**II. Утилиты программирования**

|  |  |  |
| --- | --- | --- |
| II - 1 | **Вступление** | Утилиты программирования |
| II - 2 | **Соглашения** | Использование утилит |
| II - 9 | **ROM** | Написание кода только для чтения |
| II - 11 | **anat** | Транслятор A-Natural |
| II - 13 | **as.80** | Ассемблер A-Natural для 8080 |
| II - 15 | **c** | Драйвер многопроходной команды |
| II - 18 | **cpm** | Обслуживание дискет CP/M |
| II - 20 | **db** | Редактор/отладчик двоичных файлов |
| II - 28 | **dis80** | Перевод A-Natural перемещаемых файлов |
| II - 29 | **hex** | Перевод объектных файлов в формат ASCII |
| II - 33 | **lib** | Обслуживание библиотек |
| II - 36 | **link** | Объединение объектных файлов |
| II - 41 | **lord** | Порядок библиотек |
| II - 43 | **p1** | Разбор программ на языке C |
| II - 45 | **p2.80** | Генерация кода для программ 8080 |
| II - 47 | **pp** | Препроцессор |
| II - 49 | **prof** | Создание профиля выполнения |
| II - 52 | **ptc** | Переводчик с Pascal на C |
| II - 54 | **rel** | Проверка объектных файлов |

**НАЗВАНИЕ**

**Вступление** - Утилиты программирования

**ОПИСАНИЕ**

Утилиты, описанные в этом разделе, поставляются с операционной системой Idris или с кросс-компилятором в любой другой операционной системе для создания и отладки программ, написанных на ассемблере, C или Pascal. Поскольку модули могут быть отдельно переведены в объектный код, затем объединены (с помощью link) в один или несколько этапов для создания исполняемого объекта, каждый модуль может быть написан на наиболее подходящем из поддерживаемых языков.

Библиотеки могут быть созданы (с помощью lib) из модулей, которые включаются только по мере необходимости (с помощью link) при создании программы. Таким образом, предоставляемые библиотечные процедуры не накладывают никаких ограничений на пространство для программ, которые их не используют, и каждый пользователь имеет право добавить к этому набору или изменить любой из его членов.

Если явно не обозначена как машинно-зависимая, каждая утилита в этом разделе может использоваться для разработки программ для любой из машин, поддерживаемых компиляторами Whitesmiths, Ltd. Это повышает единообразие реализации языка и упрощает программирование для более чем одной целевой машины.

**ОШИБКИ**

В системах, где генератор кода (p2) взаимодействует с существующим ассемблером, не являющимся Idris, многие из этих утилит не предоставляются, следовательно, большая часть этого раздела может быть неактуальной.

**НАЗВАНИЕ**

**Соглашения** - использование утилит

**ОПИСАНИЕ**

Каждая из утилит, описанных в этом разделе, представляет собой отдельную "программу", которую можно запустить или "вызвать", набрав "командную строку", обычно в ответ на "приглашение", такое как в Idris "% ". Этот документ предоставляет систематическое руководство по соглашениям, которые регулируют порядок указания командных строк. В нем также содержится краткая информация о расположении следующих ниже описаний отдельных служебных программ, чтобы вы знали, какая информация, где и в каком формате отображается.

**Командные строки**

В общем случае командная строка состоит из трех основных частей: имени программы, необязательного ряда флагов и ряда аргументов, не связанных с флагами, обычно приводимых в таком порядке. Каждый элемент командной строки обычно представляет собой строку, отделенную пробелом от всех остальных. Чаще всего, конечно, имя программы - это просто имя (файла) утилиты, которую вы хотите запустить. Флаги, если они заданы, изменяют некоторые аспекты того, что делает утилита, и обычно предшествуют всем другим аргументам, потому что утилита должна интерпретировать их, прежде чем она сможет безопасно интерпретировать остальную часть командной строки.

Значение аргументов, не являющихся флагами, сильно зависит от запускаемой утилиты, но есть пять общих классов командных строк, представленных здесь (где это возможно) с примерами, взятыми из портативных программных средств, поскольку они работают практически одинаково во многих разных системах:

1. Имя программы и флаги, за которыми следует ряд имен файлов. Эти программы (называемые фильтрами) обрабатывают каждый файл, названный в указанном порядке. Примером может служить sort.
2. Имя программы и флаги, за которыми следует ряд аргументов, которые не являются именами файлов, а строками, которым программа дает некоторую другую интерпретацию. Программа echo - одна из таких утилит.
3. Имя программы и обязательный аргумент, который **предшествует** остальной части командной строки, за которым следуют флаги и другие аргументы, grep и tr принадлежат к этому классу.
4. Имя программы и флаги, за которыми следует ряд имен файлов "источника" и одно имя файла "назначения". Имя файла, которое должно быть сопряжено с каждым именем источника, создается путем применения "завершения пути" к имени назначения. Например, имя файла назначения может быть каталогом файлов, имена которых совпадают с именами исходных файлов. Эти программы (называемые направленными утилитами) затем выполняют некоторые операции с использованием каждой пары файлов, одним из примеров является diff.
5. Имя программы и флаги, за которыми следует командная строка, которую утилита выполняет после изменения некоторых условий, при которых она будет работать. Это, как правило, более сложные инструменты, такие как error, которые перенаправляют сообщения об ошибках в Idris.

Краткое описание классов командной строки выглядит следующим образом:

|  |  |  |
| --- | --- | --- |
| **Класс** | **Пример** | **Синтаксис** |
| filter | sort | <progname> <flags> <files> |
| string arguments | echo | <progname> <flags> <args> |
| mandatory argument | grep | <progname> <arg> <flags> <files> |
| directed | diff | <progname> <flags> <files> <dest> |
| prefix | error | <progname> <flags> <command> |

Обратите внимание, что, как правило, <flags> необязательны в любой командной строке.

**Флаги**

Флаги используются для выбора параметров или указания параметров при запуске программы. Это аргументы командной строки, распознаваемые по их первому символу, который всегда является '-' или '+'. Название флага (обычно состоящее из одной буквы) следует сразу за ведущим '-' или '+'. Некоторые флаги являются просто индикаторами YES/NO - либо они названы, либо нет, - но за другими должна следовать некоторая дополнительная информация или "значение". Это значение, если требуется, может быть целым числом, строкой символов или только одним символом. Строковые или целочисленные значения указываются в качестве оставшейся части аргумента, в котором указан флаг. Им также могут предшествовать пробелы и, следовательно, они могут быть заданы в качестве следующего аргумента.

Флаги, присвоенные утилите, могут отображаться в любом порядке, и два или более могут быть объединены в одном аргументе, если второй флаг нельзя ошибочно принять за значение, соответствующее первому. Некоторые флаги имеют только значение и не имеют имени. Они задаются как '-' или '+', за которыми сразу следует значение.

Таким образом, все следующие командные строки эквивалентны и передадут uniq флаги -c и -f:

**% uniq -c -f**

**% uniq -f -c**

**% uniq -fc**

И каждая из следующих командных строк передаст три флага -o3, -n и -4 в pr:

**% pr -o3 -4 -n file1**

**% pr -4 -nc 3 file1**

**% pr -n4 -c 3 file1**

Короче говоря, если вы укажете флаги так, чтобы их можно было понять, у утилиты тоже не должно возникнуть проблем.

Обычно, если вы даете один и тот же флаг более одного раза, запоминается только последнее вхождение флага, и повторение принимается без комментариев. Однако иногда флагу явно разрешается появляться несколько раз, и каждое событие запоминается. Такие флаги, как говорят, "складываются" и используются для указания набора параметров программы, вместо одного.

Другим специальным флагом является строка "--", которая используется в качестве признака конца флага. Как только она встречается, никакие дополнительные аргументы не интерпретируются как флаги. Таким образом, строка, которая обычно считывается как флаг, поскольку она начинается с '-' или '+', может быть передана утилите в качестве обычного аргумента, предшествуя ей аргументом "--". Строка "-" также приводит к завершению обработки флага везде, где она встречается, но, в отличие от "--", передается утилите вместо того, чтобы быть "израсходованной", по причинам, объясненным ниже.

Если вы даете утилите неизвестный флаг, она обычно отображает подсказку, напоминающую вам о том, какие флаги являются правильными. В этом сообщении кратко описывается формат командной строки, ожидаемый утилитой, и объясняется ниже в разделе синтаксис страницы псевдо-руководства. Если вы забыли, какие флаги принимает утилита, вы можете заставить ее вывести эту "сводку по использованию", присвоив ей флаг "-help", который никогда не является допустимым аргументом флага. (Если утилита ожидает обязательный аргумент, вам придется сказать "-help -help", чтобы преодолеть аргумент.)

Наконец, имейте в виду, что некоторые **комбинации** флагов для данной утилиты могут быть недопустимыми. Если утилите не понравится предоставленный вами набор, она выведет сообщение, напоминающее вам о допустимых комбинациях.

**Файлы**

Любая утилита, принимающая ряд имен входных файлов в командной строке, также будет читать свой стандартный ввод, STDIN, когда имена файлов ввода не заданы или когда встречается специальное имя файла "-". Следовательно, sort можно заставить читать STDIN, просто набрав:

**% sort**

в то время как следующие элементы объединяют file1, затем STDIN, затем file2 и записывают их в STDOUT:

**% cat file1 - file2**

Естественно, что всякий раз, когда читается STDIN, он читается до конца файла, поэтому не следует передавать его дважды одной и той же программе.

**Страницы руководства**

В оставшейся части этого документа рассматривается формат страниц руководства, описывающих каждую из утилит. Страницы руководства краткие, но полные и очень тщательно организованные. Из-за их общей скупости получение информации из них зависит от знания, где найти то, что вам нужно, и какую форму это может принять, когда вы это найдете. Страницы руководства разделены на несколько стандартных разделов, каждый из которых охватывает один аспект использования документированной утилиты. Итак, для ясности, остальная часть этого документа представлена как псевдо-справочная страница, с примечаниями к каждому разделу реальной страницы, появляющимся под обычным заголовком для этого раздела.

**НАЗВАНИЕ**

**название** - название и краткое описание утилиты

**СИНТАКСИС**

В этом разделе приводится краткое описание командной строки в одну строку, которую ожидает утилита. Краткое описание взято из сообщения с флагом -help, который вызовет его вывод большинство утилит, и указывает основные компоненты командной строки: само имя утилиты, флаги, которые принимает утилита, и любые другие аргументы, которые могут (или должны) появиться.

Флаги перечислены по имени внутри разделителей "-[" и "]". Обычно они появляются в алфавитном порядке; флаги, состоящие только из значения (см. выше), перечислены после остальных. Если флаг включает значение, то тип значения, которое он включает, также указывается одним из следующих кодов, указанных сразу после имени флага:

**Код Тип значения**

|  |  |
| --- | --- |
| \* | строка символов |
| # | целое число (размер слова) |
| ## | целое число (длинное) |
| ? | один символ |

Код '#' обозначает целое число, представимое в размере слова главного компьютера, что может ограничить его максимум 32,767 на некоторых машинах. Код "##" всегда обозначает длинное (четырехбайтовое) целое число, которое может представлять числа более двух миллиардов. Целое число интерпретируется как шестнадцатеричное, если оно начинается с "0x" или "0X", в противном случае как восьмеричное, если оно начинается с '0', в противном случае как десятичное. Всегда допускается ведущий знак '+' или '-'.

Если флаг может быть задан более одного раза (и содержит его значения), то за кодом значения следует символ '^'.

Таким образом, синтаксис pr:

**pr -[c# e# h l# m n s? t\* w# +## ##] <files>**

указывает, что pr принимает одиннадцать различных флагов, из которых -c, -e, -l и -w включают целочисленные значения размера слова, -h, -m и -n не включают вообще никаких значений, -t включает последовательность символов, -s включает один символ, а два флага +## и -## являются безымянными, состоящими только из длинного целого числа.

Обратите внимание, что флаги, введенные "-", отображаются без "-". Примерно такая же нотация используется в других разделах страниц руководства для обозначения флагов. На странице руководства pr, например, -c# будет ссылаться на флаг, указанный выше, как c#, в то время как -[c# e# w#] будет ссылаться на флаги "c# e# w#".

