Sisteme de operare
Tema 5

Exercițiul 1

Rulați toate programele prezentate. Asigurați-vă că le-ați înțeles funcționarea. Folosiți-vă de pagina 2 de manual (comanda `man`).

```
[user@fedora destination]$ make lib
gcc -Wall -g -O   -c -o error.o error.c
gcc -Wall -g -O   -c -o pathalloc.o pathalloc.c
ar rcs liblab5.a error.o pathalloc.o
[user@fedora destination]$ make
gcc -o ftw4 ftw4.c liblab5.a
gcc -o unlink unlink.c liblab5.a
gcc -o zap zap.c liblab5.a
gcc -o mycd mycd.c liblab5.a
gcc -o cdpwd cdpwd.c liblab5.a
gcc -o devrdev devrdev.c liblab5.a
```

Am creat librăria liblab4.a .

```
[user@fedora destination]$ df /home
Filesystem                   1K-blocks    Used Available Use% Mounted on
/dev/mapper/fedora_fedora-root  15718400 2620564  13097836  17% /
[user@fedora destination]$ cp ftw4.c tempfile
[user@fedora destination]$ df /home
Filesystem                   1K-blocks    Used Available Use% Mounted on
/dev/mapper/fedora_fedora-root  15718400 2620052  13098348  17% /
[user@fedora destination]$ echo mini >tempfile
[user@fedora destination]$ df /home
Filesystem                   1K-blocks    Used Available Use% Mounted on
/dev/mapper/fedora_fedora-root  15718400 2620052  13098348  17% /
[user@fedora destination]$ ./unlink
unlink error: Text file busy
[user@fedora destination]$ df /home
Filesystem                   1K-blocks    Used Available Use% Mounted on
/dev/mapper/fedora_fedora-root  15718400 2620052  13098348  17% /
```

Programul unlink.

```
[user@fedora destination]$ cp unlink tempfile
[user@fedora destination]$ ls -l tempfile
-rwxrwxrwx. 1 root root 30504 Apr  3 18:17 tempfile
[user@fedora destination]$ ls -lu tempfile
-rwxrwxrwx. 1 root root 30504 Apr  3 18:17 tempfile
[user@fedora destination]$ date
Mon Apr  3 06:17:56 PM EEST 2023
[user@fedora destination]$ ./zap tempfile
[user@fedora destination]$ ls -l tempfile
-rwxrwxrwx. 1 root root 0 Apr  3 18:17 tempfile
[user@fedora destination]$ ls -lu tempfile
-rwxrwxrwx. 1 root root 0 Apr  3 18:17 tempfile
[user@fedora destination]$ ls -lc tempfile
-rwxrwxrwx. 1 root root 0 Apr  3 18:18 tempfile
```

Programul zap.

```
[user@fedora destination]$ sudo ./ftw4 /usr
[sudo] password for user:
regular files  =   68947, 77.74 %
directories    =   10390, 11.71 %
block special  =       0,  0.00 %
char special   =       0,  0.00 %
FIFOs          =       0,  0.00 %
symbolic links =    9354, 10.55 %
sockets        =       0,  0.00 %
```

