Controlul intreruperilor (semnalelor) in Shell

In mod normal o procedura Shell se termina daca primeste o intrerupere de la terminal (de regula prin **CTRL-C**).

Comanda trap

Scop: Controleaza modul de comportare la terminare (si mai general la aparitia unui semnal)

Sintaxa:

trap [-l] [arg] [sigspec]

Optiuni:

-l - da o lista a **numelor semnalelor** si a numerelor de identificare corespunzatoare;

arg - daca lipseste sau e precedat prin **-**, toate semnalele specificate revin la tratarea lor impicita;

- daca arg e sirul vid, semnalele specificate vor ignorate;
- *apare ca o lista de c-zi intre apostrofuri*;

[sigspec] daca are val 0 **arg** specifica ce c-zi se executa inainte de terminarea procedurii prin **exit**

Controlul intreruperilor (semnalelor) in Shell

Exemplu de uitilizare:

In cazul in care se doreste **stergerea unor eventuale fisere temporale** create de procedura inainte de **terminarea** propriu-zisa a acesteia.

trap 'rm /tmp/ps$$; exit' 2

Obs: semnalul de intrerupere e identificat prin **2**, iar fis. temporal se creaza in **/tmp; exit incheie procedura shell;**

Exemplu: o procedura shell in care executia se reia dupa tratarea lui **trap**

Efect:- procedura examineaza fiecare catalog al catalogului curent, afiseaza numele catalogului si executa c-zile introduse de la terminal pina se tasteaza CTRL-d(sfirsit de fisier) sau se receptioneaza o intrerupere;

-daca intreruperea apare in timp ce se executa o comanda introdusa , ea este ignorata.

-daca intreruperea apare in timp ce se asteapta introducerea unei c-zi ea are efectul de terminare a executiei procedurii;

```
d= 'pwd'
for i in  *
do  if  [ -d $d/$i]
        then  cd $d/$i
                while  echo "$i:"
                        trap exit  2
                        read x
                do trap : 2 ; eval  $x; done
fi
done
```

Controlul joburilor

Programe care ruleaza in prim-plan ("**foreground**")

Programe care ruleaza in fundal ("**background**")

Pot fi lansate de la acelasi treminal dar **numai cel din prim plan** va comunica direct cu terminalul.

Pot exista mai multe programe in fundal, deconectate de la terminal.

Exemplu:

Lansarea executiei unei c-zi in fundal prin introducerea caracterului **&**

```
$gv  fis.ps  &
[1]  12804
$
```

Efect: lanseaza in fundal programul gv ("gostview") care vizualizeaza un fisier in format Postscript intr-o fereastra separata, redind controlul utilizatorului dupa ce afiseaza un nr de job si PID-ul procesului care ruleaza programul (12804)

Controlul joburilor

Comanda jobs

Efect: produce o lista a tuturor programelor care ruleaza in fundal si indica starea fiecaruia:

```
$jobs
[1]+ Running  gv fis.ps  &
```

Obs: Semnul + insoteste job-ul care va reveni implicit in prim plan.

Comanda **fg**

Efect: aducerea in prim plan a unui job din fundal

```
$ fg %1
```

Obs - arg. procesului e precedat de **%**;

- daca el lipseste se aduce procesul prevazut cu **+** in lista produsa de **jobs**.

Controlul joburilor

Comanda bg (background)

Un job care ruleaza in prim plan poate fi suspendat (oprit) fara a fi terminat prin CTRL-Z. C-da **bg** background permite trecerea in fundal a unui job suspendat.

```
$ gv Dev.ps.gz
CTRL-Z
[1]+ Stopped  gv Dev.ps.gz
$bg
[1]+ gv Dev.ps.gz. &
```

Efect: pt executia unui program in fundal, dar pt care s-a uitat introducerea caracterului & la sfirsitul liniei.

Se poate cere terminarea unui job utilizind kill si numarul job-ului

```
$ kill %1          (%1= in locul pid-ului)
```

Daca se actioneaza **return => mesajul**

```
[1] Terminated gv Dev.ps.gz
```

Obs: Desi au fost prez ca si c-zi externe, **bg**, **fg** si **jobs** functioneaza si pe post de c-zi interne shell.

