

## ЯЗЫК ПРОГРАММИРОВАНИЯ С И ОПЕРАЦИОННАЯ СИСТЕМА UNIX

А. Н. Маслов

Перспективным подходом к решению проблемы повышения эффективности применения ЭВМ является использование машинно-независимой операционной системы на ЭВМ различных типов. При этом увеличивается срок жизни программных средств, упрощается обучение программированию и облегчается создание локальных сетей ЭВМ.

Наиболее широкое применение получила машинно-независимая операционная система UNIX [98], написанная на языке программирования С (читается си). Поставляются версии системы UNIX для ЭВМ PDP-11, VAX-11, HP 9000, IBM 370, IBM Series/1, Eclipse MV, Interdata 8/32, целого ряда персональных ЭВМ, а также поддерживаемая аппаратно реализация на Pугамid 90X; сообщалось о работах по переносу UNIX на Cгау-2. Имеется множество систем, аналогичных UNIX, но отличающихся в отдельных аспектах. Большинство из них написано на языке С, но используются также другие языки, такие как Паскаль [33], Симула и Евклид [46].

В 1982 году [103] редакция журнала Electronics присудила премию «За достижения» авторам системы UNIX и языка программирования С, сотрудникам фирмы Bell Labs Кеннету Томпсону и Денису М. Ритчи.

Ядро операционной системы UNIX — это набор программ для управления внешними устройствами и программными процессами.

Язык заданий shell [8] является лицом системы UNIX. Он предназначен для управления исполнением команд и композиций команд. Команда может работать как с файлом, так и с внешним устройством. Для композиции команд используются средства, аналогичные управляющим конструкциям языков программирования, а также «магистраль», обеспечивающая соединение стандартного вывода одной команды со стандартным вводом другой, что сокращает длину заданий.

Компиляторы и интерпретаторы языков программирования являются обычными командами. В поставляемых версиях доступны языки С, Паскаль, Фортран, Лисп и другие. Имеются также сведения о реализации в системе UNIX языка Алгол 68.

Для управления разработкой больших программных комплексов используется команда конфигурационного управления make и система управления SCCS. Имеются системы управления базами данных (в частности, классическая система Ингре [112]) и сетевые пакеты [6, 7, 12, 17, 18, 26, 54, 59, 63, 65, 69, 72, 76, 84, 86, 89, 94, 95, 102, 116, 117].

Система UNIX используется для различных приложений, включая автоматизацию научных экспериментов [3], архитектурное проектирование [24], написание статей и книгоиздание, медицину и даже уроки танцев [42]. На основе системы UNIX созданы и разрабатываются локальные и коммутируемые сети ЭВМ. Отдельные версии системы UNIX разработаны по заказу Управления перспективных научно-исследовательских работ (ARPA) министерства обороны США. UNIX является торговой маркой фирмы Bell Laboratories (США).

Большинство статей по системе UNIX опубликовано в журналах Bell Syst. Techn. J., 1978, 57, № 6, 1980, 59, № 9, 1982, 61, № 6, 1983, 62, № 1; Electronics, 1983, 53, № 15 (перевод на русский язык: Электроника, М., «Мир») и других номерах; Mini-Micro Syst., Software-Pract. and Exper. и в подборках статей фирм — поставщиков системы UNIX.

Системе UNIX посвящено несколько монографий [4, 9, 15, 59, 70, 109, 114].

Система UNIX интересна прежде всего как образец для создания машинно-независимых операционных систем для сетей из разнотипных ЭВМ. Так, в версии UNIX BSD 4.2 реализованы протоколы межсетевого взаимодействия, что позволяет совместно использовать различное оборудование и различное сетевое программное обеспечение.

## Достоинства и недостатки системы UNIX

### *Достоинства*

### *Недостатки*

1. Язык заданий shell удобен. Он краток, не требует переделок заданий при изменении конфигурации, синтаксически похож на языки программирования высокого уровня (Алгол 68).
2. Единая ОС на разнотипных ЭВМ позволяет устранить переобучение программистов.
1. Если программист поработает в системе UNIX, то он не захочет возвращаться к другим операционным системам.
2. Система UNIX требует много дисковой памяти; ее не сэкономили при создании системы.

3. Упрощается сопровождение (не нужно по сектору сопровождения для каждого типа ЭВМ)
4. Переносимость программ между разнотипными ЭВМ повышается, но не намного.
5. Может быть взят за основу создания операционной системы локальной сети разнотипных ЭВМ.
6. Сравнительно легко включать новые компоненты, прежде всего драйверы новых устройств любого типа; добавить новую команду вообще может каждый.
7. Небольшой объем документации (ее может перенести один человек за один раз).
3. Файловая система все еще ненадежна; при наличии поврежденного участка на диске работать невозможно. Требуется регулярное копирование и другие административные действия.
4. Ядро не имеет модульной структуры, драйверы и сетевые примитивы не ортогональны остальной части ядра. Как следствие ядро занимает большую резидентную память.
5. Фактически нет хорошей версии UNIX для 16-разрядных ЭВМ.
6. В системе UNIX нет средств программирования реального времени.
7. Все компиляторы и другие языковые средства написаны по одно- или двух-просмотровой схеме, поэтому при их работе требуется много оперативной памяти, что замедляет диалог других процессов с пользователями.

Нет точного определения, что такое UNIX, имеются независимо развиваемые версии. Тем не менее все они имеют некоторые общие принципы и общие интерфейсы.

## § 1. ОБЩЕЕ ОПИСАНИЕ СИСТЕМЫ UNIX. ФАЙЛОВАЯ СИСТЕМА

Система UNIX состоит из следующих функциональных частей:

**Ядро** — программа, управляющая распределением ресурсов ЭВМ. При работе системы UNIX ядро всегда находится в оперативной памяти. Отдельные функции ядра (образующие интерфейс ядра) доступны посредством системных вызовов.

**Команды** — программы, исполняющие специфические задачи пользователя.

Shell — интерпретатор команд, осуществляющий взаимодействие пользователя с системой UNIX. Вызывается автоматически при входе пользователя в систему. Shell ожидает ввода команд с терминала, затем анализирует эти команды (им передает параметры) и исполняет их. На языке интерпретатора команд shell могут создаваться новые команды, называемые командными файлами.

Командные файлы — текстовые файлы, содержащие имена программ (команд) и управляющие конструкции языка shell. Отдельные имена могут подаваться в диалоге пользователем при исполнении командного файла. Эти имена описываются в командном файле как параметры.

Подпрограммы — последовательности машинных команд, исполняющие специфические функции. Подпрограммы не могут исполняться самостоятельно, а могут использоваться только как части программ. Одна и та же подпрограмма может одновременно использоваться в нескольких программах. Подпрограммы доступны по именам (аналогично описываемым в программе функциям).

Библиотеки — наборы функционально близких подпрограмм, хранящиеся в специально организованных архивных файлах. В частности, имеются библиотека функций обмена и библиотека математических функций.

Языки — языки программирования С, Паскаль, Фортран и другие. Компилятор с каждого языка вызывается отдельной командой. Каждый язык использует некоторые библиотеки. Компилятор транслирует текстовый файл в двоичный файл, который может быть запомнен и исполнен.

Обычные файлы — содержат программы и данные (в двоичном или текстовом виде).

Специальные файлы — описывают атрибуты внешних устройств (сетевые протоколы или драйверы). При вводе или выводе специального файла происходит обращение к фактическому устройству.

Каталог — обычный файл, содержащий ссылки на файлы. С помощью каталогов образуется иерархия файловой системы.

Команды и библиотеки размещаются в обычных файлах на диске. Как правило, команды и подпрограммы размещаются в следующих каталогах: *bin* (наиболее часто используемые команды), *lib* (наиболее часто используемые подпрограммы), *usr/bin* (менее часто используемые команды), *usr/lib* (менее часто используемые подпрограммы), *etc* (команды администратора). В версиях университета Беркли (BSD 4.1 и BSD 4.2) команды содержатся также в каталоге *usr/ucb*.

Специальные файлы располагаются в каталоге *dev*. Временные файлы системы хранятся в каталоге *tmp*. Файлы пользователей хранятся в зависимости от условий эксплуатации в

каталоге *users* или в каталоге *usr/usrarc* или еще в каких-нибудь каталогах.

Таким образом обычно начальное дерево иерархии файловой системы имеет вид:

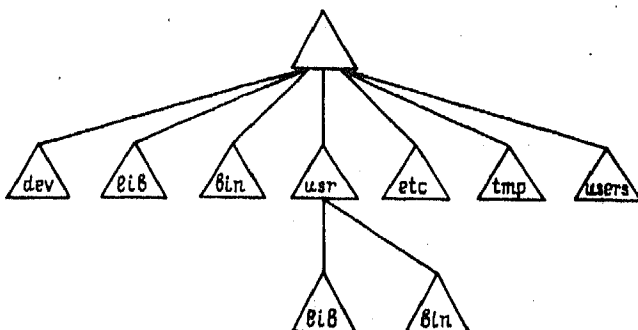


Рис. 1

После загрузки системы на каждом подключенном терминале появится приглашение

*login:*

Пользователь должен ввести имя, которое присвоено ему администратором системы и, быть может, пароль. Далее можно исполнять команды. Чтобы создать новую программу под именем *t.c*, нужно войти в редактор *ed t.c*.

Компиляция осуществляется командой *cc t.c*

Объектный код помещается в файл *a.out*, поэтому исполнение полученной программы (без параметров) осуществляется командой

*a.out*

Для выхода из системы используются команды *login*, *logout*, *bye* или *ctrl-D* (в зависимости от версии).

