

Глава 3.2. Утилиты

Выше нам уже пришлось употребить слово "утилита", не поясняя его значения. Утилита - это программа, предназначенная для выполнения какого-нибудь действия (или последовательности действий). Точным переводом этого слова было бы "пользилка". Утилиты могут иметь самые разные назначения: бывают утилиты для работы с файлами, для работы с устройствами, и редактор, с помощью которого редактируются тексты, и Модула-компилятор - тоже утилиты. Попросту говоря, утилиты - это приспособления, с помощью которых программист имеет доступ к операционной и файловой системам и т.д..

3.2.1. Ключи

Обычно утилита выполняет не одно, а несколько сходных между собой действий. Выполняемое действие зависит от ключа, с которым запущены утилита. Так, например, утилита 'ls', запущенная без ключей, выдает на экран имена файлов на указанной путем директории, кроме скрытых файлов (например, с расширителями .sym, .ref и .cod).

Будучи запущена с ключом -a

```
ls -a
```

ls выдает имена ВСЕХ файлов, в том числе скрытых.

```
ls -l
```

выдает имена файлов в "длинном формате", то есть с указанием обширной информации о каждом файле, например:

```
--rwxr-x---  1        968  Jan 18  13:08   a.m
```

Это значит, что на текущей директории находится только один файл a.m с указанными битами защиты, привязанный к одному имени, длиной 968 байт, последнее изменение внесено 18 января в 13.08.
И так далее.

Кроме символьных, существуют еще числовые ключи. С их помощью утилите можно передавать числовые параметры. Пример числового ключа:

```
where *.cod a=161288
```

- утилита находит все кодофайлы, запись в которые производилась после 16.12.1988.

3.2.2. Запуск утилиты

Таким образом, запуск утилиты выглядит так:

```
запуск ::=
<имя_утилиты> {<параметр>} {<числовой_ключ>} ['-'/'+'{<ключ>}]

<параметр>      ::= <строка>
<числовой_ключ> ::= <строка>'='<число>
<ключ>          ::= <строка>
```

Заметим, что все параметры и ключи утилит могут располагаться в командной строке в произвольном порядке.

3.2.3. Подсказка

Набор утилит относительно стабилен, но может пополняться за счет ваших собственных полезных программ, если они будут соответствовать стандарту, принятому в ОС Excelsior.

В частности, этот стандарт требует, чтобы каждая утилита имела ключ 'h', по которому она выдает краткую подсказку о себе самой (назначение и ключи).

Подробнее об этом и об утилитах вообще рассказывается в томе "Утилиты ОС Excelsior", который здесь тоже упоминался.

Помимо того, что каждая утилита выдает краткую информацию о себе по ключу 'h', существует еще утилита, ведающая службой подсказки в ОС Excelsior. Эта утилита оказывает на первых порах большую помощь, она так и называется - help. Запустив help, вы попадаете в меню, состоящее из названий всех утилит системы (а также команд shell - пользовательской оболочки). О каждой утилите вы можете получить полную информацию - от порядка запуска и этимологии названия до ключей и примеров ее употребления. О том, как пользоваться help, подскажет она сама.

Глава 3.3. Командный файл

Несколько командных строк могут быть объединены в командный файл, который создается с помощью редактора и при запуске выполняется, как если бы его строки были набраны последовательно с терминала. Имя командного файла имеет расширитель @:

<имя_файла>.@

3.3.1. Запуск командного файла

Запуск командного файла осуществляется набором имени файла:

<имя_файла>

Если на одной директории находятся кодофайл и командный файл с одним и тем же именем, то при наборе имени файла запустится кодофайл. Избежать путаницы можно, набрав имя файла с расширителем:

my_file.@

- будет исполняться командный файл;

my_file.cod

- исполнится кодофайл.

3.3.2. Прекращение командных файлов

Исполнение командного файла прекращается нажатием CTRL_C. При этом прекращаются все запущенные в нем задачи.

3.3.3. Соглашения для командных файлов

Для командных файлов приняты следующие соглашения:

1) строка комментария предваряется символом % ;

2) параметры, передаваемые командному файлу, обозначаются $1, $2, ..., $9. Например, если командный файл user.@ выглядит так:

```
cd /users
cd $1
```

, то $1 при исполнении командного файла заменяется на переданный ему параметр:

```
     user ВАСЯ
```

, что равносильно последовательности команд

```
     cd /users
     cd ВАСЯ
```

3.3.4. Пример командного файла

Приведем пример командного файла:

```
% Командный файл архивирования

% Смонтировать диск fd0 на директорию /mnt:
mou /mnt /dev/fd0

% Скопировать все изменившиеся и недостающие файлы на всех
% поддиректориях текущей директории на архивную директорию
% с сохранением имен файлов и созданием недостающих
% директорий

cp .//$1 /mnt/$@/$1 -qdb

% Демонтировать диск с директории /mnt:
mou /mnt -r

% Конец архивирования
```

3.3.5. Командный файла пользователя

На вашей приватной директории хранится командный файл profile.@. Такой файл предназначается для перехода на вашу рабочую директорию, установку удобных для вас параметров командной строки. Выглядит он приблизительно так:

```
% Установить путь поиска кодофайлов:
BIN=". /usr/flm/bin /bin /usr/bin"

% Установить путь поиска  командных файлов:
ETC=". /usr/flm/bin /etc /usr/etc"

% Установить приглашение:
PROMPT="%t %/> "

% Перейти на директорию wrk:
cd wrk

%                   Мы гостям хорошим рады,
%                   Смело в дом входите,
%                   Вытирайте ноги, гады,
%                   Чистоту блюдите.
```

Запуск командного файла пользователя осуществляется системой после входа в систему.

Если вы меняли обстановку работы, и вам после этого захотелось домой, в свою родную обстановку, наберите команду, которая так и называется: home - домой, указав, где именно ваш дом:

```
home /usr/ВАСЯ
```

Если на указанной вами директории обнаружится ваш личный "profile", он будет исполнен, и вы окажетесь у себя на директории в привычной обстановке.

Часть 4. СОЗДАНИЕ ПРОГРАММ

Программирование - это процесс исправления старых ошибок и насаждения новых.

А.Борковский
Из частной беседы

Глава 4.1. О языке Модула-2

Автором языка Модула-2 является Никлаус Вирт, профессор Высшей Технической Школы (Цюрих), который реализовал его на своей машине Лилит. Унаследовав лучшие черты языка Паскаль, Модула-2 имеет ряд отличительных особенностей, которые делают ее незаменимой при разработке программного обеспечения. Прежде всего это, конечно, модульность (и возможность разбиения модуля на определяющую и реализующую части), развитые структуры данных и управления, строгий контроль типов, наличие процедурных типов, что позволяет динамически параметризовать процедуры внешними действиями, а также наличие средств программирования низкого уровня, позволяющих ослабить жесткий контроль типов и отображать структурные данные на память.

Знание Модулы-2 на уровне языковых конструкций еще не дает возможности существенно использовать ее преимущества, поэтому, прежде чем писать тексты программ, тщательно ознакомьтесь с описанием языка. Лучше всего это сделать по книжке "Программирование на языке Модула-2" Н.Вирта в переводе В.А.Серебрякова и В.М.Ходукина, выпущенной в серии "Математическое обеспечение ЭВМ" издательством "Мир" в 1987 г.

Все дальнейшее изложение будем вести в предположении, что вы знакомы с языком.

Глава 4.2. Как пользоваться библиотеками

4.2.1. Модульность и библиотеки

Коль скоро вы собираетесь работать на Кроносе и писать свои программы на Модуле-2, нет смысла убеждать вас в пользе модульности. И все же скажем несколько слов о том, как отразилось это свойство на системе программирования, а именно ее части, именуемой библиотеками стандартных процедур.

Возможно, модульность не имела бы такого большого значения для системы программирования, если бы на Кроносе не было динамической загрузки кодов модулей, из которых состоит программа. При динамической загрузке, в отличие от статической, отсутствует этап сборки (линкования), то есть не требуется создание на диске образа задачи, в котором содержатся коды всех модулей, составляющих эту задачу. Поэтому оказалось удобным некоторые наиболее часто употребляемые процедуры объединить в модули, при инициализации которых "ничего не происходит", то есть не исполняются никакие действия. Такие модули служат только для импорта из них процедур и называются библиотеками. Коды библиотечных модулей хранятся на диске в единственном экземпляре и, следовательно, многочисленные копии процедур не загромождают дисковое пространство. Кроме того, этими процедурами можно пользоваться, как строительными блоками, не заботясь о внутреннем устройстве блоков. А вновь создаваемое программное обеспечение автоматически пополняет арсенал строительных средств.

Итак, библиотеки

а) экономят дисковое пространство;

б) ускоряют написание программы в сотни раз;

в) позволяют не изобретать велосипед;

г) дают возможность писать свои программы чужими руками.

4.2.2. Назначение и название

За время существования ОС Excelsior написано, упразднено и переписано заново множество библиотек. Чтобы овладеть всем этим богатством, необходимо знать его. Лучший способ - изучение определяющих модулей библиотек с подробными комментариями. Все это можно найти в справочном томе "Библиотеки ОС Excelsior". Там же приводится список всех библиотечных модулей с указанием их назначения.

Для начала рекомендуем ознакомиться с библиотеками StdIO, Strings, Image, Args, Streams. В этих библиотеках содержатся все процедуры, которые могут понадобиться для написания простых программ.

4.2.3. StdIO - стандартный вывод

Наиболее необходимая библиотека (во всяком случае, для начинающих) - библиотека стандартного вывода StdIO. С помощью ее процедуры print можно вывести информацию в указанном формате в так называемый файл стандартного вывода. По умолчанию это экран, но вывод может быть перенаправлен стандартным способом (см. в главе 7 раздел 3).

Остановимся на этой процедуре подробно.

```
PROCEDURE print(format: ARRAY OF CHAR; SEQ args: WORD);
```

Первый аргумент этой процедуры - формат, в котором требуется вывести перечисленную далее последовательность аргументов. Конструкция SEQ (о реализованных расширениях языка Модула-2 см. выше п.10.3) означает перечисление произвольного числа аргументов (возможно, ни одного).

Например,

```
print("Hello, hacker!")
```

напечатает строчку

```
Hello, hacker!
```

Последовательность аргументов у процедуры опущена.

-format- задается строкой ASCII-8 символов (и тогда заключается, естественно, в кавычки " или '), или строковой переменной (и тогда байт 0c означает конец строки).

Все символы строки выдаются в стандартный вывод без изменения, за исключением некоторых символов, которые процедура print интерпретирует специальным образом.

1. Символ \ интерпретируется следующим образом:

   \n отрабатывается как CR+LF.
   \r отрабатывается как CR.
   \l отрабатывается как LF.

   Если в строке требуется символ '\', предварите его '\': '\\' преобразуется в просто '\'.

   ВНИМАНИЕ! \n, \r, \l обрабатываются ТОЛЬКО в строке -format-, но не в строках-аргументах!!!

2. Символ % и следующие за ним интерпретируются процедурой как формат, в котором требуется вывести соответствующий аргумент процедуры из последовательности аргументов.

   Если требуется символ '%', предварите его '%': '%%' преобразуется в просто '%'.

   Таким образом, формат начинается символом %, далее идут модификаторы, а завершается формат базой. Формат может встречаться в строке -format- несколько раз (возможно, ни разу).

```
формат          ::= %{ модификатор }база.
```

```
модификатор     ::= ( space | "$" | "-" | "+" | "#" | ширина
                    | точность } .
ширина          ::= цифра { цифра } | "*".
                  - задает общую(!) ширину поля (с учетом
                    возможных символов основания, знака и
                    т.п.); если указанной ширины
                    недостаточно для представления
                    выводимого значения, происходит
                    автоматическое расширение поля до
                    минимально необходимого.
точность        ::= "." цифра { цифра } | ".*" .
                  - задает число значащих цифр после(!)
                    запятой в форматах 'f','e','g' или число
                    символов строки в формате 's';
```

Замечание. Если вместо спецификации ширины и/или точности указаны '*', то значения ширины и точности нужно указать в соответствующих аргументах, при этом соблюдается следующий порядок: сначала аргумент-строка, которую требуется вывести, а затем аргументы-модификаторы.

Замечание. Значения точности и ширины должны быть из диапазона [0..255]; в противном случае они принимаются равными значению по умолчанию.

```
#               - показывает число с указанием основания
                  (например: image(s,"%$10#h",12abcdefH)
                  эквивалентно image(s,"012ABCDEFh") );
-               - число пишется слева в поле установленной
                  ширины;
+               - показывает число со знаком, независимо
                  от знака числа;
$   (zero)      - дополнить до нужного количества разрядов
                  ведущими нулями;
space           - выставляется знак, если число
                  отрицательное, иначе пробел;

база            ::= ("d"|"h"|"x"|"b"|"o"|"s"|"c"|"{}"|"i"
                     |"f"|"e"|"g").
d   (Decimal)       - десятичное;
h,H (Hexidecimal)   - шестнадцатеричное
                      (h,x -- "A".."F" прописные);
x,X (Hexidecimal)   - эквивалентно 'h'
                      (H,X -- "a".."f"  строчные);
b,B (Octal)         - восьмеричное;
o,O (Octal)         - эквивалентно 'b';
s,S (String)        - строка;
c,C (Char  )        - одиночный символ;
{}  (set)           - битсет;
i,I (bIn)           - двоичное;
f,F (Float)         - вещественное число в формате:
                        [+|-]dddddddd.ddddd
                          |__  n1  __|_ t _|
                    Число цифр до запятой n1 - минимально
```

необходимое для представления числа. Число цифр после запятой t - задается точностью (по умолчанию 6).

e,E (Exponent) - вещественное число в формате:
[+|-]d.dddddE(+|-)dd или
[+|-]D.DDDDDDe(+|-)DD
Число цифр до запятой 1.
Число цифр после запятой - t - задается точностью (по умолчанию 6).
(формат 'e' - 'E' в результате;
формат 'E' - 'e' в результате);

g,G (General) - вещественное число в формате: если FLOAT(TRUNC(число))=число, то в формате dddddd; иначе в формате 'f' или 'e' - какой короче.

ЗАМЕЧАНИЕ.
Модификаторы '$' и '-', естественно, не совместимы.
Модификаторы '+' и space, естественно, не совместимы.
Модификатор точность может быть использован только с базами 's','f','e','g'.
Модификаторы '+' и space могут быть использованы только с базами 'i','f','e','g'.
Модификаторы '$', '+' и space НЕ могут быть использованы с базами 's','{}','c'.

11.3.1. Примеры употребления процедуры print

Изложение примеров будем вести в следующей форме:

Текст программы	Будет выведено

1) Примеры без аргументов

Текст программы	Будет выведено
`print("Hello, hacker!\n");`	Hello, hacker!
`print("Hello, "0c"hacker!");`	Hello,
`print("Hello,\nhacker!");`	Hello, hacker!
`print("Hello,\lhacker!");`	Hello, hacker!

```
VAR h: ARRAY [0..79] OF CHAR;  |
  h:= "Hello!"                 |
  print(h);                    |  Hello!
                               |
                               |
```

2) Примеры с аргументами

```
VAR n1,n2                      |
    : ARRAY [0..79] OF CHAR;   |
BEGIN                          |
  n1:="Вася!"                  |
  n2:="Петя!"                  |
  print("Hello, %s\n",n1);     |  Hello, Вася!
  print("Hello, %s\n",n2);     |  Hello, Петя!
                               |
```

3) С использованием модификатора "ширина"

```
                               |
VAR str: ARRAY [0..79] OF CHAR;|
BEGIN                          |
  str:="Вася!"                 |
  print("Hello, %10s\n",str);  |  Hello,      Вася!
                               |
                               |
VAR str: ARRAY [0..79] OF CHAR;|
    i : INTEGER;               |
BEGIN                          |
  str:="Вася!"                 |
  i:= 10;                      |
  print("Hello, %*s\n",str,i); |  Hello,      Вася!
                               |
                               |
```

Глава 4.3. Компиляция

Наконец текст вашей программы готов. Чтобы Кронос исполнил программу, нужно представить ее в виде последовательности машинных команд (каких именно - см. в книге "Архитектура семейства процессоров КРОНОС" описание системы команд). Функцию перевода текста программы на языке Модула-2 в последовательность команд осуществляет Модула-2 компилятор.

Кроме этой основной задачи, компилятор выполняет еще одну важную функцию - он проверяет правильность текста программы, сообщая о синтаксических и семантических ошибках (какие именно ошибки "отлавливает" компилятор, видно из п.12.3).

В этой главе мы изложим только начальные сведения о компиляции. Более подробно мы остановимся на Модула-компиляторе в книге "ОС Excelsior для всех".

4.3.1. Запуск компилятора

Запуск компилятора осуществляется с помощью утилиты mx, либо с помощью компилирующего редактора ex, который в дальнейшем мы будем называть по аналогии с TURBO-PASCAL фирмы Borland турбо-компилятором, или просто турбиной. Он отличается от текстового редактора возможностью откомпилировать программу, текст которой вы редактируете в данный момент, не записывая его на диск и не покидая редактора. В этом случае компиляция запускается последовательным нажатием клавиш F10 и m.

Модула-компилятору передается текст программы, оформленный в соответствии с описанием языка. В системе принят стандартный расширитель для имени файла, содержащего определяющий модуль - .d ; реализующий или программный модуль - .m . Если расширитель отсутствует, компилятор добавляет расширитель .m : program.m.

Сначала компилируют определяющий модуль, если он есть, затем - реализующий и только потом программный.

Если программа состоит из нескольких модулей, то компилируют сначала все определяющие модули, а затем реализующие.

4.3.2. Что получается в результате компиляции

В результате компиляции определяющего модуля (DEFINITION MODULE) получается символьный файл (симфайл) и файл ссылок (реффайл); реализующего модуля (IMPLEMENTATION MODULE) - реффайл и кодовый файл (кодофайл); программного модуля (MODULE) - реффайл и кодофайл.

В симфайле содержится информация для компилятора; в реффайле - для отладочных утилит (например, для утилиты hi (history), выдающей историю последнего неудавшегося процесса.

Кодофайл содержит собственно исполняемый код программы.

Имена симфайлов, реффайлов и кодофайлов получаются из имени модуля (не из имени файла) добавлением расширителей '.sym', '.ref' и '.cod' соответственно.

Обратите внимание, что обнаружить симфайлы, реффайлы и кодофайлы на директории с помощью утилиты ls можно лишь запустив ее с ключом -a (all).

4.3.3. Сообщения компилятора

Компилятор выдает на русском языке сообщение о характере ошибки, приводится номер строки и сама строка, в которой допущена ошибка, а также помечается предположительно место в строке, где надо искать ошибку (в случае турбо-компилятора происходит последовательное позиционирование курсора на место ошибки), например:

```
Ожидался символ ';'
  104:   PROCEDURE Next(i,j: INTEGER$, A: ARRAY OF CHAR);
Число ошибок: 1
```

Если в процессе компиляции не обнаружено ошибок, на экран выдается сообщение:

```
lines 247  time 09cp + 11io   help.cod  326 words
```

, что означает, что был скомпилирован модуль размером 247 строк, время компиляции 9 секунд, время записи - 11 секунд, кодофайл записан в файл help.cod и объем кода 326 слов.

В результате компиляции на текущей директории появятся файлы help.ref и help.cod.

4.3.3.1. Для турбо-компилятора

Если вы запустили компиляцию из турбины, компилятор сообщит количество ошибок и спозиционируется на первой встреченной ошибке, указав ее характер. После ее исправления перейти к следующей ошибке можно последовательным нажатием клавиш F10 и +.

После того, как все ошибки исправлены и компиляция прошла успешно, не забудьте записать текст программы (F10 w).

Книга вторая

ОС Excelsior ДЛЯ ВСЕХ

ПЕРВОЕ ПРЕДИСЛОВИЕ СОАВТОРА

> Разумеется, книга эта не могла бы появиться на свет без поддержки и помощи некоторых людей. Говорю я об этом для того, чтобы ответственность за нее разделить на всех поровну.
>
> Д.Даррелл. Моя семья и другие звери

Известно, что авторство отличается от соавторства приблизительно тем же, чем пение от сопения. Тем не менее мне пришлось взять на себя неблагодарную миссию быть соавтором сразу целой группы авторов - разработчиков software для процессоров Кронос. Никто не может лучше рассказать о своем детище, чем его родитель. К сожалению, рядовому читателю редко бывает нужно "лучше", он хочет - понятнее. Насколько мне удалась скромная роль переводчика - судить читателю.

Говорю все это прежде всего для того, чтобы снять с себя всякую ответственность за написанное здесь (или хотя бы частично взвалить ее на авторов). Далее в подобных предисловиях будет указано, с кого именно спрашивать за тот или иной раздел.

Соавтор

Часть 5. СИСТЕМА ПРОГРАММИРОВАНИЯ МОДУЛА-2

Предисловие соавтора

За эту часть надо спрашивать с А.Недори, который, к тому же, является создателем Модула-компилятора для Кроноса.

Соавтор

Система программирования Модула-2 представляет собой систему из четырех относительно независимых компонент:

1) компилятора с языка Модула-2;
2) универсального редактора текстов;
3) средств отладки программ;
4) набора библиотек стандартных процедур.

Универсальному редактору текстов посвящен специальный раздел.

Набор библиотек стандартных процедур, а также способы их использования описаны в справочнике "Библиотеки ОС Excelsior".

В этом разделе описывается то, что касается создания, компиляции и отладки Модула-программ: версия языка Модула-2, реализуемая компилятором "mx" для процессоров семейства Кронос, ограничения и расширения языка, способы запуска и структура компилятора, необходимое компилятору окружение, визуализатор программ и посмертный историк.

Глава 5.1. Входной язык компилятора

Язык Модула-2 реализуется компилятором в соответствии с сообщением о языке [5] и изменениями, внесенными Н.Виртом [2].

Кроме того, реализованы некоторые расширения языка. Часть изменений была внесена в язык под влиянием дальнейших работ Н.Вирта [3] и Проекта стандарта языка [4]. Компилятор не соответствует ISO-стандарту языка Модула-2, поскольку таковой стандарт в данное время отстутствует.

5.1.1. Ограничения компилятора

Список требований к компилятору взят из проекта [4]. Символом '*' помечены пункты, не соответствующие требованиям проекта. В скобках () приведены рекомендации проекта.

```
*    - размер множеств не более 32 бита    (256, SET OF CHAR);
     - число параметров процедуры не более 256            (8);
     - число импортируемых модулей не ограничено          (32);
     - вложенность процедур и модулей не ограничена        (8);
     - вложенность описания записей  не ограничена       (256);
     - вложенность описания массивов  не ограничена      (256);
     - вложенность вызовов процедур  не ограничена       (256);
     - число альтернатив в операторе выбора не более 256(256);
     - длина идентификатора не ограничена                (256);
     - число экспортируемых объектов  не ограничено       (16);
     - длина строкового литерала не более 256             (80);
     - вложенность операторов не ограничена                (8);
     - размер перечислимого типа не ограничен            (128).
```

Примечание: параметры типа открытый массив (ARRAY OF) и параметры-последовательности (SEQ-параметры) считаются за два параметра.

Дополнительные ограничения:

- число параметров у кодовой процедуры не более 7.

5.1.2. Изменения, внесенные в язык

В 5.1.2.1 содержится краткий список изменений. В 5.1.2.2 приводится описание изменений, внесенных в язык, базирующееся на Сообщении [5]. Описание выдержано в стиле Сообщения [5] с точностью до порядка нумерации разделов.

Полный синтаксис входного языка приведен в 5.1.2.3.

Примеры программ, демонстрирующие расширения, приведены в 5.1.6.

5.1.2.1. Список изменений

1) Неявная конкатенация строк.
2) Дополнительный синтаксис комментария.
3) Отсутствуют стандартные типы CARDINAL, LONGINT, LONGREAL.
4) Стандартный тип STRING.
5) Тип динамический массив.
6) Новые операции для целых "/" и REM.
7) Операция инвертирования множества.
8) Операции циклического сдвига.
9) Конструктор массивов.
10) Передача параметров по доступу.
11) Параметры-последовательности.
12) Кодовые процедуры.
13) Новые стандартные процедуры ASSERT, BITS, BYTES, LEN, REF, NEW, DISPOSE, RESIZE.
14) Переименование при импорте.
15) Все, что касается процессов и сопрограмм, реализуется средствами ОС, а не компилятора.
16) Настраиваемая динамическая поддержка.

5.1.2.2. Дополнения к Сообщению о языке Модула-2

Данный пункт содержит описание изменений входного языка, базирующееся на Сообщении [5]. При описании изменений указывается пункт сообщения, к которому оно относится, например, <8.2.4>. Если описание расширения должно быть выделено в новый пункт по логике построения сообщения, то такой пункт помечается символом "*".

<3> Словарь и изображение

2. Внесены изменения в синтакис Целого (см. 5.1.2.3) для того, чтобы описать изменения в следующем пункте.

3. Расширен синтаксис цепочки. Цепочка есть последовательность строк и представлений литер, начинающаяся со строки.

```
$   Цепочка = Строка  { Строка | ПредставлениеЛитеры }.
$   Строка  =    "'" { Литера } "'" | '"' { Литера } '"'.
```

Примеры:

```
"Привет" 12c 15c 'читателю'
'' 33c 'H' 33c 'J'
```

4. Дополнительные зарезервированные слова:

```
CODE
DYNARR
FORWARD
REM
SEQ
VAL
```

5. Дополнительные синтакисис комментария: от "--" до конца строки. Внутри комментария одного вида комментарий другого вида не рассматривается.

Примеры:

1)

```
        -- (*
           i:=0; -- этот оператор не в комментарии;
        -- *)
```

2)

```
        (* -- *) i:=0;  -- и этот оператор не в комментарии.
```

<4> Описания и правила видимости

Отсутствуют стандартные идентификаторы:

```
CARDINAL
LONGINT
LONGREAL
VAL
```

Добавлены стандартные идентификаторы:

```
ASSERT  (10.2.1*)
BITS    (10.2.1*)
BYTES   (10.2.1*)
DISPOSE (10.2.2*)
LEN     (10.2.1*)
NEW     (10.2.2*)
REF     (10.2.1*)
RESIZE  (10.2.2*)
STORAGE (14.2*)
STRING  (6.9)
```

<6> Описание типов

Добавлен тип динамический массив (динмассив).

<6.1> Основные типы

Отсутствует типы CARDINAL, LONGINT, LONGREAL.

<6.8> Тип процедура

```
$    ТипПроцедура ⇓ PROCEDURE [ СписокФормТипов ].
$    СписокФормТипов = "(" [[VAR] ФормТип
$        { "," [VAR] ФормТип } ] [ SEQ [VAR] КвалИдент ] ")"
$        [ ":" КвалИдент ].
```

Последний параметр в списке формальных параметров может быть последовательностью параметров (см. <10.1>).

<6.9*> Тип динмассив

Тип динмассив является обобщением типа указателя на случай указателя на открытый массив.

```
$  ТипДинМассив = DYNARR OF Тип.
```

DYNARR OF T следует рассматривать как POINTER TO ARRAY OF T.

Значение типа динмассив состоит из дескриптора и тела. Дескриптор динмассива содержит два поля ADR и HIGH, описывающие соотственно адрес начала массива и верхнюю границу массива. Динмассив почти всегда ведет себя как открытый массив. Значениями индекса могут быть целые числа (INTEGER) в диапазоне от 0 до верхней границы.

Стандартный тип STRING определяется следующим образом:

```
STRING = DYNARR OF CHAR
```

Примеры:

```
DYNARR OF CHAR
DYNARR OF STRING
```

Примеры использования динмассивов рассматриваются в 5.1.3 и 5.1.6. См. также пункты <8.1>, <10.2>.

<8> Выражения

В <8.1> рассматриваются изменения, касающиеся динмассивов, в <8.2> - новые операции. Новый пункт <8.3*> посвящен конструкторам массивов.

<8.1> Операнды

Если структура - динмассив D, то запись D[E] обозначает элемент D, индекс которого - текущее выражение значения E. Обозначение вида D[E1,E2,...En] означает то же, что и D[E1][E2]...[En]. Запись D^ означает дескриптор динмассива, то есть запись с двумя полями ADR и HIGH. Запись D^.f означает поле f дескриптора D (f должно быть ADR или HIGH).

Примеры:

```
str : STRING
text: DYNARR OF STRING

str[i]       (CHAR)
text[i]      (STRING)
text[i,j]    (CHAR)
str^         (см. примечание)
str^.ADR     (ADDRESS)
str^.HIGH    (INTEGER)
```

Примечание: Каждому типу динмассива соответсвует уникальный анонимный тип дескриптора вида:

```
RECORD
  ADR : SYSTEM.ADDRESS;
  HIGH: INTEGER;
END;
```

<8.2> Операции

Введены дополнительные операции типа умножения и унарная операция инвертирования множества (<8.2.3>). Операции REM и "/" для целых рассматривается в <8.2.1>, а операции "<<" и ">>" в <8.2.5*>.

```
$ ОперацияТипаУмножения = "*" | "/" | DIV
                | MOD | AND | REM | "<<" | ">>".
```

<8.2.1> Арифметические операции

В соответствии с Проектом стандарта [4] (см. п.6.6.1.2 Проекта) введены целые операции "/" и REM для целого деления и взятия остатка соответственно.

Выполняются соотношения:

```
(-x/y)=(-x/y)=-(x/y)            (для y#0)
x=(x  /  y)*y+(x REM y)         (для y#0)
x=(x DIV y)*y+(x MOD y)         (для y>0)
```

Результат операции x REM y определен при y#0 и является либо

нулем, либо целым числом, совпадающим по знаку с x и меньшим по абсолютной величине, чем y.

Результат операции x MOD y определен при y>0 и является неотрицательным целым числом, меньшим y.

Примечание. Для процессоров Кронос 2.2 операции DIV и MOD реализованы некорректно и эквивалентны операциям /, REM. Подробнее см. 5.1.4.2.

<8.2.3> Операции над множествами

Унарная операция "-" обозначает побитовую инверсию множества. Результат операции имеет тип операнда.

<8.2.5*> Операции циклического сдвига

Операции циклического сдвига ">>" и "<<". Левый операнд - любого типа длиной 1 слово, правый должен быть типа INTEGER. Тип результата операции ясен из таблицы:

Тип левого операнда	Тип результата
INTEGER	INTEGER
BITSET	BITSET
все остальные	WORD

<8.3*> Конструктор массивов

Расширен синтаксис множителя. В качестве множителя можно использовать конструктор массива.

```
$   Множитель = .... | Конструктор.
$   Конструктор = ARRAY OF КвалИдент
$      "{" КонстВыражение { "," КонстВыражение } "}".
```

Конструктор ARRAY OF T{E1,E2,...En} обозначает константное значение массива типа ARRAY [0..n-1] OF T, где n - число выражений. Тип элемента должен быть основным типом, типом перечисления, диапазона или множества.

Пример:

```
CONST
  array   = ARRAY OF INTEGER{ 1,2,3 }
  element = array[1];
```

<9.2>. Вызовы процедур

Последним формальным параметром процедуры может быть параметр-последовательность (SEQ-параметр). При вызове вместо такого параметра может быть подставлена последовательность фактических параметров любой длины (в том числе и пустая). Каждый из этих фактических параметров должен быть обозначением переменной для случая последовательности переменных (SEQ VAR) и выражением в случае последовательности значений (SEQ). Совместимость как в случае обычных параметров.

Вместо последовательности параметров в качестве такого SEQ-параметра может быть подставлен формальный параметр-последовательность. При этом их типы должны совпадать и признак VAR должен быть у обоих или у обоих отсутствовать.

В случае, если тип параметра-последовательности есть WORD, в качестве фактических параметров можно использовать параметры любого типа. При передаче структурных параметров передается их адрес, а если параметр имеет тип динмассив, то передается адрес массива (а не адрес дескриптора).

<9.2.1*> Вызов кодовых процедур

При вызове кодовой процедуры фактические параметры вычисляются в соответствии с заголовком процедуры и загружаются на стек. Далее выполняется код, полученный текстуальной вставкой тела кодовой процедуры в место вызова. Компилятор не контролирует корректность работы кодовых процедур со стеком.

<10>. Описание процедур

```
$    ОписаниеПроцедуры = ЗаголовокПроцедуры ";"
$          ( Блок | Код ) Идентификатор.
$    Код = CODE { КонстВыражение } END.
$    ЗаголовокПроцедуры = PROCEDURE Идентификатор
$          [ ФормальныеПараметры ].
$    Блок = { Описание } [ BEGIN ПослОператоров ] END.
$    Описание = CONST { ОписаниеКонстанты ";" } |
$               TYPE  { ОписаниеТипа ";" } |
$               VAR   { ОписаниеПеременной ";" } |
$               ОписаниеПроцедуры ";" | ОписаниеМодуля ";"
$               ЗаголовокПроцедуры ";" FORWARD ";".
```

В соответствии с [2,4] разрешены предварительные описания процедур. Предварительное описание представляет собой заголовок процедуры, за которым следует зарезервированный идентификатор FORWARD. Полное описание процедуры, включающее тело процедуры, должно появиться в той же области действия, что и предварительное описание, и на том же уровне видимости. Типы формальных параметров у предварительного и полного описания процедуры должны быть совместимы, а в случае открытых

массивов совместимы должны быть базовые типы массивов.

Введены кодовые процедуры. Тело кодовой процедуры представляет собой последовательность байтов. Каждый байт задается целым выражением в диапазоне 0..255. (см. <9.2>). число параметров у кодовых процедур ограничено (см. 5.1.1).

Запрещается предварительное описание кодовых процедур и экспорт из определяющего модуля.

<10.2> Формальные параметры

```
$     ФормальныеПараметры =
$          "(" [ ФПСекция { ";" ФПСекция } ]
$            [ SEQ [ VAR ] Идентификатор ":" КвалИдент ] ")"
$            [ ":" КвалИдент ].
$     ФПСекция = [ VAR | VAL ] СписИдент ":" ФормТип.
$     ФормТип = [ ARRAY OF ] Квалидент.
```

Введен новый способ передачи параметра - передача по доступу. Этот способ передачи указывается только при полном описании процедуры (нельзя использовать в процедурном типе и в предварительном описании процедуры). С точки зрения вызова передача параметра по доступу эквивалентна передаче параметра по значению. Формальный параметр, переданый таким способом, нельзя модифицировать. Присваивание такому параметру (или компоненте параметра) и передача его по сслыке запрещены. Структурный параметр, переданный по доступу, не копируется, что позволяет повысить эффективность процедуры и уменьшить требования к памяти.

Последний параметр процедуры может быть параметром-последовательностью. Такой параметр может быть передан по сслылке (SEQ VAR) или по доступу, если VAR отсутствует. Такой формальный параметр внутри процедуры аналогичен гибкому массиву.

Параметр-последовательность типа T может быть передан целиком процедуре с формальным параметром-последовательностью типа T1, если тип T совместим с типом T1 и способ передачи у обоих параметров одинаковый (оба по доступу или оба по ссылке).

Снято ограничение на поэлементное использование гибких массивов. Разрешены присваивания гибких массивов (и параметра-последовательности) целиком. Компилятор вставляет динамические проверки того, что длина массива в левой части равна длине массива в правой части оператора присваивания. Для литерных массивов размеры сравниваются не на равенство, а на больше или равно.

<10.2> Стандартные процедуры

В <10.2.1*> описываются введенные стандартные процедуры, в <10.2.2*> - настраиваиваемые стандартные процедуры работы с памятью, в данном пункте описываются изменения в существующих стандартных процедурах.

1. У процедуры HALT разрешен необязательный параметр типа INTEGER, значение которого системно-зависимо. Вызов процедуры HALT с параметром означает аварийное завершение задачи и может приводить к действиям, облегчающим отладку. Вызов процедуры HALT без параметра означает нормальное завершение задачи, неотличимое от выполнения возврата из нулевой процедуры головного модуля.

2. Процедура HIGH может применяться к массивам, гибким массивам и динмассивам. Результат имеет тип INTEGER и является константой, если операнд имеет тип массив.

3. Процедура ORD возвращает результат типа INTEGER.

4. Параметр процедуры SIZE может быть произвольным обозначением. Результат имеет тип INTEGER и является константой, если операнд не является гибким массивом или динмассивом. Размер выдается в единицах адресации (для процессоров Кронос - в словах).

5. Отсутствует процедура VAL.

<10.2.1*> Дополнительные стандартные процедуры

1. ASSERT(BOOLEAN ["," INTEGER]);

Если значение первого операнда - истина, то вызов эквивалентен пустому оператору, иначе исполнение программы завершается. Второй параметр может отсутствовать, а если он есть, то это некоторая дополнительная информация о причине завершения. Например:

```
ASSERT(adr=NIL,НетПамяти);
```

где НетПамяти - это константа ОС.

2. BITS(Обозначение | Тип)

Аналогична процедуре SIZE, но выдает размер объекта или типа в битах.

3. BYTES(Обозначение | Тип)

Аналогична процедуре SIZE, но выдает размер объекта или типа в байтах.

4. LEN(VAR x: ЛюбойМассив)

Выдает число элементов массива, гибкого массива или динмассива. Результат имеет тип INTEGER и является константой, если операнд имеет тип массив.

5. REF(VAR x: T , T2)

T - произвольный тип, T2 должен быть типом указателя на тип T (T2 = POINTER TO T). Выдает значение типа T2, являющееся адресом объекта x.

<10.2.2*> Настраиваемые стандартные процедуры

Стандартные процедуры работы с памятью выполняют операции размещения, освобождения указателей и динмассивов и изменения размера динмассивов. В описании процедуры используются обозначения P - для произвольного указателя, D - для произвольного динмассива. Перед использованием этих стандартных процедур необходимо настроить их с помощью описателя динамической поддержки (см. <14.1*>). При настройке задается соответствие между стандартной процедурой и процедурой, определенной пользователем, которая и будет вызываться при вызове стандартной процедуры.

1. NEW(P)

Выделение памяти для указателя размером SIZE(P). Должна быть настроена процедура NEW.

NEW(D [,N])

Размещение памяти для динмассива с числом элементов N. После вызова верхняя граница динмассива равна N-1. Вызов NEW(D) эквивалентен вызову NEW(D,0). При этом вызов NEW(D,N) является сокращением следующего действия:

D^.HIGH:=-1; RESIZE(D,N).

Должна быть настроена процедура RESIZE.

2. DISPOSE(P)

Освобождение памяти, занятой указателем. Должна быть настроена процедура DISPOSE.

DISPOSE(D)

Освобождение памяти, занятой динмассивом. При этом вызов DISPOSE(D) является сокращением вызова RESIZE(D,0). Должна быть настроена процедура RESIZE.

3. RESIZE(D,N)

Изменение размера динмассива. При N=0 эквивалентна вызову

процедуры DISPOSE. После вызова верхняя граница динмассива равна N-1. Сохраняет значения элементов динмассива для индексов 0..min(H0,N-1), где H0 - верхняя граница динмассива до вызова процедуры RESIZE. Должна быть настроена процедура RESIZE.

<11> Модули

Приоритет для локальных модулей не реализован.

```
$    Импорт = [ FROM Идентификатор ] IMPORT СписИмпорта.
$    СписИмпорта = ИмпортСекция { "," ИмпортСекция }.
$    ИмпортСекция = [ Идентификатор ":" ] Идентификатор.
```

В списке импорта разрешено переименование импортируемых идентификаторов (аналогично [3]), что позволяет использовать внутри модуля короткие (удобные) названия для этих идентификаторов. Запись

```
  IMPORT  in: OUT;
```

означает, что объект, видимый вне модуля под именем OUT, должен использоваться внутри модуля по имени in.

Пример:

```
    IMPORT  io: StdIO;

    io.Write('A'); -- обозначает StdIO.Write('A')
```

<12> Системно-зависимые возможности

Модуль SYSTEM содержит типы WORD, ADDRESS и стандартную процедуру ADR.

<13> Процессы

Операции создания процессов, обработки прерываний и переключения процессов реализованы в библиотечных модулях. Поэтому средства, описываемые в данном пункте Сообщения, не реализованы.

<14> Единицы компиляции

```
$    Определение = CONST { ОписаниеКонстанты ";" } |
$                  TYPE  { Идентификатор [ "=" Тип ] ";" } |
$                  VAR   { ОписаниеПеременной ";" } |
$                  VAL   { ОписаниеПеременной ";" } |
$                  ЗаголовокПроцедуры ";".
```

В определяющем модуле можно описывать переменные, которые в других модулях можно только читать (нельзя модифицировать). Такие переменные называются переменными только для чтения (read only, или VAL-переменные). В реализующем модуле, который соответствует данному определяющему, использование такой переменной не отличается от использования обычных переменных.

В заголовке программого и реализующего модуля допускается указание "приоритета". Приоритет модуля - это целое число, которое записывается в порожденный компилятором кодофайл и используется исполняющей системой в соответствии со стандартом ОС (см. 5.1.4).

```
$    ПрограммныйМодуль = MODULE Идентификатор
$         [ Приоритет ] ";" { Импорт }
$         { Описание | ДинПоддержка }
$         [ BEGIN ПослОператоров ] END Идентификатор ".".
```

Среди описаний в программном и реализующем модулях разрешается использование определения динамической поддержки (см. <14.1*>).

<14.1*> Динамическая поддержка

```
$   ДинПоддержка = WITH Идентификатор
$        ( ":" КвалИдент |  "(" СписСоответствий ")" ).
$   СписСоответствий = Соответствие { ";" Соответствие }.
$   Соответствие = Идентификатор ":" КвалИдент.
```

Понятие динамической поддержки определяет набор процедур (и, может быть, других объектов), которые компилятор будет использовать при генерации некоторых конструкций языка. При этом указывается стандартный идентификатор - имя раздела, а затем следует указание соответствий или имя модуля, содержащего необходимые объекты со стандартными именами.

В текущей версии языка (и компилятора) определен один раздел динамической поддержки - STORAGE, содержащий настраиваемые операции работы с памятью.

<14.1*> Динамическая поддержка работы с памятью

Три стандартные процедуры (см. <10.2.2*>) требуют определения динамической поддержки для раздела STORAGE.

NEW: Настраивается процедурой типа

```
     PROCEDURE (VAR ADDRESS, INTEGER);
```

,где второй параметр определяет размер требуемой памяти в словах.

```
DISPOSE: Настраивается процедурой типа
     PROCEDURE (VAR ADDRESS, INTEGER);
После вызова первый параметр равен NIL.

RESIZE: Настраивается процедурой типа
           PROCEDURE (VAR ADDRESS,
                      VAR INTEGER,
                          INTEGER,
                          INTEGER);
```

Первый параметр - адрес массива, второй - его верхняя граница, третий - требуемая длина, четвертый - размер элемента массива в байтах.

При сокращенной настройке указывается имя модуля, который содержит процедуры с именами: ALLOCATE, DEALLOCATE и REALLOCATE. Эти процедуры должны иметь типы, описанные выше. Часть процедур может отсутствовать, при этом соответствующая стандартная процедура будет не определена. Все операции над динмассивами реализуются компилятором посредством процедуры REALLOCATE. Соответственно, программа, использующая динмассивы, должна содержать настройку для процедуры RESIZE.

