Linux System Administrators' Guide 0.6

# Linux System Administrators' Guide 0.6

Lars Wirzenius


The Linux Documentation Project

This is version 0.6 of the Linux System Administrators' Guide.
Published November 15, 1997.

The LaTeX source code and other machine readable formats can be found on the Internet via anonymous ftp on `sunsite.unc.edu`, in the directory `/pub/Linux/docs/LDP`. Also available are at least Postscript and TeX .DVI formats. The official home page for the book is `http://www.iki.fi/liw/linux/sag/`. The current version can always be found at that location.

This page is dedicated to a future dedication.

In the mean time... I'd like someone who knows him let Terry Pratchett know that his way of using footnotes is rather inspiring.

# Зміст

# Розділ 1

# Деякі зауваження до перекладу

*Розкажи мені, розкажи,*
*Любиш ти чи ні ...*
*А-а-а-га-а-а ...*

*«Documentation is like sex:*
*when it is good, it is very, very good; and*
*when it is bad,*
*it is better than nothing.»*
*- D. Brandon*

Як виявляється, справа технічного перекладу, це не дуже проста справа. І хоча українське «переклад», і вводить в оману своїм звучанням, але переклад текстів зовсім чомусь не походить на перекладання чогось там з одного місця на інше. Просте підставляння українських слів замість англійських не проходить. Звичайно ж, можна зробити і таке, але хто ж тоді буде читати, те що я наперекладав. Якщо за переклад, який би він не був — технічний чи літературний, хтось збирається платити гроші, то тут все зрозуміло, мета проста і ясна («і танкі наши бистри»): треба задовольнити замовника і отримати від нього по таксі, із розрахунку по стільки-то і стільки-то за слово, або ж за сторінку. І, в принципі, для мене, як для перекладача, немає великої різниці, буде хто-небудь читати мій переклад, чи ні. Складність цієї роботи полягає в тому, що я *хочу*, щоб цей переклад хоч хто-небудь та прочитав. Оскільки я не претендую на те, щоб хто-небудь, коли небудь мені що-небудь за все це заплатив, то яка ж іще може бути мета всього цього? Звичайно ж, Ваше, Ваше і Ваше (так і твоє теж) доброзичливе ставлення, вже як не до мене, то хоча б до

написаного. («Думи мої, думи мої, лихо мені з вами, чом ви стали...»)

Які ще перепони крім моїх персональних амбіцій стоять на моєму шляху як технічного перекладача? На жаль, мушу признатися, досить багато. Основною перешкодою, на яку натикаєшся практично на кожному кроці, є неіснуюча термінологія. Скажіть щиро — чи багато хто з Вас знає, що таке технічний *український* переклад? (Га?... Шо?... Ето ви ко мнє?...) А як Ви ставитеся до написання твору в практично неіснуючому на сьогоднішній день жанрі?

Невеликий приклад з особистого життя. Ще будучи в Києві я відвідував засідання клубу системного програмування (чи як там воно називалося), які організовував В.Безруков. Ще тоді, на зорі пробудження української самоповаги в зросійщеному до безпам'ятства народі, був один хлопчина (на жаль, не пам'ятаю зовсім його імені — Агов, відгукніться!), який піднімав на засіданнях питання про українську термінологію в комп'ютерних справах. Одним із прикладів, що він приводив було таке: як би Ви переклали українською англійське *interrupt*? І справді, як? Багато хто (із тих, що живуть в Україні) дасть відповідь: «переривання», звичайно ж, отримане від російського «прерывание». Але ж українським аналогом англійського «to interrupt» є «перепиняти» чи «припиняти». Тому і перекладом для «interrupt» (іменник) має бути «перепин». І так є практично з кожним технічним терміном. На кожному кроці доводиться балансувати на лезі ножа, щоб не скотитися у прірву або ж безтямного мавпування усталених русизмів (переривання), або ж не менш безтямного калькування (транслітерації) англійських термінів українськими літерами (інтерапт).

Кілька слів всього про, якщо дозволите так висловитися, «транслітерування». Читаючи навколокомп'ютерні дописування останнього часу, з жалем помічаю, що в сучасній літературі набуває значного поширення такий собі «приблатньонно-програмьорський слєнґ», де досить вільно допускаються вислови типу «коли бравзер крахонувся, програмьор натиснув резет-кі і комп'ютер став ребутиться».

З величезною радістю (сміх крізь сльози) хочу відзначити, дорогі товаріщі, що все сказане в попередньому абзаці, відноситься тільки до російськомовної літератури. Української технічної літератури мені поки-що читати не доводилося.

Тож, Панове Оптимісти! Маємо шанс! Українська технічна література не засмічена. Зовсім. Нічим. Ну, тобто, зовсім

нічим... І кому ж, як не нам з вами стати біля її колиски. До справи-ж, хлопці, увіковічнимо наші імена на мармурових та вифарбованих у червоне, як Київський університет, стінах Матінки Історії!

## 1.1 Я

На зав'язку — трохи про себе та про своє оточення.

За походженням я — киянин. І більшу частину свого як свідомого, так і несвідомого життя провів в Україні. Але так вже трапилось, що занесла мене доля разом з моїми близькими дружиною та дітлахами аж хто-зна куди — за моря-океани, аж туди, де, як знаючі люди кажуть, сонце з моря сходить.

## 1.2 Ми

Тож посиджуємо ми тут, на березі океану Тихого, краєм ока на гейш з рікшями поглядаємо, та потрушуємося час від часу, бо якась там напасть трапилася з однією плитою океанічною і чогось їй здумалось злізти на іншу плиту — не менш океанічну (ні-ні, нічого такого еротичного).

Але, від отого посиджування та потрушування, коли-не-коли, глянь, та й нападе така собі нудьга, або, красним стилем мовлячи, «туга». І так воно собі тужиться, аж поки не витужиться в бажання — «... чи книжку почитати, як її читаєш ніби, там якесь точило, бороздить твій мозок, наче...» (ВВ). А як бажання з'явилося, то вже легше. Тоді вже пишеться.

А як вже доля розсудила працювати мені системним адміністратором Юніксу, то про що ж мені ще й писати, як не про Юнікс. Ну не буду ж я, справді, писати про **Networking Guide for Microsoft Windows 4.0 Service Pack 3**. Та й не встигну я його написати, бо той **service pack** піде в небуття, а **Microsoft** підсуне нову сви... чи то пак операційну систему з новими **bugs**, які **Microsoft** чомусь вперто продовжує називати чудернацьким словом **features**, і на вивчення яких в мене часу піде більше, ніж на все сидіння, трусіння, та писання разом взяте. Ні, кумцю, я вже краще буду писати про щось таке старе та стабільне, як, наприклад, Юнікс. І буду отримувати собі задоволення від отого пописування.

І ось на все Ваше, Пане Читачу, широке коло (дай Боже, щоб було воно широке, або хоча б щоб було) я виношу цього твора.

З величезною повагаю (ні, справді, це я серйозно) до всих тих, хто таки спромігся добратися до цього місця опусу,
Дмитро Ковальов.

І тому, досить про себе — ще трохи про твора...

## 1.3 ... і переклад

Вся документація в Інтернеті, яка стосується Лінакса - є на диво гарною документацією. В переважній більшості випадків, все написане чудовою, жвавою мовою. І перекладати такі твори є справжнім задоволенням, нема потреби вносити жодного оживляжу в переклад недолугими жартами. Тільки пливеш за думкою автора і намагаєшся не вставити в переклад чогось такого, що не ліпилося б з технічним боком справи.

А оскільки, що я десь колись чув про Юнікс, то й мені руки сверблять вставити щось таке своє (і чується сердите бурчання: «То чому ж не напишеш своє, як така сверблячка?») Тож свою сверблячку я виливаю до «Приміток перекладача». Де в чому моя думка не співпадає з авторовою (це те, що стосується технічного боку), але більшість зауважень стосується все-таки пояснень та тлумачень, щодо того, як я перекладаю тут той чи інший термін (бо мало того, щоб це все було написане українською, варто б ще, щоб це було і зрозумілим).

Декілька слів щодо зауважень про інші системи. Ориґінал твору містить дані тільки про Лінакс і зрозуміло - це ж підручник адміністратора Лінакса. Але я вважаю, що інколи було би корисним вважати також на те, як організовані інші Юнікси. Я працюю системним адміністратором Юнікса і користуюся в роботі в основному SunOS та Solaris. Під час перекладу та редагування твору часто виникають зауваження щодо подібності чи відмінності Лінакса від інших (відомих мені) Юніксів. Тому всі (чи на мою думку тільки варті уваги) такі зауваження я теж вкладаю в примітки перекладача. В основному, як показує мій власний досвід, такі зауваження стосуються саме відмінностей, аніж подібностей між системами. Прошу завважити, що всі ці замітки не носять ніякої системи, вони виникають спонтанно і, звичайно ж не претендують на будь-яку повноту.

Я не маю жадних претензій на те, що мова цього перекладу має бути стандартизована, як взірець технічного українського перекладу. Я навіть не претендую на те, що все тут написане є граматично та стилістично коректним. Я справді запрошую до

дискусій з цих питань, оскільки наша рідна мова заслуговує на увагу та повагу до себе. Якщо Ви маєте, що сказати (чого б то не стосувалося — технічних недоречностей в написаному, кострубатості виразів та невірного, з Вашого погляду, застосування термінології, тощо), я радий буду сприйняти всі зауваження.

Єдине з приводу чого я не збираюся заводити дискусії, так це з приводу «каму ета нада», та проблем «двуязичія» в Україні.

По ходу перекладу я намагався підтримувати такий собі словничок, із термінами, які можуть викликати якісь питання щодо тлумачення чи розуміння. На закінчення цього вступу приводжу те, що вийшло з цього. При подальшій роботі (коригування, правка, тощо) думаю цей словничок буде поповнюватися.

### 1.3.1 Термінологічний словничок та деякі зауваження щодо перекладу термінів

**BIOS - Base Input/Output System** Основна (або головна) система вводу-виводу. Сукупність бібліотек та невеличких програмок, що записуються в постійну пам'ять PC (Увага! тільки PC! - інші комп'ютерні системи не мають BIOS'ів) і забезпечують функціональність найнижчого рівня PC, таку, як роботу з дисками, екраном, тощо. Бібліотеки BIOS'у для вводу-виводу використовуються при роботі в DOS/Windows. Лінакс користується BIOS'ом тільки при завантаженні системи. Обмеження, закладені в BIOS ще на самих ранніх етапах існування PC (сумно відоме 640 кілобайтне обмеження основної пам'яті — та не менш сумно відомий вислів Біла Ґейтса, що ніхто і ніколи не буде потребувати пам'яті більше, ніж 640к, – обмеження в кількості циліндрів диску, доступних BIOS'у до 1024), існують, на жаль і в сучасних BIOS'ах і значно ускладнюють розробку PC Лінаксів. Інші версії Лінаксів (Sparc, Alfa, Ultra Sparc, ARM) позбавлені недоліків та обмежень BIOS'ів, але, на жаль, інші системи не є такими розповсюдженими, як PC.

**DMA - direct memory access** прямий доступ до пам'яті

**I/O - input/output** порт вводу/виводу

**LILO (LInux LOader)** завантажувач Лінакса

**MBR - Main Boot Record** Головний завантажувальний запис

**POST Power On Self Test** автоматичний тест при вмикненні живлення

**Rock Ridge extension** Розширення Рок Рідж

**ROM(read only memory)** дослівний переклад «пам'ять з доступом тільки для читання». В Україні цей тип пам'яті був більш відомий як «ПЗУ» від російського «постоянное запоминающее устройство». Я не зміг відшукати кращого варіанту в українській, тому на часі віддаю перевагу англійській абревіатурі. Маєте кращі ідеї?

**backup** резервування, архівування, створення резервних копій

**backup history** історія резервування

**boot sector** завантажувальний сектор

**booting** завантаження

**bootstrap loader** початковий завантажувач

**buffer cache** буферний кеш

**cache** кеш

**character devices** пристрої з посимвольним доступом

**compressing** компресування, компресія

**data block** блок даних

**default run level** основний робочий рівень

**directory block** блок директорії

**extended partition** розширений розділ

**file access time** час доступу до файлу

**full backup** повне резервування

**incremental backup** «доповнюючі копії"

**indirection block** блок ссилок; Пишучи переклад для indirection block, я перебирав варіанти, які були б достатньо близькими до оріґіналу і в той же час звучали якось більш-менш українською. Серед претендетів були такі, як «блок пересилань», «непрямий блок»(кривий - чи

що?). Були й інші. Але коли я намагався написати «блок пересилань», зуби мої самі собою зціплювалися, на них починав скрипіти пісок та пил далеких, невідомих (або, на жаль, занадто добре відомих) країн, пальці скрючувалися і у голові починали шуміти вітри, нашіптуючи такі до болю знайомі слова, як «сємачкі», «ваучери», «браузери», «проґрамери» та інші «заімствованія» з різних братніх мов світу. Я так і не зміг написати те, що хотів. Тому я зупинився на варіанті «блок ссилок», який хоч і не є близьким перекладом англійського варіанту, але досить близько відбиває функції даного блоку.

**inode** Оскільки саме це слово є абревіатурою в англійській (index node), подаємо його без перекладу чи транскрипції в українській. Має вимовлятися як «ай-ноуд». Його приблизне значення - вузол, вузлова точка.

**interrupt** перепин

**loading, booting** Було перекладено як «завантаження». Похідні терміни від цих слів відповідно такі: «завантажувач», «завантажувальний блок», тощо. Український переклад слова «to load» буде «завантажувати», в той час як «загружати», «загрузчик» походить від російського «грузить».

**lock** замок

**log files** файл реєстрації

**logical partition** логічний розділ

**mail hub** поштовий вузол

**main boot record** головний завантажувальний сектор

**man page** сторінка підказки

**message of the day** `/etc/motd` фраза дня

**networked file system** файлова система мережі

**NIS** Network Information System Інформаційна Система Мережі

**page fault** нестача сторінки пам'яті

**paging** пейджінґ

**partitions** дискові розділи (див. також primary partition, extended partition, logical partition)

**primary partition** основний розділ

**pipe** канал; Інші варіанти, що трапляются в літературі як «труба», «трубопровід» вже аж занадто «залізні».

**random access device** пристрій з довільним доступом

**ramdisk** віртуальний диск

**read ahead** читання наперед

**runlevel** робочий рівень

**script** скрипт; В багатьох (чи не в усіх, які мені доводилося читати) російських перекладних чи власне написаних роботах цей термін перекладається як «сценарій». Мені такий переклад не подобається страшенно. Сценарій як в українській так, власне, і в російській асоціюється перш за все з сценарієм фільму чи п'єси. І на мою думку такий переклад збиває з пантелику читача. Іншими можливими варіантами уявляються «програма командної оболонки» чи «програмний скрипт».

**seek time** час доступу

**signature** підпис (коли йдеться про підпис в заголовку диску, наприклад).

**shadow passwords** тіньові паролі

**shell** командна оболонка

**superblock** суперблок

**swapping** свопінґ

**symbolic link** символічна ссилка, посилання

**system call** системний виклик

**universal time** універсальний час

**user accounts** рахунки користувачів

**user id** ID користувача

**username** ім'я користувача

**write-through** наскрізний запис

### 1.3.2 Терміни, яким би варто знайти кращого перекладу

**root filesystem** коренева файлова система

**boot** завантаження

**user level program** програма рівня роботи користувача; Страшенно паганий переклад. Можливий варіант «програма користувача». Звучить, звичайно краще, але є не зовсім корректним, оскільки не кожна «user level program» є програмою користувача (user program). Точніше кажучи, практично жодна з них не є.

**full backup** повне резервування

**incremental backup** доповнюючі копії

# Розділ 2

# Вступ

*На початку файл був без форми і пустий; і пустота сіяла*
*над поверхнею бітів. І пальці Автора посунулися до поверхні*
*клавіатури. І Автор мовив «Нехай будуть слова», і були слова.*

Це керівництво, «Підручник системного адміністратора Лінакса», описує аспекти адміністрування при користуванні Лінаксом. Підручник призначений не для тих, хто не знає практично нічого про системну адміністрацію (приблизно як «а що ж це таке?»), а для тих, хто досяг принаймні мінімального або більш-менш пристойного рівня в користуванні системою. Цей посібник також не навчить, як встановити Лінакс. Ця інформація подана в «Посібнику по встановленню та перших кроках в Лінаксі"(Installation and Getting Started ). Нижче Ви знайдете більше інформації про існуючі посібники по Лінаксу.

Системна адміністрація - це всі ті речі, які потрібно робити для того, щоб підтримувати комп'ютер в робочому стані. До цього відносяться речі такі, як: створення резервних копій файлів (та відновлення з них тоді, коли це необхідно), встановлення нових програм, створення рахунків для користувачів (та стирання їх тоді, коли вони вже не потрібні), слідкування за тим, щоб файлові системи не псувалися, тощо. Якби комп'ютер був, скажімо будинком, то до системної адмністрації можна було би віднести прибирання, засклення вікон та інші подібні речі. Системна адміністрація не називається обслуговуванням, бо це було б занадто просто. [1]

[1] Є деякі люди, які *саме так* це і називають, але це тільки тому, що вони, бідолахи не читали цього посібника.

Цей посібник побудований таким чином, що багато які його розділи можуть використовуватися незалежно один від одного. Тобто, коли Вам потрібна, скажімо інформація з резервного копіювання[2], Ви можете прочитати саме цю частину. .[3] Є певні сподівання, що така структура полегшить користування книгою, як довідником, і дасть можливість прочитати тільки малесенький розділ тоді, коли це потрібно, замість того, щоб читати все. Однак, цей посібник перш за все і більш за все - це підручник і є довідником тільки дякуючи щасливому співпадінню.

Цей посібник не призначений для того, щоб користуватися ним як єдиним джерелом інформації. Величезна частина іншої документації по Лінаксу не менш важлива для системної адміністрації. Врешті-решт системний адміністратор - це всього-навсього користувач системи наділений спеціальними привілеями та обов'язками. Дуже важливий інформаційний ресурс - це підказки по командам (manual pages), якими треба користуватися кожного разу, коли команда невідома.

В той час, як цей посібник перш за все націлений на Лінакс, загалом він буде корисний для будь-якої іншої системи, яка базується на Юніксі. На жаль, через те, що є так багато розбіжностей серед різноманітних версій Юнікса загалом, та в системній адміністрації особливо, нам залишається мало сподівань на те, що можна буде охопити все. Навіть розповісти про всі варіанти Лінакса було би важко, вважаючи на те, як він розвивається.

Не існує одного офіційного комплекту Лінакса, отже різні люди мають різні установки в їхніх системах, які вони будують для себе. Ця книжка не націлена на будь-який конкретний комплект Лінаксу, навіть при тому, що я користуюся майже виключно системою Debian GNU/Linux. Коли це можливо я намагаюся вказати на розбіжності та вказати на альтернативні варіанти.

Я намагався показати, як речі працюють, а не просто перелічити «п'ять простих кроків» для кожного випадку. Це означає, що тут є інформація, яка не кожному потрібна, але на такі частини вказано, що, якщо Ви користуєтеся стандартною конфігурацією, то їх можна пропустити,. Якщо Ви прочитаєте все, це, звичайно, покращить Ваше розуміння системи, та зробить користування нею та її адміністрування більш приємною задачею.

[2] **Прим. перекл.:** backup *(ДК)*

[3] Якщо Вам повезло і Ви читаєте версію документа, в якому є такий розділ.

Як будь-яка робота, що пов'язана з розвитком Лінаксу, ця робота була зроблена на добровільних засадах: я робив її тому, що вважав, що буду мати задоволення від цього і тому, що я відчував, що ця робота повинна бути зробленою. Однак, як будь-яка добровільна робота, вона має межі у відношенні того, скільки часу я можу на неї відвести і відносно того, скільки досвіду та вмінь я маю. Це означає, що, можливо, ця книга може бути не такою гарною, як, якби робота сплачувалася і я мав кілька років для того, щоб довести її до ідеального стану. Я сподіваюся, що вона досить таки гарна, але я Вас попередив.

Одна річ, де я позрізав кути і не описував все до кінця, це ті області, які вже є й так добре описані в інших вільно доступних джерелах. Це стосується перш за все документації з окремих програм, таких, як наприклад деталі з використання `mkfs)`. Я описую тільки призначення програми і рівно стільки подробиць з її використання, скільки це є необхідним для цього посібника. За більш детальною інформацією я відсилаю читача до цих інших джерел. Звичайно, всі джерела, на які надаються посилання, є частиною набору повної документації з Лінаксу.

Я намагався зробити це керівництво настільки добрим, наскільки це тільки можливо. Я справді хотів би почути будь-які думки про те, як зробити цей посібник кращим. Коструbата мова, фактичні помилки, ідеї про те, які області ще можливо описати, переписати деякі розділи, інформація про те, як різні версії Юнікса роблять подібні речі - я зацікавлений в усьому цьому.

Моя контактна інформація доступна з Інтернету: `http://www.iki.fi/liw/mail-to-lasu.html`. Щоб Ваша пошта пройшла через мої фільтри, які відфільтровують сміття від листів, Вам необхідно прочитати цю сторінку.

Багато хто допоміг мені при створенні цієї книжки, посередньо чи безпосередньо. Особливу подяку я хочу висловити Мату Велшу (Matt Welsh) за натхнення та за лідерство в LDP, Енді Орему (Andy Oram) за повернення мене назад до роботи своїм неоціненним зворотнім зв'язком, Олафу Кірху (Olaf Kirch) за те, що показав мені, що це може бути зроблене та Адаму Ріхтеру (Adam Richter) з Yggdrasil та іншим тим, які мені вказали на те, що для них це теж може бути цікавим.

Стефен Твіді (Stephen Tweedie), Петер Анвін (H. Peter Anvin), Ремі Кард (Rémy Card), Теодор Цо (Theodore Ts'o) дозволили позичити мені в них їхню роботу [4] (і дали

[4] A comparison between the xia and ext2 filesystems, the device list and a description of

можливість цій книжці бути товщою, тобто виглядати солідніше). Я дуже вдячний за це, і маю почуття провини за ранні версії цього документу, в якіх не було відповідних присвят.

Додатково я хочу подякувати Марку Комарінському (Mark Komarinski) за те, що я відіслав йому цей матеріал в 1993 році та за велику кількість статей по системній адміністрації в Лінакс Джорнал (Linux Journal). Вони були дуже інформативними та натхненними.

Величезна кількість коментарів надсилалася великою кількістю людей. Моя мініатюрна чорна дірка в архіві не дає змоги відшукати їхні імена, але ось деякі з них в алфавітному порядку: Paul Caprioli, Ales Cepek, Marie-France Declerfayt, Dave Dobson, Olaf Flebbe, Helmut Geyer, Larry Greenfield and his father, Stephen Harris, Jyrki Havia, Jim Haynes, York Lam, Timothy Andrew Lister, Jim Lynch, Michael J. Micek, Jacob Navia, Dan Poirier, Daniel Quinlan, Jouni K SeppДnen, Philippe Steindl, G.B. Stotte. Мої вибачення тим, кого я забув.

## Типографічні домовленості

**Жирний текст** Використовується для позначення **нових концепцій**, **ПОПЕРЕДЖЕНЬ**, та **ключових слів** в мові.

***кусив*** Використовується для *посилення* в тексті, та інколи в лапках та на початку нового розділу.

*нахилений текст* Використовується для позначення **мета-змінних** в тексті, особливо при написанні командного рядка. Наприклад,

```
ls -l foo
```

де *foo* буде означати назву файла для команди, таких як `/bin/cp`.

`Шрифт типу друкарської машинки` Використовується щоб показати взаємодію користувача з комп'ютером (те, що відбувається на екрані) як, наприклад в

```
$ ls -l /bin/cp
-rwxr-xr-x  1 root    wheel    12104 Sep 25 15:53 /bin/cp
```

the ext2 filesystem. — Порівняння між файловими системами типів xia та ext2, список спеціальних пристроїв та опис файлової системи ext2. Все це вже не є частиною цієї книжки.

Також використовується в зразках програмного коду, будь то програма на Сі, скрипт в мові програмної оболонки (shell script) чи ще що небудь, та для того, щоб показувати фрагменти загальних файлів, таких як, наприклад, файли конфігурації. Коли це необдхідно для чіткості викладу, ці зразки та рисунки будуть заключатися в рамки.

`Клавіша` Вказує, яку клавішу треба натиснути. Ви часто будете зустрічати її в такій формі:

Press `Return` to continue.

## 2.1 Проект по документації Лінакса, The Linux Documentation Project (LDP)

Проект документації Лінакса (або LDP) - це тісна команда письменників, правщиків та редакторів, які працюють разом, щоб забезпечити повну документацію для операційної системи Лінакс. Загальний керівник проекту Грег Ханкінс (Greg Hankins).

Цей посібник є одним з ряду таких документів, що розповсюджуються LDP, серед них: «Посібник користувача Лінакса» (Linux Users' Guide), «Посібник системного адміністратора» (System Administrators' Guide), «Посібник адміністратора мережі» (Network Administrators' Guide) та «Посібник хакера ядра» (Kernel Hackers' Guide). Ці посібники доступні в таких форматах: вихідний формат LaTeX формат .dvi та вихідний формат PostScript з `sunsite.unc.edu` в директорії `/pub/Linux/docs/LDP`.

Ми заохочуємо кожного з хистом до писання чи редагування приєднуватися до нас для покращення документації по Лінаксу. Якщо Ви маєте адресу на Інтернеті, ви можете зв'язатися з Грегом Ханкінсом за адресою `gregh@sunsite.unc.edu`.

**The LDP Rhyme**[5]

A wondrous thing,
and beautiful,
'tis to write,
a book.

I'd like to sing,
of the sweat,
the blood and tear,
which it also took.

[5]The author wishes to remain anonymous. It was posted to the LDP mailing list by Matt Welsh.

It started back in,
nineteen-ninety-two,
when users whined,
«we can nothing do!»

They wanted to know,
what their problem was,
and how to fix it
(by yesterday).

We put the answers in,
a Linux f-a-q,
hoped to get away,
from any more writin'.

«That's too long,
it's hard to search,
and we don't read it,
any-which-way!»

Then a few of us,
joined together
(virtually, you know),
to start the LDP.

We started to write,
or plan, at least,
several books,
one for every need.

The start was fun,
a lot of talk,
an outline,
then a slew.

Then silence came,
the work began,
some wrote less,
others more.

A blank screen,
oh its horrible,
it sits there,
laughs in the face.

We still await,
the final day,
when everything,
will be done.

Until then,
all we have,
is a draft,
for you to comment on.

# Розділ 3

# Огляд системи Лінакс

*Подивився Владика на все, що він зробив,*
*і побачив, що це було гарно.*
*Genesis 1:31*

Цей розділ дає загальний огляд системи Лінакс. На початку розділу описані головні послуги, які надаються операційною системою. Після цього описані (практично без будь-яких подробиць) програми, які надають ці послуги. Мета цього розділу - надати загальне розуміння системи в цілому, з тим, щоб кожна окрема частина потім була би описана в окремому розділі.

## 3.1 Різні складові частини операційної системи

Операційна система Юнікс складається з **ядра** та деяких **системних програм**. Крім цього існують також **прикладні програми**, що повинні робити основну роботу. Ядро є серцем операційної системи[1]. Воно веде облік файлів на диску, запускає програми і виконує їх по черзі, виділяє пам'ять та інші необхідні ресурси різноманітним процесам, отримує пакети з мережі та передає пакети в мережу, тощо. Ядро мало що робить саме по собі, але воно забезпечує необхідні засоби, за допомогою яких всі ці послуги можуть надаватися. Крім цього воно також всім забороняє програмам доступ безпосередньо до апаратури, змушуючи кожного користуватися тільки наданими ядром засобами. Таким чином, ядро забезпечує деякий захист одних користувачів від інших.

[1] Фактично, воно часто помилково вважається самою операційною системою, але це не так. Операційна система надає набагато більше послуг, ніж звичайне ядро.

ядро
операційна система
системна програма
прикладна програма
системний виклик
режим користувача
GCC
мови програмування
компілятор Сі
документація
ігри
менеджер процесів
менеджер пам'яті
драйвер апаратури
драйвер файлової системи
менеджер мережі
область свопінгу
буфер кешу
багатозадачність

Всі засоби, що надаються ядром, доступні при використанні **системних викликів**[2]; див. сторінки підказок (man pages) з розділу 2 для більш повної інформації з цього.

Системні програми користуються засобами, що їх надає ядро для того, щоб виконувати різноманітні функції, ті, які вимагаються від операційної системи. Системні програми, так само, як і всі інші програми в системі, працюють «на вершині ядра» в режимі, який звуть **режимом користувача** [3].

Різниця між системною програмою та прикладною полягає в їх намірах: прикладні програми наміряються принести корисні результати (чи насолодження від гри, якщо це ігрова програма), в той час, як системні програми наміряються зробити так, щоб система працювала. Текстовий редактор - це прикладна програма, `telnet` - це системна програма. Різниця часто дуже розпливчаста, і є життєвою тільки для запеклих каталогізаторів.

Операційна система також може мати компілятори та відповідні їм бібліотеки (GCC та бібліотека Сі в Лінаксі, наприклад), хоча не обов'язково кожна мова програмування повинна бути частиною операційної системи. Документація та ігри також можуть належати до операційної системи. Традиційно операційна система визначалася змістом дискет чи стрічок для установки. У випадку з Лінаксом - це визначення не настільки чітке, бо Лінакс розпорошений по всіх серверах FTP по цілому світу.

## 3.2 Важливі частини ядра

Ядро Лінакса складається з кількох важливих частин: менеджера процесів, менеджера пам'яті, драйверів пристроїв, драйверів файлових систем, менеджера мережі та різноманітних шматочків та частин. Рисунок 3.1 показує деякі з них.

Можливо найважливішими частинами ядра (ніщо інше не працює без них) є менеджер пам'яті та менеджер процесів. Менеджер пам'яті піклується про виділення пам'яті та областей свопінгу для процесів, частин ядра та для буфера кешу. Менеджер процесів створює процеси та відтворює багатозадачний режим роботи перемиканням активних процесів в процесорі.

[2] **Прим. перекл.:** system calls *(ДК)*
[3] **Прим. перекл.:** user mode *(ДК)*

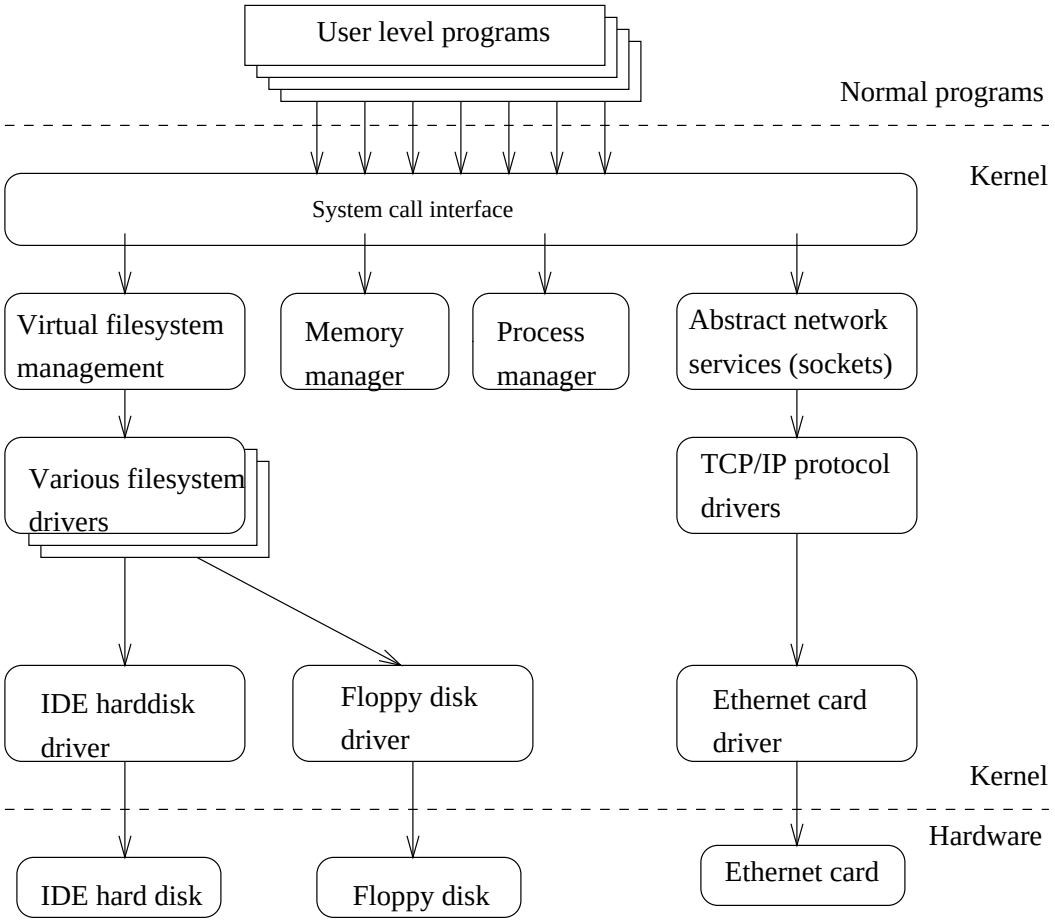


Рис. 3.1: Деякі з важливіших частин ядра Лінакса.

Найнижчий рівень ядра складається з драйверів всіх тих пристроїв, які воно підртимує. Через те, що в світі повно різноманітних пристроїв, драйверів теж багато. Дуже часто існують багато різноманітних апаратних засобів, що виконують подібні функції, але відрізняються тим, як вони керуються з боку програмного забезпечення. Подібності різних драйверів часто роблять можливою розробку «узагальнених класів» драйверів, кожен з яких виконує свої функції. Кожен член такого класу має схожий інтерфейс з іншими частинами ядра, але відрізняється тими частинами, які потребуються для реалізації апаратних функцій. Наприклад, всі драйвери дисків схожі між собою (з точки зору ядра), тобто всі вони мають такі операції, як «ініціалізувати диск», «прочитати сектор N» та «записати сектор N».

віртуальна файлова система
BSD
розетка, сокет
/sbin/init
монтування файлових систем
процес завантаження

Деякі програмні послуги, що надаються ядром, мають подібні властивості, і через це їх можна абстрагувати в класи. Наприклад, різноманітні протоколи зв'язку через мережу виділилися в один програмний інтерфейс, що носить назву «BSD socket library» або «бібліотека сокетів (розеток) BSD». Іншим зразком може служити **віртуальна файлова система** (ВФС або VFS, virtual file system) — абстрактний прошарок, що відділяє операції, які виконуються над файлами від їх конкретного втілення. Кожний окремо взятий тип файлової системи має відмінне втілення для кожної окремо взятої файлової операції. Алк коли деяка істота (фізична особа або процес) намагається використати файлову систему, запит до файлової системи проходить через ВФС, яка, в свою чергу, перенаправляє такий запит до драйвера конкретної файлової системи.

## 3.3 Основні послуги в системі Юнікс

Цей розділ дає опис деяких із найбільш важливих послуг, які надаються системою Юнікс. Ці послуги з більшими подробицями описані в пізніших розділах.

### 3.3.1 `init`

Найперша і найважливіша послуга в Юніксі надається процесом `init`. `init` є першим процесом в кожній системі Юнікс, і старт `init` є останнім кроком, який виконує ядро, коли воно завантажується. Після свого старту `init` продовжеє процес завантаження системи, виконуючи різноманітні задачі, потрібні при старті системи, такі як перевірка та монтування файлових систем, старт демонів (фонових процесів), тощо.

Точний список тих кроків, які робляться `init`'ом насправді залежить від того, до якого типу він відноситься. Є кілька різноманітностей доступних для вибору. `init` як звичайно забезпечує **однокористувацький режим**[4], в якому ніхто не може зареєструватися в систему і користувач `root` користується командною оболонкою на системній консолі; звичайний режим роботи має назву **багатокористувацького режиму**[5]. Деякі версії Юніксів узагальнюють ці поняття під назвою **«робочих рівнів»** [6], при цьому одно- та багато-

[4] **Прим. перекл.:** single user mode *(ДК)*
[5] **Прим. перекл.:** multi user mode *(ДК)*
[6] **Прим. перекл.:** run levels *(ДК)*

користувацькі режими розглядаються як два окремих робочих рівні. Крім цього можуть бути також інші, як, наприклад, рівень, в якому запускається X на системній консолі[7].

робочий рівень
однокористувацький режим
багатокористувацький режим
Система X Window

При звичайній роботі `init` забезпечує роботу `getty`, які дозволяють користувачам зареєструватися в системі та «всиновлює» процеси-сироти (ті процеси, батьки яких, тобто ті процеси від яких вони породилися, вже померли (завершилися); в Юніксі *всі* процеси *повинні* бути в єдиному дереві, тобто всіх сирот потрібно всиновити.)

вбивання процесів
відмонтовування файлових систем
зупинка процесора

Під час вимкнення системи, це саме `init` займається тим, що вбиває всі інші процеси, відмонтовує всі файлові системи та зупиняє процесор. І `init` робить все це паралельно з усіма іншими функціями, на які ще окрім цього він був сконфіґурований.

/sbin/getty
реєстрація в системі
вихід з системи
перевірка паролю

### 3.3.2 Реєстрація з терміналів (login)

Реєстрація в системі з терміналів (через лінію послідовного зв'язку) та з системної консолі (якщо X не працює в даний момент) забезпечується програмою `getty`. Для кожного терміналу, з якого вхід в систему дозволений, `init` стартує окремий примірник програми `getty`. `getty` читає ім'я користувача з клавіатури і запускає програму `login`, яка читає пароль. Якщо пароль і ім'я користувача вірні, програма `login` запускає командну оболонку (shell). Коли командна оболонка припиняє свою роботу (тобто користувач виходить з системи) або коли робота `login` переривається через те, що введений пароль і ім'я користувача не відповідають один одному, `init` помічає це і запускає новий примірник `getty`. Ядро не має ніякого відношення до процесів реєстрації в системі (і навіть не помічає їх), це все забезпечується системними програмами.

