3. Praktika

UNIX: Fitxategi-sistema eta Sarrera/Irteera: Sistema-deiak

Kontzeptuak

Fitxategi-sistema eta Sarrera/Irteerako sistema-deiak eta bere parametroak. Utilitateen/tresnen programazioa sistema-deiak erabiliz. Sistema-deien eta liburutegiko funtzioen arteko desberdintasunak.

Gaitasun espezifikoa

Sistema Eragileen Sarrera/Irteerako funtzioak erabiltzea utilitateak/tresnak programatzeko.

Urratsak

1 Sistema-deien aurkezpena.

2 Klaseko adibideak probatu.

3 Proposatutako ariketak egin.

Dokumentazioa

- Klaseko gardenkiak eta adibideak.
- Ariketen enuntziatua eta dokumentazioa.
- UNIX-eko laguntza: man orriak (***man*** komandoa)

UNIX-eko sistema-deiak (Sarrera/Irteera)

Irekiera / itxiera / irakurketa / idazketa / kokapena

```
int open (char *path, int flags, int baim);
int creat (char *path, int baim);
int close (int fd);
int read (int fd, char *buf, unsigned luz);
int write (int fd, char *buf, unsigned luz);
long lseek (int fd, long despl, int pos_kod);
/* hasiera: 0, unekoa: 1, bukaera: 2 */
```

O_RDONLY	Irakurketa soilik
O_WRONLY	Idazketa soilik
O_RDWR	Irakurketa eta idazketa
O_NDELAY	Sinkronizazioaren kontrola
O_CREAT	Sortu existitzen ez bada
O_TRUNC	Ezabatu edukia fitxategia existitzen bada
O_EXCL	Errorea itzuli existitzen bada (O_CREAT-ekin konbinatzen da)
O_APPEND	Eransketa moduan
Adierazle hauek konbina daitezke **|**, **&** eta ~ eragile logikoak erabiliz	
open deian O_WRONLY|O_CREAT|O_TRUNC balioak *creat* deiaren baliokide bihurtzen du	
O_EXCL fitxategia elkarrekiko esklusiorako erabiltzea baimentzen du	

open deiak dituen aukerak (flags).

Katalogoaren kontrola

```
int chdir (char *path);
int mkdir (char *path, int baim);
int rmdir (char *path);
int link (char *iturburu_bidea, char *helburu_bidea);
int symlink (char *iturburu_bidea, char *helburu_bidea);
int unlink (char *path);
```

Fitxategi eta dispositiboen kontrola

```
int stat (char *path, struct stat *sbuf);
int fstat (int fd, struct stat *sbuf);
int fcntl (int fd, int komandoa, int argum);
int ioctl (int fd, int komandoa, struct termio *buf);
```

st_dev:	fitxategia gordetzen duen unitatearen zenbakia (short)
st_ino:	inodo-zenbakia (ushort)
st_mode:	modua (short)
st_nlink:	lotura-kopurua (short)
st_uid:	jabearen identifikatzailea (ushort)
st_gid:	taldearen identifikatzailea (ushort)
st_rdev:	fitxategi berezientzako unitate-zenbakia (short)
st_size:	tamaina edo 0 fitxategi berezietan (long)
st_atime:	azken atzipenaren ordua (long)
st_mtime:	azken aldaketaren ordua (long)
st_ctme:	sortu zeneko ordua (long)

struct stat egituraren edukia.

POSIX makroak fitxategiaren mota konprobatzeko. TRUE bueltatzen dute baldin:

S_ISLNK(m)	softwarezko lotura bada
S_ISREG(m)	fitxategi arrunta bada
S_ISDIR(m)	katalogoa bada
S_ISCHR(m)	karaktere motako gailua bada
S_ISBLK(m)	bloke motako gailua bada
S_ISFIFO(m)	izendun buzoia (fifo-a) bada
S_ISSOCK(m)	socket-a bada

m struct stat aldagai baten ***st_mode*** eremua da

Fitxategien mota jakiteko makroak.

st_mode eremua aztertzeko honako konstanteak ere erabili daitezke:

S_IFLNK	softwarezko lotura
S_IFREG	fitxategi arrunta
S_IFDIR	katalogoa
S_IRWXU	jabearen irakurketa, idazketa eta exekuzio baimena
S_IRUSR	jabearen irakurketa baimena
S_IWUSR	jabearen idazketa baimena
S_IXUSR	jabearen exekuzio baimena

Erabilpen adibidea:

```
stat(fitx_izena, &st);
if ((st.st_mode & S_IRUSR) == S_IRUSR))
   ...
```

Fitxategien mota jakiteko konstanteak.

