Лабораторная №1

// для работы в компьютерных классах
При входе в систему необходимо выполнить следующие действия: **login**: **user**
Password: student

Выход из системы:
$ logout
login:shutdown

При вводе правильных ключевых слов перед Вами появится знак # или $, который обозначает командную строку.

Командный процессор

Командная строка – строка текста на языке shell.

Кома́ндная оболо́чка Unix (англ. *Unix shell*, часто просто «шелл» или «sh») — командный интерпретатор, используемый в операционных системах семейства Unix, в котором пользователь может либо давать команды операционной системе по отдельности, либо запускать скрипты, состоящие из списка команд. В первую очередь, под shell понимаются POSIX-совместимые оболочки, восходящие к Bourne shell (шелл Борна), появившемуся в Unix Version 7.

Наиболее общий формат команд (в квадратные скобки помещены необязательные части):

[символ_начала_команды] имя_команды [параметр_1 [параметр_2 […]]]

Для выполнения (почти) каждой простой команды*) shell порождает отдельный **процесс**, в рамках которого выполняется программа, хранящаяся в файле и имеющая имя команды. Программа может быть выполняемой, т.е. содержать машинные команды, или представлять собой shell-процедуру – содержать текст на языке shell.

Shell считывает командную строку из файла стандартного ввода и интерпретирует ее в соответствии с установленным набором правил. Дескрипторы файлов стандартного ввода и стандартного вывода, как правило, указывают на терминал, с которого пользователь зарегистрировался в системе. Если shell «узнает» во введенной строке конструкцию собственного командного языка, то он исполняет команду «своими силами», не прибегая к созданию новых процессов; в противном случае команда интерпретируется как имя исполняемого файла.

Примеры:

Простейшие командные строки содержат имя и несколько параметров. Например:

#who
#grep –n include *.c
#ls –l

Shell «ветвится», используя системный вызов fork() и порождает новый процесс, который и запускает программу, указанную пользователем в командной строке. Родительский процесс (shell) дожидается завершения процесса-потомка и повторяет цикл считывания следующей команды.

Изучение команд Shell

Задание:

---

*) Простая команда – последовательность полей с разделителями (обычно пробелами) между ними.

- Используя команду mkdir создайте директорию именем своей группы. Например, $ mkdir iu7; перейдите в созданную директорию с помощью команды cd

Например,
$ cd iu7
Команда cd работает аналогично одноименной команде в консоли Windows или Дос. Имеется одно отличие: в Unix команда чувствительна к наличию разделяющего пробела.

$ cd iu7-53
cd ..

Создайте поддиректорию, например, используя свою фамилию.
$ mkdir <student name>
и т.д.
$ cd <student name>
$ pwd
< working >

- В Unix имеется команда **man** от слова manual, выводящая на экран описание команд, например: $man cd или $man ps
  Выход из подсказки осуществляется нажатием q. Используя man изучите команды, список которых приведен ниже.
- **Команда: ls**
  Например: $ ls / -выводит список файлов из root
  $ ls -l
  $ ls -lt
  $ ls -F

$ ls –a - вывод списока всех файлов в несколько столбцов
$ ls –d - трактовать каталоги наравне с файлами других типов
$ ls –i - выдавать порядковый номер файла в файловой системе
например: вывод содержимого директории в расширенном формате
$ ls –al <Enter>
total 30
drwx-xr-x 3 startship project 96 Oct 27 08:16 bin
drwx-xr-x 2 startship project 64 Nov 1 14:19 draft
-rwx----- 2 startship project 80 Nov 8 08:41 letters
Первый символ строки обозначает тип файла.
В Linux различаются 7 типов файлов:
d – справочник или директорий (directory)
- - обычный файл (regular)
l – символическая связь (канал) – символьная ссылка
b – специальный блочный файл - блочное устройство
c – специальный символьный файл - символьное устройство
p – программный канал (pipe)
s – сокет в файловом пространстве имен (Домен – UNIX)

