- Experiment with open and close.
- The code carefully handles errors and interruption by signals.

Opening and Closing

open() –It associate a file descriptor with a file or physical device

```
SYNOPSIS
#include <fcntl.h>
#include <sys/stat.h>

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

- It associates a file descriptor.
- path parameter points to the pathname of the file or device,

(use absolute path begin with “/”, when you are not work in same directory

use relative path which is only file name with extension, when you are work in same directory of file.)

- oflag parameter specifies status flags and access modes for the opened file.

Opening and Closing

oflag argument

is constructed by using the bitwise OR (|) of the desired combination of the access mode and the additional flags

Access mode flags	
O_RDONLY	read-only,
O_WRONLY	write-only
O_RDWR.	read-write access

Note: exactly one of these designating read-only, write-only or read-write access can be used.

returns

success : a nonnegative integer (i.e. open file descriptor).

unsuccessful : –1 and sets errno.

Opening and Closing

Flags find in fcntl.h

Access mode flags	
O_APPEND	file offset to be moved to the end of the file before write
O_TRUNC	truncates the length of a regular file opened for writing to 0
O_CREAT	file is created if it doesn't exist Need to pass a third argument to open to designate the permissions
O_EXCL	avoid writing over an existing file if used as O_CREAT | O_EXCL
O_NOCTTY	prevents an opened device from becoming a controlling terminal
O_NONBLOCK	controls whether the open returns immediately or blocks until the device is ready

Opening and Closing

Example 4.11:

```
#include<stdio.h>
#include <fcntl.h>
#include <sys/stat.h>
extern int errno;
int main()
{
    // if file does not have in directory then file my.dat is created.
    int fd = open(“my.dat", O_RDONLY | O_CREAT);

    printf("fd = %d", fd);

}
```

Note: This code does no error checking.

Opening and Closing

Exercise 4.12

How can the call to open of Example 4.11 fail?

Answer:

- The open function returns –1 if the file doesn't exist,
- the open call was interrupted by a signal
- or the process doesn't have the appropriate access permissions.

Opening and Closing

```
#include<stdio.h>
#include <fcntl.h>
#include <sys/stat.h>
#include<errno.h>
extern int errno;
int main()
{
int fd = open("my.dat", O_RDONLY);
printf("fd = %d\n", fd);
 if (fd ==-1)
    {        // print which type of error have in a code
        printf("Error Number % d\n", errno);
         // print program detail "Success or failure"
        perror("Program");
    }
return 0;
}
```

```
o/p: if file not exist
fd = -1
Error Number  2
Program: No such file or directory
```

Opening and Closing

Example 4.13
The following code segment restarts open after a signal occurs.

```
int main()
{
     int myfd;
     while((myfd = open("my.dat", O_RDONLY)) == -1 && errno == EINTR) ;
     if (myfd == -1) /* it was a real error, not a signal */
              perror("Failed to open the file");
     else
    printf("fd = %d\n", myfd);
 return 0;

}
```

O/p
-if no file exist ----- Failed to open the file: No such file or directory
-if exist no permission----Failed to open the file: Permission denied

Opening and Closing

Exercise 4.14

How would you modify Example 4.13 to open my.dat for nonblocking read?

Answer:

Use O_RDONLY and the O_NONBLOCK flags.

myfd = open(“my.dat", O_RDONLY | O_NONBLOCK);

Opening and Closing

Each file has three classes associated with it:
- a user (or owner),
- a group and
- everybody else (others).

The possible permissions or privileges are
- read(r),
- write(w)
- execute(x).

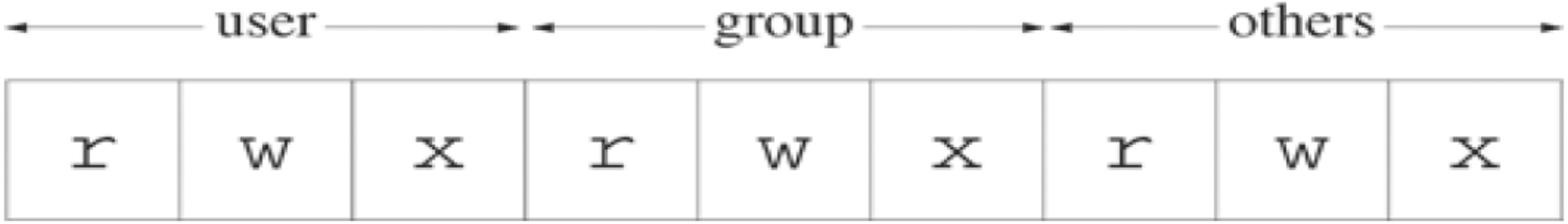
These privileges are specified separately for the user, the group and others.

When a file open with O_CREAT flag, need to specify the permissions as the third argument to open in a mask of type *mode_t.*

Opening and Closing

The file permissions were laid out in a mask of bits with 1's in designated bit positions of the mask

Figure 4.1. Historical layout of the permissions mask.



POSIX defines symbolic names for masks corresponding to the permission bits, to specify file permissions independently of the implementation.

These names are defined in sys/stat.h.

To assign desired permissions, bitwise OR is used

Table 4.1. POSIX symbolic names for file permissions.

symbol	meaning
S_IRUSR	read by owner
S_IWUSR	write by owner
S_IXUSR	execute by owner
S_IRWXU	read, write, execute by owner
S_IRGRP	read by group
S_IWGRP	write by group
S_IXGRP	execute by group
S_IRWXG	read, write, execute by group
S_IROTH	read by others
S_IWOTH	write by others
S_IXOTH	execute by others
S_IRWXO	read, write, execute by others
S_ISUID	set user ID on execution
S_ISGID	set group ID on execution

Opening and Closing

Ex: It creates a file “info.dat” in the current directory. The new file can be read or written by the user and only read by everyone else.
If the info.dat file already exists, it is overwritten.

