
Міністерство освіти і науки України
НТУУ «Київський політехнічний інститут»
Фізико-технічний інститут

# ОПЕРАЦІЙНІ СИСТЕМИ

## Комп’ютерний практикум №10

Варіант №11

**Виконав:**

Студент 2 курсу ФТІ
Групи ФІ-92
Поночевний Назар Юрійович

**Перевірив:**

Ільїн Костянтин Іванович

Київ – 2021

# Робота №10. Інтерфейс файлової системи в ОС Linux

Варіант №11

**Мета:** Ознайомитися з реалізацією файлових систем в Linux і основними структурами даних, що використовуються віртуальною файловою системою (VFS). Дослідити механізм доступу до файлів через інтерфейс віртуальної файлової системи в Linux.

**Завдання для самостійної підготовки**

1. Ознайомитись з документацією VFS для ОС Linux. Звернути увагу на архітектуру системи, структури даних, що використовує ця система.

2. Ознайомитись з правилами і прикладами використання функцій (системних викликів) для роботи з файловою системою (перелік функцій див. нижче у розділі Довідковий матеріал):

- man pages;
- книги з числа рекомендованих, зокрема [1, розд. 4.3.7], [5, розд. 13];
- стаття [17];
- інші джерела.

**Завдання до виконання**

1. Спочатку виконати завдання для самостійної підговки, тобто опанувати теорію.

2. Ознайомитись із завданням до лабораторної роботи (згідно варіанту, наданого викладачем). Варіант 11 (1)

Процес відкриває N файлів, що реально існують на диску або є новостворениими. Розробити програму, яка демонструвала б динаміку формування таблиці дескрипторів файлів і зміни інформації в її елементах (при зміні інформації в файлах). Наприклад, сценарій програми може бути таким:

- відкриття першого призначеного для користувача файлу;
- відкриття другого призначеного для користувача файлу;

● відкриття третього призначеного для користувача файлу;

● зміна розміру третього файлу до нульової довжини;

● копіювання другого файлу в третій файл.

