C Programming

C fcntl function usage

4 months ago • by Aqsa Yasin

Поскольку имя указывает на то, что fcntl сокращенно называется элементом управления "файл’. Это означает, что он основан на процессе обработки файлов. fcntl - это системный вызов. Это позволяет программе устанавливать блокировку чтения или записи. Эта функция может использоваться для изменения свойств файла, которые либо уже открыты, либо могут быть открыты с помощью любого действия, примененного к нему. Это универсальная функция, которая используется для изменения файлов многими способами, такими как открытие, чтение и запись и т. Д. Эта статья посвящена функциям управления файлами.

Fcntl in Linux

In the Linux operating system, the fcntl call is done through the descriptors. For instance, a read lock is placed on a readable file descriptor, and a similar case is for the write lock. A file descriptor represents the file number that is opened. It is convenient for the program to remember which file it is working on. When we open a file, the number that is not assigned

already and is free is given to the file in the descriptor table of the processes file. And in the case of closing a file, that assigned number is removed from the process's descriptor table.

Syntax


```
#include <fcntl.h>

int fcntl (целочисленный дескриптор, целочисленный cmd)
```

Во-первых, мы определяем библиотеку fcntl, чтобы процесс был легко выполнен. Вызов функции в основном содержит два аргумента в параметре. Одним из них является

дескриптор, как определено выше; он определяет файл, к которому должна быть применена команда управления. Другими словами, на которых атрибуты должны быть изменены. Вторая - это команда, которая применяется к указанным дескрипторам.

Как fcntl изменяет свойства файла

Функция Fcntl используется для пяти различных целей, в основном включая дублирование, установку флагов и т. Д.; Каждый из них подробно описан здесь.

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

int main(void){

    printf("duplicate an existing file descriptor (F_DUPFD) %d\n", F_DUPFD);
    printf("get file descriptor flags (F_GETFD) %d\n", F_GETFD);
    printf("set file descriptor flags (F_SETFD) %d\n", F_SETFD);
    printf("get file status flags (F_GETFL) %d\n", F_GETFL);
    printf("set file status flags (F_SETFL) %d\n", F_SETFL);
    printf("get record locks (F_GETLK) %d\n", F_GETLK);
    printf("set record locks (F_SETLK) %d\n", F_SETLK);
    printf("get asynchronous I/O ownership (F_GETOWN) %d\n", F_GETOWN);
    printf("set asynchronous I/O ownership (F_SETOWN) %d\n", F_SETOWN);
```

Cmd = F_DUPFD

Дублируйте дескриптор файлов. Новое дублированное значение возвращается функции. Это самое низкое значение, которое еще не открыто и не присвоено какому-либо другому дескриптору. Он всегда принимается как целое число, и значение всегда больше третьего аргумента. Кроме того, дублированное значение имеет свои флаги

файлового дескриптора. Новый дескриптор имеет ту же запись в таблице, что и исходный дескриптор.

Cmd = GETFD

Эта функция предназначена для возврата флагов дескриптора в качестве значения функции. Как следует из названия, мы получаем флаг после его установки.

Cmd = SETFD

Как и получение флага, эта функция используется для установки флага дескриптора. Программа установила флаг либо на 0, не закрывать при exec, либо на 1, чтобы закрыть при exec.

Cmd = F_GETFL

Эта функция возвращает флаги для состояния файла в качестве значения функции. Когда статус флага описывается как открытый флаг, то мы описываем флаги статуса.

Cmd = F_SETFL

It sets the status flag to the file. As GETFL is used to return the file status.

Cmd = F_GETOWN

This function is related to the process identity as it returns the process ID and the process group id.

Cmd = F_SETOWN

This function tends to create and set process id or group process id.

Возвращаемое значение из fcntl зависит от используемой команды. Если команды сталкиваются с ошибкой, она возвращает -1. Если каждая функция не испытывает проблем, то возвращается любое другое значение, кроме -1 . Принимая во внимание, что в случае F_GETOWN возвращаемый идентификатор может быть положительным или отрицательным значением.

Теперь мы добавим здесь несколько элементарных примеров. Для реализации кодов fcntl вам необходимо иметь текстовый редактор и терминал Linux, чтобы получить результирующее значение.

Пример 1

Рассмотрим пример, в котором мы создали, а затем записали новую строку в текстовый файл образца. В этом примере не будет использоваться участие fcntl в качестве функции. Функции этой функции будут реализованы в коде только с помощью библиотеки.

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

Мы напишем строку в код, поэтому нам нужно использовать библиотеку строк. Здесь мы взяли массив файловых дескрипторов. Кроме того, был взят массив символов, который непосредственно инициализируется строкой из нескольких символов. С помощью файлового дескриптора мы будем использовать некоторые файловые операции состояния флага файла, такие как чтение и запись в файл и т. Д. Если файл уже создан, то нужно открыть его и записать в него строку.