Alte facilitati ale interpretoarelor de c-zi

Istoria c-zilor:

Comanda history:

$ history 5

497 ls –l

498 ftp b519

499 ftp b519 timisoara.roedu.net

500 exit

501 history

Navigarea cu sageti cea mai simpla. Nu toate interpretoarele permit acest lucru

Alta varianta: **!n-** permite revenirea la c-zi anterioare in lista produsa de **history**

Exemplu: !497 –duce la reexecutarea c-zii ls –l.

!-4 adresare relativa fata de comanda curenta

Alte facilitati ale interpretoarelor de c-zi

Aliasuri

Scop: posibilitatea de a defini nume noi pentru o serie de c-zi (alias-uri).

Sintaxa (**bash**):

alias ri="rm-i"

alias clean="rm*~ .*~ core *.bak"

alias lr="ls-lR"

Efect: **ri <=> "remove cu interogare"**

clean <=> sterge din catalogul curent toate fisierele cu terminatia ~ si fisierul core

lr <=> listare lunga recursiva

Obs. Aceste definitii se introduc in fisierul de configurare al interpretorului; Aceste fisiere se genereaza in fiecare catalog si au nume specifice diverselor intrepretoare (. bash_profile)

In cazul sh unde nu exista alias se obtine un efect asemanator prin introducerea unor variabile de mediu.

Alte facilitati ale interpretoarelor de c-zi

Functii

O structura intermediara intre alias-uri si proceduri shell **= functiile- cu o structura asemanatoare cu cea a functiilor din limbajul C.**

Exemplu:

O functie echivalenta cu alias-ul **lr** :

```
lr ( ) { ls –lR “$@” ; }
```

Obs: $@ = toti parametrii din linia de c-da

@ = o variabila de mediu

Dupa definitie o functie ramine valabila pe toata durata sesiunii curente, la fel ca si un **alias**.

Functiile pot si definite si in cadrul unor **proceduri shell**.

Apeluri sistem pentru operatii cu fisierele

Apeluri sistem = interfete puse la dispozitia programatorilor de **sistemul de operare** pt a facilita **accesul la diverse servicii ale acestuia**

Sintactic apar in **programele scrise in limbaje de nivel inalt ca si un apel obisnuit de functie.**

Difera modul in care se realizeaza legatura intre program si aceste apeluri sistem:

- pt. functiile obisnuite: codul functiei – intr-o biblioteca si este adaugat programului in care se apeleaza functia in faza de editare de legaturi;
- pt apeluri sistem **codul din biblioteca** are **doar** rolul de a pregati intrarea in sistemul de operare **precum si** preluarea rezultatelor de la acesta si transmiterea lor spre programul apelant. **Tratarea lor** pr-zisa are loc intr-o secventa care face parte din **nucleul sistemului de operare**. La programul utilizatorului se adauga in faza de **editare de legaturi numai portiunea din biblioteca**;

Intrarea in sistem = regimul de lucru **privilegiat si selectarea secventei de cod care ofera serviciul solicitat. La terminarea serviciului, se revine in mod neprivilegiat si se returneaza eventualele rezultate.**

Apeluri sistem fundamentale

Descriptori de fisier

Toate fisierele deschise de un proces sunt identificate prin descriptori de fisier (un intreg ne-negativ atribuit de nucleu la deschiderea fisierului)

Deschiderea unui fisier = operatia prin care se creaza un canal de comunicare intre proces si fisier

Interpretoarele de c-zi deschid pt fiecare proces trei fisiere:

- **fisier standard de intrare- descriptor 0;**
- **fisier standard de iesire - descriptor 1;**
- **fisier standard de eroare - descriptor 2.**

Obs Exista un numar maxim de fisiere deschise cu care un proces poate opera la un moment dat, de regula 32 sau 64.

Constante simbolice pt descriptorii de fisiere standard: de forma STDIN_FILENO, STDOUT_FILENO, STDERR_FILENO …definite in fisierul antet (<unistd.h>)

Apeluri sistem fundamentale

Apelul sistem open - pt deschiderea unui fisier

#include <sys/types.h> - apelul unor functii predefinite in fisiere header
#include <sys/stat.h> prin directiva de procesare #include<fis.h>
#include <fcntl.h>

int open (const char **pathname* ,int *oflag*, …/* , mode_t *mode/)**

Efect: daca apelul se executa fara eroare valoarea ne-negativa returnata reprezinta **descriptorul de fisier** alocat fisierului deschis. La eroare se returneaza -1.

