ТЕМА 5. УПРАВЛЕНИЕ НА ФАЙЛОВЕТЕ В UNIX

1. Файлове в UNIX

Файловете в UNIX (от гледна точка на ядрото) са информационни структури за съхранение на последователност от байтове. Тяхното съдържание и вътрешна логическа структура не се интерпретират от UNIX ядрото. Грижата за това е оставена на приложните програми, който използват различни програмни библиотеки, реализиращи едно по-високо ниво на абстракция при работата с файлове. Например, библиотеки за работа с таблични структури от полета и записи и тяхното адресиране посредством ключове. Всяка от функциите в тези библиотеки работи, като използва системните извиквания на ядрото за работа с файлове.

В UNIX се различават следните седем вида файлове:

1. Регулярни (обикновени) файлове
2. Директории
3. Твърди и символни връзки
4. Файлове на устройства (блокови и символни)
5. Неименувани канали (pipe)
6. Именувани канали (FIFO)
7. Сокети

Първите три вида са задължителни за всяка Unix файлова система

Регулярните файлове могат да съдържат ASCII символи, двоични данни, изпълним двоичен код, входни и изходни данни за програми, т.е информация въвеждана от потребителя, приложна или системна програма.

Директориите са файлове, който съдържат списък с имената на файловете в тях (включително поддиректории) в комбинация със съответните индексни възли характеризиращи файловете.

Твърдите връзки са допълнителни имена (псевдоними) на файловете, а символните връзки са файлове, съдържащи пътя и името на друг файл във файловата система.

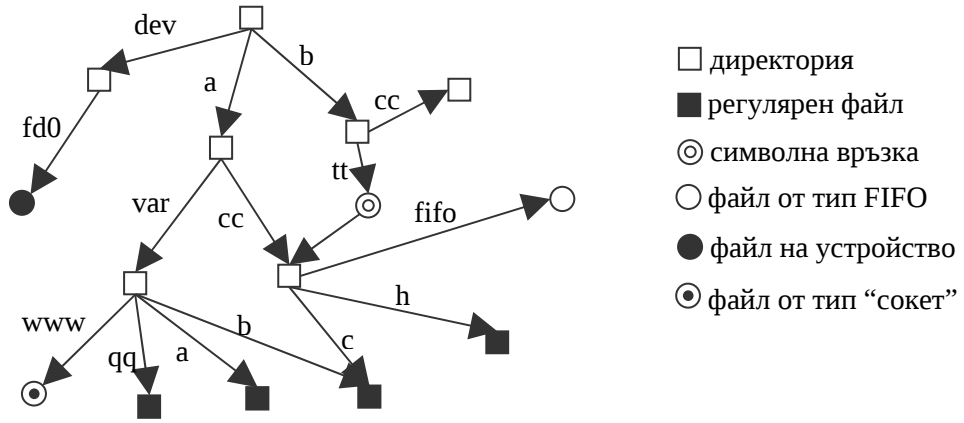
Файловете на устройства са свързани с В/И устройства и драйвери на устройства интегрирани в ядрото на ОС. Когато една програма осъществява достъп до файл на устройство, тя взаимодейства директно с

В/И устройство свързано с този файл.

Именуваните и неименуваните канали, както и сокетите са специални файлове, които се използват за комуникация между процеси.

2. Логическа структура на файловата система

Файловете от всички изброени типове са логически обединени в ориентиран ацикличен граф (фиг. 5.1). За разлика от дървовидната структура (описана в предходното упражнение) в ацикличния граф името на файла се свързва не с възела, съответстващ на файла, а с входящото в него ребро.



Фиг. 5.1. Пример за граф, показващ логическата структура на файловата система

Във вътрешните възли (т.е възлите, от които излизат ребра) на ацикличния граф могат да се разполагат само файлове от тип "директория", "символна" или "твърда" връзка. При това от възела в който се разполага файл от тип "символна" или "твърда" връзка може да излиза само едно ребро. В терминалните възли на ацикличния граф (т.е. във възлите от които не излизат ребра) могат да се разполагат файлове от всякакъв тип. Наличието на файл от тип "символна" или "твърда" връзка в терминален възел, обаче, показва, че целостта на файловата система е нарушена.

3. Индексни възли

Всички видове файлове в UNIX се администрират от операционната система, чрез техните индексни възли. Един индексен възел (i-възел) е структура от данни, която съдържа необходимата на операционната система информация, характеризираща файла. Няколко файлови имена

могат да бъдат асоциирани с един i-възел, но един i-възел се асоциира само с един файл. В индексния възел се съхраняват атрибутите на файла, правата за достъп до него и друга управляваща информация.

В UNIX се прави ясно разграничение между файл и файлови характеристики. С изключение на файловете на устройства и специалните файлове, всеки файл съдържа последователност от символи. Във файловете не се включва контролна информация, като дължина или знак за край на файла (EOF). Цялата информация необходима на файловата система за боравенето с (обработката на) един файл е включена в структура от данни наречена i-възел. Всеки файл има собствен i-възел, който файловата система използва, за да го идентифицира. Файловите системи и функциите на ядрото за боравене с тях могат да са различни за различните UNIX системи, но те трябва да осигуряват поне следните атрибути, описани в стандарта POSIX:

• Вид (тип) на файла (регулярен, директория, символно или блоково устройство, символна връзка, FIFO, pipe, сокет);

• Брой на твърдите връзки свързани с файла;

• Размер на файла в байтове;

• Адресна информация, която показва как е разположен файла върху диска;

• Идентификатор на устройство (за файловете на устройства);

• Номер на i-възела, който идентифицира файла във файловата система;

• Идентификатор на потребителя собственик на файла;

• Идентификатор на групата собственик на файла;

• Няколко времеви печати, които описват моментите на промяна в състоянието на i-възела – дата и час на последен достъп до файла и последна промяна;

• Права за достъп и режим на файла.