### 3.3.3 Syslog

Ядро та багато інших системних програм продукують повідомлення про помилки, попередження та інші важливі повідомлення. Важливим є те, щоб малася можливість пізніше (дуже часто набагато пізніше) переглянути всі ці повідомлення, тобто вони повинні записуватися в файл. Програма, яка це робить - `syslog`. Її можна сконфіґурувати так, щоб вона сортувала повідомлення в різні файли відповідно того, або хто пише повідомлення або від ступені важливості самого

[7] **Прим. перекл.:** runlevel 5 в більшості систем *(ДК)*

повідомлення про помилки
попередження
лог-файли (файли повідомлень)

повідомлення. Наприклад, через те, що повідомлення від ядра дуже часто важливіші від інших повідомлень і їх треба регулярно перечитувати щоб передбачити виникнення проблем, повідомлення від ядра дуже часто відправляються в окремий файл.

### 3.3.4 Періодичне виконання команд: `cron` та `at`

періодичне виконання команд|див. cron та at
тимчасові файли
стирання тимчасових файлів
демони!cron|див. cron
демони!at|див.at

Як користувачі, так і системні адміністратори часто мають необхідність виконувати певні команди періодично. Наприклад, системному адміністратору може знадобитись періодично старати файли в директоріях, що служать для тимчасового зберігання файлів (`/tmp` та `/var/tmp`) від старих файлів, щоб утримувати від забивання диски (не всі програми прибирають після себе так, як потрібно).

Для цього призначений сервіс `cron`. Кожен користувач має власний `crontab`, де він перечисляє ті команди, які він хоче виконати та час коли, ці команди мають виконуватися. Демон `cron` запускає команди тоді, коли потрібно.

Сервіс `at` подібний до `cron`, але він виконує команду тільки один раз: команда виконується в заданий час, але не повторюється.

### 3.3.5 Графічний інтерфейс

Ні Юнікс ні Лінакс не включають інтерфейс з користувачем в ядро. Замість цього вони надають можливість реалізувати його на рівні програм користувача. Це відноситься, як до текстового режиму роботи, так і до графічного.

Така побудова робить систему більш гнучкою, і впровадження нового інтерфейс з користувачем стає дуже легким. Але це також є і певним недоліком. Через те, що інтерфейсів створюється досить багато, система є більш складною для вивчення.

Графічне середовище, що найчастіше використовується з Лінаксом, носить назву X Windows (або просто X). X також не впроваджує інтерфейсу з користувачем, як такого. Воно тільки реалізує віконну систему, тобто ті засоби, за допомогою яких можна реалізувати графічний інтерфейс. Серед найбільш популярних стилів графічного інтерфейсу використовуються такі троє, як Athena, Motif та Open Look [8]станнім часом графічні інтерфейси з'являються, як гриби після дощу. Досить

[8]**Прим. перекл.:** О *(ДК)*

назвати такі, як `Qt`, `GTK` та `GTK+`, `JX`, тощо. Розвиток в Лінаксі пішов зараз іншим шляхом, від того, що був навіть один-два роки тому (це пишеться в квітні 1999 року). Цей розвиток відміняє навіть те, що сказано в попередньому абзаці - тепер часто немає необхідності вивчати новий інтерфейс. Можна просто настроїти необхідну оболонку так, щоб вона реалізувала знайомий графічний інтерфейс. Так, наприклад `KDE`, яка базується на `Qt`, може перемикатися в режими, які імітують такі досить відмінні між собою інтерфейси, як `Macintosh` та `Windows`.

графічний інтерфейс
Система X Window
Athena
Motif
Open Look
цетралізовані обчислення
розподілена
обчислювальна техніка
стійкість проти помилок

### 3.3.6 Мережа

Мережа - це засіб за допомогою якого два або більше комп'ютерів з'єднуються таким чином, що вони можуть вести обмін даними один з одним. Реальні методи, за допомогою яких відбувається таке сполучення,досить складні, але кінцевий результат того вартий.

Операційні системи Юнікс багаті на ресурси для роботи в мережах. Найбільш фундаментальні послуги (файлові системи, друк, збереження інформації, тощо) можуть здійснюватися при сполученні різних комп'ютерів в мережу. Це робить системну адміністрацію набагато простішою, бо дозволяє централізовану адміністрацію. Тим не менше, всі переваги мікрокомп'ютерів та розподіленої обчислювальної техніки (такі, як знижена вартість та висока витривалість) залишаються присутніми.

Однак, цей підручник дуже мало торкається питань адміністрації комп'ютерних мереж. Зверніться до Посібника адміністратора мережі за більш детальним описом цих питань (починаючи з основ роботи комп'ютерних мереж).

### 3.3.7 Реєстрація в системі при роботі в мережі

При роботі в систимі через лінію послідовного зв'язку, для кожного окремого терміналу існує своя виділена лінія послідовного зв'язку. Робота в системі виглядає інакше, коли комп'ютер підключений до мережі[9]. Для кожної особи, що з'єднується з даним комп'ютером, існує окремий віртуальний зв'язок і може існувати довільна кількість з'єднань[10]. Через це

[9] **Прим. перекл.:** Навіть, якщо комп'ютер фізично не підключений до мережі, але Ви працюєте в `X`, ситуація виглядає зовсім інакше, оскільки `X` є *суттєво* орієнтованою на роботу в мережі і розглядає будь-який комп'ютер, як працюючий в мережі. *(ДК)*

[10] Або, принаймні, дуже багато. Все ще пропускна здібність мережі залишається обмеженим ресурсом, і через це все-ще існує верхня межа кількості одночасно дозволених

для кожної нової робочої сесії (тобто для кожного нового віртуального зв'язку) неможливо запускати окремий `getty` . Крім того, існує кілька відмінних методів реєстрації в системі при роботі в мережах TCP/IP - `telnet` та `rlogin` є основними з них.

Замість фіксованих `getty`'ів при реєстрації через мережу, за процес реєстрації відповідає один демон на кожен тип з'єднання (тобто `telnet` та `rlogin` мають кожен свого демона ) і цей демон прислухається до всіх спроб ввійти в систему. Коли він чує, що хтось підключається до системи, він породжує нову копію самого себе, щоб ця копія обслуговувала саме цю конкретну спробу з'єднання. В цей час перший демон продовжує слухати, чи не з'являться інші спроби підключитися до системи. Новостворена копія демона працює подібно до `getty`.

### 3.3.8 Файлова система мережі (Network file systems, NFS)

Одна з найкорисніших справ, що їх може допомогти реалізувати комп'ютерна мережа, це спільне використання файлів за допомогою **файлової системи мережі**. Система, яка використовується найширше, була розроблена фірмою Sun і носить назву Network File System або NFS.

При роботі з NFS будь-які файлові операції, що виконуються програмою на одному з комп'ютерів, передаються по мережі іншій машині. При цьому програма думає, що всі файли, які фактично знаходяться на іншому комп'ютері, розташовані в локальній файловій системі, на цьому комп'ютері, на якому працює дана програма. Такий підхід робить обмін та спільне використання інформації надзвичайно простим, оскільки не вимагає і найменшої модифікації програм.

### 3.3.9 Пошта

Електронна пошта як звичайно є найважливішим засобом спілкування за допомогою комп'ютера. Електронний лист зберігається у файлі в спеціальному форматі, і для читання та передачі кореспонденції використовуються призначені для цього програми.

Кожен користувач має **вхідну поштову скриню** (файл спеціального формату), в якій зберігаються всі нові листи.

робочих сесій при використанні зв'язку через мережу.

поштова скриня
вхідна поштова скриня
черга принтера
спільне користування принтером
взаємовідносини
spooling

Коли хто-небудь відсилає листа, поштова програма відшуковує поштову скриню адресату і дописує листа в кінець поштової скрині. Якщо поштова скриня знаходиться на іншій машині, лист відсилається на цю машину, і ця (віддалена) машина доставляє листа до поштової скрині адресата, тим чином, який вона вважає за найкращий.

Система електронної пошти складається з багатьох програм. Доставка пошти до місцевої чи віддаленої скрині виконується однією програмою (**агентом доставки пошти** (mail transfer agent, MTA): наприклад, `sendmail` або `smail`), в той час як програм, якими користуються безпосередньо користувачі, багато і вони значно відрізняються одна від одної (**агент користувача пошти** (mail user agent, MUA): наприклад, `pine`, `elm`). Поштові скрині звичайно знаходяться в `/var/spool/mail`.

### 3.3.10 Друк

Кожного певного відтинку часу тільки одна особа може користуватися принтером. Але не розподіляти послуги принтера між користувачами є надзвичайно неекономним. Через це принтери керуються програмами, які втілюють **чергу друкування** [11]: всі роботи для друку ставляться в чергу, і коли друкування одного документу закінчується, автоматично починається друк іншого. Це звільнює користувачів від організовування черги «на принтер» та боротьби за контроль над принтером. [12]

Програмне забезпечення черги принтера **скидає** (spool) матеріали призначені для друку на диск, тобто, до того часу, поки робота перебуває в черзі, текст зберігається в файлі. Це дає змогу прикладній програмі швидко «випльовувати» роботи на друк. Прикладна програма не має чекати на те, щоб дана робота була фактично віддрукована, перш, ніж продовжувати роботу. Це дуже зручно, бо дозволяє, наприклад, віддрукувати одну версію документу, і, не чекаючи на результати друку, приступити до редагування наступної версії.

[11] **Прим. перекл.:** print spool, print queue *(ДК)*

[12] Замість цього вони формують чергу *біля* принтера, чекаючи на віддруковані матеріали, через те, що, мабуть, ніхто і ніколи не буде здатний змусити програмне забезпечення відповісти на запитання: коли саме буде віддруковане те чи інше конкретне завдання з черги. Цей невеличкий факт є надзвичайним рушієм взаємовідносин в колективі.

FSSTND
Стандарт файлової системи Лінакс|див.FSSTND
Стандарт файлової системи |див.FSSTND

## 3.4 Побудова файлової системи

Файлова система складається з багатьох частин; до файлової системи root входять `/bin`, `/lib`, `/etc`, `/dev` та кілька інших. Є окрема файлова система `usr` для програм та даних, що не змінюються з часом. Є файлова система `var`, яка містить такі дані, що змінюються, як, наприклад, (файли реєстрації повідомлень - log file). Також є файлова система `home`, де містяться файли, що належать персонально користувачам системи. В залежності від конфіґурації апаратного забезпечення та волі системного адміністратора файлові системи можуть відрізнятися від того, що тут написано, може навіть трапитися, що все разом буде знаходитися на одному дисковому розділі.

Розділ 4 описує побудову файлової системи в подробицях; «Стандарт файлової системи Лінакс» дає опис з іще більшими подробицями.

# Розділ 4

# Дерево директорій

*Два дні по тому там був Пух,*
*сидячи на своїй гілці, колихаючи ногами і*
*там же, позаду нього, були чотири*
*глечики меду...*
*(А.А. Мілн)*

Цей розділ описує важливі частини стандартного дерева директорій Лінакса, базуючись на стандарті файлової системи FSSTND. Він визначає стандартний підхід до розбиття дерева директорій на окремі файлові ситеми з різним призначенням та дає пояснення для чого той чи інший поділ потрібен. Описуються також деякі існуючі альтернативи.

## 4.1 Основи

Цей розділ базується на стандарті файлової системи Лінакса, FSSTND, версії 1.2 (див. бібліографію [?], який намагається встановити стандарт того, як повинно бути організованим дерево директорій в Лінаксі. Позитивним у створенні такого стандарту є те, що з ним буде легше писати чи переносити з іншої системи програми в Лінакс, а також адмініструвати машини з Лінаксом, через те, що все в цьому випадку повинно бути на своїх звичних місцях. За спиною цього стандарту немає жодної авторитетної інстанції, тим не менше, його підтримують майже всі робробники Лінакса. Тому перш, ніж відхилятися від цього стандарту треба добре подумати і мати на це справді важливі підстави. Стандарт FSSTND намагається слідувати традиціям Юнікса та сучасним тенденціям, що робить Лінакс знайомим тим, хто працював з іншими версіями Юнікса і навпаки.

Цей розділ не настільки деталізований, як стандарт FSSTND. Для кращого розуміння системи кожному системному адміністратору варто прочитати FSSTND.

Даний розділ не пояснює призначення кожного файлу в деталях. Наші наміри - не описати кожен файл, а дати загальні відомості про систему з точки зору файлової системи. Більш детальна інформація про кожен окремий файл міститься в інших розділах та сторінках підказок - man pages.

Все дерево директорій Лінакса повинно бути спроектовано таким чином, щоб його можна було розділяти на менші частини. Кожну частину можна помістити на окремому розділі чи окремому диску, щоб ефективніше використовувати його об'єм та полегшити створення резервних копій і виконання інших обов'язків системного адміністратора. Основні частини цього дерева - це коренева файлова система (root), файлові системи `/usr`, `/var` та `/home` (див. мал. 4.1). Кожна з частин файлової системи має своє призначення. Побудова дерева директорій така, що дає можливість ефективної роботи в мережі. Комп'ютери в мережі можуть спільно користовуватися деякими частини файлових систем, що забезпечується через пристрій з доступом тільки на читання (наприклад, CD-ROM) або через мережу з NFS.

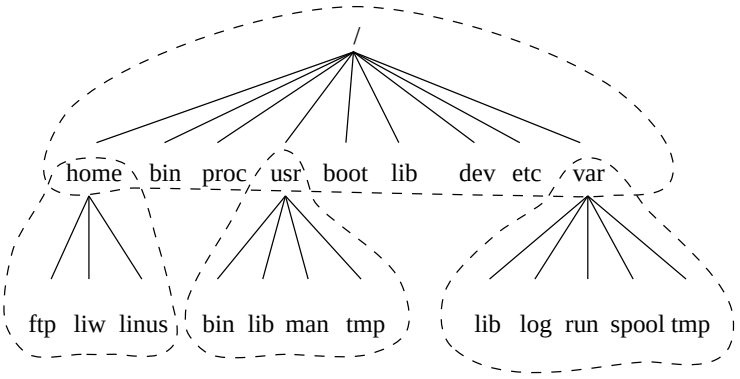


Рис. 4.1: Частини дерева директорій Юнікса. Пунктирні лінії вказують межі підрозділів дисків.

Призначення різних частин дерева директорій подано нижче.

- Коренева файлова система є специфічною для кожного комп'ютера (найчастіше вона зберігається на локальному диску, хоча може бути також і на віртуальному диску в пам'яті (ramdisk) і на диску в мережі). В цій файловій

системі знаходяться всі необхідні для завантаження системи файли і файли необхідні для приведення системи в такий стан, коли на кореневу файлову систему можна змонтувати інші файлові системи. Тобто, іншими словами, зміст кореневої файлової системи повинен бути достатнім для однокористувацького режиму роботи. Вона повинна також мати в собі засоби достатні для ремонту зіпсованої системи та для відновлення загублених файлів з резервних копій.

- У файловій системі `usr` знаходяться всі команди, бібліотеки, довідкові матеріали (man pages) та інші файли, що не змінюються в процесі звичайної роботи системи. Це дозволяє спільно користуватися одними й тими ж файлами в мережі, що в свою чергу зменшує вартість системи (дуже часто `usr` може мати сотні меґабайт) і полегшує адміністрування всієї мережі. Щоб змінити якусь програму в системі, треба модифікувати тільки головну директорію `usr`, а не всі директорії `/usr` на кожній окремій машині. Навіть якщо директорія знаходиться на локальному диску, її можна змонтувати з доступом тільки на читання (read-only). Таким чином можна зменшити шанси пошкодження файлової системи при краху комп'ютера.

- Файлова система `var` містить файли, які змінюються при звичайній роботі системи. Сюди відносяться такі файли та директорії як, наприклад, директорії «спулу» (для електронної пошти, новин, принтерів, тощо), різноманітні файли реєстрації [1], форматовані сторінки підказок [2] та деякі тимчасові файли. Все, що знаходиться в сучасних системах в `var`, традиційно містилося раніше в піддиректоріях файлової системи `usr`, але це затруднювало монтування директорії `usr` без дозволу на запис.

- Файлова система `home` містить домашні директорії користувачів системи, тобто всі «справжні» дані в системі. Виділення домашніх директорій у їх власну файлову систему робить створення резервних копій простішим. Ішні частини системи не потрібно резервувати, або, як мінімум, це не треба робити так часто. Велику

[1] **Прим. перекл.:** log files *(ДК)*
[2] **Прим. перекл.:** catman *(ДК)*

директорію `home` можливо треба буде розбити на менші файлові системи, і тоді потрібно буде розширити схему найменування на рівнях нижчих за `home`, наприклад, `/home/students` та `/home/staff`.

[3]бо користуватися для цього тою або іншою схемою автоматичного монтування директорій. Вибір в Лінаксі зводиться або до користування `amd` - демоном автомонтування BSD, або до `autofs` - «рідної» системи автомонтування в Лінаксі, яка можлива з ядрами 2.x. Як та, так і інша схема мають свої плюси та мінуси, але як ті, так і інші виходять за рамки тематики даного підручника, а скоріше належать до Системної адміністрації мережі. Можливо колись дойдуть руки і до цього.

Незважаючи на те, що різноманітні частини файлової системи Лінакса називалися файловими системами в попередніх абзацах, вони не обов'язково повинні лежати на окремих дискових розділах. В однокористувацькій невеличкій системі або в системі, де на речі дивляться просто, всі розділи легко можна розташувати на одному дисковому розділі. Дерево директорій також можна поділити на підрозділи (або файлові системи) базуючись на інших критеріях. Все залежить від розмірів дисків та від того, як дисковий простір виділяється для тих чи інших потреб. Однак при всякій схемі розподілу, важливою залишається вимога, щоб всі стандартні *назви* працювали. Навіть, якщо, скажімо `var` та `usr` знаходяться на одному підрозділі, то назви `/usr/lib/libc.a` та `/var/adm/messages` повинні існувати. Цього можна добитися, наприклад, перенісши файли із-під `/var` в `/usr/var` та зробивши `/var` символічною ссилкою на `/usr/var`.

Файлова система Юнікс групує файли відповідно до їх призначення, тобто, всі команди знаходяться в одному місці, всі файли даних - в іншому, документація - ще в іншому, тощо. Альтернативним підходом є групування файлів по їх належності до певної програми. Тобто, всі файли, що належать до Emacs'а будуть розташовані в одній директорії, всі файли від TEXá в іншій, і т.п. При такому підході проблемою є те, що значно затрудняється спільне використання деяких файлів (директорії програм часто містять обидві, як статичну так і динамічну версії файлів) і часто навіть звичайний пошук потрібного файлу (наприклад, сторінки підказки по певній програмі) перетворює роботу в маячню.

[3]**Прим. перекл.:** А *(ДК)*

## 4.2 Коренева файлова система

Взагалі кажучи кореневу файлову систему краще мати невеличкою. Вона має містити дуже критичні файли, і мала файлова система, яка змінюється часто, має більше шансів на виживання. Пошкодження кореневої файлової системи у більшості випадків означає, що система неможливо завантажити без застосування спеціальних заходів (як, наприклад, використання гнучких дисків для завантаження). Отож, нею краще не ризикувати.

Коренева директорія загалом не має ніяких файлів, хіба що крім ядра системи, яке за традицією носить назву `/vmlinuz`. Інші файли, що місяться в кореневій директорії:

`/bin` Команди необхідні для завантаження системи, і які можливо будуть використовуватися звичайними користувачами після завантаження.

`/sbin` Команди подібні до тих, що є в `/bin`, але команди з `/sbin` не призначені для звичайних користувачів системи. Хоча, можливо, вони можуть користуватися цими командами, якщо це потрібно і дозволено.

`/etc` Файли конфіґурації специфічні для цієї машини.

`/root` Домашня директорія користувача `root`.

`/lib` Бібліотеки спільного використання, необхідні для програм в кореневій файловій системі.

`/lib/modules` Модулі ядра, що підгружаються, особливо ті з них, які потрібні для завантаження системи при відновленні від пошкоджень (як, наприклад, драйвери пристроїв мережі та файлових систем).

`/dev` Файли спеціальних пристроїв.

`/tmp` Місце для запису тимчасових файлів. Програми, які використовуються після завантаження, повинні користуватися директорією `/var/tmp`, а не `/tmp` через те, що `/var/tmp` скоріше всього знаходиться на більшому диску.

`/boot` Файли, що використовуються системним завантажувачем, таким, як наприклад, LILO. Файли ядра системи часто знаходяться в цій директорії замість кореневої. Коли кількість скомпільваних ядер зростає, ця

директорія може збільшуватися досить швидко, і це може послужити причиною виділення її у власну файлову систему. Іншою причиною щоб мати директорію на своєму власному розділі, може бути та, що таким чином можна забезпечити, що ядро буде завжди знаходитися в межах 1024 цилиіндрів на IDE диску.

`/mnt` Точка для тимчасового монтування директорій системним адміністратором. Інші програми не повинні використовувати її для автоматичного монтування. `mnt` може бути розділеною на піддиректорії (наприклад, `/mnt/dosa` може бути точкою монтування гнучких дисків з файловою системою MS-DOS, а точка `/mnt/exta` може використовуватися для монтування того ж гнучкого диску тільки з файловою системою ext2 на ньому).

`/proc`, `/usr`, `/var`, `/home` Точки монтування інших файлових систем.

### 4.2.1 Директорія `/etc`

Ця директорія має безліч файлів. Деякі з них описуються нижче по тексту. Щодо інших, Ви повинні спершу визначити, до якої програми вони відносяться і звернутися до підказки по цій програмі. Більшість файлів конфіґурації мережі знаходяться в цій же директорії, але описуються в Посібнику адміністратору мережі.

`/etc/rc` **або** `/etc/rc.d` **або** `/etc/rc?.d` Скрипти чи директорії із скриптами, що виконуються при старті системи, чи при зміні робочого рівня. Детальна інформація по цих файлах (директоріям) дається в розділі про `init`.

`/etc/passwd` База даних користувачів системи. Поля бази містять ім'я користувача в системі, справжнє ім'я користувача, домашню директорію, закодований пароль та іншу інформацію про кожного з користувачів. Формат файлу описаний в сторінці підказки *passwd*.

`/etc/fdprm` Таблиця параметрів гнучкого диску. Описує як виглядають різні формати гнучких дисків. Використовується `setfdprm`. Більше інформації можна отримати з сторінки підказки *setfdprm*.

`/etc/fstab` Перелічує всі ті файлові системи, які монтуються автоматично при запуску системи командою `mount -a` (із

скрипту `/etc/rc` чи подібного до нього за призначенням). В Лінаксі містить також інформацію про облаті свопінґу, що автоматично використовуються при виконанні команди `swapon -a`. Більш детальна інформація доступна в розділі 5.8.5 і в сторінці підказки *mount*.

`/etc/group` За призначенням та форматом подібний до файлу `/etc/passwd`, але описує групи користувачів замість індивідуальних даних про них. Див. сторінку підказки *group*.

`/etc/inittab` Файл конфіґурації `init`.

`/etc/issue` Текст, який виводиться на екран командою `getty` при реєстрації в системі. Найчастіше містить короткий опис системи або запрошення до системи. Склад цього файлу залежить від бажання системного адміністратора.

`/etc/magic` Файл конфіґурації для команди `file`. Він містить в собі опис різних форматів файлів. Базуючись на цих форматах `file` визначає тип файлу. Див. сторінки підказок по *magic* та *file* для повного опису.

`/etc/motd` **Фраза дня** [4]. Ця фраза автоматично виводиться на екран після успішної реєстрації в системі. Зміст запису в файлі залежить від бажання системного адміністратора. Часто використовується для того, щоб донести якусь інформацію до кожного користувача, таку як, наприклад, попередження про вимкнення чи профілактику системи.

`/etc/mtab` Список змонтованих в даний момент файлових систем. Напочатку цей файл створюється скриптами при завантаженні системи. Після завантаження він автоматично поновлюється командою `mount`. Цей файл використовується тоді, коли потрібно отримати список змонтованих на даний час файлових систем, як, наприклад, при використанні команди `df`.

`/etc/shadow` «Тіньовий» файл паролів. Є в системах з встановленою підтримкою тіньових паролів. При використанні тіньових паролів, зашифровані паролі зберігаються в `/etc/shadow` замість `/etc/passwd`. Перший з цих двох файлів на відміну від останнього закритий для прочитання для всіх крім користувача `root`. Це підвищує безпеку системи і покращує захист від злому паролів.

[4] **Прим. перекл.:** message of the day *(ДК)*

`/etc/login.defs` Файл конфіґурації для команди `login`.

`/etc/printcap` Файл подібний до `/etc/termcap`, але містить описи принтерів. Має інший синтаксис.

`/etc/profile`, `/etc/csh.login`, `/etc/csh.cshrc` Файли, які виконуються при реєстрації в системі командними оболонками Бьорна та Сі. Ці файли дають системному адміністратору можливість встановити однакові початкові параметри для всіх користувачів системи. Див. сторінки підказки до відповідних командних оболонок.

`/etc/securetty` Визначає безпечні термінали, тобто, такі термінали, з яких дозволено реєстрацію в системі користувачеві `root`. В більшості випадків в цьому файлі вказуються тільки віртуальні термінали системи. Злом системи через мережу або модем стає неможливим (або, як мінімум, набагато важчим).

`/etc/shells` Перелічує командні оболонки «з довірою» [5]. Команда `chsh` дозволяє користувачам змінювати їх початкову програмну оболонку. Але команда змінить командну оболонку тільки на ту з них, яка є в цьому файлі. Процес `ftpd` - процес, що забезпечує FTP сервіс на машині, буде перевіряти, чи є оболонка користувача в списку `/etc/shells` і не дозволить зареєструватися в системі тим користувачам, чиїх оболонок тут нема.

`/etc/termcap` База даних властивостей терміналів. Цей файл описує які «ESC-послідовності» використовуються при роботі з тими чи іншими терміналами. «ESC» послідовності будуть працювати тільки для одного типу терміналу. Тому програми пишуться таким чином, щоб замість того, щоб безпосередньо виводити «ESC» символи на екран, вони відшукують потрібну послідовність у базі `/etc/termcap`. У результаті більшість програм може працювати з переважною більшістю терміналів. Для більш повної інформації з цього питання зверніться до сторінок підказок з *termcap*, *curs_termcap* та *terminfo*.

### 4.2.2 Директорія `/dev`

Директорія `/dev` містить файли спеціальних пристроїв для більшості можливих пристроїв. Всі файли пристроїв носять

[5] **Прим. перекл.:** trusted *(ДК)*

назви відповідно до спеціальних домовленостей. Всі вони описані в списку пристроїв (див. [?]). Файли пристроїв створюються під час установки системи, або після цього за допомогою скрипта `/dev/MAKEDEV`. Скрипт `/dev/MAKEDEV.local` - це скрипт, що створюється системним адміністратором і який створює тільки ті файли пристроїв, які є властивими для даної системи (тобто ті, що не є складовими стандартного `/dev/MAKEDEV`, такі, як файли драйверів нестандартних апаратних засобів).

## 4.3 Файлова система `/usr`

Часто файлова система `/usr` досить велика, через те, що всі програми встановлюються саме тут. Всі файли, що знаходяться в `/usr` попадають сюди при установці Лінакса, локально встановлені програми та всі інші речі часто попадають в `/usr/local`. Завдяки цьому легко можна поновити систему при появі нової версії, або навіть встановити повністю нову систему. При цьому не потрібно встановлювати всі програми з самого початку.

Нижче подано деякі з піддиректорій, що лежать «нижче» директорії `/usr` (деякі з менш важливих опущені, детальніша інформація міститься в FSSTND).

`/usr/X11R6` Система X Window (всі її файли). Для полегшення розробки та встановлення X, всі X файли не були включені в систему вцілому. [6]а відміну від Лінакса, в SunOS та Solaris всі X файли потрапляють в різні директорії під `/usr`: `/usr/bin/X11`, `/usr/lib/X11` і т.ін. Вниз від `/usr/X11R6` «росте» ціле дерево подібне до того, що є внизу під `/usr`.

`/usr/X386` Подібна до `/usr/X11R6` директорія, але призначена для версії X11 Release 5. [7]юди, що починають працювати з Лінаксом вже після того, як цей посібник перекладається можливо вже ніколи не будуть мати справу з `X11R5`, але для спокійнішого життя варто все-таки мати символічну ссилку `/usr/X386`, яка вказує на `/usr/X11R6`(так само варто мати `/usr/X11`, що вказує на ту ж директорію. Час від часу то та, то інша програма починає щось шукати в `/usr/X386`.

`/usr/bin` Майже всі команди, що ними користується користувач. Деякі команди знаходяться також в `/bin` та

[6] **Прим. перекл.:** Н *(ДК)*
[7] **Прим. перекл.:** Л *(ДК)*

`/usr/local/bin`.

`/usr/sbin` Команди для системної адміністрації, які не потрібні в кореневій файловій системі, наприклад, більшість програм серверів.

`/usr/man`, `/usr/info`, `/usr/doc` Сторінки підказки [8], документи інформації [9] GNU та різноманітні файли документації (відповідно).

`/usr/include` Файли заголовків для програм на мові Сі. Насправді, все це повинно-б бути десь в глибинах `/usr/lib`, але традиція перемагає і файли знаходяться тут.

`/usr/lib` Файли даних для програм та підсистем, які не змінюються в процесі роботи (включаючи деякі загальносистемні файли конфіґурації). Назва `/lib` походить від «бібліотека» [10] - початково тут зберігалися бібліотеки для програмування.

`/usr/local` Місце, де встановлюються специфічні для даної системи програми та інші файли.

## 4.4 Файлова система `/var`

Файлова система `var` зберігає дані, які змінюються при звичайній роботі системи. Ці дані є специфічними для кожної окремо взятої системи, тобто вони не використовуються спільно кількома комп'ютерами в мережі. [11]е завжди, в деяких випадках виявляється вигідніше монтувати NFS навіть деякі директорії під `/var`. Див. нижче про `/var/spool/mail`

`/var/catman` Кеш для відформатованих за потребою сторінок підказок. Неформатовані тексти сторінок підказок звичайно зберігаються під `/usr/man/man*`. Деякі строрінки можуть бути встановленими в уже відформатованому вигляді, і зберігаються вони в `/usr/man/cat*`. Але інші сторінки потрібно відформатувати перш, ніж переглядати. Після форматування такі стрінки зберігаються в `/var/man`, тобто при наступному перегляді вже не потрібно чекати повторного форматування. Директорію `/var/man` потрібно

[8] **Прим. перекл.:** man pages *(ДК)*
[9] **Прим. перекл.:** Info *(ДК)*
[10] **Прим. перекл.:** library *(ДК)*
[11] **Прим. перекл.:** H *(ДК)*

час від часу спорожняти так само, як це робиться для тимчасових директорій.

`/var/lib` Файли, які змінюються при нормальній роботі системи.

`/var/local` Змінні дані для програм, які встановлені в `/usr/local` (тобто програм встановлених системним адміністратором). Зауважимо, що навіть локально встановлені програми повинні користуватися іншими директоріями під `/var` коли це їм необхідно.

`/var/lock` Файли «замків» [12]. Багато які з програм відповідно до загальної домовленості створюють замки в `/var/lock`, щоб вказати, що той чи інший спеціальний файл є замкненим в даний момент (під час роботи даної програми). Інші програми, помітивши замок не намагатимуться користуватися пристроєм, якщо його спеціальний файл замкнено.

`/var/log` Файли реєстрації [13]різноманітних програм, особливо програм `login` (`/var/log/wtmp`, в якому містяться записи про всі реєстрації (входи) в систему та виходи з неї) та `syslog` (`/var/log/messages`, в якому записуються повідомлення від ядра системи та системних програм). Файли в `/var/log` потенційно можуть зростати нескінченно, і від адміністратора вимагається спорожняти їх через певні проміжки часу.

`/var/run` Файли, що містять інформацію про систему від одного старту системи до іншого. Так `/var/run/utmp` містить інформацію про осіб, які користуються системою в даний момент.

`/var/spool` Директорії для черг електронної пошти, новин, принтерів та інших можливих черг. Кожна окрема черга має свою власну директорію під `/var/spool`, тобто поштові скриньки користувачів зберігаються в `/var/spool/mail`. [14] деяких випадках буває зручніше монтувати `/var/spool/mail` як NFS директорію з поштового сервера. Наприклад, розглянемо організацію з великою розгалуженою мережею робочих станцій і одним основним розподільником пошти (поштовим вузлом (mail

[12] **Прим. перекл.:** lock *(ДК)*
[13] **Прим. перекл.:** log files *(ДК)*
[14] **Прим. перекл.:** В *(ДК)*

hub)). Якщо пошта розподіляється з головного поштового розподільника по робочих станціях, кожна робоча станція буде мати тільки поштові скрині осіб, які працюють на даній станції. Отож, неможливо прочитати свою пошту при переході на інший комп'ютер. Звичайно, можна користуватися POP-поштою, але це тема для іншої розмови і крім того вона має свої проблеми. Крім того на поштовому вузлі потрібно буде постійно підтримувати в робочому стані файл `/etc/aliases` або базу даних користувачів, в якому зберігаються адреси (поштові «псевда») користувачів системи - де, на якій робочій станції, той чи інший користувач хотів би читати свою пошту. Вихід із цієї ситуації виявляється надзвичайно простим. Достатньо організувати одну-єдину директорію `/var/spool/mail` на поштовому вузлі і монтувати її на кожній робочій станції, як проблеми відпадають самі собою. Кожне робоче місце буде мати всі поштові скриньки змонтовані в `/var/spool/mail` і кожен користувач зможе читати свою пошту на будь-якому робочому місці.

`/var/tmp` Великі тимчасові файли або такі, яким дозволено жити довше, ніж тим, що живуть в `/tmp`. (Хоча системний адміністратор може також заборонити занадто довге перебування і в `/var/tmp` також.)

## 4.5 Файлова система `/proc`

Файлова система `/proc` є ілюзорною. Вона не існує на диску. Замість цього ядро створює «зображення» файлової системи в пам'яті. Використовується вона для надання інформації про систему (початково використовувалася для надання відомостей про процеси, звідси її назва). [15]olaris має також файлову систему `proc`. Тут вона і є саме тим - вона містить тільки інформацію про процеси. Пояснення до деяких з її важливіших файлів та директорій подано далі. Подробиці `/proc` описані в сторінці підказки по *proc*.

`/proc/1` Директорія з інформацією про процес з номером 1. Кожен процес має власну директорію в `/proc`. Назва директорії відповідає номеру процесу.

`/proc/cpuinfo` Інформація про процесор, така як його тип, виробник, модель та виробнича потужність.

[15]**Прим. перекл.:** S *(ДК)*

`/proc/devices` Список драйверів пристроїв сконфіґурованих в працюючому ядрі на даний момент .

`/proc/dma` Показує які канали ПДП (прямого доступу до пам'яті) [16] використовуються на даний момет.

`/proc/filesystems` Файлові системи сконфіґуровані в ядрі.

`/proc/interrupts` Показує які перепини [17] знаходяться в користуванні.

`/proc/ioports` Які порти вводу/виводу[18] знаходяться в використанні.

`/proc/kcore` Точний образ фізичної пам'яті системи. Цей образ має точнісінько такий розмір, як і фізична пам'ять в комп'ютері, але не забирає ніякої пам'яті у системи. Цей образ створюється «на льоту» в той час, як програми звертаються до пам'яті. (Пам'ятайте: жоден файл чи директорія, які знаходяться в `/proc` не займають жодного байта пам'яті на диску чи в оперативній пам'яті (до тих пір поки Ви не скопіюєте їх куди-небудь).

`/proc/kmsg` Повідомлення, які видаються ядром. Крім цього вони також перенаправляються в `syslog`.

`/proc/ksyms` Таблиця символів ядра.

`/proc/loadavg` «Середня завантаженість» системи. Три позбавлені змісту індикатори, які показують наскільки система завантажена.[19]озволю собі не погодитися з твердженням про «змістовність» даних індикаторів. Три числа, які вказують на завантаження системи, є середня кількість процесів, які стоять в черзі на доступ до процесора, зафіксована в три різні моменти часу, а саме: на час відкриття файлу `/proc/loadavg` (Чи на час виконання команди `uptime` в системах, які не мають `/proc/loadavg`, як, наприклад, SunOS, Solaris чи Лінакс з ядром до 1.3.59), п'ять хвилин тому та п'ятнадцять хвилин тому. В системі з одним користувачем, процесор майже завжди спить, прокидаючись лиш на короткі миті щоб віддати коротке розпорядження комусь із підлеглих - дискам, пам'яті чи дисплею та щоб знову заснути після

[16] **Прим. перекл.:** DMA - direct memory access *(ДК)*
[17] **Прим. перекл.:** interruprts *(ДК)*
[18] **Прим. перекл.:** I/O - input/output *(ДК)*
[19] **Прим. перекл.:** Д *(ДК)*

цього. Тому і завантаження такої системи завжди буде не більше, ніж 0,1-0,2. Пристойно завантажений сервер покаже 2-3 процеси, яких мучить безсоння чи не дає спати турбота про користувача. Якщо ж `loadavg` фіксує навантавження біля 10-20, то це є ознакою того, що щось негаразд із даною системою - з якоїсь причини навантаження на сервер занадто велике.

`/proc/meminfo` Інформація про користування пам'яттю, як оперативною, так і свопінґом.

`/proc/modules` Завантажені в даний момент модулі ядра.

`/proc/net` Інформація про протоколи мережі.

`/proc/self` Символічна ссилка на директорію процесу тої програми, яка в даний момент заглядає в `proc`. Якщо два процеси одночасно дивляться в `/proc`, обидва з них отримують по власній ссилці. В основному це зроблено для зручності, щоб дати можливість програмам потрапляти у власні директорії процесів.

`/proc/stat` Різноманітна статистика системи, така як, скажімо, кількість нестач сторінок пам'яті[20] з часу вмикнення системи.

`/proc/uptime` Час, на протязі якого система працює.

`/proc/version` Версія ядра.

Варто зазначити, що хоча файли з `/proc` є звичайними текстовими файлами, їх фомат не завжди легкий для людського ока. Тому багато які з команд роблять тільки те, що читають файли в `/proc` і видають їх на екран у «людському» форматі. Наприклад, команда `free` читає файл `/proc/meminfo` та конвертує його в представлення пам'яті в кіло та меґа байтах та додає від себе деяку інформацію.

[20] **Прим. перекл.:** page fault *(ДК)*

# Розділ 5

# Використання дисків та інших носіїв інформації

*На пустому диску можна шукати до нескінченості.*

При установці чи поновленні системи Вам доводиться виконувати досить багато маніпуляцій з Вашими дисками. На них потрібно створити файлові системи таким чином, щоб можна було зберігати потрібні файли і залишити необхідний простір для різноманітних частин системи.