Следующие значения маски определены для типа файла:
S_IFMT 0170000 bit mask for the file type bit field
S_IFSOCK 0140000 socket
S_IFLNK 0120000 symbolic link
S_IFREG 0100000 regular file
S_IFBLK 0060000 block device

S_IFDIR 0040000 directory
S_IFCHR 0020000 character device
S_IFIFO 0010000 FIFO

Таким образом, чтобы проверить обычный файл (например), можно написать:

```
stat(pathname, &sb);
if ((sb.st_mode & S_IFMT) == S_IFREG) {
    /* Handle regular file */
}
```

Следующие три символа – права доступа: read-write-execute для пользователя (user), следующие три для группы пользователя (group) и последние три для остального «мира» (others). Столбик чисел – количество жестких ссылок на файл.
Команда ls -ial дополнительно выведет номер inode (см. информацию ниже по тексту).

- **Команда ps**

Например: вывести все процессы, связанные с терминалом

#ps -a
или #ps -xl
или #ps -ajx – выводит список системных демонов

Если в командной строке набрать ps -al на экране появится примерно следующее:

```
F PID   PPID  TTY   TIME     CMD
0  1007 1101  tty1  00:00:00 bach
0  1036 1101  tty2  00:00:00 bach
0  1424 1325  tty1  00:00:02 mc
….
1  6785 6784   tty   00 :00:09 mypr
```

Где F – флаги, показывают важную системную информацию. Флаги являются важнейшим параметром процесса.

Используя команду man в описании команды ps, найдите флаги (flags) и статус (state).

В Linux Ubuntu флаги имеет три значения:

- 1 - показывет, что был fork(), но системного вызова exec() не было. Системный вызов fork() создает новый процесс-потомок. Системный вызов exec() переводит процесс на новое адресное пространство, что делается для запуска на выполнение новой программы, которая передается exec() в качестве параметра.
- 4 – показывает, что процесс выполняется с правами супер пользователя
- 0 – показывает, что был fork() и exec(), т.е. создан новый процесс и он переведен на выполнение новой программы. Это нормальное выполнение действий в системе.

команда $ps –ax добавит статус

статус:
R – running or runnable
S – interruptble sleep
D - uninterruptible sleep (usually IO)
….
T - terminate
Z - zombie

$ ps –al

Следующее задание: напишите программу, в которой создается дочерний процесс и организуйте как в предке, так и в потомке бесконечные циклы, в которых выводятся идентификаторы процессов с помощью системного вызова getpid():

```
#include <stdio.h>
int main()
{
 int childpid;
 if ((childpid = fork())== -1)
  {
    perror("Can't fork.\n");
    return 1;
  }
 else if (childpid == 0)
  {
    while(1) printf("%d  ", getpid());
    return 0;
  }
 else
  {
    while(1)  printf(" %d  ",getpid());
    return 0;
  }
 }
```

1- запустите программу и посмотрите идентификаторы созданных процессов: предка и потомка;

2- для получения процесса зомби выполните следующие действия: a) удалите командой kill потомка и посмотрите с помощью команды `ps` его новый статус – Z; b) удалите предка.

3- Для получения «осиротевшего» процесса запустите программу еще раз, но в этот раз удалите предка и посмотрите с помощью команды `ps` идентификатор предка у продолжающего выполняться потомка; идентификатор предка будет изменен на 1, так как процесс был «усыновлен» процессом с идентификатором 1 процессом «открывшим» терминал в случае, если используется Unix BSD, или идентификатор процессов-посредников в случае, Linux Ubuntu.

- **Выполните следующие команды:**

  id - информация о текущем пользователе, если набрать id [имя пользователя], то будет получена информация об ассоциированных с данным пользователем данных.

  logname – можно узнать полное имя текущего пользовантеля.

  pwd – позволяет узнать абсолютное имя маршрутное имя текущего каталога.

  who – можно узнать активных пользователей и их терминалы.

