Федеральное агентство по образованию Российской федерации
Томский государственный университет систем управления и радиоэлектроники

В. В. Одиноков, В.П. Коцубинский, Д.А. Звонков

ЛАБОРАТОРНЫЙ ПРАКТИКУМ ПО ОПЕРАЦИОННОЙ СИСТЕМЕ UNIX

Томск 2010

СОДЕРЖАНИЕ

Лабораторная работа №1. Первоначальное знакомство с UNIX 4
1. Цель работы 4
2. Вход в систему и выход из нее 4
3. Работа с текстовым редактором 7
4. Файловая структура системы 9
5. Команды для работы с файлами 11
6. Работа в среде Midnight Commander 14
6.1. Представление на экране файловой структуры 14
6.2. Команды для работы с файлами 15
6.3. Ввод команд UNIX 17
6.4. Настройка Midnight Commander 17
7. Электронный справочник 18
8. Задание 19
Лабораторная работа №2. Дальнейшее знакомство с командами UNIX 19
1. Цель работы 19
2. Права доступа к файлам 19
3. Перенаправление ввода-вывода 21
4. Конвейеры 24
5. Применение метасимволов в командах 24
6. Командный интерпретатор и командные файлы 25
7. Задание 28
Лабораторная работа №3. Управляющие операторы командного языка 29
1 Цель работы 29
2. Двухальтернативный выбор 29
3. Многоальтернативный выбор 31
4. Цикл с перечислением 33
5. Цикл с условием 34
6. Цикл с инверсным условием 35
7. Задание 36
Лабораторная работа №4. Процессы в UNIX 37
1. Цель работы 37
2. Понятие процесса 37
3. Запуск процессов в фоновом режиме 38
4. Получение информации о процессах 38
5. Приоритеты процессов 40
6. Уничтожение процессов 41
7. Задержка процессов 42
8. Сообщения другому пользователю 43
9. Задание 44

Лабораторная работа №1. Первоначальное знакомство с UNIX

1. Цель работы

Целью выполнения настоящей лабораторной работы является получение начальных навыков работы в среде UNIX: 1) вход в систему; 2) знакомство с текстовым редактором ed; 3) применение команд shell для работы с файлами; 4) работа в среде Midnight Commander.

2. Вход в систему и выход из нее

При работе с многопользовательской системой, каковой является UNIX, каждый пользователь имеет свой уникальный идентификатор – имя пользователя. Кроме того, для избежания использования имени пользователя другим лицом, каждый пользователь может иметь пароль пользователя.

Имя пользователя — символьная строка. Например, оно может включать (одновременно) вашу фамилию, имя и отчество. Другой вариант — имя содержит номер студенческой группы и инициалы пользователя. В любом случае имя пользователя должно быть согласовано с администратором системы, который регистрирует его в «журнале» системы.

Работа с ВС, на которой установлен UNIX, начинается с включения терминала. В качестве терминала может использоваться совокупность экрана и клавиатуры или, даже, целая периферийная ЭВМ. В любом случае на экране появляется сообщение UNIX — **login**, в ответ на которое следует ввести имя пользователя. Пример ввода имени, зарегистрированного в системе:

```
UNIX     →login:
польз.   →vlad <Enter>
UNIX     →$
```

Символ $ есть приглашение UNIX к вводу команды. Его появление на экране говорит об успешном входе в систему. Пример ввода имени, незарегистрированного в системе:

```
UNIX     →login:
польз.   →vlid <Enter>
UNIX     →passwd:
```

Слово **passwd** означает просьбу ввести пароль. Может показаться странным, что в ответ на неправильное имя пользователя UNIX выводит такую просьбу. Дело в том, что таким образом система скрывает от лица, использующего неверное имя, сам факт такого использования. Поэтому вместо того, чтобы угадывать правильное имя, данное лицо будет заниматься угадыванием пароля, что заведомо безнадежно, так, как ввод любого пароля приведет к повторению вывода на экран сообщения “login”.

Применение пароля – дело добровольное. Но если вы хотите, чтобы никто (или почти никто) в будущем не разрушил результат вашей работы, или

не воспользовался бы им без вашего разрешения, без пароля не обойтись. Пароль пользователя регистрируется в системе не администратором, а самим пользователем с помощью команды **`passwd`** . Пример:

```
UNIX     $
польз.   passwd <Enter>
UNIX     Changing password for vlad
         New password:
польз.   dracon <Enter>
UNIX     Retype new password:
польз.   dracon <Enter>
UNIX     $
```

Набор пароля производится пользователем «вслепую» — без отображения набираемых на клавиатуре символов на экране. Это позволяет избежать «подсматривание» пароля.

Требования к паролю: 1) если для написания пароля используются обычные алфавитно-цифровые символы, его длина должна быть не менее 6 символов (в примере пароль пользователя `dracon` отвечает этому требованию); 2) при использовании специальных и управляющих символов допускается меньшая длина пароля.

Если пароль отвечает, хотя бы одному из приведенных двух требований, `UNIX` выводит просьбу повторить ввод нового пароля (с целью избежания ошибки при наборе пароля). Иначе `UNIX` выведет сообщение: “`Please use a longer password`” («Пожалуйста, введите длинный пароль»).

После того, как пароль зарегистрирован, он будет запрашиваться всякий раз, когда вы будете входить в систему. Пример такого входа:

```
UNIX     login:
польз.   vlad <Enter>
UNIX     passwd:
польз.   dracon <Enter>
UNIX     $
```

Если вы забудете пароль, то не сможете войти в систему без помощи администратора. Он удалит прежний пароль, а затем вы зарегистрируете в системе новый пароль. Замену пароля может выполнить и сам пользователь при условии, что прежний пароль им не забыт. Для этого вновь используется команда `passwd`. Пример:

```
UNIX     $
польз.   passwd <Enter>
UNIX     Changing password for vlad
         Old password:
польз.   dracon <Enter>
UNIX     New password:
польз.   dracon7 <Enter>
```

```
UNIX     Retype new password:
польз.   dracon7 <Enter>
UNIX     $
```

Зарегистрируйте свое имя пользователя у администратора UNIX-системы. Выполните вход в систему и установку пароля.

Примечание. Допустим, что после включения терминала вы оказываетесь не в среде UNIX, а в среде другой операционной системы, позволяющей связываться с UNIX. Тогда прежде, чем ввести команды по входу в UNIX, необходимо войти в исходную операционную систему, а затем ввести ее команду по запуску «терминала» в среде UNIX. При этом под «терминалом» понимается не аппаратное устройство, а псевдотерминал — программный процесс, связанный с UNIX.

Например, если ваш терминал находится в локальной сети, управляемой операционной системой Novell Net Ware, то вы сначала входите в эту систему под своим логическим именем, которое зарегистрировано в этой локальной сети. После этого следует набрать команду Kermit, которая осуществляет запуск программного процесса, имитирующего терминал UNIX. При этом в ответ на запрос этой программы следует ввести идентификатор используемой UNIX-системы. Далее Kermit обратится к UNIX так, как будто речь идет о включении реального терминала. В ответ UNIX передаст в Kermit версию UNIX-системы и номер «терминала» UNIX . Далее Kermit выводит эти данные на экран.

Ситуация усложняется тем, что сетевая операционная система Novell Net Ware представляет собой надстройку над другой системой, например, над MS-DOS. В этом случае подготовка к запуску UNIX может выглядеть следующим образом:

```
MS-DOS   n:\
польз.   Kermit
Kermit   [n:\]JMS-Kermit>
польз.   telnet dark.tusur.ru
UNIX     FreeBSD/:386(dark.tusur.ru)(ftyp0)
UNIX     login:
```

Вопрос о взаимодействии различных операционных систем весьма интересен, но сейчас нас интересуют лишь его технические аспекты.

Выход из системы. При завершении сеанса работы необходимо сообщить об этом UNIX командой <Ctrl>&<D> (одновременное нажатие клавиш <Ctrl> и <D>). Пример:

```
UNIX     $
польз.   <Ctrl>&<D>
UNIX     login:
```

Другой способ выхода из UNIX — использование команды **exit**.

Выйдите из системы, а затем опять войдите в нее.

3. Работа с текстовым редактором

Одной из часто используемых системных программ является **текстовый редактор**. Эта утилита вводит текстовую информацию (в коде ASCII) с клавиатуры в ОП, откуда она выводится на экран и может быть записана в файл на магнитном диске. Работая в среде UNIX, мы можем использовать несколько текстовых редакторов, например, **ed**. Редактор ed ориентирован на построчное редактирование. **Строка** – текст, введенный до нажатия клавиши <ENTER>.

Запуск редактора выполняется командой ed с параметром, в качестве которого выступает имя редактируемого текстового файла. Так как ed весьма старый редактор, то простое имя файла не должно иметь длину более 14 символов. При этом, как и в любой команде UNIX, команда ed отделяется от параметра одним или несколькими пробелами. Пример:

```
UNIX     $
польз.   ed letter<Enter>
ed       ?letter
```

В данном примере файл letter новый и в нем еще ничего нет. Поэтому в ответном сообщении редактора присутствует символ “?”. Если бы мы задали имя уже существующего файла, то ed сообщил бы длину редактируемого файла в байтах. При этом последняя строка могла бы выглядеть, например, так:

```
ed       1234
```

Допустим пока, что мы создали новый (пустой) файл. Для того, чтобы поместить в него текст, необходимо перейти в режим добавления. Это делается командой **a** редактора. Пример:

```
польз.   a<Enter>
ed       ответа нет; переход на новую строку
польз.   This is a test to see if I am<Enter>
         entering text in the file “letter”.<Enter>
         Once I have completed it I shall find<Enter>
         that I have created 4 new lines of data<Enter>
         . <Enter>
```

Обратите внимание на точку в последней строке. Она представляет собой команду выхода из режима добавления. Другого способа выхода из этого режима нет. Так как только одиночная точка в строке будет воспринята редактором как команда, а не как часть вводимого текста.

После того, как текст файла набран, его можно выводить на экран и редактировать. Для вывода файла используется команда **p** редактора. Вывод файла можно выполнять построчно или в виде группы строк. Для построчного вывода редактору передается номер первой требуемой строки, в ответ, на что

ed выводит текст этой строки на экран. Далее вывод каждой следующей строки файла предваряется нажатием клавиши <Enter>. Пример:

```
польз.   1p<Enter>
ed       This is a test to see if I am
польз.   <Enter>
ed       entering text in the file "letter".
польз.   <Enter>
ed       Once I have completed it I shall find
польз.   <Enter>
ed       that I have created 4 new lines of data
польз.   <Enter>
ed       ?
```

Символ "?" означает, что текст файла закончился.

