Laboratorul 2

Funcții sistem

1 Utilizarea funcțiilor sistem

Reamintim faptul că funcțiile de sistem (syscalls) sunt definite în secțiunea 2 a manualului sistemului de operare. Aprofundați definiția și utilizarea fiecărei funcții folosite în acest material prin apeluri de tipul

```
$ man 2 <syscall>
```

Funcțiile cele mai des întâlnite pentru manipularea fișierelor sunt `read(2)`, `write(2)`, `stat(2)`, `open(2)` și `close(2)`.

1.1 Citire și scriere

Am văzut în Laboratorul 1 cum se comportă `write(2)`. Similar, `read(2)` citește dintr-un descriptor `d` în bufferul `buf` un număr de bytes dat de `nbytes`.

```
ssize_t read(int d, void *buf, size_t nbytes);
```

Când este executată cu succes, ieșirea funcției este numărul de bytes citiți.

1.2 Accesarea fișierelor

Pentru a obține un descriptor asociat unui fișier trebuie folosită funcția `open(2)` care deschide fișierul găsit în `path` pentru scriere și/sau citire.

```
int open(const char *path, int flags, ...);
```

Ieșirea funcției este descriptorul asociat. Modul în care va fi manipulat fișierul este dat de argumentul `flags` similar funcției standard C `fopen(3)`.

```
O_RDONLY     Open for reading only.
O_WRONLY     Open for writing only.
O_RDWR       Open for reading and writing.
```

Dacă fișierul cerut nu există în sistem, se poate cere crearea lui prin adăugarea flagului `O_CREAT` la cele de scriere sau citire. În acest caz, trebuie specificate și drepturile de acces la fișier în al treilea argument.

```
#define S_IRWXU 0000700    /* RWX mask for owner */
#define S_IRUSR 0000400    /* R for owner */
#define S_IWUSR 0000200    /* W for owner */
#define S_IXUSR 0000100    /* X for owner */
```

Vezi manualul `open(2)` și tabelul din `chmod(2)` pentru mai multe detalii.

Orice fișier deschis cu `open(2)` trebuie închis cu `close(2)` când nu mai este folosit.

1.3 Informații despre fișiere

Pentru a afla detalii despre obiectele manipulate, precum dimensiunea ocupată pe disc, permisiunile de acces, data la care a fost creat și modificat ultima dată, se folosește funcția `stat(2)`.

```
int stat(const char *path, struct stat *sb);
```

În câmpurile structurii de date `stat` vor fi populate informațiile de mai sus împreună cu alte detalii.

```
struct stat {
 dev_t      st_dev;    /* inode's device */
 ino_t      st_ino;    /* inode's number */
 mode_t     st_mode;   /* inode protection mode */
 nlink_t    st_nlink;  /* number of hard links */
 uid_t      st_uid;    /* user ID of the file's owner */
 gid_t      st_gid;    /* group ID of the file's group */
 dev_t      st_rdev;   /* device type */
 struct timespec st_atim;  /* time of last access */
 struct timespec st_mtim;  /* last data modification */
 struct timespec st_ctim;  /* last file status change */
 off_t      st_size;   /* file size, in bytes */
 blkcnt_t   st_blocks; /* blocks allocated for file */
 blksize_t  st_blksize;/* optimal blocksize for I/O */
 u_int32_t  st_flags;  /* user defined flags for file */
 u_int32_t  st_gen;    /* file generation number */
 };
```

Următorul fragment de program afișează dimensiunea fișierului `foo`.

```
#include <sys/stat.h>
...
struct stat sb;
if (stat("foo", &sb)) {
        perror("foo");
```

errno	Valoare	Descriere
1	EPERM	operația nu este permisă
2	ENOENT	fișier sau director inexistent
5	EIO	eroare de comunicare intrare/ieșire (cu un dispozitiv)
9	EBADF	descriptor inexistent
12	ENOMEM	memorie insuficientă
13	EACESS	nu sunt permisiuni suficiente de acces
14	EFAULT	adresă invalidă
22	EINVAL	argument invalid

Tabela 1: Coduri de eroare uzuale

```
        return errno;
}
printf("Foo takes %jd bytes on disk\n", sb.st_size);
```

2 Tratarea erorilor

În manualele de utilizare există o secțiune importantă numită `RETURN VALUES`. Adesea valoarea la ieșirea cu succes este pozitivă, iar când apelul întâmpină o problemă utilizatorul este semnalat prin valoarea -1. În acest caz mai multe detalii se pot găsi în variabila globală `ernno`. Codul de eroare indicat are asociat un mesaj de eroare ce poate fi ușor afișat pe ecran cu ajutorul funcției `perror(3)`.

Documentația funcției `read(2)` spune următoarele:

> **RETURN VALUES**
> If successful, the number of bytes actually read is returned. Upon reading end-of-file, zero is returned. Otherwise, a -1 is returned and the global variable errno is set to indicate the error.

Așadar, un apel corect al funcției arată astfel:

```
nread = read(fd, buf, bufsz);
if (nread < 0) {
        perror("read buf");
        return errno;
}
```

În anumite cazuri se poate face un caz special și pentru `nread == 0`, semnalând că am ajuns la sfârșitul fișierului.

În Tabelul 1 puteți găsi câteva din cele mai frecvente erori semnalate de `errno`. O listă completă cu valorile posibile și semnificația lor se găsește în manual `errno(2)`.

Toate apelurile de funcții trebuie verificate corespunzător pentru toate ieșirile posibile – fie cu succes, fie fără!

3 Sarcini de laborator

1. Rescrieți programul HelloWorld de data trecută folosind numai funcții sistem.

2. Scrieți un program `mycp` care să primească la intrare în primul argument un fișier sursă pe care să-l copieze într-un alt fișier cu numele primit în al doilea argument. Exemplu apel: `./mycp foo bar`.