**Lab 5. Directoare**

**5.2.1 Apeluri de sistem pt lucrul cu Directoare**

#include <sys/types.h>

#include <dirent.h>

DIR \*opendir(const char \*name);

int closedir(DIR \*dirp);

struct dirent \*readdir(DIR \*dirp);

void rewinddir(DIR \*dirp);

\_\_\_\_\_\_\_\_\_

**5.2.1.1 opendir/close dir**

opendir 🡪 deschide fisiere

🡪 name = calea relativa/absoluta spre directorul ce se doreste sa fie deschis

🡪 rezultatul **e un pointer tip DIR\* la directory stream** --> stream-ul are asociat un **cursor ce permite parcurgerea directorului elem cu elem**

--> la deschiderera unui director , so aloca anumite structuri interne ce vor fi eliberate la apelul **closedir**

**5.2.1.2 readdir**

**readdir** 🡪 citirea urmatorului element dintr-un director

🡪 parametru: **DIR\*** returnat de opendir

🡪 returneaza un pointer la structura de tip **struct dirent**

🡪 pt dirent: standardul POSIX garanteaza ca va contine cel putin **d\_name**

**5.2.1.3 rewinddir**

**rewinddir** 🡪 mutarea cursorului asociat directory stream-ului la **incepului unui director deschis**

🡪 param stream **DIR\*** returnat de **opendir**

**5.2.1.4 listarea continutului unui director**

**| int listDir(const char \*path){**

**| DIR\* dir = NULL;**

**| struct dirent \*entry = NULL;**

**| dir = opendir(path);** *//deschide directorul --> daca descrierea esueaza se retruneaza NULL*

**| if(dir == NULL){**

**| perror("could not open directory");** // perror afiseaza mesajul dat + “:”+cauza erorii

**| return -1;**

**| }**

**| while((entry == readdir(dir))!=NULL){** //se apeleaza readdir la fiecare iteratie pentru a citi urmatorul element din director; |  **| //** cand se ajunge la finalul directorului => readdir va returna NULL;

**| printr("%s\n",entry->d\_name);**

**| }**

**| closedir(dir);** // inchidere fisier… altfel se pierde memorie

**| return 0;**

**| }**

**5.2.2 UTILIZAREA LINK-urilor**

🡪 **link** == un tip de fisier ce face legatura cu un alt fisier

**--> pe UNIX ( inclusiv Linux) exista 2 tipuri de link-uri**

=> **hard link** = **fisier obisnuit asociat aceluiasi i-node**

-> dupa ce e creat, **hardlink-ul NU se mai deosebeste de fisierul original**

**!!OBS:--> cand ultima legatura se sterge, se sterge si i-node-ul cu continutul**

=> **symbolic link** = **fisier special ce indica locatia altui fisier**

-> daca fisierul original se sterge**, symlink-ul devine INVALID**

ex:

**| #include <unistd.h>**

**| int link(const char \*oldpath, const char \*newpath);** // pentru crearea hardlink-urilor

**| int symlink(const char \*target, const char \*linkpath);** //pt creare link-urilor simbolice

--> argumente: arg1 = calea spre un fisier existent, arg2 = calea spre link-ul ce se doreste a fi creat

**5.2.3 Extragerea de info despre fisiere**

**| #include <sys/types.h>**

**| #include <sys/stat.h>**

**| #include <unistd.h>**

**| int stat(const char\* pathname, struct stat \*statbuf);**//f1--->f2 (leg sym) stat obtine info despre f2

**| int fstat(int fd, struct stat \*statbuf);**

**|int lstat(const char\* pathname,struct stat \*statbuf);**//f1--->f2(leg sym) lstat obtine info despre f1

**🡪 obtinerea metadatelor unui fisier/director: stat/fstat/lstat**

==> parametrul statbuf de tip **struct stat** --> **se vor completa info despre fisier**

==> return : 0 succes, -1 altfe;

🡪 **diferenta stat si lstat**: **doar in cadrul legaturilor simbolice**

=> daca avem un fisier f1 ce e de fapt o legatura simbolica spre f2, stat apelat pe f1 va obtine info despre f2(fisierul spre care pointeaza legatura), iar lstat va obtine info despre f1(legatura)

**🡪 metadatele dintr-un struct stat:**

=> **off\_t st\_size** = dimensiunea fisierului in octeti ( pt FISIERE OBISNUITE)

