Linux Programozás

Ismerkedés a Linux Kernellel

Virtuális állományrendszer

Állományabsztrakció

- „everything is a file”
- Az I/O perifériák úgy használhatóak mint egy fájl.
- Akár valós, akár virtuális állomány megnyithatjuk, és a leíróval kezelhetjük.
- A műveletek eltérhetnek.
- Az állományabsztrakciós felület a műveletek uniója

Állományabsztrakciós felület

- megnyitás és lezárás
- olvasás és írás
- pozicionálás
- aszinkron olvasás és írás
- könyvtár olvasása
- várakozás I/O műveletre
- I/O vezérlés
- leképezés memóriába
- szinkronizálás
- zárolás
- …

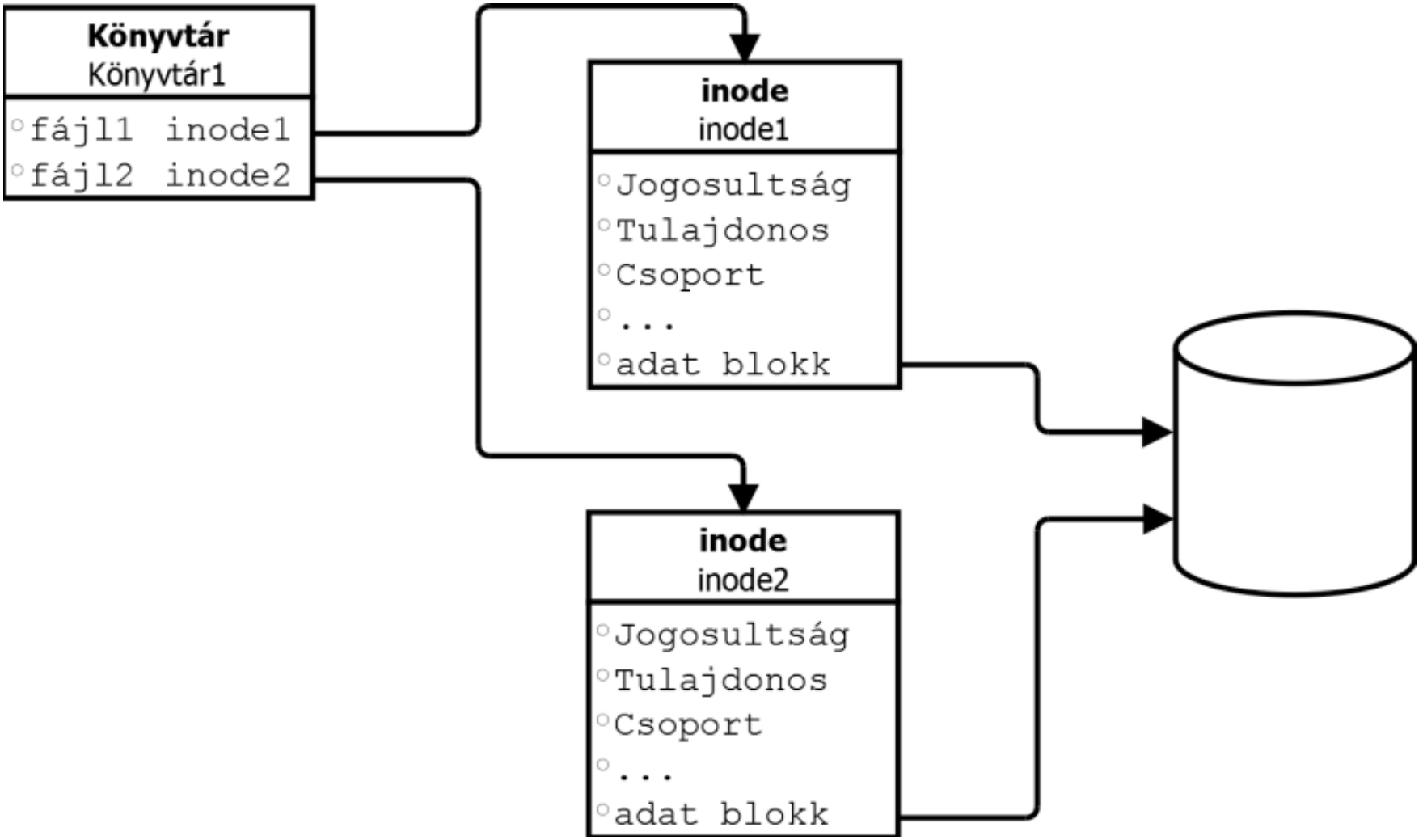
Állomány típusok

- Egyszerű állományok
- Speciális állományok
  - Eszközállományok (karakteres, blokkos)
  - Könyvtár
  - Szimbolikus hivatkozás
  - Csővezeték
  - Socket