4. Права за достъп и режим на файловете

Потенциалните потребители на един файл попадат в една от следните три категории:

1) Потребителят собственик на файла;

2) Потребителите, които принадлежат на групата, на която принадлежи и собственика на файла;

3) Всички останали потребители;

За всяка от трите категории, съществуват три типа права за достъп – четене, запис и изпълнение. По този начин, правата за достъп до един файл се състоят от девет различни бита (двоични флага).

Значенията на правата за достъп са различни за директориите спрямо останалите типове файлове. Тези разлики са описани в следващата таблица.

Таблица 5.1. Значение на правата за достъп

Достъп	*Файл*	*Директория*
четене	Съдържанието може да се види	В съдържанието може да се търси (команда ls)
запис	Съдържанието може да се променя	Съдържанието може да се променя (да се добавят или изтриват файлове)
изпълнение	Стартиране на изпълними файлове	Директорията може да се направи текуща (да се влезе в директорията)

Режимът на файла се дефинира от три допълнителни бита (флага), наречени suid (Set User ID), sgid (Set Group ID) и sticky. Тези флагове имат следните значения, когато се прилагат към изпълним файл:

• suid – Обикновенно, когато един процес изпълнява даден файл, той използва идентификатора (UID) на собственика на процеса. Ако, обаче, флагът suid е установен процесът взема UID на собственика на файла.

• sgid – При изпълнение на файл, процесът породил неговото изпълнение използва идентификатора на групата (GID) на процеса. Ако, обаче, за изпълнимия файл е установен флага sgid, процесът взема идентификаторът на групата на файла.

• sticky – Изпълним файл с установен флаг sticky съответства на заявка към ядрото да остави изпълнимия файл на програмата в паметта след като изпълнението и приключи. Приложен към директория този флаг указва, че единствено собственика на даден файл в директорията може да преименува, премества или изтрива файла. Това е полезно за управление на файлове във поделени или временни директории.

Когато процес създаде файл, то негов идентификатор на собственика

става UID на процеса. Идентификатора на групата на собственика, може да бъде или GID на процеса създател или GID на родителската директория, в зависимост от стойността на флага sgid на родителската директория.

В UNIX съществува, така наречения “супер потребител” (super user), който притежава широки правомощия за достъп до системата и е освободен от обичайните ограничения за достъп до файлове. Всяка програма с установен suid флаг, чийто собственик е супер потребителя, получава потенциално неограничен достъп до системата, независимо от това, кой е потребителя, който я е стартирал.

5. Организация на разположението на файловете върху диска

Разположението на файловете върху диска се нарича физическото разположение на файловете. То е различно от логическото разположение представено на фиг. 5.1.

Физическо разположение се прави само за регулярните файлове, директориите и файловете от тип "символна връзка". Останалите специални типове файлове и файловете на устройства не заемат място на диска, за тях не се отделят логически блокове, тъй като цялата необходима информация за работа с тях се съдържа в техния индексния възел.

Дисковите устройства върху които са разположени файловете са разделени на дялове. Дисковите дялове от своя страна са разделени на логически блокове с еднакъв размер (при съвременните UNIX файлови системи размера на логическия блок е 1024 байта).

Данните на файловете се записват в логически блокове. Разполагането е динамично – използват се толкова блока колкото са необходими, за да се запише целия файл, като не е задължително блоковете на един файл да са съседно разположени върху диска. За да се проследят логическите блокове, в които е записан всеки файл се използва индексен метод, при който част от индексите са съхранени в i-възела на файла.

Индексният възел на файла включва 39 байта с адресна информация, организирана като тринайсет 3-байтови адреса или указателя. Първите 10 адреса сочат към първите 10 блока с данни на файла. Ако файла е по-голям от 10 блока, то се използва допълнително индексиране на няколко нива:

• Единайсетия адрес в i-възела сочи към блок от диска, който съдържа следващата част от индексите. Този блок съдържа указатели към следващите блокове на файла и се нарича единичен индиректен блок.

• Ако файла съдържа още блокове, дванайсетия адрес в i-възела сочи към двоен индиректен блок, който съдържа списък с адреси на допълнителни единични индиректни блокове. Всеки от които съдържа указатели към блоковете на файла.

• Ако файла е още по-голям и съдържа още много блокове, тринайсетия адрес в i-възела сочи към троен индиректен блок, който е последното трето ниво на индексиране. Този блок съдържа списък на указатели към допълнителни двойни индиректни блока.

Описаното тринивово индексиране на логическите блокове на файла е представено на фиг. 5.2.

Общият брой на блоковете с данни във файла зависи от размера на логическия блок в системата. В UNIX System V, дължината на блока е 1024 байта. Всеки блок може да съдържа общо 256 блокови адреса. Така максималния размер на един файл в тази схема е над 16 Гбайта (табл. 5.2).