Положение и значение аргументов, не являющихся флагами, указываются "мета-понятиями", то есть словами, заключенными в '<' и '>' (например, <files> в приведенном выше примере). Каждое мета-понятие представляет собой ноль или более аргументов, фактически передаваемых в командной строке. При вводе командной строки вы вводите все, что представляет мета-понятие, и вводите его в этой точке строки. В этом примере вы должны ввести ноль или более имен файлов в точке, обозначенной <files>.

В этом разделе не предпринимается никаких попыток объяснить семантику командной строки --, например, комбинации аргументов, которые являются незаконными. Этому служит следующий раздел.

**ОПИСАНИЕ**

Этот раздел обычно содержит три части: обзор работы утилиты, описание каждого из ее флагов, затем (при необходимости) дополнительную информацию о том, как взаимодействуют различные флаги или другие аргументы, или о деталях работы утилиты, которые влияют на то, как ее использовать.

Обычно вводный обзор бывает кратким и резюмирует только то, что делает утилита и как она использует свои аргументы, не являющиеся флагами. Следующие ниже описания флагов состоят из отдельного предложения или большего количества информации о каждом флаге, представленной тем же именем флага и кодом значения, указанными в синтаксисе. В каждом описании указывается эффект, который имеет флаг, если он задан, и использование его значения, если оно есть. Параметры, указанные значениями флага, обычно имеют настройки по умолчанию, которые используются, когда флаг не установлен. Такие значения по умолчанию указаны в конце описания. Флаги перечислены в том же порядке, что и в строке синтаксиса.

Наконец, за описаниями флагов может следовать один или несколько абзацев, чтобы объяснить, какие комбинации флагов запрещены или как некоторые флаги влияют на другие. Любая дополнительная информация, представляющая интерес для обычного пользователя, также приведена здесь.

**КОД ЗАВЕРШЕНИЯ**

По завершении выполнения каждая утилита возвращает одно из двух значений, называемых успехом или неудачей. В этом разделе указано, при каких условиях утилита будет возвращать одно, а не другое. Успешный возврат обычно означает, что утилита смогла выполнить всю необходимую обработку файлов, а также все другие операции, относящиеся к утилите. В любом случае возвращаемое значение гарантированно предсказуемо. В этом разделе дается подробное описание каждой утилиты.

Обратите внимание, что возвращаемое значение часто не представляет интереса - оно даже не может быть протестировано в некоторых системах. Но когда его можно протестировать, оно играет важную роль в управлении командными сценариями.

**ПРИМЕР**

Здесь приведены один или несколько примеров того, как можно использовать эту утилиту. Примеры стремятся показать использование утилиты в реальных приложениях, если возможно, в сочетании со связанными программами.

Как правило, каждый пример окружен пояснительным текстом и отделяется от текста пустой строкой и отступом. В каждом примере строки, которым предшествует приглашение (например, "% "), представляют вводимые пользователем данные. Другие строки - это ответ утилиты на показанный ввод.

**ФАЙЛЫ**

В этом разделе перечислены все внешние файлы, которые требуются утилите для правильной работы. Чаще всего это общесистемные таблицы информации или общие имена устройств.

**СМ. ТАКЖЕ**

Здесь перечислены названия связанных утилит или руководств, которые следует изучить, если текущая утилита не выполняет то, что вы хотите, или если ее работа остается для вас неясной. Другая утилита может подойти ближе, и рассмотрение тех же проблем в разных терминах может помочь вам понять, что происходит.

Другие документы в том же разделе руководства, что и текущая страница, просто упоминаются по заголовку. Документы в другом разделе того же руководства упоминаются по названию и номеру раздела.

**ОШИБКИ**

В этом разделе описаны несоответствия или недостатки в поведении утилиты. Чаще всего это отклонения от соглашений, описанных на этой странице руководства, которые всегда будут здесь упоминаться. Здесь также будут указаны известные опасности, связанные с неправильным использованием утилиты.

**ОШИБКИ**

Есть тонкая грань между краткостью и загадочностью.

**НАЗВАНИЕ**

**ROM** - Написание кода только для чтения

**ОПИСАНИЕ**

Компилятор C никогда не генерирует самомодифицирующийся код. Если все инструкции вводятся в текстовую секцию во время генерации кода (как по умолчанию), то составной текстовый раздел, созданный программным компоновщиком, можно безопасно сделать только для чтения. Это означает, что программа, запускаемая в операционной системе, может иметь свой текстовый раздел, защищенный от записи каждый раз, когда он динамически загружается.

Это также означает, что текстовая часть автономной программы может быть "записана" в ПЗУ (постоянное запоминающее устройство) по любой из множества причин, которые люди находят для этого. Однако у автономной программы обычно есть и другие проблемы, с которыми нужно бороться.

Каждая программа имеет раздел данных в дополнение к текстовому разделу. Он содержит все статические и внешние объявления с их начальными значениями. C не дает возможности отличить параметры и таблицы, которые никогда не меняются во время выполнения программы, от переменных с важными начальными значениями, от переменных, которые не нужно инициализировать. Поэтому самый безопасный способ сделать это - предположить, что весь раздел данных состоит из переменных с важными начальными значениями - и, следовательно, исходное изображение раздела данных должно быть скопировано из некоторого места в ПЗУ по ожидаемым адресам в ОЗУ (Оперативная память с возможностью записи) при запуске программы.

Это не главное бремя. При запуске программы у отдельно стоящей программы на C должен быть установлен свой стек - для авто, аргументов и связей между вызовами функций. Просто немного больше работы, чтобы скопировать исходное изображение данных. Универсальная утилита link легко получает указание переместить раздел данных в его возможное место назначения в ОЗУ, а утилита hex может разместить его после текстового раздела в ПЗУ. Утилита link также может быть проинструктирована определять символы в конце текстового раздела и конце раздела данных в своем выходном файле. Таким образом, процедура инициализации для копирования данных из ПЗУ в ОЗУ может просто пройти через память от символа "end of text" до символа "end of data".

Есть способы уменьшить количество данных, которые должны быть инициализированы таким образом, а иногда и вовсе не инициализировать. Во-первых, это литералы - символьные строки, таблицы переключателей и константы, - которые компилятор генерирует самостоятельно. На машине, которая поддерживает отдельные пространства инструкций и данных, литералы по умолчанию направляются в раздел данных. Указание флага -x6 для генератора кода побуждает его помещать литералы в текстовый раздел. Редко, когда система на основе ПЗУ также использует отдельные пространства команд и данных.

Точно так же обширные таблицы, о которых известно, что они доступны только для чтения, могут быть собраны в один исходный файл C и скомпилированы с флагом -x7 для генератора кода, тем самым помещая все объявления данных для этого файла в текстовый раздел.

Если остальные переменные могут быть созданы так, чтобы не требовать инициализации при запуске, или требовать только простую инициализацию, такую как все нули, то секция данных может быть просто отброшена во время записи ПЗУ. Этот подход, естественно, требует, чтобы весь исходный код был доступен для изучения.

Один из случаев, когда это невозможно, - это когда библиотечные функции, доступные только в двоичном формате, включены в продукт на основе ПЗУ или для совместного использования. Библиотеки, предоставляемые Whitesmiths, Ltd., не содержат никаких необоснованных препятствий для совместного использования. Например, такая функция, как exp, содержит только таблицы в разделе данных - ее можно безопасно передать через компоновщик, чтобы ее данные были объединены в текстовый раздел.

Однако некоторые функции по необходимости содержат статические переменные с важными начальными значениями. Обычно эти функции взаимодействуют со службами операционной системы и, следовательно, не встречаются в программах на основе ПЗУ. Примерами являются onexit/exit, которые запоминают заголовок списка функций, вызываемых при выходе из программы, и sbreak, которая может содержать текущий конец области динамических данных. Другие функции со статическими переменными могут найти применение в автономных программах - примеры: alloc/free и \_when/\_raise.

Поэтому лучше всего использовать наиболее общую инициализацию запуска, описанную выше. Часто бывает слишком сложно определить, какие библиотечные функции безопасны, и еще сложнее запомнить специальные правила для обеспечения безопасности нового кода.

**НАЗВАНИЕ**

**anat** - транслятор A-Natural

**СИНТАКСИС**

**anat -[i o\* s\*] <files>**

**ОПИСАНИЕ**

Утилита **anat** переводит повествовательный ассемблер A-Natural для Intel 8080 на стандартный язык ассемблера, приемлемый для ассемблера ISIS-II asm80 или Microsoft Macro-80. Поскольку вывод генератора кода 8080, p2.80, является A-Natural, anat требуется для взаимодействия компилятора C с существующими средствами разработки программного обеспечения 8080.

Флаги:

|  |  |
| --- | --- |
| **-i** | Выдавать код в формате ISIS-II. Идентификаторы каждого "." изменены на "@", а каждый "\_" изменен на "?". По умолчанию используется Microsoft, и в этом случае каждый "." изменяется на "$", а каждый "\_" изменяется на ".". |
| **-o\*** | Имя файла можно указать с помощью операнда -o, чтобы выходные данные записывались в указанный файл, а сообщение об ошибке записывалось в STDOUT. По умолчанию для вывода используется STDOUT, а для сообщения об ошибке - STDERR. |
| **-s\*** | Выдать оператор end. С помощью начального операнда задает начальный адрес. По умолчанию в операторе end не указан начальный адрес. |

Если <files> присутствуют, они объединяются по порядку и используются в качестве входного файла вместо стандартного STDIN.

Если -o отсутствует, присутствует один или несколько файлов и первое расширение имени файла начинается с "8", тогда утилита ведет себя, как если -o было указано с использованием первого имени файла, за исключением того, что в расширение "8" заменена на "m". Таким образом,

**anat file.8**

то же самое, что

**anat -o file.m file.8**

Если присутствует -o, генерируется директива name (title для Microsoft) с использованием имени выходного файла в качестве имени модуля. Любые ".suffix" в имени удаляются. Если нет -o, но присутствуют <files>, в директиве name используется первое имя входного файла. Если ни один из них не предустановлен, директива name не генерируется.

Генерация кода начинается с cseg и возвращается к cseg в конце до того, как будут созданы какие-либо литералы. Любые символы, не определенные в конце входного файла, публикуются с помощью операторов extrn в конце.

**ОШИБКИ**

Начальная строка, используемая с -s, не имеет "." и "\_", сопоставленных в качестве идентификаторов.

**НАЗВАНИЕ**

**as.80** - ассемблер A-Natural для 8080

**СИНТАКСИС**

**as.80 -[o\* x] <files>**

**ОПИСАНИЕ**

**as.80** ассемблирует повествовательный ассемблер A-Natural для Intel 8080 в перемещаемые изображения объектов стандартного формата. Поскольку вывод p2.80 генератора кода 8080 является A-Natural, as.80 требуется для создания перемещаемых изображений, подходящих для связывания с помощью link.

Флаги:

|  |  |
| --- | --- |
| **-o\*** | Записать выходные данные в файл \*. Значение по умолчанию - xeq. При некоторых обстоятельствах вместо этого параметра может быть введено имя файла, как описано ниже. |
| **-x** | Поместить в изображение объекта только те символы, которые не определены или которые должны стать известными глобально. В настоящее время это все равно происходит. Флаг присутствует для совместимости с другими ассемблерами. |

Если <files> присутствуют, они объединяются по порядку и используются в качестве входного файла вместо стандартного STDIN.

Если -o отсутствует, и один или несколько файлов присутствуют, и расширение у первого имени файла начинается с 's', тогда as.80 ведет себя так, как если бы -o был указан с использованием первого имени файла, только с завершающей 's' измененной на 'o'. Таким образом,

**as.80 file.s**

то же самое, что

**as.80 -o file.o file.s**

Изображение перемещаемого объекта состоит из заголовка из восьми слов, за которым следует сегмент text, сегмент data, таблица символов и информация о перемещении. Заголовок состоит из значения 0x1499, количества байтов таблицы символов, количества байтов объектного кода, определенного сегментом text, количества байтов, определенных сегментом data, трех нулевых слов и смещения сегмента data, которое всегда равно размеру сегмента text. Все слова в изображении объекта сначала записываются в младший байт. Сегмент text перемещается относительно нулевого местоположения, сегмент data перемещается относительно конца сегмента text.

