Laborator 3

```
biancapinghireac@vbox:~/SO/lab3/src$ make lib
gcc -Wall -g -O   -c -o error.o error.c
make: gcc: No such file or directory
make: *** [<builtin>: error.o] Error 127
biancapinghireac@vbox:~/SO/lab3/src$
biancapinghireac@vbox:~/SO/lab3/src$ gcc --version
bash: gcc: command not found...
Install package 'gcc' to provide command 'gcc'? [N/y] y
```

Initial nu aveam gcc pentru compilare, dar l am instalat

```
biancapinghireac@vbox:~/SO/lab3/src$ which gcc
/usr/bin/gcc
biancapinghireac@vbox:~/SO/lab3/src$ make lib
gcc -Wall -g -O   -c -o error.o error.c
ar rcs liblab3.a error.o
biancapinghireac@vbox:~/SO/lab3/src$ tree
.
├── error.c
├── error.o
├── fileflags.c
├── hole.c
├── liblab3.a
├── Makefile
├── mycat.c
├── ourhdr.h
└── seek.c

1 directory, 9 files
biancapinghireac@vbox:~/SO/lab3/src$
```

Am creat biblioteca liblab3.a prin comanda make lib

```
biancapinghireac@vbox:~/SO/lab3/src$ make
gcc -o seek seek.c liblab3.a
gcc -o hole hole.c liblab3.a
gcc -o mycat mycat.c liblab3.a
gcc -o fileflags fileflags.c liblab3.a
```

Am folosit comanda make pentru a compila exemplele.

```
biancapinghireac@vbox:~/SO/lab3/src$ ./seek < /etc/motd
seek OK
```

comanda verifică dacă putem folosi locația curentă pentru a scrie in etc/motd

```
biancapinghireac@vbox:~/SO/lab3/src$ cat /etc/motd | ./seek
cannot seek
```

comandă verifică dacă se poate citi in ‘etc/motd’

```c
#include <sys/types.h>
#include "ourhdr.h"

int main(void)
{
        if (lseek(STDIN_FILENO, 0, SEEK_CUR) == -1)
                printf("cannot seek\n");
        else
                printf("seek OK\n");
        exit(0);
}
```

Codul pentru seek.c

```
biancapinghireac@vbox:~/SO/lab3/src$ ./hole
biancapinghireac@vbox:~/SO/lab3/src$ ls -l file.hole
-rw-r--r--. 1 biancapinghireac biancapinghireac 50 Mar 19 10:48 file.hole
```

```
biancapinghireac@vbox:~/SO/lab3/src$ od -c file.hole
0000000   a   b   c   d   e   f   g   h   i   j  \0  \0  \0  \0  \0  \0
0000020  \0  \0  \0  \0  \0  \0  \0  \0  \0  \0  \0  \0  \0  \0  \0  \0
0000040  \0  \0  \0  \0  \0  \0  \0  \0   A   B   C   D   E   F   G   H
0000060   I   J
0000062
```

Am creat un fișier cu o ”gaură” și după am afisat ce este in el cu octal dump

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

#include "ourhdr.h"

char buf1[] = "abcdefghij";
char buf2[] = "ABCDEFGHIJ";

int main(void)
{
    void err_sys(const char *msg){
      perror(msg);
      exit(10);
    }

        int fd;

        if ((fd = creat("file.hole", FILE_MODE)) < 0)
                err_sys("creat error");

        if (write(fd, buf1, 10) != 10)
                err_sys("buf1 write error");
        /* offset now = 10 */

        if (lseek(fd, 40, SEEK_SET) == -1)
                err_sys("lseek error");
        /* offset now = 40 */

        if (write(fd, buf2, 10) != 10)
                err_sys("buf2 write error");
        /* offset now = 50 */

        exit(0);
}
```

Codul pentru hole.c

(Muta offset ul la 40 (adica ii lasa o gaura intre 10 si 40))

```
biancapinghireac@vbox:~/SO/lab3/src$ ./mycat < file.hole
abcdefghijABCDEFGHIJbiancapinghireac@vbox:~/SO/lab3/src$
```

Mycat.c copiaza standardul input la standardul output