I arg. = **numele de cale al fisierului care va fi deschis**

al-II-lea arg. = **optiunile de deschidere**; Prin insumare logica a cte. din lista!!!!!

al-III-lea arg. = apare numai daca la deschidere are loc si **crearea** fisierului;

Apeluri sistem fundamentale

Optiuni de deschidere – constante asociate

- **O_RDONLY-** deschide fisierul pt citire;
- **O_WRONLY-** deschide fisierul pt scriere;
- **O_RDWR -** deschide fisierul pt citire si scriere (actualizare);
- **O_APPEND – adauaga la sf fisierului la fiecare op de scriere**;
- **O_CREAT – creeaza fisierul daca el nu exista + al-III-lea argument**
- **O_EXCL – genereaza o eroare daca s-a specificat si O_CREAT, in caz ca fisierul exista.**
- **O_TRUNC – lungimea fisierului se truncheaza la zero;**
- **O_NOCTTY-**
- **O_NONBLOCK- periferic orientat pe bloc sau FIFO – op cu el nu blocheaza procesul;**
- **O_SYNC – la efectuarea oricarei operatii de scriere se asteapta pina la terminarea operatiei la nivel fizic.**

Obs – valoarea descriptorului returnata de nucleu este intotdeauna cea mai mica posibila;
– pt redirectarea fisierelor standard de intrare si iesire;

Apeluri sistem fundamentale

Exemplu:

open(“fis.txt”, O_WRONLY | O_CREAT | O _TRUNC);

Trebuie inchis dupa ce a fost scris si apoi redeschis pt citire.

Varianta in care nu mai e necesara inchiderea.

open(“fis.txt”, O_RDWR | O_CREAT | O _TRUNC);

Apeluri sistem fundamentale

Apelul sistem creat

```
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
int creat (const  char *pathname , mode_t mode );
```

Efect -Se returneaza o valoare intreaga reprezentind descriptorul fisierului si care poate fi folosit pentru indicarea fisierului la apelul unor functii;

-Fisierul nou creat este implicit de lungime zero.

Apeluri sistem fundamentale

Exemplu: Se creaza un fisier cu drepturi de citire scriere si executare pt toate tipurile de utilizatori

```
#include <sys/stat.h>
int main (void) {
        int fd;
        if ((fd = creat ("fis.gol", 0777)) <0){
                printf("eroare creare\n");
                exit (1)
        }
```

Apelul sistem close

Deconecteaza de la un proces un fisier deschis.

#include <unistd.h> fuctiile biblioteca

int close (int *filedes*);

Efect: -returneaza 0 in caz de succes si -1 in caz de eroare;

-inchiderea fisierului produce eliberarea eventualelor blocari pe care procesul le detine asupra fisierului;

Apelul sistem lseek

Cu fiecare fisier, Unix asociaza un ***indicator de prelucrare*** sau ***deplasament curent*** = un interg ne-negativ relativ la inceputul fisierului, initializat la zero la deschiderea fisierului (daca nu e specificata op: O_APPEND).

Op. de citire si scriere incep de la valoarea curenta a deplasamentului si au ca efect incrementarea acestuia cu numarul de octeti prelucrati.

Este posibila si **pozitionarea explicita** a unui fisier cu ajutorul lui **lseek**.

#include <sys/types.h>

#include <unistd.h>

off_t lseek(int *filedes*, off_t *offset*, int *whence*);

Apelul returneaza noua valoare a deplasamentului in caz de succes sau -1 in caz de eroare.

Valorile posibile pentru al-III-lea parametru se specifica prin constante simbolice in functie de care se interpreteaza valoarea celui de-al II-lea parametru si deci rezultatul operatiei.

- Daca *whence* = **SEEK_SET**; deplasamentul=offset octeti fata de inceputul fisierului
- *whence* = **SEEK_CUR**; se aduna la valoarea curenta, poate poz sau neg
- *whence* = **SEEK_END**; deplasam egal cu lung curenta + offset

Prin apelul sistem **lseek** este posibil sa se determine valoarea deplasamentului curent in fisierul indicat de **descriptorul fd**:

off_t poz ;

poz = lseek (fd, 0, SEEK_CUR);

Apeluri sistem fundamentale

Apelurile sistem read si write

#include <unistd.h>

ssize_t read (int *filedes,* void **buf*, size_t *nbytes*);

La o executie fara erori a apelului, valoarea returnata e nr. de octeti cititi din fisier, iar in caz de eroare -1.