Пример:

```
WITH STORAGE (NEW: Storage.ALLOCATE; DISPOSE: dealloc);
WITH STORAGE: Heap;
```

5.1.2.3. Синтаксис входного языка

Приведен синтакис Модулы-2 из [5] дополненый расширениями языка. Символом '*' помечены измененные и добавленные правила.

```
   1    Идентификатор = Буква {Буква | Цифра}.
   2    Число = Целое | Действительное.
   3    Целое = Цифра { Цифра } | ВосьмеричнаяЦифра
   4            { ВосмеричнаяЦифра } "B" |
   5            Цифра { ШестнадцатеричнаяЦифра } "H" |
*  5.1          ПредставлениеЛитеры.
*  5.2  ПредставлениеЛитеры =
*  5.3       ВосьмеричнаяЦифра { ВосмеричнаяЦифра } "C".
   6    Действительное = Цифра { Цифра }
   7                     "." { Цифра } [Порядок].
   8    Порядок = "E" [ "+" | "-" ] Цифра { Цифра }.
   9    ШестандатиричнаяЦифра =
  10                        Цифра |"A"|"B"|"C"|"D"|"E"|"F".
  11    Цифра = ВосьмеричнаяЦифра |"8"|"9".
  12    ВосьмеричнаяЦифра =
  13                       "0"|"1"|"2"|"3"|"4"|"5"|"6"|"7".
* 14    Цепочка = Строка  { Строка | ПредставлениеЛитеры }.
* 14.1  Строка  =    "'" { Литера } "'" | '"' { Литера } '"'.
  15    КвалИдент = Идентификатор { "." Идентификатор }.
  16    ОписаниеКонстанты =
```

```
  17              Идентификатор "=" КонстВыражение.
  18    КонстВыражение = Выражение.
  19    ОписаниеТипа = Идентификатор "=" Тип.
  20    Тип = ПростойТип | ТипМассив | ТипЗапись
  21          | ТипМножество | ТипУказатель | ТипПроцедура
* 21.1        | ТипДинМассив.
  22    ПростойТип = КвалИдент | Перечисление
  23                 | ТипДиапазон.
  24    Перечисление = "(" СписИдент ")".
  25    СписИдент = Идентификатор { "," Идентификатор }.
* 26    ТипДиапазон =
  27      "[" КонстВыражение ".." КонстВыражение "]".
  28    ТипМассив = ARRAY ПростойТип { "," ПростойТип }
  29                OF Тип.
  30    ТипЗапись = RECORD ПослСписковКомпонент END.
  31    ПослСписковКомпонент = СписокКомпонент
  32                         { ";" СписокКомпонент }.
  33    СписокКомпонент ⇓   [ СписИдент ":" Тип |
  34      CASE [Идентификатор] ":" КвалИдент OF Вариант
  35               { "|" Вариант}
  36      [ ELSE ПослСписковКомпонент ] END ].
  37    Вариант = [ СписокМетокВарианта ":" ]
  38              ПослСписковКомпонент.
  39    СписокМетокВарианта = МеткиВарианта
  40              { "," МеткиВарианта }.
  41    МеткиВарианта = КонстВыражение [ ".." КонстВыражение ].
  42    ТипМножество = SET OF ПростойТип.
  43    ТипУказатель = POINTER TO Тип.
  44    ТипПроцедура ⇓ PROCEDURE [ СписокФормТипов ].
  45    СписокФормТипов = "(" [[VAR] ФормТип
* 46       { "," [VAR] ФормТип } ] [ SEQ [VAR] КвалИдент ] ")"
* 46.1     [ ":" КвалИдент ].
* 46.2  ТипДинМассив = DYNARR OF Тип.
  47    ОписаниеПеременной = СписИдент ":" Тип.
  48    Обозначение = КвалИдент { "." Идентификатор |
  49                  "[" СписВыражений "]" | "^" }.
  50    СписВыражений = Выражение { "," Выражение }.
  51    Выражение = ПростоеВыражение
  52                [ Отношение ПростоеВыражение ].
  53    Отношение = "=" | "#" | "<" | "<="
  54              | ">" | ">=" | IN.
  55    ПростоеВыражение = [ "+" | "-" ] Слагаемое
  56             { ОперацияТипаСложения Слагаемое}.
  57    ОперацияТипаСложения = "+" | "-" | OR.
  58    Слагаемое = Множитель
  59             { ОперацияТипаУмножения Множитель}.
  60    ОперацияТипаУмножения = "*" | "/" | DIV
  61             | MOD | AND
* 61.1           | REM | "<<" | ">>".
  62    Множитель = Число | Цепочка | Множество |
  63             Обозначение [ ФактическиеПараметры ] |
  64             "(" Выражение ")" | NOT Множитель |
* 64.1           Конструктор.
```

```
  65    Множество = [ КвалИдент ]
  66                "{" [ Элемент { "," Элемент } ] "}".
  67    Элемент = Выражение [ ".." Выражение ].
* 67.1  Конструктор = ARRAY OF КвалИдент
* 67.2      "{" КонстВыражение { "," КонстВыражение } "}".
  68    ФактическиеПараметры = "(" [ СписВыражений ] ")".
  69    Оператор = [ Присваивание | ВызовПроцедуры |
  70               УсловныйОператор | ОператорВыбора  |
  71               ЦиклПока | ЦиклДо | БезусловныйЦикл |
  72               ЦиклСШагом | ОператорПрисоединения |
  73               EXIT | RETURN [ Выражение ]].
  74    Присваивание = Обозначение ":=" Выражение.
  75    ВызовПроцедуры =
  76              Обозначение [ ФактическиеПараметры ].
  77    ПослОператоров = Оператор { ";" Оператор }.
  78    УсловныйОператор = IF Выражение THEN ПослОператоров
  79                  { ELSIF Выражение THEN ПослОператоров }
  80                  [ ELSE ПослОператоров ] END.
  81    ОператорВыбора = CASE Выражение OF Альтернатива
  82       { "|" Альтернатива } [ ELSE ПослОператоров ] END.
  83    Альтернатива = [ СписокМетокВырианта ":"
  84                     ПослОператоров ].
  85    ЦиклПока = WHILE Выражение DO ПослОператоров END.
  86    ЦиклДо = REPEAT ПослОператоров UNTIL Выражение.
  87    ЦиклСШагом = FOR Идентификатор ":="
  88             Выражение TO Выражение
  89             [ BY КонстВыражение ] DO ПослОператоров END.
  90    БезусловныйЦикл = LOOP ПослОператоров END.
  91    ОператорПрисоединения = WITH Обозначение
  92                              DO ПослОператоров END.
  93    ОписаниеПроцедуры = ЗаголовокПроцедуры ";"
* 94           ( Блок | Код ) Идентификатор.
* 94.1  Код = CODE { КонстВыражение } END.
  95    ЗаголовокПроцедуры = PROCEDURE Идентификатор
  96           [ ФормальныеПараметры ].
  97    Блок = { Описание } [ BEGIN ПослОператоров ] END.
  98    Описание = CONST { ОписаниеКонстанты ";" } |
  99               TYPE  { ОписаниеТипа ";" } |
 100               VAR   { ОписаниеПеременной ";" } |
 101               ОписаниеПроцедуры ";" | ОписаниеМодуля ";"
*101.1             ЗаголовокПроцедуры ";" FORWARD ";".
 102    ФормальныеПараметры =
 103         "(" [ ФПСекция { ";" ФПСекция } ]
*103.1         [ SEQ [ VAR ] Идентификатор ":" КвалИдент ] ")"
 103.2         [ ":" КвалИдент ].
*104    ФПСекция = [ VAR | VAL ] СписИдент ":" ФормТип.
 105    ФормТип = [ ARRAY OF ] Квалидент.
*106    ОписаниеМодуля = MODULE Идентификатор
 107          ";" { Импорт } [ Экспорт ] Блок Идентификатор.
 108    Приоритет = "[" КонстВыражение "]".
 109    Экспорт = EXPORT [ QUALIFIED ] СписИдент ";".
*110    Импорт = [ FROM Идентификатор ] IMPORT СписИмпорта.
*110.1  СписИмпорта = ИмпортСекция { "," ИмпортСекция }.
```

```
*110.2  ИмпортСекция = [ Идентификатор ":" Идентификатор ].
 111    МодульОпределений = DEFINITION MODULE Идентификатор
 112                  ";" { Импорт } { Определение } END
 113                  Идентификатор ".".
 114    Определение = CONST { ОписаниеКонстанты ";" } |
 115                TYPE  { Идентификатор [ "=" Тип ] ";" } |
 116                VAR   { ОписаниеПеременной ";" } |
*116.1              VAL   { ОписаниеПеременной ";" } |
 117                ЗаголовокПроцедуры ";".
 118    ПрограммныйМодуль = MODULE Идентификатор
 119      [ Приоритет ] ";" { Импорт }
*119.1    { Описание | ДинПоддержка }
 119.2    [ BEGIN ПослОператоров ] END Идентификатор ".".
*119.3  ДинПоддержка = WITH Идентификатор
*119.4      ( ":" КвалИдент |  "(" СписСоответствий ")" ).
*119.5  СписСоответствий = Соответствие { ";" Соответствие }.
*119.5  Соответствие = Идентификатор ":" КвалИдент.
 120    ЕдиницаКомпиляции = МодульОпределения |
 121      [ IMPLEMENTATION ] ПрограммныйМодуль.
```

5.1.3. Еще раз о расширениях

В этом пункте делается попытка заранее ответить на следующие вопросы читателя касательно расширений:

- Зачем нужны расщирения?
- Кому это выгодно?
- А как этим пользоваться?

Примеры использования расширений приводятся в 5.1.6.

5.1.3.1. Динмассивы

Динмассивы удобно использовать для хранения информации, размеры которой статически не известны. Необходимо понимать двойственность структуры динмассива (или тройственность?).

Динмассив представляет собой пару дескриптор + массив. Значение типа динмассив занимает два слова под дескриптор и сколько надо слов под тело массива. Если динмассив является компонентой некоторой структуры (записи, массива, динмассива), то при копировании структуры будет скопирован только дескриптор, например:

```
VAR a,b: RECORD
           dyn: DYNARR OF INTEGER;
           ptr: POINTER TO INTEGER;
         END;

   a:=b;
```

После этого присваивания поля a.dyn и b.dyn будут ссылаться на одно и то же тело массива. Ситуация аналогична ситуации для указателей: поля a.ptr и b.ptr указывают в одно место памяти.

Дескриптор динмассива (или паспорт) представляет собой запись из двух полей:

```
RECORD
  ADR : SYSTEM.ADDRESS;
  HIGH: INTEGER;
END;
```

Поле ADR содержит адрес массива, а поле HIGH - значение его верхней границы.

Запись d^ (где d - динмассив) обозначает доступ к дескриптору, d^.ADR и d^.HIGH обозначает соответствующие поля дескриптора.

Динмассив ведет себя как гибкий массив в следующих случаях:

- в индексации d[i];
- в вызовах стандартных процедур HIGH, LEN, SIZE, BYTES, BITS;
- при передаче параметра, если формальный параметр является гибким массивом;
- при передаче параметра, если тип формального параметра - последовательность слов.

В следующем примере приводятся почти все случаи использования динмассивов:

```
PROCEDURE p1(    s: ARRAY OF CHAR); END p1;
PROCEDURE p2(VAR s: ARRAY OF CHAR); END p2;
PROCEDURE p3(    s: STRING);        END p3;
PROCEDURE p4(VAR s: STRING);        END p4;
PROCEDURE p5(SEQ x: SYSTEM.WORD);   END p5;

VAR a,b: STRING;
    adr: ADDRESS;
      n: INTEGER;

   a:=b;           -- копирование строк;
   a^:=b^;         -- копирование дескрипторов;
   a^.HIGH:=1;     -- изменение верхней границы;
   a^.ADR:=NIL;    -- изменение адреса массива;
   n:=SIZE(a^);    -- n = 2;
   n:=SIZE(a);     -- n = размер массива;
   adr:=ADR(a);    -- adr = a^.ADR;
   adr:=ADR(a^);   -- adr = адрес дескриптора;
   p1(a);          -- передача массива по значению;
   p2(a);          -- передача массива по ссылке;
   p3(a);          -- передача дескриптора по значению;
   p4(a);          -- передача дескриптора по ссылке;
```

```
p5(a);          -- передача адреса массива;
p5(a^);         -- передача адреса дескриптора.
```

Стандартные процедуры NEW, DISPOSE, RESIZE (см. 5.1.2) реализуют операции выделения памяти динмассиву, возвращения памяти и изменения размера динмассива. При этом практически никогда не появляется необходимость обращаться к дескриптору динмассива.

5.1.3.2. Процедуры с переменным числом параметров

Последним параметром у процедуры может быть параметр-последовательность. При вызове процедуры вместо такого формального параметра может быть подставлена последовательность фактических параметров (в том числе и пустая). Внутри процедуры такой параметр аналогичен параметру типа ARRAY OF T. Если параметр передается не по ссылке, то присваивание элементам массива-параметра запрещено, то есть подразумевается передача по доступу.

Пример:

```
PROCEDURE Min(SEQ x: INTEGER): INTEGER;
  VAR min,i: INTEGER;
BEGIN
  ASSERT(HIGH(x)>=0);
  min:=x[0];
  FOR i:=1 TO HIGH(x) DO
    IF x[i]<min THEN min:=x[i] END
  END;
  RETURN min
END Min;

........

i:=Min(1,2,3,4);
```

Параметр-последовательность типа T может быть передан целиком процедуре с формальным параметром-последовательностью типа T1, если тип T совместим с типом T1 и способ передачи у обоих параметров одинаковый (оба по значению или оба по ссылке).

```
PROCEDURE p(SEQ x: INTEGER);
.......
  i:=Min(x);
.......
```

Замечание: если тип последовательности WORD, то фактическими параметрами могут быть не только однословные объекты, но и структуры. В этом случае передается только их адрес и копирования не происходит.

Введение в язык процедур с переменным числом параметров оказало большое влияние на библиотеки ввода/вывода. Вместо большого количества процедур вывода, таких, как WriteChar, WriteInteger, WriteReal и т.д., что характерно для других реализаций Модулы-2, наши библиотеки содержат процедуры форматного вывода, которые позволяют формировать выходной поток существенно удобней.

```
PROCEDURE print(format: ARRAY OF CHAR; SEQ x: SYSTEM.WORD);
```

Строка format содержит описание формата вывода (в стандарте Unix SYSTEM V).

Пример:

```
   print('factorial(%d)=%d\n',n,factorial(n));
```

Если n=5 (и процедура с именем factorial действительно реализует факториал), то результатом будет:

```
factorial(5)=120
```

5.1.3.3. Динамическая поддержка, определяемая пользователем

Одним из преимуществ Модулы-2 как языка программирования систем (в том числе ОС и систем реального времени) является отсутсвие необходимости в динамической поддержке (Run Time Support - RTS). Отсутствие RTS отвязывает компилятор от реальной системы, в которой будет работать полученный им код.

В то же время есть много полезных вещей, которые хочется включить в язык, но реализовать в компиляторе их по разным причинам трудно (или нежелательно). К таким вещам относятся:

- исключительные ситуации;
- аккуратные операции над кучей (NEW, DISPOSE, RESIZE);
- операторы параллельности;
- и так далее...

Реализация всего этого в компиляторе требует некоторых знаний о системе, а в качестве библиотеки их невозможно реализовать без потери надежности (или вообще невозможно реализовать, например RESIZE(любой_динмассив,новая_длина)).

Рассуждая таким образом, можно прийти к выводу о необходимости совместной реализации таких вещей в компиляторе и бибилиотеке. Компиляторная часть должна обеспечить надежность и контроль, библиотечная же часть может использовать конкретную систему. Существенным требованием к такой реализации должно оставаться отсутствие обязательного RTS, то есть: те модули, которые не пользуются новыми языковыми конструкциями, не должны изменяться от введения таковых.

В текущей версии с помощью динамической поддержки реализованы операции работы с памятью.

5.1.4. Режимы компиляции

5.1.4.1. Прагматы

Текст Модула-программы может содержать управляющие последовательности, которые мы будем называть прагматами. Прагматы служат для изменения режима компиляции внутри текста программы.

```
прагмат   = "(*$" директива "*)".
директива = { опция "+" | "-"  | "!" | "<" | ">" }
опция     = 'A'|'F'|'I'|'N'|'R'|'T'
```

Каждая опция означает действие, которое может быть включено (отключено) управляющей последовательностью. Смысл опций и их начальное состояние приводится в таблице.

N	Опция	Значение	По умолчанию
1	A	показ всех ошибок	off
2	F	аварийное окончание компиляции при первой встреченной ошибке	off
3	I	инициализация переменных типа указатель значением NIL; типа динмассив - парой (NIL,-1)	on
4	N	проверка на NIL для указателей и динмассивов. При включении автоматически включает опцию I	off
5	R	контроль выхода за границы отрезка	on
6	T	контроль индексов массивов	on

Остальные директивы определяют операции над стеком режимов трансляции.

1) < - запомнить текущее состояние опций в стеке;
2) > - восстановить последнее состояние опций из стека;
3) ! - счеркнуть весь стек режимов и перейти в исходное состояние.

Начальное состояние режимов компиляции может быть задано при запуске компилятора средствами системы программирования (см. 5.2).

5.1.4.2. Версия системы команд

Компилятор может порождать код модуля, используя три версии системы команд:

- базовая версия (версия 0). Код базовой версии может выполняться на всех моделях процессоров семейства Кронос;

- расширенная версия (версия 1). Компилятор использует набор дополнительных команд, реализованных на процессорах 2.5 и 2.6WS.

- версия рабочей станции (версия 2). Компилятор использует набор дополнительных команд, реализованных на процессоре 2.6WS.

Номер версии системы команд может быть задан при запуске компилятора (см. 5.1.2). Если номер версии не указан, то компилятор порождает код для того процессора, на котором работает.

За более подробными сведениями о версиях системы команд отсылаем читателя к книге "Архитектура процессоров семейства КРОНОС".

5.1.4.3. Приоритет модуля

В заголовке модуля может быть указан приоритет модуля - целое число, интерпретация которого зависит от ОС. В текущей версии определены следующие приоритеты:

0 - обычный модуль;
1 - уникальный модуль.

Данные уникального модуля не дублируются, если этот модуль используется несколькими задачами (подробнее см. в документации по ОС).

5.1.5. Стиль программирования

Этот пункт содержит необязательные рекомендации по разработке программ на языке Модула-2. Набор рекомендаций включает в себя правила расположения текста, способ комментирования, правила выбора идентификаторов и использования языковых конструкций. Эти правила были выработаны группой Кронос для внутреннего использования. Совокупность правил определяет "стиль" программирования. Использование единого стиля упрощает совместную работу над проектами и является простым способом увеличения читабельности программ.

Особенно важно придерживаться единого стиля при оформлении общеполезных библиотек.

5.1.5.1. Расположение текста

Отступы используются для выделения структуры программы. Ширина каждого отступа 2 символа (пробела). Следующий пример иллюстрируют рекомендуемую форму некоторых конструкций Модулы-2:

```
MODULE Example; (* 29-Mar-90. (c) KRONOS *)

CONST
  one = 1;
  two = 2;

TYPE
  Mode     = (root,path,leaf);
  node_ptr = POINTER TO node_rec;
  node_rec = RECORD
               mode : Mode;
               info : INTEGER;
               left : node_ptr;
               right: node_ptr;
             END;

VAR
  info: INTEGER;
  root: node_ptr;

PROCEDURE p(VAR n: node_ptr);
  VAR i: INTEGER;
BEGIN
  i:=0;
  n^.info:=info; INC(info);
  IF    node^.left =NIL THEN node^.left :=n
  ELSIF node^.right=NIL THEN node^.right:=n
  ELSE DEC(info);
  END;
END p;

BEGIN
  CASE node^.mode OF
    |root:
    |path:
  ELSE ASSERT(FALSE);
  END;
END Example;
```

Если языковая конструкция достаточно мала, то рекомендуется размещать ее на одной строке текста.

```
  IF a>b THEN min:=b ELSE min:=a END;
  FOR i:=0 TO HIGH(a) DO a[i]:=0 END;
```

Несколько простых операторов могут быть размещены на одной строке. Рекомендуется размещать на одной строке несколько "сильно" связанных операторов присваивания. Например, операторы, ввязывающие элемент в односвязный список:

```
n^.next:=head; head:=n;
```

5.1.5.2. Комментарии

Используются традиционные комментарии для оформления больших текстов. Комментарии в стиле Ады употребляются в следующих случаях:

- комментирование данной строки;
- пометка об изменении строки;
- указание ключевого места в тексте;
- выделение заголовка раздела.

Примеры:

```
TYPE
  node_rec = RECORD
               mode : Mode;      -- вид узла
               left : node_ptr;  -- левый  сын
               right: node_ptr;  -- правый сын
             END;

----------------------  NEW SECTION  ------------------------
                       ---------------

PROCEDURE joke(a,b: INTEGER): BOOLEAN;
  VAR min: INTEGER;
BEGIN
  IF a>b THEN min:=a ELSE min:=b END;  -- modified 29-Mar-90
  IF min<0 THEN RETURN TRUE END;
                       ----
  RETURN FALSE
END joke;
```

5.1.5.3. Именование

Как правило, все имена записываются только строчными буквами. Если имя состоит из нескольких слов, могут быть использованы два способа именования:

```
local_var      localVar
node_ptr       nodePtr
```

Рекомендуется использование подчерка.

Основные требования предъявляются к именам в определяющих модулях. Рекомендуется следующий стиль:

- процедуры в разных библиотеках, реализующие аналогичные операции, называются одинаково (Terminal.print, StdIO.print);

- существуют устойчивые пары имен, не надо смешивать имена из разных пар; например:

```
put      get
read     write
new      dispose
insert   delete
install  remove
```

- имена скрытых типов состоят только из заглавных букв;
- имена типов указателей и записей оканчиваются добавкой _ptr и _rec соответственно;

5.1.5.4. Использование языковых конструкций

В определяющих модулях не рекомендуется использование типа диапазона и типа перечисления, если набор констант может измениться. Так, не стоит заводить перечислимый тип OS_errors, но нет ничего плохого в том, чтобы определить тип

```
Week =(Monday,...,Sunday),
```

так как это от Бога.

В реализующих модулях не рекомендуется использование локальных модулей (от себя все равно не скроешь) и расквалифицирующего импорта (FROM IMPORT). Напротив, очень рекомендуется использование переименования при импорте. Это существенно увеличивает читабельность текста (всегда видно Что, Где и Почем).

В процедурах типа "open", "new", которые создают новый объект, рекомендуется делать возвращаемый объект первым параметром. Желательно воздерживаться от подряд идущих параметров одного типа, особенно - основного типа. Так, лучше писать:

```
PROCEDURE (INTEGER,BOOLEAN,INTEGER)
```

вместо

```
PROCEDURE (INTEGER,INTEGER,BOOLEAN).
```

5.1.5.5. Оформление определяющего модуля библиотек

Рекомендуется следующий вид библиотеки:

```
DEFINITION MODULE имя; (* <автор> <время написания> *)

ТЕКСТ НА МОДУЛЕ-2
```

```
(************************************************************

     Развернутый  комментарий, состоящий из описания принципов
использования  библиотеки  и  пояснения  к  каждой  процедуре,
содержащий  описания параметров процедуры, диапазон допустимых
значений и реакцию на исключительные ситуации.

************************************************************)

END имя.
```

5.1.6. Примеры программ

5.1.6.1. Рассмотрим простой модуль работы с текстом для текстового редактора. Модуль реализует операции сохранения строки текста, возвращения строки основному модуля редактора, удаления строк и так далее. В примере используется библиотека работы со строками.

```
DEFINITION MODULE edText; (* Ned 29-Mar-90. (c) KRONOS *)

PROCEDURE get(lineno: INTEGER; VAR s: ARRAY OF CHAR);
(* Выдает сожержимое строки по ее номеру. *)

PROCEDURE put(lineno: INTEGER;     s: ARRAY OF CHAR);
(* Запоминает содержимое строки *)

PROCEDURE delete(lineno,lines: INTEGER);
(* Удаляет lines строк, начиная с lineno *)

PROCEDURE insert(lineno,lines: INTEGER);
(* Вставляет lines пустых строк перед строкой lineno *)

END edText.

IMPLEMENTATION MODULE edText; (* Ned 29-Mar-90. (c) KRONOS *)

IMPORT  str: Strings;                                    (*0*)
IMPORT       Heap;

WITH STORAGE: Heap;                                      (*1*)

TYPE TEXT = DYNARR OF STRING;

VAR text: TEXT;

PROCEDURE get(lineno: INTEGER; VAR s: ARRAY OF CHAR);
BEGIN
  IF lineno>HIGH(text) THEN s:=''
  ELSE
```

```
    str.copy(s,text[lineno]);                                   (*2*)
  END;
END get;

PROCEDURE put(lineno: INTEGER; VAL s: ARRAY OF CHAR);
  VAR x: STRING;
BEGIN
  IF lineno>HIGH(text) THEN RESIZE(text,lineno+16) END;         (*3*)
  NEW(x,str.len(s)+1);                                          (*4*)
  str.copy(x,s);
END put;

PROCEDURE delete(lineno,lines: INTEGER);
  VAR i,n: INTEGER;
BEGIN
  IF lineno+lines>=LEN(text) THEN
    FOR i:=lineno TO HIGH(text) DO DISPOSE(text[i]) END;
  ELSE
    FOR i:=lineno TO lineno+lines-1 DO DISPOSE(text[i]) END;
    n:=lineno+lines;
    FOR i:=lineno TO HIGH(text) DO
      IF n<=HIGH(text) THEN text[i]^:=text[n]^;                 (*5*)
      ELSE NEW(text[i]);
      END;
      INC(n);
    END;
  END;
END delete;

BEGIN
  NEW(text);
END edText.
```

Примечания:

(*0*) Импорт модуля Strings с переименованием.
(*1*) Определение настраиваемых стандартных процедур
работы с памятью.
(*2*) Копирование строк.
(*3*) Увеличение числа элементов в динмассиве text.
(*4*) Выделение памяти для строки. Число элементов в строке
вычисляется как число элементов параметра плюс один
(для хранения символа конца строки - 0c).
(*5*) Копирование дескриптора строки.

Глава 5.2. Использование компилятора

Компилятор "mx" состоит из ядра и набора утилит. Ядро является конструтором для создания конечного продукта, каждая утилита реализует некоторый способ использования компилятора. В базовый набор постановки системы входит несколько таких утилит: пакетный компилятор "mx", турбо-компилятор "turbo2x" и утилита поддержки разработки "pm" (project manager).

Использование пакетного компилятора описывается в 5.2.2, турбо-компилятора - в 5.2.3. Все эти утилиты одинаковым образом настраиваются на окружение задачи (environment). Описание окружения приводится в 5.2.7. В 5.2.1 описываются соглашения об именовании файлов. Способы разработки новых компилирующих утилит описываются в 5.3.2.

Содержание этого раздела существенно зависит от операционной системы, в которой работает компилятор (см. описание утилит ОС Excelsior).

5.2.1. Имена модулей и файлов

При работе на Модуле-2 программисту приходится иметь дело с пятью сущностями:

- определяющие модули;
- реализующие (и программные) модули;
- симфайлы;
- реффайлы;
- кодофайлы.

Определяющий модуль содержит информацию о способе использования некоторых объектов (интерфейс), реализующий модуль определяет реализацию операций (функций). Программный модуль удобно рассматривать как реализующий модуль с пустым интерфейсом.

Для модуля с именем M рекомендуется текст определяющего модуля хранить в файле с именем M.d, а текст реализующего модуля в файле с именем M.m.

При компиляции определяющего модуля компилятор порождает так называемый симфайл, который содержит (в сжатой форме) информацию, необходимую клиентам модуля (то есть тем модулям, которые импортируют данный). Компилятор записывает симфайл для модуля с именем M в файл с именем M.sym.

При компиляции реализующего модуля компилятор порождает кодофайл (исполняемый образ модуля) и реффайл, который содержит информацию, необходимую отладчикам. Для модуля с именем M компилятор записывает кодофайл в файл с именем M.cod, а реффайл в файл с именем M.ref.

По умолчанию все файлы, которые порождает компилятор, записываются на текущую директорию (см. 5.2.7, если есть желание записывать файлы в другие места).

5.2.2. Пакетный режим

Использование утилиты "mx":

```
mx { имя_файла } [ дополнительные_аргументы ]
```

Компилятор последовательно транслирует файлы, имена которых заданы в списке параметров утилиты. Если имя файла не содержит расширителя, то к имени добавляется постфикс ".m".

В начале компиляции на терминал выдается строка, содержащая имя и версию компилятора и имя компилируемого файла:

```
Modula X  v0.7 /21-Mar-90/ [2] "a.m"
```

В квадратных скобках указана модель процессора.

При обнаружении ошибок компиляции на стандартный вывод выводится сообщение об ошибке, состоящее из пояснения типа ошибки и строки текста, в которой обнаружена ошибка. Позиция ошибки в строке помечается символом "$". Заметим, что символ "$" выставлен после ошибочной лексемы. Например:

```
Невидимый объект  -- "aaaa"
   5:   i:=aaaa$;
```

Компиляция модуля прекращается, если число ошибок больше некоторого предела (см. 5.1.4).

После завершения компиляции модуля на терминал выдается сообщение, содержащее информацию о числе строк, числе ошибок и времени, затраченном на компиляцию:

```
errors: 1  lines 7  time  01cpu + 01io
```

Если ошибки не обнаружены, то сообщение содержит информацию о числе строк, времени компиляции, а также имя порожденного файла и размер кода (только при компиляции реализующих модулей).

```
lines 7  time  01cpu + 01io  "a.cod"  1 words
```

Если компилятору задано несколько имен файлов, то по завершении компиляции компилятор выдает сообщение, содержащее суммарную статистику: число файлов, число строк, время компиляции, скорость компиляции (строк в минуту) и общий размер порожденного кода. Информация о неправильных модулях в статистику не включается.

```
files 2 lines 14 time cpu00:01 io00:01 speed 840 l/m  2 words
```

Если компилятор запущен откреплено, то строки,

содержащие имя компилятора и информацию о модулях, не выдаются.

5.2.3. Турбо-компилятор

Турбо-компилятор представляет собой утилиту, которая компилирует текст модуля, находящегося в буфере редактора. При этом полученные в результате компиляции симфайл, кодофайл и реффайл записываются на диск.

Турбо-компилятор запускается из редактора (см. "Утилиты ОС Excelsior", раздел 'ex') нажатием клавиш F0 и 'm'.

Если в тексте при компиляции обнаружены ошибки, компилятор сообщает о них последовательно. Сообщение о характере ошибки при этом появляется в служебной (информационной) строке, а курсор позиционируется в том месте текста модуля (после той лексемы), где произошла ошибка.

Переход к следующей ошибке осуществляется нажатием клавиш F0 и '+'.

После того, как исправлены все ошибки, можно запустить отлаживаемую программу и, убедившись в правильности ее работы, записать текст на диск. Таким образом, время на исполнение стандартного отладочного цикла редактирование-компиляция-запуск может быть заметно сокращено.

5.2.4. Перечень сообщений компилятора

Приводим сообщения компилятора с комментариями там, где они требуются.

5.2.4.1. Сообщения типа "Слишком много"

1) Переполнена хеш-таблица (слишком много имен)

Текущая реализация не умеет работать с таким количеством переменных, процедур и других именованных объектов.

2) Переполнение стека выражений!

Выражение не удалось вычислить на стеке в 7 элементов.

3) Слишком сложное условное выражение

Много написано между условными операторами. Аппаратное ограничение. Не бывает.

4) Слишком большой размер типа

Размер типа превышает maxInt.

5) Слишком много параметров

Параметров у процедуры больше, чем позволяет текущая реализация (>256).

6) Ограничение транслятора: слишком много

Процедур или переменных больше, чем допускает текущая версия.

7) Размах меток оператора выбора > 256

5.2.4.2. Сообщения типа "Ожидалось"

1) Неожиданный конец исходного текста!
   Ожидалось "END имя_модуля".

2) Ожидался идентификатор

3) Должно быть имя блока
   Должно быть имя процедуры или модуля.

4) Отсутствует 'h' после шестнадцатеричного

5) Незакрытая или слишком длинная строка!
   В конце строки текста не стоит ".

6) Ожидался символ
   Например, ";" или ")".

7) Ожидалось константное выражение
   Например, границы массива.

8) Ожидался оператор
   Как правило, вызвано мусором в тексте модуля.

9) Код записывается байтами! [0..0FFh]
   Ошибка в кодовой процедуре.

10) Некорректная информация в симфайле
   Начало было похоже на симфайл, а дальше...

11) Ошибка в заголовке симфайла
   Мусор в содержимом симфайла.

12) Ошибка в конструкторе типа

13) Ошибка в заголовке модуля
   Ошибка в самой первой строке программы.

14) Незакрытый комментарий, начавшийся в строке

5.2.4.3. Семантические ошибки

1) Невидимый объект
   Объект не описан или не проимпортирован.

2) Повторно объявлен

Такой объект уже был описан.

3) Рекурсивное определение объекта
Например, CONST A=A+1.

5.2.4.4. Ошибки при работе с типами

1) Недопустимое преобразование типа
Можно преобразовывать только в тип той же длины.

2) Типы несовместимы

3) Должен быть тип указателя

4) Скрытый тип должен быть однословным и не литерным

5) Должен быть тип
Идентификатор не является типом.

6) Должен быть скалярный тип

7) Должен быть простой (1 слово) тип

5.2.4.5. Ошибки в выражениях

1) Неправильное константное выражение
Выражение правильное, но не константное.

2) Неправильное выражение

3) Переполнение (исчерпание) в константном выражении
Выдается, например, при делении на 0.

5.2.4.6. Вызовы процедур и функций

1) Вызов процедуры в выражении
Вместо функции вызвана процедура.

2) Вызов функции в позиции оператора

3) Неправильное число параметров
Лишние или недостающие параметры в процедуре.

4) Это не процедура

5.2.4.7. Ошибки в версиях

1) Некорректная версия симфайла
Что-то устарело.

2) Конфликт версий (по времени компиляции)
См. раздел о конфликте версий.

5.2.4.8. Встретилось неожиданно (не на месте)

1) Непонятный знак игнорируется
Например, $ или %, @.

2) Должен быть массив
Попытка проиндексировать не массив ([после идентификатора).

3) Должна быть запись
Встретилась ".".

4) Ошибка в описаниях
Мусор до BEGIN или отсутствует BEGIN.

5) Недопустимо в определяющем модуле
Например, VAL-параметр или BEGIN.

6) EXIT вне LOOP'а
Скорее всего, лишний END внутри LOOP-цикла.

7) Такая метка уже была
Одинаковые метки в CASE-операторе.

8) Должен быть тип множества
Использование { не по делу.

9) RETURN можно писать только в процедуре
В теле модуля нельзя.

5.2.4.9. Прочие ошибки

1) Неправильный (неконстантный) или вырожденный отрезок
Отрезок в описании типа должен быть константным и правая граница больше левой.

2) Не реализовано
Не реализовано в компиляторе.

3) Должна быть переменная
Объект существует, но не является переменной.

4) Не обладает адресом
Выдается, например, при попытке присвоить значение константе.

5) Не обладает значением
Например, вложенная процедура.

6) Недопустимое использование формального типа
ARRAY OF можно использовать только при описании заголовка процедуры.

7) Переменная цикла должна быть локальной
Должна быть описана в минимальном объемлющем блоке.

8) Это не модуль
Попытка проимпортировать что-то другое (или из чего-то другого).

9) Выход за границы диапазона
Выход за границы массива или битсета.

10) Экспорт невозможен. Объект уже объявлен

11) Доступ к дескриптору только с ключом $U+ (unsafe)

12) Нереализованная процедура
Ошибка в FORWARD-описании.

13) Попытка подсунуть чужой симфайл
Симфайл не модуловский. Не бывает.

14) Повторный FORWARD

15) Недопустимое использование идентификатора модуля
Попытка проиндексировать имя модуля.

16) Неправильный синтаксис строки
Ошибка между "".

17) Разрешено только на уровне единицы компиляции

18) Недоступная RTS процедура
NEW, DISPOSE, RESIZE не определены.

19) Спецификатор VAL недопустим в описании процедурного типа

20) В CASE нужна хоть одна альтернатива

21) Переменная цикла не может быть VAR-параметром

22) Присваивание VAL-переменной (Только Для Чтения)

23) Разрешено только в определяющем модуле
Попытка описать скрытый тип или VAL-переменную.

24) Копирование динмассивов только с ключом $X+
Временно.

5.2.5. Порядок компиляции

Сначала компилируют определяющий модуль, если он есть, затем - реализующий и только потом программный.

Если программа состоит из нескольких модулей, то компилируют сначала все определяющие модули, а затем реализующие.

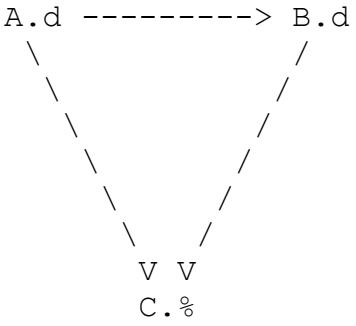
При компиляции необходимо следить за порядком импорта, а именно: чтобы скомпилировался модуль M, импортирующий какой-нибудь объект из модуля N, требуется симфайл N.sym. Из этого следует вывод, что определяющий модуль N должен быть скомпилирован раньше, чем определяющий модуль M. Кроме того, этот симфайл должен быть видим, т.е. лежать на директории, указанной в пути. Например, симфайлы всех библиотек принято хранить на директории /$v/sym, где $v - имя носителя.

Порядок компиляции реализующих модулей после того, как скомпилированы определяющие, значения не имеет. Это имеет важное методологическое значение. Так, скажем, один и тот же модуль, задействованный в большом количестве задач, может иметь разные реализации, которые можно заменять без последующей перекомпиляции этих задач.

5.2.6. О конфликте версий

При несоблюдении правильного порядка компиляции может произойти конфликт версий, возникающий в процессе компиляции или при запуске задачи.

Первое происходит в случаях, которые можно описать схемой:



Здесь стрелки изображают порядок импорта объектов, C.% - определяющий (C.d) либо реализующий (C.m) модуль.

Если сначала был скомпилирован модуль A.d, затем B.d, а потом по какой-то причине снова A.d, то при компиляции модуля C возникает конфликт версий симфайлов A.sym и B.sym.

Ситуация, в которой возникает конфликт версий кода при запуске задачи, такова: если модуль C импортирует объекты из модуля A, а порядок компиляции оказался следующим:

A.d C.m A.d A.m

, то при запуске задачи возникает конфликт версий кодофайлов C.cod и A.cod.

5.2.7. Среда компиляции

Среда компиляции состоит из набора строк, определяющих поиск симфайлов и файлов, содержащих исходный текст, директории на которые будут записаны симфайлы, реффайлы и кодофайлы и режимы компиляции. Компилятор при запуске настраивается на окружение:

"SYM" - определяет пути поиска симфайлов; состоит из имен директорий, разделенных пробелами;

"mxTEXT" - определяет пути поиска файлов исходных текстов; состоит из имен директорий, разделенных пробелами;

"mxOUT" - определяет имена директорий, на которые будут записываться выходные файлы компилятора; состоит из пар равенств вида образец = путь, разделенных символом ":". Если имя выходного файла не сопоставляется ни с одним из образцов, то файл будет записан на текущую директорию.

"mxERRLIM"- определяет макисмальное число ошибок компиляции;

"mxCPU" - определяет версию системы команд. Если окружение задачи не содержит строки "mxCPU", то компилятор порождает код для той модели процессора, на которой работает.

"mxFLAGS" - определяет режимы компиляции. Содержит последовательность пар символов, состоящих из литеры, определяющей режим компиляции, за которым следует символ "+" или "-". Символ "+" обозначает включение режима, "-" - выключение.

Строки с указанными именами могут быть заданы в окружении задачи (об окружении см. в разделе, посвященном пользовательской оболочке): или параметрами при запуске компилятора.

Примеры:

```
SYM       ". /sym /usr/sym"
mxTEXT    ". new_project"
```

```
mxOUT     "*.cod=my_bin: mx*.sym=/usr/sym"
mxFLAGS   "N+T-"
mxERRLIM  "8"
mxCPU     "2"
```

Глава 5.3. Реализация компилятора

5.3.1. Детали реализации

Рассматриваются тонкие места входного языка, которые не определены Сообщением [5]. Речь будет идти о совместимости типов.

5.3.1.1. Тип T1 совместим с типом T0, если истинно одно из следующих утверждений:

1) T1 описан, как T1=T0 (т.е. они идентичны);
2) T1 описан как диапазон T0;
3) T0 описан как диапазон T1;
4) T0 и T1 описаны как диапазоны одного и того же типа;
5) один из типов есть ADDRESS, а другой - тип INTEGER (или диапазон типа INTEGER) или тип указателя.

Заметим, что отношение совместимости симметрично.

5.3.1.2. Тип T1 обобщенно совместим с типом T0, если истинно одно из следующих утверждений:

1) типы T1 и T0 совместимы;
2) один из типов есть WORD, а другой - любой однословный тип;
3) T1 и T0 есть процедурный типы и они процедурно совместимы.

Отношение обобщенной совместимости симметрично.

5.3.1.3. Тип T1 совместим по присваиванию с типом T0, если истинно одно из следующих утверждений:

1) типы T1 и T0 обобщенно совместимы;
2) T1 и T0 есть типы массивов литер и длина (число элементов) массива T1 не меньше длины массива T2;
3) T1 есть массивов литер, а T2 константа типа CHAR;
4) T1 и T0 есть типы массивов, гибких массивов или динмассивов и хотя бы один из них является типов гибкого массива или динмассива и их базовые типы совместимы;

Отношение совместимости по присваиванию несимметрично.

Отличия от Сообщения [5]:

- в утверждении 2 по сообщению правым операндом может быть только цепочка (т.е. строковая константа);
- разрешены присваивания гибких массивов целиком; при исполнении проверяется, что размеры массивов равны, а для массивов литер, что число элементов левого операнды не меньше числа элементов правого операнда.

5.3.1.4. Тип T1 процедурно совместим с типом T0, если истинны ВСЕ следующие утверждения:

1) если T1 есть процедура-функция, то T2 тоже есть процедура-функция и типы результатов обобщенно совместимы;
2) число параметров у процедур одинаково;
3) соответствующие параметры имеют одинаковый способ передачи;
4) у параметров типа гибкий массив типы элементов обобщенно совместимы, типы остальных параметров обобщенно совместимы.

5.3.1.5. Передача параметров

Основное правило:

Тип параметра-переменной должен быть обобщенно совместимы с типом фактического параметра. Тип параметра-значения должен быть совместим по присваиванию с типом фактического параметра. Если тип формального параметра - гибкий массив слов, то тип фактического параметра должен быть массивом (любым), а типы элементов формального и фактического параметров должны быть совместимы.

Параметры-последовательности:

Вместо формального параметра-последовательности может быть подставлена последовательность параметров любой длины (возможно пустая) или один фактический параметр, являющийся формальным параметром-последовательностью.

В первом случае тип каждого фактического параметра должен быть обобщенно совместим или совместим по присваиванию с типом последовательности (в зависимости от способа передачи). Во втором случае способ передачи последовательностей должен совпадать, а типы быть обобщенно своместимы.

Исключения:

1) Если тип формального параметра - массив слов, то фактическим параметром может быть объект любого структурного типа той же длины.

2) Если тип формального параметра - гибкий массив слов, то фактическим параметром может быть объект любого структурного типа.

3) Если тип последовательности есть WORD, то типы фактических параметров могут быть любыми. При этом если тип фактического параметра есть тип записи, массива, гибкого массива или динмассива, то в качестве параметра будет передан адрес структуры.

5.3.2. Структура компилятора

5.3.2.1. Интерфейс компилятора

Тексты модулей приводятся в качестве примера и могут не полностью совпадать с библиотеками текущей версии ОС и компилятора.

Модуль coolDefs определяет представление операций ввода/вывода.

```
DEFINITION MODULE coolDefs; (* Ned 06-Jan-90. (c) KRONOS *)

IMPORT  SYSTEM;

CONST -- виды модуля/файла
  def    = 0;   -- определяющий модуль
  imp    = 1;   -- реализующий модуль
  main   = 2;   -- программный модуль
  text   = 3;   -- текст
  sym_in = 4;   -- входной симфайл
  sym_ou = 5;   -- выходной симфайл
  ref    = 6;   -- реффайл
  code   = 7;   -- кодофайл

TYPE
  PRINT  = PROCEDURE (ARRAY OF CHAR, SEQ SYSTEM.WORD);
  io_ptr = POINTER TO io_rec;
  io_rec = RECORD
             kind : INTEGER;             -- вид модуля/файла
             doio : PROCEDURE (io_ptr); -- операция в/в
             done : BOOLEAN;             -- результат операции
             print: PRINT;               -- для сообщений
             buf  : STRING;              -- буфер в/в
             len  : INTEGER;             -- длина в/в
             exts : SYSTEM.ADDRESS;      -- расширение
           END;

TYPE
  INI    = PROCEDURE (VAR    io_ptr,     -- дескриптор в/в
                       ARRAY OF CHAR,    -- имя модуля/файла
                       INTEGER,          -- вид модуля/файла
                       PRINT             -- для сообщений
                      );
  EXI    = PROCEDURE (VAR io_ptr);
  ERROR  = PROCEDURE (
                       INTEGER,          -- номер строки
                       INTEGER,          -- позиция в строке
                       ARRAY OF CHAR,    -- строка текста
                       ARRAY OF CHAR,    -- формат
                       SEQ SYSTEM.WORD   -- аргументы
                       );
END coolDefs.
```

Модуль определяет тип дескриптора ввода/вывода, виды модулей/файлов, вид операций открытия/закрытия дескриптора и сообщения об ошибках.

В процедуре сообщения об ошибках первые два параметра определяют позицию ошибки в тексте, а последние два - текст поясняющего сообщения.