=> **mode\_t st\_mode** = indica tipul si permisiunile fisierului

**🡪 pentru determinarea unui fisier se poate folosi:**

 **S\_ISREG** - returneaza o valoare nenula pentru un fisier OBISNUITE

 **S\_ISDIR** - returneaza o valoare nenula pentru un director

 **S\_ISLNK** -returneaza o valoare nenula daca avem o legatura simbolica

🡪 **st\_mode** identifica permisiunile unui fisier

==>**cei mai putin semnificativi 9 biti ai campului se refera la permisiuni**

==> pentru a extrage doar acei biti ---> '&' + masca 0777

**5.2.3.1 Exemplu: afisare tip si permisiune fisiere**

**| char filePath[512];**

**| struct stat statbuf;**

**| ....**

**| while((entry == readdir(dir))!=NULL){**

**| snprintf(filePath,512,"%s/%s",path, entry->d\_name);**//concatenarea caii spre folderul curent cu sep / si cu numele fis curent

**|** // se prefera snprintf in loc sprintf pt ca permite specificarea dimensiunii

**| if(lstat(filePath,&statbuf)==0){** // e nevoie de path-ul complet …de ex daca dau doar “f1” in loc de “d1/f1” se vor

//Cauta metadatele despre f1 in folder curent, nu in d1/f1

// daca se returneaza 0 inseamna ca lstat s-a executat cu succes si in statbuf avem metadate

**| if( S\_ISREG(statbuf.st\_mode)){**

**| printf("[reg]");**

**| }else if(S\_ISDIR(statbuf.st\_mode)){**

**| printf("[dir]");**

**| }else if(S\_ISLNK(statbuf.st\_mode)){**

**| printf("[lnk]");**

**| }else {**

**| printf("[unk]");}**

**| printf("[%o] ",statbuf.stmode&0777); //afisare permisiune** aplicand masca de biti 0777

**| }else{**

**| printf("[err]"); //**

**| }**

**| printf("%s\n",entry->d\_name);**

**| }**

**| ....**

**|**

**5.2.3.2 Exemplu: listare recursiva a elementelor unui folder**

**| void listRec(const char\* path){**

**| DIR \*dir = NULL;**

**| struct dirent \*entry=NULL;**

**| char fullPath[512];**

**| struct stat statbuf;**

**|**

**| dir = opendir(path); //deschidere fisier**

**| if(dir == NULL){**

**| perror("could not open directory");**

**| return;}**

**|**

**| while((entry = readdir(dir)) != NULL){** //citire element cu element

**| if(strcmp(entry->d\_name, ".")!=0 && strcmp(entry->d\_name,"..") != 0){ // altfel intram in bucla infinita**

**//** pt ca folderele ne vor duce la dir curent sau la parinte

**| snprintf(fullPath,512,"%s/%s",path, entry->d\_name);** // determinare path complet

**| if(lstat(fullPath, &statbuf) == 0){** //iau metadatele din noul path format

**| printf("%s\n",fullPath);**

**| if(S\_ISDIR(statbuf.st\_mode)){**

**| listRec(fullPath);** // daca elementul curent e director, il parcurg si pe el

**| }**

**| }**

**| }**

**| }**

**| closedir(dir);**

**|}**

**|**

**5.3.1 intrebari recap**

1. ce rol au elementele "." si ".." din fiecare director?

**--> . -> trimite la folderul curent**

**--> .. -> trimite la folderul parinte**

2. ce fct se folosesc pentru a citi continutul unui director?

**🡺 opendir (imi returneaza o structura de tip DIR\*) + readdir( primeste un parametru DIR\* si returneaza un dirent in caz de succes)**

3. cu ce apel de sistem pentru operatii pe fisiere e echivalent rewinddir?

**--> lseek(fd,0,SEEK\_SET)** 🡪 seteaza cursorul la inceputul fisierului iar **rewinddir seteaza cursorul la inceputul directorului**

4. care e diferenta intre un hard link si un symbolic link?

**🡪 hardlink --> fisier obisnuit asociat aceluiasi i-node**

**--> dupa creare, hardlink-ul nu se mai deosebeste de fisierul original**

**--> cand ultima legatura se sterge=> se sterge si i-node-ul cu continutul**

**🡪 symlink --> fisier special ce pointeaza la alt fisier**

**--> daca fisierul original se sterge, symlink-ul devine INVALID**

5.dandu-se o cale in sistemul de fisiere, cum aflam daca eleemntul e fisier sau director?

**lstat(filepath,&statbuf);**

**S\_ISDIR(statbuf.st\_mode);**

**S\_ISREG(statbuf.st\_mode);**

6. in ce camp dintr-o structura de tip struct stat se gasesc perimisiunile dintr-un fisier?

**-- pe ultimii 9 cei mai simnificativi biti**