Для вывода на экран группы строк, в редактор передаются одновременно номер первой и номер последней выводимой строки файла, разделенные запятой. При этом для обозначения последней строки файла можно использовать символ **$** . Пример:

```
польз.   1,$p<Enter>
ed       This is a test to see if I am
         entering text in the file "letter".
         Once I have completed it I shall find
         that I have created 4 new lines of data
```

Для выполнения редактирования файла, во-первых, устанавливается требуемая **текущая строка**, а во-вторых, выполняется требуемая операция редактирования. Для установки требуемой текущей строки можно использовать построчный вывод текста. Получив на экране требуемую строку, следует ввести одну из команд редактирования: **a** — добавление текста после текущей строки; **i** — добавление текста перед текущей строкой; **c** — замена текущей строки текстом новой строки (строк); **d** — удаление текущей строки. Пример:

```
польз.   $p<Enter>
ed       that I have created 4 new lines of data
польз.   a<Enter>
         I will now enter two new lines of<Enter>
         text to see if it is accepted.<Enter>
         .<Enter>
```

Команда редактора $p требует от него вывода на экран последней строки файла. Поэтому две строки, набранные в примере пользователем, добавляются в конец файла. Пример добавления текста перед текущей строкой:

```
польз.   3p<Enter>
ed       Once I have completed it I shall find
```

```
польз.   i<Enter>
         I am now inserting two lines of<Enter>
         text to demonstrate how it works.<Enter>
         .<Enter>
```

Пример замещения текста текущей строки текстом другой строки:

```
польз.   3p<Enter>
ed       Once I have completed it I shall find
польз.   c<Enter>
         After completion, I shall find.<Enter>
         .<Enter>
```

В данном примере текущая строка заменена текстом одной строки. На самом деле число новых строк может быть любым. В следующем примере производится удаление двух строк:

```
польз.   3p<Enter>
ed       I am now inserting two lines of
польз.   d<Enter>
польз.   d<Enter>
```

Одной командой можно удалить целую группу строк. В следующем примере производится удаление всего содержимого файла:

```
польз.   1,$d<Enter>
польз.   1,$p<Enter>
ed       ?
```

Если текст файла достаточно большой, рекомендуется выполнять промежуточное запоминание файла на диске, используя команду **w**. Выполнив ее, редактор выведет на экран новую длину файла.

Выход из редактора ed в UNIX выполняет команда редактора – **q.** Пример:

```
польз.   q<Enter>
UNIX     $
```

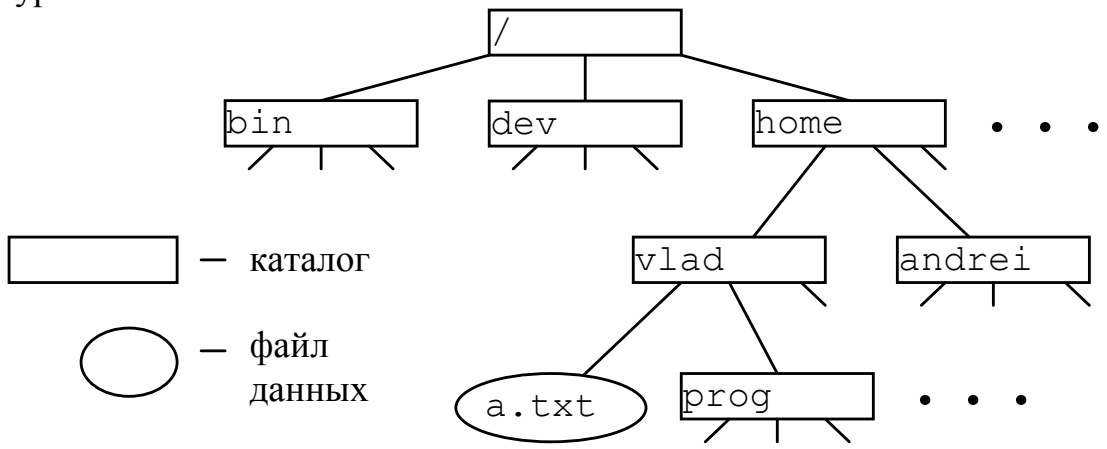
Выполните создание нового текстового файла, а затем выйдите из редактора в UNIX. Далее скорректируйте этот файл, добавив новые строки в конец, в начало и в середину файла. Затем удалите некоторые строки и проверьте результаты редактирования с помощью вывода файла на экран.

4. Файловая структура системы

В ведении UNIX находятся различные периферийные устройства — устройства внешней памяти и устройства ввода-вывода. Информация, расположенная на конкретном ПУ, разделена на поименованные части,

называемые **файлами**. В то время, как устройство внешней памяти (а точнее — установленный на нем носитель) содержит большое количество файлов, устройство ввода-вывода рассматривается в качестве одного файла, в который можно записывать информацию (устройство вывода) или из которого информацию можно читать (устройство ввода).

Для того, чтобы ориентироваться в огромном количестве файлов, используются служебные файлы, называемые **каталогами**. Каждый каталог содержит сведения о нескольких других файлах, в число которых, возможно, входят и каталоги. Эти каталоги являются по отношению к первому каталогу «дочерними». В результате все файлы, находящиеся в ведении UNIX, объединены в единую древовидную файловую структуру. Фрагмент этой структуры:



Корневой каталог имеет имя "/". Его дочерние каталоги: bin — содержит файлы (а точнее — сведения о файлах) с системными утилитами; dev — содержит файлы устройств ввода-вывода; home — содержит личные каталоги пользователей; другие каталоги.

После того, как вы успешно вошли в систему, UNIX назначает вам каталог, «дочерний» к каталогу home. В качестве имени этого каталога может использоваться имя пользователя, например — vlad. Теперь каталог vlad является корнем вашей собственной файловой структуры, которую вы можете «выращивать» на протяжении одного или многих сеансов работы в среде UNIX. (Следует запомнить, что уничтожать свой корневой каталог ни в коем случае нельзя.)

В любой конкретный момент времени один каталог в файловой структуре пользователя является особенным. Это **текущий каталог пользователя.**

Обратим внимание на то, что благодаря древовидной файловой структуре каждый файл или каталог имеет **имя-путь**, уникальное для данной системы. Пример имени-пути: /home/vlad/a.txt. Данное имя-путь можно использовать в любой команде UNIX, работающей с данным файлом.

Если адресуемый файл является «потомком» текущего каталога, то в качестве его имени можно использовать «смещение» относительно текущего каталога. Например, если текущим каталогом является /home, то записанное выше имя-путь файла может быть заменено более коротким именем vlad/a.txt Обратите внимание на отсутствие в начале этого имени символа "/". Его наличие всегда говорит о том, что записано полное имя-путь.

5. Команды для работы с файлами

Команда **pwd** выводит на экран полное имя-путь текущего каталога. Пример:

```
UNIX     $
польз.   pwd<ENTER>
UNIX     /home/vlad
         $
```

Команда **ls** выводит на экран содержимое текущего каталога в алфавитном порядке. В простейшем случае команда не содержит параметров и позволяет вывести лишь имена файлов (в том числе и каталогов). Пример:

```
UNIX     $
польз.   ls<ENTER>
UNIX     a.txt
         prog
         $
```

Замена текущего каталога производится с помощью команды **cd**, в качестве параметра которой задается имя нового текущего каталога. Это имя представляет собой или полное имя-путь, или «смещение» относительно текущего каталога. Следующий пример иллюстрирует первый вариант:

```
UNIX     $
польз.   cd /home/vlad/prog<ENTER>
UNIX     $
```

Пусть текуший каталог есть `/home/vlad`, тогда каталог `/home/vlad/prog` можно сделать текущим следующим образом:

```
UNIX     $
польз.   cd prog<ENTER>
UNIX     $
```

Для перехода по файловой структуре вверх, можно использовать символы “**..**” . Пример перехода вверх на один уровень:

```
UNIX     $
польз.   pwd<ENTER>
UNIX     /home/vlad/prog
польз.   cd ..<ENTER>
UNIX     $
польз.   pwd<ENTER>
UNIX     /home/vlad
         $
```

Пример перехода вверх на два уровня:

```
UNIX     $
польз.   pwd<ENTER>
UNIX     /home/vlad/prog
польз.   cd ../..<ENTER>
UNIX     $
польз.   pwd<ENTER>
UNIX     /home
         $
```

Создание новых каталогов выполняет команда **mkdir**, параметрами которой являются имена новых каталогов. Перед применением этой команды в качестве текущего каталога должен быть установлен тот каталог, «дочерними» по отношению к которому являются новые каталоги. Пример:

```
UNIX     $
польз.   pwd<ENTER>
UNIX     /home/vlad
польз.   mkdir c b<ENTER>
UNIX     $
польз.   ls<ENTER>
UNIX     a.txt
         b
         c
         prog
         $
```

Удаление каталогов выполняет команда **rmdir**, параметрами которой являются имена удаляемых каталогов. Перед применением этой команды в качестве текущего каталога желательно установить тот каталог, «дочерними» по отношению к которому являются удаляемые каталоги. Пример:

```
UNIX     $
польз.   ls<ENTER>
UNIX     a.txt
         b
         c
         prog
         $
польз.   rmdir c b<ENTER>
UNIX     $
польз.   ls<ENTER>
UNIX     a.txt
         prog
         $
```

Удаление файлов выполняет команда **rm**, параметрами которой являются имена удаляемых файлов. Применение этой команды аналогично применению команды `rmdir`.