Цей розділ пояснює всі ці початкові дії. В більшості випадків після того, як Ваша система встановлена і обладнана, Вам не потрібно проходити через всі ті ж самі етапи знову, крім хіба що, як при використанні гнучких дисків. Але Вам потрібно буде повернутися до цього розділу, якщо Ви додаєте нові диски до системи, чи хочете точніше керувати використанням дискового простору.

Основні задачі при адмініструванні дисків такі:

- Форматування диску. Щоб підготувати диск до роботи, на цьому етапі виконуються різноманітні кроки, такі, як наприклад, перевірка дефектних блоків диску. (В даний час для більшості дисків форматування не потрібно.)
- Розбиття диску на розділи. Диск потрібно розділяти на окремі підрозділи в тому випадку, коли необхідно виділити простір для різних видів діяльності, які (види діяльності) не повинні заважати один одному. Одним з приводів для розділу диску може бути бажання встановити кілька операційних систем на одному диску. Іншим - бажання тримати на іншому розділі ті файли, що не відносяться до загальносистемних, з метою полегшення резервування даних та щоб зберегти систему від суцільного краху.

- Створення файлової системи (потрібного типу) на кожному окремому з розділів диску. Диск не означає практично нічого для Лінакса до того часу, поки на ньому не створена файлова система. [1]рім, хіба що своп-простору, який в багатьох Юніксах не вимагає від системного адміністратора практично ніяких дій. Юнікс може користуватися своп-простором зразу ж після того, як створений розділ диску. В Лінаксі спочатку треба виконати компнду `mkswap`, яка створює «файлову систему». Насправді Лінакс записує в заголовку своп-розділу тільки спеціальний підпис[2], а користується все-таки «сирим» підрозділом. Після цього можна створювати файли та звертатись до них.

- Монтування різних файлових систем, щоб сформувати єдину структуру (або автоматично, або ж вручну, тільки тоді, коли це необхідно. Вручну змонтовані файлові системи у більшості випадків потрібно і розмонтувувати також вручну.).

В розділі 6 міститься інформація про віртуальну пам'ять та дисковий кеш, про які теж варто мати деяке поняття, при користуванні дисками.

Даний розділ пояснює, що Вам потрібно знати про гнучкі та жорскі диски, CD-ROM та приводи магнітної стрічки.

## 5.1 Два типи пристроїв

Юнікс, а також і Лінакс, розрізняють два відмінних між собою типи пристроїв: блочні пристрої з довільним доступом[3] (якими є диски) та пристрої з посимвольним доступом[4] (прикладами яких є магнітні стрічки та послідовні лінії зв'язку), деякі з яких можуть бути з послідовним доступом, а інші - з довільним. Кожен пристрій, який підтримується системою, представлений в файловій системі **спеціальним файлом пристрою**. Під час читання чи запису в файл пристрою, дані приходять чи відправляються до пристрою представленого спеціальним файлом. Таким чином не потрібні жодні спеціальні програми (або ж якісь спеціальні методології програмування, як, наприклад, перехват перепинів чи

[1] **Прим. перекл.:** к *(ДК)*
[2] **Прим. перекл.:** signature *(ДК)*
[3] **Прим. перекл.:** random access *(ДК)*
[4] **Прим. перекл.:** character devices *(ДК)*

опитування послідовного порту) для того тільки, щоб звертатися до пристроїв. Наприклад, для того, щоб просто відправити файл на принтер, достатньо просто виконати

```
$ cat filename > /dev/lp1
$
```

і зміст файлу буде роздрукований на принтері (звичайно ж файл повинен бути в форматі зрозумілому для принтера). Однак, зважаючи на те, що важко назвати гарною звичкою пересилання файлу прямо на принтер кожним окремим користувачем, люди звично користуються спеціальною програмою для цього (в більшості систем `lpr`). Ця програма гарантує, що тільки один файл передається на принтер для друку в кожен момент часу і автоматично передає на друк наступний, як тільки впорається з друкуванням попереднього. Щось подібне потрібно також для більшості інших пристроїв в системі. Справді, користувачі не повинні турбуватися про якісь там «пристрої» практично ніколи.

Через те, що пристрої є звичайними файлами в файловій системі (в директорії `/dev`, дуже легко просто подивитись які з пристроїв існують за допомогою команди `ls` чи будь-якої іншої підходящої команди. Перший стовпчик на екрані при виконанні команди `ls -l` містить тип файлу та дозволи на цей файл. Наприклад, дивлячись на спеціальний файл послідовного порту на своїй системі я бачу:

```
$ ls -l /dev/cua0
crw-rw-rw-   1 root     uucp       5,  64 Nov 30  1993 /dev/cua0
$
```

Перша літера в першому стовчику, тобто '`c`' в вищенаведеному прикладі в `crw-rw-rw-` показує доскіпливому користувачеві тип файлу, тобто, в цьому випадку, символьний спеціальний файл. Для звичайних файлів цей символ буде '`-`', для директорій - '`d`', а для блочних пристроїв - '`b`'. Детальна інформація подана з цього питання в сторінці підказки для `ls`.

Завважте, що навіть якщо пристрої не встановлені на даному комп'ютері, файли спеціальних пристроїв існують все-одно. Отже, те, що Ви маєте файл `/dev/sda`, ще не свідчить про те, що Ви справді маєте SCSI диск підключений до системи. Зберігання всіх цих пристроїв в системі просто робить програми установки трохи простішими і від цього стає легшою установка нового обладнання (не потрібно

відшуковувати вірні параметри для апаратного пристрою та створювати спеціальні файли для нього).

## 5.2 Жорсткі диски

Даний підрозділ ознайомить читача з термінологією, що має відношення до жорстких дисків. Якщо Ви вже знайомі з термінами та концепціями, Ви можете пропустити його.

Рисунок 5.1 показує схематичну діаґраму найважливіших частин жорсткого диску. Жорсткий диск складається з однієї або більше циркулярних **пластин**[5] на яких одна або обидві **поверхні** покриті магнітною речовиною, яка і використовується для запису даних. Для кожної такої поверхні існує власна **записуючо-відтворююча головка**, яка аналізує або змінює записані на поверхні дані. Всі пластини обертаються на спільній осі, типова швидкість обертання складає 3600 обертів на хвилину, але диски з вищою виробничою потужністю мають вищі швидкості. Головки можуть пересуватися вздовж радіуса пластин, і цей рух, поєднаний з обертанням пластин, дає головкам можливість доступу до будь-якої частини диску.

Процесор (CPU) і реальний диск спілкуються через **дисковий контролер**. Це звільнює всі інші частини комп'ютера від необхідності знати, як саме користуватися тим чи іншим диском, бо дискові контролери можуть виготовлятися таким чином, щоб мати один і той-же інтерфейс до комп'ютера. Коротше кажучи, комп'ютер просто може сказати «гей, дисче, дай мені те, що я хочу», замість того, щоб передавати довгі послідовності електричних сигналів тільки щоб посунути головку у відповідне положення і чекати після цього, щоб потрібне місце диску попало під читаючу головку. Насправді, інтерфейс до дискового контролера все ще досить складна штука, але все-таки набагато простіше, ніж він міг би бути. Крім того, контролер може робити деякі інші речі, такі, як кешування або автоматичну заміну дефектних секторів.

Наведені вище дані, практично все, що необхідно знати середньо-пересічному користувачеві про обладнання. Крім згаданого є ще ціла в'язка всіляких інших причандаль, таких, як двигун, що обертає пластини, електронні штучки, які керують роботою механічних частин, але все це не має

[5]Пластини виготовляються з твердої речовини, такої, як алюміній, звідки і походить назва жорсткого диску

відношення до розуміння принципів роботи жорстких дисків.

Магнітні поверхні, як звичайно, розділені на концентричні кільця, що носять назву **доріжок**, а ці в свою чергу поділені на **сектори**. Такий поділ використовується для визначення положення на поверхні жорсткого диску та для виділення вільного простору для файлів. Щоб визначити потрібне місце на диску звичайно кажуть «поверхня 3, доріжка 5, сектор 7». Найчастіше число секторів є сталим для всіх доріжок на диску, але деякі диски приділяють більше секторів на зовнішні доріжки (всі сектори будуть мати в такому випадку однаковий фізичний розмір, а, отже, більше секторів поміститься на ближчих до зовнішнього периметру доріжках). Типовий сектор має 512 байт даних. Диск сам по собі не може оперувати об'ємами даних меншими, ніж розмір сектора.

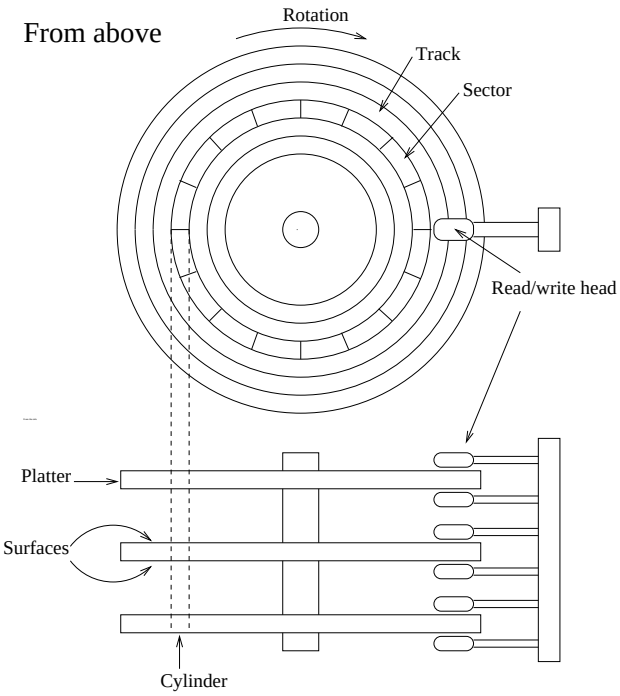


Рис. 5.1: Схематичне зображення основних частин жорсткого диску.

Кожна поверхня диску розділена на доріжки (і сектори) так само, як і інші. Це означає те, що, якщо головка на одній з поверхонь знаходиться над певною доріжкою, то головки на всіх інших магнітних поверхнях знаходяться над такою ж поверхнею. Всі відповідні доріжки разом взяті називають **циліндром**. Пересування головок з однієї доріжки (циліндру)

до іншої займає певний час. Отже, дані краще розташувати таким чином, щоб дані, до яких доступ повинен виконуватися одночасно (наприклад, файл), розташовувалися в одному і тому ж циліндрі. В такому разі відпадає необхідність пересування головок при доступі до цих даних і, отже, це підвищує швидкість роботи. Не завжди виявляється можливим розташувати файли таким чином, і файли, які виявляються записаними розкиданими по кількох різних фізичних розділах диску носять назву **фрагментованих**.

Число поверхонь, або головок (що, по суті, є одним і тим же), циліндрів та секторів суттєво відрізняються[6]. Сукупність всіх цих параметрів називають **геометрією** жорсткого диску. Геометрія диску часто записується в спеціальній пам'яті, що живиться від батарейки, і носить назву **CMOS RAM**[7]казане в попередньому абзаці відноситься, по суті, тільки до PC-Лінаксів.

Запис геометрії диску в CMOS є специфічною ознакою PC BIOS'ів. Інші системи користуються більш розвиненими методами визначення геометрії дисків. Всього лиш кілька слів для прикладу - в SunOS та Solaris є спеціальна база даних типів дисків, в якій ставиться у відповідність різні типи дисків та їх геометрії. Тип диску визначається системою з його етикетки, яка записана в заголовку самого диску. Sparc Лінакс теж не користується BIOS'ом для визначення геометрії дисків з тої простої причини, що комп'ютери Sun просто не мають BIOS'ів. Я не знаю, як саме Sparc Лінакс визначає геометрію дисків, тому буду радий, якщо хтось з Вас поділиться зі мною знаннями.

Також більш сучасні BIOS'и (практично всі, що випускаються на сьогоднішній день - я не зустрічав в останній час жодної нової материнської плати PC без цієї установки) вміють визначати типи дисків під'єднаних до системи. І від користувача не вимагається вносити зміни в установки BIOS'у після зміни дисків.

. З цієї пам'яті операційна система дізнається про диск під час старту системи або драйверів.

На жаль BIOS[8][9] має певні обмеження закладені при проектуванні, через які виявляється неможливим звертатися до

[6]**Прим. перекл.:** Від однієї моделі диску до іншої, звичайно, а не в процесі роботи. *(ДК)*

[7]**Прим. перекл.:** C *(ДК)*

[8]BIOS - це певна програма для початкового старту системи, яка записується в ROM комп'ютера. Вона відпрацьовує початковий старт системи від вмикнення живлення до того моменту, коли управління передається операційній системі.

[9]**Прим. перекл.:** (див. ROM) *(ДК)*

доріжок, номер яких більший за 1024 (тобто вказувати цей номер в CMOS RAM), що є занадто малим для сучасних жорстких дисків. Щоб перейти через цю межу контролер придурюється, що він не знає справжньої геометрії диску і **переводить адреси**, запрошені ком'ютером в такі, які більше підходять до рельної ситуації. Наприклад, жорсткий диск може мати 8 головок, 2048 доріжок і 35 секторів[10]. Його контролер може дурити комп'ютер, що він має 16 головок, 1024 доріжки і 35 секторів на доріжку, не перевищуючи таким чином верньої дозволеної межі доріжок і обчислює справжні адреси на диску виводячи їх з тих, що надаються йому комп'ютером. Насправді математика позаду такого обчислення може бути значно складнішою, бо числа можуть бути не такими гарними та гладенькими, як в наведеному прикладі (але, знову ж таки, це не має відношення до загального розуміння принципу роботи). Таке переформування фізичних адрес викривлює уявлення системи про справжню організацію дисків і робить безперспективними спроби оптимізації швидкості доступу за рахунок розташування даних в межах одного циліндру.

Ці проблеми переобчислення адрес стосуються тільки IDE дисків. SCSI диски використовують послідовні номери секторів (тобто, контролер переводить послідовний номер сектора в триплет «головка-циліндр-сектор») і зовсім інший метод для спілкування з процесором, отже вони повністю ізольовані від описаної проблеми. Відмітьте, однак, що і у випадку із SCSI дисками комп'ютер також може не знати справжньої геометрії диску.

Через те, що Лінакс часто не має і найменшого поняття про справжню геометрію диску, він навіть і не думає помістити якісь дані в межах одного циліндру. Замість цього він намагається приписати кожному файлу послідовні сектори, і це в результаті дає приблизно такий же виграш в часі. Ця проблема навіть більше ускладнюється кешами контролера дисків та автоматичним «читанням наперед»[11] контролера.

Кожен жорсткий диск має свій власний спеціальний файл. Загалом може бути два або чотири жорстких диски IDE в системі. Вони відомі під назвою `/dev/hda`, `/dev/hdb`, `/dev/hdc` та `/dev/hdd`. SCSI диски мають назви `/dev/sda`, `/dev/sdb` і т.д. Відповідні домовленості щодо назв дисків існують також для інших типів пристроїв. Детальні дані про типи пристроїв маються в [?]. Майте на увазі, що спеціальні файли для цілих

[10] чисто умовні числа

[11] **Прим. перекл.:** read ahead *(ДК)*

дисків мають доступ до дисків цілком, не звертаючи уваги на розділи дисків (про це йдеться далі), і при цьому виявляється надзвичайно легко наробити лиха з усіма розділами, якщо не бути уважним. Спеціальні файли дисків частіше всього використовуються, щоб отримати доступ до MBR [12] (що теж буде обговорюватися далі).

## 5.3 Гнучкі диски

Гнучкі диски складаються з гнучкої мембрани покритої з одного або обох боків магнітною речовиною, що схожа на ту, яка використовується при виготовленні жостких дисків. Гнучкий диск сам власне не має записуючої чи читаючої магнітної головки. Головка входить в склад приводу. Гнучкий диск аналоґічний до однієї пластини жорсткого диску з тією різницею, що його можна виймати з комп'ютера і один і той же привід можна використовувати для багатьох дисків, в той час, як жорсткий диск - нероздільний.

Подібно до жорсткого диску гнучкий також розділений на доріжки і сектори (де дві відповідні доріжки на обох боках диску утворюють циліндр), але об'єм інформації, що записується на гнучкий диск значно менший.

Привід гнучкого диску в більшості випадків може користуватися кількома різними типами дисків, наприклад, 3.5 дюймовий дисковід може використовувати як диски відформатовані на 720 кБайт, так і диски відформатовані на 1.44 МБайт. Приймаючи до уваги, що з кожним окремими форматом система повинна поводитися трохи інакше, ніж з іншим, і система повинна розуміти, скільки інформації містить той чи інший диск, стає зрозумілим, чому для кожного дисководу мається по кілька різних спеціальних файлів. Отже, `/dev/fd0H1440` відповідає першому дисковому приводу (`fd0`), який має бути 3.5 дюймовим дисководом, який використовує 3.5-дюймові дискети високої щільності запису (`H`), об'ємом 1440 кБайт (`1440`), або, просто кажучи, звичані трьохдюймові дискети. Більше інформації про назви спеціальних пристроїв дисководів має [?].

Назви дисководів в Лінаксі досить складні, отож був створений спеціальний тип спеціального файлу для дискет, який самостійно визначає тип дискети, яка знаходиться в дисководі. Він по черзі намагається прочитати перший сектор

[12] **Прим. перекл.:** main boot record - головний завантажувальний запис *(ДК)*

дискети, користуючись різними форматами до тих пір, поки не відшукає потрібний. При цьому, звичайно, потрібно, щоб дискета спершу була відформатована. Автоматичні пристрої дискет називаються `/dev/fd0`, `/dev/fd1` і т.д.

Параметри, які автоматичний драйвер використовує для доступу до дискет можна задавати за допомогою програми `setfdprm`. Це може знадобитися, якщо Ви користуєтесь дискетами, які не відповідають ніяким стандартам форматування, тобто мають незвичайну кількість секторів, або, якщо з якоїсь причини автоматичне визначення формату на спрацьовує, або ж, якщо відповідний спеціальний файл драйвера на знайдено в системі.

Додатково до всіх стандартних форматів дискет Лінакс може працювати з багатьма нестандартними форматами. Для деяких з них потрібні спеціальні програми для фотматування. Ми не будемо зараз зупинятися на них, але Ви можете самостійно подивитися файл `/etc/fdprm`. В ньому визначаються всі ті формати, які підтримує `setfdprm`.

Операційна система повинна знати, коли замінюється диск в приводі, щоб не використовувалися дані від попереднього диску (які можуть залишатися певний час в пам'яті). На жаль сигнальна лінія, що служить для цього часто буває зіпсованою, і, крім того, що найгірше, зіпсовану лінію часто буває важко помітити при роботі в MS-DOS. Якщо Ви помічаєте, що привід дискети починає себе поводити «дивно», то описане може бути проблемою цього. Цьому можна зарадити тільки одним чином - відремонтувавши привід.

## 5.4 Приводи CD-ROM

Привід CD-ROM використовує покритий полімером диск для оптичного считування інформації. Інформація записується на поверхні диску[13] у вигляді маленьких 'дірочок' розташованих по спіралі від центру до країв. Привід спрямовує лазерний промінь для считування інформації із спіральної канавки. Коли промінь попадає у впадину, він відбивається одним чином, а коли на гладку поверхню - іншим. Завдяки цьому можна легко закодовувати інформацію у вигляді двійкових кодів. Все інше - дурнички - звичайна механіка.

Порівняно з жорсткими дисками, приводи CD-ROM –

[13]Тобто на поверхні всередині диску — на металевому диску всередині полімерного покриття.

повільні. Якщо типовий жорсткий диск має час доступу[14] менше 15 мілісекунд, швидкому CD-ROM'у потрібно на це десяті долі секунди. Дійсна швидкість передачі інформації досить висока - сотні кілобайт за секунду. Повільність роботи приводів не дозволяє використовувати їх для заміни жорстких дисків, хоча це і можливо (деякі компанії продають компакти Лінакса з 'живою' файловою системою на ньому, яка робить установку Лінакса простішою і одночасно зберігає дисковий простір). Найбільш придатні компакт-диски для установки нового програмного забезпечення, бо швидкість роботи не має такого великого значення при цьому.

Існує кілька різних способів організації інформації на компакті. Найбільш відомий стандарт - міжнародний стандарт ISO 9660. Він визначає дуже мінімалістську файлову систему, яка навіть примітивніша за MS-DOS'івську. З іншого боку вона настільки проста, що кожна сучасна операційна система здібна використовувати її і відображати її на 'рідну' файлову систему.

Для нормального використання в Юніксі ISO 9660 не годиться. Через це було розроблене розширення до неї, назване «Розширення Рок Рідж»[15]. Рок Рідж дозволяє використання довгих імен файлів, символічні ссилки та інші Юніксовські причандалля, і робить звичайний компакт більш-менш схожим на сучасну файлову систему Юнікса. Крім того файлова система Рок Рідж все ще залишається нормальною файловою системою ISO 9660, що дозволяє використовувати її іншими операційними системами. Лінакс підтримує обидві з них, як ISO 9660 так і Рок Рідж, причому розширення розпізнаються та використовуються автоматично.

Але файлова система - все ще тільки пів дороги. Більшість компакт-дисків мають записані програми для доступу до даних на цьому ж диску, і, на жаль, більшість цих програм не працюють в Лінаксі (крім, хіба що, за допомогою dosemu - емулятора MS-DOS для Лінакса).

Привід компакт-диску доступний через відповідний спеціальний файл пристрою. Привід може бути підключений до комп'ютера одним із кількох можливих способів: через SCSI інтерфейс, через звукову плату або через EIDE. Хакерські зміни до програм, для того, щоб це стало можливим випадають з поля зору цієї книжки, але тип з'єднання визначається спеціальним файлом. Деталі дивіться в [?].

[14] **Прим. перекл.:** seek time *(ДК)*
[15] **Прим. перекл.:** Rock Ridge extension *(ДК)*

## 5.5 Стрічки

Привід магнітної стрічки використовує стрічку подібну до[16] магнітофонних касет. Стрічка є послідовною за своєю натурою, що означає, що для того, щоб дістатися до якогось певного місця на стрічці, Ви повинні пройти через всі попередні записи. На відміну від стрічки, до даних на диску можна доступатися в довільному порядку, тобто Ви можете перестрибнути до будь-якого бажаного місця на диску. Через послідоний доступ до даних на стрічці, стрічки - надзвичайно повільні пристрої.

З іншого боку, стрічки відносно дешеві при виготовленні, саме через те, що їм не потрібно бути швидкими. Крім того їх можна зробити довгими і тому вони можуть містити великі об'єми даних. Тому, в основному, стрічки використовуються для архівування та створення резервних копій, при яких не потрібні високі швидкості, але необхідні дешевизна та велика ємність.

## 5.6 Форматування

**Форматування** - це процес запису спеціальних позначок на магнітних носіях, які використовуються для позначення доріжок та секторів. До форматування магнітна поверхня - це повна мішанина магнітних сигналів. Після цього в хаос вноситься певних порядок за рахунок проведення меж доріжок та магнітних позначок, які розділяють їх на сектори. Справжній процес не настільки простий, але це не стосується суті справи. Що є насправді важливим - це, те, що диском неможливо користуватися без попереднього форматування.

Трохи вводить в оману термінологія, що застосовується тут - в MS-DOS слово 'форматування' відноситься до процесу створення файлової системи (про що йдеться трохи далі). Але насправді існує два процеси, які часто об'єднуються, особливо для дискет. Тоді, коли треба відрізняти ці два процеси, те, що справді є форматуванням називають **ф**орматуванням низького рівня[17]ow-level formatting , а про створення файлової системи говорять, що це - **ф**орматування високого рівня[18]igh-level formatting. В світі Юнікса про ці два процеси говорять 'форматування' та 'створення файлової системи', отже це і будуть ті терміни, які використовуються в цій книжці.

[16]Але, звичайно ж, зовсім іншу.
[17]**Прим. перекл.:** l *(ДК)*
[18]**Прим. перекл.:** h *(ДК)*

Для IDE та деяких SCSI дисків форматування фактично виконується на заводі і його не потрібно робити вдруге, тобто, більшість людей взагалі не повинні турбуватися про форматування. Насправді, форматування може навіть привести до того, що диск буде працювати гірше, наприклад, через те, що, можливо, форматувати необхідно якимось спеціальним чином, так, щоб автоматично замінювати дефектні сектори.

Диски, які потрібно або можна форматувати, часто вимагають спеціальної програми для цього - логіка, що використовується для форматування відрізняється від диску до диску. Форматуюча програма часто або знаходиться в BIOS'і контролера або постачається у вигляді програми для MS-DOS, ні тим, ні іншим не просто користуватися із Лінакса.

Під час форматування можна помітити деякі дефектні місця на диску, що називаються **дефектними блоками** або **дефектними секторами**. В деяких випадках привід турбується про них самостійно, але навіть якщо це так, якщо кількість дефектів збільшується, то щось треба з ними робити для того, щоб не користуватися дефектними місцями на диску. Логіка, що застосовується для цього, вбудована в файлову систему. Далі буде йтися про те, як додати таку інформацію до файлової системи. Додатково можна створити невеликий розділ на диску, який буде покривати тільки дефектні області, бо при занадто великій кількості збійних блоків навіть файлова система може мати з ними певні труднощі.

Дискети форматуються командою `fdformat`. Як параметр команді повинна надаватися назва спеціального файлу дискети. Наприклад, для форматування 3.5-дюймової дискети високої щільності використовується така команда:

```
$ fdformat /dev/fd0H1440
Double-sided, 80 tracks, 18 sec/track. Total capacity 1440 kB.
Formatting ... done
Verifying ... done
$
```

Відмітимо, що якщо Ви хочете користуватися спеціальним файлом з автоматичним визначенням типу дисекети, Ви *повинні* встановити параметри дискети командою `setfdprm` до цього. Наступні команди будуть мати той ж результат, що й попередні:

```
$ setfdprm /dev/fd0 1440/1440
```

```
$ fdformat /dev/fd0
Double-sided, 80 tracks, 18 sec/track. Total capacity 1440 kB.
Formatting ... done
Verifying ... done
$ $
```

Слід користуватися типом спеціального файлу, що відповідає типу дискети і не варто форматувати дискету на об'єми більший, ніж той, на який вона розрахована.

`fdformat` також перевірить дискету, тобто, помітить збійні блоки. Він буде намагатися записати (відформатувати) дефектні блоки по кілька разів (Ви це почуєте, звук приводу при цьому змінюється суттєво). Якщо дискета не надто зіпсована (невеликі часточки бруду, що попали на записуючу головку, деякі дрібні помилки читання, тощо), `fdformat` пропустить їх, але суттєві помилки спричинять зупинку процесу перевірки. Ядро надрукує повідомлення про кожну з помилок вводу/виводу, всі вони будуть направлятися на консоль або в файл `/usr/adm/messages`[19]обто мається на увазі `/var/log/messages`. Більшість Юніксів традиційно користуються директорією `/var/adm` для всіх файлів реєстрації повідомлень, алі Лінакс змінив цю традицію і ввів директорію `/var/log`, тож вона і мала б бути тут вказаною, але в ориґіналі твору було вказано саме цю назву. Наприклад, в моєму RedHat 5.0, на якому це пишеться, такої директорії (`/usr/adm`) немає, тож будьте уважними з назвами директорій. якщо працює `syslog`. `fdformat` однак не буде повідомляти, де саме трапилася помилка, але саму це якраз дуже мало кого хвилює в наш час, оскільки при дешевизні гнучких дисків зараз набагато легше (і часто дешевше -Д.К.) викинути збійну дискету.

```
$ fdformat /dev/fd0H1440
Double-sided, 80 tracks, 18 sec/track. Total capacity 1440 kB.
Formatting ... done
Verifying ... read: Unknown error
$
```

Для того, щоб відшукати збійні блоки на будь-якому диску (в тому числі і на дискеті) можна скористатися командою `badblocks`. Вона не форматує диску, тобто нею можна скористуватися навіть для перевірки існуючих файлових систем. Для прикладу розгляньте команду, яка перевірила 3.5 дюймову дискету і видала інформацію про два збійних блоки:

[19]**Прим. перекл.:** т *(ДК)*

```
$ badblocks /dev/fd0H1440 1440
718
719
$
```

Команда `badblocks` просто друкує номери знайдених нею збійних блоків. Більшість файлових систем вміють обходити збійні блоки. В файловій системі міститься список із збійними блоками. Цей список створюється при створенні самої файлової системи, але до нього можна додавати нові блоки і пізніше. Спочатку список збійних блоків створюється командою `mkfs` (Яка створює файлову систему)[20]росто для довідки: в SunOS та Solaris аналоґом команди `mkfs` є команда `newfs`, різниця невелика, але спробуй-но відформатуй диск в SunOS, якщо до цього користувався тільки Лінаксом., але після цього перевіряти файлову систему потрібно командою `badblocks` і додавати нові блоки до списку збійних треба командою `fsck`. Опис команд `mkfs` та `fsck` іде далі.

Більшість сучасних дисків достатньо «розумні» для того, щоб визначати свої власні збійні блоки і щоб спробувати їх відновити, використовуючи інші зарезеровані для цієї мети блоки. Ця операція непомітна («прозора») для операційної системи. Якщо Вас цікавить, чи це є подібна функція у Ваших дисках, то, мабуть, ця функція повинна бути описана в документації до дисків. Але навіть такий диск може відмовити, коли число збійних блоків збільшується занадто. Але, напевне, що коли таке трапиться, то диск буде нарешті вже настільки застарілим, що, певне, у Вас не виникне навіть і бажання ним користуватися.

## 5.7 Розділи

[21]

Жорсткий диск можна розділити на кілька **розділів**. Кожен такий розділ може працювати, як окремий диск. Ідея полягає в тому, що, наприклад, якщо Ви маєте диск, то, можливо, захочете встановити кілька операційних систем на ньому. В цому випадку можна розділити диск на кілька підрозділів. Кожна операційна система користується своїм розділом, як їй захочеться і не допускається до інших розділів. Таким чином різні операційні системи можуть мирно

[20]**Прим. перекл.:** П *(ДК)*
[21]**Прим. перекл.:** partitions *(ДК)*

співіснувати на одному і тому ж жорсткому диску. Якби це було не можливо, потрібно було б купувати окремий диск для кожної встановленої системи.

Розділи неможливо створити на дискетах. Не існує ніяких технологічних перешкод щоб це зробити, але оскільки такі розділи будуть занадто малими, навряд чи комусь це знадобиться. Оскільки компакт диски набагато зручніше використовувати як одне ціле, вони теж практично ніколи не розбиваються на розділи. І дуже рідко виникає потреба мати кілька операційних систем на одному компакті [22].

### 5.7.1 Головний завантажувальний запис, завантажувальні сектори та таблиця розділів.

[23].

Всі дані про те, як диск розділений на розділи, зберігаються в його найпершому секторі (тобто - в першому секторі першої доріжки на першій магнітній поверхні). Цей перший сектор [24] і є **головним загрузочним записом** (MBR) диску, це є той запис, який читається BIOS'ом і починає відпрацьовувати, коли комп'ютер стартує. В MBR записується невелика програма, яка читає таблицю розділів дисків, перевіряє, який з розділів є активним (тобто, з нього можна завантажити систему) і читає перший сектор цього розділу - **завантажувальний сектор** цього розділу (MBR - це теж завантажувальний сектор, але він - спеціальний серед секторів і тому називається інакше). В цьому іншому завантажувальному секторі записана інша невеличка програмка, яка читає початок операційної системи записаної в даному розділі (якщо, звичайно, з цього розділу можна завантажитися) і після цього передає управління операційній системі.

Схеми розбиття на розділи не вмонтовані в апаратуру комп'ютера, і навіть BIOS не знає про них. Це всього-навсього домовленість, яка виконується багатьма операційними системами. Правда, не всіма операційними системами, але ця

[22] **Прим. перекл.:** Хоча це і можливо і навіть, як свідчить мій власний досвід, використовується дуже часто. Компакт може мати кілька підрозділів, видимих для різних систем. Наприклад, дуже багато програмного забезпечення розпровсюджується в вигляді диску з двома розділами - один з файловою системою FAT для MS-DOS та його похідних Віндовсів, а другий з файловою системою HFS для Макінтошів. Коли цей диск вставляється в Windows, то він, звичайно-ж, не бачить файлової системи для Мака (HFS) і навпаки. І тільки в Лінаксі можна змонтувати обидві з них. *(ДК)*

[23] **Прим. перекл.:** Головний завантажувальний запис:MBR - Main Boot Record, завантажувальні сектори: boot sectors, таблиця розділів: partition table *(ДК)*

[24] **Прим. перекл.:** з номером 0, як і все, що стосується комп'ютерів, починається з 0, а не з 1 *(ДК)*

меншість з них є скоріше виключенням. Деякі операційні системи можуть користуватися одним розділом на диску і поділяють цей розділ всередині на підрозділи. Системи цього типу мирно співіснують з іншими системами (включаючи Лінакс), і для них не потрібні ніякі спеціальні підходи. Але операційні системи, які не підтримують розділи диску не можуть співіснувати з іншими системами зовсім.

Для власної безпеки завжди краще мати таблицю розділів диску записаною в себе на аркуші паперу, так що навіть зіпсовану таблицю можна було б відновити після краху (якщо зіпсована тільки таблиця розділу, Ви не загубите своїх файлів). Зіпсовану таблицю можна відновити за допомогою команди `fdisk`, а подивитися, яка Ваша таблиця - командою `fdisk -l`:

```
$ fdisk -l /dev/hda

Disk /dev/hda: 15 heads, 57 sectors, 790 cylinders
Units = cylinders of 855 * 512 bytes

   Device Boot  Begin   Start     End  Blocks   Id  System
/dev/hda1           1       1      24   10231+  82  Linux swap
/dev/hda2          25      25      48   10260   83  Linux native
/dev/hda3          49      49     408  153900   83  Linux native
/dev/hda4         409     409     790  163305    5  Extended
/dev/hda5         409     409     744  143611+  83  Linux native
/dev/hda6         745     745     790   19636+  83  Linux native
$
```

### 5.7.2 Розширені та логічні розділи

Початкова схема розділу дисків на розділи, яка прижилася в світі «пі-сі» дозволяла мати всього чотири розділи. Дуже швидко виявилося, що цього недостатньо (просто тому, наприклад, що багато хто хоче мати більше чотирьох систем на своєму комп'ютері: Лінакс, DOS, OS/2, Minix, FreeBSD, NetBSD чи Windows/NT). Але в основному тому, що часто краще мати кілька розділів для однієї системи. Наприклад, простір для свопінґу для підвищення швидкості роботи (див. далі) краще помістити на його власний розділ замість того, щоб тримати на основному.

Щоб перейти через ці проектні обмеження пізніше були введені **розширені розділи**[25]. Це нововведення дозволяє

[25]**Прим. перекл.:** extended partition *(ДК)*

розбити **основний розділ**[26] на підрозділи. Таким чином розділений основний розділ називають **розширеним розділом**, а підрозділи, на які розбитий основний розділ - **логічними розділами**[27]. Вони працюють так саме, як і основні [28] розділи, але процес створення їх інший. Швидкість їх роботи не відрізняється від основних розділів.

Структура диску може мати вигляд, як показано на рисунку 5.2. Диск розділений на три основні розділи, другий з яких розділений на два логічних розділи. Частина диску не належить жодному розділу. Весь диск, як вцілому, так і кожний з його основних розділів мають по завантажувальному сектору.

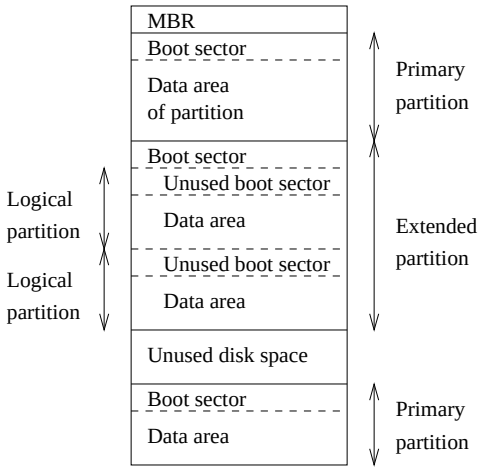


Рис. 5.2: Зразок розділу диску.

### 5.7.3 Типи розділів диску

Таблиці розділів диску або завантажувальні сектори (один з яких є MBR, а інші знаходяться на розширених розділах) мають по одому байту (тобто - один байт на кожен дисковий розділ - чи то основний, чи то логічний), в якому записується тип відповідного розділу. Це робиться для того, щоб можна було визначати тип операційної системи, якій належить даний розділ, або призначення даного розділу. Мета цього - запобігти випадкам, коли дві різні операційні системи можуть випадково скористуватися одним і тим же розділом. Однак, насправді,

[26] **Прим. перекл.:** primary partition *(ДК)*
[27] **Прим. перекл.:** logical partition *(ДК)*
[28] Нелогічні?

операційні системи не дуже то стурбовані різними байтами розділів. Наприклад, Лінакс не звертає на цей байт жодної уваги. Що ще гірше - деякі з операційних систем користуються цим байтом невірно. Наприклад, як мінімум деякі версії DR-DOS'у ігнорують найстарший біт даного байту, в той час, як інші - відносяться до нього з повагою.

Жодна стандартизаційна інстанція не може вказати, яке значення байту відповідає чому, але деякі найбільш вживані величини приведені в таблиці 5.1. Ця ж сама таблиця мається в команді `fdisk` Лінакса.