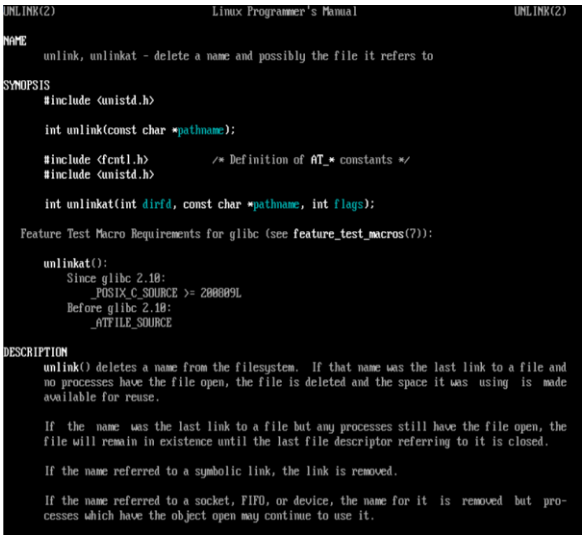
Programul ftw4.

```
[user@fedora destination]$ ./cdpwd
cwd = /var/spool
[user@fedora destination]$ ls -l /var/spool
total 4
drwxr-x--x. 34 root abrt 4096 Apr  3 18:12 abrt
drwx------.  2 abrt abrt    6 Mar 10  2022 abrt-upload
drwxr-xr-x.  2 root root   63 Feb 28 12:49 anacron
drwx------.  3 root root   31 Mar 31  2022 at
drwx------.  2 root root    6 Jul  1  2022 cron
drwxr-xr-x.  2 root root    6 Aug  9  2022 lpd
drwxrwxr-x.  2 root mail   29 Aug  9  2022 mail
drwxr-xr-x.  2 root root    6 Mar  7  2022 plymouth
```

Programul cdpwd.

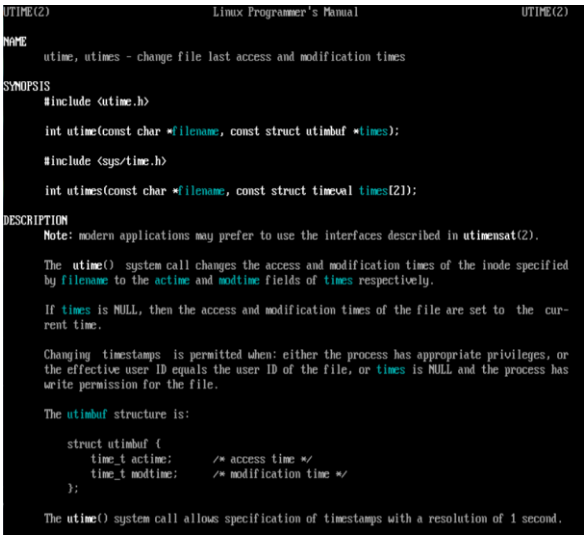
```
[user@fedora destination]$ ./devrdev / /dev/sr0 /dev/tty
/: dev = 253/0
/dev/sr0: dev = 0/5 (block) rdev = 11/0
/dev/tty: dev = 0/5 (character) rdev = 5/0
```

Programul devrdev.



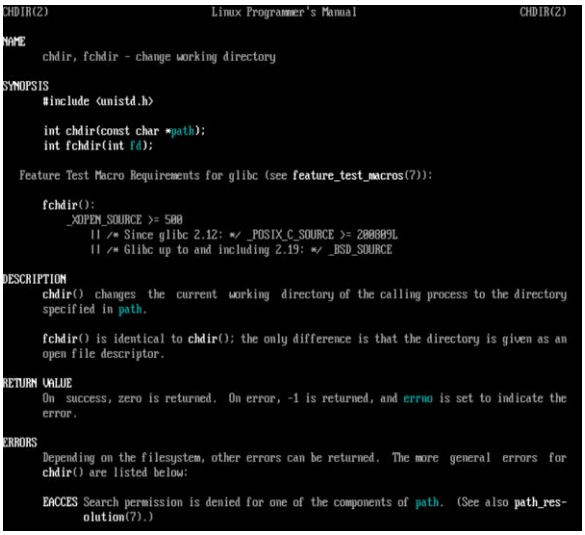
```
UNLINK(2)                  Linux Programmer's Manual                 UNLINK(2)

NAME
       unlink, unlinkat - delete a name and possibly the file it refers to

SYNOPSIS
       #include <unistd.h>

       int unlink(const char *pathname);

       #include <fcntl.h>           /* Definition of AT_* constants */
       #include <unistd.h>

       int unlinkat(int dirfd, const char *pathname, int flags);

   Feature Test Macro Requirements for glibc (see feature_test_macros(7)):

       unlinkat():
           Since glibc 2.10:
               _POSIX_C_SOURCE >= 200809L
           Before glibc 2.10:
               _ATFILE_SOURCE

DESCRIPTION
       unlink() deletes a name from the filesystem.  If that name was the last link to a file and
       no processes have the file open, the file is deleted and the space it was  using  is  made
       available for reuse.

       If  the  name  was the last link to a file but any processes still have the file open, the
       file will remain in existence until the last file descriptor referring to it is closed.

       If the name referred to a symbolic link, the link is removed.

       If the name referred to a socket, FIFO, or device, the name for it  is  removed  but  pro-
       cesses which have the object open may continue to use it.
```

Pagina de manual pentru programul unlink.