Модуль coolIO реализует операции файлового ввода/вывода.

```
DEFINITION MODULE coolIO; (* Ned 04-Mar-90. (c) KRONOS *)

IMPORT  comp: coolDefs;

PROCEDURE ini(VAR io: comp.io_ptr;
                name: ARRAY OF CHAR;
                unit: INTEGER;
               print: comp.PRINT);

PROCEDURE exi(VAR io: comp.io_ptr);

PROCEDURE set(text_path,sym_path,out: ARRAY OF CHAR; print:
comp.PRINT);

PROCEDURE dispose;

END coolIO.
```

Модуль mxPars реализует операцию compile, которая параметризована операциями ввода/вывода.

```
DEFINITION MODULE mxPars; (* Ned 06-Dec-87. (c) KRONOS *)

IMPORT SYSTEM;

IMPORT  comp: defCompiler;

PROCEDURE compile(text : comp.io_ptr;
                  ini  : comp.INI;
                  exi  : comp.EXI;
                  error: comp.ERROR;
                  print: comp.PRINT;
                  opts : BITSET;
                  cpu  : INTEGER;
                  );

PROCEDURE fault(format: ARRAY OF CHAR; SEQ args: SYSTEM.WORD);

END mxPars.
```

Процедура compile в качестве параметров получает открытый дескриптор файла входного текста, процедуры открытия/закрытия дескриптора, процедуры вывода ошибок и сообщений, множество режимов компилятора и номер версии системы команд.

Глава 5.4. Средства отладки и визуализации

К средствам отладки программ относятся историк, выдающий историю неудавшегося процесса в терминах текста программы, и визуализатор кода и других атрибутов программы.

5.4.1. Визуализация M-кода

Визуализацию кода осуществляет утилита vx.

Для работы утилиты необходим, как минимум, кодофайл. При этом можно узнать следующее:

- коды процедур (имена процедур недоступны);
- строковый пул;
- список внешних и, при необходимости, прогуляться по дереву импорта;
- мультиглобалы;
- общую информацию о модуле (версия компилятора, время компиляции и т.д.).

Кроме того, при наличии реффайла утилита выдает информацию "по полной программе":

- описание процедуры: имя, параметры, локалы;
- типы;
- глобалы: имя, тип;
- структурные константы.

Подробное описание утилиты vx читайте в справочнике по утилитам.

5.4.2. Посмертный историк

Историю неудавшегося процесса выдает утилита hi.

Подробное описание утилиты hi читайте в справочнике по утилитам.

5.4.3. Симфайлы, кодофайлы и реффайлы

Этот пункт пока не написан. Возможно, он появится не скоро, или даже вовсе никогда не появится, потому что информация, которую ему следовало бы содержать, по мнению разработчика, является делом текущей версии компилятора и подвержена периодическим изменениям. Как следствие, эта информация уже изложена в соответствующих "холодных" (cool) библиотеках.

Читайте о симфайлах в комментарии к библиотеке coolSym.

Читайте о кодофайлах в комментарии к библиотеке visCode.

Читайте о реффайлах в комментарии к библиотеке xRef.

Глава 5.5. Еще о Модула-X компиляторе

Предисловие соавтора

Этот раздел написан А.Недорей (Ned), автором ныне здравствующего Модула-компилятора (mx) вскоре после создания последнего. Первоначально это задумывалось как статья, не предназначающаяся для публикации в официальных изданиях. Поэтому я рада, что изложенная в ней информация впервые увидит свет на страницах нашей книги. Этот материал адресуется разработчикам компиляторов, отладчиков и комплексаторов (?), а также посмертным историкам.

ВВЕДЕНИЕ
В котором автор попробует понять, зачем ему понадобилось писать еще один Модула-2 компилятор

Причины (как обычно) бывают субъективные и объективные, причем важными являются только субъективные (в данном случае - ровно одна субъективная причина), а объективные предназначены для того, чтобы убедить себя и других в необходимости делать то, что хочется. Так вот, субъективная причина была очень проста: хотелось написать компилятор, который можно развивать, который послужит основой для других разработок.

Теперь перечислим те объективные соображения, которые послужили толчком к началу разработки компилятора. Они приводятся в некотором случайном порядке (не по важности).

1. Любое программное обеспечение морально устаревает, принципы его построения, интерфейсы с ОС, СП и пользователем с какого-то момента не соответствует изменившимся взглядам разработчиков и пользователей.

2. Если два года назад (когда был написан наш предыдущий компилятор, далее - m2) проблемы переносимости нас не волновали (хоть бы для Кроносов что-нибудь написать), то теперь мы собираемся переносить свое ПО на все 32-разрядные микропроцессоры, и в том числе на разные Кроносы (а система команд для кристалла (4.X) достаточно существенно отличается от системы команд семейства 2.X).

3. При разработке и эксплуатации компилятора m2 входной язык был расширен, часть этих изменений стали стандартными, часть в процессе использования были признаны ненужными. Разработка нового компилятора позволяет реализовать принятые расширения естественно (без ограничений), так как набор расширений известен с начала разработки.

4. И так далее.

КОНЕЦ Введения.

ПРИМЕЧАНИЕ:

Далее в тексте используется терминология, введенная в [5,6], например, иннерфейс, экстерфейс [5], фильтр [6].

ГЛАВА ПЕРВАЯ
В которой изложены особенности входного языка

Особенности входного языка изложены в документации (см).

ГЛАВА ВТОРАЯ
(Из трех взглядов).
В которой автор показывает mx со всех трех сторон.

ВЗГЛЯД 1. mx со стороны пользователя

Вид с этой стороны не должен вызвать интереса у людей, знакомых с m2. Как и m2, mx един в нескольких лицах:

- пакетная версия (утилита mx);
- турбо-компилятор (фильтр turbo2x);
- турбо-тестер (фильтр tester2x);
- и так далее.

Пакетная версия читает исходный текст из файла и пишет результаты компиляции (кодофайл и симфайл) на диск.

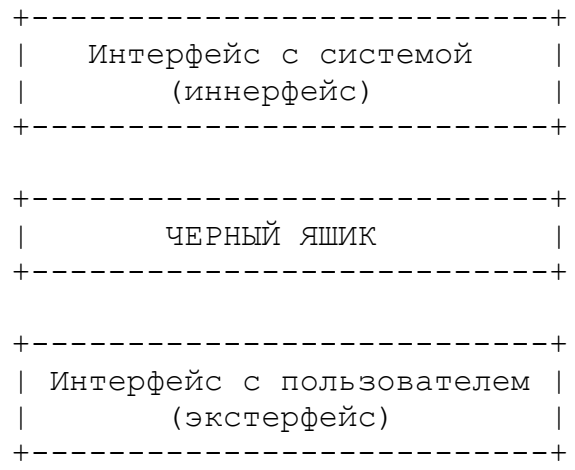
Фильтр turbo2x читает текст из буфера редактора, а результаты пишет в файлы.

Турбо-тестер (см. главу 3) используется для отладки компилятора. Он читает текст (множество единиц компиляции) из буфера редактора, пытается выполнить те тесты, которые скомпилировались, сравнивает результаты компиляции и запуска с тем, что должно было быть, и сообщает о расхождениях. Кодофайлы и симфайлы хранятся в оперативной памяти и уничтожаются после выполнения теста.

ВЗГЛЯД 2. mx со стороны разработчика системы программирования

Все утилиты, описанные в первом взгляде, пользуются некоторой библиотекой, которая позволяет просто писать разные (с точки зрения пользователя) версии Модула-компиляторов. Собственно говоря, эта библиотека и есть mx, а все утилиты реализуют методы ее использования. Внутри данного взгляда под словом "пользователь" мы будем понимать пользователя этой "библиотеки" (т.е. разработчика очередного "пускача").

Библиотека mx - это слоеный пирог (терминология из [5]):



Интерфейс с системой определяет модуль mxSystem. Этот модуль предоставляет остальному компилятору все возможности, которые нужны от системы, кроме операций чтения исходного текста, записи выходных файлов и выдачи сообщений об ошибках. Операции ввода/вывода определяются каждой компилирующей утилитой.

Интерфейс с пользователем определяют модули defCompiler, mxPars и mxIO. Модуль defCompiler определяет представление операций ввода/вывода. Модуль mxPars реализует операцию compile, которая параметризована операциями ввода/вывода. Модуль mxIO предоставляет стандартные реализации операций ввода/вывода.

Продемонстрируем использование этих модулей при написании пакетного компилятора:

```
MODULE компилятор;

IMPORT  comp: defCompiler;
IMPORT    mx: mxPars;
IMPORT    io: mxIO;

VAR
  text: io.io_ptr;
  name: ARRAY [0..255] OF CHAR;

BEGIN
  инициализация;
  WHILE есть_что_компилировать DO
    следующее_имя_файла(name);                  (*0*)
    io.ini(text,name,comp.text,mx.fault);       (*1*)
    IF text^.done THEN                          (*2*)
      mx.compile(text,io.ini,io.exi,....);      (*3*)
    END;
    io.exi(text);                               (*4*)
  END;
END компилятор.

(*0*)  Выбор следующего имени файла;
```

```
(*1*)  Начало работы с текстом из файла с именем name;
(*2*)  Если файл успешно открылся;
(*3*)  Компиляция;
(*4*)  Конец работы с текстом.
```

ВЗГЛЯД 3. mx изнутри

Теперь рассмотрим то, что находится в черном ящике для взгляда 2. Компилятор состоит из 5 модулей:

```
mxScan  -- лексический анализ;
mxObj   -- таблица объектов, видимость, импорт/экспорт;
mxSym   -- чтение/запись симфайлов;
mxCmd   -- примитивы генерация кода;
mxGen   -- генерация кода;
mxPars  -- синтаксический анализ.
```

Модуль	Число строк		размер кода
	определение	реализация	(в байтах)
mxScan	115	646	3624
mxObj	279	679	2988
mxSym	13	602	3468
mxCmd	215	1780	7184
mxGen	254	2177	11084
mxPars	39	1486	7732
ИТОГО	915	7370	36080

Все генерация сосредоточена в модулях mxGen и mxCmd. Модуль mxCmd содержит примитивы генерации (аппаратно-зависимые), а модуль mxGen языково-зависимую генерацию. При генерации делается щелевая оптимизация (аналогично m2) и оптимизация структур управления. На тестах со сложными управляющими структурами оптимизация управления может дать существенное ускорение. Так, известный тест "dhrystone", скомпилированный mx, работает на 10% быстрее по сравнению с m2.

Интерфейс с модулем mxGen выдержан в машинно-независимых терминах, что позволяет надеяться на простоту переноса компилятора на другие архитектуры.

ГЛАВА ТРЕТЬЯ
В которой описывается процесс отладки компилятора

Для отладки компилятора использовался турбо-тестер (упомянутый в предыдущей главе) и набор тестов [7], дополненный автором. Набор тестов содержит тесты, проверяющие корректность реализации различных конструкций языка, тесты на

граничные условия, тесты на исключительные ситуации и диагностику ошибок.

Примеры:

Следующие два теста проверяют корректность реализации следующего правила из описания языка:

"Тип T1 может использоваться в описании типа указателя T, тесктуально предшествующего описанию T1, если T и T1 описаны в одном блоке."

Первый тест проверяет то, что такое описание типа указателей разрешено; второй - то, что компилятор проверяет ограничение правила, то есть что оба типа должны быть описаны в одном блоке.

```
$x
MODULE T6P7D4;  (* Conformance *)

FROM InOut IMPORT WriteString;

TYPE T  = POINTER TO T1;

TYPE T1 = RECORD
            X: INTEGER;
            Y: BOOLEAN
          END;

BEGIN
   WriteString('PASS...6.7-4',12)
END T6P7D4.

$
MODULE T6P7D5;  (* Deviance *)

FROM InOut IMPORT WriteString;

TYPE P = POINTER TO T;

PROCEDURE p;
  TYPE T = BOOLEAN;
BEGIN
   WriteString('DEVIATES...6.7-5',16)
END p;

BEGIN
  p;
END T6P7D5.
```

Специальные строки в тексте, начинающиеся символом '$', указывают турбо-тестеру, что должна дать проверка данным тестом.

```
УправляющаяСтрока = "$" [ "c" | "x" ]
```

Символ "x" указывает, что тест должен быть скомпилирован и удачно исполнен; символ "c" - что тест должен быть скомпилирован и неудачно исполнен (исключительная ситуация при исполнении); отсутствие символов указывает на то, что компилятор должен обнаружить ошибки в тесте. Турбо-тестер выдает список имен тестов, при проверке которых результат не соответствует требуемому.

Следующий тест проверяет, ведется ли контроль границ индексов при исполнении программы.

```
$c
MODULE T8P1D5;

FROM InOut IMPORT WriteString;

VAR a: ARRAY [2..5] OF INTEGER;
    i: INTEGER;

BEGIN
  i:=1;
  A[i]:=1;
  WriteString('ERROR IS NOT DETECTED...8.1-5',30)
END T8P1D5.
```

Турбо-тестер позволяет исполнить полный (около 250 штук) или частичный набор тестов после каждого изменения компилятора и проверить корректность этого изменения и то, что изменение не повлияло на реализацию других конструкций языка. Благодаря наличию системы тестов и мощной поддержки тестирования удалось очень быстро получить надежный и стабильный компилятор.

ГЛАВА ЧЕТВЕРТАЯ
В которой автор говорит всем,
кто поддерживал его в этой работе
и отдельно Д.Г.Фон-Дер-Флаассу,
принимавшему активное участие в разработке алгоритмов
оптимизации структур управления,

БОЛЬШОЕ СПАСИБО

Л И Т Е Р А Т У Р А

1. N.Wirth Programming in Modula-2. Third, Corrected Edition. Springer-Verlag. Berlin, Heidelberg, New-York, 1985.

2. N.Wirth. A fast and compact compiler for Modula-2. ETH, Zurich, July 1985, 64, pp. 1-22.

3. N.Wirth. The Programming Language Oberon.

4. Second Working Draft Modula-2 Standard. Document D103. BSI Modula-2 Standartisation Working Group (IST/5/13).

5. Н.Вирт Программирование на языке Модула-2. Москва, Мир, 1987.

6. Кузнецов Д.Н., Недоря А.Е., Тарасов Е.В., Филиппов В.Э. КРОНОС - автоматизированное рабочее место профессионального программиста. В сб.: Автоматизированное рабочее место программиста. Новосибирск, 1988.

7. Кузнецов Д.Н., Недоря А.Е. Турбо-компиляция. Что дальше? (рукопись).

Часть 7. SHELL: ПОЛЬЗОВАТЕЛЬСКАЯ ОБОЛОЧКА ОС

Предисловие соавтора

Эта часть написана в соавторстве с А.Хапугиным (Hady), реализовавшим ныне здравствующую пользовательскую оболочку, а также интерпретатор командныхх файлов mshell.

Соавтор

Пользовательская оболочка системы shell - основное средство общения с операционной системой. С помощью shell можно изменить обстановку работы как себе, так и окружающим, запустить или прекратить задачу, а также узнать кое-что о состоянии ОС.

Для работы с пользовательской оболочкой достаточно этого описания. Но если читатель хочет продвинуться дальше, понять смысл средств и ограничений shell, мы посоветуем ему ознакомиться с комментариями к библиотекам Shell и exeCut.

Глава 7.1. Окружение и его параметры

> Не имея достаточно средств и будучи отфутболенным различными бюрократами, она у меня пока не полностью автоматизирована. Вопросы задаются устным образом, и я их печатаю и ввожу к ей внутрь, довожу, так сказать, до ейного сведения. Отвечание ейное, опять же через неполную автоматизацию, печатаю снова я.
>
> А. и Б. Стругацкие. Сказка о Тройке

7.1.1. Приглашение к вводу команды

Большую часть времени, занимаемого утилитой у компьютера, она тратит на ввод команд с клавиатуры и попытку их исполнить. Выдавая на экран строчку-приглашение, утилита приглашает пользователя ввести строку-команду.

Стандартное приглашение состоит из текущего времени и текущей директории и выглядит приблизительно так:

```
12:35 dir $
```

Если оно не подходит вам, его легко поменять, набрав PROMPT="приятная_глазу_строка". Теперь в качестве приглашениея будет выдаваться

приятная_глазу_строка

В нее могут быть включены последовательности из двух символов %/ и %t, которые перед выводом строки-приглашения на экран заменяются на имя текущей директории и на текущее время соответственно.

Например, было указано приглашение

PROMPT="Время: %t Директория: %/ >> "

Если сейчас 12 часов 35 минут, а текущая директория - "/usr/John/games", то строка-приглашение будет выглядеть теперь следующим образом:

Время: 12:35 директория: games >>

Впрочем, такое длинное приглашения не очень удобно, поэтому будем считать, что установлено приглашение "]", и именно им будем предварять изображаемые далее командные строки.

Конец 7.1.1.

Установив новое приглашение, вы, сами того не зная, изменили значение одного из параметров своего окружения, а именно параметра PROMPT.

Каждая задача запускается в некотором окружении, содержащем необходимую (по крайней мере, не вредную) для ее исполнения информацию. Окружение задачи (а shell - тоже задача) состоит из набора ПАРАМЕТРОВ, у которых есть имя ("PROMPT") и значение (теперь "]").

Откуда берется окружение вновь запускаемой задачи? Его создает запускающая задача (в нашем случае shell) при запуске. Получает она его простым копированием почти всех (подробности см. в п.7.3.6) параметров своего собственного окружения.

7.1.2. Изменение параметров окружения

С клавиатуры можно задать или изменить (если уже задано) значение любого параметра, набрав строку "ИМЯ=тело", и в окружении вашей shell параметр ИМЯ примет значение "тело".

Обратите внимание: если "тело" не содержит пробелов, то брать его в кавычки необязательно.

Удалить параметр из окружения еще проще: достаточно сказать "ИМЯ=", и параметр с именем "ИМЯ" исчезнет.

7.1.3. Изменение параметров чужого окружения

Можно сменить окружение не только вашей shell, но и любой другой задачи. Для этого перед именем параметра надо поставить номер задачи, у которой требуется сменить параметр, и отделить

этот номер от имени точкой, например:

```
]69.PROMPT="ex.shell>>"
```

Вместо номера годится и имя уже существующего параметра, если его значение - запись какого-нибудь целого числа, как, например, у параметров FATHER, TASK или BASE. Для этого нужно поставить имя этого параметра вместо номера и предпослать ему символ "$, например:

```
]$FATHER.INFO="кафолик недорезанный"
```

Такая возможность дает, естественно, почву для мелкого хулиганства, но есть у нее и серьезные применения, о которых - в главе про командные файлы.

Следует при этом помнить, что если номер указан неправильно, или задачи с таким номером не существует, то вся конструкция "НОМЕР.ИМЯ" будет проинтерпретирована как имя нового параметра.

7.1.4. О приватных параметрах

Есть еще одна маленькая тонкость: если перед именем (или номером задачи и именем, если речь идет о чужом окружении) параметра поставить ".", то эта точка не войдет в имя получившегося параметра, а станет признаком приватности параметра: такой параметр не будет копироваться в окружения задач, запущенных вашей shell. А если вам все-таки хочется начать имя параметра с точки, то поставьте перед ней символ "\".

7.1.5. Имя параметра вместо его значения

Если требуется передать значение параметра в качестве аргумента какой-либо команде, вместо значения можно написать имя параметра, предварив его символом "$".

При выполнении любых действий shell заменяет в строке аргументов действия все комбинации вида "$ИМЯ" на значения параметров с именем "ИМЯ", если они есть в окружении, и только потом исполняет действие.

Глава 7.2. Команды shell

Кроме параметра "PROMPT", в окружении вашей shell может быть (и обычно бывает) много других параметров. Некоторые из них заметным образом влияют на поведение shell, а некоторые отражают ее состояние. Значения основной части параметров демонстрирует команда "set".

7.2.1. Как узнать текущие значения параметров (set)

По команде "set" shell печатает на экране состояния тех параметров, которые считает существенными. Команда

```
]set
```

вызовет появление на экране следующей информации:

```
 USER       "Iam"
 HOME       "/usr/Iam"
 NAME       "shell"
 TASK       "155"
 FATHER     "66"
 BASE       "155"
 SHELL      "shell"
 STK        "8"
 CEMETRY    "1"
 CD         "/usr/Iam/my_dir"
 BIN        ". /bin /usr/bin"
 SYM        ". /sym /usr/sym"
 ETC        ". /etc /usr/etc"
 REF        ". /ref /usr/ref"
 TTY        "/dev/tty2"
 KEY        "/dev/key2"
 PROMPT     "%t %/ $ "
 INSERT     "1"
 ECHO       "0"
 BELL       "1"
```

Еще раз заметим, что это не все параметры вашего окружения, наверняка есть еще и другие. Можно узнать и их значения. Наберите

```
]set S*
```

```
 SYM        ". /sym /usr/sym"
 SCR        "/dev/bmap"
 STK        "4"
 SHELL      "mshell"
```

Можно заметить, что кроме того, что напечатаны параметры,

имена которых начинаются на "S", появился еще один параметр с именем "SCR".

Аргумент команды set "S*" использован как регулярный образец, и команда напечатала все параметры, чьи имена сопоставились с этим образцом. Если учесть, что с образцом "*" сопоставляются все строки, то ясно, что посмотреть совсем все параметры можно, набрав

```
]set *

 FATHER     "66"
 BASE       "48"
 TASK       "48"
 ARGS       ">t"
 NAME       "t"
 HOME       "/usr/hady"
 USER       "hady"
 REF        ". /ref /usr/ref"
 SYM        ". /sym /usr/sym"
 PROMPT     "%t %/ $ "
 BELL       "1"
 LP         "/dev/lp0"
 CPD        "/dev/mou0"
 MSG        "/dev/LATIN"
 SCR        "/dev/bmap"
 ETC        ". /etc /usr/etc"
 BIN        ". /bin /usr/bin"
 STK        "4"
 ECHO       "0"
 SHELL      "mshell"
 KEY        "/dev/key0"
 TTY        "/dev/tty0"
 CD         "/usr/hady/sh/doc"
```

Конец 7.2.1.

Ни команда "set", ни модификации окружения не приводят к запуску каких-либо задач, эти команды - внутреннее дело самой shell. Есть еще несколько команд, чье исполнение полностью находится в компетенции shell. Остановимся на них подробнее.

7.2.2. Информация о задачах (ps)

По команде "ps" shell выдает на экран некоторую информацию о запущенных в настоящий момент задачах, например:

```
] ps

0153    run  ex            shell.doc
0066    run  shell
```

Первая колонка - номера задач. Номер, в отличие от имени

задачи, уникален, и потому более удобен для работы с задачей (прекращение, удаление, история завершения).

Вторая колонка - состояние задачи. Кроме "run" (запущена) задача может находиться в состояниях "stop" (остановлена), "load" (загружена) или еще каком-нибудь.

Третья колонка - имена задач.

Четвертая - некая дополнительная информация. В приведенном примере видно, что редактор "ex" сейчас занят файлом "shell.doc", содержимое которого у вас перед глазами. Заметим, что эта дополнительная информация - всего лишь строчка окружения задач с именем "INFO". Попробуйте набрать

```
] .INFO="кафолик недорезанный"
```

а потом спросить "ps":

```
] ps
```

и вы убедитесь в том, что программы тоже могут грубить:

```
0153    run  ex            shell.doc
0066    run  shell         кафолик недорезанный
```

У команды "ps" могут быть модификаторы "-tty", "-usr", "-mem", "-l", "-all".

7.2.2.1. Модификатор usr

Наберите

```
]ps -usr
```

```
0153    run  Iam   soft   ex            shell.doc
0066    run  Iam   soft   shell
```

Появились две новые колонки: имена пользователя и его группы.

7.2.2.2. Модификатор tty

```
]ps -tty
```

Модификатор "tty" добавит колонку с информацией о терминале, с которого запущена задача.

7.2.2.3. Модификатор mem

```
]ps -mem
```

Модификатор mem добавит информацию о памяти, занятой задачей.

7.2.2.4. Модификатор l

Модификатор "l" заменяет все три перечисленных модификатора:

```
] ps -l

 0153  run  Iam  soft  62K+0624  /dev/tty2  ex     shell.doc
 0066  run  Iam  soft  22K+0316  /dev/tty2  shell
```

7.2.2.5. Модификатор all

Обратите внимание: все показанные задачи запущены одним пользователем - вами, как будто других на машине нет. Может, это и так, но не всегда. Просто вам показывают НЕ ВСЕ задачи, а только запущенные с вашим пользовательским кодом. Чтобы увидеть ВСЕ задачи, используйте модификатор "all":

```
] ps -all

...TASKS...
0153    run  ex            shell.doc
0066    run  shell
0015  + run  login
0012  + run  DK6fd4110
0010  + run  TT6FCT
0009  + run  SIq6
0008  + run  CCDqMOUSE
0007  + run  DK6qSCSI
0006  + run  tmspo
0005  + run  LEXICON
0002  + run  TT6FCT
0001  + run  DK6dq615
```

Задач стало больше! Теперь это действительно все задачи. Модификатор "-all" можно применять в комбинации с другими для получения расширенной информации. Например,

```
] ps -all -mem
...TASKS...
0153    run   62K+0624  ex            shell.doc
0066    run   22K+0316  shell
0015  + run   13K+0976  login
0012  + run    7K+0476  DK6fd4110
0010  + run    9K+0936  TT6FCT
0009  + run    9K+0936  SIq6
0008  + run   11K+0496  CCDqMOUSE
0007  + run    7K+0936  DK6qSCSI
0006  + run   13K+0976  tmspo
0005  + run   15K+0352  LEXICON
0002  + run    8K+0920  TT6FCT
0001  + run    5K+0132  DK6dq615
```

ясно показывает, что больше всех памяти отъел редактор с файлом "shell.doc", а что поделаешь?

В конце строки можно указать номера задач, информацию о которых вы хотите получить. Если модификатор all отсутствует, вы получите информацию только об указанных задачах.

Символ "+" сразу после номера задачи обозначает, что эта задача сейчас находится в режиме привилегированного доступа. Этот режим позволяет иногда обойти ограничения, связанные с защитой файлов. Перейти в него можно с помощью команды "su".

7.2.3. Привилегированный доступ (su и us)

Команда "su" используется для переведения вашего shell'а в режим привилегированного доступа. Чтобы добраться до защищенных от вас файлов, наберите "su":

```
] su
```

Вот незадача! Требуют пароль:

```
password:
```

Если вы знаете его - введите, и тогда вы, работая под своим именем, сможете попользоваться привилегированным доступом. Если не знаете - не сможете. А если вы администратор системы, никто не помешает вам сделать в своем пользовательском коде специальную пометку, которая избавит вас от необходимости знать и набирать пароль: у вас его даже не спросят.

Поработав в привилегированном режиме, не забудьте от него отказаться - так спокойнее. Это делается командой "us".

7.2.4. Как гулять по директориям (cd)

> Через несколько поколений картофелю наскучил оседлый образ жизни, он самостоятельно выкопался и перешел к кочевому состоянию.
>
> С.Лем. Звездные дневники Ийона Тихого

Иногда возникает желание (или даже необходимость) сменить текущую директорию. Сделать это просто:

```
my_dir] cd my_another_dir
```

```
my_another_dir]
```

Этот номер пройдет гладко, если на директории my_dir действительно имеется директория my_another_dir. Если ее нет, то вам намекнут на ошибку:

```
my_dir] cd my_another_dir
#no such entry: change_dir("my_another_dir")
```

С помощью этой команды можно узнать и имя текущей директории. Достаточно набрать просто "cd":

```
my_dir] cd
#current directory: /usr/Iam/my_dir
```

Здесь нелишне упомянуть, что имя текущей директории хранится в окружении под именем "CD" (этот параметр показывает команда "set"). Более того: команда "cd any_name" полностью эквивалентна команде "CD=any_name".

7.2.5. Показ распределения памяти (mem)

Команда "mem" демонстрирует распределение оперативной памяти компьютера:

```
] mem

...MEMORY TOTAL...
   available  1024K+0000
   core        216K+0472
   free        356K+0068
   busy        451K+0484
```

Строчка "available" обозначает полный объем памяти, доступной системе, "core" - объем памяти, занятой системой, "free" - объем свободной памяти, а "busy" - объем памяти, занятой всеми задачами.

7.2.6. История задачи (his)

> Откинув уши на плечи, он рассказал мне историю своих соотечественников.
>
> С.Лем. Звездные дневники Ийона Тихого

Команда "his" печатает на экране историю последней неудачно закончившейся задачи. Если таковых еще не было, на экран выдается

```
#no history
```

С помощью команды his можно узнать историю любой задачи, еще находящейся в памяти, даже если она не завершена. Для этого достаточно указать этой команде номер нужной задачи.

Замечание. Параметр окружения HISTORY хранит историю последней неудачно завершившейся задачи или строчку "#no history", если задачи не было. При вызове без указания номера задачи команда просто распечатывает содержимое параметра.

7.2.7. Задержка на несколько секунд (delay)

Команда

```
] delay N
```

, где N - целое число, приведет к тому, что shell просто ничего не делает N секунд. Такое действие может показаться странным, но оно бывает полезным в некоторых "хитрых" случаях администратору системы.

Администратору системы будут полезны и следующие две команды: "mount" и "unmount".

7.2.8. Монтирование дисков (mount,unmount)

Команда "mount" позволяет изменить файловое дерево методом монтирования нового диска вместо какой-нибудь директории. Например,

```
] mount /mnt /dev/fd0
```

приводит к тому, что если устройство /dev/fd0 существует, и если оно - диск с нормальной файловой системой, а также существует директория /mnt (и если она доступна пользователю), то после исполнения вышеозначенного действия вместо директории /mnt в файловом дереве будет корневая директория носителя, находящегося на устройстве /dev/fd0.

Правда, при таком способе монтирования на этот диск записать ничего не удастся: он защищен системой от записи. При желании писать на него после монтирования нужно добавить модификатор "-ro":

```
] mount /mnt /dev/fd0 -ro
```

Демонтировать диск и получить, таким образом, доступ к содержимому директории /mnt можно командой "unmount":

```
] unmount /mnt
```

Замечание. Вообще-то в системе есть специальная утилита mou для монтирования и демонтирования дисков. Программисты обычно пользуются ею, потому что она удобней и много чего умеет.

7.2.9. Управление задачами (stop, kill, wait)

> - ... И я, старший здесь, нарушая законы и правила, первый говорю: смерть!
>
> Рядовые члены Тройки вздрогнули и разом заговорили.
>
> - Которого? - с готовностью спросил Хлебовводов, понявший, по-видимому, только последнее слово.
>
> А. и Б. Стругацкие. Сказка о Тройке

Команды управления могут быть названы таковыми с некоторой натяжкой, поскольку все управление сводится к прекращению (stop), удалению (kill) или ожиданию окончания (wait) задачи. Команды запускаются одинаково:

```
] wait   <номера_задач>
```

дождаться окончания задач с указанными номерами. Это ожидание, если оно стало слишком тягостным, может быть в большинстве случаев прервано с клавиатуры нажатием CTRL_C.

```
] stop   <номера_задач>
```

- прекратить исполнение задач с указанными номерами, не выгружая их из памяти. После выполнения этой команды можно посмотреть историю задачи.

```
] kill   <номера_задач>
```

- прекратить исполнение задач с указанными номерами и выгрузить их из памяти. Если задача уже была остановлена, то команда только выгружает ее из памяти. После этого от задачи не остается уже ничего, даже ее трупа.

Номера задач в строке аргументов перебираются до тех пор, пока не встретится конец строки или неправильный номер.

Заметим, что отсутствующую задачу можно считать нормально законченной, следовательно, все три команды не считают ее отсутствие ошибкой.

7.2.10. Завершение работы с shell

Команда bye заканчивает работу с пользовательской оболочкой. После набора команды shell исчезает.

Конец 7.2.10.

Итак, мы познакомились с окружением shell, его параметрами, которые можно изменять как явно, так и с помощью команд. Перечислим еще раз все команды shell:

set - показ текущих значений параметров;
cd - смена текущей директории;
ps - показ информации о задачах;
mount, unmount - монтирование и демонтирование носителя;
his - показ истории задачи;
mem - показ распределения памяти;
kill - удаление задачи;
wait - ожидание завершения задачи;
stop - остановка исполнения задачи;
delay - задержка на несколько секунд;
su,us - переход к привилегированному доступу и обратно;
bye - завершение работы с shell.

Глава 7.3. Запуск задач

Если введенная строка не является командой shell, то shell интерпретирует ее как имя запускаемой задачи и пытается запустить эту задачу.

7.3.1. Простой случай

Простейший способ запустить задачу - набрать ее имя:

```
] ls
```

запустится задача "ls", которая показывает имена файлов, расположенных на текущей директории.

Если дополнить имя задачи расширением ".cod", то произойдет то же самое. При этом нужно помнить, что поиск кода задачи будет происходить сначала по памяти - по прямым предкам новой задачи, а затем по диску - по директориям, описанным строкой "BIN" окружения пользователя (читайте об этом в разделе о задачах в ОС Excelsior). Если код не найден, то shell сообщит об этом.

Можно указать полное имя кодофайла. Например,

```
 ] /bin/ls
```

заставит shell временно поменять текущую директорию на /bin, загрузить задачу, а затем вернуться на прежнюю директорию и запустить загруженную задачу.

Часто возникает необходимость передать задачам какие-нибудь аргументы. Аргументы вводятся после имени задачи. На них распространяется Замечание из п.1.10. Например:

```
 ]ls $HOME
```

печатает имена файлов собственной директории пользователя.

```
 ]echo $HISTORY
```

выдает значение параметра HISTORY, если он есть в окружении; иначе строчку $HISTORY.

7.3.2. Запуск независимых задач

По способу исполнения задачи в ОС Excelsior делятся на независимые (открепленные) и зависимые. Завершения независимой задачи никто не ждет; она исполняется параллельно с

исполнением других задач. Завершения зависимой задачи shell ожидает и освобождает по завершении занимаемые ею ресурсы: закрывает открытые задачей файлы, освобождает память из-под задачи.

Для того, чтобы задача запустилась как независимая, поместите символ "&" в конец строки аргументов:

```
]<задача> <аргументы> &
```

Например:

```
]ls -1 >listing &
```

позволит записать в файл listing список всех файлов текущей директории в одну колонку, а программист тем временем может заниматься редактированием какого-нибудь текста.

Символ "&" из строки аргументов изымается и задаче не передается. При необходимости передать именно его в качестве последнего аргумента предварите его символом "\".

7.3.3. После того, как задача запущена

7.3.3.1. Прекращение зависимой задачи

После запуска зависимой (неоткрепленной) задачи shell дожидается ее окончания, и только после этого принимает новые команды.

При желании почти всегда задачу можно прервать нажатием CTRL_C на клавиатуре, или открепить нажатием CTRL_X (случаи, когда это невозможно, будут описаны позднее).

7.3.3.2. Параметр CHAIN

При нормальном окончании зависимой задачи перед ее удалением из памяти shell проверяет в ее окружении наличие параметра "CHAIN", и если он есть - интерпретирует его значение как следующую команду.

Замечание. Среди значений параметра "CHAIN" не исполняются команды, описанные в п.п.1.2.-1.8., а также команда "bye". (Вдумчивый читатель, прочитавший уже описания библиотек Shell и exeCut, наверняка понял, откуда это ограничение, и теперь имеет повод слегка повеселиться над нами. Что ж, нет в мире совершенства ...)

7.3.3.3. О трупах задач зависимых и независимых

Если зависимая задача завершилась нормально, ее "труп" (коды, процедурный стек, глобалы, созданные задачей структуры)

удаляется системой из памяти автоматически.

Если же она завершилась аварийно, ее труп удаляется из памяти только тогда, когда в окружении shell нет параметра с именем "CEMETRY", или его значение - ноль. В противном случае несчастные останки остаются в памяти до явного удаления (например, командой "kill").

Остаются в памяти также и трупы аварийно законченных независимых задач до явного их уничтожения командой kill.

Кстати, пока труп не удален из памяти, можно ознакомиться с историей болезни с помощью команды his (программисты обычно пользуются утилитой hi, которая предоставляет информацию в более удобной форме, см. справочник по утилитам).

Теперь вы, пожалуй, знаете, как запустить задачу, но некоторые тонкости этого действия заслуживают более подробного разговора.

7.3.4. Управление деревом задач

Замечание. При возникновении неясностей, касающихся организации дерева задач, советуем обратиться к разделу, посвященному задачам в ОС Excelsior.

Любая задача создается на базе другой задачи. Термин "создание на базе" здесь обозначает, что новая задача становится потомком другой, на чьей базе она создана, а задача-база, в свою очередь, - предком новой задачи.

Что означает отношение потомок-предок? Это означает, что кодофайлы для новой задачи сначала будут разыскиваться по ее прямым предкам, начиная от задачи-базы, а также задача может разделять глобалы какого-либо модуля только с одним из своих предков. И еще: при любом окончании задачи все ее потомки бессердечно уничтожаются.

Номер задачи-базы shell извлекает из параметра BASE своего окружения. Отсутствие такого параметра приводит к невозможности запустить какую-либо задачу.

Замечание. По соглашениям ОС Excelsior iV, все задачи начинают нумероваться с единицы (1). Если значение параметра BASE равно 0, это означает, что в качестве базы задачи используется ядро ОС. Это может быть полезным, например, при запуске драйверов.

Ясно, что запускающая задача (в нашем случае shell) в общем случае не имеет к базе никакого отношения - только то, что в окружении запускающей задачи указан номер задачи-базы для всех запускаемых задач.

7.3.5. Управление поиском кодофайлов

Перед именем задачи в строке может находиться некая информация, заключенная в фигурные скобки "{}". В частности, в таких скобках можно указать способы поиска кодофайлов.

Так, комбинация "-имя_модуля" обозначает, что модуль "имя_модуля" нужно обязательно загрузить с диска, даже если он был найден у одного из предков; комбинация "+имя_модуля" обозначает, что модуль "имя_модуля" необходимо найти у одного из предков и не копировать его глобалы (считать его уникальным); комбинация "=имя_модуля имя_файла" означает, что модуль "имя_модуля" нужно взять с диска, причем искать его в файле "имя_файла".

Например:

```
]{+myEditor} turbo2x
```

запустит в редакторе фильтр turbo2x, который является Модула-компилятором. Так простой текстовый редактор превращается в турбину.

Необходимо учесть, что здесь действует Замечание из п.7.2.10 о копировании комбинаций типа "$NAME".

Строчки "+имя_модуля", "-имя_модуля", "=имя_модуля имя_файла" можно занести в параметр окружения "ALIAS". После этого ВСЕ запуски задач будут осуществляться с указанными изменениями.

7.3.6. Управление окружением запускаемых задач

Каждая задача запускается с некоторым окружением. Окружение задачи получается простым копированием всех параметров окружения запускающей задачи (в нашем случае shell). Надо учесть, что параметры, помеченные как приватные, не копируются.

При этом над окружением новой задачи производятся следующие действия:

- удаляется параметр "CHAIN";
- создаются параметры "TASK", "FATHER", "NAME";
- если отсутствует, то создается параметр "BASE", со значением, равным "TASK".

И уже потом из части строки между скобками "{}", если она есть, выбираются все комбинации вида "NAME=string" и используются как указания к модификации окружения новой задачи.

Замечание. Такие комбинации выбираются, начиная с символа "{", до первой неподходящей комбинации; вся остальная

строка до "}" считается изменением поиска кодофайлов, о котором говорилось в п.7.3.5.

Таким образом, запуск `ls ../`
эквивалентен `{ CD=.. } ls`

Глава 7.4. Запуск командных файлов

Если попытка запустить задачу провалилась из-за отсутствия кодофайла (файла с расширением .cod), но в одной из директорий, перечисленных в параметре "ETC" окружения, обнаружился файл с именем "имя_задачи.@" (или "имя_задачи.sh"), то shell решит, что нужно запустить командный файл с этим именем. При желании запустить именно командный файл, даже если существует кодофайл с таким именем, укажите имя с требуемым расширением.

Иерархия при запуске задачи без явно указанного расширения будет такая: shell сначала разыскивает файл с расширением ".@"; не найдя, ищет файл с расширителем ".cod"; если же и такого не находит, пытается найти файл с расширителем ".sh".

Имеются две возможности в зависимости от того, с каким командным файлом мы имеем дело - ".sh" или ".@".

7.4.1. Запуск со специальным интерпретатором и без него

Если имя командного файла имеет расширение ".sh", то происходит следующее.

Для запуска командного файла shell ищет в своем окружении параметр с именем "SHELL", считает его содержимое именем установленного интерпретатора командных файлов и запускает задачу с таким именем, поправив для нее строку параметров: в ее начало будет вписано имя командного файла с префиксом "<". Например, если в вашем окружении "SHELL=mshell" и вы набрали строку

```
] { CD=/usr/bin } /usr/games/comp.@ poker xonix
```

, то это эквивалентно запуску

```
] { CD=/usr/bin } mshell </usr/games/comp.@ poker xonix
```

Отметим, что прелюдия в скобках "{}" относится к задаче "mshell".

Если параметр "SHELL" в окружении отсутствует, то командный файл с расширением ".sh" не может быть запущен.

Из всего сказанного можно сделать два тревожных вывода: во-первых, командный файл интерпретируется ОТДЕЛЬНОЙ ЗАДАЧЕЙ, во-вторых - он интерпретируются не shell, а чем-то другим.

Первое замечание очень важно, и мы его подробно обсудим ниже (7.4.3.), а по поводу второго скажем, что shell тоже умеет интерпретировать командные файлы, и имя shell можно смело указывать в строчке окружения SHELL.

Замечание. Более того, командный файл может быть запущен и

проинтерпретирован именно тем экземпляром задачи shell, "под которым" вы работаете. Для этого командный файл должен иметь расширение ".@".

7.4.2. Как shell интерпретирует командный файл

Командный файл полагается текстовым файлом с разделителем между строками ASCII.NL (код 036с). Строки не должны быть длиннее 255 символов (в противном случае выдается сообщение об ошибке). Если в прочитанной строке первый символ, отличный от пробела, равен "%", то строка считается комментарием и не исполняется.

В строках, прочитанных из файла, комбинации вида "$1","$2.."$9" заменяются на соответствующие аргументы командного файла с номерами соответственно 1,2..9 (в приведенном выше примере $1 обозначает "poker", $2 обозначает "xonix", а остальные обозначают пустую строку). Комбинации "\$" заменяются на "$", с отменой специального значения "$", а комбинации "\\" заменяются на "\". Результирующая (после всех подстановок) строка тоже не должна быть длиннее 255 символов.

Если в окружении shell'а-интерпретатора параметр "ECHO" есть и не равен "0", то обработанная строка будет напечатана на экране терминала. В тех же случаях печатаются на экране строки-комментарии.

После всего этого строка будет исполнена, как будто ее ввели с клавиатуры, кроме команды "bye".

Учитывая, что утилита shell вполне подходит для интерпретации командных файлов, имеет смысл именно ее имя хранить в окружении под именем "SHELL" (спросите "set", наверняка в вашем окружении так оно и есть).

7.4.3. Кто исполняет командный файл?

Тот факт, что командный файл исполняется отдельной задачей, приводит к некоторым особенностям, которые необходимо учитывать при его написании.

Первая особенность: у командного файла есть СВОЙ экземпляр окружения. Это значит, что значения параметров окружения командного файла могут отличаться от параметров вашей shell, и что модификации окружения по умолчанию относятся к окружению именно командного файла, а не shell.

Поясним это на примере. Набранные с клавиатуры команды

```
]echo $TASK
].INFO="кафолик недорезанный"
```

приведут к тому, что отпечатается номер вашей shell, и в именно в окружении shell появится ругачая строчка.

Эти же команды, помещенные в командный файл t.sh, после исполнения последнего приведут к тому, что напечатается другой номер (а именно - номер интерпретирующей shell), а окружение

вашей shell не изменится совсем (изменится окружение shell-интерпретатора, которое пропадет после завершения его работы).

Таким образом, чтобы с помощью командного файла изменить параметры именно вашего окружения, нужно предварить их имена номером вашей shell. Если вы хотите, чтобы командный файл t.sh исполнил то, что от него ожидается, исправьте его следующим образом:

```
echo $FATHER
.$FATHER.INFO="кафолик недорезанный"
```

Вторая особенность: при исполнении командного файла клавиатурные прерывания (CTRL_X, CTRL_C) действуют не на ту задачу, которая запущена из командного файла, а на весь командный файл в целом, то есть CTRL_C прерывает, а CTRL_X открепляет тот shell, который интерпретирует командный файл.

Глава 7.5. Когда CTRL_C не действует

> Включив тормоза, я вдруг обнаружил, что они не действуют и что корабль камнем падает на планету. Выглянув в люк, я увидел, что тормозов вообще нет.
>
> С.Лем. Звездные дневники Ийона Тихого

Как уже говорилось, по соглашениям ОС Excelsior существуют два символа, на которые установлена специфическая реакция: CTRL_X и CTRL_C. Первый обозначает открепление задачи, второй - ее прекращение. Эти символы действуют на все запущенные вашим shell задачи. Поскольку ваша shell запущена не ей самой, она не прекращается по нажатии этих клавиш.

Если вы запустили командный файл, то нажатием "CTRL_U" на своей клавиатуре вы прервете исполнение ВСЕГО командного файла, а не только той команды, которая исполнялась в момент нажатия.

Правда, существует исключительный случай, когда эти символы не действуют. Система предоставляет возможность задаче иметь собственный обработчик символов, посланных с клавиатуры. Перехватив у shell посланный CTRL_C, задача, имеющая такой обработчик, может отреагировать на него так, как считает нужным - например, никак не отреагировать.