Переименование файла (каталога) выполняет команда **mv**, в качестве первого параметра которой выступает старое имя файла (каталога), а в качестве второго параметра — новое имя файла (каталога). Перед применением команды желательно установить тот текущий каталог, в котором находится переименуемый файл (каталог). Пример:

```
UNIX     $
польз.   ls<ENTER>
UNIX     a.txt
         prog
         $
польз.   mv a.txt b.txt<ENTER>
UNIX     $
польз.   ls<ENTER>
UNIX     b.txt
         Prog
         $
```

Вывод текстовых файлов на экран выполняет команда **cat**. В следующем примере на экран выводится содержимое одного файла:

```
UNIX     $
польз.   cat letter<ENTER>
UNIX     This is a test to see if I am
         entering text in the file "letter".
         Once I have completed it I shall find
         that I have created 4 new lines of data
         $
```

Вывод на экран содержимого нескольких файлов:

```
UNIX     $
польз.   cat b c a<ENTER>
UNIX     bbbbb
         ccccc
         aaaaa
         $
```

Команда `cat` позволяет объединить содержимое нескольких файлов не только на экране, но и поместить сцепление файлов в какой-то другой файл. Пример:

```
UNIX     $
польз.   cat b c a >e<ENTER>
```

```
UNIX     $
польз.   cat e<ENTER>
UNIX     bbbbb
         ccccc
         aaaaa
         $
```

Еще одно применение команды `cat` — ввод текста с клавиатуры в файл. Пример:

```
UNIX     $
польз.   cat >x <ENTER>
         This is a test to see if I can
         enter nev text into the file x.
         <ctrl>&<d>
UNIX     $
```

Обратите внимание, что завершение набора текста здесь производится точно также, как и выход из `UNIX` (командой `<ctrl>&<d>`).

Копирование файла выполняет команда **`cp`**, в качестве первого параметра которой записывается имя копируемого файла, а в качестве второго — имя файла-копии. Пример:

```
UNIX     $
польз.   cp a x<ENTER>
UNIX     $
польз.   cat x<ENTER>
UNIX     aaaaa
         $
```

6. Работа в среде `Midnight Commander`

6.1. Представление на экране файловой структуры

Утилита **`Midnight Commander`** предназначена для того, чтобы предоставить пользователю `UNIX` удобный интерфейс для общения с этой системой. Это обеспечивается, во-первых, наглядным выводом на экран информации о файловой структуре системы. Во-вторых, `Midnight Commander` существенно упрощает для пользователя ввод команд `UNIX`.

Для того, чтобы запустить `Midnight Commander`, достаточно набрать команду `UNIX` — **`mc`**. Часто эту команду включают в командный файл, выполняемый после загрузки системы, и поэтому она выполняется автоматически.

В любом случае в верхней части экрана появляются две серые панели, каждая из которых содержит перечень файлов, «зарегистрированных» в одном из каталогов файловой структуры системы. При этом в заголовке каждой панели указано имя каталога. Ниже располагается командная строка `UNIX` с

обычным ее приглашением и мерцающим курсором. В последней строке экрана находится список клавиш <F1> – <F10> с кратким обозначением их функций.

Каталоги, отображаемые на левой и правой панелях, могут совпадать или не совпадать, но в любом случае одна из панелей, называемая активной, отображает текущий каталог. Заголовок активной панели выделяется белым цветом. Кроме того, одно из имен файлов на активной панели выделено псевдокурсором. В отличие от обычного курсора (он находится в командной строке UNIX) **псевдокурсор** генерируется не аппаратурой, а программой (в графическом режиме экрана). Переключение активной панели производится нажатием клавиши **<Tab>**.

Перемещение псевдокурсора внутри активной панели производится с помощью клавиш управления курсором — вниз, вверх, влево, вправо. Нажатие клавиши **<End>** приводит к установке псевдокурсора на последнюю, а **<Home>** — на первую строку панели. Щелчком левой клавиши мыши можно установить псевдокурсор в любую позицию не только активной, но и соседней панели (происходит смена активной панели).

В общем случае панель содержит строки трех типов:

1 строку “**..**” , обозначающую выход в “родительский каталог” данного каталога;
2 строки с именами подкаталогов данного каталога;
3 строки с именами файлов данного каталога.

В последней строке панели, как правило, записано имя выделенного файла, его длина, дата и время создания или последней модификации.

Для того, чтобы перейти на подкаталог текущего каталога, достаточно установить на него псевдокурсор и нажать <Enter>. Для возврата в родительский каталог требуется установить псевдокурсор на “**..**” и нажать <Enter>. Перемещаясь подобным образом вверх-вниз по файловой структуре логического диска, можно сделать текущим любой каталог на нем.

Прекращение работы Midnight Commander происходит в результате нажатия клавиши **<F10>**. В ответ на появившийся затем вопрос о намерении прекратить работу следует нажать <Enter>.

6.2. Команды для работы с файлами

Midnight Commander предоставляет пользователю команды для работы с файлами, намного более удобные, чем соответствующие команды UNIX. Рассмотрим их.

Создание каталога. Для этого достаточно установить в заголовке активной панели “родительский” каталог по отношению к вновь создаваемому каталогу, а затем нажать клавишу **<F7>**. На экране появится диалоговое окно с приглашением набрать имя нового каталога. В ответ следует набрать имя каталога прописными или строчными буквами и нажать клавишу <Enter>. В результате на активной панели появится имя нового каталога.

Копирование файла. Для этого требуется установить в заголовке одной из панелей “родительский” каталог по отношению к копируемому файлу, а в заголовке другой панели — “родительский” каталог по отношению к файлу-

копии. Далее следует установить псевдокурсор на копируемый файл и нажать клавишу **<F5>**. На экране появится диалоговое окно с сообщением о готовности выполнить копирование. В ответ достаточно нажать клавишу <Enter>. (Для отмены этой или другой команды следует нажать **<Esc>**.) В результате на второй панели появится имя скопированного файла.

Для того, чтобы создать копию файла в том же каталоге, что и исходный файл, необходимо обеспечить, чтобы старый и новый файл имели разные имена. Для этого следует в диалоговом окне, появившемся после нажатия <F5>, набрать имя файла-копии, а уж затем нажать <Enter>.

Копирование каталога выполняется аналогично копированию обычного файла данных. При этом копируется все поддерево файловой структуры, начинающееся с данного каталога.

Копирование нескольких “дочерних” файлов (каталогов) текущего каталога можно ускорить, используя выделение группы файлов. Для выделения файла необходимо установить на его имя псевдокурсор и нажать клавишу **<Ins>**. В результате имя файла высвечивается белым цветом, и оно включается в группу. (Для исключения файла из группы необходимо повторить эти же два действия — установку псевдокурсора и нажатие <Ins>.) После того, как требуемая группа файлов выделена (в нижней строке панели информация об общем числе выделенных файлов и их общем объеме), ее можно скопировать последовательным нажатием <F5> и <Enter>.

Уничтожение файла. Для этого требуется установить псевдокурсор на имя уничтожаемого файла и нажать клавишу **<F8>**. На экране появится диалоговое окно с просьбой подтвердить намерение удалить файл. В ответ достаточно нажать <Enter> (для отмены — <Esc>).

Выделив группу файлов (аналогично выделению при копировании) ее можно уничтожить нажатием <F8> и последующими нажатиями <Enter> в ответ на вопросы Midnight Commander.

Переименование файла. Для этого следует установить псевдокурсор на требуемый файл и нажать клавишу **<F6>**. В появившемся диалоговом окне следует набрать новое имя файла, а затем нажать <Enter>.

Редактирование текстового файла. Для работы с ранее созданным текстовым файлом с помощью редактора **vi** необходимо установить псевдокурсор в каталоге файлов на нужный файл и нажать клавишу **<F4>**.

Создание текстового файла. Для создания нового файла необходимо запустить с командной строки один из текстовых редакторов, например, **ed** или **vi** (этот редактор встроен в Midnight Commander, вызвать встроенный редактор для редактирования нового файла можно, нажав клавиши **<Ctrl>** и **<F4>**).

Примечание. Нажатие функциональной клавиши <Fi> можно заменить последовательным нажатием двух клавиш — <Esc> и <i>, где i — номер функциональной клавиши.

6.3. Ввод команд UNIX

В отличие от рассмотренных выше операций с файлами, для выполнения которых Midnight Commander предоставляет пользователю свои команды, формат остальных команд UNIX остается без изменений. При этом помощь Midnight Commander заключается в ускорении набора этих команд.

Крайний случай — пользователь набирает команду в командной строке UNIX полностью вручную, не пользуясь помощью Commander. Такой метод используется для ввода коротких или редко используемых команд.

Другой метод позволяет обойтись вообще без записи в командную строку. Установив псевдокурсор на имени исполняемого файла (такое имя начинается с символа «*»), следует нажать <Enter>. Данный метод обычно применяется тогда, когда команда не имеет параметров.

Третий метод заключается в том, что команда UNIX переписывается в командную строку из протокола команд. Протокол — список последних команд (не более 16), сохраненный в Commander. Для вывода на экран протокола команд достаточно нажать последовательность клавиш:

<F9> → Command → Command History → Ok

Установив псевдокурсор на требуемую команду в протоколе, следует нажать <Enter>.

Если программа, запускаемая любым способом из Commander, выводит какие-то данные на экран, то чтение их будет невозможно из-за присутствия на экране панелей. Для того, чтобы убрать с экрана обе панели, требуется одновременно нажать **<Ctrl>&<O>**. Для восстановления панелей достаточно опять нажать **<Ctrl>&<O>.**

Удаление (восстановление) только левой панели производится нажатием последовательности клавиш:

<F9> → Left Listing mode → Ok

Удаление (восстановление) только правой панели:

<F9> → Right → Listing mode → Ok

6.4. Настройка Midnight Commander

Commander предоставляет своим пользователям возможность выполнять настройку формата информации, выводимой на экран.

В результате нажатия клавиши **<F9>** в верхней части экрана появляется главное (горизонтальное) меню из пяти пунктов: **Left** (левая), **Files** (файлы), **Commands** (команды), **Options** (параметры), **Right** (правая). Одна из позиций меню выделена псевдокурсором. Для выбора требуемого пункта меню достаточно установить на него псевдокурсор (с помощью клавиш управления курсором) и нажать <Enter>. Это же можно сделать щелчком левой клавиши

мыши. В результате подобного выбора на экране появится ниспадающее меню, выбор в котором позволит выполнить требуемое действие.

Пункт горизонтального меню **Files** позволяет с помощью своих ниспадающих меню выдавать команды по работе с файлами. Многие из этих команд могут быть введены другим способом — нажатием клавиш <F1> — <F10>. Примеры других команд: **Chmod** — установка прав доступа к файлу; **Chown** — замена владельца файла.

