МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ «ЧЕРНІГІВСЬКА ПОЛІТЕХНІКА»
Кафедра кібербезпеки та математичного моделювання

АДМІНІСТРУВАННЯ UNIX-ПОДІБНИХ СИСТЕМ

КОНСПЕКТ ЛЕКЦІЙ
для здобувачів
першого (бакалаврського) рівня вищої освіти
освітньо-професійної програми «Кібербезпека»
спеціальності 125 Кібербезпека та захист інформації

Обговорено і рекомендовано
на засіданні кафедри
Кібербезпеки та математичного
моделювання
Протокол №2
від 13 лютого 2024 р.

Чернігів 2024

Адміністрування Unix-подібних систем. Конспект лекцій для здобувачів першого (бакалаврського) рівня вищої освіти освітньо-професійної програми «Кібербезпека» спеціальності 125 Кібербезпека та захист інформації. – Чернігів: НУ «Чернігівська політехніка», 2024 – 170 с.

Укладачі: СЕМЕНДЯЙ СЕРГІЙ МАТВІЙОВИЧ, старший викладач кафедри кібербезпеки та математичного моделювання;
ПЕТРЕНКО ТАРАС АНАТОЛІЙОВИЧ, доцент кафедри кібербезпеки та математичного моделювання, кандидат технічних наук;
ГРЕБЕННИК АЛЛА ГРИГОРІВНА, старший викладач кафедри кібербезпеки та математичного моделювання;
КАЛЬЧЕНКО ДМИТРО ВОЛОДИМИРОВИЧ, викладач кафедри кібербезпеки та математичного моделювання, доктор філософії

Відповідальний за випуск – ТКАЧ ЮЛІЯ МИКОЛАЇВНА,
завідувач кафедри кібербезпеки та математичного моделювання, доктор педагогічних наук, професор

Рецензент – ШЕЛЕСТ МИХАЙЛО ЄВГЕНОВИЧ,
професор кафедри кібербезпеки та математичного моделювання, доктор технічних наук, професор

ЗМІСТ

ВСТУП

Конспект лекцій призначений для засвоєння курсу «Адміністрування Unix-подібних систем» здобувачами вищої освіти освітньо-кваліфікаційного рівня «бакалавр» спеціальності 125 – «Кібербезпека» та закріплення необхідних у подальшій роботі знань з основ системного адміністрування.

Матеріал дисципліни тісно пов’язаний зі спеціальними дисциплінами, що викладаються у вищих навчальних закладах технічного профілю.

Даний конспект лекцій ознайомить студентів з основними принципами адміністрування Unix-подібних систем та заходами із забезпечення безпеки систем, допоможе формуванню у студентів навичок системного адміністратора, вивченню методик, необхідних для управління автономною системою, протоколів, що використовуються в UNIX-системах, а також способів побудови, розширення та адміністрування мереж та інтернет-серверів.

У змістовому модулі 1 “Основи адміністрування” (лекції 1-4) проводиться загальний огляд систем UNIX та Linux з погляду системного адміністратора. Розглядтиметься високорівневе мережеве програмне забезпечення. Серед тем, що вивчаються, можна виділити систему доменних імен (DNS), мережеву файлову систему (NFS), інструменти мережевого управління.

У змістовому модулі 2 “Робота в мережі” (лекції 5-8) вивчаються протоколи, що використовуються в системах UNIX, а також способи налаштування, розширення та експлуатації мереж та серверів з виходом в Інтернет, приватних віртуальних мереж, розглядаються заходи із забезпечення безпеки систем.

ЛЕКЦІЯ 1. Основні завдання системного адміністратора.

1.1 Історія UNIX

UNIX – операційна система, створена у 1970 році для комерційних цілей. Спочатку система була доступна виключно для великих компаній і організацій, які могли сплатити вартість ОС. На базі UNIX пізніше були розроблені Linux, BSD та Mac OS.

Сьогодні варто виділити основні особливості UNIX-системи, які дозволяють не втрачати їй своєї популярності і затребуваності:

✓ Підтримка різного обладнання

✓ Робота з командним рядком

✓ Мультизадачність

✓ Розрахований на багато користувачів інтерфейс та інші переваги системи.

1.1.1. Історія розвитку UNIX

Спочатку система UNIX створювалася «для себе». Розробники компанії Bell Labs AT&T зробили початкову програму на комп'ютері PDP-7, потім основну частину програми, яка була написана на асемблері, була переписана на мову С. Завдяки цьому UNIX можна було використовувати на різних архітектурах. Після того, як код системи потрапив до університету Берклі, було створено свою версію операційної системи під назвою BSD UNIX (Berkeley Software Distribution).

Ім'я UNIX походить від співзвучного **MULTICS (Multiplexed Information and Computing Service)**, це один із початкових проєктів розробки ОС.

Також абревіатура UNIX означає Uniplexed Information and Computing System (Об'єднана інформаційно-обчислювальна система).

До 1979 року вийшло шість редакцій UNIX:

У листопаді 1971 року з'явилася перша. Вона вже включала компілятор FORTRAN та версії ПЗ, які використовуються навіть сьогодні.

У червні 1972 року з'явилася друга версія. UNIX успішно встановлювалася на комп'ютерах компанії AT&T.

У лютому 1973 року випустили 3-ту версію. Вона була доступна разом з компілятором Сі та pipes.

У листопаді 1973 року вже представили 4 редакцію, яку повністю переписали на мову Сі.

У червні 1974 року UNIX 5 редакції був доступний на 50 робочих системах.

У травні 1975 року з'явилася вже 6 редакція.

ОС розпочала свій шлях у освітніх закладах. В 1976 Юнікс портували на Interdata 8/32 комп'ютер, а після успішної появи 6 редакції, почалося створення клонів системи. Інші розробники оцінили користь та можливості нової ОС та розпочали розробку подібних систем на базі оригінальної UNIX.

У січні 1979 року була випущена 7 версія системи. Основна відмінність від попередніх – більш стабілізована файлова система. Також у ОС з'явилися нові інструменти. З цієї версії UNIX розділилася на 2 потоки – BSD та System V.

Сьогодні власником ТМ UNIX є The Open Group. Винятково сумісні та сертифіковані системи специфікації Single UNIX кваліфікуються, як UNIX (Unix-like).

1.1.2. Навіщо використовується UNIX?

Сьогодні UNIX використовується виключно з корпоративною метою, наприклад, для установки на сервери та додаткове робоче обладнання. Для домашньої установки UNIX не використовується, тому що у нього немає графічної оболонки, тобто інтерфейсу для управління, а необхідне ПЗ потрібно ставити руками.

UNIX-подібні ОС сьогодні – це багатозадачні ОС, створені для великої кількості користувачів. UNIX був розроблений для різних машин, у тому числі для систем POSIX, включаючи Linux та Mac OS, тому не дивно, що сьогодні на базі UNIX працює багато проєктів.

Сьогодні найбільші постачальники UNIX-систем – це компанії **IBM, HP, SUN.**

UNIX в основному використовують компанії для своїх серверів, які мають певні вимоги через застаріле програмне забезпечення.

1.1.3. Які основні переваги систем UNIX?

UNIX це відкритий вихідний код, який можна адаптувати під свій бізнес і не використовувати зайвого. Це максимально зручно та практично.

Система безпечна, не потрібні додаткові антивірусники, як на ОС Windows. Грамотне налаштування сервера та ваші дані будуть у повній безпеці на UNIX подібній системі. Крім того, є безліч безкоштовних антивірусних програм, які допоможуть вам спати спокійно.

Можливість використання на застарілих моделях комп'ютерів. Це дуже зручно, якщо у вас є обладнання, яке ще потрібно підтримувати у працездатному вигляді та забезпечувати його функціональність та надійність. У UNIX-системах дуже гнучкі налаштування.

На рівні протоколів та даних UNIX дозволяє проводити інтеграцію для забезпечення продуктивності системи. Це безперечний плюс, UNIX – це відкриті стандарти для комерційного використання.

Велика кількість інструкцій в Інтернеті та фахівців, які готові допомогти у налаштуванні системи. Ви ніколи не залишитеся наодинці, якщо раптом знадобиться допомога в налаштуванні системи.

1.1.4. UNIX-подібні системи

Наведемо нижче список найбільш затребуваних і популярних ОС, які використовують UNIX-системи. UNIX є прабатьком багатьох сучасних ОС. Важливо відзначити, що більшість з них повністю безкоштовні, а в мережі є велика кількість інструкцій для роботи з цими системами навіть для користувачів-початківців.

Centos – Для бізнесу та корпоративних клієнтів, для встановлення на віртуальні та виділені сервери

Ubuntu – Для загального використання, як безкоштовний аналог Windows OS, але на базі Linux

Mint – Сучасна ОС загального призначення для вирішення щоденних завдань користувачів

OpenSuse – Для встановлення на сервер або домашній ПК

Debian – Для серверних рішень, для роботи ПЗ

ArchLinux – Система, що постійно оновлюється, для встановлення сучасних серверних програм

Slackware – Для комп'ютерів на базі процесорів Intel

Red Hat – Корпоративна система для виробничих підприємств

Gentoo – Для організації домашньої або офісної робочої станції

FreeBSD – Для організації веб-серверів та інтернет-шлюзів

OpenBSD – Для організації роботи різних апаратних платформ

Mac OS X – Для роботи продуктів від компанії Apple

HP-UX – Комерційна ОС для установки на продукти від компанії Hewlett-Packard

Solaris – Для підтримки корпоративних інформаційних систем.

1.2. Ініціалізація користувачів.

До кола обов'язків системного адміністратора входить створення облікових записів для нових користувачів, видалення облікових записів тих користувачів, які вже не працюють у системі, і розв'язання всіх проблем, що виникають під час «системного життя» своїх підопічних (наприклад, «підказка» забутих паролів). Процес управління записами можна автоматизувати, але низку рішень, пов'язаних із включенням у систему нового користувача (місце розміщення його початкового каталогу, комп'ютер, на якому має бути створено обліковий запис, тощо), має ухвалювати тільки адміністратор.

Якщо необхідно припинити доступ користувача до системи, його обліковий запис має бути анульовано. Усі файли, що відносяться до цього користувача, необхідно видалити, щоб вони не займали місце на диску.

1.3. Підключення і видалення апаратних засобів.

Після придбання нового програмного забезпечення його потрібно інсталювати та протестувати, часто в кількох операційних системах і на різному обладнанні. Якщо програми працюють нормально, користувачам необхідно повідомити про їхню наявність і місцезнаходження. Пакети оновлень, що випускаються для виправлення помилок і усунення прогалин у системі безпеки, мають без проблем встановлюватися в локальних системах.

Локальні програми й адміністративні сценарії слід відповідним чином упакувати й організувати у вигляді, що забезпечує сумісність із процедурами оновлення, які використовуються в системах вашого сайту. Нові версії ваших програмних продуктів, перш ніж використовувати їх у всьому вузлі мережі, необхідно представляти для тестування.

1.4. Моніторинг системи

Великі системи вимагають невсипущого контролю. Не варто сподіватися, що користувачі завжди самі повідомлятимуть вам про труднощі (якщо, звісно, вони не зіткнуться по-справжньому із серйозними проблемами). Зазвичай користувачі йдуть шляхом найменшого опору, вважаючи, що на розв'язання проблеми в них піде менше часу, ніж на її опис і повідомлення про її виникнення.

Існує безліч обов'язкових щоденних операцій, а саме: перевірка правильності функціонування електронної пошти та веб-служб; перегляд журнальних файлів на предмет наявності ранніх ознак несправностей; контроль над підключенням локальних мереж; контроль доступності системних ресурсів (зокрема, перевірка наявності вільного місця на диску). Усі ці рутинні операції чудово піддаються автоматизації, та й безліч готових систем моніторингу можуть допомогти системним адміністраторам у вирішенні цього завдання.

1.5. Пошук несправностей

Збої систем неминучі. Завдання адміністратора – діагностувати збої в системі та в разі потреби викликати фахівців. Як правило, знайти несправність буває набагато складніше, ніж усунути її. Реакція системного адміністратора на збої в UNIX-подібних системах повинна бути структурованою та ефективною, щоб якнайшвидше відновити нормальну роботу системи. Ось варіант послідовності кроків, яких слід дотримуватись, а також рекомендацій системному адміністратору щодо необхідних дій:

- **Оцінка ситуації:** Спочатку потрібно швидко оцінити серйозність збою. Це допоможе визначити, чи це критична проблема, яка потребує негайної реакції.
- **Збір інформації:** Зібрати якомога більше інформації про збій. Перевірити логи подій, журнали, вивід команд тощо. Це допоможе зрозуміти причину збою.
- **Припинити незаплановані операції**: Якщо це можливо, відключіть або зупиніть всі незаплановані операції або сервіси, які можуть призвести до подальших проблем.
- **Відновлення збережених даних:** Якщо збій вплинув на доступ до даних, спробуйте відновити дані з резервних копій.
- **Виявлення причини збою:** Аналізуйте логи, виведення команд, діагностичну інформацію та інше, щоб зрозуміти причину збою. Це може бути пов'язано з апаратурою, програмним забезпеченням, конфігурацією тощо.
- **Виправлення проблеми**: Після визначення причини збою, внесіть необхідні зміни для виправлення проблеми. Це може включати в себе виправлення конфігурації, відновлення пошкоджених файлів, оновлення програм тощо.
- **Тестування і перевірка:** Переконайтеся, що внесені зміни вирішили проблему і не спричинили нових. Проведіть тестування, щоб переконатися в стабільності системи.
- **Комунікація**: Якщо збій вплинув на користувачів або клієнтів, повідомте їх про ситуацію, можливий вплив і орієнтовний час відновлення послуг.

- **Запобігання майбутнім збоям:** Ретельно аналізуйте причини збою і вживіть заходів для запобігання їм в майбутньому, таких як покращення моніторингу, оновлення програм, реалізація резервних систем тощо.
- **Документування**: Після відновлення системи, докладно задокументуйте причини збою, вжиті заходи для відновлення та попередження, а також вчинені дії. Це допоможе в майбутньому при подібних проблемах.

Загалом, ефективна реакція на збої в UNIX-подібних системах вимагає швидкості, методичності і систематичності у вирішенні проблеми.

1.6. Ведення локальної документації

Під час налаштування конфігурації системи під конкретні вимоги дуже скоро виявляється, що вона значно відрізняється від базової конфігурації, яка описана в документації. А оскільки за реалізацію цих налаштувань відповідає системний адміністратор, він має документувати всі зміни. Адміністратор має також вести облік обслуговування всіх апаратних засобів, реєструвати стан резервних копій, документувати розводку мережевих кабелів і локальні правила роботи в системі. Рекомендації, що стосуються ведення документації, розглянемо у наступних лекціях.

1.7. Стеження за безпекою системи

Системний адміністратор відповідає за реалізацію стратегії захисту і повинен періодично перевіряти, чи не порушено безпеку системи. У системах із низьким рівнем безпеки цю процедуру можна звести до кількох елементарних перевірок на предмет несанкціонованого доступу. У системах із високим рівнем безпеки зазвичай застосовується складна мережа пасток і програм контролю. Питання безпеки більш детально розглядатимемо у наступних лекціях.

1.8. Надання допомоги користувачам

Про необхідність надання допомоги користувачам у розв'язанні різноманітних проблем рідко згадують у посадовій інструкції системного адміністратора, хоча виконання такого роду обов'язків «з'їдає» більшу частину його робочого часу. Системних адміністраторів тероризують найрізноманітнішими запитаннями, починаючи від «Вчора моя програма працювала, а сьогодні ні! Що ви поміняли?» до «Я пролила каву на клавіатуру! Чи потрібно тепер полити її водою, щоб змити каву?».

У більшості випадків ваша реакція на всі ці «сигнали тривоги» набагато більше впливає на вашу оцінку як адміністратора, ніж реальні технічні навички, якими, можливо, ви володієте. Ви можете або нарікати на несправедливість такого стану речей, або радіти з того, що успішним розв'язанням однієї-єдиної проблеми ви можете заслужити таку саму прихильність працівників (або начальства), як і п'ятигодинним налагодженням у нічний час.

1.9. Безліч ролей системного адміністратора

Деяких системних адміністраторів початкового рівня просять виконувати одне завдання, а зі збільшенням досвіду вони доходять до виконання більшої кількості

завдань. Іноді системні адміністратори починають із завантаженості великою кількістю завдань, а згодом спеціалізуються.

У невеликій компанії може знадобитися, щоб її єдиний системний адміністратор узяв на себе багато завдань. У міру зростання організації певні завдання можуть бути передані новим системним адміністраторам. Іноді ви можете виявити, що певна роль вам не подобається і ви намагаєтеся уникнути її під час зміни роботи. Розгляд цих завдань також може допомогти керувати вашою кар'єрою в плані того, у яких компаніях ви хочете працювати: у невеликих компаніях зазвичай вимагають від людей виконання кількох завдань, у більших компаніях часто потрібні більш спеціалізовані співробітники, а в транснаціональних корпораціях є настільки вузькоспеціалізовані співробітники, що стороннім це може здатися диким. Технічні компанії поважають і заохочують тих, хто виконує завдання із впровадження нових технологій, тоді як в інших компаніях часто не схвалюють занадто сильних змін.

1.9.1 Поширені позитивні ролі

Деякі ролі в компанії важливіші за інші, якісь із них хороші, а інші – погані. Тут ми наведемо багато поширених ролей і пояснимо, яку користь вони приносять компанії, як ці люди отримують задоволення від роботи і чого від них очікують користувачі.

Установник

Дехто розглядає системного адміністратора як людину, яка встановлює «всяку всячину». Цю роль найчастіше бачать користувачі, і тому вона зазвичай асоціюється з роботою системного адміністратора. Користувач рідко бачить інші, можливо, важливіші посади, наприклад людей, які проєктують інфраструктуру.

Цінність установників для компанії полягає в їхній здатності доводити роботу до кінця і бачити, що робота виконується. Часто вони є останньою і найважливішою ланкою в ланцюзі розроблення.

Коли установку виконують у великому масштабі, об'єкт, що встановлюється, зазвичай попередньо налаштовують у якомусь центральному місці розташування. Людина, яка стане хорошим установником, – це та, кому приносить задоволення зустрічатися з користувачами і допомагати їм і хто отримує задоволення від багаторазового якісного виконання одного й того самого завдання. З іншого боку, за менш масштабного встановлення передбачається, що установник буде більш досвідченою людиною, тому що можливо більше несподіваних ситуацій.

Якщо ви установник, важливо бути доброзичливим і ввічливим. Установник – це публічна особа організації; люди будуть думати, що вся організація діє так, як ви.

Ремонтник

Усе ламається. Дехто розглядає системного адміністратора як ремонтника. Точно так само, як люди телефонують комусь, коли ламається їхня посудомийна машина, вони телефонують комп'ютерному ремонтнику, коли ламається їхній комп'ютер. Крім того, системні адміністратори ремонтують більші й іноді більш ефемерні речі, наприклад «Інтернет» або «базу даних». Користувача мало цікавить, чи полягає реальна проблема в простому обриві кабелю, чи вона набагато серйозніша.

Цінність ремонтників для компанії полягає в їхній здатності повертати компанію до життя, коли технологічні проблеми блокують бізнес. Ремонтники отримують задоволення від усвідомлення того, що вони допомогли одній людині або цілій компанії. Вони отримують задоволення від складності хорошої загадки або секрету.

Якщо ви ремонтник, користувачам потрібно знати, що вам не байдужі їхні проблеми. Їм потрібно відчувати, що їхні проблеми – найважливіші у світі.

Співробітник підтримки

Співробітник підтримки – це людина, яка підтримує роботу раніше побудованих систем. У співробітників підтримки дуже добре виходить виконувати вказівки, дані їм у вигляді письмової інструкції або за допомогою навчання. Вони не прагнуть поліпшити систему, вони хочуть підтримувати її такою, яка вона є.

Цінність підтримуючих співробітників для компанії полягає в тому, що вони вносять у середовище стабільність. Такі люди не будуть нічого ламати, намагаючись це поліпшити або замінити. Також вони не витрачатимуть весь день на читання журналів про те, що можна поставити нового. Коли компанії витрачають гроші на встановлення будь-якої системи, їм потрібно буде підтримувати її стабільність протягом часу, достатнього для того, щоб вона себе окупила, перш ніж замінювати її на щось новіше.

Співробітники підтримки отримують задоволення від усвідомлення того, що їхня робота є елементом великої справи, що підтримує діяльність організації. Вони схильні радіти тому, що вони не ті люди, яким доводиться з'ясовувати, як проєктувати і встановлювати наступне покоління систем, і навіть можуть зневажати тих, хто хоче замінити їхню стабільну систему чимось новим.

Якщо ви співробітник підтримки, користувачам потрібно від вас дві протилежні якості: вони хочуть знати, що ви підтримуєте стабільність їхнього світу, і бажають, щоб ви були гнучким, якщо їм потрібна індивідуалізація.

Запобіжник

Запобігання проблемам – завдання, невидиме більшості користувачів, і співробітник, який його виконує, шукає проблеми й усуває їх до того, як вони стануть видимими. Запобіжники виконують закулісне планування і профілактичне обслуговування, яке не дає змоги проблемам з'явитися взагалі. Хороший запобіжник відстежує показники для пошуку тенденцій, а також завжди має бути в курсі подій, щоб знати, які проблеми можуть виникнути надалі.

Цінність запобіжника для компанії полягає в попередженні проблем, а це дешевше за усунення проблем за їхньої появи.

Запобіжники отримують задоволення від усвідомлення того, що їхня робота запобігла проблемам, про можливе виникнення яких ніхто не знав. Їхня радість таємна. Вони отримують задоволення від продумування дій у довгостроковій перспективі, а не від успішного вирішення екстреної проблеми.

Типові користувачі не підозрюють про існування такої людини, але їхнє керівництво знає. Керівництво розраховує, що в цієї людини ті ж самі пріоритети, що й у неї.

Герой

Системний адміністратор може бути героєм, який здійснює свій щоденний подвиг. Як пожежник, який виносить людей з палаючої будівлі, герой отримує лестощі та похвалу. Мережа не працювала, але тепер вона працює. Демоверсія не була готова, але системний адміністратор працював усі вихідні, щоб провести мережу в цю частину будівлі. Герої отримують задоволення від своєї роботи, коли їх хвалять після її виконання.

Цінність героїв для компанії дуже велика: керівництво завжди заохочує героя. Кумедно, але запобіжники часто змушені боротися за подібне позитивне враження, незважаючи на те що їхній внесок може бути не меншим, а то й більшим.

Герої отримують задоволення від усвідомлення того, що вони володіють якимись ключовими знаннями, без яких компанія не може жити. Роль героя – не з тих, які передбачають здорове життя поза роботою. Вони жертвують ночами, вихідними та відпустками, часто не помічаючи цього. Особисте життя відходить на другий план. Зрештою герої вигорають і стають мучениками, якщо керівництво не знаходить будь-якого способу полегшити їхню напругу.

Користувачі розраховують, що герой знайдеться завжди і скрізь. Вони воліли б мати справу тільки з героєм, тому що ця ефектна суперзірка стала кимось, на кого вони можуть покластися. Однак користувачам потрібно знати, що, якщо вони отримають бажане, герой згорить на роботі. Для пошуку нових героїв потрібен час.

Універсал

Ця людина здобула репутацію співробітника, який може вирішити будь-яку проблему. Універсали трохи схожі на героїв, але вони більш координовані та більше покладаються на інфраструктуру. Замість того, щоб бігати і гасити пожежі або ремонтувати сервер з третьої дня в п'ятницю до третьої ночі в неділю, ця людина є тим, до кого керівництво піде у разі виникнення масштабних питань, де необхідні глибокі знання. Керівництво знає, що універсал дійде до кореня проблеми, з'ясує проблеми, що лежать в основі, та усуне їх. Це може бути проблема інфраструктури (налаштування параметра бази даних), або проблема процесу (як забезпечити, щоб нові користувачі мали загальну конфігурацію, необхідність створити нову автоматизовану систему і практично все що завгодно).

Цінність наявності універсала під рукою в тому, що він може зробити те, що не можуть зробити інші.

Як і герой, ця людина може згоріти на роботі в разі зловживання її роботою, але коли вона працює, то отримує задоволення від усвідомлення того, що її рішення стане елементом стандартних процедур, які будуть використовуватися надалі.

Користувачі розраховують, що, коли універсал погодиться розв'язати проблему, він доведе роботу до кінця і зможе надати точну оцінку часу або, принаймні, періодичні повідомлення про стан, поки таку оцінку дати не можна.

Творець інфраструктури

Корпоративна мережа залежить від великої кількості інфраструктур: DNS, директорій, баз даних, скриптів, комутаторів тощо. Типовий користувач не бачить нічого з цього, якщо не відбувається збій, який пояснюється такими незрозумілими фразами, як «Була проблема з DNS-сервером».

Чим більша компанія, тим ціннішими стають творці інфраструктури. Хороша інфраструктура аналогічна міцному фундаменту, на якому можна побудувати будинок. Ви можете побудувати будинок на хиткому фундаменті і зміцнити його за допомогою складніших і дорожчих проєктів, але в довгостроковій перспективі дешевше почати з міцного фундаменту. У невеликих компаніях практично немає інфраструктури. Більші компанії виграють від амортизації витрат на якісну інфраструктуру за дедалі більшою і більшою користувацькою базою. Коли малі компанії виростають і стають великими, часто це відбувається плавно через завбачливе наймання системних адміністраторів з «великими планами» щодо інфраструктури.

Творці інфраструктури отримують задоволення від довгострокового планування, удосконалення наявних систем, розширення великих систем до величезних, а також від переробки старих систем та їхньої заміни на новіші. Творці інфраструктури пишаються своєю здатністю не тільки будувати дуже великі системи, а й створювати витончені способи переходу на них.

Якщо ви творець інфраструктури, у вас є дві групи користувачів. Основній масі користувачів потрібно, щоб комп'ютерна інфраструктура була надійною, а нова інфраструктура встановлювалася вчора. Інші ваші користувачі – це системні адміністратори, чиї системи покладаються на інфраструктуру, яку ви будуєте. Системним адміністраторам потрібна документація та інфраструктура, яка є надійною та простою для їхнього розуміння, і вона потрібна їм зараз, бо, якщо ви не вкладетеся в строк, їхні проєкти теж затримаються.

Творець політики

Політики - основа інформаційних технологій. Вони поширюють вимоги вищих керівників компанії і визначають, як, коли і чому все потрібно робити. Часто системних адміністраторів просять створювати політики в інтересах свого керівництва. Соціальні проблеми неможливо вирішити технологічним шляхом. Деякі соціальні проблеми можуть бути вирішені тільки за допомогою письмової політики.

Цінність творців політики для компанії в тому, що вони вирішують деякі проблеми і запобігають виникненню нових. У міру зростання компанії поширення інформації стає дедалі складнішим і важливішим.

Творці політик отримують задоволення від усвідомлення того, що їхні знання, навички та особистий досвід були внесені в політику, яка поліпшила організацію. Крім того, вони отримують задоволення від того, що є координаторами, які можуть отримати визнання багатьох різних спільнот.

Якщо ви творець політики, користувачі розраховують, що ви будете враховувати їхні побажання. Це потрібно робити на початковому етапі процесу. Якщо питати думку людей після ухвалення основних рішень, це позбавляє їх впливу. Ваше бажання слухати оцінять.

Системний клерк

У системних клерків дуже мало влади та відповідальності щодо ухвалення рішень. Таким системним адміністраторам дають вказівки, які потрібно виконувати, наприклад «Створіть обліковий запис для Петренка» або «Виділіть IP-адресу». Якщо системний клерк – помічник системного адміністратора вищого рівня, це може бути

хорошим місцем. Насправді це прекрасний спосіб розпочати кар'єру. Однак, бачили системних клерків, які перебували під керівництвом нетехнічних керівників, яких дратувало, що клерк не міг виконувати завдання за межами своїх нормальних обов'язків.

Цінність системних клерків для компанії пов'язана з виконанням завдань, які інакше відволікали б старших системних адміністраторів від більш спеціалізованих завдань, і заміною відсутніх системних адміністраторів. Крім того, системний клерк є чудовим кандидатом на посаду більш старшого системного адміністратора при її звільненні. Клерк уже знає середовище, а менеджер із підбору персоналу знає клерка. Однак, якщо в середовищі немає старших системних адміністраторів, клерки часто є «цапами-відбувайлами» через погане комп'ютерне середовище, тоді як реальна проблема полягає в нестачі розуміння керівництвом технічної роботи.

Клерк отримує задоволення від добре зробленої роботи, від навчання новим навичкам і від прагнення до кар'єрного зростання. Якщо ви клерк, користувачам потрібне негайне виконання їхніх запитів, незалежно від того, чи є вони розумними.

Приклад: компанія, де були тільки системні клерки. Компанії потрібен баланс між старшими системними адміністраторами та клерками. В одній компанії були тільки системні клерки. Їхнє навчання охоплювало початкові навички роботи з UNIX: виконання резервного копіювання, створення облікових записів, виділення IP-адрес і IP-підмереж, встановлення нових програм і додавання нових вузлів. Клерки були жертвою синдрому «завжди можна додати ще одного»: нові замовлення сліпо виконувалися за першою вимогою без загального плану для підвищення ємності. Наприклад, новий вузол міг бути доданий у підмережу без будь-якого планування ємності мережі. Якийсь час це працювало, але врешті-решт призвело до перевантаження підмереж. Користувачі скаржилися на повільні мережі, але у клерків не було навичок проєктування мереж для усунення проблеми. Користувачі вирішували цю проблему самі, вимагаючи приватні підмережі, щоб отримати власний виділений канал. Клерки неохоче виділяли нові IP-підмережі, і користувачі підключали їх до решти мережі за допомогою маршрутизуючої підстанції з двома мережевими картами. Такі з'єднання були ненадійними, тому що вузли повільно маршрутизували пакети, особливо під час перевантаження. Чим більш перевантаженими ставали основні мережі, тим більше створювалося виділених підмереж. Зрештою повільна робота мережі стала, головним чином, зумовлена повільними з'єднаннями між цими приватними ділянками. Обчислювальні сервери компанії теж страждали від відсутності планування ємності. Користувачі встановлювали свої обчислювальні сервери навіть незважаючи на те, що проблеми зі швидкістю роботи, з якими вони стикалися, були, найімовірніше, пов'язані з повільною маршрутизацією, що забезпечується робочими станціями. Ці нові, швидкі сервери перевантажували мережу з пропускною здатністю 10 Мбіт/с. особливо через те, що часто вони були на порядок швидшими, ніж вузли, які виконували маршрутизацію.