- I arg – descriptorul fisierului din care se citeste
- al-II-lea arg – adresa in memorie a tamponului unde se depun octetii cititi
- al-III-lea arg – nr de octeti ce se doresc a fi cititi

Exemplu:

```
ssize_t m;
char sir[10];
m=read(fd, sir,10);
```

Pentru scriere intr-un fisier:

```
#include <unistd.h>
ssize_t write (int filedes,  const void *buf, size_t  nbytes);
```

Efect- valoarea rezultata este egala cu nr de octeti scrisi adica al treilea argument al apelului; in caz de eroare -1.

OBS: Erorile de scriere au de cele mai multe ori drept cauze umplerea perifericului(disc, partitie de disc), sau depasirea dimensiunii maxime de fisier acceptata de o anumita implementare.

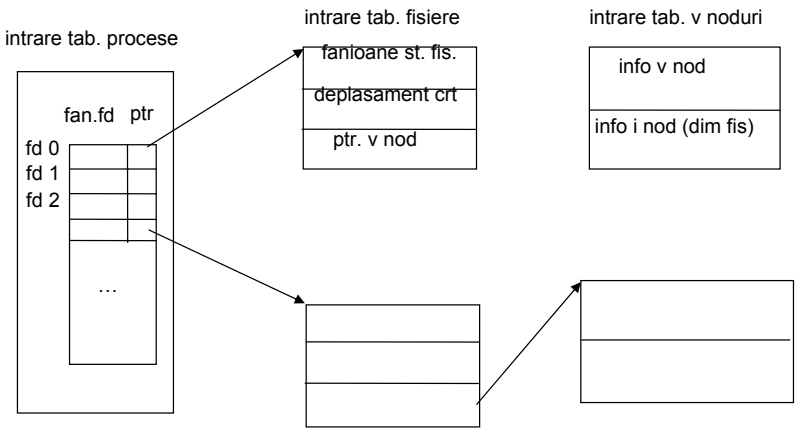
Exemplu:

```
ssize_t n;
n=write(fd, "abcdefghij", 10);
```

Partajarea fisierelor - nucleul Unix ofera suport pentru partajarea fisierelor deschise intre mai multe procese

- nucleu utilizeaza structuri de date

structuri de date in nucleu pt fisiere deschise



1. Pt fiecare proces se rezerva o intrare intr-o tabela de procese;

   - Ca parte a acestei intrari apare o tabela de descriptori de fisier, cu cite o intrare pentru fiecare fisier deschis de proces.

   - Fiecare descriptor de fisier contine fanioanele descriptorului si un pointer la o intrare in tabela de fisiere deschise.

2. Intrarea in tabela de fisiere deschise are urmatoarele cimpuri:

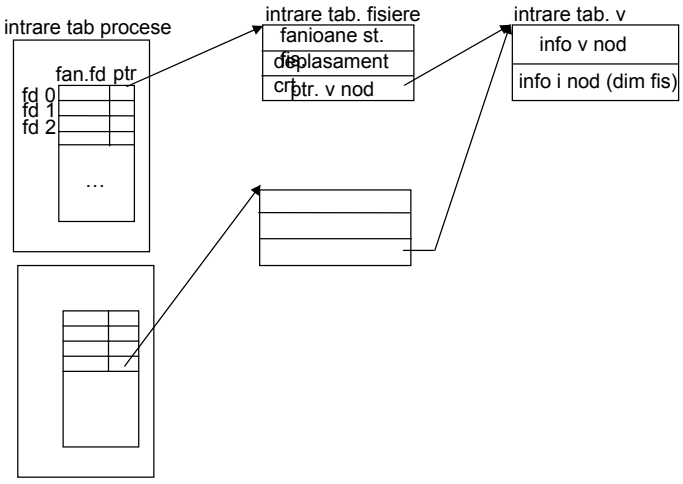
   - fanioanele de stare ale fisierului;

   - deplasamentul curent al fisierului;

   - un pointer la tabela de v-noduri;

3. Pt fiecare fisier fizic distinct deschis exista o intrare in tabela de v-noduri. In esenta o asemenea intrare contine informatii despre fisier si pointeri spre functiile care opereaza asupra fisierului. Info despre fisier se citesc de pe disc cand este deschis fisierul.