Табл. 5.1: Типи розділів жостких дисків (з програми `fdisk` Лінакса).

| | | | | | |
|---|---|---|---|---|---|
| 0 | Empty | 40 | Venix 80286 | 94 | Amoeba BBT |
| 1 | DOS 12-bit FAT | 51 | Novell? | a5 | BSD/386 |
| 2 | XENIX root | 52 | Microport | b7 | BSDI fs |
| 3 | XENIX usr | 63 | GNU HURD | b8 | BSDI swap |
| 4 | DOS 16-bit <32M | 64 | Novell | c7 | Syrinx |
| 5 | Extended | 75 | PC/IX | db | CP/M |
| 6 | DOS 16-bit ≥32M | 80 | Old MINIX | e1 | DOS access |
| 7 | OS/2 HPFS | 81 | Linux/MINIX | e3 | DOS R/O |
| 8 | AIX | 82 | Linux swap | f2 | DOS secondary |
| 9 | AIX bootable | 83 | Linux native | ff | BBT |
| a | OS/2 Boot Manag | 93 | Amoeba | | |

### 5.7.4 Розділення жорсткого диску на розділи

Існує багато програм для створення та знищення розділів. Більшість операційних систем мають свої власні варіанти таких програм, і тому, взагалі кажучи, уявляється доцільним користуватися в кожній операційній системі її власною програмою для створення розділів цієї операційної системи, на той випадок, якщо така програма робить щось дивне чи не задокументоване, або таке, чого інші робити не вміють. Багато варіантів таких програм називаються `fdisk` , включаючи і Лінакс[29]дається, що це відноситься тільки до 'PC-похідних' операційних систем. 'Справжні' Юнікси (типу SunOS, Solaris) мають програму, що називається `format`. Основна відмінність між цими двома в тому, що `format` (наслідуючи кращі традиції Юніксів) розуміє команди, тоді, коли вони приходять не з клавіатури, а з скрипту (із STDIN), тобто мається можливість автоматизувати процес розбиття диску на розділи, що

[29]**Прим. перекл.:** З *(ДК)*

надзвичайно важливо в близьких до промислових умов, коли потрібно множити ('клонувати') комп'ютерні системи покладаючись на автоматизацію. Можливість скриптування процесів завжди була і буде найбільшою перевагою Юнікса над іншими загальновживаними системами.. Подробиці використання `fdisk` наведені в сторінці підказки для неї. Команда `cfdisk` подібна за функціями до `fdisk`, але має кращий вигляд на екрані (повноекранний інтерфейс).

При використанні IDE дисків стартовий розділ (тобто розділ, на якому міститься ядро операційної системи) повинен лежати повністю в межах перших 1024 циліндрів диску. Це вимагається через те, що при старті системи диск використовується через BIOS (до того, як система переходить в захищений режим[30]нову ж таки - данина 'важкому дитинству' PC пов'язаномих з DOSом. Відомий всім максимум у 640 кБайт оперативної пам'яті - ще один із прикладів спадщини. Саме через те, що PC працює в стандартному режимі (standard mode) під час завантаження, виникає необхідність компресувати ядро Лінакса програмою `gzip` - ядро не поміщається повністю в перші 640кБайт пам'яті. І ще раз - це особливість тільки PC-Лінакса, ні інші Лінакси, ні інші Юнікси не мають потреби компресувати ядро.), а BIOS не знає про те, що робити з більше, ніж 1024-ма циліндрами. Інколи можливо користуватися стартовим розділом, який не весь лежить в межах 1024 циліндрів. Але це працює тільки до тих пір, поки всі файли, які читає BIOS, знаходяться у вказаних межах. І оскільки є практично неможливим керувати розташуванням файлів на диску, то уявляється *страшенно поганою ідеєю* користуватися таким розділом. Ви ніколи не зможете сказати напевне, чи завантажиться Ваш комп'ютер після поновлення ядра чи після дефрагментації диска. Тому будьте особливо обережні і визначайте Ваш стартовий розділ так, щоб він від початку до кінця лежав на перших 1024 циліндрах [31]ожливо в когось виникне питання, чому не виникає така проблема при роботі з DOS або з його похідними - Windows'ами. Справа не в кращій організації Windows, а навпаки - в примітивнішому підході до процесу старту системи. Якщо Ви коли-небудь в DOS'і користувалися командою `sys` (яка переносить файли операційної системи з стартового диску на інший диск, наприклад на дискету - робить цей диск системним або стартовим), то можливо помітили, що не у всіх випадках

[30]**Прим. перекл.:** З *(ДК)*
[31]**Прим. перекл.:** М *(ДК)*

таке перенесення можливе. Якщо дискета була тільки, що відформатована, то система перенесеться залюбки, а якщо Ви до цього встигли записати на неї хоча б один файл, то команда `sys` відмовиться встановлювати систему. Це відбувається тому, що DOS встановлює системні файли тільки в перших секторах диску і не може прочитати ядро системи (в DOS'і це два невидимих файли, що завжди лежать в корні диску C: - MSDOS.SYS або PCDOS.SYS залежно від виробника системи (Якщо ще є люди, які пам'ятають, що DOS виробляв колись не тільки Майкрософт, і не завжди в минулі роки DOS мав префікс «MS») та IO.SYS), якщо це ядро знаходиться в іншому місці диску.

Результатом такого підходу і є те, що: з одного боку система завжди автоматично знаходиться в межах 1024 перших циліндрів, а з іншого - несистемний диск можна зробити системним тільки наново відформатувавши його (що, можливо, не всім до смаку). Крім того, що ще гірше, систему неможливо поновити (тобто поновити її ядро) записавши замість старого ядра нове на диск. А мріяти про те, щоб мати на диску два ядра (йдеться не про два різних розділи з різними системами на них, а саме про один розділ з кількома ядрами, які можна стартувати за бажанням).

В будь-якоому Юніксі, ( Лінакс не є виключенням), можливо вказати яке саме ядро Ви хочете стартувати. Але оскільки Лінаксу дісталися в спадщину обмеження, існуючі в світі PC, доводиться трохи хитрувати, щоб їх обійти - і вимога мати ядро на перших циліндрах є саме такими хитрощами..

Деякі новіші версії BIOS'ів з IDE дисками, фактично, вміють поводитися з дисками, що мають більше ніж 1024 циліндри. Якщо Ви маєте таку систему, Ви можете забути про те, що тільки що прочитали, але якщо Ви не впевнені, то покладіть ядро в перші 1024 циліндри.[32]

Кожен розділ повинен мати парне число секторів, оскільки файлова система Лінакса користується блоком розміром 1 кБайт, тобто двома секторами. Непарна кількість секторів

[32]**Прим. перекл.:** Для цього багато і не треба. Далі буде йтися про те, що кореневу файлову систему завжди краще мати невелику - маючи на ній абсолютний мінімум директорій необхідних для завантаження системи. Мій особистий досвід свідчить, що самий песимістичний прогноз щодо кореневої файлової системи - це максимум 50МБайт (а в більшості випадків достатньо 15-20МБайт). Тобто, такий розділ достатньо малий і на будь-якому сучасному диску (із величезними циліндрами), помістити цей розділ першим достатньо для того, щоб бути певним, що він повністю лежить в потрібних межах. На старих же дисках цієї проблеми не існує взагалі, оскільки такі диски часто мають менше, нід 1024 циліндри. Ситуація трохи погіршується, якщо Вам треба грузити ще й Windows з цього диску, але це вже інше питання. *(ДК)*

не принесе Вам ніяких прикрощів крім того тільки, що останній сектор не буде використовуватися, і деякі з версій `fdisk` попередять Вас про це (ну а крім того - це просто непристойно).

Якщо Вам потрібно змінити розмір розділу диску, зробити практично це можна єдиним способом - зробивши резервну копію всього, що є на цьому розділі (або всього диску, що набагато надійніше), стерши цей розділ, створивши новий і поновивши на цьому розділі все з резервної копії. Якщо розмір розділу збільшується, можливо Вам потрібно буде також відрегулювати (із створенням резервних копій і відновленням) розміри сусідніх розділів також.

І оскільки такий процес досить болісний, бажано встановити вірні розміри розділів прямо з першого разу, або ж придбати апаратуру для швидкого і безболісного створення резервних копій і відновлення. Якщо Ви тільки встановлюєте систему з носіїв, які до того ж не вимагають частої зміни (як, наприклад, компакт-диски), Ви маєте можливість трохи побавитися з різноманітними конфіґураціями при цьому. Не маючи ніяких власних даних в новій системі, простіше змінювати розділи по кілька разів.

Для DOS'у існує програма `fips`, яка розповсюджується з Лінаксом, за допомогою якої можна змінювати розміри розділів DOS'у без резервування та відтворення даних, але для інших систем це все ще необхідно робити.

### 5.7.5 Спеціальні файли пристроїв та розділи диску

Кожен розділ і кожен розширений розділ має свій власний спеціальний файл пристрою. За домовленістю назви цих файлів утворюються приєднанням номера розділу диска після назви файлу самого диску. За домовленістю також перші чотири розділи (з номерами 1–4) є основними розділами, незалежно від їх кількості, а номери 5–8 зарезервовані за логічними розділами, незалежно від того, скільки мається основних розділів на диску і від того, на якому з основних розділів ці логічні розділи знаходяться. Наприклад, `/dev/hda1` - це перший основний розділ на першому IDE диску, а `/dev/sdb7` - третій розширений розділ на другому SCSI диску. Повний список файлів пристроїв наведений в [**?**].

## 5.8 Файлові системи

### 5.8.1 Що таке файлові системи?

**Файлова система** - це метод та структура даних, які використовуються операційною системою для збереження інформації про файли на диску чи розділі диску. Тобто, іншими словами, - це метод організації збереження файлів на диску. Термін часто також вживається як синонім до слів «диск» або «дисковий розділ», коли мова йде про файли розташовані на даному диску чи розділі. Отже, якщо Ви почуєте «Я маю дві файлові системи», знайте, що при цьому мається на увазі, два підрозділи, які служать для запису файлів, або у випадку з Лінаксом - це два підрозділи із файловими системами типу ext2 («розширена файлова система», як її називають в Лінаксі). Відмінності між диском чи розділом на диску та файловою системою, яка на ньому створена суттєві. Лише деякі програми (включаючи сюди ті, які створюють файлові системи) можуть працювати безпосередньо з секторами[33] на диску або його розділі. Якщо на цьому диску або розділі вже існує файлова система, вона може такими програмами бути знищена або серйозно пошкоджена. Більшість програм працюють, на відміну від згаданих кількох, з файловими системами і, отже, не зможуть працювати з розділами, які не мають файлової системи на них (або, що приблизно те ж саме, - мають файлову систему не того типу, що потрібен).

Перед тим, як використовувати диск або розділ як файлову систему, його треба ініціалізувати. При цьому на диск записується деяка інформація для підтримання файлової системи у відносному порядку. Цей процес відомий як **створення файлової системи**.

Більшість файлових систем Юніксів мають дуже схожу загальну структуру, але все-таки розходяться в подробицях. Основні поняття включають **суперблок**[34], **inode**1.3.1, **блок**

[33]**Прим. перекл.:** В англійській мові разом з іменником «сектор» в даному випадку вживається прикметник «raw»(raw disk, raw sector, raw partition), який дослівно мав би перекладатися як «сирий» або «неприготований». Ця термінологія відноситься, наприклад, до дискового підрозділу без створеної на ньому файлової системи, або просто до спеціального файлу дискового підрозділу чи всього диску. Як протилежність до нього використовується «cooked» - тобто розділ із файловою системою на ньому. (аналогічно - cooked device, cooked partition, тощо) - тобто «приготований» (зварений, підсмажений). Дослівний переклад українською після цього виглядав би досить «цікавим», але на мій погляд - це занадто. Тому я, сподіваючись на високий інтелектуальний рівень українського читача, просто опускаю в таких випадках прикметник. Хочу вірити, що читач сам може визначити, де йде мова про сире і де про варене. *(ДК)*

[34]**Прим. перекл.:** superblock *(ДК)*

**даних**[35], **блок директорії**[36] та **блок ссилок**1.3.1. Суперблок містить інформацію про файлову систему в цілому, таку, наприклад, як її розмір (детальна інформація, яка тут мала б знаходиться, залежить від конкретного типу файлової системи). В inode зберігається вся інформація про окремий файл, крім його назви [37]ожен файл має inode, але не кожен файл має свій *власний* inode. Власне, для кращого розміння варто підкреслити, що саме inode і є файлом, а та назва, що бачить користувач, є всього-навсього етикеткою, яка причеплена до inode. Якщо inode має кілька етикеток, то кажуть, що між файлами встановлені «жорсткі ссилки» - hard link. Тобто дві різні етикетки можуть вказувати на один і той же файл - для користувача це виглядає так, ніби то один файл має кілька назв.

Слід також відчувати різницю між жорсткими та м'якими (символічними) ссилками (soft link або symbolic link, symlink). Якщо жорстка ссилка це не що інше, як просто інша назва одного і того ж файлу, то символічна ссилка - це спеціальний вид файлу. Якщо подивитися на символічну ссилку, наприклад, командою `ls -l`, то помітно, що символічна ссилка - це файл дуже невеликого розміру, часто 10-14 байт. Що ж таке ці байти? Це - просто маршрут до оригіналу (як кажуть «батька») цієї символічної ссилки.

Так, наприклад, якщо символічна ссилка була створена такою командою, як `ln -s /usr/X11R6 /usr/X11` , то файл `/usr/X11` буде символічною ссилкою на `/usr/X11R6`, і його розмір буде рівно 10 байт - тобто кількість символів у `/usr/X11R6`, разом із '/'.

```
# ln -s /usr/X11R6 /usr/X11
#ls -l
#total 42
#lrwxrwxrwx 1 root root 10 Mar 31 08:22 X11 -> /usr/X11R6/
# [...]
```

Від звичайних файлів (в тому числі і від жорстких ссилок) символічні ссилки відрізняються першим символом в рядку, що вказує тип файлу - тобто в тому байті inode, де записується тип файлу, у символічної ссилки записано «l».

. Назва файлу записується в директорії разом з номером inode'у. inode зберігає номери кількох блоків, в яких записаний

[35] **Прим. перекл.:** data block *(ДК)*
[36] **Прим. перекл.:** directory block *(ДК)*
[37] **Прим. перекл.:** K *(ДК)*

сам файл. Але в inode'і достатньо місця тільки для кількох блоків даних, однак, якщо потрібно буде більше, для ссилок на наступні блоки динамічно виділяється більше місця. Ці блоки, що виділяються динамічно, є непрямими блоками. Тобто, сама їх назва вказує на те, що перш, ніж прочитати дані з блоку, потрібно знайти, де знаходиться сам блок, користуючись ссилкою на нього.

Як звичайно файлові системи Юніксів дозволяють мати файли з **«дірками»** (це робиться за допомогою команди `lseek`, краще дізнатися про яку можна з сторінки підказки). Дірка в файлі означає, що файлова система просто робить вигляд, нібито якесь певне місце в файлі має розмір нуль байтів, але жодного дискового простору при цьому не резервується на це в самому файлі (тобто файл насправді буде використовувати трохи менше дискового простору). Це трапляється особливо часто для невеликих двійкових файлів, бібліотек для спільного користування в Лінаксі, деяких баз даних та деяких спеціальних випадків. (Дірки створюються записуванням певної спеціальної величини в блоці ссилок або в inode'і. Ця спеціальна адреса означає, що для блоку даних не відводиться простору на диску, тобто в файлі - дірка.)

Дірки досить корисні. На системі автора просте вимірювання показало, що дірки зберігають близько 4 МБайт з 200 МБайт диску. Щоправда система має відносно мало програм і не має баз даних. Засіб для вимірювання дірок описаний в додатку А. [38]аявність дірок в файлах інколи може спричинити деякі неприємнощі. Тому треба пам'ятати про те в яких файлах дірки можуть траплятися і що треба робити, щоб ці дірки «обходити».

Так, наприклад, одним з найбільш розповсюдженіших типів файлів, в яких можуть траплятися дірки є файли баз даних dbm. Такі файли використовуються в базах даних мап NIS 1.3.1 або в базах даних користувачів, які використовуються в `sendmail` версій 8.x.

Через те, інколи виявляється, що сума розмірів всій файлів баз даних може бути більшою, ніж розмір дискового розділу, який містить такі файли. Якщо в мережі мається кілька серверів NIS, то вони найчастіше будуються за принципом - один головний сервер (master) і всі інші сервери-підлеглі (slave). При необхідності обновити бази даних на підлеглих серверах використовується команда `xfer`, яка фактично робить

[38]**Прим. перекл.:** Н *(ДК)*

копіювання файлів баз даних з головного сервера на підлеглі. Це ж саме може в принципі робити команда `rcp`, і якщо файли не мають дірок, то дія обох команд ідентична. Але команда `rcp` «не розуміє» дірок і у випадку файлів з дірками використання команди `rcp` приводить до того, що копія виявляється більшою ориґіналу і дисковий розділ підлеглого сервера може переповнитися в той час, як на головному сервері ще буде повно вільного місця.

### 5.8.2 Галоп по файлових системах

Лінакс підтримує кілька файлових систем, серед яких найважливішими є:

**minix** Найстаріша з усіх і вважається найбільш заслуговуючою на довіру. В той же час, вона досить обмежена за можливостями (відсутні деякі часові відбитки[39], довжина назви файлу обмежена 30-ма літерами , може мати максимум 64 МБайти на файлову систему).

**xia** Видозмінена версія файлової системи minix, яка розширила межі довжин назви файлу та розмір файлової системи, але не додала нових можливостей до системи. Це - не дуже популярна файлова система, але, як повідомлялося, працює досить гарно.

**ext2** Найбільш багата можливостями рідна файлова система Лінакса, на даний момент - найбільш вживана, була спроектована таким чином, щоб її легко можна було розширювати та доповнювати новими можливостями, тобто нові версії коду файлових систем не будуть вимагати перебудови існуючих файлових систем.

**ext** Старіша версія - попередниця `ext2`, яка була спроектована без думок про розширення. Її практично вже немає в нових системах, і ті хто мав її в старих, вже замінили її новою версією.

Додатково до цього, існує підтримка для деяких «іноземних» файлових систем, що робить простішим обмін файлами з іншими системами. Всі ці іноземні системи працюють практично так само, як і рідні файлові системи Лінакса, але можливо вони не мають тих чи інших можливостей, які присутні в Юніксах, або мають деякі смішні обмеження або ще що.

[39] **Прим. перекл.:** time stamps *(ДК)*

**msdos** Сумісність з DOS (а також з OS/2 та Windows NT) - файлові системи FAT.

**umsdos** Розширення файлової системи `msdos` для Лінакса - для вживання довгих назв файлів, з можливістю присвоєння власників файлам, дозволів, ссилок та файлів пристроїв. Це розширення дозволяє користуватися звичайною файловою системою `msdos`, так як нібито це - файлова система Лінакс, і таким чином відпадає необхідність користуватися спеціальною системою для Лінакса при наявності комп'ютера з встановленим DOS'ом.

**iso9660** Стандартна файлова система компакт-дисків, має популярне розширення «Рок Рідж», яке дозволяє вживати довгі назви файлів. Розширення Рок Рідж при необхідності підключається автоматично.

**nfs** Файлова система мережі [40] - файлова система, яка дозволяє спільно на багатьох комп'ютерах в мережі користуватися файлами і дозволяє простий доступ до них з будь-якого з комп'ютерів мережі.

**hpfs** Файлова система OS/2.

**sysv** Файлові системи SystemV/386, Coherent та Xenix.

[41]о цього можна додати деякі додаткові системи не згадані в ориґіналі.

**vfat** Файлова система застосовувана в Windows 95 - розширення файлової системи DOS, яке дозволяє користуватися довгими назвами файлів.

**hfs** «Ієрархічна файлова система» (hierarchical file system), яка застосовується в операційній системі MacOS на Макінтошах. Ця система може мати такі два застосування - для читання дискет записаних на Макінтоші, та (в поєднанні з реалізацією протоколу рідної Макінтошівської мережі AppleTalk) для побудови файл сервера для мережі Мас'ів (Лінакс дозволяє, наприклад, змонтувати Макінтошевський компакт-диск та експортувати його для всієї мережі Мас'ів).

**ufs** Файлова система, що застосовується в SunOS та Solaris. В термінолоґії SunOS - це файлова система типу «4.2», а в

[40]**Прим. перекл.:** networked file system - NFS *(ДК)*
[41]**Прим. перекл.:** Д *(ДК)*

Solaris'і вона ж носить назву «ufs». Дуже важлива файлова система особливо для Sparc Linux'а, оскільки дозволяє спільно користуватися файловими системами SunOS/Solaris і Sparc Лінакса.

Вибір файлової системи залежить від ситуації. Якщо Ви хочете мати доступ до одних і тих же файлів з Лінакса та з якоїсь іншої операційної системи (встановленої на цьому ж комп'ютері), то повинна використовуватися файлова система саме цієї операційної системи. Якщо ж Ви маєте вільний вибір, тоді найкращим вибором буде файлова система ext2, оскільки вона має всі необхідні функції і достатньо швидка.

Крім згаданих вище файлових систем в Лінаксі є також файлова система `proc`, змонтована в директорії `proc`, яка насправді не є файловою системою, хоча і дуже на неї схожа. Файлова система `proc` полегшує доступ структур даних всередині ядра і процесів (звідки і назва). Вона подає користувачеві ці структури у вигляді файлової системи, і до даних в такому вигляді можна застосовувати звичайні засоби для роботи з файлами. Наприклад, для того, щоб отримати список всіх процесів в системі можна скористуватися командою

```
$ ls -l /proc
total 0
dr-xr-xr-x   4 root     root            0 Jan 31 20:37 1
dr-xr-xr-x   4 liw      users           0 Jan 31 20:37 63
dr-xr-xr-x   4 liw      users           0 Jan 31 20:37 94
dr-xr-xr-x   4 liw      users           0 Jan 31 20:37 95
dr-xr-xr-x   4 root     users           0 Jan 31 20:37 98
dr-xr-xr-x   4 liw      users           0 Jan 31 20:37 99
-r--r--r--   1 root     root            0 Jan 31 20:37 devices
-r--r--r--   1 root     root            0 Jan 31 20:37 dma
-r--r--r--   1 root     root            0 Jan 31 20:37 filesystems
-r--r--r--   1 root     root            0 Jan 31 20:37 interrupts
-r--------   1 root     root      8654848 Jan 31 20:37 kcore
-r--r--r--   1 root     root            0 Jan 31 11:50 kmsg
-r--r--r--   1 root     root            0 Jan 31 20:37 ksyms
-r--r--r--   1 root     root            0 Jan 31 11:51 loadavg
-r--r--r--   1 root     root            0 Jan 31 20:37 meminfo
-r--r--r--   1 root     root            0 Jan 31 20:37 modules
dr-xr-xr-x   2 root     root            0 Jan 31 20:37 net
dr-xr-xr-x   4 root     root            0 Jan 31 20:37 self
-r--r--r--   1 root     root            0 Jan 31 20:37 stat
-r--r--r--   1 root     root            0 Jan 31 20:37 uptime
```

```
-r--r--r--   1 root     root            0 Jan 31 20:37 version
$
```

(Крім наведених тут буде ще кілька додаткових файлів, які не мають відповідних процесів. Приклад, наведений тут, трохи скорочений порівняно з реальною ситуацією.)

Хоча `/proc` і називається файловою системою, жодна її частина навіть і не торкається диску. Вона вся існує тільки в уяві ядра. Коли будь-хто намагається звернутися до будь-якої з частин файлової системи `proc`, ядро подає її так, ніби то вона справді існує десь. Отже, навіть якщо Ви власними очима бачите тут багато-мегабайтний файл `/proc/kcore`, він не займає жодного байту пам'яті ні на диску, ні в пам'яті (до того часу, поки Ви не почнете його копіювати куди-небудь).

### 5.8.3 Якою файловою системою користуватися?

В більшості випадків немає жодного сенсу в користуванні багатьма файловими системами одночасно. Зараз найбільш популярною є ext2fs, і, напевно тому, краще всього користуватися нею. В залежності від конкретних вимог до структури, швидкості, надійності, сумісності, тощо, слід розлядати також ті чи інші системи поза ext2fs. Але кінцеве конкретне рішення завжди залежить від конкретних умов.

### 5.8.4 Створення файлової системи

Файлові системи створюються (або «ініціалізуються») за допомогою команди `mkfs`. Насправді ж, для кожної файлової системи існує своя власна команда. Команда `mkfs` всього-навсього фасад до цілої групи команд, і для створення тої або іншої файлової системи, застосовується відповідна їй команда. Тип файлової системи, яку треба створити, задається за допомогою параметру `-t тип_файлової_системи` команді `mkfs`.

Програми, які викликаються командою `mkfs`, мають різні параметри. Загальні (і найбільш важливі) параметри наведені нижче, але конкретніші дані можна завжди відшукати в сторінці підказки по конкретній команді.

`-t` *тип_файлової_системи* Виберіть тип файлової системи.

`-c` Шукати збійні блоки та поновити їх список.

`-l` *назва_файлу* Прочитати попередній список збійних блоків з файлу *назва_файлу*.

Щоб створити на дискеті файлову систему типу ext2, потрібно виконати такі команди:

```
$ fdformat -n /dev/fd0H1440
Double-sided, 80 tracks, 18 sec/track. Total capacity 1440 kB.
Formatting ... done
$ badblocks /dev/fd0H1440 1440 > bad-blocks
$ mkfs -t ext2 -l bad-blocks /dev/fd0H1440
mke2fs 0.5a, 5-Apr-94 for EXT2 FS 0.5, 94/03/10
360 inodes, 1440 blocks
72 blocks (5.00%) reserved for the super user
First data block=1
Block size=1024 (log=0)
Fragment size=1024 (log=0)
1 block group
8192 blocks per group, 8192 fragments per group
360 inodes per group

Writing inode tables: done
Writing superblocks and filesystem accounting information: done
$
```

Спершу дискета форматується (параметр `-n` забороняє перевірку дискети на збійні блоки). Після цього збійні блоки шукаються командою `badblocks`, результати роботи якої відправляються в файл. І, нарешті, створюється файлова система, і при цьому використовується список збійних блоків, створений командою `badblocks`.

Замість команди `badblocks` можна скористуватися параметром `-c`, при цьому відпадає необхідність в додатковому файлі. Таке використання команди показано в наступному прикладі.

```
$ mkfs -t ext2 -c /dev/fd0H1440
mke2fs 0.5a, 5-Apr-94 for EXT2 FS 0.5, 94/03/10
360 inodes, 1440 blocks
72 blocks (5.00%) reserved for the super user
First data block=1
Block size=1024 (log=0)
Fragment size=1024 (log=0)
1 block group
8192 blocks per group, 8192 fragments per group
```

```
360 inodes per group

Checking for bad blocks (read-only test): done
Writing inode tables: done
Writing superblocks and filesystem accounting information: done
$
```

Параметром `-c` зручніше користуватися, ніж командою `badblocks` і додактковим файлом, але для перевірки файлової системи після її створення краще все-таки вживати `badblocks`.

Процес створення файлової системи на жорсткому диску відрізняється від процесу для дискет тільки тим, що при відпадає крок форматування.

### 5.8.5 Монтування та розмонтування

Перед тим, як користуватися файловою системою, її треба **змонтувати**. Після монтування, для того, щоб підтримувати в робочому порядку, система переймає на себе обов'язки по контролю файлової системи. Оскільки всі файли в Юніксі розташовані в єдиному дереві, після монтування виявляється, що все, що знаходилось до монтування в окремій файловій системі, стає складовою директорією в цьому дереві.

Наприклад, рисунок 5.3 показує три окремі файлові системи, кожна зі своїм власним коренем (кореневою директорією). Після того, як дві останні файлові системи змонтовані в точках `/home` та `/usr` першої файлової системи, утворюється єдине дерево директорій показане на рисунку 5.4.

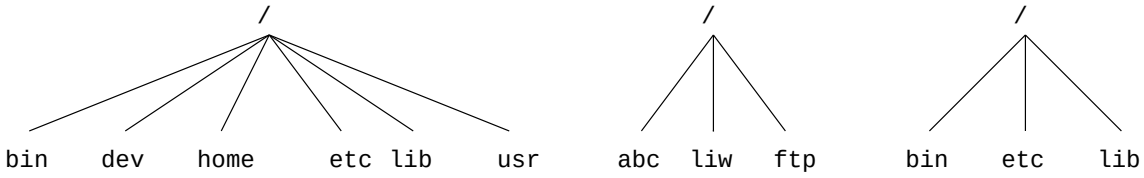


Рис. 5.3: Три окремі файлові системи.

Операції монтування виконуються наступними командами:

```
$ mount /dev/hda2 /home
$ mount /dev/hda3 /usr
$
$
```

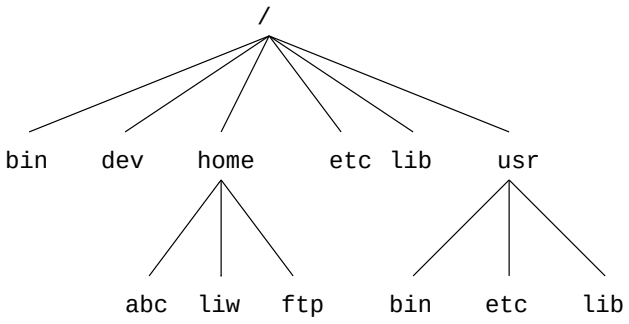


Рис. 5.4: `/home` та `/usr` змонтовано.

Команді `mount` потрібно давати два арґументи. Перший з них - це спеціальний файл пристрою, який відповідає диску чи підрозділу з файловою системою. Другий - директорія, в якій потрібно змонтвуати дану файлову систему. Після того, як виконано ці дві команди, все, що знаходиться в змонтованих фалових системах, буде виглядати як склад директорій `/home` та `/usr`. Кажуть, що «`/dev/hda2` **змонтовано** на `/home`», і, аналоґічно для `/usr`. Щоб подивитися, що знаходиться в тій, чи іншій файловій системі (після її монтування), слід дивитися, що знаходиться в тій директорії, на якій ця файлова система змонтована, так, як ніби-то це - звичайна директорія. Відмітьте різницю між спеціальним файлом пристрою (`/dev/hda2` та директорією, на яку його змонтовано (`/home`. Перший з них дає доступ до «сирого» диску (або розділу), в той час, як друга - описує доступ до «готової» директорії на диску. Директорію, в якій проводиться монтування файлової системи, називають **точкою монтування**.

Лінаксом, як уже відмічалося, підтримуються файлові системи багатьох типів. Команда `mount` намагається вгадати тип файлової системи, яку вона монтує. Але крім цього можна також користуватися параметром `-t` *тип_файлової_системи*, щоб явно вказати тип. Інколи це необхідно, бо буває, що шлях здогадок підводить. Наприклад, щоб змонтувати файлову систему DOS на дискеті, потрібно виконати команду:

```
$ mount -t msdos /dev/fd0 /floppy
$
```

Від директорії, на вершині якої монтується файлова система, не вимагається, щоб вона була пустою, але необхідно, щоб вона існувала. Однак, всі файли, які існували в цій директорії до монтування будуть недоступними після того, як

файлова система змонтована. (Будь які файли, відкриті на час монтування будуть залишатися доступними, так само, як і файли в даній директорії, які мають жорсткі ссилки на себе з інших частин файлової системи, можуть бути доступними з використанням цих інших імен.) Цим не завдається жодної шкоди, і інколи такий ефект можна використати на користь. Наприклад, уявімо собі, що `/tmp` та `/var/tmp` є синонімами, та `/tmp` є символічною ссилкою на `/var/tmp`. Під час старту системи, коли файлова система `/var` ще не змонтована, для тимчасових файлів використовується директорія `/var/tmp`, яка знаходиться на кореневій файловій системі. Після того, як `/var` змонтована, директорія `/var/tmp` в кореневій директорії робиться недоступною, і замість неї використовується змонтована на її вершині файлова система. Якби директорія `/var/tmp` не існувала, то було б неможливо користуватися тимчасовими файлами до того, як змонтована `/var`.

Якщо Ви не збираєтесь записувати нічого в файлову систему, яку монтуєте, Ви можете скористуватися параметром `-r` для того, щоб змонтувати цю файлову систему в режимі **тільки читання**[42]. Отримавши цей параметр, ядро зупинить будь-яку спробу записати що-небуть в файлову систему і ядро не буде змінювати час доступу[43] до inode'ів файлів в даній файловій системі. Монтування в режимі «тільки читання» необхідне для носіїв, на які запис неможливий, таких, наприклад, як компакт-диски[44]ісля роботи з Лінаксом спроба змонтувати компакт в SunOS або в Solaris'і може спочатку збити системного адміністратора з пантелику. В Лінаксі, якщо Ви намагаєтеся змонтувати компакт-диск, не вказавши параметра `-r`, то `mount` тільки видасть попереження, але диск все-таки змонтує. Але в SunOS чи Solaris'і Ви по-перше, отримаєте попередження, а по-друге, повинні будете повторити монтування все-таки з використанням цього параметра, бо без нього файлова система не змонтується. Додайте до цього ще те, що назви одних і тих-же файлових систем відрізняються в різних системах (`iso9660` в Лінаксі і `hsfs` в SunOS/Solaris), і зрозумієте насолодження від користування кількома Юніксами відразу..

Уважний читач мабуть уже відмітив одну головоломку, а саме: як же монтується «сама перша» файлова система - та,

[42]**Прим. перекл.:** readonly *(ДК)*
[43]**Прим. перекл.:** file access time *(ДК)*
[44]**Прим. перекл.:** П *(ДК)*

яка називається **кореневою файловою системою**[45], бо вона містить кореневу директорію. Звичайно ж, із зрозумілих причин, цю файлову систему не можна змонтувати на вершині іншої. Відповідь в цьому випадку дуже проста - це звичайно ж фокус.[46][47]. Коренева файлова система чарівним способом виявляється змонтованою при запуску системи, і можна завжди розраховувати на те, що вона змонтована - якщо неможливо змонтувати кореневу файлову систему, система просто не завантажиться. Назва тої файлової системи, яка магічно монтується як коренева, або компілюється безпосередньо в саме ядро, або встановлюється за допомогою LILO чи `rdev`[48].

При старті системи коренева система завжди монтується спочатку в режимі «тільки читання». Пізніше із стартових скриптів запускається `fsck` для перевірки цілісності файлової системи, і потім файлова система **перемотовується**[49] в режимі дозволу запису. Загалом команда `fsck` не повинна виконуватися на змонтованих файлових системах, оскільки будь-які зміни до файлової системи, над якою вона виконується, *викличуть* великі проблеми. Але, оскільки коренева система в даному випадку змонтована з доступом тільки на читання, `fsck` може коригувати потрібні ділянки без проблем, і операція перемонтування запише всі потрібні зміни з пам'яті на диск.

В багатьох системах при старті потрібно монтувати також інші файлові системи (крім кореневої та своп). Всі вони описуються в файлі `/etc/fstab`[50]бо `/etc/vfstab` у випадку із Solaris'ом. Подробиці про формат файлу є в сторінці підказки *fstab*. Те, як конкретно додаткові файлові системи монтуються

[45] **Прим. перекл.:** root filesystem *(ДК)*

[46] Про це можна довідатися дослідивши вихідні тексти ядра або Посібник по програмуванню ядра

[47] **Прим. перекл.:** Kernel Hackers' Guide *(ДК)*

[48] **Прим. перекл.:** Лінакс у відношенні встановлення кореневої файлової системи є набагато гнучкішим, ніж такі системи, як SunOS та Solaris. У відношенні монтування кореня SunOS стоїть на найнижчому рівні, тобто його гнучкість практично рівна нулю. Корінь завжди має бути на першому підрозділі диску - це скомпільовано прямо в ядро. Якщо ядро не може знайти корінь на першому розділі, процес старту спиниться і система буде питати про альтернативний корінь (те ж стосується і своп-розділу на диску, який завжди має бути другим розділом - теж «закомпільовано» в ядро. На відміну від Лінакса, SunOS не може працювати без своп-простору і не знайшовши потрібного розділу, просто не завантажиться). Тобто в результаті, запитавши про корінь, система завантажиться все-таки в сякий-такий робочий стан, але сама перегрузитися вже не зможе. В Solaris'і можна вказати розділ кореневої системи, зробивши запис в файлі `/etc/system`, але вказати альтернативний корінь під час старту системи можливо тільки, якщо Ви маєте альтернативний файл `/etc/system`. *(ДК)*

[49] **Прим. перекл.:** re-mount *(ДК)*

[50] **Прим. перекл.:** a *(ДК)*

залежить від багатьох факторів, і це може конфіґурувати кожен окремий системний адміністратор так, як це потрібно. Коли розділ про процес старту системи буде закінчено, в ньому можна буде це прочитати.

Якщо файлова система більше не потрібна, її можна розмонтувати командою `umount`[51]. Для `umount` потрібен один параметр - або точка монтування файлової системи, або її спеціальний файл. Наприклад, щоб розмотувати директорії, змонтовані в попередньому прикладі, потрібно виконати команди:

```
$ umount /dev/hda2
$ umount /usr
$ $
```

Подальші відомості про команду дивіться в сторінці підказки по ній. Не забудьте, що завжди потрібно розмонтовувати змонтовану дискету. *Не витягуйте просто так дискету з дисководу!* Через кешування диску в пам'яті, дані можуть справді записуватися на диск багато пізніше і не обов'язково записані на диск, якщо його ще не розмонтовано [52]. Отже, якщо Ви витягнете дискету зарано - отримаєте сміття замість даних. Якщо Ви тільки читали з дискети, то, можливо нічого страшного не трапиться, але, якщо записали щось (навіть випадково), результатом може бути катастрофа.

Для монтування та розмонтовування файлових систем потрібно мати привілеї супер-користувача, тобто тільки користувач `root` може це робити. Причиною цього є те, що, якщо будь-хто може монтувати чи розмонтовувати, скажімо дискету на будь-якій директорії, то, в цьому випадку було б дуже просто створити, наприклад, Троянського коня замаскованого під `/bin/sh`, або іншу часто вживану програму. Однак, звичайним користувачам інколи треба монтувати дискети, і для цього існує кілька способів:

- Дати користувачеві пароль `root`'а. Звичайно ж це найгірше рішення з точки зору безпеки, але і найпростіше. Воно спрацьовує в тих випадках, коли немає необхідності турбуватися про безпеку, тобто годиться на багатьох персональних системах не підключених до мережі.

[51] Звичайно ж це повинна була б бути команда `unmount`, але літера `n` містично зникла десь в середині 70-х і вже більше не з'являлася. Будь-ласка, поверніть її в Bell Labs, NJ, якщо Ви її коли-небудь знайдете.

[52] **Прим. перекл.:** Операція розмонтування гарантує, що дані записуються з кешу на фізичні носії. *(ДК)*

- Скористуватися програмою типу `sudo` щоб дозволити користувачам користування програмою `mount`. Це все ще не ідеальне рішення з точки зору безпеки, але все-таки не дає користувачам безпосереднього доступу до паролю `root`'а[53].
- Рекомендувати користувачам вживати пакет програм `mtools`, для оперування файлами DOS без монтування файлових систем. Цей пакет - дуже зручна штука, якщо все, що потрібно, це копіювати файли з DOSівських дискет і назад, але страшенно незручний в усіх інших випадках.
- Перечислити пристрої дисководів разом з відповідними параметрами для монтування в файлі `/etc/fstab`.

