PRAKTIKA: Sistema deiak

Helburuak:

-Fitxategiak kudeatzen dituzten sistema deiak ezagutzea.

-Liburutegien funtzio tipikoak erabili beharrean, linuxeko APIak erabiltzen dituzten programak sortu.

Erreferentziak:
http://blog.txipinet.com/index.php/Informatica/
http://www.e-ghost.deusto.es/docs/2006/ProgramacionGNULinux.odt

#### Fitxategien Sistema-Deiak

Sistema Deiak ulertzeko modurik errezena oinarrizko funtzioak erabiltzea da, hau da, fitxategiak maneiatzen dituztenak daude oinarrizkoen artean. Esanda dago, UNIX-en eta GNU/LINUX-en konkretuki dena fitxategi bat dela eta honez gero syscall-ak UNIX-en sistema programazioaren oinarria dira.

Has gaitezen fitxategi bat sortzen. Fitxategi bat irekitzeko bi aukera ezberdin ditugu, open() eta creat(). Aspaldi, open() erabiliz, bakarrik jadanik sortuta zeuden fitxategiak ireki ahal genituen. Beharrezkoa genuen creat() dei bat egitea open() deitu aurretik. Gaur egun, open() erabiliz fitxategi bat sortu dezakegu, bere erazagupenari, parametro berri bat erantsi zaiolako:

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

Ikusten denez, open() berria aspaldiko open()-aren eta creat()-aren funtzionaltasunen batura bat da. Beste berezitasunen artean azpimarratu dezakegu “mode” parametroa orain “mode_t” dela. Mota hau UNIX-en askotan erabiltzen da eta normalean “int” edota “unsigned int” bati dagokio. Hala ere, aurreko bertsioetako bateragarritasunagatik mota hori mantentzen da. Horregatik, syscall hauek erabiltzeko, hurrengo goiburu-fitxategiak gehitu behar ditugu:

```
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
```

open() funtzioaren parametroak:

- pathname: path + fitxategiaren izena, kate baten helbide bezala edo kakotzen artean.
- “flags” fitxategia zertarako eta nola zabalduko dan adierazteko. Hurrengo taulan hartu dezaketen balioak daude:

| Adierazlea | Edukia | Deskribapena |
|---|---|---|
| O_RDONLY | 0000 | Fitxategia soilik irakurtzeko irekitzen da. |
| O_WRONLY | 0001 | Fitxategia soilik idazteko irekitzen da. |
| O_RDWR | 0002 | Fitxategia idazteko eta irakurtzeko irekitzen da. |
| O_SEQUENTIAL | 0020 | Fitxategia sekuentzialki atzituta izateko (zintak kudeatzeko erabiliak). |
| O_TEMPORARY | 0040 | Fitxategiak denbora izaera du. |
| O_CREAT | 0100 | Fitxategia sortu aurkitu ezean. |
| O_EXCL | 0200 | O_CREAT aukerarekin batera zehazten bada, existitu ezkero hutsitzen du. |
| O_NOCTTY | 0400 | Fitxategia terminal-gailu (TTY) bat baldin bada, ez da prozesu kontrolen terminalean bihurtuko (CTTY). |
| O_TRUNC | 1000 | Fitxategia hutsitzen du. |
| O_APPEND | 2000 | Idazkera erakuslea fitxategiaren bukaeran kokatzen du, eta datu berriak bukaeran idatziko dira. |
| O_NONBLOCK | 4000 | Fitxategiaren irekiera ez-blokeatzailea izango da. O_NDELAY –aren baliokidea da. |
| O_SYNC | 10000 | Fitxategian egiten diren idazketak sinkronoak dira. |
| O_ASYNC | 20000 | Fitxategian egiten diren idazketak asinkronoak dira. |
| O_DIRECT | 40000 | Disko-sarbidea zuzenki egingo da, buffer barik. |
| O_LARGEFILE | 100000 | Soilik oso handiak diren fitxategientzako. |
| O_DIRECTORY | 200000 | Katalogo bat denean. |
| O_NOFOLLOW | 400000 | Lotura sinbolikoak ez jarraitzeko aukera. |

