МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ОДЕСЬКИЙ ДЕРЖАВНИЙ ЕКОЛОГІЧНИЙ УНІВЕРСИТЕТ

ТКАЧ Т.Б.

UNIX- ПОДІБНІ ОПЕРАЦІЙНІ СИСТЕМИ

Конспект лекцій

Одеса
Одеський державний екологічний університет
2017

ЗМІСТ

1 ІСТОРІЯ СТВОРЕННЯ ТА ПРИЧИНИ ПОПУЛЯРНОСТІ UNIX 4
1.1 Історія створення системи 4
1.2 Основні характеристики системи 7
1.3 Відмінності між Windows і UNIX 9
1.4 Стандарти 10
2 АРХІТЕКТУРА ОС UNIX 12
2.1 Структурна організація ОС UNIX 12
2.1.1 Структура ядра ОС UNIX 13
2.2 Функції ОС UNIX 17
2.2.1 Середа користувача UNIX 20
3 КОРИСТУВАЧІ ТА ГРУПИ системИ 22
4 ПОНЯТтя і призначення ФАЙЛУ 25
4.1 Імена файлів в UNIX-подібних ОС 25
4.2 Типи файлів 26
4.3 Права доступу до файлів 29
4.4 Управління правами доступу 31
5 Файлова система UNIX-подібних ОС 33
5.1 Поняття логічної файлової системи 33
5.1.1 Логічна файлова система – основні каталоги та їх призначення 34
5.2 Фізична організація файлової системи 36
5.2.1 Основні компоненти фізичної файлової системи UNIX - подібних ОС 36
5.3 Орієнтація та навігація у файловій системі 39
5.4 Управління файловою системою 40
5.4.1 Створення фізичної файлової системи 40
5.4.2 Монтування і демонтування фізичних файлових систем 41
5.4.3 Перевірка та відновлення цілісності файлових систем 43
5.4.4 Отримання інформації про файлові системи 44
5.5 Файлова система /proc 44
5.6 NFS 45
5.7 VFS 45
5.8 Файлові системи, що журналізуються 46
6 Процеси 47
6.1 Типи процесів: 47
6.2 Атрибути процесу 48
6.3 Структура процесу 49
6.4 Логічна організація процесів 51
6.5 Функціонування процесу 51
6.6 Управління процесами 52
6.7 Життєвий цикл процесу 54
6.8 Контекст процесу 56
6.8.1 Перемикання контексту 57
6.8.2 Створення процесу 57
6.8.3 Завершення виконання процесу 58
6.9 Планування 59
6.9.1 Алгоритми планування 59
6.9.2 Пріоритети процесів в UNIX 60
6.9.3 Планування в Linux 62
7 МІжпроцесна взаємодія (IPC) 64
8 Управління памятю 69

8.1 Основні поняття 69
8.1.1 Віртуальна память (ВП) 69
8.1.2 Методи організації пам’яті 70
8.1.3 Адресний простір процесу 70
8.1.4 Методи управління ОП 71
8.2 Апаратна частина ВП 73
8.3 Системні виклики управління пам’яттю 74
9 Подсистема введення-виведення 76
9.1 Багатошарова модель підсистеми введення-виведення 77
9.2 Багаторівневі драйвери 78
10 МЕРЕЖА В UNIX 81
10.1 Семирівнева модель OSI 81
10.2 Протоколи Internet: TCP/IP 82
10.2.1 Мережевий інтерфейс в UNIX 83
10.3 Конфігурація IP-мереж 84

1 ІСТОРІЯ СТВОРЕННЯ ТА ПРИЧИНИ ПОПУЛЯРНОСТІ UNIX

1.1 Історія створення системи

UNIX (Uniplex Information and Computing Services) - високошвидкісна, ефективна і гнучка ОС, яка використовується як на мейн-фреймах, так і на персональних комп'ютерах.

UNIX-подібна операційна система – операційна система, яка утворилася під впливом UNIX. Термін включає вільні (відкриті) операційні системи, утворені від UNIX компанії Bell Labs або адаптовані під їх можливості, комерційні і запатентовані розробки, а також версії, засновані на вихідному коді UNIX.

Операційна система (ОС) - це організована сукупність програм і даних, яка виконує функції посередника між користувачами і комп'ютером. ОС служить двом цілям: по-перше, зробити комп'ютерну систему зручною для використання, і, по-друге, ефективно використовувати апаратні засоби комп'ютера.

На першій стадії свого розвитку ОС призначалися для виконання базових завдань з управління апаратними засобами. В цьому відношенні ОС були розраховані не на користувача, а на апаратні засоби. ОС відрізнялися жорсткістю, змушуючи користувача пристосовуватися до вимог ефективності використання апаратних засобів.

Операційна система UNIX створювалася в кілька етапів. Все починалося в 1965-69 рр. в Bell Labs концерну AT & T в рамках проекту MULTICS (Multi-user Timesharing Interactive Computing System – об'єднані інформаційні та обчислювальні послуги) для великої машини General Electric GE-645. У той час AT & T могла розробляти, але не продавати комп'ютерні продукти, в результаті діючих в США антимонопольних законів. Від Bell Labs в проекті брали участь два співробітника – Кен Томпсон і Денніс Рітчі.

Ця ОС замислювалася як багатокористувачева, багатозадачна, з ієрархічної файлової системою. Однак ця система зазнала невдачі, частково через занадто високі цілі, які не відповідали рівню комп'ютерів того часу, а почасти й через те, що вона розроблялася на мові PL / 1 , а компілятор PL / 1 затримувався і взагалі погано працював після своєї запізнілої появи.

Завершено проект не було, але деякі учасники проекту вирішили врятувати ідеї MULTICS. У 1969 р Кен Томпсон написав ОС, яку назвали UNICS (Uniplexed Information and Computing Service – роз'єднані інформаційні та обчислювальні послуги), підкреслюючи її відмінність від розрахованої на багато користувачів MULTICS. Незабаром UNICS почали називати UNIX.

Зайнятися продовженням проекту співробітників підштовхнуло бажання встановити гру. У Кена Томпсона була написана ним ще за часів роботи над MULTICS гра "Space Travel" – "Космічна подорож". Він запускав її на комп'ютері GE-645, але вона працювала на ньому не дуже добре. Тоді Томпсон і Рітчі вирішили перенести гру на машину PDP-7 фірми DEC, що має 4096 18-бітних слів, телетайп і хороший графічний дисплей. Але у PDP-7 було неважливе програмне забезпечення, і, закінчивши перенесення гри, Томпсон вирішив реалізувати на PDP-7 ту файлову систему, над який він працював на GE-645. З цієї роботи і виникла перша версія UNIX. Вона вже включала характерну для UNIX файлову систему, засновану на індексних дескрипторах, мала підсистему керування процесами і пам'яттю, а також дозволяла двом користувачам працювати в режимі поділу часу. Система була написана на асемблері.

1970 рік вважається роком народження UNIX. Саме від першого січня 1970 року відраховується системний час в UNIX.

Спочатку UNIX написана на асемблері для DEC PDP-7. Потім до роботи за цим проектом було залучено Денніса Рітчі, який в той час вже розробив мову B. У 1973 році він запропонував переписати основну частину UNIX на B. У процесі здійснення цієї ідеї, мова B настільки удосконалився, що перетворився в C. Таким чином, було досягнуто небачене тоді якість – мобільність. На відміну від всіх попередніх ОС, на 100 відсотків написаних на асемблері для певної машини, UNIX мала тільки 10 відсотків (1000 рядків) коду на асемблері. Для того щоб працювати на довільній машині, нова ОС потребувала написання декількох сторінок на асемблері і компіляторі мови C.

Спочатку ОС UNIX вважалася дослідним продуктом, тому перші версії UNIX розповсюджувалися безкоштовно по факультетах обчислювальної техніки багатьох відомих університетів.

Широке поширення UNIX одержав з 1974 року, після опису цієї системи Томпсоном і Рітчі в комп'ютерному журналі CACM. UNIX одержав широке поширення в університетах, так як для них він поставлявся безкоштовно разом з вихідними кодами на C. Широке поширення ефективних C-компіляторів зробило UNIX унікальної для того часу ОС через можливість переносу на різні комп'ютери. Університети внесли значний вклад у поліпшення UNIX і подальшу його популяризацію. Ще одним кроком на шляху отримання визнання UNIX як стандартизованого середовища стала розробка Деннісом Рітчі бібліотеки введення-виведення stdio. Завдяки використанню цієї бібліотеки для компілятора C, програми для UNIX стали легко переносяться.

До 1975 р. AT&T, в яку входить Bell Labs, була єдиним офіційним розробником UNIX, незважаючи на безліч додатково написаних функцій активними користувачами.

Однак, AT&T була в скрутному становищі і не могла займатися бізнесом, які не мають відношення до телефонії і телеграфії.

Ініціатива розвитку системи перейшла до Каліфорнійському університету в Берклі. У 1975 р. вийшла версія BSD – Berkeley Software Distribution.

Поступово стали з'являтися і інші версії.

У 1980 р. Microsoft випустила версію UNIX для ПК, названу XENIX.

У 1982 р. вийшла версія 7 BSD. Це була перша 32-бітна версія.

AT&T відновила роботу над системою і випустила версію System III. За нею пішла версія System V, яка стала серйозним програмним продуктом.

Виникла необхідність стандартизувати ОС.

ATA&T передала права на UNIX своєму підрозділу USL (UNIX System Laboratories). USL розробляла стандартну систему, що об'єднує основні версії UNIX. У 1991 р USL випускає System V версії 4.

Права на UNIX набуває компанія Nowell. У 1993 р. виходить версія UNIX від Nowell – UNIXWare.

Nowell в свою чергу передає права на UNIX компанії Santa Cruz Operation – SCO, яка теж випускає свою версію під назвою SCO UNIX.

Т.ч., є дві гілки розвитку UNIX: та, яка базується на BSD і та, яка базується на System V. (тобто. Від AT&T і від Berkeley). Всі інші є їх клонами.

Для того, щоб дізнатися, до якої гілки відноситься система треба подивитися як відбувається висновок на принтер: якщо командою lp, то до System V, якщо lpr, то до BSD.

ОС Linux є клоном BSD.

Сьогодні існують десятки різних версій ОС UNIX. У багатьох випадках виробник ОС є і виробником апаратної платформи, для якої ця ОС призначена.

Наприклад,

Sun OS, Solaris Sun Microsystems

HP-UX Hewlett Packard

AIX IBM

IRIX Silicon Graphics

Digital UNIX Digital Equipment Corporation (DEC)

SCO UNIX Santa Cruz Operation

І т.д.

Це все комерційні версії, в яких творці намагалися якомога ефективніше вирішити питання реалізації тієї чи іншої підсистеми.

Існують і некомерційні UNIX-системи, наприклад, Linux і FreeBSD.

У 1991 році була створена некомерційна версія UNIX – Linux. Її автор – Лінус Торвальдс студент університету Хельсінкі. З тих пір популярність Linux стрімко зростає завдяки вільному поширенню цієї ОС. Багато програмістів взяли активну участь в її вдосконаленні, розширенні можливостей і усунення помилок.

Linux поширюється як ПО, що охороняється авторським правом за генеральною відкритою ліцензією (GPL) GNU в рамках Фонду безкоштовного програмного забезпечення. Основна вимога цієї ліцензії полягає в тому, що ви маєте право переробити і передати кому завгодно копії Linux. Вихідні тексти відкриті, що дозволяє легко змінювати або налаштовувати ПО відповідно до потреб конкретного користувача. Linux є переносимою системою, тобто існують версії, що працюють на комп'ютерах різних марок (PC, Apple Macintosh, робочі станції Sun). По суті, Linux є ПК-версією UNIX.

1.2 Основні характеристики системи

ОС UNIX має наступні основні характеристики:

- переносимість;
- витісняюча багатозадачність на основі процесів, що працюють в ізольованих адресних просторах в віртуальної пам'яті;
- підтримка одночасної роботи багатьох користувачів;
- підтримка асинхронних процесів;
- ієрархічна файлова система;
- підтримка незалежних від пристроїв операцій введення-виведення (через спеціальні файли пристроїв);
- стандартний інтерфейс для програм (програмні канали, IPC) і користувачів (командний інтерпретатор, що не входить в ядро ОС);
- вбудовані засоби обліку використання системи.

Особливості UNIX, що відрізняють дане сімейство від інших ОС:

Файлова система деревоподібна з урахуванням регістру символів в іменах, дуже слабкі обмеження на довжину імен.

Немає підтримки структурованих файлів ядром ОС, на рівні системних викликів файл є потік байт.

Командний рядок знаходиться в адресному просторі процесу, що запускається, а не вираховується системним викликом з процесу інтерпретатора команд.

Поняття «змінних оточення».

Запуск процесів викликом fork(), тобто можливість клонування поточного процесу з усім станом.

Введення / вивід тільки через дескриптори файлів.

Інтерпретатор команд – звичайний додаток, що спілкується з ядром звичайними системними викликами.

Команда командного рядка є не більше ніж ім'я файлу програми, не потрібно спеціальна реєстрація і спеціальна розробка програм як команд.

Не прийнятий підхід до програми, яка задає користувачеві питання про режими своєї роботи, замість цього використовуються параметри командного рядка.

Простір імен пристроїв на диску в каталозі /dev, піддається управлінню адміністратором.

Широке використання текстових файлів для зберігання налаштувань, на відміну від двійковій бази даних налаштувань, як, наприклад, в Windows.

Широке використання утиліт обробки тексту для виконання повсякденних завдань під управлінням скриптів.

«Розкрутка» ОС після завантаження ядра шляхом виконання скриптів стандартним інтерпретатором команд.

Широке використання конвеєрів (pipe).

Всі процеси, крім init, рівні між собою, не буває «спеціальних процесів».

Використання двох рівнів привілеїв процесора.

Концепція сигналу унікальна для UNIX.

UNIX існує вже чотири десятиліття. Це дуже великий термін для ОС. І сьогодні, незважаючи на появу більш простих і зручних з точки зору адміністратора систем, UNIX займає провідне місце. При цьому мова йде не про конкретну версію, а про систему як таку.

Перелічимо основні риси UNIX, що дозволяють зрозуміти причини довгожительства цієї системи:

1. Код системи написаний на мові високого рівня С, що зробило її простою для розуміння, змін і перенесення на інші платформи. За оцінками одного з творців UNIX, Денніса Рітчі, система на мові С мала на 20-40% більшій розмір, а продуктивність її була на 20% нижча від аналогічної системи, написаної на асемблері. Однак ясність і переносимість, а в результаті – і відкритість системи зіграли вирішальну роль в її популярності. Можна сміливо сказати, що UNIX є однією з найбільш відкритих систем.

2. UNIX – багатозадачна розрахована на багато користувачів система з широким спектром послуг. Один потужний сервер може обслуговувати запити великої кількості

користувачів. При цьому необхідно адміністрування тільки однієї системи. Система може виконувати різні функції – працювати як обчислювальний сервер, що обслуговує сотні користувачів, як сервер бази даних, як мережевий сервер, що підтримує найважливіші сервіси мережі (telnet, ftp, електронну пошту, службу імен DNS і т.д.), або навіть як мережевий маршрутизатор.

3. Наявність стандартів. Незважаючи на різноманіття версій UNIX, основою всього сімейства є принципово однакова архітектура і ряд стандартних інтерфейсів. Досвідчений адміністратор без великих труднощів зможе обслужити іншу версію системи, для користувачів перехід на іншу версію і зовсім може виявитися непомітним.

4. Простий, але потужний модульний призначений для користувача інтерфейс. Маючи в своєму розпорядженні набір утиліт, кожна з яких вирішує вузьку спеціалізовану задачу, ви можете конструювати з них складні комплекси.

5. Використання єдиної ієрархічної файлової системи, що легко обслуговується. Файлова система – це не тільки доступ до даних, що зберігаються на диску. Через уніфікований інтерфейс файлової системи здійснюється доступ до терміналів, принтерів, мережі і навіть до пам'яті.

6. Дуже велика кількість додатків, в тому числі вільно розповсюджуваних, починаючи від найпростіших текстових редакторів і закінчуючи потужними системами управління базами даних.

1.3 Відмінності між Windows і UNIX

Операційні системи UNIX і Windows досить сильно відрізняються в реалізації різних сервісів і служб.

В UNIX-подібних ОС графічна система існує окремо від ядра і функціонує як звичайна програма. В операційних системах Windows графічна система інтегрована в ядро. У разі використання операційної системи на робочій станції, особливо при запуску графікоємких додатків, можливо, краще, коли графічна система входить в ядро – в цьому випадку вона може швидше працювати. А при роботі на сервері краще відділення графічної системи від ядра ОС, так як вона завантажує пам'ять і процесор. У разі UNIX/Linux графічну систему можна просто відключити, до того ж, якщо системний адміністратор її все-таки хоче використовувати, наприклад, в Linux є кілька графічних оболонок на вибір, деякі з них (наприклад, WindowMaker) досить слабо завантажують машину. Ця ж особливість UNIX-подібних операційних систем дозволяє запускати ці ОС на машинах з досить скромними обсягами ОЗП і т.п. У разі Windows графічна система

занадто тісно інтегрована в ОС, тому вона повинна запускатися навіть на тих серверах, на яких вона зовсім не потрібна.

Наступна відмінність в методиці поділу прав доступу в Windows 2000 і UNIX/Linux. У першому – поділ прав доступу засноване на ACL (access control lists), тобто, наприклад, можна налаштувати систему таким чином, щоб адміністратор не мав можливості керувати файлами користувачів. У UNIX/Linux же завжди є привілейований користувач – root, який має доступ абсолютно до всього. Тобто теоретично модель безпеки в Windows краще: щоб повністю заволодіти добре налагодженою системою Windows, хакеру доведеться ламати більше, в UNIX / Linux же досить зламати доступ до root. (В UNIX / Linux використовуються більш старі технології, тим не менш, деякі дистрибутиви Linux зараз починають підтримувати ACL, серед них – ASPLinux 7.3 Server Edition). Але в Windows не так швидко, як в Linux, закладаються "дірки", що відноситься до плюсів відкритої моделі розробки. В результаті виявляється, що в Windows за статистикою більше дірок. Але, знову ж таки, точно про кількість дірок в Linux і Windows можна буде сказати тільки тоді, коли кількість користувачів обох видів ОС буде приблизно однаковим.

У Linux підтримуються кілька файлових систем, найбільш просунуті – це Ext3, Ext4, XFS. ОС Windows зав'язана за великим рахунком на одну файлову систему – NTFS або FAT 32. Файлові системи Ext3, Ext4, XFS за оцінками працюють швидше.

Принципова ж відмінність в тому, що в UNIX / Linux взагалі немає поняття диска, фізичного або логічного. Вся робота з пристроями зберігання даних організовується через спеціальні файли пристроїв, які відображають фізичний носій або його частини (розділи) в файлову систему.

Можна перерахувати ще ряд відмінностей UNIX-подібних операційних систем від Windows, наприклад, вбудовану підтримку віддаленого доступу в UNIX і відсутність її в Windows за замовчуванням (вона реалізується в серверних версіях Windows, а також за допомогою додаткових коштів, наприклад, Citrix). В UNIX / Linux і Windows сильно розрізняються мережеві підсистеми (IP-stack), по ряду оцінок мережева підсистема UNIX / Linux ефективніше.

1.4 Стандарти

Велика кількість різних варіантів системи UNIX привело до необхідності стандартизувати її, щоб спростити переносимість додатків і позбавити користувача від необхідності вивчати особливості кожного різновиду UNIX. З цією метою ще в 1980 р.

була створена для користувача група /usr/group. Найперші стандарти були розроблені в 1984-1985 рр. В даний час найбільш важливими є такі нормативні документи:

- POSIX 1003.1-1988, що визначає програмний інтерфейс додатків (API, Application Programming Interface). Він використовується не тільки в UNIX, але і в інших операційних системах. У 1990 він був прийнятий інститутом IEEE як IEEE 1003.1-1990.
- POSIX 1003.2-1992, що визначає поведінку утиліт, в тому числі командного інтерпретатора.
- POSIX 1003.1b-1 993, що доповнює POSIX 1003.1-1988. Визначає підтримку систем реального часу.
- POSIX 1003.1c-1995 року, що доповнює POSIX 1003.1-1988. Визначає нитки (threads), відомі також як pthreads.

Всі стандарти POSIX об'єднані в документі IEEE +1003.

На початку 1990-х років The Open Group запропонувала інший, схожий на POSIX стандарт – Common API Specification, або Spec 1170. Стандарт набув великої популярності, ніж POSIX, оскільки був доступний безкоштовно, в той час як IEEE вимагало чималу плату за доступ до свого стандарту.

У 1998 році були розпочаті роботи по об'єднанню даних стандартів. Завдяки цьому в даний час дані стандарти майже ідентичні. Спільний стандарт називається Single UNIX Specification Version 3 (SUS3) і доступний для безкоштовного завантаження в Інтернеті.

З метою сумісності кілька творців UNIX-систем запропонували використовувати ELF-формат систем SVR4 для довічних і об'єктних файлів. Єдиний формат повністю забезпечує відповідність довічних файлів в рамках однієї комп'ютерної архітектури.

Структура каталогів деяких систем, зокрема, GNU / Linux, визначена в стандарті Filesystem Hierarchy Standard. Однак у багатьох відношеннях цей тип стандарту є спірним, і він, навіть всередині спільноти GNU/Linux, далеко не універсальний.

2 АРХІТЕКТУРА ОС UNIX

Згадаймо, що під архітектурою операційної системи розуміють структурну і функціональну організацію ОС на основі деякої сукупності програмних модулів.

UNIX спочатку була задумана як система, незалежна від конкретної платформи, що вигідно відрізняло її від інших ОС.

2.1 Структурна організація ОС UNIX

Структуру UNIX найпростіше представити у вигляді двох шарів. Першим є ядро. Воно безпосередньо взаємодіє з обладнанням і забезпечує переносимість всього іншого програмного забезпечення на комп'ютери з різними апаратним забезпеченням. Другим шаром є програмне забезпечення, прикладне або системне: командний інтерпретатор, графічна оболонка і т. д. (рис. 2.1).

Ядро являє собою відносно невелику частину програм, яка дозволяє всім іншим програмам спілкуватися з периферійними пристроями, регулює доступ до файлів, управляє пам'яттю і процесами.

