Лабораторная №1

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 начинает командную строку.

Командный процессор

Командная строка – строка текста на языке shell.

Для выполнения (почти) каждой простой команды*) shell порождает отдельный процесс, в рамках которого выполняется программа, хранящаяся в файле, имеющим имя команды. Программа может быть выполнимой, 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.

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

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

---

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

Выход из подсказки осуществляется нажатием q. Используя man изучите команды, список которых приведен ниже.

Задание: исследовать следующие команды, используя команду man.

Например: команда cd

$man cd

Команда cd работает аналогичено одноименной команде в консоли Windows. Имеется одно отличие - в Unix команда чувствительна к наличию разделяющего пробела.

$ cd iu7-53

cd ..

Команда : ls

Например: $ ls / -выводит список файлов из root

$ ls -l

$ ls -lt

$ ls -F

$ ls –a - вывод список 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 директория 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:

d – справочник или директорий

- - обычный файл

l – символическая связь (канал) – символьная ссылка

b – блочный файл

c – символьный файл

p – программный канал (pipe)

Следующие три символа – права доступа: read-write-execute для user, следующие три – group и ещё три – others.

Команда ps

Например: вывести все процессы, связанные с терминалом

#ps -a

или #ps -xl

или #ps -ajx – выводит список системных демонов

На экране появится примерно следующее:

PID TTY TIME CMD

1007 tty1 00:00:00 bach

1036 tty2 00:00:00 bach

1424 tty1 00:00:02 mc

команда $ps –ax добавит статус

статус: R-поцесс выполняется

T-остановлен

D-ожидает ввода/вывода

Z-зомби

S-спит <20s
I-спит >20s
$ ps –al

Важнейшим параметром процесса являются флаги. Используя мануал найти значения флагов и связать полученную информацию с материалом лекций.

Задание**:**

напишите программу, в которой создается дочерний процесс и организуйте как в предке, так и в потомке бесконечные циклы – например, в одном выводите «1» ,во втором – «2» или идентификаторы процессов с помощью системного вызова getpid() и getppid();

```
#include <stdio.h>
int main()
{
 int childpid, pearentpid;
 if ((childpid = fork())== -1)
  {
    perror("Can't fork.\n");
    return 1;
  }
 else if (childpid == 0)
  {
    while (1) printf(" childpid = %d ", getpid());
    return 0;
  }
 else
  {
    while(1)  printf(" pearentpid = %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 предкка у продолжающего выполняться потомка – идентификатор предка будет изменен на 1, так как процесс был «усыновлен» процессом с идентификатором 1 , процессом «открывшим» терминал.

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»

mknod – создает именованный канал типа fifo
формат команды: mknod [опции] имя {bc} старший номер младший номер
mknod [опции] имя p
p – для fifo
b – для блочного файла
c – для символьного (небуферизованного) 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
$ ls -l

tee
jobs
ln
id
find ... Например: #find /-name "signal.h" -print
quota
echo
kill <proc numb> Например: kill –9 1234

Link. *В Unix к одним и тем же файлам можно обращаться под разными именами***.**
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.

Links

Names of a file are also called *links* to that file. Or *hard links* if it is

necessary to distinguish them from *soft links*. All hard links are equivalent.

A soft link, or symbolic link, symlink in short, is a very different animal. It is a very small file that contains the name of another file, and has the property that the system name lookup process will usually automatically follow this redirection. All hard links to a file live in the same filesystem. Symlinks to the file can live in different filesystems. The filesystem keeps track of the number of hard links to a file, but not of the number of symlinks. If you remove the file a symlink pointed to, the symlink becomes a *dangling* symlink.

Since the Unix file hierarchy should be a tree, there should be precisely one hard link to a directory. Symlinks however can point at arbitrary path names. A file tree walker should be careful not to follow symlinks (or otherwise keep track of all files visited).

On a Unix-like operating system any file or directory can have multiple names because of the operating system's use of inodes instead of names to identify files and directories. Additional names can be provided by using the *ln* command to create one or more *hard links* to a file or directory.

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

Description

link() creates a new link (also known as a hard link) to an existing file.
If *newpath* exists it will *not* be overwritten.
This new name may be used exactly as the old one for any operation; both names refer to the same file (and so have the same permissions and ownership) and it is impossible to tell which name was the "original".

So the name of the file is stored within the directories' information structure. For example:

При создании файла имя файла *(в Unix имя файла не является идентификатором файла в системе, а предоставляет пользователю удобный способ обращения к файлам и их именования - символьный уровень файловой системы)* заносится в каталог и для адресации файла на диске создается указатель inode (I – Node номер), который содержит следующую информацию о файле:

- тип файла;
- бит защиты;
- links;
- данные о пользователе и группе;
- размер файла;
- временные данные;
- указатели на прямые блоки;
- указатели на косвенные блоки;
- указатели на вторичные косвенные блоки.

Files can have multiple names. If multiple names hard link to the same inode then the names are equivalent; i.e., the first to be created has no special status. This is unlike symbolic links, which depend on the original name, not the inode (number).

An inode may have no links. An unlinked file is removed from disk, and its resources are freed for reallocation but deletion must wait until all processes that have opened it finish accessing it. This includes executable files which are implicitly held open by the processes executing them.
It is typically not possible to map from an open file to the filename that was used to open it. The operating system immediately converts the filename to an inode number then discards the filename. This means that the getcwd() and getwd() library functions search the parent directory to find a file with an inode matching the working directory, then search that directory's parent, and so on until reaching the root directory. SVR4 and Linux systems maintain extra information to make this possible.

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.

inode

The VFS `inode` data structure holds information about a file or directory on disk.