Таким образом, иерархия файлов в системе UNIX строится посредством каталогов (как промежуточных вершин) с корневым каталогом (обозначается литерой */*) и листьями — обычными или специальными файлами. Каталог при каждом имени файла содержит ссылку на четверку: тип файла, имя владельца, имя группы владельца, ссылка на содержимое файла *>*. Тип файла задает его класс (каталог, простой файл, специальный файл) и режим использования: девять разрядов — три разряда для пользователей, три разряда для группы, в которую входит пользователь, и три разряда для всех остальных. В каждой тройке разрядов первый определяет возможность записи, второй — возможность чтения и третий — возможность исполнения (например, восьмеричная запись режима *777* обозначает возможность всех действий, а *755* возможность всех действий

для владельца и возможность чтения и исполнения для всех остальных).

В каждом каталоге (кроме корневого каталога) содержится ссылка на объемлющий каталог (обозначается двумя точками) и на себя (обозначается точкой). Один из каталогов считается основным каталогом пользователя, а другой является текущим. При входе в систему текущим является основной каталог пользователя. Файлы текущего каталога доступны по имени, а файлы остальных каталогов — по косвенному имени, которое строится (через разделитель /) из каталогов на пути от текущего каталога до файла. Например, на рис. 1 из каталога *lib* можно адресоваться по имени *../bin/lisp* к вызову Лисп-интерпретатора. Можно также строить косвенное имя, начиная от корневого каталога, тогда последнее косвенное имя было бы */usr/bin/lisp*. Косвенное имя от корневого каталога называется абсолютным, а от текущего каталога — относительным.

Имеются следующие команды для работы с файловой системой:

- cd* — перейти в основной каталог пользователя;
- cd d* — перейти в каталог *d*;
- chmod r f* — изменить режим защиты файла *f* на режим *r*;
- chown i f* — изменить имя владельца файла *f* на имя *i*;
- chgrp i f* — изменить имя группы файла *f* на имя *i*;
- cmp f1 f2* — сравнить файлы *f1* и *f2*;
- cat f1 f2 ... fn* — слить файлы *f1, ... fn* в один файл и выдать его на стандартный вывод;
- cp f1 f2* — копировать файл *f1* в файл *f2*;
- grep w f* — поиск строчек в файле *f*, содержащих слово *w*;
- file f* — определить тип файла *f*;
- ls d* — выдать список имен каталога *d* (на стандартный вывод);
- ls -l d* — выдать список имен и их типов в каталоге *d*;
- mkdir d* — создать новый пустой каталог с именем *d*;
- mv f1 f2* — переименовать файл *f1* в файл *f2*;
- od f* — выдать восьмеричный дамп файла *f*;
- print f* — печать файла *f*;
- pwd* — выдать имя текущего каталога (на стандартный вывод);
- rm f* — удалить файл *f*;
- rmdir d* — удалить каталог *d* (если он пустой!);
- wc f* — определить число символов, слов и строчек в файле *f*;

Многие команды имеют режимы (например, команда *ls -l*), уточняющие их выполнение. Особенно разнообразны режимы команды *dd* (переслать файл с преобразованием), *find* (поиск файла), *sort* (сортировка файла), *tar* (работа с ленточным архивом), *test* (проверка типа файла и других условий).

Описание любой команды можно получить на экране терминала с помощью команды

*man команда*

описание режимов работы команды *man* также можно получить на экране терминала, набрав командную строчку

*man man*

(имеется режим поиска по ключевому слову в описании команды). Команда *man* полезна даже опытному программисту, т. к. современные версии системы UNIX имеют сотни команд с многочисленными режимами.

Выше указанных знаний достаточно, чтобы начать работать в системе UNIX для пользователя, знающего язык C или какой-нибудь другой из имеющихся в UNIX. Редакторы текстов в системе UNIX аналогичны редакторам в других операционных системах. В зависимости от типа терминала могут применяться строчные, экранные и полиэкранные редакторы текстов.

## § 2. ЯЗЫК ЗАДАНИЙ SHELL. КОМАНДА MAKE

Язык заданий shell в различных версиях системы UNIX различен, но по мере совершенствования его основные конструкции унифицируются. Язык shell реализуется диалоговым интерпретатором, который считывает командные строки и интерпретирует их. Язык shell применяется для взаимодействия с системой UNIX и для составления новых команд (т. е. как язык программирования). Основной конструкцией языка shell является команда (с возможными параметрами и переназначениями обменов). Команда продолжается до конца строки или до специального разделителя.

При описании языка shell будем различать последовательности команд и задания, которые кроме команд могут содержать управляющие конструкции (циклы, условные команды и т. д.). Последовательность команд будем называть простой, если она не содержит параметров.

Если задание помещено в некотором файле *f*, то его можно исполнить командой *sh f* или, если предварительно исполнить команду *chmod 755 f*, которая делает файл исполняемым, то можно исполнить просто *f*. Задание (команда) может иметь параметры трех типов: общие, позиционные и специальные. Имя общего параметра это последовательность букв и цифр, начинающаяся с буквы (здесь символ подчеркивание относится к буквам!).

Общему параметру может быть присвоено значение

*parameter=value*,

где *value* — любая строка или заключенная в символы обратного ударения команда. При использовании общего параметра ему предшествует литера \$ или  $\$$ , обозначающая разымено-

вание. Например, последовательность команд

```
d = `pwd`  
cd ..  
f = /usr/ f1  
cd $d  
echo f = $f  
cp $f f2
```

выдает сообщение  $f = /usr/ f1$  (команда *echo* просто выдает на стандартный вывод свои аргументы) и осуществляет пересылку файла *f1* из каталога *usr* в файл *f2* текущего каталога. Общий параметр можно использовать в качестве подслово, тогда он заключается в фигурные скобки.

Например,  $\${f}5$  будет обозначать  $/usr/ f 15$ .

Позиционные параметры  $\$0, \$1, \dots, \$9$  получают свои значения при вызове команды с параметрами (параметры отделяются от команды и последующих параметров пробелами). Например, при вызове

```
cc t1.c t2.c t3.c
```

$\$0 = cc, \$1 = t1.c, \$2 = t2.c, \$3 = t3.c,$

остальным присваивается пустая строка. Позиционных параметров может быть более десяти, но десятый и последующие параметры остаются недоступными до исполнения нужного числа раз команды *shift*, которая осуществляет присваивания  $\$1 = \$2, \$3 = \$4$  и т. д.,  $\$9 =$  <первый из недоступных параметров>, и т. д.

Специальные параметры (автоматически устанавливаемые):

$\#$  — число непустых позиционных параметров (включая недоступные);

— режимы, установленные при вызове или специальными командами; представляется строкой (для каждого режима есть своя представляющая литера);

? — десятичное число, выданное последней синхронной (не фоновой) командой;

\$ — идентификатор процесса исполняемой текущей процедуры;

! — номер процесса последней асинхронной (фоновой) команды.

Этим параметрам нельзя присваивать значения. Имеются также системные параметры: HOME, PATH, SHELL, MAIL, TZ, PS1, PS2 и IFS.

Имеются четыре конструкции для изменения параметров:

$\{parameter:—word\}$  — если *parameter* установлен и ненулевой, то используется его значение, иначе используется значение *word*;

$\{parameter:=word\}$  — если *parameter* не установлен или он нулевой, то ему присваивается *word*; во всех случаях ис-

пользуется его (новое) значение, такая установка не применима к позиционным параметрам;

`{parameter:?word}` — если *parameter* установлен и он ненулевой, используется его значение; в противном случае, печатается *word* и завершается текущий процесс; если *word* опущено, печатается сообщение «параметр нулевой или не установлен»;

`{parameter:+word}` — если параметр установлен и он ненулевой, используется *word*, иначе используется нулевое значение. В некоторых версиях двоеточие после *parameter* не ставится.

Примеры:

`echo ${1:-`pwd`}` если параметр `$1` установлен, то выдается его значение, иначе выдается текущий каталог;

`pr {s:=*}>dev/lp` если *s* установлен, то печатается файл (или файлы) с соответствующим именем, иначе печатаются все файлы текущего каталога (\* обозначает любую последовательность букв и цифр);

`cd ${arg:?«not set»}` если *arg* не установлен, то команда *cd* не исполняется: а печатается «*arg*:not set»;

`{s:+} echo cd $s` если *s* установлен, то исполняется команда `cd $s`; здесь приходится использовать команду `echo` из-за недоработки в программе `shell`; команда `echo` выдает последующую строчку на стандартный вывод (обычно на терминал, а здесь на исполнение).

Имеются четыре литеры, которые используются для выделения других литер, это обратная черта (`\`), одиночная кавычка (`'`), двойная кавычка (`"`) и обратное ударение (```).

Обратная черта используется перед специальными управляющими литерами, чтобы они рассматривались как обычные литеры. К специальным относятся следующие: `?*[]\`"'"`&`; `( ) < > { }` и литера «новая строчка». Обратная черта перед «новой строчкой» приводит к тому, что «новая строчка» игнорируется.

Строка, ограниченная одиночными кавычками, рассматривается как строка литер, включая специальные литеры (кроме одиночной кавычки), которые также трактуются как литеры.

Строка, ограниченная двойными кавычками, рассматривается как строка литер, исключая специальные литеры (перед ними нужно ставить обратную черту).

Двойные и одиночные кавычки используются в команде `echo`, чтобы не осуществлялась подстановка параметров, или в присваиваниях параметрам, чтобы можно было использовать в значениях параметров пробелы и специальные литеры.

Обратное ударение ограничивает строку, которая рассматривается как задание; это задание выполняется, и его выдача подставляется вместо исполнявшейся строки.

Литера # используется как начало примечания, продолжающегося до конца строчки.

Важным средством программирования является переназначение обмена, которое может иметь следующие формы:

- 1) команда  $> f1$
  - 2) команда  $< f2$
  - 3) команда  $\gg f1$
  - 4) команда  $\ll$ ограничитель
- ⋮  
ограничитель.