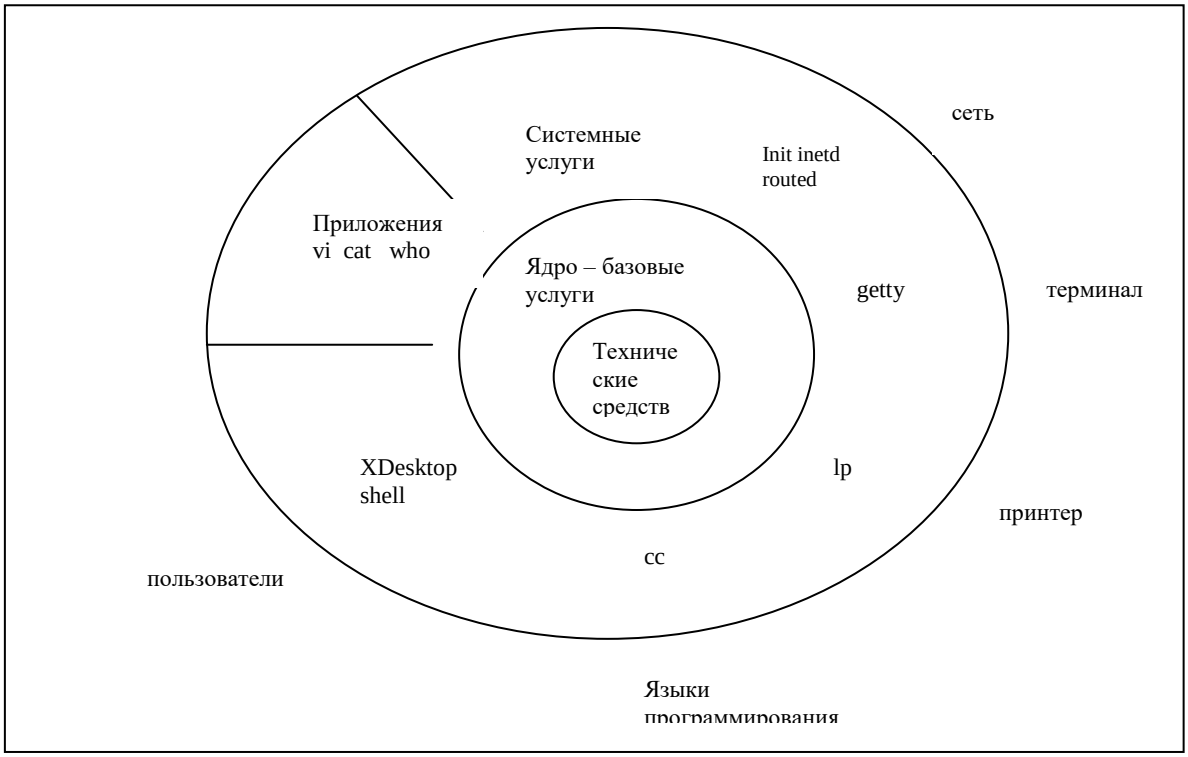

Рисунок 2.1 – Структура ОС UNIX

Таким чином, ядро спирається на апаратні засоби. А другий рівень структури ОС складають додатки як системні, що забезпечують функціональність системи, так і прикладні, що забезпечують користувальницький інтерфейс UNIX. Незважаючи на зовнішню різнорідність додатків, схеми їх взаємодії з ядром однакові.

2.1.1 Структура ядра ОС UNIX

Ядро – єдиний файл, що розміщується в кореневому каталозі, який не може перебувати де-небудь ще в діючій системі. Коли система завантажується, вона ініціює саме цей файл, який і залишається активним до закриття системи. Будь-який інший процес викликається при необхідності і зупиняється, якщо він не потрібен.

Основою операційної системи UNIX є ядро (kernel). Ядро являє собою спеціальну програму (або кілька програмних модулів, в разі модульного ядра), яка постійно знаходиться в оперативній пам'яті і працює, поки працює операційна система. Ядро безпосередньо взаємодіє з апаратною частиною комп'ютера, ізолюючи прикладні програми від особливостей її архітектури. До послуг ядра відносяться операції введення-виведення, створення і управління процесами, їх синхронізації і взаємодії між процесами. Всі додатки запитують послуги ядра за допомогою системних викликів.

Другий рівень складають додатки чи завдання, як системні, що визначають функціональність системи, так і прикладні, що забезпечують користувальницький інтерфейс. Незважаючи на зовнішню різнорідність додатків схеми їх взаємодії з ядром однакові.

Структура ядра представлена на рисунку 2.2.

Ядро складається з декількох підсистем, які завжди беруть участь в роботі UNIX, а також ряду завантажувальних модулів і драйверів пристроїв, які керують інтерфейсним устаткуванням.

Три основні підсистеми:

- файлова
- управління процесами і пам'яттю
- управління введенням-виведенням

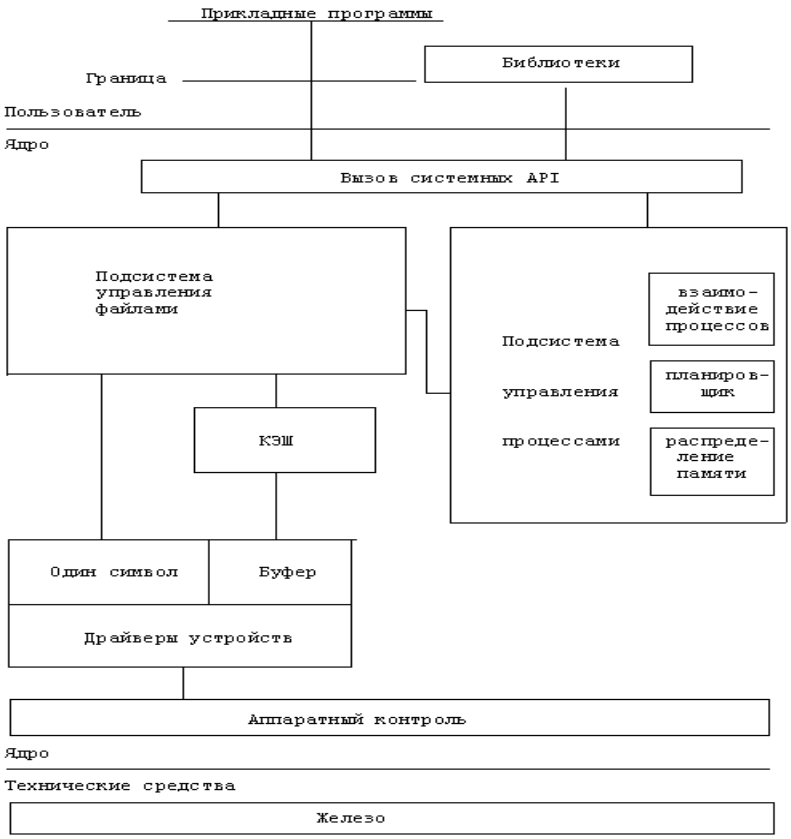

Рисунок 2.2 – Структура ядра UNIX

Файлова підсистема забезпечує уніфікований інтерфейс доступу до даних. Це означає, що одні й ті ж функції open (), read (), write () можуть використовуватися як при читанні або запису даних на диск, так і при виведенні тексту на принтер або термінал. Якщо дані направляються на периферійний пристрій, ф/пс забезпечує перенаправлення запитів відповідним модулям підсистеми вводу/виводу.

Файлова система UNIX характеризується:

- ієрархічною структурою,
- узгодженою обробкою масивів даних,
- можливістю створення і видалення файлів,
- динамічним розширенням файлів,
- трактуванням периферійних пристроїв як файлів.

Файлова система організована у вигляді дерева з однієї вихідної вершиною, яка називається коренем або кореневим каталогом. Кореневої каталог розгалужується на кілька підкаталогів. Кожен підкаталог, в свою чергу може поділятися на кілька додаткових каталогів. Ця структура перейшла з UNIX в більшість сучасних ОС.

Файлове дерево може бути довільного розміру. Але можуть існувати обмеження на довжину імені каталогу і загальну довжину колійного імені.

Підсистема управління процесами виконує такі функції:

- створення і видалення процесів
- розподіл системних ресурсів
- синхронізацію процесів
- міжпроцесна взаємодія

У загальному випадку число активних процесів перевищує число процесів комп'ютера, але в кожен момент часу може виконуватися тільки один процес. До завдань ОС входить управління доступом процесів до обчислювальних ресурсів так, щоб створювалося відчуття одночасного виконання декількох завдань.

У підсистему управління процесами входить спеціальна програма-планувальник. Планувальник запускає процеси на виконання відповідно до їх пріоритетів, вирішує конфлікти між процесами в конкуренції за системні ресурси, стежить за тим, щоб якийсь процес не захопив монопольно системні ресурси, що розділяються.

Це стандартний планувальник UNIX використовує чергу процесів. Зазвичай використовуються дві черги: стандартна черга і черга "реального часу". Зазвичай процеси з черги "реального часу" запускаються раніше процесів зі стандартної черги. Усередині кожної черги високопріоритетні процеси активізуються раніше менш пріоритетних.

Планувальник Linux сильно відрізняється від інших планувальників систем типу UNIX. Замість планування запуску процесів з високим пріоритетом в першу чергу, Linux використовує кругове планування, однак дозволяє процесам з високим пріоритетом запускатися швидше і на більш тривалі терміни.

Планувальник Linux більш ефективний з точки зору продуктивності.

Модуль управління пам'яттю забезпечує розміщення оперативної пам'яті. Якщо для всіх процесів оперативної пам'яті недостатньо, частини процесів переміщаються у вторинну пам'ять (в спеціальній частині жорсткого диска).

UNIX як всі сучасні ОС реалізує механізм віртуальної пам'яті: процес виконується у власному логічному адресному просторі, яке може значно перевищувати доступну фізичну пам'ять.

Модуль взаємодії між процесами відповідає за обмін сигналами між процесами.

Взаємодія прикладних задач з ядром відбувається за допомогою стандартного інтерфейсу системних викликів. Процес посилає системний виклик ядра, зовні схожий на звичайний виклик бібліотечної функції. Ядро виконує дії, запитані процесом, і повертає необхідні дані.

Підсистема введення/виведення виконує запити інших підсистем. При цьому вона забезпечує необхідну буферизацію даних і взаємодіє з драйверами пристроїв – спеціальними модулями ядра, безпосередньо обслуговуючими зовнішні пристрої.

Всі звернення до ядра системи можна розділити на дві категорії: програма викликає підсистему управління файлами або підсистему управління процесами.

Перша відповідає за все, що пов'язано з файлами: управління, розміщення, доступ. Процеси ж – це, в загальному випадку, будь-які програми, які виконуються. Тому підсистема управління процесами служить для їх життєздатності, синхронізації і управління.

Важливо також і те, що файлова підсистема і підсистема управління процесів можуть спілкуватися один з одним: будь-який процес може викликати системні API для роботи з файлами.

Ядро – це зв'язковий, до якого звертаються за допомогою системних викликів (запитуючи якусь послугу). Зв'язок цей – не односторонній: ядро може і повертати в разі потреби якісь дані. Основною перевагою ядра є сувора стандартизація системних API. За рахунок цього багато в чому досягається переносимість коду між різними версіями UNIX і абсолютно різним апаратним забезпеченням.

Системні виклики забезпечують:

- зіставлення дій користувача з запитами драйверів пристроїв;
- створення і припинення процесів;
- реалізацію операцій введення-виведення;
- доступ до файлів і дисків;
- підтримку функцій терміналу.

Системні виклики перетворять процес, який працює в режимі користувача, в захищений процес, який працює в режимі ядра. Це дозволяє процесу викликати захищені процедури ядра для виконання системних функцій.

Коли інсталюється ОС, будується ядро саме для роботи на цій машині. Враховуються всі апаратні засоби і будується найбільш оптимальний для конкретної машини файл.

Ядро написано на мові С, в разі необхідності його можна переробити, що дає додаткову мобільність системи. На С звичайно написані не всі без винятку частини ядра, а тільки ті, які має сенс переробляти.

2.2 Функції ОС UNIX

ОС UNIX реалізують наступний ряд функцій:

• Забезпечення інтерфейсу користувача.

В даний час ОС UNIX надає користувачеві можливість вибирати зручний інтерфейс. Графічний багатовіконний інтерфейс, техніка перетягування – все це стирає відмінності в роботі з UNIX і, наприклад, Widows. Однак, для UNIX в будь-якому випадку базовим все одно залишається інтерфейс командного рядка.

Інтерфейс здійснюється за допомогою спеціальної програми, яка називається командним інтерпретатором (або оболонкою).

Той факт, що командний інтерпретатор є звичайним процесом, дозволяє просто замінити його на інший.

Командний інтерпретатор займає важливе місце в ОС завдяки таким властивостям:

- це перша програма, з якої починається робота користувача
- це зручний засіб програмування
- основна ініціалізація ОС відбувається в результаті виконання сценаріїв командного інтерпретатора.

Командна оболонка або командний інтерпретатор – програма, яка використовується для організації діалогу людини і системи.

Існує чимало командних інтерпретаторів.

Sh (Bourn shell). Початковий командний інтерпретатор. Розроблено Стівеном Борном в середині 70-х років. Як командна мова забезпечує взаємодію з конкретним процесом засобами UNIX та виконання інших команд в системі. Як мова програмування містить безліч конструкцій для розробки складних програм. У ньому відсутні засоби редагування ком рядка і можливість управління завданнями, наявні в більш пізніх версіях. Однак використовується і досі (в System 5).

Csh (c-shell). Розроблено Біллом Джоем з Каліфорнійського університету. Відрізняється поліпшеними діалоговими можливостями, способом привласнення та експортування змінних в середу, керуючими конструкціями і рядом інших моментів; теж підтримує історію та редагування командного рядка. Погано підтримує перенаправлення вводу / виводу, мало придатна для створення складних сценаріїв.

Kshell (korn shell). Написана Девідом Корном (AT&T). Сумісна з shell, але включає в себе багато засобів cshell (підтримку псевдонімів, історія команд, редагування)

Bash (Bourne again shell). Проведена Free Software Foundation. Вільно розповсюджуваний у вигляді вихідних текстів інтерпретатор, який об'єднує все краще з інших інтерпретаторів з зручними можливостями редагування командного рядка і роботи

з історією команд. В даний час – фактичний стандарт. Можливості відповідають можливостям kshell, але поширюється безкоштовно.

Tcsh (Tom c shell) Повністю сумісний з csh, але в ньому усунені помилки і вдосконалений інтерфейс.

Команди, що запускаються в командному інтерпретаторі можуть бути або функціями, що визначені користувачем, або вбудованими командами інтерпретатора, або виконуваними файлами – прикладними програмами і утилітами. У будь-якому випадку, синтаксис їх виклику однаковий.

Загальний вигляд командного рядка

<ім'я команди> <опції> <аргументи>

ім'я команди – ім'я відповідного файлу

опції – режими виконання команди

аргументи – інформація, необхідна для виконання команди (об'єкти над якими виконуються операції)

Аргументи команди в більшості випадків – це імена файлів.

Наприклад, команда *ls -l a.txt* означає: виконати команду ls (відображення інформації про файл) з ключем -l (розширена інформація) для файлу a.txt

Перш ніж виконати команду інтерпретатор виконує наступні дії:

- аналізує синтаксис і якщо виявлена помилка, видає повідомлення
- виконує підстановки: замінює всі зазначені імена змінних їх значеннями, формує списки файлів, замінюючи шаблони
- робить відповідні перенаправлення вводу / виводу
- передає команді аргументи і виконує її.

Послідовність команд можна об'єднати в одному файлі і виконувати його як звичайну команду. Такий файл буде називатися сценарієм командного процесора.

• Забезпечення автоматичного запуску.

Всі операційні системи забезпечують свій автоматичний запуск. Для дискових операційних систем в спеціальній (системній) області диска створюється запис програмного коду. Звернення до цього коду виконують програми, що знаходяться в базовій системі вводу-виводу (BIOS). Завершуючи свою роботу, вони дають команду на завантаження і виконання вмісту системної області диска.

Рівень виконання

Рівень виконання (run level) – це один з можливих режимів роботи системи. Кожен рівень виконання позначається номером (від 0 до 6). У будь-який момент часу система може перебувати на одному з рівнів виконання. Зміна режиму роботи проводиться за

допомогою виклику telinit з параметром, що вказує номер рівня виконання, на який слід перейти, наприклад, telinit 5.

Рівень 0 – зупинка системи.

Рівень 1 – однокористувальницький режим роботи – система ініціалізує мінімум служб і надає командний рядок суперкористувачеві без проведення аутентифікації. Як правило, цей режим використовується для відновлення системи.

Рівень 2 – розрахований на багато користувачів режим – користувачі можуть працювати на різних терміналах, вхід в систему з процесом аутентифікації.

Рівень 3 – розрахований на багато користувачів мережевий режим – багатокористувацький режим, в якому здійснюється настройка мережі і запускаються різні мережеві служби.

Рівень 4 – практично не використовується, зарезервований для визначення власних режимів роботи ОС.

Рівень 5 – запуск графічної підсистеми – на додаток до рівня 3 виробляється також старт графічної підсистеми X Window, реєстрація в системі здійснюється також в графічному режимі.

Рівень 6 – перезавантаження системи – при включенні цього режиму зупиняються всі запущені програми і проводиться перезавантаження.

Перехід на певний рівень виконання має на увазі виконання певного набору процедур ініціалізації і певний набір системних служб, які повинні виконуватися на даному рівні. Конкретний список таких процедур і служб може бути заданий адміністратором системи.

У файлі конфігурації /etc/inittab містяться параметри, що визначають, що повинен робити процес init на кожному з рівнів виконання. У ньому задаються команди, які повинні бути виконані (або повинні продовжувати виконуватися), коли система переходить на певний рівень виконання. Рівень виконання за замовчуванням також задається в файлі /etc/inittab в рядку типу id:3:initdefault.

Перезапуск і зупинка системи

Команда shutdown – це найбезпечніший і найбільш коректний спосіб зупинити або перезавантажити систему.

Прапори:

-h виконати зупинку системи

-r виконати перезавантаження після зупинки системи

-t time вказує затримку (в секундах) перед зупинкою системи; відсутність прапора означає негайну зупинку

-f після перезавантаження запустити утиліту перевірки дисків fsck

Наприклад:

shutdown -h -t 10 # зупинка через 10 секунд

shutdown -r now # негайне перезавантаження

shutdown -r 0 # негайне перезавантаження

shutdown -f -t 30 "THE POWER IS FAILING"

Зазначено повідомлення, що посилається під час очікування зареєстрованим користувачам.

Команда halt виконує всі основні операції, необхідні для зупинки системи. Щоб викликати цю команду, можна в командному рядку вказати shutdown -h або безпосередньо halt. Команда halt реєструє в журнальному файлі подію зупинки, знищує несуттєві процеси, виконує команду sync (для запису вмісту оперативної пам'яті на диск – синхронізацію), чекає завершення операцій дискової записи, а потім припиняє роботу ядра.

При вказівці команди halt -n команда sync пригнічується. Команда halt -q ініціює майже негайну зупинку без синхронізації, знищення процесів і реєстрації події.

Команда reboot ідентична команді halt. Різниця полягає в тому, що система перезавантажується, а не зупиняється. Режим перезавантаження викликається також командою shutdown -r. Крім цього, команда reboot також підтримує прапори -n і -q.

Команда poweroff використовується для зупинки системи і вимикання комп'ютера.

Передача процесу init сигналу TERM: init – це завжди процес з ідентифікатором 1. З метою відправки сигналу можна скористатися командою kill, яка приймає як параметр ім'я (номер) сигналу і номер процесу: kill -TERM 1

Знищення процесу init за допомогою передачі їй сигналу KILL: процес init настільки важливий для роботи системи, що якщо його знищити за допомогою команди kill -KILL 1 або kill -9 1, то більшість систем автоматично перезавантажиться. Це дуже грубий спосіб перезавантаження.

2.2.1 Середа користувача UNIX

Сценарій роботи

В ОС UNIX реалізується наступний сценарій роботи в системі:

- при включенні терміналу активізується процес getty, який є сервером термінального доступу і запускає програму login, яка в свою чергу запитує у користувача ім'я та пароль;

- якщо користувач зареєстрований і ввів правильний пароль, login запускає програму, яка вказана в останньому полі запису користувача в файлі /etc/passwd, тобто командний інтерпретатор;

- командний інтерпретатор виконує командний файл ініціалізації і видає на термінал користувача запрошення. З цього моменту користувач може вводити команди;

- командний інтерпретатор зчитує ввід користувача, виробляє синтаксичний аналіз введеного рядка, підстановку шаблонів і виконує дію або повідомляє про помилку;

- по закінченню роботи користувач завершує роботу, вводячи команду exit.

Файл ініціалізації

Отже, при вході користувача в систему виконується його командний файл ініціалізації.

Файл ініціалізації виконує наступні функції:

- встановлює шлях пошуку програми;
- ініціалізує термінал;
- визначає розташування поштової скриньки.

Файл ініціалізації знаходиться в особистому каталозі користувача. Для різних командних інтерпретаторів виконуються різні файлу ініціалізації.

sh .profile

csh .login і .cshrc

bash .profile і .kshrc

ksh .profile і .bashrc

3 КОРИСТУВАЧІ ТА ГРУПИ СИСТЕМИ

Перш ніж почати роботу в UNIX-подібній операційної системи, необхідно стати користувачем системи.

Користувач, з точки зору системи, не обов'язково людина.

Користувач – це об'єкт, який володіє певними правами, може запускати на виконання програми і володіти файлами.

Як користувачів можуть виступати, наприклад, віддалені комп'ютери або групи користувачів з однаковими правами і функціями. Такі користувачі називаються псевдокористувачами. Більшість користувачів, як правило, є реальними людьми, які зареєстровані в системі і запускають ті чи інші програми.

Кожен користувач має своє унікальне ім'я (login), але система розрізняє користувачів за ідентифікатором, відповідному імені. Він називається ідентифікатором користувача (UID). Кожен користувач є членом однієї або декількох груп.

Група – це користувачі, які мають схожі завдання. Кожна група має своє унікальне ім'я. Система ж розрізняє групи за ідентифікатором (GID).

UID і GID визначають, якими правами володіє користувач в системі.

Присвоєння унікального ідентифікатора користувача виконується при створенні системним адміністратором нового реєстраційного імені. Значення ідентифікатора користувача і групи – не просто числа, які ідентифікують користувача, – вони визначають власників файлів і процесів.

В системі існує один користувач, що володіє необмеженими можливостями. Це привілейований користувач (root). Його UID = 0 GID = 0.

Інформація про користувачів зберігається в файлі /etc/passwd, а інформація про користувачів в файлі /etc/group.

Файл /etc/shadow використовується в системах з тіньовим зберіганням паролів, де вони винесені з доступного всім користувачам на читання файлу /etc/passwd для підвищення безпеки системи.

Кожен рядок файлу /etc/passwd складається з семи полів, розділених двокрапкою

Ось ці поля:

1 - name – реєстраційне ім'я користувача

2 - passwd-encod – пароль користувача в закодованому вигляді

3 - UID – ідентифікатор користувача

4 - GID – ідентифікатор первинної групи

5 - comments – коментарі, зазвичай це повне ім'я користувача

6 - home-dir – домашній каталог користувача

7 - shell – ім'я програми, яку ОС запускає в якості командного інтерпретатора.