### 1.5.1 Taula “Flag”-en balioak

Lerroa nahiko luzea da eta baloreak bat baino gehiago kateatzeko aukera dago. Hau da, balore ezberdinen artean OR logiko bat egitea, nahi dugun efektua lortuz. Horrela, creat() -aren parekoa izango den hurrengo open()-a lor dezakegu:

```
open( pathname, O_CREAT | O_TRUNC | O_WRONLY, mode )
```

“mode” argumentua fitxategien sistemaren barnean baimenak zehazteaz arduratzen da (chmod komandoarekin egiten genuen bezala). Balore posibleen taula honako hau da:

| Adierazlea | Edukia | Deskripzioa |
|---|---|---|
| S_IROTH | 0000 | Gainerako Erabiltzaileek irakurtzeko aukera |
| S_IWOTH | 0001 | Gainerako Erabiltzaileek idazteko aukera |
| S_IXOTH | 0002 | Gainerako Erabiltzaileek exekutatzeko aukera |
| S_IRGRP | 0010 | Taldeko Erabiltzaileek irakurtzeko aukera |
| S_IWGRP | 0020 | Taldeko Erabiltzaileek idazteko aukera |
| S_IXGRP | 0040 | Taldeko Erabiltzaileek exekutatzeko aukera |
| S_IRUSR | 0100 | Jabeak irakurtzeko aukera |
| S_IWUSR | 0200 | Jabeak idazteko aukera |
| S_IXUSR | 0400 | Jabeak exekutatzeko aukera |
| S_ISVTX | 1000 | Fitxategian “sticky bit”-a gehitu |

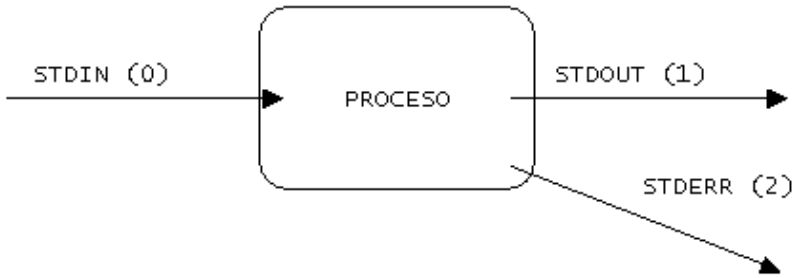
| | | |
|---|---|---|
| S_ISGID | 2000 | Fitxategian SUID bita gehitu |
| S_ISUID | 4000 | Fitxategian SGID bita gehitu |
| S_IRWXU | S_IRUSR + S_IWUSR + S_IXUSR | Jabeak irakurtzeko,idazteko eta exekutatzeko aukera |
| S_IRWXG | S_IRGRP + S_IWGRP + S_IXGRP | Taldeko Erabiltzaileek irakurtzeko,idazteko eta exekutatzeko aukera |
| S_IRWXO | S_IROTH + S_IWOTH + S_IXOTH | Gainerako Erabiltzaileek irakurtzeko,idazteko eta exekutatzeko aukera |

Taula1 "Mode" argumentuaren balore posibleen lista

Balore guzti hauek goiburu-fitxategi batean zehazten dira eta horregatik barneratzea komeni da:

```
#include <sys/stat.h>
```

open() fitxategi bat ondo ireki ezkero, irekitako fitxategia maneiatzeko fitxategiaren deskribatzailearen zenbakia itzultzen du. Prozesu bakoitzak, fitxategiak errez maneiatzeko, fitxategien deskribatzaileen taula bat erabiltzen du. Hasieran, taula horren 0,1 eta 2 sarrerak STDIN, STDOUT eta STDERR fitxategiek betetzen dituzte, hau da, sarrera estandarra, irteera estandarra eta errore estandarra.




1.Irudia Prozesu baten hasierako fitxategien deskribatzaileak.