```
Fd[0] = открыть("sample.txt ", O_RDWR);
```

Этот оператор откроет файл с именем ‘sample.txt - используя флаг O_RDWR. Теперь, чтобы добавить строку, определенную ранее, она будет введена в файл.

```
Write(fd[0], Bf1, strlen (buf1));
```

С помощью опции read строка будет отображаться при выполнении кода из файла. Оба файловых дескриптора закрыты в конце.

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

int main (void)
{
    int fd[2];
    char buf1[22] = "Linux hint website";
    char buf2[12];

    // assume sample.txt is already created
    fd[0] = open("sample.txt", O_RDWR);
    fd[1] = open("sample.txt", O_RDWR);

    write(fd[0], buf1, strlen(buf1));
    write(1, buf2, read(fd[1], buf2, 22));

    close(fd[0]);
    close(fd[1]);

    return 0;
}
```

После сохранения файла мы будем использовать компилятор GCC для выполнения файла.

файл $ gcc –o file .c

$ ./file

```
aqsa@aqsa-VirtualBox:~$ gcc -o file file.c
aqsa@aqsa-VirtualBox:~$ ./file
Linux hint websiteaqsa@aqsa-VirtualBox:~$
```

Когда код будет выполнен, вы увидите, что отображается строка, которую мы записали в коде в файл. Это утверждение было извлечено из файла. Когда вы перейдете к файлам в Ubuntu, вы увидите sample.txt файл. Вы заметите, что строка записывается в файл через код при открытии файла.

Пример 2

Это пример команды F_GETFL. Он возвращает флаг состояния файла в качестве значения функции. Во-первых, файл будет открыт; если он еще не создан, то O_CREAT создаст файл; во-первых, все эти функции состояния флага указаны ниже, чтобы было легко понять.

O_RDONLY: Эта функция открывает файл только для чтения.

O_WRONLY: имеет дело только с целью записи.

O_RDWR: Это также для процесса чтения и записи.

O_APPEND: добавляется к каждой функции записи в текущем файле.

Это основной флаг, который используется в примере. Возвращаясь к примеру, если возвращаемое значение при открытии файла меньше 0, то выводится сообщение об ошибке.

```c
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <fcntl.h>
#include <sys/types.h>
int main(void){

    int fflags, rVal, fd;

    if((fd = open("file1", O_CREAT | O_APPEND, S_IRWXU))<0){
        printf("error opening file.\n");
        exit(EXIT_FAILURE);
    } else {
        printf("File opened.\n");
    }

    if((rVal = fcntl(fd, F_GETFL))<0){
        printf("Error getting file status flags.\n");
        exit(EXIT_FAILURE);
    } else {
        printf("File status flags retrieved.\n");
    }

```

После открытия необходимо получить файл; если состояние файла равно -1, то выводится сообщение об ошибке; в противном случае статус файла будет получен. Теперь с помощью режима доступа мы получим флаг файлового дескриптора. Все параметры будут проверены с помощью оператора if-else. Опция в соответствии со

статусом файла будет выбрана соответственно. В конце концов, если состояние дескриптора файла больше 1, это означает, что файл теперь закрыт.

```
    //O_ACCMODE has the value of three
    fflags = rVal & O_ACCMODE;
    if(fflags == O_RDONLY){
        printf("File is read only.\n");
    } else if (fflags == O_WRONLY){
        printf("File is write only.\n");
    } else if (fflags == O_RDWR){
        printf("File is read / write.\n");
    } else {
        printf("No idea.\n");
    }


    if(close(fd)<0){
        printf("Error closing file.\n");
        exit(EXIT_FAILURE);
    } else {
        printf("File descriptor closed.\n");
    }


    return 0;
}
```

Скомпилируйте код; вы увидите, что все операторы отображаются в соответствии со статусом fd либо файл открыт, либо закрыт.

```
aqsa@aqsa-VirtualBox:~$ gcc -o file file.c
aqsa@aqsa-VirtualBox:~$ ./file
File opened.
File status flags retrieved.
File is read only.
File descriptor closed.
aqsa@aqsa-VirtualBox:~$
```

Заключение

Статья ‘C: использование функции Fcntl’ содержит особенности функции fcntl. Файловый дескриптор играет жизненно важную роль в процессе обработки файлов относительно любой функции. Мы также использовали некоторые примеры флагов в случае чтения и записи в файл. Оба примера, объясненные здесь, будут полезны для вас при использовании Fcntl на языке программирования C.

ОБ АВТОРЕ

Акса Ясин

Я целеустремленный специалист в области информационных технологий со страстью к писательству. Я технический писатель и люблю писать для всех версий Linux и Windows.

Просмотреть все сообщения

RELATED LINUX HINT POSTS

- Arithmetic Shift and Logical Shift in C
- C Uppercase to Lowercase
- How to bit flip in C
- How to reverse an array in C
- String to integer in C
- How to Debug Segmentation Faults in C?
- Strchr Metohd in C

Linux Hint LLC, editor@linuxhint.com
1309 S Mary Ave Suite 210, Саннивейл, Калифорния
94087