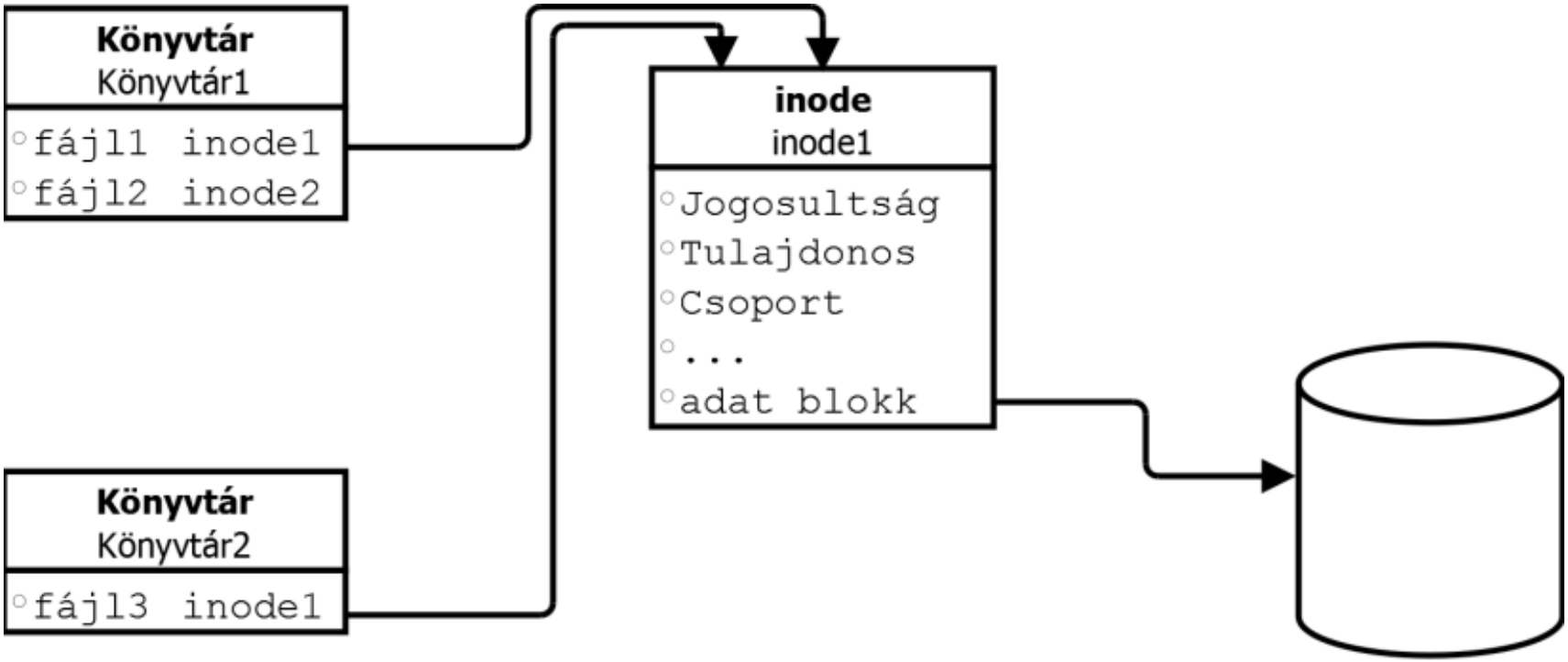
Inode (1)

- Az állomány egyedi azonosítója
- Információs tároló, ami mindent tartalmaz kivéve:
  - az állomány neve
  - az állomány tartalma
- on-disk vs. in-core inode