TCGETA	Informazioa lortu *termio* egituran
TCSETA	Informazioa finkatu *termio* egituran oinarrituz

struct termio egiturari dagozkion komandoak.

Erabiltzaile anitz

```
int chmod (char *path, int modua);
int chown (char *path, int jabea, int taldea);
int access (char *path, int modua);   /* R_OK, W_OK, X_OK */
int umask (int modua);
```

C liburutegiko funtzioak (Sarrera/Irteera)

Katalogoekin aritzeko

```
DIR *opendir (char *path);
int closedir (DIR *dir);
struct dirent *readdir (DIR *dir);  /* izena: emaitza->d_name */
```

```
struct dirent {
    long d_ino;                    // Inode zenbakia
    off_t d_off;
    unsigned short int d_reclen;   // d_name eremuan luzera
    char d_name[NAME_LEN];         // Sarreraren izena (0 bukaeran)
};
```

struct dirent egitura**.**

Irekiera / itxiera / irakurketa / idazketa

```
FILE *fopen (char *path, char *mota);
int fclose (FILE *fd);
int fread (void *buf, int tam, int osa_kop, FILE *fd);
int fwrite (void *buf, int tam, int osa_kop, FILE *fd);
```

Karaktere kateekin lan egiteko C liburutegiko funtzioak

- strcpy (str1, str2) → str2 str1-en kopiatzen da
  `char *strcpy (char *helburu, const char *jatorri);`
- strcmp (str1, str2) → bi karaktere kateak konparatzeko
  `int strcmp (const char *str1, const char *str2);`
- strcat (str1, str2) → kateamendua: str1 := str1 + str2
  `char *strcat (char *str1, const char *str2);`
- strlen (str) → karaktere katearen luzera itzuli
  `int strlen (const char *str);`
- atoi (str) → string motatik integer motara (Ascii TO Integer)
  `int atoi (const char *str);`
- strstr (str1, str2) → str2 str1-en azpikatea den edo ez esan
  `char *strstr (const char *str1, const char *str2);`

kopiatu adibide-programa

Fitxategi bat kopiatzen du sarrera estandarretik irteera estandarrera, *read* eta *write* deiak erabiliz.

```
#include <stdio.h>
#include <stdlib.h>
#include <errno.h>

#define errore(a) {perror(a); exit(1);};
#define BUFSIZE 512

main(int argc, char *argv[])   /* kopiatu.c */
{
   int n;
   char buf[BUFSIZE];

   /* irakurketa eta idazketa zikloa */
   while ((n = read(0, buf, BUFSIZE)) > 0)
      if (write(1, buf, n) != n) errore("write");

   if (n == -1) errore("read");
}
```

buztana adibide-programa

Programa honek parametro gisa pasatzen zaion fitxategiaren azken 10 karaktereak idazten ditu irteera estandarrean.

```
#include <stdio.h>
#include <stdlib.h>
#include <errno.h>
#include <fcntl.h>

#define errore(a) {perror(a); exit(1);};

main(int argc, char *argv[])   /* buztana.c */
{
   int fd, k;
   char buf[80];

   if (argc != 2) errore("argumentuak gaizki");

   if ((fd = open(argv[1], O_RDONLY, 0666)) == -1) errore("open");

   if (lseek(fd, -10, 2) == -1) errore("lseek");

   if ((k = read(fd, buf, 10)) != 10) errore("read");

   if (write(1, buf, k) != k) errore("write");

   if (close(fd) == -1) errore("close");
}
```