```
int main()
{
int fd;
mode_t fdmode = (S_IRUSR | S_IWUSR | S_IRGRP | S_IROTH);
if ((fd = open("info.dat", O_RDWR | O_CREAT, fdmode)) == -1)
   perror("Failed to open info.dat");

return 0;

}
```

Opening and Closing

Program 4.9:

- It copies a source file to a destination file.
- Both filenames are passed as command line arguments (the open function for the destination file has O_CREAT | O_EXCL)
- the file copy fails if that file already exist

```
#include <fcntl.h>
#include <stdio.h>
#include <unistd.h>
#include <sys/stat.h>
#include <errno.h>
#define READ_FLAGS O_RDONLY
#define WRITE_FLAGS (O_WRONLY | O_CREAT | O_EXCL)
#define WRITE_PERMS (S_IRUSR | S_IWUSR)
#define BLKSIZE 10
```

```
int copyfile(int fromfd, int tofd) {
char *bp;
char buf[BLKSIZE];
int bytesread, byteswritten;
int totalbytes = 0;
for ( ; ; ) {
/* handle interruption by signal */
   while (((bytesread = read(fromfd, buf, BLKSIZE)) == -1) && (errno == EINTR)) ;
   if (bytesread <= 0) /* real error or end-of-file on fromfd */
              break;
   bp = buf;
   while (bytesread > 0) {
/* handle interruption by signal */
      while(((byteswritten = write(tofd, bp, bytesread)) == -1 ) && (errno == EINTR)) ;
      if (byteswritten <= 0) /* real error on tofd */
        break;
      totalbytes += byteswritten;
      bytesread -= byteswritten;
      bp += byteswritten;
   }
   if (byteswritten == -1) /* real error on tofd */
     break;
 }
 return totalbytes;
}
```

```
int main(int argc, char *argv[]) {
int bytes;
int fromfd, tofd;
if (argc != 3) {
fprintf(stderr, "Usage: %s from_file to_file\n", argv[0]);
return 1;
}
if ((fromfd = open(argv[1], READ_FLAGS)) == -1) {
perror("Failed to open input file");
return 1;
}
if ((tofd = open(argv[2], WRITE_FLAGS, WRITE_PERMS)) == -1) {
perror("Failed to create output file");
return 1;
}
bytes = copyfile(fromfd, tofd);
printf("%d bytes copied from %s to %s\n", bytes, argv[1], argv[2]);
return 0; /* the return closes the files */
}
```

o/p:
Run 1:
./a.out doc1.dat file2.doc
19 bytes copied from doc1.dat to file2.doc

Run2:
$ ./a.out doc1.dat file2.doc
Failed to create output file: File exists

Note: immediately after performing the copy and does not explicitly close the file descriptor..

Opening and Closing

Close(): To release open file descriptors

SYNOPSIS

```
#include <unistd.h>
int close(int fildes);
```

fildes: representing the open file whose resources are to be released.

returns:

0: If successful,

-1: If unsuccessful and sets errno

errno	cause
EBADF	`fildes` is not a valid file descriptor
EINTR	the `close` function was interrupted by a signal

Opening and Closing