  Например: name ttyS4 Sept 22 13:25

  mail

  date

  cp

  cat

  sort

  rm

  rmdir

newgrp – смена группы (команда относится к необязательным и входит в расширение «Мобильность пользователей»
tee
jobs
ln
id
find ... Например: #find /-name "signal.h" -print
quota
echo

Особое внимание обратите на команды:

- **mknod** – создает буфер

формат команды: mknod [опции] имя {bc} старший номер младший номер
mknod [опции] имя p
если указывается опция p, то создается именованный канал типа fifo
опция b – для блочного файла
опция c – для символьного (не буферизованного) файла

- **chmod** – изменяет права доступа к файлу. Для каждого файла устанавливаются права доступа: первые 3 символа – права доступа для создателя файла; следующие 3 символа – для группы и последние 3 – для остального мира (others).
  r – разрешает читать или копировать содержимое файла,
  w – разрешает вносить изменения в файл или копировать его содержимое,
  x - разрешает исполнять файл.
  Синтаксис команды:
  chmod кто+права файл(ы) <Enter>
  или
  chmod кто-права файл(ы) <Enter>
  где
  кто – одна из трех групп пользователей: u – владелец, g – группа, o – остальные пользователи;
  + или - наличие или отсутствие права;
  права – r – read, w – write, x – execute;
  файлы – имя или имена файлов (полные имена с путем или в текущей директории).
  Например: $ chmod u-w list

- **kill** <proc numb> Например: kill –9 1234 уничтожает процесс. Опция -9 означает безусловное уничтожение процесса.

Файловая система Unix

Файловая система в Unix – иерархическая ("деревянная"), состоит из файлов и каталогов. На каждом разделе диска (partition) создается собственная независимая файловая система. Отдельные файловые системы входят в единое дерево каталогов и файлов системы вместе. Такая операция называется "монтированием". Выглядит это примерно так:
mount - F ufs /dev/dsk/m197_c0d0s5 /home1
mount - F ufs /dev/dsk/m197_c0d0s4 /usr

Получить доступ к файлам "несмонтированной" файловой системы невозможно. Только смонтированная файловая система, ее каталоги и файлы видны в дереве каталогов. Порочная пракика MS-DOSa - сколько разделов, столько и "дисков" (a: b: c: d: e:... k: l: m: n:) в Unix не применяется. В Unix всегда есть ровно одно общее дерево каталогов, и, по большому счету, пользователям совершенно все равно, на каком именно диске или разделе диска расположены его файлы /usr/spool/moshkow или /home1/moshkow/bin/mcopy...
Файловая система Unix кэшируется буферным кэшем. Операция записи на диск выполняется не тогда, когда это приказывает выполняемый процесс, а когда операционная система сочтет нужным это сделать. Это резко поднимает эффективность и скорость работы с диском, и повышает опасность ее использования. Выключение питания на "горячей", работающей Unix-машине приводит к разрушениям структуры файловой системы.
При каждой начальной загрузке Unix проверяет - корректно ли была выключена машина в прошлый раз, и если нет - автоматически запускает утилиту fsck (File System Check) - проверку и ремонт файловых систем. .

Внутренняя структура файловой системы Unix

Для поддержки большого количества файловых систем в Unix реализован интерфейс VFS/VNODE. В Linux это интерфейс – VFS. Linux **vnode** не поддерживает, а только inode (index node). В Linux виртуальная файловая система построена на четырех основных структурах: struct superblock, struct inode, struct dentry, struct file.

Каждый каталог и файл файловой системы имеет уникальное полное имя (в ОС UNIX это имя принято называть full pathname - имя, задающее полный путь, поскольку оно действительно задает полный путь от корня файловой (“/”) системы через цепочку каталогов к соответствующему каталогу или файлу; мы будем использовать термин "полное имя", поскольку для pathname отсутствует благозвучный русский аналог). В конце полного имени находится собственно имя файла (short name). Каталог, являющийся корнем файловой системы (корневой каталог), в любой файловой системе имеет предопределенное имя "/" (слэш). Полное имя файла, например, /bin/sh означает, что в корневом каталоге должно содержаться имя каталога bin, а в каталоге bin должно содержаться имя файла sh. Коротким или относительным именем файла (relative pathname) называется имя (возможно, составное), задающее путь к файлу от текущего рабочего каталога (существует команда и соответствующий системный вызов, позволяющие установить текущий рабочий каталог).