oihartzun_on_off adibide-programa

Gailu baten ECHO ezaugarria aldatzen duen programa inplementatzen da adibide honetan.

```
#include <stdio.h>
#include <stdlib.h>
#include <errno.h>
#include <termio.h>
#define errore(a) {perror(a); exit(1);};

main(int argc, char *argv[])   /* oihartzun_on_off.c */
{
   struct termio tm;

   if (ioctl(0, TCGETA, &tm) == -1) errore("ioctl 1");
   if (tm.c_lflag & ECHO)
      tm.c_lflag = tm.c_lflag & ~ECHO;
   else
      tm.c_lflag = tm.c_lflag | ECHO;
   if (ioctl(0, TCSETA, &tm) == -1) errore("ioctl 2");
}
```

kat adibide-programa

```
#include <stdio.h>
#include <stdlib.h>
#include <errno.h>
#include <sys/types.h>
#include <dirent.h>
#define errore(a) {perror(a); exit(1);};

main(int argc, char *argv[])   /* kat.c */
{
   DIR *dir;
   struct dirent *sarrera;

   if (argc != 2) errore("argumentuak gaizki");
   if ((dir = opendir(argv[1])) == NULL) errore("opendir");
   while ((sarrera = readdir(dir)) != NULL) {
      write(1, sarrera->d_name, strlen(sarrera->d_name));
      write(1, "\n", 1);
   }
   if (closedir(dir) == -1) errore("closedir");
}
```

berrizendatu adibide-programa

```
#include <stdio.h>
#include <stdlib.h>
#include <errno.h>
#define errore(a) {perror(a); exit(1);};

main(int argc, char *argv[])   /* berrizendatu.c */
{
   if (argc != 3) errore("argumentuak gaizki");
   if (link(argv[1], argv[2]) == -1) errore("link");
   if (unlink(argv[1]) == -1) errore("unlink");
}
```

kopiatu-kop adibide-programa

Fitxategi bat kopiatzen du sarrera estandarretik irteera estandarrera, *read* eta *write* sistema-deiak erabiliz. Aldiko zenbat byte kopiatu argumentu bezala pasatzen zaio.

```
#include <stdio.h>
#include <stdlib.h>
#include <errno.h>
#define errore(a) {perror(a); exit(1);};
#define BUFSIZE 512

main(int argc, char *argv[])   /* kopiatu-kop.c */
{
   int n, kop = BUFSIZE;
   char buf[BUFSIZE];

   if (argc == 2) {
      kop = atoi(argv[1]);
      if ((kop < 1) || kop > BUFSIZE)) errore("kop");
   }

   /* irakurketa eta idazketa zikloa */
   while ((n = read(0, buf, kop)) > 0)
      if (write(1, buf, n) != n) errore("write");

   if (n == -1) errore("read");
}
```

fkopiatu-kop adibide-programa

Fitxategi bat kopiatzen du sarrera estandarretik irteera estandarrera, *fread* eta *fwrite* liburutegiko funtzioak erabiliz. Aldiko zenbat byte kopiatu argumentu bezala pasatzen zaio.

```
#include <stdio.h>
#include <stdlib.h>
#include <errno.h>
#define errore(a) {perror(a); exit(1);};
#define BUFSIZE 512

main(int argc, char *argv[])   /* fkopiatu-kop.c */
{
   int n, kop = BUFSIZE;
   char buf[BUFSIZE];

   if (argc == 2) {
      kop = atoi(argv[1]);
      if ((kop < 1) || kop > BUFSIZE)) errore("kop");
   }

   /* irakurketa eta idazketa zikloa */
   while ((n = fread(buf, sizeof(char), kop, stdin)) > 0)
      if (fwrite(buf, sizeof(char), n, stdout) != n) errore("fwrite");

   if (n == -1) errore("fread");
}
```

ARIKETAK – Fitxategi-sistema eta Sarrera/Irteera

1. Berbideraketak erabiliz, probatu itzazu ***kopiatu-kop*** eta ***fkopiatu-kop*** programak (1) tamaina desberdineko fitxategiak kopiatuz (txikia eta handia), eta (2) argumentu bezala kopuru desberdinak erabiliz (adibidez, aldiko 512, 256, eta 2 byte). Probak egiterakoan, ***time*** tresna erabili ezazu behar den denbora neurtzeko:

```
time kopiatu-kop [kop] <f1 >f2
```

```
time fkopiatu-kop [kop] <f1 >f2
```

Aztertu itzazu exekuzio denborak, eta arrazoitu emaitza.