```c
#include "ourhdr.h"

#define BUFFSIZE 8192

int main(void)
{
    void err_sys(const char *msg){
        perror(msg);
        exit(10);
   }
        int n;
        char buf[BUFFSIZE];

        while ( (n = read(STDIN_FILENO, buf, BUFFSIZE)) > 0)
                if (write(STDOUT_FILENO, buf, n) != n)
                        err_sys("write error");

        if (n < 0)
                err_sys("read error");

        exit(0);
```

Cod pentru mycat.c

```
biancapinghireac@vbox:~/SO/lab3/src$ ./fileflags 0 < /dev/tty
read only
```

```
biancapinghireac@vbox:~/SO/lab3/src$ ./fileflags 1 > temp.foo
biancapinghireac@vbox:~/SO/lab3/src$ cat temp.foo
write only
```

```
biancapinghireac@vbox:~/SO/lab3/src$ ./fileflags 2 2>> temp.foo
write only, append
```

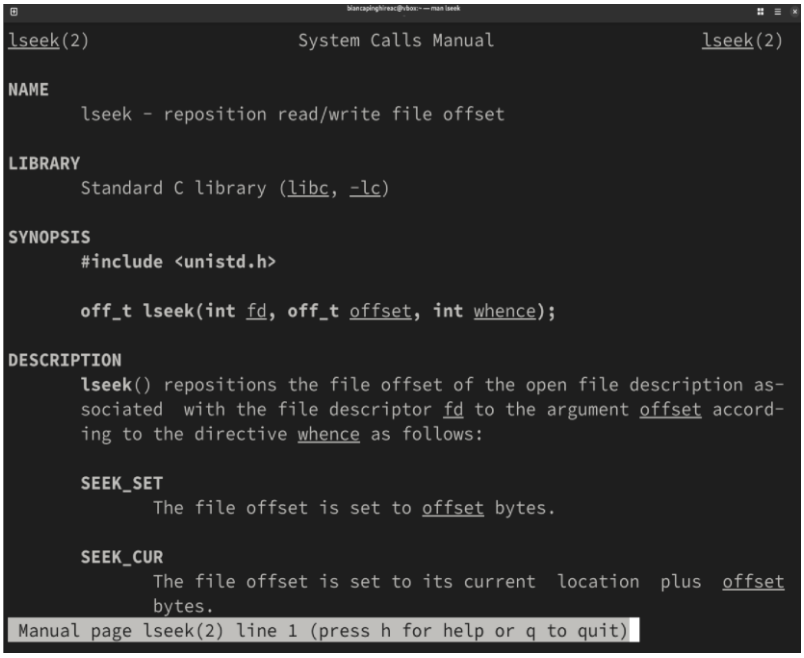
```
biancapinghireac@vbox:~/SO/lab3/src$ ./fileflags 5 5<> temp.foo
read write
```

„fileflags” este un program care primeste un descriptor de fisier ca argument si apeleaza fcntl(F_GETFL) pentru a afisa modul in care acel descriptor este deschis (read only, write only, read write) si eventualele flag-uri suplimentare (append, nonblocking, etc.).

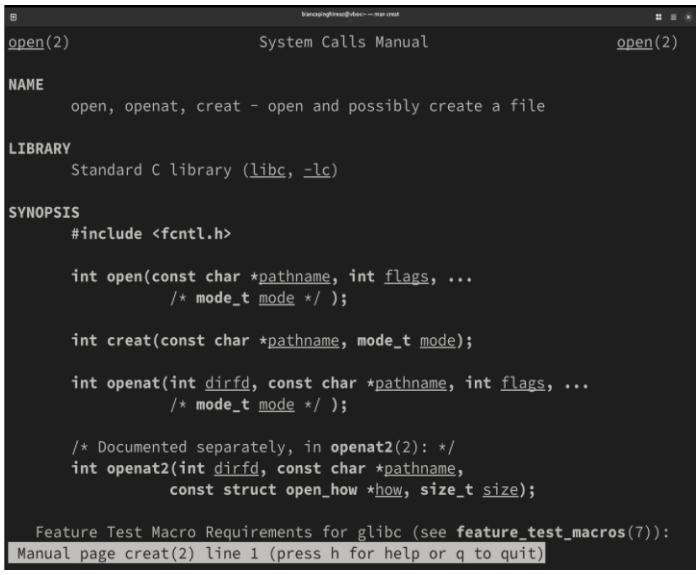
```c
#include <sys/types.h>
#include <fcntl.h>
#include "ourhdr.h"
#include <stdio.h>
#include <stdlib.h>
#include <stdarg.h>

