

Хмельницький національний університет

В.С. Чернишенко

М.М. Ясько

С.В. Чернишенко

**ОПЕРАЦІЙНА СИСТЕМА UNIX ТА ПРИНЦИПИ РОБОТИ
З ВІДКРИТИМ ПРОГРАМНИМ ЗАБЕЗПЕЧЕННЯМ**

Навчальний посібник

Зміст

ВСТУП.....	4
1. РОЛЬ ВІДКРИТОГО ПО У РОЗВИТКУ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ	7
1.1. Відкрите та вільне програмне забезпечення: основні визначення	7
1.2. Організаційні принципи руху відкритого та вільного програмного забезпечення	8
1.3. Переваги вільного ПЗ і його значення для розвитку ІКТ	11
1.4. Національні пріоритети України з розвитку та використання вільного ПЗ	14
2. ОПЕРАЦІЙНІ СИСТЕМИ ТИПУ UNIX	16
2.1. Історія, версії та основні характеристики ОС UNIX	16
2.1.1. Хронологія основних подій в історії ОС UNIX	16
2.1.2. Сучасні версії ОС UNIX	20
2.1.3. Архітектура ОС UNIX	21
2.2. ОС Linux	24
2.2.1. Характерні особливості GNU/Linux	25
2.2.2. Дистрибутиви Linux	28
2.2.3. Ubuntu Linux	33
2.2.4. Mandriva	35
2.3. Управління доступом	37
2.3.1. Користувачі та групи	37
2.3.2. Системні реєстраційні імена	41
2.3.3. База даних облікових записів	48
2.3.4. Отримання списку зареєстрованих користувачів	50
2.3.5. Засоби створення, зміни та видалення облікових записів користувачів	53
2.3.6. Засоби створення, зміни та видалення груп	57
3. ФАЙЛОВА СИСТЕМА ОС UNIX	59
3.1. Ієрархічні файлові системи	59
3.1.1. Поняття логічної файлової системи	59
3.1.2. Каталоги в ОС Unix	60
3.1.3. Зміна поточного каталогу	61
3.1.3. Файли в ОС UNIX	62
3.1.4. Отримання інформації про файли	63
3.1.5. Типи файлів	65
3.1.6. Визначення типу файлу	70
3.1.7. Основні команди для роботи з файлами	71
3.2. Структура і властивості файлових систем	88
3.2.1. Основні каталоги і їх призначення	88
3.2.2. Основні компоненти файлових систем UNIX	91
3.2.3. Огляд файлових систем	95
3.3. Управління файловою системою	99
3.3.1. Створення фізичної файлової системи	99
3.3.2. Монтування та демонтування фізичних файлових систем	100
3.3.3. Отримання інформації про файлові системи	103
4. УПРАВЛІННЯ ПРОЦЕСАМИ	107
4.1. Процеси в ОС UNIX	107
4.1.1. Типи процесів	107

4.1.1. Атрибути процесу	110
4.1.3. Життєвий цикл процесу в UNIX і основні системні виклики	111
4.1.4. Створення і завершення процесу.....	115
4.1.5. Управління станом процесу	119
4.2. Командний інтерпретатор.....	127
4.2.1. Структура командного рядка	128
4.2.2. Створення сценаріїв.....	133
4.2.3. Змінні і привласнення	136
4.2.4. Оператори командного інтерпретатора	139
4.2.5. Робота з сигналами та інформацією від користувача	145
4.2.6. Функції в командному інтерпретаторі	147
4.2.7. Управління завданнями.....	150
4.3. Графічні оболонки.....	153
4.3.2. X Window System.....	154
4.3.3. Клієнт-серверна модель.....	155
4.3.4. Інтерфейси користувача	156
4.3.6. Деякі інші графічні оболонки	158
5. ВІДКРИТИ СИСТЕМНІ УТИЛТИ	161
5.1. Засоби обробки тексту	161
5.1.1. Основні утиліти для обробки текстів.....	161
5.1.3. Утиліта grep. Пошук в тексті за зразком.....	164
5.1.4. Редактор vi	166
5.1.5. Робота в мережі	169
5.1.6. Резервне копіювання і відновлення.....	171
5.2. Файлові менеджери.....	172
5.3. Архіватори.....	180
6. ВІЛЬНІ ОФІСНІ СИСТЕМИ.....	183
6.1. Відкритий офісний пакет програм	183
6.1.2. Текстовий процесор Writer	183
6.1.3. Електронні таблиці Calc.....	184
6.1.4. Інструмент для створення презентацій Impress	185
6.1.5. Реляційна база даних Base	185
6.2. Робота з графікою.....	187
6.2.1. Векторний графічний редактор Draw	187
6.2.2. Редактор растрової графіки Gimp	188
6.2.3. Програми для перегляду зображень.....	190
Завдання для самостійної роботи.....	191
Контрольні питання	191
Література	192

ВСТУП

Вільне програмне забезпечення (ВПЗ) – досить цікавий феномен у розвитку сучасних інформаційно-комунікаційних технологій (ІКТ). Його популярність досить велика й поступово зростає, що пояснюється декількома обставинами. Перша причина такої популярності – це романтичне ставлення до своєї професії молодих програмістів та користувачів ПО, які, з одного боку протестують проти перетворення інформації на товар, а з другого хочуть виділитися з натовпу «звичайних» користувачів. Друга – це позиція справжніх професіоналів – розробників та системних адміністраторів, які хочуть мати змогу контролювати всі процеси, яким мають справу, що є неможливим при роботі з «закритим», комерційним кодом.

Є, нарешті, й третя причина, яка має найбільшу вагу. Це бажання серйозних користувачів, ѹ, у першу чергу, державних структур різних країн світу, забезпечити безпечність й секретність своїх даних, уникнути можливих сюрпризів, випадкових чи запланованих, з боку комерційних розробників ПО, упередити всяку можливість шантажу з боку останніх. Для відкритих ресурсів, якщо подібна загроза ѹ існує, то ѹ можна надійно уникнути, залучивши до команди кваліфікованого ІКТ-фахівця, який завжди може самостійно «залатати дірки» в системі захисту ПО, чи вилучити з коду «небезпечні» фрагменти.

Відкрите програмне забезпечення (англ.: *open source*) - це програмне забезпечення, що розповсюджується не у вигляді готових для виконання машинних кодів (які дуже важко аналізувати й модифікувати), а у вигляді текстових програм на одній з алгоритмічних мов (найчастіше – одній з версій мови Сі). Код таких програм доступний для перегляду, вивчення та змін. Крім аспекту безпечності, це дозволяє допомогти в доопрацюванні самої відкритої програми, а також використовувати код для створення нових програм і виправлення в них помилок через запозичення вихідних кодів, якщо це дозволяє ліцензія, або вивчення алгоритмів, які використовуються, структур даних, технологій, методик та інтерфейсів (оскільки вихідний код може істотно доповнювати документацію, а за відсутності такої – сам слугує документацією).

Відкрите програмне забезпечення має великі перспективи у зв'язку із прийняттям урядом рішень щодо забезпечення національної безпеки у сфері ІТ на основі

впровадження відкритого та вільного ПЗ у державні і бюджетні організації. Підготовка спеціалістів з цієї галузі набуває державного значення. Очікується започаткування спеціального державного проекту, східного з російським проектом "Пінгвін", який спрямований на впровадження технологій Лінукс и та відкритого ПО в школі и вузі й передбачає: підготовку відповідних підручників та методичного забезпечення; організацію технічної підтримки та Інтернет-порталів; навчання викладачів технологіям Лінукс та ВПЗ.

Навчальний посібник, що пропонується читачу, покликаний внести певний вклад у рішення цієї задачі. Його зміст відповідає вимогам останніх Галузевих стандартів вищої освіти України для освітньо-кваліфікаційного рівня «бакалавр» з напрямків підготовки, пов'язаних з професійною роботою з комп'ютерами. Зокрема, це стосується напрямків з галузі знань «системні науки та кібернетика»; наприклад, у стандарті напрямку «прикладна математика», посібник може бути використаний студентами при вивченні таких дисциплін як «Програмне забезпечення обчислювальних систем», «Комп'ютерні мережі» (обов'язкова програма) та «Unix-подібні операційні системи», «Операційні системи та системне програмування» (рекомендовані дисципліни).

Наголос у цьому посібнику зроблено на опису відкритих операційних систем з родини UNIX. Операційна система (ОС) — це базовий комплекс програмного забезпечення, що виконує управління апаратним забезпеченням комп'ютера або віртуальної машини; забезпечує керування обчислювальним процесом і організує взаємодію з користувачем.

Головними функціями операційної системи є:

- Виконання на вимогу програм користувача тих елементарних (низькорівневих) дій, які є спільними для більшості програмного забезпечення і часто зустрічаються майже у всіх програмах (ввід і вивід даних, запуск і зупинка інших програм, виділення та вивільнення додаткової пам'яті тощо).
- Стандартизований доступ до периферійних пристройів (пристрої введення-виведення).
- Завантаження програм у оперативну пам'ять та їх виконання.

- Керування оперативною пам'яттю (розподіл між процесами, організація віртуальної пам'яті).
- Взаємодія між процесами: обмін даними, синхронізація.
- Керування доступом до даних енергозалежних носіїв (твердий диск, оптичні диски тощо), організованим у тій чи іншій файловій системі.
- Забезпечення користувачького інтерфейсу.
- Мережеві операції, підтримка стеку мережевих протоколів.

До ОС UNIX відноситься велика кількість операційних систем, котрі можна умовно поділити на три категорії — System V, BSD та Linux. Сама назва «UNIX» (Юнікс) є торговою маркою, що належить «The Open Group», котра власне й ліцензує кожну конкретну ОС на предмет того, чи відповідає вона стандарту. Тому через ліцензійні чи інші неузгодження деякі ОС, котрі фактично є Юнікс-подібними, не визнані такими офіційно.

Системи UNIX можуть використовуватись на великій кількості процесорних архітектур. Вони популярні як серверні системи у бізнесі, як настільні системи в академічному та інженерному колах. Найбільш поширені вільні варіанти UNIX, такі як Linux та Free BSD, серед них особливо — орієнтовані на кінцевого користувача дистрибутивів Linux, в першу чергу Ubuntu, Mandriva, Red Hat Enterprise та Suse. Linux є популярною системою на стільницях розробників програмного забезпечення, системних адміністраторів та інших ІТ-спеціалістів.

Посібник призначений для початкового ознайомлення з архітектурою, особливостями і основними засобами ОС UNIX та супутніми відкритими комп'ютерними програмами. Засвоєння матеріалу дозволить читачу вільно працювати у середовищі ОС UNIX і дасть достатні базові знання для подальшого вивчення, за необхідністю, методів адміністрування або програмування цієї операційної системи.

1. РОЛЬ ВІДКРИТОГО ПО У РОЗВИТКУ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ

1.1. Відкрите та вільне програмне забезпечення: основні визначення

Відкрите програмне забезпечення (англ. *open source software*) - це програмне забезпечення з відкритим вихідним кодом. Будь-яка людина, яка добре знається на цьому, може вільно використовувати та змінювати його. «Відкрите програмне забезпечення» – це більше, ніж просто технічне визначення. Це філософія, яка базується на принципі, що програмами мають можливість користуватися, змінювати їх та модифікувати під свої завдання всі бажаючі, не сплачуючи при цьому вартості ліцензій.

Як визначається у «Вікіпедії», вихідний код відкритих програм доступний для перегляду, вивчення та зміни, що дозволяє допомогти в доопрацюванні самої відкритої програми, а також використовувати код для створення нових програм і виправлення в них помилок через запозичення вихідних кодів, якщо це дозволяє ліцензія, або вивчення алгоритмів, які використовуються, структур даних, технологій, методик та інтерфейсів (оскільки вихідний код може істотно доповнювати документацію, а за відсутності такої слугує документацією).

Термін «вільне програмне забезпечення» (англ. *free software*) визначає програмний продукт, який розповсюджується безкоштовно, і має довгу історію використання. Термін «*open source*» був запропонований у 1998 році Еріком Реймондом і Брюсом Перенсом саме як уточнення терміну «*free software*», який, на їх думку, є в англійській мові неоднозначним й «бентежить» багато комерційних підприємців.

Переважна більшість відкритих програм є одночасно вільними. Визначення відкритого і вільного ПЗ не повністю збігаються один з одним, але близькі, і більшість ліцензій відповідають обом. У той же час існують програми, які мають відкритий вихідний код, але не є вільними. Наприклад, код програми UnRAR, розпаковувальника RAR-архівів, знаходить у відкритому доступі, але ліцензія забороняє використовувати його для створення RAR-сумісних архіваторів. Взагалі, існує цілий клас програм, що називаються комерційним ПЗ с відкритим вихідним кодом, або Open Core, для яких термін «*open source*» використовується для не вільного програмного забезпечення.

Відмінність між рухами відкритого та вільного ПЗ полягає в основному в пріоритетах. Прибічники терміна «*open source*» роблять наголос на ефективності відкритих кодів як методу розроблення, модернізації та супроводу програм. Прибічники терміна «*free software*» вважають, що саме права на вільне поширення, модифікацію і вивчення програм є головним набутком вільного відкритого ПЗ.

За думкою Річарда Столлмана, розрекламованість «*open source*» певною мірою шкодить ідеї вільного ПЗ, оскільки деякі розробники відкритого ПЗ є прибічниками власницького ПЗ, й користувачі можуть зупинитися на вільному програмному забезпеченні, не доходячи до поняття про вільне ПЗ. Він відмічає, що деякі компанії, ворожі до вільного ПЗ, – наприклад, Microsoft, – використовують тільки вираз «*open source*», запобігаючи використання виразу «*free software*».

В той же час один з ініціаторів підходу, Брюс Перенс, наполягає на тому, що концепція відкритого ПЗ завжди була лише способом пояснити підприємцям ідею вільного ПЗ, і цієї мети було досягнуто.

1.2. Організаційні принципи руху відкритого та вільного програмного забезпечення

У 1985 році американський вчений Річард Столлман (*Richard Stallman*) заснував **Фонд вільного програмного забезпечення** (*Free Software Foundation*, FSF). Метою цього фонду було усунення всіх заборон та обмежень на поширення, копіювання, модифікацію та вивчення програмного забезпечення. Сайт фонду - <http://www.fsf.org> , логотип представлено на рис. 1. З часу створення й до середини 1990-х кошти Фонду використовувались в першу чергу для найму розробників для розробки вільного ПЗ. Починаючи з середини-кінця 1990-х, вільне програмне забезпечення створюється багатьма компаніями й приватними особами, тому співробітники й добровольці Фонду працюють головним чином над юридичними та організаційними питаннями у галузі вільного ПЗ.

У рамках Фонду вільного ПЗ було почато розроблення проекту GNU – проекту створення вільного програмного забезпечення. Абревіатура GNU відкривається рекурсивно – Gnu is Not Unix, тобто те, що належить проекту GNU, не є частиною

Unix (тому що на той час навіть саме слово UNIX вже було зареєстрованою товарною маркою, тобто перестало бути вільним).

Те, що ПЗ, яке розробляється в рамках проекту GNU, є вільним, не означає, що воно поширюється без ліцензії і не має ніякого статусу в юридичному сенсі. Програми, які розробляються в рамках руху Open Source, поширюються на умовах ліцензії General Public License (GPL). Програмне забезпечення, яке поширюється під цією ліцензією, можна як завгодно доопрацьовувати, модифікувати, передавати або продавати іншим особам за умови, що результат такої переробки теж поширюватиметься під ліцензією *copyleft*. Остання умова – найважливіша і визначальна в цій ліцензії. Вона гарантує, що результати зусиль розробників вільного ПЗ залишаться відкритими і не стануть частиною будь-якого ліцензованого звичайним способом продукту. Вона також відрізняє вільне ПЗ від ПЗ, яке поширюється безкоштовно. Словами творців FSF, ліцензія GPL "робить ПЗ вільним і гарантує, що воно залишиться вільним".

Іншими словами, GPL надає користувачам свободу:

- запуску програм із будь-якою метою;
- вивчення принципів роботи програм;
- модифікації програм;
- поширення копій;
- поліпшення програм і випуску нових версій у публічний доступ.

Фонд вільного програмного забезпечення має необхідні засоби й можливість контролювати виконання GNU General Public License (GPL) та інших ліцензій GNU. Фонд розглядає біля 50 випадків порушення GPL на рік й намагається примусити порушників виконувати вимоги ліцензії без судових розглядів.

Існує Видавництво Фонду (<http://www.gnupress.org>), ціллю якого є «видавництво недорогих книг з інформатики з використанням вільно поширюваних ліцензій».

FSF підтримує *Free Software Directory*, каталог вільного ПО – список програмних пакетів, які були ідентифіковані як вільне програмне забезпечення. Є можливість додавати вільне ПЗ до каталогу через веб-форму — fsf.org. Призначення каталогу — слугувати пошуковою системою з вільного ПЗ, а також джерелом інформації про те, чи була певна програма перевірена на відповідність критеріям вільного ПЗ.

Фонд надає хостінг free software-проектам через свій веб-сайт Savannah: <http://savannah.gnu.org> та <http://savannah.nongnu.org>

25 листопада 2002 р. Фонд відкрив програму Асоційованого членства (англ. *FSF Associate Membership*) для приватних осіб. В березні 2005 р. Фонд включав більш ніж 3400 асоційованих членів. 5 березня 2003 для комерційних організацій була започаткована програма Корпоративного патронажу (англ. *Corporate Patron*). На квітень 2004 р. Фонд мав 45 корпоративних членів.

Фонд вільного програмного забезпечення (*Open Source Initiative*, OSI), (<http://opensource.org/>, логотип представлений на рис. 1) – організація, що має ціллю просунення та пропонує технічну підтримку відкритого програмного забезпечення (ВПЗ).

Організацію було засновано в лютому 1998 р. Брюсом Перенсом та Еріком Реймондом, коли Netscape Communications Corporation оприлюднила вихідний код Netscape Communicator як ВПЗ через зниження прибутку й конкуренції з Microsoft Internet Explorer.

Open Source не є торговельною маркою організації Open Source Initiative. В той же час, для розробників, які бажають використовувати логотип цієї організації, існує вимога використовувати термін Open Source тільки відносно до ліцензій, що схвалені OSI. Існує спеціальний комітет, який вирішує, чи може ПЗ з певною ліцензією бути віднесено до Open Source. Критерії, якими він при цьому керується, наведено в The Open Source Definition.



Рис. 1. Логотипи FSF та OSI

Винесене визначення OSI визнається як керівництво багатьма іншими установами (наприклад, порталом Sourceforge.net). OSI накладає на публічну ліцензію десять вимог, які були розроблені на основі Debian Free Software Guidelines.

Якщо раніше ВПЗ в основному використовувалося кваліфікованими спеціалістами і не було орієнтоване на повсякденне використання звичайними користувачами, то зараз ситуація змінюється. Все більше користувачів персональних комп'ютерів переходять на ВПЗ з різних причин; і серед них часто ті, хто не має жодного уявлення про системне програмування. В принципі, це не заважає використанню сучасного відкритого ПЗ, однак знання хоча б основних принципів функціонування відкритих операційних систем та деяких ключових стандартних операцій, разом з шляхами їх виконання, можуть значно підвищити ефективність роботи користувача. Просвітницька робота в цьому напрямку є, очевидно, необхідною, й зацікавленість у ний з боку широких верств комп'ютерних користувачів буде постійно зростати.

Це, зокрема, стосується відкритих операційних систем, якими є більшість сучасних версій операційної системи UNIX, яким, головним образом, присвячений цій посібник. Перші версії Linux використовувались як серверні платформи й були орієнтовані, у першу чергу, на функціональність та надійність. З ними працювали професіонали, яких не дуже цікавила простота керування, інтуїтивна зрозумілість й різні графічні прикраси; Linux мав інтерфейс командного рядка. Коли ж він почав використовуватись також як ОС для персональних комп'ютерів, кількість його користувачів багаторазово збільшилась й з'явився інтерес до графічних оболонок і прикладних програм для Linux. У 1996 р. став розроблятись графічний інтерфейс KDE, ще через рік стартував проект GNOME. На цей час це найпопулярніші середовища для робочого стола Linux. Користувач Windows при переході на Linux сьогодні практично не побачить різниці – та ж зручність роботі, стильний дизайн, інтуїтивно зрозумілий інтерфейс, аналогічні панелі, меню, іконки. При цьому Linux дає користувачу значно більші можливості щодо налагодження системи.

1.3. Переваги вільного ПЗ і його значення для розвитку ІКТ

Відкрите ПЗ має очевидні переваги перед комерційним. Перше й найочевидніше – велика частина програм із відкритим кодом поширюється безкоштовно і, при

цьому, легально. Користувач приймає GPL – суспільну ліцензію, єдине обмеження якої полягає в тому, що будь-які зміни, здійснені користувачем у програмі, мають бути надані співтовариству на тих же вільних підставах. Усі користувачі змінених версій програм мають такі самі права на їх вивчення, модифікування та поширення.

На відміну від GPL ліцензії комерційного ПЗ дуже рідко дають користувачеві такі права і, як правило, навпаки, прагнуть їх обмежити. Розробники комерційного ПЗ оберігають секрети своїх рішень і не розкривають ні внутрішньої архітектури, ні форматів подання даних, ні інтерфейсів. При цьому найсерйознішим недоліком комерційних рішень є їх монолітність, практично всі вони поширюються у вигляді великих «в'язок», в які внесено всі необхідні для роботи компоненти. Замінити окремий компонент, якось модифікувати його неможливо, можливість інтеграції з продуктами інших компаній або заміни частини модулів на сторонні також майже ніколи не передбачена. Тим часом за допомогою стандартних програм далеко не завжди можна ефективно вирішити якісь специфічні завдання.

Відкрите ПЗ, на відміну від комерційного, будується за модульним принципом, тобто будь-яка програма збирається з окремих складових, і всі її компоненти документовані та відкриті, як і інтерфейси між ними. У розробників відкритого ПЗ немає секретів, їм не потрібно щось приховувати, щоб не втратити прибуток, тому вони не винаходять своїх закритих форматів даних або програмних інтерфейсів, а користуються тими, які є міжнародним стандартом. Це ще одна безперечна перевага Open Source перед закритим комерційним ПЗ. Можна легко інтегрувати додатки від різних розробників, замінювати окремі компоненти програм. Будь-який кваліфікований IT-фахівець може зібрати рішення, яке буде ідеально підігнано під потреби конкретної організації.

Ще одна перевага відкритого ПЗ – це вища, ніж у комерційних програм, безпека роботи. Відкритий вихідний код можна вільно проаналізувати, щоб переконатися в його коректності та відсутності не документованих можливостей – різного роду «закладок», «чорних ходів», навмисно залишених програмістом можливостей обходу захисту і тому подібного. Комп'ютери, які працюють під керуванням Linux, нечутливі практично до всіх відомих видів вірусів.

У відкритого ПЗ є й інші переваги, не менш значущі ніж ті, що перелічені вище:

- відкрите програмне забезпечення швидше тестиється та удосконалюється, ніж закрите комерційне. Швидкість реакції розробників на необхідність внесення змін відчутно вища; оновлення випускаються більш оперативно.
- завдяки доступності програмних кодів, відкриту систему можна у будь-який момент передати на підтримку власному IT-фахівцеві. Компанія, що використовує Linux, не зіткнеться з проблемою прихильності до одного підрядчика та неможливістю в найкоротші терміни усунути помилки впровадження, вона не залежатиме від політики та технічної підтримки іноземних виробників ПЗ.
- на сьогоднішній день світ продуктів Open Source включає практично всі необхідні компоненти для автоматизованого управління всіма бізнес-процесами: серверні рішення, десктопні системи, спеціалізовані бізнес-додатки.

Ми розглянули переваги відкритого ПЗ з точки зору користувача. Відмітимо ще дві критично важливі переваги, які стосуються загальних перспектив розвитку ІКТ у всесвітньому масштабі.

1. Відсутність загрози, що через якісь проблеми у фірмі-розробниці розвиток тих чи інших технологій обірветься або уповільниться.
2. Можливість для молодих фахівців, за їх власною ініціативою, прийняти участь в реальних IT проектах. Це особливо важливо, якщо взяти до уваги загальне зниження рівня теоретичної підготовки IT фахівців в університетах в останні роки, яке, у значній мірі, пов'язане з переорієнтацією вищої освіти у галузі на підготовку користувачів, а не розробників. Відкрите ПЗ дозволяє підтримати кількість спеціалістів, які вміють не лише використовувати, а й розробляти ПЗ, на більш-менш задовільному рівні.

Останній аспект є особливо важливим на національному та регіональному рівнях: навіть країни, які не мають розвиненої ІКТ індустрії, мають потребу у наявності певної кількості власних спеціалістів-розробників, наприклад, для підготовки та реалізації національних програм розвитку прикладних ІКТ з урахуванням місцевих потреб та особливостей.

Привабливі сторони відкритого ПЗ стають все більш зрозумілими для людей, які приймають політичні рішення. Більшість країн Європи мають власні програми з

використання у державних установах, розвитку та викладання у навчальних закладах відкритого програмного забезпечення.

Відповідна активність має місце останні роки у Росії. У 2008 р. уряд РФ запропонував «Концепцію розвитку розробки та використання вільного програмного забезпечення в РФ». У 2010 р. Президент Д. Мєдвєдов висловив думку, що «якщо ми повністю будемо залежити від іноземного програмного забезпечення протягом довгих років, мі від цієї залежності не позбавимося. І будемо розвиватися всупереч світовим тенденціям». Він закликав до роботи над вітчизняним пакетом відкритих програм с тим, щоб через три роки повністю перейти на нього.

У грудні 2010 прем'єр-міністр РФ В.Путін підписав «План переходу федеральних органів влади та федеральних бюджетних закладів на використання вільного програмного забезпечення», який відноситься до періоду з 2011 по 2015 р. Зокрема, у плані згадується про створення у 2012 р. національного репозиторію вільного ПЗ, який має бути не тільки «складальним» репозиторієм дистрибутиву Linux, але місцем зберігання цілої бібліотеки вільного ПЗ. Серед кандидатів на цю роль називають один з найстаріших репозиторіїв ВПЗ в Росії, Sisyphus, який підтримує фірма «Альт Лінукс», розробник однойменного дистрибутива ОС Linux.

1.4. Національні пріоритети України з розвитку та використання вільного ПЗ

Незважаючи на всі свої набутки, до недавнього часу відкрите програмне забезпечення не користувалося широкою популярністю в нашій країні. Однією з причин була та обставина, що найвідоміші комерційні програми, які на Заході є досить дорогими, у нас можна було придбати за символічні гроші, а то й скачати безкоштовно з Інтернету.

Зараз ситуація змінюється, використання неліцензійного програмного забезпечення може обернутися серйозними штрафами та конфіскацією офісної техніки. Багато установ, які не хочуть витрачати дуже суттєві кошти на придбання ліцензійної версії Windows та всіх необхідних для роботи програм, переходят на Linux. Рядових співробітників часто лякає подібна альтернатива. Вони чекають, що нова система виявиться складною в освоєнні, що відкрите ПЗ не зможе повністю замінити звичні комерційні продукти. З одного боку, як було зазначено вище, це не зовсім вірно, відкрите ПЗ стає все більш «дружнім» для користувача. З іншого боку, певна перепідготовка користувачів все ж таки може виявитися потрібною, і ця проблема досить інтенсивно обговорюється в українському суспільстві.

З іншого боку, з причин, які обговорювалися раніше, українські політики та-кож занепокоєні проблемами відкритого ПЗ. Перший проект Закону про викорис-

тання вільного ПЗ було розглянуто Верховною радою у 2005 р. У статті 2.4 цього законопроекту, зокрема, зазначалося: «Суб'єкти державного сектору для провадження всіх публічних сервісів та створення інформації, що знаходиться в публічному використанні, мають віддавати перевагу використанню Вільного програмного забезпечення або відповідних Вільних ліцензій на програмне забезпечення, визначених в статті 1.21 – 1.22 цього Закону. Використання для вказаних цілей такого ПЗ, ліцензія на яке надає користувачеві менше прав, ніж ліцензія на Відкрите (Вільне) програмне забезпечення, можливе у випадках та за процедурою, визначених статтею 6 цього Закону.» У пункті 6, де говориться про можливість використання, у деяких випадках, комерційного продукту, при цьому оговорено: «Можливості, що надаються за цим пунктом, не знімають із суб'єкта господарювання або державної установи зобов'язання дослідження можливих альтернатив у сфері Відкритого (Вільного) ПЗ, та застосування таких альтернатив у випадку їх появи.»

Занепокоєність проблемами інформаційної безпеки, які пов'язані з використанням «закритого» комерційного ПЗ, демонструє пункт 6.9: «При застосуванні іншого, ніж Вільне (Відкрите) ПЗ, відповідний суб'єкт державного сектору повинен гарантувати збереження за його допомогою суспільно-важливих, та (або) таких, що знаходяться в публічному користуванні, даних – у відкритих форматах.»

Нажаль, того часу закон не було прийнято. Наступна концепція «Програма використання в органах державної влади програмного забезпечення з відкритим кодом» даної була затверджена Кабінетом міністрів наприкінці 2009 р. Вона досить близька до попередніх відповідних документів. Срок виконання програми розрахований на 4 роки (з 2011 року по 2014 рік). Загальний бюджет даної програми повинен становити близько 45 млн. гривень (41 млн. гривень із держбюджету, ще 4 - з інших джерел).

Державний комітет зв'язку й інформатизації України підготував проект цієї програми у 2010 р. Головною метою діяльності визначена оптимізація видатків бюджетних коштів, а також вирішення проблеми використання неліцензійного програмного забезпечення в державних органах. За розрахунками чиновників, економія від переходу держапарату на відкрите ПЗ може становити 87%.

Серед очікуваних результатів програми - удосконалення нормативно-правової бази, виконання науково-дослідних робіт зі створення й використання відкритого ПЗ, створення інфраструктури для його розробки, формування й поширення, координація й державна підтримка інфраструктури використання відкритого ПЗ в органах державної влади, створення локалізованого базового комплекту дистрибутива, адаптованого для потреб органів державної влади. Це дало привід журналістам говорити про створенні «державного Лінукса».

Крім цього, Держкомзв'язку планує провести оцінку відкритого ПЗ на відповідність вимогам нормативних документів з технічного та (якщо буде потреба) криптографічного захисту інформації, створити фонд алгоритмів і ПЗ з відкритим кодом, організувати навчання й підвищення кваліфікації користувачів ПЗ з відкритим кодом в органах державної влади.

Напевно, використання ВПЗ в Україні буде розширятися й, відповідно, буде зростати потреба у спеціалістах з цієї галузі. Це робить вивчення відповідного навчального матеріалу українськими студентами важливою соціально-економічною задачею.

2. ОПЕРАЦІЙНІ СИСТЕМИ ТИПУ UNIX

2.1. Історія, версії та основні характеристики ОС UNIX

Історія ОС UNIX почалася в 1969 році в одному з підрозділів **AT&T Bell Laboratories**, коли на EBM DEC PDP-7 **Кен Томпсон** (Ken Thompson), **Денніс Річі** (Dennis Ritchie) та інші програмісти почали роботу над операційною системою, названої ними спочатку Unics (*UNiplexed Information and Computing System*). Протягом перших 10 років розвиток UNIX відбувалося, в основному, в Bell Labs. Відповідні початкові версії називалися "Version n" (Vn) і призначалися для EOM DEC PDP-11 (16-бітова) і VAX (32-бітова).

Версії Vn розроблялися групою Computer Research Group (CRG) у Bell Labs. Підтримкою займалася інша група, Unix System Group (USG). Розробкою також займалася група Programmer's WorkBench (PWB), привнесший систему управління вихідним кодом **sccs**, іменовані канали і ряд інших ідей. У 1983 році ці групи були об'єднані в одну, **Unix System Development Lab.**

2.1.1. Хронологія основних подій в історії ОС UNIX

Нижче в хронологічному порядку представлені найбільш суттєві версії і події в історії UNIX аж до березня 2004 року, а також деяка інформація про взаємозв'язки між ними:

Очікується:

- подальший розвиток гілки Linux 2. 6 і оновлення по старим гілкам ядер (2. 0, 2. 2, 2. 4);
- Solaris 10 (*вийшла*);
- послідовний розвиток FreeBSD, OpenBSD, NetBSD, інших гілок BSD.
- подальший розвиток Mac OS X і Darwin;
- подальше послідовний розвиток IRIX.

1971 V1. Перша версія UNIX Time-Sharing System на асемблері для PDP-11/20.

- Включала файлову систему, системний виклик **fork** () для породження процесів, утиліти типу **cat**, **ed**, **roff**. Використовувалася для обробки текстів при підготовці патентів. Системний виклик **pipe** () і підтримка програмних каналів з'явилися в **V2**.
- 1973** **V4.** Версія, переписана на мові C, що зробило UNIX легко переноситься на інші платформи. Мова C створювався для розробки ОС UNIX.
- 1974** **V5.** Поява перших версій в Bell Labs (PWB / UNIX, MERT).
- 1975** **V6.** Перша версія UNIX, широко поширенна за межами Bell Labs, зокрема, в університетах. З цього часу починається появу безлічі інших версій i UNIX стає популярною ОС. На базі цієї версії в Каліфорнійському університеті в Берклі (UCB) створювалася **1. xBSD** (для PDP-11). Версія **2. xBSD** (Berkeley Systems Development) для PDP-11, створена групою **Computer Systems Research Group** (CSRG) в Берклі. Підтримка мережі DARPA, перша реалізація стека протоколів TCP / IP. Командний інтерпретатор **csh**. У подальших версіях (до 1980): підтримка віртуальної пам'яті, **termcap**, **curses**, редактор **vi**.
- 1979** **V7. "Остання справжня UNIX"**, включала компілятор мови C, командний інтерпретатор **sh**, систему **uucp**, була перенесена на 32-розрядний VAX. При цьому розмір ядра становив близько **40 Кбайт!**
- 1981** **4. 1BSD:** управління завданнями, автоматичне конфігурування ядра. **System III** - перший комерційний UNIX від AT & T, реалізація *іменованих каналів* (FIFO).
- 1982** UNIX починають використовувати творці робочих станцій: **SunOS 1. 0** (на базі 4. 1BSD) від Sun Microsystems і **HP-UX** (на базі System III) від Hewlett-

Packard.

4. 2BSD: повна підтримка TCP/IP, сокетів, Ethernet. Файлова система UFS з **1983** підтримкою довгих імен файлів і символічних посилань. AT & T **System V:** підтримка основних утиліт і засобів BSD.

SVR2: функції в командному інтерпретаторі **sh**, перші спроби стандартизації. **SCO XENIX** - перший комерційний UNIX на Intel-архітектурі. Створення **1984** **Free Software Foundation** (FSF) і початок проекту GNU - створення вільно поширюваної UNIX-подібної ОС і відповідних утиліт.

1985 **V8** (модулі STREAMS). Поява архітектури мікроядра **Mach**. Реалізації стандарту SVR2: SCO XENIX SystemV/286, Interactive 386/ix. Поява ОС Minix.

4. 3BSD для VAX. **SVR3:** модулі STREAMS з V8, TLI, підтримка динамічно **1986** завантажуваних бібліотек. **V9** (доповнення з 4. 3BSD). Поява операційних систем **AIX** (IBM) і **A/UX** (Apple).

1987 **SVR3. 2:** SCO XENIX SV/386. Поява ОС **IRIX** (SVR3. 0).

4. 3BSD Tahoe - 4. 3BSD з кодами. Створення **SVR4** на базі System V, BSD та SunOS (X11, NFS, система віртуальної пам'яті, спільні бібліотеки). Додаткові: командний інтерпретатор **ksh**, ANSI C, можливості підтримки національних мов, відповідність стандартам POSIX, X / Open. Поява комп'ютера NeXT з ОС **NeXTSTEP** (4. 3BSD + Mach 2. 0).

4. 3BSD Reno: підтримка різних платформ, NFS, SLIP, Kerberos. **SUN 1990** **Solaris 1** (SunOS 4. 1. 4). Поява **OSF / 1** від Open Software Foundation: мікроядро Mach 2. 6 + SVR4, SMP, нитки, Motif GUI.

1991 **BSD Net2 (4. 3BSD Lite)** - не містить спірного коду AT & T. Поява ОС **GNU HURD**. Поява ОС **Linux** (на базі Minix). Виділення з AT & T окремого підрозділу

USL (Unix System Laboratories), що володіє кодом AT & T UNIX і System V.

- 4. 4BSD:** віртуальна пам'ять як в Mach 2. 5, журналізируема файлова система UFS. Закриття CSRG в Берклі. **SVR4. 2:** журналізируема файлова система **Veritas FS**, списки контролю доступу ACL, динамічно завантажувані модулі ядра. **USL UnixWare 1** - реалізація SVR4. 2. **SunOS 5 = Solaris 2** (SunOS 4 + SVR4).
- 1993** Поява OC **FreeBSD**. Solaris 2. 2. NeXTSTEP 3. 2. IRIX 5. 3, HP-UX 9. 04, AIX 4. 0, Linux 0. 99, UnixWare 1. 1.
- 1994** OSF 1.3: мікроядро Mach 3, підтримка **64-бітових платформ**. FreeBSD 2. 0, SCO OpenDesktop 3.2.4. **UnixWare 1.1.2**. Linux 1.0.9. USL куплена Novell.
- 1995** Поява **OpenBSD** і **NetBSD**. Solaris 2. 5. Поява **Digital UNIX** (DEC OSF / 1). Поява SCO OpenServer 5. 0. **UnixWare 2. 0:** SVR4. 2 MP від Novell. Novell продає UnixWare і весь вихідний код AT & T компанії **SCO**. Вихід HP-UX 10 (з додатками з UnixWare).
- 1996** FreeBSD 2. 1. 6. OpenBSD 2. 0. IRIX 6. 3. Linux 2. 0. 21. OpenSTEP 4 - завершення проекту NeXTSTEP. SCO UnixWare 2. 1. Мікроядро Mach 4.
- 1997** FreeBSD 2. 2. 5, OpenBSD 2. 2, NetBSD 1. 3, Solaris 2. 6 (під SPARC і Intel), SCO OpenServer 5. 0. 4. IRIX 6. 4. GNU Hurd 0. 2 (+ Mach 4). Linux 2. 0. 28.
- 1998** FreeBSD 3. 0 (+4. 4 BSD), Solaris 7, DigitalUNIX 4, SCO: OpenServer 5. 0. 5, **UnixWare 7 (SVR5)**. HP-UX 11. 0. **Linux 2. 0. 36**. IBM: проект Monterey (AIX 4. 3 + SVR5).
- 1999** FreeBSD 3. 4. OpenBSD 2. 6, NetBSD 1. 4. Поява **Mac OS X** і проекту **Darwin** (Mach 4 + FreeBSD 3. 1). Solaris 8 beta. Компанію DEC купив Compaq: **Tru64 Unix V. 5. 0.** (DigitalUNIX). IRIX 6. 5. 6. SCO: OpenServer 5.

