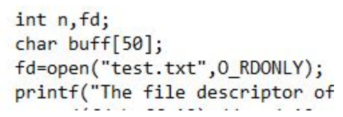
Dextutor


```
int n,fd;
char buff[50];
fd=open("test.txt",O_RDONLY);
printf("The file descriptor of
```

Program on open() system call

6 Comments / Programs / By Baljit Singh Saini

In the previous section on read/write system call we learned how to read from the standard input device and how to write to a standard output device. But, normally we would either read from a user-created file or write to a user-created file. Hence, the question comes that how do know the file descriptor of these files because to read or to write the first parameter in the read()/write() system calls is the file descriptor. We can use the *open()* system call to get the file descriptor of any file.

Syntax

```
        #include <sys/types.h>
        #include <sys/stat.h>
        #include <fcntl.h>
            int open(const char *pathname, int flags);
            int open(const char *pathname, int flags, mode_t mode);
```

The open() system call has two syntax. We discuss the first one here:

```
int open(const char *pathname, int flags);
```

The first parameter is the name of the file that you want to open for reading/writing. The second parameter is the mode in which to open the file i.e., for reading or for writing. For reading from a file, the flag used is O_RDONLY, for writing O_WRONLY and for both reading and writing O_RDWR.
Other commonly used flags are O_CREAT, O_APPEND, O_TRUNC, O_EXCL
On success, the open() system call return the file descriptor of the file. This descriptor now can be used in read()/write() system call for further processing. The value of the file descriptor will always be a positive number greater than 2. Whereas on failure it returns -1.

Program 1: Write a program using open() system call to read the first 10 characters of an existing file "test.txt" and print them on screen.

```
//open.c
#include<stdio.h>
#include<unistd.h>
#include<sys/types.h>
```

```
int n,fd;
char buff[50];
fd=open("test.txt",O_RDONLY); //opens test.txt in read mode and the file descriptor is saved in integer fd.
printf("The file descriptor of the file is: %d\n,fd); // the value of the file descriptor is printed.
n=read(fd,buff,10);//read 10 characters from the file pointed to by file descriptor fd and save them in buffer (buff)
write(1,buff,n); //write on the screen from the buffer
}
```

How it works?

First, create a file “test.txt” and write some content into it(more than 10 characters). The open() system call opens the file test.txt in read-only mode and returns the file descriptor. This file descriptor is saved in variable ‘fd’. You can print it to check the value of file dexcriptor of the file. next, use read() to read 10 characters from the file into the buffer. Finally, the buffer values are printed on the screen. The file descriptor used here is ‘0’ which is the file descriptor for standard output device.

Output

Step1: create the file test.txt and write “1234567890abcdefghij54321” into it

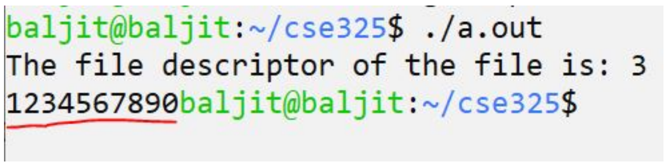
$nano test.txt

Step2: compile the program

$gcc open.c

Step3: run

$./a.out



```
baljit@baljit:~/cse325$ ./a.out
The file descriptor of the file is: 3
1234567890baljit@baljit:~/cse325$
```

Syntax 2:

The second syntax is used when the file involved does not already exist in the system and you want to create it on the go.

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

The first parameter is the file name. The second parameter is the mode (read/write). In this case apart from writing O_RDONLY, O_WRONLY and O_RDWR you also need to write O_CREAT, to create the file. The third parameter secifies the permissions on the created file (read/write/execute).
Note: O_RDONLY and O_WRONLY flags are different from read and write permission in the manner that even if a file has write permision on it it will not open in write mode until you specify the O_WRONLY mode.

Now, we extend Program 1 to read from file text.txt and write the contents into a npon existing file “towrite.txt”

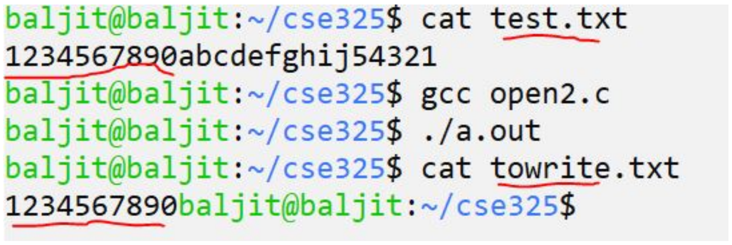
Program2: To read 10 characters from file “test.txt” and write them into non-existing file “towrite.txt”

```
//open2.c
#include<unistd.h>
#include<sys/types.h>
#include<sys/stat.h>
#include<fcntl.h>
int main()
{
write(fd1,buff,n);
}
```

How it works?