В пункте **Command** находится большое количество вспомогательных команд MC. Примеры таких команд: **Find File** — поиск файла; **Swap panels** — переключение панелей; **Show directories size** — показ размера каталогов.

Пункты горизонтального меню **Left** и **Right** предназначены для настройки левой и правой панелей соответственно. При этом, установив в ниспадающем меню режим **Listing mode** режим **Brief** (краткий), мы обеспечим вывод на соответствующей панели лишь одних имен файлов. Задание режима **Full** (полный) позволяет выводить на экран не только имена файлов, но и их основные характеристики (размер в байтах, дату и время создания или последней модификации). **Name**, **Extension**, **Time**, **Size** — группа полей в ниспадающем меню, позволяющая выполнить сортировку имен файлов, выводимых на панель, соответственно по имени, расширению имени, времени создания или модификации, размеру файла. Для интеграции в сетевое пространство предусмотрены пункты **Network link** и **FTP link**.

Пункт горизонтального меню **Options** позволяет настраивать интерфейс MC. Например, в ниспадающем меню можно установить режимы: **Layout** — задание информации, выводимой вне панелей; **Learn Keys** — задание функциональных клавиш, используемых для работы с MC; **Virtual FS** — задание параметров для установления связи. В этом же ниспадающем меню можно выбрать пункт **Configuration** (конфигурация). На экране появится диалоговое окно, в котором можно установить дополнительные параметры настройки Commander.

7. Электронный справочник

Утилита **man** сообщает пользователю UNIX о интересующей его команде, системном вызове, библиотечной функции и о многих других элементах, используемых при работе с этой операционной системой. Для этого достаточно набрать в командной строке слово man, за которым следует название интересующего элемента UNIX. Например, для вывода на экран информации о самом man, достаточно набрать:

```
$ man man
```

Содержание электронного справочника разбито на ряд тематических разделов, которые пронумерованы. Начальные разделы:

4 прикладные утилиты;

5 системные вызовы;
6 библиотечные функции.

Получите на экране информацию о известных вам командах UNIX.

8. Задание

Для успешной сдачи работы требуется выполнить наизусть следующие операции:

1 войти в UNIX с паролем;
2 создать трехуровневое поддерево каталогов;
3 с помощью ed создать в одном из новых каталогов текстовый файл;
4 вывести файл на экран;
5 выполнить добавление текста в начало, в середину и в конец файла;
6 вывести файл на экран;
7 произвести переименование файла;
8 выполнить копирование файла (исходный файл и файл-копия должны располагаться в разных каталогах);
9 удалить созданные файлы и каталоги;
10 выполнить операции 2-9 с помощью Midnight Commander;
11 выйти из UNIX.

Лабораторная работа №2. Дальнейшее знакомство с командами UNIX

1. Цель работы

Целью выполнения настоящей лабораторной работы является развитие навыков работы в среде UNIX: 1) использование в командах shell метасимволов и перенаправление ввода-вывода; 2) запуск конвейеров программ; 3) применение в командах shell переменных; 4) построение командных файлов; 5) изменение прав доступа к файлам.

2. Права доступа к файлам

Наряду с процессами **файлы** являются наиболее важными объектами, находящимися под управлением ОС. Так как UNIX многопользовательская система, то большое значение имеет защита файлов от несанкционированного доступа (преднамеренного или непреднамеренного).

Различают три основных формы доступа к файлу:

чтение файла — **r**;

запись в файл — **w**;

выполнение файла — **x**.

Выполнение файла означает, что после того, как он будет записан в ОП, его содержимое может интерпретироваться ЦП как машинная программа.

Применительно к каталогу перечисленные формы доступа имеют особенный смысл. Операция «чтение каталога» означает, что вы хотите получить перечень имен файлов, записанных в данном каталоге. Получение никакой другой информации о файлах эта форма доступа не предполагает.

Для получения дополнительной информации о файлах, «дочерних» к каталогу, необходимо иметь к нему форму доступа «выполнение». Эта же форма необходима для того, чтобы сделать каталог текущим. Более того, чтобы сделать каталог текущим, необходимо иметь форму доступа x ко всем каталогам в имени-пути данного каталога. Иначе, в данный каталог мы не попадем.

Очевидно, что для создания файла (в том числе и каталога) мы должны иметь форму доступа w к родительскому каталогу. Эта же форма доступа требуется и при удалении файла. Может показаться странным, но для удаления файла какие-то формы доступа к нему самому не требуются. При этом если форма доступа w к файлу есть, команда удаления файла не выдает на экран дополнительного запроса подтвердить удаление файла. Иначе – такой запрос выдается.

Различные пользователи файла должны иметь к нему и различные формы доступа. Для любого файла различают следующие типы пользователей:

1 **суперпользователь** — администратор системы;
2 **владелец** — лицо, создавшее файл;
3 **владелец-группа** — группа лиц, коллективно использующих файл. Обычно, хотя и не всегда, владелец входит в эту группу;
4 остальные пользователи.

Формы доступа к файлу, разрешенные для данного пользователя, называются **правами доступа** этого пользователя. Суперпользователь всегда имеет неограниченные права доступа (`rwx`). Права доступа остальных пользователей могут быть ограничены.

Для того, чтобы узнать права доступа к файлам текущего каталога, воспользуемся командой **`ls`** с флагом **`-l`** . **Флаг** — параметр команды, который может принимать только два значения. Одно из этих значений обозначается буквой, которой предшествует символ «**-**». Второе значение — по умолчанию. Если команда имеет несколько флагов, порядок их перечисления безразличен. Пример:

```
UNIX     $
польз.   ls -l
UNIX     -rw-r--r--  1  vlad gro 1120  Dec15 15:03 abc.txt
         drwxr-xr--  2  vlad gro 11500 Aug28 17:38 mac
         $
```

Информация о файле, выданная UNIX, состоит из 8 колонок. В первой позиции первой колонки записан тип файла (**`d`** — каталог, «**-**» — прочий файл). В остальных девяти позициях первой колонки записаны права доступа к файлу: первые три символа — права доступа владельца; вторые три символа — права доступа владельца-группы; последние три символа — права доступа прочих пользователей.

В приведенном примере для первого файла владелец имеет права доступа `rw` (понятно, что «выполнять» текстовый файл бессмысленно), а все остальные пользователи, включая и членов группы, могут только читать информацию из

файла. Что касается второго файла (это каталог), то относительно его владелец обладает всеми правами доступа; член группы — всеми правами, за исключением записи, а остальные пользователи могут только читать файл.

Что касается остальных колонок информации о файлах, то в них содержится:

1 число **жестких связей** — количество каталогов, в которых «зарегистрирован» данный файл;
2 имя владельца файла;
3 имя владельца-группы;
4 длина файла в байтах;
5 дата последнего изменения файла;
6 время последнего изменения;
7 имя файла.

Владелец файла может не только вывести права доступа на экран, но и внести в них изменения. Для этого используется команда chmod. Ее общий формат:

```
      | u |  | + |  | r |
chmod | g |  | - |  | w| <имя файла> [<имя файла…>]
      | o |  | = |  | x |
      | a |
```

где “**u**” — владелец; “**g**” — группа; “**o**” — остальные пользователи; “**a**” — все пользователи; “**+**” — добавить; “**-**“ — удалить; “**=**” — присвоить.

Пример. Следующая команда лишает членов владельца-группы файла `abcfile` права на запись и выполнение этого файла:

```
chmod g-wx abcfile
```

Применительно к каталогу сочетание прав `r` и `x` позволяет получить интересный эффект, называемый «темным каталогом». Он заключается в том, что для всех пользователей, кроме владельца, запрещено чтение каталога, но разрешено его выполнение. Поэтому только «посвященные» пользователи, знающие о наличии файла в каталоге, имеют возможность работать с ним.

Проанализируйте права доступа в своем корневом каталоге и создайте в нем «темный» подкаталог.

3. Перенаправление ввода-вывода

Набор вами на клавиатуре любой команды `UNIX` приводит к запуску соответствующей программы (прикладной программы, утилиты или лингвистического процессора). Запущенная программа выполняет преобразование и (или) перенос своих входных данных, получая при этом свои выходные данные. Откуда берет программа свои входные данные? Она считывает их или с устройства ввода-вывода, или из файла на диске. Между этими вариантами нет особой разницы, если вспомним, что устройства ввода-

вывода в UNIX «маскируются» под файлы (эти файлы находятся в каталоге /dev).

Обычным (стандартным) источником входных данных для программы является клавиатура. В частности, на клавиатуре набираются параметры команды — наборы символов, отделенные от кода команды и друг от друга минимум одним пробелом. Стандартным получателем выходных данных программы является экран.

Как и при работе с другими файлами, экран и клавиатура должны быть «открыты» прежде, чем программа начнет их использовать. Одним из результатов операции открытия файла является присвоение ему номера (индекса), уникального для выполняемой программы. Получив индекс файла, программа может использовать его в качестве заменителя имени файла. В частности, клавиатура получает номер 0, а экран «открывается» дважды — с номерами 1 и 2. «Файл» 1 считается стандартным получателем обычных выходных данных программы, а «файл» 2 — стандартным получателем выходного сообщения программы в случае ее ошибочного завершения. Подобное представление одного и того же экрана в виде двух файлов не приводит к противоречиям, так как выдача данных будет произведена программой только в один из этих файлов.

Файлы стандартного ввода (0), стандартного вывода (1) и стандартного вывода ошибки (2) могут быть заменены в команде на другие файлы. Такая операция называется **перенаправлением ввода-вывода**

Для перенаправления ввода программы, в команде записывается выражение: «< имя файла». В качестве примера рассмотрим команду cat. Записанная без параметров, она выводит на экран символьную строку, набранную на клавиатуре. Теперь сравним два варианта применения этой команды, которые делают одно и то же (выводят на экран содержимое файла file1):

```
$ cat file1
$ cat < file1
```

В первом случае имя файла передается в программу cat в качестве ее входного параметра. Во втором случае программа cat никакого входного параметра не получает. Просто до ее вызова UNIX перенаправляет стандартный вход с клавиатуры на файл file1.

Для перенаправления стандартного вывода программы, в команду ее вызова помещается конструкция «> имя файла». Пример:

```
$ cat > file2
```

Данная команда записывает текст, набранный на клавиатуре (до нажатия <ctrl>&<d>), в файл file2. При этом если файл file2 ранее не существовал, то он создается. Иначе — прежнее содержимое файла file2 теряется. Заметим, что в данном примере программа cat выполняет функции

простейшего текстового редактора, позволяя быстро создавать текстовые файлы.