Partajarea unui fisier intre doua procese independente



- Dupa fiecare operatie **write** deplasamentul in fisier se incrementeaza cu numarul de octeti scrisi (daca se depaseste dimensiunea curenta a fisierului se va modifica valoarea acestuia din i-nod, = cu valoarea curenta a deplasamentului.
- Daca fisierul e deschis cu O_APPEND, se pozitioneaza fanionul corespun-zator din intrarea in tabela de fisiere- deplasamentul in fisier la dimensiunea curenta din i-nod **=>** scrierea are loc la sfirsitul fisierului.
- Apelul lseek are ca efect doar modificarea deplasamentului in fisier; nu se produce nici o operatie de transfer pr-zisa.

Obs

-Este posibil ca mai multi descriptori de fisier sa indice spre aceeasi intrare in tabela de fisiere (dupa un apel sistem **dup** sau **fork**);

- Fanioanele din descriptorul de fisier – se aplica unui singur descriptor intr-un anumit proces;

- **Fanioanele de stare ale fisierului sunt valabile pentru toti descriptorii din toate procesele care indica spre aceeasi intrare in tabela de iesire**.

Operatii atomice

a) Adaugarea la un fisier

Optiunea O_APPEND- la deschiderea unui fisier nucleul efectueaza implicit pozitionarea la sfirsitul fisierului inaintea scrierii, ca parte a executiei **write**, fara a fi necesar un apel sistem special.

b) Crearea unui fisier

Optiunile O_CREAT si O_ EXCL. Verificarea existentei fisierului ce urmeaza a fi creat se realizeaza implicit de catre nucleu ca parte a executiei apelului **open.**

Apelurile sistem dup si dup2

Un descriptor de fisier existent poate fi duplicat prin:

```
#include <unistd.h>
int dup (int filedes) ;
int dup2(int filedes, int filedes2);
```

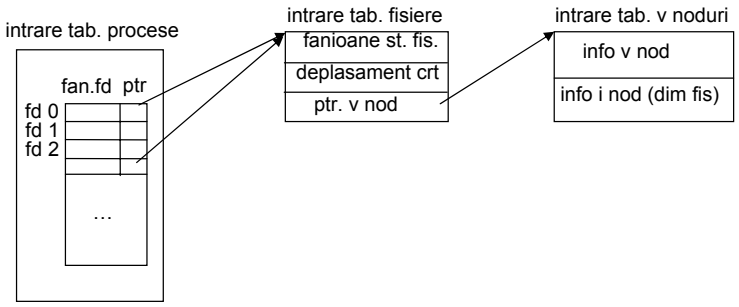
Efect: ambele returneza noul descriptor de fisier in caz de succes si -1 la eroare.

Descriptorul returnat de **dup** este cel mai mic ca valoare intre cele diponibile.

Pt **dup 2** se specifica in apel valoarea noului descriptor ***filedes***.

Noul descriptor va partaja cu cel pe care il duplica aceeasi intrare in tabela de fisiere deschise.

Apelurile de duplicare se folosesc pt redirectare.



Apeluri sistem pentru cataloage

Starea fisierelor

Tipuri de fisiere

Drepturi de acces la fisiere

Stabilirea proprietarului pentru fisiere si cataloage

Stabilirea drepturilor de acces la creare

Schimbarea drepturilor de acces la un fisier existent

Alte apeluri cu efect asupra cataloagelor

Crearea si stergerea cataloagelor

Apelurile chdir, fchdir si getcwd

Deschiderea citirea si inchiderea cataloagelor

Starea fisierelor

Informatiile despre starea unui fisier, pastrate in **i-nodul** fisierului devin accesibi le unui program prin apelurile sistem **stat** cu prototipurile:

#include <sys/types.h>
#include <sys/stat.h>
int stat (const char *_pathname_, struct start stat *_buf_);
int fstat (int _filedes_, struct start stat *_buf_);
int lstat (const char *_pathname_, struct *_buf_);

1) - da info despre fisierul specificat prin nume de cale nu e necesar sa fie deschis;
2) - pt fisiere deschise identificate prin descriptor;
3) – similar cu 1) dar fisierul specificat e legatura simbolica

Al-II-lea arg. e pointer spre o structura in care se vor returna info.