В каждом каталоге содержатся два специальных имени, имя ". " – определяющее текущий каталог, и имя ".. ", именующее "родительский" каталог данного каталога, т.е. каталог, непосредственно предшествующий данному в иерархии каталогов.

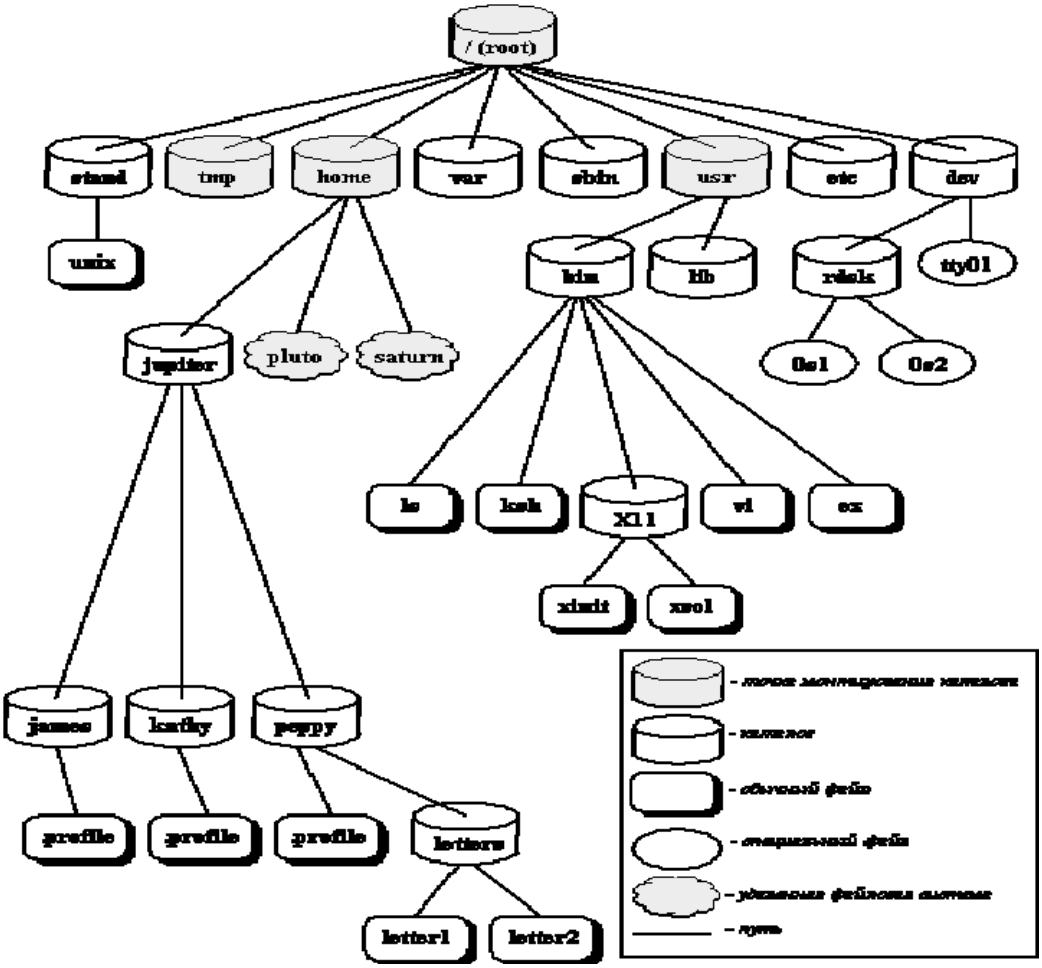

Рис.1 Структура каталогов файловой системы UNIX поддерживает многочисленные утилиты, позволяющие работать с файловой системой и доступные как команды командного интерпретатора.