Для того, чтобы прежнее содержимое файла не было уничтожено, и чтобы новый текст добавился к нему, используется конструкция «>> имя файла». В следующем примере набранные на клавиатуре символы добавляются в конец файла file2:

```
$ cat >> file2
```

В следующих примерах команда cat выполняет копирование файла file1 в файл file2 (файл file1 при этом сохраняется):

```
$ cat file1 > file2
$ cat < file1 > file2
```

Во втором примере используются две операции перенаправления ввода-вывода. Результат не изменится, если эти операции поменять местами. Отсутствие путаницы объясняется тем, что имя файла всегда записывается справа от операции перенаправления.

Слева от операции перенаправления может быть записан номер того файла, который перенаправляется (по умолчанию для «<» это 0, а для «>» — 1). Например, для подавления вывода об ошибках некоторой программы run, ее можно вызвать следующим образом:

```
$ run 2 > /dev/null
```

где /dev/null является псевдоустройством, удаляющим все введенные в него символы.

Для перенаправления ввода программы иногда полезна конструкция: «<< слово». Ее наличие в команде означает, что текст, следующий за ней, используется далее в качестве входного файла программы. Конец этого файла определяется вторичным появлением этого слова. Пример:

```
UNIX     $
польз.   cat > file2 << ! <ENTER>
         This is a test to see if I am<ENTER>
         entering text in the file “letter”. <ENTER>
         Once I have completed it I shall find<ENTER>
         that I have created 4 new lines of data<ENTER>
         ! <ENTER>
```

В этом примере текст, набранный на клавиатуре и заканчивающийся символом «!», помещается в качестве нового содержимого файла file2. Таким образом программа cat опять работает в качестве простого текстового редактора.

Выполните команду cat, используя различные операции перенаправления ввода-вывода.

4. Конвейеры

На основе простых команд UNIX можно создавать более сложные команды. **Конвейер** — сложная команда UNIX, полученная соединением двух или большего числа команд символом «|». При этом выходные данные команды, стоящей слева от этого символа, являются входными данными для команды, расположенной правее. Иными словами в качестве выходного файла одной программы используется входной файл для другой. Пример конвейера:

```
$ ls -l | sort -r<ENTER>
```

Команда **ls** выдает в свой выходной файл перечень имен файлов в текущем каталоге. (Флаг **-l** задает выдачу имен файлов в один столбец.) Команда **sort** сортирует строки своего входного файла. При этом наличие флага **-r** требует, чтобы порядок сортировки был обратным. Этот же эффект можно было бы достичь следующими двумя командами:

```
$ ls -l >file1; sort -r <file1<ENTER>
```

Пример конвейера из трех команд:

```
$ ls -1 | fgrep ".txt" | wc <ENTER>
```

Команда **fgrep** определяет наличие в своем входном файле заданной последовательности символов (в примере — `.txt`). Все строки входного файла, в которых заданная последовательность найдена, помещаются в выходной файл этой программы.

Команда **wc** выдает статистику о своем входном файле. Флаги этой команды: **-l** — вывод числа строк; **-w** — вывод числа слов; **-c** — вывод числа символов. По умолчанию все три флага установлены (`-lwc`). Поэтому флаги записываются в этой команде только тогда, когда требуется ограничить выходную статистику. Следует отметить, что данная команда может применяться для выдачи статистики о нескольких файлах. В этом случае имена этих файлов задаются в качестве параметров команды `wc`.

Результатом выполнения приведенного выше конвейера является вывод на экран числа строк, слов и символов в файле, строки которого являются именами файлов с расширением `.txt` текущего каталога.

Поместите в один из своих каталогов несколько файлов и выполните для него приведенные выше конвейеры.

5. Применение метасимволов в командах

Многие команды позволяют задавать в качестве своих параметров не один, а несколько имен файлов. Для облегчения набора этих имен используются метасимволы. **Метасимвол** — служебный символ, обозначающий в имени файла один или несколько обычных символов. Это следующие метасимволы:

***** — заменяет в имени файла любое число любых символов;

? — заменяет в имени файла один (и только один) любой символ;

[ ] — заменяют в имени файла один символ, удовлетворяющий списку символов в квадратных скобках. Список может быть задан перечислением допустимых значений символа (без всяких разделительных символов) или задан в виде диапазона изменения символа (первое и последнее допустимое значения, разделенные «**-**»)

В имени файла метасимволы могут быть записаны любое число раз.

Пример. Следующая команда выводит на экран (в один столбец) список файлов заданного каталога, имеющих расширение `.txt` :

```
$ ls -l /usr/vlad/*.txt<ENTER>
```

Пример. Команда выводит на экран список файлов заданного каталога, имеющих в имени слово “`abcd`” :

```
$ ls -l /usr/vlad/*abcd*<ENTER>
```

Пример. Следующая команда выводит на экран список файлов текущего каталога, имена которых начинаются с набора символов “`file`”, за которым стоит произвольный символ:

```
$ ls -l file?<ENTER>
```

Пример. Следующая команда выводит на экран список файлов текущего каталога, имена которых начинаются с набора символов “`file`”, за которым стоит символ `a`, `b` или `c`:

```
$ ls -1 file[abc]<ENTER>
```

Пример. Следующая команда выводит на экран список файлов текущего каталога, имена которых начинаются с набора символов “`file`”, за которым стоит любой символ в диапозоне от `a` до `z`:

```
$ ls -1 file[a-z]<ENTER>
```

Выполните команду `ls` с различными метасимволами для ранее созданного вашего каталога.

6. Командный интерпретатор и командные файлы

После того, как мы набрали команду `UNIX` и нажали `<ENTER>`, эта команда поступает на вход командного интерпретатора `UNIX`, имеющего название **`shell`**. Это название собирательное, объединяющее целый ряд реальных интерпретаторов. Некоторые из них:

1 **`Bourne shell`** (`/bin/sh`);

2 **`C shell`** (`/bin/csh`);

3 **Korn shell** (/bin/ksh);

4 **Bourne Again shell** (/usr/local/bin/bash).

Имя используемого командного интерпретатора выводится на экран сразу же после вашего входа в UNIX. Далее на экране появляется приглашение **shell** к вводу команды. Обычно это символ “$”. Вводить команды можно по одной, в виде конвейера, а также в виде обычной последовательности команд, размещенных в одной строке и разделенных символом «;». Кроме того, в командной строке можно задать имя командного файла.

Командный файл — файл, содержащий список команд shell. Другие названия таких файлов: **скрипты** и **сценарии**. Применение таких файлов позволяет избежать повторения набора часто используемых команд. (Аналогом скрипта UNIX, в операционной системе MS-DOS является файл с расширением .bat). Так как по своей форме командный файл представляет собой обыкновенный текстовый файл, то для его получения и редактирования может быть использован любой текстовый редактор, например, ed.

Примером командного файла является **.profile**. Это «инициализационный» файл, который выполняется сразу же после вашего входа в систему. Файл .profile записывается самим пользователем в свой корневой каталог. (Данный файл является аналогом файла autoexec.bat в MS-DOS). Он может содержать, например, приглашение к последующей работе, указание путей поиска файлов, «переделку» приглашений shell. Рассмотрим выполнение этих действий.

Для выдачи приглашения удобно использовать команду echo. Она выводит в свой выходной файл (на экран) последовательность символов, заданную в качестве параметра этой команды. Пример:

```
$ echo “Good morning !”<ENTER>
```

В результате на экран будет выведена строка: Good morning !

Многие возможности командных файлов реализуются с помощью переменных. **Переменная** — символьная строка, которой присвоено имя. **Имя переменной** — совокупность букв, цифр и знаков подчеркивания. Причем имя не может начинаться с цифры. Строка-значение переменной не может иметь пробелов.

Определение переменной производится одновременно с присваиванием этой переменной значения с помощью оператора присваивания “=”. При этом имя переменной стоит слева, а значение — справа от “=”. Оператор присваивания может размещаться как в командной строке, так и в командном файле. Например, в следующем операторе присваивания переменной var1 присвоено значение length:

```
var1 = length
```

Самой переменной length также может быть присвоено значение:

```
length = 40
```

Если переменной присваивается значение символьной строки, то эту строку следует заключить в кавычки, например:

```
message = “Hello”
```

Полученное значение переменной может использоваться в последующих командах. Для этого имени переменной в команде должен предшествовать символ $. Пример:

```
$ var=user1<ENTER>
$ echo $var<ENTER>
user1
$ echo ${var}1<ENTER>
user11
$
```

В данном примере показано, что для отделения имени переменной от последующих символов это имя следует заключать в фигурные скобки.

Некоторые переменные shell являются зарезервированными. Они используются для хранения состояния командного интерпретатора и запускаемых им программ. Вот некоторые из этих переменных:

5 **HOME** — полное имя корневого каталога пользователя;

6 **PATH** — путь в файловой структуре, используемый для поиска файлов. Если требуется указать несколько путей, то они разделяются символом «**:**». Для задания текущего каталога используется символ «**.**»;

7 **PS1** — приглашение shell в командной строке (обычно — $);

8 **PS2** — вторичное приглашение shell в командной строке (обычно — >). Выводится на экран в том случае, когда вводимая вами команда занимает более одной строки.

Меняя значения этих и других зарезервированных переменных, можно влиять на работу командного интерпретатора. Обычно это делают в файле .profile.

Кроме переменных в командных файлах используются **параметры.** Параметр отличается от переменной своим именем. Например, параметр с именем 0 обозначает в скрипте имя этого же скрипта, а параметры с именами 1,2,…,9 используются для передачи информации на вход данного командного файла. Как и для переменных, для записи значения параметра перед его именем записывается символ $.

Пример. Следующий командный файл выполняет вывод на экран своего имени, а также вывод своих входных параметров:

```
echo script $0
echo $1 $2 $3
```

Запуск скрипта может быть выполнен из командной строки аналогично запуску простой команды: в ответ на приглашение `shell` набирается имя скрипта, за которым следуют его параметры, разделенные пробелами. Например, пусть приведенный выше скрипт имеет имя `test1.sh`, тогда он может быть запущен следующим образом:

```
$ test1.sh a1 a2 a3
script test1.sh
a1 a2 a3
```

где последние две строки на экране есть результат работы скрипта.