Информация о перемещении состоит из двух байтовых потоков, один для сегмента text и один для сегмента data, каждый из которых заканчивается нулевым контрольным байтом. Управляющие байты в диапазоне [1, 31] вызывают пропуск такого количества байтов в соответствующем сегменте. Байты в диапазоне [32, 63] пропускают 32 байта плюс 256 байт управления минус 32 плюс количество байтов, указанное следующим байтом перемещения.

Все остальные байты управления управляют перемещением следующего слова в соответствующем сегменте. 1-взвешенный бит и 2-взвешенный бит всегда равны нулю для 8080. Код символа - это управляющий байт, сдвинутый вправо на два места, минус 16.

Код символа 47 заменяется кодом, полученным из байта или байтов, следующих в потоке перемещения. Если следующий байт меньше 128, то код символа равен его значению плюс 47; в противном случае код будет таким: этот байт минус 128, умноженный на 256, плюс 175 плюс значение следующего байта перемещения после этого.

Нулевой символьный код не требует дальнейшего перемещения; 1 означает, что к слову нужно добавить изменение смещения текста; 2 означает, что необходимо добавить изменение смещения данных; 3 не используется. Другие коды символов требуют добавления к слову значения записи таблицы символов, индексированной кодом символа минус 4.

Каждая запись таблицы символов состоит из слова значения, байта флага и девяти-байтового имени, дополненного конечными NUL. Значимые значения флага: 0 для неопределенных, 4 для определенных абсолютных, 5 для определенных текстовых перемещаемых и 6 для определенных данных перемещаемых. К этому добавляется 010, если символ должен быть известен глобально.

**СМ. ТАКЖЕ**

link, p2.80, rel

**НАЗВАНИЕ**

**c** - драйвер многопроходной команды

**СИНТАКСИС**

**c -[f\* o\* p\* v +\*] <files>**

**ОПИСАНИЕ**

Драйвер **c** разработан для облегчения многопроходных операций, таких как компиляция, ассемблирование и связывание исходных файлов, путем расширения команд из сценария-прототипа для каждого из его аргументов файла. Лучше всего это описать на примере. Скрипт для компиляции Pascal на PDP-11 может выглядеть так:

p:/etc/bin/ptc ptc

c:/etc/bin/pp pp -x -i/lib/

:/odd/p1 p1 -cem

:/odd/p2.11 p2

s:/etc/bin/as as

o::/etc/bln/link link –lp.11 –lc.11 /lib/Crts.o

Драйвер c предназначен для установки под несколькими псевдонимами. Каждый псевдоним обычно подразумевает имя файла ".proto", содержащего соответствующий скрипт, который нужно искать так же, как оболочка ищет файл, названный командой. То есть, если псевдоним содержит '/', то выполняется поиск только по названному каталогу, в противном случае поиск выполняется по каждому каталогу в текущем пути выполнения. Если бы приведенный выше сценарий назывался pc.proto, его можно было бы неявно вызвать, запустив с под псевдонимом pc. Флаг -f может использоваться для переопределения этой процедуры поиска.

По соглашению псевдоним c является синонимом для драйвера компилятора C на основной машине, а pc - для драйвера Pascal на хосте.

Учитывая приведенный выше сценарий, c создаст файл ".o" для любых файлов аргументов, заканчивающихся на ".p", выполнив следующий список команд:

**Команда Аргументы**

/etc/bin/ptc ptc –o T1 file.p

/etc/bin/pp pp –o T2 –x –i/lib/ T1

/odd/p1 p1 –o T1 –cem T2

/odd/p2.11 p2 –o T2 T1

/etc/bin/as as –o file.o T2

где file.p - файл аргументов, а T1 и T2 - временные промежуточные элементы.

Файлы, имена которых заканчиваются на ".c", будут обрабатываться, начиная со строки сценария с надписью "o:"; аналогично, файлы ".s" будут обрабатываться, начиная со строки с надписью ".s". Как правило, все созданные файлы ".o", а также любые файлы, суффикс которых не совпадает с префиксом строки, собираются и передаются программе link для окончательной привязки. Файл передается программе link только в том случае, если его имя еще не указано в векторе аргументов строки link. Любая ошибка, о которой сообщает любая из программ обработки, завершает обработку текущего файла. Если какие-либо файлы завершены, программа link не запускается.

Флаги:

|  |  |
| --- | --- |
| **-f\*** | Взять ввод прототипа из файла \* вместо файла ".proto", подразумеваемого псевдонимом командной строки. Если STDIN, читается. |
| **-o\*** | Поместить вывод программы строки link в файл \* (добавив -o\* перед списком аргументов). Добавление происходит только в том случае, если установлен этот флаг. Естественно, его следует указывать только в том случае, если программа принимает спецификацию -o. |
| **-p\*** | Префикс имен всех постоянных выходных файлов с именем пути \*. Если указан флаг -o, то выходное имя файла строки link также будет иметь префикс \*, если оно еще не содержит '/'. Все остальные выходные файлы будут удалены из любого ранее существовавшего пути перед добавлением префикса. |
| **-v** | Перед выполнением каждой команды вывести ее аргументы в STDOUT. По умолчанию выводится только имя каждого обработанного файла и имя компоновщика, когда (и если) он запускается, за каждым именем следует двоеточие и новая строка. |
| **+\*** | Остановить производство на линии прототипа с префиксом \*. Данная строка сопоставляется с полями префиксов каждой строки прототипа, и выполнение команд прототипа останавливается после строки, предшествующей соответствующей. В приведенном выше примере +c приведет к преобразованию файлов ".p" в файлы ".c" без последующей обработки, +s обработает файлы ".p" или ".c" в ".s", а +o запретит связывание. Любой файл, который не обрабатывается, вызывает сообщение об ошибке. |

Каждая строка в файле прототипа состоит до четырех частей: строка префикса (возможно, null), оканчивающаяся двоеточием, путь, определяющий программу, которую нужно запустить, и одна или две группы до 16 строк, группы разделены двоеточием. Вектор аргументов создается для каждой программы из первой строки в первой группе, спецификации -o, оставшихся строк в первой группе и одного из файлов аргументов. Вторая группа строк, если присутствует, добавляется к вектору после файла аргументов. Каждая запись в командной строке отделяется произвольным пробелом. Последняя строка в файле может содержать только поле префикса (и двоеточие), и в этом случае она определяет только суффикс выходного файла для предыдущей строки. Последняя строка может также иметь поле префикса, оканчивающееся двойным двоеточием, определяя его как "строку link" со специальными характеристиками, указанными выше.

Обратите внимание, что строка прототипа с пустым полем префикса не может быть указана пользователем в качестве точки входа или выхода.

Если последняя строка прототипа не является только префиксом и не является строкой link, то ее вывод записывается во временный файл, который удаляется.

**КОД ЗАВЕРШЕНИЯ**

**c** возвращает успех, если ему удалось передать все свои файлы командной строки хотя бы одной из программ-прототипов (включая программу строки link), и если все контролируемые им исполнения вернули успех.

**ПРИМЕР**

Файл прототипа, предназначенный только для проверки синтаксиса языка Си, без генерации кода, может быть:

**c:/etc/bin/pp pp -x -i/lib/**

**:/odd/p1 p1**

Этот прототип не будет создавать выходных файлов, так как вывод p1 будет отправлен во временный файл.

**ОШИБКИ**

Нужен какой - то способ указать флаги для программ-прототипов.

Эта утилита в настоящее время предназначена для использования только в хост-системах Idris/UNIX.

**НАЗВАНИЕ**

**cpm** - обслуживание дискет CP/M

**СИНТАКСИС**

**cpm -[+cpm dec b c d p r t v x f\*] <files>**

**ОПИСАНИЕ**

Утилита **cpm** читает и записывает файловые системы, совместимые с CP/M, записанные на стандартные восьмидюймовые IBM-совместимые дискеты с программным разделением. Такие файловые системы управляют дискетой, разделяя ее на 243 кластера по 1024 байта. Можно выделить до 16 кластеров. В справочник можно ввести до 64 экстентов. Файл состоит из одного или нескольких экстентов, пронумерованных последовательно с нуля. Все экстенты, принадлежащие одному файлу, имеют одинаковое восьмизначное имя и трехсимвольный тип. Имена и типы лучше всего ограничивать буквами одного регистра и цифрами для максимальной переносимости.

Флаги:

|  |  |
| --- | --- |
| **+cpm** | Свернуть сектора на каждой дорожке по рецепту:  **sec = (sec \* 6 + (13 <= sec)) % 26;**  где sec находится в интервале [0, 26], т.е. на единицу меньше, чем номер аппаратного сектора. Эта опция необходима на тех машинах, которые записывают сектора дискет без свертки, т.е. начиная с сектора 0 дорожки 1 для поиска адресных 0L и последовательно принимая последующие сектора. |
| **-dec** | Разделить сектора, чтобы исправить свертку DEC. Рецепт DEC следующий:  **sec = ((sec << 1) + (13 <= sec) + trk \* 6) % 26;**  **trk =+ 1;**  где sec такое же, как указано выше, а trk находится в интервале [0, 77]. Эта опция необходима в таких системах DEC, как RT-11 и RSX-11M. Преобразование CP/M также происходит, как указано выше. |
| **-b** | Обрабатывать все копируемые файлы как двоичные изображения. По умолчанию используется текстовый режим с возможной обработкой возврата каретки, нулей, кодов Control-Z и т.д. |
| **-c** | Создать новую дискету с пустым каталогом, затем скопировать именованные файлы на дискету. |
| **-d** | Удалить именованные файлы с дискеты. |
| **-f\*** | Доступ к дискете осуществляется в виде имени файла \*. По умолчанию используется "dx0:", если присутствует флаг -dec, в противном случае "/dev/rx0". |
| **-p** | Распечатать файлы, т.е. скопировать их с дискеты в STDOUT. Если файлы не указаны, печатаются все файлы. |
| **-r** | Заменить названные файлы на дискете |
| **-t** | Составить таблицу каталога дискеты, показывая все выделенные экстенты в алфавитном порядке. |
| **-v** | Подробно описывать операции, называя файлы удаленным "d:", добавленным "a:" или извлеченным "x:". Вывести количество свободных экстентов и кластеров, если указано -t. По умолчанию это достаточно тихая работа, за исключением ошибок. |
| **-x** | Извлечь файлы, т.е. скопировать их с дискеты в файлы с тем же именем. Если файлы не указаны, извлекаются все файлы. |

Может появиться не более одного из флагов -a, -d, -p, -r, -t, -x, по умолчанию -t.

**КОД ЗАВЕРШЕНИЯ**

Утилита **cpm** возвращает успех, если структура каталогов на дискете не повреждена и передача файлов не завершилась неудачно, иначе - сбой.

**ПРИМЕР**

Чтобы создать дискету и проверить ее в операционной системе DEC:

**>cpm -dec -o cscrip.cmd lscrip.cmd**

**>cpm -dec -rb prog.com**

**>cpm -dec**

**>cpm -dec -p scrip.cmd**

**НАЗВАНИЕ**

**db** - редактор/отладчик двоичных файлов

**СИНТАКСИС**

**db -[a c n p u] <file>**

**ОПИСАНИЕ**

Программа **db** - интерактивный редактор, предназначенный для работы с двоичными файлами. Работает в одном из трёх режимов, 1) в качестве двоичного файлового редактора, для проверки и исправления произвольных файлов прямого доступа, таких как образы диска или кодированные файлы данных; 2) в качестве "подготовительного" отладчика, для проверки и исправления исполняемых или перемещаемых файлов, и 3) в качестве "посмертного" отладчика для проверки основных образов, созданных системой, возможно, с использованием информации из исходного исполняемого файла. В двух последних случаях db распознает структуру объектных, исполняемых и core файлов и выполняет специальную интерпретацию адресов для моделирования размещения кода и данных во время выполнения.

Программа db содержит дизассемблер, включенный путем указания режима m в командах o, p или l (также используется командами d и s и в контекстном поиске). Дизассемблер интерпретирует файл как машинные инструкции и переводит их в обычную символьную нотацию, используя таблицу символов и информацию о перемещении, если таковая имеется, по адресам вывода. Этот "машинный режим" является исходным по умолчанию, используемым для файлов, не отмеченных как абсолютные.