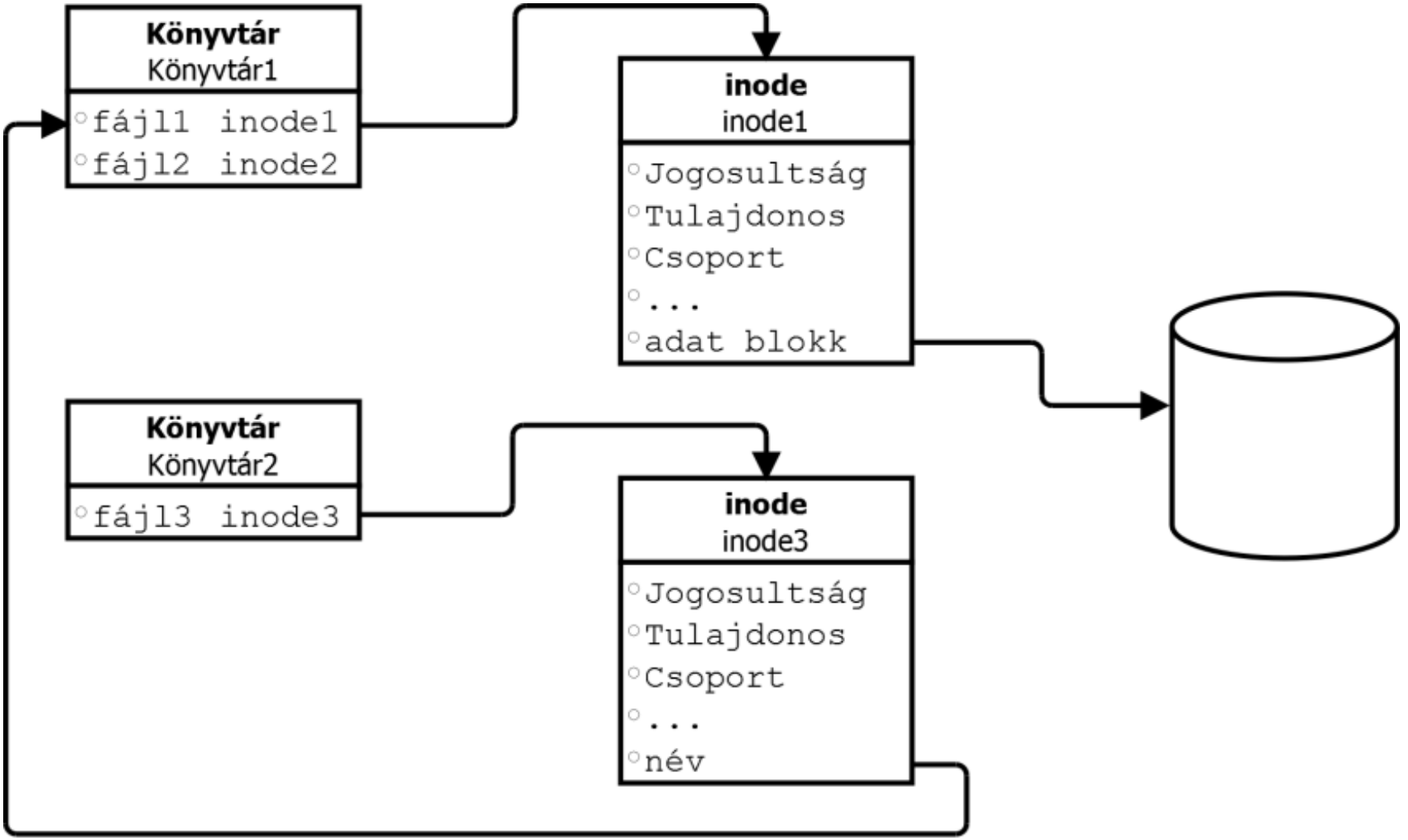
Inode (2)


Könyvtár
Könyvtár1
fájl1 inode1
fájl2 inode2
inode
inode1
Jogosultság
Tulajdonos
Csoport
...
adat blokk
inode
inode2
Jogosultság
Tulajdonos
Csoport
...
adat blokk

Merev hivatkozás (hard link)



Szimbolikus hivatkozás (symlink)



Alkalmazás fejlesztés

A Linux programozási felülete

- Fejlesztői könyvtárak
- Kernel felület – rendszerhívások
  - Felhasználói módból hívjuk, de kernel módban fut
  - Egyedi szám azonosítja
- A Linux a POSIX szabványt követi nagyrészt

