**Робота №10. Інтерфейс файлової системи в ОС Linux**

### 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.

### Довідковий матеріал

Тут наведено лише мінімальну інформацію, достатню хіба що для того, щоби зрозуміти, про що йде мова. Решту інформації необхідно здобути з джерел, названих вище.

Назвемо системні функції, що забезпечують звернення до існуючих файлів, такі як open(), read(), write(), lseek() і close(), потім функції створення нових файлів, а саме, creat() і mknod(), і, нарешті, функції для роботи з індексними дескрипторами (i-node) або для пересування по файловій системі: chdir(), chroot(), chown(), stat() і fstat().

Більш складні системні функції: pipe() і dup() — мають важливе значення для реалізації каналів в shell; mount() і umount() розширюють видиме для користувача дерево файлових систем; link() і unlink() змінюють ієрархічну структуру файлової системи.

### Функції для роботи з файловою системою

Функції для роботи з файловою системою і їх зв'язок з іншими алгоритмами можна звести у такі таблиці:

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Повертають де-  скриптори файлу | Використовують алгоритм namei | Призначають індексні дескриптори | Працюють з атрибутами файлу | Здійснюють введення-  виведення з файлу | Працюють  зі  структурою  файлових систем | Здійснюють керування деревами |
| open creat | open stat creat link | creat  mknod | chown  chmod | read  write | mount  umount | chdir  chown |

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| dup  pipe close | chroot unlink chdir mknod  chown mount chmod umount | link  unlink | | | stat | lseek | |  | |  | |
| Алгоритми роботи з файловою системою на нижньому рівні | | | | | | | | | | |  |
| namei | | | | ialloc ifree | | | alloc free | | | | bmap |
| iget iput | | | |
|  | | | Алгоритми роботи з буферами | | | | | |  | | |
| getblk | | | brelse bread breada | | | | | | bwrite | | |

Системні функції класифікують на кілька категорій, хоча деякі з функцій присутні більш, ніж в одній категорії:

1. Системні функції, які повертають дескриптори файлів для використання іншими системними функціями.
2. Системні функції, що використовують алгоритм namei для аналізу імені шляху пошуку.
3. Системні функції, які призначають і звільняють індексний дескриптор з використанням алгоритмів ialloc та ifree.
4. Системні функції, які встановлюють або змінюють атрибути файлу.
5. Системні функції, що дозволяють процесу проводити введення-виведення з використанням алгоритмів alloc, free і алгоритмів виділення буфера.
6. Системні функції, які змінюють структуру файлової системи.
7. Системні функції, що дозволяють процесу змінювати власне уявлення про структуру дерева файлової системи.

Виклик системної функції open() (відкрити файл) — це перший крок, який повинен зробити процес, щоб звернутися до даних у файлі.

Для читання з файлу використовується функція read(), а для запису в файл write(). Звичайне використання системних функцій read() і write() забезпечує послідовний доступ до файлу, однак процеси можуть використовувати виклик системної функції lseek() для зазначення місця у файлі, де буде проводитися введення-виведення, а також здійснення довільного доступу до файлу.

Процес закриває відкритий файл, коли процесу більше не потрібно звертатися до нього, за допомогою функції close().

Системна функція open() дає процесу доступ до файлу, що вже існує. Системна функція creat() створює в системі новий файл. Системна функція mknod() створює в системі спеціальні файли, в число яких входять іменовані канали, файли пристроїв і каталоги.

Зміна власника або режиму (прав) доступу до файлу є операцією, яка здійснюється над індексним дескриптором, а не над файлом. Це робиться операціями chown() і chmod(), відповідно.

Системні функції stat() і fstat() дозволяють процесам запитувати інформацію про статус файлу: тип файлу, власника файлу, права доступу, розмір файлу, кількість зв'язків, номер індексного дскриптора і час доступу до файлу.

Системна функція dup() копіює дескриптор файлу в перше вільне місце в таблиці користувальницьких дескрипторів файлу, повертаючи новий дескриптор користувачеві. Вона діє для всіх типів файлів.

Системна функція mount() пов'язує файлову систему із зазначеного розділу на диску з існуючою ієрархією файлових систем, а функція umount() вилучає файлову систему з ієрархії. Функція mount(), таким чином, дає можливість звертатися до даних в дисковому розділі як до файлової системи, а не як до послідовності дискових блоків.