Fitxategien deskribatzaileen taula hau, hotel baten moduan ulertu dezakegu, non hasieran lehenengo hiru logelak STDIN, STDOUT eta STDERR bezeroek okupatzen dituzten. Bezero gehiago etorri ezkero (fitxategi gehiago ireki ezkero), bezero hauek hurrengo logeletan sartuko dira. Horrela, prozesuaren hasieran irekitako fitxategi batek, normalean, 2-ra hurbiltzen den fitxategiaren deskribatzaile bat izango du. "Hotel" honetan, logela baxuena beti bezero berrienarentzat izango da. Hau guztia kontuan izan beharko dugu etorkizunean.
Ongi, orain existitzen ez direnean fitxategiak irekitzen eta sortzen badakigu, baina ezin ditugu fitxategiak irekita utzi, hau da, ondo itxi gabe. Badakizue C-k bere programatzaileak pertsona arduratsu moduan hartzen dituela. Fitxategi bat ixteko nahikoa da syscall close()-ari fitxategiaren deskribatzailea argumentu moduan pasatzea:

```
int close( int fd)
```

Ariketa 1:

- Hurrengo adibidearekin ariketa1.c osotu.
  subl ariketa1.c
- Konpilatu warning guztiak ezabatu arte.

Nahiko erreza da. Ikus dezagun adibide batekin nola funtzionatzen duen hau guztia:

```
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <unistd.h>
#include <stdio.h>
#include <stdlib.h>

int main( int argc, char *argv[] )
{
  int fd;
if( (fd = open( argv[1], O_RDWR )) == -1 )
  {
    perror( "open" );
    exit( -1 );
}
printf( "Irekita dagoen fitxategia %d deskribatzailea du\n", fd );
close( fd );
return 0;
}
```

Hasieran beharrezko goiburu-fitxategiak ditugu, honaino azaldu dugun bezala. Segidan, fitxategiaren deskribatzailea barnean edukiko duen “fd” aldagaia deklaratzen dugu, eta open()-i dei egiten diogu, dei honen emaitza “fd”-n gordetzen dugularik. “fd” -1 izan ezkero, fitxategia irekitzean errore bat eman duela esango dugu eta programatik irtengo gara. Ordea, beste edozein kasutan, programaren gauzatzearekin jarraituko dugu, fitxategiaren deskribatzailea pantailatik erakutsiz eta ondoren fitxategia itxiz. Programa honen funtzionamendua hemen ikus dezakegu:

```
txipi@neon:~$ gcc ariketa1.c -o  ariketa1
txipi@neon:~$ ./ ariketa1 ariketa1.c
Irekita dagoen fitxategia 3 deskribatzailea du.
```

Hurrengo pauso logikoa, maneiatzen ditugun fitxategietan irakurri eta idatzi, ahal izatea da. Horretarako, antzeko bi “syscall” erabiliko ditugu: read() eta write(). Erazagupenak:

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

Lehenengoak “fd”-n zehaztutako fitxategiaren deskribatzailetik “count” bytak irakurtzen saiatzen da, “buf” buffer-an gordetzeko. Saiatzen dela esaten dugu, read()-ek irakurritako byte kopurua itzultzen duelako eta balore hau “count” aldagaiarekin konparatuz, eskatzen genuen byte kopurua irakurri duen edo ez jakin dezakegu. Irakurritako byte-ak zenbatzeko erabilitako datu motak pixka bat arraroak izan daitezke baina GNU/Linux bertsio honetan soilik zenbaki osoak dira, goiburu fitxategian ikus daiteken bezala.

2. Ariketa:

ssize_t eta size_t, ze motatakoa diren jakiteko, aurreprosezadorea erabili. Hau da, .c .i batean bihurtu eta aurkitu ssize_t eta size_t ze motatakoak diren.

```
Gcc -o ariketa1.i -E ariketa1.c
gedit ariketa1.i
ssize_t long int bat da
```

size_t long unsigned int bat da

write() funtzioaren erabilera oso antzekoa da. “buf” bufferra, idatzi nahi dugunarekin betetzea nahikoa izango da. “count”-en bere tamaina zehaztuko dugu eta “fd”-n bere fitxategiaren deskribatzailearekin idatziko dugun fitxategia zehaztu:

```
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>

#define STDOUT 1
#define SIZE 512
```