Таблица 5.2 Капацитет на файл в UNIX

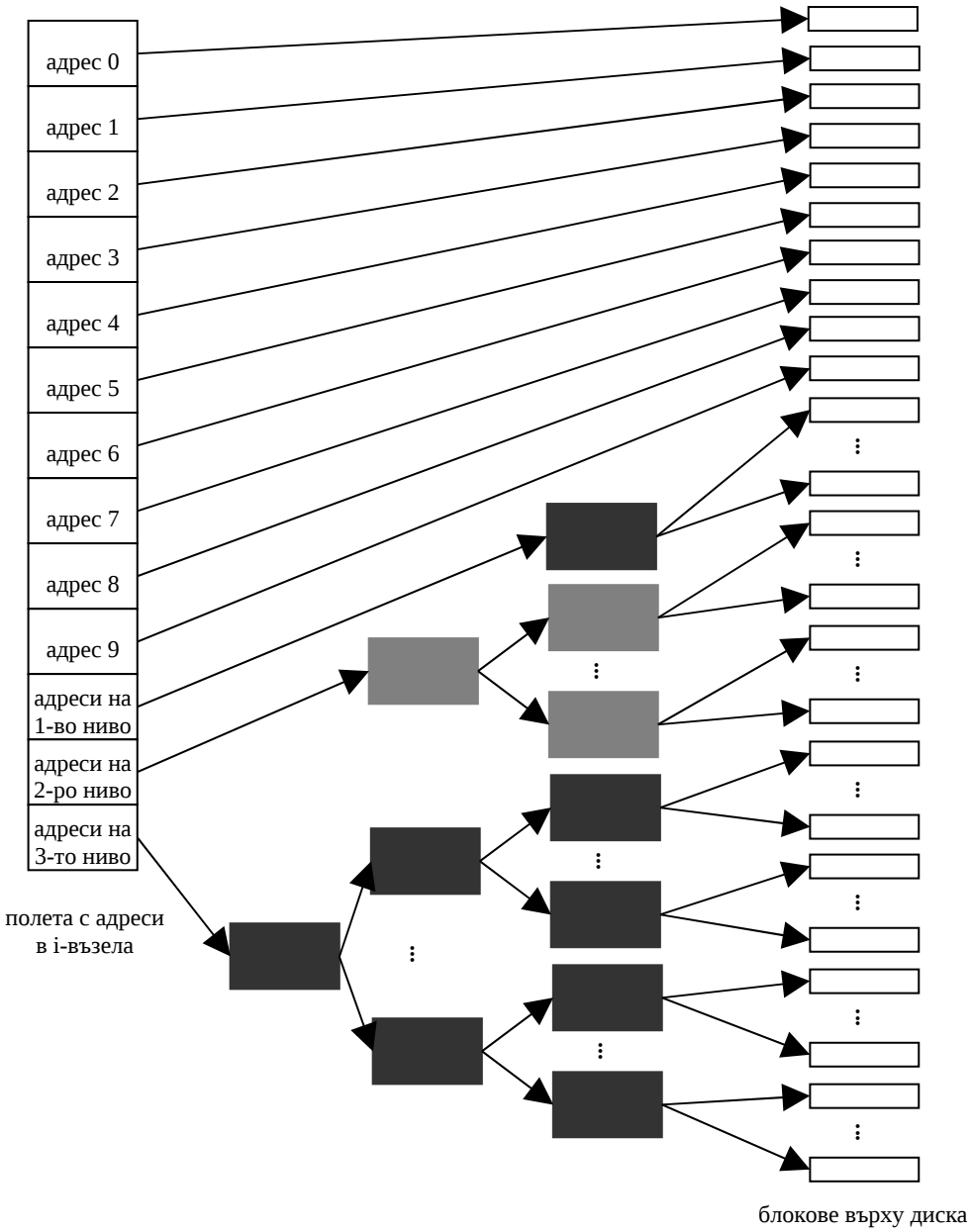
Ниво на адресиране	*Брой на блоковете*	*Размер в байтове*
Директно	10	10K
Единично индиректно	256	256K
Двойно индиректно	256 × 256 = 65K	65M
Тройно индиректно	256 × 65K = 16M	16Г

Описаният по-горе метод на индексиране има слените предимства:

1. Индексният възел е с фиксиран и относително малък размер, което позволява да бъде зареден в основната памет за дълги периоди от време.

2. Достъпът до файловете с по-малък размер може да се осъществи с единично индексиране или без индексиране, което намалява времето за обработка и достъп до диска.

3. Теоретичният максимален размер на файла е достатъчно голям, за да задоволи изискванията на почти всички приложения.



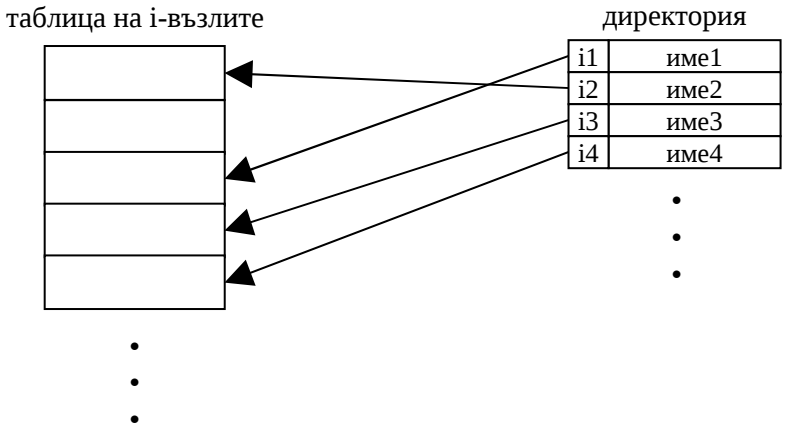
Фиг. 5.2. Метод за адресиране на логическите блокове в UNIX

6. Организация на директориите (каталозите) в UNIX

Директориите са файлове, които за разлика от регулярните файлове, (чиято структура не се задава от операционната система, а зависи от приложните програми) имат строго установена структура, зависеща от типа на използваната файлова система.