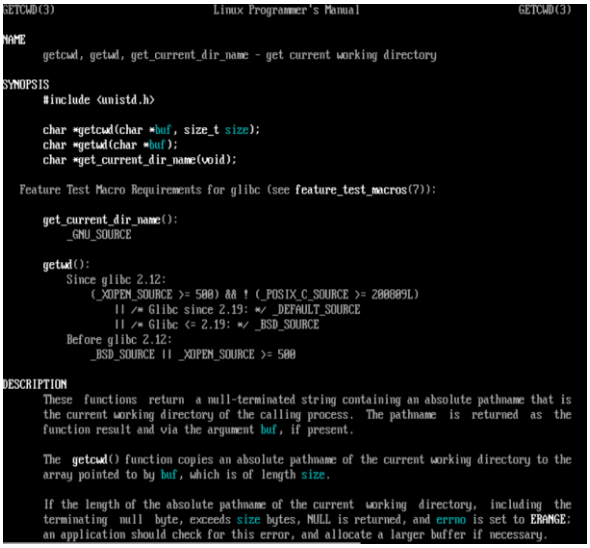
```
UTIME(2)                   Linux Programmer's Manual                  UTIME(2)

NAME
       utime, utimes - change file last access and modification times

SYNOPSIS
       #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]);

DESCRIPTION
       Note: modern applications may prefer to use the interfaces described in utimensat(2).

       The  utime()  system call changes the access and modification times of the inode specified
       by filename to the actime and modtime fields of times respectively.

       If times is NULL, then the access and modification times of the file are set to  the  cur-
       rent time.

       Changing  timestamps  is permitted when: either the process has appropriate privileges, or
       the effective user ID equals the user ID of the file, or times is NULL and the process has
       write permission for the file.

       The utimbuf structure is:

           struct utimbuf {
               time_t actime;       /* access time */
               time_t modtime;      /* modification time */
           };

       The utime() system call allows specification of timestamps with a resolution of 1 second.
```

Pagina de manual pentru funcția utime.



```
CHDIR(2)                    Linux Programmer's Manual                    CHDIR(2)

NAME
       chdir, fchdir - change working directory

SYNOPSIS
       #include <unistd.h>

       int chdir(const char *path);
       int fchdir(int fd);

   Feature Test Macro Requirements for glibc (see feature_test_macros(7)):

       fchdir():
           _XOPEN_SOURCE >= 500
               || /* Since glibc 2.12: */ _POSIX_C_SOURCE >= 200809L
               || /* Glibc up to and including 2.19: */ _BSD_SOURCE

DESCRIPTION
       chdir()  changes  the  current  working  directory of the calling process to the directory
       specified in path.

       fchdir() is identical to chdir(); the only difference is that the directory is given as an
       open file descriptor.

RETURN VALUE
       On  success, zero is returned.  On error, -1 is returned, and errno is set to indicate the
       error.

ERRORS
       Depending on the filesystem, other errors can be returned.  The more  general  errors  for
       chdir() are listed below:

       EACCES Search permission is denied for one of the components of path.  (See also path_res-
              olution(7).)
```

