

- All resources (terminals, printers, disks, tapes, cd-roms, sound cards, network adapters) in Unix are accessed through files.
- We discuss ordinary files, i.e. the files containing data stored in a file system on a disk. It is worth noting that such files should be treated as an array of bytes.

Before any information is read or written, a file must be opened or created.

**Table 1 Basic system calls to operate on files**

<b>Function</b>	<b>Description</b>
open	open or create a file
read	read data from a file into a buffer
write	write data from a buffer to a file
dup	duplicates the file descriptor to the place pointed by the system
dup2	duplicates the file descriptor to the place pointed by the function parameter
lseek	Moves the pointer of the current position
close	close a file

## Descriptors

An open file is referenced by a non-negative integer value called *descriptor*. The descriptor is an index to process open files table. Each process owns a private descriptor table.

All read/write operations take a value of descriptor to identify an open file.

Three values of a descriptor have special meanings:

- 0 – standard input
- 1 – standard output
- 2 – standard error output

---

```
#include <fcntl.h>
```

```
int open(char * path, int flags [, int mode ] );
```

---

path                    points to the pathname of a file

flags                   way of accessing a file:

- O\_RDONLY            Open for reading only.
- O\_WRONLY            Open for writing only.
- O\_RDWR             Open for reading and writing.

Above arguments may be concatenated by “OR” with following arguments:

- O\_APPEND            If set, the seek pointer will be set to the end of the file prior to each write.
- O\_CREAT             If the file exists, this flag has no effect. Otherwise, the file is created, and the owner ID of the file is set to the effective user ID of the process.
- O\_TRUNC             If the file exists and is a regular file, and the file is successfully opened O\_RDWR or O\_WRONLY, its length is truncated to zero and the mode and owner are unchanged. O\_TRUNC has no effect on FIFO special files or directories.

mode                    access rights

RETURNS: **success** : file descriptor

**error**: -1

```
open(argv[1], O_CREAT|O_WRONLY, 0666)
```

---

```
#include <fcntl.h>
```

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

---

RETURNS: **success** : 0  
          **error**: -1

Frees descriptor referenced by `fd`. After closing, the descriptor may be reused again.

---

```
#include <fcntl.h>
```

```
int dup(int oldfd);
```

---

oldfd                    current file descriptor

RETURNS: **success** : newfd  
          **error**: -1

Duplication of oldfd. Function creates additional descriptor for an open file, which is identified by oldfd. New descriptor is created on the first free position ( it gets the minimal possible number)

---

```
#include <fcntl.h>
```

```
int dup2(int oldfd, int newfd,);
```

---

oldfd                    current file descriptor  
newfd                    new file descriptor

RETURNS: **success** : newfd  
          **error**: -1

Duplication of oldfd at a place pointed by newfd. Similarly to dup, function creates additional descriptor for an open file identified by oldfd, in the place indicated by newfd

---

```
#include <fcntl.h>
```

```
int read(int fd, char *buf, int nbyte);
```

---

fd            file descriptor

buf           the address of buffer to which the data is read (Buffer length has to be equal to or greater than nbyte )

nbyte        number to bytes to read

RETURNS: **success** : number of bytes actually read and placed in the buffer  
**error**: -1

If nbyte is not zero and read returns 0, then EOF (end of file) has been reached.

#### Example 1

---

```
char array[10];  
read(0,array,10);        /* read 10 chars from standard  
                          input */  
  
int numbers[3];  
read(0,numbers,3);      /* read only 3 bytes not integers  
                          (integers is 2 or 4 bytes long) */  
  
read(0,numbers,sizeof(numbers)*sizeof(int)); /* correct  
                                              filling of array */  
  
float *size;  
                          read(0, size, sizeof(float)); /*  
WRONG!! - read into unallocated memory */  
  
float cc;  
read(0, cc, sizeof(cc)); /* WRONG !! - second  
parameter should be pointer  
                          not value */
```

---

---

```
#include <fcntl.h>
```

```
int write(int fd, char *buf, int nbyte);
```

---

fd            descriptor that references the object to which data is to be written

buf           the address of buffer from which the data that is to be written is get

nbyte        number to bytes to write

RETURNS: **success** : number of bytes actually written

**error**: -1

### Example 2

---

```
char array[10];
```

```
write(1,array,10);     /* write 10 chars to standard  
                          output */
```

```
int number;
```

```
write(2,&number,sizeof(number));     /* write integer to  
                                          standard error, the  
                                          argument number is passed  
                                          by pointer, not by value !!  
                                          */
```

---

---

```
#include <unistd.h>
```

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

---

fd            file descriptor

offset       number of bytes that indicates how far the pointer of the position should be moved

whence       the parameter that determines the position upon which the pointer of current position is moved

RETURNS: **success** : new current pointer position counted with respect to the beginning of the file

**error:** -1

The movement of the pointer of the current position. A new position has number offset, which is calculated from the following place:

- the beginning of the file if `offset = SEEK_SET`
- the end of the file if `offset = SEEK_END`
- the current position if `offset = SEEK_CUR`

`offset < 0`       - the pointer is moved towards the beginning of file

`offset > 0`       - the pointer is moved towards the end of file

### Example 3 Writing data into file

---

```
#include <fcntl.h>

void main() {
int fd;
int n,m,i;
char buf[10];

/* open file /tmp/xx for writing */
fd = open("/tmp/xx",O_WRONLY|O_CREAT, 0644 );

/* check if open finished successful */
if (fd < 0) {
/* open failed */
exit(1);
}

/* read data from standard input */
n = read(0,buf, sizeof(buf));

/* write data info file */
m = write(fd,buf,n);

printf("Read %d, write %d   buf: %s\n",n,m,buf);

/* close descriptor */
close(fd);
}
```

---

#### Example 4 Copying files

---

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

#define MAX 512

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

char buf[MAX];
int from, to;
int nbajt;

if (argc<3){
fprintf(stderr, " Wrong nmber of parameters. Usage:\n");
fprintf(stderr, "%s <source file> <destination file>\n",
argv[0]);
exit(1);
}

from = open(argv[1], O_RDONLY);
to = open(argv[2],O_RDWR|O_CREAT, 0640);

while((nbajt = read(from, buf, MAX)) > 0)
    write(to, buf, nbajt)

close(from);
close(to) ;
exit(0);
}
```

---

#### Example 4 File size

---

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

int main(int argc, char* argv[]){
int desc;
long size;

if (argc < 2){
fprintf(stderr, "Wrong number of
parameters.Usage:\n");
fprintf(stderr, "%s <nazwa pliku>\n", argv[0]);
exit(1);
}
desc = open(argv[1], O_RDONLY);
size = lseek(desc, 0, SEEK_END);

printf("File size %s: %ld\n", argv[1], size);

close(desc);
exit(0);
}
```

## Exercises:

1. Write a program that checks whether the file which name is given as an argument is a text file. (Hint: you can use the function `isascii`)
2. Write a program that finds the longest line of file which name is given as an argument.
3. Write a program that finds the amount of baits in the longest line of file which name is given as an argument.
4. Write a program that converts small letters into capital ones in the file passed as the argument
5. Write a program that in the file which name is given as the last argument writes the content of files which names are given in the command line before the latest argument
6. Write a program that counts all words in the file which name is given as the argument. Assume, that word is a sequence of small or capital letters, numbers and underline sign. Other characters are words separators