void err_sys(const char *fmt, ...) {
    va_list ap;
    va_start(ap, fmt);
    vfprintf(stderr, fmt, ap);
    va_end(ap);
    exit(1);
}

void err_quit(const char *fmt, ...) {
    va_list ap;
    va_start(ap, fmt);
    vfprintf(stderr, fmt, ap);
    va_end(ap);
    exit(1);
}

void err_dump(const char *fmt, ...) {
    va_list ap;
    va_start(ap, fmt);
    vfprintf(stderr, fmt, ap);
    va_end(ap);
    abort(); // abort() pentru a genera un core dump
}

int main(int argc, char *argv[])
{

        int accmode, val;


        if (argc != 2)
                err_quit("usage: fileflags <descriptor#>");

        if ( (val = fcntl(atoi(argv[1]), F_GETFL, 0)) < 0)
                err_sys("fcntl error for fd %d", atoi(argv[1]));

        accmode = val & O_ACCMODE;
        if      (accmode == O_RDONLY)   printf("read only");
        else if (accmode == O_WRONLY)   printf("write only");
        else if (accmode == O_RDWR)     printf("read write");
        else err_dump("unknown access mode");

        if (val & O_APPEND)     printf(", append");
        if (val & O_NONBLOCK)   printf(", nonblocking");
#if !defined(_POSIX_SOURCE) && defined(O_SYNC)
        if (val & O_SYNC)       printf(", synchronous writes");
#endif
        putchar('\n');
        exit(0);
}
```

Cod pentru fileflags.c

Pagini de manual al comenzilor utilizate:

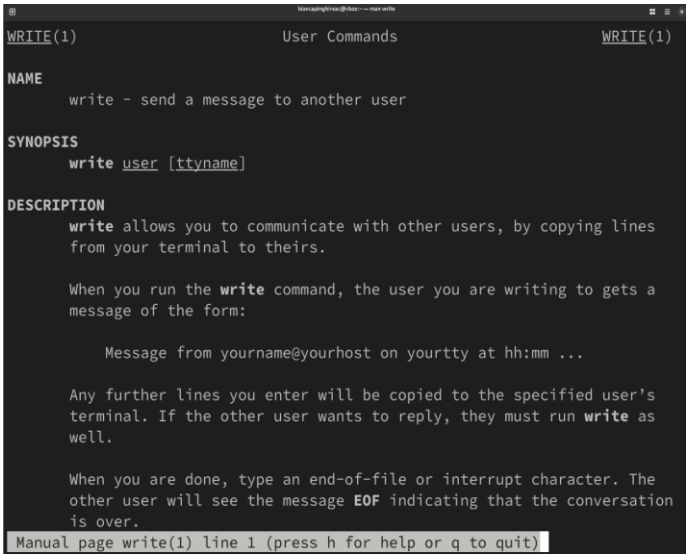


lseek



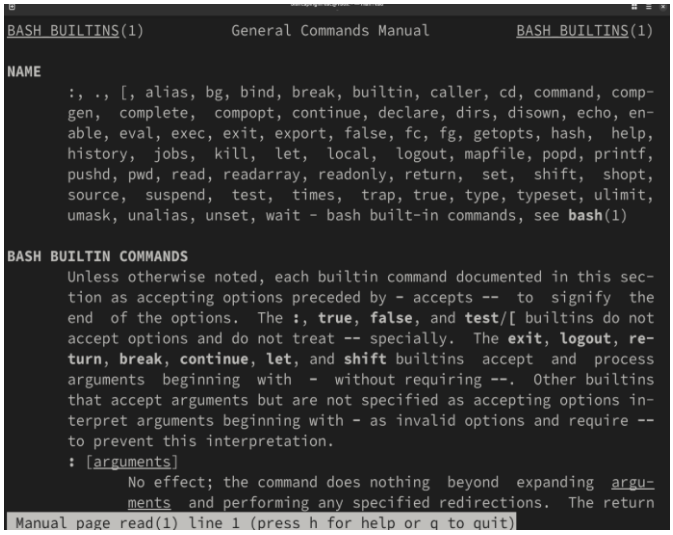
```
open(2)                    System Calls Manual                    open(2)