Állománykezelés (alacsony szintű)

Az állományleírók

- Az állomány megnyitásakor kapjuk vissza.
- Hivatkozásként szolgál az állományra.
- Csak a folyamaton belül értelmezett.
- Kis pozitív egész szám
- Foglalt leírók:
  - 0: bemenet
  - 1: kimenet
  - 2: hibakimenet

Állományok megnyitása

```
int open(const char *pathname, int flags, mode_t mode);
int creat(const char *pathname, mode_t mode);
```

Flag	Jelentés
O_CREAT	Ha az állomány nem létezik, akkor létrehozza.
O_EXCL	Az O_CREAT opcióval együtt használatos. Az open() hibával tér vissza, ha az állomány már létezik.
O_NOCTTY	A megnyitott fájl nem lesz a processz kontrollterminálja.
O_TRUNC	Ha az állomány létezik, akkor tartalma törlődik, és a mérete 0 lesz.
O_APPEND	Az állomány végéhez fűzi íráskor az adatokat.
O_NONBLOCK	Az állományműveletek nem blokkolják a processzt.
O_SYNC	Kikapcsolja a write cache-t az állomány írásakor.
O_NOFOLLOW	Ha az állomány szimbolikus link, akkor hibával tér vissza.

Állományok megnyitása (2)

- A hozzáférés irányának megadásához bites VAGY művelettel adjuk hozzá az alábbi flagek valamelyikét.

Flag	Jelentés
O_RDONLY	Olvasásra való megnyitás.
O_WRONLY	Írásra való megnyitás.
O_RDWR	Olvasásra és írásra való megnyítás.

- creat() = open()+O_CREAT|O_WRONLY|O_TRUNC

Állományok bezárása

int close(int fd);

Hálózati állományrendszer esetén visszatérhet hibával.

Speciális leírók

Az első három leíró minden folyamat esetén automatikusan rendelkezésre áll:

- 0 (STDIN_FILENO): standard bemenet
- 1 (STDOUT_FILENO): standard kimenet
- 2 (STDERR_FILENO): hiba kimenet

Olvasás, írás

- Olvasás:

  ssize_t read(int fd, void *buf, size_t count);

- Írás:

  ssize_t write(int fd, const void *buf, size_t count);

Pozicionálás

off_t lseek(int fd, off_t offset, int whence);

Opció	Jelentés
SEEK_SET	Az állomány eleje.
SEEK_CUR	Az aktuális pozíció.
SEEK_END	Az állomány vége.

Gap-et hozhatunk létre, ha az állomány vége után pozicionálunk.

Állomány rövidítése

```
int truncate(const char *path, off_t length);
int ftruncate(int fd, off_t length);
```

Buffer szinkronizációk

- Állomány buffer kiírása
  - adat + metadata: (mint az O_SYNC flag)

    int fsync(int fd);
  - adat:

    int fdatasync(int fd);
- Merevlemez buffer
  - Minden merevlemez:

    void sync(void);
  - Csak az állományt tartalmazó:

    void syncfs(int fd);

Állományleírók duplikálása

- Duplikálás (mentés):

  int dup(int oldfd);

  - Új descriptor ugyanaz az állomány
- Duplikálás cserével:

  int dup2(int oldfd, int newfd);

  - Bezárja a *newfd*-t és az *oldfd* másolata lesz
- Duplikálás cserével + flagek:

  int dup3(int oldfd, int newfd, int flags);

  - dup2() + flags

I-node információk kiolvasása

```
int stat(const char *path, struct stat *buf);
int fstat(int fd, struct stat *buf);
int lstat(const char *path, struct stat *buf);
```

I-node információk kiolvasása

Típus	Mező	Leírás
dev_t	st_dev	Az állományt tartalmazó eszköz azonosítója.
ino_t	st_ino	Az állomány on-disk inode-száma.
mode_t	st_mode	Az állomány jogai és típusa.
nlink_t	st_nlink	A referenciák száma erre az inode-ra.
uid_t	st_uid	Az állomány tulajdonosának felhasználóazonosítója.
gid_t	st_gid	Az állomány tulajdonosának csoportazonosítója.
dev_t	st_rdev	Ha az állomány eszközleíró, akkor ez a mező tartalmazza a major és minor számot.
off_t	st_size	Az állomány mérete bájtokban.
unsigned long	st_blksize	A fájlrendszer blokkmérete.
unsigned long	st_blocks	Az állomány által allokált blokkok száma.
time_t	st_atime	Az állományhoz való legutolsó hozzáférés időpontja.
time_t	st_mtime	Az állomány legutolsó módosítási időpontja.
time_t	st_ctime	A legutolsó változtatás időpontja.