На той час, коли організація найняла старшого системного адміністратора, мережа являла собою болото з ненадійних підмереж, погано налаштованих обчислювальних серверів і древніх файлових серверів. У мережі було 50 підмереж на приблизно 500 користувачів. Розчищення і модернізація мережі зайняла близько двох років.

Лаборант

Лаборант – це системний адміністратор вузькоспеціалізованого обладнання. Наприклад, у хімічній дослідницькій компанії лаборант може відповідати за невелику мережу, яка об'єднує всі мікроскопи та пристрої моніторингу. У компанії з виробництва телекомунікаційного обладнання лаборант може підтримувати все обладнання в приміщенні забезпечення сумісності протоколів, маючи в наявності примірник кожної версії продукту, продукти-конкуренти та набір генераторів трафіку. Лаборант несе відповідальність за встановлення нового обладнання, інтеграцію систем для спеціалізованих проєктів, а також повинен досить добре розумітися на сфері діяльності користувачів, щоб перекладати їхні бажання в завдання, які йому потрібно виконати. У лаборанта зазвичай є невелика мережа або група мереж, яка з'єднана з головною корпоративною мережею і залежить від неї за більшістю служб; якщо він недурний, він також заведе друзів у сфері корпоративних служб, щоб радитися з ними в технічних питаннях.

Цінність лаборантів для компанії полягає в тому, що вони дають змогу дослідникам зосередитися на проєктуванні експериментів, а не на їх виконанні. Крім того, лаборанти цінні своїми великими знаннями технічної інформації.

Лаборант отримує задоволення від вчасно завершеного експерименту або презентації. Проте, якщо він не отримує безпосередніх привітань від дослідників, з якими працює, він може образитися. Лаборантам слід пам'ятати, що їхні дослідники вдячні незалежно від того, показують вони це чи ні. Дослідники будуть довше працювати з лаборантами, якщо останніх запрошують на церемонії вручення нагород, вечері тощо.

Якщо ви граєте роль лаборанта, користувачі хочуть знати, чи можна щось зробити, а не як це буде зроблено. Вони хочуть, щоб їхні вимоги були виконані, хоча з'ясувати в них ці вимоги – ваш обов'язок. У цій сфері можуть сильно допомогти навички активного слухання.

Шукач продуктів

Шукач продуктів читає всі технічні журнали та огляди, і коли хто-небудь запитує: «Чи існує програмне забезпечення для стиснення кліпів?», він може порадити не тільки кілька систем стиснення кліпів, а й способи, за допомогою яких можна визначити, який із них найкраще підходить для конкретної програми. Він також знає, де можна знайти подібний продукт.

Цінність такого шукача для компанії полягає в його здатності бути в курсі останніх новинок. Керівникам не потрібно пильно стежити за цією людиною, бо вони жахнуться, виявивши, що вона проводить половину робочого дня за читанням журналів і веб-сайтів. Керівники повинні порівняти це з тим часом, який шукач економить усім іншим.

Шукачі продуктів отримують задоволення, коли всі використовують правильні продукти. Ці люди можуть дратувати інших співробітників групи, навіть тих, кому вони допомагають, тому що всі хочуть мати час на те, щоб «полазити» в Інтернеті й бути в курсі новинок, але в більшості людей (мимоволі) інші пріоритети.

Якщо ви є шукачем продуктів, користувачам потрібні короткі зведення, а не деталі. Якщо ви описуватимете всі деталі, які з'ясували з цієї теми, що подібно до

багатогодинного читання довгої і сумбурно написаної статті, вони вас уникатимуть. Будьте небагатослівні.

Проєктувальник рішень

Проєктувальники рішень відіграють ключову роль у компанії. Незабаром після того, як вони почують, у чому полягає проблема, у них уже буде рішення, краще, ніж будь-хто міг очікувати. Це може стосуватися як невеликих питань, як-от встановлення сервера електронного факсу для спрощення процедури надсилання факсів, так і великих питань, наприклад створення електронної версії бухгалтерії. На відміну від шукачів продуктів, проєктувальники рішень з більшою часткою ймовірності створять щось із нуля або інтегрують кілька менших продуктів.

Цінність проєктувальників рішень для компанії полягає в їхній здатності усувати перешкоди та спрощувати бюрократичні процеси.

Проєктувальник рішень отримує задоволення від свідомості того, що люди охоче використовують його рішення, оскільки це означає, що людям вони подобаються.

Якщо ви є проєктувальником рішень, користувачам потрібно, щоб проблему було вирішено відповідно до їхнього бачення, а не того, що ви розумієте під проблемою або що заощадить компанії гроші. Наприклад, якщо звіти про витрати надсилаються факсом до головного офісу, ви маєте створити спосіб, що дає змогу передавати дані в електронному вигляді, щоб у головному офісі не доводилося передруковувати всі дані. Однак ваші клієнти не отримають користі від того, що ви заощадите час співробітникам головного офісу; їм потрібно просто спростити підготовку документа. Це можна вирішити створенням кращого користувальницького інтерфейсу або системи, яка могла б завантажувати їхній корпоративний кредитний рахунок із сайту провайдера служби. Ваших користувачів не цікавитиме, чи буде результат потім надіслано факсом до головного офісу для повторного ручного введення.

Шукач спеціалізованих рішень

Шукач спеціалізованих рішень може в екстреній ситуації створити рішення для, здавалося б, нерозв'язної проблеми. Це людина, яка може якимось магічним шляхом встановити безпечне з'єднання з Місяцем, щоб представити вашу велику презентацію його мешканцям. Такі люди можуть знати про засоби більше, ніж звичайна людина, яка користується цими засобами, можливо, завдяки їх вивченню. На відміну від героя, який повертає ситуацію в нормальне русло, усуваючи проблеми, ця людина створює рішення.

Цінність шукача спеціалізованих рішень полягає в його здатності знаходити рішення, незважаючи на той факт, що технологія не є настільки гнучкою, як того вимагають деякі особливі ситуації, або на те, що ваша корпоративна мережа має слабкі місця, виправленням яких ви ще не займалися. Перше – це ситуація, яка з часом стає кращою. Друге показує відсутність належного технічного управління.

Шукач спеціалізованих рішень отримує задоволення, рятуючи ситуацію. Як і герой, шукач спеціалізованих рішень може «згоріти» на роботі від перевантажень.

Якщо граєте роль шукача спеціалізованих рішень, користувачі хочуть, щоб відбувалися дива, і не хочуть нагадувань, що надзвичайної ситуації можна було

уникнути за рахунок кращого планування, оскільки це рідко стосується їхніх помилок.

Шукач незатребуваних рішень

Деякі системні адміністратори займаються тим, що впроваджують рішення, які не були затребувані. Це може бути як добре, так і погано. Одного системного адміністратора нагородили за встановлення користувачам системи надсилання факсів без використання паперу, яку не просили ставити, але яка незабаром призвела до серйозного підвищення продуктивності. Вона була заснована на відкритому програмному забезпеченні і використовувала вже наявний модемний пул, так що реальна вартість цієї системи була нульовою. Цьому ж системному адміністратору одного разу оголосили догану за те, що він надто багато часу проводив за «самодіяльними проєктами», і змусили зосередитися на завданнях, які він має вирішувати.

Цінність для компанії співробітників, які займаються незатребуваними рішеннями, полягає в тому, що зазвичай вони перебувають ближче до користувачів і в змозі побачити ті їхні потреби, які вище керівництво не побачило б або не зрозуміло. Крім того, ці системні адміністратори краще обізнані про нові продукти, ніж їхні менш технічно просунуті користувачі.

Люди на цій посаді отримують задоволення, коли виявляють, що їхні здогадки про те, що може бути корисним, виявляються правильними.

Якщо ви перебуваєте на цій ролі, користувачі хочуть, щоб ви правильно здогадувалися про те, що буде або не буде для них корисним; дуже важливо регулярно, у визначений час, з ними розмовляти. Вони турбуватимуться, чи не позначаться ці нові проєкти негативно на термінах виконання безпосередньо ваших проєктів, а особливо їхніх власних. Керівництво турбуватиметься про вартість вашого робочого часу та будь-які матеріальні витрати, особливо коли незатребувана служба не використовується.

Експерт за викликом

Експерт за викликом завжди може дати пораду. Ця людина зарекомендувала себе як така, що знається на всіх або майже на всіх аспектах системи. Іноді експерт за викликом має вузьку спеціалізацію; в інших випадках його знання є всебічними.

Цінність експертів за викликом для компанії полягає в тому, що до цієї людини завжди можна звернутися, коли людям необхідна порада – або точне рішення, або просто хороша початкова точка для досліджень.

Експерт за викликом отримує задоволення від допомоги людям і від гордості за займану роль. Оскільки технології стрімко розвиваються, йому потрібен час для розширення своїх знань, чи то читання відповідних журналів, участь у конференціях, чи то експерименти з новими продуктами.

Якщо ви є експертом за викликом, вам слід нагадувати людям, що можна працювати і самостійно. В іншому разі ви не впораєтеся з усіма взятими зобов'язаннями.

Викладач

Викладач навчає користувачів працювати з доступними службами. Викладач може відволіктися від процесу усунення неполадки з принтером, щоб пояснити користувачеві, як краще користуватися електронними таблицями, і часто пише більшу частину користувацької документації.

Викладач важливий для компанії, тому що результатом його роботи є більш ефективне використання робочих інструментів. У викладача тісні взаємовідносини з користувачами, внаслідок чого він знає, які в людей є проблеми. Він стає засобом, через який можна дізнатися про потреби користувачів.

Викладач отримує задоволення від усвідомлення того, що його документацію використовують і поважають і що люди працюють краще завдяки його праці.

Якщо ви перебуваєте на ролі викладача, користувачі хочуть, щоб ви розумілися на сфері їхньої діяльності, на особливостях їхньої роботи і, найважливіше, на тому, що їх не влаштовує в засобах, якими вони користуються. Вони хочуть, щоб документація містила відповіді на запитання, які в них виникають, а не на ті, які є важливими на думку розробників.

Блюститель політики

Блюститель (охоронць) політики – це людина, яка каже «ні», коли хтось хоче зробити щось, що суперечить політиці, а також присікає дії порушників. Охоронець політики однаково залежить від двох чинників: від письмових правил і підтримки керівництва. Правила мають бути записані й опубліковані, щоб усі були про них обізнані. Якщо правила ніде не записані, дії охоронця будуть безпідставними, оскільки йому доведеться створювати правила на ходу, а його колеги можуть мати іншу думку щодо того, що правильно, а що ні. Другий фактор – це підтримка керівництва. Політика не має сили, якщо керівництво змінює правила щоразу, коли хтось просить зробити виняток. Керівнику не слід затверджувати політику, а потім постійно змінювати її при проханні зробити виняток. Найчастіше блюститель політики має повноваження відключити мережевий кабель, якщо порушення створює глобальні проблеми, а зв'язатися з порушником швидко неможливо. Якщо керівництво не підтримує рішення блюстителя політики, він не зможе виконувати свою роботу. Якщо керівництво підтверджує політику, але потім дозволяє виняток після відмови блюстителя, він втратить вплив і бажання або підстави продовжувати свою діяльність.

Цінність блюстителя політики для компанії полягає в тому, що політика компанії приводиться у виконання. Якщо важливу політику не здійснювати повністю, то сенс її наявності втрачається.

Блюститель політики отримує задоволення, усвідомлюючи, що він намагається утримувати курс компанії відповідно до напряму, встановленого керівництвом, а також забезпечуючи жорстке дотримання правил по всій компанії.

Якщо ви є блюстителем політики, користувачі хочуть просто виконувати свою роботу і не розуміють, чому так багато перешкод (правил) заважають їм це зробити. Замість того щоб говорити «ні», може бути правильніше з'ясувати, чого вони хочуть досягти, і допомогти їм досягти своїх цілей, не порушуючи політики.

Перестрахувальник

Комусь у групі слід турбуватися про те, що щось може піти не так. Коли пропонується рішення, ця людина запитає: «А що станеться в разі невдачі?» Звісно, ця людина не може визначати всі рішення, інакше проєкти ніколи не будуть завершені або вийдуть за межі бюджету. Цю людину слід врівноважити оптимістом. Однак, якщо ніхто не стежить за потенційними небезпеками, група може побудувати «картковий будиночок».

Цінність для компанії перестраховика відчувається тільки в надзвичайних ситуаціях. Половина системи відмовила, але інша половина продовжує працювати завдяки грамотно встановленим засобам управління. Результатом роботи цієї людини може бути загальна відмовостійкість системи.

Ця людина отримує задоволення від власної впевненості в безпеці та стабільності.

Якщо ви перебуваєте на цій посаді, люди навколо вас можуть втомитися від того, що ви постійно вимагаєте, щоб вони «пристебнули паски». Важливо втілювати свої рішення в життя, а не просто висловлювати свою думку на кожному кроці. Ніхто не любить слухати скарги на кшталт «Цієї невдачі не сталося б, якби люди мене послухали» або «Наступного разу ви не будете так просто мене ігнорувати». Можливо, краще розділяти відповідальність, замість того щоб визначати винних, і працювати над подальшим поліпшенням, замість того щоб таємно радіти своїй прозорливості: «У майбутньому нам буде потрібно написати скрипти, які справляються із ситуацією переповнення дисків». Ввічливий інструктаж віч-на-віч набагато ефективніший, ніж публічні скарги.

Обережний проєктувальник

Обережний проєктувальник довго планує кожен крок проєкту, у якому він бере участь. Він будує хороші тестові макети і ніколи не бентежиться, коли щось іде не так, тому що вже з'ясував, що робити.

Цінність обережного проєктувальника для компанії полягає в тому, що він виконує важливі завдання надійно та бездоганно.

Ця людина отримує задоволення від завершення завдання й усвідомлення того, що воно завершене належним чином, а також спостерігаючи, як перші користувачі використовують його без затримок. Вона пишається своєю роботою.

Якщо ви граєте цю роль, інші можуть звикнути покладатися на те, що ваша робота бездоганна. Вам часто дають завдання, в яких не можна припускатися помилок. Продовжуйте працювати так, як ви завжди працювали, і не дозволяйте важливості завдань переважити всі інші чинники. Не забувайте, що ваша ретельна робота потребує часу, а інші постійно поспішають і можуть обуритися, дивлячись на те, як ви працюєте. Переконайтеся, що ви розвиваєте талант прогнозування того, як багато часу вам знадобиться, щоб завершити завдання. Ви ж не хочете, щоб вас вважали людиною, яка не може виконати завдання вчасно.

Проєктувальник пропускної здатності

Проєктувальник пропускної здатності розширює систему в міру її зростання. Ця людина помічає, коли що-небудь переповнюється, виснажується або стає перевантаженим. Хороші проєктувальники пропускної здатності приділяють увагу

структурам використання і знають про зміни бізнесу, які можуть їх зачепити. Вправні проєктувальники пропускної здатності встановлюють системи, які проводять такий моніторинг автоматично і створюють графіки, що передбачають, коли пропускної здатності буде недостатньо. Виробники можуть допомогти проєктувальникам пропускної здатності, документуючи дані, які можуть бути їм корисними, як-от необхідна кількість оперативної пам'яті та дискового простору залежно від кількості користувачів.

Цінність проєктувальників пропускної здатності для компанії полягає в тому, що вони запобігають заторам трафіку. Це ще одна роль, яка залишається непоміченою, якщо роботу виконано належним чином. (Типова ситуація – підрозділи намагалися прискорити роботу сервера, додаючи більше оперативної пам'яті, хоча реальною проблемою була перевантажена мережа, і навпаки).

Проєктувальник пропускної здатності отримує задоволення від усвідомлення того, що проблемам запобігли і що люди беруть попередження до уваги, а також від визначення реального джерела проблем.

Якщо ви є проєктувальником пропускної здатності, користувачі хочуть, щоб у вас були точні дані та рішення, які не коштуватимуть грошей. Виправдовувати витрати – ваша робота. Як завжди, дуже важливо пояснити ситуацію мовою клієнта.

Адміністратор бюджету

Адміністратор бюджету веде рахунок грошам, які залишилися в бюджеті, і допомагає скласти бюджет на наступний рік. Ця людина знає, на що слід спрямувати грошові витрати, коли це необхідно, і як можна розтягнути бюджет ширше.

Цінність адміністратора бюджету для компанії полягає в тому, щоб контролювати витрати на системне адміністрування, забезпечувати фінансування завдань, які потребують виконання, у межах розумного – навіть якщо вони були несподіваними, і надавати достовірні відомості, щоб керівництво могло здійснити фінансове планування на наступний рік.

Адміністратор бюджету отримує задоволення від того, що він не виходить за межі бюджету і водночас справляється з фінансуванням додаткових важливих проєктів, які не були включені до бюджету.

Якщо ви є адміністратором бюджету, користувачі хочуть, щоб ви залишалися в межах бюджету, готували гарний бюджетний план на наступний рік, точно визначали, які проєкти є найважливішими, щоб забезпечити фінансування всіх пріоритетних завдань, і показували, яким чином гроші, які вони дозволили вам витратити, приносять їм вигоду.

Адвокат користувачів

Адвокат користувачів може допомогти людині заявити про її потреби. Він перекладач і лобіст, посередник між користувачем і його керівництвом. Адвокат не просто рекомендує рішення, а й пояснює користувачеві, як презентувати його ідею керівнику, і присутній при цьому, на випадок якщо людині знадобиться допомога.

Цінність адвоката користувачів для компанії полягає в тому, щоб допомагати користувачам отримувати те, що їм потрібно, незважаючи на бюрократію і труднощі в спілкуванні.

Адвокат отримує задоволення від усвідомлення того, що він комусь допоміг. Він також знає, що, взаємодіючи з керівництвом, він може представити свою групу системного адміністрування в хорошому світлі, і відіграє роль корисного координатора. Часто ви допомагаєте користувачеві отримати те, що він хоче, за допомогою системи, а не обходячи її. Це особливо важливо, якщо ви самі брали участь у створенні цієї системи.

Якщо ви є адвокатом, користувачі хочуть, щоб ви зрозуміли їх, перш ніж почнете пропонувати рішення.

Технократ

Технократ – це прихильник нових технологій. Коли систему необхідно полагодити або замінити, він віддає перевагу новій системі, навіть якщо вона ще не доведена до бездоганного стану. Він зневажає тих, хто вважає зручними старі системи, які ще «досить гарні». Технократ може бути хорошою противагою перестраховику.

Цінність технократа для компанії полягає в тому, що він утримує компанію від технічного застою.

Технократ отримує задоволення від того, що занурений у море останніх технологій, – можна сказати, від синдрому «нової іграшки».

Якщо ви технократ, користувачі хочуть, щоб ви зосередилися на реальній цінності рішення, а не бездумно віддавали перевагу всьому новому.

Продавець

Продавець не обмежений лише матеріальними аспектами. Він може продавати конкретну політику, нову службу або план. Він може продавати саму групу системного адміністрування – або вищому керівництву, або користувачам. Продавець займається тим, що з'ясовує потреби користувачів, а потім переконує їх у тому, що його товар відповідає їхнім потребам. Нові служби легше продати, якщо користувачі були залучені до процесу визначення характеристик і вибору засобів.

Цінність продавця для компанії полягає в тому, що він спрощує роботу групи системного адміністрування. Чудова система, яку ніколи не приймуть користувачі, марна для компанії. Чудова політика, яка економить компанії гроші, марна, якщо користувачі оминають її, тому що не розуміють її переваг. Продавець отримує короткочасне задоволення від продажу, але для справжнього, тривалого задоволення продавець повинен розвинути взаємовідносини з користувачами і відчувати, що сильно допомагає їм.

Якщо ви є продавцем, користувачі хочуть, щоб ви розуміли і поважали їхні потреби. Вони хочуть, щоб із ними розмовляли, а не вказували.

Контактер із постачальником

Контактер із постачальником підтримує зв'язок з одним або більше постачальниками. Він може знати лінію продуктів постачальника краще, ніж будь-хто ще в групі, і бути в курсі наступних продуктів. Він є ресурсом для інших системних адміністраторів, таким чином скорочуючи час дзвінків продавцю виробника.

Цінність для компанії контактера з постачальником полягає в тому, що це людина, яка розуміє і спеціалізується на потребах компанії щодо взаємодії з постачальником.

Контактер із постачальником отримує задоволення від того, що він є експертом, якого всі поважають, від того, що він перший дізнається новини виробника, і від безкоштовних обідів і футболок, які він отримує.

Якщо ви є контактером із постачальником, користувачі хотітимуть, щоб ви знали все про виробника, неупереджено ставилися до виробників-конкурентів і жорстко торгувалися під час встановлення цін.

Провидець

Провидець дивиться на загальну картину і має уявлення про напрямок, у якому компанії слід іти.

Цінність провидця для компанії полягає в тому, що він підтримує зосередженість групи на тому, що відбувається.

Провидець отримує задоволення, коли він озирається на багато років назад і бачить, що в довгостроковій перспективі його робота дуже важлива. Усі ці поступові поліпшення в підсумку привели до виконання основних завдань.

Якщо ви є провидцем, користувачі хочуть знати, що станеться в майбутньому і не хочуть занадто сильно турбуватися про довгострокову перспективу. Репутація вашої групи в сенсі здібностей щодо виконання плану впливає на можливість продавати ваші задуми користувачам.

«Мати»

«Мати» опікується користувачами. Це складно пояснити без прикладу. Один системний адміністратор щоранку ходив кімнатами, зупиняючись біля робочого столу кожної людини, щоб подивитися, як ідуть справи. Він міг виправити дрібні неполадки і нагадати про більші проблеми поточного дня. Він відповідав на безліч дрібних запитань щодо інтерфейсу користувача, які, на думку користувача, були занадто несуттєві, щоб ставити їх службі підтримки. Користувачі здійснювали великий перехід (з X-терміналів на настільні емулятори X-терміналів), і така материнська турбота була якраз тим, що їм було потрібно. За ці ранкові прогулянки він відповідав на сотні запитань і розв'язував десятки проблем, які в іншому разі були б надіслані в службу підтримки. Користувачі звикли до такого рівня обслуговування і незабаром стали покладатися на його ранкові візити як на один із факторів, що підвищують продуктивність їхньої діяльності.

Цінність «матері» для компанії полягає у високому ступені підтримки, що може бути дуже важливо за часів великих змін або під час обслуговування нетехнічних користувачів. Особистісне спілкування також додає впевненості в тому, що потреби клієнтів будуть зрозумілі точно.

«Мати» отримує задоволення від людяності взаємин, які вона розвиває зі своїми клієнтами.

Якщо ви є «матір'ю», користувачі хочуть знати, що їхні нагальні потреби виконано, і стануть приділяти менше уваги довгостроковій стратегії. Ви ж повинні не забувати про майбутнє і не надто сильно занурюватися в сьогодення.

Спостерігач

Спостерігач стежить за тим, наскільки добре йдуть справи. Іноді спостерігач використовує застарілі технології, застосовуючи ті самі служби, що і його користувачі. Хоча системні адміністратори можуть мати власний файловий сервер, ця людина зберігає свої файли на одному сервері з клієнтами, щоб вона могла «відчути їхній біль». У міру того як ця людина стає більш досвідченою, вона автоматизує моніторинг, а потім переглядає вихідні дані системи моніторингу і витрачає час на виправлення неполадок, а не просто стирає попередження.

Цінність спостерігача для компанії полягає в тому, що проблеми помічаються до того, як користувачі починають скаржитися. Це може створити враження, що мережа працює безвідмовно.

Спостерігач отримує задоволення від того, що першим помічає проблему і розв'язує її до того, як користувачі про неї повідомлять, а також від усвідомлення, що проблемам запобігли за допомогою моніторингу характеристик пропускної здатності.

Якщо ви є спостерігачем, користувачі, найімовірніше, не знатимуть про ваше існування. Якби вони знали, вони б захотіли, щоб ваше тестування симулювало їхнє реальне робоче навантаження і було наскрізним. Наприклад, недостатньо знати, що сервер електронної пошти працює. Ви маєте переконатися, що повідомлення можна надіслати, передати, доставити та прочитати.

Координатор

У координатора чудові навички спілкування. Він прагне перетворити спонтанні дискусії на зустрічі з ухвалення рішень. Його часто просять провести нараду, особливо велику, на якій складно підтримувати зосередженість на питанні.

Координатор важливий тим, що забезпечує більш спокійне протікання процесів. Він може і не вживати багато заходів, але допомагає групам людей домовитися про те, що необхідно зробити і хто буде це робити. Він робить наради ефективними і живими. Координатор отримує задоволення від вигляду людей, які дійшли згоди у своїх цілях, і від прояву ініціативи для досягнення цих цілей.

Якщо ви є координатором, інші члени вашої групи хочуть, щоб ви координували всі їхні переговори. Важливо навчати інших людей бути координаторами і створювати середовище, в якому всі мали б хороші навички спілкування.

Користувач/системний адміністратор

Іноді користувач одночасно є і системним адміністратором, можливо, тому, що користувач має якісь певні обов'язки, які потребують привілейованого доступу, або він раніше був системним адміністратором і зберіг деякі з цих обов'язків.

Цінність користувача/системного адміністратора для компанії полягає в тому, що він може підмінити інших системних адміністраторів, коли вони перебувають у відпустці. Насправді в ситуаціях, коли є тільки один системний адміністратор, дуже корисно навчити когось із користувачів питань, пов'язаних зі щоденними операціями, щоб він зміг замінити системного адміністратора на час відпустки. Може бути дуже корисно мати додаткову людину, яка вміє змінювати стрічки в системі резервного копіювання, створювати облікові записи користувачів і вирішувати десять найпоширеніших проблем.

Користувач/системний адміністратор отримує задоволення від своєї ролі, якщо він має доступ системного адміністратора або привілейованого користувача. Крім того, він може отримувати свого роду «бойову надбавку» за навчання суміжної професії.

Якщо ви є користувачем/системним адміністратором, основна група системного адміністрування хоче знати, що ви не заважаєте їхнім планам, дотримуєтеся правильних процедур і дотримуєтесь тих самих етичних правил, що й вони. Інші користувачі хочуть знати, що ви зможете вирішити багато проблем, якщо вас про це попросять.

Помічник користувачів

Системний адміністратор, який допомагає користувачам, бачить свою роботу в тому, щоб концентруватися на запитах живих користувачів, а не на технічних процесах, у яких він бере участь. Він розглядає свою роботу як допомогу користувачам у роботі з системою, яка досить статична. Основні зміни в системі походять від зовнішніх сил, таких як група програмування систем.

Системний адміністратор, який допомагає користувачам, важливий для компанії як завжди доступний живий представник усієї групи системного адміністрування. Багато користувачів ніколи не бачать задню лінію підтримки.

Системний адміністратор, який допомагає користувачам, отримує задоволення від особистих взаємин, які він розвиває, і від теплого почуття, викликаного усвідомленням, що він допоміг реальній живій людині.

Якщо ви є системним адміністратором, який підтримує користувачів, останні хочуть, щоб їхні завдання виконувалися відповідно до їхнього розкладу. Якщо вони виявляють, що ситуація екстрена, вони очікують, що ви все кинете і прийдете їм на допомогу.

Навігатор з політики

Навігатор з політики розуміє правила і норми бюрократичної системи і може допомогти іншим проходити через них або в обхід них. Навігатор може допомогти будь-кому пройти через бюрократичний процес або обійти політику без її порушення.

Навігатор з політики важливий для групи системного адміністрування, оскільки допомагає швидше виконувати завдання, коли доводиться мати справу з бюрократичною системою і коли необхідно обійти систему, не піддаючи її ризику порушення.

Навігатор з політики отримує задоволення від усвідомлення, що його зв'язки і знання сприяли виконанню проєкту.

Якщо ви є навігатором з політики, ваші користувачі хочуть, щоб завдання було виконано, незалежно від того, чи залишаєтеся ви в межах норм системи. Це може поставити вас у скрутне становище, коли користувачеві здається, що легше порушити політику або працювати в обхід системи.

1.9.2 Негативні ролі

Далі описано деякі негативні ролі, яких слід уникати.

Екстремал

Іноді, системний адміністратор може бути настільки захопленим новою технологією, що шукає шляхи випробувати її на користувачах до того, як вона буде розроблена до стану готовності. Замість того щоб бути на передньому краї, він тримає компанію на вістрі технологій. У такому разі виходить, що користувачі завжди страждають від недоліків нових, ще не відпрацьованих служб.

Консерватор

Протилежністю екстремала є співробітник, який відтягує використання будь-якої нової технології. У цієї людини, яка прагне уникнути ризику, завжди є улюблене виправдання, наприклад незадоволеність планом скасування змін. Вона задоволена поточною версією операційної системи, поточним дистрибутивом операційної системи, поточною маркою комп'ютера, кількістю каналів зв'язку і типом мережевої топології. Ця людина не помічає відсутності оновлення технологій і проблем, до яких це призводить. За іронією долі раніше ця людина завжди була на передових позиціях, але тепер загрузла в рутині. Вона може бути тією самою людиною, яка відмовилася від мейнфреймів і освоювала UNIX або сміялася над користувачами робочих станцій, які не переходили на персональні комп'ютери. Однак цей час минув, вона знайшла щось, із чим їй зручно, і тепер сама стала такою людиною, з якої посміялася б багато років тому.

Панікер

Ця людина турбується про те, що ще не відбувається і з великою часткою ймовірності ніколи не станеться. Майже все системне адміністрування пов'язане з управлінням ризиками, але ця людина вважає, що будь-який ризик неприйнятний. Ця людина уповільнює проєкти і не дає їм зрушити з мертвої точки. Вона пророкує загибель або невдачу без будь-яких фактів, що підтверджують це. Іноді вона просто відчуває дискомфорт від своєї нездатності щось контролювати або від нестачі навчання. Часом ця людина витрачатиме багато робочого часу, працюючи над проблемами, яких ви навіть не бачите, та ігноруватиме більш термінові завдання. Найбільша небезпека цієї людини в тому, що, коли вона дійсно матиме рацію, її проігнорують.

Ковбой

Ковбой приступає до усунення неполадок у системах або розроблення нових служб без належного планування, роздумів про наслідки або розроблення плану скасування змін. Він не думає про те, щоб попередньо запитати свого керівника або порадитися з користувачами. Він не перевіряє належним чином результати своєї праці, щоб переконатися, що все працює, і йде додому, нікому не повідомляючи про це. Він вважає себе винятково талановитим і продуктивним і думає, що інші намагаються поставити на його шляху занадто багато бюрократичних перешкод і що вони недооцінюють його таланти. Ковбой нічого не документує і просто знає, що він безцінний для компанії.