Для каждой поддерживаемой целевой машины в db встроен свой дизассемблер. Таким образом, могут быть варианты с такими именами, как db11, db80 и т.д. По соглашению отладчик для хост-машины называется db.

Флаги:

|  |  |
| --- | --- |
| **-a** | Считать <file> абсолютным двоичным файлом. |
| **-e** | Использовать файл core вместе с <file> и работать в режиме отладки, core будет исходным текущим файлом. |
| **-n** | Подавить вывод номера текущего байта, который обычно предшествует каждому отображаемому элементу. |
| **-p** | Перед чтением ввода команды вывести в STDOUT символ подсказки '>'. По умолчанию приглашение не выводится. |
| **-u** | Открыть <file> в режиме обновления. По умолчанию <file> открыт только для чтения. |

Может быть указано не более одной из -a и -c. Любой файл изображения ядра всегда должен иметь имя core. Если <file> не указан, предполагается xeq.

Программа db имеет много общего с текстовым редактором e. Она работает с текущим двоичным файлом, соответствующим образом ядра или с обоими, считывая из STDIN серию односимвольных команд и записывая любые выходные данные в STDOUT. Точно так же, как командам в e могут предшествовать адреса строк, командам db могут предшествовать адреса байтов, синтаксис которых является надмножеством синтаксиса адресов e. Аналогичные команды в двух редакторах имеют одинаковые имена. Поведение по умолчанию также аналогично.

Программа db не накладывает никакой структуры на данные, содержащиеся во входных файлах, за исключением того, что подразумевается текущим режимом ввода/вывода, который определяет длину "элементов", с которыми работают многие команды. Вне машинного режима элементом является один, два или четыре байта файла. В машинном режиме элемент - это инструкция, разобранная, начиная с текущего местоположения, и имеет разную длину. Порядок байтов для многобайтовых элементов определяется целевой машиной, как и другие машинно-зависимые характеристики, такие как структура таблицы символов и размер int. Таким образом, исследование файла, созданного для какой-либо машины, отличной от целевой, не рекомендуется. Если <file> нельзя интерпретировать как объектный файл в стандартном формате для целевой машины, он рассматривается как абсолютный. Если основной файл не начинается с управляющих байтов, подходящих для цели, db прерывает работу.

**Интерпретация адреса и синтаксис**

В абсолютном файле байтовый адрес - это просто расположение байта относительно начала файла (первый байт имеет нулевой адрес). В объектных файлах или файлах ядра адрес состоит из двух компонентов: места в памяти, которое байт занял бы, если бы текущий файл был фактически загружен, и флага, указывающего, находится ли это место в сегменте text или в сегменте data/bss программы. (Сегмент bss программы следует сразу же за сегментом data.) Диапазоны адресов в двух сегментах могут перекрываться. Адрес, который может относиться к любому пространству, устраняется, как описано ниже.

Байтовый адрес представляет собой последовательность из одного или нескольких следующих терминов:

Символ обращается к «текущему байту» (или "точке").

Как правило, точка является первым байтом последнего элемента, на который влияет последняя команда (см. описание команды ниже). Точка также относится к тому же сегменту, что и этот элемент.

Символ адресует последний байт сегмента, на который в данный момент ссылается точка. Для абсолютного файла не имеет значения. Запрос на изучение или изменение данного адреса приведет к поиску этого байта файла. Запрос выполняется успешно, если это делает последующий вызов ввода-вывода.

В файле объекта символы 'T', 'D' и 'B' представляют собой константы, которые имеют значение первого адреса в сегментах text, data и bss соответственно, а также имеют соответствующий компонент сегмента. В файле core 'T' и 'D' имеют одинаковое значение, в то время как 'B' устанавливается на адрес верхней части стека во время дампа, как записано на панели файла ядра. Все три имеют нулевое значение, если текущий файл является абсолютным.

Целое число n обращается к n-му байту абсолютного файла, где байты нумеруются от нуля. В файле объекта или ядра целое число n обращается к n-му байту "памяти", а сегмент, заданный самым последним термином, имеет явный компонент сегмента (или сегмент точки, если такого термина не существует). Наиболее простым способом обращения к n-му байту одного из двух сегментов является добавление целого числа n к одной из констант 'T', 'D' или 'B'. Целое число интерпретируется как восьмеричное, если оно начинается с "0", как шестнадцатеричное, если оно начинается с "0x" или "0X", и как десятичное в противном случае.

"'x", за которой следует любая буква нижнего регистра, обращается к байту, которому ранее была назначена эта буква, как описано в команде k ниже.

Регулярное выражение, заключенное в '/', вызывает контекстный поиск в текущем сегменте (или всем файле, если он абсолютный), начиная с элемента после текущего байта и продвигаясь к концу текущего сегмента (или файла). Синтаксис для регулярных выражений такой же, как и для e, то есть реализованный подпрограммами переносимой библиотеки C amatch, match и pat. Каждый элемент в искомой области преобразуется в текущий выходной формат и сравнивается с заданной строкой поиска. Поиск останавливается на первом совпадающем элементе, значение термина является адресом этого элемента. Если поиск не удался, выдается сообщение об ошибке.

Регулярное выражение, заключенное в '%' или '?' вызывает обратный поиск. Контекстный поиск начинается с элемента перед текущим байтом и продолжается в направлении начала текущего сегмента (или файла, если он абсолютный), как описано выше. Обратный поиск невозможен, если текущий режим вывода - машинный.

Строка, начинающаяся с '\_' или '"', считается именем символа, заканчивается пробелом или одним из операторов, приведенных ниже. Начальный '\_' считается частью имени. Начальный '"' вводит имена, не начинающиеся с '\_', и отбрасывается. Значение термина - это адрес символа. Сегмент, на который ссылается термин, - это тот сегмент, в котором определяется символ.

За термином может следовать произвольное число символов '\*', каждый из которых приводит к тому, что значение, вычисленное слева от него, обрабатывается как указатель на сегмент данных. Содержимое указанного места с размером int становится значением термина. Добавление термина с помощью '>' приводит к тому, что он аналогичным образом обрабатывается как указатель на текстовый сегмент.

Адрес определяется путем сканирования терминов слева направо. Два термина, разделенные плюсом '+' (или минусом '-'), разрешаются термином, представляющим собой сумму (или разницу) значений двух исходных терминов. Суммируются два последовательных члена, не разделенных оператором. Если термин явно относится к одному из двух сегментов, то этот сегмент становится сегментом, к которому относится весь адрес.

Если адрес начинается с '+' или '-', то точка считается термином слева от '+' или '-'. Символ '^' в этом контексте эквивалентен '-'. Таким образом, "^3", стоящее отдельно, считается таким же, как ".-3".

Символ '+' ('-' или '^'), за которым не следует термин, принимается за значение +1 (-1). Таким образом, '-' само по себе означает ".-1", "++" означает ".+2", "6---" означает "3" и так далее.

Адрес может быть сколь угодно сложным, если он дает значение в пределах сегмента, на который он ссылается. (Опять же, для абсолютного файла будет принят любой адрес.) Перед командой может быть любое количество адресов, если последние один или два являются допустимыми для этой команды. Для команды, требующей двух адресов, ошибка заключается в том, что второй адрес меньше первого или адреса относятся к разным сегментам. В режиме отладки при указании на основной файл принимаются адреса стека.

В режиме машинной инструкции адреса должны быть указаны с осторожностью, db будет разбирать начиная с указанного адреса, но если адрес не является началом действительной инструкции, вывод не имеет смысла. Чтобы обеспечить допустимую разборку, начните с начала текстового раздела или с адреса символа, относящегося к тексту.

**Синтаксис команды**

Каждая команда db состоит из одного символа, которому необязательно может предшествовать один или несколько адресов, обычно указывающих включающий диапазон байтов в файле, на который должна воздействовать команда. Адреса разделяются запятой ',' или двоеточием ':'. Если используется двоеточие, то точка устанавливается в положение и сегмент, указанные в адресе слева от двоеточия, до того, как остальная часть команды будет интерпретирована.

За символом команды могут следовать дополнительные параметры. Несколько команд можно вводить в строке, разделяя их точкой с запятой. Каждая команда, для которой требуются адреса, имеет значения по умолчанию, поэтому адреса часто можно не указывать. Программа db сообщает о несуществующем адресе.

В приведенных ниже описаниях команд адреса в скобках являются байтовыми адресами по умолчанию, которые будут использоваться для команды, если они не введены. Круглые скобки нельзя вводить вместе с командой, они используются в качестве синтаксической записи. Если команда описана без круглых скобок, адреса не требуются. Для двухадресной команды предоставление только одного адреса по умолчанию приведет к тому, что второй будет идентичен указанному. Это ошибка - предварять команды адресов, которые не требуют каких-либо действий.

Команды:

|  |  |  |
| --- | --- | --- |
| **(fp)a** | - | Отобразить автоматический фрейма стека. Допустимо, только если core является текущим файлом, db сохраняет указатель fp на текущий кадр стека, инициализированный указателем, указанным панелью, выгруженной в файл core. "a" может предшествовать адрес, который будет интерпретирован (и запомнен) как адрес автоматического фрейма стека. |
| **b(1)** | - | Резервное копирование одного или нескольких кадров стека. Действителен только в том случае, если ядром является текущий файл, за «b» может следовать ноль, который повторно инициализирует текущий указатель кадра стека fp на тот, который находится в панели, или отсчет количества кадров, которые должны быть зарезервированы. После "b" и этого счетчика также может следовать "a", который будет резервировать один кадр стека и отображать значения кадров автоматического стека. |

|  |  |  |
| --- | --- | --- |
| **c** | - | Отобразить панель, состоящую из сохраненных регистров и причины смерти. Регистры отображаются в шестнадцатеричном формате и символически, если они значимы. Допустимо только в том случае, если core является текущим файлом. |
| **(.,.)d** | - | Печатать элементы, которые различаются между исполняемыми и core файлами. |
| **e file** | - | Ввести новый изменяемый файл данных в том же режиме, что и раньше, затем запомнить имя файла. Работает только с флагом -a. |
| **f** | - | Отобразить имя изменяемого файла данных. |
| **(.)kx** | - | Отметить адрес, запомнив его как x, где x - любая строчная буква, пригодная для последующего использования в качестве указателя с \*x. |
| **(.,.)l<mode>** | - | Посмотреть по указанному адресу, т.е. отобразите элемент в текстовом представлении вывода, но не изменяя указанный адрес. Если присутствует <mode>, он изменяет режим, указанный в последней команде 'o'. |
| **(.)n** | - | Отобразить адрес как символ плюс смещение. |
| **o<mode>** | - | Изменить представление выходного текста на <mode>, где <mode> - это 'm', указывающее на разборку машинных инструкций или любую из комбинаций [a|h|o|u] [c|s|i|l] для ASCII, шестнадцатеричного, восьмеричного или беззнакового отображения целых чисел типа char, short, int или long. Если <mode> не указан, для режима устанавливается значение по умолчанию, используемое при вызове db: 'm', если флаг -a не был задан, и без знака short в противном случае. |
| **(.,.)p<mode>** | - | Напечатать элемент в текстовом представлении вывода, затем изменить его, указав последний указанный адрес. Если присутствует <mode>, он изменяет режим, указанный в последней команде 'o'. |
| **q** | - | Выход |

|  |  |  |
| --- | --- | --- |
| **(.)r file** | - | Считать двоичное содержимое указанного файла и заменить целевой файл с текущего байта, расширяя файл по мере необходимости. Это допустимо только с флагом -a, файл не должен быть <file> редактируемый db. |
| **(.,.)s/pattern/new/[g]** | - | Для всех элементов в диапазоне разверните, чтобы вывести текстовое представление, отредактируйте текст, изменив экземпляры шаблона на новый, затем обновите элемент, используя отредактированный текстовый шаблон. Если указан суффикс g, изменяются все совпадения в элементе, в противном случае изменяется только крайний левый. |
| **(.)u** | - | Войти в режим обновления, т. е. рассматривайте каждую последующую строку ввода как выходное текстовое представление элемента и замените адресованный элемент. Последовательные элементы заменяются, расширяя файл по мере необходимости, пока не появится строка ввода, содержащая только '.'. Это не работает в режиме 'm'. Ввод должен быть введен в текущем режиме, по одному элементу в строке; в противном случае db будет жаловаться. |
| **(0,$)w file** | - | Записать указанный диапазон файла редактируемого db в файл, файл не должен совпадать с <file> редактируемого db. Эта команда допустима только с флагом -a. |
| **z file** | - | Указать db на файл. Если файл не указан, z сообщает, на какой файл, core или исполняемый, в данный момент указывает. |
| **(.)<newline>** | - | Эквивалентно вводу ". + size", где siz - это количество байтов, отображаемых в текущем режиме. Эту форму можно использовать для просмотра одной инструкции или элемента за раз. Если байтовый адрес предшествует <newline>, будет отображаться адресуемый элемент. Точка будет установлена на отображаемый байт. |

|  |  |  |
| --- | --- | --- |
| **!<command>** | - | Отправить все символы в строке справа от '!' к системе, чтобы интерпретироваться как системную команду. "!cd" обрабатывается db напрямую. Каждое вхождение последовательности "\f" будет заменено текущим запомненным файлом до того, как будет произведен переход на системный уровень. Эта команда не меняет точку. Доступно только в системах Idris/UNIX. |
| **=** | - | Отобразить байт числа точки. |

В системах Idris/UNIX при вводе ASCII DEL db прерывает текущую задачу, если она существует, и отображает '?'. После этого он будет готов снова принять команды db. Точка обычно плохо определена на данном этапе.