Системна функція link() пов'язує файл з новим ім'ям в структурі каталогів файлової системи, створюючи для існуючого індексного дескриптора новий запис у каталозі.

Системна функція unlink() видаляє з каталогу точку входу для файлу.

Наявність віртуальної файлової системи дає ядру можливість підтримувати одночасно множину файлових систем, таких як мережні файлові системи або файлові системи з інших операційних систем. Процеси користуються для звернення до файлів звичайними функціями системи, а ядро встановлює відповідність між загальним набором файлових операцій і операціями, специфічними для кожного типу файлової системи.

### Структури даних ядра (системні таблиці)

При здійсненні операцій введення-виведення в файл, специфікований призначеним для користувача дескриптором файлу fd, ОС Linux ставить у відповідність використовуваному системному виклику послідовність програмних запитів до апаратури комп'ютера за допомогою цілого ряду пов'язаних наборів даних, структура яких підтримується самою ОС Linux, її файлової системою і системою керування введеннямвиведенням.

Основною із згаданих структур можна вважати таблицю дескрипторів файлів. Таблиця дескрипторів файлів являє собою структуру даних, що зберігається в оперативній пам'яті комп'ютера, елементами якої є копії дескрипторів файлів inode, по одній на кожен файл ОС Linux, до якого була здійснена спроба доступу. При виконанні операції відкриття файлу в ОС Linux спочатку по повному імені файлу визначається елемент каталогу, де в полі імені міститься ім'я файлу, для якого проводиться операція відкриття файлу. У знайденому елементі каталогу з поля посилання витягується порядковий номер дескриптора файлу (inode). Потім дескриптор файлу з відповідним номером копіюється в оперативну пам'ять, в її область, яка називається таблицею дескрипторів файлів (якщо він до цього там був відсутній).

З таблицею дескрипторів файлів тісно пов'язана інша структура даних, яка називається таблицею відкритих файлів. 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.

Один і той самий файл 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. Однак з цього правила є один виняток, який стосується випадку, коли файл, відкритий процесом, потім відкривається процесом-нащадком, породженим за допомогою системного виклику fork(). При виникненні такої ситуації операції відкриття файлу, здійсненої процесом-нащадком, буде поставлений у відповідність той з існуючих елементів таблиці відкритих файлів (в тому числі положення покажчика читання-запису), який свого часу був поставлений у відповідність операції відкриття цього файлу, здійсненої процесом-предком.

Третій набір даних називається масивом файлових дескрипторів процесу (fd). 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.

В результаті 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.

#### 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 1

Процес відкриває N файлів, що реально існують на диску або є новоствореними. 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.

### Варіант 2

Процес створив новий файл і перепризначив на нього стандартний потік виведення. Розробити програму, яка демонструвала б динаміку створення таблиць, пов'язаних з цією подією (таблиця відкритих файлів, масив файлових дескрипторів процесу). Наприклад, сценарій програми може бути таким:

 неявне відкриття стандартного файлу введення;

 неявне відкриття стандартного файлу виведення;

 неявне відкриття стандартного файлу виведення помилок;

 відкриття призначеного для користувача файлу;

 закриття стандартного файлу введення (моделювання close (0));

 отримання копії дескриптора призначеного для користувача файлу (моделювання dup(fd), де fd — дескриптор призначеного для користувача файлу);

 закриття призначеного для користувача файлу (моделювання close(fd), де fd — дескриптор призначеного для користувача файлу).

Після кожного з етапів друкуються таблиця описателей файлів, таблиця файлів, таблиця відкритих файлів процесів.

### Варіант 3

Нехай два процеси здійснюють доступ до одного і того ж файлу, але один з них читає файл, а інший пише в нього. Настає момент, коли обидва процеси звертаються до одного й того ж блоку диска. Нехай деяка гіпотетична ОС використовує ту ж механіку керування введенням-виведенням, що і ОС UNIX, але не дозволяє, як в ситуації, описаній вище, звертатися до одного блоку файлу.