От потребителска гледна точка, един файл от тип "директория" съдържа списък на имената на файловете, намиращи се непосредствено в директорията и указателите към свързаните с тях индексни възли (фиг.

5.3). Всеки директориен запис съдържа име на файл или поддиректория и цяло число, наречено индексен номер (i-номер). Когато се осъществява обръщение към определен файл или директория, нейният i-номер се използва като индекс в таблицата на индексните възли.



Фиг. 5.3. Организация на директориите в UNIX

7. Организация на файловата система в UNIX

Всеки дисков дял в UNIX има собствена файловата система, която се състои от следните елементи:

• Блок за начално зареждане: Съдържа кода необходим за първоначалното зареждане на операционната система;

• Суперблок: Съдържа служебна информация, необходима а работата на файловата система, като: тип на файловата система; размер на логическия блок в байтове; размер на дисковия дял, размер на таблицата с индексни възли; брой на свободните индексни възли; брой на свободните индексни възли (т.е. оставящият брой файлове, които могат да бъдат записани във файловата система); брой на свободните блокове за разполагане на файлове;

• Таблица на индексните възли: Съдържа индексните възли на всички файлове;

• Блокове с данни: Наличното пространство за съхранение на файлови данни и поддиректории.

При първото обръщение към файловата система данните от

суперблока се зареждат в адресното пространство на ядрото, за да се ускорят последващите обръщения към него.

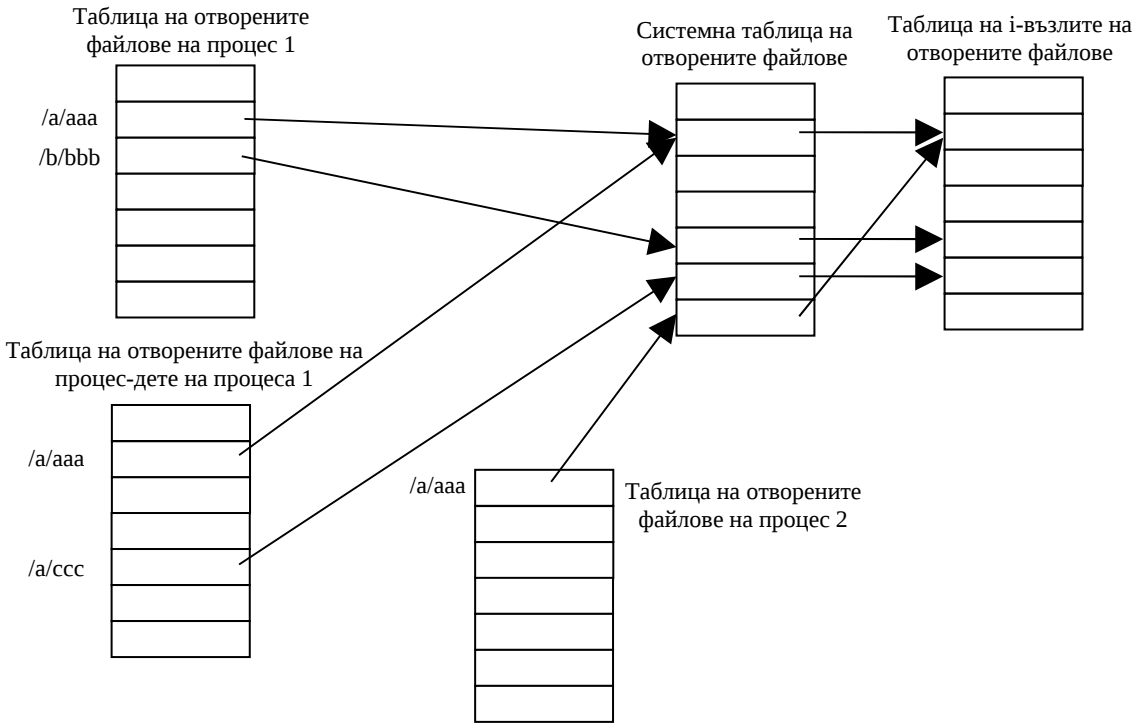
8. Отваряне и работа с файлове в UNIX

Информацията необходима за работа с един файл (атрибутите на файла и неговото разположение на физическия носител) се съдържа в асоциирания с файла индексен възел. Тази информация се използва всеки път когато се извършва някаква операция с файла, например четене или запис. За да се ускори достъпа и работата с файла, информацията в неговия индексен възел се прочита само веднъж при първото обръщение към файла (при неговото отваряне) и се зарежда в основната памет в адресното пространство на ядрото на ОС.