Для реалізації останнього варіанту треба додати до `/etc/fstab` приблизно такий рядок:

```
/dev/fd0 /floppy msdos user,noauto 0 0
```

Стовпчики в рядку такі: пристрій, який треба змонтувати, директорія, в якій потрібно монтувати файлову систему, тип файлової системи, параметри, частота створення резервних копій (використовується командою `dump`), та номер проходу для `fsck` (Цей параметр служить для того, щоб встановити порядок, в якому команда `fsck` повинна перевіряти файлові ситеми при старті системи. 0 означає, що перевірка не потрібна).

Параметр `noauto` запобігає монтуванню даної файлової системи при старті системи (тобто команда `mount -a` не змонтує цю файлову систему). Параметр `user` дозволяє монтувати цю файлову систему будь-якому користувачеві, і, (з міркувань безпеки) забороняє виконання будь-яких програм (як нормальних, так і з встановленим setuid) та використання будь-яких спеціальних файлів пристроїв із змонтованої таким чином файлової системи.[54]ідмінності SunOS/Solaris в даному випадку полягають в тому, що параметр `user` відсутній в обох цих системах. Тобто монтування та розмонтування файлових систем дозволено *тільки* системному адміністратору - `root`'у. Але крім описаних способів є ще деякі, на яких ми зупинимося трохи далі по тексту, і як виявляється не все ще втрачено, навіть, якщо Ви адмініструєте SunOS/Solaris. Якщо такий рядок включено в файл `/etc/fstab`, то будь-який користувач може змонтувати дискету виконавши команду:

[53]Вимагає подумати кілька секунд.
[54]**Прим. перекл.:** В *(ДК)*

```
$ mount /floppy
$
```

Дискету можна (і, звичайно ж, потрібно) розмонтовувати відповідною командою `umount`.

[55] деякі нюанси при монтуванні файлової системи, вказаної в `/etc/fstab`. Саме головне при цьому - команда `mount` *повинна* мати тільки один параметр - або ім'я спеціального файлу пристрою, або точку монтування файлової системи. Тільки при цій умові `mount` загляне в `/etc/fstab` для того, щоб знайти там інші необхідні параметри. Якщо при монтуванні Ви скористаєтесь стандартною формою команди `mount спец_файл точка_монтування`, то команда не буде ссилатися на `/etc/fstab`, а намагатиметься змонтувати файлову систему безпосередньо, і, оскільки Ви - не `root`, видасть помилку:

```
$ mount : only root can do that
$
```

Якщо Вам потрібно забезпечити можливість монтування кількох типів дискет, Ви повинні забезпечити також і відповідні точки монтування для всіх них і відповідні рядки для кожного типу в `/etc/fstab`. Параметри можуть бути різними для них всіх. Наприклад, щоб дати доступ до обох файлових систем - MS-DOS та ext2 на дискетах, Вам потрібно мати такі рядки в `/etc/fstab`:

```
/dev/fd0    /dosfloppy    msdos   user,noauto  0  0
/dev/fd0    /ext2floppy   ext2    user,noauto  0  0
```

Для файлових систем MS-DOS (не тільки дискети, а взагалі - всі з них), було б бажано обмежети доступ використанням `uid` чи `gid` та параметром `umask`, які в подробицях описані в сторінці підказки *mount*. Якщо Ви не надто безпечні, то монтування файлової системи MS-DOS дає (як мінімум) доступ на читання для будь-кого, що взагалі-то не бажано.

### 5.8.6 Перевірка цілісності файлових систем за допомогою `fsck`

Файлові системи - дуже складні створіння, і тому, в деякому сенсі, вони схильні до помилок. Цілісність файлової системи чи наявність в ній помилок можна перевірити за допомогою команди `fsck`. Команді можна вказати, що вона повинна

[55] **Прим. перекл.:** Є *(ДК)*

виправляти всі незначні помилки, які вона відшукає, і попереджати користувача, якщо трапляються такі з них, які не можна виправити. На щастя, бібліотеки вживані для файлових систем вже відлагоджені досить добре, і проблеми з ними трапляються досить рідко (або їх взагалі не буває). Збої в файлових системах найчастіше трапляються через перебої в електропостачанні, збої в «залізі» чи через помилки операторів, як, наприклад, не вимкнена за правилами система.

Більшість систем виконують `fsck` автоматино при старті системи, так, що більшість помилок знаходяться (і, дай Боже, виправляються) до того, як система починає використовуватися. Використання зіпсованої файлової системи приводить до того, що погане стає ще гіршим: якщо структури даних файлової системи зіпсовані, використання цих структур може тільки зробити їх ще гіршими, що приведе до ще більших втрат даних. Але, з іншого боку, повна перевірка файлової системи на великих файлових системах за допомогою `fsck` може займати досить довгий час. І, через те, що помилки практично ніколи не трапляються при корректному вимкненні системи, для того, щоб не затягувати час старту системи, в Лінаксі вдаються до деяких хитрощів. Перший трюк: якщо існує файл `/etc/fastboot`, то перевірка файлових систем не робиться. Другий трюк: файлова система ext2 має спеціальний маркер, який вказує на те, чи була ця файлова система відмонтована вірно при попередньому монтуванні. Знаючи, що файлова система була розмонтована «чистою» (якщо прапорець вказує на це) `e2fsck` (версія `fsck` спеціалізована для перевірки файлової системи ext2), може не перевіряти цю файлову систему. При цьому, звичайно, робиться припущення, що чисте розмонтування не приносить системі проблем. Чи працює перший з трюків (з файлом `/etc/fastboot`) на Вашій системі залежить від стартових скриптів системи, але трюк з прапорцем монтування ext2 спрацьовує кожного разу, коли Ви користуєтесь `e2fsck`. Щоб змусити команду ігнорувати цей прапорець, це потрібно явно вказати за допомого параметра. (Деталі дивіться в сторінці підказки *e2fsck*).

Автоматична перевірка спрацьовує тільки для тих систем, які монтуються автоматично при старті системи. Для перевірки інших файлових систем, як, наприклад, дискет, користуйтесь `fsck`, запускаючи її вручну. [56]оманда `mount` в Лінаксі має певний інтелект. Якщо Ви вручну монтуєте файлову систему, яка не

[56]**Прим. перекл.:** К *(ДК)*

була перевірена, команда видасть попередження про це і скаже, що рекомендується скористатися `e2fsck` для перевірки.

Якщо `fsck` при перевірці натрапляє на файлову систему, яку вона не може відремонтувати, тр Вас може спасти тільки одне з двох: або глибокі пізнання про будову файлових систем взагалі і даного типу конкретно, або гарна резервна копія. Друге - зробити просто, а першого Ви можете досягти за допомогою друзів, підписної групи новин та підписного листа по Лінаксу чи задопомогою інших засобів знаходження інформації. Я хотів би розповісти про це більше, але білі плями в освіті та досвіді не дають мені цього зробити. Програма Теодора Цо [57] `debugfs` може Вам допомогти при цьому.

Користуйтеся `fsck` тільки на розмонтованих файлових системах, ніколи на змонтованих (за виключенням кореневої файлової системи при старті, яка монтується при цьому тільки на читання). Через те, що `fsck` працює з «сирими» розділами диску, вона може змінювати файлові системи так, що система про це навіть не здогадається. А якщо операційну систему дурити, то неприємності *трапляться*.

### 5.8.7 Перевірка зіпсованих блоків за допомогою `badblocks`

Перевірку збійних блоків варто виконувати періодично. Це можна робити командою `badblocks`. Вона видає список знайдених збійних блоків. Цей список потім можна передати `fsck`, щоб вона записала ці дані в структури даних файлової системи з тим, щоб операційна система могла б ними потім скористатися щоб обходити збійні блоки при записі даних. Наступний приклад показує як це зробити.

```
$ badblocks /dev/fd0H1440 1440 > bad-blocks
$ fsck -t ext2 -l bad-blocks /dev/fd0H1440
Parallelizing fsck version 0.5a (5-Apr-94)
e2fsck 0.5a, 5-Apr-94 for EXT2 FS 0.5, 94/03/10
Pass 1: Checking inodes, blocks, and sizes
Pass 2: Checking directory structure
Pass 3: Checking directory connectivity
Pass 4: Check reference counts.
Pass 5: Checking group summary information.

/dev/fd0H1440: ***** FILE SYSTEM WAS MODIFIED *****
```

[57] **Прим. перекл.:** Theodore T'so *(ДК)*

```
/dev/fd0H1440: 11/360 files, 63/1440 blocks
$
```

Якщо `badblocks` повідомляє про зіпсований блок, який вже використано під файл, `e2fsck` спробує перемістити цей блок в інше місце. Якщо блок було пошкоджено серйозно, може виявитися, що файл, зіпсовано також.

### 5.8.8 Боротьба з фраґментацією

Не завжди можливо записати файл на диск як неперервну послідовність блоків. Про файл, який записано з розривами («не нерозривна» послідовність блоків) кажуть, що він **фраґментований**. На прочитання фраґментованого файлу потрібно більше часу оскільки читаюча/записуюча головка повинна зробити для цього більше переміщень. Тому краще було б позбутися фраґментації, хоча в системах, які мають добрий буфер з «читанням наперед» це і невелика проблема.

Файлова система ext2 намагається утримувати фраґментацію на мінімумі, розташовуючи всі блоки файлу разом, навіть якщо їх не можна записати в послідовних секторах. Ext2 ефективно розташовує вільні блоки, які знаходяться по сусідству з іншими блоками файлу. Тобто для ext2 рідко коли буває необхідним турбуватися про фраґментацію. Існує програма для дефраґментації ext2, дивіться [**?**].

Існує багато програм дефраґментації для MS-DOS, які переписують блоки туди-сюди, щоб позбутися фраґментації файлів. Для інших систем дефраґментації можна досягти переписавши файлову систему цілком на резервні носії і відновивши її знову. Створення резервної копії перед дефраґментацією взагалі непагана ідея для будь-якої такої програми, оскільки багато чого може трапитися під час роботи програми.

### 5.8.9 Інші програми для всіх файлових систем

Існують інші засоби придатні на щось корисне при роботі з файловими системами. `df` показує скільки вільного дискового простору залишилося на файловій системі (системах). `du` підраховує скільки місця на диску займає та чи інша директорія та всі її файли. Обидві програми можна використовувати в полюванні за місцем на диску.

`sync` записує всі не записані досі блоки, які знаходяться в буфері кешу, на диск (див. розділ 6.6). Дуже рідко трапляються ситуації, коли це потрібно робити вручну - процес-демон `update` виконує це автоматично. Це може знадобитись в катастрофічних ситуаціях, коли `update` або його процес-помічник `bdflush` вмирає, або Вам потрібно вимнути напругу, і немає часу чекати, поки `update` зробить свою справу.

### 5.8.10 Інші засоби для файлової системи ext2

Додатково до програм, які створюють (`mke2fs`) та перевіряють цілісність (`e2fsck`) файлової системи, і якими можна користуватися або безпосередньо, або ж через їх незалежну від типу файлової системи «фасадну» програму, існують деякі інші додакткові засоби для ext2.

`tune2fs` служить для підгонки параметрів файлової системи. Деякі з найбільш цікавих параметрів такі:

- Найбільша допустима кількість монтувань. Якщо файлова система була змонтована і розмонтована без перевірки багато разів, `e2fsck` насильно перевіряє файлову систему, навіть якщо вона була розмонтована чисто. Для систем, що використовуються для розробки чи для випробовування, корисною порадою було б цей номер зменшити.

- Максимальний час між перевірками. `e2fsck` також може примусово перевіряти файлову систему, якщо час між перевірками досягнуто, навіть якщо прапорець чистоти встановлено, і система монтується не дуже часто. Однак, цей параметр можна відмінити.

- Кількість блоків зарезервованих для `root`'а. Ext2 резервує певну частину блоків для використання `root`'ом. Таким чином, навіть, якщо файлова система переповнюється, для системного адміністратора доступний деякий простір, і ще можна дещо зробити не стираючи файлів. Як звично, такий резервний простір встановлюється на рівні 5%, що для більшості дисків є припустимою величиною. Для дискет, однак, не має ніякого сенсу резервування жодного блока.

Сторінка підказки по *tune2fs* надасть Вам більше інформації по програмі.

```
dumpe2fs 0.5b, 11-Mar-95 for EXT2 FS 0.5a, 94/10/23
Filesystem magic number:  0xEF53
Filesystem state:         clean
Errors behavior:          Continue
Inode count:              360
Block count:              1440
Reserved block count:     72
Free blocks:              1133
Free inodes:              326
First block:              1
Block size:               1024
Fragment size:            1024
Blocks per group:         8192
Fragments per group:      8192
Inodes per group:         360
Last mount time:          Tue Aug  8 01:52:52 1995
Last write time:          Tue Aug  8 01:53:28 1995
Mount count:              3
Maximum mount count:      20
Last checked:             Tue Aug  8 01:06:31 1995
Check interval:           0
Reserved blocks uid:      0 (user root)
Reserved blocks gid:      0 (group root)

Group 0:
  Block bitmap at 3, Inode bitmap at 4, Inode table at 5
  1133 free blocks, 326 free inodes, 2 directories
  Free blocks: 307-1439
  Free inodes: 35-360
```

Рис. 5.5: Зразок роботи `dumpe2fs`

`dumpe2fs` показує інформацію про файлову систему ext2, в основному інформацію з супер-блока. Зразок того, що видає ця програма показано на рис. **??**. Деякі дані, що видаються є досить технічними і потребують розуміння того, як працює файлова система (див. додаток **??**, але більшість зрозуміла навіть системним адміністраторам.

`debugfs` є відладчиком файлової системи. Він дозволяє прямий доступ до структур даних, що записуються на диск, тобто його можна використовувати для відновлення тих дисків, які зіпсовані настільки, що `fsck` відновити їх вже не може. Про цю програму також просочувалися дані, що `debugfs` можна використовувати також для відновлення стертих файлів. Однак, робота з цією програмою вимагає дуже глибоких знань того, що Ви хочете зробити. Недостатні знання можуть знищити всі Ваші дані.

`dump` та `restore` служать для створення резервних копій

файлової системи ext2. Вони належать до традиційних засобів Юнікса по створенню резерних копій, але є специфічними для файлової системи ext2. Детальніша інформація про це є в розділі 11.

## 5.9 Диски без файлових систем

Не всі диски чи підрозділи використовуються як файлові системи. Підрозділ, що містить простір для свопінґу, наприклад, не має файлової системи. Більшість дискет використовуються подібно до стрічок, так, що `tar` чи інші подібні програми записують дані безпосередньо на диск без файлової системи. Завантажувальні дискети Лінакса не мають файлової системи, а містять тільки ядро системи.

Відмова від створення файлової системи має ті переваги, що дисковий простір використовуєься більш повно — без зайвої втрати його на облікові функції файлової системи. Крім цього такі диски є більш сумісними з іншими системами. Наприклад, для всіх систем формат `tar` є одним і тим же, не дивлячись на те, що файлові системи відрізняються від однієї системи до іншої. Ви швидко звикнете користуватися дисками без файлових систем в разі необхідності. Завантажувальні диски Лінакса теж часто не мають файлової системи, хоча можуть і мати.

Ще однією причиною для користування «сирими» дисками є необхідність зробити копії «образу» або зображення з них. Наприклад, якщо на диску знаходиться файлова система, яка тільки частково пошкоджена, варто б зробити точну копію файлової системи, перш, ніж намагатися її відновити, оскільки в разі неудачі Ви завжди зможете почати спочатку. Один із способів це зробити - за допомогою команди `dd`:

```
$ dd if=/dev/fd0H1440 of=floppy-image
2880+0 records in
2880+0 records out
$ dd if=floppy-image of=/dev/fd0H1440
2880+0 records in
2880+0 records out
$ $
```

Перша команда `dd` створює точний образ дискети в файлі `floppy-image`, а друга з них записує це зображення на дискету (Сподіваємось, що користувач здогадається замінити дискету в

дисководі між двома цими командами. Інакше ця пара команд має мало сенсу.).

## 5.10 Виділення дискового простору

### 5.10.1 Схеми розділу дисків

Дуже нелегко розбити диск на розділи найкращим чином. І що ще гірше - не існує ніякої універсальної поради як це робити. Занадто багато факторів впливають на кінцевий результат.

Традиційно вважається доцільним створення (відносно) невеликої кореневої файлової системи, на якій будуть зберігатися `/bin`, `/etc`, `/dev`, `/lib`, `/tmp` та інші необхідні для початку роботи системи речі. Таким чином коренева файлова система (на окремому розділі, або ж на своєму власному диску) - це все, що потрібно для того, щоб привести систему в робочий стан. Пояснення до цього може бути таке – якщо коренева файлова система невеличка і не дуже активно використовується, то вона має менше шансів бути зіпсованою під час краху системи. Тобто, її легше можна відновити після краху. Після цього Ви можете створити окремі розділи диску (чи окремі диски) для дерева директорій, що йде під `/usr`, для домашніх директорій користувачів (найчастіше під `/home`) та для простору свопінґу. Відділення домашніх директорій користувачів в свій окремий розділ має ті переваги, що створення резервних копій в цьому разі стає простішим, оскільки не має сенсу дуже часто архівувати програми, що знаходяться під `/usr` [58]ід себе можу додати, що це також зменшує проблеми з системою. Якщо система не користується квотами (рідко хто має бажання псувати відносини з користувачами), то рано чи пізно, не залежно від розміру диску, виділеного під домашні директорії, він переповниться. І навіть якщо Ви й змушені будете розсилати користувачам повідомлення: «Панове, чи не будете Ви настільки ласкавими зтерти непотрібні файли в Ваших домашніх директоріях», або «Панове: `$(cd /home; du -s * | sort -nr | head -10 | awk ' { print $2}') | xargs rm -rf»!` (Якщо не зрозуміло, що це означає, не варто поспішати друкувати команду в командному рядку, щоб подивитися, що вона робить. Спробуйте спочатку розібратися.) Але при всьому цьому система залишається в робочому стані, і не залежно

[58] **Прим. перекл.:** В *(ДК)*

від того наскільки довго користувачі будуть борсатися в файловій системи заповненій на 99-100%, Ви зможете зловтішно усміхаючись, встановлювати нові програми під `/usr/local/games`.

Крім того, якщо комп'ютери об'єднані в мережу, можна також використовувати одну і ту ж директорію `/usr` із сервера спільно для багатьох комп'ютерів (наприклад, користуючись NFS). Таким чином зменшується загальний дисковий простір, необхідний для всієї системи (економія може становити десятки або сотні мегабайт помножені на число машин в мережі).

Проблеми, які виникають коли Ви маєте багато розділів, полягають в основному в тому, що вільний простір на диску виявляється розділеним на багато невеликих шматків розподілених по всіх розділах. В наш час, коли диски та операційні системи стають (як ми на це сподіваємось) більш надійними, багато хто віддасть перевагу одному єдиному розділу, на якому зберігаються всі файли. З іншого боку, створення резервної копії та відновлення невеликого розділу буде менш болючим.

Для невеликого диску (якщо Ви не займаєтесь програмуванням ядра системи), найкраще мати один єдиний розділ. Для великих дисків, можливо краще мати кілька великих розділів, на той випадок, якщо щось буде не так, як хотілося б. (Слід відзначити, що ми вживаємо 'малі' і 'великі' у дуже відносному смислі - чим більші Ви маєте диски, тим більшим буде 'велике').

Якщо Ви маєте кілька дисків, Ви можливо захочете створити кореневу систему (разом з `/usr`) на одному диску, а домашні директорії помістити на іншому.

Будьте готові до деякого експериментування з різноманітними схемами поділу дисків на розділи (з часом, не тільки під час встановлення системи). Це - добрий шматок роботи, оскільки вимагає встановлення системи від самого початку кілька разів, але, це єдиний надійний спосіб впевнитися, що Ви робите все вірно[59].

[59]**Прим. перекл.:** Дозволю собі знову невелику сваволю - не погоджусь з автором щодо перевстановлення системи багато разів. Юнікс на те і є Юнікс, що не вимагає в багатьох випадках того, чого вимагають інші системи - перегрузок та перевстановлень системи. Як тільки система приведена в більш-менш робочий стан, тобто встановлена конфіґурація, створені користувачі системи, тощо, немає ні найменшого сенсу її знищувати. Навіть якщо Ви помилилися з розподілом дискового простору з першого разу (або здобули в процесі роботи необхідний досвід для того, щоб зробити цей розподіл кращим, Вам достатньо мати такий-сякий вільний диск (або добрий, надійний засіб для резервної копії - стрічка чи ще що-небудь), щоб перенести на нього потрібні розділи диску та повернути їх на первісний диск після його перерозподілу. Пам'ятайте, що в Юніксі (а

### 5.10.2 Вимоги до дискового простору

Документація до тих комплектів Лінакса, які Ви будете встановлювати, дасть Вам деяке поняття про те, скільки дискового простору Вам необхідно для встановлення системи. Додайте до цього необхідне місце на диску для програм, що їх потрібно встановлювати самостійно. Ці дані допоможуть Вам спланувати використання дискової пам'яті, але крім цього необхідно приготуватися до майбутнього розширення системи і зарезервувати простір під нього.

Конкретні дані залежать від того, що саме збираються робити Ваші користувачі. Звичайно ж більшість людей хочуть мати стільки дискового простору, скільки це тільки можливо, але величина простору, маючи яку вони зможуть жити спокійно залежить від багатьох чинників. Деякі люди займаються друкуванням текстів і можуть легко обійтися кількома меґабайтами, але інші займаються обробкою величезних зображень і потребують для своєї роботи ґіґабайтів.

До речі, порівнюючи розміри дисків в кілобайтах, меґабайтах та ґіґабайтах, зважайте на те, що ці величини можуть бути різними. Деякі виробники дисків хочуть вважати, що кілобайт - це 1000 байт і що меґабайт - 1000 кілобайт, хоча весь комп'ютерний світ в обох випадках користується множником 1024. Тобто, мій 345 МБ диск насправді є 330 МБайтним.[60]

Про виділення місця під свопінґ читайте в розділі 6.5.

### 5.10.3 Деякі зразки розділу дисків

Раніше я користувався диском об'ємом 109 МБайт, тепер я маю 330 МБайтний. Я поясню, як я розділив ці диски на підрозділи.

109 МБайтний диск я розбивав тисячами способів тоді, коли мої потреби змінювалися. Було два типових сценарії. Спочатку я користувався MS-DOS'ом разом з Лінаксом. Мені потрібно було біля 20 МБайт простору на диску, щоб мати мінімальний DOS, компілятор Сі, редактор, кілька інших програмок та трохи вільного простору, щоб не відчувати клаустрофобії. Для

тим паче в Лінаксі) Ви маєте доступ до всіх системних ресурсів за допомогою звичайний команд - Ви можете переносити систему з диску на диск, встановлювати завантажувальні сектори на ті диски, які Ви забажаєте, можете змінювати головний завантажувальний блок (MBR) за бажанням та відновлювати його, не забувайте одного єдиного правила при цьому - RTFM (або, як це приблизно можна перекласти «вивчайте матеріальну частину». Сподіваємось допитливий читач зможе відшукати більш дослівний переклад цієї абревіатури самостійно). *(ДК)*

[60] Sic transit discus mundi.

Лінакса я мав 10 МБайтний своп-підрозділ і решта (тобто 79 МБайт) належала всім Лінаксівським файлам на одному розділі. Я експериментував з окремими розділами для кореня, `/usr`, `/home`, але у мене не було ніколи достатньо простору одним шматком настільки великим, щоб мене це зацікавило.

Коли я втратив потребу в DOS-і, я переформатував диск так, що я мав 12 МБайтний розділ під свопінґ і знову ж таки решта йшла на один розділ.

330 МБайтний диск я розділив на кілька підрозділів приблизно так:

| | |
|---|---|
| 5 MB | коренева файлова система |
| 10 MB | підрозділ для свопінґу |
| 180 MB | файлова система `/usr` |
| 120 MB | файлова система `/home` |
| 15 MB | чорновий розділ |

Чорновий розділ використовувася для деяких експериментів з різномнітними речами, які вимагають власного підрозділу, тобто для різних версій Лінакса чи для порівняння швидкостей файлових систем. Коли цей розділ не використовувався ні під що, я робив з нього простір під свопінґ (я люблю мати *багато* відкритих вікон).

### 5.10.4 Як додати нові диски до Лінаксу

Додання нового дискового простору до Лінаксу - справа проста, якщо вся апаратура встановлена і сконфіґурована (встановлення апаратури не розглядається в цій книжці). Ви форматуєте диски так, як Вам потрібно, створюєте підрозділи та файлові системи, так як це було описано раніше та додаєте відповідні рядки до `/etc/fstab`, щоб підрозділи монтувалися автоматично.

### 5.10.5 Економія дискового простору

Найкраща порада - це не встановлювати непотрібні програми. Більшість версій Лінакса запропонує при установці деякі варіанти для вибору тих пакетів, що будуть встановлені. Проаналізувавши список, Ви можете прийти до висновку, що Вам не потрібна більшість із цих пакетів. Це збереже Вам море місця. Навіть, якщо Вам потрібна якась певна програма, можливо, що Вам не потрібна вона вся. Наприклад, деяка документація може не знадобитися, так само, як можуть бути непотрібними деякі Elisp файли для GNU Emacs'a, деякі шрифти для X11 або деякі бібліотеки для програмування.

Якщо не можна стерти певні пакети, можливо їх можна стиснути. Програми стискання файлів, такі, як `gzip` та `zip` стиснуть (так само як і розширять) індивідуальні файли чи групи файлів[61]радиційним для Юніксів є `compress` для стискання файлів та `uncompress` чи `zcat` для їх розширення. `gzexe` може стискати та розширяти файли непомітно для користувача (програми, які не використовуються, стискаються та розширюються пізніше, тоді коли вони знадобляться). Експериментальна система `DouBle` зможе стискати всі файли в файловій системі непомітно для програм, які ними користуються. Якщо Ви знайомі з такими продуктами, як Stacker для DOS[62], то принцип роботи той же.

[61] **Прим. перекл.:** T *(ДК)*
[62] **Прим. перекл.:** чи для Macinosh'a *(ДК)*

# Розділ 6

# Керування пам'яттю

*Minnet, jag har tappat mitt minne,*
*Ӧar jag svensk eller finne*
*kommer inte ihåg...*
*(Bosse Ӧsterberg)*

Цей розділ описує принципи роботи менеджера пам'яті Лінакса, такі як віртуальна пам'ять та дисковий кеш. Описані призначення, принципи роботи та ті деталі, які треба приймати до уваги системному адміністратору.

## 6.1 Що таке віртуальна пам'ять?

Лінакс підтримує **віртуальну пам'ять**. При використанні диску, як уявного розширення до фізичної пам'яті, ефективний розмір пам'яті, що використовується зростає відповідно. Якщо якісь розділи пам'яті не використовуються на даний момент, ядро запише їх на диск і зможе використати звільнений в оперативній пам'яті простір під щось інше. Якщо скинуті на диск області пам'яті знадобляться знову, ядро прочитає їх знову з диску в пам'ять. Все це виконується непомітно для користувача. Програми в Лінаксі бачать тільки, що система має більший об'єм пам'яті і навіть не підозрюють, які саме з ділянок пам'яті знаходяться в даний момент на диску, а які в оперативній пам'яті. Звичайно ж, читання та запис на диск є значно повільнішими, порівняно з оперативною пам'яттю (в тисячі разів повільніше). Тому при користуванні віртуальною пам'яттю програми будуть працювати повільніше. Та частина диску, що використовується для віртульної пам'яті називається **простором свопінґу**.

Для свопінґу Лінакс може використовувати або звичайний файл в файловій системі, або окремий розділ, виділений спеціально для цього на диску. Віртуальна пам'ять з використанням підрозділу на диску працює швидше, ніж свопінґ у файл. Але змінити розмір файлу для свопінґу набагато легше і швидше, і це не потребує переформатування диску (і, можливо, встановлення всього з самого початку). Якщо Ви знаєте, скільки простору під свопінґ Вам потрібно, Ви можете відразу форматувати весь диск з виділенням підрозділу під свопінґ, але якщо Ви ще непевні, то Вам краще попробувати спочатку попрацювати з файлом, покористуватися системою деякий час і потім вирішити, який розмір розділу для свопінґу Вам потрібен.

Крім того варто знати, що Лінакс може користуватися кількома областями для свопінґу одночасно. Це означає, що, якщо великі розділи для свопінґу потрібні тільки інколи, замість того, щоб тримати їх постійно, можна додавати ці області тільки на той час, коли саме вони необхідні.

Дещо про термінологію: комп'ютерні спеціалісти відрізняють два різних режими користування віртувальною пам'яттю - свопінґ [1] (перенесення області пам'яті цілого процесу на диск) та пейджінґ[2] (запис на диск окремих сторінок пам'яті, непотрібних в даний момент). Пейджінґ є більш ефективним, і це є саме те, що робить Лінакс. Але за традицією всі називають це свопінґом.[3]

## 6.2 Створення простору для свопінґу

Файл для свопінґу - це самий звичайний файл з точки зору ядра. Єдина річ, яка має значення для ядра, це те, що цей файл не має дірок і що він приготований для свопінґу командою `mkswap`. Однак, цей файл повинен бути на локальному диску, тобто не може знаходитись на файловій системі NFS[4].

Замітка про дірки є важливою. Своп-файл створюється на диску таким чином, що ядро має можливість швидко

[1] **Прим. перекл.:** swapping *(ДК)*

[2] **Прим. перекл.:** paging *(ДК)*

[3] І цим без потреби доводять маститих метрів комп'ютеризації до стану кипіння.

[4] **Прим. перекл.:** Що є надзвичайним недоліком Лінакса порівняно з SunOS чи Solaris. Маючи можливість створювати своп-файл на NFS Ви можете створювати бездискові клієнти - тобто робочі станції, які можуть завантажуватись з мережі, монтувати всі файлові системи через NSF з сервера та мати своп-файл змонтований так само з сервера. Перші дві пункти можливі в Лінаксі, але неможливість створення своп-файлу через NFS обмежує робочу пам'ять бездискового Лінакс-клієнта до фізичної пам'яті локального комп'ютера, що, звичайно ж, в багатьох випадках не є достатнім *(ДК)*

«скинути» сторінку з пам'яті на диск, не проходячи через весь процес виділення сектора на диску. Оскільки дірка в файлі означає, що для певного шматка файлу не зарезервовано блоків на диску, ядро буде записувати інформацію в неіснуючі блоки (можливо зайняті іншими файлвми).

Вірний спосіб для створення файлу свопінґу є такий:

```
$ dd if=/dev/zero of=/extra-swap bs=1024 count=1024
1024+0 records in
1024+0 records out
$
```

де `/extra-swap` - це назва файлу, призначеного під свопінґ, а його розмір дано після `count=`. Краще всього, якщо розмір буде кратний чотирьом, бо ядро скидає у своп-файл сторінки пам'яті розміром в 4 кілобайти. Якщо розмір буде не кратним 4, то останні кілька кілобайт не будуть використані.

Підрозділ для свопінґу теж є самим звичайним підрозділом. Він створюється точнісінько, як будь-який інший підрозділ. Єдиною відмінністю є те, що він використовується як сирий диск - без файлової системи на ньому. Краще, якщо підрозділ, призначений під свопінґ буде мати тип 82 - Лінакс своп-підрозділ. Навіть при тому, що ядро не звертає уваги на типи розділів, така установка зробить список підрозділів трохи чіткішим.

Після створення або файлу або підрозділу для свопінґу, на ньому потрібно записати «підпис»[5] своп-файлу. Цей підпис містить деяку адміністративну інформацію і він використовується ядром. Команда для створення такого підпису є `mkswap`:

```
$ mkswap /extra-swap 1024
Setting up swapspace, size = 1044480 bytes
$
```

Відмітимо, що навіть після цієї операції своп-простір все ще не використовується - він існує, але ядро не використовує його як віртуальну пам'ять.

Будьте дуже обережними при використанні команди `mkswap`, оскільки вона не перевіряє, чи використовується цей дисковий простір чимось іще. *Ви можете запросто знищити важливі файли і цілі розділи диску цією командою* На щастя командою `mkswap` потрібно користуватися тільки при встановленні системи.

[5] **Прим. перекл.:** signature *(ДК)*

Менеджер пам'яті Лінакса обмежує розмір кожної ділянки своп-простору до приблизно 127 МБайт (з різних технічних причин справжній розмір є $(4096 - 10) \times 8 \times 4096 = 133890048$ байт чи 127,6875 мегабайт). Однак, можна користуватися шістнадцятьма такими розділами одночасно, що робить максимальний розмір простору для свопінґу рівним приблизно 2 Ґіґабайтам.[6]

## 6.3 Використання свопінґу

Підготований для використання простір свопінґу можна привести в дію командою `swapon`. Ця команда каже ядру, що своп-простір можна використовувати. Параметром до команди дається маршрут, вказуючий яким простором саме користуватись. Отже, для того, щоб почати свопінґ на тимчасовому файлі, можна скористуватися такою командою:

```
$ swapon /extra-swap
$
```

Розділи і файли для свопінґу будуть підключатися автоматично при старті системи, якщо вони вказані в файлі `/etc/fstab`.

```
/dev/hda8       none        swap        sw     0     0
/swapfile       none        swap        sw     0     0
```

Стартовий скрипт виконає команду `swapon -a`, яка почне свопінґ на всіх файлах і розділах свопінґу, вказаних в `/etc/fstab`. Отже, командою `swapon` потрібно користуватися тільки тоді, коли виникає потреба в додатковому просторі для свопінґу.

Прослідкувати за використанням своп-простору можна за допомогою команди `free`. Вона скаже скільки всього використовується своп-пам'яті.

```
$ free
             total       used       free     shared    buffers
Mem:         15152      14896        256      12404       2528
-/+ buffers:            12368       2784
Swap:        32452       6684      25768
$
```

[6] Ґіґабайт тут, ґіґабайт там, скоро ми будемо говорити про справжню пам'ять.

Перший рядок (`Mem:`) показує скільки є фізичної пам'яті. Стовпчик «всього»[7] не відображає пам'ять, яка використовується ядром, яка завжди приблизно дорівнює мегабайту. Стовпчик використаної пам'яті[8] показує кількість пам'яті, що використовується (другий рядок не рахує буферів). І стовчик вільної пам'яті показує пам'ять, що не використовується зовсім. Стовпчик «shared» вказує об'єм пам'яті, що спільно використовується кількома процесами - чим більше, тим краще. Стовпчик буферів демонструє розмір дискових буферів на даний момент.

Останній рядок (`Swap:`) показує всю ту ж інформацію відносно своп-простору. Якщо в цьому рядку стоять суцільні нулі - свопінґ не приведено в дію.

Ту ж саму інформацію можна отримати командою `top` або подивившись файл `/proc/meminfo` в файловій системі `proc`. Поки що неможливо отримати інформацію про конкретне використання тої чи іншої ділянки своп-простору.

Припинити використання своп-простору можна командою `swapoff`. Найчастіше немає потреби цього робити за виключенням тимчасових своп файлів. Всі сторінки, що містяться в даний час на даній ділянці своп простору спочатку будуть перенесені в пам'ять. Якщо фізичної пам'яті не вистачає для цього, то система скине ці сторінки на якусь іншу ділянку свопу. Якщо ж віртуальної пам'яті не досить для того, щоб втримати всі сторінки в пам'яті система безкінечно перекидати сторінки пам'яті туди-сюди, з диску в пам'ять і з пам'яті знову на диск. [9] Через довгий період часу вона повинна відійти, але на протязі цього часу від системи не добитися ніякого толку. Перш, ніж забирати у системи своп впевніться, що системі досить пам'яті, щоб вижити (за допомогою команди `free`).

Весь простір свопінґу, приведений в дію командою `swapon -a`, так само можна деактивізувати однією командою `swapoff -a`. Вона шукає в файлі `/etc/fstab`, що потрібно деактивізувати. Весь своп-простір, що було приведено в дію вручну продовжуватиме працювати [10]ідмінності Лінакса і SunOS/Solaris. В SunOS взагалі неможливо працювати без своп-простору. Якщо при старті система бачить, що не виділено простору для свопінґу, вона спиняється і чекає на те,

[7] **Прим. перекл.:** total *(ДК)*

[8] **Прим. перекл.:** used *(ДК)*

[9] **Прим. перекл.:** Можливо, що багато процесів просто підуть у смітник. Дуже часто це виглядає, як установка Windows 95 - комп'ютер здається підвішеним у безповітряному просторі. *(ДК)*

[10] **Прим. перекл.:** В *(ДК)*

що користувач вкаже, яким розділом користуватися для свопінґу. SunOS має єдину команду `swapon`, із синтаксисом таким самим, як і Лінакс. Але команди `swapoff` в SunOS немає, тобто деактивізувати своп простір під час роботи системи неможливо. Solaris багато в чому подібна до Лінакса в цьому відношенні, тільки має інший формат команди. Команда для керування свопом в Solaris'і одна єдина - `swap`, але коли використовується з різними параметрами виконує те ж, що і `swapon`, `swapoff` та `free` (в деякому відношенні - `swap -l` показує активізовані своп підрозділи та файли в системі, та наскільки вони використовуються, без показу проценту використання фізичної пам'яті.)

Крім того, як уже відмічалося раніше по тексту, і SunOS і Solaris вміють користуватися NFS для створення файлів свопінґу на NFS сервері..

Інколи, навіть при величезному об'ємі невикористаної фізичної пам'яті, своп пам'ять виявляється зайнятою. Це трапляється, якщо в якийсь момент деякий процес вимагав багато пам'яті і сторінки пам'яті виявилися скинутими на диск. Після того, як процес закінчив свою роботу, ті сторінки, які були скинуті в своп, так і залишаються там без їх автоматичної очистки, до того часу, поки зайнята ними пам'ять не знадобиться для іншого процесу. Про це немає зовсім ніякої причини турбуватися, але, якщо Ви про це знаєте, у Вас може бути трохи спокійніше на душі.

## 6.4 Спільне використання простору свопінґу з іншими системами

Віртуальна пам'ять використовується багатьма системами. Оскільки вона потрібна кожній системі тільки в той час, коли ця система працює, то тоді, коли дана система не активна, дисковий простір, виділений під віртуальну пам'ять, гуляє. Було б більш ефективним, якби малася можливість користуватися одним і тим, же дисковим простором в кількох операційних системах. Це виявляється можливим, але вимагає деяких хакерських дій від системного адміністратора. Деякі поради щодо того, як це зробити є в Tips-HOWTO.