Приклад. У компанії середнього розміру, що виробляє апаратне забезпечення для комп'ютерів, посаду старшого системного адміністратора обіймав ковбой. Його керівництво залучило групу консультантів, щоб перепроєктувати мережу і

побудувати план переходу від старої мережі до нової. План було підтверджено, обладнання замовлено, розклад складено й погоджено з клієнтами, і плани з тестування було створено спільно з користувачами. Коли прийшло нове обладнання, ковбой затримався на роботі, викинув усе старе обладнання і встановив нове. Він не використовував нову архітектуру; він ігнорував усі пункти з плану переходу, що стосувалися непов'язаних завдань, які потрібно було вирішити в процесі переходу; він також не обтяжував себе проведенням тестування. Наступного дня, коли люди прийшли на роботу, багато хто з них виявив, що у них немає мережевого з'єднання, – серед них були генеральний директор, цілий відділ підтримки користувачів, низка керівництва ковбоя і багато інженерів. Служба підтримки та інша група системного адміністрування не мали ні найменшого поняття, що сталося, тому що він нікому нічого не сказав і навіть не спромігся прийти на роботу в цей день вчасно – зрештою, він же затримався вчора на роботі! Він був усе ще гордий тим, що зробив, і зовсім не відчував сорому, тому що вважав план переходу і тестування зайвою тратою часу і грошей. Він насолоджувався, показуючи, що зміг зробити все це сам за кілька годин, тоді як консультантам для цього знадобилося набагато більше часу. Він не звернув уваги на вартість того масштабного збою, який він спричинив, і шкоди, завданої репутації групи системного адміністрування.

Раб, цап-відбувайло чи швейцар

Іноді системні адміністратори грають роль рабів, цапів-відбувайлів або швейцарів. Раби – це співробітники, які виконують усі вимоги без запитань, навіть якщо вони могли б запропонувати краще рішення, поглянувши уважніше на картину загалом. Іноді інші люди використовують системних адміністраторів як цапів-відбувайлів. Їх звинувачують у всьому поганому, що відбувається. Системних адміністраторів звинувачують у тому, що проєкт запізнюється, навіть якщо користувачі не повідомляли їм про свої потреби. Іноді системні адміністратори розглядаються як швейцари: люди, які не становлять жодної цінності для бізнесу компанії, а є некваліфікованими робітниками, від яких самі лише витрати. Усі три ролі пов'язані з проблемами керівництва, яке не розуміє завдання системного адміністрування у своїй власній організації. Однак обов'язок системних адміністраторів – виправити такий стан речей, покращуючи взаємодію та працюючи над помітністю своєї групи в організації.

1.9.3 Групові ролі

Деякі групові ролі повинні бути присутніми в групі системного адміністрування, за винятком мученика.

Наскрізний експерт

Наскрізний експерт розуміє використовувану технологію з найнижчих до найвищих рівнів. Він украй важливий для розв'язання неясних завдань, таких як масштабні відмови. Незрозумілі завдання, як правило, є результатом декількох одночасних збоїв або невідомих взаємодій між різними сферами, їхнє розв'язання вимагає знань у всіх сферах. Масштабні відмови зачіпають безліч підсистем і

вимагають глибокого розуміння загальної архітектури для виявлення реальної проблеми.

Сторонній

Під час тривалого збою системні адміністратори, які працюють над проблемою, іноді опиняються в ситуації, аналогічній творчому глухому куту письменників. Вони починають ходити по колу, не здатні зрушити ситуацію з мертвої точки. Роль, яка тут найкраще підходить, – це сторонній, який привнесе свіже бачення ситуації. Якщо попросити когось пояснити те, що відбувається в цей момент, люди зазвичай знаходять рішення. Іноді роль цієї людини полягає в тому, щоб переконати інших попросити допомоги ззовні або передати проблему у вищі органи чи службу підтримки виробника. В іншому випадку роль цієї людини може полягати в тому, щоб просто визначити, що настав час здатися: є план скасування, і його потрібно застосувати.

Спеціаліст за рівнями

Це людина, яка визначає, на якому рівні потрібно вирішувати конкретне завдання. Люди часто схильні думати, що ті чи інші проблеми можна вирішити на їхньому рівні. Техніки вважають, що їм потрібно старанніше працювати. Програмісти думають, що їм необхідно впровадити нове програмне забезпечення, щоб вирішити проблеми. Однак важливу роль відіграє людина, яка розуміє устрій усіх рівнів, включно з керівництвом, і може підказати, на якому з них найкраще розв'язати проблему. Після тижнів, проведених у спробі розв'язати проблему, ця людина заявить, що дешевше і, можливо, ефективніше буде просто звернутися на верхній рівень керівництва, щоб сказати, що цей метод неможливо виконати. Можливо, краще знайти найближчого за старшинством керівника в структурі організації, який має вищу посаду, ніж два користувачі, що сперечаються, і попросити його вирішити цю проблему. Крім того, найімовірніше, це та людина, яка нагадає вам старий афоризм Інтернету: «Технології не можуть розв'язати соціальні проблеми», і почне пошук заміни політики.

Мученик

Мученик вважає, що ніхто не виконує так багато роботи, як він. Обурений тим фактом, що ніхто, крім нього, не працює так довго, і незадоволений відсутністю друзів або фінансового успіху, він не може зрозуміти, як інші люди можуть «зробити це», коли вони роблять для цього так мало. Ця людина проводить багато часу, оплакуючи проблеми світу – здебільшого її світу. Це може статися внаслідок простого морального і фізичного виснаження або низької самооцінки.

Моральне та фізичне виснаження відбувається, коли людина не встановлює рівновагу між роботою та відпочинком. На жаль, у сучасному інтенсивному житті деякі люди виростають, не розуміючи необхідності відпочинку. Вони вважають, що вони завжди мають бути «в роботі». Цей крайній ступінь робочої етики може бути секретом їхнього успіху, але без духовного еквівалента профілактичного ремонту він веде до морального і фізичного виснаження. У цьому разі старанніша праця тільки погіршує ситуацію. Людям, які досягли цієї межі, може не допомогти навіть тривалий

відпочинок; і вони можуть стати ще більш роздратованими, якщо їм нема чого робити.

Самооцінка – це те, що ми набуваємо (або не набуваємо), коли ми молоді. Теоретики пізнання вважають, що наш душевний стан – смуток чи радість – це показник не того, погані чи хороші події з нами відбуваються, а того, як ми на них реагуємо. Ми можемо бути незадоволені будь-якими хорошими подіями, якщо наша самооцінка була пошкоджена до такої міри, що в будь-якій ситуації ми відчуваємо власну нікчемність. Коли таке відбувається, ми перетворюємо хороші новини та події на привід для занепокоєння: «Йому сподобалася моя робота над тим проєктом! Що як я не зможу щоразу виправдовувати ці очікування? Що якщо мої колеги стануть мені заздрити і образяться на мене?»

Той, хто виконує монотонну роботу

Деякі типи особистостей мають схильність до монотонної роботи. Цих людей слід цінувати, оскільки вони вирішують невідкладні завдання. Неможливо переоцінити важливість автоматизації, але деякі процеси не можна автоматизувати, наприклад фізичну доставку нових машин, або вони недостатньо монотонні, щоб автоматизація була фінансово ефективною. Ця роль дуже важлива для групи. Ця людина може взяти на себе монотонну роботу, звільнивши від неї більш кваліфікованих співробітників. Ці маленькі завдання часто є чудовим тренуванням для завдань вищого рівня.

Громадський директор

Громадський директор підвищує моральний дух групи, знаходячи причини для спільних урочистостей; дні народження, ювілеї, дні прийняття на роботу – це лише мала їх частина. Спілкування людей у неробочій обстановці може сприяти створенню згуртованого колективу. Ключ успіху на цій посаді – бути впевненим у тому, що люди не відчуватимуть примусу і що ви не перегинаєте палицю, наражаючись на ризик того, що вашому начальнику така діяльність здасться марною тратою часу.

Пан Перерва

Під час надзвичайної ситуації дуже важлива присутність когось, хто помічає, що люди дуже втомилися, і переконує їх зробити перерву. Люди можуть вважати, що завдання надто важливе, щоб припинити роботу над ним, але ця людина бачить марність вимучених спроб і наполягає на перерві, яка всіх освіжить. Микола часто зникав і повертався з піцами та напоями. Часто, відволікаючись на деякий час від проблеми, люди зможуть подивитися на неї по-іншому і знайдуть кращі рішення.

Є й інші ролі, які можуть існувати в групі системного адміністрування, але перелічені вище заслуговують на особливу увагу.

ЛЕКЦІЯ 2. Сценарії й командна оболонка

2.1 Основи роботи з командною оболонкою

Командна оболонка bash забезпечує зручний інтерфейс керування UNIX-системами та дає змогу автоматизувати процес роботи за допомогою скриптів. Це значно скорочує часові витрати та зменшує кількість помилок, що напряму відображається на стабільності роботи системи. Розглянемо основні принципи побудови та використання bash на прикладі застосування базових команд та створення найпростішого bash-скрипта.

2.1.1. Bash – основні поняття

Bash (Bourne again shell) є вдосконаленим варіантом відомої командної оболонки *sh* (Bourne shell). Її основне призначення – забезпечити взаємодію між користувачем та ядром операційної системи. Така взаємодія може відбуватися двома шляхами. По-перше, за допомогою набраних у терміналі команд (інтерфейс командного рядка). По-друге, шляхом запуску на виконання заздалегідь підготовлених файлів сценаріїв (shell-скриптів).

Для забезпечення вказаних можливостей, у *bash* передбачена наявність власної системи команд та мовних конструкцій. Внутрішні, або інтегровані у оболонку команди, під час роботи знаходяться у оперативній пам'яті комп’ютера, а зовнішні зберігаються у вигляді файлів у системному каталозі, зазвичай, у */usr/bin* або */bin*.

Внутрішні команди, зокрема, дозволяють виконувати наступні дії:

- Виконання операцій над файловою системою;
- Робота зі змінними;
- Управління сценаріями;
- Керування запущеними процесами;
- Забезпечення вводу-виводу з периферійних пристроїв.

Bash-сценарії або shell-скрипти дозволяють автоматизувати процес керування системою, починаючи з автоматизації процесу оновлення пакетів і закінчуючи майже повним контролем над роботою операційної системи. Їх перевагами над деякими із своїх «конкурентів», зокрема, є:

- «Прозорість» коду для користувача, оскільки команди зберігаються у текстовому форматі;
- Структурованість сценарію, тобто є можливість вказати послідовність виконання команд;
- Сумісність із більшістю UNIX-систем.

Можна виділити декілька умовно конкуруючих аналогів Bash. Це, наприклад, такі командні процесори:

- *sh* – «застарілий» стандарт для UNIX-систем;

- *ksh* та *zsh* – розширені варіанти sh;
- *csh* та *tcsh* – засновані на мові C та мають відповідний синтаксис;
- *mksh* – орієнтована на створення сценаріїв;
- *fish* – має спрощений склад команд і робить акцент на зручність роботи;
- *dash* – більш «легкий» програмний засіб з обмеженою функціональністю.

Робота з командним рядком Bash

Для полегшення вивчення матеріалу розіб'ємо наш розгляд команд на кілька логічних блоків, кожен з яких включає команди певної орієнтованості.

Отримання інформації про поточну оболонку та її налаштування

Для можливості повноцінної роботи з оболонкою необхідно орієнтуватися у програмному середовищі та вміти виконати елементарні дії, щоб отримати потрібну інформацію.

Спочатку з'ясуємо, яка оболонка активна для нашої системи Ubuntu 20.04. Для цього звернемося до змінної програмного середовища SHELL. Введемо у терміналі:

```
$ echo $SHELL
```

Ми отримали шлях до місцезнаходження файлів активної оболонки – */bin/bash*. Отже, це дійсно *bash*. Її характерною ознакою є специфічний формат запрошення у командному рядку, котрий має наступний вигляд:

ім'я поточного користувача@ім'я хоста:поточний каталог #($)

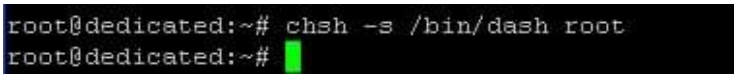
Символом у кінці рядка позначаються привілеї поточного користувача: # – для користувачів з правами суперкористувача *(root), $* – для звичайних користувачів. Отже, у нашому випадку користувачем є адміністратор, а хост має ім'я *dedicated*. Знак тільди (~) після імені хоста вказує на те, що ми знаходимося у домашній директорії суперкористувача *root*.

Для перегляду всіх доступних у системі shell-оболонок введемо у терміналі:

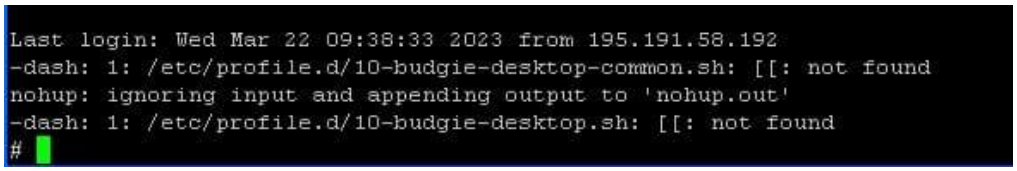
```
$ cat /etc/shells
```

Ми бачимо доволі значний перелік доступних командних процесорів, і у випадку чого, завжди можемо обрати будь-який з них. Для цього існує команда *chsh (change shell)* з параметром *s*. Для прикладу, змінимо поточний процесор на *dash*:

$ chsh -s /bin/dash root



Після виходу з системи і повторної авторизації під тим же ім'ям, ми бачимо активною оболонку *dash* замість попередньої:



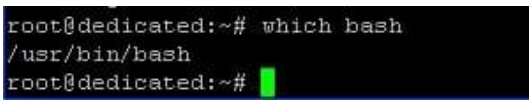
Вигляд запрошення у командному рядку став іншим. У ньому залишилася лише ознака привілей поточного користувача – *»#»*. Повернемо попередній процесор за допомогою наступної команди:

$ chsh -s /bin/bash root

Після виходу з системи та повторної авторизації ми знову повернулися до попереднього налаштування середовища:

Для того, щоб побачити шлях до розміщення файлів поточної оболонки також можна використати наступну команду:

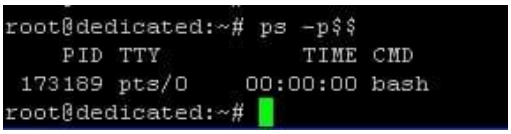
$ which bash



Можна переконатися, що команда *which* »працює».

Ще одним способом для з'ясування того, який командний процесор є активним на даний час, є використання команди *ps*, котра повертає інформацію про запущений процес із заданим ідентифікатором. Ідентифікатором для оболонки завжди є символи *»$$»*. Введемо у терміналі:

$ ps –p$$

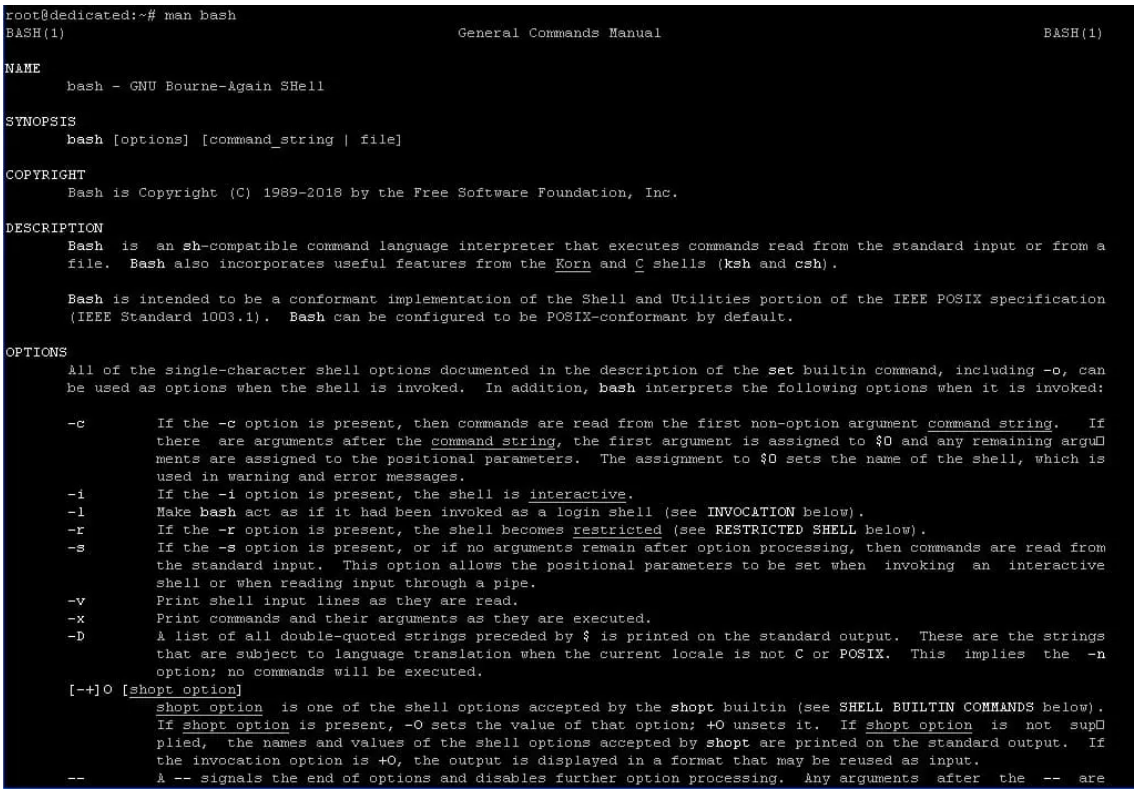


Отже, це також реальний шлях для визначення активної оболонки системи.

2.1.2. Отримання довідкової та іншої інформації про bash-процесор

Для отримання довідкової інформації про командний процесор, параметри запуску та іншої інформації, введемо у терміналі:

```
$ man bash
```



Після натискання на клавішу *»h»* отримаємо додаткову пояснювальну інформацію:

Натиснемо *»q»* та вийдемо з довідки.

Для з'ясування типу команд, котрі використовуються, тобто, чи є вони внутрішніми або зовнішніми, існує наступна конструкція:

type <ім'я_команди>

Для прикладу, визначимо тип команди *cd*. Введемо у терміналі:

$ type cd

```
root@dedicated:~# type cd
cd is a shell builtin
root@dedicated:~#
```

Таким чином, *cd* відноситься до внутрішніх команд. Тепер перевіримо таким же чином команду *whereis*:

$ type whereis

```
root@dedicated:~# type whereis
whereis is /usr/bin/whereis
root@dedicated:~#
```

Ми бачимо, що вказана команда є зовнішньою, оскільки вона зберігається у файлі, до якого виведено повний шлях доступу: */usr/bin/whereis*.

Можна, навіть, одразу визначити список внутрішніх команд процесора:

$ help

```
root@dedicated:~# help
GNU bash, version 5.0.17(1)-release (x86_64-pc-linux-gnu)
These shell commands are defined internally.  Type `help' to see this list.
Type `help name' to find out more about the function `name'.
Use `info bash' to find out more about the shell in general.
Use `man -k' or `info' to find out more about commands not in this list.

A star (*) next to a name means that the command is disabled.

 job_spec [&]                                            history [-c] [-d offset] [n] or history -anrw [filename] or h>
 (( expression ))                                        if COMMANDS; then COMMANDS; [ elif COMMANDS; then COMMANDS; ]>
 . filename [arguments]                                  jobs [-lnprs] [jobspec ...] or jobs -x command [args]
 :                                                       kill [-s sigspec | -n signum | -sigspec] pid | jobspec ... or>
 [ arg... ]                                              let arg [arg ...]
 [[ expression ]]                                        local [option] name[=value] ...
 alias [-p] [name[=value] ... ]                          logout [n]
 bg [job_spec ...]                                       mapfile [-d delim] [-n count] [-O origin] [-s count] [-t] [-u>
 bind [-lpsvPSVX] [-m keymap] [-f filename] [-q name] [-u name]>  popd [-n] [+N | -N]
 break [n]                                               printf [-v var] format [arguments]
 builtin [shell-builtin [arg ...]]                       pushd [-n] [+N | -N | dir]
 caller [expr]                                           pwd [-LP]
 case WORD in [PATTERN [| PATTERN]...) COMMANDS ;;]... esac      read [-ers] [-a array] [-d delim] [-i text] [-n nchars] [-N n>
 cd [-L|[-P [-e]] [-@]] [dir]                            readarray [-d delim] [-n count] [-O origin] [-s count] [-t] [>
 command [-pVv] command [arg ...]                        readonly [-aAf] [name[=value] ...] or readonly -p
 compgen [-abcdefgjksuv] [-o option] [-A action] [-G globpat] [>  return [n]
 complete [-abcdefgjksuv] [-pr] [-DEI] [-o option] [-A action] >  select NAME [in WORDS ... ;] do COMMANDS; done
 compopt [-o|+o option] [-DEI] [name ...]                set [-abefhkmnptuvxBCHP] [-o option-name] [--] [arg ...]
 continue [n]                                            shift [n]
 coproc [NAME] command [redirections]                    shopt [-pqsu] [-o] [optname ...]
 declare [-aAfFgilnrtux] [-p] [name[=value] ...]         source filename [arguments]
 dirs [-clpv] [+N] [-N]                                  suspend [-f]
 disown [-h] [-ar] [jobspec ... | pid ...]               test [expr]
 echo [-neE] [arg ...]                                   time [-p] pipeline
 enable [-a] [-dnps] [-f filename] [name ...]            times
 eval [arg ...]                                          trap [-lp] [[arg] signal_spec ...]
 exec [-cl] [-a name] [command [arguments ...]] [redirection ..>  true
 exit [n]                                                type [-afptP] name [name ...]
 export [-fn] [name[=value] ...] or export -p            typeset [-aAfFgilnrtux] [-p] name[=value] ...
 false                                                   ulimit [-SHabcdefiklmnpqrstuvxPT] [limit]
 fc [-e ename] [-lnr] [first] [last] or fc -s [pat=rep] [comman>  umask [-p] [-S] [mode]
 fg [job_spec]                                           unalias [-a] name [name ...]
 for NAME [in WORDS ... ] ; do COMMANDS; done            unset [-f] [-v] [-n] [name ...]
 for (( exp1; exp2; exp3 )); do COMMANDS; done           until COMMANDS; do COMMANDS; done
 function name { COMMANDS ; } or name () { COMMANDS ; }  variables - Names and meanings of some shell variables
 getopts optstring name [arg]                            wait [-fn] [id ...]
 hash [-lr] [-p pathname] [-dt] [name ...]               while COMMANDS; do COMMANDS; done
 help [-dms] [pattern ...]                               { COMMANDS ; }
root@dedicated:~#
```

Тобто, ми отримали повний список команд разом із параметрами їх запуску, що є дуже зручно.

2.2. Скрипти та їх використання

Як ми переконалися з результатів попереднього розгляду, можливості командного інтерфейсу *bash* є доволі потужними, але найбільший виграш з точки зору керування системою дає використання груп команд, або shell-скриптів. Такі файли, зазвичай, мають розширення *.sh* і можуть викликатися інтерпретатором кількома способами, про що ми поговоримо трохи пізніше.

Для прикладу, створимо найпростіший скрипт із ім'ям *one_lesson*, у складі якого буде лише одна команда *echo*. Введемо у терміналі:

```
$ nano one_lesson.sh
```

У результаті відкриється вікно редактора *nano*, де ми введемо наступний текст:

```
#!/bin/bash
#Це наш перший скрипт
echo “One lesson!”
```

Розберемо кожний рядок сценарію.

Перші два символи першого рядка (#!) дають вказівку інтерпретатору оболонки на виконання команд, вказаних у файлі, інакше він сприймав би їх як текст і не виконував би. Інші символи цього ж рядка вказують шлях до розміщення файлів інтерпретатора. У даному випадку, це *bash*.

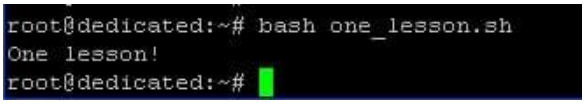
Другий рядок починається з символу *»#»*, що є ознакою коментаря, і тому інтерпретатор буде сприймати текст, вказаний після зазначеного символу лише як текст і, відповідно, не буде виконувати команди з нього, навіть, якщо вони б там були.

Третій рядок є безпосередньо командою, котра виводить результати на пристрій виводу.

Збережемо введені дані (ctrl+O) та вийдемо з редактора (ctrl+X).

Запустимо підготовлений скрипт на виконання. Для цього наберемо у терміналі наступну команду:

```
$ bash one_lesson.sh
```



Як бачимо, результатом виконання скрипта буде виведення на термінал слів «One lesson!».

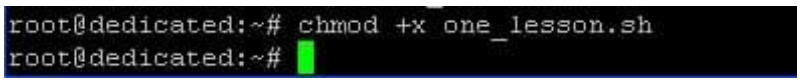
У випадку, якщо файл скрипта знаходиться в іншій директорії, необхідно безпосередньо в команді вказати до нього шлях. Шлях можна визначити за допомогою внутрішньої команди для виводу поточного каталогу *pwd*. Приклад:

```
$ bash ttt/bash/newscript.sh
```

Це був перший спосіб запуску скриптів. Інший спосіб полягає у присвоєнні файлу сценарію атрибуту «виконавчий» та використання символів «./» для його запуску.

Щоб змінити статус нашого файлу, введемо у терміналі:

```
$ chmod +x one_lesson.sh
```



Тепер запустимо скрипт:

```
$ ./one_lesson.sh
```

Тобто, вказаний спосіб запуску сценаріїв є працездатним.

Такі скрипти можуть бути доволі складними, утвореними із великої кількості пов'язаних між собою команд, кожна з яких має свій власний синтаксис. Це означає, що починати треба з бази – мовних конструкцій *bash*-процесора.

2.3. Мовні конструкції та внутрішні змінні Bash

Можливості *bash* дозволяють контролювати стан програмного середовища, створювати сценарії та управляти їх виконанням. Усе це стає можливим, зокрема, завдяки підтримці внутрішніх та зовнішніх змінних, наявності спеціальних конструкцій та операторів. Кожен з елементів вказаного арсеналу має свій власний синтаксис та правила використання. Розглянемо їх та перевіримо в дії.

2.3.1 Змінні

У *bash* відсутні типи даних. Всі змінні, котрі використовуються можна розділити на внутрішні або влаштовані у оболонку, та зовнішні, котрі створюються користувачем для різних задач. І, хоча вони мають різне походження та призначення, принцип їх використання однаковий. Ознакою змінної є наявність символу «$» попереду її назви.

Будь-яка змінна може зберігати числові значення, окремі символи або рядки символів. Розглянемо правила та приклади використання обох видів змінних.

2.3.2. Внутрішні змінні

Внутрішні змінні зберігають значення системних параметрів, необхідних для налаштування потрібної конфігурації середовища та отримання системної інформації. Приклади найбільш вживаних змінних:

- *$BASH_VERSION* – зберігає версію оболонки;
- *$HOME* – домашній каталог користувача;
- *$GROUPS* – назви груп, до котрих належить поточний користувач;
- *$HOSTTYPE* – ідентифікація апаратної архітектури машини;
- *$PS1* – запрошення командного рядка;
- *$PATH* – шлях пошуку виконавчих файлів, котрий використовується оболонкою;
- *$PPID* – ідентифікатор батьківського процесу;
- *$PWD* – поточний каталог;
- *$SECONDS* – час роботи сценарію.

Розглянемо використання декотрих з них. Для прикладу, однією з найважливіших є змінна *$PATH*. Її використання підвищує гнучкість та безпечність Linux-систем. Вона слугує для вказівки *bash* де шукати виконавчі файли. Зазвичай, такі файли знаходяться у каталогах */bin*, котрих може бути кілька в різних місцях. Це дозволяє не вказувати повний шлях до програми у командному рядку, а лише її назву. Значення змінної завантажується до пам'яті при завантаженні файлів конфігурації оболонки. Таких файлів є кілька, але який з них «прочитається» оболонкою залежить від того, яким чином була проведена реєстрація в системі. Ось ці файли:

/etc/bash.bashrc – глобальний файл сценарію конфігурації оболонки;

~ /.bashrc – персональний файл конфігурації;

~ /.bash_profile – файл ініціалізації користувача;

/etc/profile – глобальний файл ініціалізації.

Переглянемо існуюче на даний час значення вказаної змінної. Для цього введемо у терміналі:

```
$ echo $PATH
```

```
root@dedicated:~# echo $PATH
/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin:/usr/games:/usr/local/games:/snap/bin
root@dedicated:~#
```

Ми бачимо директорії, налаштовані в *$PATH* по замовченню. Всі вони перераховані через двокрапку. Існує ще один спосіб переглянути значення вказаної змінної:

$ printenv PATH

```
root@dedicated:~# printenv PATH
/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin:/usr/games:/usr/local/games:/snap/bin
root@dedicated:~#
```

Як бачимо, результат однаковий. Ще один спосіб дізнатися шлях до виконавчого файлу програми, це використання спеціальної команди which разом із потрібною програмою. Введемо у терміналі:

$ which whoami

```
root@dedicated:~# which whoami
/usr/bin/whoami
root@dedicated:~#
```

Ми отримали шлях до місцезнаходження виконавчих файлів програми *whoami*

Можна тимчасово додати каталог у *$PATH*. Це значення буде діяти лише на час роботи поточного сеансу терміналу. При закритті терміналу каталог автоматично видаляється. Для додавання нового каталогу використовуємо команду *export*:

$ export PATH=«/NewDirectory:$PATH»

```
root@dedicated:~# export PATH="/NewDirectory:$PATH"
root@dedicated:~#
```

Перевіримо поточне значення змінної:

$ echo $PATH

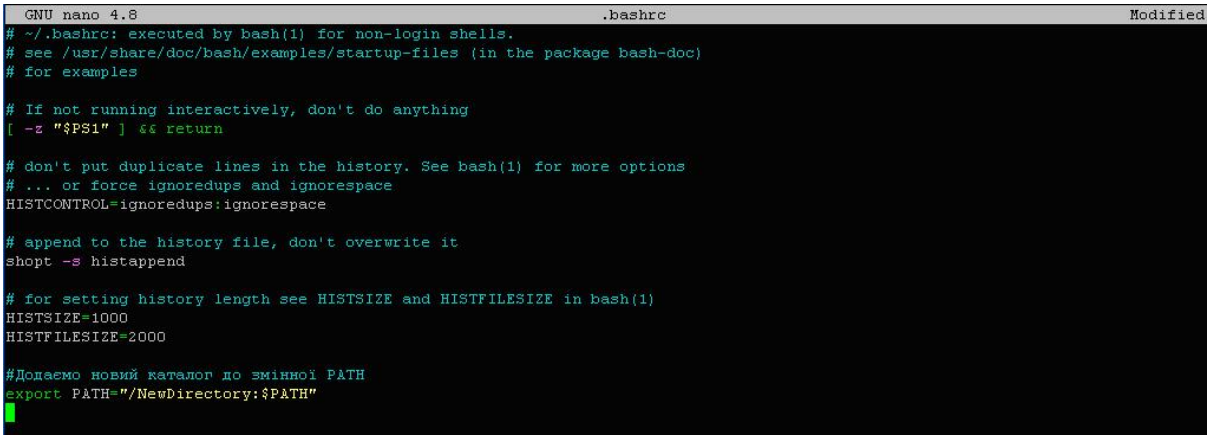
```
root@dedicated:~# echo $PATH
/NewDirectory:/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin:/usr/games:/usr/local/games:/snap/bin
root@dedicated:~#
```

Отже, новий каталог додано.