	0. 5a, UnixWare 7. 1. 1. AIX 4. 3. 3. Linux 2. 2. 13.
2000	FreeBSD 4. 0-4. 2. OpenBSD 2. 8. NetBSD 1. 5. Solaris 8 . Apple: Mac OS X Server, Darwin 1. 2. 1. Tru64 Unix V. 5. 1. IRIX 6. 5. 10. SCO: OpenServer 5. 0. 6. Компанія SCO продала всі свої ОС компанії Caldera (Caldera OpenLinux). Hurd A1, Linux 2. 4. 0 , 2. 2. 18. HP-UX 11i. AIX 5L alpha (проект Monterey).
2001	FreeBSD 4. 4. OpenBSD 3. 0. NetBSD 1. 5. 2. Mac OS X 10. 1. 2. SUN: Solaris 8 10/01, Solaris 9 alpha (не для Intel-архітектури). Tru64 Unix V. 5. 1A, IRIX 6. 5. 13. SCO OpenServer 5. 0. 6a. Hurd h3. Caldera: OpenUNIX 8 : UnixWare 7. 1. 1 + LKP = Linux 2. 4. 0 - прозора підтримка Linux-додатків. Linux 2. 4. 17 , 2. 0. 39, 2. 2. 20. AIX 5L v. 5. 1.
2002	FreeBSD 4. 7. OpenBSD 3. 2. NetBSD 1. 6. Mac OS X 10. 2 (Jaguar). Darwin 6. 2. SUN: Solaris 8 12/02, Solaris 9 OE. QNX 6. 2. IRIX 6. 5. 18. Debian GNU / Hurd J2. SCO: UnixWare 7. 1. 3 (протягом OpenUNIX 8. Caldera знову стала SCO . . .). Linux (вплив IRIX і AIX, зокрема, файлові системи): 2. 5. 2-2. 5. 52 - експериментальне ядро. 2. 4. 20 , 2. 2. 23. HP-UX 11i v1. 6. AIX 5L v. 5. 2.
2003	FreeBSD 5. 1 , FreeBSD 4. 9. OpenBSD 3. 4 . NetBSD 1. 6. 1. Mac OS X 10. 3. 2. Darwin 6. 6-7. 2. SUN: Solaris 9 OE 12/03, Solaris 9 x86 PE, Solaris 10 Preview. QNX 6. 2. 1. IRIX 6. 5. 22. Debian GNU / Hurd K5. Tru64 Unix V5. 1B-1. SCO: UnixWare 7. 1. 3/OKP (OpenServer Kernel Personality), OpenServer 5. 0. 7. Linux: 2. 6. 0 , 2. 4. 23 , 2. 2. 25. HP-UX 11i v2 (в тому числі, для Intel Itanium).
2004	FreeBSD 5. 2. NetBSD 1. 6. 2. Solaris 10 Software Express 02/04. IRIX 6. 5. 23. Linux: 2. 6. 3 , 2. 4. 25, 2. 2. 26, 2. 0. 40.

2.1.2. Сучасні версії ОС UNIX

Отже, на початок 2004 року на платформі Intel x86 представлена такі основні версії UNIX:

- FreeBSD 5.2;

- OpenBSD 3.4;
- NetBSD 1.6.2;
- Linux 2.0, 2.2, 2.4, 2.6 у вигляді безлічі різних дистрибутивів;
- Solaris 9;
- SCO OpenServer 5.0.7 і UnixWare 7.1.3

На інших платформах (основні версії):

- Linux 2.6.x (практично всі платформи);
- NetBSD 1.6.2 (практично всі платформи);
- Mac OS X 10.3.2 (PowerPC);
- AIX 5L v5.2 (PowerPC);
- Solaris 9, 10 (SPARC);
- HP-UX 11i (PA-RISC, Intel Itanium);
- Tru64 Unix V.5.1B-1 (Alpha);
- IRIX 6.5.23 (MIPS)

ОС UNIX має такі основні характеристики:

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

2.1.3. Архітектура ОС UNIX

У будь-якій операційній системі можна виділити 4 основних частини:

- ядро;

- файлову структуру;
- інтерпретатор команд користувача;
- утиліти.

Архітектура ОС UNIX - багаторівнева. На нижньому рівні, безпосередньо над устаткуванням, працює *ядро* операційної системи. Функції ядра доступні через *інтерфейс системних викликів*, що утворюють другий рівень. На наступному рівні працюють *командні інтерпретатори*, команди й утиліти системного адміністрування, комунікаційні *драйвери* та *протоколи*, - все те, що зазвичай відносять до *системного програмного забезпечення*. Нарешті, зовнішній рівень утворюють *прикладні програми* користувача, мережні та інші комунікаційні служби, СУБД і утиліти.

Основні функції ядра UNIX (яке може бути *монолітним* або *модульним*) включають:

- планування і перемикання процесів;
- управління пам'яттю;
- обробку переривань;
- низькорівневу підтримку пристрій (через драйвери);
- керування дисками й буферизація даних;
- синхронізацію процесів і забезпечення коштів між процесами взаємодії (IPC).

Системні виклики забезпечують:

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

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

Системні виклики забезпечують програмний інтерфейс для доступу до процедур ядра. Вони забезпечують управління системними ресурсами, такими як пам'ять, простір на дисках і периферійні пристрой. Системні виклики оформлені у вигляді бібліотеки. Багато системні виклики доступні через командний інтерпретатор.

Процеси користувача, як правило, мають такі характеристики:

- захищенні від інших користувальницьких процесів;
- не мають доступу до процедур ядра, окрім як через системні виклики;
- не можуть безпосередньо звертатися до простору пам'яті ядра.

Простір (пам'яті) ядра - це область пам'яті, в якій процеси ядра (процеси, що працюють у контексті ядра) реалізують служби ядра. Будь-який процес, що виконується в просторі ядра, вважається працюють у режимі ядра. Простір ядра - привілеїйована область; користувач отримує доступ до неї тільки через інтерфейс системних викликів. Для користувача процес не має прямого доступу до всіх інструкцій і фізичних пристройів, - їх має процес ядра. Процес ядра також може змінювати карту пам'яті, що необхідно для *перемикання процесів* (zmіни контексту).

Для користувача процес працює в режимі ядра, коли починає виконувати код ядра через системний виклик.

Оскільки для користувача процеси і ядро не мають загального адресного простору пам'яті, необхідний механізм передачі даних між ними. При виконанні системного виклику, аргументи виклику і відповідний ідентифікатор процедури ядра передаються з користувацького простору в простір ядра. Ідентифікатор процедури ядра передається або через апаратний реєстр процесора, або через стек. Аргументи системного виклику передаються через область клієнтів викликає процесу.

Користувацька область процесу містить інформацію про процес, необхідну ядру:

- кореневий і поточний каталоги, аргументи поточного системного виклику, розміри сегмента тексту, даних і стека для процесу;
- покажчик на запис в таблиці процесів, що містить інформацію для планувальника, наприклад, пріоритет;
- таблицю дескрипторів файлів користувача процесу з інформацією про відкриті файлах;
- стек ядра для процесу (порожній, якщо процес працює в режимі користувача).

Процес користувача не може звертатися до простору ядра, але ядро може звертатися до простору процесу.

ОС UNIX забезпечує ряд стандартних системних програм для вирішення завдань адміністрування та підтримки файлової системи, зокрема:

- для настройки параметрів конфігурації системи;
- для перекомпонування ядра (якщо вона необхідна) і додавання нових драйверів пристрій;
- для створення та вилучення облікових записів користувачів;
- створення і підключення фізичних файлових систем;
- установки параметрів контролю доступу до файлів.

Для вирішення цих завдань системне ПЗ (працює в режимі користувача) часто використовує *системні виклики*.

2.2. ОС Linux

Linux - це розрахована на багато користувачів мережева операційна система з мережевою віконною графічною системою X Window System. ОС Linux підтримує стандарти відкритих систем та протоколи мережі Internet і сумісна з системами Unix, DOS, MS Windows. Всі компоненти системи, з вихідними текстами включно, поширюються з ліцензією на вільне копіювання та установку для необмеженої кількості користувачів.

ОС Linux широко пошиrena на платформах Intel PC 386/486/Pentium/Pentium Pro та завойовує позиції на інших платформах (DEC AXP, Power Macintosh і ін.).

Розробка ОС Linux виконана Лінусом Торвалдсом (Linus Torvalds) з університету Гельсінкі та не підлягаючи підрахунку поширилося командою з тисяч користувачів мережі Internet, співробітників дослідницьких центрів, фондів, університетів і т. і.

Внаслідок того, що вихідні коди Linux поширюються вільно і загальнодоступні, до розвитку системи з самого початку підключилася велика кількість незалежних розробників. Завдяки цьому на сьогоднішній момент Linux - найсучасніша, найстійкіша система, яка швидко розвивається та майже миттєво вбирає в себе найостанніші технологічні новини. Вона володіє всіма можливостями, які властиві сучасним

повнофункціональним операційним системам типу UNIX. Наведемо короткий список цих можливостей.

2.2.1. Характерні особливості Gnu/Linux

Gnu/Linux - загальна назва Unix-подібних операційних систем на основі одноіменного ядра та зібраних для нього бібліотек і системних програм, розроблених в рамках проекту GNU. ОС Linux може вільно встановлюватися та використовуватися на персональних комп'ютерах, серверах і суперкомп'ютерах разом з пакетами вільного відкритого програмного забезпечення.

На відміну від більшості інших операційних систем, Gnu/Linux не має єдиної «офіційної» комплектації. Замість цього Gnu/Linux поставляється у великій кількості так званих дистрибутивів, в яких програми GNU з'єднуються з ядром Linux та іншими програмами.

Найбільш відомими дистрибутивами Gnu/Linux є Slackware, Debian Gnu/Linux, Red Hat, Fedora, Mandriva, SUSE, Gentoo, Ubuntu. З дистрибутивів російських розробників найбільш відомі ALT Linux і AspLinux.

Більше 60% серверів в світі використовують операційну систему Linux, що характеризується високим рівнем надійності й безпеки в експлуатації. Десятки тисяч комп'ютерних вірусів для Windows не можуть поширюватися на комп'ютерах з операційною системою Linux.

У Linux є кореневий каталог («/»), в якому лежать директорії з різними назвами і призначеннями. Наприклад, каталог /home/імя_користувача слугує для зберігання призначених для користувача даних та більшості налаштувань для програм, які запускаються. Причому дані з цього каталогу можуть знаходитися на іншому фізичному носієві, або іншому розділі жорсткого диску.

Жорсткі диски, лазерні приводи та знімні носії (і всі інші пристрої) представлені в Gnu/Linux у вигляді спеціальних файлів, розташованих в директорії /dev. Назва ide-пристрой починається з латинських букв hd, третій символ визначається залежно від того, на якому ide-каналі розташований пристрій, а четвертий залежить від таблиці розділів жорсткого диска. Якщо на диску «Primary Master» (перший диск на першому ide-каналі) є два розділи, назва першого буде hda1, а другого - hda2. Якщо

є привід «Secondary slave» (другий пристрій на другому ide-каналі), його назва буде hdd. Аналогічно для Scsi/sata-пристрійв: їх назви матимуть вигляд sda1, sda2, sdb і так далі.

Будь-який прихованний файл або тека починаються з крапки, так наприклад, призначені для користувача налаштування, розташовані в директорії користувача, починаються з крапки і файлові менеджери за умовчанням їх не відображують. У Gnu/Linux немає реєстру, системні налаштування зберігаються в текстових файлах, переважно розташованих в директорії /etc.

Взагалі, про ОС Linux можна сказати, що вона:

- дає можливість безкоштовно і легально мати сучасну ОС;
- характеризується високою швидкодією;
- працює надійно, стійко, без зависань;
- не чутлива до вірусів;
- дозволяє використовувати повністю можливості сучасних ПК, знімаючи обмеження, властиві DOS і MS Windows з використання пам'яті машини і ресурсів процесора;
- ефективно управляє багатозадачністю і пріоритетами, фонові завдання (тривалий розрахунок, передача електронної пошти по модему, форматування дискети і тому подібне) не заважають інтерактивній роботі;
- дозволяє легко інтегрувати комп'ютер в локальні й глобальні мережі, в т.ч. в Internet; працює з мережами на базі Novell і MS Windows;
- дозволяє виконувати представлені у форматі завантаження прикладні програми з інших ОС - різних версій Unix, DOS і MS Windows;
- забезпечує використання величезного числа програмних пакетів, накопичених за роки існування Unix і вільно поширюваних разом з вихідними текстами;
- надає багатий набір інструментальних засобів для розробки прикладних програм будь-якого рівня складності, включаючи системи класу клієнт-сервер, об'єктно-орієнтовані, з багатовіконним текстовим й графічним інтерфейсом, придатних для роботи як в Linux, так і в інших ОС;
- включає добірну документацію та вихідні текстів всіх компонент, в тому числі, ядра самої ОС;

- надає всім бажаючим можливість спілкування й спільної роботи через Internet з будь-якими з розробників ОС Linux, та внести свій вклад, ставши, таким чином, співавтором системи.

Linux – це повноцінна 32-х розрядна (64-х розрядна на платформі x64) операційна система, яка дозволяє ефективно використовувати комп'ютер. Linux перетворює персональний комп'ютер IBM PC на справжню робочу станцію.

Ще більш суттєвою, ніж на устаткуванні, є економія на програмному забезпеченні, оскільки Linux поставляється з вільною ліцензією, що означає безкоштовне необмежене копіювання системи. Ядро, редактори, транслятори, СУБД, мережева підтримка, графічні інтерфейси, ігри і маса іншого програмного забезпечення, об'ємом в тисячі мегабайт, можуть отримуватися безкоштовно на законній основі.

Для багатьох користувачів в Україні вільна ліцензія - це єдина можливість легально забезпечити себе повноцінним набором програмного забезпечення.

До характерних особливостей Linux як ОС можна віднести:

- багатозадачність: декілька програм можуть виконуватись одночасно;
- розрахований на багато користувачів режим: декілька користувачів можуть одночасно працювати на одній машині;
- захист пам'яті процесу; збій програми не може викликати зависання системи;
- економне завантаження: Linux прочитує з диска лише ті частини програми, які дійсно необхідні для виконання;
- розділення сторінок запису між екземплярами виконуваної програми; тобто процеси-екземпляри програми можуть використовувати при виконанні одну і ту ж пам'ять;
- віртуальна пам'ять має сторінкову організацію (тобто на диск з пам'яті витісняється не весь неактивний процес, а окрема сторінка); вона створюється в самостійних розділах диска і файлах і має об'єм до 2 Гбайт; ъє розмір може змінюватись під час виконання програм;
- спільна пам'ять програм і дискового кешу (вся вільна пам'ять використовується для буферизації обміну з диском);
- динамічно завантажуванні бібліотеки;
- сертифікація за стандартом Posix.1, сумісність із стандартами System V і BSD;

- управління завданнями в стандарті POSIX;
- емуляція співпроцесора в ядрі, тому додаток може не піклуватися про емуляцію співпроцесора. Звичайно, якщо співпроцесор в наявності, то він і використовується;
- підтримка національних алфавітів і угод, в т.ч. для української мови; можливість додавати нові;
- множинні віртуальні консолі: на одному дисплеї декілька одночасних незалежних сесій роботи, які перемикаються з клавіатури;
- підтримка ряду поширених файлових систем (MINIX, Xenix, файлові системи System V); наявність власної передової файлової системи ємністю до 4 Терабайт і з іменами файлів до 255 знаків;
- прозорий доступ до розділів DOS (або OS/2 FAT): розділ DOS виглядає як частина файлової системи Linux; підтримка VFAT (WNT, Windows 95);
- спеціальна файлова система UMSDOS, яка дозволяє встановлювати Linux у файлову систему DOS;
- доступ (тільки читання) до файлової системи HPFS-2 OS/2 2.1;
- підтримка всіх стандартних форматів CD ROM;
- підтримка мережі TCP/IP, включаючи ftp, telnet, NFS і т.д.

Користувач дістає доступ до системи в результаті реєстрації. На консоль система виводить запрошення `login:`, а користувач у відповідь вводить своє ім'я і пароль доступу. Після перевірки прав доступу система видає запрошення до вводу команди на виконання програм, приймає команди і виконує їх.

ОС Linux документована так повно, як не одна з комерційних ОС. По Linux є книги, довідники і те, що недоступно в комерційній системі - повні вихідні тексти ОС з поясненнями й коментарями. Об'єм документації з Linux безперервно поповнюється й розширюється, динамічно відстежуючи розвиток системи.

2.2.2. Дистрибутиви Linux

У будь-якій операційній системі можна виділити 4 основних частині:

- ядро,
- файлову структуру,

- інтерпретатор команд користувача,
- утиліти.

Ядро - це основна, визначаюча частина ОС, яка керує апаратними засобами і виконанням програм. Файрова структура - це система зберігання файлів на зовнішніх пристроях. Інтерпретатор команд або оболонка - це програма, яка організує взаємодію користувача з комп'ютером. І, нарешті, утиліти - це просто окремі програми, які, взагалі кажучи, нічим принципово не відрізняються від інших програм, які запускаються користувачем, хіба лише своїм основним призначенням - вони виконують службові функції.

Як вже говорилося вище, якщо бути точним, то слово "Linux" позначає лише ядро. Тому, коли йдеться про операційній системі, правильніше було б говорити "операційна система, заснована на ядрі Linux". Ядро ОС Linux розробляється під загальним керівництвом Лінуса Торвальдса і поширюється вільно (на основі ліцензії GPL), як і величезна кількість іншого програмного забезпечення, утиліт і прикладних програм. Одним з наслідків вільного поширення ПЗ для Linux стало те, що велика кількість різних фірм і компаній, а також просто незалежних груп розробників стали випускати так звані дистрибутиви Linux.

Дистрибутив - це набір програмного забезпечення, що включає всі 4 основних складових частини ОС, тобто ядро, файлову систему, оболонку і сукупність утиліт, а також деяку сукупність прикладних програм. Зазвичай всі програми, що включаються в дистрибутив Linux, поширюються на умовах GPL, так що може скластися враження, що дистрибутив може випустити хто завгодно, точніше будь-хто, хто не полінується зібрати колекцію вільного ПЗ. І якась міра правди в такому твердженні є. Проте розробник дистрибутиву повинен принаймні створити програму інсталяції, яка встановлюватиме ОС на комп'ютер, на якому жодної ОС ще немає. Крім того, необхідно забезпечити дозвіл взаємозалежності й і протиріч між різними пакетами (і версіями пакетів), що теж є нетривіальним завданням.

Кожен дистрибутив характеризується такими складовими:

- Програма-завантажувач, призначена для ініціалізації апаратної частини, завантаження (зазвичай) урізаної версії системи, ініціалізації носіїв, тощо.

- Програма для вибору графічного середовища та налаштування локалізації, потрібних сервісів.
- Програма початкової конфігурації.
- Програма управління пакетами, що дає можливість встановити пакети на працючу систему, оновити пакети і таке інше
- Набори пакетів, необхідних користувачеві, спеціалізованість дистрибутиву (загального призначення, рятувальні, «живі», мікро і таке інше, а також орієнтованість на вирішення конкретних завдань — кластерні дистрибутиви, дистрибутиви для специфічних галузей науки та інше.)
- Розробник (технічні, адміністративні, фінансові та інші рішення, покладені в основу дистрибутиву, наявність підтримки користувачів).
- Співтовариство (компетентність користувачів, взаємодопомога, обмін досвідом серед користувачів).
- Інші характеристики, такі як простота встановлення та безпека, легкість налаштування, довжина життя, стабільність розвитку дистрибутиву, платність дистрибутиву та інше.

На даний момент існує більше 1000 дистрибутивів Linux, сайт Distrowatch.com пропонує статистику по 349 дистрибутивам, і це серйозно ускладнює вибір. Для початкового користувача можна порадити такі дистрибутиви, як Debian, Slackware, Ubuntu, Mandriva, OpenSUSE, Fedora. Розглянемо декілька з них. В даний час найбільш популярними є дистрибутиви Ubuntu та Mandriva, яким будуть присвячені окремі параграфи.

Slackware

- Один із найстаріших дистрибутивів GNU/Linux
- Використовує принцип KISS (Keep it simple, stupid)
- Механізм управління пакетами – утиліта pkgtools

Назва Slackware походить від слова Slack, що означає ледачий, недбайливий, розхитаний, розслаблений, млявий, а також ледарювати.

Debian

- GNU/Hurd, GNU/NetBSD і GNU/FreeBSD
- Найбільш завершений і найбільш використовуваний дистрибутив Debian - GNU/Linux
- Має декілька гілок зберігання ПЗ
- Підтримує велику кількість апаратних платформ

Гілки збереження ПЗ розподіляються по трьом гілкам:

- стабільну (stable), таку, яка містить пакети, що увійшли до останнього офіційного дистрибутиву (оновлення пакетів у ньому відбувається лише для усунення вразливостей);
- тестовану (testing), з якої буде формуватися наступний стабільний дистрибутив;
- нестабільну (unstable), в якій пакети готовуються до розміщення в тестовану гілку.

Debian — найбільш суворий із всіх дистрибутивів у відношенні ліцензій програм. Має найбільше сховище пакетів - готових до використання програм, - і якщо навіть не за їх кількістю, то за кількістю архітектур, які підтримуються: починаючи з ARM, яка використовується у вбудованих пристроях, найбільш популярних x86 та PowerPC, нових 64-розрядних AMD і закінчуючи IBM S/390, яка використовується в мейнфреймах.

Gentoo

- Портируемість та оптимізація під конкретну машину
- Найшвидший пінгвін :)
- Механізм управління пакетами – утиліта portage

Gentoo (виголошується «дженту») – це англійська назва виду пінгвінів.

Хоча основний мотив використання Gentoo Linux — збільшення швидкості роботи системи за рахунок оптимізації під конкретну машину, приріст продуктивності вагоміший на сучасних комп'ютерах. Компіляція системи на старому процесорі може зайняти до тижня чистого машинного часу. Це має просте пояснення: чим старі-

ше процесор — тим більче його набір інструкцій до стандартного мінімального, а виграш у продуктивності досягається в основному за рахунок використання потужніших інструкцій нових процесорів. Для старих процесорів може бути розумно встановлювати кроскомпільовані пакети, тобто бінарні пакети, які заздалегідь відкомпілювалися на швидкій машині.

Значно більше Gentoo Linux пристосована для максимального використання можливостей новітніх процесорів, для встановлення рекордів продуктивності.

Red Hat Linux

- Система загального призначення
- Розробки велися до 2003 року
- Остання версія - Red Hat Linux 9
- Механізм управління пакетами – утиліта RPM

Компанія почала свою роботу в 1993 році, і на даний момент налічує більш як 700 співробітників та 27 підрозділів по всьому світу; одна з найбільших компаній, яка випускає Linux. Головний офіс компанії знаходиться в місті Ролі (Raleigh, North Carolina), штат Каліфорнія, США.

До 2002 року основним продуктом Red Hat була операційна система загального призначення Red Hat Linux, в травні 2002 року відбувся випуск корпоративної операційної системи Red Hat Linux Advanced Server 2.1 (пізніше перейменована в Red Hat Enterprise Linux AS 2.1), розробленої на основі Red Hat Linux 7.2 в рамках окремого проекту.

Fedora Core

- Побудова системи повністю з вільного програмного забезпечення
- Проект спонсорується Red Hat Inc. та підтримується громадськістю
- Використовується як майданчик для створення корпоративних дистрибутивів

У 2003 році Red Hat змінила політику випуску дистрибутивів, відмовившись від випуску версій коробочок Red Hat Linux (остання версія коробочки Red Hat Linux 9)

і перетворивши внутрішній процес розробки Red Hat Linux на відкритий проект Fedora (фетровий капелюх — англ.), що не забезпечується офіційною підтримкою, але підтримуваний співтовариством розробників і експертів по Linux, найбільш активну частину якого складають співробітники Red Hat.

В результаті корпоративне рішення називається Red Hat Enterprise Linux, а вільно поширюваний відкритий дистрибутив — Fedora Core. Проект Fedora задуманий компанією, як тестовий майданчик для нових технологій і компонентів системи, які пізніше можуть бути використані в корпоративних дистрибутивах. Деякі члени співтовариства вважають подібну практику хибною і засуджують політику Red Hat.

SUSE (Gesellschaft für Software und Systementwicklung)

- Основні розробки орієнтовані на настільні комп'ютери
- Зручний інсталятор і система управління пакетами та налаштування YAST
- Novell Linux Desktop (NLD) - корпоративна версія

Про ці дистрибутиви було написано немало позитивних оглядів за їх інсталятор та систему налаштування YAST, розроблені програмістами SUSE.

Документація, яка йде з коробочними версіями, постійно відзначається як якнайповніша, детальніша та найзручніша. Дистрибутив отримав значну частку ринку в Європі та Північній Америці, але не продається в Азії та інших частинах світу.

З купівлєю SUSE фірмою Novell в кінці 2003 року почалося цілеспрямоване завоювання корпоративного ринку. На основі дистрибутиву SUSE випускається дистрибутив для корпоративних клієнтів Novell Linux Desktop (NLD).

Переваги: професійна увага до деталей, легка для користувачів система налаштування YAST.

Недоліки: бажано редагувати конфігураційні файли лише через yast2, але не завжди це можливо.

Управління пакетами: YAST (RPM), доступні third-party APT (RPM) репозиторії.

2.2.3. Ubuntu Linux

Чому користувачі обирають Ubuntu? Тому що:

- цей дистрибутив простий і продуманий для непідготовленого користувача;
- він дуже легко встановлюється;
- в ньому неважко маніпулювати програмним забезпеченням і зручно тримати систему актуальною;
- у нього великий репозиторій;
- він не позбавлений можливості редагування конфігураційних файлів уручну;
- і, мабуть, найкоштовніше - це величезне, досить доброзичливе і активне співтовариство користувачів.

Ubuntu - досить широке поняття, яке включає як самий дистрибутив, так і його модифікації. У сімействі Ubuntu існує декілька різновидів. Їх відмінності обґрунтовані політикою, спрямованістю і встановленим за умовчанням WM/DE. На сьогоднішній день існує сім офіційних редакцій.

На основі робочого середовища GNOME існують чотири дистрибутиви:

- «звичайний» Ubuntu (ubuntu.com),
- націлений на роботу з мультимедійними застосуваннями Ubuntu Studio (ubuntustudio.org),
- Gobuntu (<http://www.ubuntu.com/products/whatisubuntu/gobuntu>), який містить лише вільне ПЗ
- створений для навчання Edubuntu (edubuntu.org).

Є спеціальний серверний дистрибутив, який не містить графічних застосувань:

- Ubuntu Server (<http://www.ubuntu.com/products/whatisubuntu/serveredition>).

І ще два дистрибутиви, які включають різні оточення робочого столу:

- заснований на легкому середовищі робочого столу Xfce Xubuntu (xubuntu.org);
- заснований на KDE Kubuntu (kubuntu.org).

Окрім цих семи, існує величезна кількість інших, заснованих на Ubuntu, наприклад християнська, мусульманська, католицька, мінімалістська версії, а також без-

ліч адаптованих під різні мови. Список деяких, найбільш відомих можна поглянути, наприклад, тут: <http://tinman321.blogspot.com/2008/08/ubuntu-based-distros.html>

Розрізнюються також й варіанти постачання: окрім CD/DVD з режимом LIVECD (у назві образу ISO з такою редакцією вказується «desktop») є і звичайний режим текстової установки (образ містить назву «alternate»). Актуальна на даний момент версія Ubuntu, випущена 2 липня 2008 року:

- 8.04.1, вона має статус LTS (Long Time Support), що означає, що підтримка цього дистрибутива
- випуск оновлень і виправлень безпеки здійснюється не 18 місяців, як завжди, а 36.

Існує декілька способів отримати настановний диск Ubuntu:

- Викачати будь-який iso-образ дистрибутиву для запису можна з офіційного сайту. Доступне завантаження за протоколами FTP, HTTP і BitTorrent: <http://cdimage.ubuntu.com/>
- Замовити безкоштовні диски звичайною поштою можна лише на офіційному сайті: <http://shipit.ubuntu.com/>. Зазвичай посилка доставляється протягом 3-4 тижнів.
- Купити диск з Ubuntu можна в будь-якому Інтернет-магазині.
- Узяти у знайомих або розповсюджувачів в своєму місті.

2.2.4. *Mandriva*

Основна редакція носить назву Free, тому що вона містить лише вільне (відкрите) програмне забезпечення в тому сенсі, в якому це розуміє Фонд вільного програмного забезпечення (Free Software Foundation (FSF)) і Ініціатива відкритого вихідного коду (Open Source Initiative (OSI)). Через відсутність пропрієтарних драйверів Free може здатися новачкам складнішою у використанні, чим редакції One або Powerpack. Користувачі Free можуть встановити не вільні драйвери побудувавши репозиторії і включивши не вільні джерела. Mandriva Linux Free є повноцінною редакцією і включає велику кількість всіляких програм для різних сфер застосування: пакет офісних програм, програми для роботи в Інтернеті, мультимедіа і так далі. Ця редакція ідеальна для людей, які бажають використовувати лише вільне програмне забезпечення на своїх комп'ютерах. Редакція Free доступна на DVD для 32-х (i586) і 64-х (x86_64) розрядних процесорів.

Mandriva 2010

- нова стабільна версія дистрибутиву, відомого своєю доброзичливістю до недовірчого користувача. Mandriva містить все необхідне для продуктивної роботи з офісними документами, виходу в Інтернет, відтворення аудіо і відео та розваг. Mandriva славиться своїм центром управління
- пакет утиліт для всеосяжного налаштування системи: у Mandriva ви зможете на будувати супутниковий Інтернет або батьківський контроль (заборонивши або вирішивши відвідини певних веб-узлів), не удаючись до ручного редагування конфігураційних файлів в командному рядку

При цьому графічні конфігуратори ніяк не обмежують просунутих користувачів, адже це Linux. Але якщо у вас немає особливого бажання лізти в командний рядок, в Mandriva ви з легкістю зможете цього уникнути.

- розробники поклопоталися про те, щоб «все просто працювало», тому користуватися всією красою Linux можна відразу після встановлення

Як завжди, до складу Mandriva включено найостанніше стабільне ядро з підтримкою нового устаткування, KMS для відеокарт Intel і фреймворком tomo, який замінив Apparmor. У х-сервері тепер реалізовані Dri2 і UXA, що добре відіб'ється на продуктивності і стабільноті роботи на деяких відеоадаптерах. У робоче оточення KDE тепер включена реалізація семантичного десктопа Nepomuk, а також майстер міграції з Kde3 (це середовище більше не підтримується, для нової версії Mandriva пакети з KDE3 не збираються).

Mandriva робить ще один крок назустріч мобільним пристроям: тепер можливе встановлення повноцінного оточення Moblin. Користувачів Gnome теж не обіздили: остання стабільна версія їх улюбленого середовища включена в дистрибутив. Mandriva 2010 містить три технології віртуалізації: KMS, XEN і Virtualbox, причому останній тепер підтримує роботу додатків OpenGL 2.0 і Direct3d 8/9. Openoffice в Mandriva збирається з гілки GO-OO, а значить включає всі останні напрацювання проекту і покращену підтримку форматів Microsoft Office.

Основні компоненти:

- ядро 2.6.31, glibc 2.10, GCC 4.4.1

- X.Org 7.5, KDE 4.3.2., Gnome 2.28
- OpenOffice.org 3.1.1, Mozilla Firefox 3.5
- Qt 4.5.3, QtCreator 1.2, Tcl/Tk 8.6

2.3. Управління доступом

2.3.1. Користувачі та групи

UNIX - багатокористувацька операційна система. *Користувачі*, що займаються загальними завданнями, можуть об'єднуватися в *групи*. Кожен користувач обов'язково належить до однієї або декількох груп. Всі команди виконуються від імені певного користувача, що належить в момент виконання до певної групи.

У багатокористувацьких системах необхідно забезпечувати захист об'єктів (файлів, процесів), що належать одному користувачеві, від всіх інших. ОС UNIX пропонує базові засоби захисту та обміну файлами на основі відстеження користувача і групи, що володіють файлом, трьох *рівнів доступу* (для користувача-власника, для користувачів групи-власника, і для всіх інших користувачів) і трьох базових *прав доступу* до файлів (на читання, на запис та на виконання). Базові засоби захисту процесів засновані на відстеженні принадлежності процесів користувачам.

Для відстеження власників процесів і файлів використовуються числові ідентифікатори. *Ідентифікатор користувача і групи* - ціле число (зазвичай) в діапазоні від 0 до 65535. Присвоєння унікального ідентифікатора користувача виконується при закладі системним адміністратором нового реєстраційного імені. Значення ідентифікатора користувача і групи - не просто числа, які ідентифікують користувача, - вони визначають власників файлів і процесів. Серед користувачів системи виділяється один користувач - *системний адміністратор* або *суперкористувач*, що володіє всією повнотою прав на використання і конфігурування системи. Це користувач з ідентифікатором 0 і реєстраційним ім'ям **root**.

При поданні інформації людині зручніше використовувати замість відповідних ідентифікаторів символільні імена - реєстраційне ім'я користувача і назва групи. Відповідність ідентифікаторів і символічних імен, а також інша інформація про користувачів і групи в системі (*облікові записи*), як і більшість іншої інформації про кон-

фігурацію системи UNIX, за традицією, представлена у вигляді текстових файлів. Ці файли - **/etc/passwd**, **/etc/group** та **/etc/shadow** (у системах з тіньовим зберіганням паролів) - детально описані нижче.

Файл **/etc/passwd**. Кожен рядок (обліковий запис) у файлі **/etc/passwd** описує одного відомого системі користувача і має сім розділених двокрапкою полів. Приклад запису:

```
spot:x:502:502:Linux User,,,:/root/spot:/bin/sh
```

Призначення полів цього запису представлено в наступній таблиці.

Таблиця 1. Поля файлу **/etc/passwd і їх призначення**

Поле	Призначення
Ім'я користувача (реєстраційне ім'я)	Містить символічне ім'я користувача, яке використовується при реєстрації в системі. У межах однієї машини повинно бути унікальним. Реєстраційне ім'я має складатися з алфавітно-цифрових символів (нижнього регістру), без пробілів, з максимальною довжиною, яка визначається конкретної ОС. Найбільш часто використовується максимальна довжина - вісім символів. Дублювання імен користувачів, призводить до певних ускладнень. Наприклад, дублікати з'являються тоді, коли адміністратор використовує в імені більше 8 символів. Тоді для системи jarmstrong той же, що jarmstroff . Коли ім'я так продубльовано, система використовує першу знайдену для нього запис у файлі /etc/passwd і ігнорує наступні.
Пароль	Поле зберігає зашифрований пароль. Допускається порожнє поле. При використанні системи тіньового зберігання паролів, в цьому полі знаходиться тільки мітка пароля (x), а зашифрований пароль зберігається в іншому місці. Правила завдання пароля зазвичай знаходяться у файлі /etc/default / passwd , (наприклад, директива PASSLENGTH = число в цьому файлі задає мінімальну кількість символів в паролі). Деякі системи також вра-

Таблиця 1. Поля файлу /etc/passwd і їх призначення

Поле	Призначення
	ховують регистр, а в деяких передбачається використання як мінімум одного не алфавітно-цифрового символу.
Ідентифікатор користувача	Поле зберігає числовий ідентифікатор користувача, який пов'язаний з його реєстраційним ім'ям. Будь-який створений користувачем файл або запущений процес асоціюється з його числовим ідентифікатором.
Ідентифікатор групи	Містить числовий код групи. Будь-який створений користувачем файл асоціюється з його ідентифікатором групи. Зазначена тут група є основною (первинної) для даного користувача.
Коментар	Містить коментар - будь-яку алфавітно-цифровий рядок. Це поле може містити інформацію про реального власника реєстраційного імені. ОС UNIX не задає його формат. Деякі програми друку та електронної пошти використовують це поле для виводу справжнього імені користувача.
Початковий каталог	Визначає початковий каталог користувача. Коли користувач починає сеанс роботи, система поміщає його в даний каталог. Користувач повинен мати відповідні права доступу до нього.
Початкова команда	Визначає командну середовище користувача (зазвичай запускається один з <i>командних інтерпретаторів</i> UNIX, але, теоретично, можна вказати будь-яку команду). Це поле можна змінювати.

Файл /etc/group. Цей файл співвідносить числові ідентифікатори груп з символьними іменами. Кожен рядок файла /etc/group містить чотири поля. Поля розділяються крапкою з комою. Призначення полів цього запису представлено в табл. 2.

Таблиця 2. Поля файлу /etc/group та їх призначення

Поле	Призначення
------	-------------

Таблиця 2. Поля файлу /etc/group та їх призначення

Поле	Призначення
Назва групи	Містить (унікальне) символьне ім'я групи.
Пароль групи	Групи можуть мати паролі, хоча використання паролів груп - явище рідкісне. У прикладі це поле порожнє - це означає, що пароль відсутній.
Ідентифікатор групи	Містить числовий код групи.
Список користувачів	Містить список реєстраційних імен користувачів цієї групи. Імена в цьому списку розділяються комами. Користувачі можуть належати до декількох груп і, при необхідності, перемикатися між ними за допомогою команди newgrp .

Приклад запису з файлу /etc/group:

bin::2:root,bin,daemon

Файл /etc/shadow. Цей файл використовується в системах з *тіньовим зберіганням паролів*, де вони винесені з доступного всім користувачам на читання файлу /etc/passwd для підвищення безпеки системи. Тут (крім власне зашифрованих паролів) зберігаються додаткові обмеження, пов'язані з реєстраційним ім'ям і паролем користувача. Доступ до цього файла на читання має тільки користувач **root**, а працюють з ним команди **passwd** і **login**.

Файл містить по одному запису з восьми полів, розділених двокрапкою, для кожного облікового запису в системі. Призначення полів цього запису представлено в табл. 3.

Таблиця 3. Поля файла /etc/shadow і їх призначення

Номер поля	Призначення
1	Ім'я користувача.
2	Зашифрований за особливим алгоритмом (зазвичай, DES чи MD5) пароль.

Таблиця 3. Поля файлу /etc/shadow і їх призначення

Номер поля	Призначення
3	Кількість днів між 01. 01. 1970 (початком ери UNIX) і вдень останньої зміни пароля.
4	Мінімальна кількість днів між змінами пароля.
5	Термін дії пароля користувача.
6	За скільки днів система буде починати попереджати користувача про необхідність зміни пароля.
7	Скільки днів користувач може не працювати в системі, перш ніж його реєстраційне ім'я буде заблоковано.
8	Дата, після якої ім'я користувача не можна буде використовувати в системі.

2.3.2. Системні реєстраційні імена

Кожна версія ОС UNIX резервує декілька спеціальних реєстраційних імен для зумовлених системних цілей. Так, в UNIX SVR4 системними вважаються реєстраційні імена, відповідні ідентифікаторами від 0 до 100. Найбільш часто резервуються реєстраційні імена, представлені в табл. 4.

Таблиця 4. Системні реєстраційні імена в ОС UNIX SVR4

Реєстраційне ім'я	Призначення
root	Реєстраційне ім'я суперкористувача, адміністратора системи, відповідне ідентифікатору 0. Єдине ім'я, обов'язково наявне в будь-якій UNIX-системі. Користувач root не пов'язаний жодними обмеженнями з доступу. Для виконання більшості програм адміністрування використовується реєстраційне ім'я root , забезпечує

Таблиця 4. Системні реєстраційні імена в ОС UNIX SVR4

Реєстраційне ім'я	Призначення
	гарантований доступ до необхідних ресурсів.
daemon	Власник процесів, що реалізують власні служби.
sys	Власник виконуваних користувальницьких системних команд UNIX (часто відповідає ідентифікатору 0).
bin	Власник стандартних користувацьких утиліт UNIX (часто відповідає ідентифікатору 0).
adm	Псевдовласник, що володіє файлами системи журналізації.
cron	Псевдовласник, що володіє відповідними файлами, від імені якого виконуються процеси підсистеми запуску програм за розкладом.
news	Псевдовласник, від імені якого виконуються процеси системи телеконференцій (<i>дискусійних груп чи груп новин</i>).
nobody	Псевдовласник, що використовується при роботі мережової файлової системи NFS.
uucp	Псевдовласник підсистеми UUCP, що дозволяє передавати поштові повідомлення і файли між UNIX-хостами.
lp , lpd	Псевдовласник, від імені якого виконуються процеси системи друку, що володіє відповідними файлами.

Точно так само ставляться і системні групи у файлі **/etc/group**. У SVR4 зарезервованими вважаються імена груп з ідентифікаторами від 0 до 100.