Глава 7.6. Общее описание утилиты shell

> - Высочайшие достижения нейтронной мегалоплазмы! - провозгласил он. - Ротор поля наподобие дивергенции градуирует себя вдоль спина и там, внутре, обращает материю вопроса в спиритуальные электрические вихри, из коих и возникает синекдоха отвечания...
>
> А. и Б. Стругацкие. Сказка о Тройке

Shell - универсальная утилита и имеет несколько функций, которые зависят от ключа, с которым запускается утилита.

7.6.1. Раскрутка системы после загрузки

Способ запуска:

shell root=<имя_директории>

используется в файле конфигурации системы при ее раскрутке.

При раскрутке системы выполняются следующие действия:

- текущая директория устанавливается на указанную директорию;
- на текущей директории ищется командный файл с именем "profile.@" и исполняется;
- shell заканчивается.

7.6.2. Ведение диалога с пользователем

Запуск:

shell -home <имя_директории_пользователя>

осуществляется утилитой входа в систему login или администратором непосредственно.

При этом выполняются следующие действия:

- текущая директория устанавливается на указанную директорию;
- исполняется командный файл с именем $TTY_up.@;
- исполняется командный файл "profile.@" с текущей директории;
- в окружение заносятся параметр HOME с именем указанной директории и параметр USER с именем пользователя;

- начинается диалог с пользователем.

7.6.3. Интерпретация командных файлов

shell "<"имя_командного_файла { аргументы }

Может быть запущена пользователем для исполнения командного файла.

По пути, указанному в ETC, ищется файл с указанным именем, интерпретируется как командный файл, после чего утилита завершается.

7.6.4. Запуск без аргументов

shell

Ведет диалог, который заканчивается либо по команде bye, либо по нажатии клавиши Esc.

Глава 7.7. Приложение: перечень параметров окружения

shell использует следующие параметры окружения:

SHELL имя утилиты, которая вызывается при обнаружении командного файла. Параметр копируется обычным образом; его отсутствие приводит к невозможности исполнить командный файл.

STK размер стека для запускаемых задач в килобайтах. Копируется обычным образом. Если в окружении отсутсвует, то размер считается равным 4.

ALIAS строка управления поиском модулей.

BIN строка имен директорий, на которых производится поиск кодофайлов.

ETC строка имен директорий, на которых производится поиск командных файлов и файлов данных.

CD имя текущей директории. shell поддерживает ее корректность.

TTY имя устройства-экрана. Копируется обычным образом. При модификации этого параметра происходит захват специфицированного устройства.

KEY имя устройства-клавиатуры. Копируется обычным образом. При модификации этого параметра происходит захват специфицированного устройства.

HOME имя собственной директории пользователя.

TASK номер задачи "shell".

BASE номер задачи, на базе которой запускаются другие задачи. Если параметр отсутствует, то создается при запуске shell со значением "$TASK". Отсутствие приводит к ошибке при запуске задач.

FATHER номер задачи-родителя. Для задач, запущенных shell, этот номер берется из параметра BASE запускающего shell'a.

SON номер последней запущеной shell'ом задачи.

NAME имя задачи.

USER имя пользователя.

CEMETRY [0/1] -
хоронить ли трупы неоткрепленных задач при их аварийном завершении. При отсутствии параметр считается равным 1.

CHAIN post-mortem команда. Если в окружении нормально завершенной неоткрепленной задачи будет найдена строка с таким именем, то она будет исполнена.

PROMPT формат строчки-приглашения на ввод команды.

INSERT включение-выключение режима вставки при редактировании строки.

BELL включение-выключение звукового сигнала при редактировании строки.

BIN директории, на которых ищутся необходимые кодофайлы.

ETC директории, на которых ищутся командные файлы.

ECHO включение эхо-режима: показ всех команд перед исполнением.

Глава 7.8. Приложение: синтаксис команд shell

```
command ::= shell_command
          | change_environment0
          | task_control
          | mount_unmount
          | run_task .

shell_command ::= set | mem | ps | his |
                  su  | us  | cd | bye |
                  home| delay .
su       ::= "su" .
us       ::= "us" .
set      ::= "set"  [ pattern ] .
mem      ::= "mem" .
bye      ::= "bye" .
his      ::= "his"  [ number ] .
cd       ::= "cd"   [ new_cd_name ] .
home     ::= "home" [ new_home_dir_name ] .
delay    ::= "delay" number .
ps       ::= "ps"   { modifyer } { number } .
modifyer ::= "-mem" | "-tty" | "-usr" | "-l" | "-all" .

change_environment0 ::=
               ["."] [ task_number "." ] var_name "=" string .

task_control ::=  "stop"     task_number { task_number }
               |  "kill"     task_number { task_number }
               |  "wait"     task_number { task_number } .

task_number ::= целое беззнаковое число | $name .

run_task ::=
         [ "{" {change_environment1} {alias} "}" ]
                              task_name { args } [ "&" ] .

change_environment1 ::= ["."] var_name "=" string .
alias     ::= interface|unload|rename .
interface ::= "+" module_name .
unload    ::= "-" module_name .
rename    ::= "=" module_name file_name .

mount_unmount ::= "mount"   directory device ["-ro"]
                | "unmount" directory .

string ::= name | '"' { char } '"' | "'" { char } "'" .
```

ЧАСТЬ 8. ФАЙЛОВАЯ ПОДСИСТЕМА

Предисловие соавтора

Эта часть целиком написана Д.Кузнецовым (Leo). Я постаралась, насколько это было возможно, сохранить не только всю информацию, но и прелесть своеобразного языка, на котором пишет автор (Leopold-Russian), несколько изменив, по его желанию, только орфографию и синтаксис, а также произведя разбивку повествования на главы в соответствии с общим стилем книги.

Предисловие автора

Перед тем, как приступить к обсуждению основных концепций, положенных в основу ФАЙЛОВОЙ СИСТЕМЫ ОС Excelsior, мы считаем необходимым дать короткие, но точные, простые, но полные определения используемым в описании терминам и словам, употребленным в специальных значениях. Мы приложили колоссальные усилия для составления этого краткого, но емкого глоссария, которым и хотим предварить знакомство читателя с новейшими концепциями, изложенными в последующем описании.

Тем не менее, мы осознаем, что не смотря ни на какие усилия, в текст глоссария, формальный по духу своему, могли, к сожалению, вкрасться неточности и погрешности. Мы будем искренне признательны всем, кто выскажет свои замечания и укажет на необходимость дополнений.

Короткий глоссарий

ФАЙЛ - (м.род) полезная фитюлька, точное определение коей с ходу дать возможным не представляется.
(Син. FILE).

УСТРОЙСТВО - (ср.род) эдакая штука, устроенная разработчиками для пущего увеселения программеров. Его полезно от программера прятать, или, на худой конец, скрывать, чем и занята СИСТЕМА УПРАВЛЕНИЯ ФАЙЛАМИ (см.).
(син. DEVICE).

СИСТЕМА УПРАВЛЕНИЯ ФАЙЛАМИ - (ж.род) плод ...героически-бесполезных усилий по... упрятыванию от программера всех фитюлек первой необходимости в повседневном обиходе.
(Син. FILE SYSTEM, BASIC INPUT/OUTPUT).

НОСИТЕЛЬ - (м.род) можно носить, если габариты позволяют. Встречаются квадраты плоско-гибкие примерно 5 и 8 дюймов (син. FLOPPY) - удобны в переноске. Огнестрельные (син. WINCHESTER) представители не стреляют, монтаж/демонтаж (син. mount/unmount) осложнен, к переноске в большинстве своем пригодны. Встречаются крупногабаритные монстры... некоторые представители успешно используются для хранения всякой абракадабры (см.).

ИНФОРМАЦИЯ - (ж.род) абракадабра вроде той, что вы уже прочли(?) и еще прочтете(!)
(Син. INFORMACIJA (~tion?)).

ДИСКОВОД - (дик.род) носители водит, пока их не носят, иногда 50 оборотов в секунду, ингода 60, а иногда - 6, а жаль.
(син. [FLOPPY|WINCHESTER|XBZ] DISK DRIVE).

ТЕРМИНАЛ - (неопр.род) очень похож на ЧБ или ЦВ телевизор, но футбол и новости не всегда, изображение в основном из БУКВ, ЦИФР и КОРЮЧЕК, похоже на ВЕЗЕР РИПОТ и так же не понятно... Ксати, это - устройство...

ВЕЗЕР РИПОТ - прогноз погоды (син. WEATHER REPORT).

СИМВОЛ - КОРЮЧКА, БУКВА или ЦИФРА.
БАЙТ - емкость для корючек объемом 8 БИТ.
БИТ - емкость для "чего или ничего".

КЛАВИАТУРА - (ж.род) а чем же я, по-вашему, пишу?
(син. KEYBOARD = кнцлрские кнпк на дуб.дск).

СПЕЦИАЛЬНОЕ УСТРОЙСТВО - (ср.род) не ТЕРМИНАЛ,
с е и л н е не КЛАВИАТУРА,
п ц а ь о НЕ дисковод,
иногда бывают на самом деле (мышки,кошки,кодировщики, шифровальщицы и т.п.) но в основном измышляются мстительными программерами для запутывания разработчиков устройств, когда предпоследние все же ткнутся в софтвер.

СУ доверять нельзя (как и всему Спец.Х.), проверьте сначала как следует, вдруг оно не настоящее, а только кажется?

(Син. SPECIAL [sometimes and somewhere - VIRTUAL] DEVICE).

СИСТЕМА - (мн.ч.) совокупление ЗАДАЧ с РЕСУРСАМИ.

ЗАДАЧА - (хор.род) полноправно-полномочный представитель ПРИКЛАДНОЙ ПРОГРАММЫ, на который начисляются РЕСУРСЫ (если таковые есть в наличии).

ПРИКЛАДНАЯ ПРОГРАММА - (ж.род) озадачивание СИСТЕМЫ.

РЕСУРС - (дфц.род) то, чего не хватает на всех. "Начисление" сиих на ЗАДАЧУ может превратиться в "ожидание" или "разломание" последней, в зависимости от ДИСЦИПЛИНЫ управления РЕСУРСАМИ.

ДИСЦИПЛИНА - (никак.род) способность читать любую абракадабру (см.).

ПЕДАНТИЧНЫЕ ЛЮБИТЕЛИ ТОЧНЫХ ОПРЕДЕЛЕНИЙ - (ср.род ед.ч.) считают, что этот коротенький словарик - сплошной КУРАЖ (см.); а потому читать (и см.) дальше не будут.

ТЕМ, КОГО НЕ ОСТАНОВИЛО пред[ыдущее]остережение, советуем в качестве дополнения к обширным и общезначимым определениям, кои будут далее наполняться все б о л е е и б о л е е глубоким смыслом, воспользоваться также руководством по внутренней структуре файловой системы. (Мы его никогда не читали (да и не видали вовсе), но точно знаем, что оно-то существует а потому забыли название ("ищущий да обрящет")).

НУ ПОЕХАЛИ; С БОГОМ... (син. "ну и Бог с ним...").

Глава 8.1. Виды файлов

Существует три основных вида файлов: обычные файлы, директории и устройства. Устройства, в свою очередь, бывают трех категорий: диски, терминалы (клавиатуры) и специальные. С точки зрения системы управления файлами, все три вида файлов являются никак не структурированными одномерными массивами байтов.

8.1.1. Обычные файлы

Обычными (ordinary) файлами называются все файлы, не являющиеся директориями или устройствами. После открытия они предоставляются прикладным программам для чтения и записи информации (произвольной природы и длины) с произвольной байтовой позиции. Обычные файлы можно рассматривать как одномерные энергонезависимые байтовые массивы изменяемого размера.

Примечание. Под энергонезависимостью понимается свойство не изменяться при выключении источника энергии.

8.1.2. Директории

Директории обеспечивают отображение (paths) имен файлов в сами файлы и индуцируют древовидную структуру на файловой системе в целом. В целях обеспечения целостности структуры файлового дерева непосредственное изменение содержимого директорий недоступно для пользовательских программ и выполняется только под контролем самого BIO. Директории можно (но не рекомендуется) читать как обычные файлы. В различных реализациях системы внутренний формат директорий может быть различным, а операция dir_walk позволит сделать программы, итерирующие директории, универсальными и не зависящими от этого формата.

8.1.3. Файлы-устройства

Файлы-устройства (специальные файлы) обеспечивают непосредственный доступ к управлению устройствами. Каждое поддерживаемое в BIO устройство ввода/вывода ассоциировано хотя бы с одним специальным файлом. Специальные файлы можно читать и писать как обычные файлы, но результатом запроса на запись или чтение (или выполнение операции управления устройством) будет активация соответствующего устройству драйвера вместо нормального механизма чтения/записи обычных файлов.

Вот неполный перечень выгод от унификации устройств как файлов:

- ввод и вывод у файлов и устройств схожи друг с другом настолько, насколько это возможно;
- имена файлов и устройств имеют одинаковый синтаксис и значение; таким образом, программам, ожидающим имя файла в качестве параметра, может быть с успехом передано имя некоторых устройств, а иногда и наоборот;
- на устройства распространяется общий механизм защиты от несанкционированного доступа;
- поскольку ввод и вывод унифицированы, один и тот же набор операций может быть использован и для файлов, и для устройств.

Устройства делятся на три важные категории: структурированные (или, по традиции, диски), неструктурированые (по традиции - терминалы и сериальные устройства) и специальные.

... не все файлы одинаковы, и следует считать, что в этом (возможно, не всегда выгодном) их свойстве отражено все многообразие проявлений ... мира

Глава 8.2. Параллелизм

Все операции над файлами защищены от одновременного их использования в нескольких задачах. Если с файлом работают более, чем одна задача, то система обеспечивает взаимное исключение задач при исполнении любых операций над этим файлом.

Тем не менее, BIO не обеспечивает взаимного исключения процессов (threads) одной задачи при выполнении операций над файлами. Кроме этого, BIO использует глобальные переменные задачи для информирования прикладной программы о своем состоянии. Поэтому, если в прикладной программе необходимо совместное использование несколькими процессами операций из BIO, данная программа обязана позаботиться о синхронизации и взаимном исключении этих процессов, в противном случае последствия одновременного выполнения операций (даже над разными файлами) могут быть непредсказуемо печальны.

... . (Гай Юлий Цезарь)

Глава 8.3. Корневая и текущая директории

При исполнении любой программы в системе существует два предопределенных файла: корневая (root) и текущая (current) директории. Корневая директория определяется в процессе загрузки системы и обычно не изменяется по ходу работы системы. Текущая директория может быть изменена любой прикладной программой по ее усмотрению на любую (доступную) другую.

В некоторых исключительных ситуациях (при автономном (stand-alone) исполнении прикладной программы) эти две директории могут оказаться недоступными.

BIO поддерживает открытым файл текущей директории (cd). Корневая директория может быть открыта нормальным способом.

При запуске задач ими наследуется текущая директория. Каждая задача имеет свою независимую текущую директорию, и смена ее никак не сказывается на других задачах.

Корневая директория может быть сменена только привилегированным пользователем, и ее смену ощутят сразу все задачи (ежели попытаются воспользоваться корневой директорией).

... "Смотри в КОРЕНЬ!" (Козьма Пр.)

Глава 8.4. Имена файлов

> - Они, конечно, откликаются, если их позвать? - небрежно заметил Комар.
> - Я не знала, что они умеют... откликаться.
> - Зачем же им имена, - возразил Комар, - если они на них не откликаются?
>
> Л.Кэрролл. Алиса в Зазеркалье

8.4.1. Маршрут

Когда прикладная программа передает системе имя файла, последнее может быть специфицировано как маршрут (pathname). Маршрут есть последовательность имен директорий, разделенных символом "/", заканчивающаяся именем файла. Последовательность директорий, предшествующую имени файла, назовем префиксом. Собственно имя файла будем иногда называть последним именем маршрута.

Примечание. Ранее вместо слова "маршрут" (pathname) мы использовали не совсем удобный термин "полное имя файла", а также употребляли не вполне корректный термин "путь", заимствованный из ОС UNIX. Далее словом под словом "путь" (path) мы будем понимать именно путь поиска файла, а именно, упорядоченное множество маршрутов.

8.4.2. Поиск от корня

В системе принято следующее соглашение об интерпретации маршрута: если префикс начинается с символа "/", просмотр начинается с корневой директории (root directory). Так, например,

```
/usr/etc/some
```

вынуждает систему найти на корневой директории поддиректорию "usr", потом на "usr" найти "etc" и, наконец, на "etc" найти файл "some". Файл "some" может оказаться обычным файлом, директорией или устройством.

В качестве предельного случая: маршрут "/" ссылается на саму корневую директорию.

8.4.3. Поиск от текущей директории

Любой не начинающийся с символа "/" префикс (или пустой) вынуждает систему начать поиск с текущей директории. Простейшая форма маршрута (например, "some") ссылается на

файл, который система будет искать на текущей директории. Такие формы маршрутов позволяют программам быстро и просто указать файл на поддиректории, не вызывая необходимости помнить полные маршруты файлов (путь от корня).

8.4.4. Имена для текущей и материнской директорий

Стандартные имена "." и ".." всегда ссылаются на саму директорию и директорию, содержащую данную, соответственно. При этом имя ".." можно итерировать, двигаясь от матери к бабушке и далее к корневой директории. Имя ".." на корневой директории ссылается на нее самое.

Примечание. Корневая директория и директория, на которой смонтирован носитель, могут не содержать имени ".." в случае монтирования носителя, записанного в старых версиях системы.

8.4.5. Именование файла

Имя файла состоит из 31 или менее символов и завершается байтом с кодировкой 000c (32 bytes null terminated string). Символы в имени файла могут быть любые, кроме следующих запрещенных:

ПРОБЕЛ	" "	SPACE	040c
СЛЭШ	"/"	SLASH	057c
ДВОЕТОЧИЕ	":"		072c
НУЛЬ		NULL	000c

Допустимо (но не рекомендуется) использование символов с кодировками 000c..037c и 200c..377c, так как такое имя файла можно отобразить не на всяком терминале.

8.4.6. О привязанностях

В дальнейшем элемент директории, состоящий из имени файла и ссылки на файл, будет частенько называться "входом" (entry).

Если директория содержит вход (entry) с некоторым именем, ссылающийся на файл, то говорят, что этот файл "привязан" к директории под этим именем.

Возможно существование нескольких ссылок на файл из различных директорий. В таком случае говорят, что файл привязан к нескольким директориям и "счетчик связей" файла больше единицы. Файл считается уничтоженным тогда, когда исчезла последняя ссылка на него, т.е. он отвязан от всех директорий, к которым был привязан.

Попытка привязать файл к директории, находящейся на другом носителе (не на том, на котором расположен сам файл), приведет к ошибке ПЕРЕКРЕСТНАЯ ССЫЛКА (XCROSS), и файл

привязан не будет.

Поскольку директория содержит ссылку ".." на родительскую директорию, она не может быть привязана более чем к одной директории. Попытка привязать директорию приведет к возбуждению ошибки ЭТО ДИРЕКТОРИЯ (IS DIRECTORY).

Глава 8.5. Хранение файлов

Свойством длительного (и иногда надежного) сохранения информации обладают обычные файлы и структурированые устройства (некоторые специальные устройства также могут обладать этим свойством).

Структуированое устройство представляет собой массив байтов фиксированной длины, из(в) которого(ый) можно прочесть(записать) последовательность байтов требуемого рамера. При выходе операции чтения/записи за границы установленного на специальном устройстве носителя возбуждаются ошибки ПЛОХОЙ ПАРАМЕТР (BAD PARAMETER) или НЕПРАВИЛЬНЫЙ АДРЕС НА УСТРОЙСТВЕ (ILLEGAL DEVICE ADDRESS).

Большинство известных структурированых устройств не требует, чтобы читаемые данные были когда-либо записаны до момента чтения. Для тех, что требуют этого (некоторые типы ОЗУ-дисков - RAM-disks), в случае чтения данных, которые ранее никогда не писались, может возникать ошибка НЕТ ДАННЫХ (NO DATA) или другие ошибки.

Обычные файлы хранятся в файловых системах, располагающихся на смонтированных носителях, установленных на структурированных устройствах. Текущая реализация накладывает ограничение на максимальный размер одного файла - он не превышает 33,554,432 байта. Файл является одномерным байтовым массивом, но его длину можно увеличивать. Увеличение длины файла производится либо явной операцией, либо автоматически при попытке записи данных за концом файла. Длину файла можно также уменьшать.

Обычные файлы хранятся неплотно. В связи с этим из них можно прочитать только те данные, которые в них были когда-либо записаны. При попытке чтения не записанных ранее данных может возникнуть ошибка НЕТ ДАННЫХ (NO DATA) или получена бессмысленная информация. При отведении места на носителе под файл учитывается размер "блокировки" данного носителя. Размер блокировки любого носителя можно узнать, выполнив соответствующую операцию. Чтение и запись файлов производятся наиболее эффективно с позиций, кратных размеру блока на данном носителе. Чтение и запись с некратных позиций могут также оказаться весьма приемлемы по эффективности при достаточно большой длине запросов на ввод/вывод. Наименее эффективными могут оказаться чтение и запись маленьких порций информации с произвольных позиций большого файла.

Файловая система гарантирует правильное хранение содержимого файла от его первого байта до конца файла (eof) при выполнении любых операций чтения/записи. Сохранность данных, оказавшихся за концом файла (eof) в результате перестановки последнего, не гарантируется.

Глава 8.6. Защита файлов

Каждому пользователю в системе присвоен уникальный идентификатор пользователя (user identification) и идентификатор его группы (group identification). Идентификаторы пользователя и группы наследуются всеми запускаемыми пользователем задачами. При создании файл помечается идентификаторами пользователя и группы его владельца. Для созданного файла устанавливается маска защиты (creation mask), которая независимо специфицирует права чтения, записи и исполнения для владельца файла (owner), других членов этой же группы, а также всех остальных, не попавших в первые две категории. Для директорий право на исполнение интерпретируется как право на "просмотр" (directory walk) содержимого директории.

Два дополнительных бита специфицируют необходимость смены идентификатора пользователя и группы при запуске задачи на базе данного кодофайла.

Владельцем файла, таким образом, является ПОЛЬЗОВАТЕЛЬ (а это - нечто абстрактное, живущее вне системы), и для простоты далее будет говорится о программах, являющихся владельцами файлов, что следует понимать как "программы, запущенные программами, запущенные программами, ..., имевшими код пользователя, равный коду владельца файла".

При интерпретации маршрута проверяется, что все директории, указанные в префиксе, доступны задаче по чтению.

При открытии и создании файла прикладная программа должна указать необходимые ей права доступа к файлу. В момент открытия (создания) проверяется возможность такого доступа, и если она отсутствует, возбуждается ошибка НАРУШЕНИЕ СЕКРЕТНОСТИ (SECURITY VIOLATION) и открытия (создания) не происходит.

В случае удачного открытия права доступа программы к файлу сохраняются и не изменяются даже при изменении идентификатора пользователя (группы) этой программы или изменении фактических прав доступа к файлу. В случае попытки применения к файлу операции, противоречащей правам доступа к файлу, установленным в момент его открытия, такая операция отклоняется, и возбуждается ошибка ЗАПРЕЩЕННЫЙ ДОСТУП (ILLEGAL ACCESS).

Права доступа к файлу ассоциированы с самим файлом, и в случае, если он привязан в нескольких местах файлового дерева (см. link), изменение прав доступа к файлу одинаково заметно во всех других точках привязки файла.

... "Безобразия нарушать вздумали?" (Неизвестный)

Глава 8.7. Библиотека BIO

Переходим к непосредственному комментированию библиотеки BIO, являющейся интерфейсом файловой подсистемы.

```
 --------------------------------
|  TYPE FILE;   VAL null: FILE;  |
 --------------------------------
```

Все операции ввода/вывода применяются к объектам типа FILE, а не к самим файлам.

Значения [абстрактного] типа FILE можно породить, создавая или открывая ВНЕШНИЙ ФАЙЛ, и уничтожить, закрывая его.

Следует избегать прямого присваивания переменных типа FILE операцией ":=", поскольку после закрытия файла значения всех переменных, ссылающихся на данный файл, становятся нехорошими, их дальнейшее использование может привести в лучшем случае к ошибке "ПЛОХОЙ ДЕСКРИПТОР" (bad_desc), в худшем - к обращению к другому файлу.

Значение null для типа ФАЙЛ является предопределенным неопределенным значением. Любая операция ввода/вывода или управления файлом над файловым значением null всегда приводит к возникновению ошибки "ПЛОХОЙ ДЕСКРИПТОР" (bad_desc).

... это вам файл! а не здесь ... (Неизвестный)

```
 ------------------------------------
|  VAL done: BOOLEAN;                |
|     error: INTEGER;                |
|     ename: ARRAY [0..31] OF CHAR;  |
 ------------------------------------
```

При успешном завершении любой операции переменная done принимает значение TRUE, значения переменных error и ename не изменяются.

Если при выполнении какой-либо операции возникли ошибки, переменная done принимает значение FALSE, значение переменной error соответствует одной из произошедших ошибок. При множественных ошибках невозможно достоверно предсказать не зависящим от реализации способом, какая из ошибок будет зафиксирована. Система придерживается стратегии фиксирования наиболее "важной" ("суровой") из ошибок. В случае, если известно имя файла, в результате операции над которым произошла ошибка, оно записывается в ename. В тех случаях, когда оно не известно, в ename заносится специальное значение "UNKNOWN" (например, в случае ошибки "ПЛОХОЙ ДЕСКРИПТОР").

Система всегда пытается аннулировать результаты ошибочно завершившейся операции. При ошибках ввода/вывода важной дополнительной информацией является длина удачно обработанной части данных iolen (подробности см. далее).

... "Все в мире имеет конец, даже несчастия" (Г.Х.Андерсен)

8.7.1. Общие замечания

BIO содержит набор операций, парных друг к другу, позволяющих интерпретировать маршрут относительно текущей директории (cd) или произвольной директории (рассматривая ее как текущую). Такие операции будут описываться парами, например:

```
   dosomething(          ... path: ARRAY OF CHAR ...);
  fdosomething(cd: FILE; ... path: ARRAY OF CHAR ...);
```

при этом следует иметь в виду, что выполнение операции

dosomething(...)

полностью эквивалентно выполнению операции

fdosomething(BIO.cd,...)

и попросту является его удобным сокращением.

Все вышесказаное следует относить только к операциям, описание которых дается парно, и не следует распространять на другие похожие по звучанию и написанию имена операций (например, read, fread; write, fwrite и т.п.)

Все операции, имеющие в качестве аргумента маршрут, интерпретируют его префикс. Все имена в префиксе должны ссылаться на директории (иначе возбуждается ошибка ЭТО НЕ ДИРЕКТОРИЯ). Все директории в префиксе должны быть доступны по чтению. Если чтение запрещено, возбуждается ошибка НАРУШЕНИЕ СЕКРЕТНОСТИ (SECURITY VIOLATION).

Операции открытия и создания файла имеют в качестве параметра также способ открытия файла (mode), который определяет набор операций, применимых к открытому файлу, и способ их применения. Способ открытия - это строка из символов (специфичных для каждой операции), приведенных в произвольном порядке. Символы, не наделенные семантикой для данной операции, игнорируются, не приводя к возбуждению ошибки.

8.7.2. Об ошибках

Везде далее за описанием той или иной операции будет следовать описание ошибок, которые могут возникнуть в процессе ее исполнения. Главным признаком ошибки является ее английская аббревиатура, соответствующая общесистемному коду ошибки. Пользователь (или программа) может получить также название ошибки на русском, английском (или другом) языке, в зависимости от установленного языкового драйвера. Такие названия здесь приводятся прописными буквами. Они никоим образом не являются делом файловой системы и в принципе могут несколько отличаться от используемых в текущей реализации ОС Excelsior.

При выполнении практически любой операции над файлами возможно возникновение следующих ошибок (поэтому эти ошибки не описываются отдельно для каждой операции):

ПЛОХОЕ ИМЯ bad_name
: Имя файла либо содержит недопустимые символы, либо пустое или слишком длинное.

ПЛОХОЙ ДЕСКРИПТОР bad_desc
: В качестве значения типа FILE использовано неправильное значение, неоткрытый (или уже закрытый) файл, значение BIO.null или испорченное значение.

НЕТ ПАМЯТИ no_memory
: Нет свободной памяти для порождаемых в ходе выполнения операции структур данных.

НАРУШЕНИЕ СЕКРЕТНОСТИ sec_vio
: Невозможно открыть или создать файл с указанными правами доступа, или нет права чтения директории, встретившейся в префиксе маршрута.

ЗАПРЕЩЕННЫЙ ДОСТУП ill_access
: Права доступа к файлу не позволяют применить к нему данную операцию.

НЕПОДХОДЯЩИЙ unsuitable
: Объект, к которому применяется операция, не является объектом подходящего класса (например, операция make file system неприменима к файлу, являющемуся сериальным устройством).

ПРЕРВАННАЯ ОПЕРАЦИЯ ipted_op
: Операция была прервана в результате того, что задаче, запросившей операцию, был послан сигнал kill.

НЕПРАВИЛЬНАЯ ОПЕРАЦИЯ ill_op
: Данная операция не применима к этому объекту или не

реализована в драйвере устройства, с которым ассоциирован (или на котором хранится) этот объект.

ОШИБКА ВВОДА/ВЫВОДА io_error **

При выполнении операции ввода/вывода информации драйвер или контроллер устройства зафиксировали нарушения в работе оборудования.

ПЛОХАЯ ФАЙЛОВАЯ СИСТЕМА bad_fsys **

При работе с файловой системой на дисковом устройстве обнаружены ошибки в ее логической организации. Требуется немедленное прекращение работы с данной файловой системой, тщательная ее проверка и, возможно, коррекция и восстановление.

НЕ ОПРЕДЕЛЕНО undef **

Устройство, над котором должна быть выполнена операция (или на котором располагается файл) ЕЩЕ или УЖЕ не определено. Устройство может быть ЕЩЕ не определено в момент, когда драйвер устройства уже прошел стадию регистрации, но еще не представил системе свой обработчик запросов на ввод/вывод. Устройство может быть УЖЕ не определено в момент, когда драйвер устройства завершил свою работу (например, в связи с поломкой устройства или ошибкой в самом драйвере, приведшей к его аварийной остановке), а задача еще располагает открытыми на этом устройстве файлами. Устойство полностью исчезнет из системы только после завершения драйвера и закрытия всех файлов на этом устройстве.

Здесь и везде далее двумя звездочками (**) помечены ошибки, предсказать результаты операции при возникновении которых достаточно сложно и они могут оказаться самыми различными. В таких ситуациях файловая система на носителе, или устройство, или контроллер устройства, или процессор требуют специфического обслуживания (maintenance) или ремонта (re mount).

Если в описании операции не растолкована возбуждаемая ошибка, а только указано ее название - это означает, что причина возбуждения ошибки совпадает с описанной здесь.

8.7.3. chdir, chroot

```
 ------------------
|  VAL   cd: FILE;  |
 ------------------
```

В переменной cd [почти] всегда содержится открытый файл текущей директории. Переменная cd инициализируется в момент запуска задачи путем открытия директории с маршрутом, взятым из ОКРУЖЕНИЯ (Environment), по имени "CD". В случае неудачного открытия cd переставляется на корневую директорию.

Непосредственно после запуска задачи и до первого обращения к любой процедуре из BIO переменная done является результатом попытки открытия текущей директории, и в случае если done=FALSE, переменная error содержит код ошибки открытия этой директории.

```
PROCEDURE chdir(path : ARRAY OF CHAR);
--------------------------------------
```

Переставляет директорию cd на директорию, указанную маршрутом path. В случае ошибки cd остается на прежнем месте.

Возможные ошибки:

```
ЭТО НЕ ДИРЕКТОРИЯ           not_dir
```

имя в префиксе маршрута или сам маршрут указывают не на директорию;

```
НЕТ ТАКОГО                  no_entry
```

не найдена директория [из маршрута] с таким именем;

```
ПЛОХОЕ ИМЯ                  bad_name
НЕТ ПАМЯТИ                  no_memory
НАРУШЕНИЕ СЕКРЕТНОСТИ       sec_vio
ПЛОХАЯ ФАЙЛОВАЯ СИСТЕМА     bad_fsys  **
ОШИБКА ВВОДА/ВЫВОДА         io_error  **
НЕ ОПРЕДЕЛЕНО               undef     **
```

```
PROCEDURE chroot(path : ARRAY OF CHAR);
---------------------------------------
```

Переставляет корневую директорию файловой системы на директорию, указанную маршрутом path. В случае ошибки корневая директория остается на прежнем месте. Эта процедура может быть выполнена только привилегированным пользователем.

Возможные ошибки:

ЭТО НЕ ДИРЕКТОРИЯ not_dir
имя в префиксе маршрута или сам маршрут указывают не на директорию;

ДЛЯ ПРИВИЛЕГИРОВАННЫХ su_only
эта операция разрешена только в привилегированном режиме;

НЕТ ТАКОГО no_entry
не найдена директория [из маршрута] с таким именем;

ПЛОХОЕ ИМЯ bad_name
ПЛОХОЙ ДЕСКРИПТОР bad_desc
НЕТ ПАМЯТИ no_memory
НАРУШЕНИЕ СЕКРЕТНОСТИ sec_vio
ПЛОХАЯ ФАЙЛОВАЯ СИСТЕМА bad_fsys **
ОШИБКА ВВОДА/ВЫВОДА io_error **
НЕ ОПРЕДЕЛЕНО undef **

8.7.4. fname, splitpathname

```
PROCEDURE fname (file: FILE; VAR name: ARRAY OF CHAR);
PROCEDURE splitpathname(VAR path: ARRAY OF CHAR;
                    VAR name: ARRAY OF CHAR);
-----------------------------------------------
```

Операция fname выдает полное имя файла (с указанием маршрута от корня).

Операция splitpathname позволяет выделить из имени маршрута префикс и последнее имя.

При применении операции параметр path должен содержать спецификацию маршрута. После успешного применения операции в параметре path остается префикс маршрута, в переменную name заносится последнее имя. Префикс может оказаться пустым.

Если при выделении префикса или имени в них обнаружены символы, запрещенные к использованию в именах файлов, а также если последнее имя или одно из имен директорий в префиксе состоит более чем из 31 символа, возбуждается ошибка ПЛОХОЕ ИМЯ (BAD NAME).

Переменная name после успешного завершения операции обязательно содержит непустую строку символов, которая обязательно завершается байтом с кодом 000c.

Если объем переменной name не достаточен для вмещения последнего имени, то имя усекается без возбуждения ошибки.

Возможно возникновение ошибок:

ПЛОХОЕ ИМЯ	bad_name

8.7.5. equal, open, fopen

```
PROCEDURE equal(f0,f1: FILE): BOOLEAN;
--------------------------------------
```

Возвращает TRUE, если файловые переменные f0,f1 ссылаются на один файл, иначе FALSE.

```
PROCEDURE open (VAR file: FILE; path,mode: ARRAY OF CHAR);
PROCEDURE fopen(cd: FILE;
                VAR file: FILE; path,mode: ARRAY OF CHAR);
-----------------------------------------------------------
```

Операции open,fopen позволяют открыть файл, директорию, устройство для дальнейшей работы с ними.

Операция open является точным эквивалентом операции fopen(BIO.cd,.....).

Path содержит маршрут и имя открываемого файла. Open открывает файл и устанавливает права и методы доступа в соответствии с mode. В случае успешного открытия порождает и возвращает новое значение типа FILE. Указатель текущей позиции устанавливается в начало файла (если не указано обратное).

Значение mode конструируется в соответствии со следуюшим списком:

'r' (read) файл открывается для чтения;
'w' (write) файл открывается для записи;
'm' (modify) файл открывается для чтения и записи;
'x' (execute) файл открывается для исполнения/просмотра;
'X' (eXelent) файл открывается с максимально возможными правами доступа;
'a' (append) после открытия файла указатель текущей позиции установится в конец файла;
'd' (direct write) ожидание завершения записи;
'n' (nodelay) чтение без ожидания (читается столько, сколько есть в буфере (для последовательных и специальных устройств);
'c' (no casch) выключение кэша, чтение всегда с диска и прямая запись (для обычных файлов);

Открытие файла совсем без доступа по чтению/записи может оказаться полезным для опроса или изменения его атрибутов (особенно тогда, когда чтение/запись запрещены) или для итерации (прогулке по) директории.

Возможно возникновение ошибок:

ЭТО НЕ ДИРЕКТОРИЯ not_dir

имя в префиксе маршрута указывает не на директорию, или значение cd не является директорией;

ПЛОХОЙ ДЕСКРИПТОР bad_desc
плохое значение параметра или переменной cd;

НЕТ ТАКОГО no_entry
на директории, указанной маршрутом, не найден файл с указанным именем, или не найдена директория для одного из имен в префиксе маршрута;

НЕТ ПАМЯТИ	no_memory	
НАРУШЕНИЕ СЕКРЕТНОСТИ	sec_vio	
ПЛОХОЕ ИМЯ	bad_name	
ПРЕРВАННАЯ ОПЕРАЦИЯ	ipted_op	
ПЛОХАЯ ФАЙЛОВАЯ СИСТЕМА	bad_fsys	**
ОШИБКА ВВОДА/ВЫВОДА	io_error	**
НЕ ОПРЕДЕЛЕНО	undef	**

В случае возникновения ошибки файл 'file' не открывается и возвращается значение BIO.null.

Привилегированному пользователю (superuser), и только ему, предоставляется возможность открытия файлов устройств, для которых нет соответствующих узлов в файловом дереве (см. mknode). Для этого необходимо в качестве имени открываемого файла указать имя логического устройства, предваренное символом ":". Из-за отсутствия узла в файловом дереве (а соответственно, и списка прав доступа к устройству) такое открытие вскрывает устройство с полным набором прав (все дозволено). Такой способ открытия необходим и применяется только при установке системы на новое оборудование и при установке корня файловой системы после загрузки ОС.

8.7.6. create, fcreate, chmode

```
VAL cmask: BITSET;

PROCEDURE create
  (VAR  file: FILE; path,mode: ARRAY OF CHAR; size: INTEGER);
PROCEDURE fcreate(cd:  FILE;
   VAR  file: FILE; path,mode: ARRAY OF CHAR; size: INTEGER);
-------------------------------------------------------------
```

Операции create, fcreate позволяют создать постоянный или временный файл на носителе, смонтированном на структурированном устройстве.

Операция create является точным эквивалентом операции fcreate(BIO.cd,.....).

Path содержит маршрут и имя создаваемого файла. Create создает файл и устанавливает права и методы доступа в соответствии с mode. В случае успешного создания порождает и возвращает новое значение типа FILE.

Указатель текущей позиции устанавливается в начало файла (если не указано обратное). Указатель конца файла eof также стоит в начале.

Файл всегда создается на носителе, на котором расположена директория cd.

Параметр size задает размер (в байтах) пространства на носителе, которое должно быть предварительно (в момент создания) выделено для файла. Если создание файла прошло успешно, в файл гарантированно возможно записать минимум size байт информации. Параметр size не фиксирует максимальный размер файла (ограниченный, главным образом, размером свободного пространства на носителе). Он только указывает минимальный необходимый размер файла, известный программе. Если таковой неизвестен, можно смело использовать значение 0.

Вне зависимости от параметра size после операций create, fcreate указатель конца файла (eof) всегда установлен в 0.

Имя файла может быть пустым. В таком случае создается временный файл, который будет уничтожен после закрытия (если до этого не будет "привязан" см. link).

Нельзя создать файл с именами "." и ".."; в этом случае возбуждается ошибка ПЛОХОЕ ИМЯ (BAD NAME).

В момент создания запоминается имя файла и директория, на которую указывал маршрут. Привязывание файла к директории произойдет только в момент его ЗАКРЫТИЯ. Вплоть до закрытия файла на директории, где он создан, будет сохраняться (ежели наличествует) файл с таким же именем.

Имя файла, оставшееся в path после интерпретации префикса, должно состоять не более чем из 31 символа и завершаться 0c. В случае более длиного имени оно усекается до 31 символа (без возбуждения ошибок). Ошибка ПЛОХОЕ ИМЯ (BAD NAME) возбуждается в том случае, если в имени файла обнаружены символы, запрещенные к использованию в именах файлов (см.

ранее).

Если файл не будет закрыт самой программой, то по ее окончании (как аварийном, так и нормальном) файл будет закрыт БЕЗ привязывания к директории (purge).

Значение переменной cmask (creation mask) определяет права доступа к файлу после его привязывания. См. также операцию chcmask.

Значение mode конструируется в соответствии со следуюшим списком:

'r' (read) файл создается для чтения;
'w' (write) файл создается для записи;
'm' (modify) файл создается для чтения и записи;
'x' (execute) файл создается для исполнения/просмотра;
'X' (eXelent) файл создается с максимально возможными правами доступа;
'd' (direct write) ожидание завершения записи;
'n' (nodelay) чтение без ожидания (читается столько, сколько есть в буфере (для последовательных и специальных устройств);
'c' (no casch) выключение кэша, чтение всегда с диска и прямая запись (для обычных файлов);
'h' создать файл с признаком hidden (в момент закрытия файл будет привязан как "невидимый").

Права доступа к файлу задачи, создавшей его, определяются значением параметра mode вплоть до закрытия файла. Значение cmask не влияет на права доступа к файлу задачи, создавшей файл. Оно будет использовано только в случае, если какая-либо задача (в том числе и создавшая его) попытается открыть его после (закрытия и) привязывания.

Возможно возникновение ошибок:

ЭТО НЕ ДИРЕКТОРИЯ not_dir
имя в префиксе маршрута указывает не на директорию, или значение cd не является директорией;

НЕТ СВОБОДНОГО МЕСТА no_space
на носителе, где создается файл, нет требуемого свободного пространства размером size байт;

НОСИТЕЛЬ ПЕРЕПОЛНЕН fsys_full
на носителе, где создается файл, переполнена таблица файлов;

ПЛОХОЙ ДЕСКРИПТОР bad_desc
плохое значение параметра или переменной cd;

НЕТ ТАКОГО no_entry
не найдена директория для одного из имен в префиксе маршрута;

НЕТ ПАМЯТИ	no_memory	
НАРУШЕНИЕ СЕКРЕТНОСТИ	sec_vio	
ПЛОХОЕ ИМЯ	bad_name	
ПРЕРВАННАЯ ОПЕРАЦИЯ	ipted_op	
ПЛОХАЯ ФАЙЛОВАЯ СИСТЕМА	bad_fsys	**
ОШИБКА ВВОДА/ВЫВОДА	io_error	**
НЕ ОПРЕДЕЛЕНО	undef	**

В случае возникновения ошибки файл 'file' не создается и возвращается значение BIO.null.

... "Сделайте мне красиво ..." ...

8.7.7. du, dup

```
PROCEDURE du(cd: FILE; VAR free,used: INTEGER);
```

Операция du (disk usage) позволяет узнать размер свободного и занятого пространства (в байтах) на носителе, на котором расположена указанная директория.

Возможно возникновение ошибок:

ЭТО НЕ ДИРЕКТОРИЯ not_dir
значение cd не является директорией;

```
PROCEDURE dup(VAR new: FILE; file: FILE);
```

Операция dup предназначена для порождения нового значения типа FILE путем копирования уже имеющегося значения.

Создает новое значение типа FILE, которое ссылается на тот же файл, что и file. Для файла заводится новый указатель позиции, новые права доступа и новое множество (изначально пустое) буферов ввода/вывода.

Указатель позиции, права доступа и число возможных буферов ввода/вывода копируются из файла file.

Если файл file был создан операцией create, НЕ происходит копирования в значение new имени (под которым нужно будет привязать файл) и директории (к которой нужно будет привязать файл в момент его закрытия).

В случае отсутствия буферизации можно использовать новое значение new как дополнительный указатель текущей позиции ввода/вывода, т.к. в этом случае все изменения содержимого (и указателя конца) в любой из этих двух копий немедленно становятся доступны и для другой.

Файл file может быть и директорией. Если эта директория итерировалась с помощью dir_walk (см.) в момент дублирования, это никак не скажется на ее дубликате.

Закрытие файла file операциями close или purge никак не сказывается на его дубликате.

Возможные ошибки:

ПЛОХОЙ ДЕСКРИПТОР bad_desc
плохое значение параметра file;

НЕТ ПАМЯТИ no_memory

8.7.8. close, purge

```
PROCEDURE close (VAR file: FILE);
PROCEDURE purge (VAR file: FILE);
---------------------------------
```

Операции purge и close закрывают файл, делая невозможным дальнейшее выполнение операций над ним.

Процедуры purge и close работают абсолютно одинаково для всех файлов, кроме созданных процедурами create, fcreate.

Файлы, созданные процедурами create и fcreate, процедура purge закрывает безусловно, не пытаясь привязать к директории.