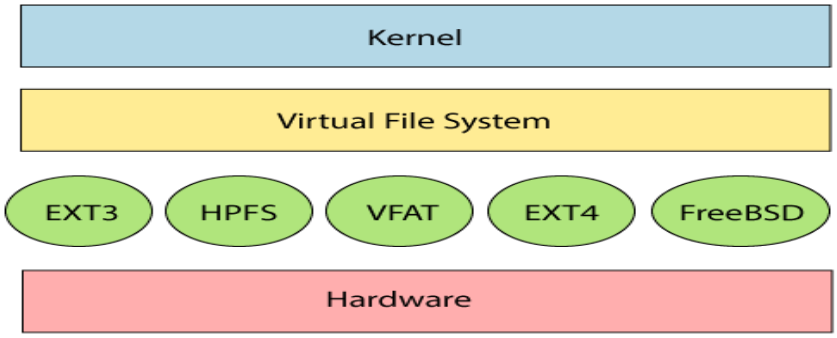

Рис.2 Структура VFS (Virtual File System – Виртуальная файловая системы).

Для отображения содержимого файла /etc в формате дерева каталогов: tree /etc

Вывод:

```
etc
|-- abrt
|   |-- abrt-action-save-package-data.conf
```

```
|   |-- abrt.conf
|   |-- gpg_keys
|   `-- plugins
|       |-- CCpp.conf
|       `-- python.conf
|-- acpi
|   |-- actions
|   |   `-- power.sh
|   `-- events
|       |-- power.conf
|       `-- video.conf
|-- adjtime
|-- aliases
```

…

Имя файла в Unix/Linux не является его идентификатором, так как файл может иметь много имен, которые называются жесткими ссылками (hard links).

Файл в Unix/Linux идентифицируется номером inode (index node – индексный узел).

inode

inode - **индексный дескриптор** — это структура данных в традиционных для ОС UNIX файловых системах (ФС), таких как UFS, ext2, ext4. В этой структуре хранится метаинформация о стандартных файлах, каталогах или других объектах файловой системы, кроме непосредственно данных и имени.

При создании файловой системы создаются также и структуры данных, содержащие информацию о файлах. Каждый файл имеет свой индексный дескриптор, идентифицируемый по уникальному номеру (часто называемому 'i-номером' или 'инодом'), в файловой системе, в которой располагается сам файл.

Индексные дескрипторы хранят информацию о файлах, такую как принадлежность владельцу (пользователю и группе), режим доступа (чтение, запись, запуск на выполнение) и тип файла. Существует определённое число индексных дескрипторов, которое указывает максимальное количество файлов, допускаемое определённой файловой системой. Обычно при создании файловой системы примерно 1 % её объёма выделяется под индексные дескрипторы.

- Номер индексного дескриптора заносится в таблицу индексных дескрипторов в определённом месте устройства; по номеру индексного дескриптора ядро системы может считать содержимое инода, включая указатели данных и прочий контекст файла.
- Номер индексного дескриптора файла можно посмотреть, используя команду `ls -i`, а команда `ls -l` покажет информацию, хранящуюся в индексном дескрипторе.

Имена файлов и содержимое каталогов:

- индексные дескрипторы не хранят имена файлов, только информацию об их содержимом;

- каталоги в Unix являются списками 'ссылочных' структур, каждая из которых содержит одно имя файла и один номер индексного дескриптора;
- ядро должно просматривать каталог в поисках имени файла, затем конвертировать это имя в соответствующий номер индексного дескриптора, в случае успеха;
- содержимое файлов располагается в *блоках данных*, на которые ссылаются индексные дескрипторы.

В ядре ОС Linux определена структура `struct inod`. В BSD системах используется термин `vnode`, буква **v** в котором указывает на virtual inode уровня ядра.

Стандарты POSIX описывают поведение файловой системы как потомка традиционных файловых систем UNIX — UFS. Регулярные файлы должны иметь следующие атрибуты:

- длина файла в байтах;
- идентификатор (ID) устройства (это идентифицирует устройство, содержащее файл);
- ID пользователя, являющегося владельцем файла;
- ID группы файла;
- *режим* файла, определяющий какие пользователи могут считывать, записывать и запускать файл;
- Timestamp указывает дату последнего изменения инода (`ctime`, *change time*), последней модификации содержимого файла (`mtime`, *modification time*), и последнего доступа (`atime`, *access time*);
- счётчик ссылок указывает количество жестких ссылок, указывающих на индексный дескриптор;
- указатели на блоки диска, хранящие содержимое файла.