Зміна чинного ідентифікатора користувача. Команда **su** призначена для тимчасової зміни діючого (ефективного) ідентифікатора користувача і сеансу користувача. Вона має такий синтаксис:

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

Команда **su** запитує пароль (у всіх користувачів, крім **root**, і якщо пароль існує). У разі відповідності пароля створюється новий сеанс від імені нового користу-

вача. У наступному прикладі зберігається середовище користувача з ім'ям **user01**, включаючи поточний робочий каталог і змінні середовища:

```
$ logname
user01
$ su informix
Password:
$ logname
user01
$ echo $LOGNAME
user01
$ set
HOME=/home/user01
LOGNAME=user01
MAIL=/var/mail/user01
PWD=/home/user01
...
$ exit
$
```

Якщо введена команда **su - реєстраційне_ім'я**, то система надає користувачеві командний інтерпретатор та середовище відповідно до зазначеного реєстраційним ім'ям:

```
$ logname
user01
$ su - informix
Password:
$ logname
user01
$ echo $LOGNAME
informix
$ set
HOME=/home3/informix
LOGNAME=informix
MAILPATH=/usr/mail/informix
```

```
PWD=/home3/informix
```

```
...  
$ exit  
$
```

Команда у форматі **su реєстраційне_ім'я -c аргументи** сприймає **аргумент** як команду, яку необхідно виконати з реєстраційним іменем нового користувача. Для виконання команди запитується пароль нового користувача і використовуються його права доступу. Після завершення виконання відбувається повернення в середу користувача, який викликав команду **su**. Таким чином, якщо користувачеві, наприклад, треба видалити файл користувача з реєстраційним ім'ям **new_user**, необхідно виконати команду:

```
$ su new_user -c "rm file"  
Password:  
$
```

Команда **su** без зазначення реєстраційного імені дозволяє отримати права користувача **root**. При цьому необхідно знати і правильно ввести пароль користувача **root**. Якщо користувач працює під реєстраційним ім'ям **root**, вводити пароль при зміні діючого ідентифікатора не потрібно.

Зміна чинного ідентифікатора групи. Відразу після реєстрації користувач працює від імені основної групи (задана в файлі /etc/passwd). Okрім основної, користувач може належати до будь-якої кількості *додаткових груп*. Ці групи задаються шляхом зазначення реєстраційного імені в четвертому полі рядка у файлі /etc/group, яка описує додаткову групу. Членство в додаткових групах або враховується при визначенні прав доступу автоматично (BSD-системи), або для переходу в додаткову групу і зміни тим самим чинного ідентифікатора групи використовується команда **newgrp** (SVR4) з таким синтаксисом:

```
/usr/bin/newgrp [ группа ]
```

Команда **newgrp** вбудована в деякі командні інтерпретатори (**sh**, **ksh**). Вона переводить користувача до нової групи шляхом запуску нового командного інтерпретатора з реальним і ефективним ідентифікатором (GID) нової групи. При цьому новий командний інтерпретатор запускається навіть якщо переход в групу завершився помилкою (наприклад, вказана неіснуюча група). Природно, в новому командно-

му інтерпретаторі будуть мати нестандартні та непорожні значення тільки змінні, експортувані в середу.

При виклику без операнда, команда **newgrp** переводить користувача в його основну групу, скасовуючи тим самим дія попередніх команд **newgrp**.

Якщо в другому полі запису відповідної групи в файлі **/etc/group** вказаний пароль (тобто якщо це поле не пусте) і користувач не вказаний у четвертому полі як член групи, при переході до групи у користувача **запитується пароль**. Єдиний спосіб створити пароль групи - скористатися командою **passwd** для завдання пароля однієї з облікових записів користувачів, а потім скопіювати зашифрований пароль з файлу **/etc/shadow** у файл **/etc/group**. Паролі груп зараз використовують рідко.

Зміна пароля і характеристик облікового запису, пов'язаних з реєстрацією.

Команда **passwd** дозволяє будь-якому користувачеві змінити пароль або отримати список атрибутів поточного пароля для свого реєстраційного_імені. Привілеїовані користувачі можуть запускати **passwd** для виконання цих функцій для будь-якого користувача, а також для установки атрибутів пароля для будь-якого користувача.

Пароль зазвичай задається адміністратором при створенні облікового запису користувача для власника **реєстраційного_імені**. Надалі користувач може змінити пароль за допомогою команди **passwd**.

Команда **passwd** має такий синтаксис:

```
passwd [реєстраційне_ім'я]
passwd [-l|-d][-f][-x max][-n min][-w warn] реєстраційне_ім'я
passwd -s [-a]
passwd -s [реєстраційне_ім'я]
```

Опції команди представлені в табл. 5. Звичайні користувачі можуть використовувати тільки опцію **-s**.

Таблиця 5. Опції команди **passwd**

Опція	Призначення
-s	Показує атрибути пароля для реєстраційного_імені користувача.

Таблиця 5. Опції команди passwd

Опція	Призначення
	Будь-який користувач може задавати дану опцію.
-l	Блокує запис пароля для реєстраційного_імені .
-d	Видаляє пароль для реєстраційного_імені , так що у користувача з цим регистраціонним_іменем пароль не вимагається.
-f	Примушує користувача змінити пароль при наступній реєстрації в системі, роблячи пароль для реєстраційного_імені застарілим.
-x max	Задає для користувача з вказаним регистраціонним_іменем кількість днів, протягом яких пароль буде дійсний.
-n min	Задає мінімальну кількість днів між змінами пароля для користувача з вказаним регистраціонним_іменем . Завжди використовуйте цю опцію з опцією -x , якщо тільки max не встановлено в -1 (старіння вимкнено). У цьому випадку, min встановлювати не потрібно.
-w warn	Визначає, за скільки днів (щодо max) користувача з даними регистраціонним_іменем будуть попереджати про майбутній застарінні пароля.
-s -a	Показує атрибути паролів для всіх користувачів.

Правила побудови паролів. При створенні паролів існують такі вимоги:

- Пароль повинен містити не менш **PASSLENGTH** символів, як визначено у файлі **/etc/default/passwd**. Значення **PASSLENGTH** повинно бути не менше 3. Враховуються тільки перші вісім символів пароля.
- Пароль повинен містити не менше двох буквених символів і однієї цифри або спеціального символу. (У даному випадку до буквеним символів відносяться всі великі та малі літери.)
- Пароль повинен відрізнятися від реєстраційного імені користувача і від будь-якого слова, одержуваного *циклічним* (circular shift) або *зворотним* (reverse shift) зрушеннем цього реєстраційного імені. (Відповідні прописні і малі літери вважаються співпадаючими.)

Ці вимоги не поширюються на користувача **root**.

Команда passwd. При використанні для зміни паролю команда **passwd** запитує у звичайних користувачів їх старий пароль, якщо він заданий. Якщо з моменту завдання старого пароля пройшло досить багато часу, **passwd** потім пропонує користувачеві двічі ввести новий пароль, інакше програма припиняє роботу. Потім **passwd** перевіряє, чи задовільняє новий пароль описаним вище правилам побудови. При введенні нового пароля вдруге, дві копії нового пароля порівнюються. Якщо вони не збігаються, цикл запиту нового пароля повторюється, але не більше двох разів.

Користувач **root** може змінювати будь-який пароль; команда **passwd** не запи- тує в нього старий пароль.

Старіння паролів. Паролі дійсні протягом обмежених періодів часу (що визнача- ються системним адміністратором), після чого їх необхідно змінити. Тому необхід- но зберігати інформацію про період активності для кожного пароля. Коли наближа- ється дата закінчення терміну дії пароля, його власникам пропонується вибрати но- вий пароль протягом певної кількості найближчих днів. Процес відстеження термі- нів дії паролів і повідомлення користувачів про необхідність змінити пароль назива- ється *старінням паролів* (password aging).

Інформація про паролі всіх користувачів системи зберігається у файлі **/etc/shadow**, який можуть читати тільки привілейовані користувачі. Кожен рядок користувача у файлі **/etc/shadow** містить чотири параметри, що визначають застаріння пароля (по- ля 3-6, див. табл. 3). Останні три з цих параметрів можна встановити опціями ко- мандного рядка **-n**, **-x** та **-w**, відповідно. При відсутності опцій, їх значення беруться з файлу **/etc/default/passwd**.

Показ атрибутів пароля. Коли команда **passwd** використовується для показу атри- бутів пароля, результати видаються в такому форматі:

```
login_name status lastchanged minimum maximum warn
```

або, якщо відсутня інформація, пов'язана з старінням пароля,

```
login_name status
```

Поля визначені таким чином: **login_name** - реєстраційне ім'я користувача; **status** - статус пароля для реєстраційного_імені (**PS** означає наявність пароля, **LK** означає, що реєстрація заблокована, а **NP** означає відсутність пароля).

Стандартні значення атрибутів. Привласнюючи значення набору параметрів у файлі `/etc/default/passwd`, адміністратор може управляти старінням і довжиною паролів. Можна поставити такі параметри:

MINWEEKS Мінімальна кількість тижнів перед тим, як пароль можна буде змінити. Відразу після установки системи цей параметр має значення 0.

MAXWEEKS Максимальна кількість тижнів, протягом яких пароль можна не змінювати. Відразу після установки системи цей параметр має значення 24.

WARNWEEKS Кількість тижнів перед старінням пароля, коли необхідно попереджати користувача. Відразу після установки системи цей параметр має значення 1.

PASSLENGTH Мінімальна кількість символів в паролі. Відразу після установки системи цей параметр має значення 6.

Зверніть увагу, що аргументи опцій команди **passwd** (**min**, **max** та **warn**), а також відповідні поля файла `/etc/shadow` задають параметри старіння в днях; тоді як відповідні поля файла `/etc/default/passwd` (**MINWEEKS**, **MAXWEEKS** і **WARNWEEKS**) – в тижнях.

2.3.3. База даних облікових записів

Для перегляду бази даних облікових записів системи призначена команда **logins**. Команда **logins** видає інформацію про користувача і системних реєстраційних іменах. Зміст видаваної інформації управляється опціями команди і може включати: реєстраційне ім'я, ідентифікатор користувача, опис облікового запису у файлі `/etc/passwd` (реальне ім'я користувача або інша інформація), ім'я основної гру-

пи, ідентифікатор основної групи, імена груп, ідентифікатори груп, початковий каталог, початковий командний інтерпретатор та чотири параметри старіння пароля.

За замовчуванням видається така інформація: реєстраційне ім'я, ідентифікатор користувача, ім'я основної групи, ідентифікатор основної групи і поле опису облікового запису у файлі **/etc/passwd**. Результат сортується за ідентифікатором користувача, в результаті чого спочатку йдуть системні реєстраційні імена, а потім - призначені для користувача.

Команда **logins** має такий синтаксис:

```
logins [-dmopstuxa] [-g групpl] [-l рег_имена]
```

Дія опцій команди logins представлено в табл. 6.

Таблиця 6. Опції команди logins

Опція	Призначення
-d	Вибирати реєстраційні імена користувачів які мають однакові ідентифікатори.
-m	Показати всі групи, до яких належить користувач.
-o	Форматувати вивід у вигляді одного рядка полів, розділених двокрапками.
-p	Вибирати реєстраційні імена без паролів.
-s	Вибирати всі системні реєстраційні імена.
-t	Сортувати результати по реєстраційному імені, а не по ідентифікатору користувача.
-u	Вибирати всі реєстраційні імена користувачів.
-x	Видати розширену інформацію про вибраного користувача. Ця розширенна інформація включає початковий каталог, початковий командний інтерпретатор і інформацію про старіння паролю, причому кожний елемент виводиться в окремому рядку. Інформація про пароль містить статус пароля (PS у разі його наявності, NP при відсутності пароля або LK для заблокованого реєстраційного імені), дату

	останньої зміни пароля, кількість днів, після яких потрібно змінити пароль, мінімальну кількість днів між змінами і за скільки днів користувач почне отримувати (при реєстрації) попередження про необхідність зміни пароля.
-a	Додає до результату два поля, зв'язаних зі старінням пароля. Вони показують, скільки днів пароль можна не використовувати, перед тим як він автоматично деактивується, і дату старіння пароля.
-g	Вибирає всіх користувачів, які належать до вказаної групи. Можна вказати декілька груп у вигляді списку через кому.
-l	Вибирає вказане реєстраційне ім'я. Можна вказати декілька реєстраційних імен у вигляді списку через кому.

У разі спільногого використання декількох опцій будуть показані облікові записи, що задовольняють будь-якому з критеріїв. У разі спільногого використання опцій **-l** та **-g** інформація про користувача буде віддана один раз, навіть якщо він належить до декількох вказаних груп.

2.3.4. *Отримання списку зареєстрованих користувачів*

Для отримання списку користувачів, які в даний час працюють в системі, можна використати команду **who** з таким синтаксисом:

/usr/bin/who [-abdhlmprstTu] [файл]

/usr/bin/who -q [-n x] [файл]

/usr/bin/who am i

Останній варіант видає рядок, відповідну запитуючій сесії, і може використовуватися для самоідентифікації.

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**. Якщо файл (який повинен мати формат **utmp(4)**) існує, інформація береться з нього.

У загальному випадку, результат має такий вигляд:

ім'я [стан] термінал час [очікування] [pid] [коментар] [статус виходу]

де: **ім'я** - реєстраційне ім'я користувача; **стан** - можливість запису на термінал; **термінал** - ім'я терміналу з каталогу **/dev**; **час очікування** - час реєстрації користувача; **pid** - час, що минув після останнього дії користувача; **коментар** - ідентифікатор процесу командного інтерпретатора; **статус виходу** - рядок коментарю з файлу **/etc/inittab**.

Опції команди **who** представлені в табл. 7.

Таблиця 7. Опції команди who

Опція	Призначення
-a	Обробляє /var/adm/utmp або вказаний файл з опціями -b, -d, -l, -p, -r, -t, -T та -u .
-b	Видає дату і час останньої перезавантаження.
-d	Видає всі процеси, припинені і не перезапущені процесом init . Для "мертвих" процесів буде видано полі статусу виходу. Це може стати в нагоді для з'ясування причини припинення процесу. Тільки для SVR4.
-h	Видає заголовки стовпців.
-l	Видає тільки термінали, на яких система очікує реєстрації користувачів. Як ім'я для них видається LOGIN . Інші поля - такі ж, як і для користувачів, але поле стану не виводиться.
-m	Видає інформацію тільки про поточний терміналі.
-nx	Видає по x користувачів в рядку. Значення x повинно бути не менше 1. Опція -n може використовуватися тільки з опцією -q .
-p	Видає інформацію про активні процеси, запущених раніше процесом init . У центрі імені видається ім'я програми, запущеної процесом init відповідно до файлом /sbin/inittab . Поля стану, терміналу і очікування в цьому випадку не мають сенсу. Поле коментарю показує ідентифікатор рядки з файлу /sbin/inittab , яка запустила цей процес. Тільки для SVR4.
-q	(Quick who) Видає тільки імена і кількість зареєстрованих користу-

Таблиця 7. Опції команди who

Опція	Призначення
	вачів. Якщо задана ця опція, інші опції ігноруються.
-r	Показує поточний рівень виконання процесу init . Тільки для SVR4.
-s	Видає тільки поля імені, терміналу і часу реєстрації. Використовується за замовчуванням.
-t	Те ж, що й опція -s , але також видаються поля стану, часу очікування, pid і коментар. У полі стану видається один з таких символів: + – термінал дозволяє запис іншим користувачам; - – термінал забороняє запис іншим користувачам; ? – можливість запису на термінал не визначена.

Розглянемо приклади виконання команди who:

```
$ who -a | more
.
system boot  Фев 23 15:39
.
run-level 3  Фев 23 15:39      3      0  S
rc2          .
root + console  Фев 27 21:34  0:28  4612      (:0)
rc3          .
sac          .
LOGIN        console  Фев 23 15:41  0:28  428
panaslog    .
netwatch    .
zsmon       .
informix + pts/1   Mar 25 10:13 15:21  1796      (khuja.domain.com)
eugene     + pts/3   Mar 22 18:23 15:24  23392     (khuja.domain.com)
serj       + pts/4   Mar 18 10:41 old    13278     (sysadm.domain.com)
serj       + pts/15  Mar 25 11:32 14:51  3004      (sysadm.domain.com)
spot        + pts/14  Mar 26 09:39   .    11615      (creator.domain.com)
alex        + pts/2   Mar 21 14:18 16:13  14526     (alex.domain.com)
informix + pts/17  Mar 21 13:19 17:50  14012     (bachin.domain.com)
informix  pts/6    Mar 25 18:34 15:05  3572      id=t800 term=0 exit=0
                                               (lyapota.domain.com)
```

```

lyapota    pts/7      Map 25 18:34 17:58   3577      id=t900 term=0 exit=0
                                         (lyapota.domain.com)
informix + pts/5      Map 5 14:48 15:33   27664     (alex.domain.com)
spot       pts/8      Map 25 18:24 15:15   8916      id=tB00 term=0 exit=0
--More--

```

У найпростішому випадку програма **who** викликається без параметрів:

```

$ who
root      console    Фев 27 21:34   (:0)
informix   pts/1      Map 25 10:13   (khuja.domain.com)
eugene    pts/3      Map 22 18:23   (khuja.domain.com)
serj      pts/4      Map 18 10:41   (sysadm.domain.com)
serj      pts/15     Map 25 11:32   (sysadm.domain.com)
spot      pts/14     Map 26 09:39   (creator.domain.com)
alex      pts/2      Map 21 14:18   (alex.domain.com)
informix   pts/17     Map 21 13:19   (bachin.domain.com)
informix   pts/5      Map 5 14:48    (alex.domain.com)
root      pts/13     Фев 27 21:35   (:0. 0)
root      pts/16     Map 25 17:24   (:0. 0)

```

Нарешті, можна використовувати команду **whoami** для самоідентифікації:

```

$ whoami
root

```

2.3.5. Засоби створення, зміни та видалення облікових записів користувачів

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

Для створення, зміни та видалення облікових записів всі версії ОС UNIX пропонують три команди, **useradd**, **usermod** та **userdel**, відповідно. Вони в більшості систем мають такий синтаксис:

```

useradd [-u ідентифікатор [-o] [-i]] [-g група]
        [-G групап[[,групап]. . . ]]] [-d каталог] [-s shell]

```

```

[-c коментар] [-m [-k skel_dir]] [-f inactive]
[-e expire] рег_ім'я

usermod [-u ідентифікатор [-o]] [-g група]
[-G груп[а][,груп[а]]. . . ] [-d каталог [-m]]
[-s shell] [-c коментар] [-l новое_рег_ім'я]
[-f inactive] [-e expire] рег_ім'я

userdel [-r] рег_ім'я

```

Ці команди дозволяють виконати лише узгоджені і допустимі зміни у файлах **/etc/passwd**, **/etc/shadow** і **/etc/group**. Команди управління обліковими записами, у загальному випадку, може виконувати тільки користувач **root**. Основні опції команд управління обліковими записами представлені в табл. 8.

Таблиця 8. Основні опції команд управління обліковими записами

Опція	Призначення
-u іден-тифіка-тор	<i>Ідентифікатор користувача (UID)</i> . Повинен бути невід'ємним цілим числом, не переважаючим MAXUID , визначений у sys/param.h . За замовчуванням використовується наступний доступний (унікальний) не застарілий UID в діапазоні користувальницьких ідентифікаторів.
-o	Ця опція дозволяє продублювати UID (зробити його не унікальним). Оскільки захист системи в цілому, а також цілісність <i>контрольного журналу</i> (audit trail) та <i>реєстраційної інформації</i> (accounting information) зокрема, залежить від однозначної відповідності кожного UID конкретної фізичної особи, використовувати цю опцію не рекомендується.
-I	Дозволяє використовувати застарілий ідентифікатор UID.
-g група	Ціличисловий ідентифікатор або символне ім'я наявної групи. Ця опція задає <i>основну групу</i> (primary group) для нового користувача. За умовчанням в SVR4 використовується стандартна група, зазначена у файлі /etc/default/useradd . В ОС FreeBSD і Linux прийнято за замовчуванням створювати для кожного користувача окрему приватну

Таблиця 8. Основні опції команд управління обліковими записами

Опція	Призначення
	основну групу, ім'я якої збігається з ім'ям користувача.
-g група [[, гру- па] . . .]	Один або декілька елементів у списку через кому, кожен з яких представляє собою цілочисловий ідентифікатор або символьне ім'я наявної групи. Цей список визначає приналежність до <i>додаткових груп</i> (supplementary group membership) для користувача. Повторення ігноруються. Кількість елементів у списку не повинно перевершувати NGROUPS_MAX-1 , оскільки загальна кількість додаткових груп для користувача плюс основна група не має перевершувати NGROUPS_MAX .
-D ката- лог	Початковий каталог (home directory) нового користувача. Довжина цього поля не повинна перевищувати певної межі (зазвичай - від 256 до 1024 символів). За замовчуванням використовується HOMEDIR/рег_імя , де HOMEDIR - базовий каталог для початкових каталогів нових користувачів, а рег_імя - реєстраційне ім'я нового користувача.
-s shell	Повний шлях до програми, яка використовується в якості початкового командного інтерпретатора для користувача відразу після реєстрації. Довжина цього поля не повинна перевищувати певної межі (зазвичай - від 256 до 1024 символів). За умовчанням в цьому полі використовується стандартний командний інтерпретатор /bin/sh . Як значення shell повинен бути зазначений існуючий виконуваний файл. В іншому випадку, користувач не зможе зареєструватися в системі.
-c ко- ментар	Будь-яка текстовий рядок. Зазвичай, це короткий опис реєстраційного імені, наприклад, прізвище та ім'я реального користувача. Ця інформація зберігається в записі користувача у файлі /etc/passwd . Довжина цього поля не повинна перевищувати 128 символів.
-m	Створює початковий каталог нового користувача, якщо він ще не

Таблиця 8. Основні опції команд управління обліковими записами

Опція	Призначення
	існує. Якщо каталог вже існує, що додається користувач повинен мати права на доступ до зазначеного каталогу.
-k skel_dir	Копіює вміст скелетного каталогу skel_dir в початковий каталог нового користувача, замість вмісту стандартного скелетного каталогу, /etc/skel . Каталог skel_dir повинен існувати. Стандартний скелетний каталог містить стандартні файли, що визначають середовище роботи користувача. Поставлене адміністратором каталог skel_dir може містити аналогічні файли і каталоги, створені для певної мети.
-f inactive	Максимально допустима кількість днів між реєстраціями, коли це ім'я ще не оголошується недійсним. Звичайно як значень використовуються позитивні цілі числа.
-e expire	Дата, починаючи з якої реєстраційне ім'я більше не можна буде використовувати; після цієї дати ніякої користувач не зможе отримати доступ під цим реєстраційним ім'ям. (Ця опція зручна при створенні тимчасових реєстраційних імен.) Вводити значення аргументу expire (що представляє собою дату) можна в будь-якому підтримуваному локаллю форматі (крім Julian date). Наприклад, можна ввести 10/6/99 або October 6, 1999 .
-l нове_рег_ім'я	Рядок друкованих символів, що задає нове реєстраційне ім'я для користувача. Вона не повинна містити двокрапок (:) і перекладів рядків (\ n). Крім того, вона не повинна починатися з великої літери.
-r	При видаленні облікового запису видалити початковий каталог користувача із системи. Цей каталог повинен існувати. Після успішного виконання команди файли і підкаталоги в початковому каталозі будуть недоступні.
рег_ім'я	Рядок друкованих символів, що задає реєстраційне ім'я для нового користувача. У ній не повинно бути двокрапок (:) і символів перевідкинення.

Таблиця 8. Основні опції команд управління обліковими записами

Опція	Призначення
	ведення рядка (<code>\ n</code>). Вона також не повинна починатися з великої літери.

Врахуйте, що знову створений обліковий запис блокується до тих пір, поки не буде виконана команда `passwd`, що задає пароль новому користувачеві.

Розглянемо ряд простих прикладів управління обліковими записами:

```
# useradd -c "Student 1" -d /home/user01 -g ixusers -m -s /bin/bash user01
# usermod -c "Student 1 of UNIX Course" -G others -s /bin/ksh user01
# userdel -r user01
```

2.3.6. Засоби створення, зміни та видалення груп

Для створення, зміни та видалення груп всі версії ОС UNIX пропонують три команди, `groupadd`, `groupmod` та `groupdel`, відповідно. Вони мають такий синтаксис:

```
groupadd [-g ідентифікатор [-o]] група
groupmod [-g ідентифікатор [-o]] [-n ім'я] група
groupdel група
```

Ці команди дозволяють виконати лише узгоджені і допустимі зміни у файлі `/etc/group`. Команди управління групами, в загальному випадку, може виконувати тільки користувач `root`. Опції і операнди команд управління групами представлена в табл. 9.

Таблиця 9. Опції команд управління групами

Опція	Призначення
<code>-g іден-тифікатор</code>	Ідентифікатор нової групи (GID). Цей ідентифікатор групи повинен бути невід'ємним десятирічним цілим числом, що не перевищує значення MAXUID , визначеного в заголовочному файлі <code><param.h></code> . За

Таблиця 9. Опції команд управління групами

Опція	Призначення
	умовчанням виділяється унікальний ідентифікатор групи, що не належить до зарезервованих. У UNIX SVR4 ідентифікатори груп в діапазоні 0-100 зарезервовані.
-o	Ця опція дозволяє задавати дублюючий (не унікальний) код групи.
-n ім'я	Рядок друкованих символів, що задає нове ім'я для групи при зміні. Рядок не повинна містити двокрапки (:) або переведення рядків (\n).
група	Ім'я створюваної, змінною або видаляється групи. Назва групи не повинно містити символи двокрапки (:) або переведення рядка (\n).

Врахуйте, що при видаленні групи просто видаляється рядок з файлу **/etc/group**. Ніякі зміни у файловій системі і в облікових записах користувачів команди **groupmod** та **groupdel** не виробляють. Відповідні дії за погодженням, при необхідності, повинен виконувати системний адміністратор - користувач *root*.

Наприкінці наведемо декілька простих прикладів управління групами:

```
# groupadd -g 101 informix
# groupmod -g 102 -o -n ixusers informix
# groupdel ixusers
```

3. ФАЙЛОВА СИСТЕМА ОС UNIX

3.1. Ієрархічні файлові системи

3.1.1. Поняття логічної файлової системи

Операційна система виконує дві основні задачі: маніпулювання даними та їх зберігання. Більшість програм в основному маніпулюють даними, але, в кінцевому рахунку, вони де-небудь зберігаються. У 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.

Файл – це іменований впорядкований набір даних на пристрой зберігання інформації; операційна система забезпечує організацію файлів в файлові системи.

Файлова система – це спосіб організації даних, який застосовується операційною системою для збереження інформації у вигляді файлів на носіях. Також цим поняттям позначають сукупність файлів та директорій, які розміщаються на логічному або фізичному пристрої. Створення файлової системи відбувається в процесі форматування.

Залежно від організації файлів на носіях даних файлові системи можуть поділятись на такі групи:

- ієрархічні файлові системи - дозволяють розміщувати файли в каталоги;
- плоскі файлові системи - не використовують каталогів;
- кластерні файлові системи - дозволяють розподіляти файли між кількома однотипними фізичними пристроями однієї машини;
- мережні файлові системи - забезпечують механізми доступу до файлів однієї машини з інших машин мережі;
- розподілені файлові системи - забезпечують зберігання файлів шляхом їх розподілу між кількома машинами мережі.

Сучасні файлові системи (ФС) являють собою ієрархічні структури каталогів. Хоча загальна концепція всіх ФС, у принципі, одна, в реалізації є деякі відмінності. Два вартих уваги приклади — це символи-роздільники каталогів та чутливість до регистра. ОС UNIX використовують як роздільник каталогів символ скісної риски

(/), у той час як MS-DOS використовує цей символ для задавання додаткових опцій у командному рядку, а як роздільник прийнято використовувати символ зворотної скісної риски (\). У Microsoft Windows прийнята та ж конвенція, за винятком китайської та корейської версій, де роздільником є знак питання (?).

Логічна файлова система в ОС UNIX (або просто *файлова система*) - це ієрархічно організована структура всіх каталогів і файлів, що починається з *кореневого каталогу*. Файлова система UNIX забезпечує уніфікований інтерфейс доступу до даних, розташованим на різних носіях, і до периферійних пристрій. Логічна файлова система може складатися з однієї або кількох *фізичних файлових (під) систем*, які є розділами фізичних носіїв (дисків, CD-ROM або дискет).

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 пристріям, відповідним модулів підсистеми введення-виведення.

Файлова структура Unix характеризується такими властивостями:

- Чітка побудова.
- Звернення до даних файлу без суперечностей.
- Захист даних файлу.

3.1.2. Каталоги в ОС Unix

Ієрархічна структура файлової системи UNIX спрощує орієнтацію в ній. Кожен каталог, починаючи з кореневого (/), у свою чергу, містить файли та інші каталоги (*підкаталоги*). Кожен каталог містить також посилання на батьківський каталог (для кореневого каталогу батьківським є він сам), представлена каталогом з ім'ям дві точки (..) і посилання на самого себе, представлена каталогом з ім'ям точки (.).

Кожен процес має *поточний каталог*. Відразу після реєстрації поточним каталогом користувача (насправді, процесу - початкової програми, зазвичай, командного інтерпретатора) стає *початковий каталог*, вказаний у файлі **/etc/passwd**.

Кожен процес може послатися (назвати) на будь-який файл або каталог у файлової системі на ім'я. Способам завдання імен файлів присвячено наступний підрозділ.

Отримання інформації про поточний каталог. Команда **pwd** видає повне ім'я поточного (робочого) каталогу. Команда **pwd** не має параметрів. Ось приклад її використання:

```
$ pwd  
/home/user01  
$
```

3.1.3. Зміна поточного каталогу

Для зміни поточного каталогу використовується команда **cd**:
cd [каталог]

Якщо **каталог** не вказано, використовується значення змінної середовища **\$ НОМЕ** (зазвичай це *початковий каталог* користувача). Щоб зробити новий каталог поточним (увійти до каталогу), потрібно мати для нього **право на виконання**. Команда **cd** є вбудованою командою інтерпретатора і використовується для зміни поточного каталогу відповідний системний виклик.

Розглянемо приклад спільноговикористання команд **cd** та **pwd** для переходів між каталогами файлової системи:

```
$ pwd  
/home/user01  
$ cd . .  
$ pwd  
/home  
$ cd user01/tmp  
$ pwd  
/home/user/tmp  
$ cd  
$ pwd  
/home/user01
```

3.1.3. Файли в ОС UNIX

У ОС UNIX у назві файлу може використовуватись будь-який символ, за винятком скісної риски, крім того, вони чутливі до регистра. В ОС UNIX підтримується три способи вказівки імен файлів:

- *Коротке ім'я.* Ім'я, яке не містить спеціальних метасимволів коса риса (/), є коротким ім'ям файлу. За короткому імені можна послатися на файли поточного каталогу. Наприклад, команда `ls -l.profile` вимагає отримати повну інформацію про **файл.profile** в поточному каталогі.
- *Відносне ім'я.* Ім'я, не починається з символу косої риси (/), але включає такі символи. Воно посилається на файл щодо поточного каталогу. При цьому для посилання на файл або каталог в якомусь іншому каталогі використовується метасимвол косої риси (/). Наприклад, команда `ls -l../../profile` вимагає отримати повну інформацію про **файл.profile** в батьківському каталогі поточного каталогу, а команда `vi doc/text.txt` вимагає відкрити в редакторі `vi` файл **text.txt** в підкаталозі **doc** поточного каталогу.
- *Повне ім'я.* Ім'я, що починається з символу косої риси (/). Воно посилається на файл, починаючи від кореневого каталогу. Це ім'я ще називають *абсолютним*, оскільки 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 вона виконується.

Інші символи, крім косої риси, не мають в іменах файлів UNIX особливого значення (це не метасимволи). Зокрема, немає системного поняття *розширення* файлу.

В ОС UNIX немає теоретичних обмежень на кількість вкладених каталогів. Тим не менш, у кожній реалізації є практичні обмеження на максимальну довжину імені файлу, що зазначається у командах (як і на довжину командного рядка в цілому). Воно задається константою **PATH_MAX** в заголовковому файлі **/usr/include/limits.h**. Так, ім'я файлу не може бути довшим 1024 символів.

3.1.4. Отримання інформації про файли

Для перегляду інформації про типи (та інших атрибутих) файлів в ОС UNIX використовується команда **ls** з таким синтаксисом:

```
ls [ -abCcdeFfgillmnopqRrstux1 ] [файл. . .]
```

Команда **ls** видає інформацію про зазначені файлах або про файлах і каталогах в поточному каталозі (якщо **файл** не заданий). Формат і подробиця видаваної інформації залежить від опцій. Основні опції команди **ls** представлені в табл. 10:

Таблиця 10. Основні опції команди ls

Опція	Призначення
-A	Видає всі файли і підкаталоги, включаючи ті, імена яких починаються з крапки (.). За замовчуванням такі файли не видаються (вони вважаються <i>прихованими</i>).
-F	Додає до імені файлу суфікс, що показує його <i>тип</i> (див. наступний розділ). Позначає каталоги косою рисою (/), виконувані файли - зірочкою (*), іменовані канали (FIFO) - вертикальної рисою , символільні посилання - "символом @" (@), а сокети - знаком рівності =.
-I	Для кожного файла видає в першому стовпці лістингу номер <i>індексного дескриптора</i> (i-node). Про індексних дескрипторах див. у розділі, присвяченому фізичним файловим системам UNIX.
-L	Видає довгий лістинг, що включає права доступу, кількість посилань, власника, групу, розмір в байтах, час останньої зміни кожного файла i, природно, ім'я файла. Якщо файл є спеціальним файлом пристрою, замість розміру видаються головний і другорядний номер пристрою. Якщо з моменту останньої зміни пройшло більше 6 місяців, воно зазвичай видається у форматі 'місяць день рік '. Для файлів, змінених пізніше, ніж 6 місяців тому, час видається у форматі 'місяць день час'. Якщо файл є символічним посиланням, в довгому лістингу після імені файла вказується стрілочка (->) і ім'я файла, на

Таблиця 10. Основні опції команди ls

Опція	Призначення
	який вказує посилання.
-R	Змінює порядок сортування на зворотний стандартному (зворотний лексикографічний або спочатку самі старі файли, в залежності від інших опцій).
-R	Рекурсивно видає вміст підкаталогів.
-T	Сортує лістинг за часом <i>про</i> ої позначки (спочатку - найновіші), а не по імені файлу. За замовчуванням використовується час останньої зміни. (Опції -u і -c дозволяють сортувати за часом останнього звернення і часу створення, відповідно.)

Як видно з синтаксису, можна задавати одночасно декілька опцій. Ось як можна подивитись детальну інформацію про файли в каталозі **/tmp**, починаючи з тих, які змінювалися найдавніше:

```
$ cd /tmp
$ ls -lrt
total 108
-rw-r--r-- 1 root root    558 2011-02-20 11:32 pup_event_modprobe.conf
-rw-r--r-- 1 root root 32138 2011-02-20 16:32 bootkernel.log
-rw-r--r-- 1 root root      0 2011-02-20 16:32 bootcnt.txt
-rw-r--r-- 1 root root    888 2011-02-20 16:32 bootsysinit.log
-rw-r--r-- 1 root root   1661 2011-02-20 16:32 xerrs.log
-rw-r--r-- 1 root root    268 2011-02-20 16:32 pup_event_ok_pin
srw----- 1 root root      0 2011-02-20 16:45 geany_socket.f54d1b79
-rw-r--r-- 1 root root   3371 2011-02-20 17:23 udevtrace-modem.log
-rw-r--r-- 1 root root 18991 2011-02-20 17:23 udevtrace.log
prw-r--r-- 1 root root      0 2011-02-20 17:23
pup_event_backend_modprobe_protect_pipe
-rw-r--r-- 1 root root   4342 2011-02-20 17:23
pup_event_backend_modprobe_protect.log
drwx----- 2 root root   4096 2011-02-20 17:36 mc-root
-rw-r--r-- 1 root root      5 2011-02-20 17:48 pup_event_sizefreem
```

```
-rw-r--r-- 1 root root      0 2011-02-20 18:10 snapmergepuppyrequest
drwxr-xr-x 2 root root 4096 2011-02-20 18:28 1
-rw-r--r-- 1 root root      4 2011-02-20 18:30 pup_event_frontend_identify1_sr0
-rw-r--r-- 1 root root     20 2011-02-20 18:30 pup_event_frontend_block1
-rw-r--r-- 1 root root      4 2011-02-20 18:30 pup_event_frontend_identify2_sr0
-rw-r--r-- 1 root root     20 2011-02-20 18:30 pup_event_frontend_block2
```

Основним форматом результатів **ls** є так званий *довгий лістинг* (задається опцією **-l**). За замовчуванням видаються тільки імена файлів:

```
$ cd /
$ ls
bin  boot  dev  etc  lib  lib64  mnt  opt
proc  root  sbin  sys  tmp  usr          var
$
```

Часто використовується опція **-F**:

```
$ ls -F
bin/  boot/  dev/  etc/  lib/  lib64@      mnt/  opt/
proc/  root/  sbin/  sys/  tmp/  usr/        var/
```

Приклади використання і результати виконання команди **ls** представлені в наступних розділах.

3.1.5. Типи файлів

У UNIX існує декілька типів файлів, що розрізняються за функціональним призначенням і діям операційної системи при виконанні тих чи інших операцій над ними. У наступних підрозділах коротко представлені основні типи файлів, їх ознаки в довгих лістингах, а також способи їх створення.

Звичайний файл являє собою найбільш загальний тип файлів, що містить дані в деякому форматі. Для операційної системи такі файли є просто послідовність байтів. До цих файлів відносяться текстові файли, двійкові дані та їх програми.

У довгому лістингу ознакою звичайного файла є дефіс (**-**) у першій позиції першого стовпця:

```
-rw-rw-r-- 1 root      sys      8296 Фев 23 15:39 ps_data
```

Звичайні файли створюються текстовими редакторами (текстові), компіляторами (виконавчі), прикладними програмами за допомогою відповідного системного виклику або шляхом перенаправлення виводу:

```
$ cd /tmp  
$ touch f1.txt  
$ ls -l >f1.txt  
-rw-r--r-- 1 spot 50 0 Mar 26 14:40 f1.txt
```

Каталог. За допомогою каталогів формується логічне дерево файлової системи. Каталог – це файл, що містить імена знаходяться в ньому файлів, а також покажчики на додаткову інформацію - метадані, що дозволяють операційній системі робити дії з цими файлами. Каталоги визначають положення файлу в дереві файлової системи. Будь-який процес, який має право на читання каталога, може прочитати його вміст, але тільки ядро має право на запис даних каталогу.

У довгому лістингу ознакою каталогу є символ **d** в першій позиції першого стовпця:

```
drwxr-xr-x 2 informix informix 115 Feb 24 13:05 txt
```

Каталоги створюються командою **mkdir**:

```
mkdir каталог ...
```

Спеціальний файл пристрою. Забезпечує доступ до фізичних пристрій. У UNIX розрізняють *символьні* (character special device) та *блочні* (block special device) файли пристрій. Доступ до пристрій здійснюється шляхом відкриття, читання та запису в спеціальний файл пристрою.

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

У довгому лістингу ознакою спеціального символного і блочного пристрій є символи **c** і **b** в першій позиції першого стовпця, відповідно:

```
$ cd /devices/pci\@0,0/pci-ide\@7,1/ide\@0  
$ ls -l | more
```

```

total 0

crw----- 1 root      sys        77,   0 Фев 14 14:03 nv@0,0:0
brw-r---- 1 root      sys        29,   0 Апр 20 2001 sd@0,0:a
crw-r---- 1 root      sys        29,   0 Апр 20 2001 sd@0,0:a,raw
brw-r---- 1 root      sys        29,   1 Апр 20 2001 sd@0,0:b
crw-r---- 1 root      sys        29,   1 Апр 20 2001 sd@0,0:b,raw
brw-r---- 1 root      sys        29,   2 Апр 20 2001 sd@0,0:c
crw-r---- 1 root      sys        29,   2 Апр 20 2001 sd@0,0:c,raw

...

```

Спеціальні файли пристрійв створюються командою **mknod** :

mknod имя b главній_номер второстепенній_номер

mknod имя c главній_номер второстепенній_номер

Головний номер пристрою задає *драйвер* (індекс в таблиці драйверів системи), або *тип* пристрою, а другорядне - *примірник* пристрою.