Після установки UNIX зазвичай містить кілька зареєстрованих користувачів. Це стандартні користувачі. Перелічимо основні з них (таблиця 3.1). (У різних версіях UID цих користувачів може відрізнятися.)

Таблиця 3.1 – Основні користувачі системи

root	СуперКористувач. Для виконання більшості функцій адміністрування потрібно вхід з цим ім'ям.
adm	Псевдокористувач, володіє файлами системи ведення журналів.
bin	Зазвичай власник виконуваних файлів, що є командами UNIX.
cron	Псевдокористувач, власник відповідних файлів, від імені якого запускаються програми за розкладом.
lp (lpd)	Псевдокористувач, від імені якого виконуються процеси системи друку.
nobody	Псевдокористувач, використовуваний в роботі NFS.
uucp	Псевдокористувач, що дозволяє передавати поштові повідомлення між UNIX-хостами (Unix to Unix copy).

Нова система також матиме кілька стандартних груп.

Таблиця 3.2 – Основні групи системи

Root (wheel)	Адміністративна група
User (users, staff)	Група, до якої, за замовчуванням, включаються всі звичайні користувачі.

Існує команда su для тимчасової зміни діючого (ефективного) ідентифікатора користувача і сеансу користувача. Вона має наступний синтаксис:

su [-] [реєстраційне_ім'я [аргументи ...]

Команда su без зазначення реєстраційного імені дозволяє отримати права користувача root. При цьому необхідно знати і правильно ввести пароль користувача root.

Команда passwd дозволяє будь-якому користувачеві змінити пароль або отримати список атрибутів поточного пароля для свого реєстраційного імені.

Для перегляду бази даних облікових записів системи призначена команда logins. За замовчуванням видається наступна інформація: реєстраційне ім'я, ідентифікатор користувача, ім'я основної групи, ідентифікатор основної групи і поле опису облікового запису в файлі /etc/passwd. Результат сортується за ідентифікатором користувача, в результаті чого спочатку йдуть системні реєстраційні імена, а потім – призначені для користувача.

Для отримання списку користувачів, що працюють зараз в системі, використовується команда who.

Утиліта who видає ім'я користувача, термінал, час реєстрації, час, що минув після останньої виконаної команди, а також ідентифікатор процесу командного інтерпретатора. Для отримання цієї інформації вона переглядає файл /var/adm/utmp.

Оскільки база даних облікових записів організована у вигляді звичайних текстових файлів, основні завдання управління обліковими записами можуть вирішуватися за допомогою звичайного текстового редактора. Однак оскільки при цьому потрібно узгоджена зміна декількох файлів, в системі для управління обліковими записами пропонується ряд утиліт командного рядка, засоби на основі меню або на основі графічного інтерфейсу.

Для створення, зміни та видалення облікових записів всі версії UNIX-подібних ОС пропонують три команди, useradd, usermod і userdel, відповідно.

Ці команди дозволяють виконати тільки узгоджені і допустимі зміни в файлах /etc/passwd, /etc/shadow і /etc/group. Команди управління обліковими записами, в загальному випадку, може виконувати тільки користувач root.

Для створення, зміни та видалення груп є три команди, **groupadd, groupmod** і **groupdel**, відповідно. Ці команди дозволяють виконати тільки узгоджені і допустимі зміни в файлі /etc/group. Команди управління групами, в загальному випадку, може виконувати тільки користувач root.

4 ПОНЯТТЯ І ПРИЗНАЧЕННЯ ФАЙЛУ

Файл – це іменована область зовнішньої пам'яті, в яку можна записувати і з якої можна зчитувати дані.

Основні цілі використання файлу:

- **Довготривале і надійне зберігання інформації**. Довготривалість досягається за рахунок використання запам'ятовуючих пристроїв, що не залежать від харчування, а висока надійність визначається засобами захисту доступу до файлів і загальною організацією програмного коду ОС, при якій збої апаратури найчастіше не руйнують інформацію, що зберігається в файлах.

- **Спільне використання інформації**. Файли забезпечують природний і легкий спосіб поділу інформації між додатками і користувачами за рахунок наявності зрозумілого людині символьного імені та сталості інформації, що зберігається і розташування файлу. Файл може бути створений одним користувачем, а потім використовуватися зовсім іншим користувачем, при цьому творець файлу або адміністратор можуть визначити права доступу до нього інших користувачів.

4.1 Імена файлів в UNIX-подібних ОС

В UNIX-подібних ОС підтримується три способи вказівки імен файлів:

Коротке ім'я. Ім'я, яке не містить спеціальних метасимволів коса риска (/), є коротким ім'ям файлу. За коротким ім’ям можна послатися на файли поточного каталогу. Наприклад, команда *ls -l .profile* вимагає отримати повну інформацію про фото .profile в поточному каталозі.

В ієрархічних файлових системах різним файлам дозволено мати однакові прості символьні імена за умови, що вони належать різним каталогам.

Відносне ім'я. Ім'я, не починається із символу косої риски (/), але включає такі символи. Воно посилається на файл щодо поточного каталогу. При цьому для посилання на файл або каталог у якомусь іншому каталозі використовується метасимвол коса риска (/). Наприклад, команда *ls -l ../.profile* вимагає отримати повну інформацію про файл .profile в батьківському каталозі поточного каталогу, а команда *cat doc/text.txt* вимагає видати вміст файлу text.txt в підкаталозі doc поточного каталогу.

Повне (абсолютне) ім'я. Ім'я, що починається із символу косої риски (/). Воно посилається на файл щодо кореневого каталогу. Це ім'я ще називають абсолютним, так як

воно, на відміну від попередніх способів завдання імені, посилається на один і той же файл незалежно від поточного каталогу. Наприклад, команда *ls -l /home/user01/.profile* вимагає отримати повну інформацію про файл .profile в каталозі /home /user01 незалежно від того, в якому каталозі виконується.

Між файлом і його повним ім'ям є взаємно однозначна відповідність.

Інші символи, крім косою риси, не мають в іменах файлів UNIX особливого значення (це не метасимволи). Зокрема, немає системного поняття розширення файлу. Імена файлів чутливі до регістру.

В ОС UNIX немає теоретичних обмежень на кількість вкладених каталогів. Проте, в кожній реалізації є практичні обмеження на максимальну довжину імені файлу, яке вказується в командах (як і на довжину командного рядка в цілому).

4.2 Типи файлів

У UNIX існує кілька типів файлів, що розрізняються за функціональним призначенням і діям операційної системи при виконанні тих чи інших операцій над ними:

- Звичайний файл (regular file);
- Каталог (directory);
- Символічне або непряме посилання (symbolic (soft) link);
- Блок-орієнтований або блоковий файл пристрою (block special device file);
- Байт-орієнтований або символьний файл пристрою (character special device file);
- Іменований канал (FIFO, named pipe);
- Доменне гніздо UNIX (socket).

У різних версіях системи можуть підтримуватися не всі файли.

Звичайний файл

Являє собою найбільш загальний тип файлів, призначений для зберігання даних в деякому форматі. Для операційної системи такі файли являють собою просто послідовність байтів. До цих файлів відносяться текстові файли, двійкові дані і програми, що виконуються. В UNIX-подібних ОС не передбачено структурування вмісту на рівні ядра, але підтримуються і послідовний і прямий методи доступу.

У довгому лістингу ознакою звичайного файлу є дефіс (-) в першій позиції першого стовпця: `-rw-rw-r--  1 root  sys  8296 Фев 23 15:39 ps_data`

Каталог

Це набір байтів, що містить інформацію про імена файлів в каталозі в оголошеному форматі. За допомогою каталогів формується логічне дерево файлової системи. Каталог – це файл, який містить імена файлів, що знаходяться в ньому, а також покажчики на

додаткову інформацію – метадані, що дозволяють операційній системі виконувати дії з цими файлами.

Каталоги визначають положення файлу в дереві файлової системи. Будь-який процес, що має право на читання каталогу, може прочитати його вміст, але тільки ядро має право на запис даних каталогу (рисунок 4.1).

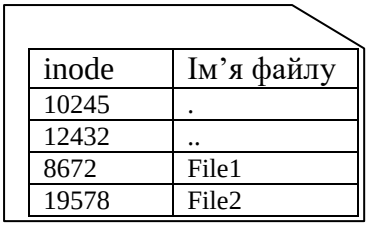

Рисунок 4.1 – Структура каталогу

Кожен каталог містить, щонайменше, два файли:

- «.» - сам каталог;

- «..» - батьківський каталог.

Каталог, що містить тільки ці два каталоги вважається порожнім.

У довгому лістингу ознакою каталогу є символ ***d*** в першій позиції першого стовпця: ***d****rwxr-xr-x 2 informix informix 115 Лют 24 13:05 txt*

Створюється каталог командою mkdir. Порожній каталог віддаляється командою rmdir. Для видалення непорожнього каталогу використовується команда rm -r.

Посилання

Каталог містить імена файлів і покажчики на їх метадані. Така архітектура дозволяє одному файлу мати кілька імен в файлової системі. Імена жорстко пов'язані з метаданими і, відповідно, з даними файлу, в той час як сам файл існує незалежно від того, як його називають в файлової системі.

Стандарт POSIX вимагає реалізувати підтримку двох типів посилань – жорстких і символічних.

Жорстким посиланням (hard link) вважається елемент каталогу, який вказує безпосередньо на деякий індексний дескриптор (рисунок 4.2). По суті це додаткове ім'я файлу. Жорсткі посилання можуть створюватися тільки в межах однієї фізичної файлової системи. Створюється таке посилання за допомогою команди ln. Дозволяється давати додаткове ім'я тільки існуючого файлу. При цьому в метаданих файлу збільшується на 1 кількість жорстких посилань. Після створення обидва імені є рівноправними.

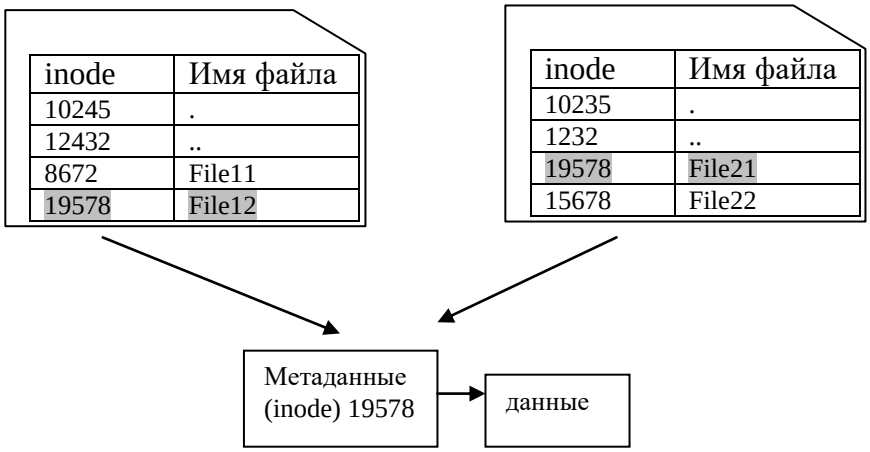

Рисунок 4.2 – Жорстке посилання

Кількість жорстких посилань файлу (а також кількість файлів в каталозі, якщо файл є каталогом) відображається у другому полі довгого лістингу:

*drwxr-xr-x **2** informix informix 115 Лют 24 13:05 txt*

Видаляється жорстке посилання командою rm. При цьому число жорстких посилань в метаданих файлу змінюється на 1. Коли воно стає рівним нулю, знищується описувач файлу і звільняється місце, займане файлом в файлової системі.

Символічне посилання (symbolic link) – це спеціальний файл, який містить шлях до іншого файлу. Вказівка на те, що даний елемент каталогу є символічною посиланням, знаходиться в індексному дескрипторі. Тому звичайні команди доступу до файлу замість отримання даних з фізичного файлу, беруть їх з файлу, ім'я якого наведено у посиланні. Така операція називається «розіменування» або «проходженням символічного посилання». Цей шлях може вказувати на що завгодно: це може бути каталог, він може навіть перебувати в іншій фізичній файлової системи, більш того, зазначеного файлу може і зовсім не бути.

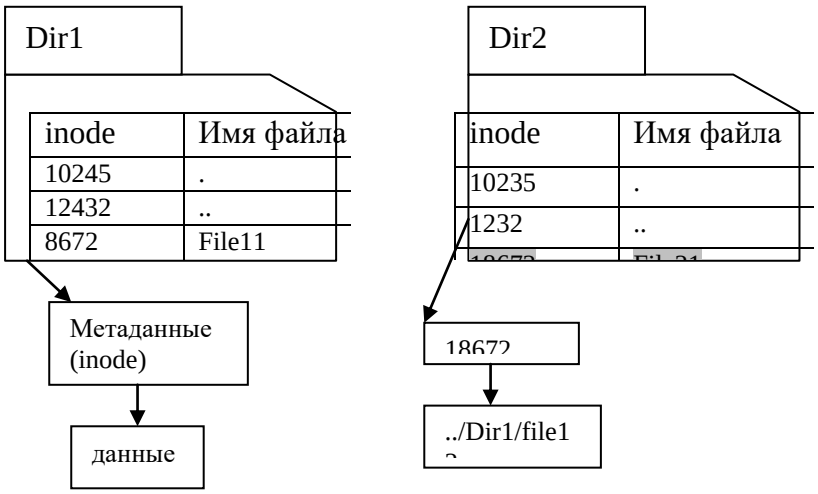

Рисунок 4.3 – Символічне посилання

Деякі системи накладають обмеження на кількість символічних зв'язків в дорозі. POSIX вимагає, щоб їх підтримувалося не менше 20, але дійсне значення залежить від конкретної реалізації.

У довгому лістингу ознакою символічної зв'язку є символ ***l*** в першій позиції першого стовпця: ***l****rwxr-xr-x 1 informix informix 115 Лют 24 13:05 txt*

Спеціальний файл пристрою

Забезпечує доступ до фізичних пристроїв. Для користувача ОС UNIX трактує пристрій так, як якщо б вони були файлами. Файли пристроїв дозволяють системі взаємодіяти з апаратними засобами і периферійними пристроями системи. За всю роботу по управлінню конкретним пристроєм відповідає спеціальна програма, яка називається *драйвером пристрою*.

В UNIX розрізняють символьні (character special device) і блокові (block special device) файли пристроїв. Доступ до пристроїв здійснюється шляхом відкриття, читання і записи в спеціальний файл пристрою.

Символьні файли пристроїв використовуються для небуферізованного обміну даними з пристроєм. Блокові файли пристроїв дозволяють проводити обмін даними у вигляді пакетів фіксованої довжини – блоків.

У довгому лістингу ознакою спеціального символьного і блокового пристроїв є символи ***c*** і ***b*** в першій позиції першого стовпця, відповідно.

FIFO - іменований канал

Цей файл використовується для зв'язку між процесами, що виконуються на одному комп'ютері, за принципом черги. Іменовані канали вперше з'явилися в UNIX System V, але більшість сучасних систем підтримують цей механізм.

У довгому лістингу ознакою іменованого каналу є символ ***p*** в першій позиції першого стовпця.

Сокет

Сокети дозволяють представити у вигляді файлу в логічній файлової системи мережеве з'єднання. У довгому лістингу ознакою сокета є символ ***s*** в першій позиції першого стовпця.

4.3 Права доступу до файлів

Права доступу до файлу є частиною системного захисту UNIX-подібних ОС. У кожного файлу є власник, який і встановлює права доступу. Тільки власник (і привілейований користувач) може змінювати права доступу.

Для кожного файлу визначаються:

- Права власника

- Права групи

- Права інших користувачів

Власник файлу це завжди один користувач. Група – один або кілька користувачів. Користувачі об'єднуються в групи з метою колективного використання файлів членами однієї групи.

Можливі дії з файлами:

- читання

- запис

- виконання

Право читання для всіх типів файлів має один і той же сенс: воно дозволяє читати вміст файлу (отримувати лістинг каталогу командою ls).

Право запису також має один і той же сенс: воно дозволяє редагувати цей файл. Для каталогу – це можливість змінювати його вміст, тобто створювати і видаляти файли.

Якщо для деякого файлу встановлений біт виконання, то файл може виконуватися як команда. У разі встановлення цього біта для каталогу, цей каталог можна зробити поточним (перейти в нього командою cd). Якщо це право для каталогу не встановлено, то права на читання і запис не враховуються.

Встановлений біт зміни ідентифікатора користувача SUID означає, що доступний користувачеві на виконання файл буде виконуватися з правами власника файлу, а не користувача, що викликав файл (як це зазвичай відбувається).

Встановлений біт зміни ідентифікатора групи SGID означає, що доступний користувачеві на виконання файл буде виконуватися з правами групи-власника файлу, а не користувача, що викликав файл (як це зазвичай відбувається).

Прикладом може бути утиліта passwd, що дозволяє користувачеві змінювати свій пароль. Для зміни пароля потрібно редагувати вміст файлу /etc/passwd і /etc/shadow. Надати права всім змінювати вміст цих файлів неможливо. Установка SUID для утиліти passwd дозволяє вирішити цю проблему. Власником файлу /usr/bin/passwd, в якому зберігається утиліта є привілейований користувач. Хоч би хто запустив утиліту на виконання, на час роботи даної програми отримує права суперкористувача, а значить, може вносити зміни в системних файлах.

Якщо біт SGID встановлений для файлу, що не доступний для виконання, він означає обов'язкове блокування, тобто незмінність прав доступу на читання та запис поки файл відкритий певною програмою.

Цей біт може бути встановлений і для каталогу, щоб управляти груповою приналежністю файлу, що встановлюється за умовчанням, що створюються в даному каталозі (не група-власника файлу, а успадковує права каталогу).

Встановлений клейкий біт для звичайних файлів раніше (за часів PDP-11) означав необхідність зберегти образ програми (тобто код і дані) в пам'яті після виконання (для прискорення повторного завантаження). Зараз при установці звичайним користувачем він скидається. Значення цього біта при установці користувачем root залежить від версії ОС і іноді необхідно. Так, в ОС Solaris необхідно встановлювати клейкий біт для звичайних файлів, використовуваних в якості області підкачки.

Установка клейкого біта для каталогу означає, що файл в цьому каталозі може бути видалений або перейменований тільки в наступних випадках:

- користувачем-власником файлу;
- користувачем-власником каталогу;
- якщо файл доступний користувачеві на запис;
- користувачем root.

Прикладом може служити каталог /tmp, який відкритий на запис для всіх користувачів, але в якому небажано видаляти чужі тимчасові файли.

4.4 Управління правами доступу

Змінювати права доступу до файлів і каталогів може їх власник або привілейований користувач. Для зміни прав доступу до файлів і каталогів застосовується команда *chmod*. Аргументи цієї команди можуть бути задані або в символьному, або в числовому (абсолютному) форматі.

Синтаксис команди chmod при використанні символьного формату:

chmod [хто] оператор права ім'я_файлу.

Параметр *хто* може приймати такі значення: u - власник, g - група, o - сторонні користувачі, a - всі користувачі.

Параметр *оператор* може приймати такі значення: + - додавання прав до наявних, - - видалення прав з наявних, = - установка прав замість наявних.

Параметр *права* може приймати такі значення: r - читання, w - запис, x - виконання, s - SUID або SGID, t - sticky, u - установка тих же прав, що і у власника, g - установка тих же прав, що і у групи, o - установка тих же прав, що і у сторонніх користувачів.

Приклади:

chmod a-x myfile # Відібрати у всіх категорій користувачів право на виконання.

chmod g-s, o + r myfile # Зняти SGID і додати стороннім користувачам право на читання.

Синтаксис команди chmod при використанні числового формату:

chmod права ім'я_файлу.

Тут параметр *права* являє собою вісімкове число. У найпростішому випадку воно складається з трьох трибітових наборів, кожен з яких відноситься до однієї з категорій користувачів. Старший біт відповідає дозволу на читання (1 - встановлено, 0 - знято), середній – дозволу на запис, а молодший – дозволу на виконання.

Наприклад, записані в символьній формі права доступу *rwxrwxrwx* мають числовий еквівалент 777, а права *rw-r-x* --- мають числовий еквівалент 650.

При установці/зняття спеціальних прав доступу зліва додається ще один трибітовий набір:

SUID	SGID	sticky	Символьний запис прав доступу
0	0	0	- - - - - - - - -
0	0	1	- - - - - - - - t (при встановленому x для сторонніх користувачів) - - - - - - - - T (x для сторонніх користувачів відсутнє)
0	1	0	- - - - - s - - - (при встановленому x для групи) - - - - - S - - - (x для групи відсутнє)
0	1	1	Встановлені SGID и sticky
1	0	0	- - s - - - - - - (при встановленому x для власника) - - S - - - - - - (x для власника відсутнє)
1	0	1	Встановлені SUID і sticky
1	1	0	Встановлені SGID і SGID
1	1	1	Встановлені SGID, SGID і sticky

Наприклад, записані в символьній формі права доступу *rw-rs - x* мають числовий еквівалент 2651, права *rwSr-sw-* мають числовий еквівалент 6652, права *r-xr-s-wT* мають числовий еквівалент 3552, а права *rw-rS-wt* мають числовий еквівалент 3643.

Приклади:

chmod 504 myfile # Власник має права читання і виконання, група не має ніяких прав і сторонні користувачі мають право читання.

chmod 5765 myfile # Для файлу встановлено SUID і sticky, власник має права читання, запису та виконання, група має права читання і запису і сторонні користувачі мають права читання і виконання.

5 ФАЙЛОВА СИСТЕМА UNIX-ПОДІБНИХ ОС

Операційна система виконує дві основні задачі: маніпулювання даними та їх зберігання. Більшість програм в основному маніпулює даними, але, в кінцевому рахунку, вони де-небудь зберігаються. В системі UNIX таким місцем зберігання є файлова система. Більш того, в UNIX всі пристрої, з якими працює операційна система, також представлені у вигляді спеціальних файлів у файловій системі.

Файлова система (ФС) – це частина операційної системи, що включає:

- сукупність всіх файлів на диску;

- набори структур даних, використовуваних для управління файлами, такі, наприклад, як каталоги файлів, дескриптори файлів, таблиці розподілу вільного і зайнятого простору на диску;

- комплекс системних програмних засобів, що реалізують різні операції над файлами, такі як створення, знищення, читання, запис, іменування і пошук файлів.

Файлова система контролює права доступу до файлів, виконує операції створення і видалення файлів, а також виконує запис/читання даних файлу. Оскільки більшість прикладних функцій виконується через інтерфейс файлової системи, отже, права доступу до файлів визначають привілеї користувача в системі.

Файлова система забезпечує перенаправлення запитів, адресованих периферійним пристроям, відповідним модулям підсистеми введення-виведення.

Файлова система UNIX забезпечує уніфікований інтерфейс доступу до даних, що розташовані на різних носіях, і до периферійних пристроїв.

Розрізняють фізичну і логічну організацію файлової системи.

5.1 Поняття логічної файлової системи

Одним з основних завдань операційної системи є надання зручностей користувачеві при роботі з даними, що зберігаються на дисках. Для цього ОС підміняє фізичну структуру даних, що зберігаються деякої зручною для користувача логічною моделлю.