**СМ. ТАКЖЕ**

link, rel

**ОШИБКИ**

В настоящее время эта утилита предназначена только для использования в хост-системах Idris/UNIX.

**НАЗВАНИЕ**

**dis80** - перевод A-Natural перемещаемых файлов

**СИНТАКСИС**

**dis80 -[c i] <files>**

**ОПИСАНИЕ**

Утилита **dis80** позволяет переводить в символическую форму перемещаемые двоичные файлы, созданные A-Natural ассемблером as.80. Эта форма совместима с существующим ассемблерным кодом на 8080.

Флаги:

|  |  |
| --- | --- |
| **-c** | Перевести каждый файл в ассемблерную программу, подходящую для ввода в ассемблер Microsoft под CP/M. Каждое подчеркивание '\_' в идентификаторе изменяется на '.', а каждое '.' изменяется на '$'. |
| **-i** | Перевести каждый файл в ассемблерную программу, подходящую для ввода в ассемблер asm80 под ISIS-II. Каждое подчеркивание '\_' в идентификаторе изменяется на '?', а каждое '.' изменяется на '@'. |

Если флаги не заданы, по умолчанию используется -c.

<files> указывает один или несколько файлов, которые должны быть в A-Natural перемещаемом формате. Если указано несколько файлов, то выводу для каждого из них предшествует строка в форме

**file:**

Вывод ассемблера начинается директивой title (name для ISIS-II), где name выводится из имени файла путем удаления любых символов справа включая крайнюю левую '.', затем добавляется '@' в случае ISIS-II и '$' в случае Microsoft.

**КОД ЗАВЕРШЕНИЯ**

Утилита **dis80** возвращает успех, если диагностика не производится, т.е. если все чтения успешны и все форматы файлов допустимы. В противном случае возвращается неудача.

**ПРИМЕР**

Для преобразования chdr.o для использования с кодом ассемблера Microsoft:

% dis80 chdr.o >chdr.mac

**СМ. ТАКЖЕ**

as.80, link

**ОШИБКИ**

Символы "\_c\_" и "\_d\_" не должны использоваться в программах A-Natural для перевода с помощью dis80.

**НАЗВАНИЕ**

**hex** - перевод объектных файлов в формат ASCII

**СИНТАКСИС**

**hex -[db## dr h m\* r## s tb## +# #] <ifile>**

**ОПИСАНИЕ**

Утилита **hex** переводит исполняемые образы, созданные с помощью link, в стандартный шестнадцатеричный формат Intel или в формат объектного файла Motorola S-record. Исполняемый образ читается из <ifile>. Если <ifile> не задан или указано имя файла, читается xeq.

Флаги:

|  |  |
| --- | --- |
| **-db##** | При обработке стандартного объектного файла в качестве входных данных переопределить смещение данных объектного модуля с помощью ##. |
| **-dr** | Вывести текстовые записи в виде записи типа 0x81, записи данных в виде записи типа 0x82 для использования с утилитой Digital Research gencmd в CP/M-86. |
| **-h** | Не создавать начальную запись для шестнадцатеричных файлов Intel. |
| **-m\*** | Вставить строку \* вместо <ifile> в запись Motorola S0, созданную при задании -s. |
| **-r##** | Интерпретировать входной файл как "необработанный" двоичный файл, а не как объектный файл. Вывод производится, как описано ниже, в любом формате, за исключением того, что начальный адрес модуля задается длинным целым числом ##. |
| **-s** | Создать S-записи, а не шестнадцатеричный формат по умолчанию. |
| **-tb##** | При обработке стандартного объектного файла в качестве входных данных переопределить смещение текста объектного модуля с помощью ##. |
| **+#** | Начать вывод с #-го байта. # должно быть между 0 и на единицу меньше, чем значение, указанное флагом -#. -4 +3 производит байты 3, 7, 11, 15, ...; -2 +0 производит все четные байты; -2 +1 выводит все нечетные байты; 0 - значение по умолчанию, которое выводит все байты. |
| **-#** | Выводить каждый #-й байт. -2 выводит каждый второй байт, -4 каждый четвертый; 1 - значение по умолчанию, которое выводит все байты. |

Вывод записывается в STDOUT. Когда входной файл является стандартным объектным файлом, смещение загрузки первой выходной записи для ее текста или сегмента данных определяется следующим образом. Если для этого сегмента явно задано смещение (с помощью "-db##" или "-tb##"), то используется оно. В противном случае, если выводятся пороговые байты входного файла, соответствующее смещение используется из заголовка объектного файла. В противном случае используется нулевое смещение загрузки. Затем значение, указанное с помощью "+#", добавляется к выбранному смещению, чтобы получить первое фактически выводимое смещение.

**Шестнадцатеричные файлы**

Файл в шестнадцатеричном формате Intel состоит из следующих записей в указанном порядке:

1. Символ '$' в строке указывает на конец (несуществующей) таблицы символов. Если указан -h, эта строка опускается.
2. Записи данных для сегмента text, если таковые имеются. Они представляют 32 байта изображения на строку, возможно, заканчивающуюся более короткой строкой.
3. Записи данных для сегмента data, если таковые имеются, в том же формате, что и записи сегмента text.
4. Конечная запись с указанием начального адреса.

Каждая запись данных начинается с символа ':' и состоит из пар шестнадцатеричных цифр, каждая пара представляет собой числовое значение байта. Последняя пара - это контрольная сумма, так что числовая сумма всех байтов, представленных в строке, по модулю 256, равна нулю. Байты в строке записи данных:

1. Количество байтов изображения в строке.
2. Два байта для смещения загрузки первого байта изображения. В смещение сначала записывается более значимый байт, чтобы он правильно читался как четырехзначное шестнадцатеричное число.
3. нулевой байт "00"
4. Байты изображения в порядке возрастания адреса.
5. Байт контрольной суммы.

Конечная запись также начинается с символа «:» и записывается в виде пар цифр с конечной контрольной суммой. Ее формат:

1. нулевой байт "00"
2. Два байта для начального адреса в том же формате, что и смещение загрузки в записи данных. Используемый начальный адрес является первым выходным значением смещения загрузки для файла.
3. Один байт "01".
4. Байт контрольной суммы.

**Файлы S-Записей**

Файл в формате Motorola S-record представляет собой серию записей, каждая из которых содержит следующие поля:

**<S field><count><addr><data bytes><checksum>**

Вся информация представлена в виде пар шестнадцатеричных цифр, каждая пара которых представляет числовое значение байта.

Поле <S field> определяет интерпретацию оставшейся части строки. Допустимыми полями S являются "S0", "S1", "S2", "S8", "S9". <count> указывает количество байтов, представленных в остальной части строки, поэтому общее количество символов в строке равно <count>\*2 + 4.

Поле <addr> указывает байтовый адрес первого байта данных в поле данных. Записи S0 имеют два нулевых байта в качестве поля адреса; записи S1 и S2 имеют поля <addr> длиной два и три байта соответственно; записи S9 и S8 имеют поля <addr> длиной два и три байта соответственно и не содержат байтов данных. Поле <addr> сначала представляется наиболее значимый байт.

Запись S0 содержит имя входного файла, отформатированное как байты данных. Если ввод был от xeq, в качестве имени используется XEQ. Записи S1 и S2 представляют собой загружаемые байты текста или сегмента данных. Обычно они содержат 32 байта изображения, выводимых в порядке возрастания адресов; последняя запись каждого сегмента может быть короче. Сначала выводится текстовый сегмент, а затем сегмент данных. Записи S9 содержат только двухбайтовый начальный адрес в поле <addr>. Записи S8 содержат трехбайтовый адрес. Используемый начальный адрес является первым выходным значением смещения загрузки для файла.

Поле <checksum> - это однобайтовое значение, такое что числовая сумма всех байтов, представленных в строке (кроме поля S), взятая по модулю 256, равна 255 (0xFF).

**КОД ЗАВЕРШЕНИЯ**

Утилита **hex** возвращает успех, если сообщения об ошибках не выводятся, то есть если все записи имеют смысл и все операции чтения и записи выполнены успешно; в противном случае он сообщает об ошибке.

**ПРИМЕР**

Файл hello.c, состоящий из:

**char \*p {"hello world"};**

при компиляции создает следующий шестнадцатеричный файл Intel:

**% hex hello.o**

**$**

**:1A00000068656C6C6F20776F726C64000E0068656C6C6F20776F726C640020**

**:02001A000000E4**

**:00000001FF**

и следующие S-записи Motorola:

**% hex -s hello.o**

**$**

**S00A000068656C6C6F2E6F44**

**S11D000068656C6C6F20776F726C64000E0068656C6C6F20776F726C64001C**

**S105001A0000E0**

**S9030000FC**

**СМ. ТАКЖЕ**

link, rel

**НАЗВАНИЕ**

**lib** - обслуживание библиотек

**СИНТАКСИС**

**lib <lfile> -[c d i p r t v3 v6 v7 v x] <files>**

**ОПИСАНИЕ**

**lib** предоставляет все функции, необходимые для создания и обслуживания библиотек объектных модулей в стандартном формате библиотеки или в формате, используемом UNIX/System III, UNIX/V6 или UNIX/V7. lib также может использоваться для сбора произвольных файлов в одно хранилище. Поле <lfile> - это имя существующего файла библиотеки или, в случае операций замены или создания, имя создаваемой библиотеки.

Флаги:

|  |  |
| --- | --- |
| **-c** | Создать библиотеку, содержащую <files>, в указанном порядке. Любой существующий <lfile> удаляется перед созданием нового. |
| **-d** | Удалить из библиотеки ноль или более файлов в <files>. |
| **-i** | Взять <files> из STDIN вместо командной строки. Любые <files> в командной строке игнорируются. |
| **-p** | Распечатать именованные файлы (сделайте -x для вывода). Если имена файлов не указаны, все файлы будут извлечены. |
| **-r** | в существующей библиотеке заменить ноль или более файлов в <files>. Если библиотеки <lfile> не существует, создайте библиотеку, содержащую <files>, в указанном порядке. Файлы в разделе <files>, отсутствующие в библиотеке, добавляются в нее в указанном порядке. |
| **-t** | Перечислить файлы в библиотеке в порядке их появления. Если задано <files>, то в списке отображаются только названные и присутствующие файлы. |
| **-v3** | Создать выходной библиотеки в формате UNIX/System III, порядок байтов VAX-11. Этот флаг имеет значение с -c или -r; это обязательно для -t и -x (предполагается формат UNIX/V7, если этот флаг не указан). |
| **-v6** | Создать выходной библиотеки в формате UNIX/V6. Имеет значение только с -c или -r. |
| **-v7** | Создать выходной библиотеки в формате UNIX/V7, порядок байтов PDP-11. Имеет значение только с -c или -r. |
| **-v** | Быть многословным. Выводится имя каждого объектного файла, с которым выполняется операция, которому предшествует код, указывающий его статус: "c" для файлов, оставшихся без изменений, "d" для тех, которые были удалены, "r" для тех, которые были заменены, и "a" для тех, которые добавлены к <lfile>. Если -v используется вместе с -t, каждый объектный файл перечисляется с указанием его длины в байтах. |
| **-x** | Извлечь файлы в <files>, которые присутствуют в библиотеке, в отдельные файлы с одинаковыми именами. Если <files> не заданы, все файлы в библиотеке будут извлечены. |

Может быть задано не более одного из флагов -[c d p r t x]. Если ничего не указано, предполагается -t. Точно так же должно присутствовать не более одного из флагов -[v3 v6 v7]. Если его нет, используется стандартный формат файла библиотеки при создании новой библиотеки с помощью -r или -c. Существующая библиотека обрабатывается в надлежащем режиме, за исключением библиотек UNIX/ III, для которых необходимо указать флаг -v3.