In this we open the “towrite.txt” in write mode and also use O_CREAT to create it with read and write permission for user, read for the group and write for others. The data is read from test.txt using file descriptor fd and stored in buff. It is then written from buff array into file towrite.txt using file descriptor fd1.

Output



```
baljit@baljit:~/cse325$ cat test.txt
1234567890abcdefghij54321
baljit@baljit:~/cse325$ gcc open2.c
baljit@baljit:~/cse325$ ./a.out
baljit@baljit:~/cse325$ cat towrite.txt
1234567890baljit@baljit:~/cse325$
```

Practice Program on open() system call

Q1. Write a program to read the contents of file F1 into file F2. The contents of file F2 should not get deleted or overwritten.

hint: use O_APPEND flag

Q2. Write a program using open() system call to copy the contents of one file into another file.

Solution Link

Viva Questions on open() system call

Q1. What does the open() system call returns on success?

Q2. Which system call is use to know the file descriptor of a file?

Q3. The value of the file descriptor of any user-created file is always greater than 2. Why?

Q4. What is the difference between O_APPEND and O_TRUNC flags used in open() system call?

Video

https://youtu.be/WxNSJAbQ8Ik

Relevant Programs

read()/write() system calls

lseek() system call

Understanding the fork() system call

6 thoughts on “Program on open() system call”

ปั๊มไลค์
JUNE 2, 2020 AT 8:08 AM

Like!! Great article post.Really thank you! Really Cool.

Log in to Reply

UNDER 25
OCTOBER 1, 2020 AT 12:34 AM

Saved as a favorite, I like your web site!

Log in to Reply

BALJIT SINGH SAINI
OCTOBER 1, 2020 AT 10:56 AM

Thanks.. Do share in your circle

Log in to Reply

파라오카지노
OCTOBER 17, 2020 AT 7:40 PM

Hi there to all, it’s in fact a fastidious for me to go to see this web page, it includes helpful Information.

Log in to Reply

바카라사이트
OCTOBER 20, 2020 AT 12:22 AM

Hi, i believe that i saw you visited my web site so i got here to return the favor?.I am attempting to find things to enhance my site!I guess its good enough to use a few of your ideas!!

Log in to Reply

SHELLA THON
DECEMBER 6, 2020 AT 4:44 AM

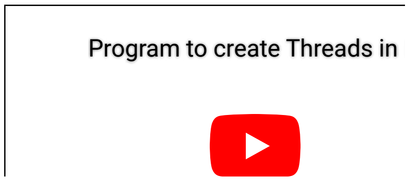
outstanding

Log in to Reply

You must be logged in to post a comment.

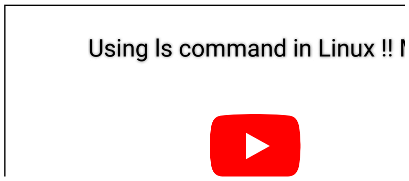
Search ...

OS Lab

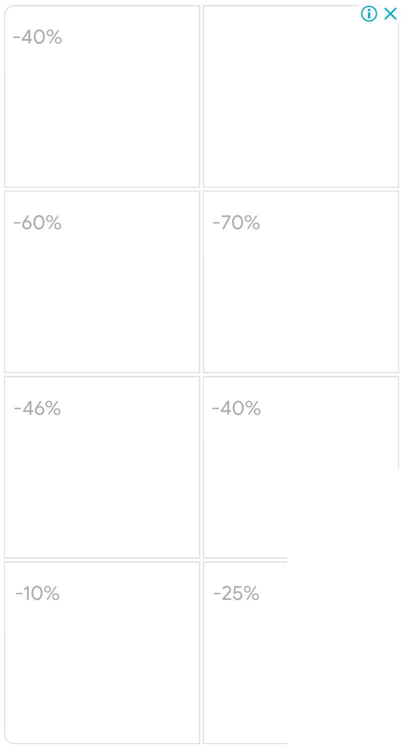


00:00 09:29

Linux Essentials Series



00:00 13:14



AJIO Sale - Get 4
AJIO.com

OS Theory

- Entity-Relationship(ER)
- Introduction to Database

- Linux
  - commands
  - Shell Scripting
- Operating System
  - CPU Scheduling
  - Deadlock
  - Disk Management
  - File Management
  - Introduction
  - IPC
  - Memory Management
  - Practice Problems
  - Process
  - Process Synchronization
  - System calls
  - Thread