3. Ariketa

Hurrengo programak parametroak behar ditu. Parametro barik deituz gero, erabiltzeko zenbat eta zein pararametroren beharra dagoan laguntzen duen mezua gehitu, ariketa3.c deitu programari.

```
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
int main( int argc, char *argv[] )
{
        int fd, readbytes;
        char buffer[SIZE];
        if (argc != 2)
           {printf (" erabili hurrengo deia: %s fitxategiaren izena", argv[0]);
             exit (-1);
           }
        if( (fd = open( argv[1], O_RDWR )) == -1 )
        {
                perror( "open" );
                exit( -1 );
        }
        while( (readbytes = read( fd, buffer, SIZE )) != 0 )
        {
/*              write( STDOUT, buffer, SIZE ); */
                write( STDOUT, buffer, readbytes );
        }

        close( fd );

        return 0;
}
```

Ikus daitekenez, hasieran bi konstante zehaztuko ditugu, STDOUT, irteera estandarra zehazten duen fitxategiaren deskribatzailea, 1 dena eta SIZE, erabiliko dugun buffer-aren tamaina adierazten duena. Segidan, beharrezkoak diren aldagaiak erazagutzen ditugu eta parametrotik pasatutako fitxategia irekitzen saiatzen gara (argv[1]), irakurtzeko eta idazteko sarbidearekin. Errore bat gertatzen bada open() funtzioak -1 itzultzen du, eta errore mezua aterako du eta bestela aurrera egingo dugu. Honen ondoren, amaitu (read()-ek irakurritako 0 byte bueltatzen dituenean) arte SIZE-naka bytak irekitako fitxategitik (“fd”) irakurriko dituen kiribil bat daukagu. Kiribilaren buelta bakoitzean iraXkurritakoa STDOUT irteera estandarretik idatziko da. Azkenean fitxategiaren deskribatzailea close() batekin itxiko dugu.

Azken finean, programa honek egiten duen gauza bakarra fitxategi baten edukia irteera estandarretik erakustea da, hau da, “cat” komandoak gehienetan egiten duena.

Aurreko programan komentatutako lerro bat dago:

```
/*                          write( STDOUT, buffer, SIZE ); */
```

Write() dei honetan, ez daukagu read() deiak itzuli duena kontuan izaten, SIZE byteak idazten saiatzen baizik, hau da, 512 byte bakoitzeko. Zer gertatuko zan, lerro hau erabiliko bazan, bestearen truke? Parametro moduan pasatzen dugun fitxategia nahiko handia bada, while kiribilaren lehenengo zikloak ondo ibiliko dira, read()-ek 512 karaktere bueltatuko duelako irakurritako byte kopuru moduan, eta write()-k behar bezala idatziko dituelako. Baina kiribilaren azken iterazioan, read()-ek 512 baino byte gutxiago irakurriko dugu, parametrotik pasatutako fitxategiaren tamaina 512 byten multiploa izatea oso arraroa izango zelako. Orduan, read()-ek 512 byte baino gutxiago irakurriko ditu eta write()-k 512 idatziko ditu. Horren ondorioz, write()-k  karaktere zaborrak idatziko ditu.

## 4. Ariketa: Kopiatu hurrengo ariketa “ariketa4.c” fitxategian eta konpilatu.

```
txipi@neon:~ $ gcc files.c -o files
txipi@neon:~ $ ./files files.c
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>

#define STDOUT 1
#define SIZE 512

int main(int argc, char *argv[])
{
      int fd, readbytes;
      char buffer[SIZE];

      if( (fd=open(argv[1], O_RDWR)) == -1 )
      {
            perror("open");
            exit(-1);
      }

      while( (readbytes=read(fd, buffer, SIZE)) != 0 )
      {
/*          write(STDOUT, buffer, readbytes); */
            write(STDOUT, buffer, SIZE);
      }

      close(fd);

      return 0;
}
@p@N¨@'@Àýÿ¿4ýÿ¿ô¢%@'@Ì8ýÿ¿D&@
                        "&@'@Xýÿ¿Ù'@Xýÿ¿@txipi@neon:~ $
```

Adibide honek erakusten duen moduan, hasieran programa ondo doa, baina read()-ak irakurritako bytak kontuan hartzen ez baditugu, azkenean karaktere zaborrak idazten amaituko dugu.