Формат стандартной библиотеки состоит из двухбайтового заголовка, имеющего значение 0177565, сначала записывается менее значимый байт, за которым следуют ноль или более записей. Каждая запись состоит из четырнадцати байтового имени, дополненного NUL, за которым следует двухбайтовая длина файла без знака, также сначала хранится менее значимый байт, а затем собственно содержимое файла. Если имя начинается с байта NUL, оно считается концом файла библиотеки.

Обратите внимание, что он несколько отличается от формата UNIX/V6, который имеет заголовок 0177555, восьми байтовое имя, шесть байтов различных атрибутов файла, специфичных для UNIX, и двухбайтовую длину файла. Более того, за файлом с нечетной длиной следует нулевой байт заполнения в формате UNIX, в то время как в формате стандартной библиотеки заполнение не используется.

Форматы UNIX/III и UNIX/V7 характеризуются заголовком 0177515, четырнадцати байтовым именем, восемью байтами специфичных для UNIX файловых атрибутов и четырехбайтовой длиной. Длина указывается в собственном порядке байтов VAX-11 для UNIX/III или в собственном порядке байтов PDP-11 для UNIX/V7. Файлы нечетной длины также дополняются до четной.

**КОД ЗАВЕРШЕНИЯ**

**lib** возвращает успех, если проблем не обнаружено, иначе - сбой. После большинства сбоев в STDERR выводится сообщение об ошибке, а файл библиотеки не изменяется. Вывод флага -t и подробные примечания записываются в STDOUT.

**ПРИМЕР**

Чтобы создать библиотеку и проверить ее содержимое:

**% lib clib -r one.o two.o three.o**

**% lib olib -tv**

**СМ. ТАКЖЕ**

link, lord, rel

**ОШИБКИ**

Если все модули удалены из библиотеки, остается остаточный файл.

Изменение файлов формата UNIX/III, UNIX/V6 или UNIX/V7 приводит к обнулению всех атрибутов всех записей файлов.

lib не проверяет файлы, слишком большие для правильного представления в библиотеке (> 65 534 байта).

**НАЗВАНИЕ**

**link** - объединение объектных файлов

**СИНТАКСИС**

**link -[a bb## b## o db## dr# d eb\* ed\* et\* h i l\*^ o\* r sb\* sd\* st\* tb## tf# t u\*^ x#] <files>**

**ОПИСАНИЕ**

**link** объединяет перемещаемые объектные файлы в стандартном формате для любой целевой машины, выборочно загружая из библиотек таких файлов, созданных с помощью lib, для создания исполняемого образа для работы в Idris или для автономного выполнения, или для ввода в другие двоичные преобразователи.

Флаги:

|  |  |
| --- | --- |
| **-a** | Сделать все элементы перемещения и все символы абсолютными. Используется для предварительного размещения кода, который будет связан в другом месте и не подлежит повторному перемещению. |
| **-bb##** | Переместить сегмент bss (block started by symbol), чтобы он начинался с ##. По умолчанию сегмент bss перемещается относительно конца сегмента data. После указания -bb## результирующий выходной файл не подходит для ввода для другого вызова link. |
| **-b##** | Установить размер стека плюс кучи в заголовке модуля вывода на ##. Idris принимает это значение, если оно не равно нулю, как минимальное количество байтов, которое необходимо зарезервировать во время выполнения для увеличения стека и области данных. Если значение нечетное, Idris будет выполнять программу с меньшим размером кучи плюс размер стека, пока можно будет обеспечить некоторый неснижаемый минимальный объем пространства за счет использования всей доступной памяти. |
| **-c** | Подавить вывод кода (.text и .data) и сделать все символы абсолютными. Используется для создания модуля, определяющего только значения символов, для указания адресов в разделяемых библиотеках и т.д. |
| **-db##** | Установить смещение данных в длинное целое число ##. По умолчанию используется конец текстового раздела, округленный до требуемой границы хранилища для целевой машины. |
| **-dr#** | Округлить смещение данных в большую сторону, чтобы убедиться, что в его значении содержится как минимум # двоичных нулей младшего разряда. Игнорируется, если указан -db##. |

|  |  |
| --- | --- |
| **-d** | Не определять символы bss и не жаловаться на неопределенные символы. Используется для частичных ссылок, т.е. Если модуль вывода должен вводиться в link |
| **-eb\*** | Если имеется ссылка на символ \*, сделать ее равной первому неиспользуемому местоположению за областью bss. |
| **-ed\*** | Если имеется ссылка на символ \*, сделать ее равной первому неиспользуемому местоположению после области инициализированных данных. |
| **-et\*** | Если имеется ссылка на символ \*, сделать ее равной первому неиспользуемому месту за текстовой областью. |
| **-h** | Скрыть заголовки в выходном файле. Это следует указывать только для автономных модулей, таких как загрузочные файлы. |
| **-i** | Взять <files> из STDIN вместо командной строки. Любые <files> в командной строке игнорируются. |
| **-l\*** | Добавить имя библиотеки в конец списка файлов для связывания, где имя библиотеки формируется добавлением \* к "/lib/lib". Таким образом, "-lc.11" производит "/lib/libc.11". В качестве флагов можно указать до десяти таких имен. Они добавляются в указанном порядке. |
| **-o\*** | Записать модуль вывода в файл \*. По умолчанию - xeq. |
| **-r** | Подавить биты перемещения. Это не следует указывать, если модуль вывода должен быть введен для link или выполняться под версией Idris, которая перемещает команды при запуске. |
| **-sb#** | Если имеется ссылка на символ \*, сделать ее равной размеру области bss. |
| **-sd#** | Если имеется ссылка на символ \*, сделать ее равной размеру области данных. |
| **-st#** | Если на символ \* есть ссылка, сделать ее равной размеру текстовой области |
| **-tb##** | Установить смещение текста в длинное целое число ##. По умолчанию используется нулевое местоположение. |
| **-tf#** | Округлить размер текста, добавив нулевые байты в раздел текста, чтобы убедиться, что в результирующем значении есть по крайней мере # двоичные нули низкого порядка. Значение по умолчанию равно нулю, т. е. без заполнения. |
| **-t** | Подавить таблицу символов. Это не следует указывать, если модуль вывода должен быть введен в link. |
| **-u\*** | Ввести символ \* в таблицу символов в качестве неопределенной общедоступной ссылки, обычно для принудительной загрузки выбранных модулей из библиотеки. |

|  |  |
| --- | --- |
| **-x#** | Указать размещение в модуле вывода входных разделов .text и .data. 2-взвешенный бит # направляет ввод .text, 1-взвешенный бит .data. Если установлен какой-либо бит, соответствующие разделы помещаются в вывод текстового сегмента; в противном случае они попадают в сегмент данных. По умолчанию используется значение 2. |

Предполагается, что секция bss всегда следует за секцией данных во всех входных файлах. Если не задано -bb##, секция bss также будет следовать за секцией данных в выходном файле. Совершенно допустимо, чтобы разделы текста и данных перекрывались, если речь идет о ссылке). Целевая машина может или не может понимать эту ситуацию (как с отдельными пространствами инструкций и данных).

Указанные <files> связаны по порядку; если файл имеет формат библиотеки, поиск по ней выполняется один раз от начала до конца. Включены только те библиотечные модули, которые определяют общедоступные символы, для которых в настоящее время имеются невыполненные неудовлетворенные ссылки. Следовательно, библиотеки должны быть тщательно упорядочены или повторно проверены, чтобы убедиться, что все ссылки разрешены. По специальному разрешению, флаги формы "-l\*" могут перемежаться между <files>. Они требуют поиска соответствующих библиотек в точках, указанных в списке файлов. В этом случае после "-l" не должно быть пробелов.

**Формат файла**

Изображение перемещаемого объекта состоит из заголовка, за которым следует текстовый сегмент, сегмент данных, таблица символов и информация о перемещении.

Заголовок состоит из байта идентификации 0x99, байта конфигурации, короткого целого числа без знака, содержащего количество байтов таблицы символов, и шести целых чисел без знака, дающих: количество байтов объектного кода, определяемого текстовым сегментом, количество байтов объектного код, определяемый сегментом данных, количество байтов, необходимых для сегмента bss, количеством байтов, необходимых для стека плюс кучи, смещением текстового сегмента и смещением сегмента данных.

Порядок байтов и размер всех целых чисел в заголовке определяются байтом конфигурации.

Байт конфигурации содержит всю информацию, необходимую для полного представления заголовка и оставшейся информации в файле. Его значение val определяет следующие поля: ((val & 07) << 1) + 1 - это количество символов в поле имени таблицы символов, так что значения [0, 8] обеспечивают нечетную длину в диапазоне [1, 15]. Если (val & 010), то целые четыре байта; в противном случае это два байта. Если (val & 020), то целые числа в сегменте данных сначала представляются младшим значащим байтом, в противном случае - самым старшим байтом; порядок байтов предполагается чисто восходящим или чисто нисходящим. Если (val & 040), то даже границы байтов устанавливаются оборудованием. Если (val & 0100), то порядок байтов текстового сегмента меняется на обратный по сравнению с порядком байтов сегмента данных; в противном случае порядок байтов текстовых сегментов такой же. Если (val & 0200) в этом файле нет информации о перемещении.

Текстовый сегмент перемещается относительно смещения текстового сегмента, указанного в заголовке (обычно нуля), в то время как сегмент данных перемещается относительно смещения сегмента данных (обычно конца текстового сегмента). Если не указан -bb##, сегмент bss перемещается относительно конца сегмента данных.

Информация о перемещении состоит из двух последовательных байтовых потоков, один для текстового сегмента и один для сегмента данных, каждый из которых заканчивается нулевым контрольным байтом. Управляющие байты в диапазоне [1, 31] вызывают пропуск такого количества байтов в соответствующем сегменте. Байты в диапазоне [32, 63] пропускают 32 байта плюс 256 байт управления минус 32 плюс количество байтов, указанное следующим байтом перемещения.

Все остальные управляющие байты управляют перемещением следующего короткого или длинного int в соответствующем сегменте. Если 1-взвешенный бит установлен в таком управляющем байте, то изменение смещения загрузки должно быть вычтено из int.

2-взвешенный бит устанавливается, если перемещается длинный int вместо короткого int. Значение управляющего байта, сдвинутого вправо на два места, минус 16, представляет собой "код символа".

Код символа 47 заменяется кодом, полученным из байта или байтов, следующих в потоке перемещения. Если следующий байт меньше 128, то код символа - это его значение плюс 47. В противном случае код равен этому байту минус 128, умноженному на 256, плюс 175 плюс значение следующего байта перемещения после этого.

Код символа, равный нулю, не требует дальнейшего перемещения; 1 означает, что к элементу необходимо добавить изменение смещения текста (short или long int); 2 означает, что необходимо добавить изменение смещения данных; 3 означает, что необходимо добавить изменение смещения bss. Другие коды символов требуют, чтобы значение записи таблицы символов, индексированной кодом символа минус 4, было добавлено к элементу.