Створювати спеціальні файли пристрійв звичайно може тільки користувач **root**. Ось як можна створити новий спеціальний файл пристрою для одного з представлених у лістингу вище пристрійв:

```

# mknod slice1 b 29 1
# ls -l slice1
brw-r---- 1 root      sys        29,   1 Мар 25 2001 slice1

```

FIFO – іменований канал. Цей файл використовується для зв'язку між процесами 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:

```

$ find / -type p -print 2>/dev/null
/var/spool/lp/fifos/FIFO
/etc/cron. d/FIFO
/etc/saf/zsmon/_pmpipe
/etc/saf/_sacpipe
/etc/saf/_cmdpipe
/etc/initpipe

```

```
/etc/utmppipe
^C
$ ls -l /etc/cron. d/FIFO
prw----- 1 root      root          0 Фев 23 15:41 /etc/cron. d/FIFO
```

Іменовані канали створюються командою `mknod`:

`mknod имя р`

Наприклад:

```
$ mknod p1 p
$ ls -l p* >p1 & cat p1
[2] 22380
prw-r--r-- 1 spot    50          0 Mar 26 15:17 p1
-rw-rw-r-- 1 root    sys         8296 фев 23 15:39 ps_data
[2]- Done                               ls -l p* >p1
$
$
```

Посилання. Каталог містить імена файлів і покажчики на їх метадані. Така архітектура дозволяє одному файлу мати декілька імен у файловій 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 у файловій системі.

Стандарт POSIX (*Portable Operating System Interface*) вимагає реалізувати підтримку двох типів посилань – жорстких і символічних. *Жорстким посиланням (hard link)* вважається елемент каталогу, який вказує безпосередньо на деякий *індексний дескриптор*. Жорсткі посилання дуже ефективні, але у них існують певні обмеження, так як вони можуть створюватися лише в межах однієї фізичної файлової системи. Коли створюється таке посилання, то файл, на який воно вказує, повинен вже існувати. Крім того, каталоги не можуть зв'язуватися жорстким посиланням.

Символічне посилання (symbolic link) - це спеціальний файл, який містить шлях до іншого файла. Вказівка на те, що даний елемент каталогу є символічним посиланням, знаходиться в індексному дескрипторі. Тому звичайні команди доступу до файла замість отримання даних з фізичного файла, беруть їх з файла, ім'я якого наведено у посиланні. Цей шлях може вказувати на що завгодно: це може бути ката-

лог, він може навіть знаходитися в іншій фізичній файлової системі, більше того, зазначеного файлу може взагалі не бути.

Деякі системи накладають обмеження на кількість символічних посилань. POSIX вимагає, щоб їх підтримувалося не менше 20, але дійсне значення залежить від конкретної реалізації. Звичайно, в описі шляху можна використовувати поєднання символічних і жорстких посилань.

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

```
$ ls >f2. txt
$ ln f3. txt f2. txt
ln: cannot access f3. txt
$ ln f2. txt f3. txt
$ ls -l f?. txt
-rw-r--r--    1 spot    50          0 Мар 26 14:40 f1.txt
-rw-r--r--    2 spot    50        643 Мар 26 15:37 f2. txt
-rw-r--r--    2 spot    50        643 Мар 26 15:37 f3. txt
$
```

У цьому прикладі створено текстовий файл із лістингом поточного каталогу, а потім створено на нього жорстке посилання. Для цього використовується команда `ln` з таким синтаксисом:

```
ln [ -fns ] ісходний_файл [ мета ]
ln [ -fns ] ісходний_файл. . . мета
```

Якщо в якості мети зазначено неіснуючий файл, або файл, який не є каталогом, використовується перша форма. При цьому кількість операндів має бути не більше двох. В результаті виконання створюється жорстке (за замовчуванням) або символічне (якщо вказана опція `-s`) посилання із заданим ім'ям **мета**. Якщо файл з таким ім'ям вже існує, він перезаписується. При виклику з одним аргументом створюється посилання на вказаний **ісходний_файл** з таким же ім'ям в поточному каталозі.

Якщо **мета** задає існуючий каталог, створюється посилання з таким же ім'ям в цьому каталозі. При наявності більше двох аргументів використовується друга форма команди, причому **мета** повинна посилатися на існуючий каталог.

Опції **-f** та **-n** вимагають, відповідно, примусово створити посилання або не створювати його, якщо **мета** задає існуючий файл.

Зверніть увагу, що перший аргумент команди **ln** повинен вказувати існуючий файл або каталог.

У довгому лістингу ознакою символічного посилання є символ **l** в першій позиції первого стовпчика. Розглянемо простий приклад створення символічної зв'язку:

```
$ ln -s f2 f4
$ ls -l f*
-rw-r--r--  1 spot  50          0 Mar 26 14:40 f1.txt
-rw-r--r--  2 spot  50          643 Mar 26 15:37 f2. txt
-rw-r--r--  2 spot  50          643 Mar 26 15:37 f3. txt
lrwxrwxrwx  1 spot  50          2 Mar 26 15:57 f4 -> f2
```

Сокети дозволяють представити у вигляді файлу в логічній файловій системі мережеве з'єднання. Створення сокетів виходить за межі даного курсу, хоча зрозуміло, що для цього ядро пропонує відповідний системний виклик.

У довгому лістингу ознакою сокета є символ **s** в першій позиції первого стовпчика. Ось які сокети можна знайти в Linux:

```
$ find / -type s -print 2>/dev/null
/var/spool/prngd/pool
/tmp/.x11-unix/X0
^C
$ ls -l /var/spool/prngd/pool
srwxrwxrwx  1 root      other      0 Mar 14 11:25 /var/spool/prngd/pool
```

3.1.6. Визначення типу файлу

Для більш точного визначення типу файлу (наприклад, якщо файл двійковий, якою програмою він міг бути створений) використовується команда **file** з таким синтаксисом:

```
file [ -h ] [ -m файл_сигнатур ] [ -f файл_списка ] файл. ...
file [ -h ] [ -m файл_сигнатур ] -f файл_списка
file -c [ -m файл_сигнатур ]
```

Утиліта **file** виконує ряд перевірок кожного з вказаних файлів і всіх файлів, зазначених у параметрі **файл_спіска**, якщо він заданий, намагаючись класифікувати

файли. Якщо файл не є звичайним, видається його тип. Якщо ж звичайний файл має нульову довжину, він класифікується як порожній (**empty**).

Якщо файл є текстовим, команда **file** перевіряє перших 512 байтів і намагається визначити, на якій мові програмування написаний файл. Якщо файл є символьчною зв'язком, відбувається перевірка і класифікація файлу, на який зв'язок вказує.

При визначенні типу файлу використовується файл сигнатур. Стандартний файл сигнатур - **/etc/magic**. У ньому зберігаються числа або рядки, що показують тип файлу:

0	string	PK\003\004	ZIP archive
0	string	MZ	DOS executable (EXE)
0	string	LZ	DOS built-in
0	byte	0xe9	DOS executable (COM)
0	byte	0xeb	DOS executable (COM)
24	long	60012	ufsdump archive file
0	string	TZif	zoneinfo timezone data file

Формат файлу сигнатур детально описаний на сторінці довідкового посібника **magic(4)**.

Якщо перевіряється файл не існує, не може бути прочитаний або його тип не вдається визначити, це не вважається помилкою. Результат тестування командою **file** не гарантує 100% коректності. Не покладайтеся на нього з повною впевненістю.

Підтримуються такі опції: **-C** - перевіряти формат файлу сигнатур; **-H** - не слідувати за символічними посиланнями; **-F** - визначає файл, що містить список файлів для класифікації; **-M** - Визначає альтернативний файл сигнатур, замість **/etc/magic**

Наведемо простий приклад:

```
$ file -f f*
f2.txt:      ascii text
f3.txt:      ascii text
f4:          symbolic link to f2
```

3.1.7. Основні команди для роботи з файлами

До основних операцій для роботи з файлами, крім створення і перегляду характеристик, можна віднести копіювання, видалення, переміщення і перейменування, а також перегляд вмісту. Команди для виконання цих дій подані в наступних підрозділах.

Копіювання файлів. Команда **cp** копіє вихідний файл у цільовий файл або каталог. Вона має такий синтаксис:

```
cp [-r] вихідний цільовий  
cp [-r] [-r] вихідний_1 ... каталог
```

Вихідний файл не повинен співпадати з цільовим. Якщо цільовий файл є каталогом, то вихідні файли копіюються в нього під тими ж іменами. Тільки в цьому випадку можна вказувати декілька вихідних файлів. Якщо цільовий файл існує і не є каталогом, його старе вміст втрачається. Права доступу, власник і група цільового файлу при цьому не змінюються.

Якщо цільовий файл не існує або є каталогом, нові файли створюються з тими ж правами доступу, що і вихідні. Час останньої зміни цільового файлу (останнього доступу, якщо він не існував), а також час останнього доступу до вихідних файлів встановлюється рівним часу копіювання. Якщо цільовий файл був зв'язком на інший файл, всі зв'язки зберігаються, а вміст файлу змінюється.

Команда **cp** підтримує такі основні опції: **-P** – зберігати інформацію про власника, по можливості - права доступу й часи доступу для нового файлу; **-R** – копіювати рекурсивно, включаючи підкаталоги. Два дефіса (**-**) дозволяють явно вказати кінець опцій командного рядка, що дає можливість команді **cp** працювати з іменами файлів, що починаються з дефіса (**-**). Якщо в одній командному рядку вказані **-** та **-**, другий дефіс буде інтерпретуватися як ім'я файлу.

Розглянемо ряд простих прикладів копіювання. Ось як, незалежно від типу, копіюється в каталог декілька файлів:

```
$ mkdir d1  
$ rm f*  
$ ls >f1.txt  
$ ln f1.txt f2.txt  
$ ln -s f1.txt f3.txt  
$ cp f1.txt f2.txt f3.txt d1  
$ ls -l d1  
total 24  
-rw-r--r--    1 spot   50          639 Mar 26 16:54 f1. txt
```

```
-rw-r--r--    1 spot  50          639 Map 26 16:54 f2. txt
-rw-r--r--    1 spot  50          639 Map 26 16:54 f3. txt
```

Приклад звичайного копіювання файлів "один в один":

```
$ cp f1.txt f5.txt
$ ls -l f*.txt
-rw-r--r--    2 spot  50          639 Map 26 16:54 f1. txt
-rw-r--r--    2 spot  50          639 Map 26 16:54 f2. txt
lrwxrwxrwx    1 spot  50          6 Map 26 16:54 f3. txt -> f1.txt
-rw-r--r--    1 spot  50          639 Map 26 16:55 f5. txt
```

Рекурсивне копіювання:

```
$ cp -r d1 d2
$ ls -l d2
total 24
-rw-r--r--    1 spot  50          639 Map 26 16:56 f1.txt
-rw-r--r--    1 spot  50          639 Map 26 16:56 f2.txt
-rw-r--r--    1 spot  50          639 Map 26 16:56 f3.txt
```

Копіювання файлів, імена яких починаються з дефіса:

```
$ ls > -1
$ cp -1 1.txt
cp: illegal option -- 1
cp: Insufficient arguments (1)
Usage: cp [-f] [-i] [-p] f1 f2
        cp [-f] [-i] [-p] f1. . . fn d1
        cp -r|R [-f] [-i] [-p] d1. . . dn-1 dn
$ cp -- -1 1.txt
$ ls -l ?.txt
-rw-r--r--    1 spot  50          666 Map 26 16:58 1. txt
```

Видалення файлів. Для видалення файлів використовується команда **rm** з таким синтаксисом:

```
rm [ -firR ] файл ...
```

При цьому відбувається видалення запису файлу з відповідного каталогу та зменшення на 1 кількості зв'язків в індексному дескрипторі. Якщо кількість зв'язків в результаті стає рівним 0, файл знищується (після його закриття усіма відкрили процесами) - відповідний індексний дескриптор стає вільним, і блоки даних файлу також звільняються.

Щоб видалити файл, користувач повинен мати право запису у відповідний каталог. Якщо немає права на запис у файл і вхідний потік пов'язаний з терміналом, на термінал видаються (у вісімковому вигляді) права доступу до файлу і запитується підтвердження; якщо введений відповідь **у** - файл видаляється, інакше - ні.

Команда **rm** сприймає такі основні опції: **-F** - видаляти без запитів підтвердження всі файли, незалежно від прав доступу до них, якщо є право запису для каталогу; **-I** - чекати підтвердження, перш ніж видалити файл (скасовує дію опції **-F**; діє навіть тоді, коли стандартний вхідний потік не пов'язаний з терміналом); **-r** - рекурсивне видалення з підкаталогами, в тому числі, не пустими; **-R** - те ж саме, що й опція **-r**.

Команда **rm** без опцій рекурсивного видалення не видаляє каталоги. Для видалення *порожніх* каталогів призначена команда **rmdir**. Якщо в каталогі є інші файли, крім посилань на поточний і батьківський каталог, команда **rmdir** його не видаляє. Ця команда має такий синтаксис:

```
rmdir [-p][-s] каталог ...
```

Команда **rmdir** сприймає такі опції: **-p** - дозволяє видалити каталог та його батьківські каталоги, якщо вони порожні (у стандартний вихідний потік видається повідомлення про видалення всіх зазначених каталогів або про збереження частини з них з яких-небудь причин); **-s** - припиняє видачу повідомлень при використанні опції **-p**.

Наведемо ще приклади видалення файлів і каталогів:

```
$ ls f* d*
dogovor_trg. sql  f1.txt          f3.txt
dtdbcache_:0      f2.txt          f5.txt
d1:
f1. txt  f2.txt  f3.txt
```

```

d2:
f1. txt  f2.txt  f3.txt
$ rm -r d1
$ rm f1.txt f2.txt
$ ls -l f*
lrwxrwxrwx    1 spot   50          6 Mar 26 16:54 f3.txt -> f1.txt
-rw-r--r--    1 spot   50          639 Mar 26 16:55 f5.txt
$ mkdir d2/d3
$ rm d2/*
rm: d2/d3 is a directory
$ ls -l d2
total 8
drwxr-xr-x    2 spot   50          69 Mar 26 17:24 d3
$ rmdir -p d2/d3
$ ls -l d2
d2: No such file or directory

```

Переміщення і перейменування файлів. Команда **mv** переміщує (перейменовує) вихідний файл (або файли) у цільовий файл (або папку). Вона має такий синтаксис:

```

mv [-f][-i] вихідний_файл цільовий_файл
mv [-f][-i] вихідний_файл_1 ... каталог

```

Ім'я вихідного файлу не повинне збігатися з ім'ям цільового файлу. Якщо цільовий файл є каталогом, то вихідні файли переміщаються в нього під тими ж іменами. Тільки в цьому випадку можна вказувати декілька вихідних файлів. Якщо цільовий файл існує і не є каталогом, його старе вміст втрачається. Якщо при цьому виявляється, що в цільовий файл не дозволена запис, то виводиться інформація про права доступу до цього файлу і з терміналу запитується підтвердження його перезапису.

Для переміщення файлу необхідно мати права на запис у вихідному і цільовому каталогі.

Команда **mv** підтримує такі опції: **-F** – примусове переміщення (якщо цільовий файл вже існує, то він віддаляється); **-I** – запитувати підтвердження видалення існуючого файла.

Розглянемо приклади:

```

$ ls -l f*
lrwxrwxrwx    1 spot   50          6 Mar 26 16:54 f3.txt -> f1.txt
-rw-r--r--    1 spot   50          639 Mar 26 16:55 f5.txt

$ mv f5.txt f4.txt
$ mv f4.txt f4.txt
mv: f4.txt and f4.txt are identical
$ mv f4.txt f3.txt
$ ls -l f*
-rw-r--r--    1 spot   50          639 Mar 26 16:55 f3.txt

$ mkdir d1
$ mv f3.txt d1
$ ls -l d1
total 8
-rw-r--r--    1 spot   50          639 Mar 26 16:55 f3.txt

```

Перегляд вмісту файлів. Стандартним засобом перегляду вмісту файлів (крім редакторів або команд типу **od**), є команда **cat**. Вона читає файли з командного рядка в заданій послідовності і поміщає їх вміст у стандартний вихідний потік. Команда **cat** має такий синтаксис:

cat [-u][-s][-v][-t][-e] [файл ...]

Якщо жоден файл не зазначено або зазначено символ дефісу (-), то команда читає стандартний вхідний потік. Команда **cat** – це корисний інструмент для *конкатенації* декількох файлів.

Команда **cat** сприймає такі основні опції: **-u** – вивід не буферизується (за замовчуванням - буферизується); **-s** – про неіснуючі файли не повідомляється; **-v** – візуалізація недрукованих символів (крім табуляції, перевода рядка й переходу до нової сторінки). Керуючі символи зображуються у вигляді ^X (CTRL + X); символ **DEL** (вісімкове 0177) - у вигляді ^?. Символи, що не входять до набору ASCII (тобто з ненульовим восьмим бітом) виводяться у вигляді **Mx**, де символ **x** визначається молодшими сім'ю бітами символ.

Розглянемо декілька прикладів використання команди **cat**:

```

$ ls *.txt > 1.txt
$ cat 1.txt

```

```

1. txt
$ cp 1. txt 2. txt
$ cat 1. txt 2. txt > 3. txt
$ ls -l *. txt
-rw-r--r-- 1 spot 50          6 Mar 26 17:55 1. txt
-rw-r--r-- 1 spot 50          6 Mar 26 17:55 2. txt
-rw-r--r-- 1 spot 50         12 Mar 26 17:56 3. txt
$ cat 3. txt
1. txt
1. txt
$ cat >4. txt
Hello!
^D
$ cat 4. txt
Hello!

```

Права доступу до файлів. Кожен користувач UNIX (не кажучи вже про системний адміністратора) повинен управляти дисковим простором. Користувач несе відповідальність за зміст свого початкового каталогу та забезпечення цілісності будь наявних у нього даних. Цілісність даних забезпечується перевіркою і зміною *прав доступу*. Захищаючи файли і каталоги, користувач запобігає неавторизованій доступ.

Кожен файл в ОС UNIX містить набір прав доступу, за яким визначається, як користувач взаємодіє з цим файлом. Цей набір зберігається в індексному дескрипторі даного файлу у вигляді цілого значення, з якого зазвичай використовується 12 бітів. Причому кожен біт використовується як перемикач, дозволяючи (значення 1) чи забороняючи (значення 0) той чи інший доступ.

Три перших біта встановлюють різні види поведінки при виконанні. Решта дев'ять діляться на три групи по три, визначаючи права доступу для власника, групи та інших користувачів. Кожна група задає права на читання, запис і виконання.

Базові біти прав доступу представлені в табл. 11. Там дано вісімкове значення, що задає відповідний біт, вигляд цього біта в першому стовпці довгого лістингу та право, що задається цим бітом.

Таблиця 11. Права доступу до файлів в ОС UNIX

Вісімкове значення	Вид у стовпці прав доступу	Право або призначення біта
4000	---S---	Встановлений ефективний ідентифікатор власника (біт SUID)
2000	----- S -----	Встановлений ефективний ідентифікатор групи (біт SGID)
1000	----- T	Клейкий (<i>sticky</i>) біт. Вид для каталогів і виконуваних файлів, відповідно.
0400	-R-----	Право власника на читання
0200	-W-----	Право власника на запис
0100	---X-----	Право власника на виконання
0040	----R----	Право групи на читання
0020	----W----	Право групи на запис
0010	-----X--	Право групи на виконання
0004	-----R--	Право всіх інших на читання
0002	-----W-	Право всіх інших на запис
0001	-----X	Право всіх інших на виконання

Біт читання для всіх типів файлів має одне і те ж значення: він дозволяє читати вміст файлу (отримувати лістинг каталогу командою **ls**).

Біт запису також має одне і те ж значення: він дозволяє писати в цей файл, включаючи і перезапис вмісту. Якщо у користувача відсутнє право доступу на запис в каталозі, де знаходиться даний файл, то користувач не зможе його видалити. Аналогічно, без цього ж права користувач не створить новий файл у каталозі, хоча може скоротити довжину доступного на запис файлу до нуля.

Якщо для деякого файла 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**).

Встановлений біт SUID означає, що доступний користувачеві на виконання файл буде виконуватися з правами (з *ефективним ідентифікатором*) власника, а не користувача, який викликав файл (як це зазвичай відбувається).

Примітка: Біт **s** є могутнім і небезпечним. Якщо користувач напише будь-яку програму, встановить для неї біт **s** командою `chmod`, а потім передасть її надкористувачу командою `chown`, його програма отримає право робити, що хоче. Для боротьби з цією небезпекою існують різні методи. У системі BSD передавати файл від користувача до користувача або від групи до групи може тільки суперкористувач. У системі System V передати свій файл іншому користувачеві може будь-який, але при цьому біт **s** скидається.

Встановлений біт SGID означає, що доступний користувачеві на виконання файл буде виконуватися з правами (з *ефективним ідентифікатором*) групи-власника, а не користувача (як це зазвичай відбувається). Якщо біт SGID встановлений для файлу, не доступного для виконання, він означає обов'язкове блокування, тобто незмінність прав доступу на читання і запис, поки файл відкритий певною програмою.

Встановлений клейкий біт для звичайних файлів раніше (за часів PDP-11) означав необхідність зберегти образ програми в пам'яті після виконання (для прискорення повторного завантаження). Зараз, при установці звичайним користувачем, він скидається. Задання 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 таких випадках:

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

Для розрахунку прав доступу необхідно скласти вісімкові значення всіх необхідних встановлених бітів. У результаті вийде чотиризначне вісімкове число. Якщо старший розряд має значення 0, його можна не вказувати.

Наприклад, якщо необхідно задати права доступу на читання, запис і виконання для власника, на читання і виконання для групи і на виконання для всіх інших користувачів, отримуємо таке вісімкове значення:

Читання для власника: **0400**

Запис для власника: **0200**

Виконання для власника: **0100**

Читання для групи: **0040**

Виконання для групи: **0010**

Виконання для інших: **0001**

Сума: **0751**

Отже, відповідні права доступу - **751**. У довгому лістингу ці права будуть представлені у вигляді "**-rwxr-x-x**" (при "складання" букви з дефісом в символному поданні залишається буква).

Зміна прав доступу до файлу. Для установки (zmіни) прав доступу до файлу використовується команда **chmod**. Вона має такий синтаксис:

chmod [-fR] абсолютні_права файл ...

chmod [-fR] символьна_zmіна_прав файл ...

Команда **chmod** встановлює права доступу до зазначених файлів. Права доступу до файлу може змінювати або встановлювати тільки його власник чи користувач **root**. Опція **-f** означає, що команда не буде повідомляти про неможливість встановлення прав доступу. Опція **-R** означає, що задане зміна прав доступу буде застосовуватися рекурсивно для всіх підкаталогів, зазначених у списку файлів.

Абсолютні права доступу задаються восьмеричним числом, розрахунок якого (відповідно до табл. 11) описаний у попередньому розділі. Опису синтаксису, використовуваного для завдання символного зміни прав доступу, присвячений наступний підрозділ.

Символьне представлення зміни прав доступу задається у вигляді списку, через кому, виразів такого виду:

[Користувачі] оператор [права]

Компонент користувачі визначає, для кого задаються або змінюються права. Він може мати значення **u**, **g**, **o** та **a**, що задають зміни прав для власника, групи, інших користувачів і всіх категорій користувачів. Якщо користувачі не вказані, права змінюються для всіх категорій користувачів. Однак при цьому не перевизначаються установки, що задаються маскою створення файлів (**umask**).

Компонент **оператор** може мати значення **+**, **-** або **=**, означають додавання, скасування права доступу і установку в точності зазначених прав, відповідно. Якщо після оператора **=** права не вказані, всі права доступу для відповідних категорій користувачів скасовуються.

Компонент **права** поставив у вигляді будь сумісної комбінації таких символів:

- r** право на читання
- w** право на запис
- x** право на виконання
- l** блокування зміни прав доступу
- s** виконання з ефективним ідентифікатором власника або групи-власника
- t** клейкий біт

Не всі поєдання символів для компонента користувачі та компонента права допустимі. Так, **s** можна задавати тільки для **u** або **g**, а **t** - тільки для **u**. Права **x i s** не сумісні з **l** і т. д. Зміни прав доступу в списку виконуються послідовно, в порядку їх перерахування.

Розглянемо приклад зміни прав доступу:

```
$ cd /tmp
$ echo "Hello" >f1.txt
$ chmod +w f1.txt
$ ls -l *.txt
-rw-r--r--  1 spot  50          0 May 27 10:52 f1.txt
$ chmod a+w f1.txt
$ ls -l *.txt
-rw-rw-rw-  1 spot  50          0 May 27 10:52 f1.txt
$ chmod u+x,g=x,o= f1.txt
```

```

$ ls -l *.txt
-rwx--x---    1 spot   50          0 Map 27 10:52 f1.txt

$ chmod ug-x,og+r,u=rwx f1.txt
$ ls -l *.txt
-rwxr--r--    1 spot   50          0 Map 27 10:52 f1.txt

$ chmod 644 f1.txt
$ ls -l *.txt
-rw-r--r--    1 spot   50          0 Map 27 10:52 f1.txt

```

Розглянемо ще один приклад, що показує значення і зміни прав доступу до каталогу:

```

$ ls -l | grep d1
drw-r--r--    2 spot   50          108 Map 26 17:39 d1

$ cd d1
bash: cd: d1: Permission denied

$ chmod 744 d1
$ cd d1
$ cd ..
$ chmod -w d1
$ cd d1
$ ls
f3. txt
$ rm f3.txt
rm: f3.txt not removed: Permission denied

```

Установка режиму створення файлу. Новий файл створюється з правами доступу, обумовленими користувача маскою режиму створення файлів. Команда **umask** (вбудована команда інтерпретатора) присвоює користувача масці режиму створення файлів вказане вісімкове значення. Три вісімкові цифри відповідають прав на читання / запис / виконання для власника, членів групи та інших користувачів, відповідно.

Команда **umask** має такий синтаксис:

```
umask [ -S ] [ маска ]
```

Якщо параметри не зазначені, команда **umask** видає поточне значення маски. За замовчуванням, значення видається і задається у вісімковому вигляді як число,

яке необхідно "відняти" з максимальних прав доступу (777 для виконуваних файлів, які створюються компіляторами, і 666 для звичайних файлів):

```
$ umask
```

```
022
```

При такій масці звичайні текстові файли будуть створюватися з правами **666 - 022 = 644**:

```
$ >f5.txt
```

```
$ ls -l f5*
```

```
-rw-r--r-- 1 spot 50 0 Mar 27 11:33 f5.txt
```

Операція "вирахування" для значення маски формально виконується як побітове логічне **I** доповнення маски і максимальних прав доступу. Розглянемо приклад розрахунку:

Двійкове значення маски: **000010010**

Доповнення маски: **111101101**

Максимальне значення прав: **110110110**

Логічне **I** попередніх двох рядків: **110100100**

Результатуючі біти прав: **110100100 (644)**

Для виконуваних файлів, що створюються, наприклад, компілятором мови С:

Двійкове значення маски: **000010010**

Доповнення маски: **111101101**

Максимальне значення прав: **111111111**

Логічне **I** попередніх двох рядків: **111101101**

Результатуючі біти прав: **111101101 (755)**

Опція **-S** вимагає видавати маску в символному вигляді, показуючи, які біти прав доступу будуть встановлені у створюваного файлу (також з урахуванням того, чи створюється виконуваний або звичайний текстовий файл):

```
$ umask -S
```

```
u=rwx,g=rx,o=rx
```

Команду **umask** доцільно включити у файли початкового запуску, що задають середовище для початкового командного інтерпретатора.

Розглянемо ще один приклад створення файлу при іншому значенні маски:

```
$ umask 257  
$ umask -S  
u=rw,g=wr,o=  
$ >f6. txt  
$ ls -l f6*  
-r--w---- 1 spot 50 0 Mar 27 11:41 f6.txt
```

Зміна принадлежності файлу. Власник файлу, а також користувач **root** може змінювати власника і групу-власника файлу. Для зміни власника (і групи-власника) файлу використовується команда **chown** з таким синтаксисом:

```
chown [-h] [-R] власник[:группа] файл ...
```

Опція **-h** вимагає змінювати власника файлу, на який вказує символічний зв'язок, а не самого зв'язку, як відбувається за умовчанням. Опція **-R** вимагає рекурсивно змінити власника у всіх підкаталогах.

Для зміни тільки групи, що володіє файлом, використовується команда **chgrp**:

```
chgrp [-h] [-R] група файл ...
```

Її опції аналогічні команді **chown**.

Врахуйте, що після передачі файлу іншому власнику, початковий власник перестає ним володіти, і буде мати права доступу, встановлені новим власником.

Розглянемо простий приклад:

```
$ ls -l  
total 2  
-rw-r--r-- 1 user01 others 6 Dec 10 16:19 testfile  
$ chown informix testfile  
$ ls -l  
total 2  
-rw-r--r-- 1 informix others 6 Dec 10 16:19 testfile  
$ logname
```

```
user01
$ chown user01 testfile
UX:chown: ERROR: testfile: Not privileged
```

Пошук файлів. У логічній файловій системі ОС UNIX - тисячі файлів, тому необхідні засоби пошуку файлів за різними критеріями. Для пошуку файлів призначена команда **find** з таким синтаксисом:

find каталог_1 ... вираз

Утиліта **find** переглядає ієархії каталогів у пошуках файлів, що задовольняють критерію, що задається виразом. Вираження будуються з елементів з використанням таких конструкцій:

(Елемент) Істинно, якщо істинний елемент у дужках (оскільки дужки - метасимвол командного інтерпретатора, їх треба екраниувати). Дужки використовуються для групування елементів.

!елемент Істинно, якщо елемент не правдивий.

елемент [-a] Істинно, якщо істинні обидва елементи. Якщо елементи просто перераховані поспіль, передбачається ця ж логічна операція I.

елемент-о Істинно, якщо істинний хоча б один елемент.
елемент

Імена знайдених (задовольняють критерію, що задається виразом) файлів за замовчуванням видаються в стандартний вихідний потік.

В якості елементів вираження використовуються основні конструкції, представлені в табл. 12. Вираз перевіряється зліва направо, з урахуванням дужок.

Таблиця 12. Основні елементи вираження в команді **find**

Елемент	Призначення або критерій істинності
-name шаблон	Істина, якщо ім'я файлу відповідає шаблоном. При ви-

Таблиця 12. Основні елементи вираження в команді `find`

Елемент	Призначення або критерій істинності
	користанні метасимволів необхідно маскувати шаблони від командного інтерпретатора.
-type тип	Істина, якщо файл - зазначеного типу. Типи файлів за-даються символами b , c , d , f , l , p та s , що познача-ють, відповідно, спеціальний блоковий пристрій, спеці-альне символьне пристрій, каталог, звичайний файл, символьну зв'язок, іменований канал і сокет.
-user користувач	Істина, якщо файл належить користувачу , вказаному за ідентифікатором або реєстраційному імені.
-group група	Істина, якщо файл належить групі , зазначеної за іден-тифікатором або імені.
-perm [-] права	Якщо дефіс не заданий, то правдивий тільки якщо права доступу в точності відповідають зазначеним (як в ко-манді chmod , простіше - абсолютні). Якщо задано де-фіс, істина, якщо в правах доступу файлу, як мінімум, встановлені ті ж біти, що і у вказаних правах.
-size n [c]	Істина, якщо файл має довжину n блоків (блок - 512 байтів) або символів (якщо зазначений суфікс c). Пе-ред розміром можна вказувати префікс + (не менше), - (не більше) або = (в точності дорівнює).
-atime n	Істина, якщо до файлу останній раз зверталися n днів то-му. Перед n в елементах -atime , -ctime та -mtime можна вказувати префікс +(не пізніше), -(не раніше) або =(рівно).
-ctime n	Істина, якщо файл створений n днів тому.
-mtime n	Істина, якщо файл був змінений n днів тому.
-newer файл	Істина, якщо файл - більш новий, ніж зазначений.

Таблиця 12. Основні елементи вираження в команді `find`

Елемент	Призначення або критерій істинності
-ls	Завжди правдивий. Видає інформацію про фото, аналогічну довгому лістингу.
-print	Правдивий завжди. Видає повне ім'я файлу в стандартний вихідний потік.
-exec команда {} \;	Істина, якщо виконана команда повертає код повернення 0. Команда закінчується замаскованої крапкою з комою. У команді можна використовувати конструкцію {}, замінюючи повним ім'ям розглянутого файлу.
-ok команда {} \;	Аналогічний exec , але отримана після підстановки імені файла замість {} команда видається зі знаком питання і виконується тільки якщо користувач ввів символ у .
-depth	Правдивий завжди. Вимагає так обходити ієархію каталогів, щоб файли будь-якого каталогу завжди оброблялися раніше, ніж сам каталог (обхід "в глибину").
-prune	Правдивий завжди. Вимагає не перевіряти файли в каталозі, можна порівняти з попереднім елементом вираження. Не діє, якщо раніше вказаний елемент -depth .

У різних версіях ОС UNIX можуть підтримуватися й інші компоненти виразів у команді **find**. Якщо командний рядок сформована неправильно, команда негайно завершує роботу.

Розглянемо декілька прикладів використання команди **find**:

```
$ find . -user spot -size +0c -ls
find: cannot read dir. /smc898: Permission denied
475898122  4 -rw-r--r--  1 spot   50          666 Mar 26 16:58. /-1
473866040  4 -rw-r--r--  1 spot   50           6 Mar 26 17:55. /1.txt
475472259  4 dr-xr--r--  2 spot   50          108 Mar 26 17:39. /d1
```

```

474199552    4 -rw-r--r--  1 spot   50          639 Mar 26 16:55. /d1/f3.txt
476732956    4 -rw-r--r--  1 spot   50          6 Mar 26 17:55. /2.txt
476732980    4 -rw-r--r--  1 spot   50          12 Mar 26 17:56. /3.txt
476142563    4 -rw-r--r--  1 spot   50          7 Mar 26 17:56. /4.txt
$ find . -name "???.txt" -print
find: cannot read dir. /smc898: Permission denied
. /d1/f3.txt
. /f1.txt
$ find . -name d1 -prune -name "???.txt" -print
find: cannot read dir. /smc898: Permission denied
$ find . -name d1 -prune -o -name "???.txt" -print
find: cannot read dir. /smc898: Permission denied
. /f1. txt
$ find . -user spot -ok rm {} \;
find: cannot read dir. /smc898: Permission denied
< rm. . . /p1 >?    y
< rm. . . /-1 >?    y
< rm. . . /1.txt >?    y
< rm. . . /mpDfa4ZT >?    y
< rm. . . /d1 >?    y
rm: Unable to remove directory. /d1: File exists
< rm. . . /d1/f3.txt >?    y
< rm. . . /2.txt >?    y
< rm. . . /3.txt >?    y
< rm. . . /4.txt >?    y
< rm. . . /f1.txt >?    y
$ find . -user spot -print
find: cannot read dir. /smc898: Permission denied
. /d1
. /d1/f3.txt

```

3.2. Структура і властивості файлових систем

3.2.1. Основні каталоги і їх призначення

Використання загальноприйнятих імен основних файлів і структури каталогів істотно полегшує роботу в операційній системі, її адміністрування і підвищує переваги.

носимість. Типова структура і призначення каталогів файлової системи UNIX представлена в табл. 13.

Таблиця 13. Основні каталоги логічної файлової системи UNIX

Каталог	Призначення і зміст
/	Кореневий каталог є основою будь-якої файлової системи UNIX. Всі інші каталоги та 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 <i>окрема</i> фізична файлова система, а сам він є точкою її монтування, про що свідчить наявність підкаталогу lost+found .
/bin	Користувальницькі виконувані програми. Зараз зазвичай є символічною зв'язком, що вказує на /usr/bin.
/dev	Каталог для спеціальних файлів пристройів. Може мати підкаталоги для різних класів і типів пристройів, наприклад, dsk , rdsk , rmt , inet (у SVR4).
/etc	Каталог для конфігураційних файлів. Може мати підкаталоги для різних компонентів і служб. Файли в UNIX - звичайні текстові.
/home	Каталог для розміщення початкових каталогів користувачів. Часто є точкою монтування окремої фізичної файлової системи.
/lib	Каталог для бібліотек. Зараз зазвичай є символічним посиланням, що вказує на /usr/lib.
/lost+found	Підкаталог, який може бути в кожному каталозі, що є точкою монтування фізичної файлової системи на диску. Кореневий каталог завжди представлений окремою фізичною файловою системою, повинен бути завжди доступний, і монтується автоматично при запуску системи. Всі інші фізичні файлові системи формально не потрібні для функціонування ОС UNIX.
/mnt	Точка монтування для файлових систем на знімних носіях або додаткових дисках. Може містити підкаталоги для окремих типів носіїв, наприклад, cdrom або floppy . Може бути порожнім.

Таблиця 13. Основні каталоги логічної файлової системи UNIX

Каталог	Призначення і зміст
/opt	Каталог для додаткового комерційного програмного забезпечення. Може бути порожнім або відсутнім (в BSD-системах).
/proc	Каталог псевдо-файльової системи, що представляє у вигляді каталогів і файлів інформацію про ядро, пам'яті і процесах, які працюють в системі.
/sbin	Каталог для системних виконуваних програм, необхідних для вирішення завдань системного адміністрування.
/tmp	Каталог для тимчасових файлів. Має встановлений <i>клейкий біт</i> і доступний для запису і читання всім користувачам. Зазвичай створюється у вигляді окремої фізичної файлової системи, в тому числі, у віртуальній пам'яті.
/usr	У цьому каталозі знаходяться виконувані програми, бібліотеки, заголовні файли, довідкові посібники (/usr/share/man), вихідні тексти ядра і утиліт системи (Linux), що ростуть файли і черги друку (/usr/spool в BSD-системах) та т . д. Часто каталог є точкою монтування окремої фізичної файлової системи. Нижче представлені основні його підкаталоги.
/usr/bin	Основні виконувані програми і утиліти.
/usr/include	Заголовки бібліотек. Може містити підкаталоги.
/usr/lib	Статичні й динамічні компонуючі бібліотеки. Може містити підкаталоги.
/usr/local	Каталог для додаткового <i>вільно поширюваного програмного забезпечення</i> (GNU). Містить структуру підкаталогів, аналогічну кореневого каталога (bin , etc , include , lib і т.і.)
/var	У UNIX System V і Linux цей каталог є замінником каталогу (/usr/spool), який використовується для зберігання зростаючих файлів різних сервісних підсистем, наприклад, файлів журналів системи. Так, основний журнал системи, що ведеться демоном syslogd , розміщується у вигляді декількох файлів в підкаталозі /var/adm , а у файлі /var/adm/messages зберігаються повідомлення часу завантаження. Має сенс створювати окрему фізичну файлову систему для розміщення цього каталогу.

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

3.2.2. Основні компоненти файлових систем UNIX

Кожен жорсткий диск складається з однієї або декількох логічних частин (груп циліндрів), званих *розділами* (partitions). Розташування та розмір розділу визначається при форматуванні диска. В ОС UNIX розділи виступають в якості незалежних пристройів, доступ до яких здійснюється як до різних носіїв даних. Зазвичай в розділі може розташовуватися тільки одна *фізична файлова система*.

Є багато типів фізичних файлових систем, наприклад FAT16 і NTFS, з різною структурою. Більше того, є безліч типів фізичних файлових систем UNIX (**ufs**, **s5fs**, **ext2**, **vxfs**, **jfs**, **ffs** і т.і.). Нижче ми розглянемо основні їх загальні особливості.

Фізична файлова система UNIX займає розділ диска і складається з таких основних компонентів:

- *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 індексному дескриптору по індексу 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 містить інформацію, необхідну для монтування та управління файловою системою в цілому. У кожній файловій системі існує тільки один суперблок, який розташовується на початку розділу. Суперблок читається в пам'ять ядра при монтуванні файлової системи і перебуває там до її відключення - *демонтажу*.