NAME
       open, openat, creat - open and possibly create a file

LIBRARY
       Standard C library (libc, -lc)

SYNOPSIS
       #include <fcntl.h>

       int open(const char *pathname, int flags, ...
                   /* mode_t mode */ );

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

       int openat(int dirfd, const char *pathname, int flags, ...
                   /* mode_t mode */ );

       /* Documented separately, in openat2(2): */
       int openat2(int dirfd, const char *pathname,
                   const struct open_how *how, size_t size);

   Feature Test Macro Requirements for glibc (see feature_test_macros(7)):
 Manual page creat(2) line 1 (press h for help or q to quit)
```

Creat



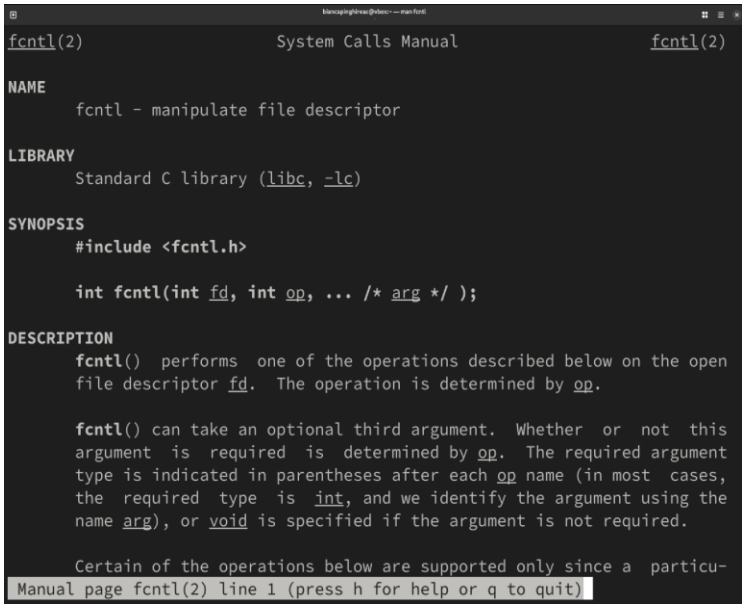
```
WRITE(1)                     User Commands                     WRITE(1)

NAME
       write - send a message to another user

SYNOPSIS
       write user [ttyname]

DESCRIPTION
       write allows you to communicate with other users, by copying lines
       from your terminal to theirs.

       When you run the write command, the user you are writing to gets a
       message of the form:

           Message from yourname@yourhost on yourtty at hh:mm ...

       Any further lines you enter will be copied to the specified user's
       terminal. If the other user wants to reply, they must run write as
       well.

       When you are done, type an end-of-file or interrupt character. The
       other user will see the message EOF indicating that the conversation
       is over.
 Manual page write(1) line 1 (press h for help or q to quit)
```

Write



```
BASH_BUILTINS(1)          General Commands Manual          BASH_BUILTINS(1)

NAME
       :, ., [, alias, bg, bind, break, builtin, caller, cd, command, comp-
       gen,  complete,  compopt, continue, declare, dirs, disown, echo, en-
       able, eval, exec, exit, export, false, fc, fg, getopts, hash,  help,
       history,  jobs,  kill,  let,  local,  logout, mapfile, popd, printf,
       pushd, pwd, read, readarray, readonly, return,  set,  shift,  shopt,
       source,  suspend,  test,  times,  trap, true, type, typeset, ulimit,
       umask, unalias, unset, wait - bash built-in commands, see bash(1)