Pagina de manual pentru funcția chdir.



```
GETCWD(3)                   Linux Programmer's Manual                   GETCWD(3)

NAME
       getcwd, getwd, get_current_dir_name - get current working directory

SYNOPSIS
       #include <unistd.h>

       char *getcwd(char *buf, size_t size);
       char *getwd(char *buf);
       char *get_current_dir_name(void);

   Feature Test Macro Requirements for glibc (see feature_test_macros(7)):

       get_current_dir_name():
           _GNU_SOURCE

       getwd():
           Since glibc 2.12:
               (_XOPEN_SOURCE >= 500) && ! (_POSIX_C_SOURCE >= 200809L)
                   || /* Glibc since 2.19: */ _DEFAULT_SOURCE
                   || /* Glibc <= 2.19: */ _BSD_SOURCE
           Before glibc 2.12:
               _BSD_SOURCE || _XOPEN_SOURCE >= 500

DESCRIPTION
       These  functions  return  a null-terminated string containing an absolute pathname that is
       the current working directory of the calling process.  The pathname  is  returned  as  the
       function result and via the argument buf, if present.

       The  getcwd() function copies an absolute pathname of the current working directory to the
       array pointed to by buf, which is of length size.

       If the length of the absolute pathname of the current  working  directory,  including  the
       terminating  null  byte, exceeds size bytes, NULL is returned, and errno is set to ERANGE;
       an application should check for this error, and allocate a larger buffer if necessary.
```

Pagina de manual pentru funcția getcwd.

Exercițiul 2

Să se scrie un program care să producă același rezultat ca și comanda `ls` (invocată fără parametri).

```
[user@fedora destination]$ ls
2       devrdev    error.o  liblab5.a  mycd.c      pathalloc.o  unlink.c
cdpwd   devrdev.c  ftw4     Makefile   ourhdr.h    tempfile     zap
cdpwd.c error.c    ftw4.c   mycd       pathalloc.c unlink       zap.c
```

Comanda ls afișează toate fișierele care se află în folder-ul current, în cazul nostrum acesta fiind folder-ul destination.

```c
#include <stdio.h>
#include <dirent.h>
#include <string.h>

int main(void)
{
        DIR *dir;
        struct dirent *entry;

        dir=opendir(".");
        if(dir==NULL)
        {
                perror("opendir");
                return 1;
        }
        while((entry=readdir(dir))!=NULL)
        {
                if((strcmp(entry->d_name,".")!=0)&&(strcmp(entry->d_name,"..")!=0))
                    printf("%s\n",entry->d_name);
        }
        closedir(dir);
        return 0;
}
```

Programul în C.

```
[user@fedora destination]$ make exercitiu2
gcc -Wall -g -O    exercitiu2.c   -o exercitiu2
[user@fedora destination]$ ./exercitiu2
2
cdpwd
cdpwd.c
devrdev
devrdev.c
error.c
error.o
exercitiu2
exercitiu2.c
ftw4
ftw4.c
liblab5.a
Makefile
mycd
mycd.c
ourhdr.h
pathalloc.c
pathalloc.o
tempfile
unlink
unlink.c
zap
zap.c
```

Am apelat programul nostru, exercițiu2, și produce același rezultat ca și comanda ls.

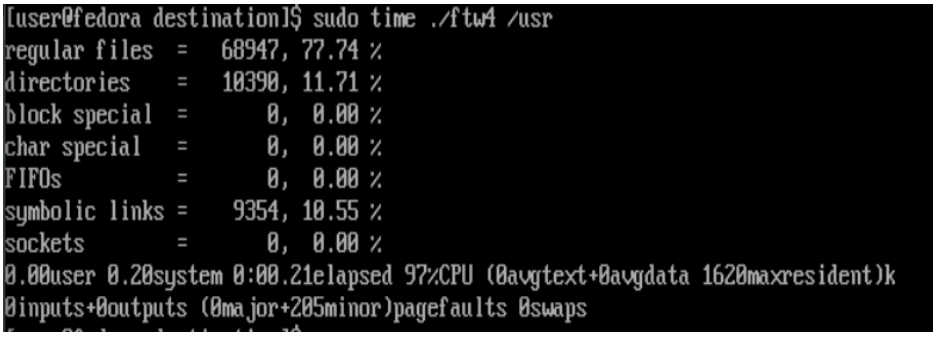
Exercițiul 3

Modificați programul `ftw4` astfel încât de fiecare dată când este întâlnit un director să se execute `chidir` la acel director, permițînd astfel utilizarea numelor de fișier și nu a căilor complete la apelul funcției `lstat`. După ce toate intrările în director au fost procesate se va executa `chdir("..")`. Comparați timpii de execuție pentru cele două variante folosind comanda de sistem `time`.