Суперблок містить:

- тип файлової системи;
- розмір файлової системи в логічних блоках, включаючи сам суперблок, масив індексних дескрипторів та 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 у суперблоці непрактично. Для індексних дескрипторів зберігатися 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 по вмісту блоку не можна визначити, вільний він чи ні. Тому необхідно зберігати список адрес вільних блоків цілком. Перелік адрес вільних блоків може займати декілька 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 читається в суперблок, і блок стає вільним. Такий підхід дозволяє використовувати дисковий простір під списки, пропорційне вільному місцю у файловій системі. Коли вільному місцю практично не залишається, список адрес вільних блоків цілком поміщається в суперблоці.

Індексний дескриптор, або `inode`, містить інформацію про файл, необхідну для обробки даних, тобто метадані файлу. Кожен файл асоційований з одним індекс-

ним дескриптором, хоча може мати декілька імен (жорстких зв'язків) у файловій 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, який розміщений у 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 даного файлу.

Кожен дескриптор містить 13 покажчиків. Перші 10 покажчиків безпосередньо посилаються на блоки даних файлу. Якщо файл більшого розміру - одинадцятий покажчик посилається на перший *непрямий блок* (indirection block) з 128 (256) посилань на блоки даних. Якщо і цього недостатньо, дванадцятий покажчик посилається на *двічі непрямий блок*, що містить 128 (256) посилань на непрямі блоки. Нарешті останній, тринадцятого покажчик посилається на *тричі непрямий блок* з 128 (256) посилань на двічі непрямі блоки. Кількість елементів в непрямому блоці залежить від його розміру.

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

Синхронізація структури файлової системи. При відкритті файлу ядро поміщає копію дискового індексного дескриптора у відповідну таблицю в пам'яті, яка містить додаткові атрибути. У подальшому 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.

Якщо відбулося позаштатне 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 (у разі аварійної зупинки системи) може привести до появи таких помилок у файловій системі:

1. Один блок адресується кількома дескрипторами (належить кільком файлів).
2. Блок позначений як вільний, але в теж час зайнятий (на нього посилається дескриптор).
3. Блок позначений як зайнятий, але в той же час вільний (ні один дескриптор на нього не посилається).
4. Неправильне кількість посилань у дескрипторі.
5. Розбіжність між розміром файлу і сумарним розміром адресованих дескриптором блоків.
6. Неприпустимі адресовані блоки (наприклад, розташовані за межами файлової системи).
7. "Загублені" файли (правильні дескриптори, на які не посилаються запису каталогів).
8. Неприпустимі номери дескрипторів в записах каталогів.

Частина цих проблем може бути усунена необхідні інструменти, **fsck** (див. далі в розділі, присвяченому управлінню файловою системою). Але принципове рішення проблеми узгодженості та цілісності даних у файлових системах UNIX мож-

ливо тільки при використанні **журналізації** - попереднього запису всіх змін дискової структури в окрему область на диску.

У **журнальованій файлової системі** після того, як *транзакція* (зміна) записана, вона може бути виконана повторно, що запобігає виникненню помилок і неузгодженостей у файловій системі і необхідність запуску програми **fsck**. Тим самим, зменшується час перевантаження у випадку збою або некоректної зупинки системи.

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

Різні версії ОС UNIX підтримують різні реалізації журналізованих файлових систем. Це, наприклад, файлова система **ufs** (Solaris), **vxfs** (Solaris, UnixWare), **RaiserFS** та **ext2, ext3, ext4** (Linux), **jfs** (AIX та Linux) та інші. Деякі файлові системи дозволяють включати і відключати журналізацію (**ufs, ext2/ext3**). Природно, журналізація трохи сповільнює роботу файлової системи, але, в більшості випадків, гарантує цілісність даних.

3.3.2. Огляд файлових систем

Операційні системи UNIX підтримують багато файлових систем. Нижче наведена коротка інформація про файлові системи, які використовуються на ций час найчастіше.

Файлова система ext2 (також відома як друга розширенна файлова система) розроблена для усунення недоліків у системі Minix, яка використовувалася у ранніх версіях Linux. Вона широко використовувалася в Linux протягом довгого часу. Ext2 не журналюється і значною мірою витіснена ext3. Використовується в основному для розділів невеликого розміру.

Файлова система ext3 доповнює можливості стандартної ext2 журналюванням і тому є результатом еволюційного розвитку дуже стабільної файлової системи. Вона забезпечує розумну продуктивність у більшості ситуацій і продовжує вдосконюватися. Оскільки ця система є розширенним варіантом системи ext2, можливо перетворювати систему ext2 на ext3 і, у разі потреби, навпаки.

Файлові системи RamFS та TmpFS. Тимчасова файлова система (віртуальний диск у старій термінології компанії Microsoft) створюється в оперативній пам'яті і

використовується звичайно для тимчасового збереження файлів з метою підвищення швидкодії та зменшення навантаження на жорсткий диск. У нижче наведеному прикладі створюється файлова система розміром 80Мб в оперативній пам'яті і монтується в директорію **/tmp/RamDisk**:

```
mount -t tmpfs none /tmp/RamDisk -o size=80m  
mount -t ramfs none /tmp/RamDisk -o size=80m
```

Зовнішньо різниця між цими двома командами не значна. Вона полягає в тому, що файлова система TmpFS створюється у віртуальній памяті (тобто може використовувати розділ підкачки) і має фіксований розмір. Файлова система RamFS використовує виключно оперативну пам'ять і може змінювати свій розмір у бік збільшення.

Файлова система vfat (також відома як FAT32) не є журнальованою та має безліч недоліків порівняно з файловими системами, які використовуються Linux. Вона застосовується для обміну даними між системами Windows і Linux, оскільки читається обома. Не слід використовувати цю файлову систему в Linux, за винятком випадків спільногого використання даних системами Windows та Linux. Якщо розпакувати архів Linux на диск з системою vfat, це призведе до втрати права доступу, наприклад, на виконання програм, а також символічні посилання, які могли зберігатися в архіві.

Файлова система SquashFS – стиснута файлова система, яка надає доступ до даних в режимі «лише для читання». SquashFS стискує файли, індексні дескриптори та каталоги, а також підтримує блоки розміром до 1024 Кбайт для кращого стискання. Крім того Squashfs є вільним ПЗ (використовується ліцензія GPL).

SquashFS призначена для використання в пристроях «лише для читання», а також в обмежених за розміром блокових пристроях/системах зберігання (тобто у вбудовуваних системах), де необхідні низькі витрати на зберігання.

Стандартна версія SquashFS використовує алгоритм стискування gzip, хоча існує проект, що дозволяє використовувати алгоритм стискування LZMA.

Squashfs використовується Live CD дистрибутивами Debian, Finnix, Gentoo, Ubuntu, Fedora, gNewSense, а також у вбудовуваних дистрибутивах, таких як прошивки маршрутизаторів OpenWRT і DD-WRT. Крім того, Squashfs використовуєть-

ся спільно з файловими системами, які створюють каскадно-об'єднане монтування, такими як UnionFS і AUFS, щоб надати можливість використовувати Live CD дистрибутиви Linux в режимі «читання-запис». Це дає переваги від використання високошвидкісного стискання Squashfs з можливістю здійснювати зміни дистрибутива під час завантаження з Live CD. Такі дистрибутиви як Slax, Debian Live і Mandriva One використовують цю комбінацію.

SquashFS вже є досить стабільною файловою системою, що послужило внесенням її до основної гілки розробки ядра Linux починаючи з ядра версії 2.6.29. Ви можете зустріти її на звичайному livecd диску Ubuntu. Файл squashfs-образу знаходиться в теці casper на диску з Ubuntu і називається filesystem.squashfs. Якщо в ядрі є підтримка даної файлової системи, то для того, щоб переглянути її зміст, достатньо примонтувати цей файл як це показано в прикладі:

```
mount -o loop /media/cdrom/casper/filesystem.squashfs /mnt/sfs
```

Після цього у каталозі /mnt/sfs можна побачити структуру каталогів звичайного дистрибутива Ubuntu.

Отримати файли з цієї файлової системи можна й іншим способом. Okрім реалізації самої даної файлової системи, існують також і утиліти для управління нею.

Файловая система UnionFS - допоміжна файлова система для Linux і FreeBSD, яка здійснює каскадно-об'єднане монтування інших файлових систем. Це дозволяє файлам і каталогам ізольованих файлових систем, відомих як гілки, прозоро перевиватися, формуючи єдину зв'язану файлову систему. Каталоги, які мають ті ж шляхи в об'єднаних гілках, спільно відображатиме вміст в об'єднаному каталозі нової віртуальної файлової системи.

Коли гілки вмонтовуються, то вказується пріоритет однієї гілки над іншою. Отже, коли обидві гілки містять файл з ідентичним ім'ям, одна гілка матиме більший пріоритет.

Різні гілки можуть одночасно знаходитися в режимі «лише читання» і тільки одна в режимі «читання-запис», таким чином, запис в об'єднану віртуальну файлову систему буде направлений на певну реальну файлову систему. Це дозволяє файлові системі виглядати змінною, але насправді, не дозволяє здійснювати запис змін у файлові системи «лише читання», цей процес також відомий як копіювання при за-

писі. Це може бути потрібно, коли носій інформації фізично дозволяє лише читання, як у випадку з Live CD дисками.

Існує дві версії UnionFS для Linux. Версія 1.x є ізольованою, яка може бути зібрана у вигляді модуля. Версія 2.x новіша, реконструйована. У січні 2007, UnionFS була включена у гілку Linux -mm tree, яку підтримує Andrew Morton, що означає подальше повне включення в основну гілку ядра Linux. Версія 2.x є найдрібнішою реалізацією каскадно-об'єднаного монтування для Linux, вона була ретельно перевірена і досліджена багатьма розробниками ядра, до того ж є найефективнішою.

Aufs (Another Union File System) - альтернативна версія UnionFS, яка здійснює каскадно-об'єднане монтування для файлових систем Linux. Розробка ведеться Junjiro Okajima з 2006. Aufs повністю переписаний код UnionFS, який націленний на поліпшення стабільності та продуктивності. Крім того, в ній представлені деякі нові поняття, такі як балансування гілки, яка записується, та інші поліпшення. Деякі з цих ідей були реалізовані в UnionFS версії 2.x.

Aufs замінила UnionFS в дистрибутиві Linux Knoppix Live CD починаючи з кінця 2006, «для кращої стабільності та продуктивності». Ця допоміжна файлова система замінила UnionFS в SLAX (і сценарії Linux-live) починаючи з версії 6.

Файлова система NFS (Network File System) – протокол мережного доступу до файлових систем, спочатку розроблений Sun Microsystems в 1984 році. Заснований на протоколі виклику вилучених процедур. NFS надає клієнтам прозорий доступ до файлів і файлової системи сервера. На відміну від FTP протокол NFS здійснює доступ тільки до тих частин файлу, до яких звернувся процес, і основне достоїнство його в тому, що він робить цей доступ прозорим. Це означає, що будь-яка програма, що може працювати з локальним файлом, з таким же успіхом може працювати з NFS файлом.

Нижче наведено приклад монтування розділу на сервері з IP-адресою 192.168.0.21

```
mount -t nfs -o nolock 192.168.0.21:/home/nfs /mnt/nfs
```

Після виконання такої команди користувач зможе працювати з файлами, розташованими на сервері, як з локальними в каталозі /mnt/nfs .

3.3. Управління файловою системою

Основними завданнями адміністрування файлових систем є створення, монтування та демонтування фізичних файлових систем, а також перевірка їх цілісності. У наступних підрозділах ми розглянемо відповідні команди і узагальнено опишемо виконувані ними дії.

3.3.1. Створення фізичної файлової системи

Команда **mkfs** створює файлову систему шляхом запису на зазначений пристрій (необхідно вказати спеціальне символьне пристрій). Файловая система створюється на основі зазначених у командному рядку типу файлової системи (ТіпФС), **спеціфічних_опцій** та **операндів**. Команда має такий синтаксис:

```
mkfs [-v][-t ТипФС][-c] [-o специфічні_опції] пристрій розмір [операнди]
```

Спеціфічні опції та операнди залежать від конкретного типу створюваної файлової системи. Їх можна подивитися на відповідній сторінці довідкового посібника (наприклад, **man mkfs_ufs** для файлової системи **ufs**).

Основні опції і параметри команди **mkfs** представлені в табл. 14.

Таблиця 14. Основні опції і параметри команди **mkfs**

Опція	Призначення
-t	Вказує тип файлової системи, яку необхідно створити. Тип файлової системи повинен бути або зазначено тут, або перебуває у файлі таблиці стандартних файлових систем (/etc/vfstab в SVR4, /etc/fstab в інших версіях UNIX) шляхом зіставлення пристрої з записом в таблиці.
-v	Видає результатуючу командний рядок, але не виконує команду. Командний рядок генерується з використанням опцій і аргументів, за-

Таблиця 14. Основні опції і параметри команди `mkfs`

Опція	Призначення
	значених користувачем, шляхом додавання до них інформації, взятої з таблиці стандартних файлових систем. Ця опція використовується для перевірки правильності командного рядка.
-c	Перевірити пристрій перед створенням файлової системи.
-o	Визначає опції, специфічні для зазначеного типу фізичного файлової системи.
пристрій	Визначає спеціальне символне пристрій, на якому буде створена файлова система.
розмір	Вказує кількість 512-байтових блоків у файловій системі. Максимальний розмір багатьох фізичних файлових систем в UNIX - 4194304 блоку розміром 512 байт (або 2 Гбайта).

3.3.2. Монтування та демонтування фізичних файлових систем

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 у **точці_монтування**, яка вже повинна існувати; точка монтування стає ім'ям кореня знову змонтованого пристрою або ресурсу. Кажуть, що ця команда монтує або підключає фізичну файлову систему або ресурс до загальноЛогічної файлової системи.

Команда `mount` має такий синтаксис:

`mount [-v | -h | -a]`

`mount [-t ТипФС] [-v] [-o 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 з відповідної таблиці. При виклику з неповним списком аргументів (наприклад, тільки з вказівкою **пристрою** або **точки_монтування**, або коли зазначені обидва ці аргументи, але не заданий тип файлової системи), **mount** буде переглядати таблицю стандартних файлових систем у пошуках відсутніх аргументів. Потім вона викликає специфічний модуль монтування для відповідного типу файлової системи.

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

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

```
umount [-v] [-o 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** та **umount** сприймають такі основні опції: **-v** - видати інформацію про версію; **-h** - видати підказку про опції даної команди; **-t** - задає тип файлової системи для монтування (тип файлової системи повинен бути або заданий, або визначається по таблиці стандартних файлових систем **/etc/fstab** в ході монтування); **-a** - монтувати всі файлові системи, перелічені в файлі **/etc/fstab**; **-o** - задає специфічні опції для зазначеного типу фізичного файлової системи.

Будь-який користувач може викликати команду **mount** для отримання списку змонтованих файлових систем і ресурсів. Наприклад:

```
$ mount
/dev/dsk/c1t0d0s0 - / ufs - no
rw,intr,largefiles,logging,onerror=panic,suid,dev=740040
/dev/dsk/c1t0d0s3 - /usr ufs - no
rw,intr,largefiles,logging,onerror=panic,suid,dev=740043
/dev/dsk/c1t0d0p0:boot - /boot pcfs - no
rw,nohidden,nofoldcase,dev=763050
/proc - /proc proc - no dev=2c00000
```

```
fd - /dev/fd fd - no rw,suid,dev=2cc0000
mnttab - /etc/mnttab mntfs - no dev=2dc0000
/dev/dsk/c1t0d0s1 - /var ufs - no
rw,intr,largefiles,logging,onerror=panic,suid,dev=740041
swap - /var/run tmpfs - no dev=1
swap - /tmp tmpfs - no dev=2
/dev/dsk/c1t0d0s4 - /home ufs - no
rw,intr,largefiles,logging,onerror=panic,suid,dev=740044
/dev/dsk/c2t0d0s1 - /fs ufs - no
rw,intr,largefiles,logging,onerror=panic,suid,dev=740401
```

Тільки користувач **root** може монтувати чи демонтувати файлові системи.

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

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

- спеціальний блочний пристрій або ім'я ресурсу;
- неформатований (спеціальний символний) пристрій для перевірки утилітою **fsck** ;
- стандартний каталог монтування;
- тип файлової системи;
- число, використовуване **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 -. Розглянемо приклад записів з таблиці стандартних файлових систем:

#device	device	mount	FS	fsck	mount	mount
#to mount	to fsck	point	type	pass	at boot	options
/dev/dsk/c1t0d0s0	/dev/rdsk/c1t0d0s0	/	ufs	1	no	logging
/dev/dsk/c1t0d0s3	/dev/rdsk/c1t0d0s3	/usr	ufs	1	no	logging
/dev/dsk/c1t0d0s1	/dev/rdsk/c1t0d0s1	/var	ufs	1	no	-
/dev/dsk/c1t0d0s4	/dev/rdsk/c1t0d0s4	/home	ufs	2	yes	logging
...						

3.3.3. Отримання інформації про файлові системи

Перевірка та відновлення цілісності файлових систем. Програма **fsck** шукає і, автоматично або в інтерактивному режимі, виправляє протиріччя у файлових системах. Якщо файлова система знаходиться в неузгодженному стані, що не можна однозначно виправити, у користувача запитують підтвердження перед спробою виконати кожне виправлення. Слід мати на увазі, що деякі виправлення призводять до певних втрат даних. Обсяг і серйозність втрати даних можна визначити з діагностичного повідомленням. Стандартним дією при кожному виправленні є очікування від користувача ствердної (**yes**) чи негативного (**no**) відповіді.

При використанні **fsck** файлова система повинна бути неактивній (розмонтувати 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** має такий синтаксис:

```
fsck [-SACVRTNP] [-t fstype] [пристрій] [-- специфічні_опції]
```

Основні опції і параметри команди **fsck** представлені в табл. 15.

Таблиця 15. Основні опції команди **fsck**

Опція	Призначення
-t	Задає тип файлової системи, яку слід перевірити. Можна задати декілька

Таблиця 15. Основні опції команди **fsck**

Опція	Призначення
	значень, розділених комами. Якщо тип не вказано, команда звертається до таблиці стандартних файлових систем.
-A	Перевіряти всі файлові системи перелічені в файлі /etc/fstab .
-N	Видає результатуючий 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.
--	Дозволяє задати опції, специфічні для типу файлової системи.

Для роботи команді **fsck** необхідно вказувати спеціальне символне пристрій.

Коренева файлова система зазвичай перевіряється при запуску автоматично. Система при запуску може автоматично перевіряти та інші фізичні файлові системи, для яких у таблиці стандартних файлових систем вказана необхідність такої перевірки. Ця перевірка може вестися паралельно, шляхом запуску окремого процесу **fsck** для кожної перевіряється файлової системи з одним і тим же порядковим номером перевірки. Паралельно має сенс перевіряти файлові системи, розташовані на різних фізичних дисках.

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

```
df [-t ТипФС] [-ahhiklp] [Пристрій | каталог | файл | ресурс ...]
```

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

Розглянемо приклади виконання команди **df**:

```
$ df -k
Filesystem      kbytes   used  avail capacity  Mounted on
/dev/dsk/c1t0d0s0    245983  20713  200672   10%       /
/dev/dsk/c1t0d0s3   3096090 1782106 1252063   59%     /usr
/dev/dsk/c1t0d0p0:boot
                    10797   1622   9175   16%     /boot
/proc              0       0       0    0%     /proc
fd                 0       0       0    0%     /dev/fd
mnttab             0       0       0    0%     /etc/mnttab
/dev/dsk/c1t0d0s1    491983  204863  237922   47%     /var
swap               324832    16  324816   1%     /var/run
swap               337828  13012  324816   4%     /tmp
/dev/dsk/c1t0d0s4   2305873 1021225 1238531   46%     /home
/dev/dsk/c2t0d0s1   6192197 5633827  496449   92%     /fs

$ df
/
          (/dev/dsk/c1t0d0s0 ): 450540 blocks  120616 files
/usr
          (/dev/dsk/c1t0d0s3 ): 2627968 blocks 338652 files
/boot
          (/dev/dsk/c1t0d0p0:boot): 18350 blocks      -1
files
/proc
          (/proc ): 0 blocks  3615 files
/dev/fd
          (fd ): 0 blocks  0 files
/etc/mnttab
          (mnttab ): 0 blocks  0 files
/var
          (/dev/dsk/c1t0d0s1 ): 574236 blocks 240784 files
/var/run
          (swap ): 647568 blocks 43108 files
/tmp
          (swap ): 647568 blocks 43108 files
/home
          (/dev/dsk/c1t0d0s4 ): 2569298 blocks 379999 files
/fs
          (/dev/dsk/c2t0d0s1 ): 1116738 blocks 688872 files
```

Якщо необхідно вивести дані про використання **inode**, використовується команда **df** з опцією **-i**. Можна виключити виведення даних за визначеню файловою системою, використовуючи опцію **-x**, або лімітувати інформацію визначеними типами файлових систем, використовуючи опцію **-t**. За необхідності їх можна використовувати декілька раз.

Команда **df** виводить інформацію тільки про файлову систему в цілому. Іноді необхідно дізнатися, скільки місця займає каталог **home**, або який розмір розділу

знадобиться, щоб розташувати каталог **/usr** в окремій файловій системі. Для виконання цих завдань використовується команда **du**.

Команда **du** виводить інформацію про файл (файли), імена яких задані як параметри. Якщо задано ім'я каталогу, то **du** визначає розмір всіх файлів та підкаталогів цього каталогу на всіх рівнях входження. Результат роботи команди може бути дуже ємним. Однак існує опція **-s** для виводу звітної інформації за каталогом. Якщо використовувати **du** для отримання інформації за декількома каталогами, можна додати опцію **-c** для виводу сумарних даних. Також можна задавати формат виводу. Для цього використовуються опції, аналогічні тим, які використовуються в команді **df**.

4. УПРАВЛІННЯ ПРОЦЕСАМИ

4.1. Процеси в ОС UNIX

4.1.1. Типи процесів

Основним ресурсом комп'ютера є його процесор (або процесори). У кожен момент часу один процесор може виконувати тільки один процес. Організація планування процесів так, щоб за рахунок їх перемикання створювалася ілюзія одночасної роботи декількох процесів - одна з основних завдань будь-якої багатокористувачкої і багатозадачної операційної системи.

В ОС UNIX основним засобом організації і одиницею багатозадачності є *процес* - унікальним чином ідентифікується програма, яка потребує отримання доступу до ресурсів комп'ютера. Операційна система маніпулює чином процесу, який являє собою програмний код, а також розділами даних процесу, визначальними середу виконання. 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.

Основою операційної системи UNIX є ядро. Ядро являє собою спеціальну програму (або декілька програмних модулів, у випадку модульного ядра), яка постійно знаходиться в оперативній пам'яті і працює, поки працює операційна система. Ядро керує усіма таблицями, що використовуються для відстеження процесів та інших ресурсів. Ядро завантажується в пам'ять під час початкового завантаження і негайно запускає необхідні процеси, зокрема процес ініціалізації операційної системи - `init`.

Дані, пов'язані з процесом, також є частиною образу процесу. Деякі з них зберігаються в регістрах, зазвичай представлених регістрами процесора. Крім того, існують динамічні області зберігання даних (*купа*), що виділяються процесом по ходу роботи при необхідності.

Ще у процесу є *стек*, що міститься в пам'яті і використовується для зберігання локальних змінних програми і передачі параметрів. Коли процес виконує звернення до функції або підпрограмі, в стек відправляється новий *фрейм*. Однією з частин

кожного фрейма є покажчик на базу попереднього фрейму, який дозволяє легко повернутися з виклику функції. При цьому важливо знати місце розташування поточного фрейма та вершину стека.

Регістри грають важливу роль у роботі процесів. Звичайно 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.

Під час виконання або в очікуванні "свого часу" процеси містяться у віртуальній пам'яті з сторінкової організацією. Частина цієї віртуальної пам'яті зіставляється з фізичною. Частина фізичної пам'яті резервується для ядра операційної системи. Користувачі можуть отримати доступ тільки до тієї, що залишилася для процесів пам'яті. При необхідності, сторінки пам'яті процесів відкачуються з фізичної пам'яті на диск, в область *підкачки*. При зверненні до сторінки у віртуальній пам'яті, якщо вона не перебуває у фізичній пам'яті, відбувається її підкачка з диска.

Віртуальна пам'ять реалізується і автоматично підтримується ядром ОС UNIX.

В ОС UNIX виділяється три типи процесів: *системні, процеси-демони і 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. Виконувані інструкції і дані 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**, що є праобразом 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, і виконуються 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 чекають, поки той або інший процес запросить певну послугу.

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

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

У наступному прикладі, що показує частину списку процесів, напівжирним виділені системні процеси. У цій ОС системними є процес-планувальник (**sched**), процес відкачування сторінок віртуальної пам'яті (**pageout**) і процес, що синхронізує файлові системи (**fsflush**). Користувальницькі процеси представлені на більш темному тлі. Всі інші процеси – це демони, реалізують ті чи інші служби. Назви команд, що починаються з дефіса, представляють початкові командні інтерпретатори користувачів.

```
$ ps -eacf | more
```

UID	PID	PPID	CLS	PRI	STIME	TTY	TIME	CMD
root	0	0	SYS	96	Фев 23	?	0:19	sched
root	1	0	TS	58	Фев 23	?	0:04	/etc/init -
root	2	0	SYS	98	Фев 23	?	0:08	pageout
root	3	0	SYS	60	Фев 23	?	87:49	fsflush
root	411	1	TS	58	Фев 23	?	0:00	/usr/lib/saf/sac -t 300
root	259	1	TS	50	Фев 23	?	2:20	/usr/sbin/nsqd
root	184	1	TS	46	Фев 23	?	0:00	/usr/lib/netsvc/yp/ypxfrd
root	68	1	TS	58	Фев 23	?	0:02	/usr/lib/sysevent/syseventd

```

root 144      1   TS  59   Фев 23 ?      0:00 /usr/sbin/in. rdisc -s
root 161      1   TS  58   Фев 23 ?      0:41 /usr/sbin/rpcbind
...
markov 5724    5723  TS  48  12:07:16 pts/1 0:00 -bash
root 3705    215   TS  54  09:46:57 ?      0:00 in. telnetd
root 6804    6803  IA  48   Map 25 ??      0:00 /usr/dt/bin/dtterm
root 87     310   TS  59   Map 19 ?      0:02 /usr/local/samba/bin/smbd
                           -D -s/usr/local/samba/lib/smb. conf
root 27210   215   TS  54   Map 27 ?      0:00 in. telnetd
root 3918    215   TS  54  10:11:00 ?      0:00 in. telnetd
spot 3697    3679  TS  38  09:46:39 pts/14 0:00 -bash
...

```

4.1.1. Атрибути процесу

Процес в UNIX має ряд атрибутів, що дозволяють операційній системі керувати його роботою. Основні атрибути подані в наступних підрозділах.

Ідентифікатор процесу (PID). Кожен процес має унікальний ідентифікатор PID, що дозволяє ядру системи розрізняти процеси. Коли створюється новий процес, ядро присвоює йому наступний вільний (тобто не асоційований ні з яким процесом) ідентифікатор. Присвоєння ідентифікатора зазвичай відбувається за зростаючий, тобто ідентифікатор нового процесу більше, ніж ідентифікатор процесу, створеного перед ним. Якщо ідентифікатор досягає максимального значення (зазвичай, 65 737), наступний процес отримає мінімальний вільний PID і цикл повторюється. Коли процес завершує роботу, ядро звільняє використовувався їм ідентифікатор.

Ідентифікатор батьківського процесу (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 чи існували раніше процесів.

Поправка пріоритету (NI). Відносний пріоритет процесу, врахований планувальником при визначенні черговості запуску. Фактичне ж розподіл процесорних ресурсів визначається пріоритетом виконання (атрибут PRI), залежать від декількох факторів, зокрема від заданого відносного пріоритету. Відносний пріоритет не змінюється системою на всьому протязі життя процесу (хоча може бути змінений кори-

стувачем або адміністратором) на відміну від пріоритету виконання, динамічно змінюваного планувальником.

Термінальна лінія (TTY) – термінал або псевдотермінал, пов'язаний з процесом. З цим терміналом за замовчуванням пов'язані стандартні потоки: *вхідний*, *вихідний* і *потік повідомлень* про помилки. Потоки (*програмні канали*) є стандартним способом між процесами взаємодії в ОС UNIX.

Процеси-демони не пов'язані з терміналом.

Реальний (UID) і ефективний (EUID) ідентифікатори користувача. 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 більш широкі права, ніж права користувача, шляхом установки *bіта SUID*, коли ефективному ідентифікатору присвоюється значення ідентифікатора власника виконуваного файлу (наприклад, користувача `root`).

Реальний (GID) та ефективний (EGID) ідентифікатори групи. 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 файла встановлений *біт SGID*, такий файл виконується з ефективним ідентифікатором групи-власника.

4.1.3. Життєвий цикл процесу в UNIX і основні системні виклики

Життєвий цикл процесу в ОС UNIX може бути розбитий на декілька станів. Перехід з одного стану в інше відбувається в залежності від настання певних подій у системі. Можливі такі 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.

3. Процес не виконується, але готовий до запуску, як тільки планувальник вибере його (стан **Runnable**). Процес перебуває в черзі на виконання і володіє всіма необхідними йому ресурсами, крім процесора.
4. Процес перебуває в стані сну (**asleep**), очікуючи 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 перебуває в стані сну.
7. Процес виконав системний виклик **exit** і перейшов у 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.

Процес починає свій життєвий шлях з 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 у стан 1 і виконується в режимі користувача.

Виконання в режимі користувача 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 у 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 наступного найбільш пріоритетному процесу. Коли ресурс ставати доступним, ядро "пробуджує процес", використовуючи виклик **wakeup**, поміщає його в чергу на виконання, і процес переходить у стан 3 готовності до запуску.

При наданні процесу обчислювальних ресурсів відбувається перемикання контексту, в результаті якого зберігається образ, чи *контекст*, поточного процесу, і ке-

рування передається новому. Переключення контексту може статися, наприклад, якщо процес перейшов у стан сну, або якщо в стані готовності до запуску перебуває процес з більш високим пріоритетом, ніж поточний. В останньому випадку ядро не може негайно перервати поточний процес і зробити перемикання контексту. Справа в тому, що перемикання контексту при виконанні в режимі ядра може привести до порушення цілісності самої системи. Тому перемикання контексту відкладається до моменту переходу процесу з режиму ядра в призначений для користувача режим, коли всі системні операції завершенні, і структури даних ядра знаходяться в нормальному стані.

Таким чином, після того як планувальник вибрал процес на запуск, останній починає своє виконання в режимі ядра, де завершує перемикання контексту. Далі стан процесу залежить від передісторії.

Нарешті, процес виконує системний виклик `exit` і закінчує своє виконання. Процес може бути також завершений 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 знаходитьсь до тих пір, поки батьківський процес не виконає системний виклик `wait`, після чого вся інформація про процес буде знищена, а батько отримає код повернення завершився процесу.

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:

- *Адресний простір процесу в режимі користувача.* Сюди входять код, дані і стек процесу, а також інші області, наприклад, колективна пам'ять або код і дані динамічних бібліотек.
- *Керуюча інформація.* Ядро використовує дві основні структури для управління процесом – `proc` та `user`. Сюди ж входять дані, необхідні для відображення віртуального адресного простору процесу у фізичну пам'ять.

- *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. Середовище процесу можна отримувати або змінювати за допомогою функцій.
- *Апаратний контекст.* Сюди входять значення загальних і ряду системних реєстрів процесора, зокрема, покажчик поточної інструкції і покажчик стека (див. на початку розділу).

Переключення між процесами, необхідне для розподілу обчислювального ресурсу, по суті, виявляється у переключенні контексту, коли контекст виконується процес запам'ятується, а відновлюється контекст процесу, вибраного планувальніком. Переключення процесу є досить ресурсномісткою операцією. Окрім збереження стану реєстрів процесу, ядро змушене виконати безліч інших дій.

Контекст перемикається в чотирьох випадках:

1. Поточний процес переходить у стан сну, чекаючи недоступного ресурсу.
2. Поточний процес завершує своє виконання.
3. Якщо після перерахунку пріоритетів у черзі на виконання є більш високопріоритетні процес.
4. Відбувається пробудження більш найпріоритетніше процесу.

Перші два випадки відповідають добровільному переключенню контексту і дії ядра при цьому досить прості. Ядро викликає процедуру перемикання контексту з функції `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. У цій ситуації ядро встановлює спеціальний прапор `runrun`, який вказує, що в черзі перебуває понад високопріоритетні 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.

Пріоритети процесів. Планування процесів та UNIX засноване на *пріоритеті* процесу. Планувальник завжди вибирає процес з найвищим пріоритетом. Пріоритет

процесу не є фіксованим і динамічно змінюється системою в залежності від використання обчислювальних ресурсів, часу очікування запуску і поточного стану процесу. Якщо процес готовий до запуску і має найвищий пріоритет, планувальник призупинить виконання поточного процесу (з більш низьким пріоритетом), навіть якщо останній не "виробив" свій часовий квант.

Ядро UNIX є *неперервним* (nonpreemptive). Це означає, що процес, що перебуває в режимі ядра (в результаті системного виклику або переривання) і виконує системні інструкції, не може бути перерваний системою, а обчислювальні ресурси передані іншій високопріоритетним процесу. У цьому стані виконуваний процес не може звільнити процесор "за власним бажанням", в результаті недоступності якого-небудь ресурсу перейшовши в стан сну. В іншому випадку система може перервати виконання процесу тільки при переході з режиму ядра в користувачький режим. Такий підхід значно спрощує рішення задач синхронізації і підтримки цілісності структур даних ядра.

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 використованого планувальника. У будь-якому випадку, процеси, що виконуються в режимі користувача, мають більш низький пріоритет, ніж працюють у режимі ядра.

4.1.4. Створення і завершення процесу

Новий процес створюється в UNIX тільки шляхом системного виклику **fork**. Процес, який зробив виклик **fork**, називається *батьківським*, а знов створений процес - *породженим*. Новий процес є точною копією батьківського. При породженні (розгалуженні) процесу перевіряється, чи достатньо пам'яті та місця в таблиці процесів для даного процесу. Якщо так, то образ поточного процесу копіюється у новий образ процесу, і в таблиці процесів виникає новий елемент. Нового процесу присвоюється новий унікальний ідентифікатор (PID). Коли зміна таблиці процесів ядра

завершується, процес додається до списку процесів, доступних для виконання та очікують у черзі планувальника подібно іншим процесам.

Породжений процес успадковує від батьківського процесу такі основні характеристики:

1. Способи обробки сигналів (адреси функцій обробки сигналів).
2. Реальні та ефективні ідентифікатори користувача та групи.
3. Значення поправки пріоритету.
4. Усі приєднані сегменти пам'яті.
5. Ідентифікатор групи процесів.
6. Термінальну лінію.
7. Поточний каталог.
8. Кореневий каталог.
9. Маска створення файлів (**umask**).
10. Обмеження ресурсів (**ulimit**).

Породжений процес відрізняється від батьківського процесу такими основними характеристиками:

1. Породжений процес має свій унікальний ідентифікатор.
2. Породжений процес має інший ідентифікатор батьківського процесу, рівний ідентифікатору породив процесу.
3. Породжений процес 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 й відповідний батьківський.
4. У породженого процесу обнулюються лічильники часу, витраченого системою для його обслуговування.

Системний виклик **fork** завершується невдачею і новий процес не породжується, якщо:

- Створити процес забороняє системне обмеження на загальну кількість процесів.
- Створити процес забороняє системне обмеження на кількість процесів в одного користувача.

- Загальна кількість системної пам'яті, наданої для фізичного введення-виведення, тимчасово виявилося недостатнім.

При успішному завершенні породженному процесу повертається значення **0**, а батьківському процесові повертається ідентифікатор породженого процесу. У випадку помилки батьківському процесові повертається **-1**, новий процес не створюється і змінної **errno** присвоюється код помилки.

Зазвичай після породження породжений процес виконує системний виклик **exec**, перекриває сегменти тексту і даних процесу новими сегментами, взятыми із зазначеного виконуваного файлу. При цьому апаратний контекст процесу ініціалізується заново.

Виконуваний файл складається з заголовка, сегменту команд і сегменту даних. Дані (глобальні змінні) складаються з ініціалізованої і неініціалізованої частин.

Якщо системний виклик **exec** закінчився успішно, то він не може повернути управління, так що викликав процес уже замінений новим процесом. Повернення з системного виклику **exec** свідчить про помилку. У такому випадку результат дорівнює **-1**, а змінній **errno** присвоюється код помилки.

Новий процес успадковує у процесу, що викликав **exec**, такі характеристики:

1. Значення поправки пріоритету.
2. Ідентифікатор процесу.
3. Ідентифікатор батьківського процесу.
4. Ідентифікатор групи процесів.
5. Термінальну лінію.
6. Поточний каталог.
7. Кореневий каталог.
8. Маску створення файлів.
9. Обмеження ресурсів.
- 10.Лічильники часу, витраченого системою на обслуговування цього процесу.
- 11.Блокування доступу до сегментів файлів.

Сон і пробудження. Процес зазвичай переводиться в стан сну при обробці системної функції. Якщо для завершення обробки запиту слід недоступний ресурс, процес знімається з процесора і переводиться в стан сну.

Стан сну - це логічне стан процесу, при цьому він не переміщається фізично в пам'яті. Перехід у стан сну, в першу чергу, визначається занесенням в системну таблицю процесів відповідного прапора стану і події, пробуджуючого процес.

Події інформують про доступність того чи іншого ресурсу. Як правило, події пов'язані з роботою периферійних пристройів. Наступ одного і того ж події може очікувати декілька процесів. Оскільки перехід зі стану в стан акт швидше логічний, то і пробуджуються всі процеси очікують дану подію. Це призводить до зміни стану з "сну" на "готовий до виконання", і відповідні процеси поміщаються в чергу на запуск. Завдання вибору процесу для запуску вирішує планувальник.

Оскільки планувальник приймає рішення про запуск процесу, ґрунтуючись на пріоритетах, єдиним способом встановити "справедливий" порядок запуску процесів є присвоєння певного пріоритету кожної події.

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 такі дії:

1. Відключаються усі сигнали.
2. У викликала процесі закриваються всі дескриптори відкритих файлів.
3. Якщо батьківський процес знаходиться в стані виклику **wait**, то системний виклик **wait** завершується, видаючи батьківському процесові як результат ідентифікатор завершився процесу, і молодші 8 біт його коду завершення.
4. Якщо батьківський процес не знаходиться в стані виклику **wait**, то завершується процес переходить у 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 завершуються процесу збігаються, то всім процесам з тим же ідентифікатором групи процесів посилається сигнал **SIGHUP**. Тим самим, завершуються і всі породжені в пріоритетному режимі процеси.

Батьківському процесові посилається сигнал **SIGCHLD** (завершення породженого процесу). Цей сигнал пробуджує батьківський процес, якщо той чекає завершення породжених процесів.

4.1.5. Управління станом процесу

Для отримання інформації про стан процесів використовується команда **ps**.

Вона має такий синтаксис:

```
ps [ -acdefjLP ]  
      [-t список_терміналов ]  
      [-p список_ідентификаторов_процесов ]  
      [-u |U список_ідентификаторов_пользователей ]  
      [-g список_ідентификаторов_лідеров_груpp ]  
      [-G список_числовіх_ідентификаторов_груpp ]
```

Основні опції команди **ps** в системах SVR4 і BSD описані в табл. 16, а основні поля в результатах виконання представлені в табл. 17.

Таблиця 16. Опції команди ps

Опція	Призначення
-A	Надає інформацію про всі процеси, крім групових, і не пов'язаних з терміналом.
-D	Надає інформацію про всі процеси, виключаючи лідерів сеансу.
-E	Надає інформацію про всі процеси.
-L	Генерує довгий лістинг.
-f	Генерує повний лістинг.
-g список	Виводить інформацію тільки про процеси, для яких зазначені ідентифікатори лідерів груп. Лідер групи - це процес, номер якого ідентичний його кодом. Командний інтерпретатор, що запускається при вході в систему, є стандартним прикладом лідера групи.
-A	Надає інформацію про всі процеси.

Таблиця 16. Опції команди ps

Опція	Призначення
-р список	Надає інформацію по процесах із зазначеними ідентифікаторами.
-т список	Надає інформацію по процесах, що мають відношення до зазначених терміналів.
-u список	Надає інформацію про всі процеси, що мають відношення до зазначених ідентифікаторів користувачів.
-U список	Надає інформацію про всі процеси, що мають відношення до зазначених іменами користувачів.

Таблиця 17. Основні характеристики процесів, що надаються командою ps

Тема	Значення
ADDR	Адреса процесу в пам'яті.
З	Частка виділеного планувальником часу ЦП.
COMD	Ім'я команди й аргументи (для опції -f).
F	<p>Прапори (шістнадцяткові), логічна сума яких дає такі відомості про процес:</p> <ul style="list-style-type: none"> • 00 - процес термінована; елемент таблиці процесів вільний; • 01 - системний процес: завжди в основній пам'яті; • 02 - процес трасує батьківським процесом; • 04 - батьківський трасувальний сигнал зупинив процес; батьківський процес чекає [див ptrace (2)]; • 08 - процес не може бути розбуджений сигналом; • 10 - процес в основній пам'яті; • 20 - процес в основній пам'яті; блокований до завершення дії; • 40 - йде сигнал до віддаленої системи; • 80 - процес в черзі на введення-виведення.
NI	Поправка пріоритету.

Таблиця 17. Основні характеристики процесів, що надаються командою ps

Тема	Значення
PID	Ідентифікатор процесу.
PPID	Ідентифікатор батьківського процесу.
PRI	Поточний пріоритет процесу.
S	Стан процесу: <ul style="list-style-type: none"> • B, W - процес перебуває в стані очікування; • I - створення процесу; • O - процес виконується; • R - перебуває в черзі готових до виконання процесів; • S - процес не активний; • T - процес трасує; • X - очікує додаткової оперативної пам'яті; • Z - процес "зомбі".
STIME	Час запуску процесу.
SZ	Розмір (в блоках по 512 байт) образу процесу в пам'яті.
TIME	Загальний час виконання для процесу
TTY	Термінальна лінія процесу
UID	Ідентифікатор користувача власника процесу
WCHAN	Адреса події, якого очікує процес. У активного процесу цей стовпець - порожній.

У залежності від переданих опцій і реалізації, команда **ps** може видавати й інші атрибути. Команду **ps** може виконувати будь-який користувач. Розглянемо простий приклад:

```
$ ps
  PID  TTY      TIME CMD
 3697 pts/14    0:00 bash
$ ps -1
 F S      UID PID PPID  C PRI NI      ADDR     SZ      WCHAN TTY      TIME CMD
```

```
8 S 31061 3697 3679 0 51 20 e3110048 499 e31100b4 pts/14 0:00 bash
$ ps -p 5726
 PID TTY      TIME CMD
 5726 pts/1    0:00 mc
```

Управління пріоритетом процесів. У всіх UNIX-системах користувачі можуть при запуску процесу задавати значення поправки пріоритету за допомогою команди **nice**. (Кажуть, що команда запускається "з-під" **nice**.) Ця команда має синтаксис:

```
nice [-інкремент | -n [-|+]інкремент] команда [аргумент ... ]
```

Діапазон значень інкременту 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
```

4.1.6. Сигнали: відправка і обробка.