```
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.).

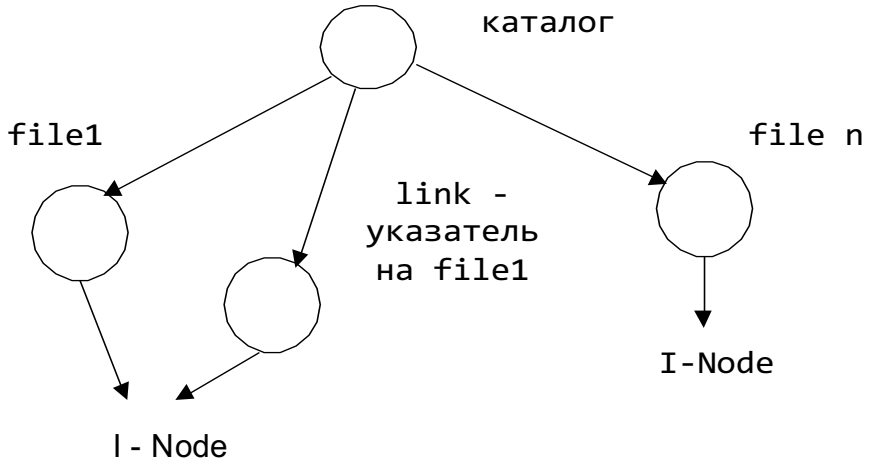

Рис.1

В этом случае создается, так называемый, hard link – link в том же носителе данных. Права доступа всех links одни и те же.

В некоторой иерархии файлов могут существовать несколько файловых систем с собственными списками inode файлов. В каждой файловой системе каждый номер , например, 1131, указывает на свой отличный от других файл.

С помощью link-механизма невозможно инсталлировать link за рамки файловой системы. Поэтому был разработан symbolic link или soft link, позволяющий создавать указатели на файлы других файловых систем.

Посмотреть с помощью команды ln информацию, которая выводится для каждого типа линка, проанализировать ее и сделать выводы.

Super User может делать кроме того указатели на каталоги.

Например: пользователи usr1 и usrIU7 хотят работать с одним и тем же файлом:

$ ln / usr / IU7 / usr1 / data1 /usr / IU7 / usrIU7 data1

Задание: введите команду ls и сделайте link на файл 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 ls

$ rm datlink
$ ls il

82343 – rw-r- - r- -1IU715937…fil.3

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

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**

Команда tree

Installation

By default the tree command is not installed. Type the following command to install the same under RHEL / CentOS / Fedora Linux:

```
# yum install tree
```

If you are using Debian / Mint / Ubuntu Linux, type the following command to

install the tree command:

```
$ sudo apt-get install tree
```

Syntax

The syntax is:

```
tree
tree /path/to/directory
tree [options]
tree [options] /path/to/directory
```

To list contents of /etc in a tree-like format:

```
tree /etc
```

Sample outputs:

```
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
|-- aliases.db
|-- alsa
|   `-- alsactl.conf
|-- alternatives
|   |-- links -> /usr/bin/elinks
|   |-- links-man -> /usr/share/man/man1/elinks.1.gz
|   |-- mta -> /usr/sbin/sendmail.postfix
|   |-- mta-aliasesman -> /usr/share/man/man5/aliases.postfix.5.gz
|   |-- mta-mailq -> /usr/bin/mailq.postfix
|   |-- mta-mailqman -> /usr/share/man/man1/mailq.postfix.1.gz
|   |-- mta-newaliases -> /usr/bin/newaliases.postfix
|   |-- mta-newaliasesman -> /usr/share/man/man1/newaliases.postfix.1.gz
|   |-- mta-pam -> /etc/pam.d/smtp.postfix
|   |-- mta-rmail -> /usr/bin/rmail.postfix
|   |-- mta-sendmail -> /usr/lib/sendmail.postfix
|   `-- mta-sendmailman -> /usr/share/man/man1/sendmail.postfix.1.gz
|-- anacrontab
|-- asound.conf
|-- at.deny
|-- audisp
|   |-- audispd.conf
|   `-- plugins.d
|       |-- af_unix.conf
|       |-- sedispatch.conf
|       `-- syslog.conf
|-- audit
|   |-- auditd.conf
....
..
..
|-- xinetd.d
|   `-- rsync
|-- xml
|   `-- catalog
|-- yum
|   |-- pluginconf.d
|   |   |-- product-id.conf
```

```
|   |   |-- protectbase.conf
|   |   |-- rhnplugin.conf
|   |   `-- subscription-manager.conf
|   |-- protected.d
|   |-- vars
|   `-- version-groups.conf
|-- yum.conf
`-- yum.repos.d
    |-- epel.repo
    |-- epel-testing.repo
    |-- ksplice-uptrack.repo
    |-- redhat.repo
    `-- rhel-source.repo
208 directories, 1452 files
```

The `-a` option should be passed to see all files. By default tree does not print hidden files (those beginning with a dot '.'). In no event does tree print the file system constructs '.' (current directory) and '..' (previous directory).:

```
tree -a
```

To list directories only, run:

```
tree -d
```