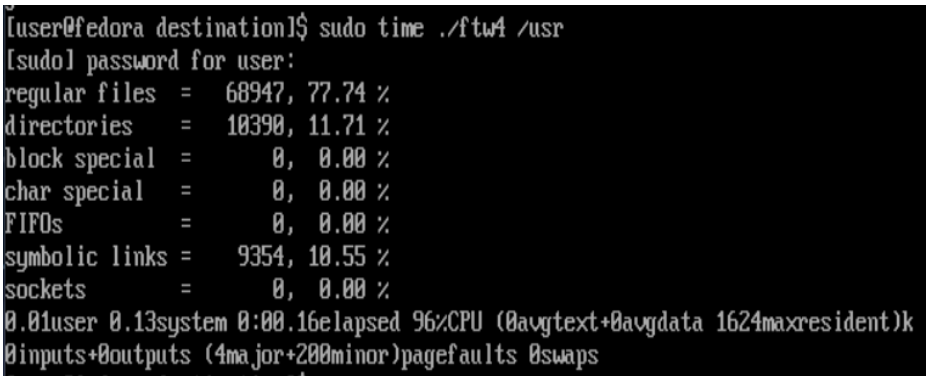
```
static int myftw(char *pathname, Myfunc *func)
{
        return dopath(func,pathname);
}

/*
 * Descend through the hierarchy, starting at "fullpath".
 * If "fullpath" is anything other than a directory, we lstat() it,
 * call func(), and return.  For a directory, we call ourself
 * recursively for each name in the directory.
 */
static int dopath(Myfunc* func,char* path)
{
        struct stat     statbuf;
        struct dirent   *dirp;
        DIR             *dp;
        int             ret;

        if (lstat(path, &statbuf) < 0)
                return func(path, &statbuf, FTW_NS);    /* stat error */

        if (S_ISDIR(statbuf.st_mode) == 0)
                return func(path, &statbuf, FTW_F);     /* not a directory */
```

```
    /*
     * It's a directory.  First call func() for the directory,
     * then process each filename in the directory.
     */

    if ( (ret = func(path, &statbuf, FTW_D)) != 0)
            return ret;

    if ( (chdir(path)) <0)
            return func(path, &statbuf, FTW_DNR);
    if((dp=opendir("."))==NULL)
    {
            if(chdir("..")<0)
                    err_ret("can't go back");
            return func(path,&statbuf,FTW_DNR);
    }

    while ( (dirp = readdir(dp)) != NULL) {
            if (strcmp(dirp->d_name, ".") == 0  ||
                strcmp(dirp->d_name, "..") == 0)
                        continue;       /* ignore dot and dot-dot  */
            if ( (ret = dopath(func,dirp->d_name)) != 0)           /* recursive */
                    break;                          /* time to leave */

    }
    if(chdir("..")<0)
            err_ret("can't go back");

    if (closedir(dp) < 0)
            err_ret("can't close directory %s", path);

    return(ret);
```

Am modificat programul ftw4.



Timpul de executare al programului inițial. (Folosind calea absolută)



Timpul de executare al programului modificat. (Folosind calea relativă)

După cum putem observa, primul program folosește calea absolută, iar în programul modificat am folosit calea relativă. Calea absolută specifică calea completă de la directorul root până la directorul/ fișierul curent (de exemplu, /myFolder/example.txt) . Calea relativă specifică calea către un fișier/director la directorul de lucru curent (de exemplu, myFolder/example.txt).

Mai sus am executat ambele programe și putem observa că programul cu calea relativă este mai rapidă decât programul cu calea absolută.