Після кожного з етапів друкується таблиця дескрипторів файлів для всіх відкритих файлів.

```cpp
#include <stdlib.h>
#include <stdio.h>
#include <unistd.h>
#include <string>
#include <iostream>
#include <fstream>

using namespace std;

int main(int argc, char *argv[]) {
    if (argc != 2) {
        printf("USAGE: ./Lab10 start\n");
        return 1;
    }

    string line;
    fstream file1, file2, file3;

    system("ls -la /proc/$$/fd");

    file1.open("file1.txt", ios::out);
    printf("\nOpened file for write: %s\n\n", "file1.txt");

    system("ls -la /proc/$$/fd");

    file2.open("file2.txt", ios::in);
    printf("\nOpened file for read: %s\n\n", "file2.txt");

    system("ls -la /proc/$$/fd");

    file3.open("file3.txt", ios::in);
    printf("\nOpened file for read: %s\n\n", "file3.txt");

    system("ls -la /proc/$$/fd");

    file3.close();
    file3.open("file3.txt", ios::out);
    printf("\nChanged file size to zero: %s\n\n", "file3.txt");

    system("ls -la /proc/$$/fd");

    while (getline(file2, line)) {
        file3 << line << '\n';
    }
    printf("\nCopied file2.txt to %s\n\n", "file3.txt");

    system("ls -la /proc/$$/fd");

    file1.close();
    file2.close();
    file3.close();

    printf("\nClose files and exit the program.\n");
    return 0;
}
```

```
nazar11@ubuntu:~/lab_8$ dir
file2.txt  file3.txt  lab_10  lab_10.cpp
nazar11@ubuntu:~/lab_8$ cat file2.txt
It is multi
line text from
the second file!

nazar11@ubuntu:~/lab_8$ cat file3.txt
Some
Temp
text here!
nazar11@ubuntu:~/lab_8$ ./lab_10 start
total 0
dr-x------ 2 nazar11 nazar11  0 Jun  4 12:49 .
dr-xr-xr-x 9 nazar11 nazar11  0 Jun  4 12:49 ..
lrwx------ 1 nazar11 nazar11 64 Jun  4 12:49 0 -> /dev/pts/0
lrwx------ 1 nazar11 nazar11 64 Jun  4 12:49 1 -> /dev/pts/0
lrwx------ 1 nazar11 nazar11 64 Jun  4 12:49 2 -> /dev/pts/0

Opened file for write: file1.txt

total 0
dr-x------ 2 nazar11 nazar11  0 Jun  4 12:49 .
dr-xr-xr-x 9 nazar11 nazar11  0 Jun  4 12:49 ..
lrwx------ 1 nazar11 nazar11 64 Jun  4 12:49 0 -> /dev/pts/0
lrwx------ 1 nazar11 nazar11 64 Jun  4 12:49 1 -> /dev/pts/0
lrwx------ 1 nazar11 nazar11 64 Jun  4 12:49 2 -> /dev/pts/0
l-wx------ 1 nazar11 nazar11 64 Jun  4 12:49 3 -> /home/nazar11/lab_8/file1.txt

Opened file for read: file2.txt

total 0
dr-x------ 2 nazar11 nazar11  0 Jun  4 12:49 .
dr-xr-xr-x 9 nazar11 nazar11  0 Jun  4 12:49 ..
lrwx------ 1 nazar11 nazar11 64 Jun  4 12:49 0 -> /dev/pts/0
lrwx------ 1 nazar11 nazar11 64 Jun  4 12:49 1 -> /dev/pts/0
lrwx------ 1 nazar11 nazar11 64 Jun  4 12:49 2 -> /dev/pts/0
l-wx------ 1 nazar11 nazar11 64 Jun  4 12:49 3 -> /home/nazar11/lab_8/file1.txt
lr-x------ 1 nazar11 nazar11 64 Jun  4 12:49 4 -> /home/nazar11/lab_8/file2.txt
```

```
Opened file for read: file3.txt

total 0
dr-x------ 2 nazar11 nazar11  0 Jun  4 12:49 .
dr-xr-xr-x 9 nazar11 nazar11  0 Jun  4 12:49 ..
lrwx------ 1 nazar11 nazar11 64 Jun  4 12:49 0 -> /dev/pts/0
lrwx------ 1 nazar11 nazar11 64 Jun  4 12:49 1 -> /dev/pts/0
lrwx------ 1 nazar11 nazar11 64 Jun  4 12:49 2 -> /dev/pts/0
l-wx------ 1 nazar11 nazar11 64 Jun  4 12:49 3 -> /home/nazar11/lab_8/file1.txt
lr-x------ 1 nazar11 nazar11 64 Jun  4 12:49 4 -> /home/nazar11/lab_8/file2.txt
lr-x------ 1 nazar11 nazar11 64 Jun  4 12:49 5 -> /home/nazar11/lab_8/file3.txt

Changed file size to zero: file3.txt

total 0
dr-x------ 2 nazar11 nazar11  0 Jun  4 12:49 .
dr-xr-xr-x 9 nazar11 nazar11  0 Jun  4 12:49 ..
lrwx------ 1 nazar11 nazar11 64 Jun  4 12:49 0 -> /dev/pts/0
lrwx------ 1 nazar11 nazar11 64 Jun  4 12:49 1 -> /dev/pts/0
lrwx------ 1 nazar11 nazar11 64 Jun  4 12:49 2 -> /dev/pts/0
l-wx------ 1 nazar11 nazar11 64 Jun  4 12:49 3 -> /home/nazar11/lab_8/file1.txt
lr-x------ 1 nazar11 nazar11 64 Jun  4 12:49 4 -> /home/nazar11/lab_8/file2.txt
l-wx------ 1 nazar11 nazar11 64 Jun  4 12:49 5 -> /home/nazar11/lab_8/file3.txt

Copied file2.txt to file3.txt

total 0
dr-x------ 2 nazar11 nazar11  0 Jun  4 12:49 .
dr-xr-xr-x 9 nazar11 nazar11  0 Jun  4 12:49 ..
lrwx------ 1 nazar11 nazar11 64 Jun  4 12:49 0 -> /dev/pts/0
lrwx------ 1 nazar11 nazar11 64 Jun  4 12:49 1 -> /dev/pts/0
lrwx------ 1 nazar11 nazar11 64 Jun  4 12:49 2 -> /dev/pts/0
l-wx------ 1 nazar11 nazar11 64 Jun  4 12:49 3 -> /home/nazar11/lab_8/file1.txt
lr-x------ 1 nazar11 nazar11 64 Jun  4 12:49 4 -> /home/nazar11/lab_8/file2.txt
l-wx------ 1 nazar11 nazar11 64 Jun  4 12:49 5 -> /home/nazar11/lab_8/file3.txt

Close files and exit the program.
nazar11@ubuntu:~/lab_8$ dir
```

```
nazar11@ubuntu:~/lab_8$ dir
file1.txt  file2.txt  file3.txt  lab_10  lab_10.cpp
nazar11@ubuntu:~/lab_8$ cat file1.txt
nazar11@ubuntu:~/lab_8$ cat file2.txt
It is multi
line text from
the second file!

nazar11@ubuntu:~/lab_8$ cat file3.txt
It is multi
line text from
the second file!

nazar11@ubuntu:~/lab_8$
```

**Контрольні запитання**

1. Яка структура дескрипторів файлів, таблиці відкритих файлів, таблиці відкритих файлів процесу?