Lagundu ahal gaituen beste funtzio bat lseek()-a da. Askotan ez gara idazteko edota irakurtzeko fitxategi baten hasieran kokatu nahi, baizik eta fitxategiaren hasierari edota amaierari erlatiboa den

desplazamendu konkretu batean kokatzea da interesatzen zaiguna. “lseek()” funtzioa posibilitate hori ematen digu eta honako prototipoa dauka:

```
off_t lseek(int fildes, off_t offset, int whence);
```

5. Ariketa:
lseek funtzioak off_t itzultzen du, off_t ze motatakoa den jakiteko aurreprozesadorea erabili.

```
Subl files.c
gcc -o files.i -E files.c
gedit files.i
typedef long int __off_t;
```

Jasotzen dituen parametroak oso ezagunak dira. “fildes” fitxategiaren deskribatzailea da, “offset” kokatu nahi garen desplazamendua da, “whence”-k kokapena adierazten du eta hurrengo balioak hartu ditzake:

| Erakuzlea | Edukina | Deskribapena |
|---|---|---|
| SEEK_SET | 0 | Erakuslea fitxategiaren hasieratik “offset” bytetara kokatzen du. |
| SEEK_CUR | 1 | Erakuslea momentuan erakusleak daukan posiziotik “offset” bytetara kokatzen du. |
| SEEK_END | 2 | Erakuslea fitxategiaren amaieratik “offset” bytetara kokatzen du. |

Taula 2 “Whence” argumentuak jaso ditzazkeen baloreen zerrenda

Adibidez, fitxategi bat irakurri nahi badugu eta 200 karaktereko goiburu bat pasatu nahi badugu, horrela egin genezake:

```
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>

#define STDOUT 1
#define SIZE 512

int main( int argc, char *argv[] )
{
   int fd, readbytes;
   char buffer[SIZE];

   if( (fd = open( argv[1], O_RDWR )) == -1 )
   {
      perror( "open" );
      exit( -1 );
   }

   lseek( fd,200, SEEK_SET );

   while( (readbytes = read( fd, buffer, SIZE )) != 0 )
   {
      write( STDOUT, buffer, SIZE );
   }
```

```
    close(fd);

    return 0;
}
```

6. Ariketa:
Aurreko kodea ariketa6.c bezala gorde eta write kodea aldatu zaborrik ez ateratzeko.

```
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>

#define STDOUT 1
#define SIZE 512

int main( int argc, char *argv[] )
{
    int fd, readbytes;
    char buffer[SIZE];

    if( (fd = open( argv[1], O_RDWR )) == -1 )
    {
        perror( "open" );
        exit( -1 );
    }

    lseek( fd,200, SEEK_SET );

    while( (readbytes = read( fd, buffer, SIZE )) != 0 )
    {
        write( STDOUT, buffer, readbytes);
    }

    close(fd);

    return 0;
}
```

Sortze, irekitzen, ixten, irakurtzen eta idazten badakigu eta honekin, denetarik egin dezakegu! Fitxategiak maneiatzeko funtzioekin amaitzeko, chmod(), chown() eta stat() ikusiko ditugu, fitxategiaren modua eta jabea aldatzeko edota beren ezaugarriak ikusi ahal izateko.

chmod() funtzioa izen bereko komanduaren erabilera dauka: fitxategi konkretu bateko sarbide moduak aldatu. Nahiz eta C erabili, gure programa fitxategi-sistemaren murrizketei helduta dago eta soilik beren jabea edo root aldatu ditzakete fitxategi konkretu baten sarbide motak. Fitxategi bat sortzean, bai creat() erabiliz nahiz open() erabiliz, fitxategi honek, konfiguratuta dagoen modu-maskararen arabera (ikus dezagun “man umask”) dagoen modu bat dauka. Hala ere, honako funtzioak erabiliz beren moduak zuzenean alda ditzakegu:

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

```
int fchmod(int fildes, mode_t mode);
```

Funtzio bakoitzaren prototipoa ikusiz, bere funtzionamendua aztertu dezakegu: lehendabizikoa, chmod(), “path” katean adierazitako fitxategiaren modua aldatzen du. Bigarrena, fchmod(), fitxategiaren ibilbidea daukan karaktere-katea jaso beharrean, fitxategiaren deskribatzaile bat jasotzen du, “fildes”. “mode” parametroa “mode_t” motatakoa da, baina GNU/Linux-en osoa den aldagai bat erabiltzearen baliokidea da. Bere balioa “chmod” komandoa deitzean erabiliko dugunaren bera da, adibidez:

```
chmod( “/home/txipi/prueba”, 0666 );
```

Fitxategiaren jabea aldatzeko hurrengo funtzioak erabiliko ditugu:

```
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);
```

Haiekin, bere ibilbidearen arabera (chown() y lchown()) eta fitxategiaren deskribatzailearen arabera (fchown()), fitxategi baten jabea eta taldea alda ditzakegu. Jabea (“owner”) eta taldea (“group”) erabiltzaileak eta taldeak identifikatzen dituzten osoak dira, hala nola “/etc/passwd” eta “/etc/group” fitxategiak azaltzen duten. Parametro haietako baten bat -1 balioarekin finkatzen badugu (“owner” edo “group”), zegoen bezala utzi nahi dugula ulertuko du. lchown() funtzioa chown() bezalakoa da, fitxategiekiko lotura sinbolikoetan ezik. Linux 2.1.81 bertsioaren aurrekoetan (2.1.46 bertsioa izan ezik), chown() ez zituen lotura sinbolikoak jarraitzen. Linux 2.1.81-etik aurrera hasi zen chown() lotura sinbolikoak jarraitzen eta syscall berri bat sortu zen, lchown(), lotura sinbolikoak jarraitzen ez zituena. Honez gero, gure programen segurtasuna hobetu nahi badugu chown() erabiliko dugu, lotura sinboliko nahasgarriekin gaizki-ulertuak saihesteko.

7. Ariketa:
Azaldu adibide batekin zer den SUID eta SGID bita.
Laguntza: http://es.wikipedia.org/wiki/Setuid

SUID-ek erabiltzaileari baimenak emango dizkio eta SGID-ek berriz erabiltzaile horren taldeei.

Edozein erabiltzaile arrunt batek fitxategiaren jabea aldatu ezkero, fitxategi horrek bit SUID edo SGID izan ezkero desaktibatu egingo dira segurtasunagatik. POSIX estandarrak, root-ek akzio bera egin ezkero gertatuko dena ez du garbi ezartzen, orduan linux-aren arabera bit SUID eta bit SGID aktibatuta edo desaktibatuta geratzea gerta daiteke. Erabilpenaren adibide bat honako hau izango litzateke:

```
gid_t grupo = 100; /* 100 es el GID del grupo users */
chown( “/home/txipi/prueba”, -1, 4);
```

ikusteko taldeak:
/etc/group
Dei honekin “/home/txipi/prueba” fitxategiaren jabea eta taldea aldatu nahi ditugula adierazten dugu, jabea zegoen moduan (-1) utziz eta taldea 100 balorearekin aldatuz, “users” taldeari dagokiona:

```
txipi@neon:~$ grep 100 /etc/group
users:x:100:
```

Geratzen zaigun gauza bakarra, fitxategi baten ezaugarrietara sartzen jakitea da, stat() funtzioa

erabiliz. Funtzio honen portaera, orain arte ikusitako funtzioek daukatenaren ezberdina da. Fitxategi baten i-nodo informazio guztia (hau da, ezaugarri guztiak) estruktura batean sartzen du. Eta stat() funtzioari estruktura baten erreferentzia pasatuko diogu. Stat syscall funtzioari deitu ostean , estruktura horretan fitxategiaren ezaugarri guztiak izango ditugu, behar bezala beteak. Honekin erlazionatutako funtzioak honakoak dira:

```
int stat(const char *file_name, struct stat *buf);
int fstat(int filedes, struct stat *buf);
int lstat(const char *file_name, struct stat *buf);
```