Можна додати новий каталог, відредагувавши вказані вище файли конфігурації за допомогою одного з редакторів тексту, наприклад, *nano*. Для того, щоб додати новий каталог лише для поточного користувача системи, необхідно відредагувати персональний файл конфігурації. Введемо у терміналі:

$ nano .bashrc

```
root@dedicated:~# nano .bashrc
```



Збережемо внесені зміни та вийдемо з редактора (ctrl+O, ctrl+X). Для того, щоб зміни негайно почали діяти, необхідно перезавантажити вказаний файл:

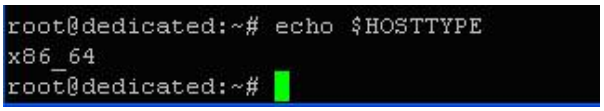
```
$ source .bashrc
```

Після цього внесені зміни вступили в силу.

Якщо ж ми хочемо додати каталог для всіх користувачів системи, то тоді необхідно відредагувати глобальний файл ініціалізації */etc/profile*.

Так само ми можемо переглядати й змінювати значення інших внутрішніх змінних оболонки. Для прикладу, розглянемо тип апаратної архітектури нашого сервера:

```
$ echo $HOSTTYPE
```



Результат – x86_64. Взагалі ж, якщо нам потрібно переглянути значення всіх змінних програмного середовища, то можна скористатися командою *printenv*:

```
$ printenv
```

Як бачимо, значення одразу кількох внутрішніх змінних виведені на термінал.

2.3.3. Зовнішні змінні

Зовнішні змінні створюються користувачем в основному для їх використання у сценаріях. Для їх коректного використання треба притримуватись встановлених домовленостей, стосовно імен змінних:

- Імена повинні починатися з букви або символу підкреслення;
- У складі імені можуть бути тільки букви, цифри та символи підкреслення;
- Враховується регістр символів;
- Використання пробілів та спец символів заборонено;
- Не можна використовувати зарезервовані системою слова, котрі використовуються для назв мовних конструкцій та системних змінних.

Для прикладу, наведемо коректні назви змінних:

*Gora_*
_fazan
mmm

Некоректні назви:

Ind-us (використовується заборонений символ «-»)
12345r (назва починається з цифри)
gh hhd (присутній пробіл у середині назви)

Присвоїти значення, тобто, по суті, створити нову змінну можна двома шляхами – пряме присвоєння та присвоєння результатів роботи команди або програми шляхом підстановки.

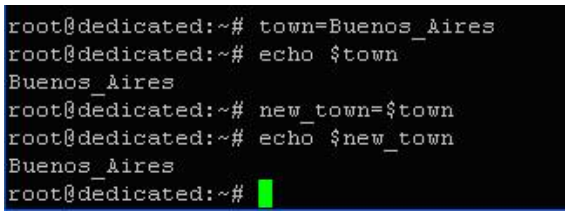
Приклад прямого присвоєння:

```
town=Buenos_Aires
```

Приклад підстановки:

```
new_town=$town
```

Тут значення змінної *$town* передалося новій змінній *$new_town*. Продемонструємо це в терміналі:



Звертаємо увагу на те, що пробіли між змінними та їх значеннями недопустимі. Наприклад, спробуємо присвоїти значення, тобто, створити змінну з використанням пробілів у команді:

town = Buenos_Aires

Як бачимо, інтерпретатор *bash* сприймає *»town»* як команду, а оскільки такої команди не існує, то він виводить повідомлення, що така команда не знайдена. Отже, це треба враховувати у подальшому.

2.4. Конструкції для введення, виведення та перенаправлення даних

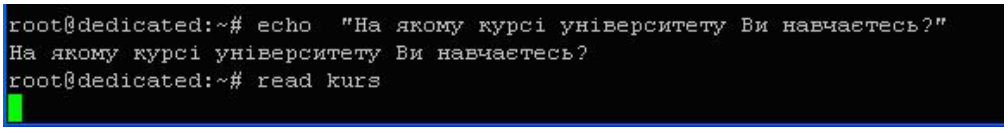
Розглянемо спочатку конструкції для вводу. Ввід даних може бути здійснений з кількох джерел:

- З клавіатури користувачем (користувацький);
- З файлу;
- За рахунок використання аргументів командного рядка.

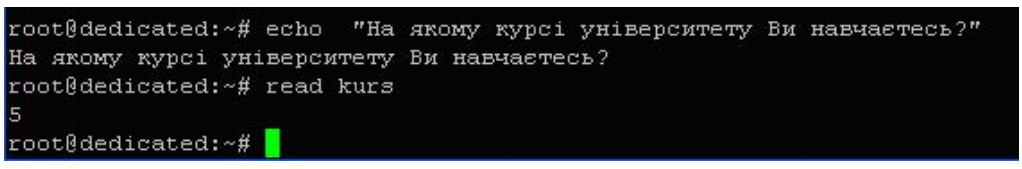
Користувацький ввід даних є можливим завдяки наявності оператора read. Продемонструємо це. Наберемо у терміналі:

$ echo «На якому курсі університету Ви навчаєтесь?»

$ read kurs

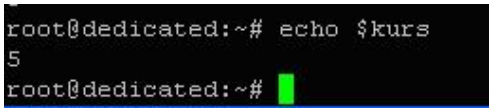


Результат введення двох команд можна побачити на скриншоті. Система очікує введення даних від користувача. Після їх введення та підтвердження вводу клавішею *Enter* результат буде наступним:



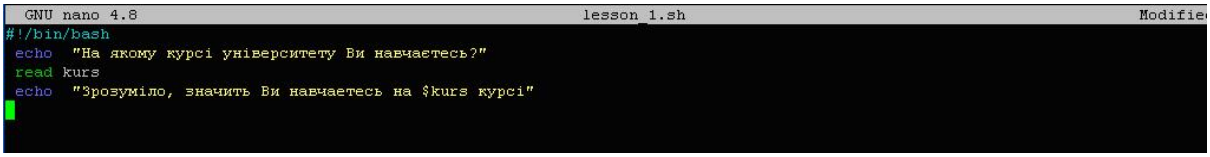
Тобто, команда автоматично вивела результат. Перевіримо, чи збереглося значення змінної *$kurs* у пам'яті. Для цього введемо команду виводу:

```
$ echo $kurs
```



Так, значення збереглося. Це означає, що такі змінні можна використовувати в сценаріях. Сформуємо сценарій із використаних нами команд. Для цього викличемо редактор та введемо в нього наступні рядки:

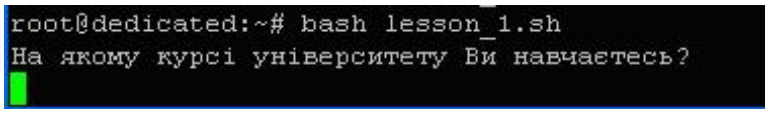
```
#!/bin/bash
echo  «На якому курсі університету Ви навчаєтесь?»
read kurs
echo  «Зрозуміло, значить Ви навчаєтесь на $kurs курсі»
$ nano lesson_1.sh
```



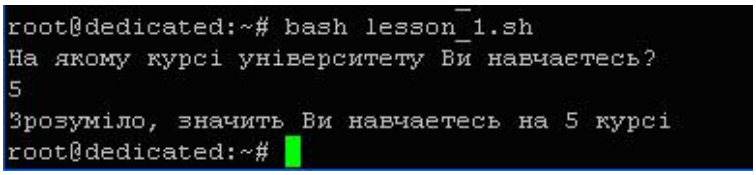
Збережемо файл та запустимо його на виконання:

```
$ bash lesson_1.sh
```

Після запуску сценарію з'являється прохання ввести дані та система знаходиться у режимі очікування вводу:



Кінцевий результат роботи сценарію показаний нижче.

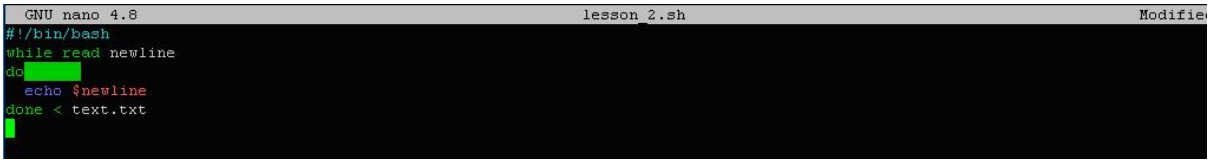


Можна переконатися, що змінна *$kurs*, яка була вставлена в останный рядок, успішно спрацювала.

Зчитування даних з файлу застосовується при необхідності вивести результати на термінал або ж у інший файл чи пристрій. Для цієї мети існує спеціальна конструкція «<«. Застосуємо цю конструкцію у нашому новому сценарію,

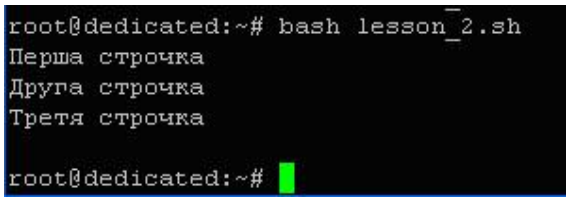
який призначений для виводу на термінал кожного рядку файлу із ім'ям *text.txt*. Викличемо редактор та вставимо в нього код, наведений нижче:

```
#!/bin/bash
while read newline
do
  echo $newline
done < text.txt
$ nano lesson_2.sh
```



Збережемо внесені зміни та закриємо файл сценарію. Після цього запустимо його на виконання

```
$ bash lesson_2.sh
```



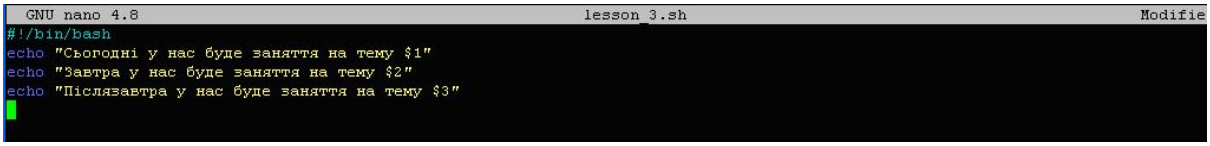
У результаті, кожен рядок текстового файлу виведений на термінал, як і планувалося.

Аргументи командного рядка є ще одним вдалим способом роботи зі змінними. Сутність його полягає у послідовній нумерації переданих скрипту або функції параметрів або змінних. Завдяки цьому, такі змінні можна легко ідентифікувати у межах коду. Це виглядає так:

```
$1, $2, $3 .... $N
```

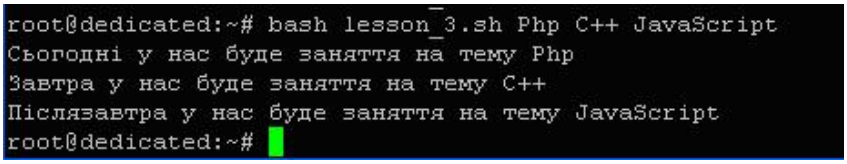
Для прикладу, створимо новий файл сценарію із ім'ям *lesson_3* та помістимо в нього наступний код:

```
#!/bin/bash
echo «Сьогодні у нас буде заняття на тему $1»
echo «Завтра у нас буде заняття на тему $2»
echo «Післязавтра у нас буде заняття на тему $3»
$ nano lesson_3.sh
```



Збережемо внесені зміни та закриємо файл. Після цього запустимо створений скрипт на виконання з передачею трьох параметрів, як показано нижче:

$ bash lesson_3.sh Php C++ JavaScript



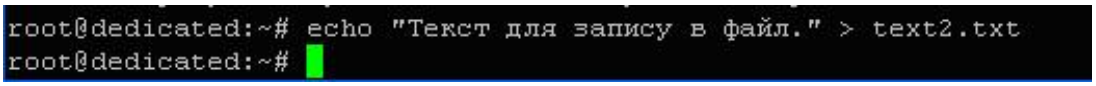
Як видно зі скриншоту, всі параметри були успішно передані та ідентифіковані, тобто, метод аргументів працює.

2.5. Виведення даних

Виведення на термінал можна забезпечити за допомогою команди *echo*, роботу з котрою ми вже не раз демонстрували.

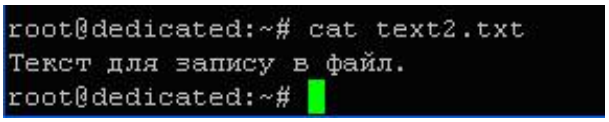
Для виводу або запису інформації у файл у *bash* застосовується конструкція «>«, котра є дзеркальним відображенням конструкції для вводу. Для прикладу, запишемо текстову інформацію у файл *text2.txt*. Введемо у терміналі:

$ echo «Текст для запису в файл.» > text2.txt



Тепер перевіримо вміст файлу за допомогою команди cat (виведення вмісту файлу):

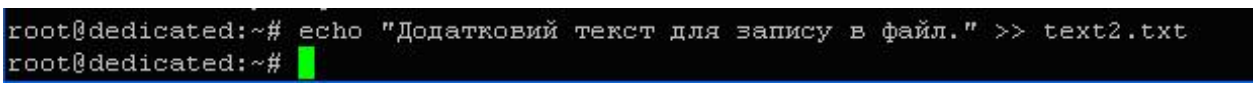
$ cat text2.txt



Можна переконатися, що запис інформації відбувся успішно.

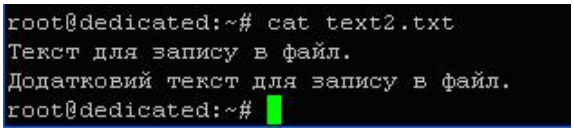
Інформацію можна також додати у кінець будь-якого файлу. Для цього існує конструкція «>>«. Скористаємося нею та додамо додаткові дані до нашого файлу. Введемо:

$ echo «Додатковий текст для запису в файл.» >> text2.txt



Знову перевіримо вміст файлу:

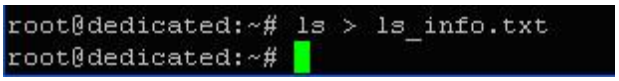
$ cat text2.txt



Як бачимо, текст успішно додався у кінець існуючих записів.

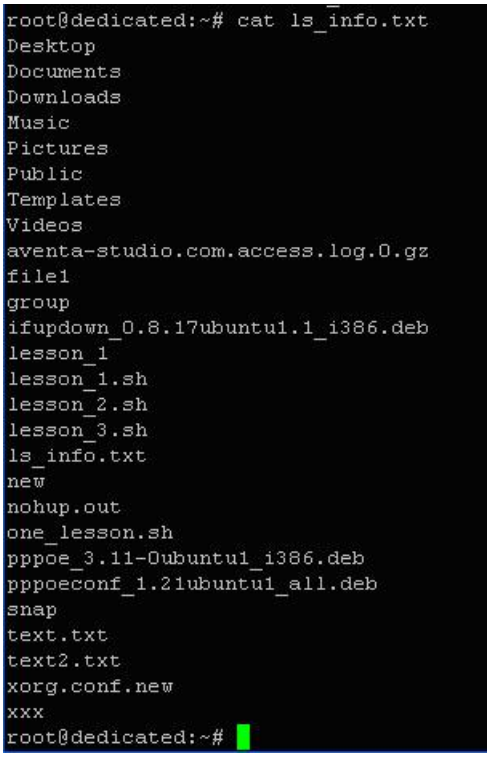
Ще один спосіб виводу інформації – це її пере направлення. Для прикладу, перенаправимо результати виконання команди у визначений файл. Введемо у терміналі:

```
$ ls > ls_info.txt
```



Зробимо перевірку:

```
$ cat ls_info.txt
```



Можна переконатися в успішності виконаної операції.

2.6. Організація обробки даних в сценаріях Bash

Оболонка *bash* не була б такою ефективною і популярною серед фахівців, якби не мала внутрішніх програмних засобів для створення доволі складних сценаріїв, котрі відповідають всім вимогам структурного програмування. Одним із таких засобів є оператори розгалуження. У *bash* вони мають деякі особливості. Поговоримо про це та перевіримо їх роботу на практиці.

2.7. Огляд управляючих конструкцій Bash для обробки даних

Bash є командним процесором із широкими можливостями для імперативного програмування, коли програми будуються у формі послідовного списку наказів або команд для інтерпретатора. Так само, як і для більшості імперативних мов програмування (C, C++, Java) у *bash* підтримується структурний підхід для організації програм, котрий полягає у представленні програми у вигляді ієрархічної структури або блоків. У відповідності до вказаного підходу, будь-яка програма, котра не містить операторів безумовного переходу будується із використанням трьох управляючих форм або конструкцій:

- Послідовність;
- Розгалуження;
- Цикли.
-

Послідовність є найпростішою формою організації обчислень, котра не вимагає наявності додаткових конструкцій чи структур і тому не викликає складнощів у використанні.

Найбільш складними формами є **розгалуження** та **цикли**. Вони дозволяють виконувати більш складну обробку числових та текстових даних, що значно підвищує рівень «інтелекту» сценаріїв. Розглянемо більш детально існуючі форми реалізації розгалуження у *bash*.

Оператори розгалуження

Оператори розгалуження застосовуються у випадках, коли певний набір команд повинен виконуватися чи не виконуватися в залежності від деякої умови або умов. Для всіх імперативних мов програмування, так само, як і для мовних засобів *bash*, існують наступні форми реалізації розгалуження:

- Умовний оператор;
- Оператор багатозначного вибору.
-

Умовний оператор

Умовний оператор є більш простою формою керування даними, котра забезпечує виконання певних команд при умові істинності значення логічного виразу, котрий, зазвичай, і є основною умовою для їх виконання.

У більшості мов програмування, як і у *bash*, для цієї мети використовується конструкція *if*, що в перекладі означає «або». Вона може мати одно або декілька розгалужень, котрі позначаються оператором *elif*. Загальна синтаксична форма оператора *if* має наступний вигляд:

```
if [[умова 1]];
then
    Команда 1
    .......
    Команда N
  elif [[умова 2 ]];
    then
       Команда 1
       .......
       Команда N
  ...................
      elif [[умова M ]];
```

```
        then
            Команда 1
            .......
            Команда N
else
    Команда 1
    .......
    Команда N
fi
```

У даному випадку здійснюється, так зване, каскадування умовних операторів за допомогою операторів *elif*. Загальна кількість розгалужень – M. Робота усього блоку відбувається наступним чином. У випадку виконання умови № 1 (повертається значення *True*) виконуються команди, вказані після найближчого оператору *then*. Після цього відбувається перехід до кінцевого оператору *fi* і робота програми завершується. Якщо ж умова № 1 не виконується (повертається значення *False*), тоді по черзі обробляються всі умовні оператори каскаду, доки якась із умов не поверне значення *True*.

У випадку, якщо ні одна із *M* умов не була виконана, йде перехід до виконання команд, розташованих за оператором *else*. Після того, як усі вони будуть оброблені, відбувається перехід до кінцевого оператору *fi* і робота програми завершується.

Розглянутий варіант є найскладнішим. Частіше ж зустрічаються скорочені версії конструкції *if*, де може бути лише одне розгалуження, або, наприклад, може взагалі не бути альтернативних команд, котрі визначаються конструкцією *else*.

У *bash* підтримується певний набір операторів та параметрів, котрі можуть бути використані у формуванні логічного виразу або умов. Наведемо кілька з них.

-a file - Повертається значення *True*, якщо файл існує;
-c file - True, якщо файл є спеціальним символом;
-d file - True, якщо файл є директорією;
-s file - True, якщо файл існує і не є порожнім;
file1 -ot file2 - True, якщо файл *file1* більш давній, ніж *file2*;
-z STRING - True, якщо текстовий рядок порожній;
-v VAR - True, якщо встановлена змінна *VAR*;
! EXPR - True, якщо значенням виразу є *False*;
EXPR1 -a EXPR2 - True, якщо обидва вирази вірні;
EXPR1 -o EXPR2 - True, якщо якесь із виразів є вірним;
arg1 AO arg2 - Повертається значення в залежності від результату арифметичної операції між операндами.
Де *file* – ім'я файлу, *EXPR* – вираз, *AO* – арифметичний оператор (-eq, -ne, -lt, -le, -gt, or –ge).

Взагалі ж, завдяки наявній у *bash* потужної довідкової служби, можна ознайомитися із всіма доступними параметрами за допомогою команди *help* з відповідним параметром. Введемо у терміналі:

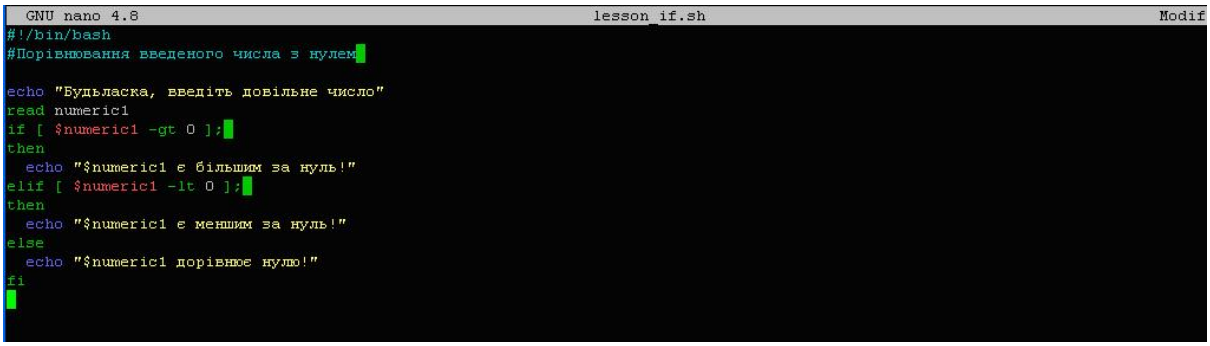
```
$ help test
```

Для прикладу, створимо сценарій із ім'ям *lesson_if*, котрий буде порівнювати введене користувачем число із нулем. Введемо у терміналі:

```
$ nano lesson_if.sh
```

У створеному файлі введемо наступний код:

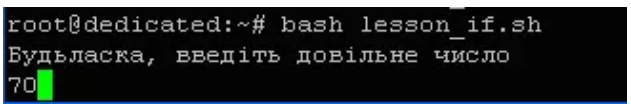
```
#!/bin/bash
#Порівняння введеного користувачем числа з нулем
echo «Будь ласка, введіть довільне число»
read numeric1
if [ $numeric1 -gt 0 ];
then
  echo «$numeric1 є більшим за нуль!»
elif [ $numeric1 -lt 0 ];
then
  echo «$numeric1 є меншим за нуль!»
else
  echo «$numeric1 дорівнює нулю!»
fi
```

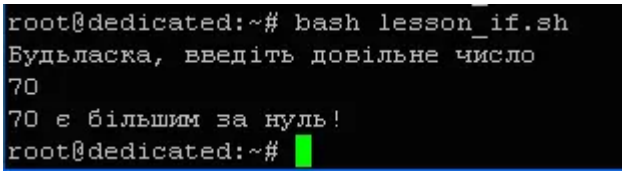


Збережемо внесені зміни (ctrl+O, Enter) та вийдемо з редактору (ctrl+X).

Тепер запустимо на виконання створений сценарій. Для цього введемо у терміналі:

```
$ bash lesson_if.sh
```





Як бачимо, сценарій працює так, як треба.

Оператор багатозначного вибору

Оператори багатозначного вибору є ще однією формою реалізації розгалуження. У кожній з імперативних мов програмування використовуються свої

конструкції для цієї мети. Наприклад, у мовах програмування C та JavaScript це оператор *switch*, а у *bash* – оператор case. Такі конструкції, зазвичай, використовуються для заміни каскадів або групи вкладених умовних операторів *if*, що дозволяє спростити код та зробити його більш читабельним та легким у використанні.

Наведемо загальну синтаксичну форму оператора *case*:

```
case Вираз in
  Шаблон_1)
     Оператори
    ;;
  Шаблон_2)
     Оператори
    ;;
  ........
    Шаблон_N)
     Оператори
    ;;
  *)
     Оператори
    ;;
esac
```

Сутність роботи оператора *case* полягає в послідовному пошуку відповідності шаблонів зазначеному виразу. У випадку, якщо така відповідність буде знайдена, починають виконуватися оператори, котрі йдуть за відповідним шаблоном. Після цього йде перехід до кінцевої конструкції *esac* і програма завершує свою роботу. Якщо ж жодної відповідності не було знайдено, виконуються оператори, вказані після конструкції «*)» і вже після цього робота програми завершується.

Вказаний алгоритм роботи відрізняється від алгоритму аналогічного оператора *switch*, котрий використовується у мові програмування C. Там після знайдення першої відповідності та виконання команд, робота оператора не завершується, а продовжується пошук інших відповідностей, і так буде доти, доки не будуть оброблені всі шаблони. І це є суттєва відмінність, котру треба враховувати у подальшому.

Сформулюємо основні правила використання оператора *case*:

- Конструкція повинна починатися з ключового слова *»case»*, за котрим йде вираз та ключове слово *»in»*;
- Можна використовувати кілька шаблонів, розділених символом «|», при цьому оператор «)» завершує їх список;
- Кожен блок «шаблон – оператори» повинен закінчуватись знаками «;;»;
- Шаблон може містити спеціальні символи;
- Виконуються лише команди, котрі відповідають першому шаблону;
- Конструкція «*)» використовується у якості кінцевого шаблону, котрий встановлено за замовчуванням;
- Якщо жоден із шаблонів не є відповідним, статус повернення дорівнює нулю. У іншому випадку статус повернення є статусом виходу виконаних команд.

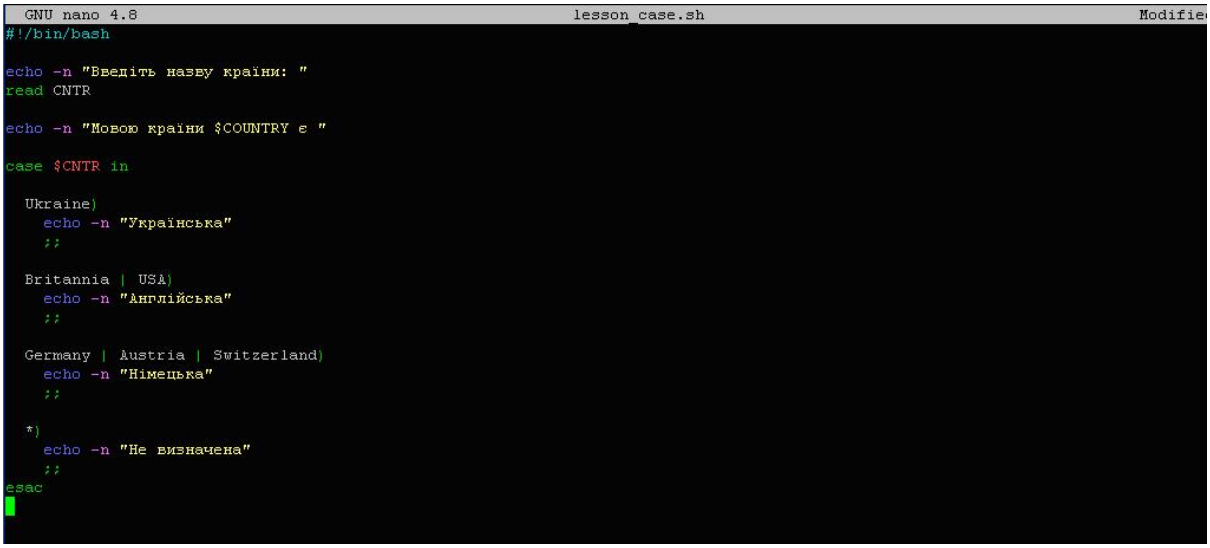
Створимо сценарій, у котрому буде продемонстровано роботу оператора *case*. Сутність сценарію полягає у автоматичному визначенні мови країни, назва котрої вводиться користувачем. Результат, тобто, визначена мова, виводиться на термінал.

Введемо у терміналі команду створення файлу сценарію:

```
$ nano lesson_case.sh
```

У вікні редактора введемо наступний код:

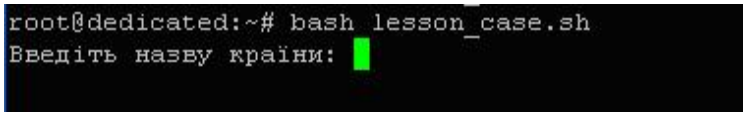
```
#!/bin/bash
echo -n «Введіть назву країни: «
read CNTR
echo -n «Мовою країни $COUNTRY є «
case $CNTR in
  Ukraine)
    echo -n «Українська»
    ;;
  Britannia | USA)
    echo -n «Англійська»
    ;;
  Germany | Austria | Switzerland)
    echo -n «Німецька»
    ;;
  *)
    echo -n «Не визначена»
    ;;
esac
```



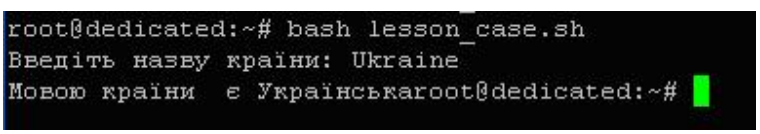
Збережемо файл (ctrl+O, Enter) та вийдемо з редактору (ctrl+X).

Після цього запустимо на виконання наш сценарій. Введемо у терміналі:

```
$ bash lesson_case.sh
```



Після введення значення «Ukraine» та підтвердження вводу отримаємо наступний результат:



Тобто, мова вказаної у запиті сценарію країни визначена вірно.