Логічна файлову систему в ОС UNIX – це ієрархічно організована структура всіх каталогів і файлів в системі з однієї вихідної вершиною, званої кореневим каталогом. Кореневої каталог розділяється на кілька підкаталогів. Кожен підкаталог в свою чергу може поділятися на кілька додаткових каталогів.

Ця структура перейшла з UNIX в більшість сучасних ОС.

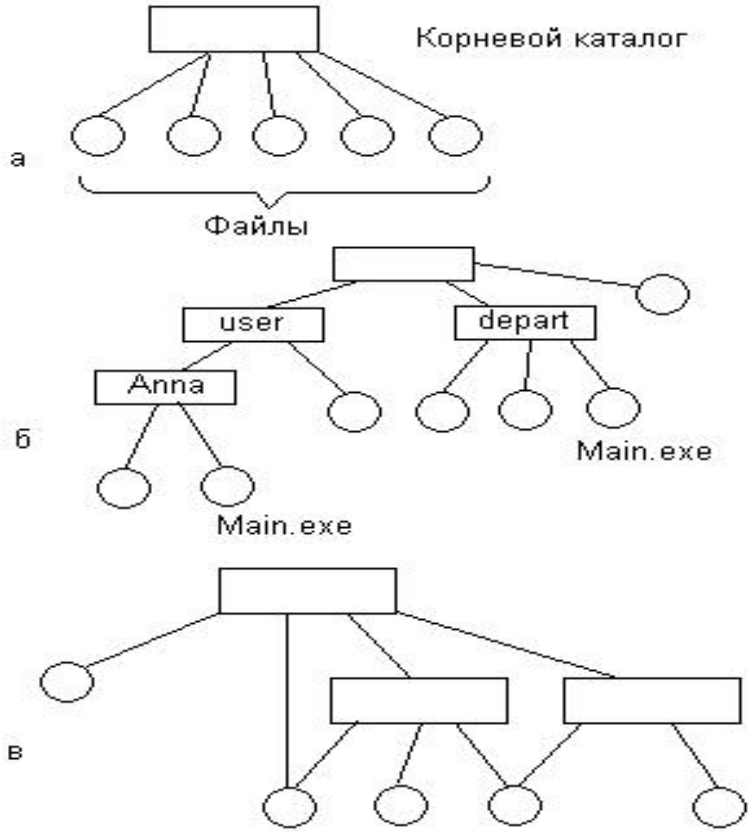

Рисунок 5.1 – Ієрархія каталогів

Граф, що описує ієрархію каталогів, може бути деревом або мережею. Каталоги утворюють дерево, якщо файлу дозволено входити тільки в один каталог (рис. 5.1, б), і мережа – якщо файл може входити відразу в декілька каталогів (рис. 5.1, в). Наприклад, в MS-DOS і Windows каталоги утворюють деревовидну структуру, а в UNIX – мережеву. Каталог самого верхнього рівня називається кореневим каталогом, або коренем.

Проте зазвичай, говорячи про файлову систему UNIX-подібних ОС, застосовують термін «файлове дерево».

5.1.1 Логічна файлова система – основні каталоги та їх призначення

Використання загальноприйнятих імен основних файлів і структури каталогів суттєво полегшує роботу в операційній системі, її адміністрування і підвищує переносимість. Типова структура і призначення каталогів файлової системи UNIX представлена в таблиці 5.1.

Таблиця 5.1 – Основні каталоги логічної файлової системи UNIX-подібних ОС

Каталог	Призначення і зміст
/	Кореневий каталог. Є основою будь-якої файлової системи UNIX. Всі інші каталоги і файли розташовуються в рамках структури, породженої кореневим каталогом (в ньому і в його підкаталогах), незалежно від їх фізичного місцезнаходження. Для кореневого каталогу обов'язково повинна створюватися окрема фізична файлова система, а сам він є точкою її монтування, про що свідчить наявність підкаталогу lost + found.
/bin (/sbin)	Містить команди, необхідні для забезпечення мінімального рівня працездатності системи. Може бути символічною посиланням, що вказує на /usr /bin.
/dev	Каталог для спеціальних файлів пристроїв. Може мати підкаталоги для різних класів і типів пристроїв.
/etc	Каталог для конфігураційних файлів. Може мати підкаталоги для різних компонентів і служб. Файли в UNIX – звичайні текстові. Використовується при запуску і конфігурації системи.
/home	Каталог для розміщення початкових каталогів користувачів. Часто є точкою монтування окремої фізичної файлової системи.
/lib	Каталог для бібліотек. Часто є символічною зв'язком, що вказує на /usr/lib.
/lost+found	Залишки пошкоджених файлів при збої.
/mnt	Каталог для розміщення точок монтування файлових систем на знімних носіях або додаткових дисках. Може містити підкаталоги для окремих типів носіїв, наприклад, cdrom або floppy. Може бути порожнім.
/opt	Каталог для додаткового комерційного програмного забезпечення. Може бути порожнім або відсутнім (в BSD-системах).
/proc	Каталог псевдо-файлової системи, що представляє у вигляді каталогів і файлів інформацію про ядро, пам'ять і процеси, які працюють в системі.
/sbin	Каталог для системних виконуваних програм, необхідних для вирішення завдань системного адміністрування.
/tmp	Каталог для тимчасових файлів. Доступний для запису і читання всім користувачам. Зазвичай створюється у вигляді окремої фізичної файлової системи, в тому числі, в віртуальної пам'яті.
/usr	В цьому каталозі знаходяться виконувані програми, бібліотеки, заголовні файли, довідкові керівництва (/usr/share/man), вихідні тексти ядра і утиліт системи (Linux), файли і черги друку, що ростуть (/usr/spool в BSD-системах) і т . Д. Часто каталог є точкою монтування окремої фізичної файлової системи. Нижче представлені основні його підкаталоги.
/var	Буферні каталоги, файли реєстрації, облікова інформація, файли різних сервісних підсистем, наприклад, файлів журналів системи. Так, основний журнал системи, що ведеться демоном syslogd, розміщується у вигляді декількох файлів в підкаталозі /var/adm. Там же, в файлі /var/adm/messages, зберігаються повідомлення часу завантаження. Має сенс створювати окрему фізичну файлову систему для розміщення цього каталогу (і, можливо, його підкаталогу /var/run).
/usr/bin	Основні виконувані програми і утиліти.

Каталог	Призначення і зміст
/usr/include	Заголовки бібліотек. Може містити підкаталоги.
/usr/lib	Статично та динамічно компоновані бібліотеки. Може містити підкаталоги.
/usr/local	Каталог для додаткового вільно поширюваного програмного забезпечення (GNU). Містить структуру підкаталогів, аналогічну кореневого каталогу (bin, etc, include, lib і т.д.).

Наявність, призначення та використання інших каталогів верхнього рівня і підкаталогів залежить від версії ОС UNIX, встановленого системного і прикладного програмного забезпечення та конфігурації системи, створеної адміністратором.

5.2 Фізична організація файлової системи

5.2.1 Основні компоненти фізичної файлової системи UNIX - подібних ОС

Кожен жорсткий диск складається з однієї або декількох логічних частин (груп циліндрів), званих розділами (partitions). Розташування і розмір розділу визначається при форматуванні диска. В UNIX-подібних ОС розділи виступають в якості незалежних пристроїв, доступ до яких здійснюється як до різних носіїв даних.

У розділі може розташовуватися тільки одна фізична файлова система. Систему можна конфігурувати так, що диск складається тільки з одного розділу.

Перші UNIX-системи підтримували одну файлову систему, структуру, яка була занесена прямо в ядро.

На сьогоднішній день підтримуються різні типи фізичних файлових систем: є безліч типів фізичних файлових систем UNIX (ufs, s5fs, ext2, ext3, vxfs, jfs, ffs і т.д.), більш того, підтримуються системи з іншою структурою, наприклад, FAT16 і NTFS.

Ми вже говорили, що є дві гілки розвитку UNIX: та, яка базується на BSD і та, яка базується на System V. (тобто від AT&T і від Berkeley). Кожній гілки відповідає своя файлова система. UNIX фірми AT&T має файлову систему S5. Файлова система з Берклі була названа Fast File System (FFS), але в даний час більше відома під назвою Unix File System (UFS)

Незважаючи на існуючі відмінності обидва типи файлових систем складаються з однакових компонентів:

ЗАВАНТАЖУВАЛЬНИЙ БЛОК
СУПЕРБЛОК
МАСИВ ІНДЕКСНИХ ДЕСКРИПТОРІВ
ОБЛАСТЬ ДАНИХ

- Завантажувальний блок – зарезервована область, яка знаходиться в самому початку файлової системи. На первинному диску вона містить фрагмент коду, який завантажує систему при запуску. На вторинних дисках він, швидше за все, буде порожнім. Завантажувальний блок є в будь-якій файловій системі незалежно від того, використовується він чи ні.

- Суперблок (superblock). Містить загальну інформацію про файлову систему.

- Масив індексних дескрипторів (ilist). Містить метадані всіх файлів файлової системи. Індексний дескриптор (inode) містить інформацію про статус файлу і вказує на розташування даних цього файлу. Ядро звертається до індексного дескриптора за індексом в масиві. Один дескриптор є кореневим для фізичної файлової системи, через нього забезпечується доступ до структури каталогів і файлів після монтування файлової системи. Розмір масиву індексних дескрипторів є фіксованим і задається при створенні фізичної файлової системи.

- Блоки зберігання даних. Дані звичайних файлів і каталогів зберігаються в блоках. Обробка файлу здійснюється через індексний дескриптор, що містить посилання на блоки даних.

Суперблок

Суперблок містить інформацію, необхідну для монтування і управління файловою системою в цілому. У кожній файловій системи існує тільки один суперблок, який розташовується на початку розділу. Суперблок зчитується в пам'ять ядра при монтуванні файлової системи і перебуває там до її відключення – демонтування.

Суперблок містить:

- тип файлової системи;

- розмір файлової системи в логічних блоках, включаючи сам суперблок, масив індексних дескрипторів і блоки зберігання даних;

- розмір масиву індексних дескрипторів;

- кількість вільних блоків;

- кількість вільних індексних дескрипторів;

- прапори;

- розмір логічного блоку файлової системи (512, 1024, 2048, 4096, 8192).

- список номерів вільних індексних дескрипторів;

- список адрес вільних блоків.

Оскільки кількість вільних індексних дескрипторів і блоків зберігання даних може бути значним, зберігання двох останніх списків цілком в суперблоці непрактично. Для індексних дескрипторів зберігається тільки частина списку. Коли число вільних

дескрипторів наближається до 0, ядро переглядає список і знову формує список вільних дескрипторів.

Такий підхід неприйнятний щодо вільних блоків зберігання даних, оскільки у вмісті блоку не можна визначити, вільний він чи ні. Тому необхідно зберігати список адрес вільних блоків цілком. Список адрес вільних блоків може займати кілька блоків зберігання даних, але суперблок містить тільки один блок цього списку. Перший елемент цього блоку вказує на блок, який зберігає продовження списку.

Виділення вільних блоків для розміщення файлу проводиться з кінця списку суперблоку. Коли в списку залишається єдиний елемент, ядро інтерпретує його як покажчик на блок, що містить продовження списку. У цьому випадку вміст цього блоку зчитується в суперблок, і блок стає вільним. Такий підхід дозволяє використовувати дисковий простір під списки, пропорційне вільному місцю в файлової системі. Коли вільного місця практично не залишається, список адрес вільних блоків цілком поміщається в суперблоці.

Індексні дескриптори

Індексний дескриптор, або inode, містить інформацію про файл, необхідну для обробки даних, тобто метадані файлу. Кожен файл асоційований з одним індексним дескриптором, хоча може мати кілька імен (жорстких зв'язків) в файлової системі, кожне з яких вказує на один і той же індексний дескриптор.

Індексний дескриптор не містить:

- імені файлу, яке міститься в блоках зберігання даних каталогу;
- вмісту файлу, яке розміщене в блоках зберігання даних.

Індексний дескриптор містить:

- номер;
- тип файлу;
- права доступу до файлу;
- кількість зв'язків (посилань на файл в каталогах) файлу;
- ідентифікатор користувача і групи-власника;
- розмір файлу в байтах;
- час останнього доступу до файлу;
- час останньої зміни файлу;
- час останньої зміни індексного дескриптора файлу;
- покажчики на блоки даних файлу (зазвичай 10);
- покажчики на непрямі блоки (зазвичай 3).

Розмір індексного дескриптора зазвичай становить 128 байтів.

Індексний дескриптор містить інформацію про розташування даних файлу. Оскільки дискові блоки зберігання даних, в загальному випадку, розташовуються не послідовно, індексний дескриптор повинен зберігати фізичні адреси всіх блоків, що належать даному файлу.

Підтримуючи множинні рівні непрямості, індексні дескриптори дозволяють відслідковувати величезні файли, не витрачаючи дисковий простір для невеликих файлів.

Для порівняння наведемо представлення каталогів.

У файлової системи MS-DOS каталог містить значення атрибутів файлів. На рисунку 5.2 представлена структура записи в каталозі, що містить просте символьне ім'я і атрибути файлу. Тут буквами позначені ознаки файлу: R – тільки для читання, A – архівний, H – прихований, S – системний.

В ОС UNIX атрибути файлів розміщуються в спеціальних таблицях, а каталоги містять тільки посилання на ці таблиці. Такий підхід реалізований, наприклад, в файлової системі ufs ОС UNIX. У цій файлової системи структура каталогу дуже проста. Запис про кожен файл містить коротке символьне ім'я файлу і покажчик на індексний дескриптор файлу, так називаєма таблиця, в якій зосереджені значення атрибутів файлу.

Рисунок 5.2 – Структура каталогів:

а – структура запису каталогу MS–DOS (32 байта),

б – структура запису каталогу ОС UNIX

5.3 Орієнтація та навігація у файловій системі

Ієрархічна структура файлової системи UNIX спрощує орієнтацію в ній. Кожен каталог, починаючи з кореневого (/), в свою чергу, містить файли та інші каталоги (підкаталоги). Кожен каталог містить також посилання на батьківський каталог (для

кореневого каталогу батьківським є він сам), представлену каталогом з ім'ям дві точки (..) і посилання на самого себе, представлену каталогом з ім'ям точка (.).

5.4 Управління файловою системою

Основними завданнями адміністрування файлових систем є створення, монтування та демонтування фізичних файлових систем, а також перевірка їх цілісності.

5.4.1 Створення фізичної файлової системи

Команда *mkfs* створює файлову систему шляхом запису на вказаний пристрій (необхідно вказати спеціальний символьний пристрій). Файлова система створюється на основі зазначених у командному рядку типу файлової системи (ТипФС), специфічних_опцій і операндів. Команда має наступний синтаксис:

mkfs [-F ТипФС] [- V] [- m] [-o специфічні_опції] пристрій розмір [операнди]

Специфічні опції і операнди залежать від конкретного типу створюваної файлової системи. Їх можна подивитися на відповідній сторінці довідкового керівництва (наприклад, man mkfs_ufs для файлової системи ufs).

Основні опції і параметри команди mkfs представлені в табл. 5.2.

Таблиця 5.2 – Основні опції і параметри команди mkfs

Опція	Призначення
–F	Вказує тип файлової системи, яку необхідно створити. Тип файлової системи повинен бути або вказано тут, або знаходитися в файлі таблиці стандартних файлових систем (/etc/vfstab в SVR4, /etc/fstab в інших версіях UNIX) шляхом зіставлення пристрої із записом в таблиці.
–V	Видає результуючий командний рядок, але не виконує команду. Командний рядок генерується з використанням опцій і аргументів, зазначених користувачем, шляхом додавання до них інформації, взятої з таблиці стандартних файлових систем. Ця опція використовується для перевірки правильності командного рядка.
–m	Повертає командний рядок, використаний для створення файлової системи. Файлова система повинна вже існувати. Ця опція забезпечує засоби отримання команди, використаної при створенні файлової системи. Для неї не застосовні специфічні_опції, розмір і операнди.
–o	Задає опції, специфічні для зазначеного типу фізичної файлової системи.
пристрій	Задає спеціальний символьний пристрій, на якому буде створена файлова система.
розмір	Задає кількість 512-байтових блоків в файловій системі. Максимальний розмір багатьох фізичних файлових систем в UNIX - 4194304 блоку розміром 512 байт (або 2 Гб).

5.4.2 Монтування і демонтування фізичних файлових систем

У загальному випадку система може мати декілька дискових пристроїв. Більш того, навіть один фізичний пристрій за допомогою засобів операційної системи може бути представлений у вигляді декількох логічних пристроїв, зокрема шляхом розбиття дискового простору на розділи. Виникає питання, яким чином організувати зберігання файлів в системі, що має кілька пристроїв зовнішньої пам'яті?

Перше рішення полягає в тому, що на кожному з пристроїв розміщується автономна файлова система, тобто файли, що знаходяться на цьому пристрої, описуються деревом каталогів, ніяк не пов'язаним з деревами каталогів на інших пристроях. У такому випадку для однозначної ідентифікації файлу користувач поряд з складовим символьним ім'ям файлу повинен вказувати ідентифікатор логічного пристрою. Прикладом такого автономного існування файлових систем є операційна система MS-DOS, в якій повне ім'я файлу включає буквений ідентифікатор логічного диска.

Іншим варіантом є така організація зберігання файлів, при якій користувачеві надається можливість об'єднувати файлові системи, що знаходяться на різних пристроях, в єдину файлову систему, яка описується єдиним деревом каталогів. Така операція називається монтуванням.

Розглянемо, як здійснюється ця операція.

Серед усіх наявних в системі логічних дискових пристроїв операційна система виділяє один пристрій, званий системним. Нехай є дві файлові системи, розташовані на різних логічних дисках, причому один, з дисків є системним.

Файлова система, розташована на системному диску, призначається кореневої. Для зв'язку ієрархій файлів в кореневій файловій системі вибирається деякий існуючий каталог. Після виконання монтування обраний каталог стає кореневим каталогом другої файлової системи. Через цей каталог вмонтовується файлова система під'єднується як піддерево до загального дерева.

Після монтування загальної файлової системи для користувача немає логічної різниці між кореневою і змонтованою файловими системами, зокрема іменування файлів проводиться так само, як якби вона з самого початку була єдиною.

Фізичні файлові системи, крім кореневої (/), вважаються знімними (removable) в тому сенсі, що вони можуть бути як доступні для користувачів, так і не доступні. Команда mount повідомляє систему, що блоковий пристрій або віддалений ресурс доступні для користувачів в точці_монтування, яка вже має існувати; точка монтування стає ім'ям

кореня знову змонтованого пристрої або ресурсу. Кажуть, що ця команда монтує або підключає фізичну файлову систему або ресурс до загальної логічної файлової системи.

Команда *mount* має наступний синтаксис:

mount [-F ТипФС] [-V] [-о специфічні_опції] {пристрій точка_монтування}

Команда mount, при виклику з аргументами, перевіряє всі аргументи, за винятком пристрою, і викликає специфічний модуль монтування для зазначеного типу файлової системи. При виклику без аргументів mount видає список всіх змонтованих файлових систем. Коли Ви визиваєте команду з неповним списком аргументів (наприклад, тільки з зазначенням пристрою або точки_монтування, або коли зазначені обидва ці аргументи, але не заданий тип файлової системи), mount буде переглядати таблицю стандартних файлових систем в пошуках відсутніх аргументів. Потім вона викликає специфічний модуль монтування для відповідного типу файлової системи.

Специфічні опції монтування залежать від типу фізичної файлової системи. Всі фізичні файлові системи можна монтувати тільки для читання (-о ro).

Зворотна процедура по відношенню до монтування називається демонтування і виконується командою *umount* з наступним синтаксисом:

umount [-V] [-о специфічні_опції] {пристрій | точка_монтування}

Для більшості типів файлових систем немає специфічного модуля демонтажу. Якщо такий модуль існує, він виконується; інакше файлова система демонтується стандартним модулем.

Команди mount і umount сприймають такі основні опції:

-v Видає результати в "новому" стилі. При цьому додатково відображається тип файлової системи і прапори. Поля точка_монтування і пристрій переставлені.

-p Видає список змонтованих файлових систем в форматі таблиці змонтованих файлових систем.

-F Задає тип файлової системи для монтування. Тип файлової системи повинен бути або заданий, або визначається по таблиці стандартних файлових систем в ході монтування.

-V Видає результуючу командний рядок, але не виконує команду. Командний рядок генерується з використанням опцій і аргументів, зазначених користувачем, шляхом додавання до них, при необхідності, інформації, взятої з таблиці стандартних файлових систем.

-о Задає специфічні опції для зазначеного типу фізичної файлової системи.

Будь-який користувач може викликати команду mount для отримання списку змонтованих файлових систем і ресурсів. **Тільки** користувач root може монтувати або демонтувати файлові системи.

Таблиця змонтованих файлових систем

Команда *mount* за замовчуванням додає запис в таблицю змонтованих файлових систем (файл /etc/mnttab в SVR4); *umount* видаляє запис з цієї таблиці. Поля в таблиці змонтованих пристроїв розділені пробілами і представляють блочне спеціальний пристрій, під'єднання, тип змонтованої файлової системи, опції монтування і час, коли файлова система була змонтована.

Таблиця стандартних файлових систем

Таблиця стандартних файлових систем (у файлі /etc/vfstab або /etc/fstab, в залежності від різновиду UNIX) описує стандартні параметри для фізичних файлових систем. Поля в таблиці (їх 7) розділені пробілами і табуляцією, і представляють, відповідно:

- Спеціальне блоковий пристрій або ім'я ресурсу, що монтується;
- Неформатований (спеціальне символьне) пристрій для перевірки утилітою fsck;
- Стандартний каталог монтування;
- Тип файлової системи;
- Число, яке використовується fsck для прийняття рішення про автоматичну перевірку файлової системи і про порядок цієї перевірки по відношенню до інших файлових систем;
- Ознака автоматичного монтування файлової системи;
- Опції монтування.

Якщо в полі немає значення, використовується дефіс.

5.4.3 Перевірка та відновлення цілісності файлових систем

При відкритті файлу, ядро поміщає копію дискового індексного дескриптора в відповідну таблицю в пам'яті, яка містить додаткові атрибути. Надалі зміна індексного дескриптора відбувається в пам'яті, і змінена структура файлової системи скидається на диск тільки при виконанні спеціальної команди, sync. Ця команда виконується при штатній зупинці системи або явно адміністратором.

У разі виникнення нештатного припинення роботи системи, структура суперблоку і масиву індексних дескрипторів на диску не відповідає структурі блоків даних і може бути неузгодженою.

Відсутність синхронізації між образом файлової системи в пам'яті і її даними на диску (в разі аварійної зупинки системи) може привести до появи помилок в файлової системі

Рішення проблеми узгодженості і цілісності даних в файлових системах UNIX-подібних ОС можливо тільки при використанні журналізації. Хоча частина цих проблем може бути усунена спеціальним інструментом – fsck.