Примененный способ запуска командного файла требует наличия права доступа к нему «выполнение». Другой способ запуска заключается в использовании в качестве имени команды самого `shell`, а в качестве параметра этой команды — имени скрипта. В этом случае не требуется «выполнение» скрипта, а лишь его «чтение». Пример:

```
$ bash test1.sh a1 a2 a3
script test1.sh
a1 a2 a3
```

Третий способ запуска скрипта: из другого командного файла аналогично обычной команде. В этом случае запускаемый скрипт называется **вложенным,** а запускающий – **главным скриптом**

7. Задание

Выполнить (желательно наизусть) следующую последовательность действий:

1 создать два каталога и поместить в один из них четыре текстовых файла, два из которых имеют в своем имени одинаковую символьную последовательность, называемую далее «словом»;

2 поместить во второй каталог скрипт, имеющий два входных параметра: имя каталога и набор символов. Скрипт выполняет действия:
   - 2.1 вывод на экран перечня файлов, «дочерних» к заданному каталогу, которые имеют в своем имени заданный набор символов;
   - 2.2 уничтожение всех остальных файлов заданного каталога;
   - 2.3 любые другие действия (по вашему желанию);

3 создать свой инициализационный скрипт, выполняющий действия:
   - 3.1 здоровается;
   - 3.2 «переделывает» приглашения `shell`;
   - 3.3 запускает вложенный скрипт, созданный в (2), задав ему в качестве параметров каталог и «слово» из (1);

4 любые другие действия (по вашему желанию);

5 выйти из UNIX, а затем войти вновь с целью демонстрации результатов выполнения инициализационного скрипта.

Примечание 1. При выполнении задания разрешается использовать только те средства shell, которые рассмотрены в данной и предыдущей работах. В частности нельзя применять управляющие операторы (рассматриваются в следующей работе).

Примечание 2. Для избирательного удаления файлов в (б) рекомендуется использовать команду rm с флагом –i, предварительно установив права доступа к файлам. При этом, для обеспечения автоматического выполнения rm, ее стандартный ввод должен быть переключен на вспомогательный файл, содержащий любой символ кроме “y”.

Лабораторная работа №3. Управляющие операторы командного языка

1 Цель работы

Целью выполнения настоящей лабораторной работы является развитие навыков программирования на языке shell путем использования в скриптах управляющих операторов if, case, for, while, until.

2. Двухальтернативный выбор

По сравнению с командными языками других ОС, язык shell наиболее приближается к обычным алгоритмическим языкам программирования по своим выразительным возможностям. В частности, он имеет управляющие операторы, позволяющие изменять последовательный ход выполнения программы. Сюда относятся **операторы выбора** if и case, **операторы цикла** while, until и for.

Оператор двухальтернативного выбора if позволяет выполнить одну из двух взаимоисключающих последовательностей команд shell, в зависимости от выполнения условия:

```
if <условие>
then
    команда 1
    ...
    команда N
else
    команда (N+1)
    ...
    команда M
fi
```

Выполнение условия обозначается цифрой **0** — **истина**, а невыполнение: **1** — **ложь**. (В языках программирования обычно используют противоположные обозначения.) В качестве условия может быть записана любая (или почти

любая) команда shell (в том числе скрипт или конвейер). При этом выполнение условия определяется кодом завершения команды.

Код завершения — целое число, возвращаемое командой после своего завершения. При успешном завершении команды код завершения всегда равен 0. Ненулевой код завершения говорит или об ошибочном, или о необычном завершении команды. Например, команда ls (вывод содержимого каталога) выдает код завершения 0, если информация обо всех файлах, указанных в команде, успешно выведена. Иначе она выдает код завершения >0 (причины: указанный файл не найден; нет права чтения указанного файла; неверно задан параметр команды). Если в качестве условия задан скрипт (или конвейер) то выполнение условия определяется кодом завершения последней команды скрипта (конвейера).

Пример. Допустим, что мы хотим найти в текущем каталоге все файлы с расширением .txt и вывести их имена на экран. Это можно сделать с помощью следующих команд shell:

```
$ if ls *.txt >file1
then cat file1
else echo "Good morning !"
fi
```

Наиболее часто в качестве условия используется команда **test**. Эта команда позволяет обнаружить наличие или отсутствие какого-то свойства у файла или у символьной строки, а также позволяет сравнить между собой две строки или два целых числа. При этом проверяемое свойство, операция сравнения, файл, строка (строки), число (числа) записываются в качестве параметров команды test. Некоторые из подобных параметров:

-s file размер файла file больше 0
-r file для файла file разрешен доступ на чтение
-w file для файла file разрешен доступ на запись
-x file для файла file разрешено выполнение
-f file файл file существует и является обычным файлом
-d file файл file существует и является каталогом
-z string строка string имеет нулевую длину
-n string строка string имеет ненулевую длину
string1 **=** string2 две строки идентичны
string1 **!=** string2 две строки различны
i1 **-eq** i2 число i1 равно числу i2
i1 **-ne** i2 число i1 неравно числу i2
i1 **-lt** i2 число i1 строго меньше числа i2
i1 **-le** i2 число i1 меньше или равно числу i2

Результатом выполнения команды test является код завершения: 0 — условие выполняется, 1 — не выполняется. В качестве примера запишем команды, выполняющие то же самое, что и последовательность команд в предыдущем примере:

```
$ ls *.txt >file1
$ if test -s file1
> then cat file1
> else echo “Good morning !”
> fi
```

При задании команды `test` вместо слова «`test`» можно использовать квадратные скобки, но этот способ работает не во всех интерпретаторах (точно работает в `bash`). Например, приведенную выше последовательность команд можно записать в виде:

```
$ ls *.txt >file1
$ if [-s file1]
> then cat file1
> else echo “Good morning !”
> fi
```

Если команды после слова `else` отсутствуют, то можно применить сокращенную форму команды `if`:

```
if <условие>
then
    команда 1
    ...
    команда N
fi
```

Пример:

```
$ if ls *.txt >file1
> then cat file1
> fi
```

Нетрудно заметить, что данную совокупность команд можно заменить всего одной командой:

```
$ ls *.txt
```

Выполните приведенные выше примеры применения оператора `if`.

3. Многоальтернативный выбор

Оператор многоальтернативного выбора **`case`** позволяет выбрать для выполнения одну из нескольких последовательностей команд. Структура оператора:

```
case <слово> in
    <шаблон1>)
```

```
        команда
        ...
        ;;
    <шаблон2>)
        команда
        ...
        ;;
    ...
    *)
    команда1
    ...
    ;;
esac
```

Слово — набор символов без пробелов. Оно поступает на вход оператора `case` и последовательно сравнивается с шаблонами. **Шаблон** — слово, которое может иметь наряду с обычными символами метасимволы (`?`, `*`, `[ ]`). Если входное слово удовлетворяет первому шаблону, то выполняется первая последовательность команд, после чего делается выход из оператора `case`. Иначе входное слово сравнивается со вторым шаблоном и так далее. Если при этом обнаружится, что входное слово не отвечает ни одному из шаблонов, то выполняется последовательность команд, которой предшествуют символы «`*)`». Обратим внимание, что любая последовательность команд заканчивается двумя символами «`;;`».

В следующем примере в качестве входного слова используется имя файла — содержимое переменной `name`. Оно используется для задания операции с файлом:

```
$ name = /usr/vlad/abc.txt
$ case $name in
>  *.txt) ed $name ;;
>  *.sh) bash $name ;;
>  *) wc $name ;;
> esac
```

В результате выполнения данного оператора `case` заданный файл обрабатывается или текстовым редактором, или командным интерпретатором, или же команда `wc` выдаст о нем статистику. (Применительно к конкретному файлу `/usr/vlad/abc.txt` будет инициирован редактор `ed`.)

Гибкость приведенной выше последовательности команд можно значительно повысить, если для задания значения переменной `name` использовать не оператор присваивания «=», а вводить это значение с клавиатуры. Для этого можно использовать команду ввода:

```
read перем.1 перем.2 ... перем.N
```

Данная команда выполняет ввод с клавиатуры строки символов. (Строка — набор символов, заканчивающийся символом <ENTER>). При этом первое слово введенной строки записывается в качестве содержимого первой переменной, второе слово – в качестве содержимого второй переменной и так далее. Если число слов больше числа переменных, то все оставшиеся слова записываются в качестве содержимого пераменной, записанной последней в списке параметров команды read. А если, наоборот, число переменных больше, то последние переменные остаются «пустыми». Пример:

```
$ read name
$ case $name in
>  *.txt) ed $name ;;
>  *.sh) bash $name ;;
>  *) wc $name ;;
> esac
```

Так как в данном примере команда read имеет всего одну переменную, то набираемая на клавиатуре строка должна содержать ровно одно слово. Так как иначе все набранные слова станут содержимым переменной name, что недопустимо при задании имени файла.

С помощью команды **read** можно обеспечить диалоговое взаимодействие между пользователем и командным файлом (точнее — с shell, выполняющим файл). **Создайте** небольшой скрипт, выполняющий вывод на экран простого меню. После этого производится ввод с клавиатуры номера варианта и выполнение соответствующего действия.

4. Цикл с перечислением

Данный цикл выполняет оператор **for**. Его структура:

```
for <переменная> in <список>
do
    команда 1
    ...
    команда N
done
```

Данный цикл выполняется столько раз, сколько слов в списке. При этом указанная переменная последовательно принимает значения, равные словам из списка. Пример:

```
$ cat > file1
This is a test to see if I am
entering text in the file “letter”.
Once I have completed it I shall find
that I have created 4 new lines of data
<ctrl>&<d>
```

```
$ for str1 in 4 text test
> do
>  echo ${str1}:
>  fgrep $str1 file1
> done
```

В данном примере оператор `cat` выполняет ввод с клавиатуры текста и записывает его в качестве содержимого файла `file1`. Далее выполняется оператор `for`, имеющий три слова в списке (`4`, `text`, `test`). Поэтому трижды выполняется совокупность операторов `echo` и `fgrep`. Каждый раз при этом переменная `str1` принимает значение соответствующего слова. В результате на экран будет выведено:

```
4:
that I have created 4 new lines of data
text:
entering text in the file “letter”.
test:
This is a test to see if I am
```

Выполните поиск в введенном с клавиатуры тексте строк с другими ключевыми словами.