2.8. Використання циклів та виразів у сценаріях Bash

Цикли є одним із основних типів управляючих мовних конструкцій *bash*, котрі дозволяють в повній мірі реалізувати можливості структурного підходу в програмах сценаріїв. Вони призначені для організації багаторазового виконання набору команд, котрі утворюють тіло циклу. Процес виконання циклу включає:

- Початкову ініціалізацію змінних;
- Перевірку умови виходу або завершення;
- Виконання тіла циклу;
- Оновлення змінної на кожній ітерації.

У *bash*, як і у більшості імперативних мов програмування, підтримується кілька різновидів циклів, що дозволяє розширити простір для їх використання. Це такі конструкції:

- For;
- While;
- Until.

Нижче розглянемо більш детально синтаксис та особливості роботи кожної із вказаних конструкцій.

Цикл For

Цей оператор є найбільш вживаним через свою гнучкость у налаштовуванні синтаксису в залежності від задач. Його загальний синтаксис має наступний вигляд:

```
for VARIABL in [список елементів]
do
   [список команд]
done
```

Роз'яснимо призначення кожного з операторів та параметрів.

VARIABL – слово, котре може складатися із латинських букв та цифр і є заповнювачем для змінної. При цьому, *$VARIABL* – індивідуальне значення змінної.

in – службовий ключ, котрий відокремлює змінну від елементів списку.

[список елементів] – значення, котрі послідовно або за певним алгоритмом приймає змінна *$VARIABL*. Алгоритм вибору може залежати від версії *bash*. Для версій <3.0 список має такий вигляд: 1 2 3 ..... N. Починаючи з версії 3.0, той самий список реалізовується за допомогою символів «..», наприклад: {1..N}. З версії 4.0 стало можливим використання конструкції типу {START..END..INCREMENT}, де START – початкове значення змінної *$VARIABL*, INCREMENT – крок ітерації, END – кінцеве значення змінної. Попередній список буде виглядати так: {1..N..1}. Вочевидь, це більш компактна та зручна форма запису елементів порівняно з іншими.

do – службове ключове слово, котре запускає ітерації обробки тіла циклу. У найпростішому випадку, за відсутності операторів переривання та переходу, кількість ітерацій буде дорівнювати кількості елементів зі списку.

[список команд] – команди, котрі утворюють тіло циклу.

done – оператор зупинки циклу.

Існує також синтаксис оператора *for* для організації роботи двох циклів та більше. Наприклад, для двох циклів синтаксис оператора може мати наступний вигляд:

```
for VARIABL in [список елементів]
do
   command_1 on $VARIABL
   .......
   command_M
done
```

Для прикладу, розглянемо різні варіанти використання вказаного оператора. Однак, для початку з'ясуємо версію оболонки *bash*, котра встановлена на нашому *Ubuntu*. Це дозволить зорієнтуватися, що до її можливостей. Для цього введемо у терміналі:

```
echo $BASH_VERSION
```

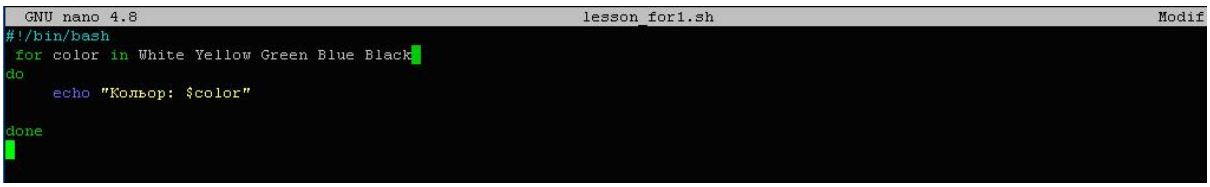
Отже, у системі встановлена свіжа версія *bash 5.0.17*.

Введемо команду створення файлу сценарію:

```
$ nano lesson_for1.sh
```

У вікні редактора введемо наступний код:

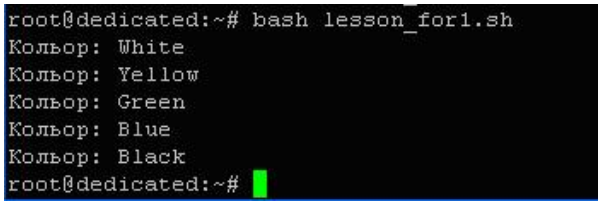
```
#!/bin/bash
 for color in White Yellow Green Blue Black
do
    echo «Кольор: $color»
done
```



У сценарії використовується найпростіший варіант списку елементів. Сутність його роботи полягає у виводі на термінал списку кольорів, впорядкованих по світлості тону – спочатку світліші, а потім темніші. Збережемо внесені зміни (ctrl+O, Enter) та вийдемо з редактору (ctrl+X).

Тепер запустимо створений сценарій. Для цього введемо у терміналі:

```
$ bash lesson_for1.sh
```



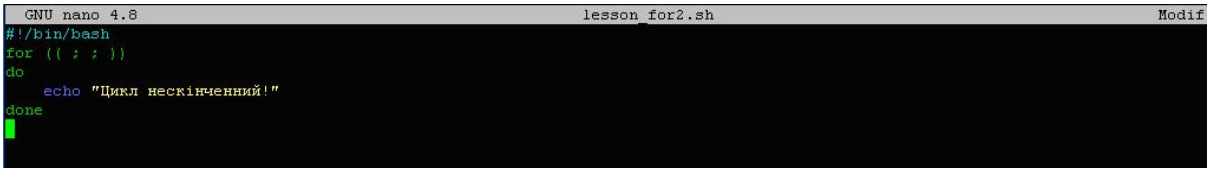
В результаті, ми бачимо вивід кольорів у тому порядку, який був у списку елементів оператора *for*.

Використаємо оператор для створення нескінченного циклу. Для цього створимо новий файл сценарію:

```
$ nano lesson_for2.sh
```

Додамо у редакторі наступний код:

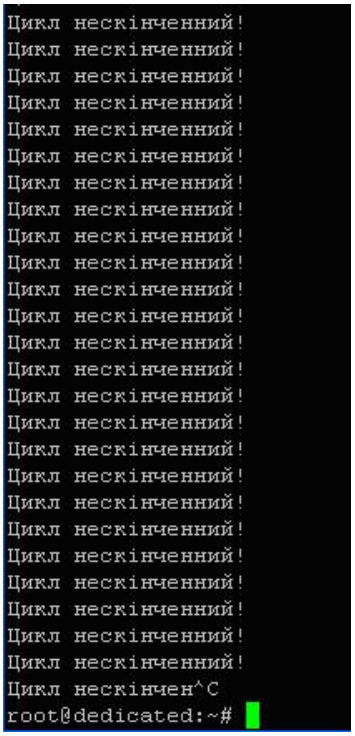
```
#!/bin/bash
for (( ; ; ))
do
   echo «Цикл нескінченний!»
done
```



Виходом сценарію повинна бути фраза «Цикл нескінченний!», котра постійно генерується. Перервати процес можна за допомогою комбінації клавіш «ctrl+C». Збережемо зміни та вийдемо з редактора.

Запустимо сценарій. Для цього введемо:

```
$ bash lesson_for2.sh
```



Можна переконатися, що цей цикл дійсно нескінченний.

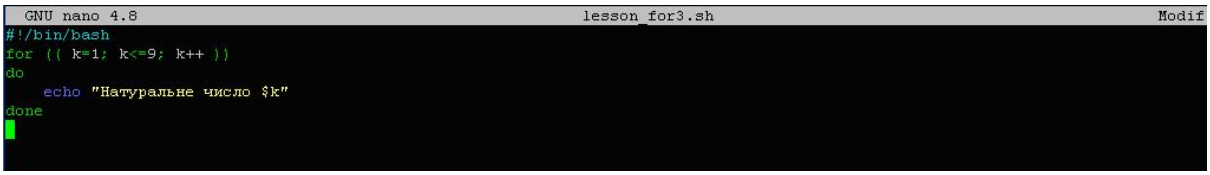
Тепер створимо сценарій з використанням більш складного варіанту запису списку елементів – типу {START..END..INCREMENT}, призначення котрого ми вже розглядали.

Запустимо редактор:

```
$ nano lesson_for3.sh
```

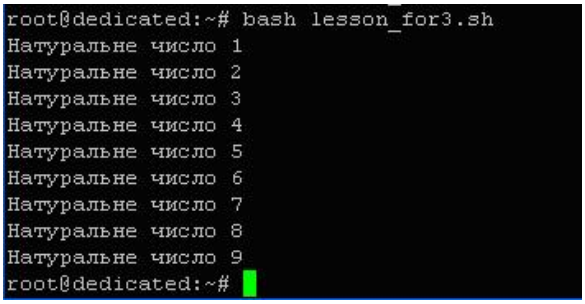
Введемо наступний код:

```
#!/bin/bash
for (( k=1; k<=9; k++ ))
do
   echo «Натуральне число $k»
done
```



Код буде виводити ряд натуральних чисел доти, доки не виконається умова завершення циклу, зазначена у логічному виразу (END-частині), тобто, до числа 9 включно. Закриємо редактор та запустимо сценарій:

```
$ bash lesson_for3.sh
```



Як бачимо, код працює.

Тепер продемонструємо роботу оператора *for* разом з конструкцією *continue*, за допомогою якої можуть ігноруватися певні ітерації циклу в залежності від умов. Для цього введемо команду виклику редактора:

```
$ nano lesson_for4.sh
```

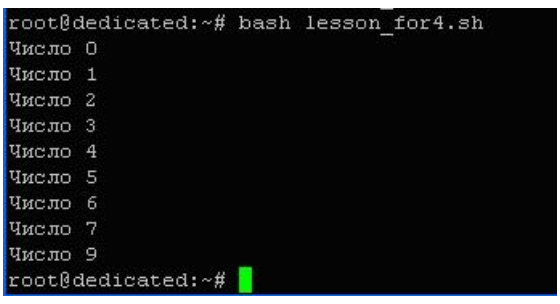
Введемо створений нами код:

```
#!/bin/bash
for j in {0..9}
do
  if [[ «$j» == '8' ]]
  then
    continue
  fi
  echo «Число $j»
done
```

У зазначеному коді для формування умови пропуску ітерації № 8 використовується оператор розгалуження *if*. Збережемо зміни та вийдемо з редактора.

Запустимо сценарій:

```
$ bash lesson_for4.sh
```



Можна переконатися, що число № 8 відсутнє у списку, що і передбачалось.

Цикл While

While є ще одним різновидом циклів, котрі використовуються у *bash*. Сутність його роботи полягає в багаторазовому виконанні тіла циклу до тих пір, поки зазначена умова буде істинною. Як тільки логічним виразом буде повернене значення *False*, робота циклу припиняється. Загальний синтаксис оператору має наступний вигляд:

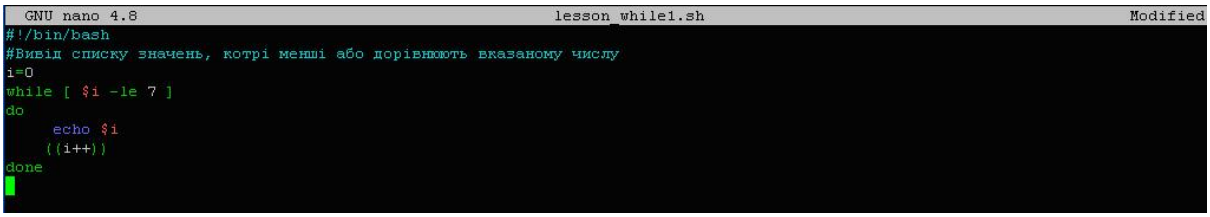
```
while [умова]

do
   [список команд]
done
```

Розглянемо найпростіший приклад його використання сумісно з оператором прирощення «++». Для цього створимо відповідний файл сценарію:

```
$ nano lesson_while1.sh
```

У вікні, що відкрилося, додамо наш код:

```
#!/bin/bash
#Вивід списку значень, котрі менші або дорівнюють вказаному числу
i=0
while [ $i -le 7 ]
do
    echo $i
   ((i++))
done
```



Робота коду полягає у збільшенні на одиницю значення змінної *$i* при кожній ітерації. Кількість ітерацій визначається вказаною умовою. У даному випадку, цикл буде виконуватися доти, доки значення змінної буде менше або дорівнюватиме числу 7. Збережемо файл та запустимо його на виконання:

```
$ bash lesson_while1.sh
```

Отже, отримано очікуваний результат.

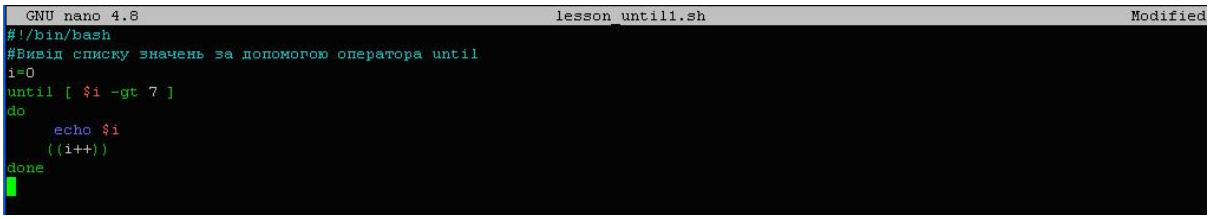
Цикл Until

Until є протилежністю *While*, оскільки тут цикл буде повторюватись доти, доки зазначена умова не стане істинною, тобто, поки логічний вираз не поверне значення *True*. Для прикладу, реалізуємо ту ж саму задачу, що і в попередньому випадку, але з використанням циклу *until*. Введемо у терміналі:

```
$ nano lesson_until1.sh
```

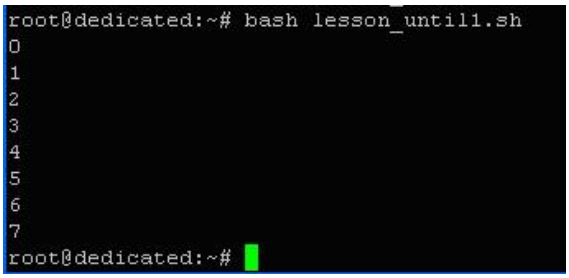
У вікні програми введемо:

```
#!/bin/bash
#Вивід списку значень за допомогою оператора until
i=0
until [ $i -gt 7 ]
do
    echo $i
   ((i++))
done
```



Збережемо зміни та вийдемо з програми. Запустимо сценарій:

```
$ bash lesson_until1.sh
```



Як бачимо, результат той самий, що і в попередньому випадку.

2.9. Вирази у Bash

Будь-яка імперативна мова програмування, так само як і *bash*, забезпечує підтримку кількох типів виразів. Це дозволяє формувати повноцінні умови виконання для управляючих операторів та виконувати будь-які обчислення. У *bash* підтримуються наступні основні типи виразів:

- Алгебраїчні;
- Булеві та логічні;
- Регулярні.

Розглянемо вказані типи виразів та продемонструємо роботу з ними.

Алгебраїчні вирази

Алгебраїчні вирази можуть складатися з кількох чисел та змінних, пов'язаних між собою за допомогою математичних операцій. *Bash*підтримує наступні базові математичні операції:

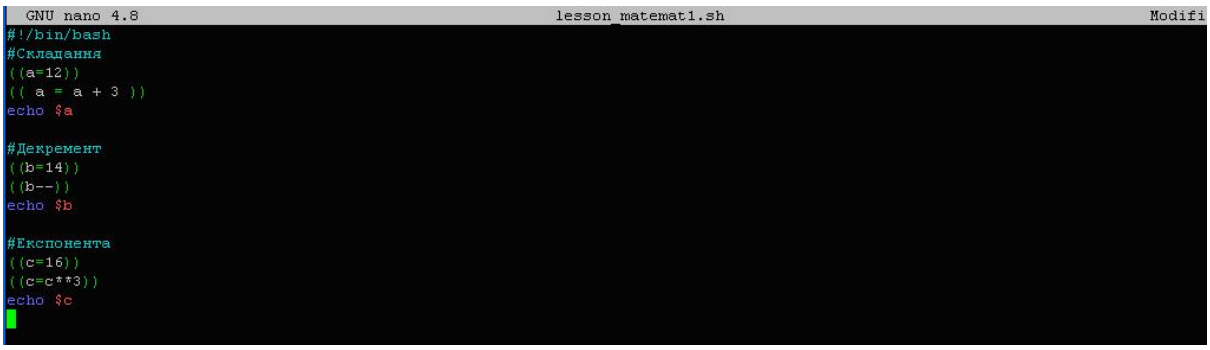
- +, - - Складання, віднімання;
- ++, -- - Інкремент, декремент;
- * - Множення;
- / - Ділення;
- % - Залишок;
- ** - Експонента.

У сценаріях вказані операції можна виконувати за допомогою подвійних дужок. Продемонструємо це на практиці. Для цього створимо сценарій із реалізацією кількох найпростіших математичних операцій. Наберемо у терміналі:

```
$ nano lesson_matemat1.sh
```

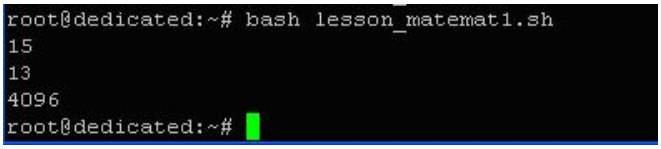
Введемо у файл код трьох блоків операцій:

```
#!/bin/bash
#Складання
((a=12))
(( a = a + 3 ))
echo $a
#Декремент
((b=14))
((b--))
echo $b
#Експонента
((c=16))
((c=c**3))
echo $c
```



Збережемо внесені зміни та закриємо файл. Тепер виконаємо сценарій:

```
$ bash lesson_matemat1.sh
```



Неважко переконатися, що всі три результати обчислень вірні. Це числа 15, 13 та 4096.

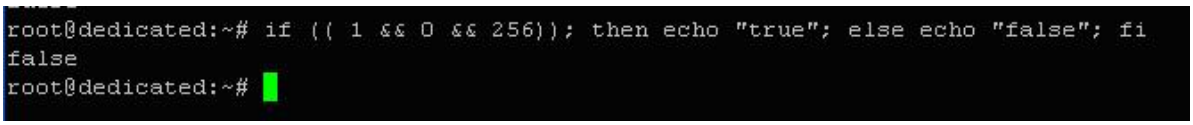
Булеві та логічні вирази

Булеві та логічні вирази у сценаріях *bash* зазвичай використовуються усередині умовних дужок, котрі мають наступний вигляд: «(( ))».

Для можливості формування логічних виразів, у *bash* застосовується кілька логічних операторів, основними з яких є оператор || (логічне АБО) та оператор && (логічне І). При перевірці умов оператор || повертає значення 0 (Успіх) якщо хоча б один із операндів має значення *True*. Оператор && повертає «успіх» лише в тому випадку, якщо обидва операнди мають значення *True*.

Наведемо приклад «роботи» вказаних операторів. Введемо у терміналі наступні команди:

```
$ if (( 1 && 0 && 256)); then echo «true»; else echo «false»; fi
```



Можна переконатися, що результатом «роботи» умовного оператора *if* є *False*, що пояснюється тим, що, як нами вже було зазначено вище, оператори && у даному випадку можуть повернути лише значення 1 *(False)*, оскільки один із операндів має значення 0 *(False)*. У зв'язку з цим, сконцентруємо увагу на тому, що код повернення будь-якої команди у випадку її успішного завершення завжди має значення 0, тобто *True*, у той час як логічне значення *True* відмінне від нуля, що і продемонстровано наведеним прикладом. Тут логічним значенням першого з операндів буде *True*, оскільки він є більшим за нуль. Те ж стосується третього операнду, котрий також має логічне значення *True*.

Окрім застосування вказаних операторів для перевірки умов у сценаріях, їх також можна використовувати у інший спосіб, зокрема, для одночасного запуску на виконання кількох команд у командному рядку або у скрипті. Синтаксис такої конструкції має наступний вигляд:

```
$ команда №1 && команда №2 .... && команда №N
```

Регулярні вирази

Регулярний вираз або *RegExp* представляє собою довільний набір символів, котрі визначають шаблон пошуку. Пошук може відбуватися у файлі або у іншому виразі. Окрім пошуку, їх можна застосовувати для перевірки умов при авторизації користувачів системи та для інших потреб.

Для роботи із *RegExp* існує зовнішня команда *grep*, основним недоліком якої є повільне виконання команд. І тому, починаючи з версії 3.0, у *bash* почали підтримуватися влаштовані регулярні вирази, такі самі, як і у багатьох мовах,

працюючих у режимі інтерпретатора, зокрема, у *perl*. Влаштовані вирази обробляються у рамках процесу *bash* і тому виконуються набагато швидше, ніж зовнішня команда *grep*. Детальну інформацію по їх синтаксису можна знайти у документації *bash* по *regex*.

Загальний синтаксис конструкції для використання влаштованих регулярних виразів має такий вигляд:

```
[[ string =~ regex ]]
```

Якщо *Regex* співпало чи увійшло до рядка *String*, тоді статусом виходу буде значення 0. У іншому випадку – значення 1. Значення внутрішнього виразу, заключеного у дужки можна отримати за допомогою внутрішньої змінної *bash* – *$BASH_REMATCH*.

Продемонструємо роботу з регулярними виразами на прикладі одного сценарію. Для цього створимо файл сценарію із ім'ям *lesson_regex*:

```
$ nano lesson_regex.sh
```

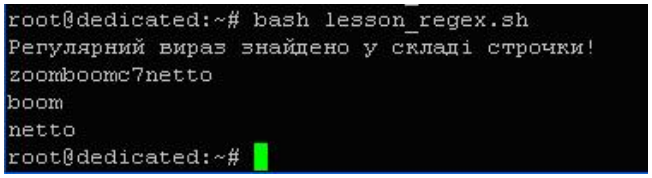
Додамо наш код:

```
REGEXP='zoom(boom)c7(.*)'
if [[ «mnczoomboomc7netto» =~ $REGEXP ]]
then
  echo «Регулярний вираз знайдено у складі рядка!»
  echo «$BASH_REMATCH»
  echo «${BASH_REMATCH[1]}»
  echo «${BASH_REMATCH[2]}»
fi
```

Вказаний код співставляє вираз *REGEXP* із рядком, котрий знаходиться у лівій частині умовного оператора *if*. За результатами співставлення виводяться підсумовуючі повідомлення.

Збережемо файл та виконаємо сценарій:

```
$ bash lesson_regex.sh
```



Як бачимо, вираз у складі рядка знайдено. Значення змінної *$BASH_REMATCH* без параметрів показує наш вираз. Ця ж змінна з параметрами показує символи, котрі

знаходяться у дужках: з параметром 1 – значення *boom*, з параметром 2 – значення *netto*.

Зовнішні команди bash для роботи з виразами та виконання складних обчислень

Будь-яка мова програмування чи командний процесор повинні мати достатній набір програмних засобів для роботи з різними типами виразів та виконання складних обчислень. Внутрішні команди, зазвичай, є компактними і їх набір обмежений. І тому, виходом із ситуації є використання, так званих, зовнішніх або додаткових команд, котрі здатні реалізувати будь-яке завдання, яке є надто складним для базового набору. І командний процесор *bash* не є винятком. Розглянемо найбільш вживані зовнішні команди *bash* та протестуємо їх можливості на практиці.

Огляд зовнішніх команд bash для роботи з виразами

Серед зовнішніх команд *bash* можна виділити ті, котрі розраховані на роботу із широким спектром виразів, і тому є універсальними. Інші є більш спеціалізованими і застосовуються лише для певних типів виразів або окремих операцій.

Найбільш вживаним представником універсальних зовнішніх засобів *bash* є команда *expr*. Вона працює практично з усіма типами доступних виразів – строковими, цілочисельними, логічними та регулярними. Її використання дозволяє значно розширити можливості оболонки по обробці різних типів даних.

До спеціалізованого засобу по роботі із регулярними виразами відноситься програма *grep*. Її застосування дозволяє розширити межі обробки регулярних виразів порівняно із внутрішнім механізмом *regex*, обробка котрого здійснюється у рамках процесу *bash*.

Для виконання складних математичних операцій існує додатковий набір спеціалізованих зовнішніх програм, таких, наприклад, як *jot, bc, factor*. Вони дозволяють виконувати специфічні завдання, наприклад, вивід числових значень у певній послідовності, як у *jot*, чи розрахувати значення числових коефіцієнтів, як у *factor*.

Кожен із наведених програмних засобів має свій власний синтаксис та правила застосування. Їх знання допоможе зорієнтуватися у командному середовищі у разі необхідності вирішення певного завдання. І тому, розглянемо більш детально роботу з ними.

Робота з програмою Expr

Як вже зазначалося, програма працює зі всіма типами виразів і є універсальною. Її застосування, як правило, не викликає складнощів, тим паче, що вона вже присутня у більшості дистрибутивів Linux і доступна для встановлення з репозиторію. Скористаємося можливостями термінальної роботи з нашою системою Ubuntu 20.0 та протестуємо програму *expr*.

Насамперед, переглянемо опис її синтаксичних конструкцій за допомогою довідкової системи. Для цього введемо у терміналі наступну команду:

```
$ expr --help
```

```
root@dedicated:~# expr --help
Usage: expr EXPRESSION
  or:  expr OPTION

      --help     display this help and exit
      --version  output version information and exit

Print the value of EXPRESSION to standard output.  A blank line below
separates increasing precedence groups.  EXPRESSION may be:

  ARG1 | ARG2       ARG1 if it is neither null nor 0, otherwise ARG2

  ARG1 & ARG2       ARG1 if neither argument is null or 0, otherwise 0

  ARG1 < ARG2       ARG1 is less than ARG2
  ARG1 <= ARG2      ARG1 is less than or equal to ARG2
  ARG1 = ARG2       ARG1 is equal to ARG2
  ARG1 != ARG2      ARG1 is unequal to ARG2
  ARG1 >= ARG2      ARG1 is greater than or equal to ARG2
  ARG1 > ARG2       ARG1 is greater than ARG2

  ARG1 + ARG2       arithmetic sum of ARG1 and ARG2
  ARG1 - ARG2       arithmetic difference of ARG1 and ARG2

  ARG1 * ARG2       arithmetic product of ARG1 and ARG2
  ARG1 / ARG2       arithmetic quotient of ARG1 divided by ARG2
  ARG1 % ARG2       arithmetic remainder of ARG1 divided by ARG2

  STRING : REGEXP   anchored pattern match of REGEXP in STRING

  match STRING REGEXP        same as STRING : REGEXP
  substr STRING POS LENGTH   substring of STRING, POS counted from 1
  index STRING CHARS         index in STRING where any CHARS is found, or 0
  length STRING              length of STRING
  + TOKEN                    interpret TOKEN as a string, even if it is a
                               keyword like 'match' or an operator like '/'

  ( EXPRESSION )             value of EXPRESSION
```

Як можна переконатися, засіб здатний виконувати математичні операції та логіку, здійснювати порівняння аргументів та використовувати шаблони пошуку для регулярних виразів.

Загальний синтаксис оператора для операції складання буде наступним:

```
expr apr1 + apr2
```

Введемо наступну команду у терміналі:

```
$ expr 25 + 5
```

```
root@dedicated:~# expr 25 + 5
30
root@dedicated:~#
```

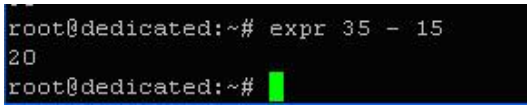
Команда одразу ж видає результат, у даному випадку дорівнює числу 30.

Загальний синтаксис оператора для операції віднімання буде таким:

```
expr apr1 - apr2
```

Введемо для перевірки:

```
$ expr 35 - 15
```



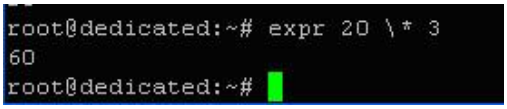
Отже, результат операції вірний.

Синтаксис оператора для операції множення:

```
expr apr1 \* apr2
```

Перевіримо це:

```
$ expr 20 \* 3
```



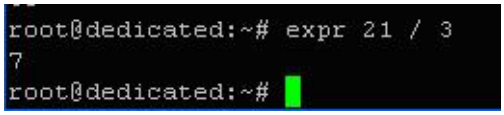
Результат вірний, це число 60.

Синтаксис оператора для операції ділення:

```
expr apr1 / apr2
```

Введемо у терміналі:

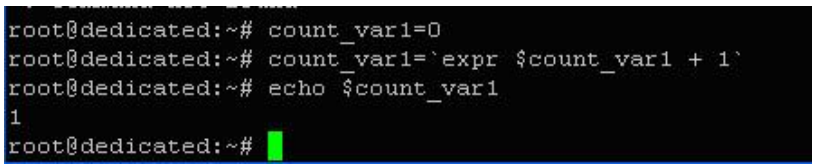
```
$ expr 21 / 3
```



Все вірно.

Оператор дозволяє організувати процес інкрементації змінної. Для демонстрації цього виконаємо кілька попередніх команд:

```
$ count_var1=0
$ count_var1=`expr $count_var1 + 1`
$ echo $count_var1
```



Тут ми спочатку присвоїли нульове значення змінній *coun_var1*, а потім за допомогою оператора *expr* збільшили її значення на 1. Цей механізм можна застосовувати у подальшому при організації обробки даних у сценаріях.

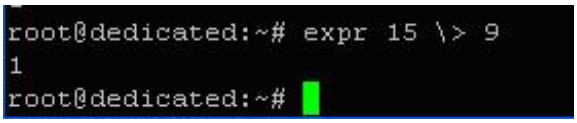
Синтаксис оператора для операції порівняння аргументів є наступним:

```
expr apr1 \> apr2
```

Результатом роботи буде значення 1, якщо перший аргумент більше другого, і 0, якщо не більше.

Перевіримо працездатність конструкції:

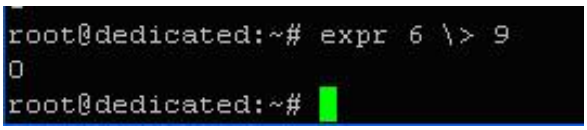
```
$ expr 15 \> 9
```



Отримали значення 1, як і повинно бути.

Тепер введемо інші числа:

```
$ expr 6 \> 9
```



Отримали нульовий результат, тобто, все вірно, оскільки число 6 не може бути більшим за число 9.

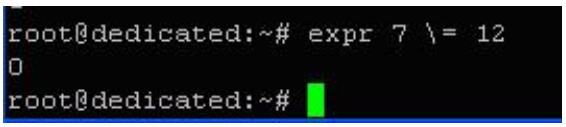
Для визначення рівності аргументів застосовується наступна конструкція:

```
expr apr1 /= apr2
```

Якщо аргументи мають однакові значення, то результатом буде значення 1, у протилежному випадку – 0.