Jogok lekérdezése

int access(const char *pathname, int mode);

Érték	Jelentés
F_OK	Az állomány létezik-e, elérhető-e?
R_OK	A processz olvashatja-e az állományt?
W_OK	A processz írhatja-e az állományt?
X_OK	A processz futtathatja-e az állományt? (Könyvtár esetén keresési jog.)

Jogok állítása

```
int chmod(const char *path, mode_t mode);
int fchmod(int fd, mode_t mode);
```

Jogok állítása

Érték névvel	Érték számmal
S_ISUID	04000
S_ISGID	02000
S_ISVTX	01000
S_IRUSR (S_IREAD)	00400
S_IWUSR (S_IWRITE)	00200
S_IXUSR (S_IEXEC)	00100
S_IRGRP	00040
S_IWGRP	00020
S_IXGRP	00010
S_IROTH	00004
S_IWOTH	00002
S_IXOTH	00001

Tulajdonos és csoport beállítása

```
int chown(const char *path, uid_t owner, gid_t group);
```

```
int fchown(int fd, uid_t owner, gid_t group);
```

```
int lchown(const char *path, uid_t owner, gid_t group);
```

- A tulajdonost csak root állíthatja.
- Állításkor a setuid és setgid bitek törlődnek.

Időbélyeg beállítása

```
#include <utime.h>

int utime(const char *filename, const struct
utimbuf *times);


#include <sys/time.h>

int utimes(const char *filename, const struct
timeval times[2]);
```

Eszközállomány létrehozása

int mknod(const char *pathname, mode_t mode, dev_t dev);

Érték	Jelentés
S_IFREG	Normál állomány
S_IFCHR	Karakter típusú eszköz állomány
S_IFBLK	Blokk típusú eszköz állomány
S_IFIFO	named pipe

Linkek létrehozása/olvasása

- Hard link:

  int link(const char *oldpath, const char *newpath);

- Szimbolikus link:

  int symlink(const char *oldpath, const char *newpath);

- Szimbolikus link olvasása:

  ssize_t readlink(const char *path, char *buf, size_t bufsiz);

Állomány átnevezése/törlése

- Állományok átnevezése:

  int rename(const char *oldpath, const char *newpath);

  - Partíciók között nem mozgatja az állományt.
  - Atomi műveletnek látszik.
- Állományok törlése:

  int unlink(const char *pathname);

Könyvtárműveletek

Munkakönyvtár

- Aktuális munkakönyvtár:

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

- Könyvtár váltás:

  int chdir(const char *path);

  int fchdir(int fd);

A gyökér könyvtár lecserélése

- Védelmi okokból egy alkönyvtárra korlátozzuk a folyamatot. (Rendszergazda jogosultság szükséges.)

```
int chroot(const char *path);
```

- Munkakönyvtárat is át kell állítanunk utána:

```
chdir("/");
```

További könyvtárműveletek

- Létrehozás

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

- Törlés (üresnek kell lennie)

  int rmdir(const char *pathname);

Könyvtár listázása

- Header: dirent.h
- Megnyitás:

  DIR *opendir(const char *name);

  DIR *fdopendir(int fd);
- Lezárás:

  int closedir(DIR *dirp);

Könyvtár listázása

- Olvasás:

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

- A soron következő elemet adja vissza:

```
struct dirent {
  ino_t          d_ino;
  off_t          d_off;
  unsigned short d_reclen;
  unsigned char  d_type;
  char           d_name[256];
};
```

- A végén NULL értékkel tér vissza.

Csővezetékek

Csővezetékek létrehozása

- Névtelen csővezeték:

  int pipe(int pipefd[2]);

- Megnevezett csővezeték:
  - mknod parancs