Aceasta structura este definita in **stat.h** astfel:

```
struct stat {
    mode_t      st_mode;        /* file type & mode (permission)*/
    ino_t       st_ino;         /* i-node number (serial number)*/
    dev_t       st_dev;         /* device number (filesystem) */
    dev_t       str_dev;        /* device number for special files */
    nlink_t     st_nlink;       /* number of links*/
```

```
    uid_t       st_uid;         /* user ID of owner */
    gid_t       st_gid;         /* group ID of owner */
    off_t       st_size;        /* size in bytes, for regular files*/
    time_t      st_atime;       /*time of last acces */
    time_t      st_mtime;       /*time of last modification */
    time_t      st_ctime;       /*time of last file status change*/
    long        st_blksize;     /*best I/O block size*/
    long        st_blocks;      /*number of 512-byte blocks allocated*/
};
```

Obs

st_xxxx reprezinta cimpul in care e codificata informatia referitoare la …

Utilizarea info returnate de structura stat se realizeaza cu ajutorul unor macrouri.

Tipuri de fisiere

Pt determinarea tipului unui fisier, codificat in cimpul **st_mode** se folosesc **macro-urile de mai jos:**

Macro st_mode	**Tipul fiiserului**
S_ISREG	fisier obisnuit
S_ISDIR( )	fisier catalog
S_ISCHR( )	fisier special catalog
S_ISBLK ( )	fisier special bloc
S_ISFIFO( )	conducta (pipe) sau fifo
S_ISLNK ( )	legatura simbolica
S_ISSOCK ( )	socket

Drepturi de acces la fisiere

Drepturi speciale: ***setuid***, ***setgid***, ***sticky bit***.

Fiecarui proces i se asociaza cel putin 6 identificatori

- Identificatorul utilizatorului real;
- Identificatorul grupului real;
- Identificatorul utilizatorului efectiv;
- Identificatorul grupului efectiv;
- Identificatorul salvat la modificarea utilizatorului;
- Identificatorul salvat la modificarea grupului.

Identificatorul utilizatorului real si identificatorul grupului real se obtin din fisierul: de parole: /etc/passwd/ la intrarea in sesiune. Nu se modifica pe parcursul sesiunii.

Identificatorul utilizatorului efectiv si identificatorul grupului efectiv sunt cei folositi la determinarea drepturilor de acces la fisiere in timpul executiei proceselor: - de regula au aceeasi valoare ca si cei reali

- intr-un proces in care se executa un program(memorat intr-un fisier executabil) ce are bitul set-user-id pozitionat pe 1, pe durata executiei acelui program, identificatorul utilizatorului efectiv devine egal cu identificatorul proprietarului fisierului.

Identificatorii savati contin copii ale identificatorilor efectivi si se utilizeaza tot in programe cu set-user-id.

Valorile bitilor set-user-id si set-group-id se verifica prin compararea valorii din cimpul st_mode cu constantele S_ISUID respectiv S_ISGID.

Pentru verificarea bitilor de acces obisnuiti se folosesc constantele din tabel

Masca st_mode	**Semnificatia**
S_IRUSR	drept citire utilizator
S_IWUSR	drept scriere utilizator
S_IXUSR	drept executie utilizator
S_IRGRP	drept citire grup
S_IWGRP	drept scriere grup
S_IXGRP	drept executie grup
S_IROTH	drept citire altii
S_IWOTH	drept scriere altii
S_IXOTH	drept executie altii

Utilizarea bitilor de acces in timpul apelurilor sistem

- **La op de deschidere a fisierului prin nume se verifica daca apelantul** (dat prin identificatorul utilizatorului efectiv) are drept de executie pt fiecare din cataloagele prezente in nume(bitul de executie se mai numeste si bit de cautare).
- Dreptul de citire pt un fisier determina daca apelantul poate deschide un fisier existent pt citire(fanioanele O_RDONLY si O_RDWR) ale apelului **open**).
- Dreptul de scriere pt un fisier determina daca apelantul poate deschide un fisier existent pt scriere (fanioanele O_WDONLY si O_RDWR) ale apelului **open**).
- Pentru a specifica fanionul O_TRUNC la **open** trebuie ca apelantul sa aibe drept de scriere asupra fisierului.
- Pentru a putea crea un fisier intr-un catalog, apelantul trebuie sa aiba drepturi de scriere si executie in acel catalog.
- Pt a sterge un fisier existent apelantul trebuie sa aiba drepturi de scriere si de executie in catalogul care contine fisierul.
- Pt a lansa in executie – fisierul trebuie sa fie unul obisnuit iar apelantul sa aiba drept de executie pe fisierul respectiv.