Введемо у терміналі:

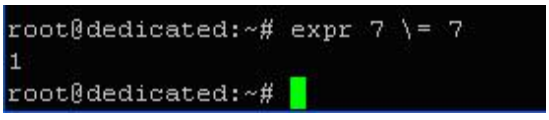
```
$ expr 7 \= 12
```



Повернуто 0, оскільки числа не співпадають.

Змінимо числа у команді:

```
$ expr 7 \= 7
```



Результатом є значення 1, тому що числа однакові.

Продемонструємо синтаксис і роботу розглянутих та інших конструкцій програми *expr* безпосередньо у сценарії. Для цього створимо файл сценарію із ім'ям *lesson_expr* та введемо в нього наведений нижче код:

```
$ nano lesson_expr.sh
```

Код сценарію:

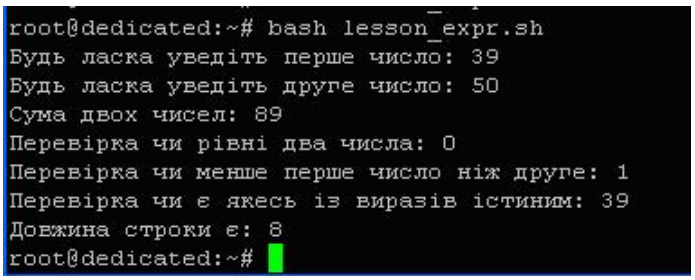
```
#!/bin/bash
len=Windows7
read -p «Будь ласка уведіть перше число: « n1
read -p «Будь ласка уведіть друге число: « n2
rezult1=`expr $n1 + $n2`
rezult2=`expr $n1 \= $n2`
rezult3=`expr $n1 \< $n2`
rezult4=`expr $n1 \| $n2`
rezult5=`expr length $len`
echo  «Сума двох чисел: $rezult1»
echo  «Перевірка чи рівні два числа: $rezult2»
echo  «Перевірка чи менше перше число ніж друге: $rezult3»
echo  «Перевірка чи є якесь із виразів істинним: $rezult4»
echo  «Довжина строки є: $rezult5»
```

Збережемо внесені у файл зміни (ctrl+O, Enter) та вийдемо з редактора (ctrl+X).

Сутність роботи наведеного коду полягає у обробці різними способами введених користувачем двох чисел, а також у виконанні операції зі строковим виразом. Строкове значення *Windows7* присвоєне змінній *$len*. *Expr* підраховує кількість символів у вказаному строковому значенні. Результати роботи всіх конструкцій фіксуються у змінних із назвою *$rezultN*, де *N* – номер операції. Потім значення вказаних змінних виводяться на екран за допомогою оператора *echo*.

Запустимо сценарій на виконання:

```
$ bash lesson_expr.sh
```



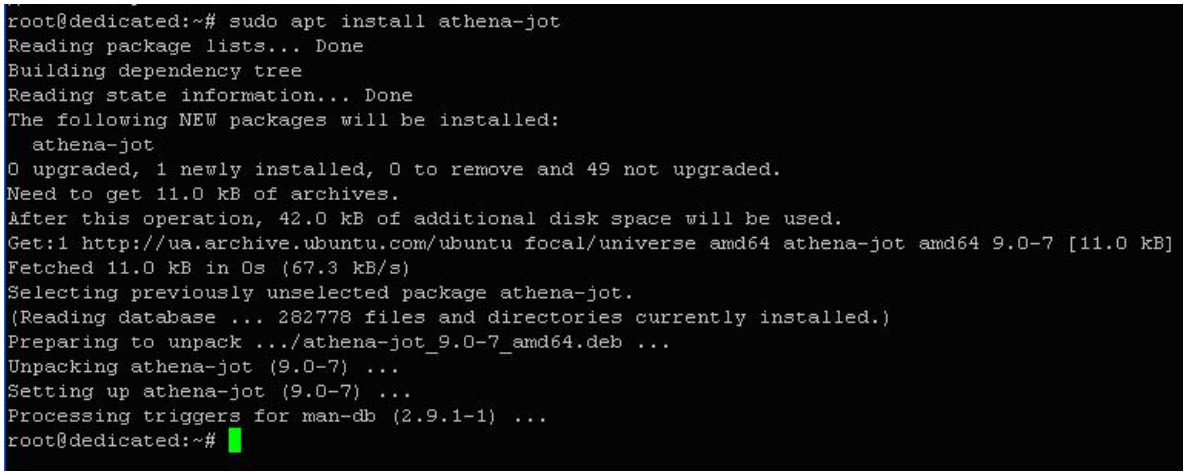
Можна переконатися, що результати всіх операцій є вірними.

Команда Jot

Засіб дозволяє створювати довільні списки чисел на основі вказаних значень початкового (базового) числа та кількості чисел, котрі виводяться у списку.

Для роботи із вказаним програмним засобом, його треба встановити із репозиторію. Для цього введемо відповідну команду:

```
$ sudo apt install athena-jot
```



Отже, програма *athena-jot* успішно встановлена у системі.

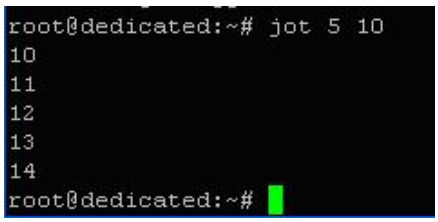
Розглянемо способи її використання та синтаксис. Перший варіант використання – це вивід заданої кількості чисел після базового числа. У цьому разі синтаксис конструкції буде виглядати наступним чином:

```
jot m n
```

Де *n* – базове число; *m* – кількість чисел, котрі будуть виведені після числа *n*.

Наведемо конкретний приклад. Введемо у терміналі:

```
$ jot 5 10
```



Тут число 5 – це кількість чисел, що виводяться, а 10 – базове число. За результатами роботи виведені значення з 10 до 14, тобто, саме базове число також включається у загальний список.

Другий варіант використання – це організація зворотного друку чисел. Синтаксис конструкції у цьому випадку буде наступним:

```
jot  m n o
```

Де *n* – базове число; *m* – кількість чисел, що передує числу *n; o* – значення числа, до котрого можна виводити список чисел.

Наведемо приклад:

$ jot 9 10 3

Можна переконатися, що числа виведені у зворотному порядку, починаючи з базового. І вивід зупинено на числі 3.

Команди Factor та Bc

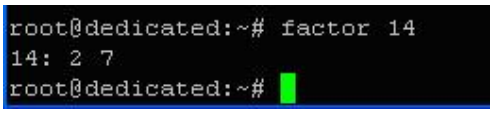
Програма *factor* використовується для розрахунку значень коефіцієнтів вказаного у команді числа.

Синтаксис наступний:

factor number

Наведемо приклад:

$ factor 14



Як бачимо, коефіцієнти числа 14 розраховані вірно.

Програма *bc* корисна у випадку необхідності швидкого виконання складних обчислень у *bash*. Вона може розраховувати значення тригонометричних функцій, отримувати квадратний корінь числа та обробляти логічні вирази. Таке різноманіття можливостей та швидкість роботи робить її дуже зручною при застосуванні у командному рядку і у сценаріях.

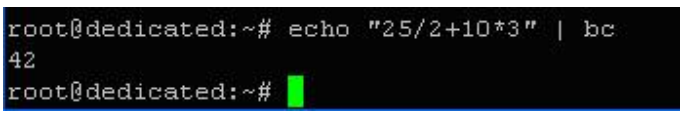
Загальний синтаксис має наступний вигляд:

echo «math_expr» | bc

Де *math_expr* – будь-який математичний вираз, котрий підлягає миттєвому розрахунку.

Наведемо приклад. Введемо у терміналі:

$ echo «25/2+10*3» | bc



Можна переконатися, що значення вказаного у конструкції виразу розраховано вірно.

Команда Grep

Засіб призначений для організації роботи із регулярними виразами, котрі є наборами символів, що задають шаблон пошуку даних. Програму можна використовувати для пошуку та заміни даних, для перевірки умов при вході та інших важливих цілей.

Загальний синтаксис програми виглядає так:

```
grep xxx file
```

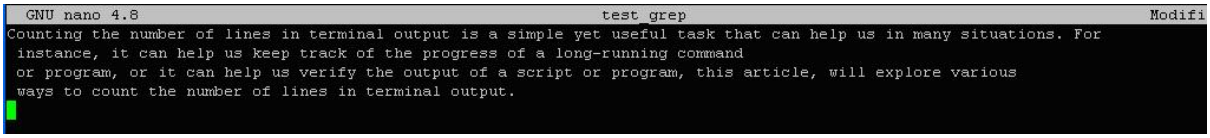
Де *xxx* – шаблон для пошуку у форматі *regex*, *file* – повна назва файлу, у котрому необхідно виконати пошук чи заміну даних.

Для можливості демонстрації роботи із засобом та для виконання повноцінних запитів, створимо файл із ім'ям *test_grep*, у котрий введемо наступний текст (скорочений варіант):

Counting the number of lines in terminal output is a simple yet useful task that can help us in many ..... will explore various ways to count the number of lines in terminal output.

Викличемо редактор nano

```
$ nano test_grep
```

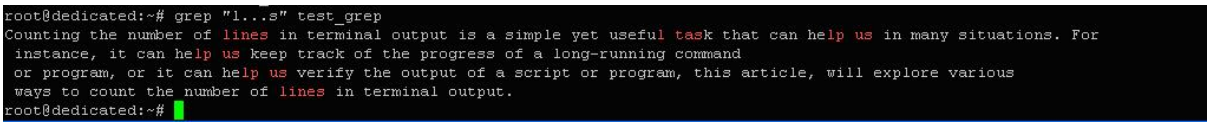


Збережемо внесені зміни та закриємо файл.

Тепер будемо застосовувати певні формати шаблонів для пошуку потрібних даних.

Введемо у терміналі:

```
$ grep «l...s» test_grep
```



Ключовим елементом виразу є символ «.», котрим позначається відсутній у текстовому рядку символ. У даному випадку, будуть шукатися послідовності символів, котрі починаються з букви «l», посередині можуть мати три будь-яких символи і закінчуються буквою «s». Результат роботи конструкції показаний вище. Як видно зі скриншоту, виділяються послідовності не тільки у межах одного слова, але й з різних слів, котрі знаходяться поруч. Також добре видно, що символом «.» позначаються не тільки існуючі букви, але й пусті символи.

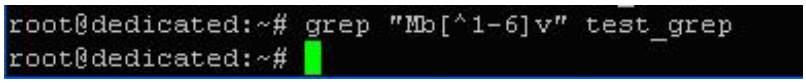
Тепер використаємо наступний шаблон пошуку:

$ grep «C[oui]g» test_grep

Ключовим елементом тут є квадратні дужки []. Ними позначаються діапазони позицій символів у шаблонах. У нашому випадку, буде відбуватися пошук всіх послідовностей символів, котрі починаються з символу «C», закінчуються символом «g» та можуть мати посередині комбінацію лише з трьох латинських букв: *o, u та i*. Тобто, ці символи є допустимими, а всі інші – ні.

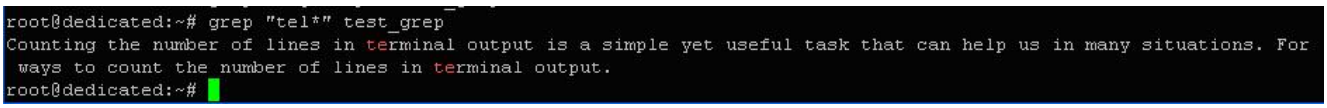
Розглянемо наступний пошуковий шаблон:

$ grep «Mb[^1-6]v» test_grep



Ключовим елементом тут є квадратні дужки із спеціальним символом у середині: [^]. Сутність його використання полягає у пошуку будь-яких знаків, окрім тих, що вказані у дужках. У нашому випадку, будуть шукатися послідовності, котрі починаються із «Mb», закінчуються буквою «v», а посередині допустимі всі символи, окрім цифр від 1 до 6.

$ grep «tek*» test_grep



Ключовим елементом тут є знак «*», котрим позначається будь-яка кількість входжень букви «k», включаючи її відсутність. Як видно з результатів, таким вимогам відповідають лише послідовності виду «te».

2.10. Запуск python скрипта в Linux

Python – дуже популярна мова програмування для написання різних системних скриптів у Linux. У Windows, там де не вистачає можливостей командної оболонки використовується PowerShell. У Linux же, коли можливостей Bash не вистачає, використовується мова Python.

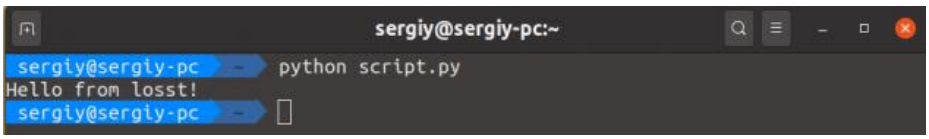
Цією мовою написана величезна кількість системних програм, серед них пакетний менеджер apt, відеоредактор OpenShot, а також безліч скриптів, які ви можете встановити за допомогою утиліти pip. У цій невеликій статті ми розглянемо як запустити Python скрипт у Linux за допомогою терміналу різними способами.

Для прикладу нам знадобиться Python скрипт. Щоб не брати будь-який з наявних скриптів, давайте напишемо свій:

```
vi script.py
print(«Hello from losst!»)
```

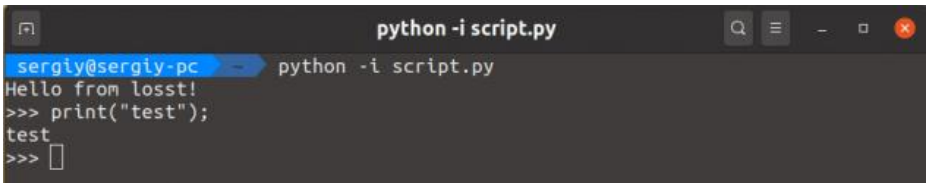
Для того щоб запустити скрипт, необхідно передати його інтерпретатору Python. Для цього просто відкрийте термінал за допомогою поєднання клавіш Ctrl+Alt+T, перейдіть до папки зі скриптом і виконайте:

```
python script.py
```



Якщо ви хочете, щоб після виконання скрипта відкрилася консоль, у якій можна інтерактивно виконувати команди мови Python, використовуйте опцію -i:

```
python -i script.py
```



Але як ви могли помітити, під час запуску apt або openshot не треба писати слово python. Це набагато зручніше. Давайте розберемося як це реалізувати. Якщо ви не хочете вказувати інтерпретатор у командному рядку, його треба вказати в самому скрипті. Для цього слід на початок скрипта додати такий рядок:

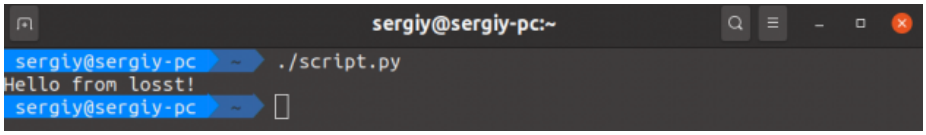
```
vi script.py

#!/usr/bin/python
```

Збережіть зміни, а потім зробіть файл скрипта виконуваним за допомогою такої команди:

```
chmod ugo+x script.py
```

Після цього можна запустити скрипт Python просто звертаючись до його файлу:

```
./script.py
```



Якщо прибрати розширення .py і перемістити скрипт у каталог, що міститься у змінній PATH, наприклад /usr/bin/, то його можна буде виконувати ось так:

```
script
```

Як бачите, запуск команди python Linux виконується досить просто і для цього навіть є кілька способів.

ЛЕКЦІЯ 3. Запуск і зупинка системи.

3.1. Огляд процесу завантаження

Процедури завантаження сильно змінилися за останні роки. Поява сучасних (UEFI) BIOS спростила початкові етапи завантаження, принаймні з концептуальної точки зору. На пізніших етапах більшість дистрибутивів Linux тепер використовують демон системного менеджера `systemd` замість традиційного `init`. `systemd` спрощує процес завантаження, додаючи управління залежностями, підтримку паралельних процесів запуску, комплексний підхід до ведення журналів тощо.

Керування завантаженням також змінилося з міграцією систем у хмару. Перехід до віртуалізації, хмарних екземплярів та контейнеризації зменшив потребу адміністраторів торкатися фізичного обладнання. Натомість з'явилося керування образами, API та панелі керування.

Під час завантаження ядро завантажується в пам'ять і починає виконуватися. Виконуються різноманітні завдання ініціалізації, після чого система стає доступною для користувачів. Загальний огляд цього процесу показано на рис.3.1.

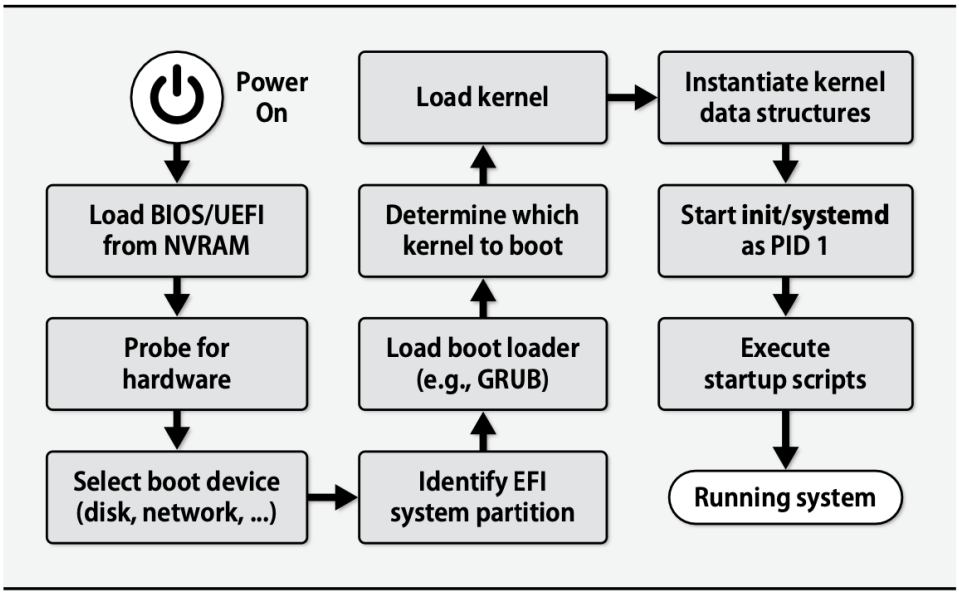


Рисунок 3.1 – Процес завантаження Linux та UNIX

Адміністратори не мають прямого інтерактивного контролю над більшістю кроків, необхідних для завантаження системи. Натомість, адміністратори можуть змінювати конфігурацію завантажувача, редагуючи конфігураційні файли для сценаріїв запуску системи або змінюючи аргументи, які завантажувач передає ядру.

Перш ніж система повністю завантажиться, необхідно перевірити та змонтувати файлові системи, а також запустити системні демони. Цими процедурами керує низка сценаріїв командного інтерпретатора (іноді їх називають «init-скрипти») або модульних файлів, які послідовно запускаються `init` або розбираються `systemd`. Точна структура сценаріїв запуску і спосіб їх виконання відрізняються у різних системах.

3.2. Початкове завантаження

Під початковим завантаженням (bootstrapping) розуміють запуск системи під час увімкнення живлення. Оскільки звичайні засоби операційної системи на даному етапі ще не доступні, система повинна «самозавантажитися», у буквальному сенсі «обслужити себе самостійно». Під час цього процесу ядро системи завантажується в пам'ять і активується. Потім виконується низка ініціалізаційних завдань, після чого система готова до обслуговування користувачів.

Початкове завантаження – це період особливої вразливості системи. Помилки в конфігурації, збої в роботі або відсутність потрібного обладнання, пошкодження файлових систем можуть завадити комп'ютеру нормально почати роботу. Налаштування режимів завантаження часто є одним із перших завдань, яке доводиться вирішувати адміністратору в новій системі, особливо під час додавання нового обладнання. На жаль, це завдання – одне з найскладніших, і для його вирішення необхідно добре знати багато інших аспектів системи.

Коли вмикається живлення, запускається на виконання завантажувальний код, що зберігається на постійному запам'ятовуючому пристрої. Він має запустити ядро. Ядро опитує стан апаратних пристроїв, а потім запускає демон init, ідентифікатор якого завжди дорівнює 1.

Перш ніж система повністю завантажиться, має бути перевірено і змонтовано файлові системи та запущено системні демони. Відповідні процедури реалізуються за допомогою сценаріїв інтерпретатора команд, які послідовно запускаються демоном init. Конкретна структура сценаріїв і спосіб їх виконання залежать від системи. Усі ці питання буде розглянуто в цій лекції.

При нормальній роботі системи самі виконують початкове завантаження в автономному режимі, після чого до них можуть отримати віддалений доступ адміністратори та користувачі. Однак адміністраторам необхідно мати інструмент відновлення системи в разі, якщо несподівана відмова дисковода або якась проблема конфігурації завадить системі нормально виконати процес початкового завантаження. Системи UNIX (замість завантаження «за повною програмою») можуть обмежитися тільки найнеобхіднішим, а саме активізацією командної оболонки на системній консолі. Цей варіант називають входом системи в так званий «однокористувацький режим», режим відновлення або профілактичний режим – усі ці терміни в цій лекції взаємозамінні. Однокористувацький режим не дає змоги виконувати мережеві операції, а для використання системної консолі вам потрібно мати фізичний доступ до неї.

У більшості систем для входу в однокористувацький режим під час початкового завантаження необхідно передати ядру в командному рядку певний параметр. Якщо система вже завантажена і працює, ви можете перевести її в однокористувацький режим за допомогою команди `shutdown` або `telinit`.

3.3 Етапи завантаження

Типова процедура початкового завантаження складається з шести окремих етапів:

- зчитування початкового завантажувача з головного завантажувального запису;

- завантаження та ініціалізація ядра;
- виявлення і конфігурування пристроїв;
- створення процесів ядра;
- втручання адміністратора (тільки в однокористувацькому режимі);
- виконання системних сценаріїв запуску.

Майже всі етапи не потребують інтерактивного контролю з боку адміністратора. Адже адміністратори мають можливість керувати завантаженням системи, редагуючи файли конфігурації для сценаріїв запуску системи або змінюючи аргументи, які передаються завантажувачем ядру.

3.4. Ініціалізація ядра

Ядро являє собою програму, і як перше завдання початкового завантаження виконується запис цієї програми в пам'ять для подальшого виконання. Ім'я файлу ядра дає його виробник, але за традицією воно називається /unix або /vmunix. У Linux-системах ядро зазвичай отримує дороговказне ім'я у вигляді варіації на тему /boot/vmlinuz.

У більшості систем завантаження здійснюється у два етапи. Спочатку в пам'ять комп'ютера з диска зчитується (за допомогою коду, записаного на постійному запам'ятовуючому пристрої) невелика програма початкового завантаження (іменована початковим завантажувачем), яка потім виконує власне завантаження ядра. Ця процедура відбувається поза UNIX-доменом і тому не стандартизована серед систем.

Ядро виконує тести, що дають змогу визначити, скільки пам'яті є в розпорядженні системи. Частина внутрішніх структур ядра має фіксований розмір, тому ядро резервує певний обсяг фізичної пам'яті для самого себе. Ця пам'ять недоступна для користувацьких процесів. Ядро видає на консоль повідомлення про загальний об'єм фізичної пам'яті та об'єм пам'яті, доступний для користувацьких процесів.

3.5. Конфігурування апаратних засобів

Одне з перших завдань ядра – дослідження компонентів апаратного забезпечення. У процесі тестування різних системних шин та інвентаризації обладнання ядро виводить на консоль коротку інформацію про кожен виявлений пристрій. У багатьох випадках ядро завантажує драйвери пристроїв як незалежні модулі ядра. Для систем, орієнтованих на персональні комп'ютери, дистрибутиви включають ядро, яке здатне працювати в більшості апаратних конфігурацій комп'ютерів, потребуючи при цьому мінімального настроювання або ж взагалі обходячись без нього.

Процес конфігурації апаратного забезпечення має бути відносно простим для адміністраторів, особливо в середовищі Linux. Готові ядра мають модульну структуру і автоматично виявляють більшу частину пристроїв. Проте ви можете зустріти нерозпізнані пристрої.

3.6. Створення процесів ядра

Після завершення етапу базової ініціалізації ядро створює в ділянці пам'яті, виділеній для користувацьких програм, кілька процесів, що «самовиконуються». Це відбувається «в обхід» стандартного системного механізму fork.

Кількість таких процесів залежить від операційної системи, хоча демон init завжди має ідентифікатор процесу (PID), що дорівнює 1. Більшість систем UNIX використовує sched як процес з ідентифікатором 0.

У Linux процес з ідентифікатором PID, що дорівнює 0, відсутній. Демон init працює в супроводі з різними оброблювачами пам'яті та сигналів ядра. Усі ці процеси мають ідентифікатори з низькими номерами, а їхні імена в лістингах команди ps укладені в квадратні дужки (наприклад, [kacpid]). Іноді імена процесів можуть містити наприкінці символ косої риски і цифру, як, наприклад, [kblockd/0]. Число вказує процесор, на якому виконується цей процес. Ця інформація може бути корисною під час налаштування багатопроцесорної системи.

Із цих процесів тільки init є повноцінним користувацьким процесом; решта фактично є частинами ядра, що були зроблені процесами з концептуальних міркувань.

Системи UNIX створюють подібні процеси ядра, але оскільки ці процеси віддзеркалюють особливості конкретної реалізації ядра, жодні імена або функції не можуть бути однаковими для різних систем. На щастя, адміністраторам ніколи не доводиться взаємодіяти з цими процесами безпосередньо.

Після створення цих процесів ядро більше не бере участі в процедурі початкового завантаження системи. До цього моменту, однак, ще не створено жодного з процесів, що керують базовими операціями (наприклад, реєстрацією користувачів у системі), і більшість демонів не запущено. Про все це подбає (у деяких випадках побічно) демон init.

3.7. Дії оператора (тільки в режимі відновлення)

Якщо систему потрібно запустити в режимі відновлення, оператор встановлює в командному рядку спеціальний прапор, а ядро передає цю інформацію демону init як повідомлення про запуск. Під час однокористувацького завантаження ви повинні отримати запрошення ввести пароль користувача root. Якщо він введений правильно, запускається інтерпретатор команд з правами суперкористувача. Можна не задавати пароль, а просто натиснути комбінацію клавіш <Ctrl+D>, після чого завантаження продовжиться в багатокористувацькому режимі.

В однокористувацькому режимі команди виконуються майже так само, як і в повністю завантаженій системі. Однак іноді монтується тільки кореневий розділ. Для того щоб можна було використовувати програми, що знаходяться поза каталогами /bin, /sbin або /etc, необхідно вручну змонтувати інші файлові системи.

У багатьох однокористувацьких середовищах кореневий каталог файлової системи монтується в режимі тільки для читання. Якщо каталог /etc є частиною кореневої файлової системи (звичайна ситуація), редагування багатьох файлів конфігурації буде неможливим. Щоб вийти з положення, доведеться почати однокористувацький сеанс із повторного монтування каталогу / в режимі читання/запису. Потрібну дію в Linux-системах виконує наступна команда.

```
# mount -o rw, remount /
```

У більшості інших систем ви можете виконати команду mount /, щоб реалізувати звернення до файлу fstab або vfstab і з'ясувати, як має бути змонтована файлова система.

Команда fsck, яка перевіряє і відновлює пошкоджені файлові системи, зазвичай виконується під час автоматичного завантаження. Якщо система запускається в однокористувацькому режимі, команду fsck доведеться вводити вручну.

Коли інтерпретатор команд однокористувацького режиму завершить свою роботу, система продовжить завантаження в нормальному режимі.

3.8. Виконання сценаріїв запуску системи

У той момент, коли система зможе виконувати сценарії запуску, її вже можна назвати UNIX. Це ще не повністю завантажена система, але «загадкових» етапів процесу завантаження більше не залишилося. Файли сценаріїв являють собою звичайні командні файли, які вибирає і запускає демон init за складним, але зрозумілим алгоритмом.

Точне місцезнаходження, вміст і організація сценаріїв запуску заслуговують на окреме вивчення.

3.9. Завершення процесу завантаження

Після виконання сценаріїв ініціалізації система повністю готова до роботи. Такі системні демони, як DNS- і SMTP-сервери, приймають і обслуговують підключення. Не забувайте про те, що демон init продовжує відігравати важливу роль навіть після завершення початкового завантаження.

У демона init є один однокористувацький і кілька багатокористувацьких «рівнів виконання», що визначають, які ресурси системи будуть доступні користувачеві.

3.10. Завантаження системи на персональному комп'ютері

До цього моменту описувалася загальна схема початкового завантаження. Тепер деякі найважливіші (і складні) її етапи необхідно розглянути докладніше, проаналізувавши особливості функціонування Intel-систем.

Завантаження системи на персональному комп'ютері – це багатоступеневий процес. Коли вмикається комп'ютер, починає виконуватися код, записаний на постійному запам'ятовуючому пристрої. Точне його місцезнаходження і структура залежать від типу обладнання. У комп'ютерах, створених спеціально для UNIX або іншої комерційної операційної системи, код «прошивається» розробником, який заздалегідь задає алгоритм під'єднання пристроїв, базової ініціалізації мережі та розпізнавання локальних файлових систем. Це дуже зручно для системного адміністратора. Йому достатньо ввести лише ім'я нового файлу ядра, а код постійного запам'ятовуючого пристрою автоматично виявить і прочитає цей файл.

На персональних комп'ютерах код початкового завантаження представлений у вигляді базової підсистеми вводу-виводу – BIOS (Basic Input/Output System), яка є надзвичайно спрощеною, як порівняти з фірмовим кодом UNIX-станцій. Насправді в системі BIOS є кілька рівнів коду: для самого комп'ютера, для відеоплати, для SCSI-адаптера, якщо такий є, та іноді для інших периферійних пристроїв, як-от мережеві плати.

Вбудованому коду BIOS відомо про пристрої, розташовані на материнській платі, зокрема про IDE- і SATA-контролери (і жорсткі диски), плату мережевого адаптера, вимірювач потужності і температури, контролер клавіатури, послідовні і паралельні порти. SCSI-адаптери знають тільки про пристрої, підключені безпосередньо до них. На щастя, за останні кілька років складні взаємодії, необхідні для забезпечення спільної роботи цих пристроїв, було стандартизовано, і тепер майже не потребують втручання оператора.