Програма fsck шукає і, автоматично або в інтерактивному режимі, виправляє протиріччя в файлових системах. При використанні fsck файлова система повинна бути неактивною (демонтувати або змонтована тільки для читання). Якщо це неможливо, необхідно забезпечити, щоб машина перебувала в стані спокою (без працюючих користувачів) і щоб відразу після завершення команди вона була перезавантажена, якщо виправляється критична файлова система, наприклад, коренева.

Команда fsck повинна виглядати так:

fsck [-F ТипФС] [-V] [-m] [пристрій ...]

fsck [-F ТипФС] [-V] [-о специфічні_опції] [пристрій ...]

Для роботи команді fsck необхідно вказувати спеціальний символьний пристрій.

Коренева файлова система зазвичай перевіряється при запуску автоматично. Система при запуску може автоматично перевіряти і інші фізичні файлові системи, для яких у таблиці стандартних файлових систем вказана необхідність такої перевірки.

5.4.4 Отримання інформації про файлові системи

Для отримання інформації про змонтовані фізичні файлові системи використовується команда df з наступним синтаксисом:

df [-F ТипФС] [-V] [-о специфічні_опції] [пристрій | каталог | файл | ресурс ...]

Опції і параметри визначають формат видаваної інформації і файлові системи, про які інформує команда. Найчастіше, команда df викликається без опцій або з опцією -k. Опція -k видає інформацію про обсяги в кілобайтах. Для кожної фізичної файлової системи видається окремий рядок, що включає (при використанні опції -k) спеціальний файл або ім'я змонтованого ресурсу, загальний обсяг, використаний обсяг, доступний обсяг для використання звичайними користувачами, відсоток вільного місця в файлової системі і точку монтування.

5.5 Файлова система /proc

Створена вона була ще в 8-й реалізації UNIX, а пізніше дороблена в 4.4BSD і SystemV. Основна концепція цієї файлової системи полягає в тому, що для кожного

процесу системи створюється підкаталог в каталозі /proc. Ім'я каталогу формується з ідентифікатора процесу в десятковому форматі.

У цьому каталозі розташовуються файли, які несуть інформацію про процес – його команду, рядки оточення і маски сигналів. Насправді цих файлів на диску немає. Коли вони зчитуються, система отримує інформацію від фактичного процесу і повертає її в стандартному форматі.

5.6 NFS

Мережі грають головну роль в UNIX з самого початку, що послужило приводом створення NFS (Network File System). Вона вперше з'явилася в складі SunOs в 1984р. Призначалася вона для роботи в гетерогенних мережах, незалежно від апаратної платформи. Тобто сервери, запущені під різними UNIX-подібними ОС, а також під ОС, відмінними від UNIX, могли отримати доступ до останньої, поки вони використовують стек TCP/IP і NFS.

NFS використовує протокол RPC (віддалений виклик процедур). Використовуючи RPC додатки на одній машині можуть викликати процес на інший, а результат відобразити на локальному дисплеї.

Кожен сервер NFS експортує каталоги, надаючи доступ до них віддаленим клієнтам. Список каталогів, що експортуються зберігається в файлі /etc/exports. При завантаженні сервера вони експортуються автоматично. Коли клієнт монтує віддалений каталог (це робиться в процесі завантаження системи), цей каталог стає частиною загального дерева каталогів клієнта.

Вибір вузла, в якому монтується віддалений каталог, цілком залежить від клієнта. Сервер не знає, де клієнт зберігає монтується каталог.

Хоча при роботі з NFS існує ряд проблем, вона розвивається з розвитком мереж. По суті, всі файлові системи мережі можна з'єднати в єдину файлову систему, при цьому локальний користувач цього не помітить.

5.7 VFS

Сучасні версії UNIX забезпечують роботу з декількома типами файлових систем різної архітектури, віддалені і навіть відмінні від файлових систем UNIX (наприклад, DOS).

Багато UNIX-подібних систем підтримують стандартний метод обміну інформацією між ядром і операціями з індексними дескрипторами, незалежно від того, яка файлова система запущена. Цей метод називається VFS.

Текст файлової системи тим самим розбивається на дві частини: верхня частина, пов'язана з розподілом таблиць ядра і структур даних, і нижня частина, створена для установки функцій, що залежать від файлової системи і викликаються через структури даних VFS.

Завдання рівня VFS полягає в управлінні таблицею, що містить по одному запису для кожного відкритого файлу, аналогічної таблиці inode для відкритих файлів в UNIX. У звичайній UNIX inode однозначно вказується парою (пристрій, номер inode). Замість цього рівень VFS містить для кожного відкритого файлу записи, звані vnode. Структура цих записів однакова для всіх файлів, незалежно від типу реальної файлової системи.

5.8 Файлові системи, що журналізуються

Для мінімізації проблем пов'язаних з цілісністю і мінімізацією часу перезапуску системи, файлова система, що журналізується, зберігає список змін, які вона буде проводити з файлової системою перед фактичним записом змін.

Ці записи зберігаються в окремій частині файлової системи, яку називають <журналом "або" логом> і є наборами пов'язаних змін файлової системи, що дуже схоже на те, як зміни, що додаються в базу даних організовані в транзакції.

Журнал виділяється з вільних блоків файлової системи і, звичайно, має розмір близько 1 Мбайта на кожен 1 Гбайт файлової системи. Журнал скидається в міру заповнення, після синхронізації структури файлової системи з диском.

Як тільки ці записи журналу (логу) безпечно записані, журналізуєма файлова система вносить ці зміни в файлову систему і потім видаляє ці записи з <логу> (журналу реєстрацій).

Записи в лог-файл ведуться до проведення змін файлової системи і зберігаються до тих пір, поки вони не будуть цілком і безпечно застосовані до файлової системи.

Різні версії ОС UNIX підтримують різні реалізації файлових систем, що журналізуються. Це, наприклад, файлова система ufs (Solaris), vxfs (Solaris, UnixWare), RaisorFS і ext3 (Linux), XFS (Linux і ін., розробка фірми Silicon Graphics), RaiserFS, журналізуєма файлова система розроблена спеціально для Linux, jfs (AIX і Linux) та інші. Деякі файлові системи дозволяють вмикати та вимикати журналізацію (ufs, ext2/ext3).

Природно, журналізація кілька уповільнює роботу файлової системи, але, в більшості випадків, гарантує цілісність даних.

6 ПРОЦЕСИ

Процес – програма в момент виконання і єдина активна сутність в системі UNIX.

Процес – програма, що ідентифікується унікальним чином, яка потребує отримання доступу до ресурсів комп'ютера.

Кожен процес запускає одну програму і спочатку отримує один потік управління. Більшість версій UNIX дозволяють процесу після того, як він запущений, створювати додаткові потоки. UNIX – багатозадачна система, так що кілька незалежних процесів можуть працювати одночасно.

6.1 Типи процесів:

В ОС UNIX виділяється три типи процесів:

- системні,
- демони
- прикладні процеси.

Системні процеси є частиною ядра і завжди розташовані в оперативній пам'яті. Системні процеси не мають відповідних їм програм у вигляді виконуваних файлів і запускаються особливим чином при ініціалізації ядра системи. Інструкції і дані, що виконуються, цих процесів знаходяться в ядрі системи, таким чином, вони можуть викликати функції і звертатися до даних, недоступним для інших процесів.

До системних процесів можна віднести і процес початкової ініціалізації, init, який є прабатьком всіх інших процесів. Хоча init не є частиною ядра, і його запуск відбувається з виконуваного файлу, його робота життєво важлива для функціонування всієї системи в цілому.

Демони – це не інтерактивні процеси, які запускаються звичайним чином – шляхом завантаження в пам'ять відповідних їм програм, та виконується у фоновому режимі. Зазвичай демони запускаються при ініціалізації системи, але після ініціалізації ядра і забезпечують роботу різних підсистем UNIX: системи термінального доступу, системи друку, мережевих служб і т.д. Демони не пов'язані ні з одним користувачем. Велику частину часу демони очікують, поки той або інший процес запросить певну послугу.

Типовим демоном є cron. Цей демон планує активність системи на хвилини, години, дні і місяці вперед. Він прокидається кожну хвилину, перевіряючи, чи не потрібно чого зробити. Якщо у нього є робота, він її виконує і відправляється спати далі.

До **прикладних** процесів належать всі інші процеси, що виконуються в системі. Як правило, це процеси, породжені в рамках користувальницького сеансу роботи. Найважливішим для користувача процесом є початковий командний інтерпретатор, який забезпечує виконання команд користувача в системі UNIX.

Призначені для користувача процеси можуть виконуватися як в інтерактивному (пріоритетному), так і в фоновому режимах. Інтерактивні процеси монопольно володіють терміналом, і поки такий процес не завершить своє виконання, користувач не має доступу до командного рядка.

6.2 Атрибути процесу

Процес в UNIX має ряд атрибутів, що дозволяють операційній системі управляти його роботою. Основні атрибути:

1) Ідентифікатор процесу (PID)

2) Ідентифікатор батьківського процесу (PPID)

3) Поточний пріоритет PRI

4) Поправка пріоритету (NICE) – відносний пріоритет

5) Термінальна лінія (TTY)

6) Реальний (UID) і ефективний (EUID) ідентифікатори користувача

7) Реальний (GID) і ефективний (EGID) ідентифікатори групи

1) Кожен процес має унікальний ідентифікатор PID, що дозволяє ядру системи розрізняти процеси. Коли створюється новий процес, ядро привласнює йому наступний вільний ідентифікатор. Присвоєння ідентифікатора зазвичай відбувається по зростаючий, тобто ідентифікатор нового процесу більше, ніж ідентифікатор процесу, створеного перед ним. Якщо ідентифікатор досягає максимального значення (зазвичай – 65737), наступний процес отримає мінімальний вільний PID і цикл повторюється. Коли процес завершує роботу, ядро звільняє ідентифікатор, що використовувався ним.

2) PPID – ідентифікатор процесу, який породив даний процес. Всі процеси в системі, крім системних процесів і процесу init, що є прабатьком інших процесів, породжені одним з існуючих або існували раніше процесів.

3) Розподіл процесорних ресурсів визначається пріоритетом виконання PRI.

4) Відносний пріоритет процесу, що враховується планувальником при визначенні черговості запуску. Відносний пріоритет не змінюється системою на всьому протязі життя процесу (хоча може бути змінений користувачем або адміністратором) на відміну від пріоритету виконання, динамічно змінюваного планувальником.

5) Термінал або псевдотермінал, пов'язаний з процесом. З цим терміналом за замовчуванням пов'язані стандартні потоки: вхідний, вихідний і потік повідомлень про помилки. Процеси-демони не пов'язані з терміналом.

6) Реальним ідентифікатором користувача даного процесу є ідентифікатор користувача, що запустив процес. Ефективний ідентифікатор служить для визначення прав доступу процесу до системних ресурсів (в першу чергу до ресурсів файлової системи). Зазвичай реальний і ефективний ідентифікатори збігаються, тобто процес має в системі ті ж права, що і користувач, що запустив його. Однак існує можливість задати процесу ширші права, ніж права користувача, шляхом установки біта SUID, коли ефективному ідентифікатором присвоюється значення ідентифікатора власника виконуваного файлу (наприклад, користувача root).

7) Реальний ідентифікатор групи дорівнює ідентифікатору основної або поточної групи користувача, що запустив процес. Ефективний ідентифікатор служить для визначення прав доступу до системних ресурсів від імені групи. Зазвичай ефективний ідентифікатор групи збігається з реальним. Але якщо для виконуваного файлу встановлений біт SGID, такий файл виконується з ефективним ідентифікатором групи-власника.

6.3 Структура процесу

Виконання процесу може відбуватися в двох режимах – ядра і завдання. У режимі завдання виконується основна частина процесу, виконуючи інструкції прикладної програми, допустимі на непривілейованому рівні захисту процесора. При цьому йому недоступні системні структури ядра. Коли процесу потрібно отримання будь-яких послуг ядра, він робить системний виклик, який виконує інструкції ядра, що знаходяться на привілейованому рівні. Виконання процесу при цьому переходить в режим ядра.

Відповідно кожен процес в системі UNIX складається з двох частин і представлений двома структурами: користувальницькою (user) і системною (proc).

При завантаженні системи в оперативну пам'ять завантажується системна таблиця процесів, яка залишається там до закінчення роботи. Записами цієї таблиці є структури proc. Таблиця процесів є одночасно масивом і двозв’язним списком у вигляді дерева.

Фізичний опис є статичним масивом покажчиків, довжина якого є константою. Покажчики описані системної змінної curproc. Curproc вказує на структуру proc активного процесу і змінюється планувальником, коли ресурси процесора передаються іншому процесу. Структура proc містить покажчик на дані структури user.

Дані user розміщуються в певному місці віртуальної пам'яті ядра і адресуються системної змінної u (рисунок 6.1).

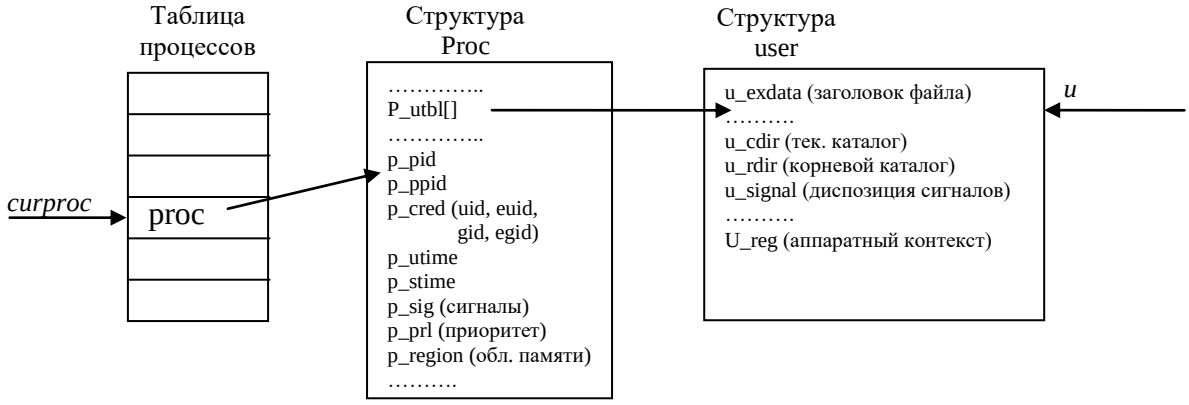

Рисунок 6.1 – Структура процесу

У таблиці процесів зберігається інформація, необхідна всім процесам, навіть тим, яких на даний момент в пам'яті немає.

Структура proc складається з наступних компонентів:

А) Різне: поточний стан, очікувані події, час до закінчення інтервалу будильника, PID і PPID процесу, ідентифікатори користувача і групи.

Б) Параметри планування: пріоритети, процесорний час, спожитий за останній розрахунковий період, кількість часу, проведена в очікуванні.

В) Образ пам'яті: покажчики на сегменти програми, даних і стека або на відповідні таблиці сторінок.

Г) Сигнали: маски, які вказують, які сигнали ігноруються, які перехоплюються, які тимчасово заблоковані або знаходяться в процесі доставки.

У структурі користувача міститься інформація, яка не потрібна, коли процесу фізично немає в пам'яті і він не виконується. Структура користувача вивантажується на диск, звільняючи місце в пам'яті, щоб не витрачати пам'ять на непотрібну в даний момент інформацію.

Структура користувача ділиться на наступні категорії:

А) Машинні регістри: заповнюються при перериванні з перемиканням в режим ядра.

Б) Стан системного виклику: інформація про поточний виклик, включаючи параметри і результати.

В) Таблиця дескрипторів файлів: по дескриптору файлу знаходиться структура даних (inode), який відповідає цьому файлу.

Г) Облікова інформація: покажчик на таблицю використання процесорного часу, максимальний розмір стека, кількість сторінок пам'яті і т.д.

Д) Стек ядра: фіксований стек для використання процесом в режимі ядра.

6.4 Логічна організація процесів

Процеси організовані в логічну структуру, що складається з самих процесів, груп процесів і сеансів. Це спрощує управління процесами.

Кожен процес належить певній групі, яка має унікальний ідентифікатор. Група може мати в своєму складі лідера групи – процес, чий ID дорівнює ID групи. Процес успадковує групу від батьківського процесу, але може покинути її і організувати власну.

Групи процесів об'єднуються в **сеанси**. Сеанси описують процеси, породжені користувачем за час роботи в системі. Кожен сеанс може мати один асоційований термінал, званий керуючим, а групи і процеси, створені в даному сеансі, успадковують цей термінал. Отже, процеси, що належать до різних сеансів, отримують доступ до різних терміналів. Наявність керуючого терміналу дозволяє ядру контролювати стандартне введення/виведення, а також посилати сигнали всім процесам, асоційованим з терміналом групи.

6.5 Функціонування процесу

З кожним процесом пов'язується власний *адресний простір*, що зазвичай називають *образом процесу*.

Адресний простір – список адрес пам'яті від деякого мінімуму до деякого максимуму, які процес може прочитати і в які він може писати.

Адресний простір (образ) містить саму програму (код), дані до неї і її стек. З усяким процесом пов'язується набір регістрів.

Виділяється чотири регістри, що мають спеціальне значення:

Регістр	Призначення
PC	Програмний лічильник – вказує на поточний рядок коду.
PS	Вказує стан процесора.
SP	Вказує на вершину стека.
FP	Вказує на поточний фрейм стека.

Крім того, існують динамічні області зберігання даних (купа), що виділяються процесу при необхідності.

Стек складається з фреймів, міститься в пам'яті і використовується для зберігання локальних змінних програми і передачі параметрів. Коли процес виконує звернення до

функції або підпрограми, в стек відправляється новий фрейм. Однією з частин кожного фрейма є покажчик на попередній фрейм, який дозволяє повернутися з виклику функції. Т.ч. треба знати місце розташування поточного фрейма і вершину стека.

Стеки, що використовуються процесами, що працюють в різних режимах, різні – певний сегментний стек використовується в призначеному для користувача режимі, а режим ядра використовує стек певної величини.

Операційна система маніпулює образом процесу. Сегмент коду містить реальні інструкції процесора, які включають як рядки, скомпільовані і написані користувачем, так і стандартний код, згенерований компілятором для системи. Цей системний код забезпечує взаємодію між програмою і операційною системою.

Оскільки призначені для користувача процеси і ядро не мають загального адресного простору пам'яті, необхідний механізм передачі даних між ними. Ідентифікатор процедури ядра передається або через апаратний регістр процесора, або через стек. Аргументи системного виклику передаються через користувачеву область процесу, що викликається.

Призначений для користувача процес не може звертатися до простору ядра, але ядро може звертатися до простору процесу.

Під час виконання або в очікуванні доступу до ресурсів процеси містяться в віртуальної пам'яті. При необхідності, сторінки пам'яті процесів відкачуються з фізичної пам'яті на диск, в область підкачування. При зверненні до сторінки у віртуальній пам'яті, якщо вона не знаходиться у фізичній пам'яті, відбувається її підкачка з диска.

6.6 Управління процесами

Процеси управляються системними викликами.

Системний виклик визначає функцію, виконувану ядром ОС від імені процесу, і є інтерфейсом найнижчого рівня взаємодії прикладних процесів з ядром. Системні виклики надають точки входу в ядро ОС, через які прикладні програми отримують можливість скористатися базовими послугами. Виконання системного виклику відбувається покроково і схоже на виконання процедури, тільки перший проникає в ядро, а другий ні.

Системні виклики всередині ядра існують як функції на мові С і їх імена починаються з префікса "sys_".

Типи системних викликів:

1. Головними системними викликами є виклики створення та закінчення процесу.

2. Для запитів про надання додаткової пам'яті або звільнення не використовуваної.

3. Для передачі інформації процесу, що працює.

4. Для передачі сигналів переривання.

Перша група системних викликів управляє життєвим циклом процесу.

Друга – розподілом пам'яті. Третя – взаємодією між процесами. Четверта – посилкою сигналів.

Розглянемо четверту групу системних викликів.

Для надсилання сигналу процесу використовується виклик kill (pid, sig). Sig – це номер сигналу.

Сигнал можна послати за допомогою команди: kill sig pid, де pid – ідентифікатор процесу.

Сигнал можна послати будь-якому родинному процесу, в тому числі і собі. Процес, що володіє привілеями суперкористувача, може послати сигнал будь-якому процесу.

Сигнали генеруються ядром і забезпечують виклик певної процедури при настанні деякої події.

Основні причини відправки сигналу:

Особливі ситуації. (ділення на нуль)

Термінальні переривання. (<Del>, <Ctrl + C>, <Ctrl + Z>)

Інші процеси. Як засіб взаємодії між процесами за допомогою виклику kill ()

Управління завданнями. Маніпулювання фоновими і поточними завданнями, повідомлення про завершення дочірнього процесу.

Квоти. Перевищення квоти ресурсів.

Повідомлення. Наприклад, про готовність пристрою.

Аларми. Пов'язано з роботою таймерів.

При отриманні сигналу процес має *три* варіанти дії:

1. Ігнорувати сигнал.

2. Вимагати дії за умовчанням. Зазвичай це завершення процесу.

3. Перехопити сигнал і обробити його.

Обробка сигналу відбувається, коли процес працює, тобто коли він обраний планувальником.

Сигнали, необхідні стандартом POSIX:

SIGABRT – перервати процес і створити дамп пам'яті

SIGALRM – вийшов час будильника

SIGFPE – помилка під час операції з плаваючою точкою

SIGHUP – переривання модемного зв'язку

SIGILL – користувач натиснув клавішу DEL

SIGQUIT – користувач натиснув клавішу припинення роботи з процесом зі створенням дампа пам'яті

SIGKILL – знищення процесу

SIGPIPE – процес пише в канал, з якого ніхто не читає

SIGSEGV – процес звернувся до невірної адреси в пам'яті

SIGTERM – звернення до процесу завершити роботу

SIGUSR1 – визначається додатком

SIGUSR2 – визначається додатком

Наприклад, сигнал SIGINT дозволяє процесу видалити створені ним тимчасові файли, тобто гідно підготуватися до смерті.

SIGKILL SIGTOP не можна ні перехопити, ні ігнорувати

Процесам, що чекають недоступні ресурси NFS посилають сигнали SIGINT або SIGQUIT.

Сигнал інтерактивного процесу можна послати з іншого терміналу. Наприклад, якщо програма зависла, можна перейти на інший термінал, обчислити pid програми за допомогою команди ps і послати їй сигнал kill -9. Відповідність між номерами та іменами сигналів можна дізнатися за допомогою команди kill -l

Інший спосіб надсилання сигналу – клавіші переривання:

<CTRL + C> відповідає SIGINT

<CTRL + Z> - SIGTOP

Таким чином можна зупинити програму, що працює в пріоритетному режимі.