Сигнали забезпечують механізм виклику певної процедури при настанні деякої події (аналогічно перериваннях). Кожна подія має свій числовий ідентифікатор (зазвичай в діапазоні від 1 до 36) і відповідну символічну константу - ім'я. При роботі з сигналами необхідно розрізняти дві фази:

1. Генерація або посилка сигналу.
2. Доставка і обробка сигналу.

Сигнал відправляється, коли відбувається певна подія, про настання якого повинен бути повідомлений процес. Сигнал вважається доставленим, коли процес, якому був відправлений сигнал, отримує його і виконує його обробку. У проміжку між цими двома подіями сигнал чекає доставки.

Сигнал може надсилятися одним процесом іншому (за допомогою відповідного системного виклику) і буде доставлений, якщо обидва процеси – одного користувача або сигнал посланий від імені користувача **root**. Сигнали надсилаються також ядром.

Ядро генерує і посилає процесу сигнал у відповідь на ряд подій, які можуть бути викликані самим процесом, іншим процесом, перериванням або якимось зовнішнім подією. Основні причини відправлення сигналу:

Виняткові ситуації	Виконання процесу викликає виключно ситуацію, наприклад, ділення на 0.
Термінальні переривання	Натискання клавіш терміналу, наприклад, <code></code> , <code><Ctrl+C></code> , <code><Ctrl+\></code> , викликає посилку сигналу поточному процесу, пов'язаного з терміналом.
Інші процеси	Процес може посылати сигнал іншому процесу або групі процесів за допомогою системного виклику kill . У цьому випадку сигнали є елементарною формою між процесами взаємодії.
Управління завданнями	Командні інтерпретатори, підтримуючи засоби управління завданнями, використовують сигнали для маніпулювання фоновими і поточними процесами. Коли процес, що виконується у фоновому режимі, робить спробу читання або запису на термінал, йому надсилається сигнал зупинки. Коли породжений процес завершує свою роботу, батьківський процес повідомляється про це також за допомогою сигналу.
Квоти	Коли процес перевищує виділену йому квоту обчислювальних ресурсів або ресурсів файлової системи, йому надсилається відповідний сигнал.
Повідомлення	Процес може запросити повідомлення про настання тих чи інших подій, наприклад, готовності пристрой і т. д. Таке повідомлення надсилається процесу у вигляді сигналу.
Будильники	Якщо процес встановив таймер, йому буде посланий сигнал, коли значення таймера стане рівним 0.
Доставка і обробка сигналу. Для кожного сигналу в системі визначена обробка за замовчуванням, яку виконує ядро, якщо процес не вказав іншої дії. У загальному випадку обробка виконується ядром.	

ному випадку можливі дії: завершити виконання процесу (з створенням образу пам'яті **core** і без), ігнорувати сигнал, зупинити процес і продовжити процес.

Слід зауважити, що будь-яка обробка сигналу, в тому числі і обробка за замовчуванням, має на увазі, що процес виконується. На системах з високою завантаженням це може привести до затримок між відправленням і доставкою сигналу, тому що процес не може отримати сигнал, поки не буде обраний планувальником, і йому не будуть надані обчислювальні ресурси.

Доставка сигналу відбувається після того, як ядро від імені процесу викликає системну процедуру **issig()**, яка перевіряє, чи існують які очікують доставки сигналі, адресовані даному процесу. Процедура **issig()** викликається ядром у трьох випадках:

1. Безпосередньо перед поверненням з режиму ядра в користувальський режим після обробки системного виклику або переривання.
2. Безпосередньо перед переходом процесу в стан сну з пріоритетом, що допускає переривання сигналом.
3. Відразу ж після пробудження з пріоритетом, що допускає переривання сигналом.

Якщо процедура **issig()** виявляє очікування доставки сигналу, ядро викликає функцію доставки сигналу, яке виконує дію за умовчанням або викликає спеціальну функцію **sendsig()**, яка запускає обробник сигналу, зареєстрований процесом. Функція **sendsig()** повертає процес в призначений для користувача режим, передає управління обробнику сигналу, а потім відновлює контекст процесу для продовження перерваного сигналом виконання.

Робота з сигналами, пов'язаними з винятковими ситуаціями, незначно відрізняється від описаної вище. Виняткова ситуація виникає при виконанні процесом певної інструкції, яка викликає в системі помилку. Якщо таке відбувається, викликається системний обробник виняткової ситуації, і процес переходить в режим ядра, майже так само, як і при обробці будь-якого іншого переривання. Оброблювач відправляє процесу відповідний сигнал, який доставляється, коли процес повертається в призначений для користувача режим.

У стані сну існують дві категорії подій, що викликали стан сну процесу: допускають переривання сигналом і не допускають такого переривання. В останньому випадку сигнал буде терпляче чекати нормального пробудження процесу.

Інформація про основні сигнали представлена в табл. 18.

Таблиця 18. Основні сигнали

Сигнал	Стандартна обробка	Значення
SIGTERM 15	Завершення процесу	Стандартний сигнал, посланий для зупинки процесу.
SIGHUP 1	Завершення процесу	Відключився термінал (або закрито термінальне вікно). Сигнал посилається всім не фоновим процесам, пов'язаним з відповідною термінальної лінією.
SIGKILL 9	Завершення процесу	Не перехоплює сигнал, що дозволяє завершити будь-який процес.
SIGILL 4	Завершення процесу і скидання образу пам'яті	На центральний процесор була послана заборонена інструкція. Це могло бути наслідком неприпустимого переходу в машинному коді програми, наприклад, спроби виконати рядок даних.
SIGTRAP 5	Завершення процесу і скидання образу пам'яті	Була встановлена пастика точки переривання процесу. Цим управляє системний виклик <code>ptrace</code> , який корисний для налагодження.
SIGFPE 8	Завершення процесу і скидання образу пам'яті	Була спроба виконати заборонену арифметичну операцію, наприклад, взяття логарифма негативного числа або ділення на 0.
SIGBUS 10	Завершення процесу і скидання образу пам'яті	Помилка на шині вводу-виводу. Звичайно це є результатом спроби виконати читання або запис поза межами пам'яті програми.
SIGSEGV 11	Завершення процесу і скидання	Це порушення сегментації - прокляття розробників програм! Воно означає, що ви спробували

Таблиця 18. Основні сигнали

Сигнал	Стандартна обробка	Значення
	образу пам'яті	отримати доступ до сегменту пам'яті забороненим чином. Може бути, це було присвоювання значення частини сегмента коду або читання з нульового адреси.
SIGPIPE 13	Завершення процесу	Програма спробувала виконати читання або запис в програмний канал, інший кінець якого вже завершив роботу. Цей сигнал допомагає завершити роботу конвеєра, коли одна з його команд дала збій.
SIGALRM 14	Завершення процесу	Програміст може встановити будильник, щоб дозволити вам в певний момент часу виконати яке-небудь дію.
SIGCHLD 18	Ігнорується	Спочатку це був сигнал завершення роботи дочірнього процесу, але зараз він означає зміну стану дочірнього процесу.
SIGTSTP 24	Зупинка процесу	Це запит від терміналу на зупинку процесу. Здійснення цього сигналу процесу відбувається при натисканні комбінації клавіш Ctrl-Z .
SIGCONT 25	Ігнорується	Цей сигнал вказує процесу на відновлення його роботи. Процесу надсилається або команда fg , або bg , а командний інтерпретатор виконує внутрішній системний виклик wait для привілейованого процесу, або не виконує його для фонового процесу.

Детальна інформація про сигнали представлена на сторінках довідкового посібника **signal**.

Процес, за допомогою системного виклику **signal()**, може задати нестандартний обробник будь-якого сигналу, крім **SIGKILL (9)**.

Виконання сигналів. Для посилки сигналів з командного інтерпретатора використовується команда **kill**. Вона має такий синтаксис:

```
kill [ -сигнал ] pid ...
```

Ця команда посилає вказаний сигнал (за замовчуванням - **SIGTERM**) всіх процесів із зазначеними ідентифікаторами. Посилати сигнал можна і не існуючому процесу - видається попередження, але іншим процесам сигнал посилається. Посланий сигнал задається по імені без префікса **SIG** або за номером, наприклад:

```
$ echo $$  
3697  
$ kill -STOP 3697
```

У результаті поточний сеанс зависає.

4.2. Командний інтерпретатор

Користувач ОС UNIX спілкується з системою через *командний інтерпретатор* (shell). Через нього відбувається доступ до команд, файлової системи і інших функцій ядра UNIX. Це звичайна програма (тобто не входить у ядро операційної системи UNIX). Її можна замінити іншою або використовувати декілька різних версій. Найбільш відомі такі версії:

- sh** Класичний інтерпретатор версії UNIX V7, інакше званий на прізвище автора *Bourne shell*.
- Ksh** Інтерпретатор *Korn shell*, що доповнює класичний shell можливостями роботи з завданнями користувача, історією роботи і дозволяє редагувати командний рядок за допомогою команд, аналогічних **vi**. Є фактично стандартом для POSIX-сумісних систем, зокрема, UNIX System V.
- Csh** Стандартний інтерпретатор BSD UNIX і похідних від нього систем. Відрізняється поліпшеними діалоговими можливостями, способом привласнення та експортування змінних в середу, керуючими конструкціями і рядом інших

моментів; теж підтримує історію і редактування командного рядка. Головне і, на мою думку, негативне його відмінність від інших інтерпретаторів, - це свої керуючі конструкції, що не збігаються з **sh**.

Bash Вільно розповсюджуваний у вигляді вихідних текстів інтерпретатор, званий "*Bourne another shell*", що поєднує все краще з інших інтерпретаторів із зручними можливостями редактування командного рядка та роботи з історією команд. В даний час - фактичний стандарт.

У рамках цього курсу ми будемо розглядати, в основному, засоби, що не виходять за межі можливостей командних інтерпретаторів **sh** і **ksh** (в класичній версії 1988 року).

4.2.1. Структура командного рядка

Командні рядки розглядаються по одній і мають певну структуру. Щоб зрозуміти її, розглянемо ряд синтаксичних визначень:

<Пробіл>:: =	<Символ пропуску> <символ табуляції>
<Ім'я>:: =	<Буква або підкреслення> {<допустимий символ імені>}
<Буква або підкреслення>:: =	<Буква> _
<Допустимий символ імені>:: =	<Буква> <цифра> _
<Параметр>:: =	<Ім'я> <цифра> * @ # ? - \$!
<Слово>:: =	<Не пробіл> {<не пробіл>}
<Проста команда>:: =	<Слово> {<пробіл> <слово>}

Отже, *проста команда* - це послідовність слів через пробіл. Натискання клавіші **Enter** при введенні команди або новий рядок при обробці сценарію є для командного інтерпретатора ознакою завершення команди. Вона обробляється і виконується.

Значенням простої команди є її *статус виходу* (див. далі) у разі нормальног завершення або (вісімкове) 200 + статус при ненормальному завершенні.

Приклад простої команди:

```
$ who
oracle pts/000 Aug 20 10:08
root console Aug 20 09:03
intdbi pts/004 Aug 20 12:45
$
```

З простих команд будуються більш складні конструкції: *конвеєри та списки*.

```
<Конвеєр> ::= <Команда> { | <команда> }

<Список> ::= <Конвеєр> {<роздільник> <конвеєр>} [<термінатор ко-
манди>]

<Роздільник> ::= & & | || | <Термінатор команди>

<Термінатор команди> ::= ; | &
```

Конвеєр - це послідовність однієї або більше команд, розділених |. Стандартний вихідний потікожної команди, крім останньої, з'єднується за допомогою *програмного каналу* зі стандартним вхідним потоком наступної команди. Кожна команда виконується як окремий процес; інтерпретатор очікує закінчення останньої команди. Статусом виходу конвеєра є статус виходу його останньої команди. Ось приклад простого конвеєра:

```
$ ls | tee save | wc
15      15      100
$
```

Список - це послідовність одного або більше конвеєрів, **розділених**, ;, &, & & чи || i, можливо, що **закінчується**, або &. З цих чотирьох **символів**, ; і & мають рівний пріоритет, який нижче, ніж у & & i || (ці символи теж мають рівний пріоритет). Крапка з комою (;) викликає послідовне виконання попереднього конвеєра (тобто командний інтерпретатор очікує закінчення конвеєра перед виконанням будь-яких команд, наступних за крапкою з комою). Амперсанд (&) викликає асинхронне виконання попереднього конвеєра (тобто командний інтерпретатор не очікує закінчення роботи конвеєра). Символ & & (||) веде до того, що наступний за ним список виконується тільки в тому випадку, коли попередній конвеєр повернув нульової (нену-

льовий) статус виходу. У список може входити довільну кількість перекладів рядків і точок з комою, що поділяють команди.

Тепер можна дати загальне визначення команди:

<Команда>::=

<Проста команда> |
<Оператор управління> |
<Визначення функції> |
<Список> | (<список>) | {<список>;}

Список у круглих дужках виконується у породженному командному інтерпретаторі. Круглі дужки зазвичай використовують для групування команд.

Список у фігурних дужках виконується в поточному командному інтерпретаторі, без породження додаткового процесу, і заміщає образ командного інтерпретатора (це аналог системного виклику **exec**).

Оператори управління та синтаксис визначення функцій розглядається далі.

Розглянемо приклад складної команди:

```
$ (sleep 5; date) & date  
[1] 1148  
Wed Aug 20 15:00:11 ???? 1997  
$ Wed Aug 20 15:00:16 ???? 1997
```

Фоновий процес починається, але відразу "засинає"; тим часом друга команда **date** видає поточний час, а інтерпретатор - запрошення для введення нової команди. Через п'ять (приблизно, залежить від завантаження системи і т. п.) секунд припиняється виконання команди **sleep** і перша команда **date** видає новий час.

Ряд символів, як було описано вище, мають для командного інтерпретатора спеціальне значення - це **метасимволи**. Вони описані в табл. 23.

Метасимволи не входять до команди і обробляються в декілька проходів до початку виконання реальних програм.

Таблиця 23. Метасимволи командного інтерпретатора

Метасимвол	Інтерпретація
>	prog> file - переключити стандартний вихідний потік в файл

Таблиця 23. Метасимволи командного інтерпретатора

Метасимвол	Інтерпретація
<code>>></code>	<code>prog>> file</code> - додати стандартний вихідний потік до файлу
<code><</code>	<code>prog <file</code> - витягти стандартний вхідний потік з файлу
<code> </code>	<code>p1 p2</code> - передати стандартний вихідний потік p1 як стандартний вхідний потік p2
<code><<str</code>	"Документ тут": стандартний вхідний потік задається в наступних рядках до рядка, що складається тільки з символів str . За умовчанням в даних інтерпретуються метасимволи <code>\</code> , <code>\$</code> і <code>``</code> . Якщо необхідно запобігти в даних інтерпретацію всіх метасимволів, необхідно екранувати рядок str , випередивши зворотною косою або взявши в подвійні або поодинокі лапки.
<code>*</code>	Визначає в імені файлу будь-який рядок з нуля або більше символів
<code>?</code>	Визначає будь-який символ в імені файлу
<code>[abc]</code>	Визначає будь-який символ з <code>[abc]</code> в імені файлу, при цьому допускаються діапазони, що задаються за допомогою дефіса <code>-</code> . Якщо першим символом після <code>[</code> є <code>!</code> , З цією конструкцією зіставляється будь-який символ, що не входить у квадратні дужки.
<code>;</code>	Роздільник команд: <code>p1; p2</code> - виконати p1 , потім p2 .
<code>&</code>	Виконує попередню команду у фоновому режимі
<code>`...`</code>	Ініціює виконання команд (и) у <code>... ; ...`</code> замінюється на отриманий в результаті виконання стандартний вихідний потік
<code>(...)</code>	Ініціює виконання команд (и) <code>...</code> у порожньому командному інтерпретаторі
<code>\$1, \$2,.. \$9</code>	Замінюються аргументами командного файлу
<code>\$var</code>	Значення змінної (<i>ключового параметра</i>) var в сеансі
<code>\$ {var}</code>	Значення var : виключає колізії у разі конкатенації змінної з наступним текстом

Таблиця 23. Метасимволи командного інтерпретатора

Метасимвол	Інтерпретація
\	\C - використовувати безпосередньо символ c , \ переведення рядка – відкидається
'...'	Безпосереднє використання того, що в лапках
"..."	Безпосереднє використання, але після того, як будуть інтерпретовані метасимволи \$, `...` і \
#	Початок коментаря
var=value	Присвоює значення value змінній var
p1 && p2	Виконати p1 ; у разі успіху виконати p2
p1 p2	Виконати p1 ; у разі невдачі виконати p2

Примітка: Більшість метасимволів розглядається по ходу викладу. Тут мова йде про тих з них, які використовуються для генерації імен файлів і екранування.

Перед виконанням команди кожне слово-аргумент команди проглядається в пошуках метасимволів ,? або [шаблон]. Якщо в слові є один з цих символів, слово розглядається як *шаблон* (зверніть увагу на синтаксичні відмінності від шаблонів **ed**). Таке слово замінюється відсортованими в алфавітному порядку іменами файлів, які відповідають шаблону. Якщо ні одне їх імені файлів не відповідає шаблону, слово залишається без змін. Символ . на початку імені файлу або відразу після /, а також сам символ /, повинні зіставлятися явно.

При такій кількості метасимволів інтерпретатору необхідно мати можливість екранувати спеціальний символ від інтерпретації. Для цього можна використовувати апострофи, лапки або зворотну косу. При цьому лапки одного виду можуть екранувати лапки іншого виду. Зверніть увагу, що лапки "", на відміну від апострофів, не екранують рядок повністю - інтерпретатор заглядає всередину лапок в пошуках \$, `...` і \.

У лапках можуть міститися переклади рядків, пробіли, табуляції, символи ; , &, (,), |, ^, <>. Задаючи ім'я файлу у лапках, можна створити файли з такими нетривіальними символами в іменах. Втім, робити це не рекомендується, тому що працювати з ними буде явно незручно.

4.2.2. Створення сценаріїв

Маючи послідовність команд, яку доведеться багаторазово використовувати, перетворимо її для зручності в "нову" команду зі своїм ім'ям і будемо використовувати її як звичайну команду. Припустимо, що вам як адміністратора належить часто підраховувати кількість користувачів, що працюють зараз в системі, за допомогою простого конвеєра

```
$ who | wc -l
```

і для цієї мети потрібна нова програма **nu**.

Першим кроком буде створення звичайного текстового файлу, що містить текст конвеєра. Можна скористатися вашим улюбленим текстовим редактором, а можна проявити винахідливість:

```
$ echo 'who | wc -l' > nu
```

Інтерпретатор є такою ж програмою, як редактор, **who** або **wc**. Раз це програма, її можна викликати і перемкнути її вхідний потік. Отже, запускаємо інтерпретатор з вхідним потоком, що надходить з файлу **nu**, а не з терміналу:

```
$ who
oracle      pts000      Aug 20 10:08
root        console     Aug 20 09:03
intdbi      pts004      Aug 20 12:45
```

```
$ cat nu
```

```
who | wc -l
```

```
$ sh < nu
```

```
3
```

```
$
```

Результат вийшов таким же, як і при введенні команди з терміналу. Як і багато інших програм, інтерпретатор обробляє файл, якщо він зазначений як аргумент; ви з тим же успіхом могли б задати:

```
$ sh nu
```

Насправді, цей виклик відрізняється, так як вхідний потік **sh** залишається пов'язаним з терміналом.

Не хотілося б вводити **sh** кожен раз, крім того, це створює відмінність між командами, написаними на мові shell, та іншими виконуваними файлами. Тому, якщо текстовий файл призначений для виконання, то інтерпретатор вважає, що він складається з команд (інтерпретатор **csh** вимагає, щоб файл починався з #).

Примітка: Якщо в першому рядку виконуваного текстового файлу зазначено:

```
#!/Повний_шлях_до_програми опції_програми
```

то даний текстовий файл буде інтерпретуватися вказаною програмою, при виклику якої будуть встановлені задані опції. Так можна виконувати, наприклад, програми командного інтерпретатора **csh**, не виходячи з **sh**. Точно так само можна автоматично викликати і інтерпретатори інших мов сценаріїв, наприклад, Perl.

Все, що вам потрібно зробити, це оголосити файл **nu** виконуваним, задавши:

```
$ chmod +x nu
```

а потім ви зможете викликати його за допомогою

```
$ nu
```

Насправді, при виконанні команди **nu** створюється новий процес інтерпретатора (породжений інтерпретатор), який і виконує команди, що містяться в **nu**. Щоб команда виконувалась у тому ж інтерпретаторі, необхідно поставити перед викликом команди крапку (.). Зауважте, що

```
$ . nu
```

виконується швидше, ніж простий виклик **nu**.

У більшості програм треба використовувати аргументи - файли, прапори і т. д. Відповідно до синтаксисом командного рядка, це можна зробити, передаючи їх після імені команди через пробіли.

Припустимо, необхідно програму **cx** для установки права доступу до файлу на виконання, так що

```
$ cx nu
```

є скорочений запис для

```
$ chmod +x nu
```

Створити такий сценарій нескладно. Залишається тільки одне питання - як отримати в програмі доступ до імені файлу-аргументу. Для цього в командному інтерпретаторі використовуються *позиційні параметри*.

При виконанні командного файлу, кожне входження **\$1** замінюється першим аргументом, **\$2** – другим і так далі до **\$9**. **\$0** замінюється ім'ям виконуваної команди. Тому, якщо файл **сx** містить рядок **chmod +x \$1**, то при виконанні команди **\$ сx** її породжений інтерпретатор замінить **\$1** першим аргументом команди **nu**.

Значення позиційним параметрами присвоюються при виклику сценарію, при виклику функції в сценарії або явно, за допомогою команди **set**.

Як бути, якщо потрібно працювати з декількома аргументами, наприклад, змусити програму **сx** робити виконуваними декілька файлів? Можна включити дев'ять аргументів на командний файл (явно можна задавати тільки дев'ять аргументів, так як конструкція **\$ 10** розпізнається як "перший аргумент, за яким слід 0"):

```
chmod +x $1 $2 $3 $4 $5 $6 $7 $8 $9
```

Якщо користувач такого командного файлу задасть менше дев'яти аргументів, то що залишилися виявляться порожніми рядками і тільки справжні аргументи будуть передані **chmod** породженим інтерпретатором. Але таке рішення не підходить, якщо кількість аргументів перевищує дев'ять.

Інтерпретатор надає спеціальний параметр **\$***, який замінюється усіма аргументами команди, незалежно від їх кількості. Тобто правильне визначення **сx** буде таким:

```
chmod +x $*
```

Всі позиційні і спеціальні параметри, підтримувані командним інтерпретатором, представлені в табл. 24.

Таблиця 24. Позиційні та спеціальні параметри командного інтерпретатора

Параметр	Призначення
\$0	Ім'я виконуваної команди
\$1, \$2, .. \$9	Замінюються аргументами командного файлу
\$#	Кількість аргументів

\$_*	Всі аргументи, що передаються інтерпретатору. "\$_*" є єдиним словом, утвореним з усіх аргументів, об'єднаних разом з пробілами.
\$_@	"\$_@" Ідентично всіх аргументів: пробіли в аргументах ігноруються, і виходить список слів, ідентичних вихідним аргументам.
\$_-	Прапори, встановлені в інтерпретаторі.
\$_?	Значення, повернене останньої виконаної командою (<i>статус виходу</i>).
\$_\$	Номер процесу інтерпретатора.
\$_!	Номер процесу останньої команди, запущеної асинхронно за допомогою &.

4.2.3. Змінні і привласнення

Подібно до більшості мов програмування, командний інтерпретатор має змінні, які називають ще *ключовими параметрами* (доступ до них здійснюється по *імені* - ключу). Змінні можна створювати, змінювати і вибирати їх значення. Для надання значення змінної застосовується конструкція: **змінна=значення**.

Зверніть увагу на відсутність пробілів до і після знака присвоювання. Згадайте, що командний інтерпретатор вважає пробіли роздільниками слів. Якщо поставити пробіл після знака присвоювання, то інтерпретатор не змінить значення змінної, але буде шукати команду з ім'ям **значення**.

Значення, що присвоюється, має виражатися одним словом, і його слід взяти в лапки, якщо воно містить метасимволи, які не потрібно обробляти.

Створення (визначення) змінної відбувається при першому присвоенні їй значення. Змінні не потрібно явно оголошувати до моменту їх використання. Якщо змінна не оголошена (не отримала значення), при зверненні до неї буде отримана порожній рядок. Всі змінні в командному інтерпретаторі - рядкові, тому тип змінної задавати не треба. Деякі команди інтерпретують рядки як числа.

Багато змінних, як, наприклад, **PATH**, мають спеціальне значення для інтерпретатора. За традицією, такі змінні називають *вбудованими* та позначають великими

ми літерами, а звичайні змінні - малими. Основні вбудовані змінні представлені в табл. 25.

Таблиця 25. Вбудовані змінні командного інтерпретатора

Змінна	Значення
\$HOME	Початковий каталог користувача.
\$PATH	Шлях для пошуку виконуваних команд.
\$CDPATH	Шлях пошуку для команди cd .
\$IFS	Список символів, що розділяють слова в аргументах
\$MAIL	Файл поштової скриньки. Командний інтерпретатор Про надходження пошти у вказаний файл.
\$MAILCHECK	Ця змінна визначає, як часто (у секундах) інтерпретатор перевірятиме надходження пошти до файлу, який визначається змінної MAIL . За замовчуванням прийняте значення 600 секунд. При установці в 0, інтерпретатор буде перевіряти пошту передожною видачею рядка-запрошення.
\$PS1	Рядок-запрошення, за замовчуванням прийнята '\$'
\$PS2	Рядок-запрошення при продовженні командного рядка, за замовчуванням прийнята '>'

Типовим прикладом використання змінних є зберігання в них довгих рядків, таких як імена файлів:

```
$ pwd  
/home/intdbi/dosapps/doc/book/unix/shell  
$ dir=`pwd`  
$ cd  
$ ln $dir/cx .  
...  
$ cd $dir  
$ pwd  
/home/intdbi/dosapps/doc/book/unix/shell
```

Вбудована команда інтерпретатора **set**, при виклику без параметрів, показує значення всіх визначених змінних.

Присвоєння значення змінної при виклику. Змінні називають *ключовими параметрами*, оскільки їм можна передавати значення на ім'я при виклику. Розглянемо приклад:

```
$ echo 'echo $x' >echox  
$ cx echox  
$ echo $x  
Hello  
$ echox  
$ x=Hi echox  
Hi
```

За замовчуванням ключові параметри можна передавати тільки до назви команди. Якщо встановити прапор **інтерпретатора-k** (**set-k**), то можна буде передавати значення змінним і після імені команди.

Експорт змінних. Кожен екземпляр командного інтерпретатора має свій набір змінних, що розміщаються в окремій області пам'яті. Якщо необхідно, щоб певна змінна в породжених процесах мала конкретне значення, необхідно *експортувати* її. Така змінна називається змінною середовища.

Для всіх експортованих змінних при запуску породженого процесу створюються їх локальні копії з тими ж значеннями. Розглянемо приклад:

```
$ x>Hello  
$ export x  
$ PS1='new$ ' sh  
new$ echo $x  
Hello  
new$ x='Good Bye'  
new$ echo $x  
Good Bye  
new$ exit  
$
```

```
$ echo $x
```

```
Hello$
```

Зміна значення змінної у породженному інтерпретаторі не впливає на її значення в батьківському інтерпретаторі.

Для перегляду значень всіх змінних середовища призначена команда **env**.

4.2.4. Оператори командного інтерпретатора

Цикли в командному інтерпретаторі. Командний інтерпретатор підтримує циклічну обробку. Найчастіше на практиці використовується цикл **for** - цикл за списком слів. Він описаний в наступному підрозділі.

Зверніть увагу, що виділені **жирним** ключові слова повинні бути первістком команди, тобто первістком словом в рядку або йти відразу після крапки з комою.

Цикл **for** має наступний синтаксис:

<Цикл for>:: = **for** <ім'я змінної> [in <список слів>] **do** <команди> **done**

<Список слів>:: = <Слово> {<пробіл> <слово>}

<Команди>:: = <Команда> {<; або новий рядок> <команда>}

Змінна послідовно отримує значення чергового слова зі списку, і для цього значення виконуються команди в тілі циклу. Цикл завершується, коли пройдено весь список слів. За замовчуванням як список слів використовуються аргументи командного рядка.

Розглянемо декілька прикладів таких циклів:

```
$ for i in 1 2 3 4 5  
> do  
>     echo $i  
> done
```

Зверніть увагу, що командний інтерпретатор розпізнає цикл, видає вторинне запрошення, і виконує цикл тільки після його завершення ключовим словом **done**.

Список слів для циклу звичайно породжується динамічно. Наприклад, шляхом розкриття шаблонів імен файлів:

```
$ for i in *. c *. h
> do
>   echo $i
>   diff -b old/$i $i
>   echo
> done | pr -h "diff `pwd`/old `pwd`" | lp &
[4] 1430
```

Можна також породжувати його командою, підставляючи її результати:

```
$ for i in `pick *. c *. h`
> do
>   echo $i:
>   diff -b old/$i $i
> done | pr | lp
```

Оператори циклу while і until. Командний інтерпретатор підтримує також традиційні цикли за умовою з таким синтаксисом:

<Оператор while> ::= **while** <команди> **do** <команди> **done**

<Оператор until> ::= **until** <команди> **do** <команди> **done**

Виконуються *команди*, що задають умова, і перевіряється код повернення останньої з них. Якщо це нуль (**істина**), виконуються команди в тілі циклу **while** або завершується виконання циклу **until**. Якщо це не нуль (**хибність**), завершується робота циклу **while** або виконується чергова ітерація циклу **until**.

На основі цих циклів часто створюються *програми-“спостережниці”*, що працюють нескінченно:

```
$ cat watchfor
# watchfor: watching for log ins and log outs. .
PATH=/usr/bin
new=/tmp/wfor1. $$
old=/tmp/wfor2. $$

>$old          # створює порожній файл

while :           # нескінчений цикл
do
  who >$new
```

```

diff $old $new
mv $new $old
sleep 60
done | awk ' />/ { $1 = "in: "; print }
           /</ { $1 = "out: "; print }'
$
```

Оператор вибору. Командний інтерпретатор підтримує виконання того чи іншого блоку команд залежно від значення деякого слова. Для цього пропонується оператор **case** з таким синтаксисом:

<Оператор вибору>:: = **case** <слово> **in**
 <Опис варіанта>) <команди>;;
 { <Опис варіанта>) <команди>; }
 esac

<Опис варіанта>:: = <Шаблон> { | <шаблон> }
<Команди>:: = <Команда> { <роздільник> <команда> }
<Роздільник>:: = <Переведення рядка> |;

Слово (зазвичай – значення змінної) порівнюється послідовно з шаблонами. Якщо сталася зіставлення (*за правилами зіставлення шаблонів імен файлів*) виконуються команди, які відповідають даному варіанту і оператор завершується. Врахуйте, що шаблон *) зіставляється з будь-яким словом, і, тим самим, задає варіант за замовчуванням.

У шаблонах оператора **case** символи . та / , на відміну від шаблонів імен файлів, не обов'язково ставити явно.

Умовний оператор. Командний інтерпретатор підтримує умовний оператор такого загального вигляду:

<Умовний оператор>:: = **if** <команди> **then** <команди>
 { **elif** <команди> **then** <команди> }
 [**else** <команди>]
 fi

Виконуються команди після **if** і перевіряється код повернення останньої з них. Якщо це 0 (**істина**) виконуються відповідні команди після **then** і виконання опера тора завершується. Якщо ж це не 0 (**хибність**), то при наявності конструкцій **elif** виконуються послідовно відповідні команди-умови і, якщо вони повертають код 0, команди після **then**, а потім оператор завершується. Якщо ні одна з умов не було істинним, виконуються команди в частині **else** і оператор завершується.

В якості умови в умовному операторі може використовуватися будь-яка команда. Однак, є стандартна команда для перевірки умов у традиційному розумінні. Це команда **test**, представлена в наступному розділі.

Перевірка умов у командному інтерпретаторі. Команда **test** має синтаксис:

<Команда test>:: = **test** <вираз> | [<вираз>]

Вираз будується з примітивів, представлених в табл., при необхідності, за допомогою таких операторів:

- !** Унарний оператор заперечення.
- A** Бінарний оператор "і".
- O** Бінарний оператор "або".
- (<Вираз>)** Взяття угруповання. Врахуйте, що дужки розпізнаються командним інтерпретатором, тому їх треба брати в лапки.

Таблиця 26. Основні примітиви команди test

Примітив	Умова
-r файл	файл існує і доступний для читання
-w файл	файл існує і доступний для запису
-x файл	файл існує і є виконуваним
-f файл	істина, якщо файл існує і є звичайним файлом (не каталогом)
-d файл	файл існує і є каталогом
-h файл	файл існує і є символічним зв'язком

Таблиця 26. Основні примітиви команди test

Примітив	Умова
-s файл	файл існує і не порожній
-t [дескриптор]	істина, якщо відкритий файл із зазначенім дескриптором (за замовчуванням, 1) асоційований з терміналом
-z s1	істина, якщо рядок s1 має нульову довжину
-n s1	істина, якщо рядок s1 має ненульову довжину
s1 = s2	істина, якщо рядки s1 і s2 ідентичні
s1 != s2	істина, якщо рядки s1 і s2 не збігаються
s1	істина, якщо рядок s1 непустий
n1 -eq n2	порівняння цілих чисел на рівність (=). Можна використовувати також і інші порівняння: -ne (! =), -gt (>), -ge (>=), -lt (<) і -le (<=).

Розглянемо приклад використання умовного оператора і команди **test**:

```
$ cat which
# which cmd: безпечна версія сценарію для видачі каталогу,
# з якого буде викликатися виконувана програма
opath=$PATH
PATH=/usr/bin

# Це гарантує використання справжніх команд
# echo, sed і test в будь-якому разі!
case $# in
0)    echo 'Usage: which command' 1>&2; exit 2
esac

for i in `echo $opath | sed 's/^:/./ :/
                                         s/:/:/.. :/g
                                         s/:$/.. /
                                         s:/ /g'`^
do
    if test -x ${i}/${1}
    then
```

```

echo $i/$1
exit 0          # команда знайдена

fi

done
exit 1          # не знайдено

$ which sed
. /sed

$ which which
. /which

```

Обчислення в командному інтерпретаторі. Обчислення можна виконувати за допомогою будь-якої програми, що сприймає свої параметри як вираз, значення якого необхідно обчислити, і видає результат обчислення в стандартний вихідний потік. Одна з таких програм, **expr** , розглянута далі. Але сучасні командні інтерпретатори включають вбудовану команду для виконання найпростіших арифметичних дій. Це команда **let** :

<Команда **let**>:: = **let** <аргумент> {<аргумент>}

Ось як її можна використовувати:

```

$ let a=5
$ echo $a
5
$ let a=a*a+34/2
$ echo $a
42

```

Зверніть увагу, що якщо навколо знаку рівності йдуть пробіли, необхідно брати вираз в лапки. Команда **let** вимагає, щоб вираз було одним словом. Крім того, для звернення до значення змінної в цій команді *не потрібно* використовувати метасимвол **\$**.

Команда expr. Однією зі стандартних програм-калькуляторів є програма **expr** . Її основні оператори представлені в табл. 27.

Таблиця 27. Основні оператори, які розпізнає команда `expr`

Оператор	Результат
Вир1 \ вир2	Повертає значення першого виразу, якщо воно не пусте і не дорівнює 0, інакше, повертає значення другого виразу.
Вир1 \ & вир2	Повертає значення першого виразу, якщо обидва вирази - не порожні і не рівні 0, інакше, повертає 0.
Вир1 {+, -} вир2	Складає або віднімає цілочисельні аргументи.
Вир1 {\ *, /, %} вир2	Множить, ділить або повертає залишок від ділення для цілочисельних аргументів.
Length рядок	Повертає довжину рядка.

Розглянемо простий приклад обчислення за допомогою `expr`:

```
$ a=5
$ echo $a
5
$ a=`expr $a \* $a + 34 / 2`
$ echo $a
42
```

Зверніть увагу, що між елементами висловлювання треба вказувати пробіли.

4.2.5. Робота з сигналами та інформацією від користувача

Перехоплення і обробка сигналів. Сигнал – це засіб міжпроцесної взаємодії в Unix-подібних, та інших операційних системах що сумісні зі стандартом POSIX. Сигнал являє собою асинхронне повідомлення що посилається процесу щоб проінформувати його про подію яка відбулася. Коли процес отримує сигнал, операційна система перериває хід його виконання, і запускає підпрограму обробки цього сигналу. Якщо в програмі явно не задана реакція на сигнал, запускається його стандартний обробник.