У режимі конфігурування можна вибрати, з якого пристрою потрібно завантажуватися. Зазвичай послідовність завантаження задається у вигляді правила, наприклад: «спочатку – DVD, потім – «флешка» (USB drive), в останню чергу – жорсткий диск». Поширеним варіантом також вважається мережеве завантаження за допомогою середовища PXE.

Коли комп'ютер визначив, з якого пристрою слід завантажуватися, зчитується його (пристрою) перший блок. Цей сегмент (512 байт) називається головним завантажувальним записом (master boot record – MBR). У ньому зберігається програма, яка повідомляє комп'ютеру про те, в якому розділі диска розташована програма вторинного завантаження (завантажувач операційної системи).

Стандартна програма, що міститься в головному завантажувальному записі, дає комп'ютеру вказівку витягти завантажувач операційної системи з першого розділу диска. У деяких системах підтримуються більш складні програми, які можуть працювати з декількома операційними системами і ядрами. Знайшовши розділ, з якого буде виконуватися завантаження системи, програма головного завантажувального запису намагається запустити завантажувальну програму, пов'язану з цим розділом. У разі успіху цій програмі передаються повноваження щодо подальшого завантаження ядра.

3.11. UEFI

Специфікація UEFI включає сучасну схему розбиття диска на розділи, відому як GPT (GUID Partition Table, де GUID означає «глобально унікальний ідентифікатор»). UEFI також розуміє файлові системи FAT (File Allocation Table - таблиця розміщення файлів), просту, але функціональну схему, яка походить з MS-DOS. Ці особливості разом визначають поняття системного розділу EFI (ESP). Під час завантаження мікропрограма звертається до таблиці розділів GPT, щоб визначити ESP. Потім вона зчитує налаштовану цільову програму безпосередньо з файлу в ESP і виконує її.

Оскільки ESP є звичайною файловою системою FAT, її можна монтувати, читати, записувати і обслуговувати за допомогою будь-якої операційної системи. Ніяких завантажувальних блоків на диску не потрібно.

Насправді, технічно взагалі не потрібен завантажувач. Ціллю завантаження UEFI може бути ядро UNIX або Linux, яке було налаштоване на пряме завантаження UEFI, таким чином створюючи завантажувач без завантажувача. На практиці, однак, більшість систем все ще використовують завантажувач, частково тому, що так легше підтримувати сумісність із застарілими BIOS.

UEFI зберігає назву шляху для завантаження з ESP як параметр конфігурації. За відсутності конфігурації він шукає стандартний шлях, зазвичай `/efi/boot/bootx64.efi` на сучасних системах Intel. Більш типовим шляхом у

налаштованій системі (у цьому випадку для Ubuntu і завантажувача GRUB) буде `/efi/ubuntu/grubx64.efi`. Інші дистрибутиви дотримуються подібної домовленості.

UEFI визначає стандартні API для доступу до апаратного забезпечення системи. У цьому відношенні це щось на кшталт мініатюрної операційної системи у власному розумінні. Вона навіть передбачає додаткові драйвери пристроїв на рівні UEFI, які написані незалежною від процесора мовою і зберігаються в ESP. Операційні системи можуть використовувати інтерфейс UEFI, а можуть брати на себе безпосереднє керування обладнанням.

Оскільки UEFI має формальний API, ви можете переглядати і змінювати змінні UEFI (зокрема пункти меню завантаження) у запущеній системі. Наприклад, `efibootmgr -v` показує наступний результат конфігурації завантаження:

```
$ efibootmgr -v
BootCurrent: 0004
BootOrder: 0000,0001,0002,0004,0003
Boot0000* EFI DVD/CDROM PciRoot(0x0)/Pci(0x1f,0x2)/Sata(1,0,0) Boot0001* EFI Hard Drive
PciRoot(0x0)/Pci(0x1f,0x2)/Sata(0,0,0) Boot0002* EFI Network
PciRoot(0x0)/Pci(0x5,0x0)/MAC(001c42fb5baf,0) Boot0003* EFI Internal Shell
MemoryMapped(11,0x7ed5d000,0x7f0dcfff)/
FvFile(c57ad6b7-0515-40a8-9d21-551652854e37) Boot0004* ubuntu HD(1,GPT,020c8d3e-fd8c-
4880-9b61-
ef4cffc3d76c,0x800,0x100000)/File(\EFI\ubuntu\shimx64.efi)
```

`efibootmgr` дозволяє змінювати порядок завантаження, вибирати наступний налаштований варіант завантаження або навіть створювати і знищувати завантажувальні записи. Наприклад, щоб встановити порядок завантаження для спроби завантаження з системного диска перед спробою завантаження з мережі і ігноруванням інших варіантів завантаження, можна скористатися командою

```
$ sudo efibootmgr -o 0004,0002
```

Тут ми вказуємо параметри `Boot0004` і `Boot0002` з наведених вище даних.

Можливість змінювати конфігурацію UEFI з простору користувача означає, що інформація про конфігурацію мікропрограми доступна для читання/запису – добре й погано одночасно. У системах (як правило, з `systemd`), які за замовчуванням дозволяють доступ на запис, `rm -rf /` може бути достатньо, щоб назавжди знищити систему на рівні прошивки; крім видалення файлів, `rm` також видаляє змінні та іншу інформацію UEFI, доступну через `/sys`. Не намагайтеся зробити це вдома!

3.12. GRUB: універсальний завантажувач

Програма GRUB (GRand Unified Boot loader), розроблена в рамках проєкту GNU, використовується як стандартний завантажувач у більшості UNIX- і Linux-систем, оснащених процесорами Intel. GRUB поставляється в складі багатьох дистрибутивів Linux і Solaris (з архітектурою x86) починаючи з версії 10. Завдання GRUB – вибрати з попередньо підготовленого списку ядро і завантажити його з опціями, заданими адміністратором.

За замовчуванням GRUB читає параметри завантаження з файлу /boot/grub/menu.lst або /boot/grub/grub.conf. Зчитування вмісту файлу конфігурації відбувається в період запуску (що саме по собі вже вражає), і це дає змогу реалізувати динамічні зміни під час кожного завантаження системи. Файли menu.lst і grub.conf відрізняються незначно, але мають аналогічний синтаксис. Системи Red Hat використовують файл grub.conf, а Solaris, SUSE і Ubuntu – файл menu.lst. Ось зразок файлу grub.conf.

```
default=0
   timeout=10
   splashimage=(hd0,0)/boot/grub/splash.xpm.gz
   title Red Hat Enterprise Linux Server (2.6.18-92.1.10.el5)
       root (hd0,0)
       kernel /vmlinuz-2.6.18-92.1.10.el5 ro root=LABEL=/
```

У цьому прикладі конфігурується єдина операційна система, яка буде завантажена автоматично (`default=0`), якщо протягом 10 секунд з клавіатури не надійдуть жодні дані (`timeout=10`). Коренева файлова система для конфігурації «Red Hat Enterprise Linux Server» розташована в розділі (`hd0,0`), що для GRUB означає звернення до першого розділу першого жорсткого диска системи (нумерація «першого розділу» та «першого диска» визначається в BIOS).

Програма GRUB завантажить ядро з файлу `/vmlinuz-2.6.18-92.1.10.e15`, а потім виведе початковий образ екрана, що зберігається у файлі `/boot/grub/splash.xpm.gz`. Шлях пошуку ядра вказується щодо розділу завантаження, який зазвичай монтується в каталозі `/boot`.

Програма GRUB підтримує потужний інтерфейс командного рядка і низку функцій, які дають змогу редагувати записи файлу конфігурації під час завантаження. Для того щоб перейти в режим командного рядка, у вікні завантаження GRUB необхідно ввести команду `c`. З командного рядка можна завантажувати операційні системи, не зазначені у файлі `grub.conf`, виводити на екран інформацію про систему і виконувати найпростішу перевірку файлової системи. Завантажувач надає також низку функцій, подібних до функцій інтерпретатора команд, зокрема – функції завершення неповністю введених команд і переміщення курсору. Будь-які дії, які можуть бути виконані за допомогою файлу `grub.conf`, можуть бути виконані і за допомогою командного рядка завантажувача GRUB.

Для одержання докладної інформації про завантажувач GRUB і його параметри командного рядка зверніться до офіційного керівництва: `gnu.org/software/grub/ manual`.

3.13. Параметри ядра

Програма завантажувача GRUB дає змогу передавати ядру параметри командного рядка. Як правило, ці параметри змінюють значення параметрів ядра, змушують його опитати конкретні пристрої, вказують шлях пошуку демона `init` або призначають конкретний кореневий пристрій.

Параметри ядра, відредаговані під час завантаження, не зберігаються. Щоб зберігати зміни на майбутні перезавантаження, відредагуйте рядок `kernel` у файлі `grub.conf` або `menu.lst`.

Ядро Linux постійно вдосконалюється за допомогою патчів, що підвищують рівень безпеки, виправлень помилок і нових функцій. Але, на відміну від інших програмних пакетів, старі версії ядра зазвичай не замінюються новими. Нові ядра інсталюються поряд зі старими, щоб у разі виникнення проблеми ви могли тут же повернутися до старого ядра. Ця угода допомагає адміністраторам відмовитися від модернізації, якщо заплата ядра руйнує їхню систему. Після деякого часу завантажувальні меню GRUB заповнюються різними версіями ядра. Зазвичай вибір версії ядра за замовчуванням не становить небезпеки, але якщо виявиться, що ваша система не завантажується після додавання заплати, подбайте про можливість вибору іншого ядра.

3.14. Мультисистемне завантаження

Оскільки на одному персональному комп'ютері може бути встановлено кілька операційних систем, звичною є ситуація, коли комп'ютер завантажується в мультисистемному режимі. Для цього завантажувач має бути правильно сконфігуровано, щоб розпізнати наявні на локальних дисках операційні системи. У наступних розділах ми опишемо деякі проблемні блоки мультисистемного завантаження, а потім розглянемо конкретні приклади конфігурації.

У кожному розділі диска може зберігатися власний вторинний завантажувач, однак завантажувальний диск повинен мати тільки один головний завантажувальний запис. Тому необхідно вирішити, який завантажувач буде «головним». Як правило, вибір диктується особливостями наявних операційних систем. Для Intel-орієнтованих UNIX- і Linux-систем найкраще як головний завантажувач вибрати GRUB. При двоваріантному завантаженні Windows-системи завжди використовуйте завантажувач GRUB (а не завантажувач Windows).

Конфігурування завантажувача GRUB для виконання мультисистемного завантаження багато в чому аналогічне конфігуруванню завантаження тільки однієї операційної системи. Перш ніж вносити зміни у файл `grub.conf` або `menu.lst`, потрібно встановити бажані операційні системи.

3.15. Процедури перезавантаження і вимкнення

Історично склалося так, що машини UNIX і Linux були чутливі до того, як їх вимикати. Сучасні системи стали менш чутливими, особливо якщо використовується надійна файлова система, але завжди корисно вимкнути комп'ютер правильно, коли це можливо.

Споживчі операційні системи минулого навчили багатьох сисадмінів перезавантажувати систему як перший крок у налагодженні будь-якої проблеми. Тоді це була адаптивна звичка, але сьогодні вона частіше призводить до втрати часу і переривання роботи. Зосередьтеся на виявленні першопричини проблем, і ви, ймовірно, помітите, що перезавантажуватиметеся рідше.

Тим не менш, варто перезавантажуватися після модифікації сценарію запуску або внесення значних змін до конфігурації. Ця перевірка гарантує, що система зможе

успішно завантажитися. Якщо ви створили проблему, але виявили її лише через кілька тижнів, ви навряд чи пам'ятатимете деталі ваших останніх змін.

3.16. Завершення роботи фізичних систем

Команда `halt` виконує основні обов'язки, необхідні для завершення роботи системи. `halt` реєструє завершення роботи, завершує несуттєві процеси, скидає кешовані блоки файлової системи на диск, а потім завершує роботу ядра. На більшості систем, `halt -p` вимикає систему як фінальне завершення роботи.

`reboot` по суті ідентична до `halt`, але призводить до перезавантаження системи, а не до зупинки.

Команда `shutdown` є надбудовою над `halt` і `reboot`, яка забезпечує вимкнення за розкладом і зловісні попередження для користувачів, які увійшли до системи. Вона з'явилася ще за часів систем з розподілом часу і наразі є значною мірою застарілою. `shutdown` не робить нічого технічно важливого, окрім зупинки і перезавантаження, тому можете не звертати на неї уваги, якщо у вас не багатокористувацька система.

3.17. Вимкнення хмарних систем

Ви можете зупинити або перезапустити хмарну систему зсередини сервера (за допомогою зупинки або перезавантаження, як описано в попередньому розділі) або з веб-консолі хмарного провайдера (або еквівалентного API).

Взагалі кажучи, вимкнення з хмарної консолі подібне до вимкнення електроенергії. Краще, якщо віртуальний сервер сам керує своїм вимкненням, але не соромтеся вимкнути віртуальний сервер з консолі, якщо він не реагує на запити. Що ще можна зробити?

У будь-якому випадку, переконайтеся, що ви розумієте, що означає вимкнення з точки зору хмарного провайдера. Було б прикро знищити вашу систему, коли все, що ви хотіли зробити – це перезавантажити її.

У всесвіті AWS операції зупинки та перезавантаження виконують те, що ви очікуєте. «Завершити» виводить пристрій з експлуатації і видаляє його з вашого інвентаря. Якщо основний пристрій зберігання даних налаштовано на «видалення при завершенні роботи», буде знищено не лише ваш екземпляр, але й дані на кореневому диску. Це цілком нормально, якщо це те, чого ви очікуєте. Ви можете увімкнути «захист при завершенні роботи», якщо вважаєте, що це неправильно.

ЛЕКЦІЯ 4. Керування доступом.

Керування доступом до ресурсів належить до сфери, яка активно досліджується в наші дні і тривалий час була однією з найпроблемніших у проєктуванні операційних систем. Для окремих користувачів операційні системи визначають облікові записи, які дають змогу виконувати безліч операцій: редагування текстових файлів, реєстрація на віддалених комп'ютерах, встановлення імені головного вузла, інсталяція нових програм тощо. Систему управління доступом можна розглядати як чорну скриньку, яка (на вході) приймає потенційні дії (пари «користувач/операція») і видає (на виході) рішення залежно від допустимості кожної дії.

У системах UNIX і Linux не існує єдиної чорної скриньки, яка реалізує управління доступом. Насправді тут можна говорити про ціле сховище чорних скриньок – і це сховище «з'їдає» пам'ять. У цій лекції ми спочатку повернемося до витоків UNIX (щоб зрозуміти, як склалася нинішня ситуація з керуванням доступом), а потім розглянемо сучасні системи управління доступом в UNIX і Linux (як у теорії, так і на практиці), після чого зупинимося на інструментах, що допомагають зробити адміністрування сфери управління доступом (і особливо в частині, пов'язаній із привілеями всемогутнього суперкористувача) відносно безболісним.

4.1 Традиційні методи керування доступом у системах UNIX

Навіть у перших і найпростіших версіях UNIX ніколи не було єдиного «центру» управління доступом. Проте існували загальні правила, які впливали на проєктування систем.

- Об'єкти (наприклад, файли і процеси) мають власників. Власники мають великий (але необов'язково необмежений) контроль над своїми об'єктами.
- Ви є власниками нових об'єктів, створюваних вами.
- Користувач root з особливими правами, відомий як суперкористувач, може діяти як власник будь-якого об'єкта в системі.
- Тільки суперкористувач може виконувати адміністративні операції особливого значення.

У системах UNIX єдиної чорної скриньки, що управляє доступом, не існує, оскільки код, який ухвалює рішення в цій царині, розосереджений по всій системі. Одні системні виклики (наприклад, `settimeofday`) обмежені користувачем root (під час виконання таких системних викликів просто перевіряється «особа» поточного користувача, і, якщо він виявиться не суперкористувачем, операція відкидається). Інші системні виклики (наприклад, `kill`) реалізують різні обчислення, які вимагають як відповідності власнику, так і забезпечення спеціальних умов для суперкористувача. Нарешті, файлова система реалізує власну систему управління доступом, причому ця система складніша за будь-яку іншу, реалізовану в ядрі. Наприклад, тільки файлова система використовує принцип UNIX-груп для управління доступом.

Ускладнення ситуації призвело до переплетення ядра і файлової системи. Наприклад, ви керуєте більшістю пристроїв і передаєте їм інформацію за допомогою файлів, які представляють їх у каталозі `/dev`. Оскільки ці файли пристроїв є об'єктами файлової системи, вони використовуються і в її семантиці управління доступом.

4.2. Керування доступом у файловій системі

Кожен файл у традиційній моделі UNIX-подібних систем належить власнику і групі, яку іноді називають «груповим власником». Власник файлу має особливий привілей, недоступний іншим користувачам системи: йому дозволено змінювати права доступу до файлу. Зокрема, власник може задати права доступу так, що ніхто, крім нього, не зможе звертатися до файлу.

Власником файлу завжди є одна людина, тоді як до групи власників можуть входити кілька користувачів. За традицією інформація про групи зберігалася у файлі `/etc/group`, але тепер її частіше зберігають на мережевому сервері NIS або LDAP.

Власник файлу має визначити, які операції можуть виконувати над файлом члени групи. Така схема допускає спільне використання файлів членами однієї групи. Наприклад, ми застосовуємо групу для управління доступом до файлів вихідного коду веб-сайту *www.admin.com*

Дізнатися ідентифікатори власників файлу можна за допомогою команди `ls -l ім'я_ файлу.`

Наприклад:

```
aix$ ls -l /home/serhii/todo
-rw------ 1 serhii stu 1258 Jun 4 18:15 /home/serhii/todo
```

Файлом володіє користувач `serhii`, а група, якій він належить, називається `stu`. Букви та дефіси в першому стовпчику означають права доступу до файлу.

Ядро і файлова система відстежують не текстові імена власників і груп, а їхні ідентифікатори. У найзагальнішому випадку ідентифікатори користувачів (скорочено UID - User ID) і відповідні їм імена зберігаються у файлі `/etc/passwd`, а ідентифікатори та назви груп (GID - group ID) знаходяться у файлі `/etc/group`. Текстові еквіваленти ідентифікаторів UID і GID визначаються виключно для зручності користувачів. Для того щоб команда на кшталт `ls` могла вивести інформацію про власника файлу в зручному для читання вигляді, вона повинна переглянути базу даних ідентифікаторів і знайти в ній потрібні імена.

4.3. Володіння процесом

Власник може посилати процесу сигнали, а також знижувати його пріоритет. Із процесами пов'язано кілька ідентифікаторів: реальний, поточний і збережений ідентифікатор користувача; реальний, поточний і збережений ідентифікатор групи, а в системі Linux – «ідентифікатор користувача файлової системи», що використовується тільки для визначення прав доступу до файлів. Взагалі кажучи, реальні номери застосовуються для обліку використання системних ресурсів, а поточні – для зазначення прав доступу. У більшості випадків реальні та поточні ідентифікатори збігаються.

Збережені ідентифікатори не чинять жодного безпосереднього впливу. Вони дають змогу програмам «приберігати» неактивні ідентифікатори для подальшого використання, тим самим сприяючи розважливому застосуванню розширених повноважень. Взагалі кажучи, ідентифікатор користувача файлової системи – це особливість реалізації мережевої файлової системи (Network File System – NFS), і зазвичай він збігається з поточним ідентифікатором користувача.

4.4. Обліковий запис суперкористувача

Обліковий запис root в UNIX належить «всесильному» користувачеві-адміністратору, відомому як суперкористувач, хоча його справжнє ім'я - «root».

Визначальною характеристикою облікового запису суперкористувача є значення UID, що дорівнює нулю. Ніщо не забороняє змінювати ім'я цього облікового запису або створювати інший запис із нульовим ідентифікатором, але такі дії ні до чого доброго не призведуть. Їхнім наслідком буде виникнення нових проломів у системі захисту, а також розгубленість і гнів інших адміністраторів, яким доведеться розбиратися з особливостями конфігурування такої системи.

Традиційна система UNIX дає змогу суперкористувачеві (тобто кожному процесу, поточний ідентифікатор користувача якого дорівнює нулю) виконувати над файлом або процесом будь-яку допустиму операцію.

Ось приклади операцій, доступних лише суперкористувачу:

- зміна кореневого каталогу процесу за допомогою команди chroot;
- створення файлів пристроїв;
- встановлення системного годинника;
- збільшення лімітів використання ресурсів і підвищення пріоритетів процесів;
- завдання мережевого імені комп'ютера;
- конфігурування мережевих інтерфейсів;
- відкриття привілейованих мережевих портів (номери яких менші за 1024);
- зупинка системи.

Процеси суперкористувача мають здатність змінювати свої ідентифікатори. Один із таких процесів – програма `login` та її графічні еквіваленти, які відображають на екрані запрошення ввести пароль під час входу в систему. Якщо введені пароль та ім'я користувача правильні, програма замінює свої ідентифікатори на відповідні ідентифікатори зазначеного користувача і запускає інтерпретатор команд. Після того як процес суперкористувача, змінивши власника, стане звичайним користувацьким процесом, відновити свій попередній привілейований стан він уже не зможе.

4.5. Використання бітів «setuid» і «setgid»

Традиційна система керування доступом в UNIX доповнена системою зміни повноважень, яка реалізується ядром у співпраці з файловою системою. Якщо коротко, то ця система дає змогу виконувати спеціально підготовлені файли з використанням привілеїв вищого рівня (зазвичай це привілеї суперкористувача). Цей механізм дозволяє розробникам і адміністраторам створювати умови для непривілейованих користувачів, за яких вони можуть виконувати привілейовані операції.

Річ у тім, що існують два спеціальних біти, які встановлюються в масці прав доступу до файлу: «`setuid`» (Set User ID - біт зміни ідентифікатора користувача) і «`setgid`» (Set Group ID - біт зміни ідентифікатора групи). Якщо запускається виконуваний файл, у якого встановлено один із цих бітів, то поточними ідентифікаторами створюваного процесу стають ідентифікатори власника файлу, а не ідентифікатори користувача, який запустив програму. Зміна повноважень дійсна тільки на час роботи програми.

Наприклад, користувачі повинні мати можливість змінювати свої паролі. Але оскільки паролі зберігаються в захищеному файлі `/etc/shadow`, користувачам потрібно використовувати команду `passwd` з повноваженнями `setuid`, щоб «посилити» свої права доступу. Команда `passwd` перевіряє, хто її виконує, і, залежно від результату, налаштовує свою поведінку відповідним чином, тому звичайні користувачі можуть змінювати тільки власні паролі, а суперкористувач – будь-які. (Це, між іншим, ще один приклад «спеціального випадку» в UNIX-системі управління доступом – правила «вбудовані» в код команди `passwd`).

4.6. Сучасна організація управління доступом

Незважаючи на те що систему управління доступом можна описати буквально двома сторінками, вона витримала перевірку часом завдяки своїй простоті, передбачуваності та здатності задовольняти більшість вимог, які висуваються до управління доступом на середньостатистичному вузлі. Усі UNIX- і Linux-версії продовжують підтримувати цю модель, яка широко використовується в наші дні. Проте ми не можемо не згадати очевидні недоліки цієї моделі.

- З погляду безпеки, суперкористувач представляє єдиний обліковий запис, який може бути причиною потенційної відмови системи. Якщо суперкористувач дискредитує себе, може порушитися цілісність усієї системи. Зломщик у цьому разі може завдати системі колосальної шкоди.
- Єдиний спосіб розділити спеціальні привілеї суперкористувача полягає в написанні setuid-програм. На жаль, як показує безперервний інтернет-потік оновлень засобів захисту систем, досить важко написати по-справжньому безпечне програмне забезпечення. До того ж, вам не слід писати програми, призначення яких можна було б висловити такими словами: «Хотілося б, щоб ці три користувачі могли виконувати завдання резервування на файловому сервері».
- Модель безпеки не є достатньо міцною для використання в мережі. Жоден комп'ютер, до якого непривілейований користувач має фізичний доступ, не може гарантувати, що він точно представляє приналежність процесів, що виконуються. Хто може стверджувати, що такий-то користувач не переформатував диск і не інсталював власну копію Windows або Linux зі своїми UID?
- У багатьох середовищах із високими вимогами до захисту системи діють угоди, які просто не можуть бути реалізовані з використанням традиційних UNIX-систем безпеки. Наприклад, згідно з державними стандартами США комп'ютерні системи повинні забороняти привілейованим користувачам (наприклад, тим, хто належить до вищої категорії допуску) публікувати документи особливої важливості на нижчому рівні безпеки. Традиційна ж організація безпеки в UNIX залежить від доброї волі та професійних навичок окремих користувачів.
- Оскільки багато правил, пов'язаних з управлінням доступу, вбудовано в код окремих команд і демонів, неможливо перевизначити поведінку системи, не модифікуючи вихідного коду і не перекомпілюючи програми. Але це реально не практикується.
- Для відстеження дій користувачів передбачено мінімальну підтримку. Ви можете легко дізнатися, до яких груп належить той чи інший користувач, але ви не в змозі точно визначити, які дії дозволені користувачеві членством у цих групах.

Через ці недоліки UNIX- і Linux-системи багаторазово зазнавали різних змін. Їхня мета – удосконалити різні аспекти підсистеми управління доступом і зробити UNIX-системи більш придатними для сайтів із високими вимогами до безпеки. Одні коригувальні технології, як-от PAM, нині мають практично універсальну підтримку. Інші системні розширення можна назвати унікальними.

4.7. Керування доступом на основі ролей

Управління доступом на основі ролей (role-based access control – RBAC) – теоретична модель, формалізована в 1992 р. Девідом Ферраільо (David Ferraiolo) і Ріком Куном (Rick Kuhn). В основі цієї моделі лежить ідея додавання рівня опосередкованості в механізм управління доступом. Замість того щоб призначати повноваження безпосередньо користувачам, їх призначають проміжним конструкціям, іменованим «ролями», а ролі, своєю чергою, призначають користувачам. Для того щоб ухвалити рішення з управління доступом, спеціальна бібліотека перераховує ролі поточного користувача і дізнається, чи має хоча б одна з цих ролей відповідні повноваження.

Між ролями і UNIX-групами можна помітити деяку схожість, і немає єдиної думки щодо того, чи відрізняються ці конструкції по суті. На практиці ролі виявляються більш корисними, ніж групи, оскільки системи, в яких вони реалізовані, дозволяють використовувати їх поза контекстом файлової системи. Ролі можуть також мати ієрархічні взаємини одна з одною, що значно спрощує адміністрування. Наприклад, ви могли б визначити роль «старшого адміністратора», який має всі повноваження «адміністратора», а також додаткові повноваження X, Y і Z.

Модель RBAC дає змогу на практиці реалізувати управління великими колекціями можливих повноважень. Більша частина зусиль витрачається на ієрархічне визначення ролі, але це робиться один раз за проєкт. Повсякденне ж адміністрування користувачів у цьому випадку спрощується. Отже, системи, які підтримують модель RBAC, зазвичай користуються її перевагами для «розщеплення» повноважень суперкористувача на безліч різноманітних фрагментів, які можуть призначатися користувачам окремо.

4.8. SELinux: Linux-системи з поліпшеною безпекою

Операційна система SELinux (як проєкт) була розроблена Агентством національної безпеки США (АНБ), але з кінця 2000 року була передана розробникам відкритого коду. Систему SELinux включено до складу ядра Linux (з версії 2.6) і тому наразі вона доступна в більшості дистрибутивів (але часто в не зовсім дієздатному стані).

SELinux – це реалізація системи примусового управління доступом доступу (mandatory access control – MAC), у якій усі привілеї призначаються адміністраторами. У середовищі MAC користувачі не можуть делегувати свої права і не можуть встановлювати параметри контролю доступу на об'єкти, якими вони (користувачі) володіють. І тому така операційна система підходить більше для вузлів зі спеціальними вимогами.

Систему SELinux можна також використовувати для контролю доступу на основі ролей, хоча це не головна мета системи.

4.9. Модулі аутентифікації, що підключаються

Модулі автентифікації, що підключаються (Pluggable Authentication Modules – PAM), утворюють технологію автентифікації, а не технологію керування доступом. Інакше кажучи, технологія PAM покликана шукати відповідь не на запитання: «Чи має право користувач X виконати операцію Y?», а на запитання: «Як дізнатися, що це дійсно користувач X?». Технологія PAM – це важлива ланка в ланцюгу управління доступом у більшості систем.

Раніше паролі користувачів перевірялися за допомогою вмісту файлу `/etc/shadow` (або його мережевого еквівалента) у момент реєстрації, щоб можна було встановити відповідний UID-ідентифікатор для командної оболонки користувача або віконної системи. У цьому разі програми, що виконуються користувачем, змушені були приймати зазначений UID на віру. У сучасному світі мереж, криптографії та пристроїв біометричної ідентифікації потрібна гнучкіша і відкритіша система. Тому і з'явилася технологія PAM.

Технологія PAM – це набір спільних бібліотек, які дають змогу інтегрувати різні низькорівневі методи аутентифікації. Адміністратори вказують ті методи автентифікації, які (з їхнього погляду) має використовувати система у відповідних контекстах. Програми, які збираються аутентифікувати користувача, просто викликають систему PAM, а не реалізують власні форми автентифікації. Система PAM, у свою чергу, викликає спеціальну бібліотеку автентифікації, задану системним адміністратором.

4.10. Мережевий протокол криптографічної аутентифікації Kerberos

Подібно до PAM, протокол Kerberos покликаний розв'язувати проблеми аутентифікації, а не управління доступом. (Він названий на честь триголового пса, який захищав вхід до царства Аїда – Цербера, або Кербера.) Але якщо PAM можна назвати структурною оболонкою аутентифікації, то Kerberos – це конкретний метод аутентифікації.

Реалізація Kerberos використовує перевірений (сторонній) сервер для виконання завдань аутентифікації в масштабах всієї мережі. Замість самоаутентифікації на своєму комп'ютері ви надаєте свої мандати (квитки) Kerberos-службі, а вона видає вам криптографічні мандати, які ви можете презентувати іншим службам як підтвердження вашої ідентичності.

4.11. Списки керування доступом

Керування доступом до файлової системи – найважливіша частина систем UNIX і Linux, і тому її вдосконалення становить першочергову мету розробників цих систем. Особливу увагу було приділено підтримці списків доступу до файлів і каталогів (access control lists – ACL) як узагальненню традиційної моделі привілеїв користувача/групи/»всіх інших», встановлюваних одразу для кількох користувачів і

груп. Списки ACL є частиною реалізації файлової системи, тому їхню підтримку в явному вигляді має забезпечувати файлова система, яку ви використовуєте. На щастя, нині практично всі файлові системи UNIX і Linux підтримують ACL-списки в тій чи іншій формі.