Розробити програму, яка демонструє "заморожування" переміщення покажчика читання-запису одного з процесів до тих пір, поки покажчик другого процесу знаходиться в цьому блоці. Показати динаміку створення всіх таблиць, пов'язаних з файлами і процесами, і зміну їх вмісту.

Після кожного з етапів друкуються таблиці відкритих файлів і масиви дескрипторів файлів процесу обома процесами.

### Варіант 4

Нехай N процесів здійснюють доступ до одного і того ж файлу на диску (але з різними режимами доступу). Розробити програму, яка демонструвала б динаміку формування таблиці відкритих файлів і зміни її елементів (при переміщенні покажчиків читання-запису, наприклад). Наприклад, сценарій програми може бути таким:

 відкриття файлу процесом 0 для читання;

 відкриття файлу процесом 1 для запису;

 відкриття файлу процесом 2 для додавання;

 читання зазначеного числа байт файлу процесом 0;

 запис зазначеного числа байт в файл процесом 1;  додавання вказаного числа байт в файл процесом 2.

Після кожного з етапів друкуються таблиці файлів всіх процесів.

### Варіант 5

Розробити програму, яка демонструвала б роботу ОС Linux при відкритті файлу процесом і читанні-записи в нього. При цьому досить показати тільки динаміку створення таблиць, пов'язаних з цією подією (таблиця дескрипторів файлу, таблиця відкритих файлів, масив файлових дескрипторів процесу). Наприклад, сценарій програми може бути таким:

 неявне відкриття стандартного файлу введення;

 неявне відкриття стандартного файлу виведення;

 неявне відкриття стандартного файлу виведення помилок;

 відкриття першого призначеного для користувача файлу;

 відкриття другого призначеного для користувача файлу;

 записування 20 байт в перший файл;  зчитування 15 байт з другого файлу;  записування 45 байт в перший файл.

Після кожного з етапів друкуються таблиця дескрипторів файлів, таблиця відкритих файлів, таблиця відкритих файлів процесів.

### Варіант 6

Розробить програму, яка демонструвала б роботу ОС Linux при відкритті файлу процесом. При цьому досить показати тільки динаміку створення таблиць, пов'язаних з цією подією (таблиця дескрипторів файлів, таблиця відкритих файлів, масив файлових дескрипторів процесу). Наприклад, сценарій програми може бути таким:

 неявне відкриття стандартного файлу введення;

 неявне відкриття стандартного файлу виведення;

 неявне відкриття стандартного файлу виведення помилок;

 відкриття першого призначеного для користувача файлу;

 відкриття другого призначеного для користувача файлу;  відкриття третього призначеного для користувача файлу.

Після кожного з етапів друкуються таблиця дескрипторів файлів, таблиця відкритих файлів, таблиця відкритих файлів процесів.

### Варіант 7

Нехай кожен з N процесів здійснює доступ до Pi файлів (i = 1..N). Далі нехай M<N процесів породили процеси-нащадки (за допомогою системного виклику fork ()) і серед цих нащадків K<M процесів додатково відкрили ще Sj файлів (j = 1..K). Розробити програму, яка демонструвала б динаміку формування масивів файлових дескрипторів процесів. Наприклад, сценарій програми може бути таким:

 процес 0 відкриває два файли (загальне число відкритих файлів, включаючи стандартні файли, дорівнює п'яти);

 процес 1 відкриває два файли (загальне число відкритих файлів, включаючи стандартні файли, дорівнює п'яти);

 процес 2 відкриває два файли (загальне число відкритих файлів, включаючи стандартні файли, дорівнює п'яти);

 процес 0 породжує процес 3, який успадковує таблицю відкритих файлів процесу 0;

 процес 1 породжує процес 4, який успадковує таблицю відкритих файлів процесу 1;

 процес 4 додатково відкрив ще два файли.

Після кожного з етапів друкуються таблиці відкритих файлів процесів, що беруть участь в даному етапі.