Възможно е няколко процеса да работят едновременно (съвместно или независимо един от друг) с един файл. В този случай е нецелесъобразно информацията от един и същ индексен възел да се зарежда неколкократно в паметта на КС. За целта в адресното пространство на ядрото се поддържа една таблица на индексните възли на отворените файлове (фиг. 5.4), а процесите получават достъп до данните в нея, чрез указатели записани в системната таблица на отворените файлове. Всеки процес от своя страна поддържа собствена таблица на отворените файлове, в която са записани поредните номера на отворените от процеса файлове (файловите дескриптори) заедно със съответен указател към системната таблица на отворените файлове. Ако един файл се използва едновременно от два независими процеса, то в системната таблица на отворените файлове съществуват два отделни записа за него. Ако процесите работят съвместно с файла (в UNIX това могат да бъда само близко родствени процеси, т.е процеси, единият от които е дете, а другия родител или процеси на които родителя е един и същ), то указателите в техните таблици на отворените файлове сочат към един и същи запис в системната таблица.

От гледна точка на потребителския процес в UNIX всеки файл представлява обособен последователен набор от байтове, снабден с указател за текущата позиция на процеса в тази последователност. Всички операции четене и запис във файла се извършват от мястото сочено от

указателя на текущата позиция. След завършване на операцията четене или запис указателя се премества след края на участъка, който е прочетен или записан. Този указател е част от контролния блок на процеса (PCB), което означава, че той е различен за независимите процеси и еднакъв за процесите, които са близко родствени.



Фиг. 5.4. Връзка между таблиците на отворените файлове

За коректната работа на няколко независими процеса с един и същ файл, е необходимо да се синхронизира тяхната работа, чрез механизъм за взаимоизключване, тъй като те използват различни указатели за текущата позиция във файла, което може да доведе до преплитане на операциите запис и четене, извършвани от двата независими процеса.

При съвместното използване на един и същи файл от близко родствени процеси, се използва един и същ указател на текущата позиция във файла. Така, операциите четене и запис изпълнени от един процес изменят стойността на този указател във всички близко родствени процеси работещи едновременно с този файл и преплитане между операциите чете или запис изпълнявани от различните процеси е невъзможна.

Цялата информация за файла, необходима на процеса за работа с него може да бъде разделена на три части:

Данни специфични за процеса;

Данни, общи за близко родствени процеси, които използват съвместно файла, например, указателят на текущата позиция;

Данни общи за всички процеси използващи файла, – атрибутите и разположението върху диска.

За съхранението на тази информация се използват три свързани структури от данни, намиращи се в адресното пространство на ядрото на операционната система – таблица на отворените файлове на процеса, системна таблица на отворените файлове и таблица на индексните възли на отворените файлове. За достъп до тази информация в блока за управление на процеса се поддържа таблица на отворените файлове. Всеки непразен елемент в нея съдържа препратка към съответстващия му елемент в системната таблица на отворените файлове, който съдържа данните необходими за съвместно използване на файла от близко родствени процеси. От своя страна, системната таблица на отворените файлове съдържа препратка до общите данни за файла, съхраняващи се в таблицата на индексните възли на отворените файлове (фиг. 2.4). Единствено таблицата на отворените файлове на процеса влиза в състава на неговия блок за управление (PCB), който се наследява от процеса родител при създаването на нов процес. Индекса (поредния номер) на елемента в тази таблица (цяло положително число) се нарича файлов дескриптор. Чрез него процеса работи с файла на потребителско ниво. В същата тази таблица се намират и препратки към данни, описващи други входно-изходни потоци, като pipe и FIFO, а също и към структури от данни, необходими за предаване на информация между отдалечени процеси в компютърна мрежа.

9. Системни извиквания за работа с файлове

Когато потребителски процес осъществява достъп до съдържанието на регулярен файл или директория, той трябва да се обърне към запомнящото устройство, в което се съхраняват данните. Тъй като един процес в потребителски режим не може пряко да взаимодейства с хардуерните компоненти от ниско ниво, всяка файлова операция трябва да се извършва в режим на ядрото. За тази цел в ОС UNIX са дефинирани

няколко системни извиквания за работа с файлове. Когато един процес иска да извърши някаква операция върху определен файл, той използва подходящо системно извикване. Един от параметрите на системното извикване е дескриптора на файла. Ето защо, преди да може да се работи с файла е необходимо да се помести информация за него в трите описани по-горе таблици и да се определи съответстващия на файла файлов дескриптор. Това се извършва, посредством отваряне на файла с помощта на системното извикване open().

10. Отваряне на файл. Системно извикване open()

Процесите имат достъп само до отворени файлове. За да отвори файл, процесът използва системно извикване:

fd = open(path, flag, mode)

Параметрите на това извикване имат следните значения:

path – посочва пътя (относителен или абсолютен) до файла, за да бъде отворен;

flag – задава, как трябва да бъде отворен файла (напр., четене, запис, четене и запис или добавяне). Този параметър също така посочва дали файл, който не съществува трябва да бъде създаден.

mode – Задава правата за достъп до новосъздаден файл.

При отваряне на файл операционната система проверява, съответстват ли заявените от процеса права за достъп с установените за файла права. В случай на съвпадение в системната таблица на отворените файлове се разполага необходимата информация за файла и ако този файл не е бил отворен до момента от други процеси, информация за него се разполага и в таблицата на индексните възли на отворените файлове. След което операционната система търси празен елемент в таблицата на отворените файлове на процеса, за да установи необходимата връзка между трите таблици и да върне на потребителския процес дескриптора на отворения файл. Накратко, системното извикване open() създава обектът „отворен файл” и връща идентификатор към него наречен файлов дескриптор.

Един обект на отворен файл съдържа:

Структури от данни за боравене с файла. Например, указател към

област от паметта (използвана за буфериране от ядрото), където се копират данните от файла, поле за отместване, което отбелязва текущата позиция във файла, от която следва да продължи следващата операция (т. нар. указател на файла) и др.