Каждая запись таблицы символов состоит из значения int, байта флага и имени, дополненного конечными NUL. Значимые значения флага: 0 для неопределенного, 4 для определенного абсолютного, 5 для определенного относительного текста, 6 для определенного относительного значения и 7 для определенного относительного bss. К этому добавляется 010, если символ должен быть известен во всем мире. Если символ все еще не определен после связывания, имеет ненулевое значение, ссылка назначает символу уникальную область в сегменте bss, длина которого определяется значением, и считает символ определенным. Это происходит только в том случае, если не было указано -d.

**КОД ЗАВЕРШЕНИЯ**

Утилита **link** возвращает успех, если в STDOUT не выводятся сообщения об ошибках, то есть, если не осталось неопределенных символов и если все операции чтения и записи выполнены успешно; в противном случае возвращается неудача.

**ПРИМЕР**

Чтобы загрузить программу C echo.o в Idris/S11:

**% link -lc.11 -t /lib/Crts.o eoho.o**

с отдельными пространствами I/D для UNIX:

**% link -lc.11 -rt –db0 /lib/Crts.o eoho.o; taout**

или с текстовой секцией только для чтения для UNIX:

**% link -lc.11 -rt -dr13 /lib/Crts.o echo.o; taout**

И чтобы загрузить версию echo 8080 под CP/M:

**a: link -hrt –tb0x0l00 a:chdr.o echo.o a:clib.a a:mlib.a**

**СМ. ТАКЖЕ**

hex, lib, lord, rel

**НАЗВАНИЕ**

**lord** - порядок библиотек

**СИНТАКСИС**

**lord -[c\* d\*^ i r\*^ s]**

**ОПИСАНИЕ**

Утилита **lord** считывает список имен модулей со связанными взаимозависимостями из STDIN и выводит в STDOUT топологически отсортированный список имен модулей, так что, если это вообще возможно, ни один модуль не зависит от более раннего модуля в списке. Каждый модуль вводится строкой, содержащей его имя, за которым следует двоеточие. Последующие строки интерпретируются как:

def - вещи, определенные модулем.

ref - вещи, на которые ссылается модуль, или

другие вещи - другие вещи.

Ref и def имеют синтаксис, заданный одним или несколькими форматами, введенными как флаги в командной строке. Каждый символ формата должен соответствовать соответствующему символу в начале строки ввода; ? будет соответствовать любому символу, кроме новой строки. Если символы формата совпадают, то оставшаяся часть строки ввода принимается как имя ref или def. Таким образом, флаг формата "-d0x????D " будет идентифицировать как допустимую def любую строку, начинающуюся с "0x", четыре произвольных символа и "D ", так что входная строка "0x3ff0D \_inbuf" будет принята как def с именем "\_inbuf"..

Флаги:

|  |  |
| --- | --- |
| **-c\*** | Добавить строку \* к выходному потоку. Подразумевает -s. Каждое имя модуля выводится с пробелом. Выходной поток заканчивается новой строкой. Следовательно, lord можно использовать для создания командной строки. |
| **-d\*** | Использовать строку \* как формат для def. |
| **-i** | Игнорировать другие вещи. По умолчанию нужно жаловаться на любую строку, которая не распознается как ref или def. |
| **-r\*** | Использовать строку \* как формат для ref. |
| **-s** | Подавить вывод def и ref. Выводить только имена модулей по порядку. |

Можно ввести до десяти форматов для def и до десяти для ref.

Если флаги -d не указаны, lord использует форматы def по умолчанию: "0x????????B ", "0x????????D ", "0x????????T ", "0x????B ", "0x????D ", "0x????T ". Если флаги -r не указаны, lord использует стандартные форматы ref: "0x????????U " и "0x????U ". Они совместимы с выводом по умолчанию rel.

При наличии циклических зависимостей между модулями lord записывает имя файла, начинающего несортированный список, в STDERR, за которым следует сообщение "not completely sorted". В общем случае перестановки делаются только при необходимости, поэтому упорядоченный набор модулей должен проходить через lord без изменений.

**КОД ЗАВЕРШЕНИЯ**

Утилита **lord** возвращает успех, если сообщения об ошибках не печатаются, в противном случае сбой.

**ПРИМЕР**

Чтобы создать упорядоченную библиотеку объектных модулей в Idris:

**% rel \*.o | lord -c"lib libx.a -c" | sh**

Для заказа набора объектов с помощью UNIX nm:

**% nm \*.o > nmlist**

**% lord < nmlist -o"ar r libx.a" I \**

**-d"??????T " -d"??????D " -d"??7???B " -r"??????U" | sh**

**СМ. ТАКЖЕ**

lib, rel

**НАЗВАНИЕ**

**p1** - разбор программ на языке C

**СИНТАКСИС**

**p1 -[a b# c e l m n# o\* r# u] <flle>**

**ОПИСАНИЕ**

Программа **p1** - это проход синтаксического анализа компилятора C. Она принимает последовательный файл лексем из препроцессора pp и записывает последовательный файл графиков потока и деревьев синтаксического анализа, пригодный для ввода в машинно-зависимый генератор p2 кода. Работа p1 в значительной степени не зависит от любой целевой машины. Параметры флага:

|  |  |
| --- | --- |
| **-a** | Компиляция кода для машин с отдельными адресами и регистрами данных. Этот флаг подразумевает -r6 (потому что в настоящее время используется только в сочетании с генератором кода MC68000). |
| **-b#** | Установить границы хранения в соответствии с #, который уменьшается по модулю 4. Граница 0 не оставляет дыр в структурах или автоматических распределениях. Граница 1 (по умолчанию) требует, чтобы короткие, входные и более длинные данные начинались на четной границе. Граница 2 та же, что и 1, за исключением того, что 4-8 байтовые данные принудительно преобразуются в кратную четырехбайтовой границе. Граница 3 такая же, как и 2, за исключением того, что 8 байтовые данные (double) принудительно преобразуются в границу, кратную восьми байтам. |
| **-c** | Игнорировать различия в регистре при проверке внешних идентификаторов на равенство и перевести все имена в нижний регистр при выводе. По умолчанию различия в регистре имеют значение. |
| **-e** | Не вызывать принудительную загрузку внешних ссылок, которые объявлены, но никогда не определены или не используются в выражении. По умолчанию загружаются все объявленные extern. |
| **-l** | Целые числа и указатели должны иметь длину 4 байта. По умолчанию 2 байта. |
| **-m** | Рассматривать каждую struct/union как отдельное пространство имен и требовать, чтобы у x.m была структура x с одним из членов m. |
| **-n#** | Игнорировать символы после первого # при проверке равенства внешних идентификаторов. По умолчанию 7; максимум - 8, за исключением того, что значения до 32 разрешены только для генератора кода VAX-11. |
| **-o\*** | Записать вывод в файл \* и записать сообщения об ошибках в STDOUT. По умолчанию для вывода используется STDOUT, а для сообщений об ошибках - STDERR. |
| **-r#** | Назначить регистрам не более # переменных в любой момент времени, где 4 уменьшается по модулю 7. Значение по умолчанию - 3 переменные регистра; значения выше 3 в настоящее время приемлемы только для генератора кода MC68000 (3 регистра данных + 3 адресных регистра максимум) и генератора кода VAX-11 (6 регистров максимум). |
| **-u** | Взять "string" как массив беззнаковых символов, а не как массив символов. |

Если присутствует <flle>, он используется в качестве входного файла вместо STDIN по умолчанию. Во многих системах (кроме Idris/UNIX) параметр -o и <flle> являются обязательными, поскольку STDIN и STDOUT интерпретируются как текстовые файлы и, следовательно, становятся поврежденными.

**ПРИМЕР**

p1 обычно помещается между pp и некоторой версией p2, как в:

**pp -x -o temp1 file.c**

**p1 -o temp2 temp1**

**P2.11 -o file.s temp2**

**СМ. ТАКЖЕ**

pp

**ОШИБКИ**

Программа p1 может быть довольно бесцеремонной в отношении точек с запятой.

**НАЗВАНИЕ**

**p2.80** - генератор кода 8080 для C программ

**СИНТАКСИС**

**p2.80 -[ck o\* p x#] <file>**

**ОПИСАНИЕ**

Программа **p2.80** - это этап генерации кода компилятора C. Она принимает последовательный файл потоковых графов и деревьев синтаксического анализа из p1 и записывает последовательный файл операторов A-Natural, подходящий для ввода в ассемблер A-Natural, as.80.

Насколько это возможно, компилятор генерирует автономный код. Но для тех операций, которые не могут быть выполнены компактно, он генерирует встроенные вызовы набора машинно-зависимых библиотечных подпрограмм. Библиотека времени выполнения 8080 задокументирована в Разделе IV этого руководства.

Флаги:

|  |  |
| --- | --- |
| **-ck** | Включить проверку переполнения стека. |
| **-o\*** | Записать вывод в файл \* и записать сообщения об ошибках в STDOUT. По умолчанию для вывода используется STDOUT, а для сообщений об ошибках - STDERR. |
| **-p** | Выполнять вызовы профилировщика при входе в каждую функцию. |
| **-x#** | Сопоставить три виртуальных раздела: для функций (04), литералов (02) и переменных (01) с двумя физическими разделами Code (бит равен единице) и Data (бит равен нулю). Таким образом, "-x4" - для отдельного пространства I/D, "-x6" - для кода ПЗУ/ОЗУ, а "-x7" - для компиляции таблиц в ПЗУ. По умолчанию 6. |

Если <file> присутствует, он используется в качестве входного файла вместо стандартного STDIN по умолчанию. Во многих системах (кроме Idris/UNIX) <file> является обязательным, поскольку STDIN интерпретируется как текстовый файл и, следовательно, повреждается.

Файлы, выводимые из p1 для использования с генератором кода 8080, должны быть сгенерированы с: "-b0", чтобы устранить дыры в структурах и автоматическом распределении, и "-n8" для внешних имен полной длины. Граница "-b1" (по умолчанию) также приемлема, если компактное хранилище менее важно, чем совместимость со структурами данных на других машинах, таких как PDP-11. Обратите внимание, что использование флагов p1 "-b2", "-b3" или "-1" приведет к странному поведению и неправильному коду в p2.80, часто без значимой диагностики.

Везде, где это возможно, каждая метка в сгенерированном коде содержит комментарий, который дает исходную строку, из которой происходит получение следующего кода.

**ПРИМЕР**

Программа p2.80 обычно следует за pp и p1, а именно:

**pp -x -o tempi file.c**

**p1 -b0 -n8 -o temp2.80 temp1**

**p2.80 -o file.s temp2**

**НАЗВАНИЕ**

**pp** - препроцессор

**СИНТАКСИС**

**pp -[c d\*^ i\* o\* p? s? x 6] <files>**

**ОПИСАНИЕ**

**pp** - это препроцессор, используемый компилятором C для выполнения #define, #include и других функций, обозначенных символом #, перед фактическим началом компиляции. Однако его можно использовать с большей пользой для большинства языковых процессоров. Варианты флага:

|  |  |
| --- | --- |
| **-c** | Не удалять /\* комментарии \*/ и не продолжать строки, заканчивающиеся на \. |
| **-d\*** | Где \* имеет форму name=def, перед чтением ввода определить имя со строкой определения def. Если =def опущено, определение принимается как "1". name и def должны быть в одном аргументе, т.е. пробелы не допускаются, если аргумент не заключен в кавычки. Таким образом можно ввести до десяти определений. |
| **-i\*** | Изменить префикс, используемый с #include <filename> со значения по умолчанию "" на строку \*. Можно указать несколько префиксов, которые нужно попробовать по порядку, через символ '|'. |
| **-o\*** | Записать вывод в файл \* и записать сообщения об ошибках в STDOUT. По умолчанию для вывода используется STDOUT, а для сообщений об ошибках - STDERR. Во многих системах (кроме Idris/UNIX) опция -o является обязательной с -x, потому что STDOUT интерпретируется как текстовый файл и, следовательно, становится поврежденным. |
| **-p?** | Изменить управляющий символ препроцессора с '#' на символ ?. |
| **-s?** | Изменить управляющий символ вторичного препроцессора с '@' на символ ?. |
| **-x** | Вывод лексем для ввода в компилятор C p1, а не строк текста. |
| **-6** | Выводить дополнительные новые строки и/или коды SOH ('\1'), чтобы номера исходных строк оставались правильными для UNIX/V6 компилятора или ptc. |

Препроцессор pp обрабатывает названные файлы или STDIN, если они не указаны, в указанном порядке, записывая полученный текст в STDOUT. Действия препроцессора подробно описаны в разделе I Руководства программиста на C.