Если файл был открыт или создан с пустым именем (см. create), close тоже просто закрывает его. Иначе close пытается привязать файл к директории, и закрывает его, если только это удалось. В случае ошибки привязывания файл остается открытым (file#BIO.null), и прикладная программа может по выбору попытаться привязать его в другое место файлового дерева или выполнить над ним операцию purge. В случае если программа так и не пришла ни к какому решению, по ее завершении над всеми созданными, но не привязанными файлами выполняется операция purge.

Если в момент закрытия или удаления файл имеет нулевое число связей (не привязан ни к одной директории), он будет удален, а пространство, занятое его данными, будет считаться свободным.

Возможно возникновение ошибок:

ЭТО ДИРЕКТОРИЯ is_dir *
: при закрытии (close) и привязывании созданного операцией create файла обнаружилась непустая директория с таким же именем;

НАРУШЕНИЕ СЕКРЕТНОСТИ sec_vio *
: при закрытии (close) и привязывании созданного операцией create файла обнаружилось нарушение прав доступа к директории;

НЕТ СВОБОДНОГО МЕСТА no_space
: при закрытии (close) и привязывании созданного операцией create файла потребовалось расширить директорию, к которой привязывается файл, и на носителе, на котором создан файл, не хватило свободного пространства;

ПЛОХОЙ ДЕСКРИПТОР bad_desc
: плохое значение параметра file;

ПРЕРВАННАЯ ОПЕРАЦИЯ ipted_op
ПЛОХАЯ ФАЙЛОВАЯ СИСТЕМА bad_fsys **
ОШИБКА ВВОДА/ВЫВОДА io_error **
НЕ ОПРЕДЕЛЕНО undef **

Символом (**) помечены ошибки, при которых файл может остаться НЕЗАКРЫТЫМ (file#BIO.null). Во всех остальных случаях файл закрывается даже в случае возникновения ошибки, и переменной file присваивается значение BIO.null.

ПРИМЕЧАНИЕ. В момент удаления файла пространство, отведенное файлу, продолжает содержать ранее записанную в него информацию. Это пространство впоследствии может быть отведено под другой файл, и программа, создавшая этот другой файл, может получить доступ ко всей (или к части) информации из удаленного файла. В случае особой важности и секретности хранимой в файле информации для предотвращения такой ситуации рекомендуется перед уничтожением файла "почистить" его содержимое (например, прописав его весь байтами с кодом 000, или любыми другими, менее секретными, чем его драгоценное содержимое).

8.7.9. link, flink

```
PROCEDURE  link (file: FILE; path,mode: ARRAY OF CHAR);
PROCEDURE flink (cd  : FILE;
                 file: FILE; path,mode: ARRAY OF CHAR);
-------------------------------------------------------
```

Операции link, flink позволяют привязать файл к директории.

Операция link является точным эквивалентом операции flink(BIO.cd,.....).

Path содержит маршрут, префикс которого указывает на директорию, и имя, под которым к ней надо привязать файл.

В случае успешного привязывания счетчик связей файла увеличивается на 1.

Параметр mode может указывать на необходимость установки атрибута "скрытый" (hidden).

Значение mode конструируется в соответствии со следуюшим списком:

'h' привязать файл с признаком hidden ("невидимый");
's' привязать файл с признаком system ("системный").

Нельзя привязать:

- файл к директории, находящейся на другом носителе (ПЕРЕКРЕСТНАЯ ССЫЛКА);
- директорию (ЭТО ДИРЕКТОРИЯ);
- файл, если существует непустая директория с таким же именем (ЭТО ДИРЕКТОРИЯ);
- файл с именами "." и ".."; в этом случае возбуждается ошибка ПЛОХОЕ ИМЯ (BAD NAME).

Имя файла, оставшееся в path после интерпретации префикса, должно состоять не более чем из 31 символа и завершаться 0c. В случае более длинного имени оно усекается до 31 символа (без возбуждения ошибок). Ошибка ПЛОХОЕ ИМЯ (BAD NAME) возбуждается в том случае, если в имени файла обнаружены символы, запрещенные к использованию в именах файлов.

Для привязывания файла к директории необходимо иметь к ней доступ по записи.

Если в момент привязывания на директории существует файл (или пустая поддиректория) с таким же именем, как имя привязываемого файла, он (она) будет отвязан(а).

Возможно возникновение ошибок:

ЭТО НЕ ДИРЕКТОРИЯ not_dir
имя в префиксе маршрута указывает не на директорию, или значение cd не является директорией;

ЭТО ДИРЕКТОРИЯ is_dir

привязываемый файл оказался директорией, или файл, отвязываемый вследствие привязывания, оказался непустой директорией;

НОСИТЕЛЬ ПЕРЕПОЛНЕН fsys_full
на носителе, куда привязывается файл, переполнена таблица файлов;

НЕТ СВОБОДНОГО МЕСТА no_space
на носителе, где располагаются файл и директория, нет свободного пространства, необходимого для увеличения размера директории;

НЕТ ТАКОГО no_entry
не найдена директория для одного из имен в префиксе маршрута;

ПЛОХОЙ ДЕСКРИПТОР bad_desc
плохое значение параметра или переменной cd;

НАРУШЕНИЕ СЕКРЕТНОСТИ sec_vio
нет права записи в директорию, к которой ведется привязывание;

ПЕРКРЕСТНАЯ ССЫЛКА xcross
файл и директория, к которой ведется привязывание, находятся на разных носителях;

ПЛОХОЕ ИМЯ	bad_name	
ПРЕРВАННАЯ ОПЕРАЦИЯ	ipted_op	
ОШИБКА ВВОДА/ВЫВОДА	io_error	**
ПЛОХАЯ ФАЙЛОВАЯ СИСТЕМА	bad_fsys	**
НЕ ОПРЕДЕЛЕНО	undef	**

В результате возникновения любой ошибки (кроме помеченных **) файл остается непривязанным, а файл, который должен был быть отвязан в результате привязывания, неотвязанным.

8.7.10. unlink, funlink

```
PROCEDURE unlink(            path: ARRAY OF CHAR);
PROCEDURE funlink(cd: FILE;  path: ARRAY OF CHAR);
-----------------------------------------------
```

Операции unlink, funlink позволяют отвязать файл или поддиректорию от директории.

Операция unlink является точным эквивалентом операции funlink(BIO.cd,.....).

Path содержит маршрут, префикс которого указывает на директорию, и имя файла, который нужно от нее отвязать.

В случае успешного отвязывания счетчик связей файла уменьшается на 1. Если при этом он стал равным 0, происходит уничтожение файла и освобождение ранее занятого им пространства.

Нельзя отвязать:

- непустую поддиректорию;
- поддиректорию с именем ".."; в этом случае возбуждается ошибка ПЛОХОЕ ИМЯ (BAD NAME).

Для отвязывании файла от директории необходимо иметь к ней доступ по записи.

Возможно возникновение ошибок:

ЭТО НЕ ДИРЕКТОРИЯ not_dir
имя в префиксе маршрута указывает не на директорию, или значение cd не является директорией;

ЭТО ДИРЕКТОРИЯ is_dir
отвязываемый файл оказался непустой директорией;

ПЛОХОЙ ДЕСКРИПТОР bad_desc
плохое значение параметра или переменной cd;

НАРУШЕНИЕ СЕКРЕТНОСТИ sec_vio
нет права записи в директорию, к которой осуществляется привязывание;

НЕТ ТАКОГО no_entry
на директории, указываемой маршрутом, не найден файл с указанным именем, или не найдена директория для одного из имен в префиксе маршрута;

```
ПЛОХОЕ ИМЯ                  bad_name
ПРЕРВАННАЯ ОПЕРАЦИЯ         ipted_op
ОШИБКА ВВОДА/ВЫВОДА         io_error **
ПЛОХАЯ ФАЙЛОВАЯ СИСТЕМА     bad_fsys **
НЕ ОПРЕДЕЛЕНО               undef    **
```

В результате возникновения любой ошибки (кроме помеченных **) файл остается неотвязанным.

8.7.11. mkdir, fmkdir

```
PROCEDURE mkdir (          path: ARRAY OF CHAR; hidden: BOOLEAN);
PROCEDURE fmkdir(cd: FILE; path: ARRAY OF CHAR; hidden: BOOLEAN);
-----------------------------------------------------------------
```

Операции mkdir, fmkdir позволяют создать новую поддиректорию.

Операция mkdir является точным эквивалентом операции fmkdir(BIO.cd,.....).

Path содержит маршрут, префикс которого указывает на директорию, и имя поддиректории, которую нужно создать.

Параметр hidden определяет видимость/скрытость созданной директории.

В случае успешного создания поддиректории на ней автоматически создается вход с именем "..", ссылающийся на материнскую директорию. Число связей директории, тем не менее, всегда равно единице независимо от числа ее поддиректорий (содержащих на нее обратную ссылку "..").

Если при создании поддиректории на директории находился файл (или пустая поддиректория) с таким же именем, он(а) автоматически отвязывается.

При наличии непустой поддиректории с таким же именем операция создания не происходит и возбуждается ошибка ЭТО ДИРЕКТОРИЯ (IS DIRECTORY).

Для создания поддиректории необходимо иметь доступ по записи к директории, на которой происходит создание.

Нельзя создать поддиректории с именами "." и ".."; в этом случае возбуждается ошибка ПЛОХОЕ ИМЯ (BAD NAME).

Возможно возникновение ошибок:

ЭТО НЕ ДИРЕКТОРИЯ not_dir
имя в префиксе маршрута указывает не на директорию, или значение cd не является директорией;

ЭТО ДИРЕКТОРИЯ is_dir
отвязываемый файл оказался непустой директорией;

ПЛОХОЙ ДЕСКРИПТОР bad_desc
плохое значение параметра или переменной cd;

НАРУШЕНИЕ СЕКРЕТНОСТИ sec_vio
нет права записи в директорию, к которой ведется привязывание;

НЕТ ТАКОГО no_entry
не найдена директория для одного из имен в префиксе маршрута;

ПЛОХОЕ ИМЯ	bad_name
ПРЕРВАННАЯ ОПЕРАЦИЯ	ipted_op
ПЛОХАЯ ФАЙЛОВАЯ СИСТЕМА	bad_fsys **
ОШИБКА ВВОДА/ВЫВОДА	io_error **
НЕ ОПРЕДЕЛЕНО	undef **

В результате возникновения любой ошибки (кроме помеченных **) поддиректория не создается.

8.7.12. fmvdir

```
PROCEDURE fmvdir(cd,to: FILE; path,name: ARRAY OF CHAR; hidden:
BOOLEAN);
-------------------------------------------------------------
--------
```

Операция fmvdir перемещает директорию с одного места в файловом дереве в другое.

Path указывает на директорию (относительно cd), которую необходимо перенести на директорию to с именем name.

Параметр hidden определяет видимость/скрытость перемещенной директории.

В случае успешного переноса директории ссылка из вход с именем ".." на ней автоматически переделывается в ссылку на директорию to. Директория отвязывается от директории, к которой она была ранее привязана. Число связей директории не изменяется. При перемещении директории вместе с ней, естественно, перемещаются и все ее потомки (поддиректории, поддиректории поддиректорий и т.д.).

Если при перемещении директории на директории to находился файл (или пустая поддиректория) с таким же именем, он(а) автоматически отвязывается.

При наличии непустой поддиректории с таким же именем при перемещении директории возбуждается ошибка ЭТО ДИРЕКТОРИЯ (IS DIRECTORY).

Для перемещения директории необходимо иметь доступ по записи к директории, содержашей данную, и к той, на которую происходит перемещение.

Нельзя переместить:

- директорию с новым именем "." и ".."; в этом случае возбуждается ошибка ПЛОХОЕ ИМЯ (BAD NAME);

- корневую директорию и директорию, на которую смонтирован носитель; в этом случае возбуждается ошибка ПЕРЕКРЕСТНАЯ ССЫЛКА (XCROSS);

- директорию между носителями; в этом случае возбуждается ошибка ПЕРЕКРЕСТНАЯ ССЫЛКА (XCROSS).

Возможно возникновение ошибок:

ЭТО НЕ ДИРЕКТОРИЯ not_dir
имя в префиксе маршрута указывает не на директорию, или значение cd или to не является директорией, или path указывает не на директорию;

ЭТО ДИРЕКТОРИЯ is_dir
отвязываемый файл оказался непустой директорией;

ПЛОХОЙ ДЕСКРИПТОР bad_desc
плохое значение переменных cd или to, или параметра;

НАРУШЕНИЕ СЕКРЕТНОСТИ sec_vio
нет права записи в директорию, из(к) которой ведется отвязывание(привязывание);

НЕТ ТАКОГО no_entry
не найдена директория для одного из имен в префиксе маршрута;

ПЕРКРЕСТНАЯ ССЫЛКА xcross
попытка переместить директорию с одного носителя на другой;

ПЛОХОЕ ИМЯ bad_name
ПРЕРВАННАЯ ОПЕРАЦИЯ ipted_op
ОШИБКА ВВОДА/ВЫВОДА io_error **
ПЛОХАЯ ФАЙЛОВАЯ СИСТЕМА bad_fsys **
НЕ ОПРЕДЕЛЕНО undef **

В результате возникновения любой ошибки (кроме помеченных **) поддиректория не перемещается.

8.7.13. check_io, buffers, seek

```
PROCEDURE check_io(halt_on_error: BOOLEAN);
-------------------------------------------
```

Операция check_io предназначена для включения/выключения аварийного завершения задачи при ошибках в операциях ввода/вывода. По умолчанию аварийное завершение запрещено и операции ввода/вывода возвращают ошибки в случае их возникновения.

```
PROCEDURE buffers(file: FILE; no,size);
---------------------------------------
```

Буферы файла используются для буферизации чтения и записи в файл. Указание числа буферов разрешает системе (но не обязывает ее) использовать no буферов. Размер буфера (в байтах) size может округляться системой до ближайшего большего, кратного blocksize байт (см. fstype).

В случае нехватки памяти для размещения буферов возбуждается ошибка no_memory. Система обеспечивает нормальное функционирование с меньшим числом буферов и даже вовсе без буферизации.

Заметим, что в случае изготовления дубликата значения типа файл операцией dup буферы для нового значения типа файл (как и текущая позиция в файле) будут другими, нежели у исходного файла. Синхронизация некогеррентного содержимого буферов может быть достигнута выполнением операции flush после каждой записи и перед каждым чтением из файла. (Правда, при такой дисциплине использования эффективность буферизации слегка ниже нуля, и поэтому выгоднее просто не использовать буферизацию при необходимости таких приложений).

В случае отсутствия буферизации операции чтения/записи используют системную кеш-память, содержащую когеррентную информацию вне зависимости от числа дубликатов дескриптора файла.

Буферизация может оказаться весьма полезной при чтении/записи файлов в построчном (getstr, putstr) или посимовльном (getch, putch) режимах. (Замечание: посимвольный режим все же существенно медленнее любых других, поэтому рекомендуется по возможности избегать его применения и пользоваться более крупноблочными операциями).

```
PROCEDURE seek(file: FILE; offset,origin: INTEGER);
---------------------------------------------------
```

Операция seek устанавливает в файле текущую позицию записи/чтения.

В зависимости от origin, значение offset задает:
origin=0 смещение от начала файла;

origin=1 смещение от текушей позиции;
origin=2 смещение от конца файла.

При всех других значениях origin возбуждается ошибка ПЛОХОЙ ПАРАМЕТР (BAD PARAMETER).

Текущая позиция в файле может указывать на произвольное место (в том числе и за конец файла). Она не может быть отрицательна, и если после seek получилась отрицательная позиция, возбуждается ошибка ПЛОХОЙ ПАРАМЕТР (BAD PARAMETER).

Если текущая позиция указывает за конец файла, применение к файлу операций чтения приведет к ошибке НЕТ ДАННЫХ (NO DATA), в то время как запись с этой позиции приведет к автоматическому расширению файла до нужных размеров и перестановке указателя конца файла (eof). При этом значения байтов между старым и новым значениями указателя конца файла могут быть не определены и даже вовсе отсутствовать (так называемый неплотный файл).

Для специальных файлов (неструктурированных устройств), кроме устанавления позиция чтения/записи, происходит обращение к драйверу специального устройства с запросом на операцию SEEK. Если операция SEEK не реализована в драйвере, будет возбуждена ошибка НЕПРАВИЛЬНАЯ ОПЕРАЦИЯ.

Возможно возникновение ошибок:

ПЛОХОЙ ДЕСКРИПТОР bad_desc
неоткрытый, или уже закрытый, или испорченный file;
ПЛОХОЙ ПАРАМЕТР bad_parm
неправильное значение параметров origin или offset;
ПРЕРВАННАЯ ОПЕРАЦИЯ ipted_op
НЕПРАВИЛЬНАЯ ОПЕРАЦИЯ ill_op
ОШИБКА ВВОДА/ВЫВОДА io_error **
НЕ ОПРЕДЕЛЕНО undef **

8.7.14. pos, eof

```
PROCEDURE pos (file: FILE): INTEGER;
PROCEDURE eof (file: FILE): INTEGER;
-----------------------------------
```

Операция pos возвращает текущее положение указателя чтения/записи в файле.

Операция eof возвращает текущий размер файла.

Указатель позиции чтения/записи может перемещаться по файлу в результате выполнения операций чтения/записи и операции seek. С помощью операции pos можно узнать текущее положение указателя чтения/записи.

Для неструктурированных устройств позиция также хранится и может изменяться в результате выполнения операций чтения/записи и операции seek. Применение операции doio() никак не влияет на указатель чтения/записи, даже если на устройстве действительно выполнялась операция обмена.

Операция eof выдает число, равное максимальному номеру байта в файле плюс единица. Для структурированных устройств eof выдает размер устройства в байтах. Для неструктурированных устройств происходит обращение к драйверу устройства с запросом READY, что позволяет программе узнать число байтов, готовых к чтению, в буфере ввода. Если операция READY не реализована в драйвере, будет возбуждена ошибка НЕПРАВИЛЬНАЯ ОПЕРАЦИЯ.

Возможно возникновение ошибок:

```
ПЛОХОЙ ДЕСКРИПТОР           bad_desc
     неоткрытый, или уже закрытый, или испорченный file;
ПРЕРВАННАЯ ОПЕРАЦИЯ         ipted_op
НЕПРАВИЛЬНАЯ ОПЕРАЦИЯ       ill_op
ОШИБКА ВВОДА/ВЫВОДА         io_error **
НЕ ОПРЕДЕЛЕНО               undef    **
```

8.7.15. fread, fwrite, read, write, get, put

```
VAL iolen: INTEGER;

PROCEDURE fread (f: FILE; buf: ADDRESS;   pos,len: INTEGER);
PROCEDURE fwrite(f: FILE; buf: ADDRESS;   pos,len: INTEGER);
PROCEDURE read  (f: FILE; buf: ADDRESS;       len: INTEGER);
PROCEDURE write (f: FILE; buf: ADDRESS;       len: INTEGER);
PROCEDURE get   (f: FILE; buf: ARRAY OF WORD; len: INTEGER);
PROCEDURE put   (f: FILE; buf: ARRAY OF WORD; len: INTEGER);
-------------------------------------------------------------
```

Операции обмена (передачи данных - data transfer) fread/fwrite, read/write, get/put осуществляют чтение/запись файлов.

Пары операций отличаются по протоколу вызова, но все они сводятся к двум базовым операциям fread/fwrite и в этом смысле почти им эквивалентны (см. примечание):

```
read (f,b,l)  == fread (f,b,0,l)
write(f,b,l)  == fwrite(f,b,0,l)

get(f,data,l) == fread (f,ADR(data),0,l)
put(f,data,l) == fwrite(f,ADR(data),0,l)
```

Операция fread для обычных файлов осуществляет чтение ранее записанных в файл данных. По окончании операции переменная iolen содержит число действительно прочитанных байтов. Это число может быть меньше, чем было заказано, если специальный файл открыт в режиме "без ожидания" (no wait) или встретился конец файла. Даже если в процессе чтения был достигнут конец файла, ошибка НЕТ ДАННЫХ не возбуждается! Если перед чтением указатель текущей позиции был строго (!) больше, чем eof, то при чтении ненулевого числа байтов возбуждается ошибка НЕТ ДАННЫХ.

Выполнение операции разрешено только в том случае, если программа обладает правом доступа к файлу по чтению.

Операция fwrite для обычных файлов осуществляет запись в файл данных. По окончании операции переменная iolen содержит число действительно записанных байтов.

Выполнение операции разрешено только в том случае, если программа обладает правом доступа к файлу по записи. В случае обычных (ordinary) файлов дополнительно требуется, чтобы носитель, на котором расположен файл, был смонтирован без запрещения записи (см. mount).

Если при записи указатель позиции в файле выходит за конец файла, файл автоматически увеличивается. Если файл должен быть увеличен на существенную длину (обычно несколько десятков килобайт), выгодно заранее (при создании) отвести

соответствующее пространство файлу или расширить его перед записью операцией extend (см.).

Если в процессе записи и увеличения файла было исчерпано пространство на носителе, ошибка НЕТ СВОБОДНОГО МЕСТА (NO FREE SPACE) возбуждается только в том случае, когда ни одного байта информации записать не удалось. Если запись частично удалась (свободное место было исчерпано не в самом начале записи), ошибка не возбуждается, а переменная iolen содержит число действительно прочитанных байтов.

Если в процессе записи носителя на структурированном устройстве был достигнут конец устройства, ошибка НЕТ ДАННЫХ возбуждается только в том случае, когда ни одного байта информации записано не было. Если запись частично удалась (конец носителя был достигнут не в самом начале записи), ошибка не возбуждается, а переменная iolen содержит число действительно прочитанных байтов.

Выполнение операции разрешено только в том случае, если программа обладает правом доступа к файлу по записи.

Примечание. Все операции, кроме fread/fwrite, дополнительно содержат проверку равенства числа переданных байтов (iolen) длине запроса на ввод/вывод len, и в случае несовпадения len и iolen при безошибочном выполнении fread/fwrite возбуждают ошибку НЕТ ДАННЫХ (NO DATA).

После выполнения операции чтения/записи указатель текущей позиции в файле продвигается на iolen байтов. Если файл расширен при записи, указатель eof файла тоже увеличивается.

Операции записи в паре с операцией seek позволяют создать "дырявые" файлы, т.е. файлы, в которых записанная информация расположена не непрерывно. Такие файлы не противоречат концепциям, заложенным в файловую систему, и все операции над ними будут выполняться абсолютно корректно, если читается всегда только ранее записанная информация.

Примечание. Полезно иметь в виду, что отведение места на носителе для хранения файла происходит порциями, кратными некоторому базовому размеру, обычно существенно большему, чем 1 байт (см. также операцию fstype). Поэтому при записи "кусочных" файлов могут появиться участки файла, информация в которые никогда не записывалась, но которые, тем не менее, можно прочитать. Значения, извлеченные из файла таким способом, с некоторой вероятностью (обычно маленькой) будут коррелировать с погодой на Марсе на момент извлечения. Разработчики не несут ответственность за неустойчивую наблюдаемость сей корреляции.

Кроме того, что размер файла можно увеличить, дописывая в

него информацию или принудительно выполняя операцию extend, размер файла можно также изменить, выполняя операции cut и end.

Операция end (см.) просто переставляет указатель конца файла (eof), не отводя дополнительного места на носителе (оно будет выделено при записи) при увеличении eof, и не утилизуя освободившегося пространства при уменьшении eof.

Следует иметь в виду, что файловая система поддерживает целостность файла только в диапазоне 0..eof-1, и в случае временного уменьшения eof и его последующего восстановления в прежнее значение целостность данных, оказавшихся на время переставления eof за пределами файла, не гарантируется! Они могут быть испорчены в результате выполнения коротких операций "дозаписи" за eof-ом, поскольку при таких "дозаписях", естественно, не производится "предчтение" записываемых блоков.

Операция cut (см.) аналогична операции end, но осуществляет утилизацию всего пространства, переставшего принадлежать файлу в результате его укорачивания. При последующем расширении файла пространство ему будут отводиться заново (и возможно, другое).

Возможно возникновение ошибок:

ЭТО ДИРЕКТОРИЯ is_dir
попытка записи в файл, являющийся директорией;

НЕТ СВОБОДНОГО МЕСТА no_space
на носителе, где располагается файл, нет свободного пространства, необходимого для увеличения его размера;

ПЛОХОЙ ДЕСКРИПТОР bad_desc
неоткрытый, или уже закрытый, или испорченный file;

ЗАПРЕЩЕННЫЙ ДОСТУП ill_access
права доступа к файлу не позволяют применить к нему данную операцию;

НЕТ ДАННЫХ no_data
попытка чтения из файла данных, которых он не содержит по той или иной причине, или попытка записать в файл данные, которые в него не влезают;

ПРЕРВАННАЯ ОПЕРАЦИЯ ipted_op
ОШИБКА ВВОДА/ВЫВОДА io_error **
ПЛОХАЯ ФАЙЛОВАЯ СИСТЕМА bad_fsys **
НЕ ОПРЕДЕЛЕНО undef **

8.7.16. doio

```
PROCEDURE doio(file: FILE; VAR r: defRequest.REQUEST);
```

Операция doio позволяет непосредственно обратиться к драйверу устройства.

Для выполнения операции doio необходимо, чтобы файл file был файлом-устройством, иначе возбуждается ошибка НЕПОДХОДЯЩИЙ (UNSUITABLE).

Для выполнении запроса с операцией READ из модуля defRequest требуется наличие права доступа к устройству по чтению. Для выполнении запроса с операцией WRITE из модуля defRequest требуется наличие права доступа к устройству по записи. Для выполнения любой другой операции необходимо наличие права доступа по исполнению.

Возможно возникновение ошибок:

ПЛОХОЙ ДЕСКРИПТОР bad_desc
: неоткрытый, или уже закрытый, или испорченный file;

НЕПОДХОДЯЩИЙ unsuitable
: файл file не является устройством;

ЗАПРЕЩЕННЫЙ ДОСТУП ill_access
: права доступа к файлу не позволяют применить к нему данную операцию;

НЕТ ДАННЫХ no_data
: попытка чтения из файла данных, которых он не содержит, или попытка записать в файл данные, которые в него не влезают;

НЕПРАВИЛЬНАЯ ОПЕРАЦИЯ ill_op
: данная операция не применима к этому файлу или не реализована в драйвере устройства;

ПРЕРВАННАЯ ОПЕРАЦИЯ ipted_op
ОШИБКА ВВОДА/ВЫВОДА io_error **
ПЛОХАЯ ФАЙЛОВАЯ СИСТЕМА bad_fsys **
НЕ ОПРЕДЕЛЕНО undef **

8.7.17. getch, putch, getstr, putstr, print

```
PROCEDURE getch(file: FILE; VAR ch: CHAR);
PROCEDURE putch(file: FILE;     ch: CHAR);

PROCEDURE getstr(file: FILE; VAR str: ARRAY OF CHAR);
PROCEDURE putstr(file: FILE;     str: ARRAY OF CHAR);
-----------------------------------------------------
```

Операции getch/putch и getstr/putstr осуществляют ввод/вывод для текстовых файлов.

Операции getch/putch читают(пишут) следующий байт из(в) файла точно так же, как это делают операции get(f,ch,1)/put(f,ch,1). Операции getch/putch могут с равным успехом применяться и к нетекстовым файлам.

Операция getstr читает из файла все байты до тех пор, пока не встретит байт с кодировкой 12c, или 15c, или 00c, или 36c ("разделитель"), или пока файл не будет исчерпан. getstr останавливается при возникновении любой из этих ситуаций (той, что случится раньше) и переписывает прочитанную информацию в строку str.

Если объем строки str оказался недостаточным для вмещения всей прочитанной информации, она усекается, и хвост этой информации теряется. Если чтение строки было остановлено по появлению разделителя, сам разделитель в файле пропускается, но в строку str не заносится. Строка str всегда завершается байтом с кодировкой 000c. Если размер строки <=0, то есть в нее нельзя положить даже байт 000c, возбуждается ошибка ПЛОХОЙ ПАРАМЕТР (BAD PARAMETER).

В переменную iolen всегда заносится длина реально прочитанной из файла информации (а не длина информации, занесенной в строку str), что позволяет определить (сравнивая iolen и BYTES(str)), произошла ли потеря информации из-за переполнения строки или нет.

Операция putstr записывает в файл все байты из строки str до тех пор, пока не встретит конец строки (байт с кодировкой 000c), или пока все содержимое строки str не будет записано.

Никакой разделитель строк в файл не записывается, тем самым последовательные вызовы putstr продолжают запись одной и той же строки. В случае необходимости записать в файл разделитель строк программа сама должна выполнить эту запись, предварительно приняв решение, какой разделитель строк использовать (см. примечание).

Следует иметь в виду, что при отсутствии буферизации побайтовый метод доступа к файлам может оказаться в десятки раз медленнее, чем обмен более крупными порциями (см. open).

Возможно возникновение ошибок:

ПЛОХОЙ ПАРАМЕТР bad_parm
строка str в операции getstr имеет размер BYTES(str)<=0;

ЭТО ДИРЕКТОРИЯ is_dir
попытка записи в файл, являющийся директорией;

НЕТ СВОБОДНОГО МЕСТА no_space
на носителе, где располагается файл, нет свободного пространства, необходимого для увеличения его размера;

ПЛОХОЙ ДЕСКРИПТОР bad_desc
неоткрытый, или уже закрытый, или испорченный file;

ЗАПРЕЩЕННЫЙ ДОСТУП ill_access
права доступа к файлу не позволяют применить к нему данную операцию;

НЕТ ДАННЫХ no_data
попытка чтения из файла данных, которых он не содержит по той или иной причине, или попытка записать в файл данные, которые в него не влезают;

ПРЕРВАННАЯ ОПЕРАЦИЯ ipted_op
ОШИБКА ВВОДА/ВЫВОДА io_error **
ПЛОХАЯ ФАЙЛОВАЯ СИСТЕМА bad_fsys **
НЕ ОПРЕДЕЛЕНО undef **

Примечание. Вопрос о разделителях строк в текстовых файлах достаточно запутан. Различные ОС и даже различные прикладные программы могут требовать различных разделителей строк в текстовых файлах.

Внутренним стандартом для фирменных прикладных программ ОС Excelsior является чтение разделителей: LF (12c), CR LF (15c 12c), CR (15c), NUL (00c) и NL (36c). Запись производится с разделителем NL (36c). При чтении текстовых файлов, содержащих произвольные разделители из перечисленных, прикладной программе следует проявлять избыточный интеллект и после чтения каждой строчки, завершенной символом CR (15c), проверять, не является ли следующий символ символом LF (12c), и если это так, игнорировать (пропускать) его.

Известные нам разделители:

Excelsior-I	NUL; NL	
MS-DOS	CR LF; CR; LF; NUL	(-)
Unix	LF	
RT-11/RSX-11/VMS ...	CR LF	
Excelsior-II	NL	(+)
Excelsior-iV	NL	(+)

(-) нет единого соглашения даже о записи файлов.
(+) фирменные программы читают любые разделители.

```
PROCEDURE print(file: FILE; format: ARRAY OF CHAR;
                                              SEQ args: WORD);
```

Операция print осуществляет вывод в файл текстовой информации в соответствии с заданным форматом.

В ходе выполнения операции print запись информации в файл может происходить неоднократно, при этом iolen равно сумме длин запросов на вывод. В случае возникновения ошибок ввода/вывода done=FALSE, а error равно последней из возникших ошибок.

Заметим, что \n записывается как байт с кодом 36c, тогда как \r\l означают CR LF (см. выше). Подробности об операции print см. в разделе о стандартном вводе/выводе в Руководсте по ОС Excelsior.

8.7.18. cut, end, extend

```
PROCEDURE cut   (file: FILE; size: INTEGER);
PROCEDURE end   (file: FILE; size: INTEGER);
PROCEDURE extend(file: FILE; size: INTEGER);
-------------------------------------------
```

Операции extend, cut, end обеспечивают возможность управления размером файла и размером отведенного для него пространства на носителе.

Операция extend отводит файлу пространство на носителе, необходимое для размещения в нем size байтов. После успешного выполнения операции extend указатель конца файла не изменяется, но в файле теперь достаточно пространства для хранения в нем size байтов, все последующие операции записи в границах 0..size-1 не потребуют отведения места на носителе. При расширении "дырявого" файла для всех "дырок", имевшихся в файле, отводится пространство на носителе. При возникновении ошибок (кроме помеченных **) в исполнении операции extend пространство, отведенное файлу, и сам файл остаются без изменений.

Операция end переставляет указатель конца файла, не отводя нового места на носителе (в случае увеличения eof) и не утилизуя освободившееся место (в случае уменьшения).

Операция cut переставляет указатель конца файла и утилизует освобождаемое пространство между новым и старым значениями eof.

Выполнение этих операций разрешено только для обычных (ordinary) файлов, не являющихся директориями и только в том случае, если программа обладает правом доступа к файлу по записи и носитель, на котором расположен файл, смонтирован без запрещения записи (см. mount).

Возможно возникновение ошибок:

ПЛОХОЙ ПАРАМЕТР bad_parm
: указанный параметр size меньше нуля или слишком велик для носителя, на котором расположен файл;

ЭТО ДИРЕКТОРИЯ is_dir
: файл является директорией;

НЕТ СВОБОДНОГО МЕСТА no_space
: на носителе, где располагается файл, нет свободного пространства, необходимого для увеличения его размера;

ПЛОХОЙ ДЕСКРИПТОР bad_desc
: неоткрытый, или уже закрытый, или испорченный file;

ЗАПРЕЩЕННЫЙ ДОСТУП ill_access
права доступа к файлу не позволяют применить к нему данную операцию;

ЗАЩИТА ЗАПИСИ write_pro
носитель, на котором открыт файл, смонтирован "только для чтения" или защищен от записи аппаратным способом (например, выключателем на устройстве или заклеиванием соответсвующего отверстия на гибком диске и т.п.);

ПРЕРВАННАЯ ОПЕРАЦИЯ ipted_op
ОШИБКА ВВОДА/ВЫВОДА io_error **
ПЛОХАЯ ФАЙЛОВАЯ СИСТЕМА bad_fsys **
НЕ ОПРЕДЕЛЕНО undef **

8.7.19. fstype

```
PROCEDURE fstype(file: FILE; VAR type,blocksize: INTEGER);
```

Операция fstype позволяет узнать тип файловой системы, в которой расположен открытый файл file, а также размер блока, используемого для отведения и утилизации пространства на носителе и организации доступа к файлам.

Операция fstype может быть применена только к обычным (ordinary) файлам и директориям. При попытке применить ее к специальным файлам произойдет возбуждение ошибки НЕПОДХОДЯЩИЙ (UNSUITABLE).

Возможно возникновение ошибок:

ПЛОХОЙ ДЕСКРИПТОР	bad_desc	
неоткрытый, или уже закрытый, или испорченный file;		
НЕПОДХОДЯЩИЙ	unsuitable	
ПЛОХАЯ ФАЙЛОВАЯ СИСТЕМА	bad_fsys	**
НЕ ОПРЕДЕЛЕНО	undef	**

8.7.20. flush

```
PROCEDURE flush (file: FILE);
-----------------------------
```

Операция flush обеспечивает проведение мер по обеспечению когеррентности информации файла в памяти и его образа на диске.

При выполнении операции flush на диск заносятся все его изменившиеся атрибуты, ожидаются окончания всех задержанных записей и производится запись всех изменившихся буферов файла.

Операция flush не эквивалентна закрытию файла и последующему его открытию, т.к. права доступа программы к файлу сохраняются неизменными при ее выполнении, но могут измениться при закрытии и последующем открытии файла (не говоря уже про то, что временный файл вообще нельзя открыть после закрытия).

Если при выполнении операции flush не обнаружено изменившихся атрибутов, задержанных записей и изменившихся буферов, операция игнорируется.

Возможно возникновение ошибок:

НЕТ СВОБОДНОГО МЕСТА no_space
: на носителе, где располагается файл, нет свободного пространства, необходимого для увеличения его размера (вызванного записью изменившихся буферов);

ПЛОХОЙ ДЕСКРИПТОР bad_desc
: неоткрытый, или уже закрытый, или испорченный file;

ЗАПРЕЩЕННЫЙ ДОСТУП ill_access
: права доступа к файлу не позволяют применить к нему данную операцию;

ПРЕРВАННАЯ ОПЕРАЦИЯ ipted_op
ОШИБКА ВВОДА/ВЫВОДА io_error **
ПЛОХАЯ ФАЙЛОВАЯ СИСТЕМА bad_fsys **
НЕ ОПРЕДЕЛЕНО undef **

8.7.21. mkfs

```
PROCEDURE mkfs  (device: FILE;
                 fstype: INTEGER;
              blocksize: INTEGER;
                  label: ARRAY OF CHAR;
                   bads: ARRAY OF INTEGER);
-----------------------------------------
```

Операция mkfs инициализирует носитель, установленный на структурированном устройстве dev, и создает на нем новую файловую систему.

Для выполнения операции mkfs нужно иметь права доступа к файлу-устройству по чтению и записи.

Массив bads содержит номера "плохих" блоков на носителе. Файловая система не будет использовать эти блоки в своей работе и включит их в файл "BAD.BLOCKS".

Строка label несет чисто информациолнный смысл, она будет возвращаться в операциях fmount/mount.

Файловая система, созданная операцией mkfs, состоит из корневой директории носителя и следующих двух или трех входов на ней:

".."	ссылка на саму корневую директорию;
"SYSTEM.BOOT"	пустой файл, предназначенный для хранения bootstrap'ного образа системы;
"BAD.BLOCKS"	файл, "закрывающий" плохие блоки на носителе (только если bads>0).

Возможно возникновение ошибок:

ПЛОХОЙ ДЕСКРИПТОР bad_desc
неоткрытый, или уже закрытый, или испорченный файл dev;

ПЛОХОЙ ПАРАМЕТР bad_parm
bads>HIGH(badb)+1

ЗАНЯТО busy
носитель на устройстве уже подмонтирован к файловому дереву или открыт другой задачей;

ЗАПРЕЩЕННЫЙ ДОСТУП ill_access
права доступа к файлу не позволяют применить к нему данную операцию;

ЗАЩИТА ЗАПИСИ write_pro
носитель, смонтированный на устройстве dev, защищен от записи аппаратным способом (например, выключателем на устройстве или заклеиванием соответсвующего отверстия на гибком диске и т.п.);

НЕПОДХОДЯЩИЙ unsuitable
файл dev не является структурированным устройством;

НЕТ СВОБОДНОГО МЕСТА no_space
носитель слишком маленький или в списке "плохих" блоков обнаружен один из тех, что необходимы для нормального устройства файловой системы;

ПРЕРВАННАЯ ОПЕРАЦИЯ ipted_op
ОШИБКА ВВОДА/ВЫВОДА io_error **
ПЛОХАЯ ФАЙЛОВАЯ СИСТЕМА bad_fsys **
НЕ ОПРЕДЕЛЕНО undef **

8.7.22. mount, fmount, unmount, funmount

```
PROCEDURE mount  (path,dev,info: ARRAY OF CHAR;
                      VAR label: ARRAY OF CHAR; ro: BOOLEAN);
PROCEDURE fmount (cd: FILE;
                  path,dev,info: ARRAY OF CHAR;
                      VAR label: ARRAY OF CHAR; ro: BOOLEAN);
------------------------------------------------------------


PROCEDURE unmount (path: ARRAY OF CHAR; method: INTEGER);
PROCEDURE funmount(cd: FILE;
                   path: ARRAY OF CHAR; method: INTEGER);
------------------------------------------------------------
```

Операции mount, fmount предназначены для подмонтирования к файловому дереву поддеревьев, расположенных на носителях, установленных на структурированных устройствах (для краткости - "монтирование устройства").

Операции unmount, funmount демонтируют с файлового дерева ранее подмонтированные к нему поддеревья.

Операции mount, unmount являются точными эквивалентами операций fmount(BIO.cd,.....), funmount(BIO.cd,.....).

Монтирование устройства возможно в любое место файлового дерева. Для монтирования устройства необходимо существование директории, на которую данное устройство будет смонтировано.

Маршрут, ведущий к директории, на которую будет смонтировано устройство, задается параметром path, файл устройства, на котором будет монтироваться носитель, задается параметром dev. Параметр info передается драйверу в момент монтирования в качестве параметра запроса MOUNT и может содержать специфическую информацию, полезную (или даже необходимую) драйверу для осуществления правильной работы с устройством. В переменной label возвращается "метка тома", которая носит чисто информационный (символический) характер и никак не влияет на работу файловой системы.

При монтировании устройства имеется возможность запретить любые модификации информации, расположенной на устройстве, независимо от индивидуальных прав доступа к каждому конкретному файлу или директории. Для этого необходимо передать операции монтирования read_only-параметр (ro), равный TRUE. Следует обратить внимание, что такое запрещение записи сродни физической защите носителя (или устройства от записи) и не сказывается на возможности программ открывать на этом устройстве файлы с требованием прав записи в операции open. Такие права будут предоставляться вне зависимости от режима, в котором смонтировано устройство (что обеспечит возможность частичной работоспособности многих программ), но при попытке модифицировать информацию на носителе, смонтированном с параметром read_only, будет возникать ошибка ЗАЩИТА ЗАПИСИ

(WRITE PROTECT), и информация модифицироваться не будет. Заметьте, что ошибка ЗАЩИТА ЗАПИСИ будет возникать в момент применения операции, изменяющей информацию на носителе, в противовес ошибке НАРУШЕНИЕ СЕКРЕТНОСТИ, происходящей в момент открытия файла.

Поскольку ссылка на устройство также хранится в файловом дереве (см. mknode) и снабжена набором прав доступа, очевидно, что для подмонтировании носителя на устройстве с требованием возможности записи на него необходимо иметь право записи на это устройство и на директорию, на которую будет произведено подмонтирование, в противном случае возникнет ошибка НАРУШЕНИЕ СЕКРЕТНОСТИ в момент подмонтирования.

После успешного подмонтирования файлового поддерева на директорию во всех маршрутах, проходящих через эту директорию, она будет ассоциироваться с корневой директорией подмонтированного носителя. Вход ".." на этой директории, тем не менее, сохранит свое нормальное значение. Прежнее содержимое директории (если таковое имеется) станет недоступным вплоть до момента демонтирования.

Процесс монтирования не сопряжен с необходимостью записи информации на носитель, к которому идет подмонтирование, что обеспечивает возможность протягивать цепочки смонтированных поддеревьев даже через смонтированные с параметром read_only носители. Никаких следов подмонтирования и демонтирования на носителе не остается.

Устройство не может быть смонтировано больше одного раза, и не может быть подмонтировано больше одного устройства на один узел файлового дерева. Этим гарантируется "деревянность" дерева, то есть отсутствие в нем циклов.

Операция демонтирования позволяет отмонтировать файловое поддерево от директории. Для успешного демонтирования необходимо указать директорию, на которую ранее было произведено монтирование, и метод обработки "плохих" файлов. В случае успешного демонтирования устройства директория восстанавливает свои нормальные, обычные свойства. Если указанная директория не является местом монтирования, возбуждается ошибка НЕ ОПРЕДЕЛЕНО (UNDEFUNED).

Для успешного демонтирования носителя все файлы на нем должны быть закрыты, иначе возникает ошибка ЗАНЯТО (BUSY), и демонтирование не выполняется. Причины наличия на устройстве незакрытых файлов могут быть следующие. Первая: наличие активных или остановленных задач, открывших или создавших файлы на этом носителе. Вторая - наличие "плохих" файлов. Первая причина не требует дополнительных средств для борьбы со своими последствиями.

"Плохие" файлы на носителе могут образовываться в результате невозможности (из-за аппаратной ошибки, например) записать на носитель изменившиеся атрибуты файлов или других внутренних структур данных. В этой ситуации задача, осуществлявшая операции закрытия (close или purge) или утрамбовывания (flush) соответствующего файла, получает

извещение о случившейся ошибке, но файл остается незакрытым и помечается как "плохой". При демонтировании носителя, на котором остались "плохие" файлы, играет роль значение параметра method. Существует три метода демонтирования таких носителей.

1) method=0. Делается попытка демонтирования носителя, и если на нем есть незакрытые или "плохие" файлы, возбуждается ошибка ЗАНЯТО (BUSY), и носитель не демонтируется.

2) method=1. Делается попытка демонтирования носителя, и если на нем етсь "плохие" файлы, все "плохие" файлы игнорируются, возбуждается ошибка ЗАНЯТО (BUSY), но носитель демонтируется!

3) method=2. Делается попытка демонтирования носителя, и если на нем есть "плохие" файлы, все "плохие" файлы игнорируются, ошибка ЗАНЯТО не возбуждается, и носитель демонтируется. Если при указании метода 2 все равно возбуждается ошибка ЗАНЯТО, это свидетельствует о том, что в системе в данный момент существуют задачи, имеющие открытые на этом носителе файлы. Необходимо либо дождаться их завершения, либо принудительно завершить их.