В первом случае стандартный вывод команды заменяется на файл  $f1$ , причем файл  $f1$  опустошается; во втором случае стандартный ввод заменяется на файл  $f2$ ; в третьем случае стандартный вывод команды заменяется на файл  $f1$ , и этот файл продолжается (производится запись в конец файла); в четвертом случае в качестве стандартного ввода рассматриваются строчки, ограниченные ограничителем (любым словом), причем, если перед первым ограничителем поставить минус, то в них будут игнорироваться начальные табуляции. Команда может иметь параметры и другие переназначения обмена.

Примеры:

$date > f$  # занесение даты в файл  $f$ ;  
 $ed z < r$  # редактирование файла  $z$  редактирующими командами из файла  $r$ ;  
 $banner abc > /dev/lp0$  # мозаичная печать букв  $abc$ .

Отличительной конструкцией языка shell является магистраль (pipe), обозначаемая литерой |. Магистраль связывает стандартный вывод левой команды (или магистрали) со стандартным вводом правой.

Например:

$ls | wc$  # выдача числа файлов в текущем каталоге.

На одной командной строчке может располагаться несколько команд. Для разделения команд или магистралей используются следующие разделители:

; — команды выполняются последовательно;

& — команды выполняются асинхронно (т. е. левая команда становится фоновой);

&& — если успешно выполняется левая команда, то выполняется правая;

|| — если левая команда не успешна, то выполняется правая.

Команды (и магистрали) могут группироваться с помощью следующих конструкций:

круглые скобки — выделяют подзадание, которое запускается как отдельный процесс (рекурсивно вызывается shell) и считывается одной командой;

фигурные скобки — используются для переназначения обмена подзаданию (новый процесс не создается);

*for* имя  
*in* список слов  
*do* список команд  
*done*

*case* \$ имя *in*  
(образцы)  
список команд;;  
:  
*esac*

*if* список команд1  
*then* список команд2  
*elif* список команд3  
*then* список команд4  
:  
*else* список команд  
*fi*

*while*  
список команд1  
*do* список команд2  
*done*  
*until* список команд1  
*do* список команд2  
*done*

- # имени в цикле присваивается очередное слово из списка слов и для него выполняется список команд;
- # может отсутствовать часть «*in* список слов», тогда в качестве списка слов используются все установленные позиционные параметры;
- # в списках команд имя используется как общий параметр;
- # имя — это разыменовываемый позиционный параметр;
- # образец — это строка литер и специальных литер \*, ?, [, ] и \, образцы разделяются литерой | и завершаются точкой с запятой;
- # две точки с запятой используются для завершения списка команд;
- # при исполнении имя сравнивается с образцами и при успешном сравнении исполняется соответствующий список команд;
- # строчки *elif* и *else* могут отсутствовать;
- # исполняется список1, если его последняя команда успешна, то исполняется список2 иначе список3 и далее аналогично, если последняя команда в списке3 успешна, то исполняется список4 иначе список5 и т. д.;
- # исполняется список1, если его последняя команда успешна, то исполняется список2 и цикл повторяется;
- # исполняется список1, если его последняя команда не успешна, то исполняется список2 и цикл повторяется.

Конструкции группирования в различных версиях языка shell могут отличаться. Как правило они аналогичны соответствующим конструкциям языка Алгол 68.

В вышеуказанных командах и конструкциях исполняются имена, которые строятся из букв, цифр, позиционных и общих параметров и специальных литер. Специальные литеры обозначают следующее:

? — одна любая не специальная литера;

\* — любая последовательность не специальных литер (возможно пустая);

[последовательность литер] — любая литера из последовательности;

[литера 1 — литера 2] — любая литера от литеры 1 до литеры 2 в лексикографическом порядке.

Если присутствуют указанные специальные литеры, то имя обозначает множество имен файлов и является образцом.

Заметим, что квадратные скобки могут использоваться также для обозначения команды *test*, т. е. записи

*test* выражение

и

[выражение]

эквивалентны.

Здесь выражение может быть таким:

- *r* имя # истинно, если файл с таким именем существует и читаем;
- *w* имя # истинно, если файл существует и в него можно писать;
- *f* имя # истинно, если файл существует и не является каталогом;
- *d* имя # истинно, если файл существует и является каталогом;
- *s* имя # истинно, если файл существует и не пуст;
- *t* дескфайла # истинно, если дескфайла это терминал;
- *t* # истинно, если *l* — это дескриптор файла;
- *z s* # истинно, если длина строки *s* нулевая;
- *n s* # истинно, если длина строки *s* ненулевая;
- s1 = s2* # истинно, если строки *s1* и *s2* равны;
- s1 != s2* # истинно, если строки *s1* и *s2* неравны;
- s1* # истинно, если длина строки *s1* ненулевая;
- n1 =op n2* # истинно, если числа *n1* и *n2* равны;
- n1 >op n2* # истинно, если *n1 op n2*, где *op* это *ne*, *gt*, *ge*, *lt* или *le* — операции сравнения;

из указанных элементарных выражений могут строиться более сложные с помощью операций:

- ! выражение # взятие отрицание;
- выражение 1 — *a* выражение 2 # логическое и;
- выражение 1 — *o* выражение 2 # логическое или;
- приоритет операции — *a* выше, чем операции — *o*, для изменения порядка действий можно использовать круглые скобки.

Пример цикла:

```
for i in bin etc usr/bin ucb games  
do cd i; make install done
```

# в каждом из каталогов из списка выполняется  
# команда *install*

Интересное расширение включено в язык C-shell [122], синтаксис конструкций группирования которого близок к языку C. Имеется команда *history*, которая показывает все выполненные ранее команды с их порядковыми номерами. Далее можно использовать дополнительные общие параметры.

! номер команды # вызывает повторное исполнение команды с указанным номером;  
!! # вызывает повторное исполнение последней команды.

Возможно также исполнение предыдущих команд с модификацией. Основное отличие языков C-shell и shell состоит в том, что язык C-shell в списке команд после последней команды ставится точка с запятой, а в языке shell не ставится.

В период отладки программа как правило состоит из нескольких файлов, которые могут компилироваться независимо. Кроме того, вспомогательные файлы могут включаться в основные файлы с помощью управляющих строчек #*include*. При исправлении одного или нескольких файлов требуется частичная перекомпиляция.

Зависимости файлов могут быть представлены правилами следующего вида:

список-имен: требуемые-файлы; команды.

Команды могут быть продолжены на следующих строчках, но эти строчки должны начинаться с табуляции. Правила зависимостей разрабатываемого комплекса программ располагаются в отдельном файле, как правило под именем *makefile*.

В системе UNIX при каждом файле хранится время его последнего изменения. Пользуясь правилами зависимостей и этими временами, команда *make* может провести нужные перекомпиляции и другие действия (причем время последнего изменения еще не созданного файла считается текущим временем).

Команда *make* может иметь два параметра. Первый задается одним из следующих вариантов:

—*f* имя файла с правилами зависимостей,

—*j*.

Во втором случае правила зависимостей вводятся со стандартного ввода.

По умолчанию первый параметр это *makefile* из текущего каталога. Второй параметр — имя анализируемого (создаваемого) файла. По умолчанию — это первое имя в первом правиле зависимости.

Анализируемый файл ищется в списках имен правил зависимости. Если оказывается, что этот файл создан ранее одного из файлов, от которых он зависит (а каждый из последних в свою очередь изменяется, если он создан ранее одного из файлов, от которых он зависит и т. д.), то выполняются команды из найденного правила зависимости. Имеется несколько правил зависимостей по умолчанию, например, *x.o* зависит от *x.c*.

Пример, пусть *makefile* это

```
prog: x.o y.o z.o; cc x.o y.o z.o
mv a.out prog
x.o y.o: defs
```

# *x.o*, *y.o* и *z.o* создаются соответственно из *x.c*, *y.c* и *z.c* по умолчанию, если последние созданы позднее.

Тогда команда *make* после изменения любого из файлов *x.c*, *y.c*, *z.c* или *defs* произведет нужные команды и создаст новый вариант для *prog*, а *make x.o* при необходимости нужный вариант для *x.o*.

В списке имен правил зависимостей могут быть не только имена имеющихся или создаваемых файлов, но и произвольные имена, которые можно понимать, как имена действий, например:

*test*:

```
prog > test 1 # запуск prog и запись результата
                в test 1;
```

```
print: test; print test 1
```

```
rm test 1 # печать файла test 1, если он существует.
```

Перед правилами зависимостей могут располагаться макроопределения:

имя = текст

а в правилах \$ (имя) всюду заменяется на текст.

По умолчанию определены макроимена:

? — множество найденных имен, которые «моложе» заданного;

< — множество имен, к которым применялись неявные правила;

\* — множество всех текущих и зависимых имен файлов.

Обычно выполняемые в команде *make* строчки команд печатаются. Чтобы подавить печать, перед строчкой команд ставится литера @.

Если нет подходящего явного или неявного правила зависимости, то выполняется правило с именем *default*, а если такого нет, то исполнение завершается и печатается сообщение об ошибке.

Команда *make* имеет уточняющие режимы.

Команда *take* перенесена с некоторой модификацией в операционную систему VAX/VMS и называется там MMS и используется совместно с системой CMS, поддерживающей использование библиотек.

В системе UNIX команда *take* неизменна с 1978 года.

### § 3. ЯЗЫК ПРОГРАММИРОВАНИЯ C

Язык программирования C [61] является машинно-независимым процедурным языком программирования. Он был реализован в рамках машинно-независимой операционной системы UNIX, последующие версии системы писались на языке C, что обеспечило ее мобильность. Имеются также реализации языка C в рамках операционных систем RSX, VMS, CP/M и других.

Язык C используется в основном для задач системного программирования, но пригоден также для написания больших вычислительных программ, программ обработки текстов, проектирования, управления оборудованием и других целей.

Стандарта на язык C нет, он продолжает развиваться. Многие идеи языка C происходят из языков BCPL и B. Однако языки BCPL и B не используют типов данных (кроме машинных слов). В языке C базисными типами являются литеры, целые и вещественные, а иерархия типов строится посредством имен, структур, массивов, объединений и функций. Использование сложных типов данных повышает эффективность и надежность программирования, хотя контроль за использованием типов не столь строгий как в языках Алгол 68, Ада или Паскаль. Имеющиеся компиляторы с языка C не обеспечивают даже предусмотренных в языке C средств контроля типов: для более полного контроля используется программа *lint*.