2. ***bikoiztu*** komandoa programatu behar da (Unix-eko ***tee*** antzekoa), sarrera estandarretik irakurritakoa irteera estandarrean eta argumentu bezala pasatako fitxategian kopiatzen duena.

```
bikoiztu fitxategia
```

3. ***ezaba2*** komandoa programatu behar da, fitxategi baten ezabaketa "berreskuragarria" egiten duena:

```
ezaba2 fitxategia
```

Komando honek fitxategiaren backup kopia bat gordetzen du `fitxategia.bck` izenarekin. Aurretik beste `fitxategia.bck` fitxategi bat egongo balitz galdu egingo litzateke. Esan ezazu zein kasu hartu behar diren kontuan komandoa zuzen exekutatu dadin.

4. ***lerroak_zenbatu*** komandoa programatu behar da (Unix-eko ***nl*** antzekoa), sarrera estandarretik lerroak irakurtzen dituena eta irteera estandarrean idazten dituena, lerro bakoitzaren hasieran lerroaren zenbakia ipiniz.

```
lerroak_zenbatu
```

Proba ezazu programa era hauetan

a) `lerroak_zenbatu`

b) `lerroak_zenbatu < datuak.txt`

Ongi funtzionatzen du bi kasuetan? Horrela ez bada, esan ezazu zergatia. Ondoren, saia zaitez arazoa konpontzen.

5. ***fitxkop*** komandoa programatu behar da, irteera estandarrean argumentu bezala pasatako katalogoak dituen sarrera kopurua idazten duena.

`sarrerakop` `katalogoa`

Egizu orain gauza bera, baina soilik fitxategi arruntak direnak zenbatuz (fitxategi arrunta dela jakiteko POSIX makroak erabili daitezke).

`fitxkop` `katalogoa`

6. ***erakutsi_inodea*** funtzioa programatu behar da, irteera estandarrean parametro bezala pasatako fitxategiaren *inode*-aren informazioa erakusten duena.

```
void erakutsi_inodea(char *izena);
```

Irteeraren formatua:

```
Lotura_kop    Jabea_ID    Taldea_ID    Tamaina    Izena
```

Aurreko ***erakutsi_inodea*** funtzioa erabiliz, ***erakutsi_inodeak*** programa inplementa ezazu, irteera estandarrean argumentu bezala pasatako fitxategien *inode*-en informazioak idazten dituena. Argumenturik ez bazaio pasatzen, sarrera estandarrari dagokion *inode*-aren informazioak idatziko ditu.

`erakutsi_inodeak` `[fitx1 [fitx2 ... [fitxn]]]`

7. Esaera zaharrak dituen fitxategi bat dugu, esaera bakoitza erregistro batean gordetzen delarik (`struct esaera`). Fitxategi hau erabili nahi dugu esaera zaharrak irteera estandarrean ausaz idatziko dituen ondoko programa inplementatuz

`esaera_zaharrak` `esaeren_fitxategia esaera_kopurua`

esaera_kopurua argumentuak irteera estandarrean idatzi beharreko esaera kopurua adierazten du. Esaera zahar bat aukeratzeko, dagoeneko inplementatutzat emango dugun `ausaz(int eremua)` funtzioa erabiliko dugu; honek ausazko zenbaki bat itzultzen du 0 eta *eremua*–1 bitarte (*eremua* fitxategiak duen erregistro kopurua izango da, kopuru hau zuk kalkulatu beharko duzularik). Ausaz ateratako zenbaki bakoitzeko, fitxategian dagokion esaera zaharraren informazioa idatziko da irteera estandarrean

```
struct esaera {
   int   urtea;
   char  egilea[20];
   char  testua[256];
}
```

Oharra: ausaz funtzioa ez da kodetu behar; aldiz, kodetuta ematen zaizue ausaz.o fitxategian.