Возможно возникновение ошибок:

ЭТО НЕ ДИРЕКТОРИЯ not_dir
имя в префиксе маршрута указывает не на директорию, или значение cd не является директорией;

ПЛОХОЙ ДЕСКРИПТОР bad_desc
плохое значение параметра или переменной cd;

НЕТ ТАКОГО no_entry
на директории, указываемой маршрутом, не найден файл с указанным именем, или не найдена директория для одного из имен в префиксе маршрута, или не найдено устройство;

ЗАНЯТО busy
носитель на устройстве подмонтирован к файловому дереву или открыт другой задачей;

ЗАПРЕЩЕННЫЙ ДОСТУП ill_access
права доступа к файлу не позволяют применить к нему данную операцию;

ЗАЩИТА ЗАПИСИ write_pro
носитель на устройстве dev защищен от записи аппаратным способом (например, выключателем на устройстве или заклеиванием соответсвующего отверстия на гибком диске и т.п.);

НЕПОДХОДЯЩИЙ unsuitable
файл dev не является структурированным устройством;

НЕ ОПРЕДЕЛЕНО undef

попытка демонтирования от директории, на которую ничего не смонтировано;

НЕТ ПАМЯТИ	no_memory	
НАРУШЕНИЕ СЕКРЕТНОСТИ	sec_vio	
ПЛОХОЕ ИМЯ	bad_name	
ПРЕРВАННАЯ ОПЕРАЦИЯ	ipted_op	
ПЛОХАЯ ФАЙЛОВАЯ СИСТЕМА	bad_fsys	**
ОШИБКА ВВОДА/ВЫВОДА	io_error	**
НЕ ОПРЕДЕЛЕНО	undef	**

8.7.23. mknode, fmknode

```
PROCEDURE mknode (          path,name: ARRAY OF CHAR);
PROCEDURE fmknode(cd: FILE; path,name: ARRAY OF CHAR);
-------------------------------------------------
```

Операции mknode, fmknode позволяют создать ссылающийся на устройство узел в файловом дереве.

Операция mknode является точными эквивалентом операции fmknode(BIO.cd,.....).

Операция mknode создает специальный узел в файловой системе на директории, указываемой префиксом маршрута path, и с именем из маршрута path, ссылающийся на устройство с именем name. Все последующие открытия этого узла будут приводить к открытию устройства с именем name. При создании узла необязательно и не проверяется наличие драйвера устройства с именем name. При создании и привязывании узла ему устанавливаются права доступа из переменной cmask.

Следует иметь в виду, что операции chaccess и chowner не смогут изменить права доступа и хозяина узла, созданного операцией mknode. Они будут изменять только права доступа и хозяина самого открытого устройства, но при закрытии файла устройства эти изменения не будут сохранены в соответствующем узле.

Изменения прав доступа к устройству можно добиться, уничтожив узел и создав вместо него новый с другими правами доступа.

Уничтожение узла, созданного операцией mknode, может быть произведено обычной операцией unlink.

Возможно возникновение ошибок:

ЭТО НЕ ДИРЕКТОРИЯ not_dir
: имя в префиксе маршрута указывает не на директорию, или значение cd не является директорией;

НОСИТЕЛЬ ПЕРЕПОЛНЕН fsys_full
: на носителе, где создается узел, переполнена таблица файлов;

НЕТ СВОБОДНОГО МЕСТА no_space
: на носителе, где создается узел, нет свободного пространства, необходимого для увеличения размера директории;

НЕТ ТАКОГО no_entry
: не найдена директория для одного из имен в префиксе маршрута;

ПЛОХОЙ ДЕСКРИПТОР bad_desc

плохое значение параметра или переменной cd;

НАРУШЕНИЕ СЕКРЕТНОСТИ sec_vio
нет права записи в директорию, к которой ведется привязывание;

ПЛОХОЕ ИМЯ bad_name
неправильное имя в указанном маршруте, или неправильное имя устройства;

ПРЕРВАННАЯ ОПЕРАЦИЯ ipted_op
ОШИБКА ВВОДА/ВЫВОДА io_error **
ПЛОХАЯ ФАЙЛОВАЯ СИСТЕМА bad_fsys **
НЕ ОПРЕДЕЛЕНО undef **

8.7.24. kind

```
PROCEDURE kind(file: FILE): BITSET;
-----------------------------------
CONST  is_dir is_tty is_disk is_spec is_hidd is_sys
```

Операция kind позволяет определить вид файла. В случае применения к BIO.null, неоткрытому или уже закрытому файлу возбуждается ошибка ПЛОХОЙ ДЕСКРИПТОР и возвращается {}.

Файл-устройство может быть определен так:

```
kind(file) * (is_tty+is_disk+is_spec) # {}
```

Вид is_hidd может комбинироваться с другими видами файлов и означает, что при открытии файла в соответствующем входе в директорию был выставлен признак hidden (скрытый), или файл был создан с параметром mode, содержащим 'h'.

Возможно возникновение ошибок:

ПЛОХОЙ ДЕСКРИПТОР bad_desc
плохое значение параметра file.

8.7.25. chmode

```
PROCEDURE chmode(file: FILE; mode: ARRAY OF CHAR);
--------------------------------------------------
```

Изменяет способы доступа, с которыми файл был открыт или создан. Параметр mode такой же, как в операции open. В случае невозможности сменить способ доступа к файлу возбуждается ошибка НАРУШЕНИЕ СЕКРЕТНОСТИ.

Синтаксис строки mode такой же, как у операций create и open (см.).

В случае возникновение ошибки НАРУШЕНИЕ СЕКРЕТНОСТИ (SECURITY VIOLATION) метод доступа к файлу не изменяется.

Возможно возникновение ошибок:

```
ПЛОХОЙ ДЕСКРИПТОР           bad_desc
    плохое значение параметра file;

НАРУШЕНИЕ СЕКРЕТНОСТИ       sec_vio
```

8.7.26. chcmask, access, owner

```
CONST  (* protection bits *)
  own_exec      gro_exec        oth_exec
  own_write     gro_write       oth_write
  own_read      gro_read        oth_read
  own_search    gro_search      oth_search

  run_uid       run_priv        unlink_pro

PROCEDURE chcmask(mask: BITSET);
--------------------------------
PROCEDURE access(file: FILE): BITSET;
-------------------------------------
PROCEDURE owner(file: FILE; VAR uid,uid: INTEGER);
--------------------------------------------------
```

Операция chcmask позволяет установить маску защиты, которая будет приписываться всем создаваемым с помощью create, mkdir и mknode файлам.

Операция access позволяет узнать маску защиты открытого или созданного файла.

Операция owner позволяет узнать uid и gid влядельца открытого или созданного файла.

Следует иметь в виду следующее: (xxx = own, gro, oth).

Признаки защиты xxx_exec и xxx_search - это один и тот же признак, отличающийся только интерпретацией для директорий, обычных файлов и файлов-устройств.

Признаки xxx_read, xxx_write, xxx_search, (xxx_exec), будучи выставленными, запрещают (!) соответствующий метод доступа.

Признак unlink_pro, будучи выставлен, запрещает отвязывать (unlink) файл до тех пор, пока этот признак не уберут (см. chaccess).

Признаки run_priv, run_uid действуют только для исполняемых файлов и означают, что при исполнении файла должен быть выставлен привилегированный режим или должен быть выставлен идентификатор группы (group id) и пользователя (user id) соответственно.

Выставление в cmask признаков run_priv, run_uid разрешено только в привилегированном режиме.

В переменной cmask всегда содержатся признаки защиты для создаваемых файлов. Переменная инициализируется в момент запуска задачи путем сканирования строки из ОКРУЖЕНИЯ (Environment) по имени "CMASK". При отсутствии такой строки или в случае ошибки в ее синтаксисе cmask инициализируется значением по умолчанию:

```
cmask = oth_write+oth_read+oth_exec+gro_write
```

, то есть в создаваемые файлы запись запрещена для всех, кроме владельца (owner), чтение и исполнение (или просмотр) разрешено только для владельца и членов его группы.

Возможно возникновение ошибок:

ПЛОХОЙ ДЕСКРИПТОР bad_desc
: плохое значение параметра file в операции access;

ДЛЯ ПРИВИЛЕГИРОВАННЫХ su_only
: попытка выставить в cmask признаки run_priv, run_uid в непривилегированном режиме. При возникновении этой ошибки признаки run_priv, run_uid игнорируются, остальные признаки выставляются в cmask;

8.7.27. chaccess, chowner

```
PROCEDURE chaccess(file: FILE; mask: BITSET);
PROCEDURE chowner (file: FILE; owner,group: INTEGER);
-------------------------------------------------
```

Операции chaccess, chowner позволяют изменить маску защиты файла и его владельца соответственно.

Для изменения признаков защиты и смены владельца необходимо быть владельцем файла или находиться в привилегированном режиме.

Описание признаков защиты см. в описании chcmask.

После смены владельца (owner) у файла вернуть файл прежнему владельцу может только новый владелец или привилегированный пользователь.

При изменении владельца файла или признаков защиты права доступа к данному файлу программы, которая произвела эти изменения, остаются прежними вплоть до закрытия файла.

Например, программа, являющаяся владельцем файла, имеет право доступа по записи к нему и выставила признак запрета записи для владельца. Несмотря на это, право записи в этот файл останется для нее прежним (т.е. она сможет продолжать записывать данные в файл) вплоть до закрытия файла. При повторном (с момента смены признаков защиты) открытии файла этой программой или любой другой с требованием права записи открытие будет отклонено из-за ошибки НАРУШЕНИЕ СЕКРЕТНОСТИ.

Возможно возникновение ошибок:

ПЛОХОЙ ДЕСКРИПТОР bad_desc
: плохое значение параметра file;

ДЛЯ ПРИВИЛЕГИРОВАННЫХ su_only
: попытка выставить признаки run_priv, run_uid в непривилегированном режиме. При возникновении этой ошибки признаки run_priv, run_uid игнорируются, остальные признаки выставляются;

НАРУШЕНИЕ СЕКРЕТНОСТИ sec_vio
: попытка изменить признаки или владельца у "чужого" файла (у файла, владельцем которого программа не является).

8.7.28. get_attr, set_attr

```
PROCEDURE get_attr(file: FILE; no: INTEGER; VAR val: WORD);
PROCEDURE set_attr(file: FILE; no: INTEGER;     val: WORD);
-----------------------------------------------------------
```

```
CONST a_links a_inode a_ctime a_wtime a_gid a_uid a_pro
```

Операции get_attr, set_attr позволяют прочитать и записать некоторые полезные атрибуты файла, не доступные иным путем.

Прочитать любой атрибут имеет право любая программа у любого файла (в любое время, и вне зависимости от местоположения компутера, Земли, Луны, Солнца и созвездия Аль Забих и др. небесных тел).

Изменять атрибуты операцией set_attr имеют право:

```
a_links   только привилегированные;
a_inode   только привилегированные;
a_ctime   только владелец или привилегированные;
a_wtime   только владелец или привилегированные;
a_gid     никто;
a_uid     никто;
a_pro     никто.
```

Атрибуты a_gid, a_uid, a_pro изменяются операциями chaccess, chowner, поэтому изменение их операцией set_attr запрещено.

Возможно возникновение ошибок:

```
ПЛОХОЙ ДЕСКРИПТОР           bad_desc
```
плохое значение параметра file;

```
ПЛОХОЙ ПАРАМЕТР             bad_parm
```
попытка изменить a_gid, a_uid, a_pro в set_attr;

```
ДЛЯ ПРИВИЛЕГИРОВАННЫХ       su_only
```
попытка изменить a_links, a_inode в непривилегированном режиме;

```
НАРУШЕНИЕ СЕКРЕТНОСТИ       sec_vio
```
попытка изменить a_ctime, a_wtime "чужого" файла (файла, владельцем которого программа не является).

8.7.29. dir_walk, end_walk, restart_walk

```
CONST sort_none         sort_reverse
      sort_name         sort_ext
      sort_time         sort_cre
      sort_eof          sort_dirfwd

PROCEDURE dir_walk(cd: FILE; sort: INTEGER);
PROCEDURE end_walk(cd: FILE);
-----------------------------------------
PROCEDURE restart_walk(cd: FILE);
--------------------------------
```

Операция dir_walk начинает итерацию (прогулку по) директории. Операция end_walk завершает итерацию директории. Операция restart_walk позволяет начать итерацию директории заново.

Для итерации директории программа должна обладать правом доступа к данной директории по просмотру, т.е. при открытии директории, которую программа собирается итерировать, следует указать флаг 'x' или 'X' в параметре mode (см. open).

Директория BIO.cd всегда открывается (если вообще открывается) с максимально допустимыми правами доступа ('X' в mode).

В момент начала итерации директории ее содержимое читается в память и, возможно, сортируется. Поэтому на протяжении итерации директории могут возникнуть две следующие ситуации:

1) за время итерации к директории привязали файл, в результате чего возник вход с новым именем, который не будет проитерирован.

2) за время итерации от директории отвязали файл, в результате чего исчез вход, который тем не менее будет проитерирован. В связи с этим не следует считать фатальной ошибку НЕТ ТАКОГО (NO ENTRY) при выполнении операции get_entry.

Параметр sort может быть образован как сумма (*):

none	не сортировать;
reverse	сортировка в обратном порядке;
name	сортировка по именам;
ext	сортировать по расширителям (**);
time	сортировка по времени;
cre	сортировка по времени создания (а не записи);
eof	сортировка по размерам;
dirfwd	директории перед файлами (иначе вперемежку).

*) в версиях системы до 1.3 реализована только сортировка sort_none.

**) для файлов с одинаковым расширителем может быть

произведена, если требуется, сортировка по именам, по временам или по размерам.

При попытке повторно применить операцию dir_walk к одной и той же директории возникнет ошибка ЗАНЯТО (BUSY). Это вовсе не означает, что директория не может одновременно итерироваться несколькими программами. Смысл этой ошибки состоит в том, что одна и та же программа пробует одновременно прогуляться по одной и той же директории. Причина запрета таких множественных прогулок очевидна - неизвестно, кто и где гуляет (да и сортировки могут быть разные).

Операция restart_walk позволяет начать итерацию директории с начала. Restart_walk не производит чтения и сортировки директории, а только переставляет на начало указатель текущего входа итерации директории. Очевидно, что restart_walk можно выполнять только после dir_walk и до end_walk.

Следует иметь в виду, что end_walk завершает итерацию директории и освобождает всю занятую память, но не закрывает самой директории, которая (в случае необходимости) должна быть закрыта операцией close.

Будьте внимательны, не закройте случайно BIO.cd!

Возможно возникновение ошибок:

ЭТО НЕ ДИРЕКТОРИЯ not_dir
значение cd не является директорией;

ПЛОХОЙ ДЕСКРИПТОР bad_desc
плохое значение параметра cd;

ПЛОХОЙ ПАРАМЕТР bad_parm
неправильное значение параметра sort;

НЕПОДХОДЯЩИЙ unsuitable
попытка закончить или рестартовать итерацию директории, для которой не выполнялось dir_walk;

ЗАНЯТО	busy	
НЕТ ПАМЯТИ	no_memory	
ЗАПРЕЩЕННЫЙ ДОСТУП	ill_access	
ПРЕРВАННАЯ ОПЕРАЦИЯ	ipted_op	
ПЛОХАЯ ФАЙЛОВАЯ СИСТЕМА	bad_fsys	**
ОШИБКА ВВОДА/ВЫВОДА	io_error	**
НЕ ОПРЕДЕЛЕНО	undef	**

8.7.30. get_entry

```
CONST e_dir e_hidden e_esc e_sys

PROCEDURE get_entry(cd: FILE; VAR name: ARRAY OF CHAR;
                              VAR mode: BITSET): BOOLEAN;
-----------------------------------------------------------
```

Операция get_entry выдает информацию об очередном (если имеется) входе итерируемой директории или сигнализирует об окончании итерации.

Операция get_entry возвращает значение TRUE, если удалось получить информацию об очередном входе.

Операция get_entry возвращает значение FALSE, если на итерируемой директории больше нет входов или get_entry применяется к недиректории или к директории, к которой не применялся dir_walk (см.).

В параметре name возвращается имя входа, в параметре mode - признаки входа. Такими признаками могут быть

e_dir	вход ссылается на директорию;
e_esc	вход ссылается на узел-устройство;
e_hidden	имя во входе считается "скрытым";
e_sys	вход ссылается на системный-файл,

а также их разумные комбинации.

Последние два признака (hidden, sys) имеют чисто утилитарный характер, то есть используются только утилитами для принятия решения о включении или невключении файла во множество файлов, над которыми выполняются операции.

Возможно возникновение ошибок:

ЭТО НЕ ДИРЕКТОРИЯ not_dir
значение cd не является директорией;

ПЛОХОЙ ДЕСКРИПТОР bad_desc
плохое значение параметра cd;

НЕПОДХОДЯЩИЙ unsuitable
попытка закончить или рестартовать итерацию директории, для которой не выполнялось dir_walk;

При любой из ошибок get_entry возвращает FALSE.

8.7.31. open_paths, close_paths, get_paths

```
TYPE PATHs;
-----------

VAL here: PATHs;  empty: PATHs;
-------------------------------
```

Значениями типа PATHs являются упорядоченные множества директорий, на которых будет производиться поиск операцией lookup (см.). Имеются два предопределенных значения типа PATHs.

Значение empty суть предопределенное неопределенное значение типа PATHs. Значение here построено операцией

```
open_paths(here,".")
```

и может быть использовано вместо любого иного значения в качестве умолчания или при неудачном открытии требуемых путей. Значение here в качестве множества директорий, относительно которых будет интерпретироваться маршрут, содержит только текущую директорию, и поэтому:

```
lookup(here,file,name,mode)
```

абсолютно эквивалентно

```
open(file,name,mode).
```

```
PROCEDURE open_paths (VAR dirs: PATHs; paths: ARRAY OF CHAR);
PROCEDURE get_paths  (VAR dirs: PATHs; envnm: ARRAY OF CHAR);
PROCEDURE close_paths(VAR dirs: PATHs);
---------------------------------------------------------------
```

Операции open_paths, get_paths, close_paths позволяют создавать новые значение типа PATHs по спецификации, заданной строкой paths (open_paths), по спецификации, извлеченной из ОКРУЖЕНИЯ (Environment), по имени envnm (get_paths), а также уничтожать значения типа PATHs (close_paths).

Синтаксис спецификации множества путей поиска файлов следующий:

```
paths   = маршрут { пробелы маршрут } 000c .
пробелы = " " { " " } .
```

В случае неправильного задания спецификации в операциях open_paths, get_paths можут возникнуть ошибки ПЛОХОЕ ИМЯ (BAD NAME) или ПЛОХОЙ ПАРАМЕТР (BAD PARAMETER). Если в ОКРУЖЕНИИ не найдено имя envnm, заданное в get_paths, возбуждается ошибка НЕТ ТАКОГО (NO ENTRY).

Порядок поиска (и вообще Порядок) в порожденном значении типа PATHs соответствует порядку перечисления маршрутов в

спецификации.

Операция close_paths закрывает все директории, составляющие множество путей (если они были открыты), и уничтожает значение типа PATHs.

Возможно возникновение ошибок:

ПЛОХОЙ ПАРАМЕТР	bad_parm
ПЛОХОЕ ИМЯ	bad_name
НЕТ ПАМЯТИ	no_memory

8.7.32. lookup

```
PROCEDURE lookup  (dirs: PATHs;
                   VAR file: FILE; name,mode: ARRAY OF CHAR);
-------------------------------------------------------------
```

Операция lookup осуществляет поиск и открытие файла, интерпретируя маршрут к файлу относительно всех директорий из множества dirs. В случае успешного поиска открытие файла происходит в точности так же, как в операции open (см.).

Следует иметь в виду следующие важные особенности операции lookup и значения типа PATHs.

Если операции lookup указывается абсолютный маршрут к файлу (начинается от корня), поиска на директориях dirs не происходит.

После выполнения операции open_paths директории из множества, определяющего места поиска, остаются неоткрытми (!) и будут открываться лишь по мере необходимости. После открытия директории из множества dirs она остается открытой вплоть до выполнения операции close_paths.

Директория, указанная в спецификации как ".", напротив, никогда (!) не открывается, вместо нее всегда используется значение переменной BIO.null.

Эти два свойства необходимо учитывать при задании спецификации, так как можно получить достаточно неожиданные эффекты, комбинируя вызовы open_paths, chdir, lookup. Поэтому рекомендуется избегать задания в спецификации относительных маршрутов (кроме "."), поскольку их интерпретация может существенно меняться при смене директории, а apriory неизвестно, на какой директории произойдет действительное открытие каждого из элементов множества dirs.

Следует также иметь в виду, что если операции lookup не удается открыть один из элементов (упорядоченного) множества путей по причинам НЕТ ТАКОГО (NO ENTRY) или НАРУШЕНИЕ СЕКРЕТНОСТИ (SECURITY VIOLATION), lookup игнорирует эту ошибку (но не какую-либо другую), откладывая открывание этого элемента до лучших времен, и переходит к попытке открытия следующего элемента множества.

Возможно возникновение ошибок:

НЕТ ТАКОГО no_entry
: файл не найден относительно ни одной из директорий;

ПЛОХОЙ ДЕСКРИПТОР bad_desc
: плохое значение параметра dirs;

ЭТО НЕ ДИРЕКТОРИЯ not_dir
: в значении dirs один из маршрутов указывает не на директорию,

а также любой из ошибок операции open (см.).

8.7.33. lock, unlock

```
PROCEDURE lock  (time_out: INTEGER; file: FILE);
PROCEDURE unlock(                   file: FILE);
-----------------------------------------------
```

Операции lock и unlock позволяют двум или более программам (да и процессам тоже!) синхронизовать свои операции над файлами.

Операция lock осуществляет попытку захвата файла. Захват файла может быть удачным или неудачным, что легко проверить, тестируя done и error.

В случае задания time_out<0 (-1, например) попытка захватить уже захваченный другим процессом (возможно, другой программы) файл затянется до тех пор, пока захватившая файл программа не отпустит его, или не завершится, или не будет остановлена, программа пытающаяся выполнить этот lock.

В случае задания time_out=0 производится попытка немедленного захвата файла, и если это не удалось, возбуждается ошибка ЗАНЯТО (BUSY).

В случае указания time_out>0 производится попытка захвата файла, и если это не удалось, производится ожидание момента, когда захватившая файл программа отпустит его, или завершится, или будет остановлена программа, пытающаяся выполнить эту операцию lock, или будет исчерпано время (в милисекундах) time_out. В последнем случае будет возбуждена ошибка ВРЕМЯ ВЫШЛО (TIME OUT).

Операция unlock производит немедленное освобождение ранее захваченного файла. Если при этом другие процессы находятся в ожидании захвата файла, первый из них будет продолжен с удачным захватом файла.

Захват файла операцией lock действует только на выполнение операции lock над тем же файлом, но не на другие операции с ним. Таким образом, lock/unlock могут служить синхронизирующими примитивами только для процессов, разработанных с учетом возможности одновременной работы с файлами, но не для произвольных процессов.

Захват и освобождение файла необходимо только для процессов, желающих выполнить последовательность операций над файлом во взаимоисключающем режиме. Взаимное исключение задач и процессов при выполнении атомарных операций обеспечивается самой файловой системой.

Все захваченные (и не освобожденные) процессом файлы будут освобождены в случае аварийного (или нормального) завершения процесса.

Следует иметь в виду, что в версиях системы до 1.3 при закрытии захваченного файла (close) не производится его автоматическое освобождение. Тем самым, программа, закрывшая

захваченный ею файл, теряет возможность его освободить вплоть до своего завершения. Будьте осторожны!

Возможно возникновение ошибок:

ПЛОХОЙ ДЕСКРИПТОР bad_desc
плохое значение параметра file;

ВРЕМЯ ВЫШЛО time_out
файл не удалось захватить за указанное время;

ЗАНЯТО busy
файл не удалось захватить немедленно.

ЧАСТЬ 9. ДИСК В ФАЙЛОВОЙ СИСТЕМЕ

Предисловие соавтора

Появлению описания диска мы обязаны А.Хапугину (Hady). Он является автором утилит проверки и починки файловой системы fschk и fsdb, и это описание адресует в помощь пользователям этих утилит.

Соавтор

Администратору системы часто приходится иметь дело с магнитными дисками: форматировать, инициализировать, копировать и, что самое страшное, редактировать низкоуровневыми средствами. Для этого необходимо иметь представление об устройстве носителя в ОС Excelsior.

> Ну и ну, сколько же можно поворачивать?! Никакая это не дорожка, а просто юла какая-то...
>
> Л.Кэрролл. Алиса в Зазеркалье

Диск представляет собой пакет, содержащий несколько рабочих поверхностей (возможно, одну). Рабочая поверхность аппаратно разбита на дорожки, каждая из которых содержит несколько секторов.

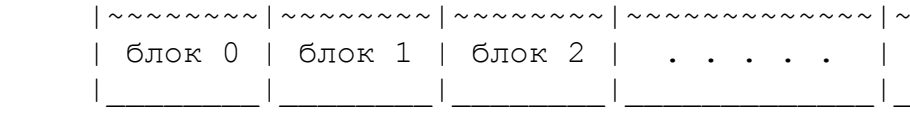
Адресация на диске, как правило, реализована аппаратно, зависит от контроллера диска и его драйвера.

Описанную структуру принято называть физической структурой диска. Информация о физической структуре диска поставляется ОС драйвером и используется только для оптимизации операций чтения/записи.

Файловая система организует другое разбиение диска, которое мы будем называть логической структурой.

9.1. Блок

Логической единицей адресации диска в ОС Excelsior является БЛОК. Размер блока на всех дисках равен 4К (4096) байт. Блоки нумеруются с 0, количество блоков определяется емкостью носителя.



Номер блока на диске называется далее ФИЗИЧЕСКИМ НОМЕРОМ (ФН).

Блок с ФН 0 зарезервирован для программы начальной загрузки; он не может быть использован для хранения файлов.

Блок с ФН 1 называется СУПЕРБЛОКОМ и используется исключительно системой (см 9.2).

Блоки с ФН от 2 до некоторого номера SYS, зависящего от размера диска, используются исключительно системой и содержат в себе ДЕСКРИПТОРЫ ФАЙЛОВ (или ИНОДЫ, см. 9.3).

Остальные блоки используются для хранения ФАЙЛОВ (см. 9.5).

9.2. Суперблок

В суперблоке содержится следующая информация:

- символьное имя носителя;
- количество файлов на носителе;
- количество блоков на носителе;
- карта файлов;
- карта блоков.

9.2.1. Метка носителя

Метка носителя - это строка в кодировке КОИ-8 длиной до 16 символов. Терминатором строки считается символ с кодировкой 0с. Метка не может содержать символы "/", "[", "]", ":".

9.2.2. Количество файлов

Количество файлов - 32-разрядное целое число, равное максимально возможному количеству файлов на диске (количеству дескрипторов файлов на диске).

9.2.3. Количество блоков

Количество блоков - 32-разрядное целое число, равное общему количеству блоков на диске, вычисленному при инициализации диска.

9.2.4. Карта файлов

Карта файлов - массив битов. Биты в карте нумеруются от 0 до N-1, где N - количество файлов на диске. Бит с номером J соответствует иноду с номером J. Если бит с номером J выставлен в "1", то инод с номером J считается свободным, в противном случае - занятым.

9.2.5. Карта блоков

Карта блоков - массив битов. Биты в карте нумеруются от 0 до N-1, где N - количество блоков на диске. Бит с номером J соответствует блоку с номером J. Если бит с номером J выставлен в "1", то блок с номером J считается свободным, в противном случае - занятым.

9.3. Область дескрипторов файлов

ОБЛАСТЬ ДЕСКРИПТОРОВ ФАЙЛОВ представляет собой одномерный массив 64-х байтовых записей с информацией о файле - инодов (см. 9.4).

Иноды нумеруются от 0 до N-1, где N - количество файлов на диске. Каждый инод либо является описанием файла, либо не занят системой.

Номер инода, описывающего файл, называется НОМЕРОМ ФАЙЛА и является внутрисистемным индикантом данного файла, так как его достаточно для организации доступа к файлу.

Инод с номером 0 на всех носителях используется для КОРНЕВОЙ ДИРЕКТОРИИ.

Инод с номером 1 зарезервирован для организации загрузки системы с данного диска.

Инод с номером 2 зарезервирован для закрытия дефектных блоков диска.

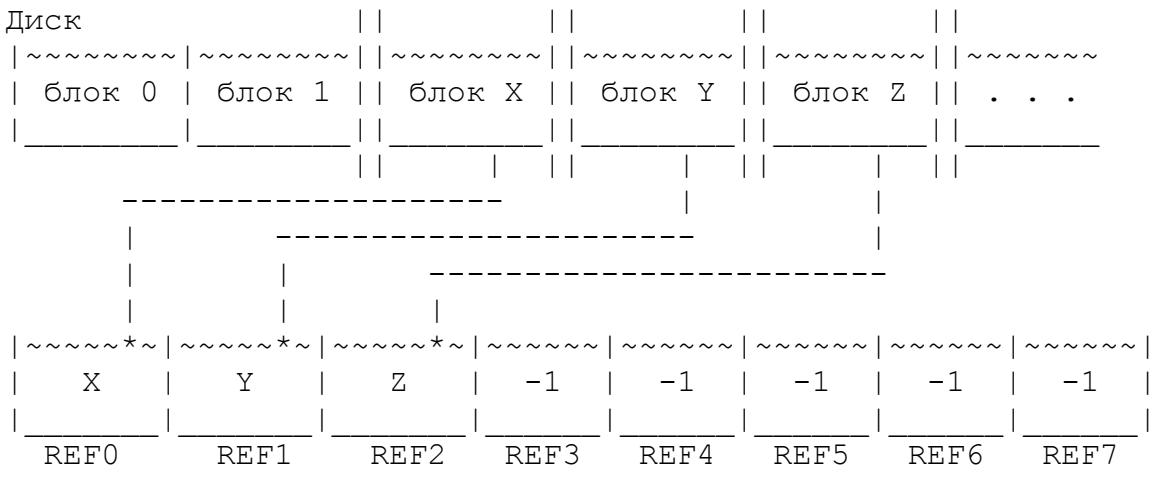
9.4. Дескриптор файла

Дескриптор файла - инод (от англ. "index node") - это специальная запись размером 64 байта, содержащая информацию ОС о файле. В ней находится:

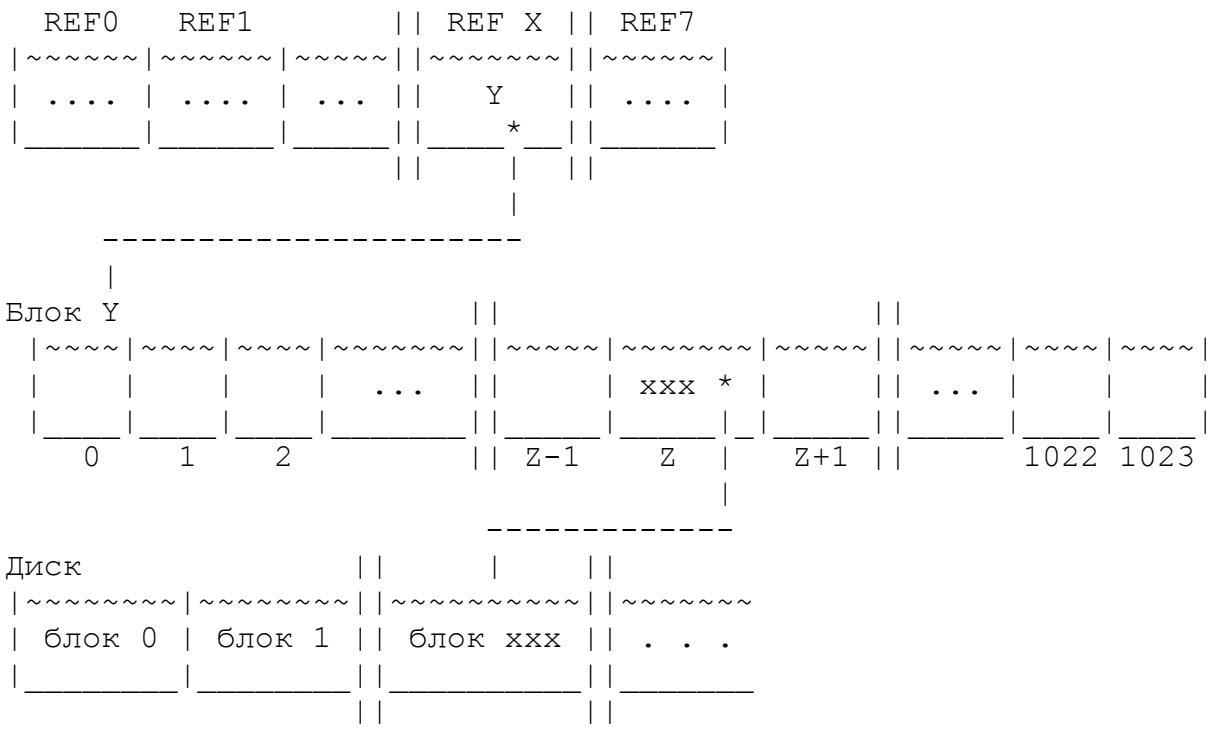
- таблица соответствия логических номеров (ЛН) (см. 9.4) блоков физическим номерам (ФН),
- время создания файла,
- время последней записи в файл,
- информация о правах доступа пользователей к файлу,
- размер файла в байтах,
- количество ссылок на файл
- специальные признаки файла.

9.4.1. Таблица соответствия номеров блоков

ТАБЛИЦА СООТВЕТСТВИЯ ЛН и ФН - REF (от англ. "REFerence" - ссылка) представляет собой массив из 8 целых чисел. Каждый элемент этого массива может ссылаться на какой-нибудь блок диска или быть незанятым. В первом случае элемент содержит ФН блока, на который он ссылается, во втором - отрицательное число.



Для файлов, содержащих не более 8 блоков, элемент таблицы с номером, равным ЛН, ссылается на соответствующий ему блок. Если файл содержит более 8 блоков (32Кбайт), то элементы данного массива ссылаются не на блоки с информацией файла, а на так называемые ИНДЕКСНЫЕ блоки - блоки, содержащие ФН блоков файла. В этом случае блок с информацией файла находится по следующей схеме:



где X=(ЛН DIV 1024); Z=(ЛН MOD 1024); xxx - ФН блока, соответствующего ЛН.

Физический блок, на который есть ссылка из REF или индексного блока файла, должен быть помечен как занятый в карте блоков (см. 9.2.5).

Значение номера блока, большее максимального номера на диске, считается некорректным и должно быть заменено на "-1".

9.4.2. Время создания файла

Время создания файла - 32-разрядное целое число, равное времени создания файла, записанному в стандарте ОС Excelsior.

9.4.3. Время модификации файла

Время модификации файла - 32-разрядное целое число, равное времени последней записи в данный файл, записанному в стандарте ОС Excelsior.

9.4.4. Информация о правах доступа

Информация о правах доступа пользователей к файлу содержится в 4 байтах.

9.4.5. Размер файла

Размер файла - 32-разрядное целое число, в котором содержится размер файла в байтах (EOF - от End Of File). При этом система полагает, что файл содержит байты с номерами [0..EOF-1].

9.4.6. Количество ссылок на файл

Количество ссылок на файл (links) - 32-разрядное целое число, обозначающее количество ссылок на данный файл из директорий (см. 9.6.1.2). Если это число равно 0, то данный инод считается свободным и должен быть помечен в карте файлов как свободный. Если это поле больше нуля, то данный инод считается занятым и должен быть помечен в карте файлов как занятый. Значение, меньшее 0 или большее максимального номера файла считается некорректным и должно быть заменено на 0.

9.4.7. Специальные признаки

Специальные признаки файла - массив битов. В настоящей реализации используются следующие признаки:

- длинный файл;
- директория;
- псевдофайл;

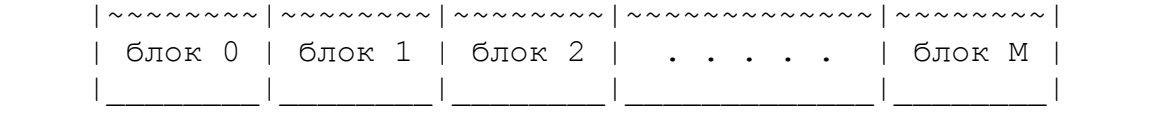
а) признак длинного файла (long) выставляется в том случае, если файл содержит более 8 блоков. В этом случае массив REF файла содержит номера не блоков с информацией, а индексных блоков (см. 9.4.1).

б) признак директории (dir) выставляется, если дескриптор описывает файл специального вида - директорию (см. 9.6).

в) признак псевдофайла (escape) обозначает, что данный дескриптор не является описанием файла. В байтах такого дескриптора, предназначенных для REF, содержится логическое имя драйвера некоторого устройства в стандарте ОС Excelsior.

9.5. Файл

Файл с точки зрения ОС представляет собой одномерный массив байтов, пронумерованых от 0 до EOF-1 (см. 9.4.4). Данный массив разбивается на блоки, которые также нумеруются с 0. Номер блока в файле называется ЛОГИЧЕСКИМ НОМЕРОМ блока.



9.6. Директория

ДИРЕКТОРИЯ - файл специального вида, доступный для чтения и записи как обычный файл, используемый ОС для организации файлового дерева. Директория представдяет собой одномерный массив УЗЛОВ - 64-байтовых записей (см. 9.6.1). Узлы в директории нумеруются от 0 до N-1, где N - текущее число узлов в директории. Размер директории должен быть не меньше 1 узла. При этом поле EOF инода, соответствующего данной директории, должно быть равно 64*N.

КОРНЕВОЙ директорией диска называется директория с номером инода 0. Узлы с номером 0 всех директорий диска, отличных от корневой, должны иметь имя "..", признаки {dir, hidden} (см. 9.6.1.3) и ссылаться (см. 9.6.1.2) на директорию, к которой данная директория привязана (см. 9.6.1.2).

9.6.1. Узел директории

Узел директории содержит следующую информацию:
- имя узла;
- номер файла, на который узел ссылается;
- специальные признаки.

Узел либо ссылается на файл, либо является незанятым. Незанятым считается узел, который либо имеет пустое имя, либо специальный признак "отвязанный файл".

9.6.1.1. Имя узла

Имя узла - массив из 32 байт, в котором содержится имя в кодировке КОИ-8. Имя может быть пустым только для свободного узла. Терминатором имени считается либо конец массива либо символ 0с. Имя не может содержать символы " ", "/", "[", "]".

Не допускается наличие нескольких узлов с одним именем.

9.6.1.2. Номер файла

Номер файла - целое число. Если данное поле содержит номер файла и узел не является свободным, будем говорить, что узел (и директория вместе с ним) ССЫЛАЕТСЯ на этот файл, а файл, в свою очередь, ПРИВЯЗАН к этой директории под именем данного узла. Значение поля "номер файла", выходящее за диапазон [0..MAX_FILE-1], где MAX_FILE - число файлов на носителе, считается некорректным.

В одной директории может содержаться несколько узлов с разными именами, ссылающихся на один файл.

9.6.1.3. Специальные признаки файла

Специальные признаки файла - это массив битов. В

настоящей реализации используются следующие признаки:

- отвязанный файл ("del", от англ. deleted)
- файл ("file")
- директория ("dir")
- скрытый файл ("hidden")
- псевдофайл ("escape")

Признаки "del" и "hidden" не зависят от состояния остальных признаков. Признак "escape" отменяет действие признаков "file" и "dir". Признак "dir" отменяет действие признака "file". Системой поддерживается отсутствие признаков, действие которых отменено другим признаком.

а) признак "отвязанный файл" выставляется, если данный узел ранее ссылался на файл, но эта ссылка была удалена. В этом случае остальные признаки сохраняются неизменными до следующего использования узла.

б) признак "файл" выставляется, если узел в настоящий момент ссылается на обычный файл (не являющийся директорией).

в) признак "директория" выставляется, если данный узел ссылается на директорию. В этом случае синонимичный признак должен быть выставлен в иноде, на который ссылается данный узел.

г) признак "скрытый файл"" выставляется для использования в системных утилитах визуализации директорий. Он обозначает, что при запуске без специальных указаний данный узел визуализировать не нужно.

д) признак "псевдофайл" дублирует в узле синонимичный признак инода, на который ссылается данный узел (см 9.4.7,б).

Часть 10. ДРАЙВЕРЫ УСТРОЙСТВ

Предисловие соавтора

Содержимое этой части - результат бесед с Д.Кузнецовым (Leo).

Соавтор

ОС Excelsior - развиваемая и легко настраиваемая система. Универсальность программного обеспечения достигается независимостью от типов устройств, входящих в конфигурацию системы. Настройка на конкретную конфигурацию производится с помощью драйверов. Это терминальные драйверы и драйверы магнитных носителей, драйверы печатающих устройств и т.д.. При необходимости новые драйверы могут быть дописаны.

10.1. Поддержка драйверов в системе

Поддержка драйверов в ОС Excelsior осуществляется на уровне ядра системы.

Кроме того, существует библиотека defReqest, которая тоже используется при написании драйверов.

10.2. Типы драйверов

В ОС известны драйверы трех типов:

1) дисковые;
2) сериальные;
3) специальные.

Дисковые драйверы обеспечивают работу с дисками: выполняют массовые операции над секторами фиксированного размера (до 4Гб).

Сериальные драйверы предназначаются для последовательных и параллельных интерфейсов. К ним можно отнести терминальные драйверы, драйверы печатающих устройств, мыши и т.п..

Все прочие устройства обслуживаются драйверами, относящимися к разделу специальных.

10.3. Структура драйвера

Очередь запросов

10.3.1. Представление системе

Для того, чтобы с устройством можно было работать,

драйвер должен представиться системе.

Прежде всего, в нем должно содержаться самоназвание - слово из семи произвольных символов, за исключением пробелов, "/", "\", ":".

Кроме того, он должен содержать две процедуры: def_drv, представляющую драйвер, и rem_drv, удаляющую драйвер.

Замечание. Процедура rem_drv в принципе не обязательна, поскольку это действие обеспечивается и самой системой.

Процедура def_drv имеет следующие параметры: имя драйвера, номер подустройства (направления), имя владельца (по сути, драйвера, с которым необходимо синхронизоваться, например, самого себя или пусто), тип устройства, процедуру обработки запроса do_io.

10.3.2. Запрос

Стандарт запроса определен в библиотечном модуле defRequest. Для непосредственного ознакомления эта библиотека с подробным комментарием приводится в приложении к этому разделу. В терминах Модулы-2 запрос - это запись со следующими полями:

```
REQUEST = RECORD
            op : INTEGER;
            drn: INTEGER;
            ofs: INTEGER;
            buf: sys.ADDRESS;
            pos: INTEGER;
            len: INTEGER;
            res: INTEGER;
          END;
```

Поле op определяет характер операции над устройством (чтение, запись и т.п.).

Поле drn содержит номер подустройства (направления).

Поле res обрабатывается средствами библиотеки defErrors для получения информации об ошибках при выполнении операции.