Указатели към функции на ядрото, които процесът може да извика. Наборът от разрешени функции зависи от стойността на параметъра flag.

Файловият дескриптор представлява едно взаимодействие между процес и отворен файл. Обектът „отворен файл" съдържа данните необходими за това взаимодействие. Един и същ обект „отворен файл" може да се идентифицира чрез няколко файлови дескриптора.

Ако няколко процеса отворят едновременно един и същ файл, ядрото на операционната система създава отделни файлови дескриптори и отделни обекти „отворен файл" за всеки процес. Файловата система на UNIX не осигурява синхронизация между В/И операции, които процесите извършват над файла. Налични са обаче няколко системни извиквания, като flock( ), които позволяват на процесите да се синхронизират помежду си при работата си с целия файл или с части от него.

За да създаде нов файл процесът също така може да използва системното извикване create( ), което се обработва от ядрото по същия начин, както open( ).

СИСТЕМНО ИЗВИКВАНЕ

open

ПРОТОТИП НА СИСТЕМНОТО ИЗВИКВАНЕ

```
#include <fcntl.h>
int open(char *path, int flags);
int open(char *path, int flags, int mode);
```

ОПИСАНИЕ НА СИСТЕМНОТО ИЗВИКВАНЕ

Системното извикване open е предназначено за изпълнение на операцията отваряне на файл. В случай, че операцията завърши успешно, извикването връща файловият дескриптор на отворения файл (неотрицателно цяло число, което се използва при извършване на други операции с този файл).
Параметърът path се явява указател към низ, съдържащ пълното или относителното име на файла.
Параметърът flags приема едно от следните три стойности:
O_RDONLY – ако над файла може да се извършва само операция четене;
O_WRONLY – ако над файла може да се извършва само операция запис;
O_RDWR – ако над файла могат да се извършват операция четене и операция запис.
Посредством операцията "побитово или (|)", тези стойности могат да се комбинират с един или няколко флага:
O_CREAT – ако файлът с указаното име не съществува, той се създава;
O_EXCL – прилага се съвместно с флага O_CREAT. При тяхното съвместно използване, ако файла с указаното име съществува, операцията по неговото отваряне не се осъществява и се връща съобщение за грешка;

O_NDELAY – изпраща изпълнението на процеса в състояние на очакване при операция отваряне или друга последваща операция над този файл;
O_APPEND – при отваряне на файл и преди изпълнението на всяка операция запис, указателят на текущата позиция във файла се установява в края на файла;
O_TRUNC – ако файла съществува, неговият размер се намалява до 0, като се запазват съществуващите атрибути на файла с изключение на времето на последния достъп до файла и неговата последна модификация.
В някои версии на ОС UNIX се прилагат допълнителни флагови стойности:
O_SYNC – всяка операция на запис във файла се блокира (т. е. процесът се привежда в състояние на очакване) до момента, когато записаната информация не бъде физически поместена на съответното по-ниско хардуерно ниво;
Параметърът mode установява атрибутите свързани с правата на достъп на различните категории потребители до новия файл при неговото създаване. Той е задължителен, ако е зададен флага O_CREAT и може да се пропусне в противен случай. Този параметър се задава като сума от следните осмични числа:
0400 – разрешена операцията четене за потребителя собственик (създател) на файла;
0200 – разрешена операцията запис за потребителя собственик (създател) на файла;
0100 – разрешена операцията изпълнение за потребителя собственик (създател) на файла;
0040 – разрешена операцията четене за групата на потребителя собственик (създател) на файла;
0020 – разрешена операцията запис за групата на потребителя собственик (създател) на файла;
0010 – разрешена операцията изпълнение за групата на потребителя собственик (създател) на файла;
0004 – разрешена операцията четене за всички останали потребители;
0002 – разрешена операцията запис за всички останали потребители;
0001 – разрешена операцията изпълнение за всички останали потребители;
Правата за достъп до създадения файл реалното се получават от комбинацията на параметъра mode и маската на текущия процес за създаване на файлове umask, по следния начин mode & ~umask.
Системното извикване open има някой особености, когато се отварят файлове от тип FIFO. Ако FIFO файлът се отваря само за четене и не е зададен флага O_NDELAY, то процесът, осъществяващ системното извикване се блокира до момента, когато някой друг процес не отвори FIFO за запис. Ако е зададен флагът O_NDELAY, то се връща стойността на файловият дескриптор, асоциирана с FIFO. Ако FIFO се отваря само за запис и не е зададен флага O_NDELAY, то процесът, осъществяващ системното извикване се блокира до момента, когато някой друг процес не отвори FIFO за четене. Ако е зададен флага O_NDELAY, то възниква грешка и се връща стойността -1.

ВРЪЩАНА СТОЙНОСТ

Системното извикване open() връща стойността на файловия дескриптор за отворения файл при, ако извикването завърши нормално и стойност -1, при възникване на грешка.