- Verificarile nucleului la incercarea de executie a oricarei operatii sunt functie de:
  - Identitatea utilizatorului efectiv(UID-ul efectiv al procesului)
  - Grupul (grupurile din care face parte acesta)
  - Identitatea proprietarului fisierului

- Verificarile sunt;
  - Daca UID-ul efectiv e zero (root) → accesul e permis
  - Daca UID-ul efectiv coincide cu proprietarul fisierului → accesul e controlat de bitii de acces pentru proprietar
  - Daca GID-ul efectiv coincide cu GID-ul fisierului se aplica drepturile de acces pt grup ale fisierului
  - In alte situatii se aplica drepturile de acces pt restul utilizatorilor (*world*)

Stabilirea proprietarului pentru fisiere si cataloage

- La crearea unui nou fisier sau catalog identitatea proprietarului acestuia va coincide cu UID-ul efectiv al procesului
- Pt stabilirea GID-ului exista variante:
  - GID-ul fisierului (catalogului) = GID-ul efectiv al acestuia
  - GID-ul fisierului (catalogului) = GID-ul catalogului in care se produce crearea (Unix BSD)

Stabilirea drepturilor de acces la creare

=drepturile initiale de acees (masca de acces sau umask)

La intrarea in sesiune se stabileste implicit o valoare umask care controleaza drepturile de acces la noile fisiere si cataloage.

Valoarea umask = complementul drepturilor de acces pt scriere si citire acordate implicit

Exemplu:

umask=02 → drepturi 664

Comanda umask

Efect: afiseaza valoarea curenta a mastii

Exemplu:

```
$ umask
002
$ touch newfile
$ mkdir newdir
$ ls –l | grep new
drwxrwxr-x   2   ada      ada      4096     May      8 21:37  newdir
-rw-rwxr--   1   ada      ada      4096     May      8 21:36  newfile
```

Stabilirea drepturilor de acces la creare

Apelul sistem umask

```
#include <sys/types.h>
#include <sys/stat.h>
mode_t umask (mode_t cmask);
```

Efect: permite ca din programe sa se controleze valoarea mastii de acces

Argumente:

cmask: **se formeaza prin SAU pe bitii intre oricare 2 constante din tabelul Masca st_mode**

efectul noii masti de creare dureaza pana la terminarea procesului curent cand se revine la valoarea initiala a mastii(uzual 002)

Schimbarea drepturilor de acces la un fisier existent

Apelurile sistem chmod si fchmod

#include <sys/types.h>

#include <sys/stat.h>

int chmod (const char **pathname*, mode_t mode);

int fchmod (int filedes, mode_t mode);

Ambele returneza zero in caz de succes si -1 in caz de eroare.

Schimbarea e permisa daca UID-ul efectiv al procesului = proprietarul fisierului sau su perutilizator

Argumentul *mode*

- da noua configuratie a bitilor de acces

- SAU logic intre constantele din tabel:

mode	**Semnificatia**
S_ISUID	activeaza set-user-ID
S_ISGID	activeaza set-group-ID
S_ISVTX	activeaza sticky-bit (“saved text”)
S_IRWXU	citire, scriere si executie pt proprietar
S_IRUSR	citire, pt proprietar
S_IWUSR	scriere, pt proprietar
S_IXUSR	executie, pt proprietar
S_IRWXG	citire, scriere si executie pt grup
S_IRGRP	citire grup
S_IWGRP	scriere grup
S_IXGRP	executie grup
S_IRWXO	citire, scriere si executie pt altii
S_IROTH	citire altii
S_IWOTH	scriere altii
S_IXOTH	executie altii

Apeluri sistem pentru schimbarea proprietarului

#include <sys/types.h>

#include <unistd.h>

int chown (const char *_pathname_, uid_t _owner_, gid_t _group_);

int fchown (int filedes, uid_t _owner_, gid_t _group_);

int lchown (const char *_pathname_, uid_t _owner_, gid_t _group_);

Returneza zero in caz de succes si -1 in caz de eroare.

OBS - Initial numai superutilizatorul poate proprietarul

- O contanta inclusa in <unistd.h. controleaza modul de opearare al apelurilor: _POSIX_CHOWN_RESTRICTED

- numai supeutilizatorul poate schimba UID

….