Язык C имеет управляющие конструкции, необходимые для создания структурированных программ (циклы, условные, переключатели и т. д.), однако операторы перехода не исключены. Допускаются арифметические операции над адресами.

К любой функции можно обращаться рекурсивно, но описание одной функции не может содержаться внутри другой.

Язык C, как и любой другой имеет свои недостатки. Синтаксис некоторых конструкций неудобен, имеются приведения только скалярных типов данных.

Тем не менее популярность языка C возрастает. Имеются сведения [108], что в фирме Pilon производительность труда программистов на языке C составляет 15—35 строчек в час, в то время как обычно производительность труда программистов значительно ниже.

**3.1. Неформальное описание основных конструкций языка C.** Операции языка C имеют дело со скалярными значениями. Составные значения используются для упорядочения доступа к их элементам. Распределение памяти статическое. Для обменов

с внешними устройствами используются системные вызовы и библиотечные функции. Управляющие конструкции традиционны. Имеется препроцессор.

В языке используются следующие скалярные типы данных (для переменных и констант):

*int* (целое)

*long* (длинное целое)

*short* (короткое целое, используется редко)

*unsigned* (беззнаковое целое)

*float* (вещественное)

*double* (длинное вещественное)

*char* (литерное).

Литера помимо своего основного назначения может использоваться в тех же случаях, что и целое.

Примеры констант:

*int*: 15 (десятичное) 07 (восьмеричное) 0xA2 (шестнадцатичное)

*long*: 15L 07L 0xA2L

*float*: 1.23 1.23e-2 2e3

*double*: 1.23 1.23e-2 2e3

*char*: 'a' '\177' '\n' (новая строка)

Переменная — это ссылка из программы на область памяти для хранения значения некоторого типа.

Примеры описания переменных скалярных типов:

*int* *x*; *char* *a*, *b*, *c*;

Имеются следующие производные типы данных:

Статические одномерные или многомерные массивы (нумерация элементов начинается с 0). Пример описания переменных, являющихся массивами:

*int* *x* [10], *y* [20];

*int* *w* [10] [20]; *char* *a* [5];

и использование этих переменных

*x* [0] = 1; *a* [0] = 'a';

здесь переменная *x* обладает именем на массив из 10 элементов, а *x* [0] — это переменная, текущее значение которой является нулевым значением массива *x*.

Имена («имя типа» указывает на область памяти, достаточную для хранения значения данного типа). Пример описания переменной типа «имя целого»

*int* \**xx*;

и использование этой переменной

(в зоне действия описания *int* *x* [10];)

*xx* = &*x* [0]; \**xx* = 1;

здесь операция & преобразует переменную в имя, указывающее

на значение переменной, а операция \* преобразует имя в переменную, текущим значением которой является значение, хранящееся в области памяти, на которую указывает имя.

Отличие имен от переменных состоит в том, что в выражении вместо переменной подставляется текущее значение переменной, а имя само является значением (адресом), над которым выполняются операции (в частности, арифметические). Кроме того переменной можно присваивать, а для изменения значения, на которое указывает имя, нужно предварительно это имя преобразовать в переменную. Таким образом,  $*xx=1$  эквивалентно  $x[0]=1$ , а  $x[i]$  эквивалентно  $*(x+i)$ .

Структуры, т. е. значения, составленные из нескольких значений определенных типов (полей).

После описания

```
struct compl {float re, im;} z;
```

шаблон *compl* обладает структурным типом, составленным из двух вещественных полей, доступ к которым определяется указателями *re* и *im*, а *z* — это переменная структурного типа *compl*.

Может быть описано «имя типа *compl*»

```
struct compl *zz;
```

Выборка *z.im* создает переменную, ссылающуюся на второе поле структуры *z*, а выражение  $z \rightarrow re$  создает переменную, ссылающуюся на первое поле структуры, именуемой *zz*. Выражение  $zz \rightarrow re$  эквивалентно  $(*zz).re$ . Таким образом, после присваиваний

$$z.im=2; zz=z; zz \rightarrow re=1;$$

переменная *z* будет ссылаться на структуру со значениями полей 1.0 и 2.0, а переменная *zz* будет ссылаться на имя, указывающее на эту структуру.

Поля структур могут быть массивами, именами или структурами, а также нижеописываемыми объединениями.

Объединение типов используется в тех случаях, когда переменная в различные моменты исполнения должна ссылаться на значения разных типов. После описания

```
union {int a; float b;} ab;
```

переменная *ab* ссылается на область памяти, достаточную для размещения как целого, так и вещественного. Выборка *ab.a* преобразует эту переменную в переменную, ссылающуюся на то же значение, но рассматриваемое как целое, а переменная *ab.b* имеет тип «вещественное».

В языке С над структурами и объединениями кроме выборки не определено других действий (некоторые компиляторы с языка С предусматривают присваивание структур).

Нет констант производных типов кроме массивов литер (строк). Строки изображаются последовательностью литер в

двойных кавычках. В конце строки компилятор подставляет литеру с кодом 0. Таким образом длина строки "abc" равна 4.

Основной конструкцией языка С является функция. Пример описания функции:

```
sum(x, y)
{return(x+y);}
```

По умолчанию тип параметров и тип результата — целые.

Другой пример:

```
double sin(x)
double x;
```

{вычисление синуса}

Примеры вызовов:

```
sum(1,2); sin(0.5).
```

Описания переменных могут быть глобальными (в начале программы) или локальными (внутри функции или блока). Однако слово *static* перед локальным описанием сохраняет значение переменной до следующего вызова функции.

Программа состоит из глобальных описаний переменных и функций. Среди функций должна быть функция *main*, с которой начинается исполнение программы.

Описание функции имеет следующую форму:

```
тип результата
название функции
(возможный список параметров)
типы параметров;
{
локальные описания
операторы
}
```

Параметр — это переменная (используемая в функции), которой при вызове присваивается значение соответствующего аргумента. При каждом вызове функции исполняются локальные описания, а затем операторы.

Язык С имеет следующие операторы.

Выражение, после которого ставится точка с запятой (даже в конце блока!), чтобы показать, что вырабатываемое выражением значение игнорируется.

Выражения в языке С формируются из изображений, идентификаторов переменных, функций и операций. Операции выполняются в соответствии с приоритетом. В таблице (см. ниже) группы операций расположены по убыванию приоритета.

При необходимости можно изменить порядок выполнения операций, заключив некоторые подвыражения в скобки.

Условный оператор:

```
if(a)b или if(a)b else c
```

отличается от условной операции

```
a?b:c
```

название	операции	ассоциативность
первичные унарные	( ) [ ] --> . ! ~ + + - - * &	слева направо справа налево слева направо
бинарные:		
мультипликативные	* / %	
аддитивные	+ -	
сдвиги	<< >>	
отношения	< <= > >=	
равенства	= = !=	
поразрядное и исключающее или	&	
поразрядное или	^	
логическое и	& &	
логическое или		
условная	? :	справа налево
присваивания	= + = -= *= /= %= >>= <<= & =	справа налево
запятая	^ = -= ,	слева направо

тем, что не вырабатывает значение. В условном операторе, условной операции и других местах условие считается истинным, если оно выдает значение, отличное от нуля.

Циклы:

*while* (*a*) *b*

выполняется *a*, если оно истинно, то выполняется *b* и цикл повторяется, иначе цикл завершается

*for* (*a1*; *a2*; *a3*) *b*;

который эквивалентен последовательности операторов

*a1*; *while* (*a2*) {*b*; *a3*};

Пример:

*for* (*i*=1; *i*<10; *i*++) *x*[*i*]=0;

переменная *i* должна быть описана; для языка С характерна операция *i*++, увеличивающая значение *i* на 1.

Завершители и переходы:

Оператор *break*; прекращает выполнение объемлющего цикла или переключателя.

Оператор *continue*; осуществляет переход на начало объемлющего цикла.

Оператор *return*; завершает выполнение функции, а оператор *return x*; завершает выполнение функции с выдачей значения *x*. Завершить выполнение программы можно системным вызовом *exit* (0);.

Оператор *goto l*; осуществляет переход к оператору, помеченному меткой *l*..

Переключатель (пример):

*int* *s*; *double* *x*, *y*; *x*=0.5;

```

switch (s)
{case 1: case 3: y=sin(x); break;
 case 2: y=cos(x); break;
 default: y=0;
}

```

переменная  $y$  при  $s=1$  или  $s=3$  получает значение  $\sin(x)$ , при  $s=2$  значение  $\cos(x)$ , а в других случаях значение 0.

Главное, не забыть оператор *break*; или другой завершитель в конце альтернативы переключателя (иначе начнет исполняться следующая альтернатива).

Неотъемлемой частью технологии программирования на языке С является препроцессор. Основные возможности препроцессора отражены в примерах:

```
#define x10 x[10]+1
```

далее в программе слово  $x10$  будет всюду текстуально заменяться на  $x[10]+1$

```
#define sum(x, y) x+y
```

далее в программе каждое вхождение  $sum(a, b)$  будет заменяться на  $a+b$  при любых  $a$  и  $b$

```
#include "f1"
```

вместо этой строчки будет вставлен файл с именем  $f1$  из каталога, в котором содержится программа (если его там нет, то поиск продолжается в специальных каталогах

```
#include <f2>
```

вместо этой строчки будет вставлен файл с именем  $f2$  из специальных каталогов (*usr/include* и других).

Для написания многих программ достаточно трех библиотечных функций обмена:

*putchar* ( $x$ ) — вывод литеры  $x$  (по стандартному каналу вывода);

*getchar* ( ) — ввод одной литеры (со стандартного ввода) и выдача ее в качестве значения функции;

*printf* ("формат.",  $x1, \dots, xn$ ) — форматный вывод (по стандартному каналу вывода).