Системный вызов `stat` считывает номер индексного дескриптора файла и некоторую информацию из него.

Пример struct inode. Эта структура может отличаться от структур в старших версиях Linux. Но она демонстрирует основные поля структуры.

```
struct inode {
    kdev_t                  i_dev;
    unsigned long               i_ino;
    umode_t                 i_mode;
    nlink_t                 i_nlink;
    uid_t                   i_uid;
    gid_t                   i_gid;
    kdev_t                  i_rdev;
    off_t                   i_size;
    time_t                  i_atime;
    time_t                  i_mtime;
    time_t                  i_ctime;
    unsigned long               i_blksize;
    unsigned long               i_blocks;
    unsigned long               i_version;
    unsigned long               i_nrpages;
    struct semaphore            i_sem;
    struct inode_operations     *i_op;
    struct super_block          *i_sb;
    struct wait_queue           *i_wait;
    struct file_lock            *i_flock;
    struct vm_area_struct       *i_mmap;
```

```
    struct page             *i_pages;
    struct dquot            *i_dquot[MAXQUOTAS];
    struct inode            *i_next, *i_prev;
    struct inode            *i_hash_next, *i_hash_prev;
    struct inode            *i_bound_to, *i_bound_by;
    struct inode            *i_mount;
    unsigned short          i_count;
    unsigned short          i_flags;
    unsigned char           i_lock;
    unsigned char           i_dirt;
    unsigned char           i_pipe;
    unsigned char           i_sock;
    unsigned char           i_seek;
    unsigned char           i_update;
    unsigned short          i_writecount;
    union {
      struct pipe_inode_info   pipe_i;
      struct minix_inode_info  minix_i;
      struct ext_inode_info    ext_i;
      struct ext2_inode_info   ext2_i;
      struct hpfs_inode_info   hpfs_i;
      struct msdos_inode_info  msdos_i;
      struct umsdos_inode_info umsdos_i;
      struct iso_inode_info    isofs_i;
      struct nfs_inode_info    nfs_i;
      struct xiafs_inode_info  xiafs_i;
      struct sysv_inode_info   sysv_i;
      struct affs_inode_info   affs_i;
      struct ufs_inode_info    ufs_i;
      struct socket            socket_i;
      void                     *generic_ip;
    } u;
};
```

После выполнения команды ln в каталог заносится новая запись, указывающая на inode существующего файла. Следовательно links имеют один и тот же inode (Рис 1.).

Жесткие и гибкие ссылки на файлы (Link)

Жёсткой ссылкой (англ. *hard link*) в UFS-совместимых файловых системах называется структурная составляющая файла — описывающий его элемент каталога.
*В Unix к одним и тем же файлам можно обращаться под разными именами***.**
Другими словами, имена файлов также называются ссылками на этот файл, причем это - жесткие ссылки.
Мягкая ссылка, или символическая ссылка, если коротко, символическая ссылка - совсем другое объект. Это файл, который содержит полное имя другого файла и позволяет осуществлять автоматический поиск системного имени.
Все жесткие ссылки на файл находятся в одной файловой системе. Симлинки на файл могут жить в разных файловых системах.
Файловая система отслеживает количество жестких ссылок на файл, но не количество символических ссылок. Если вы удалите файл, на который указывает символическая ссылка, символическая ссылка становится висячей символической ссылкой.
Поскольку файловая иерархия Unix должна быть деревом, должна быть ровно одна жесткая ссылка на каталог. Однако символические ссылки могут указывать на произвольные имена

путей. При обходе дерева файлов нужно соблюдать остороженость, чтобы не следовать символическим ссылкам (или иным образом отслеживать все посещенные файлы).