Apeluri sistem cu efect asupra cataloagelor

Crearea si stergerea cataloagelor- Apelul mkdir

#include <sys/types.h>
#include <sys/stat.h>
int mkdir (const char *_pathname_, mode_t _mode_);

Efect: - catalogul nou creat cu numele de cale dat de sirul spre care indica ***pathname***, va fi gol (se creaza doar intrarile **.** si**..**). Drepturile de acces la catalog sunt date prin ***mode + masca***.

Stergerea unui catalog gol - Apelul rmdir:

#include <unistd.h>
int rmdir (const char *_pathname_);

Obs Daca nr de legaturi devine zero prin acest apel iar catalogul nu mai e deschis in nici un proces, spatiul ocupat de catalog va fi eliberat.

Apeluri sistem cu efect asupra cataloagelor

Apelurile chdir, fchdir si getcwd

#include <unistd.h>

int chdir (const char ****pathname* **);

int fchdir (int ***filedes*** **);**

Ambele returneaza 0 in caz de succes si -1 in caz de eroare.

Efect: modifica catalogul de lucru curent al unui proces. Acesta este un atribut al fiecarui proces.

1) Cere specificarea noului catalog curent ;se face prin **nume de cale**;
2) Se utilizeaza ca argument **un descriptor de fisier**, deci noul catalog curent trebuie sa fi fost **anterior deschis de catre proces**.

Obs: la inceputul unei sesiuni de lucru catalogul curent al shell-ului este fixat automat ca fiind catalogul gazda al utilizatorului.

Nucleul pastreaza pentru fiecare proces i-nodul si identificatorul perifericului.pt catalogul de lucru curent.

Apeluri sistem cu efect asupra cataloagelor

Pt a obtine **numele de cale al catalogului** e necesara sa se porneasca de la aceasta info si sa se parcurga ascendent ierarhia de cataloage pana la radacina.

Apelul getcwd

#include <unistd.h>

char *getcwd(char ****buf*, size_t** ***size***);**

Argumentele reprezinta: adresa unui tampon (*buf*) si dimensiunea acestuia. Tamponul sa fie suficient de mare pt a retine numele de cale absolut, plus caracterul nul de terminare a sirului.

Apeluri sistem cu efect asupra cataloagelor

Exemplu: **Modificarea catalogului curent , citirea numelui de cale al acestuia si afisarea rezultatului obtinut**

```
#include <unistd>
int main (void){
    char buf[4096]
    if (chdir (“/usr/java jdk3.1/demo”) < 0 ){
    perror (“chdir error”);
    exit (1);
    }
    if (getcwd (buf , sizeof(buf)==NULL){
    perror (“getcwd error”);
    exit (2);
    }
    print (“cwd=%s\n” , buf);
    exit(0) ;
}
```

Deschiderea, citirea si inchiderea cataloagelor

Deschiderea: functie speciala din biblioteca standard de **I/E, opendir** cu prototipul:

```
#include <sys/types.h>
#include <dirent.h>
DIR *opendir(const char dirname)
```

Argumentul functiei e numele de cale al catalogului ce urmeaza a fi deschis.

Valoarea returnata e un pointer la o structura DIR definita in **<dirent.h>**

Inchiderea: functia, **closedir**

```
#include <dirent.h>
int closedir(DIR dirptr);
```

Argumentul pentru closedir trebuie sa fie un pointer la o structura DIR, ceea ce determina ca la deschiderea catalogului sa fie prevazuta salvarea pointerului returnat.

Deschiderea, citirea si inchiderea cataloagelor

Citirea: **Dupa ce un catalogul a fost deschis se pot obtin pe rind intrarile catalogului apelind functia readdir:**

```
#include <sys/types.h>
#include <dirent.h>
struct dirent *readdir(DIR dirptr);
```

Arg transmis functiei trebuie sa fie un pointer valid la o structura DIR, obtinut printr-un apel anterior **opendir**.

Valoarea returnata de functie este un **pointer la o structura dirent** cu definitie dependenta de implementare dar care trebuie sa contina cel putin urmatoarea definitie:

```
struct dirent {
ino_t  d_ino;   /*numarul i-nodului pt. intrare*/
char   d_name[NAME_MAX + 1];   /*numele fisierului*/
};
```

OBS NAME_MAX este de 255

- la primul apel **readdir** se va citi in **dirent** prima intrare a catalogului
- dupa fiecare citire cu succes indicatorul catalogului din **dir** va fi avansat la urmatoarea intrare.
- cind se ajunge la sfirsitul intrarilor catalogului **readdir** returneaza pointerul nul.