- **int lseek (int fd, int pozitie, int orig)** corespunde lui *fseek* din biblioteca standard C;
  - modificarea pozitiei curente este realizata fara operatii de citire, scriere;
  - **pozitie** poate fi una dintre: **SEEK_SET, SEEK_CUR, SEEK_END**;
  - **orig** reprezinta pozitia curenta (cu cati octeti se deplaseaza) in raport cu parametrul **pozitie; orig** poate fi si o valoare negativa;

Blocarea si deblocarea fisierelor

Ne amintim de tabelele de blocaj, care corespund fiecarei intrari din tabela de i-noduri.

Blocajele se pun cu ajutorul apelului:

- **int fcntl (int fd, int com, ...)**
  - apelul realizeaza operatiile de blocare si deblocare a fisierelor
  - **com:** **F_GETFD**
    In acest caz apelul are doar doi parametrii si returneaza **int** (masca) care contine informatie pe biti despre fisiere
    F_SETFD
    In acest caz apelul are trei parametrii, ultimul fiind **int mask** si reprezinta aceleasi informatii pe biti despre fisiere
  - constanta **FD_CLDEXEC** (true/false) semnifica inchiderea (sau nu) a descriptorilor
    ex.:

```
int attr;
attr = fcntl(fd, F_GETFD);        //citirea atributelor existente
attr |=FD_CLDEXEC;
fcntl(fd,F_SETFD,attr);
```

  - **com:** **F_GETFL**
    F_SETFL
    Obs.: Nu toate atritbutele pot fi modificate in runtime (O_APPEND, O_NONBLOCK, O_NDELAY, O_SIG)

  - **com:** (In aceste situatii apelul are trei parametrii, al treilea fiind de tip ***struct flock**, vedeti descrierea blocajelor si a acestei structuri de pe pagina 2)

    F_SETLCK **:** pune un blocaj de tip consultativ
    F_SETLKW **:** pune un blocaj de tip imperativ
    F_GETLCK **:** in structura *flock* se testeaza existenta unui blocaj incompatibil cu blocajul nostru (in acest caz campurile structurii sunt cumplute cu informatiile blocajului existent → devine folositor campul *pid*; altfel in *l_type* vom avea *F_UNLCK*)

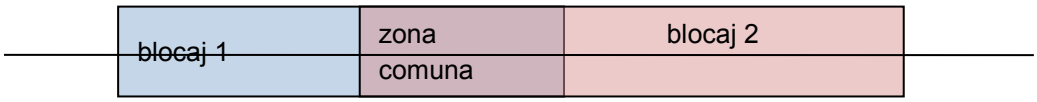
    Obs.: Pentru ca blocajele imperative sa actioneze efectiv trebuie ca fisierul pe care operam sa aiba indicatorul *set_gid* setat fara drept de executie.

Blocaje

Blocajele sunt proprietatea exculsiva a procesului care le pune. (*root* poate da *kill*, iar la terminarea procesului este ridicat blocajul).

Blocajele pot fi **exclusive** sau **partajate**.

ex.: daca cele doua blocaje din schema de mai jos sunt partajate, atunci ele pot coexista, daca sunt exclusive insa nu pot avea zona comuna



Din punct de vedere al modului de actionare al blocajului, blocajele pot fi clasificate ca fiind:

- **consultative** (doar o incercare, nu duc la blocarea operatiilor de citire/scriere ale altor procese)
- **imperative** (duc la blocarea efectiva a zonelor de memorie)

Urmatoarea structura descrie blocajele sau tentativele de blocaj:

```
struct flock{
        short l_type;       /* Valorile pe care le poate lua sunt:
                                 ▪ F_RDLCK : blocaj partajat
                                 ▪ F_WRLCK : blocaj exclusiv
                                 ▪ F_UNLCK : ridicarea blocajului
                            */
        short l_mask;       /* Descrie pozitionarea blocajului
                               (SEEK_SET,SEEK_CUR,SEEK_END)
                            */
        int l_start;    /* Descrie pozitia primului octet al zonei blocate relativ la l_mask */
        int l_len;          /* Descrie lungimea zonei blocate */
        int pid;            /* pid-ul procesului care a pus blocajul */
}
```

Operatii cu fisiere de tip director

Header-ul **dirent.h** defineste tipul **DIR**

- **DIR * opendir (char * nume)**
  - returneaza NULL daca fisierul *nume* nu exista sau exista, dar nu este de tip director

- **struct dirent * readdir (DIR * d)**
  - utilizat pentru a citi intrarile dintr-un director : pentru fiecare intrare returneaza un pointer catre o structura *dirent*; dupa ce au fost parcurse toate intrarile, returneaza NULL;

    Obs.: In *struct dirent* avem campul **char * d_name**, care reprezinta numele intrarii

- **void closedir (DIR * d)**
  - realizeaza inchiderea directorului

- **void rewinddir (DIR * d)**
  - se revine la prima intrare...

COMUNICAREA INTRE PROCESE
(continuare)

Pipes
(Fisiere de tip tub)

Sunt fisiere de tip special.

Se caracterizeaza prin:
- nu au descriptori de citire/scriere
- citirea este destructiva
- citirea si scrierea sunt blocante (exceptie: O_NONBLOCK, O_NDELAY)

Pot fi de doua tipuri: 1) cu nume
2) fara nume

1) Fisiere de tip tub cu nume

Acestea au cel putin o legatura fizica pe disc.
Din shell pot fi create folosind comanda:

$ mkfifo nume

care creeaza un fisier cu numele *nume* de tipul **.p** (pipe)

Deschiderea, citirea, scrierea si inchiderea se realizeaza cu apelurile uzuale (open cu WRONLY, read, write, close).

Obs.: Deschiderile in scriere reusesc intotdeauna. Deschiderile in citire esueaza daca nu e deschis un descriptor in scriere, astfel observam ca pipe-urile reprezinta o modalitate de sincronizare a doua procese.

2) Fisiere de tip tub fara nume

$c_1 \mid c_2 \mid \ldots \mid c_n$ → n-1 pipe-uri fara nume

Acestea nu au nicio legatura fizica pe disc, exista doar ca i-noduri in memorie (deci la inchiderea tuturor descriptorilor va disparea si pipe-ul).

- **int pipe (int * p)**
  - **p** este un vector alocat de minim doi intregi
  - apelul creeaza un i-nod in memorie pentru un fisier de tip *pipe* si pune in
    p[0] descriptorul in citire si in
    p[1] descriptorul in scriere
  - pipe-ul este disponibil doar procesului creator si descendentilor sai

Exemplu:

ls -1a | wc –l

- numara fisierele al caror nume nu incepe cu ‘**.**’

```
int p[2];
pipe (p);
if (fork () > 0){
                close (1);
                dup (p[1]);
                close (p[0]);
                close(p[1]);
                execlp(“ls”,”ls”,”-1a”,NULL);
}
else{
        close(0);
        dup(p[0]);
        close(p[0]);
        close(p[1]);
        execlp(“wc”,”wc”,”-l”, NULL);
}
```