6.7 Життєвий цикл процесу

Життєвий цикл процесу в ОС UNIX може бути розбитий на кілька станів. Перехід з одного стану в інший відбувається в залежності від настання певних подій в системі (рисунок 6.1).

Можливі наступні стани процесу:

1) Процес виконується в режимі користувача. При цьому процесором виконуються прикладні інструкції даного процесу.

2) Процес виконується в режимі ядра. При цьому процесом виконуються системні інструкції ядра від імені процесу. Існують лише три події, при яких виконання процесу переходить в режим ядра – апаратні переривання, особливі ситуації і системні виклики.

3) Процес не виконується, але готовий до запуску, як тільки планувальник вибере його (стан runnable). Процес перебувати в черзі на виконання і володіє всіма необхідними йому ресурсами, крім процесора.

4) Процес перебуває в стані сну (asleep), чекаючи недоступного в даний момент ресурсу, наприклад завершення операції введення-виведення.

5) Процес повертається з режиму ядра в режим завдання, але ядро перериває його і виробляє перемикання контексту для запуску більш пріоритетного процесу.

6) Процес щойно створений системним викликом fork і знаходиться в перехідному стані: він існує, але не готовий до запуску і не перебувати в стані сну.

7) Процес виконав системний виклик exit і перейшов в стан зомбі (zombie, defunct). Як такого процесу не існує, але залишаються записи, що містять код повернення і тимчасову статистику його виконання, доступну для батьківського процесу. Цей стан є кінцевим в життєвому циклі процесу.

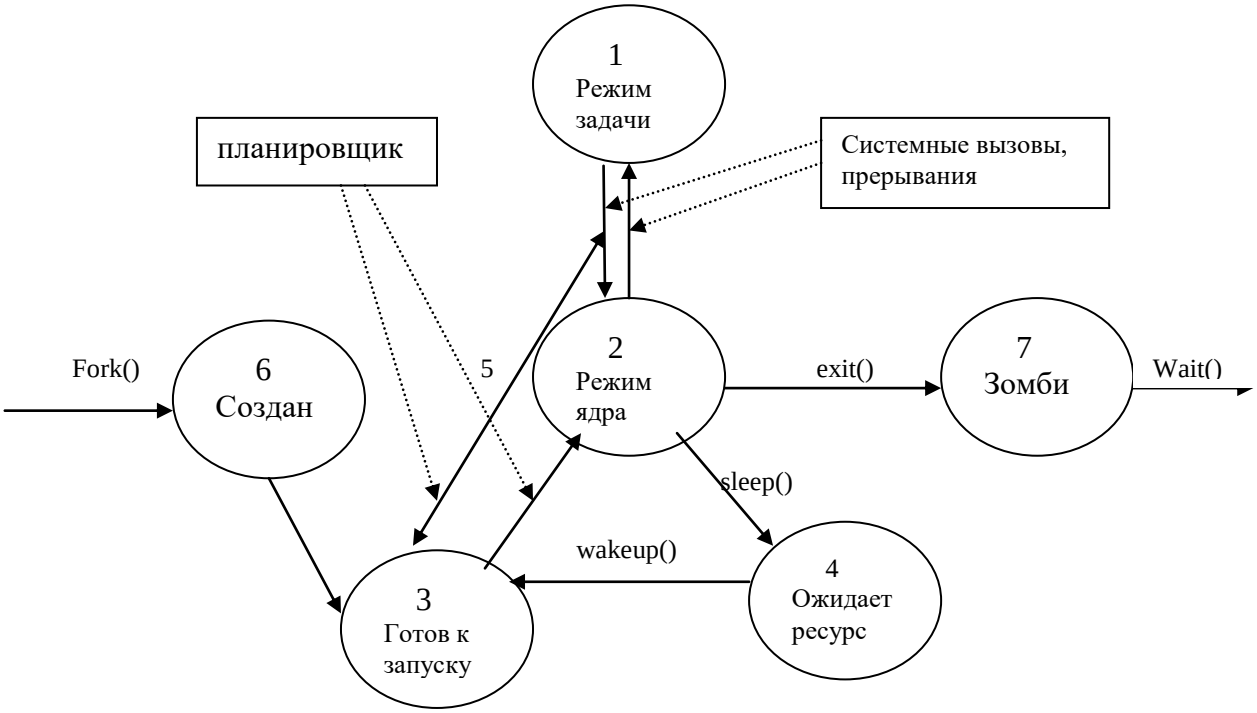

Рисунок 6.2 – Можливі стани процесу

Процес починає свій життєвий шлях зі стану 6, коли батьківський процес виконує системний виклик fork. Після того як створення процесу повністю завершено, процес завершує "дочірню частину" виклику fork і переходить в стан 3 готовності до запуску, чекаючи своєї черги на виконання. Коли планувальник вибирає процес для виконання, він переходить в стан 1 і виконується в режимі користувача.

Виконання в призначеному для користувача режимі завершується в результаті системного виклику або переривання, і процес переходить в режим ядра, в якому виконується код системного виклику або переривання. Після цього процес знову може повернутися в призначений для користувача режим. Однак під час виконання системного

виклику процесу в режимі ядра процесу може знадобитися недоступний в даний момент ресурс. Для очікування доступу до такого ресурсу, процес робить системний виклик sleep і переходить в стан 4 – сну. При цьому процес добровільно звільняє обчислювальні ресурси, які надаються наступному найбільш пріоритетному процесу.

Стан сну – це логічний стан процесу, при цьому він не переміщається фізично в пам'яті. Перехід в стан сну, в першу чергу, визначається занесенням в системну таблицю процесів відповідного прапора стану і події, що будить процес.

Коли ресурс стає доступним, ядро "пробуджує процес", використовуючи виклик wakeup, поміщає його в чергу на виконання, і процес переходить в стан 3 готовності до запуску.

Після того як планувальник вибрав процес на запуск, останній починає своє виконання в режимі ядра.

Нарешті, процес виконує системний виклик exit і закінчує своє виконання. Процес може бути також завершено внаслідок отримання сигналу. В обох випадках ядро звільняє ресурси, що належать процесу, за винятком коду повернення і статистики його виконання, і переводить процес в стан зомбі. У цьому стані процес знаходиться доти, поки батьківський процес не виконає системний виклик wait, після чого вся інформація про процес буде знищена, а батько отримає код повернення процесу, що завершився.

6.8 Контекст процесу

Кожен процес UNIX має контекст, під яким розуміється вся інформація, необхідна для опису процесу. Ця інформація зберігається, коли виконання процесу припиняється, і відновлюється, коли планувальник надає процесу обчислювальні ресурси.

Контекст процесу в ОС UNIX складається з декількох частин:

А) Адресний простір (образ) процесу в призначеному для користувача режимі. Сюди входять код, дані і стек процесу, а також інші області, наприклад, колективна пам'ять або код і дані динамічних бібліотек.

Б) Керуюча інформація. Ядро використовує дві основні структури для управління процесом – proc і user. Сюди ж входять дані, необхідні для відображення віртуального адресного простору процесу в фізичну пам'ять.

В) Середовище процесу. Змінні середовища процесу, значення яких задаються в командному інтерпретаторі або в самому процесі за допомогою системних викликів, а також успадковуються породженим процесом від батьківського і зазвичай зберігаються в нижній частині стека. Середу процесу можна отримувати або змінювати за допомогою функцій.

Г) Апаратний контекст. Сюди входять значення загальних і ряду системних регістрів процесора, зокрема, покажчик поточної інструкції та покажчик стека.

Перемикання між процесами, що регулює доступ до обчислювальних ресурсів виражається в перемиканні контексту. Контекст процесу, що виконувався запам'ятовується, а відновлюється контекст процесу, обраного планувальником. Перемикання контексту є досить ресурсномісткою операцією. Крім збереження стану регістрів процесу, ядро змушене виконати безліч інших дій.

6.8.1 Перемикання контексту

При наданні процесу обчислювальних ресурсів відбувається перемикання контексту, в результаті якого зберігається контекст поточного процесу, і управління передається новому.

Контекст перемикається в чотирьох випадках:

1. Поточний процес переходить в стан сну, чекаючи недоступного ресурсу.

2. Поточний процес завершує своє виконання.

3. Якщо після перерахунку пріоритетів в черзі на виконання є більш високопріоритетний процес.

4. Відбувається пробудження більш високопріоритетного процесу.

Перші два випадки відповідають добровільному переключенню контексту і ядро викликає процедуру перемикання контексту з функцій sleep або exit.

Третій і четвертий випадки перемикання контексту відбуваються не з волі процесу, який в цей час виконується в режимі ядра і тому не може бути негайно припинений, тому що перемикання контексту при виконанні в режимі ядра може призвести до порушення цілісності самої системи. У цій ситуації ядро встановлює спеціальний прапор, який вказує, що в черзі перебуває понад високопріоритетний процес, що вимагає надання обчислювальних ресурсів. Перед переходом процесу з режиму ядра в режим завдання ядро перевіряє цей прапор і, якщо він встановлений, викликає функцію перемикання контексту.

Перемикання контексту є досить ресурсномісткою операцією. Крім збереження стану регістрів процесу, ядро змушене виконати безліч інших дій.

6.8.2 Створення процесу

Новий процес створюється в UNIX тільки шляхом системного виклику fork. Процес, який зробив виклик fork, називається батьківським, а новостворений процес – породженим. Новий процес є точною копією батьківського. При породженні

(розгалуженні) процесу перевіряється, чи достатньо пам'яті і місця в таблиці процесів для даного процесу. Якщо так, то образ пов'язаних з поточною діяльністю копіюється в новий образ процесу, і в таблиці процесів виникає новий елемент. Новому процесу присвоюється новий унікальний ідентифікатор (PID). Коли зміна таблиці процесів ядра завершується, процес додається до списку процесів, готових до виконання.

Породжений процес успадковує від батьківського процесу наступні основні характеристики:

- Способи обробки сигналів (адреси функцій обробки сигналів).
- Реальні і ефективні ідентифікатори користувача і групи.
- Значення поправки пріоритету.
- Усі приєднані сегменти пам'яті.
- Ідентифікатор групи процесів.
- Термінальну лінію.
- Поточний каталог.
- Кореневий каталог.
- Маску створення файлів (umask).
- Обмеження ресурсів (ulimit).

Породжений процес відрізняється від батьківського процесу наступними основними характеристиками:

- має свій унікальний ідентифікатор.
- має інший ідентифікатор батьківського процесу, рівний ідентифікатору процесу, що його породив.
- має свої власні копії дескрипторів файлів (зокрема, стандартних потоків), відкритих батьківським процесом. Кожен дескриптор файлу породженого процесу має спочатку таке ж значення поточної позиції у файлі, що і відповідний батьківський. У породженого процесу обнуляються лічильники часу, витраченого системою для його обслуговування.

Батьківський і дочірній процеси мають свої власні образи. При цьому код однієї і тієї ж програми може використовуватися декількома процесами. В цьому випадку спільно використовується сегмент коду в пам'яті, але в іншому процеси ізольовані один від одного і мають різні сегменти даних і стека.

6.8.3 Завершення виконання процесу

Процес завершує роботу при виконанні системного виклику exit. Процес може сам завершити свою роботу, відповідно до алгоритму, або може бути припинений ядром.

При завершенні процесу послідовно виконуються наступні дії:

- вимкнено усі сигнали.

- В процесі, що викликав закриваються всі дескриптори відкритих файлів.

- Якщо батьківський процес знаходиться в стані виклику wait, то системний виклик wait завершується, видаючи батьківського процесу в якості результату ідентифікатор завершився процесу, і молодші 8 біт його коду завершення.

- Якщо батьківський процес не знаходиться в стані виклику wait, то процес, що завершується переходить в стан зомбі.

- У всіх існуючих нащадків завершених процесів, а також у зомбі-процесів ідентифікатор батьківського процесу встановлюється рівним 1. Таким чином, вони стають нащадками процесу ініціалізації (init).

- Якщо ідентифікатор процесу, термінальна лінія і ідентифікатор групи процесів у процеса, що завершується збігаються, то всім процесам з тим же ідентифікатором групи процесів надсилається сигнал SIGHUP. Тим самим, завершуються і всі породжені в пріоритетному режимі процеси.

- Батьківському процесу надсилається сигнал SIGCHLD (завершення породженого процесу). Цей сигнал пробуджує батьківський процес, якщо той очікує завершення породжених процесів.

6.9 Планування

6.9.1 Алгоритми планування

UNIX – багатозадачна система. Це передбачає наявність декількох активних процесів, що одночасно намагаються отримати доступ до процесора. Система повинна вибирати між процесами. Частина ОС, що відповідає за це, називається планувальником, а використовуваний алгоритм – алгоритмом планування.

Ситуації, що активізують планувальник:

- при створенні процесу проблема полягає в тому, який процес запускати: батьківський або дочірній;

- при завершенні процесу з набору готових до виконання процесів вибрати і запустити наступний;

- при блокуванні процесу на операціях введення/виводу, семафорі або ін. Треба вибрати і запустити інший процес.

Існує безліч алгоритмів планування. Їх можна розділити на дві категорії:

- без перемикань (непріоритетні), тобто такі, коли обраний процес працює до блокування;

- з перемиканнями (пріоритетні), тобто такі, коли процес працює максимально можливий фіксований час. Таке планування вимагає переривань за таймером.

Планування процесів в UNIX ґрунтується на пріоритеті процесу. При цьому використовується циклічне планування за кількома чергами.

Суть **циклічного** планування полягає в наступному. Кожному процесу надається деякий квант часу процесора. Якщо до кінця кванта процес все ще працює, він переривається і відправляється в кінець списку, а управління передається іншому процесу. Якщо процес переривається або завершується раніше, перехід відбувається в цей момент. Найсуттєвіше в цьому алгоритмі це визначення оптимальної довжини кванта. Занадто маленький квант призводить до частих перемикань контексту, а це займає деякий час (= 1мс), а занадто великий квант призводить до повільного реагування на короткі інтерактивні запити. Розумним компромісом є значення кванта близько 20-30мс.

Суть **пріоритетного** планування в наступному: кожному процесу привласнюється пріоритет, і управління передається готовому до роботи процесу з найвищим пріоритетом. Пріоритети можуть присвоюватися статично та динамічно. Наприклад, для того, щоб якнайшвидше вивести процес з режиму ядра, його пріоритет автоматично підвищується. У багатьох ОС процеси групуються в класи за пріоритетами і використовується пріоритетне планування серед класів.

Припустимо, використовується чотири класи пріоритетів. Тоді планування відбувається так. Поки в класі 4 є готові до запуску процеси, вони запускаються один за іншим згідно з алгоритмом циклічного планування, і кожному відводиться квант часу. Якщо в класі 4 немає готових до запуску процесів, запускаються процеси класу 3 і т.д.

Якщо пріоритети постійні, до процесів класу 1 процесор може не дістатися. Тому в кожній ОС існують алгоритми динамічного зміни пріоритетів.

6.9.2 Пріоритети процесів в UNIX

Кожен процес має два атрибути пріоритету: поточний пріоритет, на підставі якого відбувається планування, і відносний пріоритет, званий також поправкою пріоритету – *nice number*, який задається при породженні процесу і впливає на поточний пріоритет.

Діапазон значень поточного пріоритету різний, в залежності від версії ОС UNIX і використовуваного планувальника. (В будь-якому випадку, процеси, що виконуються в режимі користувача, мають більш низький пріоритет, ніж працюють в режимі ядра.

Процеси, що виконуються в режимі користувача мають позитивні значення пріоритетів, в режимі ядра – негативні (негативні вище)).

У всіх UNIX-системах користувачі можуть при запуску процесу задавати значення поправки пріоритету за допомогою команди *nice*.

nice [-n [- | +] інкремент] команда

Діапазон значень інкремента в більшості систем від -20 до 20. Якщо інкремент не заданий, використовується стандартне значення 10. Позитивний інкремент означає зниження поточного пріоритету. Звичайні користувачі можуть задавати тільки позитивний інкремент і, тим самим, тільки знижувати пріоритет.

Користувач *root* може задати негативний інкремент, який підвищує пріоритет процесу і, тим самим, сприяє його більш швидкій роботі:

 # nice -n -10 make

Пріоритет процесу не є фіксованим і динамічно змінюється системою в залежності від використання обчислювальних ресурсів, часу очікування запуску і поточного стану процесу.

Основні класи пріоритетів розташовані в порядку убування:

- очікування дискового введення-виведення;
- очікування дискового буфера;
- очікування термінального введення;
- очікування виконання дочірнього процесу;
- призначений для користувача пріоритет.

При роботі процесів, раз в секунду пріоритет кожного процесу перераховується за формулою:

Pri = CPU_usage + nice + base

CPU_usage – використання ЦП – середнє значення тиків (переривань) таймера в секунду, які процес працював протягом останніх кількох секунд. При кожному тику таймера лічильник використання ЦП в таблиці процесів збільшується на 1. Цей лічильник потім додається до пріоритету процесу, тим самим знижуючи його пріоритет (чим числове значення пріоритету більше, тим сам пріоритет нижче). Т.ч., поки процес виконується в режимі завдання, його поточний пріоритет лінійно зменшується. Але для того, щоб не знизити пріоритет процесу остаточно, ядро послідовно зменшує величину CPU_usage (в різних версіях для цього використовуються різні формули, якість алгоритму планування визначається як раз ефективністю обчислення (CPU_usage, внаслідок чого краще враховуються інтереси фонових процесів).

Схема нумерації поточних пріоритетів різна для різних версій UNIX. Поділ між режимами ядра і завдання також залежить від версії. Наприклад, в SCO UNIX поділ такий: режим завдання – 0-65, режим ядра – 66-95, режим реального часу – 96-127.

Від версії залежить і кількість черг планування. Наприклад, SCO UNIX має 127 черг – по одній на кожен пріоритет. BSD використовує 32 черги, кожна з яких використовує діапазон пріоритетів.

6.9.3 Планування в Linux

Потоки в Linux реалізовані в ядрі. Тому планування засноване на потоках, а не на процесах. Планувальник розрізняє три класи потоків:

1 потоки реального часу, які обслуговуються за алгоритмом FIFO;

2 потоки реального часу, які обслуговуються в порядку циклічної черги;

3 потоки поділу часу.

1 – мають найвищий пріоритет і не можуть перериватися іншими потоками, за винятком таких же.

2 – те ж, що і 1, але з тією відмінністю, що вони можуть перериватися таймером, після чого потік переміщується в кінець своєї черги.

У кожного потоку є пріоритет планування. Крім цього, з кожним процесом зв'язаний квант часу, тобто кількість тиків таймера, протягом яких процес може виконуватися. За замовчуванням кожен тик дорівнює 10мс. Цей інтервал в Linux називають «Джиффі».

Планувальник використовує квант і пріоритет для визначення величини, званої «чеснотою» (goodness).

Якщо у процесу залишився невикористаний процесорний час, він отримує бонус, що дозволяє виграти в спірних ситуаціях. У цьому випадку перевага віддається процесу, що тільки що працював, тому що в цьому випадку його сторінки і кеш ще знаходяться в пам'яті і економиться час на їх заміну.

В іншому алгоритм працює наступним чином. Крім пріоритету з кожним процесом зв'язаний квант часу, тобто кількість тиків таймера, протягом яких процес може виконуватися. Вибирається потік з максимальним пріоритетом. Під час роботи його квант зменшується на одиницю на кожному тику. ЦП віднімається при потоці при виконанні одного з наступних умов:

- квант потоку зменшився до 0;
- потік блокується на операції введення-виведення;
- в стан готовності прийшов раніше заблокований потік з вищим пріоритетом.

Т.ч. потоки, пов'язані з введенням-виведенням отримують перевагу при плануванні.

Інша властивість цього алгоритму полягає в тому, що коли потоки змагаються за право використовувати ЦП, потік з більшим пріоритетом отримує більшу частку процесорного часу.

7 МІЖПРОЦЕСНА ВЗАЄМОДІЯ (IPC)

Міжпроцесна взаємодія – зв'язок між процесами, призначена для виконання деякої задачі, передачі даних і синхронізації роботи процесів.

Проблема взаємодії між процесами розбивається на три пункти:

1. Передача даних.

2. Контроль над діяльністю процесу (гарантія того, що два процесу не перетнуться в критичних ситуаціях – наприклад, коли обидва намагаються заволодіти останнім мегабайтом пам'яті).

3. Узгодження дій (один з процесів повинен поставляти дані, а інший отримувати).

В UNIX процеси виконуються у власних адресних просторах, але можуть спілкуватися один з одним за допомогою:

1. Каналів

2. Іменованих каналиів (FIFO)

3. Сигналів (програмних переривань)

4. Повідомлень

5. Семафора

6. Пам'яті, що поділяється

Канали

Між процесами створюється канал, в який один процес може послати потік байтів, а інший процес може його зчитати. Канали можуть бути використані тільки для обміну даними між спорідненими процесами і недоступні для незалежних процесів. Процес, що створює канал, використовує системний виклик pipe(), який повертає значення файлових дескрипторів для читання і запису в канал. Цей процес і його дочірні процеси успадковують і поділяють призначені файлові дескриптори. Канал організовується за допомогою конвеєрів оболонки.

Ls / | more (обидва процеси створені процесом shell і є спорідненими).

FIFO

FIFO дуже схожі на канали, оскільки є односпрямованим засобом передачі даних. Але при цьому мають імена, які дозволяють незалежним процесам отримати доступ до цих об'єктів. FIFO є окремим типом файлу. Імя FIFO є ім'я відповідного йому файлу.

Для створення іменованого каналу використовується системний виклик mknod().

Іменований канал можна створити з командного рядка *mknod <name>*

FIFO є засобом SystemV і не використовуються в BSD.

Звичайні канали та FIFO працюють за однаковими правилами:

- якщо читається менша кількість байт, ніж знаходиться в каналі, то залишок зберігається для наступного прочитання;

- якщо треба прочитати більше, ніж знаходиться в каналі, повертається доступна кількість байт, подальшими діями керує процес, що читає;

- якщо канал порожній і жоден процес не записує в нього, то буде отримано 0 байт;

- якщо в канал записуються дані, кількість яких менше ємності каналу, то порції даних, отриманих від різних процесів не перемішуються;

- якщо в канал треба записати більшу кількість байт, ніж він може вмістити, то запис блокується до звільнення необхідного місця.

Сигнали

Сигнали – найпростіша форма взаємодії процесів. Сигнали генеруються ядром і дозволяють повідомити процес або групу процесів про настання якої-небудь події.

Процес може послати сигнал тільки членам своєї групи, в тому числі всім одночасно. Процес, що володіє привілеями суперкористувача, може послати сигнал будь-якого процесу.

У процесу є вибір: проігнорувати сигнал, перехопити його або дозволити сигналу вбити процес.

Якщо процес вибрав перехоплення сигналу, він повинен вказати процедури обробки сигналів, тоді управління перейде до обробника сигналу.

Коли процедура обробки сигналу завершує свою роботу, управління знову повертається в те місце процесу, в якому воно знаходилося, коли сигнал прийшов.