Управляющим литерам формата предшествует символ %, за которым может следовать целое — число выводимых символов. Наиболее часто используются управляющие литеры:

%d — вывод десятичного целого;

%s — вывод строки;

%e — вывод вещественного с порядком.

Неуправляющие литеры формата просто выводятся, литера, вызывающая переход на новую строку, изображается \n.

Программа на языке С должна быть создана (например, с помощью редактора) в файле с некоторым именем  $x.c$  (суффикс)  $.c$  обязателен). Компиляция осуществляется командой:

```
cc x.c
```

при отсутствии ошибок создается исполняемый код в файле с

именем *a.out*. Для исполнения программы (без параметров) нужно набрать

```
a.out
```

В качестве первой рекомендуется запустить программу:

```
main ( )  
{printf ("test\n");}
```

Программа может иметь строковые параметры. В этом случае параметры пишутся после имени команды через пробелы: *f a1...an*

Текст программы для команды *f* должен содержать функцию *main* с двумя параметрами — счетчиком числа параметров и массивом переданных параметров:

```
main (c, v)  
int c; char *v [ ];  
{...}
```

при исполнении команды *c* получает значение *n+1*, а *v* имеет массив: («имя исполняемого файла», «параметр 1»,... «параметр *n*»), т. е. в нашем случае эквивалентная программа без параметров выглядела бы так:

```
main ( )  
{char *v [ ] = {"f", "a1", ..., "an"}; int c = n; ...  
}
```

3.2. Примеры программ на языке C. Программа, определяющая «счастливый билет»

```
#define LOOP for (i=0; i<3; i++)  
main (c, v)  
char*v [ ];  
{int s1, s2; int i;  
#define s v[1]  
if (c1=2){printf ("несоответствие параметров \n"); return;}  
LOOP s1 = s1 + *s++;  
LOOP s2 = s2 + *s++;  
if (s1 == s2)  
printf ("программа %s сообщает, что билет счастливый \n"  
v [0]);  
else printf ("обычный билет \n");  
}
```

Если эту программу поместить в файл *ticket.c*, то исполнение команд

```
cc ticket.c  
a.out 123321
```

выдает на экран сообщение:

программа *a.out* сообщает, что билет счастливый.

В программе LOOP использовано для сокращенного обозначения заголовка цикла из трех шагов. Имя *v*[1] (т. е. макроним *s*) указывает на строку, подаваемую в качестве параметра команды. Операция *s++* осуществляет сдвиг по строке, а вы-

ражение `*s++` осуществляет сдвиг и выдает литеру, на которую указывало имя до сдвига. Таким образом, в `s1` накапливается сумма первых трех цифр, а в `s2` сумма трех последующих. Для сравнения используется операция `==`, т. к. просто `=` обозначает присваивание. Неправильное использование `=` является типичной ошибкой при программировании на С.

В нашем примере макроопределение для `s` вполне целесообразно поместить после локальных описаний. Однако в большинстве случаев программа имеет следующий вид:

```
#include <стандартный файл>
```

```
...
#include "файл пользователя"
```

```
...
#define имя текст
```

```
...
внешние описания
функции.
```

Следующий пример — организация списка

```
#define list struct t {struct t*s; int i;}
```

```
list x1, x2, x3;
```

```
x1.i=1; x2.i=2; x3.i=3;
```

```
x1.s=&x2; x2.s=&x3; x3.s=&x1;
```

таким образом будет организован список, показанный на рис. 2.

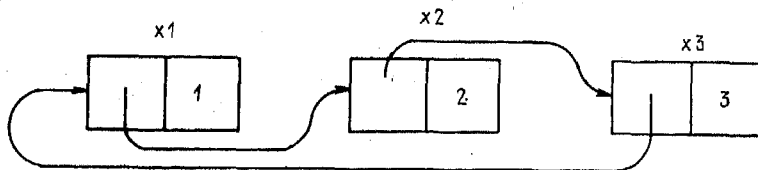


Рис. 2

который в дальнейшем может модифицироваться.

Идентификация полей структур в разных компиляторах с языка С устроена по-разному. Однако во всех случаях идентификаторы разбиваются на независимые классы: переменные, идентификаторы структур, поля структур, поля объединений и другие. В ранних версиях компиляторов с языка С требуется, чтобы в каждом блоке все доступные поля всех структур имели различные идентификаторы. При этом в выборках не контролировалось соответствие типов структуры и поля, а тип результата определялся полем. Так, при написании драйверов часто использовалась конструкция преобразования целого в адрес:

```
#define ADDR 0175610
```

```
struct {int lp; int buf;};
```

```
ADDR->lp=0;
```

```
ADDR->buf=1;
```

таким образом, *ADDR* начинает трактоваться как имя структуры, заданное физическим адресом.

В последних версиях поля, входящие только в одну структуру, можно для совместимости со старыми компиляторами применять бесконтрольно, но о несоответствии типов выдается предупреждающее сообщение. Допустимо использование одного и того же идентификатора поля в различных структурах. При выборке смысл поля определяется по типу структуры. Однако такие поля нельзя применять бесконтрольно. Пример: *struct{struct{int a1, a2;}m1, m2}p*; выборка *p.a1* ошибочна (она была бы двусмысленной), нужно писать *p.m1.a1* или *p.m2.a1*.

Для явного отключения контроля типов используется объединение. Пример:

```
union{int a; double b; char *c;}u;
```

переменная *u* ссылается на участок памяти, достаточный для размещения одного значения любого из указанных в полях типа.

Заключительный пример — упорядочивание массива методом Шелла

```
main( )
{static int a[ ] = {10, 9, 8, 7, 6, 1, 2, 3, 4, 5};
 int i;
 shell (a, 0);
 for (i=0; i<10; i++)
 printf ("%d\n", a[i]);
 }
 shell (v, n)
 int v[ ]; n
 {int gap, i, j, t;
  for (gap=n/2; gap<0; gap/=2)
  for (i=gap; i<n; i++)
  for (j=i-gap; j>0&&v[j]>v[j+gap]; j-=gap)
  {t=v[j];
   v[j]=v[j+gap];
   v[j+gap]=t;
  }
 }
```

Аналогичная программа на Фортране или на Паскале была бы вдвое длиннее.

В качестве дополнительных примеров рекомендуем задачи из [28].

**3.3. Синтаксические правила.** Синтаксис языка С описывается бесконтекстной грамматикой. Правила грамматики строятся из понятий и обозначений лексических классов. Для определения порядка исполнения конструкциям приписывается приоритет.

**3.3.1. Лексические классы.** Лексические классы (или просто классы) обозначаются словами из больших латинских и русских букв.

Классы разделяются на собственные, конечные, слова (идентификаторы) и изображения.

Собственный класс — это лексический класс, состоящий из одного слова — обозначения класса, записанного малыми буквами. Собственные классы — это: RETURN, GOTO, IF, ELSE, SWITCH, BREAK, CONTINUE, WHILE, DO, FOR, DEFAULT, CASE, SIZEOF и ENUM.

Конечные классы — это обозначения для классов, включающих конечное число лексем. Конечные классы — это: ПУСТО — пустое множество, ПЛЮС = {+}, МИНУС = {-}, ЗВЕЗДОЧКА = {\*}, И = {&}, ВОПРОС = {?}, ДВОЕТОЧИЕ = {:}, ИИ = {&&}, ИЛИИЛИ = {||}, СПРИСВАИВАНИЕМ = {=, +=, -=, \*=, /=, %=, >>=, <<=, ^=, |=, &=}, ОТНОШЕНИЕ = {<=, >=, <, >}, РАВЕНСТВО = {==, !=}, ДЕЛИТЬ = {/, %}, ИЛИ = {|}, ИСКЛЮЧЕНИЕ = {^}, СДВИГ = {>>, <<}, ПАРА = {++, --}, НЕ = {!, ~}, ПОЛЕ = {-, >, .}, ТИП = {int, char, long, float, double, unsigned, short}, ПАМЯТЬ = {extern, register, auto, static, typedef}, СТРУКТУРА = {struct, union}, СКОБКА1 = {(), {}}, КВС1 = {[], {}}, КВС2 = {[], {}}, ТЧЗАП = {;, {}}, ПРИСВОИТЬ = {=}, СТАР = {= +, = -, = \*, = /, = % = >>, = <<, = ^, = |} — операции 6 версии, а также ФС1 — левая фигурная скобка, ФС2 — правая фигурная скобка, и ЗАПЯТАЯ — символ запятая (теоретико-множественное изображение трех последних классов было бы двусмысленным).

3.3.2. Приоритеты и ассоциативность. Некоторые лексические классы обладают собственными приоритетами. Это классы (в порядке старшинства приоритета):

1. Левоассоциативный ЗАПЯТАЯ
2. Правоассоциативные СПРИСВАИВАНИЕМ, СТАР
3. Правоассоциативные ВОПРОС, ДВОЕТОЧИЕ
4. Левоассоциативный ИЛИИЛИ
5. Левоассоциативный ИИ
6. Левоассоциативный ИЛИ
7. Левоассоциативный ИСКЛЮЧЕНИЕ
8. Левоассоциативный И
9. Неассоциативный РАВЕНСТВО
10. Неассоциативный ОТНОШЕНИЕ
11. Левоассоциативный СДВИГ
12. Левоассоциативные ПЛЮС, МИНУС
13. Левоассоциативные ЗВЕЗДОЧКА, ДЕЛИТЬ
14. Правоассоциативный НЕ
15. Правоассоциативные ПАРА, SIZEOF
16. Левоассоциативные СКОБКА1, КВС1, ПОЛЕ

Начальным понятием для вывода в грамматике является слово «программа». Записанное большими буквами обозначение лексического класса в порожденной программе должно заменяться на некоторый элемент этого класса.