Поля ofs, buf, pos, len ориентированы на дисковые и сериальные драйверы и имеют следующий смысл:

поле	для дисковых драйверов	для сериальных драйверов
len	число секторов	число байт
pos	игнорируется	номер байта

```
 buf    | с какого адреса          | адрес, с которого читать
        | начинать операцию        | или по которому писать
        |                          |
 ofs    | с какого сектора         | игнорируется
        | начинать операцию        |
        |                          |
-------------------------------------------------------------
```

10.3.3. Процедура обработки запроса

Каждый драйвер реализует процедуру обработки запроса do_io. Этой процедуре в качестве параметра передается запрос:

```
PROCEDURE  do_io(VAR REQUEST);
```

Процедура может выглядеть приблизительно так:

```
PROCEDURE doio(VAR r: REQUEST);
BEGIN
  CASE r.op OF
    |READ : r.res:=my_read(r.drn,r.ofs,r.len...)
    |WRITE:

  END;
END doio;
```

10.4. Приложение: библиотека defRequest

```
DEFINITION MODULE defRequest; (* Ned 19-Sep-89. (c) KRONOS *)

(* Определяет стандарт запроса к драйверу. *)

IMPORT  sys: SYSTEM;

CONST (* dmode *)
  ready   = {0};
  floppy  = {1};
  wint    = {2};
  fmtsec  = {3};
  fmttrk  = {4};
  fmtunit = {5};
  wpro    = {6};

(*sync    = { 8};*)

CONST (* smode *)
  raw        = { 0};
  parNO      = { 1};
  parODD     = { 2};
  parEVEN    = { 3};
  stops0     = { 4};
```

```
  stops1     = { 5};
  stops1_5   = { 6};
  stops2     = { 7};
  sync       = { 8};

TYPE
  TRANSLATE = POINTER TO ARRAY CHAR OF CHAR;

--------------------- Стандарт запроса ----------------------
                      ------------------

  REQUEST = RECORD (* 28 bytes *)
              op : INTEGER;     -- код операции
              drn: INTEGER;     -- номер подустройства
              res: INTEGER;     -- результат
              CASE :INTEGER OF
              |0: ofs    : INTEGER;     -- адрес на устройстве
                  buf    : sys.ADDRESS; -- адрес буфера в памяти
                  pos    : INTEGER;     -- смещение в буфере
                  len    : INTEGER;     -- длина

              |1: (* только для операций GET_SPEC, SET_SPEC*)
                  dmode  : BITSET;
                  dsecs  : INTEGER; (* размер устройства в *)
                                    (* секторах           *)
                  ssc    : INTEGER; (* код размера сектора *)
                  secsize: INTEGER; (* длина сектора 2**ssc*)
                  cyls   : INTEGER; (* число треков        *)
                  heads  : INTEGER; (* число головок       *)
                  minsec : INTEGER; (* min номер сектора   *)
                  maxsec : INTEGER; (* max номер сектора   *)
                  ressec : INTEGER; (* число резервных сек.*)
                  precomp: INTEGER; (* прекомпенсация      *)
                  rate   : INTEGER; (* шаг головки         *)

              |2: (* только для операций GET_SPEC, SET_SPEC*)
                  smode  : BITSET;
                  baud   : INTEGER; (* baud rate code      *)
                  xon    : CHAR;    (* XON/XOFF            *)
                  xoff   : CHAR;    (* протокол            *)
                  limxon : INTEGER; (* send xon  limit     *)
                  limxoff: INTEGER; (* send xoff limit     *)
                  trtin  : TRANSLATE;
                  trtout : TRANSLATE;
              END
            END;

------------ Перечень операций над устройствами --------------
             ---------------------------------

CONST  (* L.S.B. (Low Significant Byte *)
  NOP        =  0;
```

```
  LOCK       =  1; (* LOCK   пока не используется *)
  UNLOCK     =  2; (* UNLOCK пока не используется *)

  READ       =  3; (* чтение данных с устройства           *)
  WRITE      =  4; (* запись данных на устройство          *)

  WAIT       =  5; (* ожидание данных                      *)
  READY      =  6; (* запрос о числе байтов в буфере ввода *)
  CONTROL    =  7; (* управляющая операция для клавиатур,  *)
                   (* экранов и т.д. Старшие байты содержат *)
                   (* код операции                         *)

  GET_SPEC   =  8; (* запрос спецификации устройства       *)
  SET_SPEC   =  9; (* определение спецификации устройства  *)
  POWER_OFF  = 10; (* завершение работы                    *)
  FORMAT     = 11; (* форматирование устройства            *)
  SEEK       = 12; (* позиционирование                     *)
  MOUNT      = 13; (* монтирование носителя                *)
  UNMOUNT    = 14; (* размонтирование носителя             *)


(*************************************************************

------------------------  ПРИМЕЧАНИЕ  ------------------------
                          ------------
```

Следующее описание определяет параметры каждой операции и поля, через которые эти параметры передаются. Все операции возвращают результат операции в поле res. Для всех операций поле drn содержит номер (под)устройства.

```
READ, WRITE (disk):
-------------------
        IN
            drn - номер (под)устройства
            buf - буфер ввода-вывода
            ofs - номер сектора на диске
            len - число секторов
        OUT
            res - результат операции

READ, WRITE (serial):
-----------------------------
        IN
            drn - номер (под)устройства
            buf - буфер ввода-вывода
            pos - смещение в буфере
            len - число байтов
        OUT
            len - число непрочитанных байтов
            res - результат операции

WAIT (serial):
```

```
------------
        IN
            drn - номер (под)устройства
        OUT
            res - результат операции


READY (input serial):
--------------------
        IN
            drn - номер (под)устройства
        OUT
            len - число байтов в буфере ввода
            res - результат операции

CONTROL:
-------
        IN
            op  - старшие байты содержат код
                  управляющей операциии
            drn - номер (под)устройства
        OUT
            res - результат операции

FORMAT:
------
     Драйвер  должен указать системе посредством операции
GET_SPEC, какой тип форматирования он поддерживает:
     fmtsec (посекторное);
     fmttrk (потрековое);
     fmtunit (всего устройства).


        OUT
            res - результат операции
            len - размер буфера (для fmttrk, см. далее)

     Система вызывает драйвер с "ofs"=-1 "buf"=NIL перед
форматированием.
     Если драйверу требуется буфер, он должен возвратить
"not_enough" в качестве результата и указать в "len"
необходимый размер, после чего система отведет память под
буфер и повторит операцию уже с "ofs"=-1,
"buf"=выделенный_буфер, "len"=размер_буфера.
     Если драйвер возвращает "ok" при ofs=-1, то буфер не
отводится.

     Для драйверов с форрматированием "fmtunit":
        IN  drn - номер (под)устройства
            ofs = 0

     Для драйверов с форрматированием "fmttrk":
        IN  drn - номер (под)устройства
```

```
            buf - буфер ввода-вывода
            ofs - смещение на устройстве (в секторах)
            len - размер буфера (в байтах).
```

Драйверы с посекторным форматированием пока не поддерживаются в системе.

```
**********************************************************)
END defRequest.
```

Часть 11. БАЗОВАЯ ПОДДЕРЖКА ГРАФИКИ

Предисловие соавтора

Содержимое этой части целиком лежит на совести А.Никитина (Nick), который является также автором большинства графических библиотек.

Глава 11.1. Общие сведения

11.1.1. Типы графических дисплеев

Существует много разнообразных устройств для вывода изображений, построенных с помощью машинной графики. В качестве типичных примеров можно назвать дисплеи на запоминающих трубках, векторные и растровые с регенерацией изображения на электронно-лучевых трубках. Однако в последнее время, в связи с удешевлением микросхем памяти и увеличением их емкости, широкое распространение получили растровые дисплеи.

Как дисплеи на запоминаюших ЭЛТ, так и векторные дисплеи с произвольным сканированием являются устройствами рисования непрерывных отрезков, рисуемых из любой адресуемой точки в любую другую. Растровое устройство работает совершенно иначе. Его можно рассматривать как матрицу дискретных ячеек (точек), каждая из которых может быть подсвечена. Таким образом, растровый дисплей является "точечно-рисующим" устройством. Невозможно, за исключением специальных случаев, непосредственно нарисовать отрезок из одной адресуемой точки (пиксела) в матрице в другую адресуемую точку. Отрезок можно лишь аппроксимировать последовательностью точек, близко лежащих к реальной траектории отрезка.

11.1.2. Буфер кадра

Чаще всего в растровых графических устройствах используются различного рода буферы кадра, представляющие собой большой непрерывный участок оперативной памяти компьютера. Каждый пиксел растра кодируется как минимум одним битом памяти. Для квадратного растра размером 512x512x1 бит требуется 262144 бита памяти. Изображение в буфере кадра строится поточечно. Если точка растра кодируется одним битом памяти, имеющим только два состояния (двоичное 0 и 1), то можно получить только "черно-белое" изображение. Буфер кадра является цифровым устройством, тогда как электронно-лучевая трубка - аналоговое устройство, для работы которого требуется электрическое напряжение различных уровней. Поэтому при считывании информации из буфера кадра и ее выводе на графическое устройство с растровой ЭЛТ должно происходить

преобразование из цифрового представления в аналоговый сигнал. Такое преобразование выполняет цифро-аналоговый преобразователь (ЦАП). Каждый пиксел буфера кадра, прежде чем он будет отображен на растровой ЭЛТ, должен быть считан и преобразован.

Интенсивность каждого пиксела на ЭЛТ управляется содержимым соответсвующих пикселов в буфере кадра. Двоичное число, полученное из буфера кадра, интерпретируется как уровень интенсивности между 0 и 2^N-1. С помощью ЦАП это число преобразуется в напряжение между 0 (темный экран) и 2^N-1 (максимальная интенсивность свечения). Очевидно, что всего можно получить 2^N уровней интенсивности, и расширение диапазона одновременно доступных цветов или полутонов серого может быть лишь получено путем увеличения разрядности слова, кодирующего состояние пиксела.

11.1.3 Таблица цветов (палитра)

Число доступных уровней интенсивности можно увеличить, воспользовавшись таблицей цветов, или, как ее еще называют, палитрой. После считывания из буфера кадра получившееся число используется как индекс в таблице цветов. В этой таблице должно содержаться 2^N элементов. Каждый ее элемент может содержать W бит, причем W может быть больше N. В последнем случае можно получить 2^W значений интенсивности, но одновременно могуть быть доступны лишь 2^N из них. Для получения на экране других значений интенсивности палитру следует изменить (перезагрузить).

Поскольку существует три основных цвета, линейная комбинация которых позволяет получить все остальные цвета, то можно реализовать простой "цветной" буфер кадра, разбив слово таблицы цветов на три равные части по M бит. Каждая из этих трех частей будет задавать интенсивность одного из основных цветов. Таким образом можно получить 2^N цветов из (2^M)^3, возможных, т.е. при M=4 (2^4)^3 = 4096 цветов, а при M=8 уже 16777216, что намного превосходит возможности глаза человека, различающего не более нескольких тысяч оттенков.

Глава 11.2. Битовые карты

Если разрядность слова, кодирующего состояние пиксела в буфере кадра растрового графического дисплея, равна 1, и эти пикселы расположены друг за другом в слове памяти процессора, то такого рода буфер кадра называют битовой плоскостью, или битовой картой (bitmap). Имея одну битовую плоскость, можно, очевидно, получить только "черно-белое" изображение. Для увеличения градаций "серой" шкалы или количества цветов нужно использовать несколько битовых плоскостей. При этом слово, кодирующее состояние пиксела на экране ЭЛТ, пакуется из соответствующих пикселов каждой из N используемых битовых плоскостей и интерпретируеся как уровень интенсивности между 0 и 2^N-1.

Таким образом, битовая карта представляет собой непрерывный участок памяти компьютера, интерпретируемой как один или несколько двумерных массивов битов определенного размера. Обычно размер памяти, занимаемой одной битовой строкой, выровнен по границе входного слова процессора. Количество необходимых при этом слов можно вычислить по простой формуле:

WPL (Words Per Line) = (W + N-1) DIV N ,

где W - ширина битовой плоскости, N - ширина слова процессора в битах.

Глава 11.3. Библиотеки определения

Для поддержки стандартной работы с графическими дисплеями и решения задач машинной графики, и тем самым облегчения переноса больших графических програмных систем, в ОС Excelsior имеется набор библиотек нижнего уровня, состоящий из модулей: defScreen, Screen - подержка работы с графическими устройствами отображения, defBMG, BMG - представление битовых карт, рисование на них графических примитивов и текста, defFont, Fonts - описание и работа с различного рода шрифтами. Этот набор по мере развития операционной системы и прикладного програмного обеспечения будет расширяться.

11.3.1. Модуль defBMG

В модуле defBMG (текст прилагается, см.11.5.1) вводятся два базовых типа, определяющих стадартное для всех библиотек нижнего уровня описание битовых карт.

В записи BMD - дескриптора битовой карты - обзначены поля W и H, определяющие ее горизонтальный и вертикальный размеры соответственно, WPL - количество слов памяти для одной битовой строки, массив layers содержит в себе базовые адреса битовых слоев, PATTERN - шаблон для рисования линий. Поля W, H, WPL, BASE и PATTERN используются графическими командами процессора. Поле mask - шаблон имеющихся слоев в данной битовой карте (что очень удобно). Тип BITMAP удобен для низкоуровневых прикладных библиотек графики.

11.3.2. Модуль defScreen

В данном модуле (текст прилагается, см.11.5.2) вводятся базовые типы для описания характеристик графических дисплеев, их особенностей, а также задаются коды стандартных команд для программ-драйверов этих устройств.

11.3.2.1. Представление экранов, тип операции, маска записи

Определим буфер кадра как двумерный массив размером ScrWidth-x-ScrHeight n-разрядных слов, кодирующих цвет точек экрана.

```
Screen = ARRAY [0..ScrWidth-1][0..ScrHeight-1]
           OF ARRAY [0..n-1] OF BIT
```

Color(X,Y) - некоторая функция, выдающая по координатам точки ее цвет

Mode - двуместная побитовая логическая функция между содержимым экрана и заданным цветом операции Color

Mask - маска записи.

Тогда операция постановки точки с координатами (X,Y) цвета Color и маской записи Mask определяется так:

```
Screen[X,Y]:=(Screen[X,Y]-mask) +
             (Screen[X,Y] Mode Color)*Mask .
```

Эта формула, записанная в нестрогом виде, несколько проясняет смысл введенных выше понятий.

11.3.2.2. Область отсечения (Clip Rectangle)

Для удобства ограничения доступной к изменению содержимого экрана области, а также для описания относительных координат, введем понятие "области отсечения", предварительно оговорив несколько моментов: область отсечения никогда не должна выходить за пределы экрана и ее линейные размеры должны быть сторого больше нуля.

Теперь все координаты будем отсчитывать относительно точки (0,0) области отсечения, а постановку на экран точки с координатами, выходящими за пределы области отсечения, игнорировать.

11.3.2.3. Описание прямоугольного блока

Для вышеописанной области отсечения, а также для для описания произвольных прямоугольных областей или блоков будет удобно следующее описание:

```
TYPE  BLOCK = RECORD x,y,w,h: INTEGER END;
```

где x,y - соответсвенно горизонтальная и вертикальная координаты нижнего левого угла прямоугольника, а w,h - его ширина и высота.

11.3.2.4 Инструмент (Tool)

Для работы с содержимым экрана необходимо уметь задать цвет, которым мы хотим рисовать, тип операции над содержимым экрана и заданным цветом, и очень удобно иметь маску записи (см 11.1.1). Помимо этого необходимо уметь просто и удобно задать координаты и размер доступной для изменений области экрана, т.е. определить "область отсечения" (см. 11.3.2.2).

Так как весь этот "инструментарий" необходимо постоянно иметь под рукой, логично было бы собрать его в один "ящик":

```
TYPE  TOOL = RECORD
               mode : INTEGER;
               mask : BITSET;
               color: BITSET;
               back : BITSET;
               clip : BLOCK;
               zX,zY: INTEGER;
             END;
```

Описание полей:

- mask - маска записи (см. 11.3.2.1);
- color - "рабочий" цвет для рисования графических примитивов (линия, круг и т.д.), а также цвет рисования символов;
- back - цвет фона, на котором рисуются символы (очень удобно иметь при выводе на экран различного рода текстов);
- zX,zY - центр относительных координат;
- clip - описание прямоугольника, определяющего область отсечения (задается относительно zX и zY);
- mode - тип операции (см 11.1.1), в данной версии может принимать следующие значения:

```
CONST
   rep = 0; (* destinator := sourse                      *)
   or  = 1; (* destinator := destinator OR  sourse     *)
   xor = 2; (* destinator := destinator XOR sourse     *)
   bic = 3; (* destinator := destinator AND NOT sourse *)

       normal  = 0; (* зарезервировано *)
       reverse = 4; (* зарезервировано *)
```

11.3.2.5. Палитра (Palette)

Для удобного описания палитры (см. 11.1.3) введены два типа данных. COLOR - описывает соотношение интенсивностей основных цветов (r - красный (rED), g - зеленый (gREEN), b - синий (bLUE)). PALETTE задает соответствие между кодом цвета, т.е. индексом в массиве, и соотношением основных цветов. Иными словами, определяет палитру цветов, одновременно доступных к изображению.

11.3.2.6 Описание Экрана

Поскольку физические экраны имеют различные характеристики, например: тип устройства, количество битов, кодирующих цвет пикселов, конкретную реализацию буфера кадра (bitmap, pelmap и т.д.), размеры области буфера кадра, отображемой на физическом экране дисплея, и ее текущее

положение в кординатах буфера кадра, шаг его изменения по горизонтали и вертикали, текущую палитру, соотношение расстояний между элементами изображения по вертикали и горизонтали на физическом экране, все эти и некоторые другие параметры объединены в одном типе:

```
TYPE
  STATE   = POINTER TO STATUS;
  STATUS  = RECORD
              type : INTEGER;
              kind : INTEGER;
              W,H  : INTEGER;
              bpp  : INTEGER;
              ldcx : INTEGER;
              ldcy : INTEGER;
              dx   : INTEGER;
              dy   : INTEGER;
              xpix : INTEGER;
              ypix : INTEGER;
              RGB  : INTEGER;
              pal  : PALETTE;
              ext  : INTEGER;
            END;
```

```
magic     - волшебное слово, которое не следует изменять;
type      - число, определяющее тип устройства, например:
            type=0 => устройство IGD480;
bpp       - число бит на точку;
ldcx,ldcy - текущее положение отображаемой области;
W и H     - ширина и высота области в элементах изображения;
dx,dy     - шаг изменения положения  отображаемой области бу-
            фера кадра;
xpix,ypix - соотношение расстояний между элементами изображе-
            ния по вертикали и горизонтали, соотвественно, на
            физическом экране;
pal       - текущая палитра;
RGB       - дипазон  значений  интенсивностей  красного,
            синего и зеленого цветов;
desc      - ссылка на  дескриптор представления буфера кадра,
            например, указатель на переменную типа BMD;
kind      - вид конкретной реализации буфера кадра, в  данной
            версии модуля может принимать следующие значения:
```

```
bitmap = 444D4224h; (* $BMD *)
pelmap = 44504124h; (* $APD *)
other  = 48544F24h; (* $OTH *)
```

11.3.2.7. Управляющие коды для драйверов

Для стандартного общения с драйверами различных графических дисплеев в модуле описаны соответсвующие константы - управляющие коды:

```
 CONTROL  = 01h; --
_init     = 02h; -- команда инициализации драйвера,
_set_ldcx = 03h; -- установка горизонтальной координаты поло-
                    жения отображаемой области буфера кадра,
_set_ldcy = 04h; -- установка вертикальной координаты положе-
                    ния отображаемой области буфера кадра,
_set_rgb  = 05h; -- запись нового содержимого палитры,
_get_rgb  = 06h; -- чтение текущего содержимого палитры.
```

Замечание. Все графические библиотеки для растровых дисплеев с различного рода реализациями буфера кадра должны придерживаться изложенных здесь правил изменения содержимого буфера кадра в соответствии со значениями маски записи, устанавливаемого цвета и типа операции.

11.3.3. Модуль defFonts

В данном модуле (текст прилагается, см.11.5.3) вводятся базовые типы для описания характеристик шрифтов и их особенностей.

Здесь шрифтом мы называем набор графических представлений символов, расположенных в порядке, определенном стандартом КОИ-8, для отображения текстов на графическом экране. Для представленных битовыми матрицами шрифтов размеры знако-места не должны превышать 128x128 точек.

11.3.3.1. Дескриптор шрифта

Объявленный в модуле тип FNTD (FoNT Descriptor) есть запись, в которой хранятся все основные характеристики шрифта.

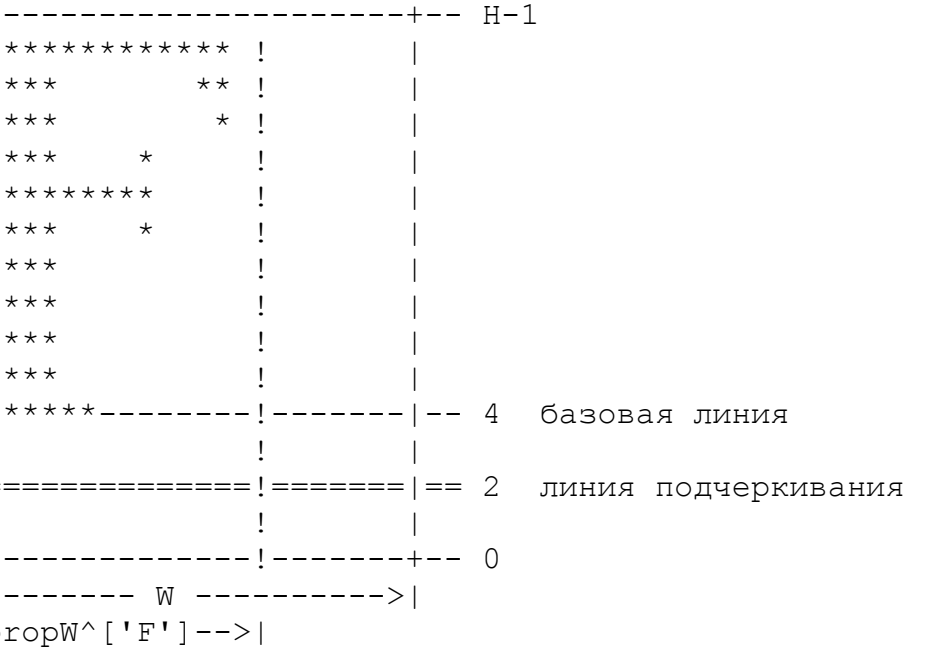
```
TYPE
  FONT  = POINTER TO FNTD;
  BTAB  = ARRAY CHAR OF CHAR;
  ITAB  = ARRAY CHAR OF INTEGER;
  BTPTR = POINTER TO BTAB;
  ITPTR = POINTER TO ITAB;

  FNTD = RECORD
           W,H  : INTEGER;
           BASE : SYSTEM.ADDRESS;
           rfe  : SYSTEM.WORD;
           magic: INTEGER;
           state: BITSET;
           size : INTEGER;
           bline: INTEGER;
           uline: INTEGER;
```

```
          space: INTEGER;
          fchar: CHAR;
          lchar: CHAR;
          propW: BTPTR;  (* #NIL only of state*prop#{} *)
          propX: BTPTR;  (* #NIL only of state*prop#{} *)
          cellW: BTPTR;
          cellH: BTPTR;
          cellY: BTPTR;
          cellX: BTPTR;  (* allways zero now *)
          bases: ITPTR;
        END;

CONST
   prop   = {0};
   italic = {1};
   packed = {2};
```

Поля W и H определяют максимальный размер знако-места для данного шрифта, bline и uline - положение базовой линии и линии подчеркивания относительно нижнего края знако-места.

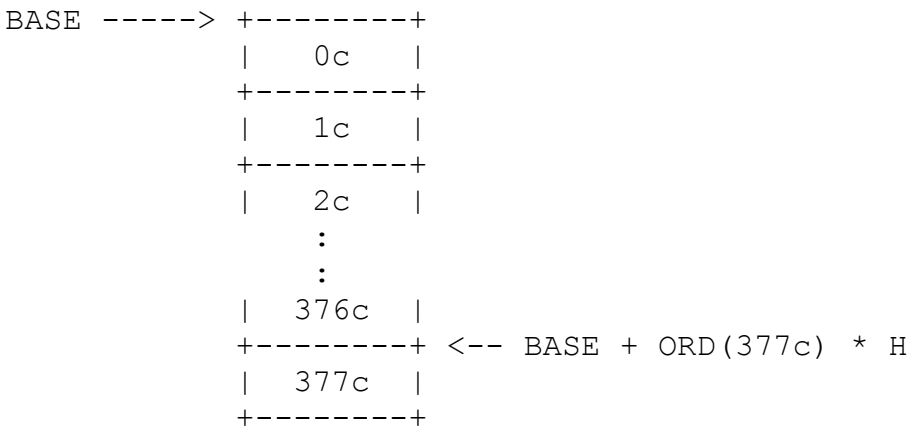


В массиве, на который указывает значение поля propW, хранятся смещения от начала знако-места заданного символа ch до позиции печати следующего в строке. Поля rfe1, rfe2 и rfe3 зарезервированы для расширений, возможных при дальнейшем развитии графической поддержки в системе. Объяснение семантики остальных полей дескриптора можно встретить далее по тексту.

11.3.3.2. DCH-шрифты

В системе команд процессора КРОНОС 2.6 есть команда отображения символа с шириной и высотой, не превышающими 32 на битовые карты. Подробное описание команды можно найти в

документации по системе команд процессора. Специально для этой графической команды в дескриптор шрифта введено поле BASE - указатель на начало непрерывной области памяти, где хранятся образы символов. При этом они имеют одинаковый размер и расположены ровно друг за другом.



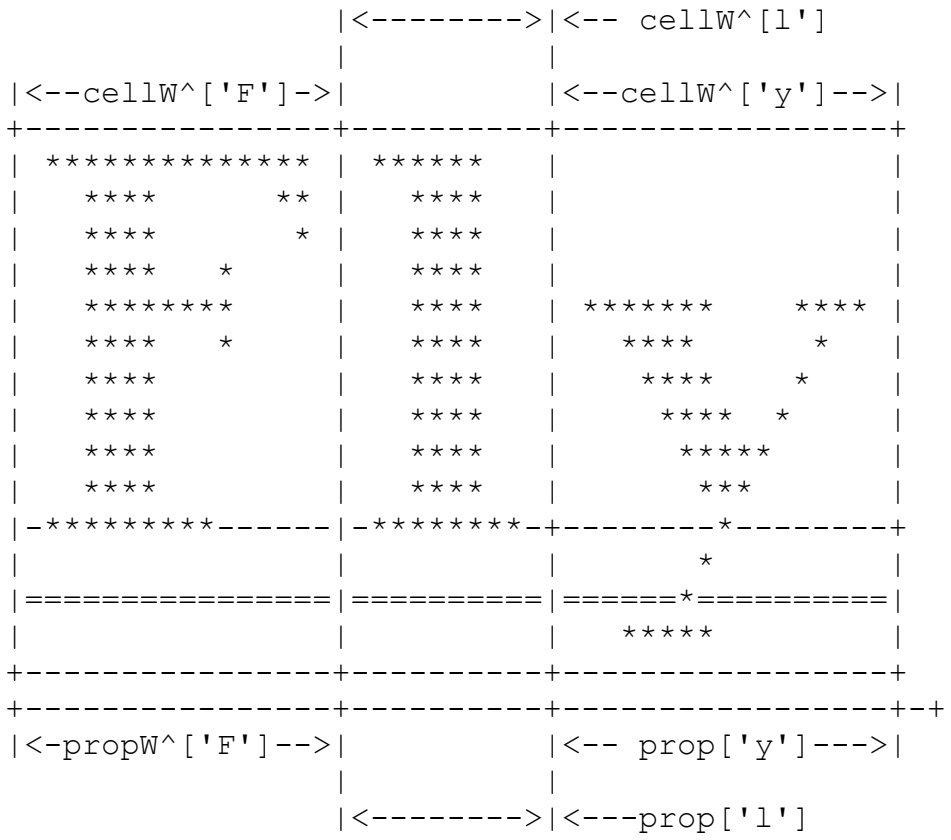
Однако расстояния между символами при печати могут отличаться от ширины знако-места, но превосходить их. На это указывает отсутсвие в слове состояния шрифта state признака constW.

О том что вы имеете дело с DCH-шрифтом, говорит наличие признака dchar в слове состояния, и для этих шрифтов значение поля size равно высоте шрифта, умноженной на количество хранимых символов шрифта.

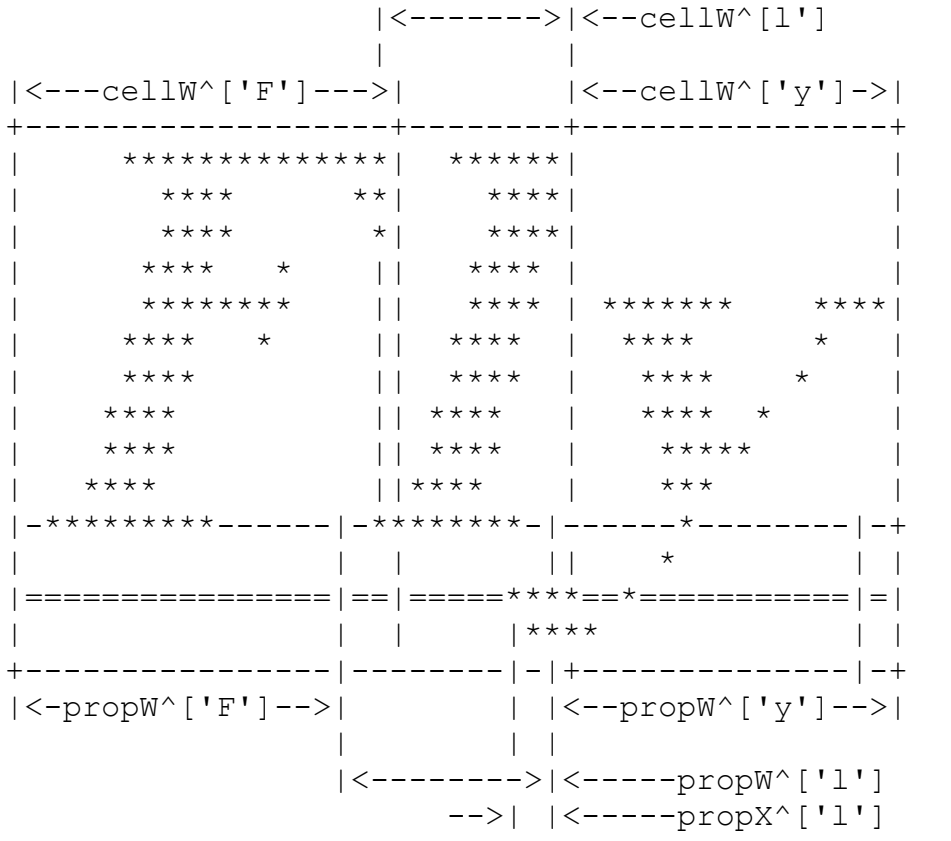
11.3.3.3. PACKED-шрифты

Для удобства хранения шрифтов большого размера предусмотрена возможность компактизации их содержимого, например, за счет того, что символы могут иметь одинаковое начертание, или быть неотображаемыми, т.е. не иметь своего битового образа. Для выделения подобного рода шрифтов в слове состояния выставляется признак packed, а также соответсвующим образом расписываются следующие поля поля дескриптора шрифта: указатели на массивы ширины и высоты хранимых образов - cellW и cellH, а также смещения этих образов по вертикали относительно нижнего края полного знако-места - charY, массив указателей на области памяти, занимаемых образами символов - bases. И если указатель в этом массиве равен NIL, то символ не имеет своего образа, например, по причине того, что не является отображаемым.

В случае пропорционального ненаклоненного шрифта смещение от начала знако-места заданного символа ch до позиции печати следующего в слове, хранящееся в массиве propW^[ch], равно ширине его знако-места ORD(cellW^[ch]).

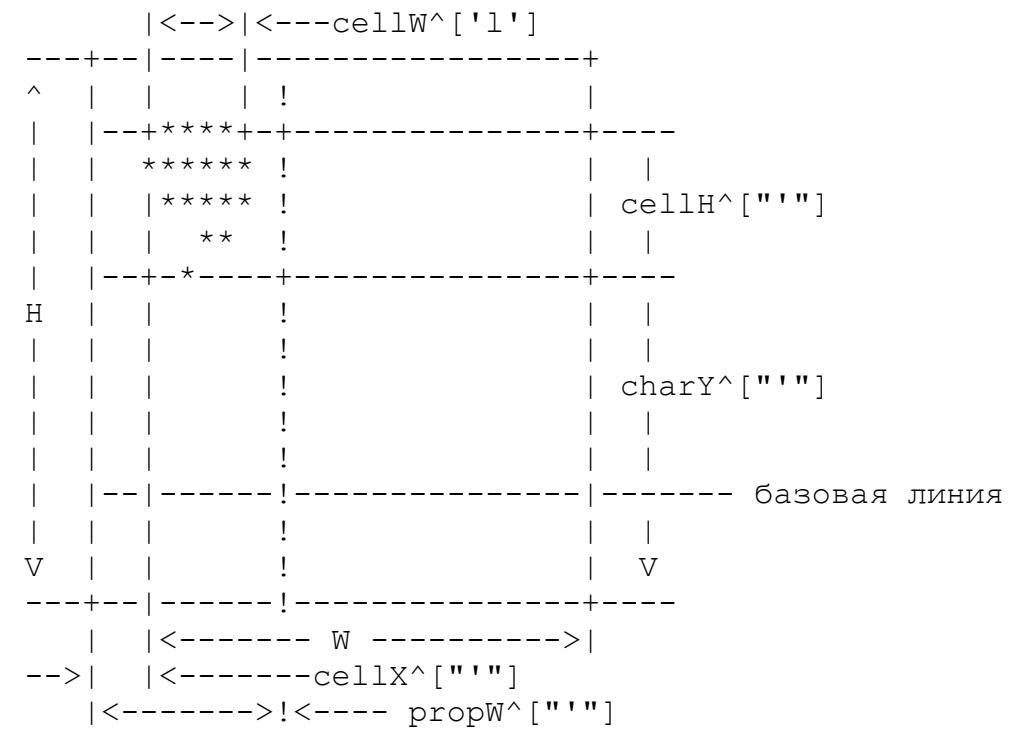


Из рисунка становится ясно, что это утверждение не верно для наклонных шрифтов, выделяемых признаком ital.



В случае более плотной упаковки графических образов символов хранится только минимальный прямоугольник,

охватывающий область значащих пикселов (на рисунке это прямоугольник размером ORD(cellW^["'"]) на ORD(cellH^["'"]) точек). При этом необходимое для корректного отображения символа смещение нижнего левого угла этого прямоугольника относительно базовой линии хранится в массиве, на который указывает поле charY в дескрипторе шрифта, и в слове состояния отсутствует признак constH.



11.3.3.4. Дополнения

Остальные типы и константы, объявленные в модуле defFont, введены из соображений удобства написания прикладных программ, использующих работу со шрифтами.

Глава 11.4. Прикладные библиотеки

11.4.1 Модуль BMG (BitMap Graphics)

Модуль BMG (текст прилагается, см.11.5.4) предназначен для поддержки графического вывода на битовые карты и содержит набор процедур нижнего уровня, представляющих собой достаточный базис для написания графических систем более высокого уровня.

Данная библиотека является "чистой" и пригодна для использования в многопроцессных системах.

11.4.1.1 Процедуры работы с блоками

Описанные ниже процедуры призваны оградить пользователя от лишних размышлений при копировании и других действиях с прямоугольными битовыми блоками.

```
PROCEDURE cross(VAR des: BLOCK; blk0,blk1: BLOCK);
```

Процедура cross вычисляет пересечение двух блоков blk0, blk1 и параметры результирующего прямоугольника пересечения заносит в дескриптор блока des. Если же пересечение пусто, то ширина и высота в des будут равны 0.

```
PROCEDURE bblt   (des: BITMAP; dtool: TOOL;   x,y: INTEGER;
                  sou: BITMAP; stool: TOOL; block: BLOCK);
```

bblt (bIT bLOCK tRANSFER) - процедура пересылки битового блока-источника block из битовой карты sou в битовую карту des. При этом левый нижний угол блока попадает в точку с координатами x,y; блок в источнике sou клипируется в соответствии с stool, а в приемнике des в соответствии с dtool. Как и все графические операции, bblt выполняется с определенной модой операции, которая задается значением параметра dtool.mode.

11.4.1.2. Процедуры стирания и заполнения

```
PROCEDURE erase(bmd: BITMAP);
```

Процедура erase предназначена для полной очистки слоев в битовой карте.

```
PROCEDURE fill(bmd: BITMAP; tool: TOOL; block: BLOCK;
                w: INTEGER; SEQ pattern: SYSTEM.WORD);

PROCEDURE pattern(bmd: BITMAP; tool: TOOL; block: BLOCK;
```

```
                    w: INTEGER; pattern: ARRAY OF SYSTEM.WORD);
```

Эти процедуры заполняют с помощью заданного инструмента прямоугольную область block битовым шаблоном шириной w и высотой, определяемой по правилу:

(Размер pattern в словах) DIV (ширина w, в словах)

((HIGH(pattern)+1) DIV ((w+31) DIV 32))

11.4.1.3. Процедуры рисования графических примитивов

Во всех нижеописанных процедурах bmd - указатель на битовую карту, где будет изображаться заданный примитив, tool - инструмент рисования.

```
PROCEDURE dot(...  x,y: INTEGER);
```
Постановка точки.

```
PROCEDURE line(...  x0,y0,x1,y1: INTEGER);
```
Рисование отрезка прямой по координатам его концов.

```
PROCEDURE dline(... X0,Y0,X1,Y1: INTEGER; VAR r: SYSTEM.WORD);
```
Рисование отрезка штрих-пунктирной прямой по координатам его концов. Шаблон задается переменной r. При этом при последовательном рисовании двух линий, если начало второй лежит в конце первой, рисунок шаблона не нарушится, что обеспечивается возвращением соответствующего значения этой переменной.

```
PROCEDURE rect (...  x0,y0,x1,y1: INTEGER);
PROCEDURE frame(...  x0,y0,x1,y1: INTEGER);
```
Рисование прямоугольника и рамки по координатам их диагональных вершин.

```
PROCEDURE arc(...  xc,yc,xa,ya,xb,yb,r: INTEGER);
```
Рисование дуги окружности с центром в точке (xc,yc) радиуса r, от луча [(xc,yc),(xa,ya)) к лучу [(xc,yc),(xb,yb)).

```
PROCEDURE arc3(...  x0,y0,x1,y1,x2,y2: INTEGER);
```
Рисование дуги окружности от точки (x0,y0) через (x1,y1) до точки (x2,y2).

```
PROCEDURE  circle(...  xc,yc,rad: INTEGER);
PROCEDURE circlef(...  xc,yc,rad: INTEGER);
```
Рисование окружности и круга: xc, yc - координаты центра, rad - радиус.

```
PROCEDURE ring(... xc,yc,r1,r2: INTEGER);
```
Рисование круглого кольца по координатам центра xc, yc, внутреннему и внешнему радиусам.

```
PROCEDURE ellipse0(...   xc,yc,rx,ry: INTEGER);
PROCEDURE ellipse0f(...   xc,yc,rx,ry: INTEGER);
```

Эллипс и закрашенный эллипс: xc, yc - координаты центра; rx, ry - величины полуосей, параллельных осям X и Y соответственно.

```
PROCEDURE polyline0(...   SEQ xy: INTEGER);
PROCEDURE polyline1(...       xy: ARRAY OF INTEGER);
```

Рисование ломаной линии, заданной набором точек. Первой в паре лежит координата по X, второй - координата по Y. Если в последовательности или массиве количество элементов нечетно, то последний элемент при рисовании не принимается во внимание.

```
PROCEDURE trif(... x0,y0,x1,y1,x2,y2: INTEGER);
```

Рисование закрашенного треугольника по координатам трех его вершин.

```
PROCEDURE grid(... block: BLOCK; xstep,ystep: INTEGER);
```

```
PROCEDURE scroll(... x,y: INTEGER);
```

Содержимое прямоугольной области tool.clip сдвигается по горизонтали на расстояние x, по вертикали - на y. Направление сдвига задается знаком x (положительный - влево) и y (положительный - вниз). Освободившееся при этом место заполняется в соответствии с инструментом tool.

```
PROCEDURE offset(bmd: BITMAP; x,y,layer: INTEGER;
            VAR adr: SYSTEM.ADDRESS; VAR bitoffset: INTEGER);
```

По координатам x и y и номеру слоя layer выдает адрес слова в памяти и битовое смещение в этом слове, соответствующее данной точке. Если указанный слой в bmd отсутствует, в adr возвращается NIL.

```
PROCEDURE  write(... str: ARRAY OF CHAR; pos,len: INTEGER);
PROCEDURE xwrite(... str: ARRAY OF CHAR;
                              pos,len: INTEGER): INTEGER;
```

Отображают из строки str с позиции pos len символов шрифтом fnt. Процедура xwrite возвращает горизонтальную координату точки, следующей непосредственно за последней точкой выведенной строки.

```
PROCEDURE  print(... f: ARRAY OF CHAR; SEQ arg: SYSTEM.WORD);
PROCEDURE xprint(... f: ARRAY OF CHAR;
                         SEQ arg: SYSTEM.WORD): INTEGER;
```

Печатают значения параметров arg в формате, заданном в строке f. Процедура xwrite возвращает горизонтальную координату точки, следующей непосредственно за последней точкой выведенной строки.

```
PROCEDURE writech(...  x,y: INTEGER; fnt: FONT; ch: CHAR);
```
Отображает символ ch из шрифта fnt в позиции (x,y).

```
PROCEDURE lenght(fnt: FONT; fmt: ARRAY OF CHAR;
                        SEQ arg: SYSTEM.WORD): INTEGER;
```

Подсчитывает ширину области в пикселах, которую займет строка, отображенная шрифтом fnt, при печати значений параметров arg в формате, заданном в строке fmt.

```
PROCEDURE width (fnt: FONT; fmt: ARRAY OF CHAR;
                        SEQ arg: SYSTEM.WORD): INTEGER;
```

Подсчитывает ширину области в пикселах, которую займет строка, отображенная шрифтом fnt, при печати значений параметров arg в формате, заданном в строке fmt.

```
PROCEDURE margin (fnt: FONT; fmt: ARRAY OF CHAR;
                         SEQ arg: SYSTEM.WORD): INTEGER;
```

Возвращает величину, на которую необходимо сместить печатаемую строку вправо, если требуется вписать эту строку в прямоугольник, ширина которого получена с помощью процедуры width, вызванной с теми же аргументами, что и margin.

11.4.3. Модуль Screen

Модуль Screen (текст прилагается, см.11.5.5) предназначен для поддержки и стандартизации управления экранами.

11.4.3.1. Переменные "done", "error"

Значение переменной "done" говорит о том, была ли ошибка при исполнении операций с экраном, а "error" является индикантом последней произошедшей ошибки (см. описание модуля defErrors). Приведем в качестве примера способ выявления ошибки и один из многих способов реакции на него:

```
operation(...);
IF NOT done THEN Terminal.perror(error,"%%s"); HALT END
```

11.4.3.2 Процедуры управления

Модуль при инициализации открывает устройство по его символическому имени взятому из параметра SCR окружения задачи; если это удалось, производится его захват, закрывается для других задач доступ к нему; дескриптор экрана помещается в

переменную state модуля. По окончании задачи устройство освобождается. Если открыть устройство не удалось, переменные done и error примут соответствующие значения.

```
PROCEDURE set_ldcx(x: INTEGER);
PROCEDURE set_ldcy(y: INTEGER);
```

Данные процедуры производят установку (если устройство располагает такой возможностью) координат левого нижнего угла видимой области экрана (Left Down Corner of visible area) по горизонтали и вертикали соответственно.

```
PROCEDURE set_palette(p: PALETTE; from,len: INTEGER);
```

Записывает len цветов в аппаратную палитру устройства, заданного дескриптором scr, начиная с цвета с номером from по образцу, заданному в переменной "p".

```
PROCEDURE op(cmd: INTEGER; SEQ args: SYSTEM.WORD);
```

Если когда-нибудь кто-нибудь будет иметь устройство с какими-нибудь возможностями, не отраженными в данном модуле, то именно эта процедура даст возможность использовать их полностью.