Повідомлення

Цей метод використовує два примітиву: send і receive, які скоріше є системними викликами, ніж структурними компонентами мови, тому їх легко можна помістити в бібліотечні процедури. Перший запит посилає повідомлення заданому адресату, а другий – отримує повідомлення від зазначеного адресата. Якщо повідомлення немає, другий запит блокується до надходження повідомлення, або негайно повертає код помилки.

Повідомлення розміщуються в адресному просторі ядра у вигляді односпрямованого зв'язного списку і є системним ресурсом, що розділяється. Кожна черга повідомлень має свій унікальний ідентифікатор. Процеси можуть записувати і зчитувати повідомлення з різних черг. Процес, що послав повідомлення в чергу, може не очікувати читання цього повідомлення будь-яким іншим способом. Він може закінчити своє виконання, залишивши в черзі повідомлення, яке буде прочитано пізніше.

Дана можливість дозволяє процесам обмінюватися структурованими даними, які мають такі атрибути:

- тип повідомлення (дозволяє мультиплексувати повідомлення в одній черзі);

- довжина даних в байтах (може бути нульовою);

- дані (можуть бути структурованими).

Для кожної черги ядро створює заголовок, де зберігається інформація про права доступу до черги, її поточний стан, покажчик на перше й останнє повідомлення. Кожен елемент списку є окремим повідомленням. Для створення черги повідомлень або для доступу до існуючої використовується системний виклик msgget().

В одній черзі можна мультиплексувати повідомлення від різних процесів.

Мультиплексування: припустимо серверний процес обмінюється даними з декількома клієнтами. Тоді для такого обміну можна використовувати одну чергу повідомлень. Для цього повідомленнями, які направляються від будь-якого з клієнтів до сервера, присвоюється тип 1. Якщо клієнт ідентифікований (наприклад, має свій pid), то сервер може передавати повідомлення, привласнюючи тип повідомлення, рівний цьому ідентифікатору.

Семафори

Це ціла змінна, яка дорівнює нулю в разі відсутності сигналів активізації або деякому позитивному числу, відповідному кількості відкладених сигналів, що активізують.

Семафори не призначені для обміну великими обсягами даних як у випадку з FIFO або повідомлень. Замість цього вони управляють доступом до того чи іншого ресурсу.

Необхідним є дотримання таких умов:

- значення семафора має бути доступне різним процесам. Тому семафор знаходиться не в адресному просторі процесу, а в адресному просторі ядра;

- операція перевірки і зміни семафора повинна бути реалізована у вигляді атомарної по відношенню до інших процесів операції (не переривати іншими процесами). Єдиним способом гарантувати це є виконання процесу в режимі ядра.

Т.ч. семафори є системним ресурсом, дії над яким здійснюється через інтерфейс системних викликів.

Семафори мають наступні характеристики:

- представляє не один лічильник, а групу, що складається з декількох лічильників, об'єднаних спільними ознаками (дескрипторами, правами доступу);

- кожен з них може приймати будь-яке позитивне значення в межах, визначених системою.

Для кожної групи семафорів ядро підтримує спеціальну структуру, яка містить:

- значення семафора;
- покажчик на перший елемент масиву;
- число семафорів в групі;
- час останньої операції.

Значення конкретного семафора зберігається у внутрішній структурі sem:

- значення;
- ідентифікатор процесу, який виконав останню операцію над семафором;
- число процесів, що очікують збільшення значення семафора;
- число процесів, що очікують обнуління семафора.

Можливі три операції над семафорами або комбінації з них:

- значення збільшується на одиницю;
- процес очікує обнуління семафора;
- процес очікує, поки значення семафора не стане більше або рівною величині semop, що обчислюється з значень семафора.

Реалізація потоків

Реалізація потоків залежить від того, підтримуються вони ядром чи ні. Перевага реалізації потоків в просторі користувача полягає в тому, що вони легко реалізуються без необхідності зміни ядра, а перемикання потоків відбувається дуже ефективно.

Недолік у тому, що якщо один з потоків заблокується, всі потоки процесу блокуються. Ядро вважає, що існує тільки один потік, і не передає управління процесу потоку, поки блокування не знімається. При використанні потоків на рівні користувача вони повністю реалізуються в динамічній бібліотеці в просторі користувача.

Як і процесами, управління потоками відбувається за допомогою механізму системних викликів.

Синхронізація потоків може здійснюватися за допомогою м'ютексів. **М'ютекс** є перемикач, що має два значення (0 або 1) і охороняє будь-який спільно використовуваний ресурс, наприклад буфер. Передбачається, що потоки блокують м'ютекс перед зверненням до ресурсу і розблокують, коли ресурс їм більше не потрібен.

М'ютекси призначені для короткочасного блокування і не призначені для довготривалої синхронізації. Для цього надаються змінні стану. Використовуються вони так: один потік чекає, коли змінна прийме певне значення, а інший потік сигналізує йому зміною цієї змінної.

М'ютекси вважаються семафорами, а змінні стану – ні.

ОС Linux підтримує потоки в ядрі. Створюються вони системним викликом clone(). Потоки можуть створюватися як в поточному процесі, так і в новому. Якщо потік знаходиться в поточному процесі, він спільно з іншими потоками використовує адресний простір і будь-яка зміна кожного байта в адресному просторі будь-яким потоком тут же стає видимою всім іншим потокам процесу. Якщо ж потік створюється в новому процесі, він отримує копію адресного простору вихідного процесу, але наступні зміни в пам'яті вже не видно іншим потокам. В обох випадках потік отримує свій власний стек, при цьому покажчик стека задається параметром створення потоку.

8 УПРАВЛІННЯ ПАМЯТЮ

Основна (оперативна) пам'ять (ОП) завжди була критичним ресурсом. Тому первинною функцією всіх ОС було забезпечення поділу основної пам'яті між конкуруючими процесами.

8.1 Основні поняття

8.1.1 Віртуальна память (ВП)

ОП представлена у вигляді послідовності байтів, кожен з яких має свою унікальну адресу, звану фізичною адресою. Ці адреси використовує процесор. Якби процес під час виконання використовував фізичні адреси виникли такі проблеми:

- захисту адресних просторів процесів;

- розподілу існуючого адресного простору (кожен процес повинен займати безперервну і непересічну послідовність адрес);

- обмеження число процесів, що працюють в системі (якщо вичерпається весь обсяг ОП, більше процесів запустити буде не можна).

Перераховані проблеми вирішуються за допомогою ВП. Віртуальні адреси транслюються в фізичні відповідно до таблиць відображення ядра під управлінням спеціального механізму апаратного рівня MMU (рисунок 8.1).

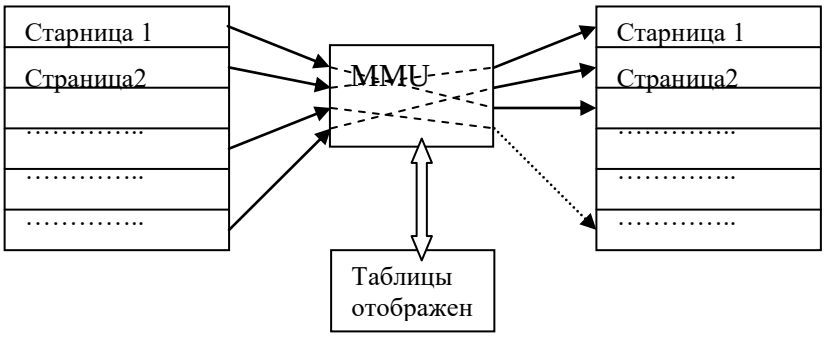

Рисунок 8.1 – Відповідність віртуальних і фізичних адрес

Тепер:

- кожен процес буде виконуватися у власному адресному просторі, що належить тільки йому;

- послідовні віртуальні адреси виділити набагато легше, ніж фізичні;

- ВП не обмежена розмірами ОП.

ВП складається з двох частин – ОП і області підкачки – свопінга – дискового простору, де можуть зберігатися тимчасово ділянки процесу, які не використовуються.

8.1.2 Методи організації пам’яті

У сучасних процесорах використовується два методи об'єднання адресного простору:

1. сегменти – області змінного розміру;

2. сторінки – області фіксованого розміру.

При використанні **сегментів** пам'ять розділяється на кілька логічних елементів, кожен з яких складається з безперервної послідовності адрес, що лежать в заданому діапазоні і відображаються в безперервну послідовність фізичних адрес.

Віртуальний адресу при цьому складається з двох частин:

- дескриптора сегмента (адреса, розмір, права доступу);

- зміщення відносно початку сегмента.

Переваги: низька фрагментація.

Недоліки:

- сегмент повинен цілком бути присутнім в ОП під час виконання процесу, що призводить до проблем одночасного виконання декількох великих процесів;

- обмін областями змінного розміру досить складний.

Сторінковий механізм забезпечує більшу гнучкість. У цьому випадку віртуальний адресний простір розбивається на блоки однакового розміру, звані сторінками. (Напр., Intel використовує розбиття по 4Кбт). Сторінка, як і сегмент має цілком знаходитися в ОП, але за рахунок того, що вона має менший і фіксований розмір забезпечити це легше. Сегменти складаються з сторінок.

Переваги:

- система оперує областями досить малого розміру;

- допускає, щоб частина сегмента перебувала в ОП, а частина в області свопінгу;

- сторінки сегмента можуть розташовуватися в фізичної пам'яті в довільному місці і порядку (схоже на схему розташування файлів). Сегменту досить знати з яких сторінок він складається.

8.1.3 Адресний простір процесу

Під адресний простір процесу відводиться певний набір адрес, починаючи від 0 до деякого максимуму. У найпростішому випадку максимальна величина адресного простору менше обсягу ОП. Тоді процес може заповнити свій адресний простір, і пам'яті вистачить на те, щоб утримувати його цілком.

Якщо ж образ процесу повністю не поміщається в ОП, то ОС зберігає в ОП лише частину адрес, а іншу частину зберігає на диску в певному місці і при необхідності змінює їх місцями. Такий механізм називається механізмом віртуальної пам'яті (рисунок 8.2).

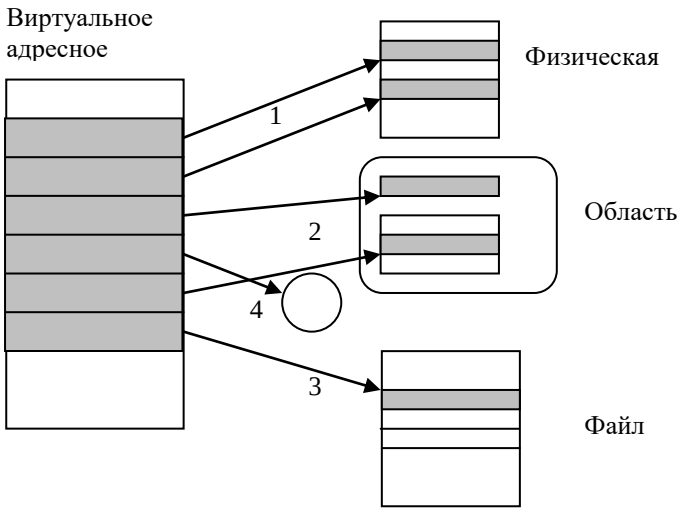

Рисунок 8.2 – Механізм віртуальної пам'яті

Можливе місцезнаходження фізичних сторінок процесу:

1. Віртуальна адреса асоційована зі сторінкою фізичної пам'яті.

2. Сторінка переміщена в область свопінгу (звернення до цієї сторінки призведе до сторінкової помилки, що зажадає від ядра її розміщення в пам'яті).

3. Адресуєма сторінка знаходиться в файлі на диску (- || -).

4. Адресуєма сторінка відсутня в пам'яті і вона не асоційована ні з областю свопінга, ні з файлом (в цьому випадку розміщується нова сторінка, заповнена нулями).

Основна ідея віртуальної пам'яті – забезпечення захисту призначених для користувача програм однієї від іншої і надання ОС можливості динамічно гнучко перерозподіляти ОП між одночасно підтримуваними користувацькими процесами. Процес «бачить» тільки власний адресний простір.

8.1.4 Методи управління ОП

Свопінг (підкачка)

Це найбільш простий метод управління оперативною пам'яттю.

Переміщенням даних між пам'яттю і диском управляє верхній рівень планувальника – свопер. Дані на диск вивантажуються в тому випадку, коли у ядра закінчується вільна пам'ять.

До версії 3BSD більшість систем ґрунтувалося на повному свопінгу. Коли завантажується більше процесів, ніж могло поміститися в пам'яті, деякі з них вивантажуються на диск. Причому вивантажуються процес цілком, таким чином процес міг бути або в пам'яті, або на диску.

Цим переміщенням управляє верхній рівень планувальника – процес під назвою свопер (процес 0).

Сторінкове заміщення

Починаючи з версії 4BSD підкачка здійснюється посторінково. Процесу не потрібно повністю перебувати в пам'яті. Сторінки з сегментами тексту, даних і стека завантажуються динамічно, в міру звернення до них.

Сторінкова підкачка реалізується частково ядром, частково сторінковим демоном. Як всі демони, сторінковий демон періодично запускається і дивиться, чи є для нього робота. Якщо він виявляє, що кількість сторінок у списку вільних сторінок занадто мало, він ініціалізує дію зі звільнення додаткових сторінок.

В UNIX-подібних системах використовується деякий полегшений варіант алгоритму підкачки, заснований на використанні поняття робочого набору. Основна ідея полягає в оцінці робочого набору процесу на основі використання апаратно встановлюваних ознак звернення до сторінок ОП.

Періодично для кожного процесу сторінковим демоном проводяться наступні дії:

- проглядаються таблиці відображення всіх сегментів ВП цього процесу;

- якщо елемент таблиці відображення містить посилання на описувач фізичної сторінки, то аналізується ознака звернення;

- якщо ознака встановлена, то вважається, що сторінка входить в робочий набір даного процесу, і скидається в нуль лічильник старіння даної таблиці;

- якщо ознака не встановлена, то до лічильника старіння додається одиниця, а сторінка набуває статусу кандидата на вихід з робочого набору процесу;

- якщо при цьому значення лічильника досягає деякого критичного значення, вважається що сторінка вийшла з набору процесу, і її описувач заноситься в список сторінок, які можна відкачати в зовнішню пам'ять.

По ходу перегляду елементів таблиць відображення в кожному з них ознака звернення гаситься.

Відкачування сторінок, що не входять в робочі набори процесів, робить спеціальний системний процес – stealer. Він починає працювати, коли кількість сторінок у списку вільних сторінок досягає встановленого нижнього порога. Функцією цього процесу є аналіз необхідності відкачування сторінки в відповідну область зовнішньої

пам'яті (тобто або в системну область підкачки – swapping для анонімних сторінок, або в певний блок файлової системи для сторінки, що входить в сегмент відображуваного файлу).

Очевидно, робочий набір будь-якого процесу може змінитися під час його виконання. Іншими словами, можлива ситуація, коли процес звертається до ВП, відсутньої в ОП. В цьому випадку зазвичай виникає апаратне переривання, в результаті якого починає працювати ОС. Подальший хід подій залежить від обставин. Якщо список описувачів вільних сторінок не порожній, то з нього вибирається деякий описувач, і відповідна сторінка підключається до ВП процесу.

Але якщо виникає вимога сторінки в умовах, коли список порожній, то починає працювати механізм свопінгу. Основний привід для застосування іншого механізму полягає в тому, що просте відібрання сторінки у будь-якого процесу, включаючи той, який зажадав сторінку, потенційно вело б до ситуації trashing, оскільки руйнувало б робочий набір деякого процесу. Будь-який процес, зажадавши сторінку не з свого поточного робочого набору, стає кандидатом на свопінг. Йому більше не даються ресурси процесора, і описувач процесу стає в чергу до системного процесу – swapper. У цій черзі може перебувати кілька процесів. Свопер по черзі здійснює повний свопинг цих процесів (тобто відкачування всіх сторінок їх ВП, які присутні в ОП), поміщаючи відповідні описувачі фізичних сторінок в список вільних сторінок, до тих пір, поки кількість сторінок в цьому списку не досягне встановленої в системі верхньої межі. Після завершення повного свопінга кожного процесу одному з процесів з черги до своперу дається можливість спробувати продовжити своє виконання.

Сторінкове заміщення має ряд важливих переваг у порівнянні зі свопінгом:

1. Розмір програми обмежується лише розміром ВП (на 32-розрядному процесорі це становить 4 Гб).

2. Запуск програми відбувається дуже швидко, так як не треба завантажувати в пам'ять всю програму цілком.

3. Значно більша кількість програм може виконуватися одночасно.

4. Переміщення окремих сторінок між ОП і вторинної пам'яттю вимагають менших витрат, ніж переміщення процесу цілком.

8.2 Апаратна частина ВП

Управління пам'яттю UNIX ділиться на апаратно-залежну та апаратно-незалежну частини.

Ідея апаратної частини механізму віртуальної пам'яті полягає в тому, що адреса пам'яті, що виробляється командою, інтерпретується апаратурою не як реальна адреса

деякого елементу основної пам'яті, а як деяка структура, різні поля якої обробляються різним чином.

Універсальна апаратно-незалежна частина зв'язується з конкретною апаратною реалізацією за допомогою апаратно-залежної частини.

На апаратно-незалежному рівні сегментна організація віртуальної пам'яті кожного процесу описується структурою *as*.

На рівні сторінок підтримується два види описових структур. Для кожної сторінки фізичної ОП існує описувач, що входить в один з трьох списків.

Перший список включає описувачі сторінок, що не допускають модифікації або відображаються в область зовнішньої пам'яті будь-якого файлу. Для таких сторінок не потрібний простір в області підкачки (вони або зовсім не вимагають відкачування, або відкачування виробляється в інше місце).

Другий – це список описувачів вільних сторінок (які не підключені до жодної віртуальної пам'яті). Такі сторінки вільні для використання і можуть бути підключені до будь-якої ВП.

Третій – включає описувачі структур анонімних сторінок (ті, які можуть змінюватися, але для яких немає «рідного місця» у зовнішній пам'яті).

У будь-якому описувачі фізичної сторінки зберігаються копії ознак звернення і модифікації сторінки, що виробляються конкретною використовуваної апаратурою.

Для кожного сегмента підтримується таблиця відображення, що зв'язує адреси віртуальних сторінок, що входять до нього з описувачем відповідних їм фізичних сторінок з першого або третього списків описувачів фізичних сторінок для віртуальних сторінок, присутніх в ОП, або з адресами копій сторінок у зовнішній пам'яті для віртуальних сторінок, відсутніх в ОП. (Підтримується окрема таблиця для кожного приватного сегмента і одна загальна для кожного сегмента, що розділяється).

Введення подібної узагальненої моделі організації ВП дозволило добитися того, що звернення до пам'яті, які не потребують втручання ОС, виробляються безпосередньо з використанням конкретних апаратних засобів. Разом з тим, всі найбільш відповідальні дії ОС, пов'язані з управлінням ВП, виконуються в апаратно-незалежній частині з необхідними взаємодіями з апаратно-залежною частиною.

8.3 Системні виклики управління пам’яттю

Управління пам'яттю відбувається за допомогою системних викликів.

Стандартом POSIX системні виклики для управління пам'яттю не визначаються. Кожна версія UNIX має свої виклики управління пам'яттю. Найбільш поширені з них:

g = brk (addr) Змінити розмір сегмента даних

a = mmap (addr, len, flags, ...,) Показати файл на пам'ять

a = unmap (addr, len) Скасувати відображення

9 ПОДСИСТЕМА ВВЕДЕННЯ-ВИВЕДЕННЯ

Підсистема управління введенням-виведенням дозволяє процесам підтримувати зв'язок з периферійними пристроями, наприклад, дисками, терміналами, принтерами і мережею, з одного боку, і з модулями ядра, які керують пристроями і іменуються драйверами пристроїв, з іншого.

Кожен пристрій введення-виведення обчислювальної системи – диск, принтер, термінал і т. п. – забезпечено спеціалізованим блоком управління, званим **контролером**. Контролер взаємодіє з *драйвером* – системним програмним модулем, призначеним для управління цим пристроєм. Контролер періодично приймає від драйвера виведену на пристрій інформацію, а також команди управління, які говорять про те, що з цією інформацією потрібно зробити (наприклад, вивести у вигляді тексту в певну область терміналу або записати в певний сектор диска). Під управлінням контролера пристрій може деякий час виконувати свої операції автономно, не вимагаючи уваги з боку центрального процесора. Цей час залежить від багатьох факторів – обсягу виведеної інформації, ступеня інтелектуальності керуючого пристроєм контролера, швидкодії пристрою і т. п. Навіть найпримітивніший контролер, який виконує прості функції, зазвичай витрачає досить багато часу на самостійну реалізацію подібної функції після отримання чергової команди від процесора. Це ж справедливо і для складних контролерів, так як швидкість роботи будь-якого пристрою введення-виведення, навіть самого швидкісного, зазвичай істотно нижче швидкості роботи процесора.

Процеси, що відбуваються в контролерах, протікають в періоди між видачами команд незалежно від ОС. Підсистема введення-виведення повинна спланувати в реальному масштабі часу (в якому працюють зовнішні пристрої) запуск і припинення великої кількості різноманітних драйверів, забезпечивши прийнятний час реакції кожного драйвера на незалежні події контролера. З іншого боку, необхідно мінімізувати завантаження процесора завданнями введення-виведення, залишивши якомога більше процесорного часу на виконання користувальницьких потоків.

Дане завдання є класичною задачею планування систем реального часу і зазвичай вирішується на основі багаторівневої пріоритетної схеми обслуговування по перериваннях. Для забезпечення прийнятного рівня реакції всі драйвери (або частини драйверів) розподіляються по кільком пріоритетним рівням відповідно до вимог до часу

реакції і часу використання процесора. Для реалізації пріоритетної схеми зазвичай задіюється загальний диспетчер переривань ОС.

9.1 Багатошарова модель підсистеми введення-виведення

Багатошарова побудова програмного забезпечення, характерне для операційних систем взагалі, виявляється особливо природним і корисним при побудові підсистеми введення-виведення. При великій різноманітності пристроїв введення-виведення, що володіють істотно різними характеристиками (принтер і диски, графічний монітор і мережевий адаптер і т. п.), ієрархічна структура програмного забезпечення дозволяє дотримати баланс між двома досить суперечливими вимогами: з одного боку, необхідно врахувати всі особливості кожного пристрою, а з іншого боку, забезпечити єдине логічне представлення і уніфікований інтерфейс для пристроїв всіх типів. При цьому нижні шари підсистеми введення-виведення повинні включати індивідуальні драйвери, написані для конкретних фізичних пристроїв, а верхні шари повинні узагальнювати процедури управління цими пристроями, надаючи загальний інтерфейс якщо не для всіх пристроїв, то принаймні для груп пристроїв, що володіють деякими загальними характеристиками, наприклад для принтерів певного виробника або для всіх лазерних принтерів і т. п.