Системното извикване open() използва набор от флагове, чрез които да се конкретизира операцията по отваряне на файл. Те се прилагат при последващите операции над файла или непосредствено при неговото отваряне. Флаговете O_RDONLY, O_WRONLY, O_RDWR се явяват са взаимоизключващи се. Наличието на един от тях не допуска наличието на другите два. Тези флагове описват набор от операции, които са разрешени за изпълнение върху файла при неговото успешно отваряне: само четене, само запис, четене и запис. Както е известно от предходните упражнения, всеки файл притежава атрибути, които определят правата за достъп до него на различните категории потребители. Ако файлът със зададеното име

съществува на диска и правата за достъп на потребителя от чието име се изпълнява текущият процес не противоречи на заявения набор от операции, операционната система сканира таблицата на отворените файлове от началото към края за наличие на свободен елемент. Първият срещнат свободен елемент се запълва и неговият индекс се връща в качеството на файлов дескриптор на отворения файл. Ако файлът не е на диска, правата му а достъп не отговарят на заявената операция или липсва свободно място в таблицата на отворените файлове, се констатира възникване на грешка.

В случаят, когато се предполага че файлът липсва на диска, за да бъде създаден се използва флага O_CREAT. В този случай, ако файлът съществува, то всичко ще протече по разгледания ред. Ако файлът липсва, той се създава с набора права, указани в параметрите на системното извикване open.

В случай, че е необходимо файлът да бъде създаден в момента на неговото отваряне, трябва да се използва комбинацията от флагове O_CREAT и O_EXCL.

11. Операции с отворен файл. Системни извиквания read(), write(), lseek()

Регулярните файлове в UNIX могат да бъдат адресирани или последователно или произволно, докато при файловете на устройства и именуваните канали се използва само последователен достъп. И при двата вида достъп ядрото съхранява указател към файла в обекта „отворен файл”, който показва текущата позиция, от която следва да се извърши следващата операция четене или запис.

За извършване на потоковите операции четене на информация от файл и запис на информация във файл се използват ситемните извиквания read() и write().

По подразбиране при последователен достъп системните извиквания read( ) и write( ) следят текущата позиция сочена от указателя на файла. За да се реализира произволен достъп до отворен файл е необходимо да се промени стойността на този указател. Това може да се извърши във всеки един момент с помощта на системното извикване lseek(). При отваряне на

файл, ядрото установява указателя на файла да сочи към неговия първи байт (отместване 0).

Особеност на това системно извикване е възможността да се премести текущата позиция след края на файла (т.е. стойността на указателя да стане по-голяма от дължината на файла). Ако това се случи при следващата операция запис ще възникне промеждутък от края на файла до текущата позиция, който ще бъде запълнен с нулеви байтове, а при операция четене ще бъде върната стойност нула за броя на прочетените байтове.

Системното извикване lseek( ) изисква следните параметри:

newoffset = lseek(fd, offset, whence);

които имат следните значения:

fd – задава дескриптора на отворения файл;

offset – задава отместването (цяло число със знак), което ще се използва за изчисляване на новата позиция на указателя на файла;

whence – определя дали новата позиция (отместването) да се изчисли спрямо началото на файла или спрямо края.

Системното извикване read() изисква следните параметри:

nread = read(fd, buf, count);

които имат следните значения:

fd – задава дескриптора на отворения файл;

buf – задава адреса на буфера в адресното пространство на процеса, към който да се прехвърлят данните;

count – определя броя байтове, които да бъдат прочетени

Обработвайки това системно извикване ядрото се опитва да прочете зададения брой байтове от файла определен от дескриптора fd, започвайки от текущата стойност на полето за отместване на отворения файл. В някой случаи – край на файла, празен канал и т.н. ядрото не успява да прочете зададения брой байтове. В този случай върнатата в nread стойност показва броят на ефективно прочетените байтове. Указателят на файла също се обновява, като се добави nread към неговата предишна стойност. Параметрите на системното извикване write() са подобни.

ИМЕ

read и write.

ПРОТОТИП

```
#include <sys/types.h>
#include <unistd.h>
size_t read(int fd, void *addr, size_t nbytes);
size_t write(int fd, void *addr, size_t nbytes);
```

ОПИСАНИЕ

Системните извиквания read и write са предназначени за осъществяване на потоковите операции въвеждане (четене) и извеждане (запис) на информация над файл, pipe, FIFO и socket (канал за връзка).
Параметърът fd се явява файлов дескриптор на предварително създаден потоков канал за връзка, чрез който се изпраща или получава информация. Стойността на fd се връща като резултат на едно от следните системни извиквания open(), pipe() или socket().
Параметърът addr представлява начален адрес на област от паметта, в която се съхранява информацията за предаване или в която се разполага приетата информация.
Параметърът nbytes за системното извикване write определя броя байтове, които се извличат от паметта с начален адрес addr и се предават по канала за връзка. Параметърът nbytes за системното извикване read определя броя байтове, които трябва да се получат от канала за връзка и разположат в паметта с начален адрес addr.

ВРЪЩАНИ СТОЙНОСТИ

При успешно завършване на системното извикване, то връща броя на реално предадените или приети байтове. Тази стойност (по-голяма или равна на 0) може да не съвпада със стойността на параметъра nbytes. Тя може да е по-малка поради лиса на достатъчно място на диска или липса на информация при нейното приемане. При възникване на грешка се връща отрицателна стойност.

ОСОБЕНОСТИ

При работа с файлове записа или четенето от тях, започва от място, което се определя от указателя на текущата позиция във файла. Стойността на указателя се увеличава според броя на реално прочетените или записани байтове. Ако системното извикване read върне стойност 0, то файлът е прочетен до края.