Наличие дополнительного символа управления препроцессором обеспечивает два уровня параметризации. Например, вызов

**pp -c –p@**

расширит определение и ifdef условий, оставив все команды # и комментарии нетронутыми. При вызове pp без аргументов будут расширены команды @ и #. Флаг -s# будет эффективно отключать дополнительный управляющий символ.

**ПРИМЕР**

Стандартный стиль написания C-программ:

**/\* name of program**

**\*/**

**#include <std.h>**

**#define MAXN 100**

**COUNT things[MAXN];**

**etc.**

Использование идентификаторов только в верхнем регистре не требуется для pp, но настоятельно рекомендуется, чтобы отличать параметры от обычных программных идентификаторов и ключевых слов.

**СМ. ТАКЖЕ**

p1, ptc

**ОШИБКИ**

Несбалансированные кавычки ' или " могут не встречаться в строке даже при отсутствии флага -x. Плавающая константа длиной более 38 цифр может быть неправильно скомпилирована на некоторых хост-компьютерах.

**НАЗВАНИЕ**

**prof** - создание профиля выполнения

**СИНТАКСИС**

**prof -[a f +l n r s t +z] [<ofile>] [<pfiles>]**

**ОПИСАНИЕ**

Утилита **prof** сопоставляет один или несколько файлов статистики, записанных во время выполнения программы, с таблицей символов соответствующего исполняемого файла и создает отчет, профилирующий выполнение. Статистика содержится в файлах "profile" стандартного формата. Поле <ofile>, указанное в командной строке, - это программа, выполнение которой привело к получению статистики.

Флаги:

|  |  |
| --- | --- |
| **-a** | Сортировать выходные данные по возрастающему порядку адресов. |
| **-f** | Сортировать выходные данные в порядке убывания частоты вызовов каждой функции. |
| **+l** | Включить локальные символы в выходные данные. По умолчанию включаются только глобальные символы. |
| **-n** | Сортировать выходные данные в порядке увеличения имени символа. |
| **-r** | Изменить значение по умолчанию для выходной сортировки. |
| **-s** | Подавить заголовок по умолчанию. |
| **-t** | Сортировать выходные данные в порядке уменьшения времени, затраченного на каждую функцию. Это ключ сортировки по умолчанию. |
| **+z** | Включить символы в выходные данные, для которых не было записано ни записей, ни времени выполнения. По умолчанию они исключаются. |

Если <ofile> не задан, в качестве имени файла объекта используется "xeq". Если <pfiles> не заданы, prof ожидает, что данные профилирования поступят из файла "profil". Однако если после [<ofile> в командной строке появляются какие-либо имена файлов, они будут считываться для профилирования данных. Если задано более одного файла, prof будет накапливать данные из каждого файла по очереди и выводить один отчет, содержащий итоговые данные из всех прочитанных файлов данных. Таким образом, результаты нескольких инструментальных прогонов данной программы могут быть объединены для уменьшения статистической погрешности. Естественно, если заданы <pfiles>, также должен быть указан <ofile>.

Профилирование включает в себя две статистики: подсчет вызовов каждой функции в программе и запись того, какая часть времени выполнения тратится внутри каждой функции. В отчете prof выводит по одной строке для каждого относительного текста символа в <file> (с учетом +l и +z), показывающих имя символа, процент общего времени выполнения, проведенного в области между этим символом и следующим более высоким, время в этой области в миллисекундах, количество вызовов, выполненных непосредственно по адресу этого символа, и среднее количество времени, записанное на вызов. Кроме того, prof рассматривает конечное местоположение, доступное для подсчета записей функций, как местоположение переполнения, в котором вызовы учитываются для всех функций, которым не может быть предоставлено собственное местоположение. Если этот счетчик не равен нулю, prof выводит его с именем "other syms".

ОС Idris записывает время выполнения, периодически проверяя PC исполняемой программы, используя его значение для индексации массива, элементы которого соответствуют допустимому диапазону PC. Затем выбранный элемент массива увеличивается. Количество записей функций поддерживается вызовами процедуры подсчета, которую каждый компилятор Whitesmiths C может вывести, указав флаг -p в генераторе кода. Таким образом, функция должна быть инструментирована, чтобы для нее сохранялось количество записей. Сама процедура подсчета указывается во время link как часть кода запуска во время выполнения.

**КОД ЗАВЕРШЕНИЯ**

Утилита prof возвращает успех, если все входные файлы доступны для чтения и соответствуют их ожидаемому формату.

**СМ. ТАКЖЕ**

Описание формата файла профиля находится в Разделе III Руководства программиста Idris, а код инициализации среды выполнения, используемый для профилирования в Idris, задокументирован вместе с системным интерфейсом Idris в Разделе III.a этого руководства. Функции настройки портативного профилирования описаны в Разделе IV Руководства программиста Idris, а процедура подсчета вводов машинно-зависимых функций описана в Разделе IV настоящего руководства.

**ОШИБКИ**

Ошибки выборки могут возникать, если коэффициент масштабирования, используемый для записи местоположения PC, не превышает наименьшей границы, с которой может начинаться инструкция; то есть данный элемент массива может перекрывать тела функций, и PC, захваченный одной функцией, может обрабатываться так, как если бы он был захвачен другой, prof делает все возможное для компенсации, разделяя значение перекрывающегося элемента массива между задействованными функциями.

Эта утилита в настоящее время предназначена для использования только в хост-системах Idris.

**НАЗВАНИЕ**

**ptc** - переводчик с Pascal на C

**СИНТАКСИС**

**ptc -[c f k m# n# o\* r s#] <ifile>**

**ОПИСАНИЕ**

**ptc** - это программа, которая принимает в качестве входных строк текст Pascal и производит в качестве выходных данных соответствующую программу на C, которая является приемлемой для компилятора C Whitesmiths, Ltd. Если присутствует <ifile>, для перевода используется программа на языке Pascal, в противном случае ввод берется из STDIN.

Флаги:

|  |  |
| --- | --- |
| **-c** | Передать комментарии программе C. |
| **-f** | Установить для вещественных чисел одинарную точность (float). Значение по умолчанию - double. |
| **-k** | Разрешить определение типов указателей с использованием идентификаторов типов из внешних блоков. По умолчанию используется стандарт ISO, т.е. указываемый тип должен быть определен в том же объявлении типа, что и определение типа указателя. |
| **-m#** | Определить количество битов в MAXINT, исключая бит знака, например, MAXINT становится 32767 для -m15, 1 для -m1 и т.д. Значение по умолчанию для MAXINT - 32766 [sic]. Допустимые значения для 4 находятся в диапазоне [0, 32]. Объявление MAXINT больше размера указателя (16 или 32) даст непредсказуемые результаты. На целевой машине с 16-битными указателями # должно быть меньше 16. |
| **-n#** | Определить # количеством значащих символов во внешних именах. По умолчанию внешние имена состоят из 8 символов. |
| **-o\*** | Записать программу C в файл \* и диагностику в STDOUT. По умолчанию это STDOUT для программы C и STDERR для диагностики. |
| **-r** | Отключить проверку границ массива во время выполнения. |
| **-s#** | Определить # количество битов в максимально допустимом размере набора, то есть размер всех наборов, базовый тип которых является целым числом, становится указанной степенью двойки. Допустимые значения находятся в диапазоне [0, 32]. По умолчанию 8 (256 элементов). |

Реализация операционной системы CP/M на Intel 8080 и Zilog Z/80 ограничивает допустимое значение для # диапазоном [0, 16]. Для максимальной переносимости это ограничение должно соблюдаться.

Идентификаторы отображаются в верхнем регистре, чтобы не конфликтовать с теми, которые объявлены как зарезервированные слова в C. Кроме того, могут создаваться объявления структуры, содержащие противоречивые объявления полей; и объявления присутствуют для библиотечных функций, которые могут не потребоваться. Все эти прегрешения прощаются при использовании соответствующих опций компилятора Си.

**КОД ЗАВЕРШЕНИЯ**

Программа **ptc** возвращает успех, если не производит диагностику.

**ОШИБКИ**

Выражение сложного набора может создавать очень длинные строки, которые усекаются pp.

**НАЗВАНИЕ**

**rel** - проверка объектных файлов

**СИНТАКСИС**

**rel -[d g i o s t u v] <files>**

**ОПИСАНИЕ**

Утилита **rel** позволяет проверять перемещаемые объектные файлы в стандартном формате для любой целевой машины. Такие файлы могли быть выведены ассемблером, объединены с помощью link или заархивированы lib. Утилиту rel можно использовать либо для проверки их размера и конфигурации, либо для вывода информации из таблиц символов.

Флаги:

|  |  |
| --- | --- |
| **-d** | Вывести все определенные символы в каждом файле, по одному в каждой строке. Каждая строка содержит значение символа, код, указывающий, к какому значению относится значение, и имя символа. Значения выводятся как количество цифр, необходимых для представления целого числа на целевой машине. Коды перемещения: 'T' для относительного текста, 'D' для относительных данных, 'B' для относительного bss, 'A' для абсолютного или '?' для всего, что rel не распознает. Строчные буквы используются для локальных символов, прописные - для глобальных. |
| **-g** | Печатать только глобальные символы. |
| **-i** | Вывести все глобальные символы с интервалом в байтах между последовательными символами, указанными в каждом поле значения. Подразумевает флаги -[d u v]. |
| **-o** | Вывести значения символов в восьмеричном формате. Значение по умолчанию-шестнадцатеричное. |
| **-s** | Отобразить в десятичном формате размеры текстового сегмента, сегмента данных, сегмента bss и пространства, зарезервированного для стека времени выполнения плюс кучи, а затем сумму всех размеров. |
| **-t** | Список информации о типе для этого файла. Для каждого объектного файла выводятся следующие данные: размер целого числа на целевом компьютере, порядок байтов на целевом компьютере, соблюдаются ли аппаратными средствами четные границы байтов, изменяется ли порядок байтов текстового сегмента по сравнению с порядком сегментов данных и максимальное количество символов, разрешенных во внешнем имени. Если файл является библиотекой, то выводится тип библиотеки и приведенная выше информация для каждого модуля в библиотеке. |
| **-u** | Перечислить все неопределенные символы в каждом файле. Если также указан параметр -d, каждый неопределенный символ отображается с кодом 'U'. Значение каждого символа, если оно не равно нулю, представляет собой пространство, которое должно быть зарезервировано для него во время загрузки, если оно не определено явно. |
| **-v** | Сортировать по значению. Подразумевает флаг -d выше. Символы равного значения отсортированы по алфавиту. |

Если флаги не указаны, значение по умолчанию -[d u], то есть перечислены все символы, отсортированные в алфавитном порядке по имени символа. Если выбрано несколько флагов -[d s t u], то сначала выводится информация о типе, затем размеры сегментов, а затем список символов, указанный с помощью -d или -u.

<files> задает ноль или более файлов, которые должны быть в перемещаемом формате, или стандартном библиотечном формате, или библиотечном формате UNIX/V6, или библиотечном формате UNIX/V7. Если указано более одного файла или библиотеки, то имя каждого отдельного файла или модуля предшествует любому информационному выводу для него, каждое имя сопровождается двоеточием и новой строкой. Если задан -s, также выводится строка итогов. Если <files> не указаны или они встречаются в командной строке, используется xeq.

**КОД ЗАВЕРШЕНИЯ**

Утилита **rel** возвращает успех, если диагностика не производится, то есть если все операции чтения успешны и все форматы файлов допустимы.

**ПРИМЕР**

Чтобы получить список всех символов в модулях

**% rel alloc.o**

**0x00000074T \_alloc**

**0x00000000U \_exit**

**0x00000IfeT \_free**

**0X000000beT \_nalloc**

**0X00000000U \_sbreak**

**0X00000000U \_write**

**СМ. ТАКЖЕ**

lib, link, lord