Файловий дескриптор має структуру невід'ємного цілого числа, що повертається системними викликами, такими як create(), open() або pipe(). Після отримання файлового дескриптора процес може використовувати його для подальшої роботи з файлом, наприклад за допомогою системних викликів read(), write() або close(). Кожен елемент file таблиці відкритих файлів містить інформацію про режим відкриття файлу, специфікований при відкритті файлу, а також інформацію про становище покажчика читання-запису. При кожному відкритті файлу в таблиці відкритих файлів з'являється новий елемент file.

2. Яким є ланцюжок відповідності дескриптора файлу, відкритого процесом, і файлом на диску?

Один і той самий файл на диску ОС Linux може бути відкритий декількома не пов'язаними один з одним процесами, при цьому йому буде відповідати один елемент таблиці дескрипторів файлів inode і стільки елементів таблиці відкритих файлів file, скільки разів цей файл був відкритий.

3. Опишіть функціональну структуру операції введення-виведення (пули, асоціація їх з драйверами, способи передачі інформації і т.д.).

Спочатку іде взаємодія з самим пристроєм (клавіатура, принтер, ін.), у пристроя є так званий контролер, який зчитує дані в буфер і за допомогою драйверів пристрою передає буфер до ядра ОС. Ядро в свою чергу за допомогою дескрипторів створює необхідні потоки, виділяє пам’ять у вигляді файлу і надає доступ певним програмам на читання чи запис у цей файл.

4. Яким чином здійснюється підтримка пристроїв введення-виведення в ОС Linux?

В ОС Linux є файлові пристрої, які працюють з файловою системою, таблицями дескрипторів, драйверами та іншими елементами ОС для того щоб легко, як зі звичайними файлами, можна було виконувати дії над пристроями.

5. Яка структура таблиці відкритих файлів і масиву файлових дескрипторів процесу після відкриття файлу?

Кожен елемент масиву файлових дескрипторів процесу (fd) містить покажчик місця розташування відповідного елементу таблиці відкритих файлів (file), який у свою чергу містить посилання на елемент таблиці дескрипторів файлу (inode). В реалізації VFS у Linux це посилання насправді здійснюється через об'єкти елементів каталогу (dentry), впроваджені для кешування часто використовуваних елементів каталогів.

6. Яка структура таблиці відкритих файлів і масиву файлових дескрипторів процесу після закриття файлу?

Після закриття файлу видаляється елемент таблиці дескрипторів файлу (inode) та звільняється покажчик місця розташування відповідного елементу таблиці відкритих файлів (file), щоб будь-який наступний відкритий файл міг його зайняти.

7. Яка структура таблиці відкритих файлів і масиву файлових дескрипторів процесу після створення каналу?

Оскільки ядро реалізує канали всередині файлової системи і оскільки канал не існує до того, як його будуть використовувати, ядро має при створенні каналу призначити йому індекс. Воно також призначає для каналу пару користувальницьких дескрипторів і відповідні їм записи в таблиці файлів: один з дескрипторів для читання з каналу, а інший для запису в канал. Оскільки ядро користується таблицею файлів, інтерфейс для виклику функцій read, write і ін. узгоджується з інтерфейсом для звичайних файлів. В результаті процес немає потреби знати, чи веде він читання або запис в звичайний файл або в канал.

8. Яка структура таблиці відкритих файлів і масиву файлових дескрипторів процесу після створення нового процесу?

Кожному процесу в ОС Linux відразу після породження ставиться у відповідність масив файлових дескрипторів процесу. Якщо, в свою чергу, зазначений процес породжує новий процес, наприклад, за допомогою системного виклику fork(), то процесу-нащадку ставиться у відповідність масив файлових дескрипторів процесу, який в перший момент функціонування процесу-нащадка є копією масиву файлових дескрипторів процесу-предка.

**Висновок**

ОС Linux має гнучку та ефективну організацію файлової системи, що дозволяє швидко, легко та безпечно отримати доступ до конкретного файлу, виконати над ним певні дії та закрити. Все це можливо через певну інкапсуляцію відкритих файлів у вигляді файлових дескрипторів всередині таблиці відкритих файлів кожного окремого процесу. Також, мені дуже подобається, що в ОС Linux доступ до каналів та пристроїв виконується за тим же інтерфейсом, як і до звичайних файлів.