## 6.5 Виділення своп простору

Дехто радить мати своп пам'ять як мінімум вдвічі більшою від фізичної пам'яті системи, але це застаріла порада [11]ля того, щоб зрозуміти звідки така порада з'явилася - декілька слів з історії. Юнікси попередніх років користувалися своп-підрозділом системи для дампу пам'яті.

Під час навчання в університеті я підробляв оператором на M4030, і кілька разів мені довелося бачити, як виглядав дамп пам'яті там. Під час краху (в Юніксі кажуть, що ядро викликає стан паніки, або панікує) системи M4030 намагалася скинути поточний стан *всієї* своєї пам'яті на папір - тобто дамп пам'яті виглядав як нескінченне друкування потоків шістнадцядкових кодів. Кілометри і кілограми паперу із стовпчиками чисел по всій його ширині.

Приблизно те ж саме робили і Юнікси, з єдиною відмінністю, що вся пам'ять скидалася на диск. Диск в цьому випадку використовувася, як сирий пристрій без файлової системи (важко вимагати від системи, щоб в стані паніки вона ще дбала про якісь там файли). Пам'ять скидалася на своп розділ диску і робиться від гори до низу - тобто з протилежного кінця до того, з якого система починає користуватися диском під свопінґ. Коли систему після цього все таки вдасться завантажити, то в верхній частині розділу ще залишається образ пам'яті системи від часу паніки, навіть якщо система вже почала свопінґ деяких сторінок пам'яті на диск. Лінакс не користується такою схемою дампу пам'яті, тому рекомендації мати своп вдвічі більший від фізичної пам'яті не мають під собою основи.

. Ось кілька рекомендацій щодо того, як вірно визначити об'єм віртуальної пам'яті:

1. Визначіться із загальним об'ємом пам'яті, яка Вам потрібна. Це найбільший об'єм пам'яті, який Вам можливо знадобиться коли-небудь. Тобто, це сума всіх об'ємів всіх програм, якими Ви хочете користуватися одночасно. Можна дізнатися про це, запустивши всі програми, якими Ви захочете одночасно користуватися.

   Наприклад, якщо Ви хочете працювати в X, Вам потрібно виділити під це біля 8 МБайт, gcc сам вимагає кількох меґабайт (деякі файли вимагають надзвичайно великого об'єму - до десятків маґабайт, але звичайно близько

[11] **Прим. перекл.:** Д *(ДК)*

чотирьох має бути досить), і так далі. Ядро займе біля меґабайта, командні оболонки [12] та деякі невеликі програмки-утілітки можливо потребують кілька сотень кілобайт (скажімо меґабайт всі разом). Немає ніякої потреби вираховувати дуже точні величини, досить приблизні округлення - це саме те що потрібно в цьому випадку. Але при цьому краще б бути трошки песимістом.

Пам'ятайте, що якщо системою будуть користуватися кілька людей одночасно, вони всі будуть користуватися пам'яттю. Однак, якщо двоє користувачів користуються однією і тією ж програмою одночасно, загальне використання пам'яті при цьому не вдвічі більше, оскільки деякі сторінки пам'яті використовуються спільно - всі спільні бібліотеки та всі програми в пам'яті існують тільки в одному екземплярі.

`free` та `ps` дадуть Вам уяву про потреби пам'яті.

2. Додайте дещо на всяк випадок. Обрахований об'єм може бути не достатньо точним, через те, що: Ви забули деякі програми, які тепер Вам просто необхідно додати до попереднього списку. Тому непогано б мати деяку вільну пам'ять про всяк випадок. Парочка-друга меґабайт має бути досить. (Звичайно ж, краще мати більше, ніж менше, але пам'ятайте, що невикористана віртувальна пам'ять - просто загублена, тому немає сенсу створювати надто великий запас. Дивіться далі про додання віртуальної пам'яті в процесі роботи. Крім того, оскільки завжди приємніше мати справу з круглими числами, можна округлити обраховане число до найближчого цілого меґабайту.

3. Обрахувавши свої потреби в пам'яті Ви будете знати, скільки пам'яті потрібно всього. Щоб дізнатися потрібний розмір своп-простіру, просто віднімiть від цього числа об'єм Вашої оперативної пам'яті. (В деяких версіях Юнікса, Вам потрібно виділити простір під фізичну пам'ять також, тобто об'єм пам'яті, обрахований на другому кроці, і буде тим, що потрібно виділити під своп. Тобто, Вам не потрібно нічого віднімати.

4. Якщо обчислена таким чином своп-пам'ять є занадто великою (більше, ніж у кілька разів більша за фізичну

[12] **Прим. перекл.:** shell *(ДК)*

пам'ять), скоріше всього Вам варто вкласти гроші в додаткову фізичну пам'ять, інакше швидкість роботи системи буде незадовільною.

Хоча б яку небудь своп-пам'ять варто мати в будь-якому випадку, навіть якщо Ваші обрахунки показують, що Вам вона не потрібна. Лінакс користується віртульною пам'яттю досить агресивно, щоб мати якомога більше фізичної пам'яті вільною. Лінакс викидає на диск ті сторінки пам'яті, які не використовуються на даний момент, щоб звільнити пам'ять під майбутній свопінґ, навіть, якщо нічого його не потребує на даний момент. Таким чином майбутній свопінґ може бути зроблений швидше - не потрібно чекати на те, що сторінки будуть скинені на диск - це вже було зроблено тоді, коли диск не був нічим не зайнятий.

Своп-простір можна поділити між кількома дисками. В деяких випадках це може підвищити швидкість роботи, залежно від швидкості роботи дисків та того, як до дисків здійснюється доступ. Можливо Вам варто трохи поекспериметувати, але пам'ятайте при цьому, що вірно експериментувати досить важко. Не варто довіряти будь-кому, хто запевнятиме Вас, що та чи інша схема має надзвичайну перевагу над іншою такою ж.

## 6.6 Буферний кеш

[13]

Швидкість читання з диску[14], є набагато меншою, ніж швидкість доступу до (справжньої) пам'яті. Часто трапляється так, що Вам потрібно прочитати одні й ті ж дані кілька разів з диску на протязі відносно короткого відтинку часу. Простий приклад: Ви читаєте листа, який прийшов електронною поштою, потім відкриваєте цього ж листа в редакторі для того, щоб дати на нього відповідь, потім відкриваєте програму електонної пошти, яка відішле цю відповідь. Або інший випадок: скільки разів виконується команда `ls` в системі з багатьма користувачами? Швидкість роботи системи можна підняти, якщо читати всі дані з диску тільки один раз, і тримати їх в пам'яті після цього доти, доки вони більше не потрібні. Таким чином підвищується швидкість всіх операцій читання з диску крім самої першої. Це називають **дисковим**

[13] **Прим. перекл.:** The buffer cache *(ДК)*

[14] крім звичайно швидкості читання з RAM-диску, із зрозумілих причин

**буфером**, і пам'ять, яка використовується при цьому називають **буферним кешем**.

Пам'ять дуже часто обмежена, і навіть більше того, її часто недостатньо. Тому дисковий буфер не вдається зробити достатньо великим (тобто, він не може зберігати всі дані, які Вам потрібні). Коли кеш заповнюється, дані, до яких не було доступу на протязі найдовшого часу стираються, і вивільнена пам'ять використовується під нові дані.

Дисковий буфер працює також для запису. З одного боку, дані записані на диск, часто читаються після цього знову в пам'ять (наприклад, програма записана на диск редактором, читається після цього компілятором). Отож, непогано мати тільки-що записані дані і в кеші також. З іншого боку, помістивши дані тільки в кеш, і не записуючи насправді їх на диск, операція запису робиться значно швидшою. Справжній запис на диск виконується після цього, на фоні якоїсь іншої роботи, не заповільнюючи їх роботи.

Більшість операційних систем мають дискові буфери та кеш (хоча в деяких випадках вони називаються інакше), але не всі з цих систем працюють за одними й тими ж принципами. Деякі з них є з **наскрізним записом**[15] - дані записуються на диск без затримки (але, звичайно ж, тримаються в кеші). Кеш, у якому запис відбувається пізніше, називається **кешем з відкладеним записом**. Відкладений запис значно ефективніший, ніж наскрізний запис, але в той же час, більш схильний до помилок: якщо щось негаразд з системою - або зникає напруга (хтось висмикнув шнур з розетки), або дискету вийняли з приводу до того, як дані з буферу записані на неї - всі дані, які були в буфері на цей момент, загублені. Інколи це навіть може означати, що файлова система на диску (якщо така була там) не в повному порядку. Бо не записані на диск дані можуть бути деякими контрольними даними файлової системи.

Через це не можна вимикати систему без виконання відповідної процедури вимикання (див. підрозділ 7), просто витягнувши шнура з розетки. Дискету з дисководу можна витягувати тільки після розмонтовування її, або ж тільки після того, як команда, яка виконувала запис на дискету, повідомить Вам про те, що вона закінчила писати. Команда **sync зливає**[16] буфер, тобто примусово записує всі незаписані дані на диск, і нею треба користуватися тоді, коли треба впевнитися, що все

[15] **Прим. перекл.:** write-through *(ДК)*
[16] **Прим. перекл.:** flushes *(ДК)*

записано безпечно на диск. В системах з традиційними Юніксами існує програма `update`, яка виконується в фоновому режимі і робить `sync` кожні 30 секунд, тобто в більшості випадків немає необхідності робити `sync`. В Лінаксі є інший демон `bdflush`, який робить чистішу роботу - він виконує `sync` частіше для того, щоб запобігти раптовим короткочасним зависанням системи, які інколи трапляються при високих нагрузках на диск викликаних командою `sync`.

В Лінаксі `bdflush` запускається процесом `update`. В більшості випадків немає причини піклуватися про них, але, якщо з деяких причин трапляється так, що демон `bdflush` вмирає, то ядро видасть попередження про це і його потрібно запустити вручну (`/sbin/update`).

Дисковий кеш звичайно тримає в буфері не файли, а блоки - найменші одиниці дискового вводу/виводу (в Лінаксі вони рівні 1 кбайту). Таким чином тримаються в буфері також директорії, суперблоки, інформація файлової системи та дані дисків без файлових систем.

Ефективність кешу в першу чергу залежить від його розміру. Занадто малий кеш практично не робить нічого - всі ті дані, які він тримає в буфері будуть злиті на диск раніше, ніж кеш встигне їх використати. Критичний розмір кешу залежить від того, скільки даних читається і записується, і як часто потрібен доступ до одних і тих же даних. Єдиний шлях це дізнатися - це шлях експерименту.

Якщо дисковий кеш має фіксований розмір, то його надмірний розмір теж знижує ефективність роботи. Кеш забирає оперативну пам'ять від системи, пам'ять стає меншою, що в свою чергу приводить до підвищення частоти свопінґу, а свопінґ - такий же повільний, як і інші дискові перації. Для підвищення ефективності використання пам'яті Лінакс автоматично використовує всю вільну пам'ять під дисковий кеш, і автоматично зменшує розмір кешу, коли програмам потрібно більше пам'яті.

В Лінаксі від адміністратора не вимагається ніяких додаткових дій для користування кешем - все відбувається автоматично. Крім вимикання комп'ютера за правилами та виймання дискети з дисководу тоді, коли потрібно, Вам немає про що турбуватися.

Windows 95|seecrying
crying
завантаження
booting
starting a computer
вмикнення комп'ютера

# Розділ 7

# Завантаження та вимкнення системи

*Start me up*
*Ah... you've got to... you've got to*
*Never, never never stop*
*Start it up*
*Ah... start it up, never, never, never*
*You make a grown man cry,*
*you make a grown man cry*
*(Rolling Stones)*

Цей розділ пояснює, що відбувається тоді, коли система з Лінаксом вмикається та вимикається і те, як вмикати та вимикати Лінакс за правилами. Якщо не робити цього так, як потрібно, то можна пошкодити чи навіть загубити файли.

## 7.1 Огляд процесів старту та вимкнення

Дія вмикнення комп'ютера та завантаження його операційної системи[1] називають **завантаженням**[2]. Назва походить від образу, коли комп'ютер витягує сам себе з болота за мотузки на чоботах, але процес сам по собі трохи більш реалістичний.

Стартуючи, на самому початку, комп'ютер читає невеликий шматочок комп'ютерного коду, який носить назву **початковий завантажувач**[3], який в свою чергу вантажить та стартує

[1] На ранніх комп'ютерах було занадто мало просто вмикнути комп'ютер, Ви повинні були б ще вручну виконати певні дії для того, щоб операційна система завантажилась в пам'ять. Ці нові сучасні цяцьки вміють робити це вже самі.

[2] **Прим. перекл.:** booting, Дослівний переклад цього терміну означає щось ніби «взування чобіт». Подальші пояснення в цьому абзаці трактують саме цю назву. *(ДК)*

[3] **Прим. перекл.:** bootstrap loader *(ДК)*

початковий завантажувач
bootstrap loader
завантажувальний сектор, boot sector
дисковий кеш, buffer cache
втрати даних, data loss

операційну систему. Початковий завантажувач найчастіше записується в певному місці жорсткого диску чи дискети. Причина такого двоступеневого процесу старту операційної системи полягає в тому, що операційна система велика і складна, а той початковий код, який читає з диску комп'ютер, повинен бути дуже малим - всього кілька сотень байт - для того, щоб не приводити до зайвого ускладнення програмного забезпечення, вбудованого «в залізі» комп'ютера.

Різні комп'ютери виконують початкове завантаження по різному. У світі PC комп'ютер (або точніше його BIOS) читає перший сектор (що називають **завантажувальним сектором**) з гнучкого або жорсткого диску. Початковий завантажувач є частиною цього сектора. Він читає та вантажить операційну систему з іншого місця на диску (або з якогось іншого місця).

Коли Лінакс (ядро) прочитано в пам'ять, він проводить ініціалізацію апаратного забезпечення і після цього запускає `init`. `init` стартує інші процеси та дозволяє користувачам реєструватися в системі та виконувати свою роботу. Подробиці цього процесу розглядатимуться далі.

При вимкненні системи всім процесам повідомляється, що вони повинні закінчити свою роботу (це змушує їх закрити всі відкриті на даний час файли та виконати інші подібного роду дії), після цього відмонтовуються файлові системи та області свопінґу, і врешті-решт, на екрані друкується повідомлення про те, що можна вимкнути напругу. Якщо ця процедура не виконана як слід, можуть трапитися неприємі речі, а саме: може виявитися, що буферний кеш файлової системи не злито на диск. Це означає, що всі дані, які перебували в буфері загублено і, що файлова система пошкоджена, або інколи навіть, що нею не можна користуватися.

## 7.2 Старт системи зблизька

Лінакс можна завантажити або із гнучкого, або із жорсткого диску[4] доданням деяких дрібничок PC можна зробити бездисковим клієнтом - тобто комп'ютером, в якому немає ні гнучких ні жорстких дисків, і який можна грузити з сервера в мережі. Справжня знахідка для обчислювальних залів на багато користувачів чи для навчальних класів в школах чи університетах. Для робочих станцій Sun Sparc не треба нічого

[4] **Прим. перекл.:** З *(ДК)*

додавати взагалі - кожен Sparc - це готовий бездисковий клієнт, який буде грузитися звідти, звідки скаже адміністратор. Теж саме - нові моделі Macintosh'ів, які побудовані з підтримкою завантаження з мережі (і для яких, до речі, теж існує версія Лінакса).

. Розділ «Установка» в Посібнику по установці та перших кроках в Лінаксі [5] розкаже Вам, як зробити так, щоб Лінакс грузився або звідти, або звідти.

Під час початкового старту комп'ютера BIOS виконує різноманітні тести, перевіряючи, що з апаратурою все в порядку,[6] і після цього починає справжнє завантаження. Він вибере дисковий пристрій (в більшості випадків - перший дисковід, якщо в нього вставлена дискета, якщо ж ні - то перший жорсткий диск, якщо такий встановлено в комп'ютері. Інколи порядок пошуку може змінюватися.) і прочитає самий перший сектор цього диску. Цей сектор називають **завантажувальним сектором**[9], у відношенні до жорстких дисків цей сектор називається також **головним завантажувальним сектором**[10], оскільки на диску може бути кілька розділів, і кожен із розділів має свій власний завантажувальний сектор.

BIOS
головний завантажувальний сектор, master boot record
MBR
завантажувальний сектор розділу, partition boot record
завантажувальний запис, boot record!master|seeMBR
boot record!partition|seepartition boot record
завантаження з дискети, booting!from floppy
дискета!завантаження з, floppy!booting from|дивзавантаження

Завантажувальний сектор містить невелику програмку (малу настільки, що вона може поміститися в один сектор). Відповідальністю цієї програмки є: прочитати операційну систему з диску та стартувати її. При завантаженні Лінакса з дискети, в завантажувальному секторі записана програма, яка просто-напросто читає кілька сотень перших блоків (в залежності від справжнього розміру ядра) в попередньо визначену ділянку пам'яті. Дуже часто на завантажувальній дискеті файлової системи немає. Ядро просто записується в послідовності секторів. Це значно полегшує процес старту системи. Однак, можливо також вантажити Лінакс з дискети з файловою системою, користуючись LILO (the LInux LOader)[11]. [12]

При завантаженні з жорсткого диску, код (програма) в головному завантажувальному секторі аналізує таблицю розділів диску (яка також знаходиться в головному

[5] **Прим. перекл.:** Installation and Getting Started guide ([?]) *(ДК)*
[6] це називається **power on self test**[7] або POST[8]
[9] **Прим. перекл.:** boot sector *(ДК)*
[10] **Прим. перекл.:** main boot record *(ДК)*
[11] **Прим. перекл.:** завантажувач Лінакса *(ДК)*
[12] **Прим. перекл.:** В світі Лінакса для Sparc та для Alfa процесорів назва LILO трансформується в SILO та MILO відповідно *(ДК)*

завантаження!з жорсткого диску
жорсткий диск!завантаження|див. завантаження
таблиця підрозділів
активний підрозділ
LILO

завантажувальному блоці) і знаходить активний підрозділ. Для того, щоб було зрозуміло, що з такого розділу можна грузитися, він позначається спеціальним прапорцем - бітом. Після цього код завантажувача з MBR читає завантажувальний сектор активного розділу і передає на нього керування процесору. Код в завантажувальному секторі активного розділу робить те ж саме, що і код в завантажувальному секторі дискети: читає ядро з диску та запускає його на виконання. Звичайно ж, подробиці старту відрізняються. Не має ніякого сенсу тримати на диску окремий розділ тільки для того, щоб тримати на ньому ядро. Тож, завантажувальний код в завантажувальному секторі диску не може просто читати все підряд з послідовності секторів. Він повинен спершу визначити де операційна система розмістила ядро на диску. Є кілька різних шляхів вирішення питання пошуку ядра на диску, але найбільш вживаний - це застосування LILO. (Подробиці того, як саме це зробити, виходять за межі тематики цієї дискусії, отож дивіться документацію по LILO, щоб зрозуміти ці процеси краще.)

При завантаженні комп'ютера з LILO, як правило, він стартує Ваше «основне»(«робоче») ядро. LILO можна сконфіґурувати так, що він буде завантажувати інші ядра, або навіть одну з кількох оперативних систем. Крім того, LILO дає користувачеві вибір того, яке ядро чи яку систему стартувати при даній завантаженні. LILO можна настроїти так, що при натисканні клавіш `alt`, `shift`, чи `ctrl` під час завантаження, він зупиниться, утримається від стандартного завантаження і спитає яку систему завантажити. Також LILO можна настроїти таким чином, що він завжди буде зупинятися при завантаженні на певний час щоб спитати, яку систему грузити. Якщо LILO не отримає відповіді на протязі цього часу, то завантажить основне ядро.

При використанні LILO ядру системи також можна передавати параметри з командного рядка.

**МЕТА:** Існують також інші завантажувачі окрім LILO. Інформація про них буде додана пізніше, в наступних версіях. loadlin.

Як завантаження з дискети, так і завантаження з жорсткого диску мають як свої плюси, так і мінуси. Але загалом завантаження з жорсткого диску виглядає набагато краще, оскільки можна позбутися цих дискет, які будуть тягатися скрізь і всюди. Крім того - це швидше. Однак, інколи можливо трохи важче встановити систему так, щоб вона завантажувалась з жорсткого диску. Тому багато хто спочатку завантажують систему з дискети. Після того, як система в основному встановлена і сконфіґурована, встановлюють LILO і починають грузитися з жорского диску.

Коли ядро системи вже прочитане в пам'ять системи, відбувається приблизно

таке:

стисканна ядра
відео плата
конфіґурація ядра
відео режими
текстовий режим
конфіґурація обладнання
конфіґурація драйверів пристроїв
повідмлення системи при старті
завантаження!повідомлення
коренева файлова система
файлові системи!коренева
монтування!коренева файлова система
LILO
паніка ядра
паніка
коренева файлова система тільки на читання
перевірка файлової системи
fsck

- Ядро Лінакса записується на диск стисненим (заархівованим). Отже воно має бути розархівованим спочатку. Початок ядра містить невеличку програмку, яка виконує цю операцію[13].

- Якщо Ваша система має відеоплату типу супер-VGA, то Лінакс зможе це розпізнати. Тоді Лінакс дасть можливість скористуватися спеціальними текстовими режимами, такими, як, наприклад, 100 стовпчиків по 40 рядків. Відео режим екрану можна встановити під час компіляції ядра. Тоді під час старту ядро не буде задавати зайвих запитань. Крім того цього ж можна добитися за допомогою LILO або `rdev`.

- Після цього ядро починає перевіряти яке обладнання має комп'ютер (жорсткі диски, приводи дискет, плати мережі...), та конфіґурує деякі з драйверів знайдених пристроїв відповідно. Під час цього на екран виводяться повідомлення про знахідки ядра. Наприклад, під час завантаження на своєму екрані я бачу:

```
LILO boot:
Loading linux.
Console: colour EGA+ 80x25, 8 virtual consoles
Serial driver version 3.94 with no serial options enabled
tty00 at 0x03f8 (irq = 4) is a 16450
tty01 at 0x02f8 (irq = 3) is a 16450
lp_init: lp1 exists (0), using polling driver
Memory: 7332k/8192k available (300k kernel code, 384k reserved, 176k data)
Floppy drive(s): fd0 is 1.44M, fd1 is 1.2M
Loopback device init
Warning WD8013 board not found at i/o = 280.
Math coprocessor using irq13 error reporting.
Partition check:
  hda: hda1 hda2 hda3
VFS: Mounted root (ext filesystem).
Linux version 0.99.pl9-1 (root@haven) 05/01/93 14:12:20
```

  Повідомлення на Вашому екрані будуть іншими. Вони залежать від обладнання, встановленого в системі, від версії Лінакса та від конфіґурації системи.

- Після цього ядро робить спробу змонтувати кореневу файлову систему. Місцезнаходження її задається під час компілювання ядра, командою `rdev` чи за допомогою LILO. Тип файлової системи визначається автоматично. Якщо ядро не може змонтувати кореневу файлову систему (наприклад, з тої причини, що Ви забули включити драйвер даної системи в ядро під час його компіляції), ядро панікує і зупиняє систему (все одно йому нема чого робити).

  Звичайно коренева файлова система монтується в режими тільки читання (це встановлюється так само, як і її місце). Це дає можливість перевіряти файлову систему під час монтування, бо не дуже гарно перевіряти файлову систему тоді, коли вона змонтована на запис.

- Після цього ядро стартує програму `init`, яка знаходиться в `/sbin/init`. `init` запускається в фоновому режимі[14] традиційних Юніксах самим

[13] **Прим. перекл.:** Це відноситься тільки до Лінакса i86, тобто для IBM-сумісних PC, і не стосується інших Лінаксів, таких як Sparc, Alfa, PowerPC, тощо, бо ці процесори не мають існуючої в світі PC межі на 640 кбайт пам'яті. *(ДК)*

[14] **Прим. перекл.:** В *(ДК)*

першим процесом є завжди процес з номером 0 - swapper. Цей процес стартує (народжує) один єдиний процес - процес init, який (як це не дивно) отримує номер 1. Всі інші процеси крім цих двох можуть народжуватися і вмирати поки працює процесор, отримують різні номери, але процеси 0 і 1 живуть поки жива система і вмирають разом з нею. init - багатодітний батько всіх інших процесів системи.

і завжди отримує номер 1. `init` виконує різні стартові функції. Що саме він робить залежить від його конфіґурації, дивіться розділ 8 (не написаний поки що). Як мінімум він стартує деякі важливі демони.

- Після цього `init` переходить в багато користувацький режим і стартує `getty` для віртуальних консолів та послідовних портів. Команда `getty` - це програма, яка дозволяє користувачам реєструватися в системі, використовуючи при цьому віртуальні консолі або послідовні порти. В залежності від того, як він сконфіґурований `init` може стартувати також інші програми.
- Після цього вже завантаження системи закінчена і система працює в своєму нормальному режимі.

## 7.3 Трохи більше про вимкнення

Важливою є процедура вимикання системи. Після невірного вимкнення Ваша файлова система може відправитися в смітник або будете мати попсовані файли. Це відбувається тому, що Лінакс записує дані на диск не відразу, а через певний інтервал і він не зможе записати дисковий кеш на диск при раптовому вимкненні. Дисковий кеш значно підвищує швидкість роботи, але також підвищує ймовірність помилок при невірному вимкненні живлення.

Іншим аргументом проти раптового вимкнення напруги, може бути те, що в багатозадачній системі може бути безліч процесів, які відбуваються одночасно в фоновому режимі, і вимкнення напруги може бути катастрофою. Виконуючи визначену послідовність при вимкненні системи Ви також забезпечуєте вірність вимкнення всіх фонових задач.

Команда для вимикання системи в Лінаксі - `shutdown`. Нею можна користуватися одним із двох способів.

В системі, де Ви - єдиний користувач, користуйтеся командою наступним чином. Вимкніть всі працюючі програми, вийдіть з усіх робочих сесій (з усіх віртуальних консолів), зареєструйтеся в системі як `root` на одній з консолей (або залишіться на одній консолі, якщо Ви працювали як `root`, але при цьому краще перейти в кореневу директорію, щоб позбутися проблем з відмонтуванням файлових систем). Після цього виконайте команду `shutdown -h now` . Якщо Ви хочете вимкнути систему з затримкою, замініть текст `now` знаком плюс за яким іде кількість хвилин затримки. Хоча в системі з одним користувачем це навряд чи потрібно.

Якщо ж Ваша система використовується багатьма користувачами, використовуйте команду `shutdown -h +`*`час повідомлення`*, де *час* - це час в хвилинах до зупинки системи, а *повідомлення* - це коротке пояснення причини зупинки системи.

```
# shutdown -h +10 'We will install a new disk. System should
> be back on-line in three hours.'
#
```

Це повідомлення попередить всіх, хто користується системою, що система вимикається через десять хвилин. І користувачам краще б вийти з системи,

якщо вони не хочуть загубити свої дані. Повідомлення друкується на кожному терміналі, на якому хто-небудь зареєструвався в системі, включаючи `xterm`'и.

```
Broadcast message from root (ttyp0) Wed Aug 2 01:03:25 1995...

We will install a new disk. System should
be back on-line in three hours.
The system is going DOWN for system halt in 10 minutes !!
```

Повідомлення повторюється автоматично кілька разів перед тим, як система вимкнеться з все коротшими і коротшими проміжками між послідовними попередженнями.

Потім починається справжнє вимкнення системи. Відмонтовуються всі файлові системи за винятком кореневої, вбиваються всі процеси користувачів (якщо все ще хто-небудь залишається в системі), вимикаються демони, відмонтовуються всі файлові системи, і зупиняється все взагалі. Коли все це зроблено, `init` друкує на екрані повідомлення про те, що Ви можете вимкнути машину. Тоді, *і тільки тоді*, можете простягати свої руки до вимикача живлення.

Інколи, хоча на стабільній системі дуже рідко, неможливо вірно вимкнути систему. Наприклад, якщо ядро панікує і поводить себе дивно, неможливо подати йому будь-яку команду. Тож вимкнути систему вірно дещо проблематично. І практично єдине, що Вам залишається, це сподіватися, що нічого поки-що не зіпсовано і вимкнути напругу. Якщо Ваші проблеми з системою трохи менші, ніж описані тільки що (наприклад, хтось сів на клавіатуру), і як ядро так і демон `update` все ще працюють, тоді варто зачекати кілька хвилин, щоб `update` міг записати всі буфери на диск і вимкнути напругу вже після цього.

Дехто вимикає систему виконуючи команду `sync`[15] тричі, чекають, поки зупиниться запис/читання з диску і вимикають напругу після цього[16]рацюючі в SunOS/Solaris'і добавляють ще і `halt` в кінці. `halt` зупиняє роботу процесора. Але в Лінаксі `halt` робить те ж саме, що і `shutdown -h -` відмотовує файлові системи і т.п.. Якщо в системі немає працюючих програм, то це майже те ж саме, що і користуватися `shutdown`. Але такий процес не відмонтовує ніяких файлових систем і це може призвести до деяких проблем з прапорцем «чистоти» файлової системи ext2. Отже, метод потрійного-sync'у *не рекомендується*.

(Якщо Вас це цікавить: причина *потрійного* sync'у в тому, що на початку історії Юнікса це давало досить часу, щоб закінчилися всі операції вводу/виводу.)

повідомлення про вимкнення
вимкнення!послідовність
крах
крах системи
вимкнення!термінове
невірне вимкнення|диввимкнення, термінове
буферний кеш
термінове вимкнення|диввимкнення, термінове
вимкнення!потрійний sync
перегрузка

## 7.4 Перевантаження

Перевантаження означає повторне завантаження системи. Це можна зробити спершу вимкнувши систему повністю, вимкнувши напругу і потім вмикнувши її знову. Простіший спосіб - це сказати команді `shutdown`, перегрузити систему замість того, щоб вимикати систему повністю. Для цього їй треба вказати параметр `-r`, тобто виконати таку команду `shutdown -r now`.

Більшість Лінаксів сконфіґуровані таким чином, що при натисканні на клавіатурі трьох клавіш ctrl-alt-del виконується команда `shutdown -r now`. [17] Це призводить до перегрузки системи. Однак, це теж можна змінити, і краще це зробити, щоб дати користувачам деякий час перед перегрузкою системи.

[15] `sync` зливає на диск дисковий кеш

[16] **Прим. перекл.:** П *(ДК)*

[17] **Прим. перекл.:** Невелике уточнення - тільки при роботі у текстовому режимі на терміналі. В X Windows «трьох-клавішний салют» не робить нічого. X Windows перехоплює сигнали від клавіатури і трактує їх по своєму. *(ДК)*

ctrl-alt-del
init!та ctrl-alt-del
однокористувацький режим
init!однокористувацький режим
завантаження!з аварійної дискети
привід дисководу
резевні копії
установочні диски
аварійні дискети!використання
аварійні дискети!виготовлення
віртуальний диск

Системи, до яких доступ вільний, можна настроїти так, щоб при натисканні ctrl-alt-del нічого не відбувалося.

## 7.5 Однокористувацький режим

Команда `shutdown` також може використовуватися для переводу системи в однокористувацький режим, тобто такий режим, в якому ніхто не може зареєструватися в системі крім користувача `root` з системної консолі. Цей режим використовується системними адміністраторами для виконання тих операцій, які неможливо виконати в звичайному режимі. Однокористувацький режим з більшими подробицями описано в розділі **??**.

## 7.6 Аварійні завантажувальні дискети

Не завжди комп'ютер може завантажитися з жорсткого диску. Наприклад, якщо Ви помилилися при конфіґурації LILO, може статися так, що система не буде грузитися. Для таких ситуацій потрібно мати запасний спосіб завантеження системи, який би працював завжди (як мінімум до тих пір, поки працює апаратура). Для типового PC це означає завантеження з дискети.

Більшість Лінаксів, що розповсюджуються на сьогодні, дозволяють створити **аварійну завантажувальну дискету**. Варто створити її як можна швидше. Однак, інколи завантажувальні дискети мають тільки ядро, передбачаючи, що Ви будете користуватися іншими програмами з придбаного комплекту для виправлення всіх Ваших негараздів. Буває так, що цих програм не досить. Наприклад, Вам потрібно відновити кілька файлів з резервних копій зроблених раніше, а не з установочних дисків.

Таким чином, може буде потрібно створити власні аварійні дискети з кореневою файловою системою на них. Інформацію про те, як це зробити, можна знайти в документі написаному Ґрехам Чепменом [18] *Bootdisk HOWTO* ([**?**]). Звичайно, потрібно потім підтримувати свої завантажувальну та кореневу дискету в порядку і поновлювати їх при необхідності.

При аварійному завантаженні дисковід, який використовується для монтування кореневої файлової системи, неможливо використовувати для інших цілей. Якщо система має всього один дисковід, це приносить великі незручності. Однак, якщо в системі досить пам'яті, завантажувальну дискету можна створити таким чином, що дискета з кореневою файловою системою прочитається в пам'ять, і в пам'яті створиться віртуальний диск [19]. Щоб це було можливо, ядро системи повинно бути настроєне відповідним чином. Після того, як коренева файлова система змонтована з віртуального диску, дисковід може використовуватися для монтування інших дискет.

[18] **Прим. перекл.:** Graham Chapman *(ДК)*
[19] **Прим. перекл.:** ramdisk *(ДК)*

# Розділ 8

# init

*Uuno on numero yksi*

В цьому розділі описується процес `init`. Процес `init` є першим процесом користувацького рівня роботи, який стартується ядром. Він має багато важливих обов'язків, таких як стартування `getty` для того, щоб користувачі могли реєструватися в системі. `init` запускає робочі рівні в системі та піклується (всиновлює) про процеси, які стали сиротами. Цей розділ пояснює, як можна сконфіґурувати `init` та використати різноманітні можливості робочих рівнів.

## 8.1 `init` приходить першим

`init` - це одна з тих абсолютно необхідних програм, які життєво необхідні для роботи Лінакса. Але Ви можете абсолютно забути про `init` і не турбуватися про його роботу. Добре відлагоджена (куплена чи іншим чином придбана) версія Лінакса постачається з конфіґурацією `init`'а, яка буде добре працювати з більшістю систем. В таких системах робити будь-яку настройку для `init`'а абсолютно непотрібно. Про `init` згадують тільки тоді, коли через послідовний порт або модем треба підключити до системи термінал, щоб підключатися до своєї системи ззовні (але не для того, щоб дзвонити з свого комп'ютера на інший), або ж тоді, коли треба змінити стартовий робочий рівень.

Після того, як ядро стартувало себе (тобто прочитане в пам'ять, почало працювати та провело ініціалізацію драйверів та структур даних), воно (ядро) закінчує роботу над самим собою і завершує процес старту системи, запускаючи на виконання процес, що відноситься до користувацького рівня роботи [1] - процес `init`. Таким чином, процес `init` - це завжди самий перший процес користувацького рівня роботи (номер процесу `init`'а завжди є 1).

Ядро шукає `init` в кількох місцях - тих які в минулому використовувалися для нього. Але вірне місце для `init`'а в Лінаксі - це `/sbin/init`. Якщо ядро не може запустити `init`, воно намагається стартувати `/bin/sh`, і якщо навіть це не спрацьовує, система здається і її старт не відбувається.

Колии `init` починає свою роботу, він продовжує початий ядром процес завантаження. `init` виконує деякі адміністративні кроки, такі, як перевірка файлових систем, очистка `/tmp`, запускає різноманітні сервіси системи та стартує `getty` для кожного терміналу та віртуальної консолі, на яких дозволена робота користувачам (див. розділ 9).

[1] **Прим. перекл.:** user level program *(ДК)*

Після того, як із стартом системи закінчено, `init` слідкує за терміналами. Коли один з користувачів закінчує роботу на терміналі, `init` перезапускає `getty` на цьому терміналі, щоб в системі міг зареєструватися наступний користувач. Крім того `init` усиновлює процеси-сироти. Коли один процес стартує інший процес, то цей другий процес називається дитиною першого. Якщо процес-батько вмирає (закінчує свою роботу) раніше, ніж помре його дитина, процес `init` миттєво усиновлює цю дитину - вона стає дитиною `init`'а. Пояснення цього дуже технічне, але варто про це знати, оскільки це дає розуміння списків процесів і графів дерев процесів.[2]

Існує кілька різних варіатів `init`'а. Більшість Лінаксів користуються варіантом, відомим як `sysvinit` (написаний написаний Міґелем ван Смуренбурґом[3]. Він базується на дизайні `init`'а System V. BSD версії Юніксів мають інший `init`. Основна відмінність - в робочих рівнях. System V має робочі рівні, BSD Юнікси - ні. Ця різниця - несуттєва, але ми будемо розглядати тільки `sysvinit`.

## 8.2 Настройка `init` для старту `getty`: файл `/etc/inittab`

На початку роботи `init` читає конфіґураційний файл `/etc/inittab`. Під час роботи системи він не буде більше звертатися до цього файлу і перечитає його, тільки якщо йому послано сиґнал HUP[4]. Дякуючи цьому не потрібно перегружати систему, після змін в `/etc/inittab`.

Синтаксис файлу `/etc/inittab` трохи заплутаний, тому ми почнем з простого випадку конфіґурації ліній `getty`. Рядки в файлі `/etc/inittab` складаються з чотирьох полів розділених двокрапками:

*id:робочі рівні:дія:процес*

Поля описані трохи далі. Додатково `/etc/inittab` може мати пусті рядки та рядки, які починаються символом номера[5] - ('verb|#|'). Обидва типи таких рядків ігноруються.

**id** Це поле визначає назву рядка. Для рядків конфіґурації `getty` воно вказує термінал, де працює даний `getty` (літери, що йдуть після `/dev/tty` в назві спеціального файлу пристрою). В інших випадках це поле не має ніякого значення (крім хіба що обмежень по довжині). Але всі рядки в `/etc/inittab` повинні мати відмінні між собою поля.

**робочі рівні** Ті робочі рівні, на яких даний рядок треба виконувати. Робочі рівні задаються у вигляді однозначних чисел (цифр) без розділових знаків. (Робочі рівні описані в наступному розділі).

**дія** Визначає спосіб роботи даного рядка. Якщо в цьому полі вказано `respawn`, то програма буде запускатися заново кожного разу, коли вона закінчить роботу, якщо ж вказано `once` - програма виконається один раз.

**процес** Команда для виконання.

Для того, щоб `getty` стартував на першому віртуальному терміналі (`/dev/tty1`) на всіх нормальних багатокористувацьких робочих рівнях (2–5) слід додади до `inittab` такий рядок:

---

[2] `init`'у самому вмирати не дозволено. Його не можна вбити навіть сигналом SIGKILL.