Багатошаровість структури, безумовно, полегшує вирішення більшості перерахованих в попередньому розділі задач підсистеми введення-виведення, таких як простота включення нових драйверів, підтримка декількох файлових систем, динамічне завантаження-вивантаження драйверів і інших.

Узагальнена структура підсистеми введення-виведення представлена на рис. 9.1.

З рисунка видно, що програмне забезпечення введення-виведення ділиться не тільки на горизонтальні шари, але і на вертикальні. Це пояснюється тим, що для такого різноманітного світу, як зовнішні пристрої, важко забезпечити однаковість у разі необхідності розділення функцій управління на шари. Тому загальний принцип багатошаровості залишається справедливим, однак для пристроїв певного типу він реалізується по-різному, зі своєю кількістю шарів і їх функціями. У поданій структурі наведено приклади трьох вертикальних підсистем, що керують дисками, графічними пристроями, такими як монітори, принтери та плотери, і мережевими адаптерами. Природно, до цього переліку можна додати й інші, наприклад підсистему управління символьними терміналами або будь-якими спеціалізованими пристроями, такими як аналого-цифрові і цифро-аналогові перетворювачі.

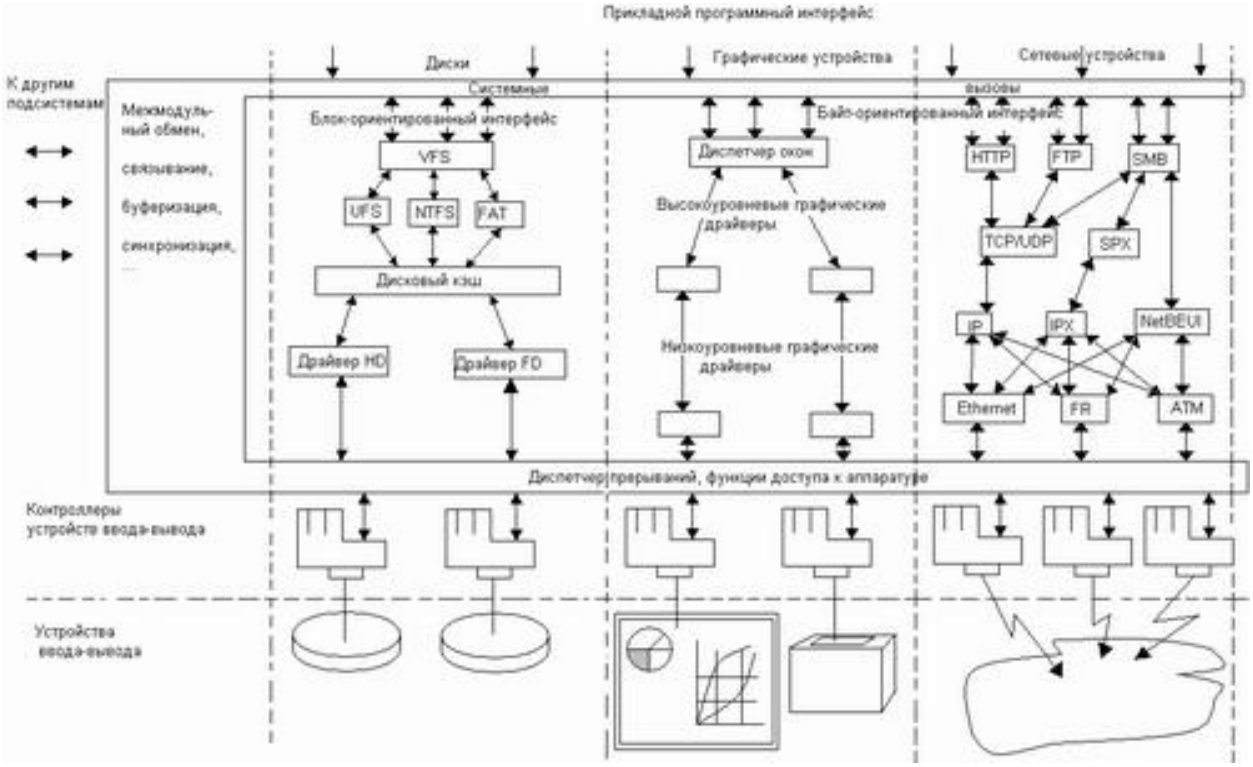

Рисунок 9.1 – Структура підсистеми введення-виведення

У кожній вертикальній підсистемі існує кілька шарів модулів. Нижній шар утворюють так звані апаратні драйвери пристроїв, назва яких відображає той факт, що вони управляють апаратурою зовнішніх пристроїв, здійснюючи обмін байтами і блоками байтів, і не мають, як правило, справи з більш високорівневими питаннями логічної організації даних, наприклад з файлами або складними графічними об'єктами.

9.2 Багаторівневі драйвери

Спочатку термін «драйвер» застосовувався в досить вузькому сенсі: під драйвером розумівся програмний модуль, який:

- входить до складу ядра операційної системи, працюючи в привілейованому режимі;

- безпосередньо керувати зовнішнім пристроєм, взаємодіючи з його контролером за допомогою команд введення-виведення комп'ютера;

- обробляє переривання від контролера пристрою;

- надає прикладному програмісту зручний логічний інтерфейс роботи з пристроєм, екрануючи від нього низькорівневі деталі управління пристроєм і організації його даних;

- взаємодіє з іншими модулями ядра ОС за допомогою суворо обумовленого інтерфейсу, що описує формат переданих даних, структуру буферів, способи включення

драйвера до складу ОС, способи виклику драйвера, набір загальних процедур підсистеми введення-виведення, якими драйвер може користуватися, і т. п.

Поступово, у міру розвитку операційних систем і ускладнення структури підсистеми введення-виведення, поряд з традиційними драйверами в операційних системах з'явилися так звані високорівневі драйвери, які розташовуються в загальній моделі підсистеми введення-виведення над традиційними драйверами. Поява високорівневих драйверів можна вважати подальшим розвитком ідеї багатошарової організації підсистеми вводу-виводу. Замість того щоб концентрувати всі функції по управлінню пристроєм в одному програмному модулі, у багатьох випадках набагато ефективніше розподілити їх між кількома модулями в сусідніх шарах ієрархії. Традиційні драйвери, які стали називати апаратними драйверами, низькорівневими драйверами, або драйверами пристроїв, підкреслюючи їх безпосередній зв'язок з керованим пристроєм, звільняються від високорівневих функцій і займаються тільки низькорівневими операціями. Ці низькорівневі операції складають фундамент, на якому можна побудувати той чи інший набір операцій в драйверах більш високих рівнів.

При такому підході підвищується гнучкість і розширюваність функцій з управління пристроєм – замість жорсткого набору функцій, зосереджених в єдиному драйвері, адміністратор ОС може вибрати необхідний набір функцій, встановивши потрібний високорівневий драйвер. Якщо різним додаткам необхідно працювати з різними логічними моделями одного і того ж фізичного пристрою, то для цього достатньо встановити в системі кілька драйверів на одному рівні, що працюють над одним апаратним драйвером.

Кількість рівнів драйверів в підсистемі введення-виведення зазвичай не обмежується будь-якою межею, але на практиці найчастіше використовують від двох до п'яти рівнів драйверів – занадто велика кількість рівнів може знизити швидкість операцій введення-виведення. Кілька драйверів, які керують одним пристроєм, але на різних рівнях, можна розглядати як набір окремих драйверів або як один багаторівневий драйвер.

Високорівневі драйвери оформляються за тими ж правилами і дотримуються тих же внутрішніх інтерфейсів, що і апаратні драйвери. Єдиною відмінністю є те, що високорівневі драйвери, як правило, не викликаються по перериваннях, так як взаємодіють з керованим пристроєм через посередництво апаратних драйверів.

В операційній системі UNIX всі драйвери розділені на два великі класи: блок-орієнтовані (block-oriented) драйвери і байт-орієнтовані (character-oriented) драйвери. Цей поділ є більш загальним, ніж показане на рисунку 9.1 поділ на вертикальні підсистеми.

Наприклад, драйвери графічних пристроїв і драйвери мережевих пристроїв відносяться до класу байт-орієнтованих.

Блок-орієнтовані драйвери керують пристроями прямого доступу, які зберігають інформацію в блоках фіксованого розміру, кожен з яких має власну адресу. Найпоширеніший зовнішній пристрій прямого доступу – диск. Адресуємість блоків призводить до того, що для пристроїв прямого доступу з'являється можливість кешування даних в оперативній пам'яті, і ця обставина значно впливає на загальну організацію введення-виведення для блок-орієнтованих драйверів.

Пристрої, з якими працюють байт-орієнтовані драйвери, які не адресовані і не дозволяють виробляти операцію пошуку даних, вони генерують або споживають послідовності байт. Прикладами таких пристроїв, які також називають пристроями послідовного доступу, служать термінали, рядкові принтери, мережні адаптери.

Блок-або байт-орієнтованість є характеристикою як самого пристрою, так і драйвера. Очевидно, що якщо пристрій не підтримує обмін адресованими блоками даних, а дозволяє записувати або зчитувати сукупність електронних даних, то і пристрій, і його драйвер можна назвати байт-орієнтованими. Для байт-орієнтованого пристрою неможливо розробити блок-орієнтований драйвер. Пристрій прямого доступу з блочною адресацією є блок-орієнтованим, і для управління ним природно використовувати блок-орієнтований драйвер. Однак блок-орієнтованим пристроєм можна керувати і за допомогою байт-орієнтованого драйвера. Так, диск можна розглядати не тільки як набір блоків, але і як набір байт, перший з яких починає перший блок диска, а останній завершує останній блок. Фізичний обмін з контролером пристрою як і раніше здійснюється блоками, але байт-орієнтований драйвер пристрою буде перетворювати блоки в електронних даних. Для пристроїв прямого доступу часто розробляють пару драйверів, щоб до пристрою можна було звертатися і по байт-орієнтованих, і по блок-орієнтованих інтерфейсів в залежності від потреб.

Розподіл всіх драйверів на блок-орієнтовані і байт-орієнтовані виявляється корисним для структурування підсистеми управління введення-виведення. Проте необхідно враховувати, що ця схема є спрощеною – є зовнішні пристрої, драйвери яких не відносяться ні до одного класу, наприклад таймер, який, з одного боку, не містить адресуємої інформації, а з іншого боку, не породжує потоку байт. Це пристрій тільки видає сигнал переривання в деякі моменти часу.

10 МЕРЕЖА В UNIX

Мережі стали невід'ємною складовою сучасних обчислювальних систем. Технології взаємодії між інформаційними системами розвиваються з 1970-х років, не набагато випередивши розвиток UNIX. Операційна система UNIX майже з самого народження інтегрувала в себе технології організації локальних мереж, на її основі потім була побудована мережа Internet, що поширена нині по всьому світу.

Організація взаємодії між пристроями і програмами в мережі є складним завданням. Мережа об'єднує різне устаткування, різні операційні системи та програми – їх успішна взаємодія була б неможлива без прийняття загальноприйнятих правил, стандартів.

В області комп'ютерних мереж існує безліч міжнародних та галузевих стандартів, серед яких слід особливо виділити міжнародний стандарт OSI і набір стандартів IETF (Internet Engineering Task Force).

10.1 Семирівнева модель OSI

У моделі OSI, званої також моделлю взаємодії відкритих систем (Open Systems Interconnection – OSI) і розробленої Міжнародною Організацією по Стандартам (International Organization for Standardization – ISO), засоби мережної взаємодії діляться на сім рівнів, для яких визначені стандартні назви і функції.

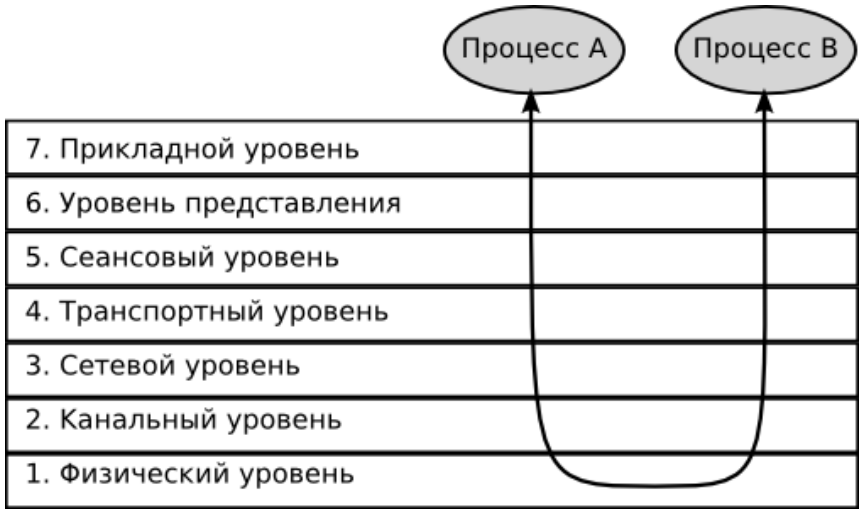

Рисунок 10.1 – Рівні ISO OSI

Кожен рівень надає інтерфейс до вищого рівня, приховуючи деталі реалізації. При побудові транспортної підсистеми будь-якої програми найбільший інтерес представляють функції фізичного, канального і мережевого рівнів, тісно пов'язані з використовуваним в даній мережі обладнанням: мережевими адаптерами, концентраторами, мостами, комутаторами, маршрутизаторами. Функції прикладного і сеансового рівнів, а також рівня представлення реалізуються операційними системами і системними додатками кінцевих вузлів. Транспортний рівень виступає посередником між цими двома групами протоколів.

Процеси, що знаходяться на різних вузлах мережі, при обміні інформацією взаємодіють тільки з самим верхнім рівнем, але дані при просуванні по мережі проходять спочатку через всі рівні – від самого верхнього до самого нижнього – на одному вузлі і потім у зворотному порядку на іншому вузлі.

10.2 Протоколи Internet: TCP/IP

Протоколи TCP/IP (Transmission Control Protocol/Internet Protocol) були розроблені на замовлення Міністерства оборони США 30 років тому для організації зв'язку в рамках експериментальної мережі ARPAnet і являє собою набір загальних протоколів для різнорідного обчислювального середовища. Перша реалізація стека TCP/IP була створена в стінах університету Берклі для операційній системи UNIX. Популярність UNIX і вдалі ідеї, закладені в TCP/IP, привели до утворення і бурхливому розвитку міжнародної мережі Internet. Всі протоколи сімейства TCP/IP проходять стандартизацію в організації IETF через випуск так званих RFC-документів (Request For Comment).

Протоколи, що входять в TCP /IP, частково відповідають моделі OSI. Стек TCP/IP підтримує на нижньому рівні всі популярні стандарти фізичного і канального рівнів: для локальних мереж – Ethernet, Token Ring, FDDI, для глобальних – PPP, ISDN. Основними протоколами стека, що дали йому назву, є протоколи IP і TCP, що відносяться до мережевого і транспортного рівнів відповідно. IP забезпечує просування пакету по мережі, TCP гарантує надійність його доставки.

За довгі роки використання стек TCP/IP поповнився безліччю протоколів прикладного рівня: FTP, SMTP, HTTP і т. П.

Рисунок 10.2 – Відповідність стека TCP/IP моделі OSI

Оскільки стек TCP/IP спочатку створювався для глобальної мережі, він має багато особливостей, що дають йому перевагу перед іншими протоколами, коли мова заходить про глобальні комунікації. Це можливість фрагментації пакетів, гнучка система адресації, простота широкомовних запитів. На сьогодні TCP/IP – найпоширеніший протокол обчислювальних мереж.

Для UNIX-систем TCP/IP історично був першим і залишається основним мережевим протоколом.

Розглянемо, як в UNIX здійснюється адміністрування мережевих з'єднань на основі TCP/IP.

10.2.1 Мережевий інтерфейс в UNIX

Основою мережевої підсистеми UNIX є *мережевий інтерфейс*. Мережевий інтерфейс – це абстракція, що зв'язує канальний і мережевий (протокол TCP/IP) рівні мережі в UNIX.

Мережевий інтерфейс створюється в момент завантаження драйвера мережевого пристрою (наприклад, мережевої карти) або створення логічного з'єднання (наприклад, в разі PPP). Кожен мережевий інтерфейс в системі має унікальне ім'я, що складається з типу пристрою і номера (0 або більше для однотипних пристроїв). Під типами пристроїв в різних UNIX-системах може розумітися вид протоколу канального рівня (Ethenet - eth) або назва драйвера пристрою (Realtek - rl).

Інтерфейс має набір параметрів, велика частина яких відносяться до мережевого рівня (IP-адреса, маска мережі і т. п.). Важливим параметром мережевого інтерфейсу є апаратна адреса (у разі Ethernet апаратна адреса називається MAC-адресом і складається з

шести байтів, які прийнято записувати в шістнадцятковій системі числення і розділяти двокрапкою).

Основною утилітою для отримання доступу до параметрів мережевих інтерфейсів в UNIX служить **ifconfig**. В даний час в ряді систем поряд з нею використовується більш сучасна утиліта **ip**, з відмінним синтаксисом параметрів командного рядка.

Утиліта ifconfig, викликана в командному рядку без параметрів, виводить відомості про всі налаштовані в даний момент мережеві інтерфейси. Щоб переглянути інформацію про конкретний інтерфейс, необхідно вказати його ім'я в якості параметра ifconfig.

Зазвичай виконання ifconfig вимагає прав суперкористувача або дозволяється користувачам, що входять в спеціальну «адміністративну» групу (наприклад, netadmin). Налаштування мережевих параметрів, пов'язаних з інтерфейсом, виконується за допомогою тієї ж утиліти ifconfig.

Для діагностики трафіку на канальному рівні застосовується спеціальні програми. Найпоширенішими в UNIX є **tcpdump** і **ethereal**. При «прослуховуванні» каналу, ці програми взаємодіють із заданим мережевим інтерфейсом.

10.3 Конфігурація IP-мереж

Мережева адреса

В IP-мережах кожному мережевому інтерфейсу присвоюється певна єдина на всю глобальну мережу адреса, яка не залежить від середовища передачі даних і завжди має один і той же формат.

Формат адреси залежить від версії протоколу. У найбільш поширеній зараз четвертій версії адреса складається з чотирьох байт, що записуються традиційно в десятковій системі числення і розділяються крапкою. Тоді як в шостій версії протоколу IP адреса складається вже з 16 байт і зазвичай записується в шістнадцятковій системі числення.

Для того щоб призначити інтерфейсу IP-адресу, досить виконати **ifconfig**, вказавши після імені інтерфейсу IP-адреса: *ifconfig eth0 192.168.1.1*

Маршрутизація

Як було сказано вище, IP-адреса включає дві частини: адресу підмережі та адресу конкретного вузла в рамках цієї підмережі. Маска підмережі показує, скільки біт в IP-адресі містять адресу підмережі (решта – це адреса вузла). Таким чином, маючи в своєму розпорядженні IP-адресу вузла призначення і маску підмережі завжди можно визначити, чи стосується вузол призначення тієї ж підмережі. В цьому випадку пакети до них будуть доставлятися безпосередньо через канальний рівень.

Більш складне питання постає, якщо IP-адреса вузла-адресата не входить в локальну мережу вузла-відправника. Адже і в цьому випадку пакет необхідно відіслати якомусь абоненту локальної мережі, з тим, щоб той перенаправив його далі. Цей абонент, маршрутизатор, підключений до декількох мереж, і йому ставиться в обов'язок пересилати пакети між ними за певними правилами. У найпростішому випадку таких мереж дві: «внутрішня», до якої підключені комп'ютери, і «зовнішня», що з'єднує маршрутизатор з усією глобальною мережею. Таблицю, керуючу маршрутизацією пакетів, можна переглянути за допомогою утиліти **netstat -r** або **route** (обидві утиліти мають ключ -n, що змушує їх використовувати у видачі IP-адреси, а не імена комп'ютерів).

Комп'ютер або апаратний пристрій, що здійснює маршрутизацію між локальною мережею і Internet зазвичай називається *шлюзом*.

Щоб редагувати таблицю маршрутизації використовується та ж утиліта **route**. Утиліта **ip** об'єднує можливості як по налаштуванню мережевих інтерфейсів, так і таблиці маршрутизації.

Службовий протокол ICMP

Є такі протоколи рівня IP, дія яких цим рівнем і обмежується. Наприклад, службовий протокол ICMP (Internet Control Message Protocol), призначений для передачі службових повідомлень.

Прикладом застосування ICMP є утиліта **ping**, яка дозволяє перевірити працездатність вузлів в мережі. Інше застосування ICMP – повідомляти відправнику, чому його пакет неможливо доставити адресату, або передавати інформацію про зміну маршруту, про можливість фрагментації і т. п.

Ще одне завдання, яке можна вирішити використанням протоколу ICMP, є визначення маршруту проходження пакету. У UNIX існує ряд альтернативних команд для визначення приблизного маршруту проходження пакету по мережі: **traceroute, tracepath** і т. п.

Утиліта **traceroute** показує список абонентів, через яких проходить пакет по шляху до адресата, і витрачений на це час. Однак список цей приблизний. Строго кажучи, невідомо, яким маршрутом йшла чергова група пакетів, тому що з тих пір, як була відправлена попередня група, який-небудь з проміжних маршрутизаторів міг змінити рішення і відправити нові пакети іншим шляхом.

Інформація про з'єднання

Транспортних протоколів в TCP/IP два – це TCP (Transmission Control Protocol, протокол управління з'єднанням) і UDP (User Datagram Protocol). UDP забезпечує більш високу швидкість обміну даними за рахунок простоти пристрою. Призначені для

користувача дані містяться в єдиний транспортний пакет-датаграмму, якому приписують звичайні для транспортного рівня дані: адреси і порти відправника і одержувача, після чого пакет йде в мережу шукати адресата. Перевіряти, чи був адресат здатний цей пакет прийняти, чи дійшов пакет до нього і не зіпсувався чи по дорозі, надається наступному – прикладному – рівню.

Інша справа – TCP. Цей протокол дуже піклується про те, щоб дані, що передаються дійшли до адресата в цілості й схоронності. Для цього робляться такі дії:

- встановлення з'єднання;
- обробка підтвердження коректної доставки;
- відстеження стану абонентів.

Для перегляду всіх існуючих в даний момент мережевих з'єднань можна скористатися командою **netstat.**

Згідно зі стандартами Internet для більшості протоколів прикладного рівня існують стандартні порти, на яких відповідні додатки повинні приймати з'єднання. Наприклад, веб-сервер, що виконує обробку з'єднань по протоколу HTTP, повинен працювати на порту з номером 80.

У UNIX існує прозорий механізм іменування протоколів прикладного рівня. У файлі /etc/services можна побачити список відповідності імен протоколів номерам портів. Цей файл використовується базової системної бібліотекою мережевої взаємодії, так що у всіх утиліти замість номера порту можна вказувати ім'я відповідного протоколу.