### Варіант 8

Процес створив новий файл і перепризначив на нього стандартний потік введення. Розробити програму, яка демонструвала б динаміку створення таблиць, пов'язаних з цією подією (таблиця дескрипторів файлів, таблиця відкритих файлів, масив файлових дескрипторів процесу). Наприклад, сценарій програми може бути таким:

 неявне відкриття стандартного файлу введення;

 неявне відкриття стандартного файлу виведення;

 неявне відкриття стандартного файлу виведення помилок;

 зчитування зі стандартного файлу введення 5 байт;

 відкриття призначеного для користувача файлу;

 закриття стандартного файлу введення (моделювання close(0));

 отримання копії дескриптора призначеного для користувача файлу (моделювання dup(fd), де fd — дескриптор призначеного для користувача файлу);

 закриття призначеного для користувача файлу (моделювання close(fd), де fd — дескриптор

призначеного для користувача файлу);

 читання зі "стандартного" файлу введення 10 байт.

Після кожного з етапів друкуються таблиця дескрипторів файлів, таблиця відкритих файлів, масив файлових дескрипторів процесу.

### Варіант 9

Нехай процес, який відкрив N файлів, перед породженням процесу-нащадка за допомогою системного виклику fork() закриває K<N файлів. Процес-нащадок відразу після породження закриває M<N-K файлів і через деякий час завершується (в цей час процес-предок очікує його завершення). Розробити програму, яка демонструвала б динаміку зміни даних в системі керування введеннямвиведенням ОС Linux (таблиці відкритих файлів і масиви файлових дескрипторів процесів). Наприклад, сценарій програми може бути таким:

 відкриття процесом-предком стандартних файлів введення-виведення і чотирьох призначених для користувача файлів для зчитування;

 закриття процесом-предком двох призначених для користувача файлів;

 процес-предок породжує процес, який успадковує таблиці файлів і відкритих файлів процесу-предка;  завершується процес-нащадок.

Після кожного з етапів друкуються таблиці відкритих файлів і масиви файлових дескрипторів для обох процесів.

### Варіант 10

Нехай процес здійснює дії у відповідності з наступним фрагментом програми:

main () {

...

fd = creat (temporary, mode); /\* Відкрити тимчасовий файл \*/

...

/\* Виконання операцій запису-читання \*/

... close (fd);

}

Розробити програму, яка демонструвала б динаміку зміни даних системи керування введенням-виведенням ОС Linux (таблиця дескрипторів файлів, таблиця відкритих файлів, масив файлових дескрипторів процесу).

1. Для вашого варіанту розробити програму, що моделює роботу системи керування введенням-виведенням ОС Linux з ведення структур (таблиць), які відстежують операції введення-виведення в системі. Після кожного кроку роботи програми щодо операцій роботи з файлами, вона повинна роздруковувати у вигляді таблиць поточну інформацію стану таблиць відкритих файлів, файлів і дескрипторів файлів (не слід намагатись друкувати усю таблицю inode вашої файлової системи!).

**Вказівка:** користуйтесь stat()/fstat(). Інформацію, що отримана зі структури stat, доповнену ім'ям файлу, і слід в лабораторних роботах трактувати як таблиці дескрипторів файлів. У тих завданнях, де потрібно відстежувати динаміку створення і модифікації таблиць файлів і таблиць відкритих файлів процесу, ці таблиці повинні програмно моделюватися при виникненні обставин, вказаних в завданнях лабораторної роботи. Ніяких дій по створенню процесів в програмах виконувати не потрібно.

1. Налагодити і зневадити складену програму, використовуючи інструментарій ОС Linux.
2. Усі отримані результати оформити у вигляді протоколу.
3. Захистити лабораторну роботу, продемонструвавши викладачеві роботу розробленої вами програми та її код, а також відповівши на контрольні запитання.

#### Контрольні запитання

1. Яка структура дескрипторів файлів, таблиці відкритих файлів, таблиці відкритих файлів процесу?
2. Яким є ланцюжок відповідності дескриптора файлу, відкритого процесом, і файлом на диску?
3. Опишіть функціональну структуру операції введеннявиведення (пули, асоціація їх з драйверами, способи передачі інформації і т.д.).
4. Яким чином здійснюється підтримка пристроїв введеннявиведення в ОС Linux?
5. Яка структура таблиці відкритих файлів і масиву файлових дескрипторів процесу після відкриття файлу?
6. Яка структура таблиці відкритих файлів і масиву файлових дескрипторів процесу після закриття файлу?
7. Яка структура таблиці відкритих файлів і масиву файлових дескрипторів процесу після створення каналу?

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