Натискання певної комбінації клавіш в терміналі в якому запущений процес, змушує систему відправляти певні сигнали, наприклад:

- **Ctrl-C** посилає сигнал **INT (SIGINT)**, за замовчуванням це змушує процес завершитись.
- **Ctrl-Z** посилає сигнал **TSTP (SIGTSTP)**, за замовчуванням це змушує процес призупинити виконання.
- **Ctrl-** посилає сигнал **QUIT (SIGQUIT)**, за замовчуванням, це змушує процес завершити роботу, та зберегти дамп.

Стандартні комбінації клавіш можна змінити командою **stty**.

У програмах командного інтерпретатора можна перехоплювати і обробляти сигнали. Для цього використовується команда **trap**, що встановлює з моменту виконання обробник у вигляді послідовності команд (*одним словом*) для всіх перерахованих сигналів. Ця команда має такий синтаксис:

<Оператор trap>:: = **trap** <послідовність команд> <список сигналів>

<Список сигналів>:: = <Сигнал> {<пробіли> <сигнал>}

Розглянемо приклад реалізації команди **nohup** , що дозволяє запустити програму так, щоб вона продовжувала працювати при вимиканні терміналу:

```
$ cat nohup
# nohup: no kill and hangup
trap "" 1 15
if test -t 2>&1
then
    echo "Redirect stdout to 'nohup.out'"
    exec nice -5 $* >>nohup.out 2>&1
else
    exec nice -5 $* 2>&1
fi
$
```

Запит інформації у користувача. Командний інтерпретатор дозволяє, при необхідності, запитувати у користувача інформацію, яка міститься у зазначену змінну. Для цього використовується команда **read**:

```
$ read greeting
Hello, world!
$ echo $greeting
Hello, world!
$
```

На практиці має сенс перед запитом видати запрошення за допомогою команди **echo**. Наприклад, ось так:

```
$ cat pick
# pick: select arguments

PATH=/bin:/usr/bin

for i                      # for each argument, try $*, "$*" and
"$@"
do
    echo -n "$i? " > /dev/tty
    read responce
    case $responce in
        y*)  echo $i;;
        q*)  break
        esac
done </dev/tty
$
```

Представлена вище програма **pick** видає кожне зазначене в якості аргументу слово в готельній рядку зі знаком питання і вимагає від користувача підтвердити необхідність його видачі в стандартний вихідний потік. Оскільки ця програма може використовуватися в інших сценаріях, вхідний і вихідний потоки яких перенаправлені, вона взаємодіє безпосередньо з поточним терміналом (через пристрій **/dev/tty**).

4.2.6. Функції в командному інтерпретаторі

Стандартним способом розбиття програм на модулі в командному інтерпретаторі є оформлення необхідних дій у вигляді окремого виконуваного файлу з програмою командного інтерпретатора - створення нової команди. Тим не менше, для деяких модулів такий підхід може виявитися неефективним і надлишковим, так як модулі можуть не представляти самостійного значення поза програмою, в якій вони використовуються. Тому в сучасних версіях командних інтерпретаторів пропонується можливість створювати і викликати функції.

Синтаксис визначення функції. Для визначення функцій використовується ключове слово **function**. Функції читаються і зберігаються внутрішньо командним інтерпретатором. Функції виконуються як команди, причому аргументи передаються як *позиційні параметри*. Синтаксис визначення функції такий:

```
<Визначення функції>::= function <ідентифікатор> {<спісок команд>} |  
                                <Ідентифікатор> () {<спісок команд>}
```

де **спісок команд** задає команди, які виконуються як тіла функції. Команди зазвичай розділяються крапкою з комою або перекладами рядків.

Виконання та використання функцій. Функції виконуються викликав їх процесом і використовують всі його файли і поточний робочий каталог. Сигнали, що перехоплюють викликає процесом, всередині функції обробляються стандартним чином. Сигнали, не перехоплює або ігноровані функцією, припиняють її виконання і передаються викликала команді.

Зазвичай змінні спільно використовуються викликає програмою і функцією. Проте, спеціальна команда **typeset**, використовувана всередині функції, дозволяє визначати локальні змінні, область дії яких - поточна функція і всі викликані нею функції.

Для виходу з функції використовується спеціальна команда **return**. У випадку помилки в функції, управління передається викликає команді.

Ідентифікатори певних функцій можна отримати за допомогою опцій **-f** або **+f** спеціальної команди **typeset**. Текст функцій показується при використанні опції **-f**. Визначення функції можна скасувати за допомогою опції **-f** спеціальної команди **unset**.

Зазвичай при виконанні сценарію командним інтерпретатором ніякі функції не задані. Опція **-xf** команди **typeset** дозволяє експортувати функцію для використання сценаріями, виконуваними без окремого виклику інтерпретатора. Функції, які повинні бути визначені для всіх викликів інтерпретатора, необхідно ставити в файлі початкового запуску за допомогою опцій **-xf** команди **typeset**.

Розглянемо класичний приклад ітеративної реалізації функції обчислення факторіала:

```
# test. sh - test shell functions

factorial () {
    typeset i
    typeset n

    i=1; n=1
    while [ $i -le $1 ]
    do
        let n=n*i
        let i=i+1
    done
    echo $n
    return
}

a=`factorial $1`
echo $a
```

При виклику ця програма, як і очікувалося, обчислить факторіал свого першого параметра:

```
$ test.sh 5
120
```

Часто у виді функцій оформляється видача повідомень про параметри виклику програми. У будь-якому випадку, якщо завдання може бути розбито на підзадачі, вирішення цих підзадач має сенс оформлення у вигляді окремої команди, якщо вони корисні не тільки в контексті розв'язуваної задачі, або у вигляді функції в іншому випадку.

4.2.7. Управління завданнями

Управління завданнями - це механізм для відстеження процесів, які породжуються з поточного сеансу роботи з терміналом. Можна запустити будь-яке число завдань. Вони можуть працювати, завершитися або перебувати в інших станах. Для управління завданнями можна використовувати декілька команд, щоб простежити результати їх роботи або вимагати від системи повідомлення про закінчення завдання.

Файли початкового запуску командного інтерпретатора. Стандартна середовище для роботи командних інтерпретаторів задається у *файліах початкового запуску*, які автоматично виконуються в початковому інтерпретаторі за допомогою команди крапка (.), тобто без породження. Файли початкового запуску розміщуються в початковому каталозі користувача і називаються **Profile** (**sh**, **ksh**) або **bash_profile** (**bash**). Змінні, складові середу, у файлі початкового запуску треба експортувати.

Розглянемо приклад вмісту файлу початкового запуску:

```
INFORMIXDIR=/usr/inf. 731
INFORMIXSERVER=onarturo7
ONCONFIG=onconfig
SQLHOSTS=sqlhosts
PATH=$PATH:$INFORMIXDIR/bin
LD_LIBRARY_PATH=$LD_LIBRARY_PATH:$INFORMIXDIR/lib/esql:$INFORMIXDIR/lib
DB_LOCALE=ru_ru. 8859-5
CLIENT_LOCALE=ru_ru. 8859-5
KAIOON=1
NODEFDAC=YES
DBDATE=DMY4/
DBTIME='%H:%M:%S %d/%m/%Y'
DBMONEY=. 4
export KAIOON
export INFORMIXDIR INFORMIXSERVER LANG PATH ONCONFIG SQLHOSTS
export DBDATE DBTIME DBMONEY NODEFDAC
export DB_LOCALE CLIENT_LOCALE
```

Запуск завдання у фоновому режимі. Фоновий режим дозволяє продовжити використання сеансу роботи з терміналом, поки виконується команда. Для запуску команди у фоновому режимі, досить до команди додати символ амперсанд (**&**). Командний інтерпретатор поверне номер завдання та ідентифікатор процесу:

```
$ make &  
[2] 254  
$
```

Якщо завдання не вимагає від користувача введення, воно продовжує свою роботу до повного завершення. Якщо команді потрібен введення, вона переходить в стан очікування, на екран виводиться відповідне повідомлення, яке виглядає приблизно так:

```
[1] + Suspended (tty input) programm 0
```

У даному випадку в очікуванні введення призупинилося виконання програми **programm**. Користувачеві необхідно перевести з фонового режиму в привілейований і виконати введення.

Перегляд стану завдань. За допомогою команди **jobs** користувач має можливість переглянути стан своїх завдань і отримає список всіх завдань запущених в сеансі роботи з терміналом.

```
$ jobs  
[1]  Stopped (user)          du  
[2]- Stopped (user)          du -a /home/intdbi  
[3]+ Stopped (user)          du -r /home/intdbi  
$
```

Команда **jobs** приймає два прапори. Прапор **-l** включає ідентифікатор процесу з номером завдання.

```
$ jobs -l  
[1]  1351 Stopped (user)      du  
[2]- 1381 Stopped (user)      du -a /home/intdbi  
[3]+ 1383 Stopped (user)      du -r /home/intdbi  
$
```

Прапор **-р** замінює номер завдання на ідентифікатор процесу.

```
$ jobs -p
```

```
1351
```

```
1381
```

```
1383
```

```
$
```

Ідентифікатор процесу може використовуватися при зверненні до команди **ps**.

Номер завдання дозволяє командному інтерпретатору спостерігати за процесами. Його можна розглядати як головний елемент групи процесів, оскільки для користувача завдання породжує будь-які команди або в конвеєрі, або, як підзавдань.

Переведення завдання в привілейований режим. Команда **fg** переводить завдання в привілейований режим. При наявності призупиненого завдання, його можна зробити привілейованим (перевести на передній план) за допомогою команди **fg #номер_завдання** (або **fg номер_завдання** в **bash**). Після цього завдання або виведе на екран повідомлення про те, що йому потрібно від терміналу, або буде приймати очікуваний введення. Перевівши завдання у привілейований режим, можна призупинити його виконання, натиснувши комбінацію клавіш **Ctrl-Z**, і зайнятися ним пізніше.

Будь-яке завдання зі списку, наданого командою **jobs**, доступне, якщо користувач захоче зробити його привілейованим, навіть у тому випадку, коли воно вже працює у фоновому режимі. Якщо в цьому списку наведено тільки одне завдання, то при використанні команди **fg** користувачеві не потрібно задавати його номер. Якщо номер завдання не заданий, передбачається поточне завдання.

Переведення завдання у фоновий режим. За допомогою команди **bg** можна відновити у фоновому режимі роботу призупиненого або зупиненого завдання. Для цього потрібно вказати відповідний номер завдання, після чого воно перейде у фоновий режим, і буде працювати до свого завершення, або поки йому знову не знадобиться введення з терміналу.

Команда очікування завершення процесу. Це остання суттєва команда управління завданнями. При введенні **wait** припиняється робота командного інтерпретатора до тих пір, поки не будуть завершені всі фонові завдання. Сюди входять і будь-які зупинені завдання, тому при введенні **wait** варто переконатися, що всі фонові завдання працюють. Команда **wait** може також приймати в якості параметра номер завдання. У цьому випадку командний інтерпретатор припиняється до тих пір, поки не завершиться виконання зазначеного завдання.

4.3. Графічні оболонки

4.3.1. Функції графічної оболонки

Термін “графічна оболонка” може використовуватися для чого завгодно, від простого менеджера вікон до повнофункціонального набору додатків для робочого столу типу KDE або GNOME. Строго кажучи, майже усі функції ОС, що є доступними для користувача, мають бути підтримані повнофункціональною графічною оболонкою, але найголовнішими функціями користувацького інтерфейсу є функції управління завданнями та файлами.

Перші версії Unix не мали графічного інтерфейсу. Тобто в той час існував суверий Unix з командним рядком, і основні користувачі комп'ютерів – учені та інженери – навіть не мріяли про щось, окрім "чорного" терміналу. І лише у 1984 році відбувся анонс графічної підсистеми X Window System (Project Athena), яка зародилася в надрах Массачусетського технологічного інституту і з того часу стала основною графічною системою для ОС сім'ї Unix.

Графічна оболонка Linux складається з декількох частин:

- X Window System - частина графічного інтерфейсу користувача, яка дозволяє використовувати відеоадаптер у графічному режимі. Вона може обробляти події миші та клавіатури, виводити на екран текст і графічні зображення, у тому числі й зображувати на екрані вікна.
- Менеджери вікон (Window managers) - частина графічного інтерфейсу користувача, яка дозволяє керувати розмірами та розміщенням вікон на екрані, згортасти і

розвертати вікна, а також яка відповідає за зовнішній вигляд вікон (наприклад, вигляд заголовків, рамок тощо).

- Елементи керування (toolkit, widget set) - набір стандартних елементів інтерфейсу користувача, таких як кнопки, комбіновані списки, поля для введення тексту, розділени і таке інше.

Більшість графічних оболонок Linux зазвичай містять також засоби для вирішення завдань, які найбільш часто зустрічаються, - редагування файлів, модифікація списку користувачів, перегляд каталогів і таке інше.

Відзначимо, що всі перелічені складові частини графічної оболонки не є частиною самої операційної системи. Тому, в принципі, можна вибрати будь-яку з наявних реалізацій у будь-якій з її складових частин. Проте нерідко одна або навіть декілька графічних оболонок входять до складу найбільш поширених дистрибутивів Linux і можуть бути встановлені в процесі інсталяції цієї операційної системи.

Найбільш популярні зараз дві графічні оболонки: **KDE** (K Desktop Environment) і **GNOME** (GNU Network Object Model Environment). Є й інші графічні оболонки, але вони зараз не настільки широко застосовуються, як **KDE** і **GNOME**.

4.3.2. X Window System

В UNIX-подібних операційних системах для виведення графіки майже завжди використовується X Window System.

X Window System — віконна система, що забезпечує стандартні інструменти, і протоколи для побудови графічних інтерфейсів користувача. Майже всі сучасні операційні системи підтримують X Window System, але переважно вона закріпилася в UNIX-подібних системах як стандарт «де-факто».

X Window System забезпечує базові функції графічного середовища: відображення і переміщення вікон на екрані, взаємодію з мишею і клавіатурою. X Window System не визначає деталей інтерфейсу користувача — цим займаються менеджери вікон, яких розроблено безліч. Через це зовнішній вигляд програм у середовищі X Window System може дуже різнятися; різні програми можуть використати цілком несхожі один на одного інтерфейси.

У X Window System передбачена мережева прозорість: графічні програми можуть виконуватися на іншій машині в мережі, інтерфейс при цьому буде передаватися через мережу і відображатись на локальній машині користувача. У X Window System терміни «клієнт» і «сервер» мають незвичне багатьом значення: «сервер» означає локальний дисплей користувача (дисплейний сервер), а «клієнт» — програму, яка цей дисплей використовує (вона може виконуватися на віддаленому комп'ютері).

Система X Window System була розроблена у Массачусетському технологічному інституті (MIT) в 1984 році. Нинішня (на червень 2006 р.) версія протоколу — X11 — з'явилася у вересні 1987 р. Проект X очолює фонд X.Org Foundation. Зразкова реалізація (англ. «reference implementation») системи вільно доступна на умові ліцензії MIT і подібних до неї ліцензій.

X Window System часто називають X11 чи X, неформально «ікс?».

4.3.3. Клієнт-серверна модель

X Window System використовує клієнт-серверну модель взаємодії: X-сервер обмінюється повідомленнями з різними клієнтськими програмами. Сервер приймає запити виведення графіки (вікон) і відправляє назад введення користувача (з клавіатури, миші чи сенсорного екрану). X-сервер може бути:

- системною програмою, яка контролює виведення зображення на персональному комп'ютері;
- додатком, який відображає графіку у вікно іншої дисплейної системи;
- виділеним компонентом апаратного забезпечення.

Ця клієнт-серверна термінологія — термінал користувача як «сервер» і віддалені програми як «клієнти» — найчастіше заплутує нових користувачів X, оскільки звичайно ці терміни мають зворотні значення. Але X Window System приймає точку зору програм, а не кінцевого користувача: локальний дисплей надає послуги відображення графіки програмам, і виступає у ролі серверу. Віддалені програми користуються цими послугами, а тому відіграють ролі клієнтів.

Протокол, за допомогою якого спілкуються сервер і клієнт, є прозорим для мережі: клієнт і сервер можуть знаходитися як на одній і тій же машині, так і на різних. Зокрема, вони можуть працювати на різних архітектурах під керівництвом різних операційних систем — результат буде одинаковим. Клієнт і сервер можуть навіть безпечно взаємодіяти через Інтернет за допомогою тунелювання з'єднання крізь зашифрований мережевий сесії.

Щоб запустити віддалену клієнтську програму, що виводить графіку на локальний X-сервер, користувач зазвичай відкриває емулятор терміналу та підключається до віддаленої машини за допомогою telnet або SSH. Потім він віddaє команду, яка вказує дисплей, на який слід виводити графіку (наприклад, `export DISPLAY=[ім'я_комп'ютера_користувача]:0` при використанні `bash`). Нарешті, користувач запускає клієнтську програму. Вона підключається до локального X-серверу та буде відображати графіку на локальний екран, і приймати введення від локальних пристрій введення. Інший варіант — використовувати невелику допоміжну програму, яка підключається до віддаленої машини та запускає на ній потрібну клієнтську програму.

4.3.4. Інтерфейси користувача

X Window System навмисно не визначає, як повинен виглядати інтерфейс користувача програми - кнопки, меню, заголовки вікон і т. д. Ці питання вирішуються на рівні віконних менеджерів, інструментаріїв елементів інтерфейсу, середовищ робочого столу і на рівні окремих додатків. З цієї причини візуальне подання X-інтерфейсів зазнало величезних змін з плином часу.

Віконний менеджер керує розміщенням і зовнішнім виглядом вікон додатків. Він може створювати інтерфейс, подібний Microsoft Windows або Macintosh (наприклад, так працюють менеджери вікон Kwin в KDE та Metacity в GNOME), або зовсім інший стиль (наприклад, в фреймових віконних менеджерах, таких, як Ion). Віконний менеджер може бути простим і мінімалістичним (як twm — базовий віконний менеджер, що постачається з X), а може пропонувати функціональність, близьку до повноцінного робочого середовища (наприклад, Enlightenment).

Багато користувачів використовують X разом з повним середовищем робочого столу, яка включає в себе віконний менеджер, різні програми та єдиний стиль інтерфейсу. Найбільш популярні середовища робочого столу — GNOME та KDE. У стандарті Single UNIX Specification вказане середовище CDE. Проект freedesktop.org намагається забезпечити взаємодію між різними середовищами, а також компоненти, необхідні для конкурентоспроможного робочого столу на основі X.

Сьогодні рівень продуктивності графічних комп'ютерних систем забезпечується шляхом використання найбільш передових графічних функцій. Виробники апаратного забезпечення, як правило, реалізують ці можливості у пропрієтарних драйверах, причому ці драйвери, зазвичай, пишуться в першу чергу для систем Microsoft Windows (як для самих розповсюджених на ринку). Драйвери багатьох старих графічних плат піддалися зворотній розробці в рамках проектів XFree86 та X.Org. Проте деякі виробники розглядають свої розробки в області високопродуктивного відео як комерційну таємницю, або ж як патентовані винаходи, які не хочуть розкривати.

Багато нинішніх реалізацій X керують відеоапаратурою напряму. Нестійкий X-сервер може зробити дисплей непридатним до використання навіть у тих випадках, коли сама операційна система продовжує нормально функціонувати. При цьому може знадобитися перевантаження всієї системи. Технологія Direct Rendering Infrastructure (DRI) покликана усунути цю проблему.

Протокол X не надає ніяких засобів для роботи зі звуком. Підтримка звукової апаратури та відтворення звуків покладається на операційну систему. Оскільки користувачам все частіше необхідний звук, ця ситуація призвела до появи різних несумісних один з одним звукових підсистем. У минулому багато програмісти ігнорували мережеві проблеми, і просто використовували локальні звукові API операційної системи. Перше покоління клієнт-серверних звукових систем включає в себе rplay та Network Audio System. Більш сучасні системи — PulseAudio, esound в GNOME та aRts в KDE. Також розпочато розробку нової системи — Media Application Server.

Незалежність від апаратури та відділення клієнтів від серверів впливає на продуктивність системи. Мережева прозорість X вимагає, щоб клієнти і сервер працювали окремо один від одного. У минулому це істотно знижувало продуктивність

окремо системи — порівняно з Microsoft Windows та Mac OS, де віконна підсистема вбудована глибоко в саму операційну систему. Для нормальної роботи X Window System рекомендувалося від 4 до 8 Мб оперативної пам'яті — значно більше (на ті часи), ніж для Windows або Mac OS.

Поточні версії Windows і Mac OS X мають внутрішній поділ графічної підсистеми, схоже на клієнт-серверне поділ в X, і мають приблизно ті ж вимоги до ресурсів, що X з KDE або GNOME. Велика частина накладних витрат у X тепер припадає на затримку при передачі даних по мережі між клієнтом і сервером. Існує поширений міф, згідно з яким при локальному використанні X Window System її мережеві можливості (непотрібні в даному випадку) негативно позначаються на продуктивності. Насправді сучасні реалізації X використовують в такому випадку локальні сокети та загальну пам'ять (напр. MIT-SHM), вимагаючи дуже незначних накладних витрат.

4.3.6. Деякі інші графічні оболонки

GNOME є «дружньою» до користувача графічною оболонкою, яка дозволяє легко використовувати і настроювати свої комп'ютери. У GNOME є панель (для запуску додатків та відображення їх стану), робочий стіл (де можуть бути розміщені дані й додатки), набір стандартних інструментів і додатків для робочого столу, а також набір угод, які полегшують спільну роботу й узгодженість додатків. Користувачі різних операційних систем або оболонок при використанні такої потужної графічної оболонки, яку забезпечує GNOME, повинні відчувати себе зручно й комфортно.

KDE є простою у використанні сучасною графічною оболонкою. Ось лише де-що з того, що дає користувачеві KDE:

- Зручний сучасний робочий стіл.
- Робочий стіл, повністю прозорий для роботи в мережі.
- Інтегрована система допомоги, яка забезпечує зручний і погоджений доступ до системи допомоги з використання робочого столу KDE і його застосувань.
- Одноманітний зовнішній вигляд і керування у всіх додатках KDE.
- Стандартизовані меню й панелі інструментів, комбінації клавіш, колірні схеми тощо.
- Інтернаціоналізація: у KDE підтримується більше 40 мов.

- Централізована одноманітна конфігурація робочого столу в діалоговому режимі.
- Велика кількість корисних застосувань для KDE.

Для KDE існує пакет офісних застосувань, який виконаний за технологією «kparts» з KDE, що складається з програми для роботи з електронними таблицями, презентаційної програми, органайзера, клієнта для читання телеконференцій та інших програм.

Кращим довідником з KDE є онлайнова документація. KDE поставляється з власним веб-браузером, який називається Konqueror, десятками корисних застосувань і детальною документацією.

Адміністратору розрахованої на багато користувачів системи, може бути потрібно мати графічний екран входу в систему для запрошення користувачів. У KDE є альтернативний менеджер kdm, який був розроблений для того, щоб мати привабливий вигляд і більшу кількість опцій, які настроюють вхід у систему. Зокрема, користувачі можуть легко вибирати (за допомогою меню), яку оболонку (KDE, GNOME або щось інше) запускати після входу в систему.

Xfce є графічною оболонкою, побудованою на основі інструментального пакета GTK, використовуваного в GNOME, але набагато легше, і призначений для тих, кому потрібний простий, ефективно працюючий робочий стіл, який легко використовувати і настроювати. Візуально він дуже схожий на CDE, який є в комерційних unix-системах. Ось деякі з переваг Xfce:

- Простий, легкий в обігу робочий стіл, який повністю настроюється за допомогою миші, з інтерфейсом drag and drop тощо.
- Головна панель з меню, аплетами і можливостями зі швидкого запуску додатків.
- Інтегрований віконний менеджер, менеджер файлів, керування звуком, модуль сумісності з **GNOME** та інше.
- Можливість використання тем (оскільки використовується **GTK**).
- Швидкий, легкий та ефективний: ідеальний для застарілих/слабких машин або для машин з обмеженою пам'яттю.

Існують й інші види графічних оболонок. Наприклад, оболонка **FVWM** чимось нагадує оболонку **Explorer** з Windows 95. А оболонка **Blackbox** — просто зразок мі-

німалізму в дизайні віконного менеджера. Втім, вибір тієї або іншої графічної оболонки — справа індивідуальна.

5. ВІДКРИТИ СИСТЕМНІ УТИЛІТИ

Крім розглянутих вище функцій, ОС UNIX пропонує десятки корисних утиліт і засобів для різних спеціальних потреб. Деякі з цих засобів є, де facto, стандартними складовими будь яких версій ОС, інші можуть включатися у склад ОС за бажанням користувача. Розглянемо декілька стандартних засобів, таких як утиліти роботи з текстами, підтримки мережевої роботи, резервного копіювання, а також такі важливі й популярні додаткові програми як файлові менеджери та архіватори.

5.1. Засоби обробки тексту

5.1.1. Основні утиліти для обробки текстів

Обробка текстів – одна з найпоширеніших сфер застосування комп’ютерів. Відповідно, з перших версій ОС UNIX цьому виду сервісу приділялася значна увага. Основні утиліти для обробки текстів представлені в табл. 28. Таблиця у краткій формі представляє загальні відомості про відповідні програми. Деякі з них (а також утиліти `grep` та `vi`) будуть розглянуті більш докладно у наступних параграфах.

Таблиця 28. Основні утиліти обробки текстів

Утиліта	Призначення
awk	Мова обробки шаблонів. Дозволяє виконувати довільну програму при виявленні в тексті певного рядка, відповідного шаблону. По синтаксису аналогічний С. Містить безліч вбудованих функцій. Використовується для обробки і перетворення текстових даних, що складаються із стовпців і рядків, а також для побудови звітів та аналізу журналів.
diff	Команда, що порівнює два файли і видає знайдені розбіжності в різних форматах.
ed	Стандартний рядковий текстовий редактор. Сприймає команди зі стандартного входного потоку, змінює файли і часто використовується в сценаріях.
ex	Розширена версія редактора ed . Підтримує безліч установок, які можна запам'ятовувати для кожного користувача у файлі .exrc в його початковому каталозі.

Таблиця 28. Основні утиліти обробки текстів

Утиліта	Призначення
head	Видає вказану кількість початкових рядків з файлу.
more	Дозволяє переглядати файл посторінково в обох напрямках, шукати в ньому за шаблоном.
pr	Форматує файл або вхідний потік для друку, розбиваючи його на сторінки і забезпечуючи, при необхідності, заголовками.
Sed	Потокове версія редактора ed . Дозволяє ефективно виконувати пошук і заміну в стандартному вхідному потоці або зазначених файлах.
tail	Видає вказану кількість кінцевих рядків з файлу.
Tr	Перетворює символи у вхідному потоці, замінюючи одні ланцюжка на інші. Підтримує весь набір символів.

5.1.2. Редактор **ed**. Регулярні вирази та зіставлення зі зразком.

Стандартні засоби редагування в ОС UNIX з'явилися давно і орієнтовані на найпростіші текстові термінали. До таких засобів можна віднести рядковий редактор **ed** та екранний редактор **vi**. При першому знайомстві вони можуть здатися складними і явно застарілими з точки зору "дружності", проте в світі UNIX добре не те, що є самим новим і "гарним", а, швидше, те, що використовується давно, багатьма людьми і є в будь-який версії. Це повною мірою відноситься до стандартних засобів обробки текстів.

Ефективність обробки тексту визначається ефективністю пошуку необхідних фрагментів. Для завдання зразків пошуку в ОС UNIX використовується ряд *метасимволів регулярних виразів*, що вперше з'явилися в редакторі **ed** та представлених у табл. 19. Прості приклади регулярних виразів і пов'язаних з ними шаблонів пошуку представлені в табл. 20.

Таблиця 19. Метасимволи регулярних виразів

Метасимвол	Опис
c	Будь-який конкретний символ задає збіг з таким же символом
\C	Скасовує спеціальний зміст символу c
^	Відповідає початку рядка, коли ^ починає зразок
\$	Відповідає кінцю рядка, коли \$ закінчує зразок
.	Співпадає з будь-яким символом
[...]	Відповідає одному будь-якому символі в ... ; припустимі діапазони типу az
[^...]	Відповідає будь-якому символу, що не входить до ... ; припустимі діапазони
r*	Відповідає нульовому або більше числа входжень r , де r - символ або [...]
&	Використовується тільки в правій частині команд заміни (s); вставляє фрагмент, який співпав із зразком
\(..\)	Позначає регулярний вираз; знайдені рядки доступні як \1 , \2 і т. д. до \9 в лівій і правій частинах відповідної команди заміни s , а також у шаблонах пошуку одразу після закриття відповідної круглої дужки.

Таблиця 20. Приклади використання регулярних виразів

Зразок	Відповідність
/^\$/	порожній рядок, тобто тільки кінець рядка
/./	непорожня рядок, принаймні один символ
/^/	всі рядки
/thing/	thing де-небудь у рядку
/^thing /	thing на початку рядка
/thing\$ /	thing в кінці рядка

Таблиця 20. Приклади використання регулярних виразів

Зразок	Відповідність
/^thing\$/	рядок, що складається лише з thing
/thing.\$ /	thing плюс будь-який символ в кінці рядка
/\thing\//	/thing/ де-небудь у рядку
/[Tt]hing/	thing або Thing де-небудь у рядку
/thing[0-9]/	thing , за якою йде одна цифра
/thing[^0-9]/	thing , за якою йде не цифра
/thing1.*thing2/	thing1 , потім будь-який рядок, потім thing2
/^thing1.*thing2\$/	thing1 на початку і в кінці thing2

Помічені регулярні вирази. Щоб маніпулювати не тільки цілими фрагментами, вибіраними регулярними виразами, але і їх частинами, використовуються *помічені регулярні вирази*: якщо конструкція \(..\) з'являється в регулярному виразі, то частина відповідного їй фрагмента доступна як \1. Допускається використання до дев'яти помічених виразів, на які посилаються \1, \2 і т. д.

Ось низка прикладів використання помічених регулярних виразів:

s /\(..\)\(..*\) / \2\1 /	Помістити 3 перших символу в кінець рядка
/\(..*\)\1 /	Знайти рядки, які містять повторювані суміжні ланцюжка символів
s ^\(..*\)\.\(..*\) / \1.\2 /	Перенести залишок рядка після першої точки на наступний рядок

5.1.3. Утиліта **grep**. Пошук в тексті за зразком

Програми сімейства **grep** здійснюють пошук шаблону (задається мовою регулярних виразів) у зазначених файлах або у вхідному потоці і (звичайно) видають відповідні шаблоном рядки у вихідний потік. У це сімейство входять три програми, **grep**, **egrep** та **fgrep**, що відрізняються алгоритмами (а, значить, швидкістю ро-

боти і використовуваними ресурсами системи) і, частково, можливостями при за-
вданні шаблонів.

Виклик команди **grep** має такий синтаксис:

grep [опції] [регулярний_вираз] [файл ...]

Команда шукає рядки, що задаються шаблоном у вигляді *обмеженого регулярного виразу* (використовують підмножина допустимих алфавітно-цифрових і спеціальних символів), аналогічного використовуваним в **ed**, у вказаних файлах або у вхідному потоці. Можливі опції наведено в табл. 21.

Таблиця 21. Опції командного рядка grep

Опція	Призначення
-B	Перед кожним рядком видавати номер блоку, в якому вона знайдена. Це може стати в нагоді при визначенні номера блоку на контекст
-C	Видає тільки кількість рядків, які відповідають шаблону
-I	Ігнорувати різницю між великими та малими буквами
-H	Запобігає видачу імені файлу перед співпадаючим рядком. Використовується при многофайловом пошуку.
-L	Видає імена файлів, що містять збіглися рядки, один раз, розділяючи їх переведенням рядка. Не повторює імена файлів, якщо шаблон знайдено більше одного разу.
-N	Передує кожен рядок її порядковим номером (перший рядок має номер 1)
-S	Пригнічує видачу повідомлень про помилки, пов'язаних з не існуванням файлів або недоступністю для читання
-V	Видає всі рядки, крім тих, що містять шаблон
-E te	Шукає спеціальне вираз se (повне регулярний вираз, що починається з -)
-F файл	Бере список повних регулярних виразів з файла

Будьте уважні при використанні символів \$, *, [, ^, |, (,) та \ в шаблоні, оскільки вони також мають значення для командного інтерпретатора. Краще укладати шуканий шаблон в апострофи: '... '.

Статус виходу дорівнює 0, якщо знайдені збігаються рядки, 1 - якщо рядки не знайдені і 2 якщо є синтаксична помилка або недоступні файли (навіть якщо збіги знайдені).

Розглянемо прості приклади:

```
*\ ) \1'  
abc abc  
$ echo abc abc | grep 'c a'  
abc abc  
$ echo abc abc | grep '^c a'  
$ cd $INFORMIXDIR/etc  
$ grep -n $INFORMIXDIR  
^C  
$ grep -n tmp *. sh  
beta_evidence. sh:306:      DUMPDIR=/tmp  
bldutil. sh:40:# remove tmp salvage_file  
bldutil. sh:55:      RESFILE=/tmp/bldutil. $$  
evidence. sh:302:      DUMPDIR=/tmp  
logevent. sh:46:TMPFILE=${TMPDIR:-/tmp}/$PROG. `date +%y-%m-%d-%H%M-%S`
```

5.1.4. Редактор **vi**

vi (redit) - консольно-орієнтований текстовий редактор. Він дозволяє бачити одночасно цілу сторінку тексту, переміщатися по ньому курсором і безпосередньо бачити зміни, що вносяться.

Редактор **vi** є спадкоємцем строкового редактора **ex**, який, у свою чергу, є розширенням базового текстового редактора **ed**. Тим самим, забезпечується спадкоємність засобів редагування і використання ефективного механізму пошуку і заміни на базі регулярних виразів.

Для виклику редактора **vi** у найпростішому випадку використовується такий синтаксис:

```
vi [+ рядок] [файл]
```

У результаті, вказаний файл відкривається у вікні редактора. Якщо файл не вказано, редагується спочатку порожній буфер (тобто новий файл, ім'я якого спочатку не задане).

Рядок, на якому відкривається файл, можна задавати таким чином:

+ **Номер** Рядок із зазначенім **номером**

+ Останній рядок файлу

+ / **Re** Рядок, відповідна вказаною регулярним виразом **re**

Зазначена рядок буде знаходитися в центрі екрану (якщо тільки файл не менше, ніж розмір екрану) і на її початку буде встановлено курсор.

Редактор **vi** підтримує декілька **режимів роботи**:

Командний режим Нормальний і початковий режим. По завершенні інших режимів відбувається повернення в командний режим. Для форсованого переходу в цей режим використовується клавіша **Esc**

Режим вводу У режим введення входять при завданні однієї з таких команд: **a A i I o O c C s SR**. При цьому може набиратися довільний текст. З цього режиму виходять або по **Esc**, або він автоматично переривається редактором. При цьому зазвичай подається звуковий сигнал.

Режим останнього рядка Читання для команди: **/?** або **!;** Припиняється натисканням клавіші **Enter**.

Основні команди редактора **vi** представлені в табл. 22.

Таблиця 22. Зведення основних команд редактора vi

Рух курсору	
H (Ctrl-h)	курсор вліво
J (Ctrl-n)	курсор вниз

K (Ctrl-p)	курсор вгору
L (Space)	курсор право
Ctrl-u	Перехід вгору на половину екрану
Ctrl-d	Перехід вниз на половину екрану
Ctrl-f	На сторінку вперед (PageDn)
Ctrl-b	На сторінку назад (PageUp)
0	Перехід на початок поточного рядка
\$	Перехід в кінець поточного рядка
nG	Перехід на рядок з номером n

Додавання тексту

a	Додати текст після курсору
A	Додати текст наприкінці поточного рядка
i	Вставити текст перед курсором
I	Вставити текст на початку поточного рядка
o	Утворити новий рядок під поточною
O	Утворити новий рядок над поточною

Зміна тексту

~	Змінити регістр символу над курсором
r	Заміна одного символу
R	Заміна символів

Видалення тексту

x	Видалення символу
dd	Видалення рядка
Ndd	Видалення N рядків

Пошук і заміна

/ Str	Пошук рядка str вперед. str може бути регулярним виразом
? / Str	Пошук рядка str тому

n	Повторити пошук в тому ж напрямку
N	Повторити пошук у зворотному напрямку
: [Range] s / old / new / [g]	Замінити old на new у вказаному діапазоні рядків range . new і old можуть бути регулярними виразами, а range задається аналогічно діапазону рядків у редакторі ed . Суфікс g означає замінити у всьому файлі.

Копіювання тексту

yy	Копіювання рядка в цілому
Nyy	Копіювання N рядків
p	Вставити з буфера після (курсору, поточного рядка)
P	Вставити з буфера перед (курзором, поточним рядком)

Вихід з редактора

: Wq ENTER	Запис і вихід. Записати текст з буфера у файл і вийти.
: X ENTER	Умовна запис і вихід. Записати текст з буфера тільки при наявності змін і вийти з редактора.
: Q!ENTER	Закінчити редагування без запису змін.

Інші команди

!	Виконати одну команду інтерпретатора
.	Повторити останню команду
u	Скасувати дію останньої команди
J	З'єднати рядки
Ctrl-G	Показати номер поточного рядка

Курсор можна переміщати і клавішами переміщення курсору або клавішами **PageUp**, **PageDn**, але ці можливості, на відміну від описаних у таблиці, підтримуються не на всіх терміналах.

5.1.5. Робота в мережі

Основні команди мережевих служб наведені в табл. 29. Ми не будемо зосереджуватись на поясненні кожної з команд: з одного боку, вони зрозумілі без пояснень

людям, обізнаним в Інтернеті, оскільки з UNIX поширилися на всю глобальну мережу; з іншого боку, їх пояснення викликало б необхідність розгляду принципів побудови комп’ютерних мереж, що виходить за межі цього посібника.

Таблиця 29. Основні мережеві команди

Утиліта	Призначення
ftp	Клієнт для обміну файлами з віддаленою машиною по мережі. Дозволяє переглядати каталоги, створювати каталоги на віддаленій машині, завантажувати і вивантажувати файли за допомогою стандартного набору команд. Допускає автоматизацію операцій з обміну файлами.
netstat	Видає різноманітну статистику про роботу мережі, вміст таблиць маршрутизації і т. д.
ping	Посилає спеціальний пакет ICMP , яке потребує відповіді від віддаленого сервера. Дозволяє перевірити доступність віддаленого хоста і швидкість роботи мережі. Зазвичай доступна тільки користувачеві root .
rlogin	Програма віддаленої реєстрації. Дозволяє працювати з віддаленою машиною так само, як з локальною. Підтримує довірчі відносини.
rsh	Віддалений командний інтерпретатор. Дозволяє виконати будь-яку команду інтерпретатора на віддаленій машині і отримати її вихідний потік на локальній. Підтримує довірчі відносини.
ssh	Захищений командний інтерпретатор, функціонально аналогічний програмам telnet , rsh і rlogin , який передає паролі і дані в зашифрованому вигляді.
telnet	Програма віддаленого підключення до зазначеної мережової служби. Зазвичай використовується для віддаленої реєстрації. Не підтримує довірчі відносини.
traceroute	Програма трасування пакетів. Показує маршрут, по якому будуть направлятися пакети на вказаний хост і швидкість передачі на кожному з переходів. Доступна тільки користувачеві root .

5.1.6. Резервне копіювання і відновлення

Закінчимо огляд стандартних засобів UNIX утилітами резервного копіювання. Корисні команди для створення резервних копій та відновлення ОС представлені в табл. 30, яка містить досить повну інформацію для користувача.

Таблиця 30. Основні засоби резервного копіювання і відновлення

Утиліта	Призначення
compress	Стискає (упаковує) вказаній файл, звичайно видаляючи вихідний варіант. Стиснуті файли звичайно мають суфікс .Z і відновлюються командою uncompress .
cpio	Команда створення архівів. Поміщає всі вказані файли, включаючи вміст підкаталогів, в архів, що видається в стандартний вихідний потік. Отримує архів з вхідного потоку і розкриває в поточному каталозі. Підтримує різні платформи і формати, в тому числі, формат архівів tar .
dr	Команда копіювання блоків даних з одного файлу або пристрою в інший. Дозволяє виконувати перетворення при копіюванні.
Gzip	Утиліта GNU для стиснення зазначеного файлу. Упаковує файли краще, ніж compress . Працює на всіх plataформах. Стиснуті файли звичайно мають суфікс .gz і відновлюються командою ungzip (gzip -d).
Tar	Утиліта архівування. Поміщає всі вказані файли, включаючи вміст підкаталогів, в архівний файл. Створює необхідні каталоги і файли при розархівуванні.