ИМЕ

lseek()

ПРОТОТИП

```
#include <sys/types.h>
#include <unistd.h>
off_t lseek(int fd, off_t offset, int whence);
```

ОПИСАНИЕ

Системното извикаване lseek е предназначено за промяна на положението на указателя на текущата позиция в отворен регулярен файл.
Параметърът fd е дескриптора, съответстващ на файла, т. е. стойността, върната от системното извикване open().
Параметърът offset съвместно с параметъра whence определят новото положение на указателя на текущата позиция по следния начин:
• Ако стойността на параметъра whence е равна на SEEK_SET, то новата стойност на указателя е offset байтове от началото на файла. Стойността на offset в този случай трябва да бъде положителна.
• Ако стойността на параметъра whence е равна на SEEK_CUR, то новата стойност на указателя е старата стойност на указателя + offset байтове. При това новата стойност не трябва да е отрицателна.
• Ако стойността на параметъра whence е равна на SEEK_END, то новата стойност на указателя е дължината на файла + offset байтове. При това новата стойност не трябва да е отрицателна.
Системното извикване lseek позволява да се промени текущата стойност на указателя да сочи след края на файла (т.е. да превишава размера на файла). При това положение възниква промеждутък, след края на файла до новото положение на указателя, който ако се извърши операция запис се запълва с

```
нулеви байтове.
Типът данни off_t е синоним на типа данни long.
```

ВРЪЩАНИ СТОЙНОСТИ

```
При нормално завършване системното извикване връща новото положение на
указателя на текущата позиция в байтове спрямо началото на файла и -1 при
възникване на грешка.
```

Ако при отварянето на файла със системното извикване open() е бил установен флага O_APPEND, то всяка операция за запис във файла винаги ще добавя новите данни в неговия край, независимо от предишното положение на указателя на текущата позиция.

12. Затваряне на файл.Системно извикване close()

След завършване на работата с файла, когато съдържанието му вече не е необходимо на един процес, процесът може да затвори файла като използва системното извикване close(). Така се изпразват буферите на файла, освобождават се заделените за него ресурси от операционната система, а елемента от таблицата на отворените файлове, съответстващ на файловият дескриптор, се отбелязва като свободен.

Когато един процес завърши (с помощта на функцията exit(), извикана явно или неявно), ядрото затваря всички все още незатворени от процеса файлове.

С помощта на системното извикване close() се синхронизира информацията за файла, съдържаща се в таблицата на индексните възли на отворените файлове, с информацията на диска. След изпълнение на системното извикване close(), ако в системата съществува поне един процес, който да използва файла, то елемента с информация за него заделен в таблицата на индексните възли на отворените файлове не се освобождава За всеки индексен възел се поддържа брояч за количеството отворени файлове, който се увеличава с 1 при всяко отваряне на файла със системното извикване open() и се намалява с 1 при всяко негово затваряне с close(). Премахването на елемента от таблицата на индексните възли на отворените фалове и окончателното синхронизиране на данните в паметта и тези на диска се извършва само в случай, че при поредното затваряне на файла неговия брояч стане 0.

ИМЕ

```
close
```

ПРОТОТИП

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

ОПИСАНИЕ

Системното извикване close е предназначено за коректно завършване на работата с файлове и други В/И обекти, които се описват в операционната система чрез файлови дескриптори: pipe, FIFO, socket.
Параметърът fd се явява дескрипторът на съответния обект. Неговата стойност е върната от едно от системните извиквания open(), pipe() или socket().

ВРЪЩАНИ СТОЙНОСТИ

Системното извикване връща стойност 0 при нормално завършване и стойност -1 при възникване на грешка.

За илюстрация на системните извиквания за отваряне и работа с файлове е дадена следната програма:

```
/*Програма 5_1.c, илюстрираща използването на системните извиквания
open(), write() и close() за запис на информация във файл */
#include <sys/types.h>
#include <fcntl.h>
#include <stdio.h>
int main(){
    int fd;
    size_t size;
    char string[] = "Hello, world!";
    /* Обновяване на маската на създадения файл от текущия процес, така
    че правата за достъп до създадения файл да съответстват на
    параметърът в извикването open() */
    (void)umask(0);
    /* Отваряне на файл с име myfile в текущата директория
     само за четене от него. Ако файлът не съществува, се създава
     с права за достъп 0666, т.е. read-write за всички категории
     потребители */
    if((fd = open("myfile", O_WRONLY | O_CREAT,
        0666)) < 0){
        /* При неуспешно отваряне на файла, се отпечатва съобщение
           и работата на програмата се прекратява */
        printf("Can\'t open file\n");
        exit(-1);
    }
    /* Във файла се записват 14 байта от масива string, т.е.
    целия низ "Hello, world!" заедно със знака за край на
    низа */
    size = write(fd, string, 14);
    if(size != 14){
        /* Ако са се записали по-голямо количество байтове,
        се съобщава за грешка */
        printf("Can\'t write all string\n");
        exit(-1);
    }
    /* Затваряне на файла */
    if(close(fd) < 0){
        printf("Can\'t close file\n");
    }
    return 0;
}
```

Програма 5_1.c, илюстрираща използването на системните извиквания open(), write() и close() за запис на информация във файл

Задача 1: Въведете тази програма, компилирайте и я изпълнете. Анализирайте полученият резултат.

Съвет: Обърнете внимание на използваното системно извикване *umask()* с параметър *0* така, че правата за достъп до създадения файл, точно да съответстват на указаните в системното извикване *open()*.

Задача 2: Изменете предходната програма, така че да чете записаната по-рано информация във файла и да я отпечатва на екрана. Всички излишни оператори е желателно да се изтрият.