В Unix-подобной операционной системе любой файл или каталог может иметь несколько имен и поэтому *в Unix имя файла не является его идентификатором в системе, а предоставляет пользователю удобный способ обращения к файлам по их именам - символьный уровень файловой системы.* Для идентификации файлов используются inode. Жесткие ссылки создаются с помощью команды ln. Все жесткие ссылки имеют один и тот же inode. Поэтому struct inode содержит количество жестких ссылок.

Команда **ln** связывает новое имя с уже имеющимся файлом, что предоставляет возможность обращаться к нему под различными именами. Новое имя называют link к старому имени. Т.е. создается еще одно равноправное имя файла.

```
#include <unistd.h>
int link(const char *oldpath, const char *newpath);
```

- **Задание:** введите команду ls и сделайте link на файл, например, fil.3 и затем ls

  $ ls -il
  …
  82343 – rw – r – r- -r- - 1 IU715937 … fil.3
  …
  $ ln fil.3 datlink
  $ ls il

  82343…….2IU715937 …….datlink
  82343…….2IU715937……...fil.3

  После этого уничтожьте файл datlink и вызовите ls

  $ rm datlink
  $ ls il

  82343 – rw-r- - r- -1IU715937…fil.3

Чтобы создать новую символическую ссылку (не получится, если она уже существует):

ln -s /path/to/file /path/to/symlink

Чтобы создать или обновить символическую ссылку:

ln -sf /path/to/file /path/to/symlink

Переключение ввода/вывода : осуществляется с помощью символа > или <

#ls > filelist
#cat f1 f2 f3 >> temp - склеивает три файла и записывает в конец temp
who > temp
sort < temp
mail Mary,Joe,Tom < letter

Программные каналы (pipe)

Linux так же, как Unix поддерживает программные каналы двух типов: именованные и неименованные. ***Именованные*** **программные каналы** создаются командой mknod. Программный канал - это специальный файл, в который можно «писать» информацию и из которого эту информацию можно «читать». Причем порядок записи информации и последующего чтения – FIFO (очередь).

Именованный канал имеет имя, которое указывается при вызове команды mknode:

#mknod [опции] <имя> p

Использовать именованный канал может любой процесс, «знающий» имя канала.

Для демонстрации работы именованного программного канала :

- создаем именованный программный канал командой mknod с именем pipe;
- направляем текст в программный канал:
  echo [текст] > pipe
- меняем консоль;
- получаем через канал текст и, используя команду tee**, выводим** на экран:
  tee < pipe

Неименованные программные каналы в командной строке создаются с помощью символа '| '.

Неименованные программные каналы могут использоваться для передачи сообщений только между процессами-родственниками, т.е. между процессами, которые имеют общего предка.

Конвейеры создаются с помощью неименованных программных каналов.

Конвейееры и примеры их использования

С помощью конвейеров удается комбинировать результаты выполнения разных команд, получая в результате новое качество.

Например, команда ls выводит информацию о файлах, а команда wc подсчитывает число строк в файле. Объединяя в конвейер эти команды, получаем возможность подсчитать количество файлов в каталоге:

ls –al / bin | wc –l

Если нужна информация о файлах текущего каталога, модифицированных в октябре, то поможет следующий конвейер:

ls –al | grep “Oct”

Команда grep играет роль фильтра, который пропускает только часть файлов.

Трех ступенчатый конвейер считает файлы модифицированные в октябре:

ls –al | grep “Oct” | wc –l

Четырех ступенчатый конвейер не только считает число файлов, но и помещает эту информацию в файл tmp / tmpinf :

ls –al | grep “Oct” | tee / tmp /tmpinf | wc –l

Рассмотрите следующие примеры:

#ls -l | tee filetee

#cat filename > filename1 | pwd > filename|ls|sort

#find /usr -name "*.let" –print | more

Приоритеты процессов

В фазе “пользователь” приоритет процесса имеет 2 составляющие: пользовательскую и системную. Значения этих составляющих задают поля дескриптора процесса p_nice и p_cpu.

Начальное значение пользовательской составляющей равно константе NZERO (=20). Пользовательская составляющая может быть изменена системным вызовом nice с аргументом, определяющим величину изменения поля p_nice в пределах от 0 до NZERO для непривилегированного процесса и от –NZERO до +NZERO для привилегированного.