Порождающее правило строится из понятий и обозначений

лексических классов. Понятие — это слово из малых букв и символа дефис. Для каждого понятия имеется определяющее его правило, начинающееся с этого понятия с последующим двоеточием. Далее следуют конструкции (порождения этого понятия), разделяемые точкой с запятой. Каждая конструкция состоит из последовательности понятий и обозначений лексических классов, разделяемых пробелами. В конце правила ставится точка.

Приоритет конструкции или записывается явно перед конструкцией, или (в противном случае) определяется приоритетом самого левого (определяющего) обозначения лексического класса в конструкции.

Понятия конструкции имеют приоритет старше приоритета этой конструкции. Однако для определяющего вхождения лево-ассоциативных (правоассоциативных) лексических классов приоритеты понятий, входящих в конструкцию левее (правее) определяющего вхождения, могут совпадать с приоритетом конструкции.

При применении правила приоритет конструкции не должен быть старше приоритета понятия.

### 3.3.3. Грамматика.

программа:

список-внешних-определений.

список-внешних-определений:

список-внешних-определений, внешнее-определение;

ПУСТО.

внешнее определение:

возможная-спецификация ТЧЗАП;

возможная-спецификация список-инициализируемых-описаний ТЧЗАП;

возможная-спецификация описатель-функции текст-функции.

текст-функции:

список-описаний блок.

список-описаний:

список-описаний описание;

ПУСТО.

описание:

спецификация список-описателей ТЧЗАП;

спецификация ТЧЗАП;

возможная-спецификация:

спецификация;

ПУСТО

спецификация:

ПАМЯТЬ тип;

тип ПАМЯТЬ;

ПАМЯТЬ;

тип

тип:

ТИП;  
ТИП ТИП;  
описатель-структуры;  
перечисление;  
СЛОВО.

описатель-структуры:

СТРУКТУРА СЛОВО ФС1 список-описаний-типа ФС2;  
СТРУКТУРА ФС1 список-описаний-типа ФС2;  
СТРУКТУРА СЛОВО.

список-описаний-типа:

описание-типа;  
список-описаний-типа описание-типа.

описание-типа:

тип список-описателей ТЧЗАП;  
тип ТЧЗАП.

список-описателей:

описатель;  
список-описателей ЗАПЯТАЯ описатель.

описатель:

описатель-функции;  
простой-описатель;  
(1) простой-описатель ДВОЕТОЧИЕ выражение;  
(1) ДВОЕТОЧИЕ выражение.

простой-описатель:

УМНОЖИТЬ простой-описатель;  
простой-описатель СКОБКА1 СКОБКА2;  
простой-описатель КВС1 КВС2;  
простой-описатель КВС1 выражение (1) КВС2;  
СЛОВО;  
СКОБКА1 простой-описатель СКОБКА2.

описатель-функции:

УМНОЖИТЬ описатель-функции;  
описатель-функции СКОБКА1 СКОБКА2;  
описатель-функции КВС1 КВС2;  
описатель-функции КВС1 выражение (1) КВС2;  
СКОБКА1 описатель-функции СКОБКА2;  
СЛОВО СКОБКА1 список-слов СКОБКА2;  
СЛОВО СКОБКА1 СКОБКА2

список-слов:

СЛОВО;  
список-слов ЗАПЯТАЯ СЛОВО.

список-инициализируемых-описаний:

список-инициализируемых-описаний  
инициализируемое-описание;  
ПУСТО.

инициализируемое-описание:

спецификация список-инициализируемых-описателей ТЧЗАП;  
спецификация ТЧЗАП.

список-инициализируемых-описателей:

инициализируемый-описатель;

список-инициализируемых-описателей ЗАПЯТАЯ

инициализируемый-описатель.

инициализируемый-описатель:

простой-описатель;

простой-описатель ПРИСВОИТЬ инициализатор;

простой-описатель инициализатор;

описатель-функции.

инициализатор:

(1) выражение;

ФС1 список-инициализации ФС2;

ФС1 список-инициализации ЗАПЯТАЯ ФС2.

список-инициализации:

(1) инициализатор;

список-инициализации ЗАПЯТАЯ инициализатор.

блок:

ФС1 список-инициализируемых-описаний

список-операторов ФС2.

список-операторов:

оператор список-операторов

ПУСТО.

оператор:

выражение ТЧЗАП;

блок;

IF СКОБКА1 выражение СКОБКА2 оператор;

IF СКОБКА1 выражение СКОБКА2 оператор ELSE

оператор;

WHILE СКОБКА1 выражение СКОБКА2 оператор;

DO оператор WHILE СКОБКА1 выражение СКОБКА2

ТЧЗАП;

FOR СКОБКА1 вв ТЧЗАП вв ТЧЗАП вв СКОБКА2

оператор;

SWITCH СКОБКА1 выражение СКОБКА2 оператор;

BREAK ТЧЗАП;

CONTINUE ТЧЗАП;

RETURN ТЧЗАП;

RETURN выражение ТЧЗАП;

GOTO СЛОВО ТЧЗАП;

ТЧЗАП;

метка оператор.

метка:

СЛОВО ДВОЕТОЧИЕ;

CASE выражение ДВОЕТОЧИЕ;

DEFAULT ДВОЕТОЧИЕ.

выражение:

(1) в.

вв:

выражение;  
ПУСТО.

в:

в ЗВЕЗДОЧКА в;  
в ЗАПЯТАЯ в;  
в ДЕЛИТЬ в;  
в ПЛЮС в;  
в МИНУС в;  
в СДВИГ в;  
в ОТНОШЕНИЕ в;  
в РАВНО в;  
в И в;  
в ИСКЛЮЧЕНИЕ в;  
в ИЛИ в;  
в ИЛИИЛИ в;  
в ИИ в;  
в СПРИСВАИВАНИЕМ в;  
в ВОПРОС в ДВОЕТОЧИЕ в;  
в СТАР в;  
первичное.

первичное:

п.

п:

п ПАРА;  
ПАРА п;  
ЗВЕЗДОЧКА п;  
И п;  
МИНУС п;  
НЕ п;  
SIZEOF п;  
(16) СКОБКА1 имя-типа СКОБКА2 п;  
(15) SIZEOF СКОБКА1 имя-типа СКОБКА2;  
п КВС1 в КВС2;  
п СКОБКА1 СКОБКА2;  
п СКОБКА1 параметры СКОБКА2;  
п ПОЛЕ СЛОВО;  
СЛОВО;  
ИЗОБРАЖЕНИЕ;  
СКОБКА1 в СКОБКА2.

параметры:

выражение;  
выражение ЗАПЯТАЯ параметры.

имя-типа:

тип абстрактный-описатель.

абстрактный-описатель:

ПУСТО;  
СКОБКА1 СКОБКА2;  
СКОБКА1 абстрактный-описатель СКОБКА2 СКОБКА1

**СКОБКА2;**

**ЗВЕЗДОЧКА** абстрактный-описатель;

абстрактный-описатель КВС1 КВС2;

абстрактный-описатель СКОБКА1 СКОБКА2.

**перечисление:**

ENUM СЛОВО ФС1 список-перечисляемых ФС2;

ENUM ФС1 список-перечисляемых ФС2;

ENUM слово.

**список перечисляемых:**

перечисляемое;

перечисляемое ЗАПЯТАЯ список-перечисляемых.

**перечисляемое:**

СЛОВО;

СЛОВО ПРИСВОИТЬ ИЗОБРАЖЕНИЕ.

**3.4. Дополнительные возможности С-компиляторов.** В работе [110] описан мобильный отладчик Strace, написанный на языке С препроцессор. Он вставляет отладочные операторы в С — программу.

Графические средства системы UNIX описаны в [20, 25, 27].

Абстрактные типы данных (аналог пакетов в языке Ада) реализованы [113] в качестве препроцессора, а после года эксплуатации включены в специальный вариант С-компилятора.

Кросс-компилятор языка С для микропроцессоров Intel 8080 и 8085, создающий код, комплексируемый с ассемблером и позволяющий создавать системы реального времени, описан в [81].

Имеется препроцессор языка С для введения средств обработки исключительных ситуаций.

## ЗАКЛЮЧЕНИЕ

Интерес к системе UNIX за последние годы возрастает. Многие локальные и коммутируемые сети ЭВМ строятся на основе системы UNIX, также на основе системы UNIX создано рабочее место программиста PWB/UNIX [22].

В СССР ведутся работы по созданию операционных систем, аналогичных системе UNIX, но с улучшенными эксплуатационными характеристиками. Например, на ВДНХ демонстрировалась машинно-независимая операционная система (МНОС) на СМ-4, разработанная в ИПК Минавтопрома.

В настоящем обзоре изложены только основы программирования в системе UNIX: язык заданий shell, язык программирования С, некоторые команды. Материалы для дальнейшего изучения указаны в прилагаемой библиографии.