BASH BUILTIN COMMANDS
       Unless otherwise noted, each builtin command documented in this sec-
       tion as accepting options preceded by - accepts --  to  signify  the
       end  of the options.  The :, true, false, and test/[ builtins do not
       accept options and do not treat -- specially.  The exit, logout, re-
       turn, break, continue, let, and shift builtins  accept  and  process
       arguments  beginning  with  -  without requiring --.  Other builtins
       that accept arguments but are not specified as accepting options in-
       terpret arguments beginning with - as invalid options and require --
       to prevent this interpretation.
       : [arguments]
              No effect; the command does nothing  beyond  expanding  argu-
              ments  and performing any specified redirections.  The return
 Manual page read(1) line 1 (press h for help or q to quit)
```

Read



```
fcntl(2)                  System Calls Manual                  fcntl(2)

NAME
       fcntl - manipulate file descriptor

LIBRARY
       Standard C library (libc, -lc)

SYNOPSIS
       #include <fcntl.h>

       int fcntl(int fd, int op, ... /* arg */ );

DESCRIPTION
       fcntl()  performs  one of the operations described below on the open
       file descriptor fd.  The operation is determined by op.

       fcntl() can take an optional third argument.  Whether  or  not  this
       argument  is  required  is  determined by op.  The required argument
       type is indicated in parentheses after each op name (in most  cases,
       the  required  type  is  int, and we identify the argument using the
       name arg), or void is specified if the argument is not required.

       Certain of the operations below are supported only since a  particu-
 Manual page fcntl(2) line 1 (press h for help or q to quit)
```

Fcntl

EXERCITIUL 2:

Dacă deschidem un fisier pentru read-write cu flag-ul pentru append, se poate citi din orice pozitie specificată cu lseek? Se poate scrie peste datele existente?

Poți citi din orice poziție, însă dacă fișierul este deschis cu flag-ul `O_APPEND`, toate operațiile de scriere vor fi redirecționate automat la sfârșitul fișierului, indiferent de offset-ul setat prin lseek(). Cu alte cuvinte, nu se poate suprascrie conținutul existent.

Exemplu:

```c
#include <fcntl.h>
#include <string.h>
#include <unistd.h>
#include <stdio.h>

int main(){

        int fd = open("test.txt", O_RDWR | O_CREAT | O_APPEND, 0666);
        write(fd, "Primul text\n",12);

        lseek(fd, 0, SEEK_SET); //vrem sa mergem la inceput
        write(fd, "Incercare suprascriere\n",23);

        //demonstram ca se poate citii de oriunde
        lseek(fd, 4, SEEK_SET);
        char buffDem[16];
        int n = read(fd,buffDem, 4); //citesc 4 caractere
        buffDem[n] = '\0';
        printf("Citire de la offset 4: %s\n", buffDem);

        //citim fisierul pt vizualizare
        lseek(fd, 0, SEEK_SET);
        char buff[128];
        int m = read(fd, buff, sizeof(buff)-1);
        buff[m] = '\0';
        write(STDOUT_FILENO, buff, m);

        close(fd);
        return 0;
}
```

```
biancapinghireac@vbox:~/SO/lab3/src$ ./exemplu
Citire de la offset 4: ul t
Primul text
Incercare suprascriere
```

Dupa cum se poate observa nu se suprasrie la inceput, dar se poate citi de oriunde

EXERCITIUL 3:

```
biancapinghireac@vbox:~/SO/lab3/src$ nano hole2.c
biancapinghireac@vbox:~/SO/lab3/src$ gcc hole2.c -o hole2
biancapinghireac@vbox:~/SO/lab3/src$ ./hole2
biancapinghireac@vbox:~/SO/lab3/src$ cat file.hole2
abcdefABCDEFabcdefghbiancapinghireac@vbox:~/SO/lab3/src$ od file.hole2
0000000 061141 062143 063145 000000 000000 000000 000000 000000
0000020 000000 000000 041101 042103 043105 000000 000000 000000
0000040 000000 000000 000000 000000 061141 062143 063145 064147
0000060
```

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

#include "ourhdr.h"

char buff1[]="abcdef";
char buff2[]="ABCDEF";
char buff3[]="abcdefgh";

int main(void){
        void err_sys(const char *msg){
                perror(msg);
                exit(10);
        }
        int fd;

        if((fd = creat("file.hole2",FILE_MODE)) < 0)
                err_sys("cannot create file");
        if(write(fd,buff1,6) != 6) //offset - 6
                err_sys("write error");
        if(lseek(fd,20,SEEK_SET) == -1) //offset - 20
                err_sys("lseek error");
        if(write(fd,buff2,6) != 6) //offset - 26
                err_sys("write(2) error");
        if(lseek(fd,40,SEEK_SET) == -1) //offset - 40
                err_sys("lseek(2) error");
        if(write(fd,buff3,8) != 8) //offset -48
                err_sys("write(3) error");
        exit(0);
}
```

Am schimbat offset-ul de mai multe ori petru a face “gauri”

```
biancapinghireac@vbox:~/SO/lab3/src$ gcc mycat2.c -o mycat2
biancapinghireac@vbox:~/SO/lab3/src$ ./mycat2 < file.hole2 > file.nohole2
biancapinghireac@vbox:~/SO/lab3/src$ od -tx1 file.nohole2
0000000 61 62 63 64 65 66 41 42 43 44 45 46 61 62 63 64
0000020 65 66 67 68
0000024
biancapinghireac@vbox:~/SO/lab3/src$ cat file.nohole2
```

```c
#include "ourhdr.h"

#define BUFFSIZE 8192

int main(void){
        void err_sys(const char *msg){
                perror(msg);
                exit(10);
        }
        ssize_t  n;
        char buff[BUFFSIZE];

        while((n = read(STDIN_FILENO, buff, BUFFSIZE)) > 0) {
                for(ssize_t i=0;i<n;i++){
                        if(buff[i] != 0){
                                if(write(STDOUT_FILENO, &buff[i],1) !=1)
                                        err_sys("write error");
                        }
                }
        }
        if(n<0)
                err_sys("read error");
        exit(1);
}
```

citeste date bloc cu bloc de la intrarea standard (read), apoi parcurge fiecare octet din buffer. Daca acel octet nu este zero (0x00), il scrie (write) la iesirea standard. Octetii 0 sunt

sariti (nu se scriu), iar daca apar erori de citire sau scriere, se afisează un mesaj și programul se opreste

EXERCITIUL 4:

```
biancapinghireac@vbox:~/SO/lab3/src$ ./fileflags 2 2>> temp.foo
write only, append
```

Redirecționarea erorilor standard (descriptorul 2):
in acest caz, descriptorul 2 (stderr) este redirectionat in fisierul „temp.foo” cu flag-ul „append” („write only, append”).

```
biancapinghireac@vbox:~/SO/lab3/src$ ./fileflags 5 5<> temp.foo
read write
```

Redirecționarea unui descriptor suplimentar (5) în mod citire-scriere:
aceasta deschide fisierul „temp.foo” pentru a fi accesat de descriptorul 5 in mod „read write”. Simbolul „<>” din shell inseamna citire-scriere pe acelasi fisier.

Alte exemple de folosire:

./fileflags 0 5 0< input1.txt 5< input2.txt
Explicatie:

- Descriptorul 0 citeste din input1.txt ⇒ “read only” afisat de fileflags 0.
- Descriptorul 5 citeste din input2.txt ⇒ “read only” afisat de fileflags 5.
- putem avea doua fisiere de intrare diferite pe descriptorii 0 și 5.

---

exec 6</dev/tty
./fileflags 6
Explicatie:

- exec 6</dev/tty deschide /dev/tty (/dev/tty este, de obicei, “terminalul de control” al sesiunii, adica exact tastatura si ecranul cu care lucrezi in acel moment) in mod read-only pentru descriptorul 6
- Cand rulezi “./fileflags 6”, vei vedea “read only”
- Asta iti permite sa folosesti descriptorul 6 pentru a citi date direct de la tastatura, lasand descriptorul 0 liber, de exemplu, pentru un fisier