Глава 11.5. Тексты определяющих модулей библиотек

11.5.1. Текст библиотеки defBMG

```
DEFINITION MODULE defBMG; (* nick 01-Jun-90. (c) KRONOS *)

IMPORT  SYSTEM;

TYPE

  BITMAP = POINTER TO BMD;

  BMD    = RECORD
             W,H    : INTEGER;
             WPL    : INTEGER;
             BASE   : SYSTEM.ADDRESS;
             PATTERN: BITSET;
             mask   : BITSET;
             layers : ARRAY [0..7] OF SYSTEM.ADDRESS; -- base
addrs of layers
           END;
END defBMG.
```

11.5.2. Текст библиотеки defScreen

```
DEFINITION MODULE defScreen; (* nick 04-Oct-90. (c) KRONOS *)

IMPORT  SYSTEM;

TYPE
  COLOR   = RECORD
              r: INTEGER;       (* intensity of red   beam *)
              g: INTEGER;       (* intensity of green beam *)
              b: INTEGER;       (* intensity of blue  beam *)
            END;

  PALETTE = DYNARR OF COLOR;

CONST           (* screen kind *)
   bitmap = 444D4224h; (* $BMD *)
   pelmap = 44504124h; (* $APD *)
   other  = 48544F24h; (* $OTH *)

TYPE
  STATE   = POINTER TO STATUS;
  STATUS  = RECORD
              type : INTEGER;  (* type of screen device
*)
              kind : INTEGER;  (* kind of implementation
*)
              W,H  : INTEGER;  (* width = W pixells, height = H
pix  *)
              bpp  : INTEGER;  (* bIT pER pIXELL
*)
              ldcx : INTEGER;  (* curren left down corner X
position *)
              ldcy : INTEGER;  (* curren left down corner Y
position *)
              dx   : INTEGER;  (* step by horizontal shifter
*)
              dy   : INTEGER;  (* step by vertical   shifter
*)
              xpix : INTEGER;  (* distance between pixells at X-
axis *)
              ypix : INTEGER;  (* distance between pixells at Y-
axis *)
              RGB  : INTEGER;  (* range of beam's intensity
*)
              pal  : PALETTE;  (* current palette
*)
              ext  : INTEGER;  (* some extra descriptor
*)
            END;

  BLOCK   = RECORD x,y,w,h: INTEGER END;
```

```
  TOOL    = RECORD
              mode : INTEGER;     (* operation mode        *)
              mask : BITSET;      (* write mask            *)
              color: BITSET;      (* color/foreground      *)
              back : BITSET;      (* for text background   *)
              clip : BLOCK;       (* clipping rectangle    *)
              zX,zY: INTEGER;     (* (0,0) abs coordinates *)
            END;

CONST           (*  operation modes *)
   rep = 0;  or  = 1;  reverse = 4;
   xor = 2;  bic = 3;  normal = 0;

CONST           (* Driver Control Codes *)
   CONTROL  = 01h;
  _init     = 02h;
  _set_ldcx = 03h;
  _set_ldcy = 04h;
  _set_rgb  = 05h;
  _get_rgb  = 06h;
  _refresh  = 07h;
  _refreshw = 08h;

END defScreen.
```

11.5.3. Текст библиотеки defFont

```
DEFINITION MODULE defFont;  (* nick 07-May-90. (c) KRONOS *)
                            (* Leo  28-Jan-91. (c) KRONOS *)
IMPORT  SYSTEM;

TYPE
  FONT  = POINTER TO FNTD;
  BTAB  = ARRAY CHAR OF CHAR;
  ITAB  = ARRAY CHAR OF INTEGER;
  BTPTR = POINTER TO BTAB;
  ITPTR = POINTER TO ITAB;

  FNTD = RECORD
           W,H  : INTEGER;
           BASE : SYSTEM.ADDRESS;
           rfe  : SYSTEM.WORD;
           magic: INTEGER;
           state: BITSET;
           size : INTEGER;
           bline: INTEGER;
           uline: INTEGER;
           space: INTEGER;
           fchar: CHAR;
           lchar: CHAR;
           propW: BTPTR;  (* #NIL only of state*prop#{} *)
           propX: BTPTR;  (* #NIL only of state*prop#{} *)
           cellW: BTPTR;
           cellH: BTPTR;
           cellY: BTPTR;
           cellX: BTPTR;  (* allways zero now *)
           bases: ITPTR;
         END;

CONST
   prop   = {0};
   italic = {1};
   packed = {2};

END defFont.
```

11.5.4. Текст библиотеки BMG

```
DEFINITION MODULE BMG; (* nick 02-Mar-90. (c) KRONOS *)

IMPORT  SYSTEM, defScreen, defBMG, defFont;

TYPE
  TOOL  = defScreen.TOOL;   BMD    = defBMG.BMD;
  BLOCK = defScreen.BLOCK;  BITMAP = defBMG.BITMAP;

CONST
  rep = defScreen.rep;          xor = defScreen.xor;
  or  = defScreen.or;           bic = defScreen.bic;

---------------------- BitBlock Procedures ---------------------
                       ---------------------

PROCEDURE cross(VAR des: BLOCK; blk0,blk1: BLOCK);

PROCEDURE bblt(des: BITMAP; dtool: TOOL;
               x,y: INTEGER;
               sou: BITMAP; stool: TOOL; block: BLOCK);

---------------  Graphic Primitive Procedures  -----------------
               --------------------------------

PROCEDURE erase(bmd: BITMAP);

PROCEDURE fill (bmd: BITMAP;  t: TOOL;  block: BLOCK;
                  w: INTEGER; SEQ pattern: SYSTEM.WORD);

PROCEDURE pattern(bmd: BITMAP;  t: TOOL; block: BLOCK;
                  w,h: INTEGER; pattern: ARRAY OF SYSTEM.WORD);

PROCEDURE grid(bmd: BITMAP; t: TOOL; block: BLOCK; xstep,ystep:
INTEGER);

PROCEDURE dot  (bmd: BITMAP; t: TOOL; X,Y: INTEGER);

PROCEDURE line (bmd: BITMAP; t: TOOL; X0,Y0,X1,Y1: INTEGER);
PROCEDURE dline(bmd: BITMAP; t: TOOL; X0,Y0,X1,Y1: INTEGER;
                                        VAR r: SYSTEM.WORD);
PROCEDURE rect (bmd: BITMAP; t: TOOL; X0,Y0,X1,Y1: INTEGER);
PROCEDURE frame(bmd: BITMAP; t: TOOL; X0,Y0,X1,Y1: INTEGER);
PROCEDURE arc  (bmd: BITMAP; t: TOOL; X0,Y0,xa,ya,xb,yb,r:
INTEGER);
PROCEDURE arc3 (bmd: BITMAP; t: TOOL; x0,y0,x1,y1,x2,y2:
INTEGER);

PROCEDURE circle (bmd: BITMAP; t: TOOL; X,Y,R: INTEGER);
PROCEDURE circlef(bmd: BITMAP; t: TOOL; X,Y,R: INTEGER);
PROCEDURE ring   (bmd: BITMAP; t: TOOL; x0,y0,r1,r2: INTEGER);
```

```
PROCEDURE ellipse0 (bmd: BITMAP; t: TOOL; xc,yc,rx,ry: INTEGER);
PROCEDURE ellipse0f(bmd: BITMAP; t: TOOL; xc,yc,rx,ry: INTEGER);

PROCEDURE polyline0(bmd: BITMAP; t: TOOL; SEQ xy: INTEGER);
PROCEDURE polyline1(bmd: BITMAP; t: TOOL;     xy: ARRAY OF
INTEGER);

PROCEDURE trif(bmd: BITMAP; t: TOOL; x0,y0,x1,y1,x2,y2: INTEGER);

-----------------------------------------------------------------

PROCEDURE scroll(bmd: BITMAP; tool: TOOL; x,y: INTEGER);

PROCEDURE offset(bmd: BITMAP; x,y,layer: INTEGER;
             VAR adr: SYSTEM.ADDRESS; VAR bitoffset: INTEGER);

-----------------------------------------------------------------

PROCEDURE lenght(fnt: defFont.FONT; fmt: ARRAY OF CHAR;
                                SEQ arg: SYSTEM.WORD): INTEGER;

PROCEDURE width (fnt: defFont.FONT; fmt: ARRAY OF CHAR;
                                SEQ arg: SYSTEM.WORD): INTEGER;

PROCEDURE margin(fnt: defFont.FONT; fmt: ARRAY OF CHAR;
                                SEQ arg: SYSTEM.WORD): INTEGER;

PROCEDURE write(bmd: BITMAP;  tool: TOOL;  x,y: INTEGER;  fnt:
defFont.FONT;
                str: ARRAY OF CHAR;    pos,len: INTEGER);

PROCEDURE xwrite(bmd: BITMAP;  tool: TOOL;  x,y: INTEGER;  fnt:
defFont.FONT;
                 str: ARRAY OF CHAR;    pos,len: INTEGER):
INTEGER;

PROCEDURE print(bmd: BITMAP;  tool: TOOL;  x,y: INTEGER;  fnt:
defFont.FONT;
                fmt: ARRAY OF CHAR;    SEQ arg: SYSTEM.WORD);

PROCEDURE xprint(bmd: BITMAP;  tool: TOOL;  x,y: INTEGER;  fnt:
defFont.FONT;
                 fmt: ARRAY OF CHAR;    SEQ arg: SYSTEM.WORD):
INTEGER;

PROCEDURE writech(bmd: BITMAP; tool: TOOL; x,y: INTEGER;
                  fnt: defFont.FONT;        ch: CHAR);

END BMG.
```

11.5.5. Текст библиотеки Screen

```
DEFINITION MODULE Screen; (* Leo & nick 26-Mar-90. (c) KRONOS *)

IMPORT  SYSTEM;
IMPORT  defScreen;

TYPE
  STATUS  = defScreen.STATUS;
  STATE   = defScreen.STATE;
  COLOR   = defScreen.COLOR;
  PALETTE = defScreen.PALETTE;

VAL done: BOOLEAN;
   error: INTEGER;
   state: STATE;

PROCEDURE nop;
PROCEDURE set_ldcx(x: INTEGER);
PROCEDURE set_ldcy(y: INTEGER);
PROCEDURE set_palette(p: PALETTE; from,len: INTEGER);
PROCEDURE refresh (x,y,w,h: INTEGER);
PROCEDURE refreshw(x,y,w,h: INTEGER);

PROCEDURE op(cmd: INTEGER; SEQ args: SYSTEM.WORD);

CONST
   bitmap = defScreen.bitmap;
   pelmap = defScreen.pelmap;
   other  = defScreen.other;

PROCEDURE loophole(kind: INTEGER; VAR ext: SYSTEM.WORD);

PROCEDURE attach(name: ARRAY OF CHAR);
(* attach screen other then $SCR *)

END Screen.
```

11.5.6. Текст библиотеки Fonts

```
DEFINITION MODULE Fonts; (* Leo 27-Jan-91. (c) KRONOS *)

IMPORT  defFont;

TYPE
  FONT = defFont.FONT;

CONST
  prop  =defFont.prop;
  packed=defFont.packed;
  italic=defFont.italic;

VAL font: FONT;    (* default constant font always in memory *)
    done: BOOLEAN;
   error: INTEGER;

PROCEDURE new    (VAR fnt: FONT; w,h: INTEGER; f,l: CHAR; state:
BITSET);
PROCEDURE dispose(VAR fnt: FONT);

PROCEDURE read (VAR fnt: FONT; file_name: ARRAY OF CHAR);
PROCEDURE write(    fnt: FONT; file_name: ARRAY OF CHAR);

PROCEDURE pack  (fnt: FONT);   (* pack unpacked font           *)
PROCEDURE unpack(fnt: FONT);   (* unpack font for "dch" ussing *)

PROCEDURE copy(des: FONT; to: CHAR; sou: FONT; from: CHAR);
(* des[to]:=sou[from]                                          *)
(* des must be previously "new" with appropriate state,f,l,w,h *)

END Fonts.
```

Часть 12. ПОДСИСТЕМА WINDOWS

Предисловие соавтора

Эта часть будет настолько сильно изменяться, что можно рассматривать ее как проект (или сигнальный вариант) описания подсистемы окон. Необходимость в таком варианте возникла хотя бы потому, что читатель не имеет иной возможности ознакомиться с текстами библиотек и комментариями к ним.

Начало работе с окнами положил Д.Кузнецов (Leo), его перу принадлежат библиотеки pmPUP, pmWnd, pmWM. Продолжил тему А.Никитин (Nick), он соавтор библиотеки Wnd, являющейся естественным развитием библиотеки pmWnd. Предполагается, что в скором времени он завершит работу над presentation manager, базирующимся на библиотеке Wnd.

12.1. Текст библиотеки pmPUP

Это было самое начало работы с окнами. Здесь реализованы так называемые POPUP-окна (окна, образующие стек - в фиксированный момент времени доступно только верхнее окно для изменения информации в нем) и операции над ними, позволяющие использовать библиотеку для изготовления Presentation Manager. Рекомедуется использовать их, если требуется не очень большое количество окон и не требуется двигать их по экрану - это будет быстро.

```
DEFINITION MODULE pmPUP; (* Leo 18-Jan-91. (c) KRONOS *)

IMPORT  SYSTEM;       (* It might be a Presentation Manager *)
IMPORT  defScreen;
IMPORT  defFont;
IMPORT  BIO;

TYPE POPUP;
     MENU;
     DEBUG;
     ROLLER;
     DIREX;
     TEXT = DYNARR OF STRING;

VAL
    done: BOOLEAN;
   error: INTEGER;

   pnull: POPUP;
   mnull: MENU;
   rnull: ROLLER;
   dnull: DIREX;
   gnull: DEBUG;

   black: BITSET;  (* by default two last layers of display used
*)
  shadow: BITSET;  (* {2,3} for 4 plane display, for example
*)
  normal: BITSET;
  bright: BITSET;

      ch: CHAR;    (* last key read from keyboard by PM *)
    time: INTEGER; (* timeout substract time of waiting *)
   mx,my: INTEGER; (* mouse coordinates driven by PM    *)
 timeout: INTEGER;

    font: defFont.FONT;
   sfont: defFont.FONT;  (* special signs font font^.H x font^.H
*)

CONST (* for sfont: (sfont generated automaticaly) *)
```

```
  empty=0c;  (* empty (black) char         *)
  utria=1c;  (* triangle directed to up    *)
  dtria=2c;  (* triangle directed to down  *)
  ltria=3c;  (* triangle directed to left  *)
  rtria=4c;  (* triangle directed to right *)

PROCEDURE setplanes(shadow,normal,bright: BITSET);
PROCEDURE setcolors;

PROCEDURE pushtimeout(milisec: INTEGER);
PROCEDURE poptimeout;

---------------------------- blocks ----------------------------
                              --------

PROCEDURE block(b: defScreen.BLOCK; fill,pressed: BOOLEAN);

PROCEDURE panel(b: defScreen.BLOCK; VAR inter: defScreen.BLOCK;
                      fill,pressed: BOOLEAN);

PROCEDURE switch(b: defScreen.BLOCK; pressed: BOOLEAN);

PROCEDURE button(b: defScreen.BLOCK; pressed: BOOLEAN);

PROCEDURE inblock(x,y: INTEGER; b: defScreen.BLOCK): BOOLEAN;

PROCEDURE inblocks(x,y: INTEGER; SEQ b: defScreen.BLOCK):
INTEGER;

---------------------------- messages ----------------------------
                             ----------

PROCEDURE mwait(cpdkeys: BITSET; SEQ kbkeys: CHAR);
PROCEDURE wait (SEQ kbkeys: CHAR);

PROCEDURE message(   xc,yc: INTEGER; f: ARRAY OF CHAR; SEQ a:
SYSTEM.WORD);
PROCEDURE perror (er,xc,yc: INTEGER; f: ARRAY OF CHAR; SEQ a:
SYSTEM.WORD);

---------------------------- popups ----------------------------
                              --------

PROCEDURE pnew    (VAR pup: POPUP; x,y,w,h: INTEGER);
PROCEDURE pdispose(VAR pup: POPUP);

PROCEDURE popen (pup: POPUP);
PROCEDURE pclose(pup: POPUP);

PROCEDURE pclosed(pup: POPUP): BOOLEAN;

PROCEDURE pblock (pup: POPUP; VAR block: defScreen.BLOCK);
```

```
--------------------------- dialogbox ---------------------------
                             -----------

PROCEDURE diabox(xc,yc,w: INTEGER;
                 VAR str: ARRAY OF CHAR;
               promptfmt: ARRAY OF CHAR; SEQ args: SYSTEM.WORD);

PROCEDURE confirm(xc,yc,w: INTEGER;
                  VAR str: ARRAY OF CHAR;
                promptfmt: ARRAY OF CHAR; SEQ args: SYSTEM.WORD);


---------------------------- debug ----------------------------
                              -------

PROCEDURE gnew    (VAR debug: DEBUG; x,y,w,h: INTEGER);
PROCEDURE gdispose(VAR debug: DEBUG);

PROCEDURE gopen (debug: DEBUG);
PROCEDURE gclose(debug: DEBUG);

PROCEDURE gprint(debug: DEBUG; fmt: ARRAY OF CHAR; SEQ args:
SYSTEM.WORD);

---------------------------- rollers ----------------------------
                             ---------

CONST
  xup = {0};    xlf = {2};
  xdw = {1};    xrg = {3};

PROCEDURE rnew    (VAR rol: ROLLER; x,y,w,h: INTEGER; exit:
BITSET;
                   titfmt: ARRAY OF CHAR; SEQ args:
SYSTEM.WORD);
PROCEDURE rdispose(VAR rol: ROLLER);
(* roller always maked with title *)

PROCEDURE rsettext(rol: ROLLER; text: TEXT; top,line: INTEGER);
(* if line<0 or top<0 (or both) automaticaly setted by internal
AI *)

PROCEDURE rsetstr (rol: ROLLER;            alt: INTEGER;
                   fmt: ARRAY OF CHAR; SEQ arg: SYSTEM.WORD);

PROCEDURE rgettext(rol: ROLLER; VAR text: TEXT);

PROCEDURE ropen (rol: ROLLER);
PROCEDURE rclose(rol: ROLLER);

PROCEDURE rselect  (rol: ROLLER);
PROCEDURE rselected(rol: ROLLER): BOOLEAN;
```

```
PROCEDURE rchoose(rol: ROLLER; alt: INTEGER);

PROCEDURE ralt   (rol: ROLLER): INTEGER;

PROCEDURE rblocks(rol: ROLLER; VAR main,txt,tit,off,up,rl,dw:
defScreen.BLOCK);

---------------------------- menus ----------------------------
                              -------

PROCEDURE mnew    (VAR m: MENU; x,y,w,h: INTEGER; exit: BITSET;
                  titfmt: ARRAY OF CHAR; SEQ arg: SYSTEM.WORD);
PROCEDURE mdispose(VAR m: MENU);

PROCEDURE mprint(m: MENU;                 alt: INTEGER;
               fmt: ARRAY OF CHAR;  SEQ args: SYSTEM.WORD);

PROCEDURE mread(m: MENU; alt: INTEGER; VAR s: ARRAY OF CHAR);

PROCEDURE mhotkey(m: MENU; alt: INTEGER; hotkey: CHAR; capequ:
BOOLEAN);

PROCEDURE mopen (m: MENU);
PROCEDURE mclose(m: MENU);

PROCEDURE mselect  (m: MENU);
PROCEDURE mselected(m: MENU): BOOLEAN;

PROCEDURE mchoose(m: MENU; alt: INTEGER);

PROCEDURE malt(m: MENU): INTEGER;

PROCEDURE mblocks(m: MENU; VAR main,txt,tit,off:
defScreen.BLOCK);

---------------------------- direx ----------------------------
                              -------

CONST (* items *)
  dirs     = {0};
  files    = {1};
  hidden   = {2};
  devices  = {3};
  all      = dirs + files + hidden + devices;
  standard = dirs + files;

PROCEDURE dnew    (VAR drx: DIREX;  x,y,w,h: INTEGER; exit:
BITSET;
                    cdname: ARRAY OF CHAR;
                   pattern: ARRAY OF CHAR;
                     items: BITSET);
PROCEDURE ddispose(VAR drx: DIREX);
```

```
PROCEDURE dopen (drx: DIREX);
PROCEDURE dclose(drx: DIREX);

PROCEDURE dchoose(drx: DIREX; filename: ARRAY OF CHAR);
PROCEDURE dselect(drx: DIREX);

PROCEDURE dselected(drx: DIREX): BOOLEAN;

PROCEDURE dfilename(drx: DIREX; VAR fname: ARRAY OF CHAR);
PROCEDURE dfullname(drx: DIREX; VAR fname: ARRAY OF CHAR);
PROCEDURE dopenfile(drx: DIREX; VAR  f: BIO.FILE; biomode: ARRAY
OF CHAR);
PROCEDURE dcd      (drx: DIREX; VAR cd: BIO.FILE);

PROCEDURE dblocks(drx: DIREX; VAR main,txt,tit,off,up,rl,dw:
defScreen.BLOCK);

-----------------------------------------------------------------

PROCEDURE zoom;

END pmPUP.

-----------------------------------------------------------------
```

NOTES:

1. "*dispose" never change "done" & "error" by itself!
2. "*blocks" defined only after first "*open" or "*select"

ch keeps value of last key readen from Keyboard by PM (000c if none)

mx,my keep values of x- & y-coordinates of mouse cursor driven by PM.
(any panel set cursor on it "off" block at open time)

time keeps the value of the time not spended for waiting.
Application may use the value timeout-time to
calcuate the time spended by human for last input.

```
PROCEDURE setplanes(black,shadow,normal,bright: BITSET);
PROCEDURE setcolors;
-------------------
```

By default PM used last (leftmost or higest) 2 planes on dispay. Application may orevride this by calling "setplanes".

PM not changed screen palette, till he is asked about it by calling "setcolors". The last one set 4 gradations of grey (if it possible) for current plane combination used for colors

```
black, shadow, normal & bright.


PROCEDURE pushfont(font: defFont.FONT);
PROCEDURE popfont;
-----------------

     Changes default font for printing text in PM.
     Be carefull! Current "font" linked with any "new"
popup, menu, roller, direx and will be used with it
as long as this object exist.
     Application may push and pop fonts freely, but it
will not dispose appropriate font before all object
referenced to it disposed!

PROCEDURE pushtimeout(milisec: INTEGER);
PROCEDURE poptimeout;
--------------------
     Changes current timeout time (in miliseconds).
     This timeout will be actual for all procedures
waiting mouse or keyboard input (such as "wait",
"message", "perror", "select" and so on) until next
push or pop call will be executed.
     To prevent timeout waiting execute call
"pushtimeout(-1)" or "pushtimeout(MAX(INTEGER))"
(realy it will be timeout for 24 days and nights).
     Aoolication may determine that "timeout" condition
was occured by testing PM.time<=0!

PROCEDURE inblock (x,y: INTEGER;     b: defScreen.BLOCK):
BOOLEAN;
PROCEDURE inblocks(x,y: INTEGER; SEQ b: defScreen.BLOCK):
INTEGER;
------------------
     "inblock"  returns TRUE when x,y IN block "b".
     "inblocks" returns the number of the block
                in wich x,y points otherwise -1.

PROCEDURE mwait(cpdkeys: BITSET; SEQ kbkeys: CHAR);
PROCEDURE wait (SEQ kbkeys: CHAR);
---------------

  "mwait" Waits for one of "kbkeys"  pressed on keyboard
         or one of "cpdkeys" pressed on mouse
         or timeout occured
    wait(....,0c)    waits any key pressed on keyboard
    wait(time,{})    waits for CR or ESC
    wait(time,{1,2}) waits for buttons 1 or 2 on mouse
If cpdkeys={} "mwait" do not touch mouse at all.
  "wait" is equal to mwait({0..CPD.state^.nokeys-1},kbkeys);

  "done" always TRUE.
```

```
PROCEDURE message(xc,yc   : INTEGER; f: ARRAY OF CHAR; SEQ a:
SYSTEM.WORD);
PROCEDURE perror (xc,yc,er: INTEGER; f: ARRAY OF CHAR; SEQ a:
SYSTEM.WORD);
-----------------

     "message" print message (format,args) in the appropriated
box
with center xc,yc on the screen and waits, driving the mouse
cursor, until one of the next conditions will happen:
        - timeout occured (see timeout)
        - CR of ESC pressed on the Keyboard;
        - leftmost  mouse key pressed when cursor in "off" button
          of message panel;
        - rightmost mouse key pressed (cursor anywhere);
"perror"  is equal to sequence of calls:
          Lexicon.perror(bump,er,f,a);
          message(t,xc,yc,"%s",bump).
     Both procedures set "done"=FALSE only when it's impossible
to open appropriate window or parameters is bad. "done"=TRUE at
least
human press ESC or timeout happens.

PROCEDURE pclosed(pup: POPUP): BOOLEAN;
-----------------

   Window opened or closed?
   returns TRUE of FALSE; never change "done", "error".
   returns TRUE for bad POPUP object.



NEWS


   psave
   prestore

   direx:  bright for directories.

   exit keys on CPD (diabox, message e.t.c)

   block for menu N'th alt
```

12.2. Текст библиотеки pmWnd

Библиотека реализует механизмы работы с топологией окон и базовый набор графических примитивов над ними.

```
DEFINITION MODULE pmWnd; (* Leo 09-Apr-91. (c) KRONOS *)

IMPORT  SYSTEM, defScreen, defFont;

TYPE
  TOOL   = defScreen.TOOL;
  BLOCK  = defScreen.BLOCK;
  FONT   = defFont.FONT;
  WINDOW = POINTER TO WNDESC;
  PAINT  = PROCEDURE (WINDOW,INTEGER,INTEGER,INTEGER,INTEGER);
  WNDESC = RECORD
             x,y   : INTEGER;
             w,h   : INTEGER;
             mode  : BITSET;
             fore  : BITSET;
             back  : BITSET;
             mask  : BITSET;
             image : BOOLEAN;
             closed: BOOLEAN;
             board : PAINT;
             paint : PAINT;
             mgr   : SYSTEM.WORD;
             obj   : SYSTEM.WORD;
             desc  : SYSTEM.WORD;
             inner : TOOL;
             full  : TOOL;
           END;

VAL done: BOOLEAN;
   error: INTEGER;

    scrW: INTEGER;  (* phisical screen W    *)
    scrH: INTEGER;  (* phisical screen H    *)
    scrM: BITSET;   (* phisical screen Mask *)

     top: WINDOW;
  bottom: WINDOW;
 desktop: WINDOW;

    kind: INTEGER;   CONST
            bitmap = 0;
            pelmap = 1;
            gamma  = 2;
            ega    = 3;
            vga    = 4;

CONST (* window modes: *)
```

```
  scr    = {0};
  img    = {1};
  deep   = {2};
  normal = scr+img;

  (* tool modes: *)

  xor    = defScreen.xor;
  or     = defScreen.or ;
  bic    = defScreen.bic;
  rep    = defScreen.rep;

PROCEDURE new    (VAR wnd: WINDOW);
PROCEDURE dispose(VAR wnd: WINDOW);

PROCEDURE open  (wnd: WINDOW);
PROCEDURE close (wnd: WINDOW);

PROCEDURE move  (wnd: WINDOW; x,y : INTEGER);
PROCEDURE resize(wnd: WINDOW; w,h : INTEGER);
PROCEDURE mask  (wnd: WINDOW; fore,back: BITSET);

PROCEDURE inner (wnd: WINDOW; x,y,w,h: INTEGER);

PROCEDURE upperthen(wnd: WINDOW; und: WINDOW): BOOLEAN;
(* TRUE iff window "wnd" upper then "und". (Note:
upperthen(w,w)=FALSE!) *)

PROCEDURE ontop   (wnd: WINDOW);
PROCEDURE onbottom(wnd: WINDOW);

PROCEDURE putover (wnd: WINDOW; under: WINDOW);
PROCEDURE putunder(wnd: WINDOW; over : WINDOW);

PROCEDURE refreshbox  (wnd: WINDOW; x,y,w,h: INTEGER);
PROCEDURE refreshboard(wnd: WINDOW; x,y,w,h: INTEGER);
PROCEDURE refresh     (wnd: WINDOW);
PROCEDURE refreshall;

PROCEDURE savebox(wnd: WINDOW; x,y,w,h: INTEGER);

PROCEDURE locate(x,y: INTEGER): WINDOW;
PROCEDURE up    (wnd: WINDOW ): WINDOW;
PROCEDURE dw    (wnd: WINDOW ): WINDOW;

PROCEDURE image  (wnd: WINDOW; on   : BOOLEAN);
PROCEDURE painter(wnd: WINDOW; paint: PAINT);
PROCEDURE boarder(wnd: WINDOW; board: PAINT);

PROCEDURE object (wnd: WINDOW; obj  : SYSTEM.WORD);
PROCEDURE manager(wnd: WINDOW; mgr  : SYSTEM.WORD);

-----------------------------------------------------------------
```

```
PROCEDURE mode(wnd: WINDOW; mode: BITSET);

-----------------------------------------------------------------

PROCEDURE erase(wnd: WINDOW);
PROCEDURE fill (wnd: WINDOW; tool: TOOL;  block: BLOCK;
                 w: INTEGER; SEQ p: SYSTEM.WORD);

PROCEDURE bblt(des: WINDOW; dtool: TOOL;  x,y: INTEGER;
              sou: WINDOW; stool: TOOL;  blk: BLOCK);

PROCEDURE pattern(wnd: WINDOW;  t: TOOL; block: BLOCK;
                 w,h: INTEGER; p: ARRAY OF SYSTEM.WORD);

PROCEDURE grid  (wnd: WINDOW;  t: TOOL; block: BLOCK;
xstep,ystep: INTEGER);

PROCEDURE dot   (wnd: WINDOW; tool: TOOL; x, y       : INTEGER);
PROCEDURE line  (wnd: WINDOW; tool: TOOL; x0,y0,x1,y1: INTEGER);
PROCEDURE dline (wnd: WINDOW; tool: TOOL; x0,y0,x1,y1: INTEGER;
VAR r: SYSTEM.WORD);
PROCEDURE hline (wnd: WINDOW; tool: TOOL; x0,y0,x1   : INTEGER);
PROCEDURE vline (wnd: WINDOW; tool: TOOL; x0,y0,y1   : INTEGER);
PROCEDURE rect  (wnd: WINDOW; tool: TOOL; x0,y0,x1,y1: INTEGER);
PROCEDURE frame (wnd: WINDOW; tool: TOOL; x0,y0,x1,y1: INTEGER);
PROCEDURE scroll(wnd: WINDOW; tool: TOOL; x,y        : INTEGER);

PROCEDURE trif  (wnd: WINDOW; tool: TOOL; x0,y0,x1,y1,x2,y2:
INTEGER);

PROCEDURE arc   (wnd: WINDOW; tool: TOOL; xc,yc,xa,ya,xb,yb,r:
INTEGER);
PROCEDURE arc3  (wnd: WINDOW; tool: TOOL; x0,y0,x1,y1,x2,y2:
INTEGER);

PROCEDURE ring   (wnd: WINDOW; tool: TOOL; xc,yc,r0,r1: INTEGER);
PROCEDURE circle (wnd: WINDOW; tool: TOOL; x,y,r: INTEGER);
PROCEDURE circlef(wnd: WINDOW; tool: TOOL; x,y,r: INTEGER);

PROCEDURE print (wnd: WINDOW; tool: TOOL; x,y: INTEGER;
                 fnt: defFont.FONT;
                 fmt: ARRAY OF CHAR;     SEQ  arg: SYSTEM.WORD);

PROCEDURE xprint(wnd: WINDOW; tool: TOOL; x,y: INTEGER;
                 fnt: defFont.FONT;
                 fmt: ARRAY OF CHAR; SEQ  arg: SYSTEM.WORD):
INTEGER;

PROCEDURE write (wnd: WINDOW; tool: TOOL; x,y: INTEGER;
                 fnt: defFont.FONT;
                 str: ARRAY OF CHAR;  pos,len: INTEGER);
```

```
PROCEDURE xwrite(wnd: WINDOW; tool: TOOL; x,y: INTEGER;
                 fnt: defFont.FONT;
                 str: ARRAY OF CHAR;  pos,len: INTEGER): INTEGER;

PROCEDURE writech(wnd: WINDOW; tool: TOOL; x,y: INTEGER;
                  fnt: defFont.FONT;        ch: CHAR);

PROCEDURE lenght(fnt: defFont.FONT;
                 fmt: ARRAY OF CHAR; SEQ arg: SYSTEM.WORD):
INTEGER;

PROCEDURE width (fnt: defFont.FONT;
                 fmt: ARRAY OF CHAR; SEQ arg: SYSTEM.WORD):
INTEGER;

PROCEDURE margin(fnt: defFont.FONT;
                 fmt: ARRAY OF CHAR; SEQ arg: SYSTEM.WORD):
INTEGER;

END pmWnd.
```

12.3. Текст библиотеки pmWM

В интерактивных графических системах возникает необходимость в непосредственном управлении топологией оконной структуры. Библиотека реализует возможности: передвижение окон, изменение их размеров, тасование окон, добавление пользователем новых операций над окнами и связанными с ними структурами данных.

```
DEFINITION MODULE pmWM; (* Leo 23-Apr-91. (c) KRONOS *)

IMPORT  SYSTEM, pmWnd, defFont;

TYPE WINDOW = pmWnd.WINDOW;

VAL done: BOOLEAN;
   error: INTEGER;

  ssfont: defFont.FONT;  (* special signs font         *)

  active: WINDOW;        (* last activated window      *)
 abutton: INTEGER;       (* last switched button no    *)

  closed: BOOLEAN;       (* active window m.b. closed  *)

   moved: BOOLEAN;       (* active window m.b. moved   *)
   moveX: INTEGER;       (* new coordinates for        *)
   moveY: INTEGER;       (*        active  window      *)

 resized: BOOLEAN;       (* active window m.b. resized *)
 resizeW: INTEGER;       (* new sizes for              *)
 resizeH: INTEGER;       (*        active  window      *)

VAR
   black: BITSET;
  shadow: BITSET;
  normal: BITSET;
  bright: BITSET;

CONST (* ssfont chars *)
  ssempty   = 00c;
  ssmove    = 01c;
  ssresize  = 02c;
  ssclose   = 03c;
  ssontop   = 04c;
  ssonbot   = 05c;
  sszoomin  = 06c;
  sszoomout = 07c;
  ssleft    = 10c;
  ssright   = 11c;
  ssup      = 12c;
  ssdw      = 13c;
```

```
  sspan     = 14c;
  sseye     = 15c;
  sszoom    = 16c;
  ssgrid    = 17c;
  sscascade = 20c;
  ssfire    = 21c;
  ssicon    = 22c;
  sstool    = 23c;

PROCEDURE new    (VAR w: WINDOW);
PROCEDURE dispose(VAR w: WINDOW);

PROCEDURE enable (w: WINDOW); (* enable/disable WM control for
*)
PROCEDURE disable(w: WINDOW); (* move, resize, close (default
enable) *)

PROCEDURE monitor;

CONST (* corners *)
  luc = 0;  ruc = 2;
  ldc = 1;  rdc = 3;

PROCEDURE button (w: WINDOW; no,corner,x,y,w,h: INTEGER); (*
inner sizes *)
PROCEDURE pressed(w: WINDOW; button: INTEGER): BOOLEAN;
PROCEDURE toggle (w: WINDOW; button: INTEGER; pressed: BOOLEAN);

PROCEDURE buttoncolors(w: WINDOW; no: INTEGER; fore,pressed,back:
BITSET);

PROCEDURE print  (w: WINDOW; button: INTEGER; font: defFont.FONT;
                             format: ARRAY OF CHAR; SEQ args:
SYSTEM.WORD);

END pmWM.
```

12.4. Текст библиотеки Wnd

```
DEFINITION MODULE Wnd; (* nick 14-Dec-91. (c) KRONOS *)

(*   Реализует механизмы работы с топологией окон и базовый
набор графических примитивов над ними.
     От pmWnd отличается тем, что понятие окна расширено
(введено понятие "подокна"), что позволяет создавать
древовидные оконно-подоконные структуры.
*)

IMPORT  SYSTEM, defScreen, defFont, defBMG;

TYPE
  TOOL   = defScreen.TOOL;
  BLOCK  = defScreen.BLOCK;
  FONT   = defFont.FONT;

  WINDOW = POINTER TO WNDESC;

  PAINT  = PROCEDURE (WINDOW,INTEGER,INTEGER,INTEGER,INTEGER);

  RESIZE = PROCEDURE (WINDOW,INTEGER,INTEGER);

  WNDESC = RECORD (* READ ONLY! *)
             x,y    : INTEGER;  (* relative "desk"      *)
             w,h    : INTEGER;  (* pixels               *)
             sx,sy  : INTEGER;  (* relative "desktop"   *)
             mode   : BITSET;   (* method of drawing    *)
             fore   : BITSET;
             back   : BITSET;
             mask   : BITSET;
             image  : BOOLEAN;  (* has image ?          *)
             closed : BOOLEAN;  (* is  closed now?      *)
             visible: BOOLEAN;  (* is  visible now?     *)
             tool   : TOOL;     (* default tool         *)
             desk   : WINDOW;   (* own    desktop       *)
             top    : WINDOW;   (* upper  subwindow     *)
             bottom : WINDOW;   (* downer subwindow     *)

             resize : RESIZE;   (* pre resize action    *)
             corner : INTEGER;  (* resize will move it  *)
             minw   : INTEGER;  (* min sizes of window  *)
             minh   : INTEGER;
             maxw   : INTEGER;  (* max sizes of window  *)
             maxh   : INTEGER;
             refresh: PAINT;
           END;

CONST (* corners: *) ruc=0; rdc=1; ldc=2; luc=3; (* default "ruc"
*)
```

```
VAL done: BOOLEAN;
   error: INTEGER;

    scrW: INTEGER;  (* phisical screen W    *)
    scrH: INTEGER;  (* phisical screen H    *)
    scrM: BITSET;   (* phisical screen Mask *)

 desktop: WINDOW;

       B: defBMG.BITMAP;

CONST (* window modes: *)
  scr    = {0};  (* drawing on screen in "fore" layers only *)
  img    = {1};  (* drawing in image  in "fore" layers only *)
  deep   = {2};  (* drawing in image  and on screen in all
possible layers *)
  glass  = {3};  (* drawing over subwindows *)
  whole  = {4};  (* refresh whole window after resize     *)
  normal = scr+img; (* default window "mode" after create *)

  (* tool modes: *)

  xor    = defScreen.xor;
  or     = defScreen.or ;
  bic    = defScreen.bic;
  rep    = defScreen.rep;

PROCEDURE create (VAR wnd: WINDOW; desk: WINDOW; x,y,w,h:
INTEGER;
                fore,back: BITSET; refresh: PAINT);

PROCEDURE dispose(VAR wnd: WINDOW);

PROCEDURE image(wnd: WINDOW; x,y,w,h: INTEGER);
(* never be called. may by used only as "create" or "painter"
parameter *)

PROCEDURE painter(wnd: WINDOW; refresh: PAINT);
(* set another refresh procedure for the window.
 * if "refresh" will stay to "image" then new image created
 *)

PROCEDURE open  (wnd: WINDOW);
PROCEDURE close (wnd: WINDOW);

PROCEDURE move  (wnd: WINDOW; x,y: INTEGER);

PROCEDURE resize(wnd: WINDOW; w,h: INTEGER);

PROCEDURE moveandresize(wnd: WINDOW; x,y,w,h: INTEGER);

PROCEDURE resizer(wnd: WINDOW; resize: RESIZE);
PROCEDURE minmax (wnd: WINDOW; minW,minH,maxW,maxH: INTEGER);
```

```
PROCEDURE corner (wnd: WINDOW; corner: INTEGER);
(* only "ruc", "rdc" are implemented yet *)

PROCEDURE upperthen(wnd: WINDOW; und: WINDOW): BOOLEAN;
(* TRUE iff window "wnd" upper then "und". (Note:
upperthen(w,w)=FALSE!) *)

PROCEDURE ontop   (wnd: WINDOW);
PROCEDURE onbottom(wnd: WINDOW);

PROCEDURE putover (wnd: WINDOW; under: WINDOW);
PROCEDURE putunder(wnd: WINDOW; over : WINDOW);

PROCEDURE pass(wnd: WINDOW; desk: WINDOW);

PROCEDURE refreshbox(wnd: WINDOW; x,y,w,h: INTEGER);
PROCEDURE refresh   (wnd: WINDOW);
PROCEDURE refreshall(wnd: WINDOW);

PROCEDURE savebox(wnd: WINDOW; x,y,w,h: INTEGER);

PROCEDURE locate(x,y: INTEGER): WINDOW;
PROCEDURE up    (wnd: WINDOW ): WINDOW;
PROCEDURE dw    (wnd: WINDOW ): WINDOW;

PROCEDURE assign(wnd: WINDOW; name: ARRAY OF CHAR;     obj:
SYSTEM.WORD);
PROCEDURE object(wnd: WINDOW; name: ARRAY OF CHAR; VAR obj:
SYSTEM.WORD);
PROCEDURE delete(wnd: WINDOW; name: ARRAY OF CHAR);

PROCEDURE iterobjects(wnd: WINDOW);
PROCEDURE nextobject (wnd: WINDOW; VAR name: ARRAY OF CHAR;
                                   VAR obj : SYSTEM.WORD):
BOOLEAN;

-----------------------------------------------------------------

PROCEDURE mode(wnd: WINDOW; mode: BITSET);

-----------------------------------------------------------------

PROCEDURE erase(wnd: WINDOW);
PROCEDURE fill (wnd: WINDOW; tool: TOOL;  block: BLOCK;
                 w: INTEGER; SEQ p: SYSTEM.WORD);

PROCEDURE bblt(des: WINDOW; dtool: TOOL;  x,y: INTEGER;
               sou: WINDOW; stool: TOOL;  blk: BLOCK);

PROCEDURE pattern(wnd: WINDOW;  t: TOOL; block: BLOCK;
                  w,h: INTEGER; p: ARRAY OF SYSTEM.WORD);
```

```
PROCEDURE grid  (wnd: WINDOW;  t: TOOL; block: BLOCK;
xstep,ystep: INTEGER);

PROCEDURE dot   (wnd: WINDOW; tool: TOOL; x, y       : INTEGER);
PROCEDURE line  (wnd: WINDOW; tool: TOOL; x0,y0,x1,y1: INTEGER);
PROCEDURE dline (wnd: WINDOW; tool: TOOL; x0,y0,x1,y1: INTEGER;
VAR r: SYSTEM.WORD);
PROCEDURE hline (wnd: WINDOW; tool: TOOL; x0,y0,x1   : INTEGER);
PROCEDURE vline (wnd: WINDOW; tool: TOOL; x0,y0,y1   : INTEGER);
PROCEDURE rect  (wnd: WINDOW; tool: TOOL; x0,y0,x1,y1: INTEGER);
PROCEDURE frame (wnd: WINDOW; tool: TOOL; x0,y0,x1,y1: INTEGER);
PROCEDURE scroll(wnd: WINDOW; tool: TOOL; x,y        : INTEGER);

PROCEDURE trif  (wnd: WINDOW; tool: TOOL; x0,y0,x1,y1,x2,y2:
INTEGER);

PROCEDURE arc   (wnd: WINDOW; tool: TOOL; xc,yc,xa,ya,xb,yb,r:
INTEGER);
PROCEDURE arc3  (wnd: WINDOW; tool: TOOL; x0,y0,x1,y1,x2,y2:
INTEGER);

PROCEDURE ring   (wnd: WINDOW; tool: TOOL; xc,yc,r0,r1: INTEGER);
PROCEDURE circle (wnd: WINDOW; tool: TOOL; x,y,r: INTEGER);
PROCEDURE circlef(wnd: WINDOW; tool: TOOL; x,y,r: INTEGER);

PROCEDURE print (wnd: WINDOW; tool: TOOL; x,y: INTEGER;
                 fnt: defFont.FONT;
                 fmt: ARRAY OF CHAR;     SEQ  arg: SYSTEM.WORD);

PROCEDURE xprint(wnd: WINDOW; tool: TOOL; x,y: INTEGER;
                 fnt: defFont.FONT;
                 fmt: ARRAY OF CHAR; SEQ  arg: SYSTEM.WORD):
INTEGER;

PROCEDURE write (wnd: WINDOW; tool: TOOL; x,y: INTEGER;
                 fnt: defFont.FONT;
                 str: ARRAY OF CHAR;  pos,len: INTEGER);

PROCEDURE xwrite(wnd: WINDOW; tool: TOOL; x,y: INTEGER;
                 fnt: defFont.FONT;
                 str: ARRAY OF CHAR;  pos,len: INTEGER): INTEGER;

PROCEDURE writech(wnd: WINDOW; tool: TOOL; x,y: INTEGER;
                  fnt: defFont.FONT;        ch: CHAR);

PROCEDURE lenght(fnt: defFont.FONT;
                 fmt: ARRAY OF CHAR; SEQ arg: SYSTEM.WORD):
INTEGER;

PROCEDURE width (fnt: defFont.FONT;
                 fmt: ARRAY OF CHAR; SEQ arg: SYSTEM.WORD):
INTEGER;
```

```
PROCEDURE margin(fnt: defFont.FONT;
                 fmt: ARRAY OF CHAR; SEQ arg: SYSTEM.WORD):
INTEGER;

END Wnd.

(*
PROCEDURE resize(wnd: WINDOW; w,h: INTEGER);
-------------------------------------------

      wnd^.resize(wnd,w,h)

   called (after possible image resize and) as resize
   notification before redraw.
   old coordinates and sizes are still in wnd^.(sx,sy,w,h).

      There are two kinds of action that may be done by
   wnd^.resize according to new window size

   For windows with IMAGE:
      moves and resize subwindows if need it.

      (image already resized when wnd^.resize called)

       set wnd^.mode(img) and redraw all extended rectangle of
       window (if exist). They will be refreshed later
       automaticly.

      set wnd^.mode(normal) and redraw all retained rectangles of
      window if them exist and it's needed. Mode normal need to
      refresh this rectangles because automaticly refresh of
      retained rectangle will NOT be executed.


   For windows WITHOUT image:
      moves and resize subwindows if need it.
      set wnd^.mode(normal) and redraw all retained rectangles of
      window if them exist and it's needed. Mode normal need to
      refresh this  rectangles because automaticly refresh of
      retained rectangle will NOT be executed.

      be ready to redraw new parts of window when wnd^.refresh
      will be called.


     (0,0) coordinates still linked to left down corner of the
     window  after resize.


     when corner =
       ruc: resize chage sizes moving right upper corner

       rdc: resize chage sizes moving right down  corner
```

```
            and so on


PROCEDURE moveandresize(wnd: WINDOW; x,y,w,h: INTEGER);
-------------------------------------------------------


      wnd^.resize(wnd,w,h)

   called (after possible image resize and) as resize
   notification  before redraw.
   old coordinates and sizes are still in wnd^.(sx,sy,w,h).

   After moveandresize whole window will de redrown automaticly.

*)
```