Начальное значение системной составляющей в фазе пользователь равно 0. Ее изменение зависит от времени использования процессора. Для формирования системной составляющей приоритета используются прерывания от аппаратного таймера, которые генерируются 50 раз в секунду. Каждое прерывание по таймеру увеличивает значение поля p_cpu на 1.

Результирующий приоритет процесса в фазе «пользаватель» определяется по формуле:
p_pri = (p_nice – NZERO) + (p_cpu/16) + P_USER, где P_USER - константа, по умолчанию равная 50.

Фоновый режим :

Например:
```
#cat *.let > myprog&
#jobs
  ...
  # sort *.let > doc & lpr *.let &
  #at <time>
 #nice <name>&

 #nice -15 proc&
```

В приведенных примерах знак **&** снижает приоритет процесса.

Снижение приоритета приводит к созданию фоновых процессов.

Для организации более "глубокого" **фонового процесса**, чем это позволяет сделать оператор **&** используется команда **nohup** (no hand up - не отключаться).

Команда nohup принимает в качестве аргументов командную строку. Однако, чтобы процесс действительно выполняется в фоновом режиме, nohup следует использовать с &.

Если с помощью nohup запустить процесс, он не будет прекращен системой несмотря на отключение терминала или модема.

Запущенная с помощью nohup команда сразу не начнет выполняться. Если процесс необходимо начать позднее или запускать его периодически, то следует прибегнуть к услугам демона cron.

Демон cron - это процесс в фоновом режиме, запущенный программой init Linux. Cron предоставляет услуги планировщика всем процессам Linux.

Можно заставить cron запускать программу в определенное время.

Команда crontab - для каждого пользователя создается его собственный файл со списком заданий в каталоге

/usr/spool/cron/crontabs.

Например, если вы пользователь и именем kiv, то

/usr/spool/cron/crontabs/kiv

Традиционные редакторы Linux

В Linux-системе может быть любое число редакторов, но два стандартных ed и vi есть всегда.

Редактор vi (visual) остается в Linux одним из наиболее широко исползуемых.
Строковый редактор ed используется редко.

Редактор ed.

Команды ed представляют собой простые символы.
Для создания файла

```
$ ed poem
poem: no such file or directory
a  начать добавление строк
  ...
.  признак конца ввода
w poem  пишем строки в poem
q  выход
```

Для редактирования старого файла

```
$ ed poem
121
a
....
.
q
?
w
139
q
$
```

Временная передача управления shell с помощью !

```
$ ed poem
139
! wc poem
  6       7139poem
!
q
```

Редактор vi.

Для начала работы наберите

```
$ vi <filename>
```

Выполнение этой команды переводит Вас в командный режим редактора. Для перехода в режим редактирования нажмите символ - a; в режиме ввода каждая клавиша получает свое буквенно-цифровое значение.

Для возврата в командный режим нажмите клавишу Esc.

После этого для входа в режим построчного редактирования надо нажать клавишу : .

Для выхода из редактора нажмите две буквы - zz.

Команда :w работает как Save As.

Например,

```
:w booklist  <enter>
```

сохранит файл с именем booklist.

Выйти из редактора можно с помощью команды :q . В отличие от zz команда :q файл не сохраняет.

Если вы вошли в vi, не указав имени файла, то не сможете выйти из него посредством команды zz. Вам следует сохранить файл - команда :w <file>, а затем выйти - :q ; команду zz можно заменить - :wq.

Команда :q! обеспечивает выход без сохранения изменений.

Текстовый редактор joe <name>.c (если установлен)

Компиляция программ:

#gcc -o programma.o programma.c –ls

Задание: написать программу вывода на экран сообщения «Hellow, world!».

```
#include <stdio.h>
int main (void)
{
  printf("Hello, World!\n");
  return 0;
}
```

Запуск на выполнение: **./a.out**

Список литературы

1. Кристиан К. Операционная система UNIX. - М., Финансы и статистика, 1985, 320 стр.
2. Баурн С. Операционная система UNIX. - М., Мир, 1986, 464 стр.