## ЛИТЕРАТУРА

1. Леонас В. В., Цанг Ф. Р., Отчет № 0284.0046187. Технический проект программного обеспечения мультипроцессорной обработки МВК «Эльбрус-1Б12», ВНИИ центр, 1984, 400 стр.
2. Марчук А. Г., Предложения по созданию переносимой операционной системы. Теор. вопр. параллел. программир. и многопроцессор. ЭВМ. Новосибирск, 1983, 31—45 (РЖМат, 1984, 4Г360)
3. Ступин Ю. В., Методы автоматизации физических экспериментов и установок на основе ЭВМ. Москва, Энергоатомиздат, 1983, 288 стр.
4. Banahan M. F., Rutter A., UNIX-the book. Sigma Technical Press, 1982, Distributed by John Wiley & Sons Ltd., 265 pp.
5. Bergeron R. F., Rochkind M. I., Software tools and components. Bell Syst. Techn. J., 1982, 61, № 6, 1177—1195
6. Blair G. S., Hutchison D., Shepherd W. D., MIMAS—A network operations system for strathnet. Proc. 3rd. Int. Conf. on Distributed Comput. Syst., Miami/Fort Landerdale, Fl, USA, 18—22 Oct. 1982, IEEE, New York, USA, XIX—901, 1982, pp. 212—217
7. —, Mariani J. A., Shepherd W. D., A practical extension to UNIX\* for interprocess communication. Software—Pract. and Exper., 1983, 13, № 1, 45—58. (РЖМат, 1983, 10В860)
8. Bourne S. R., The UNIX shell. Bell Syst. Techn. J., 1978, 57, № 6, 1971—1990
9. —, The UNIX system. Addison-Wesley Publishing Company, 1982, 350 pp.
10. Breiland I. R., Friedernson R. A., Designer's workbench: the user environment. Bell Syst. Techn. J., 1980, 59, № 9, 1767—1792
11. Bruce J. W., Kemmerer A., Popek G. J., Specification and verification of the UCLA Unix security. Commun ACM, 1980, 23, № 2, 118—131
12. Bruner J. D., Reeves A. B., An image processing system with computer network distribution capabilities. Proc. of PRIP 82. IEEE Comp. Soc. Conf. on Pattern Recogn. and Image Processing. Las Vegas, UN, USA, 14—17 June 1982», IEEE, New York, USA, XII+701 pp., 1982, 447—50
13. Burkowski F. J., Mackey W. F., Micro-C: a universal high level language for microcomputers. MIMI'77; Proc. Int. Symp. on Mini and Micro Comput., Montreal, Que, Nov., 11—18, 1977, IEEE, New York, № 4, 1978, 27—31
14. Cabrera L. F., Paris J. F., Comparing user response times on paged and swapped UNIX by the terminal probe method. Proc. Comput. Performance Evaluation Users Group, 17th Meeting, San-Antonio, TX, USA, 16—19, Nov., 1981, NBS, Washington, DC, USA, XVI+348 pp., 1981, 157—168
15. Christian K., The UNIX operating system. A Wiley-Interscience Publication John Wiley & Sons, 1983, 318 pp.
16. Cleary A., Bird R., Variations on operating systems. Data Process., 1982, 24, № 8, 18—19
17. Cohen H., Kaufeld J. C., Jr, The network operations center system. Bell. Syst. Techn. J., 1978, 57, № 6, 2289—2304
18. Collinson R. P. A., The Cambridge ring and Unix. Software—Pract. and Exper., 1982, 12, № 6, 583—594
19. Conrad S. A., Wolfenson L. B., A complex number facility for the C programming language. IEEE Trans. Biomed. Eng., 1982, 29, № 2, 145—146 (РЖМат, 1982, 8В992)
20. Corbett C., Witten I. H., On the inclusion and replacement of documentation graphics in computer type-setting. Comput. J., 1982, 25, № 2, 272—277 (РЖМат, 1982, 11В1168)
21. Dahmke M., Microcomputer operating systems. Byte Books, Subsidiari of McGraw-Hill, 1982, 230 pp.
22. Dolotta T. A., Haisht R. C., Mashey J. R., Programmer's workbench. Bell Syst. Techn. J., 1978, 57, № 6, 2177—2200
23. —, Olsson S. B., Petrucelly A. G., Editors, UNIX user's manual. Holt, Reinhart and Winston, 1983, V 1, 2

24. *Ellenberger D. J.*, AIDE—a tool for computer architecture design. Proc. Conf. ACM IEEE 18th Design Automation. Nashville, TN, USA, 29 June—1 July 1981, IEEE, New York, USA, XVIII+897 pp., 1981, 796—803
25. *Ewald R. H., Maas L. D.*, Interactive graphics on the CRAY-1 supercomputer. Los Alamos scientific Labs. NM. Dept. of Energy, 1978, 19 p.
26. *Fabry R. S.*, Proposal for configuration control for the ARPA standart version of the UNIX operating system., Tech. Rept., Calif. Univ., Berkeley. Electronics Res. Labs., 1980, 4 p.
27. *Ferrin T. E., Langridge R.*, Interactive computer graphics with the UNIX time-sharing system. Comput. Graphics, 1980, 13, № 4, 320—331
28. *Feuer A. R.*, The C puzzle book. Prentice-Hall, Inc., Englewood Cliffs, NJ, 1982, VIII+174 pp.
29. *Fitzhorn P. A., Johnson G. R.*, C: toward a concise syntactic description. SIGPLAN Notic., 1981, 16, № 12, 14—21
30. *Freedman D.* Portable operating systems fight for 16-bit machines. Mini-Micro Syst. (USA), 1982, 15, № 9, 237—240
31. *Gauthier R.*, Using the UNIX system. A Prentice-Hall Company Reston, Virginia, 1981, 297 pp.
32. *Geus A. F. de*, A new structure for the UNIX kernel. TW Rept. Math. Inst. Rijks Univ. Groningen, 1983, № 256, 86 pp., ill. (PYKMar, 1984, 2Г357)
33. *Gien M.*, SOL: a UNIX environment in Pascal. Compcon 83. Intellectual leverage for the information society. San Francisco, CA, USA, 28 Feb.—3 March 1983, IEEE, New York, USA, XXXIV+527 pp., 1983, 184A-D
34. *Greenberg R. B.*, The UNIX operating system and the XENIX standart operating environment. Byte, 1981, 6, № 6, 248—264
35. *Groff J. R., Weinberg P. N.*, Understanding UNIX: A conceptual guide. Que Corporation, 1984, 256 pp.
36. —, —, UNIX: using the Bourne Shell. Que Corporation, 1984, 280 pp.
37. *Grzelakowski M. E., Campbell J. H., Dubman M. R.*, The 3B20D processor and DMERT operating systems: DMERT operations system. Bell Syst. Techn. J., 1983, 62, № 1, 303—322
38. *Hancock L., Krieger M.*, The C Primer. McGraw-Hill International Book Company, 1982, IX+235 pp.
39. *Harvold D. M.*, High speed data acquisition: running a real-time process and a time-shared system (UNIX) concurrently. Software—Pract. and Exper., 1980, 10, № 4, 273—281
40. *Hays A. V., Jr., Richmond B. J., Optican L. M.*, A UNIX-based multiple-process system, for real-time data. Electron. Conventions, El. Segundo, CA, USA, 127 pp., 1982, 2—1/1—10
41. *Heffler M. J.* Description of a menu creation and interpretation system. Software—Pract. and Exper., 1982, 12, № 3, 269—281
42. *Herbinson-Evans D.*, A dancer among us (computer application). Austral. Comput. Sci. Commun., 1981, 3, № 1A, 68—73
43. *Herman J. G., Bernstein S. L.*, Monitoring, control and management of the defense data network. Conf. Rec. Eascon 82. 15th Ann. Electr. and Aerospace syst. conf. Washington, DC, USA, IEEE, New York, 1982, XIII+438 pp.
44. Hewlett-Packard, Elec. News, 1982, Nov. 22, pp. 18, 24
45. *Hogan T.*, The C programmers handbook. R. G. Brady, 1984, 288 pp.
46. *Holt R. C.*, Concurrent Euclid the UNIX system and TUNIS. Addison-Wesley Publishing Company, 1983, 323 pp.
47. —, TUNIS: a UNIX look-alike written in concurrent EUCLID. Oper. Syst. Rev., 1982, 16, № 1, 4—5
48. *Hughes P.*, The operating system of the future. Microcomputings, 1982, 6, № 6, 28—31
49. *Hwang Kai, Croft W. J., Goble G. H., Wah B. W., Briggs F. A., Simmons W. R., Coates C. L.*, A UNIX-based local computer network with load balancing. Computer, 1982, 15, № 4, 55—65
50. Information transfer software. New Sci., 1982, Aug. 5, pp. 366
51. *Ivte E. L.*, The programmer's workbench—a machine for software deve-