5. Цикл с условием

Данный цикл реализуется оператором **`while`**. Его структура:

```
while <условие>
do
    команда 1
    ...
    команда N
done
```

Аналогично оператору `if`, условие задается одной из команд `shell`. Пока эта команда возвращает код возврата, равный `0`, повторяется последовательность команд, заключенная между словами **`do`** и **`done`**. (При этом возможна ситуация, когда тело цикла не будет выполнено ни разу.) Пример:

```
$ while read var1 var2
> do
>  case $var1 in
>   1) echo $var2 >>file1;;
>   2) echo $var2 >>file2;;
>   *) echo $var2 >>file3;;
```

```
>  esac
> done
```

Особенностью приведенного примера является использование вложенных управляющих структур: оператор выбора `case` вложен в оператор цикла `while`. Выполнение цикла начинается с выполнения оператора `read`, который вводит строку с клавиатуры, записывая ее первое слово в качестве содержимого переменной `var1`, а все последующие слова — в качестве содержимого `var2`. Допустим, что ввод строки символов завершился успешно и команда `read` выдала код возврата `0`. В этом случае в зависимости от значения переменной `var1` (`1`, `2` или любое другое значение) содержимое введенной строки (за исключением первого слова) записывается в один из трех файлов.

Выполнение данного цикла продолжается до тех пор, пока вместо набора очередной строки, вы не наберете комбинацию клавиш `<ctrl>&<d>`, что означает для файла-клавиатуры «конец файла». В этом случае команда `read` возвратит ненулевой код возврата, и выполнение цикла завершится.

Перенаправив стандартный ввод для команды `read`, ее можно использовать для считывания значений переменных из любого текстового файла. Как и при считывании с клавиатуры, одно выполнение `read` обеспечивает чтение из файла одной строки — последовательности слов до кода `<ENTER>`. Встретив конец файла, `read` возвращает ненулевой код возврата. Например, переделаем записанную выше совокупность команд для обработки строк файла `file`:

```
$ while read var1 var2 <file
> do
>  case $var1 in
>   1) echo $var2 >>file1;;
>   2) echo $var2 >>file2;;
>   *) echo $var2 >>file3;;
>  esac
> done
```

Выполните «разделение» любого текстового файла, используя приведенную выше последовательность команд.

6. Цикл с инверсным условием

Данный цикл реализуется оператором **`until`**. Его структура:

```
until <условие>
do
    команда 1
    ...
    команда N
done
```

Команды, заключенные между **do** и **done**, повторяются до тех пор, пока заданное условие не выполняется. Первое же выполнение условия означает выход из цикла. (При этом возможна ситуация, когда тело цикла не будет выполнено ни разу.) Нетрудно заметить, что операторы `while` и `until` будут выполнять одно и то же, если условие одного из них противоположно условию другого.

Пример. Приведенная ниже последовательность команд выполняет то же самое, что и пример использования `while`, с тем отличием, что завершение ввода определяется не нажатием клавиш <ctrl>&<d>, а вводом какого-то слова, например, слова «!!»

```
$ until [var1 = !!]
> do
>  read var1 var2
>  case $var1 in
>   1) echo $var2 >>file1;;
>   2) echo $var2 >>file2;;
>   *) echo $var2 >>file3;;
>  esac
> done
```

Обратите внимание, что в качестве условия записана команда `test`.

7. Задание

Требуется разработать программу на языке `shell` (без использования команды `find`), выполняющую поиск в заданном поддереве файловой структуры всех файлов, имена которых отвечают заданному шаблону. Результатом работы программы является перечень имен искомых файлов на экране.

Примечание. Программа состоит из двух скриптов. Главный скрипт выполняет вывод на экран приглашения ввести с клавиатуры имя-путь начального каталога и шаблон поиска. Далее он выполняет ввод этих данных с клавиатуры и выводит на экран перечень искомых файлов в начальном каталоге поиска (если они там есть). Затем он вызывает для каждого подкаталога вложенный скрипт, передав ему два входных параметра: 1) относительное имя подкаталога; 2) шаблон поиска.

Вложенный скрипт выполняет поиск в заданном каталоге искомых файлов, а для каждого подкаталога вызывает точно такой же скрипт. (При выполнении любого скрипта запускается новый экземпляр `shell`, поэтому рекурсивное выполнение скриптов не приводит к каким-либо трудностям.)

Лабораторная работа №4. Процессы в UNIX

1. Цель работы

Целью выполнения настоящей лабораторной работы является развитие навыков работы в среде UNIX: 1) управление программными процессами, используя команды shell; 2) обмен сообщениями с другими пользователями этой же системы UNIX.

2. Понятие процесса

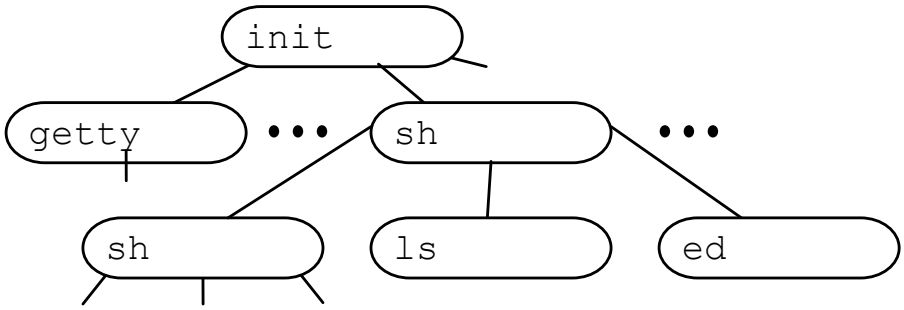
Важнейшим понятием операционной системы UNIX является понятие процесса. Краткое определение: **процесс** — выполняющаяся программа. Более подробное определение: **процесс** — программа, располагающая окружением, достаточным для своего выполнения. **Окружение программы** образуют системные программы и системные структуры данных, обслуживающие данную программу. Эти программы и структуры данных реализуют **виртуальную машину прикладной программы.**

Так как UNIX является мультипрограммной системой, то в любой момент времени в ней существует не один, а много процессов. Для того чтобы различать эти процессы между собой, каждому из них UNIX присваивает номер, уникальный для всей системы. Этот номер называется **идентификатором процесса (PID)**. При появлении в системе нового процесса, ему присваивается следующий (по возрастанию) свободный номер. Как только будет распределен наибольший в системе номер процесса, следующему процессу будет назначен идентификатор, наименьший из освободившихся к данному моменту времени.

За исключением нескольких системных процессов, любой процесс в UNIX имеет «родителя». Это — процесс, создавший данный процесс. Общим предком всех порождаемых процессов является процесс **init** (PID=1), созданный в результате начальной загрузки системы. Процесс init порождает для каждого терминала, зарегистрированного в системе, свой процесс **getty**, выполняющий ожидание включения соответствующего терминала.

Дождавшись подсоединения к терминалу, процесс getty переходит на выполнение программы **login**, запрашивающей у пользователя его имя и пароль. После этого процесс login (бывший getty) запускает программу, указанную в последнем поле записи пользователя в файле паролей /etc/passwd. Это может быть имя любой программы, но обычно оно соответствует одному из интерпретаторов shell. В результате процесс, выполнявший программу login (а еще раньше — getty), переходит на выполнение программы shell.

Если в ответ на приглашение shell вы набираете в командной строке имя программы, это приводит к созданию и инициированию соответствующего программного процесса. В результате создается целое дерево процессов. Фрагмент этого дерева:



Как видно из рисунка, процесс-shell может порождать другие процессы-shell. Это происходит, если shell-отец встретил команду, являющуюся именем скрипта. При этом для выполнения скрипта всегда создается новый процесс-shell.

3. Запуск процессов в фоновом режиме

Что делает процесс-отец, после того, как он инициировал дочерний процесс? Основные два способа его поведения сводятся к следующему. Во-первых, он может ничего не делать, а просто ждать завершения дочернего процесса. При этом дальнейшее поведение процесса-отца может определяться полученным кодом завершения дочернего процесса. Именно такой способ поведения процесса-отца (shell), использовался нами во всех предыдущих работах.

Второй способ поведения процесса-отца: после запуска дочернего процесса он продолжает свое выполнение. Для этого дочерний процесс должен быть запущен не в основном (оперативном), а в **фоновом режиме**. В этом режиме процесс не использует терминал, а выполняет ввод-вывод, используя другие файлы. Для подобного запуска обычная команда shell должна завершаться символом **&**. В следующем примере программа prog3 запускается в фоновом режиме:

```
$ prog3 >file1 &<ENTER>
205
$
```

где 205 — номер созданного процесса. В ответ на приглашение shell можно выполнить набор следующей команды.

Запустите процесс с бесконечным циклом (используя оператор while) в фоновом режиме. Выведите на экран информацию об этом процессе.

4. Получение информации о процессах

Для получения такой информации используется команда **ps**. Ее применение без флагов позволяет вывести на экран минимум информации о ваших процессах. Пример:

```
$ ps
PID   TTY    TIME  CMD
145    2     0:01 sh
```

```
313    2     0:03 ed
431    2     0:01 ps
$
```

В выдаче команды:

1. TTY — номер терминала (co — операторская консоль; ? — процесс не управляется терминалом);
2. TIME — затраты времени ЦП на выполнение процесса;
3. CMD — имя команды, для выполнения которой процесс создан.

Применение команды **ps** с флагом **-l** позволяет вывести на экран полную информацию о ваших процессах. Пример:

```
$ ps -l
F S UID  PID PPID CPU PRI NICE TTY TIME CMD
1 S vlad 701 683    0   8   10 P1  0:00.04 bash
1 R vlad 712 701  137 133   20 P1  0:54.04 sh sc1
1 R vlad 713 701  288 132   10 P1  2:37.91 sh c1
1 O vlad 720 701    0  96   10 P1  0:00:00 ps -l
```

В выдаче команды появились новые столбцы:

1. F — комбинация битов (записанная в восьмеричной системе), определяющая тип процесса (1 — программа процесса находится в ОП; 2 — системный процесс; 4 — программа процесса занимает фиксированное место в ОП (используется в процессах-драйверах); 10 — программа процесса выгружена из ОП; 20 — процесс трассируется другим процессом). Например, если F=3, то это означает, что процесс системный и его программа находится в ОП;
2. S — состояние процесса (O — «Задача»; S — «Сон»; R — «Готов»; I — создается; Z — «Зомби»);
3. UID — имя пользователя-владельца процесса;
4. PPID — номер процесса-родителя;
5. CPU — степень использования ЦП;
6. PRI — текущий приоритет процесса;
7. NICE — относительный пользовательский приоритет процесса. Значение 10 в данном столбце означает, что пользовательский приоритет имеет значение по умолчанию;
8. ADDR — адрес программы процесса (в ОП — для резидентных процессов; на диске (в области свопинга) — для остальных процессов);
9. SZ — размер программы процесса в блоках (по 512 байт);
10. WCHAN — номер события, которого ожидает процесс в состоянии сна.