Hau da, chown(), fchown() eta lchown()-en antzekoak, baina fitxategiaren jabeak zehaztu ordez, bigarren parametro bezala “stat” moduko estruktura batekiko erakusle bat behar dute:

```
struct stat {
  dev_t        st_dev;      /* dispositivo */
  ino_t        st_ino;      /* numero de inode */
  mode_t       st_mode;     /* modo del fichero */
  nlink_t      st_nlink;    /* numero de hard links */
  uid_t        st_uid;      /* UID del propietario*/
  gid_t        st_gid;      /* GID del propietario */
  dev_t        st_rdev;     /* tipo del dispositivo */
  off_t        st_size;     /* tamaño total en bytes */
  blksize_t    st_blksize;  /* tamaño de bloque preferido */
  blkcnt_t     st_blocks;   /* numero de bloques asignados */
  time_t       st_atime;    /* ultima hora de acceso */
  time_t       st_mtime;    /* ultima hora de modificación */
  time_t       st_ctime; /* ultima hora de cambio en inodo */
};
```

8. Ariketa:

Linux-en dena da fitxategi bat, eta edozein fitxategi i-nodo batekin identifikatzen da.“ls -i “ komandoarekin jakin dezakegu fitxategiaren i-nodoa zein den. Sortu bi fitxategi “p1” eta “p2” “/tmp” katalogoan eta dagozkion inodoak pantailaratzen duen programa egin.

```
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <stdlib.h>
#include <stdio.h>

int main( int argc, char *argv[] )
{
  struct stat estructura;
  if( ( lstat( argv[1], &estructura ) ) < 0 )
  {
    perror( "lstat" );
    exit( -1 );
  }

  printf( "i-nodo: %lu\n", estructura.st_ino );
  return 0;
}
```

Ikusten dugunez, fitxategiaren oso informazio garrantzitsua ikus dezakegu. Hurrengo adibideak hau

guztia nola funtzionatzen duen erakusten du:

```
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <stdlib.h>
#include <stdio.h>

int main( int argc, char *argv[] )
{
  struct stat estructura;
  if( ( lstat( argv[1], &estructura ) ) < 0 )
  {
    perror( "lstat" );
    exit( -1 );
  }
  printf( "Fitxategiaren propietateak” <%s>\n", argv[1] );
  printf( "i-nodo: %lu\n", estructura.st_ino );
  printf( "modua: %#o\n", estructura.st_mode );
  printf( "loturak: %ld\n", estructura.st_nlink );
  printf( "jabea: %d\n", estructura.st_uid );
  printf( "taldea: %d\n", estructura.st_gid );
  printf( "dispositibo mota: %ld\n", estructura.st_rdev );
  printf( "tamaina totala, byte-tan: %ld\n", estructura.st_size );
  printf( "blokearen gustuko tamaina: %ld\n", estructura.st_blksize );
  printf( "esleitutako bloke kopurua: %ld\n", estructura.st_blocks );
  return 0;
}
```

Aurreko kodean esanguratsuak diren zenbait xehetasun daude: “%#o” erabiltzen dugu, fitxategiaren sarbide modua era zortzitar eran erakusteko. Hala ere, ezagutzen ditugun lau zenbaki baino gehiago agertzen dira. Hau, eremu horretan fitxategiaren motari buruz ere informazioa agertzen zaigulako da (fitxategi bat, gailu bat, sekuentziala, FIFO bat, etab. dena)

Aurreko kodearen egikaritzearen emaitza bat honakoa izan daiteke:

```
txipi@neon:~$ gcc stat.c -o stat
txipi@neon:~$ ./stat stat.c
Fitxategiaren propietateak” <./prueba.txt>
i-nodo: 3149211
modua: 0100600
loturak: 1
jabea: 1000
taldea: 1000
dispositibo mota: 0
tamaina totala, byte-tan: 9
blokearen gustuko tamaina: 4096
esleitutako bloke kopurua: 8
```