- lopment. *Communs. ACM*, 1977, 20, № 10, 746—753 (PЖМАТ, 1978, 3B773)
52. *Jalics P. J., Heines T. S.*, Transporting a portable operating system: UNIX to an IBM minicomputer. *Communs ACM*, 1983, 26, № 12, 1066—1071
  53. *Johnson S. C.*, Language development tools on the UNIX system. *Computer*, 1980, 13, № 8, 16—21
  54. —, *Ritchie D. M.* Portability of C programs and the UNIX system. *Bell Syst. Techn. J.*, 1978, 57, № 6, 2021—2048
  55. *Karshmer A. I., Depree D. J., Phelan J.*, The New Mexico State University ring-star system: a distributed UNIX environment. *Software—Pract. and Exper.*, 1983, 13, № 12, 1157—1168 (PЖМАТ, 1984, 6Г318)
  56. *Kernighan B. W., Lesk M. E., Ossana J. F., Jr.*, Document preparation. *Bell Syst. Techn. J.*, 1978, 57, № 6, 2115—2135
  57. —, *Mashey J. R.*, The UNIX programming environment. *Computer*, 1981, 14, № 4, 12—24
  58. —, *Morgan S. P.* The UNIX operating system: a model for software design. *Science*, 1982, 215, 779—783
  59. —, *Pike R.*, UNIX programming environment. Prentice-Hall, 1984, 368 pp.
  60. —, *Plauger P. J.*, Software tools, Addison—Wesley, Reading, MA, 1976, 338 pp.
  61. —, *Ritchie D. M.*, The C programming language. A Prentice-Hall Company, Englewood Cliffs, New Jersey, 1978. DEC Edition, 1983, 240 pp.
  62. KSOS secure UNIX operating system users manual. (Kernelized secure operating system). Ford Aerospace and Communic. Corp., Palo Alto, CA. Western Development Labs. Div., 1980, Dec, 184 p.
  63. *Kummerfeld R. L., Lauder P. R.*, The Sydney UNIX network. *Austral. Comput. J.*, 1981, 13, № 2, 52—57
  64. *Landy M. S., Cohen Y., Sperling G.*, HIPS: a UNIX-based image processing system. *Comput. Vision, Graph., and Image Process.*, 1984, 25, № 3, 331—347 (PЖМАТ, 1984, 9Г399)
  65. LAN9000 Local Area Network. HP9000 Data Communications, Hewlett-Packard, 1983, p. 16
  66. *Lee G., Wiegandt D.*, Interfacing CAMAC to a UNIX system. *Interfaces in Computing*, 1983, 1, 329—337
  67. *Libes D.*, UNIX and CP/M. *Microsystems*, 1983, 4, № 1, 26—34
  68. *Lima I. G., Hopkings R., Marshall L., Mundy D., Treleaven P.*, Decentralised control flow-based on UNIX. *SIGPLAN Notic.*, 1983, 18, № 6, 192—201
  69. *Lind H. O.*, UNIX NSW front and enhancements. Vol. II. Bolt Beranek and Newman, Inc., Cambridge, MA, Rome Air Development center, Griffiss AFB, NY, 1981, June, 70 p.
  70. *Lomuto A. N., Lomuto N.*, A UNIX primer. Prentice-Hall, Englewood Cliffs, NJ, USA, XVI+739 pp., 1983.
  71. *Landwehr C. E.*, A survey of formal models for computer security. *Naval. Res. Lab.*, Washington, DC., 1981, Sept., 40 p.
  72. *Luderer G. W. R., Che H., Marshall W. T.*, A virtual circuit switch as the basis for distributed systems. *J. Telecommun. Networks*, 1982, 1, № 2, 147—160
  73. —, *Maranzano J. F., Tague B. A.*, UNIX operating system as a base for application. *Bell Syst. Techn. J.*, 1978, 57, № 6, 2201—2207
  74. *Lycklama H.*, UNIX on a microprocessor. *Bell Syst. Techn. J.*, 1978, 57, № 6, 2087—2101
  75. —, *Bayer D. L.*, The MERT operating system. *Bell Syst. Techn. J.*, 1978, 57, № 6, 2049—2086
  76. *Martin M. R.*, UNIX and local computer networking. *IEEE Comput. Soc.*, 1982, № 397, 318—322
  77. *McDonald P. H., Thompson T. J.*, Designer's workbench: the programmer environment. *Bell Syst. Techn. J.*, 1980, 59, № 9, 1793—1809
  78. *McGilton H., Morgan R.*, Introduction the UNIX system. McGraw—Hill, 1983, XIX+556 pp.

79. *Morgan C., Sufrin B.*, Specification of the UNIX filing system. IEEE Trans. Software Eng., 1984, 10, № 2, 128—142 (PKMar, 1984, 10F535)
80. *Morgan S. P.*, UNIX system: making computers easier to use. Bell Lab. Rec., 1978, 56, № 11, 308—313
81. *Murray C. G., Gray P. M. D.*, Using language C for real-time control of microprocessor systems with small memory. J. Microcomput. Appl., 1982, 5, № 1, 1—12
82. *Myers C., Munsey G.*, A multiprocessor minicomputer designed for UNIX. Comput. Des., 1982, 21, № 2, 87—96 (PKMar, 1982, 12B1042)
83. *Nagelberg E. R., Pilla M. A.*, RBCS/RCMAS — converting to the MERT operating system. Bell. Syst. Techn. J., 1978, 57, № 6, 2275—2287
84. *Nowitz D. A., Lesk M. E.*, Implementation of a UNIX network. Comput. Commun., 1982, 5, № 1, 30—34
85. *O'Neile L. A.*, Designer's workbench: philosophy. Bell Syst. Techn. J., 1980, 59, № 9, 1757—1765
86. *Panzieri F., Shrivastava S. K.*, Reliable remote calls for distributed UNIX: an implementation study. Proc. 2nd Symp. Reliab. Distrib. Software and Database Syst., Pittsburg, P, 19—21 July, 1982, Silver, Spring, Md, 1982, 127—133 (PKMar, 1983, 12B1133)
87. *Perلمان G.*, The design of an interface to a programming system and MENUNIX. A menu-based interface to UNIX (User manual). California Univ., San Diego, La Jolla, Center for Human Inf. Process., 1981, Nov., 19 p.
88. *Plum T.*, Learning to programming in C. Cardiff, N. J., Plum Hall, 1983, 365 pp.
89. *Pohm A. V., Davis J. A., Christiansen S., Bridges G., Horton R. E.*, Local network of mini and microcomputers for experiment support. Comput. Networks, 1979, 3, № 6, 381—387
90. *Popek G. J.*, Secure reliable processing systems. Calif. Univ., Los Angeles. Dept. of Compt. Sci., Semi-annual Tech. Rept., 1981, July, № 79—30, 216 p.
91. *Preister R. G.*, Programming in C. Prentice-Hall, 1984, 206 pp.
92. *Purdun J.*, C programming guide. Que Corporation, 1983, 250 pp.
93. —, *Leslie T., Stagemoller A.*, C programming library. Que Corporation, 1984, 365 pp.
94. *Rashid R. F.*, Inter-process communication facility for UNIX. Proc. of IFIP W. G. 6.4 Int. Workshop on local Networks, North-Holland Publ. Co., Amsterdam, Neth and New York, NY, USA, 1981, 319—354
95. —, *Robertson G. G.*, ACCENT: a communication oriented network operating system kernel. Oper. Syst. Rev., 1981, 15, № 5, 64—75
96. *Ritchie D. M.*, UNIX time-sharing system: a retrospective. Bell. System. Techn. J., 1978, 57, № 6, 1947—1969
97. —, *Johnson S. C., Lesk M. E., Kernighan B. W.*, The C programming language. Bell Syst. Techn. J., 1978, 57, № 6, 1991—2019
98. —, *Thompson K.*, UNIX time-sharing system. Commun ACM, 1974, 17, № 7, 365—375
99. —, —, UNIX time-sharing system. Bell Syst. Techn. J., 1978, 57, № 6, 1905—1929
100. —, —, Some further aspects of the UNIX time-sharing system. Mini-micro software, 1981, 6, № 3, 9—11
101. *Roome W. D.*, Programmer's workbench new tools for software development. Bell Lab. Rec., 1979, 57, № 1, 19—25
102. *Rowe L. A., Birman K. P.*, A local network based on the UNIX operating system. IEEE Trans. Software Eng., 1982, 8, № 2, 137—146 (PKMar, 1982, 9B785)
103. *Rozenblat A.*, 1982 award for achievement. Electronics, 1982, 55, № 21, 108—111
104. *Rutter A.*, The big red book of C. Sigma Technical Press., 1984, 160 pp.
105. *Ryer M.*, Developing an Ada-programming support environment. Mini-micro Syst. (USA), 1982, 15, № 9, 223—726

106. *Samish F.*, UNIX: still just over the horizon. *Micro Decis.*, 1983, № 20, 166—168
107. *Schindler M.*, Technology forecast software. *Electron. Des.*, 1983, 31, № 1, 244—262
108. *Sideloottem T. O., Wortman L.*, C programming tutor. R. G. Brady, 1984, 288 pp.
109. *Silvester P. P.*, The UNIX™ system guidebook. New York e. a.: Springer, 1984. xii, 207 pp., ill. ISBN 0-387-90906-0 US (PJKMar, 1984, 9Г221K)
110. *Steffen J. L.*, Experience with a portable debugging tool. *Software-pract. and Exper.*, 1984, 14, № 4, 332—334 (PJKMar, 1984, 9Г317)
111. *Stonebraker M.*, Requiem for a data base system. *Mem. Rept., Calif. Univ., Berkeley, Electron. res. lab.*, 1979, Jan., 37 pp.
112. —, *Wong E., Kreps P., Held G.*, The design and implementation of INGRES. *Calif. Univ., Berkeley, Electronics Res. Lab. Air. Force Office of Sci. Res., Bolling AFB, DC*, 1976, 36 p.
113. *Stroustrup B.*, Classes: an abstract data type facility for the C language. *SIGPLAN Notic.*, 1982, 17, № 1, 42—51
114. *Thomas R. A., Emerson S., Yates J., Campbell J.*, A business guide to the UNIX system. Addison-Wiley, 1984, 488 pp.
115. —, *Yates J.*, A user guide to the UNIX system. Osborn/McGraw-Hill Berkley, California, 1982, 520 pp.
116. *Thomas R. H.*, Unix NSW front end. Bolt Beranek and Newman Inc., Cambridge, MA, Rome Air Development Center, Griffiss AFB, NY, 1980, Mar. 108 p.
117. —, *Lind H. O., Toner S. G.*, UNIX NSW front and enhancements. Vol. 1. Bolt Beranek and Newman Inc., Cambridge, MA, Rome Air Development Center, Griffiss AFB, NY, № 4, 1981, June, 203 p.
118. *Thompson K.*, UNIX implementation. *Bell Syst. Techn. J.*, 1978, 57, № 6, 1931—1946
119. *Thompson T. J.*, Designer's workbench: providing a production environment. *Bell Syst. Techn. J.*, 1980, 59, № 9, 1811—1825
120. *Unger B. W., Mutalik P. R.*, Modular implementation and simulation of the UNIX operating system. *Proc. Summer. Comput. Simul. Conf. Newport. Bearch., Calif.*, July, 24—26, 1978.
121. Unity of VMS and UNIX. ICP Software J., *Spec. Rep.*, 1982, p. 25
122. *Waite M., Martin D., Prata S.*, UNIX primer plus. Howard W. Sams. & Co., Inc., USA, 1983, 407 pp.
123. *Withington P. T., Solomon D. J.*, A secure message system for the distributed processing project. *Fin. Tech. Rept., Mitre Corp., Bedford, MA., Rome Air Development Center, Griffiss AFB., NY*, 1981, Mar., 37 pp.
124. *Wolker B. J., Kemmerer R. A., Poperk G. J.*, Specification and verification of UCLA UNIX security kernel. *Communs ACM*, 1980, 23, № 2, 118—131