Последние три столбца из перечисленных, на рисунке не показаны. Что касается остальных столбцов, то мы рассмотрим их более внимательно. Во-первых, заметим, что выдача команды ps отражает состояние процессов на

момент времени, непосредственно предшествующий попаданию на ЦП той машинной команды программы ps, которая реализует системный вызов для вывода на экран. Во-вторых, на этот момент в системе существовали четыре процесса, принадлежащих данному пользователю vlad:

1 интерактивный процесс shell (программа bash), выполняющий обслуживание пользователя в оперативном режиме. На рассматриваемый момент времени данный процесс находился в состоянии “Сон”, так как ожидал завершения своего дочернего процесса, выполняющего в оперативном режиме программу ps;

2 два фоновых процесса, порожденных оперативным shell в ответ на команды пользователя. Каждый из этих двух процессов выполняет свой shell, производящий интерпретацию одного и того же скрипта sc1, содержащего бесконечный цикл. Оба фоновых процесса запущены пользователем с разными пользовательскими приоритетами, но в один и тот же момент времени, с помощью следующего конвейера:

```
$ nice -10 sh sc1 | sh sc1 &;
```

На момент получения выдачи программой ps, оба процесса находились в состоянии “Готов”;

3 интерактивный процесс выполнения программы ps в оперативном режиме. На рассматриваемый момент времени данный процесс находился в состоянии «Задача».

Для того, чтобы получить информацию о всех процессах системы, а не только о своих, используются флаги **–a** и **–x**. При этом флаг **–a** задает выдачу информации о процессах, управляемых с терминала, а флаг **–x** об остальных процессах.

Если нам нужна информация не о всех, а только о некоторых процессах, мы можем выделить эти процессы с помощью конвейера команд ps и fgrep. Например, следующий конвейер выводит информацию о процессах, дочерних процессу с номером 325, а также информацию о нем самом:

```
$ ps -ax | fgrep “325”
```

5. Приоритеты процессов

Как мы видели ранее, команда ps с флагом –l выводит на экран три разных приоритета процесса: CPU, PRI и NICE. Они используются при распределении времени ЦП между параллельными (то есть одновременно существующими) процессами по принципу: чем больше число-приоритет, тем меньше времени ЦП получит данный процесс. Краткая формулировка этого же принципа: «чем приоритет больше, тем он хуже».

PRI — текущий приоритет процесса на данный момент времени. Именно он используется при назначении времени ЦП процессу. Чем PRI больше, тем данный процесс будет реже выбираться среди других готовых процессов для

выполнения на ЦП. А величина выделяемого при этом кванта времени, скорее всего, будет короче. Среди других факторов на величину PRI влияют величины **CPU** (системная составляющая приоритета) и **NICE** (пользовательская составляющая приоритета).

UNIX предоставляет своим пользователям две команды, позволяющие изменять NICE (а следовательно и PRI) у требуемого процесса. При этом следует отметить, что обычный пользователь может только ухудшать NICE у своих процессов. А администратор системы может не только ухудшать, но и улучшать NICE для любого процесса системы.

Команда **nice** используется для запуска процесса с приоритетом NICE, отличном от принятого по умолчанию. В следующем примере программа prog5 (а точнее, соответствующий процесс) запускается с NICE, ухудшенным на 10:

```
$ nice -10 prog5
```

Пример команды, улучшающей NICE (команда доступна только администратору):

```
$ nice --10 prog5
```

Команда **renice** позволяет задать NICE для уже существующего процесса. В следующем примере процессу с номером 168 присваивается NICE, равная 15:

```
$ renice 15 168
```

Проверьте уровни приоритета у ранее запущенного процесса с бесконечным циклом до и после применения команды renice.

6. Уничтожение процессов

Как следует из вышесказанного, в текущий момент времени могут существовать несколько активных (то есть находящихся в состоянии выполнения на ЦП или готовых к такому выполнению) процессов, управляемых с одного терминала. Один из этих параллельных процессов выполняется в оперативном, а остальные – в фоновом режиме. Существует команда shell, позволяющая влиять из данного процесса-shell на ход выполнения других процессов. Это команда уничтожения процесса **kill**. Ее формат:

kill [-i] <pid>

где i — флаг, задающий тип сигнала, передаваемого уничтожаемому процессу (а точнее — его окружению, то есть ОС); pid — номер уничтожаемого процесса.

Различные сигналы завершения процесса различаются степенью «жесткости». В частности, если флаг опущен, то в процесс передается сигнал «плавного» завершения процесса, позволяющий ему выполнить некоторые свои последние действия (например, уничтожить временные файлы). Некоторые из сигналов завершения могут быть вообще проигнорированы процессом. Примером «жесткого» сигнала, приводящего к безусловному уничтожению процесса является сигнал 9.

Например, следующая команда выполняет уничтожение фонового процесса, запущенного последним:

```
$ kill $!
```

где `$!` — имя переменной `shell`, содержащей `pid` последнего фонового процесса.

Следует отметить, что обыкновенный пользователь может уничтожать только свои собственные процессы. Привилегированный пользователь (администратор) может уничтожить любые процессы в системе.

Выполните уничтожение ранее запущенного процесса с бесконечным циклом.

7. Задержка процессов

Процесс-`shell` может задержать собственное выполнение с помощью одной из двух команд: `sleep` и `wait`. Команда sleep задерживает выполнение `shell` на заданное число секунд. Например, следующая команда выполняет задержку на один час:

```
$ sleep 3600
```

Нетрудно догадаться, что ввод данной команды из командной строки задержит прежде всего вас самих. Напротив, применение команды `sleep` в скрипте может быть весьма полезным. Например, следующий скрипт через час выведет сообщение (содержимое файла `file1`) на терминал пользователя с именем `sam` (подробнее о выводе сообщений в следующем разделе):

```
sleep 3600
write sam <file1
```

Команда **`wait`** задерживает выполнение своего `shell` до завершения процесса с заданным номером, а также возвращает код возврата этого процесса. Например, следующая команда задерживает процесс-`shell` до завершения процесса с номером `125`:

```
$ wait 125
```

Запустите процесс, выводящий на ваш экран сообщение через каждые 5 секунд.

8. Сообщения другому пользователю

Рано или поздно у вас могло появиться желание побеседовать с другими пользователями UNIX. Для этого, во-первых, желательно ознакомиться со списком пользователей, работающих в данный момент в системе. Это можно сделать с помощью команды **who**. Она выводит список пользователей с указанием для каждого из них имени, терминала и времени входа в систему.

Посылка сообщения на экран другого пользователя может быть выполнена командой **write** с указанием имени получателя сообщения. Например, пусть мы вошли в систему под именем vlad и хотим послать сообщение пользователю с именем sergei. Для этого набираем команду:

```
$ write sergei
```

Ввод этой команды приведет к последствиям как для нашего терминала, так и для терминала другого пользователя. Во-первых, наш shell будет ждать ввода сообщения до тех пор, пока мы не введем конец файла, то есть <ctrl>&<d>. Во-вторых, ввод нами этой команды приведет к появлению на другом экране сообщения: message from vlad tty1 (сообщение от vlad и терминала tty1)

Получив эту строку, пользователь sergei может поступить двояко. Во-первых, он может перейти к ожиданию собственно сообщения (оно поступит после нажатия нами <ctrl>&<d>). Во-вторых, sergei может набрать ответную команду: write vlad. Если отправка нашего сообщения (момент нажатия <ctrl>&<d>) совпадет по времени с работой другого пользователя на клавиатуре, то на его экране произойдет наложение двух текстов. Аналогично, наложение двух текстов может произойти и на нашем терминале, если во время набора нами сообщения начнет поступать ответное сообщение. Для предотвращения подобных неприятных ситуаций необходимо, чтобы диалог двух пользователей соответствовал некоторому протоколу.

Протокол — алгоритм диалога двух удаленных партнеров, предотвращающий порчу информации. Например, простейший протокол применения команд write заключается в следующем. Допустим, что мы получили на своем экране сообщение «message from ...». Тогда мы будем ждать появления в конце строки одиночной буквы «o», которая означает, что пришла наша очередь набирать сообщение. Набрав его, мы поместим в конец строки букву «o», а уж затем нажмем <ctrl>&<d>. Если мы хотим прекратить диалог, то набираем в конце строки не одну, а две буквы «o».

Вне зависимости от того, применяем ли мы этот или другой подобный протокол, важно помнить, что за его выполнение UNIX не несет никакой ответственности. Выполнение подобного «прикладного» протокола полностью основано на добровольном согласии партнеров диалога соблюдать требования протокола.

Возможно, что в данное время вы не хотите вести какие-то диалоги со своими «соседями» по UNIX. Тогда вам следует набрать команду **mesg** с

параметром **n**. В результате все сообщения, идущие на ваш терминал, будут отменены до тех пор, пока вы не наберете эту же команду **mesg**, но с параметром **y**.

9. Задание

Выполните наизусть следующую последовательность действий:

1. запуск из командной строки в фоновом режиме скрипта, выполняющего задержку на 10 секунд;
2. автоматическое ожидание завершения запущенного скрипта;
3. одновременный запуск в фоновом режиме двух одинаковых скриптов, выполняющих бесконечные циклы, с разными приоритетами;
4. по истечению нескольких минут получение на экране и объяснение информации о затратах времени ЦП двух запущенных ранее бесконечных процессов;
5. уничтожение всех созданных фоновых процессов;
6. обмен двумя-тремя сообщениями с другим пользователем, работающим в системе.

Примечание. При выполнении задания 3 можно обеспечить одновременность запуска скриптов, поместив их в общий конвейер (несмотря на отсутствие информационного обмена между скриптами).