5.2. Файлові менеджери

5.2.1. BeeSoft Commander

BeeSoft Commander – молодий проект, нехитрий файловий менеджер, написаний в дусі UNIX-way. Згідно з неписаним правилом, програма повинна виконувати лише свої прямі обов'язки і нічого зайвого.

Файловий менеджер використовує бібліотеку QT і дуже швидко запускається з середовища KDE. Панель інструментів не налаштовується, проте на ній зібрані фактично всі основні функції програми. Ви можете працювати з декількома вкладками. Права кнопка миші відповідає за виділення файлів - данна старої традиції Norton Commander. Багато операцій над файлами виконуються за допомогою функціональних клавіш, згідно зі старим стандартом, закладеному також Пітером Нортоном.

BeeSoft Commander містить непоганий FTP клієнт, що дозволяє підключатися до декількох серверів, зберігати налаштування з'єднань. Розробники не стали обтяживати себе реалізацією перегляду і редактування різних документів. За допомогою вбудованого редактора ви можете змінювати лише текстові документи. У разі виявлення інших типів файлів, програма або перемикається в HEX-режим, або відображається хаотичні набір символів. Редагування при цьому недоступне.

BeeSoft Commander не потребує складному конфігуруванні. Ви можете вказати шрифти та кольори панелей, налаштувати ширину колонок. І, фактично, на цьому все задоволення від подорожі з налагодження закінчується.

5.2.2. emelFM2

emelFM2 - простий файловий менеджер, який використовує бібліотеку GTK2. Існують збірки для GTK1 під назвою emelFM1.

Робоче вікно файлового менеджера складається з двох панелей і вікна відображення статусу виконуваних команд. Ви можете міняти розкладку панелей, змінюючи положення роздільника з вертикального (за умовчанням) на горизонтальний. Всі панелі інструментів можна переміщати всередині робочого вікна, розташовуючи їх не тільки у верхній частині, але і з боків, знизу і навіть в області роздільника.

Ви можете додавати власні інструменти на панелі. Треба зауважити, що ця функція реалізована не дуже зручно через відсутність заздалегідь визначених кнопок. Кожен новий елемент необхідно створювати фактично з нуля. При цьому ви не можете призначити на кнопку, наприклад, виклик зовнішнього застосування або переход в довільний каталог.

Для каталогів, посилань, пристройів і файлів ви можете призначати довільні кольору, проте додавання власних типів відсутня.

Робота з архівами, перейменування, сортування і декілька інших функцій реалізуються за допомогою плагінів. Вбудований текстовий редактор дуже примітивний, однак ви можете підключити зовнішні програми на перегляд і редагування документів.

У налаштуваннях файлового менеджера ви можете вказати каталоги, до яких найчастіше звертаєтесь. Історія каталогів викликається за допомогою стрілки, розташованої в правій частині введення локальної адреси. Натискаючи на неї, ви відкриваєте спливаючий список раніше відвіданих каталогів.

Деякі властивості файлового менеджера можуть завести в глухий кут. Ви копіюєте великий файл. При цьому на екрані не відображається ніякої інформації, про копіювання говорить лише різке уповільнення роботи програми та палаючий індикатор жорсткого диска. Перервати процедуру копіювання або зробити паузу, зрозуміло, при цьому не можна. Виконання багатьох операцій або зовсім ніяк не відображається, або це відбувається на примітивному рівні.

5.2.3. *Gentoo*

Коли вимовляється слово «*Gentoo*», то, в першу чергу, згадується популярний дистрибутив Linux, основна особливість якого – збірка всіх пакетів з вихідних текстів під час установки. Одноіменний файловий менеджер не має прямого відношення до згаданого дистрибутиву. Файловий менеджер *Gentoo* – повністю незалежний проект.

Gentoo використовує бібліотеку GTK1, що дозволяє програмі працювати на високій швидкості (написаний на звичайному C, не C++). Користувачам Windows, звичним до управління файлами за допомогою Провідника або Total Commander буде дуже незвично і незручно перший час, так як *Gentoo* сильно відрізняється від цих

продуктів. Втім, за позірним великою кількістю одноманітних кнопок ховається непогано продумана структура управління.

Перше, про що варто відразу забути - це стандартні функціональні кнопки управління основними діями з файлами (Fx). За умовчанням, в Gentoo задіяні лише клавіші F1, F5 і F8, які виконують незвичні дії. Наприклад, «F8» - це запуск програми.

Виділення файлів здійснюється за допомогою лівої кнопки миші. Ви просто клацаете по файлу - він виділяється. Не відпускате кнопку миші, проведіть їй за списком файлів - буде зроблено автоматичне виділення.

Всі основні операції над файлами здійснюються за допомогою кнопок, розташованих в нижній частині робочого вікна. Gentoo володіє багатозадачністю, і ви можете, наприклад, запустити копіювання, а в цей час виконувати в програмі будь-які інші операції. Якщо в Windows практично всі файлові менеджери мають даною можливістю, то в Linux ситуація дещо складніша.

Кнопка **Configure** викликає вікно налаштувань Gentoo. Файловий менеджер має величезний потенціал, але багато що псує неприємна для багатьох *NIX-програм риса – велика складність процесу встановлення. Наприклад, для того, щоб призначити клавіші, необхідно вручну (!) ввести назву всіх елементів, що беруть участь в комбінації і, знову ж таки вручну (!!), набрати ім'я виконуваної команди. Навіщо простому користувачеві знати імена системних команд файлового менеджера!? Щоб змінити зовнішній вигляд панелей, необхідно вручну (зловісний союзник налаштування багатьох програм *NIX) вводити координати початкової позиції елементів, розміри елементів.

Заголовки вікон не підтримують кирилицю. Це відома проблема додатків GTK1 в дистрибутивах, що використовують локаль UTF8.

Gentoo - швидкий і потужний файловий менеджер, настроювання якого, однак, вимагає певних знань і хоча б трохи терпіння.

5.2.4. *GNOME Commander*

GNOME Commander – класичний, двохпанельний файловий менеджер для середовища GNOME. Програма вміє обробляти MIME-типи робочого середовища, що дозволяє, запускаючи документи на виконання, відкривати відповідні інструменти

обробки. Наприклад, якщо ви запустите файл з типом ODT (Open Document Text), то автоматично відкривається OpenOffice.org з поточним документом в робочому вікні. Даний принцип є стандартом де-факто в Windows, однак у Linux не всі програми дозволяють використовувати змінні оточення робочого середовища. Підсвічування типів файлів також береться з системних значень. За неї відповідає мінліва LS_COLORS. Значки документів, згідно з типами, беруться з оточення GNOME.

Якщо в системі встановлений пакет SAMBA, що дозволяє здійснювати навігацію по локальній мережі, то цей процес можна здійснювати прямо з GNOME Commander. Основні операції з файлами можна виконувати як з допомогою традиційних функціональних клавіш, так і викликом контекстного меню. В правому куті заголовка панелей знаходяться кнопки виклику історії перегляду каталогів, а також закладки. Настройка закладок здійснюється із головного меню файлового менеджера.

Панель інструментів не має засобів для налаштування, це можна зробити спеціальним набором кнопок. Программа має досить обмежені можливості для перегляду документів, що обмежуються обробкою простого текста і графіки. Але можна підключити два зовнішніх модуля, що відповідають за переглід і редактування.

Середовище GNOME развивається в сторону спрощення, покращення єргономіки. GNOME Commander — це на диво простий, лаконічний продукт, який не вимагає багато часу на вивчення.

5.2.5. Konqueror та Krusader

Найбільш популярним робочим середовищем в UNIX є KDE, обидва менеджери спираються саме на нього.

Якщо провести паралель між програмним забезпеченням Windows і Linux, то можна сказати, що Konqueror — це аналог Explorer, а Krusader - це щось середнє між Frigate і Total Commander.

Konqueror — це і броузер і файловий менеджер одночасно. Він може працювати як в режимі «проводника» (не зовсім коректне порівняння, яке застосовується до UNIX), так і з використанням двох стандартних панелей. За вибір режиму відповідають профілі. За замовчуванням Konqueror має шість профілів, два з яких відно-

сяться до управління файлами - Midnight Commander і «оглядач з вкладками». Використовуючи профіль оглядача, ви маєте бічну панель режимів, дерево каталогів і файлову панель. Бічна панель містить в собі виклик аудіо-плеєра Amarok, відображення закладок, перехід в домашню папку, роботи з мережевими ресурсами з використанням протоколів FTP, HTTP, SLP і SMB, перехід в кореневий каталог, а також виклик численних сервісів. Наприклад, якщо ви підключаете до ПК iPod, то можете відразу звертатися до його ресурсів, використовуючи префікс «ipod: /». Тут же можна виявити оглядач аудіо-дисків, bluetooth пристройів, принтерів, пристройів зберігання даних та іншого обладнання. Ви можете також переглядати і запускати будь-яке програмне забезпечення встановлене в системі (не тільки KDE), використовуючи префікс «applications: /». Фактично, Konqueror містить копію меню стартовою панелі.

Всі документи усередині файлової панелі мають значки, згідно типам, зазначеним у налаштуваннях KDE. Всі зображення кешуються, для них створюються мініатюри. Ви можете переглядати фотографії в окремій вкладці файлового менеджера. При цьому робоче вікно Konqueror складається з панелі, що містить список всіх зображень у цій папці, а також з вікна перегляду поточного документа.

Для того щоб потрапити на FTP-ресурс навіть не потрібно створювати нове з'єднання. Просто напишіть в адресному рядку «FTP: / / ». Після цього достатньо зберегти відвіданий адреса в закладках, щоб в майбутньому здійснювати відвідування одним клацанням миші. Для навігації по локальній мережі, наберіть в адресному рядку «SMB: /». Файловий менеджер відобразить список всіх доступних комп'ютерів у вашій локальній мережі. Ви можете додавати адресу будь-якого комп'ютера в якості закладки.

Перегляд величезної кількості типів документів може здійснюється в окремих вкладках файлового менеджера. Для перегляду відео запускається ядро Kaffeine, для прослуховування музики використовується Amarok. Завдяки тісній інтеграції програм робочого середовища KDE, вам не потрібно запускати окремі програми, всі операції можуть здійснюється всередині файлового менеджера.

Налаштування панелей Konqueror здійснюється штатними засобами робочого середовища, яка надає потужні та гнучкі інструменти управління. Ви можете міняти

розмір значків їх розташування. KDE підтримує зміну тим оформлення. Будь-яка операція може бути прив'язана до довільного поєднанню клавіш.

Файловий менеджер містить не тільки історію відвідувань, але і список найпопулярніших папок. Ви можете записувати оптичні диски з Konqueror, шифрувати файли, використовувати групове перейменування за допомогою утиліти KRename. Список можливостей Konqueror можна продовжувати дуже довго, досить лише згадати всю міць KDE.

Незважаючи на величезну кількість функцій, реалізованих у файловому менеджері, не можна сказати, що це складний продукт. Інтерфейс грамотно продуманий, його концепція вироблялася роками. Більшість властивостей файлового менеджера застосовано до всієї робочої середовищі. Якщо ви освоїлися в KDE, то його файловий менеджер здається лише невеликим доповненням.

Krusader також відчуває себе в середовищі KDE, як риба у воді. Файловий менеджер використовує безліч переваг робочого середовища, тісно інтегруючись з деякими її компонентами.

Робоче вікно складається з традиційних двох панелей, внизу яких розташовуються вкладки. Уподобання та історія відвідувань викликаються з кнопок, розташованих у правій частині заголовків панелей.

Відвідування FTP-серверів і навігацію по ресурсах локальної мережі можна здійснювати шляхом введення адреси з префіксами FTP і SMB. Ви можете також додавати часто відвідувані ресурси в закладки. Krusader володіє власним інструментарієм збору та управління вашими улюбленими ресурсами, не пересічним зі стандартними закладками KDE (Konqueror).

Втім, зв'язок між Krusader і Konqueror простежується багато в чому. Наприклад, контекстне меню Krusader має окремий пункт «Меню Konqueror», звідки можна викликати всі аналогічні команди штатного файлового менеджера KDE. Налаштування панелей інструментів виконана також з використанням стандартних функцій KDE.

На відміну від Konqueror, Krusader не обтяженій функціями браузера. Інтерфейс Krusader менш перевантажений, а строгое проходження стандарту управління Norton Commander допомагає застосувати старі навички у роботі з новим продуктом. Вдала знахідка розробників - меню користувача. Ви можете додавати команди

файлового менеджера, системні команди, а також призначати виклик зовнішніх додатків на гарячі клавіші. При роботі з мишею достатньо лише звернутися до головного меню. Кожен елемент може містити власний значок, що підвищує наочність і зручність роботи.

Два потужних файлових менеджера середовища KDE заслужено носять гучні назви - «завойовник» і «хрестоносець». За довгі роки існування, ці продукти встигли завоювати лояльність великого числа користувачів вільної ОС.

5.2.6. Midnight Commander ma Worker

Midnight Commander - один з найстаріших файлових менеджерів у світі UNIX-систем. Незважаючи на те, що технології давно пішли вперед, і життя без графічного інтерфейсу існує лише в жарких комірчинах системних адміністраторів, програма продовжує розвиватися і як і раніше залишається затребуваною серед користувачів Linux.

Для роботи Midnight Commander не потрібно ніяких графічних бібліотек, головне, щоб завжди під рукою була консоль / термінал. Ранні версії файлового менеджера мали проблеми з російськими кодуваннями, але сьогодні всі помилки виправлені, і ви можете переглядати та редагувати файли, використовуючи кодові сторінки, що не збігаються з локаллю Linux. Текстовий редактор Midnight Commander має підсвічування синтаксису.

За часів Norton Commander, створення великого меню команд користувача було дуже популярним явищем. Midnight Commander має аналогічною функцією, що викликається все тієї ж заповітної клавішею F2. Якщо ви натиснете в меню F4, то зможете відредактувати текстовий файл його настройок.

Засоби перегляду документів Midnight Commander мають велике обмеження - відсутність графічного інтерфейсу. Формально, файловий менеджер може переглядати навіть зображення. Програма правильно розпізнає тип даних, але замість картинки відображає властивості зображення. Цікаво, але Midnight Commander, не вміючи читати документи типу RTF, відмінно справляється з DOC. А адже DOC - це закритий формат, який має структуру на порядок складніше документа «зі складним форматуванням» (Rich Text Format).

Midnight Commander дозволяє створювати закладки, має історію відвідин каталогів. На відміну від інших програм, де подібні елементи частіше реалізовані у вигляді компактних, малоінформативних меню, Midnight Commander дозволяє додавати докладні описи елементів і відображає їх у великому вікні.

Ви запускаєте який-небудь тривалий процес, наприклад, копіювання. Файловий менеджер дозволяє виконувати подібні операції у фоновому режимі. При цьому на екрані не буде відображатися прогрес виконання завдання. Однак ви можете викликати менеджер фонових процесів і подивитися стан тих чи інших завдань.

Midnight Commander - старий, надійний програмний продукт. Незважаючи на те, що сьогодні слово «консоль» у багатьох викликає іронію та асоціації з далеким минулим, файловий менеджер, що працює в текстовому режимі, залишається і донині одним із самих популярних продуктів у своєму класі.

Worker — один із самих незвичайних файлових менеджерів. Перше, на що слід звернути увагу — це відсутність необхідності в установці будь-яких додаткових графічних бібліотек крім X11.

Все управління зосереджене в кнопках, розташованих в нижній частині робочого вікна. Групи кнопок можна переключати за допомогою миші. Кожна кнопка викликає окрему функцію. В якості функцій можуть виступати як вбудовані інструменти Worker, так і зовнішні утиліти, призначенні для обробки файлів.

Worker тісно інтегрується з Midnight Commander і дозволяє використовувати правила обробки архівів даного консольного файлового менеджера. Крім того, по замовчуванню, MC використовується в якості редактора файлів.

В Worker включені тільки інструменти для перегляду текстових файлів. Всі інші дії (перегляд графіки і відео, редагування інших типів даних) виконуються зовнішніми програмами. Текстовий редактор по замовчуванню — xedit, тому що він використовує лише бібліотеку X11.

Серед попередньо заданих функцій (кнопок) можно зустріти перетворення аудіо-інформації, конвертування графіки, монтування пристройів тощо.

Цей файловий менеджер дуже незвичайний, що, звичайно, затрудняє його освоєння. З іншої сторони, всі конкуренти обо використовують могутні графічні бібліотеки.

теки (QT, GTK), або працюють в текстовому режимі. Worker в єтом свете представляється неким компромисснім решением.

5.2.7. *Nautilus*

Nautilus - досить незвичайний файловий менеджер для Linux. Його концепція багато в чому нагадує *Провідник* для Windows.

Робоче вікно Nautilus складається з бічної панелі і головного вікна перегляду файлів і папок. У бічній панелі ви можете відображати дерево каталогів, інформацію про поточний елементі головній панелі, список найбільш важливих місць для відвідування. Крім того, ви можете складати і зберігати в бічній панелі свої нотатки.

Основна привабливість файлового менеджера полягає у вмінні створювати мініатюри для величезної кількості типів документів. Крім мініатюр для графічних типів даних, Nautilus розуміє формат Open Document. При цьому ви бачите невелику копію перших сторінок документів.

Ви можете включити відображення адресного рядка і з її допомогою здійснювати навігацію по FTP і SMB ресурсам. Інструмент «Комп'ютер» в точності повторює «Мій комп'ютер» Windows. Ви відразу бачите повний список локальних і мережевих ресурсів.

Файловий менеджер не містить власних інструментів перегляду і редактування, однак, розуміючи MIME-типи середовища GNOME, при виклику документів, запускає відповідні програми на виконання. Знову простежується чітка аналогія з Провідником Windows,

Так як Nautilus не містить традиційних двох панелей, копіювання файлів виконується або за допомогою контекстного меню (копіювати, вставити), або шляхом перетягування елементів мишею між двома запущеними копіями файлового менеджера.

5.3. Архіватори

5.3.1. *RAR* для *Linux*

Linux-версія всім відомого архіватора WIN RAR - потужний засіб для створення архівів та керування ними. Можливості:

- повна підтримка архівів RAR і ZIP;
- оригінальний високоефективний алгоритм стискання даних;
- спеціальний алгоритм мультимедіа-стискання;
- оболонка з підтримкою технології перетягнути-і-залишити (drag & drop);
- інтерфейс командного рядка;
- управління архівами інших форматів (CAB, ARJ, LZH, TAR, GZ, ACE, UUE, Bz2, JAR, ISO, 7z, Z) ;
- підтримка безперервних (solid) архівів, у яких міра стискання може бути на 10 – 50% більше, ніж при звичайних методах стискання, особливо при упакуванні значної кількості невеликих схожих файлів;
- підтримка багатотомних архівів.

5.3.2. PeaZip

Архіватор може працювати з архівами у форматах ZIP, RAR, CAB, JAR, LZH, 7z, Bzip2, Gzip/tgz, TAR, у власному форматі PEA і таке інше. Також може витягувати вміст iso-образів дисків, дистрибутивів програм, створених за допомогою додатка Nullsoft Install System (NSIS), і витягувати файли стислих html-довідок (CHM). Параметри архівації, які можна вибрати при створенні архіву, змінюються залежно від формату. Наприклад, у zip-архівах можна встановлювати рівень та метод стискання, пароль на відкриття архіву та спосіб створення (новий архів або оновлення того, що існує). Також Peazip вміє розбивати великі файли на рівні частини певного розміру і збирати їх назад у вихідний файл.

5.3.3. File Roller

Фактично, **File Roller** – це не самостійна програма, а просто графічна оболонка для консольних програм архівації. Завдяки File Roller засоби архівації для Linux, які працюють з командного рядка, знаходять зручний та зрозумілий інтерфейс і стають доступними навіть для недосвідчених користувачів. File Roller може працювати з архівами zip, 7z, tar, lha, rar, lzh, ear, jar, war і багатьма іншими, а також здатний читати iso-образи. Можна переглядати вміст архівів, додавати файли в архів або вида-

ляти їх з нього методом перетягування. Є чотири міри стискання, які можна обрати при створенні архіву, від граничної швидкості праці до граничного стискання файлів.

5.3.4. Tar

Tar (англ. tape archive) - формат бітового потоку або файлу архіву, а також назва традиційної для Unix програми для роботи з такими архівами. Програма tar була стандартизована в Posix.1-1998, а також пізніше в Posix.1-2001. Спочатку програма tar використовувалася для створення архівів на магнітній стрічці, а в даний час tar використовується для зберігання декількох файлів усередині одного файлу, для поширення програмного забезпечення, а також за прямим призначенням — для створення архіву файлової системи. Однією з переваг формату tar при створенні архівів є те, що в архів записується інформація про структуру каталогів, про власника і групу окремих файлів, а також тимчасові мітки файлів. Як і інші утиліти Unix, **tar** - спеціалізована програма, яка дотримується філософії Unix: «робити лише одну річ» (у даному випадку - працювати з архівами формату tar), «але робити її добре». Тому tar не створює стислих архівів, а використовує для стискання зовнішні утиліти, такі, як gzip та bzip2. Раніше для стискання використовувалася також утиліта **compress**, яка сьогодні практично не використовується.

6. ВІЛЬНІ ОФІСНІ СИСТЕМИ

6.1. Відкритий офісний пакет програм

6.1.1. Пакет *Open Office*

Сучасну операційну систему важко уявити без зручних засобів роботи з різноманітними документами, тобто без пакету офісних програм. В операційних системах UNIX з цією метою найчастіше використовують **Open Office**, який включає:

- текстовий редактор Writer з багатими можливостями для створення листів, книг, звітів, інформаційних бюллетенів, брошур та інших документів, сумісний з MS Office;
- електронні таблиці Calc, що мають сучасні засоби аналізу, побудови діаграм і можливості ухвалення рішень, очікувані від високоякісних електронних таблиць, сумісний із MS Office;
- редактор презентацій Impress, що забезпечує всі загальні засоби представлення мультимедіа, такі як спеціальні ефекти, анімація та засоби малювання;
- база даних Base;
- векторний графічний редактор Draw;
- растроївий графічний редактор Gimp, програма для створення та обробки растроївої графіки.

Розглянемо можливості цих програм детальніше.

6.1.2. Текстовий процесор Writer

Openoffice.org Writer - текстовий процесор та візуальний редактор гіпертекстів, входить до складу Openoffice.org і є вільним програмним забезпеченням (випускається під ліцензією LGPL).

Writer схожий на Microsoft Word і функціональність цих редакторів приблизно однакова. Надає користувачеві сучасний інструментарій для набору, редагування і форматування документів.

Поряд із звичним інтерфейсом володіє функціональністю:

- створення й оформлення абзаців тексту і сторінок;
- додавання розділів і колонтитулів;
- вставка зображень і мультимедійних об'єктів;
- попередній перегляд і друкування документів;
- запис змін і рецензування текстів;
- автоматичне оформлення змісту, покажчиків і виносок;
- додавання макросів та елементів керування;
- робота з таблицями;
- перевірка орфографії тощо;
- концепція вживання стилів, чудове вирішення, яке полегшує створення та форматування документів.

Робота зі стилями підтримується у всіх компонентах офісного пакета, але саме в OpenOffice.org Writer вона реалізована найповніше.

П'ять груп стилів: абзаци, символи, сторінки, врізки та списки - надають користувачеві безліч можливостей для легкого форматування документів. Спеціальна панель «Стилі і форматування» дозволяє керувати стилями, змінюючи, що існують, та створюючи нові. І чим складніший документ, чим частіше доводиться змінювати його, тим більш очевидною стає перевага стильового оформлення.

Інший інструмент - Навігатор - надає можливості швидкого переходу по документу, вибираючи як орієнтири заголовки, посилання, врізки або інші об'єкти.

Як і для інших компонентів, для Writer існує велика кількість доповнень (розширень), що поліпшують базову функціональність компонента або такі, що надають додаткові функції, наприклад, перевірку граматики або публікацію в Mediawiki.

Writer підтримує велику кількість форматів для імпорту й експорту файлів, у тому числі збереження в PDF і імпорт docx. А відправка файла електронною поштою здійснюється одним натисненням кнопки.

6.1.3. Електронні таблиці Calc

OpenOffice.org Calc - додаток для роботи з електронними таблицями. З його допомогою можна аналізувати дані, які вводяться, займатися розрахунками, прогнозувати, зводити дані з різних листів і таблиць, будувати діаграми та графіки.

Calc - актуальний у бізнес-середовищі компонент Openoffice.org для роботи з електронними таблицями. Інструмент, якому віддають перевагу бухгалтери і менеджери для створення звітності.

Покрокове введення формул у вічка електронних таблиць за допомогою Майстра полегшує формування складних та вкладених формул, демонструє описи кожного параметра і кінцевий результат на будь-якому етапі введення.

Умовне форматування і стилі вічок дозволяють упорядкувати готові дані, а звідні таблиці та графіки показують підсумки роботи.

Більше двох десятків форматів імпорту й експорту файлів, включаючи функції імпорту тексту, дозволяють оперувати практично будь-якими даними. Також за допомогою спеціального інструмента можна імпортувати дані з інших джерел, наприклад, баз даних, а можна створити оновлюваний діапазон, щоб дані, що імпортуються, завжди були актуальні.

Підтримуються зв'язки між різними електронними таблицями і спільне редактування даних (починаючи з версії Openoffice.org 3.0).

Доступні різноманітні налаштування для друку готових листів на принтері: масштаб, поля, колонтитули. А вбудована перевірка орфографії, як у текстовому редакторі, дозволить поліпшити якість готового звіту.

6.1.4. Інструмент для створення презентацій Impress

Openoffice.org Impress – це програма для створення презентацій, у плані інтерфейсу і функціональності, практично ідентична Powerpoint. Може експортовувати презентації у формат SWF, що дає можливість переглядати їх на будь-якому комп'ютері зі встановленим flash-плеєром, також може створювати з презентацій pdf-файлі. Здатна працювати з презентаціями, створеними в Powerpoint, та зберігати їх у споріднених для нього форматах. У Impress не так багато готових презентацій, проте за бажання в Інтернеті можна знайти безліч шаблонів, створених користувачами.

6.1.5. Реляційна база даних Base

Openoffice.org Base - відкрита реляційна база даних, що дозволяє створювати, редактувати і обробляти табличні дані.

Користувачам надається досить великий набір засобів обробки даних і таблиць: редактори форм, запитів, звітів, таблиць БД. З їх допомогою можна аналізувати дані, які вводяться, займатися розрахунками, прогнозувати, зводити дані з різних листів і таблиць, будувати діаграми і графіки.

Редактор запитів дозволяє створювати практично весь спектр sql-запітів до баз даних на вибірку, зміну, додавання даних. Підтримуються вкладені запити і запити з параметрами. Візуальна частина редактора спростить користувачеві-початківцю процес конструювання запитів.

Редактор форм дозволяє скористатися практично будь-яким стандартним інструментом керування вмістом БД. Тут надаються елементи керування текстовими, числовими, бінарними полями, списками, «випадаючими» списками, таблицями, кнопки, календар та інші елементи.

За допомогою Base можна додавати, видаляти, редагувати записи баз даних: MYSQL, HSQLDB (зазвичай поставляється в комплекті з Openoffice.org), POSTGRESQL, Db2, Oracle.

Також можлива робота з таблицями DBF, MS Access, адресними книгами, текстовими файлами, а також електронними таблицями, створеними в Openoffice.org Calc або MS Excel.

Доступ до зовнішніх джерел даних здійснюється за допомогою ODBC, JDBC, SDBC та інших технологій. Повний перелік підтримуваних технологій і джерел даних залежить від використовуваної операційної системи.

Складніше керування і обробка даних виконуються за допомогою макросів і засобів мови програмування OObasic.

Редактор таблиць допоможе створювати таблиці й керувати ними у вбудованій базі даних HSQLDB, а також із деякими можливими обмеженнями для інших баз даних.

У Base є вбудований майстер звітів із базовими функціями зі створення звітів або таблиць чи запитів. Проте існує кращий інструмент створення звітів - Sun Report Builder. Цей інструмент дозволяє створювати звіти практично будь-якої складності. Він підтримує групування, виконання вбудованих призначених для користувача фу-

нкцій, умовне форматування. Результатом роботи SRB є документ Writer або Calc, які надалі можна роздрукувати або редагувати.

За допомогою Base можна створювати джерела даних. Наприклад, організувавши підключення до зовнішньої бази даних у Base та відфільтрувавши потрібні дані sql-запитом, можна працювати з цими даними в Ooo Calc.

6.2. Робота з графікою

6.2.1. Векторний графічний редактор Draw

Openoffice.org Draw - векторний графічний редактор, за функціональністю порівнянюваний з **COREL DRAW**, входить до складу Openoffice.org.

Векторний графічний редактор Draw є інструментом малювання, який використовує векторну графіку. Він містить ряд сервісів, які дозволяють швидко створювати всі види малюнків. Векторна графіка дозволяє зберігати і відображувати зображення у вигляді векторів (дві точки і лінія), а не у вигляді набору пікселів (точок на екрані). Векторна графіка спрощує збереження та масштабування зображень.

Графічний редактор Draw ідеально інтегрований у систему Openoffice.org, що дозволяє здійснювати обмін малюнками між будь-якими модулями системи дуже просто. Наприклад, якщо ви створюєте малюнок у Draw, то легко можете використовувати його в документі Writer за допомогою копіювання і вставки. Ви також можете працювати з графікою безпосередньо в модулях Writer і Impress, використовуючи підмножину функцій та інструментів з Draw.

Пакет включає повнофункціональні «конектори» між фігурами, які можуть використовувати різні стилі ліній і дозволяють виконувати креслення, наприклад блок-схеми. Необхідність наявності векторного редактора у складі офісного пакета не викликає сумнівів. Сфера застосування такого редактора досить широка: від простих малюнків і оголошень до схем, діаграм і креслень.

Окрім власних об'єктів, можна вставляти в малюнки діаграми, формули та інші елементи, створені в інших компонентах Openoffice.org. Draw також підтримує експорт растрових зображень більшості форматів, як поширених, так і спеціальних. Draw має всі необхідні інструменти, властиві векторним редакторам:

- Солучені лінії, розмірні лінії, таблиці.
- Робота з текстом і текстовими ефектами.
- Зміна колірної заливки, тіні, прозорості.
- Розміщення, прив'язка та керування об'єктами за допомогою слайдів, шарів та напрямних.
- Підтримуються різні операції над об'єктами: додання, віднімання, групування та перетворення фігур.
- Малювання тривимірних об'єктів.
- Малювання за допомогою графічних примітивів, кривих Безье;
- Ефекти: освітлення, морфінг та дублювання.

Окрім власних об'єктів, можна вставляти в малюнки діаграми, формули та інші елементи, створені в інших компонентах OpenOffice.org. Draw також підтримує експорт растроїв зображень більшості форматів, як поширених, так і спеціальних.

Використання Галереї (сховища об'єктів) дозволяє упорядкувати наявні зображення, а підтримка стилів тексту економить час при виготовленні однотипних об'єктів.

Готовий малюнок або креслення можна роздрукувати на принтері або експортувати в раstroвe зображення або, наприклад, в PDF.

Максимальний розмір малюнка в Draw - 300x300 см. Починаючи з версії 3.0 OpenOffice.org, Draw підтримує не лише експорт, але й імпорт PDF (необхідна установка спеціального розширення OpenOffice.org).

6.2.2. Редактор растроїв графіки Gimp

Користувачам, які звикли працювати з Adobe Photoshop, інтерфейс GIMP спочатку здаватиметься незручним. Замість звичного вікна програми з головним меню і лінійкою інструментів користувач бачить невелике віконце з купою різних кнопок. Всі зображення відкриваються в окремих вікнах і біля кожного вікна є власна панель головного меню.

Стартове вікно виконує функції своєрідної панелі швидкого доступу, на якій зібрані всі найчастіше використовувані функції. Останні інструменти можна викликати декількома способами – за допомогою головного меню робочого вікна документа, гарячих клавіш або клацнувші по зображеню правою кнопкою миші (замість

звичного меню властивостей поточного об'єкта відкриється повний список функцій редактора, якого можна перетворити на незалежну панель, клацнувши по верхній кромці меню, яке відкрилося).

Відсутність єдиного робочого простору ускладнює роботу на початку, але з цим можна досить швидко освоїтися.

Звичайно, можливості GIMP значно скромніші, ніж в Adobe Photoshop. Професіонали, що працюють з Photoshop та використовують його за повною програмою, не зможуть також ефективно працювати в GIMP, але для любителів цей пакет – незамінний. Багато користувачів використовує можливості Photoshop не більш ніж на 20%, для невеликих нескладних проектів цей «монстр» просто не потрібний. За допомогою GIMP можна створювати логотипи, ретушувати фотографії, перетворювати зображення в різні формати тощо. GIMP підтримує графічні планшети й може бути використаний не лише для обробки зображень, але і для малювання.

У GIMP є всі стандартні інструменти малювання: пензлик, олівець, аерограф, штамп. Усі інструменти можна набудовувати, міняти товщину ліній, форму, прозорість і так далі. Підтримується робота із шарами, допускається редагування окремих каналів, є підтримка альфа-каналу. Можна працювати з текстом за допомогою стандартного інструмента, а також малювати художні емблеми за допомогою спеціальних сценаріїв. Можна працювати з анімованими зображеннями, кадри відображуватимуться як окремі шари. Наявні різні інструменти для виділення (прямокутник, еліпс, вільне, розсіяне і «розумне» виділення, криві Безье), можливість перетворювати зображення за допомогою таких функцій, як обертання, масштабування, нахил і відзеркалення, є «криві», гістограма і традиційні регулювальники, автоматичні режими, які дозволяють «покращувати» зображення одним клацанням кнопки миші. Небажані зміни можна «відкотити», аж до моменту відкриття файла. У GIMP є фільтри, їх не так багато, як у Photoshop, але цілком достатньо.

Можливості GIMP можуть розширюватися за рахунок підключення додаткових модулів, яких у Мережі налічується більше ста. Більше того, нещодавно з'явилася можливість підключати до GIMP плагіни від Photoshop.

6.2.3. Програми для перегляду зображень

Eye of GNOME

Офіційна програма каталогізації та перегляду зображень робочого середовища GNOME. Надає користувачеві базові можливості роботи із зображеннями: масштабування, повноекранний режим перегляду, інтерполяція при збільшенні зображення. Підтримує такі формати даних як ANI, BMP, GIF, ICO, JPEG, PCX, PNG, PNM, RAS, SVG, TGA, TIFF, WBMP, XBM, XPM, а також відображає EXIF-метадані про зображення.

Ksquirrel

«Переглядач» зображень для графічного середовища KDE. Більш функціональний чим попередній. Є навігатор по диску, файлове дерево, ескізи та розширені ескізи, є динамічна підтримка форматів, інтерфейс DCOP, підтримка KIPI плагінів та KEXIF. Підтримує 57 форматів зображень, включаючи SVG, CRW, NEF, MNG, Jpeg2000, XCF, DXF, JPEG, APNG та ін.

Завдання для самостійної роботи

1. Напишіть скрипт, який підмонтує ISO-образ в файлі CD.iso та запустить на виконання скрипт install.sh
2. Напишіть скрипт, який перевірить, чи існує в файлі /etc/resolv.conf символний рядок nameserver 192.168.54.186. Якщо такий рядок не існує, то скопіювати існуючий файл в resolv.conf.old і додати даний рядок в цей файл.
3. Напишіть скрипт, який всі файли з розширенням .sh зробить виконуваними.
4. Виявіть, чи існує інтерфейс eth0.
5. Напишіть скрипт, який виявить, чи існують в даній системі компілятор gcc і чи правильно він працює.
6. Напишіть скрипт, який перевірить, чи існують файли, передані йому як аргументи, та видайте попередження, які файли відсутні.
7. Напишіть скрипт, який підмонтує флешку/dev/sdb1 і виявить, чи існує в архіві casper.squashfs на цій флешці файл linuxrc в кореневій директорії.
8. Напишіть скрипт, який перевірить, чи існують виконувані файли, передані йому як аргументи, та видайте попередження, які файли відсутні або не є виконуваними.
9. Виявіть, чи існує в файлі names.txt символний рядок John Doe.
10. Напишіть скрипт, який підмонтує ISO-образ в файлі CD-ROM.iso та надрукуйте файл isolinux.cfg в кореневій директорії цього образу.
11. Напишіть скрипт, який переіменує всі файли з розширенням *.htm в *.html.
12. Напишіть скрипт, який виявить, чи існують в даній системі броузери firefox, seamonkey, opera, safari.
13. Виконайте асинхронно всі файли програм в директорії /home/Startup
14. Підрахуйте, скільки файлів програм знаходиться в директорії /usr/local/bin
15. Замініть в файлі /etc/profile значення змінної LANG=en_US на LANG=uk_UA
16. Напишіть скрипт, який всі файли з розширенням .ps зробить тільки для читання.

Контрольні питання

1. Що таке файл?
2. Назвіть властивості файлу.
3. Що таке каталог?
4. Наведіть приклади системних каталогів та їх призначення.
5. Які дії визначені над файлами?
6. Для чого призначенні права доступу для файлів та каталогів?
7. Як можна задати права доступу?
8. Призначення прав доступу. Команди chmod, chown, chgrp.
9. Фільтри. Регулярні вирази. Команда grep.
10. Для чого слугує файловий менеджер?
11. Що таке буфер обміну?
12. Як скопіювати або вирізати об'єкт у буфер обміну?
13. Як вставити об'єкт з буфера обміну?
14. Які є способи вилучення об'єкта? Яка між ними відмінність?
15. Як об'єднати усі об'єкти деякого каталогу в групу?
16. Як об'єднати декілька об'єктів у групу?
17. Чи можна скасувати дію команди, яку виконали?
18. Як зберегти створений файл у деякому каталозі?

Література

1. Беляков М. И. , Рабовер Ю. И. , Фридман А. Л. Мобильная операционная система: Справочник – М. : Радио и связь, 1991. - 208 с.
2. Дайсон П. Операционная система UNIX. Настольный справочник – М\.\. : ЛО-РИ, 1997. - 400 с.
3. Дегтярев Е. К. "Введение в UNIX" - М.: МП "Память", 1991. - 96 с.
4. Топхем Д. , Чыонг Х. В. Юникс и Ксеникс –М. : Мир, 1988. - 392 с.
5. Керніган Б. В. , Пайк Р. UNIX – универсальная среда программирования – М.: Финансы и статистика, 1992. - 304 с.
6. Кевін Р. , Фостер-Джонсон Є. UNIX: справочник – СПб: Пітер Ком, 1999. - 384 с.
7. Максимальная безопасность в Linux – К: Изд-во "ДиаСофт", 2000, - 400 с.
8. Немет Є. , Снайдер Г. , Сибиасс С. , Хейн Т. Р. UNIX: руководство системного администратора – К. : BHV, 1997 - 832 с.
9. Открытое программное обеспечение - <http://it.vahu.ru/articles/opensoft.html>
10. Робачевский А. М. Операционная система UNIX – СПб.: БХВ–Санкт-Петербург, 1999. - 528 с.
11. «Свободное ПО» и «ПО с открытым кодом» -
http://www.info-foss.ru/quickstart/freesoft/freeOpensource_soft
12. Україна створить державний Linux -
<http://ukranews.com/uk/news/ukraine/2010/08/31/25919>
13. Шенк Т. Red Hat Linux для системных администраторов. Єнциклопедия пользователя – К: Издательство "ДиаСофт", 2001. - 672 с.
14. Єбен М. , Таймен Б. FreeBSD. Єнциклопедия пользователя – К: ООО "ТИД "ДС", 2002. – 736 с.
15. Ясько М. М. , Сорокін В. В. Файловоі системи UNIX. – Д.: РЕВ ДНУ, 2010. - 64с.