[3] **Прим. перекл.:** Miquel van Smoorenburg *(ДК)*

[4] наприклад, якщо Ви як root виконаєте команду `kill -HUP 1`

[5] **Прим. перекл.:** В англійській мові. Українською можна було б сказати «знаком діез» *(ДК)*

```
1:2345:respawn:/sbin/getty 9600 tty1
```

Перше поле вказує, що це рядок для `/dev/tty1`. В другому кажеться, що команда повинна виконуватися на робочих рівнях 2,3,4 та 5. Третє поле вказує, що після закінчення роботи цієї програми, її треба перезапустити знову (тобто, користувач може зареєструватися в системі, вийти з системи і буде мати можливість зареєструватися знову). Останнє поле рядка виконує команду `getty` на першому віртуальному терміналі.[6]

Якщо Вам потрібно додати терміналів чи модемів в систему, Вам потрібно додати більше рядків в `/etc/inittab` - по одному рядку на кожен термінал чи модем. Як це зробити дивіться в підказках по *init*(8), *inittab*(5) та *getty*(8).

Якщо вказана в `inittab` програма не може стартувати вірно, вона помирає зразу ж після старту. `init` настроєний стартувати її знову. Програма стартує, вмирає, стартує знову, зупиняється... і т.д. Це займає страшенну кількість системних ресурсів. Тому `init` веде облік кількості разів, що та чи інша програма стартувала, і якщо це трапляється занадто часто, він затримує старт на п'ять хвилин.

## 8.3 Рівні роботи

**Робочим рівнем** називають стан, в якому перебуває `init` та система в цілому, і який визначає, які саме системні сервіси знаходяться в роботі. Робочі рівні визначаються номерами, див. таблицю **??**. Ще досі не прийшли до згоди, щодо того, як саме нумерувати робочі рівні від 2 до 5. Деякі системні адміністратори використовують робочі рівні для того, щоб визначити, які підсистеми працюють. Тобто: чи працює X, мережа і т.д. Інші стартують чи зупиняють підсистеми одну по одній без зміни робочих рівнів. Їм здається, що робочі рівні - це вже занадто для їхньої системи. Вам треба визначатися з цим самостійно. Або, може, навіть краще залишити все так, як воно вже є в тій версії Лінакса, що Ви маєте.

Табл. 8.1: Run level numbers

| | |
|---|---|
| 0 | Halt the system. |
| 1 | Single-user mode (for special administration). |
| 2–5 | Normal operation (user defined). |
| 6 | Reboot. |

Робочі рівні конфіґуруються в `/etc/inittab` такими рядками:

```
l2:2:wait:/etc/init.d/rc 2
```

Перше поле тут - це довільна етикетка. Друге показує, що цей рядок відноситься до другого робочого рівня. Третє поле вказує, що `init` повинен дочекатися, поки закінчиться виконання команди, даної в четвертому полі. В даному випадку команда `/etc/init.d/rc` виконує скрипт, який виконує всі команди (запускає необхідні демони та системи), які необхідно виконати для робочого рівня 2.

Команда в четвертому полі і виконує всю ту роботу, яка необхідна для встановлення робочого рівня. Вона встановлює сервіси, які ще не працюють, та зупиняє сервіси, які не повинні працювати на даному робочому рівні. Що це за

[6] Різні версії `getty` можуть працювати по різному, тому перш, ніж змінювати рядок, прочитайте сторінку підказки для даної команди. І саме головне - прочитайте її саме для тої команди, якою Ви користуєтесь.

команда, та як конфіґуруються різні робочі рівні, залежить від того, яку саме систему Ви маєте.

При старті `init` шукає в `/etc/inittab` рядок, який вказує основний робочий рівень[7]:

```
id:2:initdefault:
```

Можна вказати, що `init` повинен завантажити систему не в основний робочий рівень, а в якийсь інший. Це робиться за допомогою надання командного параметра для ядра, такого як `single` або `emergency`.[8] Це дає змогу ввійти в однокористувацький режим (робочий рівень 1), який описано в розділі 8.5.

Під час роботи системи робочий рівень можна змінити за допомогою команди `telinit`. При зміні робочого рівня, `init` виконує відповідні команди із файлу `/etc/inittab`.

## 8.4 Спеціальні конфіґурації в /etc/inittab

`/etc/inittab` має деякі спеціальні можливості, які дозволяють `init`'у реагувати на відповідні умови. Ці спеціальні функціональні можливості позначені спеціальним ключовим словом в третьому полі. Деякі приклади:

`powerwait` Команда, яка виконується `init`'ом в разі, коли пропадає напруга в мережі. Дає можливість безпечно вимкнути систему при раптовому зникненні напруги. Для цього повинен використовуватися резервний блок живлення (UPS), і програма, яка слідкує за UPS. Ця програма подає `init`'у сигнал, що напруга в мережі зникла.

`ctrlaltdel` Команда, яка виконується `init`'ом, коли користувач натискає `control-alt-del` на клавіатурі консолі. Зважте, що системний адміністратор, замість перегрузки, може приписати іншу дію для клавіш `control-alt-del`. Наприклад, просто ігнорувати - якщо система знаходиться в загально доступному місці.[9]

`sysinit` Команда, яка виконується під час завантаження системи. Ця команда найчастіше стирає `/tmp`.

Список, наведений вище, не виключний. Щоб дізнатися про всі можливості, дивіться сторінку підказки *inittab*(5) для свого Лінакса.

## 8.5 Завантаження в однокористувацький режим

Важливим робочим рівнем є **однокористувацький режим роботи** (робочий рівень 1). Звичайно тільки системний адміністратор може користуватися машиною в однокористувацькому режимі і працює рівно стільки системних сервісів, скільки це необхідно для підтримання системи в робочому стані. Однокористувацький режим потрібен для виконання кількох системних функцій,[10]. Наприклад, для виконання `fsck` на розділі `/usr` потрібно відмонтувати даний підрозділ, але цього не можна зробити, якщо не зупинені практично всі системні сервіси.

Працюючу систему можна перевести в однокористувацький режим за допомогою команди `telinit`, і вказати команді параметр 1. При завантаженні

[7] **Прим. перекл.:** default run level *(ДК)*

[8] Параметри ядру можна передати за допомогою LILO. Див. розділ 8.5

[9] Чи стартувати `nethack`.

[10] скоріше всього Ви не будете користуватися ним для гри в `nethack`

цього ж можна добитися, вказавши ядру ключове слово `single` або `emergency` в командному рядку ядра. Ядро, в свою чергу, передає цей командний рядок далі - до `init`'а. `init` зрозуміє, що систему не треба грузити в основний робочий режим. Як саме Ви передаєте командний рядок ядру залежить від того, як Ваша система завантажується.

Інколи необхідно завантажитись в однокористувацький режим, для того, щоб виконати команду `fsck` вручну, ще до того, як що-небудь змонтоване. Або тоді, коли що-небудь негаразд із файловою системою `/usr`. Будь-які дії над зіпсованою файловою системою приведуть тільки до того, що вона буде зіпсована ще більше - тобто `fsck` треба виконувати якомого раніше.

Якщо при завантаженні системи `fsck` видає серйозні помилки, `init` автоматично переведе систему в однокористувацький режим. Це робиться щоб запобігти використанню системи із зіпсованою файловою системою, яку `fsck` не спроможний відремонтувати автоматично. Випадки такого псування відносно рідкі і зв'язані із зіпсованим диском чи експериментальною версією ядра, але до них теж треба бути готовими.

Нормально настроєна система, завантажившись в однокористувацький режим, перш, ніж запустити командну оболонку, повинна спитати пароль користувача `root`. Без паролю дуже просто, надрукувавши простенький рядок в LILO, отримати доступ до всієї системи як `root`. (Але якщо файл `/etc/passwd` був пошкоджений при аварії файлової системи, Ви не зможете ввійти в свою систему, і на цей випадок краще мати під руками аварійну завантажувальну дискету.)

# Розділ 9

# Реєстрація в системі та вихід з системи

*This chapter needs a quote. Suggestions, anyone?*

Цей розділ описує те, що відбувається при реєстрації користувача в системі та виході з неї. В деталях описана також деяка взаємодія фонових процесів, файлів реєстрації[1] та файлів конфіґурації.

## 9.1 Реєстрація в системі через термінали

На рис. 9.1 показано, що саме відбувається при реєстрації користувачів з терміналу. Перш за все `init` перевіряє, чи є програма `getty` для з'єднання з терміналом (чи консоллю). `getty` сидить на терміналі і слухає, чекає поки користувач повідомить про те, що він готовий для реєстрації в системі (зміст цієї фрази просто означає, що користувач повинен надрукувати що-небудь). Коли `getty` помітить, що користувач ввів щось з клавіатури, він виводить на екран привітання з файлу `/etc/issue`, просить користувача ввести ім'я і врешті-решт запускає програму `login`. `login`'у передається параметром ім'я користувача і він, в свою чергу, попросить користувача ввести пароль для входу в систему. Якщо ім'я і пароль відповідають один одному, `login` передає управління командній оболонці, яка вказана для користувача в `/etc/passwd`, якщо ж ні - просто закінчує свою роботу (можливо давши користувачеві ще один шанс спробувати з паролем). Після цього `init` помічає, що термінал звільнився і стартує новий `getty` на цьому терміналі.

Відмітимо, що єдиний новий процес при цьому - це той, який створений `init`'ом (за допомогою системної функції `fork`), а `getty` та `login` тільки замінюють програму, що працювала до них (за допомогою системної функції `ecxec`).

Для того, щоб помітити вхід користувача в систему потрібна окрема програма на кожному послідовному порті, оскільки помітити активність на послідовній лінії може бути досить складно. Крім того, швидкість передачі та інші параметри, які дуже важливі для телефонних ліній, можуть змінюватися від дзвінка до дзвінка, і тому `getty` пристосовується до властивостей послідовного порта.

Існує кілька версій `getty` та `init`, кожна з яких має як свої позитивні, так і негативні якості. Дізнатйтеся про версії, встановлені саме в Вашій системі, але корисно знати також і про інші версії (можна скористуватися мапою програмного

[1] **Прим. перекл.:** log files *(ДК)*

забезпечення Лінакса для пошуку потрібних програм). Якщо Ви не користуєтесь телефонними каналами чи послідовними лініями зв'язку для під'єднання до інших систем чи Інтернету, то скоріше всього Вам нема про що турбуватися в відношенні `getty`, але про `init` все-таки корисно дещо знати.

## 9.2 Реєстрація в системі через мережу

Два комп'ютера в мережі, як звично, зв'язані одним фізичним кабелем. Коли вони спілкуються через мережу, програми, які приймають участь в спілкуванні, з'єднані через **віртуальне з'єднання**, яке можна можна вважати уявним кабелем. Поки програми знаходяться на обох кінцях з'єднання, вони тримають монополію на кабель. Однак, оскільки кабель всього-навсього уявний, а не справжній, то операційна система може мати насправді кілька уявних з'єднань при одному фізичному. Таким чином, маючи всього-навсього один кабель, кілька програм можуть спілкуватися, навіть не підозрюючи про інші з'єднання. Можливо навіть приєднати кілька комп'ютерів до одного і того-ж кабеля. Кожне віртуальне з'єднання встановлюється між двома комп'ютерами, але інші в цей час можуть не звертати уваги на існуюче з'єднання, оскільки вони не приймають в ньому участі.

Це - складне, але при тому ж занадто абстраговане відображення дійсності. Однак, його достатньо для того, щоб зрозуміти, чому реєстрація в системі через мережу трохи відрізняється від реєстрації з терміналу. Уявне з'єднання встановлюється тоді, коли на двох комп'ютерах є програми, які бажають встановити зв'язок. Оскільки в принципі можливо зареєструватися з будь-якого комп'ютера в мережі на будь-якому іншому, існує надзвичайна кількість потенційних уявних з'єднань. Через це непрактично стартувати `getty` для кожного потенційного користувача (тобто для кожної спроби реєстрації).

Існує єдиний процес inetd (аналогічний до `getty`, який займається *всіма* з'єднаннями в мережі. Коли він помічає вхідний запит про реєстрацію в мережі (тобто, він помічає, що якийсь процес іззовні просить про зв'язок з процесом на даному комп'ютері), `initd` стартує новий процес, який буде займатися цим одним єдиним запитом. Оригінальний процес залишається живим і продовжує слухати, чекаючи нових запитів на реєстрацію.

Щоб життя не здавалося малиною, існує кілька комунікаційних протоколів для реєстрації в мережі. Два найбільш важливих з них - це `telnet` та `rloign`. Додатково до реєстрації ще є багато інших типів умовних з'єднань (для FTP, Gopher, HTTP та для багатьох інших сервісів мережі). Неефективно мати по одному процесу на кожен тип з'єднання, які тільки слухають і чекають. Тож замість цього вживається всього один «слухач», який вміє розпізнавати тип з'єднання і запускати відповідну програму, яка буде забезпечувати потрібний сервіс. Цей «слухач» називається `inetd`. Більш повну інформацію про нього можна отримати з «Посібника по системній адміністрації мережі Лінакса».

## 9.3 Що робить `login`

Програма `login` займається перевіркою користувача, впевнюючись, що ім'я користувача та його пароль відповідають один одному. Після цього вона встановлює відповідне середовище для даного користувача, встановлюючи відповідні дозволи на послідовну лінію зв'язку та стартує командну оболонку для користувача.

Під час початкової установки `login` також виводить на екран невеличкий

файл `/etc/motd` (коротенька фраза дня[2]) і перевіряє електронну пошту. Ці функції можна заборонити створивши файл `.hushlogin` в домашній директорії користувача.

Якщо існує файл `/etc/nologin`, то реєстрація в системі заборонена. Звичайно такий файл створюється командами типу `shutdown`. `login` перевіряє існування такого файлу, і якщо він існує, він відмовляє в реєстрації користувачеві, але перш, ніж закінчити роботу, виводить зміст цього файлу на екран.

`login` записує (через `syslog`) всі спроби реєстрації в системі, які не вдалися, він також записує *всі* реєстрації в системі користувача `root`. Обидва типи записів можуть використовуватися для відслідковування спроб вторгнень в систему.

Всі користувачі, на даний момент зареєстровані в системі, записані в файлі `/var/run/utmp`. Цей файл дійсний тільки до тих пір, поки система не перевантажується або вимикається. Він перечислює всіх користувачів разом з терміналами (або з'єднаннями), на яких вони зареєстровані, разом з деякою додатковою інформацією. Команди `who`, `w` та інші звертаються до файлу `utmp` за інформацією.

Всі успішні реєстрації в системі записані в `/var/log/wtmp`. Цей файл може рости безмежно, отже його варто чистити періодично, встановивши, наприклад, щотижневий cron для цього.[3]. Переглядати файл `wtmp` можна за допомогою команди `last`.

Обидва файли - як `utmp`, так і `wtmp` - записані в двійковому форматі (див. підказку по *utmp*), і, на жаль, їх дуже незручно аналізувати без використання спеціальних команд.

## 9.4 X та xdm

## 9.5 Керування доступом

Традиційно база даних користувачів в Юніксі записувалася в файлі `/etc/passwd`. Деякі системи користуються **тіньовими паролями**[4]. В таких системах паролі користувачів записані в файлі `/etc/shadow`. [5]е завжди. SunOS записує тіньові паролі в `/etc/security/passwd.adjunct`. Організації з великими комп'ютерними мережами користуються NIS для розподіленого зберігання інформації про користувачів. Крім того, можна встановити автоматичне поширення інформації до всіх комп'ютерів в мережі з одного центрального сервера.

База даних користувачів містить в собі не тільки інформацію про паролі, а також деяку додаткову інформацію про користувачів, таку як їхні імена, домашні директорії та командні оболонки. Ця (інша) інформація повинна бути загально доступною, так, щоб кожен міг її прочитати. Отже, паролі треба зберігати в закодованому вигляді. При цьому є один суттєвий мінус - будь-хто може читати закодовані паролі і озброївшись криптографічними методами, може їх розкодувати. Тіньові паролі намагаються запобігти цьому. В системах з тіньовими паролями паролі записуються окремо від інших даних про користувачів, в файлі відкритому на читання тільки `root` (паролі все ще залишаються закодованими). Однак, встановлення тіньових паролів вже після встановлення системи може бути проблематичним.

---

[2] **Прим. перекл.:** message of the day *(ДК)*

[3] **Прим. перекл.:** Багато які з Лінаксів роблять це з самого початку автоматично *(ДК)*

[4] **Прим. перекл.:** shadow passwords *(ДК)*

[5] **Прим. перекл.:** Н *(ДК)*

З тіньовими паролями чи без них, завжди варто бути певним, що всі паролі в системі надійні, тобто, що їх не легко вгадати. Для злому паролів може використовуватися програма `crack`. Будь-який пароль вгаданий цією програмою вважається ненадійним. Програму `crack` можуть використовувати також і ті, хто намагається зломати паролі, але нею варто користуватися і системному адміністратору для того, щоб перевіряти стійкість паролів. До вживання добрих паролів може змусити також програма `passwd`. Це трохи ефективніше в термінах процесорного часу, оскільки зломка паролів вимагає досить таки інтенсивної роботи прооцесора.

База даних груп користувачів зберігається в файлі `/etc/group`. В системах з тіньовими паролями може існувати також файл `/etc/shadow.group`.

Як правило `root` не може зареєструватися в системі з віддаленого терміналу або з мережі. Це дозволяється тільки на терміналах, перелічених в файлі `/etc/securetty`. Для цього треба мати фізичний доступ до одного із цих терміналів. Однак можна зареєструватися в системі під іменем іншого користувача з будь-якого терміналу. Після цього, щоб стати `root`'ом треба виконати команду `su`.

## 9.6 Старт командної оболонки

Коли починає працювати інтерактивна командна оболонка, вона автоматично приводить в дію кілька попередньо визначених файлів. Різні оболонки використовують різні стартові файли під час свого старту. Проконсультуйтесь з документацією своєї оболонки, щоб визначити які саме файли використовуються в Вашому випадку.

Більшість оболонок на початку виконують деякий глобальний файл, наприклад оболонка Б'йорна `/bin/sh` виконує файл `/etc/profile`. Потім виконується файл `.profile` з домашньої директорії користувача. Файл `/etc/profile` дає можливість системному адміністратору встановити загальні для всіх користувачів системи властивості, такі, наприклад, як єдина доріжка для пошуку файлів для виконання PATH, особливо, якщо потрібно встановити якісь спеціальні маршрути пошуку файлів в системі додатково до стандартних. З іншого боку `.profile` дозволяє змінювати параметри свого середовища кожному окремому користувачеві, і якщо це необхідно, відміняти загальносистемні установки.

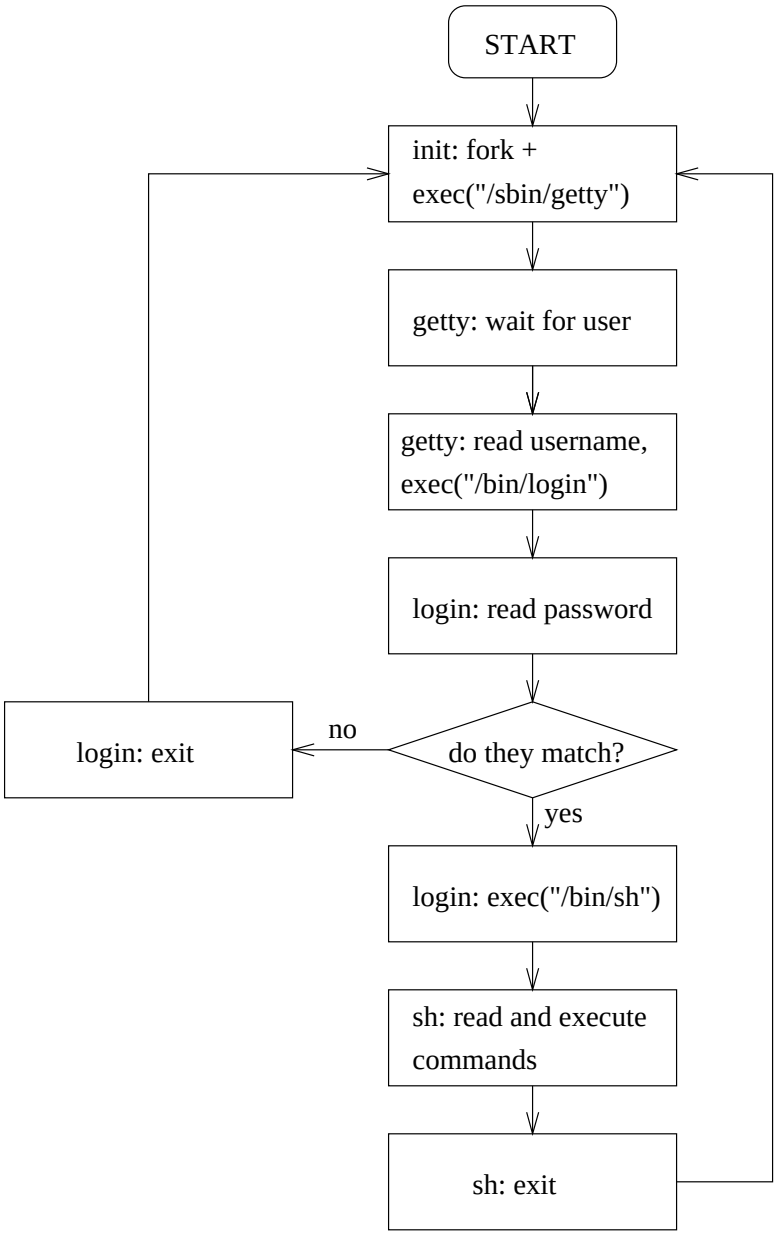


Рис. 9.1: Вхід в систему через термінал: взаємодія `init`, `getty`, `login` та командної оболонки.

# Розділ 10

# Створення та керування рахуками користувачів

*Чим схожі між собою системні адміністратори і торговці наркотиками: і ті й інші вимірюють свій товар в K, і ті й інші мають користувачів. (Старий комп'ютерний жарт з бородою).*

Цей розділ навчить Вас створювати рахунки користувачів[1], як змінювати властивості цих рахунків і як їх стирати. Різні версії Лінакса користуються різними засобами для цього.

## 10.1 Що таке рахунок?

Якщо комп'ютер використовується багатьма, необхідно ввести деякі відмінності між користувачами цього комп'ютера, з тою метою, щоб приватні файли різних користувачів не заважали особистим файлам інших. Навіть якщо комп'ютер використовується тільки одним користувачем кожного моменту часу, це варто зробити[2]. Тож кожному з користувачів присвоюється унікальне ім'я. Саме це ім'я використовується для реєстрації в системі.

Але одного імені, звичайно ж, не досить. **Рахунок** - це всі файли, ресурси та інформація, що відноситься до певного користувача. Термін цей подібний до того, який використовується в банках, а в банках до імені користувача ще прив'язуються певні фінанси асоційовані з цим іменем. Швидкість з якою фінанси витрачаються залежить від того, наскільки активно користувач використовує ресурси банку. Наприклад, дисковий простір може мати вартість, яка обраховується в меґабайтах на день, а обчислювальна потужність може обраховуватись посекундно.

## 10.2 Створення користувача

Для ядра Лінакса користувачі системи є просто числами. Кожен користувач має свій унікальний номер (ціле число), який називають **ID користувача**[3] або **uid**. Для комп'ютера набагато швидше і простіше мати справу з числами, ніж з текстовими іменами. База даних поза межами ядра призначає кожному ID

[1] **Прим. перекл.:** user accounts *(ДК)*

[2] Якщо моя сестра буде читати мої освідчення в коханні, мені не дуже це сподобається

[3] **Прим. перекл.:** user id *(ДК)*

користувача текстове ім'я, або **ім'я користувача**[4]. Ця база даних містить в собі також і іншу інформацію про користувачів.

Щоб створити користувача, Вам потрібно додати інформацію про цього користувача до бази даних користувачів і створити домашню директорію для нього. Напевно ще буде потрібно навчити користувача та створити робоче середовище для нього.

Більшість Лінаксів мають одну або іншу програму для створення рахунків користувачів. Дві таких програми, що працюють з командного рядка - це `adduser` та `useradd`. Можуть бути також інші варіанти, які працюють з віконцями та керуються мишою. Якою б програмою Ви не користувались, вручну залишається доробити дуже невелику частину роботи. Навіть якщо деталі роботи програм відрізняються, вони роблять додання нового користувача до системи тривіальною задачею. Однак, розділ 10.2.4 розповість, як можна робити все це вручну.

### 10.2.1 `/etc/passwd` та інші корисні файли

Основною базою даних з інформацією про користувачів в Юніксі є текстовий файл `/etc/passwd` (який називають файлом **паролів**[5]), в якому перелічені всі дійсні користувачі системи і та інформація, що відноситься до них. Інформація про кожного користувача записана на одному рядку в файлі, і поля в такому рядку розділені двокрапками. Кожен рядок має сім таких полів:

1. Ім'я користувача.
2. Зашифрований пароль.
3. ID користувача в числовому вигляді.
4. ID групи користувача в числовому вигляді.
5. Повне ім'я користувача та інша інформація про користувача.
6. Домашня директорія.
7. Командна оболонка (програма, яка виконується при вході в систему).

З більшимим подробицями формат файлу описаний в *passwd*(5).

Будь-який користувач системи може прочитати файл паролів. Отже, можна побачити повне ім'я користувача з цього файлу. Це означає також, що пароль (друге поле цього файлу) також доступний будь-якому користувачеві. Паролі зберігаються в закодованому вигляді, тож, теоретично, ніби-то проблеми немає. Але слабкою стороною систем криптоґрафії є те, що їх можна зламати, особливо якщо пароль погано вибраний. Наприклад, якщо пароль короткий або паролем вибране якесь слово із словника. Тож, тримати пароль в файлі паролів не рекомендується.

Багато Лінаксів мають **тіньові паролі**. Це - альтернативний спосіб зберігання паролів: закодований пароль зберігається в окремому файлі `/etc/shadow`, і цей файл доступний для читання тільки `root`'у. Файл `/etc/passwd` в цьому випадку містить тільки спеціальні показники на паролі в файлі `/etc/shadow` замість самих паролів. Таким чином тільки ті програми, які мають setuid можуть перевірити файл тіньових паролів, а всі інші програми можуть прочитати всі поля файлу `/etc/passwd`, але не мають доступу до файлу `/etc/shadow`[6]

[4]**Прим. перекл.:** username *(ДК)*

[5]**Прим. перекл.:** password file *(ДК)*

[6]Так-так, як виявляється файл паролів тепер має в собі всі дані про користувача *крім* самого пароля. Незбагненні шляхи програмного розвитку.

### 10.2.2 Вибір числового ID користувача та ID групи

Як правило, Ви можете не турбуватися про числове значення ID користувача та номер групи, але якщо Ви збираєтесь користуватися NFS (файловою системою мережі), Вам слід подбати про те, щоб користувачі мали одні й ті ж числові значення ID в усій мережі. Через те, що NFS також відрізняє користувачів по числовим значенням їхніх ID. Якщо в Вашій конфіґурації не присутні NFS сервери, можете покластися на програму в виборі значень ID.

Якщо Ви користуєтесь NFS, Вам треба винайти певний спосіб для синхронізації рахунків користувачів на різних системах. Один із способів полягає в використанні NIS (див. [**?**]).

**META:** this is wrong place? Однак не варто повторно використовувати одні й ті ж числа для ID (та для імен користувачів), бо нові користувачі можуть отримати доступ до файлів попередніх.

### 10.2.3 Робоче оточення: `/etc/skel`

Після створення нової домашньої директорії, ця директорія наповнюється файлами з директорії `/etc/skel`. Системний адміністратор може створювати файли в `/etc/skel` які будуть забезпечувати додаткову конфіґурацію для користувачів. Наприклад, адміністратор в файлі `/etc/skel/.profile` може встановити змінну EDITOR, яка буде встановлювати редактор, який більш зручний в користуванні для користувачів системи.

Однак, варто намагатися підтримувати `/etc/skel` мінімального розміру, оскільки після буде практично неможливо коригувати вже існуючі файли користувачів. Наприклад, якщо назва редактора змінюється, всі користувачі повинні будуть відредагувати свої `.profile` файли. Звичайно ж, адміністратор може намагатися зробити це за допомогою скрипта, тобто автоматично, але, напевне, це порушить логіку роботи чийогось файлу[7] цього не писав. Я всього навсього перекладач. Інколи в вільний від перекладів час, потроху займаютсь системною адміністрацією, за що і отримую гроші на свій хліб.

Взагалі керування робочими середовищами користувачів заслуговує на окрему книжку. Стандартів в цій області немає, ніхто не присвячує цій темі жодних ессе, мемуарів... А в цей час страждають беззахисні користувачі. Але частіше трапляється, що страждають від цього саме системні адміністратори, - хто його знає, які можуть бути користувачі? Взагалі кажучи «порушення логіки роботи файлу користувача» занадто сильний вираз, сила якого може в деяких випадках може переважувати вартість позиції системного адміністратора з усіма витікаючими звідси результатами.

Тож щоб порушути табу мовчання в цій області... Якщо Ваша організація має дуже багато користувачів (я маю на увазі « *дуже* багато»), то скоріше всього вони будуть логічно більш-менш ділимі на певні групи, тобто: «професори» та «студенти» чи «хакери» та «бугалтери і завгоспи». Тож можна замислитися над питанням: «А чи варто взагалі бугалтерам та завгоспам давати доступ до редагування їхніх «персональних» `.profile`'ів? Чи може створити для них один єдиний `/home/ENVIRON/ZAVHOSP/profile` і зробити `/.profile` посиланням (softlink) на цей файл. Для користувачів же з рівнем інтелекту вищим від середньо-загально-галузевого варто передбачити в `.profile` логіку типу: `[ -x ${HOME}/.profile-custom ] && { . ${HOME}/.profile-custom}`

Ще можу додати, що можливо директорія `/etc/skel` здається непоганою ідеєю, але, (з свого власного досвіду) тільки для систем від малих до середніх - від одного до пів-десятка комп'ютерів (можливо до десятка, якщо кожен з комп'ютерів має одного-двох постійних користувачів). Для більших систем

[7] **Прим. перекл.:** *Я (ДК)*

користування цією директорією виявляється непрактичним, якщо `/etc/skel` містить більше, ніж один-два файли і у переважній більшості випадків нехтується.

Коли це тільки можливо треба намагатися записувати загальні системні конфіґурації в загально-системні файли, такі, як `/etc/profile`. При цьому Ви можете змінювати системні установки без необхідності втручатися в файли користувачів.

### 10.2.4 Створення нових користувачів вручну

Щоб створити новий рахунок в системі вручну, виконайте таке:

1. Відредагуйте файл `/etc/passwd` за допомогою команди `vipw`. Будьте уважні з синтаксисом файлу. *Не редагуйте файл безпосередньо якимось редактором* Команда `vipw` замикає файл `/etc/passwd` так, що інші команди не можуть редагувати один і той же файл одночасно з Вами. В полі паролів файлу поставте '`*`' для того, щоб не можна було зареєструватися в системі з цим рахунком.
2. Якщо при створенні рахунку Вам треба також створити нову групу, відредагуйте файл `/etc/group` комадною `vigr`.
3. Створіть нову домашню директорію для користувача командою `mkdir`.
4. Скопіюйте файли з `/etc/skel` в нову домашню директорію.
5. Відкоригуйте права власності директорії командами `chown` та `chmod`. Найбільш корисним може бути використання параметру `-R`. Від системи до системи вірні права доступу трохи відрізняються, але звичайно наступна команда - це те що Вам треба:

```
cd /home/newusername
chown -R username.group .
chmod -R go=u,go-w .
chmod go= .
```

6. За допомогою команди `passwd` (1) встановіть пароль.

Після встановлення паролю цей рахунок буде працювати. Не варто встановлювати пароль до того, як закінчено все інше, бо перш, ніж Ви встигнете закінчити свою роботу, користувач може вже зареєструватися в системі.

Інколи необхідно створювати пусті рахунки[8], які не використовуються людьми. Наприклад, для встановлення сервера FTP з анонімним доступом (тобто таким, з якого будь-хто може звантажувати файли для себе, не маючи рахунку на цьому сервері), Вам необхідно створити рахунок, який називається *ftp*. В таких випадках немає необхідності встановлювати пароль (останній крок в попередніх інструкціях). Насправді, якщо пароль для такого рахунку не встановлено, то ніхто не зможе зареєструватися в системі з таким іменем (щоб перейти в цей рахунок, звичайному користувачеві треба спочатку стати `root`, оскільки `root` може ставати будь-яким користувачем.

## 10.3 Зміна властивостей рахунків користувачів

Є кілька команд, які можуть змінювати різні властивості рахунків (тобто певні поля в файлі `/etc/passwd`):

[8] Неіснуючі користувачі?

`chfn` Змінює повне ім'я користувача.

`chsh` Змінює командну оболонку користувача.

`passwd` Змінює пароль користувача.

Системний адміністратор цими командами може змінювати рахунок будь-якого користувача. Звичайні користувачі можуть змінювати тільки свої власні рахунки. Якщо потрібно заборонити використання цих команд для звичайних користувачів, наприклад, в системі, де дуже багато початківців, це можна зробити за допомогою команди `chmod`.

Все інше потрібно робити своїми власними руками. Наприклад, для того, щоб змінити ім'я користувача, треба відредагувати файл `/etc/passwd` (за допомогою команди `vipw` - пам'ятайте про це!). Так само, для того, щоб додати користувачів до інших груп, чи, навпаки, виключити з певних груп, потрібно змінити файл `/etc/group` (за допомогою `vigr`). Деякі з цих подій трапляються досить рідко, і виконувати їх треба дуже обережно. Наприклад, після зміни імені користувача, стара адреса електронної пошти також перестане працювати для цього користувача, якщо Ви не створите псевдо для нової адреси.[9]

## 10.4 Виключення користувача зі списку

Щоб виключити користувача, треба спочатку стерти всі його файли, поштові скрині, поштові псевда, роботи, що знаходяться в друці, роботи `cron`, `at` та всі інші посилання на цього користувача. Після цього зітріть відповідний рядок з `/etc/passwd` та `/etc/group` (не забудьте стерти ім'я користувача з усіх інших груп також). Можливо, спершу варто заборонити користування рахунком (див. далі) перш, ніж приступати до стирання файлів, щоб користувач не міг користуватися рахунком в той час, як Ви його стираєте.

Пам'ятайте також, що файли, що належать певному користувачеві, можуть знаходитися також поза межами його домашньої директорії. Їх можна відшукати за допомогою команди `find`.

```
find / -user ім'я користувача
```

Запам'ятайте однак, що ця команда буде працювати *дуже* довго на великих дисках. Якщо на Вашій системі змонтовані також диски з мережі (див. розділ 3.3.8), то будьте обережні, щоб не відправити в смітник всю мережу або сервер.

Деякі Лінакси мають спеціальні команди для виконання описаної дії. Пошукайте в системі `deluser` або `userdel`. Однак, все це дуже легко робиться вручну, і крім того команда може не робити всього, так, як треба.

## 10.5 Тимчасова заборона користування рахунком

Інколи буває потрібно тимчасово зробити рахунок недіючим, не знищуючи його зовсім. Наприклад, може користувач не сплачує за користування рахунком, або системний адміністратор може запідозрити, що пароль користувача зломаний.

Найкращий спосіб заборонити користування рахунком, це змінити командну оболонку такого користувача на програму, яка буде всього-навсього друкувати повідомлення. При такому підході будь-хто, пробуючи зареєструватися в системі відразу ж зрозуміє, чому він не може цього зробити. Програма може

[9]Можливо користувач змінив(ла) ім'я після одруження, і хоче зареєструватися з новим іменем.

повідомляти користувачеві, що він має звернутися до системного адміністратора щоб виправити проблему.

Крім цього можливо також змінити ім'я користувача чи пароль, але в цьому випадку користувач не знатиме, в чому справа. Збити з пантелику користувача означає принести більше клопоту самому собі.[10][11] а жаль не вказаний тут *стандартний* метод заборони користування рахунком в Юніксах: якщо попереду пароля в `/etc/passwd` поставити зірочку («*»), то ніхто не зможе зареєструватися в системі з цим іменем. Коли Вам треба буде знову дозволити користування цим рахунком, Вам просто треба буде стерти цю зірочку. Особисто мені це уявляється трохи кращим, ніж запам'ятовувати оболонку, яку вживав користувач до заборони рахунку і повертати все на свої місця пізніше. Адже після відміни заборони можна помилитися і записати в цьому полі не ту оболонку, яка потрібна користувачеві і порушити тим самим купу скриптів користувача. (Згадайте, що писалося про збитого з пантелику користувача і уявіть собі, що можна написати про збитого з пантелику системного адміністратора.) Крім того, чи варто писати користувачеві, що він має звертатися до системного адміністратора, якщо його рахунок не працює, - а куди ж йому бідоласі ще йти?!

Дуже прості програмки, які видають на екран повідомлення - це скрипти 'tail'[12]:

```
#!/usr/bin/tail +2
This account has been closed due to a security breach.
Please call 555-1234 and wait for the men in black to arrive.
\intnote{Цей рахунок закрито через небезпеку злому. Зателефонуйте по
номеру 555-1234 і чекайте на чоловіка в чорному.}
```

Перші два символи ('`#!`') повідомляють ядру, що залишок рядка є командою, яку треба виконати над цим файлом. Команда `tail` в цьому випадку надрукує на стандартний вивід все крім першого рядка.

Якщо рахунок `biilg` підозрюється в небезпеці злому, системному адміністратору слід зробити щось таке:

```
# chsh -s /usr/local/lib/no-login/security billg
# su - tester
This account has been closed due to a security breach.
Please call 555-1234 and wait for the men in black to arrive.
#
```

Команда `su` в цьому прикладі звичайно ж використовується для перевірки, що все працює так, як треба.

Такі скрипти треба тримати в окремій директорії, щоб назви скриптів не перемішувалися із звичайними командами користувачів.

---

[10] Але це може бути *таке* задоволення, якщо Ви BOFH.

[11] **Прим. перекл.:** H *(ДК)*

[12] **Прим. перекл.:** tail - хвіст *(ДК)*

# Розділ 11

# Резервні копії

*Апаратура непередбачувано надійна.*
*Програмне забезпечення непередбачувано ненадійне.*
*Люди непередбачувано ненадійні.*
*Природа непередбачувано надійна.*

Цей розділ пояснює чому, як і коли робити резервне копіювання, і як відновлювати загублені речі з резервних копій.

## 11.1 Про важливість мати резервні копії

Ваші дані важливі. Їх відновлення буде варте Вашого часу і зусиль, а це - варте грошей, або, як мінімум, власниих переживань та сліз. Деякі дані неможливо створити заново, наприклад, якщо це - результати якогось експерименту. Оскільки це - вкладення капіталу, Ви маєте їх захищати та намагатися не загубити.

Ви можете загубити свої дані в основному з чотирьох причин: зіпсована апаратура, помилки в програмах, дії людей та природні лиха.[1] Хоча сучасна апаратура стає все більш надійною, вона все таки може псуватися, як інколи здається, несподівано. Найбільш важливі для збереження даних пристрої - це жорсткі диски, які залежать від невидимих, крихітних магнітних полів і живуть в світі переповненому електромагнітними шумами. Сучасні програми навіть і не намагаються бути надійними. Залізобетонно-надійні програми в сучасному світі - це виключення, а не правило. Люди досить ненадійні, вони або роблять помилки, або, будучи підлими по натурі, просто навмисне псують дані. Природа, може навіть і не диявольска по своїй натурі, але може приносити сюрпризи навіть коли вона чудова та лагідна. Беручи все це до уваги, залишається тільки дивуватися, як все оце ще досі працює.

Створення резервних копій - це шлях до збереження вкладеного в створення даних капіталу. Маючи кілька резервних копій Ви можете не турбуватися про те, скільки даних (який їх відсоток) знищено. Вартість повернення даних буде всього навсього вартістю відновлення даних з резервної копії.

Найбільш важливим є робити резервування даних вірно. Так само, як і на все інше, на це впливає повсякденне життя, і тому рано чи пізно резервна копія зіпсується. Частиною роботи з резервування є перевірка, що резервні копії працюють. Напевно б під час відновлення даних Вам не хотілося дізнатися, що Ваше резервування не працювало[2] Навіть більше того, найжахливіший крах

[1] Ще є п'ята причина - «щось інше».
[2] Не смійтеся - це сталося не з однією людиною.

може статися якраз під час створення резервних копій. Якщо у Вас є один носій для резервування, може виявитися, що цей носій зіпсований, і Ви опиняєтесь на попелищі, залишеному на місці важкої роботи.[3] Або, відновлюючи що-небудь з резервної копії, може раптом виявитися, що щось дуже важливе, щось таке, наприклад, як база даних сервера на 15 000 користувачів, залишилося не заархівованим. І що ще краще, всі Ваші копії можуть працювати бездоганно, але саме той привід стрічки, яким Ви користувалися, був єдиним в своєму роді і стоїть тепер наповнений по вінця водою.

Коли Ви починаєте планувати створення резевних копій, то паранойя є необхідним станом душі.

## 11.2 Вибір носіїв для створення резервних копій

Найважливіше рішення, що стосується резервування даних - це вибір носіїв для створення резервних копій. Вважати треба на вартість, надійність, швидкість та зручність користування.

Вартість важлива, оскільки, напевне у Вас буде резервних копій по об'єму в кілька разів більше, ніж даних. Слід розраховувати на дешеві носії.

Надійність надзвичайно важлива - зіпсована резервна копія може довести мужнього героя до сліз. Носії резевних копій повинні зберігати дані без втрат по кілька років. Спосіб їх використання також впливає на їх надійність. Жорсткий диск - дуже надійний. Але якщо диск використовується як носій для резервного копіювання - він не дуже надійний, бо він знаходиться в тому ж комп'ютері, що і основний диск.

Швидкість, як правило, не дуже важлива, якщо тільки Ви можете зробити все потрібне резервування без переривання. Нічого страшного, що запис резервної копії займає дві години, якщо при цьому не потрібно Ваше втручання. Але, якщо Ви не встигаєте робити резервування за той час, поки комп'ютер не завантажений справжньою роботою, швидкість набуває вагу.

Наявність носіїв та пристроїв - очевидна необхідність, - Ви не зможете зробити резервну копію, якщо таких носіїв, як потрібно, не існує. Але наявність вибраного пристрою в майбутньому є не настільки очевидою вимогою (і саме для того комп'ютера, що Ви маєте). Не маючи потрібного пристрою Ви просто не зможете відновити дані після катастрофи.

Зручність в користуванні є надзвичайно важливим фактором у визначенні того, як часто Ви створюєте резервні копії. Чим простіше це робити, тим краще. Користування носіями не повинно бути важким чи втомлюючим.

Найбільш типові варіанти носіїв для резервування - це дискети та магнітні стрічки. Дискети дуже дешеві, досить надійні і є в наявності. Але для великих об'ємів - незручні в користуванні. Стрічки по вартості лежать в діапазоні від дешевих до дещо дорогших, досить надійні, досить швидкі і достатньо доступні. Вважаючи на їх об'єм, стрічки також досить зручні в користуванні.

Є також інші варіанти для вибору. Інколи вони не настільки доступні, але коли вибір не є проблемою, вони можуть бути кращими в інших відношеннях. Наприклад, магнітно-оптичні диски мають позитивні якості, успадковані як від дискет - (довільний доступ до даних дозволяє швидко відшукати потрібний файл), так і від стрічок (можуть зберігати величезну кількість даних).

[3] Вже тут був, і вже це бачив...

## 11.3 Вибір засобів для створення резервних копій

Для створення резервних копій існує безліч засобів. Традиційні засоби Юнікса - це команди `tar`, `cpio` та `dump`. Додатково до цього існують інші пакети (як безкоштовні, так і комерційні). На вибір конкретного засобу впливає також вибір носія.

`tar` та `cpio` - подібні між собою, і з точки зору резервного копіювання майже однакові. Обидві програми можуть зберігати дані на стрічках та читати їх звідти. Обидві можуть користуватися практично будь-якими носіями для запису, оскільки драйвери низького рівня в ядрі піклуються про роботу з пристроями і всі пристрої виглядають практично однаково для програм рівня користувача. Деякі версії традиційних Юніксовських `tar` та `cpio` можуть мати деякі проблеми з незвичайними файлами (символічними ссилками, спеціальними файлами пристроїв, файлами з занадто довгими назвами, тощо [4]наприклад, `tar` в SunOS не може копіювати файли спеціальних пристроїв (`/dev/*`) ), але Лінаксівські версії поводяться з усим цим так, як треба.

`dump` відрізняється тим, що він читає безпосередньо файлову систему , а не файли через файлову систему. Крім того він написаний спеціально для створення резервних копій, в той час, як `tar` та `cpio` насправді створені для архівування даних, хоча і можуть робити резервні копії також.

Звертання прямо до файлової системи має деякі переваги. Це робить можливим архівування файлів без зміни їх дат, щоб досягти цього ж за допомогою `tar` та `cpio` Вам потрібно було б спочатку монтувати файлову систему в режимі тільки читання. Також пряме читання з файлової системи ефективніше, тоді, коли потрібно створити резервну копію всієї файлової системи, оскільки менше часу витрачається на перевід магнітної головки. Основним недоліком є залежність від типу файлової системи - кожна програма розуміє одну єдину файлову систему. Лінаксівська версія `dump` може працювати тільки з файловою системою ext2.

`dump` також підтримує безпосередньо рівні створення резервних копій (про які буде трохи далі); при використанні команд `tar` та `cpio` це треба робити за допомогою інших засобів.

Порівняння інших засобів для створення резезвних копій виходить поза межі тематики цієї книжки. Багато засобів приведено на Linux Software Map.

## 11.4 Просте резервування

Найпростіша схема резезвування - це зарезервувати все абсолютно за один раз, а потім резервувати все, що змінилося з часу попереднього резервування. Перше резервування носить назву **повного резервування**[5], наступні після нього — «доповнюючі копії»[6]. Повна копія частіше вимагає більше роботи, ніж доповнюючі, через те, що потрібно записати більше даних на стрічку, і, можливо, що всі дані не помістяться на одну стрічку (або дискету). Відновлення файлів з доповнюючих копій може вимагати набагато більшої роботи, ніж відновлення з повної копії. Відновлення файлів можна оптимізувати таким чином, щоб відновлені файли завжди бралися з останньої повної копії. Таким чином, можливо резервування вимагає трохи більше роботи, але, може трапитися, що Вам ніколи не буде потрібно відновлювати файл з повної копії і доповнювати його з доповнюючої.

[4]**Прим. перекл.:** H *(ДК)*
[5]**Прим. перекл.:** full backup *(ДК)*
[6]**Прим. перекл.:** incremental backups *(ДК)*

Якщо Ви плануєте робити резервні копії кожного дня і маєте шість стрічок для цього, Ви можете використати стрічку 1 для першої повної копії (скажімо, в п'ятницю), і стрічки 2- 5 для доповнюючої копії (від понеділка до четверга). Потім Ви робите нову повну копію на стрічці 6 (другої п'ятниці) і починаєте робити доповнюючі копії з стрічками 2-5 знову. Намагайтеся не стерти стрічку 1 до того, часу поки Ви не будете мати нової повної копії, на той випадок, якщо щось трапиться під час створення другої повної копії. Навіть після того, як Ви зробили другу повну копію, тримайте першу копію десь в безпечному місці, раптом всі наступні копії будуть знищеними (від вогню, чи ще від чогось), Ви все-таки матимете щось в руках. Коли Вам потрібно буде зробити наступну повну копію, робіть її на стрічці 1, а стрічку 6 відкладіть.

Якщо Ви маєте більше, ніж 6 стрічок, Ви можете використовувати додаткові з них для інших повних копій. Кожного разу, коли Ви робите нову повну копію, використовуйте найстарішу стрічку. Таким чином, Ви можете мати повні копії за кілька попередніх тижнів, що дуже добре, якщо Вам потрібно відшукати якийсь старий файл, що Ви вже його стерли, або стару версію файлу.

### 11.4.1 Створення повних копій з tar

Повна копія може дуже просто бути створена командою **tar**:

```
# tar -create -file /dev/ftape /usr/src
tar: Removing leading / from absolute path names in the
archive
#
```

Зразок команди поданий тут використовує GNU версію **tar** і прийняті для цієї версії довгі командні опції. Традиційні версії **tar** розуміють тільки опції, що складаються з однієї літери. Версія GNU також може створювати копії, що не поміщаються на одну стрічку чи дискету, та такі, які мають файли з дуже довгими маршрутами. Не всі традиційні версії вміють це робити. Лінакс користується тільки GNU версією **tar**.

Якщо Ваша копія не поміщається на одну стрічку, треба користуватися опцією для створення багатотомних архівів `-multi-volume` (`-M`):

```
# tar -cMf /dev/fd0H1440 /usr/src
tar: Removing leading / from absolute path names in the
archive
Prepare volume #2 for /dev/fd0H1440 and hit return:
#
```

Відмітьте, що Ви маєте відформатувати дискети перш, ніж робити копіювання, або ж інакше скористуватися іншим вікном чи віртуальним терміналом для форматування, коли **tar** запитає про наступну дискету.

Після того, як Ви закінчили робити копіювання перевірте правильність зробленої копії, користуючись опцією `-commpare` (`-d`):

```
# tar -compare -verbose -f /dev/ftape
usr/src/
usr/src/linux
usr/src/linux-1.2.10-includes/
....
#
```

Якщо Ви не перевірите зроблену копію, то можливо Ви не знатимете, що Ваша копія не працює аж до того часу, поки Ви не втратите важливі дані.

Доповнюючі копії можна створювати за допомогою **tar** та опції `-newer` (`-N`):

```
# tar -create -newer '8 Sep 1995' -file /dev/ftape /usr/src
-verbose
tar: Removing leading / from absolute path names in the
archive
usr/src/
usr/src/linux-1.2.10-includes/
usr/src/linux-1.2.10-includes/include/
usr/src/linux-1.2.10-includes/include/linux/
usr/src/linux-1.2.10-includes/include/linux/modules/
usr/src/linux-1.2.10-includes/include/asm-generic/
usr/src/linux-1.2.10-includes/include/asm-i386/
usr/src/linux-1.2.10-includes/include/asm-mips/
usr/src/linux-1.2.10-includes/include/asm-alpha/
usr/src/linux-1.2.10-includes/include/asm-m68k/
usr/src/linux-1.2.10-includes/include/asm-sparc/
usr/src/patch-1.2.11.gz
#
```

На жаль `tar` не може визначити, коли інформація, що міститься в inode змінилася, наприклад, коли біти дозволів на файл змінилися, або, навіть, якщо назва файлу змінилася. Ці проблеми можна обійти, якщо скористуватися командою `find` та порівняти поточну файлову систему із списком заархівованих файлів. Деякі програми та скрипти для Лінакса, які вміють це робити можна знайти на різних ftp серверах.

### 11.4.2 Відновлення файлів за допомогою tar

Файли з архіву відновлюються за допомогою ощії `-extract` `tar`'у:

```
# tar -extract -same-permissions -verbose -file /dev/fd0H1440
usr/src/
usr/src/linux
usr/src/linux-1.2.10-includes/
usr/src/linux-1.2.10-includes/include/
usr/src/linux-1.2.10-includes/include/linux/
usr/src/linux-1.2.10-includes/include/linux/hdreg.h
usr/src/linux-1.2.10-includes/include/linux/kernel.h
...
#
```

Крім того, якщо Ви вкажете назви файлів (чи директорій) в командному рядку, `tar` розархівує тільки вказані файли та директорії (разом з усіма файлами та піддиректоріями всередині них):

```
# tar xpvf /dev/fd0H1440
usr/src/linux-1.2.10-includes/include/linux/hdreg.h
usr/src/linux-1.2.10-includes/include/linux/hdreg.h
#
```

Якщо Ви хочете переглянути, які файли маються в архіві, користуйтеся опцією `-list` (`-t`):

```
# tar -list -file /dev/fd0H1440
usr/src/
usr/src/linux
usr/src/linux-1.2.10-includes/
usr/src/linux-1.2.10-includes/include/
usr/src/linux-1.2.10-includes/include/linux/
```

```
usr/src/linux-1.2.10-includes/include/linux/hdreg.h
usr/src/linux-1.2.10-includes/include/linux/kernel.h
...
#
```

Майте на увазі, що `tar` завжди читає створені архіви послідовно - від початку до кінця. Тобто, з великими архівами він працює досить повільно. Але користуватися довільним доступом для стрічок чи інших пристроїв з послідовним доступом неможливо.

`tar` не може вірно поводитися із стертими файлами. Якщо Вам потрібно відновити файлову систему з повної та доповнюючої копії, і Ви стерли кілька файлів у проміжку між створенням двох копій, вони знову будуть існувати після того, як ви відновите файлову систему. Це може бути великою проблемою, якщо у файлі містяться такі дані, які не повинні вже бути присутніми.

## 11.5 Багаторівневі копії

Просте резервування описане попередньо достатнє в більшості випадків для персонального вжитку, або для невеликих центрів. Для більш «важкого» застосування необхідне вживання багаторівневого резервного копіювання.

Простий метод має два рівні копіювання: повний та доповнюючий. Це поняття можна узагальнити на довільну кількість рівнів. Повна копія буде носити назву копії рівня 0, доповнюючі копії називатимуться копіями рівнів 1, 2, 3, ... При кожному доповнюючому копіюванні, Ви архівуєте все, що змінилося з часу попередньої копії такого-ж або попереднього рівня.

Призначення такої системи резервування в тому, що вона дозволяє довшу **історію резервування** [7] дешевими засобами. В попередньому прикладі історія резервування йшла назад до часу попереднього повного резервування. Її (історію) можна подовжити тільки за рахунок придбання додаткових стрічок, по одній стрічці на тиждень, але це може бути занадто дорого. Довша історія копій корисна, оскільки зіпсовані чи зниклі файли не завжди виявляються швидко. Тому, навіть версія файлу, що трохи старіша, ніж хотілося б, краще, ніж ніякого файлу зовсім.

З багаторівневим копіюванням історію резервування можна подовжити дешевшими засобами. Наприклад, якщо купити десять стрічок, то можна використовувати стрічки 1 та 2 для щомісячних копій (перша п'ятниця кожного місяця), стрічки з 3 по 6 для щотижневих копій (інші п'ятниці; Відмітьте, що місяць може мати п'ять п'ятниць. Тому нам потрібно чотири стрічки), а стрічки з 7 по 10 для щоденних копій (з понеділка до четверга). Всього з чотирма новими стрічками ми, таким чином, змогли подовжити глибину резевних копій від двох тижнів (після того, як всі денні стрічки використані) до двох місяців. Звичайно, ж ми не зможемо відновити будь-яку версію будь-якого файлу на протязі цих двох місяців, але те, що можна відновити, часто виявляється достатнім.

Рисунок 11.1 показує, який рівень резервування використовується кожного дня, і які копії можуть бути відновленими в кінці місяця.

Рівні резервування можна також використовувати для зменшення часу відновлення файлової системи до мінімуму. Якщо Ви маєте батато доповнюючих копій із стало зростаючими номерами, Вам потрібно відновити всі з них, щоб повністю відновити файлову систему. Замість цього, якщо скористатися номерами рівнів резервування, які не монотонні, можна скоротити кількість копій, які потрібні для відновлення всієї файлової системи.

Щоб зменшити кількість необхідних для повного відновлення стрічок, треба користуватися меншими номерами для кожного наступного доповнюючого

[7] **Прим. перекл.:** backup history *(ДК)*

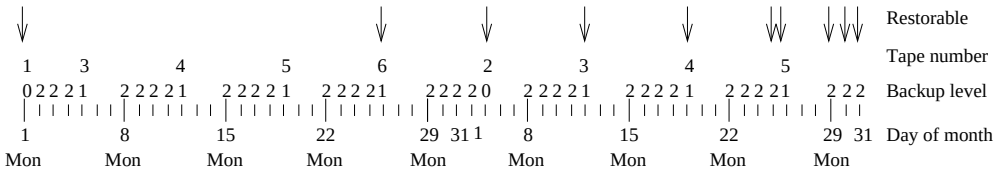


Рис. 11.1: Проста багаторівнема схема резервування.

резервування. Однак, в цьому випадку, час для створення кожної такої копії збільшується (кожне резервування копіює все з часу попереднього повного резервування). Сторінка підказки до команди `dump` подає кращу схему резервування, яка подана в таблиці 11.2. Користуйтеся такою послідовністю стрічок: 3, 2, 5, 4, 7, 6, 9, 8, 9... Така схема утримує час створення копій і відновлення на мінімумі. Максимум, що Вам потрібно резервувати, це результати дводенної роботи. Кількість стрічок для відновлення залежить від того, наскільки довго часу проходить між повними копіюваннями, але ця кількість менша, ніж при використанні простих послідовнестей.

Рис. 11.2: Ефективна схема резервування з використанням багаторівневого копіювання

| Tape | Level | Backup (days) | Restore tapes |
|---|---|---|---|
| 1 | 0 | n/a | 1 |
| 2 | 3 | 1 | 1, 2 |
| 3 | 2 | 2 | 1, 3 |
| 4 | 5 | 1 | 1, 2, 4 |
| 5 | 4 | 2 | 1, 2, 5 |
| 6 | 7 | 1 | 1, 2, 5, 6 |
| 7 | 6 | 2 | 1, 2, 5, 7 |
| 8 | 9 | 1 | 1, 2, 5, 7, 8 |
| 9 | 8 | 2 | 1, 2, 5, 7, 9 |
| 10 | 9 | 1 | 1, 2, 5, 7, 9, 10 |
| 11 | 9 | 1 | 1, 2, 5, 7, 9, 10, 11 |
| ... | 9 | 1 | 1, 2, 5, 7, 9, 10, 11, ... |

Гарна схема, звичайно ж, може зменшити кількість роботи, але це означає, що є багато речей, про які треба пам'ятати. І Ви повинні вирішити, чи ця схема варта того.

`dump` підтримує багаторівневе резервування. Для `tar` та `cpio` це треба робити за допомогою скриптів.

## 11.6 Що архівувати

Звичайно ж треба створювати резервні копії з якнайбільшої кількості файлів. Основні виключення - це програми, які можна легко перевстановити,[8] але навіть

[8]Ви повинні самі вирішувати, що означає «легко» для Вас. Для деяких людей перевстановлення з десятків дискет - легко

ці програми можуть мати файли конфіґурації, які треба архівувати, якщо Ви не бажаєте робити всю конфіґурацію з самого початку. Інше основне виключення з схеми резервування - це файлова система `proc`, оскільки вона завжди має тільки ті дані, що автоматично зґенеровані ядром системи під час роботи, і архівувати їх *завжди* було і буде нездоровою ідеєю. Особливо непотрібен файл `/proc/kcore`, оскільки це - всього лиш образ оперативної пам'яті комп'ютера, і до того ж досить великий образ.

Сірі області - це новини, лоґ-файли (файли реєстрації) та багато чого в `/var`. Ви повинні вирішити, що з них важливе.

Явні речі, які потребують архівації - це файли в домашніх директоріях користувачів (`/home`) та системні файли конфіґурації в `/etc`, але можливо інші файли можуть бути розкиданими по всіх файлових системах.

## 11.7 Резервування з компресією

Резервування займає прірву місця, яке, в свою чергу, може коштувати прірву грошей. Щоб зменшити об'єм зарезервованих даних можна скористуватися програмами для компресування даних. Є кілька шляхів для цього. Деякі програми самостійно можуть підтримувати компресування даних; наприклад, опція `-gzip` (`-z`) для GNU `tar` відправляє всі дані через канал (ріре) до програми компресії `gzip`, перш, ніж записати дані на носій.

На жаль, архівування з компресією може спричинити проблеми. Механізм роботи програм компресії приводить до того, що, якщо хоча б один біт в архіві псується і стає невірним, то і весь архів стає невірним. Деякі програми компресії мають вбудовані механізми корекції помилок, але не існує такого методу, який би міг справитися з великою кількістю помилок. Це означає, що якщо резервна копія скомпресована тим чином, яким це робить GNU `tar`, тобто коли весь архів компресується як одне ціле, то єдина помилка в архівації може привести до того, що вся копія буде загублена. Резервні копії мають бути надійними, і тому такий метод компресії не дуже гарна ідея.

Альтернативою може бути компресування файлів поодинці. Це означає, що якщо навіть один файл втрачено, всі інші будуть цілі. Загублений файл міг би бути зіпсованим все одно, навіть без компресії, тому цей підхід до архівування не набагато гірший від того, коли компресія не використовується зовсім. Програма `afio` (варіант команди `cpio`) вміє це робити.

Компресування займає деякий час, що може привести до того, що програма архівування не зможе записувати дані на стрічку з потрібною швидкістю.[9] Цього можна позбутися, якщо пропускати дані через буфер (або внутрішній, якщо програма достатньо для цього розумна, або використовуючи іншу програму), але навіть в цьому випадку архівація може не працювати, так як треба. Це може, однак, бути проблемою тільки на повільних комп'ютерах.

[9] Якщо привід стрічки не отримує дані з достатньою швидкістю, він повинен зупинитися, це призводить до того, що архівування стає ще повільнішим, і, в той же час може бути поганим для приводу і для стрічки.

# Розділ 12

# Підримка вірного часу в системі

*Час - це іллюзія. Обідній час - іллюзія вдвічі.*
*(Дуґлас Адамс.)*

Цей розділ пояснює як система з Лінаксом підтримує час, і що Вам потрібно для того, щоб не мати проблем. В більшості випадків Вам не потрібно робити нічого стосовно підтримки часу, але тим не менше непогано було б розуміти, що відбувається.

## 12.1 Часові пояси

Вимір часу базується на періодичному природньому явищі, такому, як чередування світлих та темних періодів, що викликаються обертанням планети. Загальний час, який складається з суми світлого та темного періодів завжди постійний, хоча довжина дня та ночі змінюються. Одна постійна константа - це полудень.

Полудень - це такий час, коли сонце знаходиться в своїй найвищий точці - апогеї. Оскільки Земля кругла,[1] полудень трапляється в різний час в різних куточках планети. Це приводить до поняття **місцевого часу**. Люди вимірюють час в різних одиницях, більшість з яких так або інакше прив'язані до природніх явищ. І до того часу, поки Ви знаходитесь в одному місці, для Вас не має ніякого значення, що місцевий час відрізняється від іншого місцевого часу.

Як тільки у Вас виникає потреба спілкуватися з людьми, що знаходяться на значній відстані від Вас, Вам потрібно відшукати спільну точку виміру часу. В наш час люди різних країн спілкуються між собою, тому були вироблені глобальні стандарти часу. Такий час носить назву **універсальним часом** (universal time, UT або UTC, який раніше називався Стандартним часом за Ґрінвічем, Greenwich Mean Time або GMT, бо він був місцевим часом в Ґрінвічі у Великобританії). Коли люди, що знаходяться в різний часових поясах мають необхідність в спілкуванні, вони можуть виражати час в універсальному часі і позбутися, таким чином, недорозумінь, які можуть траплятися від мішанини часів.

Кожен місцевий час називається часовим поясом. Але хоча географія і може дозволити всім людям, що мають один і той же полудневий час, жити в одному часовому поясі, але політика дозволити цього не може. З багатьох причин, багато країн користуються **літнім часом**, тобто, вони переставляють свій

[1] Як показують найновіші дослідження

годинник, щоб мати більше світлого часу поки вони працюють, і потім повертають стрілки годинників назад в зимовий час. Інші країни цього не роблять. Ті, які це роблять не можуть погодитися, коли саме переводити стрілки годинників, і правила змінюються кожного року. Все це вносить неабиякі складнощі в визначення часових поясів.

Визначати часові пояси найкраще або за місцем або за різницею часу відносно Ґрінвічського часу. В США та деяких інших країнах часові пояси мають назви та абревіатури, що складаються з трьох літер. Однак, деякі абревіатури співпадають з іншими, і тому ними не можна користуватися, якщо не вказана також і країна. Кращим є поняття місцевого часу в Гельсінкі, ніж, скажімо, місцевий східньо-європейський час, бо не всі країни східної Європи дотримуються одних і тих же правил.

Лінакс включає в себе пакет часових зон, в якому мається інформація про всі існуючі часові пояси. Ця інформація може легко поновлюватися, якщо правила змінюються. Все, що треба робити системному адміністратору, це вибрати відповідний часовий пояс. Крім цього, кожен користувач може встановити свій власний часовий пояс, оскільки багато хто працює з комп'ютерами через мережу, знаходячись на величезній відстані від комп'ютера. При зміні правил переходу на літній час в Вашому місцевому часовому поясі, не забудьте змінити хоча б цю інформацію в Вашій Лінакс системі. Крім зміни часового поясу в Вашій системі та поновлення системної інформації про часові пояси у Вас немає інших турбот щодо часу, як у системного ажміністратора.

## 12.2 Програмний та апаратний годинник

Персональний комп'ютер має годинник, який працює від батарейки. Дякуючи батарейці годинник в комп'ютері може працювати навіть тоді, коли комп'ютер вимкнений з мережі. Вірний час в апаратному годиннику можна встановити з вікна установки в BIOS'і, або з будь-якої операційної систми, що працює не комп'ютері.

Ядро Лінакса веде відлік часу незалежно від апаратного годинника. Під час завантаження системи, Лінакс встановлює свій годинник на той же час, що показує апаратний годинник. Після цього два годинники працюють незалежно один від одного. Лінакс має свого власного годинника, бо звіряти час з апаратним дуже повільно і складно.

Годинник в ядрі системи завжди показує час в універсальносу часі. Таким чином ядру не треба знати нічого про часові пояси – простота організації приводить до більш високої надійності і дає простіший механізм для поновлення інформації про часові пояси. Кожен процес самостійно займається перетворенням поясових часів (використовуючи ті засоби, які надаються пакетом часових поясів).

Апаратний годинник може бути встановленим як на місцевий час, так і на універсальний. Взагалі, краще мати годинник встановленим на універсальний час, бо в цьому випадку Вам не потрібно переводити Ваш годинник у комп'ютері, коли час у Вашому поясі переводиться на літній (універсальний час не має літнього часу). На жаль, деякі операційні системи (включаючи MS-DOS, Windows, OS/2) завжди вважають, що апаратний час встановлено у місцевий час зони. Лінакс може працювати з обома установками, але тоді треба не забувати переставляти годинник, коли літній час вводиться чи відміняється (інакше апаратний годинник не буде показувати місцевий час).

## 12.3 Відображення та установка часу

В системі Дебіан (Debian) часовий пояс системи визначається символічною ссилкою `/etc/localtime`. Ця ссилка показує на файл даний часового поясу, який описує даний часовий пояс. Файли даних часових поясів зберігаються в `/usr/lib/zoneinfo`. Інші комплектації Лінаксів можуть робити це інакше.

Користувач може встановити свій власний часовий пояс встановленням змінної середовища TZ. Якщо ця змінна не встановлена, тоді вважається, що часовий пояс користувача той же самий, що і у системи. Синтаксис встановлення часової змінної TZ описаний на сторінці *tzset* (3).

Команда `date` показує час і дату в даний момент.[2] Наприклад:

```
$ date
Sun Jul 14 21:53:41 EET DST 1996
$
```

Час зараз: Неділя, 14-го липня 1996 року, близько за десять десята вечора в часовому поясі, що називається «EET DST» (це може бути літній час східної Європи — East European Daylight Savings Time). Крім цього `date` може також показувати універсальний час:

```
$ date -u
Sun Jul 14 18:53:42 UTC 1996
$
```

`date` також використовується для переводу програмного годинника системи:

```
# date 07142157
Sun Jul 14 21:57:00 EET DST 1996
# date
Sun Jul 14 21:57:02 EET DST 1996
#
```

Синтаксис команди трохи закручений, тому за деталями звертайтесь до сторінки підказки *date*. Встановлювати час може тільки `root`. Не дивлячись на те, що кожен користувач може встановлювати свій часовий пояс, годинник в системі - один на всіх [3]рочитавши останню фразу не можу стримати себе від налету знайомих до болю анекдотів радянських часів: «Чому протигази лежать на різних поличках?», «Вони розкладені по партіям», «Партія у нас одна — покладіть протигази всі разом!».

`date` показує та встановлює тільки програмний годинник системи. Команда `clock` може синхронізувати програмний та апаратний годинники. Вона використовується під час завантаження системи — читає апаратний годинник і переводить програмний годинник на той же самий час. Якщо Вам потрібно змінити обидва годинники, Вам спершу треба перевести програмний за допомогою `date`, а потім встановити апаратний за допомогою `clock -w`.

Параметр `-u` команди `clock` вказує їй, що апаратний годинник показує універсальний час. Параметром `-u` *треба* користуватися вірно. Інакше Ваш комп'ютер страшенно розгубиться і не буде знати, який же насправді час.

Годинники треба переводити дуже обережно. Багато що в Юніксі залежить від годинника. Наприклад, демон `cron` періодично виконує команди. Після переводу годинника він може розгубитися, і не зрозуміє: виконувати команду чи ні. На одній з ранніх систем Юнікс хтось встановив годинник на двадцять років вперед і `cron` захотів виконувати всі періодичні команди за двадцять років за один раз. Сучасні версії `cron`'у вміють вірно поводитися з цим, але все одно

[2]Будьте обережними з командою `time`, яка *не* показує час.

[3]**Прим. перекл.:** П *(ДК)*

треба бути обережним. Великі стрибки в часі вперед та назад більш небезпечні, ніж малі. [4]собисто мені здається це відноситься до розряду таких собі навколо комп'ютерних баєчок. Оскільки `cron` зроблений таким чином, що він не пам'ятає свого стану, то навряд чи він стане виконувати команди, тільки тому, що хтось переставив годинник. «Не пам'ятає стану» означає ось що: `cron` не веде ніяких записів про те, які команди виконані, а які ні, і якщо вони виконувалися, то коли саме. Працює він так: він постійно спить, і просинається кожної хвилини тільки для того, щоб перевірити, чи запланована на *цю* хвилину якась команда. Якщо така команда є, то він її запускає (або ставить в чергу на виконання), якщо ж ні, то він каже всім: «Добраніч», і іде спати далі (аж на цілу хвилину). Отже якщо хтось, (із якимись злими умислами, або ж просто з невігластва) переставив годинник на двадцять років вперед (можливо намагаючись таким чином позбутися проблеми 2000 року(?)), то ... `cron` прокинеться, подивиться на годинник і почне виконувати команди, які заплановані на *ту* хвилину, через 20 років, і навіть не згадає, як ці 20 років пролетіли, і чи робив він що-небудь поки вони пройшли чи ні. Отже, єдина проблема може бути з `cron`'ом, тільки коли він ставить програми в чергу на виконання. Якщо команда з якоїсь причини не виконується за відведений їй час (це не значить, що команда не *встигла* виконатися, ні, просто вона не змогла *запуститися*), то черга команд збільшується на одну команду. Якщо і наступна не виконується, то ... (див. раніше). Зі мною особисто був випадок, коли один з користувачів жалівся, що його `crontab` не працює. При перевірці виявилося, що його `cron` роботи були заплановані запускатися кожної хвилини, але з якихось причин кільканадцять команд не виконалися. Через це черга команд переповнилася і замість того, щоб виконувати заплановані на дану хвилину команди, `cron` кожної хвилини жалівся (через `/var/adm/messages`): «Черга команд занадто довга, не можу виконувати наступну команду і тому лягаю знову спати» - при цьому і так занадто довга черга ставала довшою іще на одну команду.

## 12.4 Коли годинник відстає

Програмний годинник в Лінаксі не завжди точний. Його робота підтримується періодичними **перепинами таймера**, який ґенерується апаратними засобами PC. Якщо система досить сильно завантажена, на обробку перепину від таймера може піти занадто багато часу, і програмний годинник починає тоді відставати. Апаратний годинник працює незалежно, і в більшості випадків точніше. Якщо Ви перегружаєте свій комп'ютер часто (що трапляється з більшістю систем, які не є серверами), Ви будете мати досить точний час.

Якщо Вам потрібно відрегулювати апаратний годинник, то найпростіше це зробити, перегрузивши комп'ютер, ввійшовши в BIOS та встановити годинник звідти. Якщо це Вам не підходить, встановіть час з `date` та `clock` (в такій саме послідовності), але будьте готовими до перегрузки системи, якщо деякі з її частин починають себе дивно поводити.

Комп'ютер під'єднаний до мережі (навіть якщо це самий звичайний модем) може перевіряти свій годинник автоматично, звіряючись з деякими іншими системами. Якщо деяка система вміє підтримувати дуже точний час, то рівняючись на неї Ваш комп'ютер теж буде мати точний час. Це може бути зроблено за допомогою команд `rdate` або `netdate`. Обидві команди звіряють годинник з іншими системами в мережі, а `netdate` може звірятися з кількома системами одночасно, і переводити місцевий годинник відповідно. Виконуючи ці команди регулярно, Ви будете мати дуже точний годинник в Вашій системі.

**META:** say something intelligent about NTP

[4]**Прим. перекл.:** О *(ДК)*

# Додаток A

# Вимірювання дірок

Цей додаток містить цікаву частину програми, яка використовується для вимірювання потенційних дірок в файлових системах. Вихідний текст книжки має повний текст програми. (`sag/measure-holes/measure-holes.c`).

```
int process(FILE *f, char *filename) {
        static char *buf = NULL;
        static long prev_block_size = -1;
        long zeroes;
        char *p;

        if (buf == NULL || prev_block_size != block_size) {
                free(buf);
                buf = xmalloc(block_size + 1);
                buf[block_size] = 1;
                prev_block_size = block_size;
        }
        zeroes = 0;
        while (fread(buf, block_size, 1, f) == 1) {
                for (p = buf; *p == '\0'; )
                        ++p;
                if (p == buf+block_size)
                        zeroes += block_size;
        }
        if (zeroes > 0)
                printf("%ld %s\n", zeroes, filename);
        if (ferror(f)) {
                errormsg(0, -1, "read failed for `%s'", filename);
                return -1;
        }
        return 0;
}
```

# Додаток B

# Термінологічний словник (початковий варіант)

*The Librarian of the Unseen University*
*had unilaterally decided to aid comprehension*
*by producing an Orang-utan/Human Dictionary.*
*He'd been working on it for three months.*
*It wasn't easy. He'd got as far as 'Oook.'*
*(Terry Pratchett, «Men At Arms»)*

Це коротенький список визначень понять, які відносяться до Лінаксу та системної адміністрації. Ссилка вказує на сторінку, де це поняття зустрічається або вперше, або ж це найважливіше місце для цієї ссилки.

**амбіції** Акт написання дурнувастих речень та пассажів із надією, що вони попадуть коли небудь в Лінаксівський «cookie-file».

**прикладна програма** (p. 19) Програма, яка робить хоч що-небудь корисне. Результатом використання прикладної програми є саме те, ради чого цей комп'ютер купувався. Див. також системні програми, операційна система.

**демон** Процес, який теліпається десь там на фоні, як звично, нікому не помітний, аж до того часу, поки що-небудь не спонукає його до дії. Наприклад, демон `update` просинається кожні тридцять секунд щоб злити буферний кеш, або `sendmail` демон, який просинається тоді, коли хто-небудь відсилає свій лист електронною поштою.

**файлова система** (p. 64) Методи та структури даних, якими користується операційна система, для того щоб вести облік файлів на диску та підрозділі диску; спосіб у який файли організовані на диску. Крім того використовується, при посиланні на підрозділ чи диск, який використовується для зберігання файлів чи тип файлової системи.

**словник термінів** Список слів та пояснень щодо того, що ці слова означають. Не треба плутати це із словником, який також, до речі, є списком слів та їх пояснень.

**ядро** (p. 19) Частина операційної системи, яка втілює взаємодію з апаратурою та спільне використання ресурсів. Див. також системну програму.

**операційна система** (p. 19) Програмне забезпечення, яке розподіляє копм'ютерні системні ресурси (процесор, пам'ять, дисковий простір, мережа і т.п.) між всіма користувачами системи та прикладними

програмами, якими вони користуються. Керує доступом до системи для забезпечення безпеки системи. Див. також ядро, системна програма, прикладна програма.

**системний виклик** (p. 19) Послуги, які надаються прикладним програмам ядром і спосіб, в який вони стартують. Див. розділ 2 сторінок підказок (man pages).

**системна програма** (p. 19)

Програми, які втілюють верхній рівень функціональності операційної системи, тобто, ті речі, які не залежать напряму від «заліза». В деяких випадках вимагають спеціальних прав доступу для свого виконання (наприклад, для доставки електронної пошти), але загалом розглядаються як частина операційної системи (наприклад, компілятор). Див. також прикладна програма, ядро, операційна система.