Підтримка ACL-списків у загальному випадку реалізується у двох формах: у вигляді проєкту стандарту POSIX, який (хоч і без формального підтвердження) здобув широке визнання, і системи, стандартизованої протоколом NFSv4, який базується на ACL-списках Microsoft Windows.

4.12. Управління доступом у реальному світі

Незважаючи на наявність описаних вище чудових можливостей операційних систем, у більшості вузлів для завдань системного адміністрування, як і раніше, використовується обліковий запис суперкористувача. Багато скарг на традиційну систему мають реальне підґрунтя, але й альтернативним варіантам притаманні серйозні проблеми. Тому особливого значення набувають такі додаткові програмні інструменти, як `sudo` (докладніше трохи нижче), які певною мірою дають змогу подолати розрив між критеріями простоти використання і безпеки.

Часто для ухвалення рішень в особливих обставинах використовуються можливості POSIX або засоби керування доступом на основі ролей (наприклад, коли необхідно дозволити перевстановлення принтера або демона кожному, хто працює в конкретному відділі), тоді як під час розв'язання щоденних завдань ваша адміністративна команда продовжує покладатися на утиліту `sudo` і обліковий запис суперкористувача. У деяких випадках (наприклад, якщо це обумовлено в контракті) для вузлів необхідно використовувати такі потужні й «ударостійкі» системи, як SELinux.

Оскільки доступ із правами суперкористувача є обов'язковою умовою системного адміністрування та центральною точкою для безпеки систем, дуже важливо правильно користуватися правами та обов'язками облікового запису `root`.

4.13. Пароль суперкористувача

Якщо ви використовуєте процедури та інструменти, описані в цій лекції, ймовірно, будете здивовані, як насправді використовується пароль суперкористувача. У більшості випадків адміністраторам не потрібно знати його взагалі.

Проте для облікового запису `root` пароль необхідний. Він має бути секретним і водночас таким, що запам'ятовується, щоб ви могли ним скористатися (тобто не забути) навіть через великі інтервали часу. Для «запам'ятовування» паролів можна використовувати який-небудь зберігач паролів або систему депонування паролів.

Найважливішою характеристикою хорошого пароля є його довжина. Пароль користувача `root` має складатися щонайменше з восьми символів, оскільки навіть семисимвольні паролі зламуються досить легко. У системах, де застосовуються паролі DES (Data Encryption Standard – стандарт шифрування даних), задавати більш довгий пароль немає сенсу, тому що обробляються тільки перші вісім символів.

Теоретично найбезпечніший пароль складається з випадкового набору букв, розділових знаків і цифр. Такий пароль, однак, важко запам'ятати і, як правило, незручно вводити, тому, якщо системний адміністратор записує пароль на папірці або

вводить його занадто повільно, про оптимальний рівень безпеки системи говорити не доводиться.

У системах, які підтримують паролі довільної довжини, ви можете використовувати як пароль абсурдну фразу. Можна також перетворити цю фразу на пароль, записавши тільки другі літери кожного слова або застосувавши яке-небудь інше перетворення, що легко запам'ятовується. Безпека пароля значно зросте, якщо включити в нього цифри, розділові знаки і великі літери.

Якщо ваш вузол включає сотні комп'ютерів, то чи потрібно вам готувати сотні root-паролів? Це залежить від середовища і толерантності до ризику, але, найімовірніше, ні. Практика показує, що для комп'ютерів-клонів (наприклад, настільних АРМ) цілком підійде один і той самий root-пароль. При цьому **серверам слід підбирати унікальні паролі. Усі важливі частини мережі та інфраструктура, що забезпечує маршрутизацію, мають бути захищені окремо.**

Постарайтеся точно записати, які комп'ютери спільно використовують root-пароль. Важливо також мати структурований спосіб зміни root-паролів для таких комп'ютерів. Недбалість у цьому питанні безпосередньо пов'язана з великим ризиком порушення безпеки і головним болем адміністраторів.

Пароль суперкористувача слід змінювати в таких випадках:

- мінімум кожні три місяці;
- щоразу, коли будь-хто, хто знає пароль, звільняється з організації;
- коли, на вашу думку, безпеці системи щось загрожує.

Часто кажуть, що паролі ніколи не слід записувати, однак цю думку слід висловити точніше: паролі ніколи не повинні бути доступними для випадкових людей.

Паролі суперкористувача та інші важливі паролі краще записувати в зашифрованому вигляді, щоб адміністратори могли скористатися ними за крайньої потреби.

4.14. Файлова система

Операційна система Linux підтримує велику кількість файлових систем, на сьогодні найбільш широко використовуються: ext2, ext3, ext4, ReiserFS та xfs. Також сучасні ОС Linux сумісні з файловими системами, що використовуються в ОС Windows, такими як NTFS та FAT32, але використання цих файлових систем в Linux небажане, бо ці системи розроблялись під іншу операційну систему і в зв'язку з цим підтримка Windows-розділів в ядрі Linux реалізована за допомогою сторонніх утиліт/драйверів, що накладає деякі обмеження (наприклад, ОС Linux не має можливості розмежовувати права доступу до файлів на розділах NTFS).

4.14.1. Структура файлової системи

В операційній системі Windows при відкритті теки “Мій комп'ютер” користувач звик до наступної картини: зазвичай один або більше твердих дисків (частіш за все логічних) що називаються починаючи з латинської літери С. Кожен з дисків є кореневою текою. Так, наприклад, якщо в системі є три диска, то буде три кореневих теки (скоріш за все C, D та E), кожен з яких містить вкладені теки та файли. Іншими словами в системі буде існувати три дерева. Оскільки час від часу треба

користуватися компакт-дисками та USB-накопичувачами, то періодично будуть "виростати" ще кілька дерев.

В дистрибутивах Linux все дещо інакше. Файлова система цілісна і має лише одну кореневу теку, яка позначається косою рискою – слеш (/). Тут треба звернути увагу на відмінність від Windows. В ній при формуванні повної адреси використовується обернена коса риска "\". В Linux при формуванні повної адреси завжди використовується "/".

Файлова структура Linux має певну структуру тек.

В ній все впорядковано та "лежить" на своєму місці. Кожна тека має власне призначення, яке регламентується документом під назвою FHS (Filesystem Hierarchy Standart – стандарт структури файлової системи).

Таблиця 4.1 – Короткий опис призначення основних тек згідно даного стандарту FHS

Тека	**Опис**
/	Коренева тека, що містить всю файлову ієрархію
/bin/	Основні утиліти, які необхідні як в однокористувацькому режимі, так і при звичайній роботі всім користувачам (наприклад: cat, ls, cp)
/boot/	Завантажувальні файли (в тому числі файли завантажувача, ядро, initrd, System.map). Часто виноситься на окремий розділ.
/dev/	Основні файли пристроїв (наприклад, /dev/null, /dev/zero)
/etc/	Загальносистемні конфігураційні файли (назва походить від *et cetera*).
/etc/opt/	Файли конфігурації для /opt/.
/etc/X11/	Файли конфігурації X Window System версії 11.
/home/	Містить домашні теки користувачів, які, в свою чергу, містять персональні налаштування та дані користувача. Часто виноситься на окремий розділ.
/lib/	Основні бібліотеки, що необхідні для роботи програм з /bin/ и /sbin/
/media/	Точки монтування для змінних носіїв, таких як CD-ROM, DVD-ROM (вперше описано в FHS-2.3)
/mnt/	Містить тимчасово змонтовані файлові системи
/opt/	Додаткове програмне забезпечення
/proc/	Віртуальна файлова система, що надає стан ядра операційної системи та запущених процесів у вигляді файлів
/root/	Домашня тека користувача root
/sbin/	Основні системні програми для адміністрування та налаштування системи, наприклад: init, iptables, ifconfig
/srv/	Дані, що специфічні для оточення системи
/tmp/	Тимчасові файли
/usr/	Вторинна ієрархія для даних користувача; містить більшість програмних додатків користувача та утиліт, що використовуються в багатокористувацькому режимі. Може бути змонтована через мережу в режимі "лише для читання" та бути загальною для декількох комп'ютерів
/usr/bin/	Додаткові програми для всіх користувачів, які не є необхідними в однокористувацькому режимі.

/usr/include/	Стандартні файли заголовків.
/usr/lib/	Бібліотеки для програм, що знаходяться в /usr/bin/ та /usr/sbin/.
/usr/sbin/	Додаткові системні програми (такі як демони різних мережевих сервісів).
/usr/share/	Архітектурно-незалежні загальні дані.
/usr/src/	Вихідні коди (наприклад, тут містяться вихідні коди ядра).
/usr/X11R6/	X Window System, версії 11, реліз 6.
/usr/local/	*Третинна ієрархія* для даних, що специфічні для даного хосту. Зазвичай містить такі підтеки, як bin/, lib/, share/
/var/	Файли, що часто змінюються, наприклад файли реєстрації (log-файли), тимчасові поштові файли, файли спулерів.
/var/lib/	Постійні дані, що змінюються програмами в процесі роботи (наприклад, бази даних, метадані пакетного менеджера та інш.).
/var/lock/	Лок-файли, що вказують на зайнятість деякого ресурсу.
/var/log/	Різні файли реєстрації (log-файли).
/var/mail/	Поштові скриньки користувачів
/var/run/	Інформація про запущені програми (в основному, про демони).
/var/spool/	Завдання, що чекають на обробку (наприклад, черги друку, непрочитані або не відправлені листи).
/var/spool/mail/	Місце для поштових скриньок користувачів (застаріле).
/var/www/	Файли веб-сайтів (наприклад, ієрархія файлів віртуальних хостів).
/var/tmp/	Тимчасові файли, які повинні бути збережені між перезавантаженнями.

4.12. Користувачі та групи в Linux

Операційна система Linux є багатокористувацької системою в якій може одночасно існувати досить велика кількість користувачів.

В Linux робота з користувачами містить набір наступних маніпуляцій: додавання, видалення або модифікація налаштувань користувача або групи.

Крім облікових записів, які використовують люди для роботи з системою, в Linux передбачені облікові записи для системних користувачів: з точки зору системи це такі ж користувачі, однак ці облікові записи використовуються виключно для роботи деяких програм-сервісів. Наприклад, стандартний системний користувач mail використовується програмами доставки пошти.

Коли в системі створюється новий користувач, він додається щонайменше в одну групу. В системі при створенні нового облікового запису створюється спеціальна група, назва якої співпадає з іменем нового користувача і користувач включається в цю групу. В подальшому адміністратор може додати користувача і в інші групи.

Механізм груп може використовуватись для організовування спільного доступу кількох користувачів до певних ресурсів. Наприклад на сервері компанії для кожного проєкту може бути створена окрема група, в яку будуть входити облікові записи співробітників, що працюють над цим проєктом. При цьому файли, що відносяться до проєкту, можуть належати цій групі та бути доступними для її членів. В системі

також існує певна кількість груп (наприклад bin), які використовуються для керування доступом системних програм до різних ресурсів. Як правило членами таких груп є системні користувачі, а користвачі-люди до таких груп не входять.

Також за допомогою груп можуть бути надані права, що є необхідними для виконання певних задач. Наприклад для додавання, видалення або налаштування принтера в системі користувач повинен входити в групу lpadmin.

4.13. Керування користувачами та групами

Створення користувача
```
# adduser username
```
Створення групи
```
# addgroup groupname
```
Додавання користувача в групу
1 спосіб
```
# addgroup username groupname
```
2 спосіб
```
# usermod -aG groupname username
```
Видалення користувача з групи
```
# delgroup username groupname
```
Видалення користувача
```
# deluser username
```

4.14. Права доступу до файлів та тек

Оскільки система Linux з самого початку розроблялась як багатокористувацька система, в ній передбачений такий механізм, як права доступу до файлів та тек. Він дозволяє розмежовувати повноваження користувачів, які працюють в системі. Зокрема, права доступу дозволяють окремим користувачам мати «особисті» файли та теки.

Правильне налаштування прав доступу дозволяє підвищити надійність системи, захищаючи від змін або видалення важливі системні файли. А оскільки зовнішні пристрої з точки зору Linux також являються об'єктами файлової системи, механізм прав доступу можна застосовувати й для керування доступом до пристроїв.

У будь-якого файла у системі є власник – один з користувачів. Однак кожен файл одночасно належить ще й до деякої групи користувачів системи. Кожен користувач може входити в будь-яку кількість груп, і в кожну групу може входити довільна кількість користувачів з числа тих, які визначені в системі.

Права доступу визначаються по відношенню до трьох типів дій: читання, запис та виконання. Ці права можуть бути надані трьом класам користувачів: власнику файла (користувачу), групі, якій належить файл, а також всім іншим користувачам, що не входять в цю групу. Право на читання дає користувачу змогу читати вміст файла або, якщо такий доступ дозволений до теки, продивлятися вміст теки (використовуючи команду `ls`). Право на запис дає можливість користувачу записувати або змінювати файл, а право на запис для теки – можливість створювати або видаляти файли в цій теці. Врешті, право на виконання дозволяє користувачу запускати файл як програму або сценарій командної оболонки (зрозуміло, що ця дія має сенс лише в тому випадку, коли файл є програмою або сценарієм). Володіння правами на виконання для теки дозволяє перейти (командою `cd`) в цю теку.

Візуальне значення прав доступу записується двома способами – в символьному вигляді, або у вигляді вісімкового числа. У символьному вигляді це `r`, `w`, `x` або їх комбінація.

У вісімковому відображенні кожному виду доступу відповідає число

`x` відповідає 1, `w` – 2, а `r` – 4

Комбінація символьних значень в вісімковому вигляді буде виглядати як сума відповідних значень.

Символьний вигляд	Вісімковий вигляд
r	4
w	2
x	1

Так `rwx` – тобто повний доступ у вісімковому вигляді буде 1 + 2 + 4 = 7
`rw-` – доступ на читання та запис – 2 + 4 = 6
а `r-x` – доступ на читання и виконання – 1 + 4 = 5

Повний запис прав доступу до файлу складається з трьох частин: право власника, право групи та право всіх інших та записується відповідно

`rwxr-xr--` – в символьному вигляді або

`754` – в вісімковому вигляді

Перегляд встановлених прав доступу:

```
$ ls -l
```

Зміна прав доступу до файла

1 спосіб – явне вказання всіх прав `chmod 640 ~/test.txt`

2 спосіб – зміна певного права доступу `chmod g+w ~/test.txt`

Аналогічним чином можна змінювати і власника файлу

```
# chown newuser:newgroup ~/test.txt
```

4.15. Атрибути файлів і тек за замовчуванням. umask.

`umask` – функція середовища POSIX, що змінює права доступу, які надаються новим файлам та текам за замовчуванням. `umask` зазвичай встановлюється у вісімковій системі числення.

Фактично, `umask` вказує, які біти треба скинути в правах на файл, що встановлюються – кожен встановлений біт `umask` забороняє встановлення відповідного біту прав.

Для себе слід запам'ятати – якщо ви створюєте файл, то значення маски віднімається від значення 666, а якщо створюєте теку, то значення маски віднімається от значення 777.

Продивитися поточне значення `umask`

```
$ umask
```

За замовчуванням значення `umask` – 22. Це значить, що всі файли створюються з правами 666 – 22 = 644, тобто `rw-r--r--`, або, простіше кажучи, у володаря файла є права на читання та запис, але не на виконання, а у інших (як у групи, так і у всіх інших користувачів) права лише на читання.

Зміна `umask`

```
$ umask 002
```

Тепер при створенні нового файлу він з'явиться з правами 666 - 2 = 664, тобто `rw-rw-r--`.

В Unix-подібних системах програма запускається з правами користувача, який викликав вказаний додаток. Це забезпечує додаткову безпеку, так як процес з правами користувача не зможе отримати доступ на запис до важливих системних файлів, наприклад `/etc/passwd`, який належить суперкористувачу `root`. Якщо на виконуваний файл встановлений біт suid, то при виконанні ця програма автоматично змінює «ефективний userID» на ідентифікатор того користувача, який є власником цього файла. Тобто, в незалежності від того – хто запускає цю програму, вона при виконанні має права власника цього файла.

Встановлення біту `suid` на файли, що не є виконувальними, не має сенсу. Також немає рації встановлювати цей біт на теку. В останніх версіях ядра Linux додане ще одне обмеження – з міркувань безпеки suid-біт може застосовуватись лише до бінарних файлів, але аж ні як щодо скриптів.

Коли груповий s-біт встановлюється для теки, то кожен файл, що буде створено в цій теці буде віднесено до тієї групи, до якої відноситься і сама тека (але не користувач, що створює цей файл). Використання цього біта має сенс лише відносно теки.

Встановлення SGID-біта

```
# chmod g+s ~/testdir
```

Зняття SGID-біта

```
# chmod g-s ~/testdir
```

4.16. Додаткові атрибути файлів

Крім права на читання/запис/виконання файлові системи ext2, ext3 та ext4, які використовуються в Linux підтримують додаткові атрибути файлів.

Необхідно зразу відзначити, що в більшості випадків вам не прийдеться з ними працювати, але знати про те, що вони існують потрібно. Коротко опишемо деякі з них.

A - не оновлювати час доступу до об'єкту. Теоретично встановлення цього атрибуту має підвищити продуктивність файлової системи і, відповідно, системи в цілому.

a - вказує, що в файл можна додавати інформацію, але не можна видаляти.

d - вказує на те, що не треба робити резервні копії файла. Файл буде проігноровано командою `dump`.

i - вказує на те, що файл не можна видаляти та модифікувати.

s - вказує, що при видаленні файла місце де був розміщений файл буде перезаписано нулями.

u - вказує на те, що при видаленні файла його треба кудись зберегти.

Повний список атрибутів можна побачити прочитавши `$man chattr`.

Існують і інші атрибути. Про них ви можете прочитати в довідці. Також треба звернути увагу на те, що не всі атрибути працюють на всіх файлових системах.

Змінювати додаткові атрибути можна командою `chattr`, а для того, щоб їх продивитися існує команда `lsattr`.

4.17. Монтування дисків. Mount та fstab.

Розглянемо процедуру монтування файлових систем. Монтування може здійснюватися автоматично, або вручну.

`mount` – утиліта командного рядка в UNIX-подібних системах. Застосовується для монтування файлових систем.

4.17.1. Ручне монтування локальних ресурсів. Команда mount.

Варіант команди монтування, що зустрічається найчастіше, буде виглядати так:

```
mount -t тип_файлової_системи -o опції_монтування що_монтуємо куди_монтуємо.
```

Наприклад, монтування ntfs-розділу на нашому ж жорсткому диску в теку `/mnt/win` буде

виглядати так:

```
mount -t ntfs-3g -o defaults /dev/sda5 /mnt/win
```

де

`ntfs-3g` – тип файлової системи

`defaults` – всі опції монтування за замовчанням `/dev/sda5` – пристрій, який монтується

`/mnt/win` – тека, в яку буде проведено монтування

Якщо при монтуванні виникли проблеми з кирилицею на ntfs-розділі, в опціях монтування можна явно вказати кодову сторінку:

```
mount -t ntfs-3g -o locale=uk_UA.utf8 /dev/sda5 /mnt/win
```

Найчастіше утиліта mount всі потрібні параметри (в тому числі і тип файлової системи) може визначити сама. У такому випадку команда монтування може виглядати так:

```
mount /dev/sdb1 /mnt/new_disk
```

За допомогою mount можна монтувати не тільки розділи на фізичних пристроях, а й просто файли, наприклад iso-образи дисків. В такому випадку використовується пристрій loop:

```
mount -t iso9660 -o loop /home/user/ubuntu-10.04.3-server-i386.iso /home/ubuntu/
```

4.17.2. Автоматичне монтування. Файл fstab.

Автоматичне монтування файлових систем проводиться за допомогою файлу /etc/fstab. Саме в ньому й описано монтування пристроїв, яке буде здійснено при завантаженні операційної системи.

Якщо ми відкриємо цей файл в Ubuntu, то побачимо приблизно таке:

```
proc /proc proc nodev,noexec,nosuid 0 0
UUID=87c97237-0bf0-4a62-bae2-47a850643996 / ext4 errors=remount-ro 0 1
UUID=0af767a3-c740-47fe-87ad-fb495bc1444e /home ext4 defaults 0 2
UUID=45bc8486-ed86-4a86-96db-ea34e4c0b9da none swap sw 0 0
```

Кожен запис в цьому файлі містить 6 полів. Його формат простий:

`Пристрій – Точка монтування – Тип файлової системи – Опції монтування – вказування необхідності резервного копіювання – Порядок перевірки розділу.`

Пристрій може визначатися не тільки через UUID, а й через мітку тома, або просто як файл пристрою, наприклад `/dev/sda1`.

Точку монтування вибираємо самі по необхідності. Вона не обов'язково повинна знаходитися безпосередньо в кореневому розділі. Наприклад можемо на поштовому сервері окремий розділ примонтувати як `/var/mail`.

Тип файлової системи – тут все зрозуміло. В який формат відформатували, те і пишемо.

Опції монтування – тут можна дуже багато чого описати. Наприклад: монтувати пристрій тільки для читання, або в режимі читання і запису, кодування імен файлів, чи зберігати час доступу до файлів, логін-пароль для доступу (якщо це мережевий ресурс) і так далі.

Ознака необхідності резервного копіювання – зазвичай не використовується і встановлюється в 0.

Порядок перевірки розділу (0 – не перевіряти, 1 – встановлюється для кореня, 2 – для решти розділів).

Тепер деякі подробиці про опції монтування. Всі опції записуються через кому.

async – всі дії введення/виводу будуть проводитися асинхронно

noatime – не оновлювати час доступу до файлів

defaults – використовувати опції за замовчуванням: `rw, suid, dev, exec, auto, nouser` і `async`.

exec – дозволяти виконання бінарних файлів

noexec – не дозволяти виконання бінарних файлів

suid – дозволяти використовувати біти `set-user-identifier` і `set-group-identifier`.

nosuid – не дозволяти використовувати біти `set-user-identifier` і `set-group-identifier`.

ro – монтувати пристрій в режимі тільки для читання

rw – монтувати пристрій в режимі читання/запису

users – дозволяти всім користувачам монтувати і демонтувати цей пристрій

Крім цього, варто звернути увагу, що при монтуванні різних файлових систем, опції монтування можуть змінюватися. Особливо це відноситься до мережевих файлових систем, таким як `sshfs, curlftpfs, smbfs` і їм подібним.

4.18. Система Syslog і журнальні файли

Системні адміністратори, та й звичайні користувачі Linux, часто повинні дивитися лог-файли для усунення несправностей. Насправді це перше, що повинен зробити будь-який сисадмін при виникненні будь-якої помилки в системі.

Сама операційна система Linux та працюючі програми генерують різні типи повідомлень, які реєструються в різних файлах журналів. У Linux використовуються спеціальне програмне забезпечення, файли та директорії для зберігання лог файлів. Знання в яких файлах є логи яких програм допоможе вам заощадити час і швидше вирішити проблему.

Syslog – це повнофункціональна система реєстрації подій, написана Еріком Оллманом (Eric Allman). Вона виконує дві важливі функції: звільняє програмістів від стомлюючої механічної роботи з ведення журнальних файлів і передає управління журнальною реєстрацією в руки адміністраторів. До появи системи Syslog кожна програма сама обирала схему реєстрації подій, а системні адміністратори не мали змоги контролювати, яка інформація і де саме зберігається.

Система Syslog вирізняється високою гнучкістю. Вона дає змогу сортувати повідомлення за джерелами і рівнем важливості та направляти їх у різні пункти призначення: у журнальні файли, на термінали користувачів і навіть на інші комп'ютери. Однією з найцінніших особливостей цієї системи є її здатність централізувати процедуру реєстрації подій у мережі.

Більшість файлів логів Linux знаходяться в папці /var/log/. Ви можете переглянути список файлів логів для вашої системи за допомогою команди ls:

```
Rw-r--r-- 1 root root 52198 трав 10 11:03 alternatives.log
drwxr-x--- 2 root root 4096 лист 14 15:07 apache2
drwxr-xr-x 2 root root 4096 кві 25 12:31 apparmor
drwx------ 2 root root 4096 трав 5 10:15 audit
-rw-r--r-- 1 root root 33100 трав 10 10:33 boot.log
```

Нижче ми розглянемо 20 різних файлів логів Linux, розміщених у каталозі /var/log/. Деякі з цих логів зустрічаються лише у певних дистрибутивах, наприклад, dpkg.log зустрічається лише у системах, заснованих на Debian.

/var/log/messages – Містить глобальні системні логи Linux, у тому числі ті, які реєструються при запуску системи. У цей лог записуються кілька типів повідомлень: це пошта, cron, різні сервіси, ядро, автентифікація та інші.

/var/log/dmesg – Містить повідомлення, отримані від ядра. Реєструє багато повідомлень ще на етапі завантаження, в них відображається інформація про апаратні пристрої, які ініціалізуються під час завантаження. Можна сказати, це ще один лог системи Linux. Кількість повідомлень у лозі обмежена, і коли файл буде переповнений, з кожним новим повідомленням старі будуть перезаписані. Ви також можете переглянути повідомлення з цього лога за допомогою команди dmseg.

/var/log/auth.log – містить інформацію про авторизацію користувачів у системі, включаючи логіни користувача та механізми автентифікації, які були використані.

/var/log/boot.log – містить інформацію, що реєструється під час завантаження системи.

/var/log/daemon.log – включає повідомлення від різних фонових демонів

/var/log/kern.log – також містить повідомлення від ядра, корисні при усуненні помилок модулів, вбудованих в ядро.

/var/log/lastlog – відображає інформацію про останню сесію всіх користувачів. Це нетекстовий файл, для перегляду необхідно використовувати команду lastlog.

/var/log/maillog /var/log/mail.log – журнали сервера електронної пошти, запущеного у системі.

/var/log/user.log – інформація зі всіх журналів на рівні користувачів.

/var/log/Xorg.x.log – лог повідомлень X сервера.

/var/log/alternatives.log – інформація про роботу програми update-alternatives. Це символічні посилання на команди або бібліотеки за промовчанням.

/var/log/btmp – лог-файл Linux містить інформацію про невдалі спроби входу. Для перегляду файлу зручно використовувати команду last -f /var/log/btmp

/var/log/cups – всі повідомлення, пов'язані з друком та принтерами.

/var/log/anaconda.log – всі повідомлення, зареєстровані під час встановлення, зберігаються в цьому файлі

/var/log/yum.log – реєструє всю інформацію про встановлення пакетів за допомогою Yum.

/var/log/cron – щоразу, коли демон Cron запускає виконання програми, він записує звіт і повідомлення самої програми в цьому файлі.

/var/log/secure – містить інформацію, що стосується аутентифікації та авторизації. Наприклад, SSHd реєструє тут все, у тому числі невдалі спроби входу до системи.

/var/log/wtmp або /var/log/utmp – системні логи Linux **,** містять журнал входів користувачів у систему. За допомогою команди `wtmp` ви можете дізнатися, хто і коли увійшов до системи.

/var/log/faillog – лог системи linux містить невдалі спроби входу в систему. Використовуйте faillog, щоб відобразити вміст цього файлу.

/var/log/mysqld.log – файли логів Linux від сервера баз даних MySQL.

/var/log/httpd/ або **/var/log/apache2** – лог-файли linux веб-сервера Apache. Логи доступу знаходяться у файлі access_log, а помилок у error_log

/var/log/lighttpd/ – лог-файли linux веб-сервера lighttpd

/var/log/conman/ – файли логів клієнта ConMan,

/var/log/mail/ – цей каталог містить додаткові логи поштового сервера

/var/log/prelink/ – програма Prelink пов'язує бібліотеки та виконувані файли, щоб прискорити процес завантаження. /var/log/prelink/prelink.log містить інформацію про файли, які були змінені програмою.

/var/log/audit/ – містить інформацію, створену демоном аудиту auditd.

/var/log/setroubleshoot/ – SE Linux використовує демон setroubleshootd (SE Trouble Shoot Daemon) для сповіщення про проблеми з безпекою. Цей журнал містить повідомлення цієї програми.

/var/log/samba/– містить інформацію та журнали файлового сервера Samba, який використовується для підключення до спільних папок Windows.

/var/log/sa/– містить .cap файли, зібрані пакетом Sysstat.

/var/log/sssd – використовується системним демоном безпеки, який керує віддаленим доступом до каталогів та механізмів аутентифікації.

Перегляд логів у Linux

Щоб подивитися логи на Linux зручно використовувати кілька утиліт командного рядка Linux. Це може бути будь-який текстовий редактор, або спеціальна утиліта. Швидше за все, вам знадобляться права суперкористувача для того, щоб подивитися логи в Linux. Ось команди, які найчастіше використовуються для цих цілей:

- zgrep
- zmore

Перегляд логів Linux виконується дуже просто:
Дивимося лог /var/log/messages, з можливістю прокручування:

```
less /var/log/messages
```

Перегляд логів Linux у реальному часі:

```
tail -f /var/log/messages
```

Відкриваємо лог файл dmesg:

```
cat /var/log/dmesg
```

Перші рядки dmesg:

```
head /var/log/dmesg
```

Виводимо лише помилки з /var/log/messages:

```
grep -i error /var/log/messages
```

Крім того, подивитися логи на linux можна і за допомогою графічних утиліт. Програма **System Log Viewer** може бути використана для зручного перегляду та відстеження системних журналів на ноутбуці або персональному комп'ютері з Linux.

Ви можете встановити програму в будь-якій системі із встановленим X сервером. Також для перегляду логів може використовуватися будь-який графічний тестовий редактор.

ЛЕКЦІЯ 5. Мережі TCP/IP

5.1. Стандарти формування кадрів Ethernet

Важко переоцінити важливість мереж для сучасних комп'ютерів, хоча, здається, це не заважає людям намагатися це зробити. У багатьох місцях, можливо навіть у більшості, доступ до Інтернету та електронної пошти є основними способами використання комп'ютерів. Станом на 2017 рік, за оцінками internetworldstats.com, інтернет налічував понад 3,7 мільярда користувачів, або трохи менше половини населення світу. У Північній Америці рівень проникнення Інтернету наближається до 90%.

TCP/IP (Transmission Control Protocol/Internet Protocol) – це мережева система, яка лежить в основі Інтернету. TCP/IP не залежить від конкретного апаратного забезпечення або операційної системи, тому пристрої, які використовують TCP/IP, можуть обмінюватися даними («взаємодіяти»), незважаючи на їхні численні відмінності.

TCP/IP працює в мережах будь-якого розміру і топології, незалежно від того, підключені вони до зовнішнього світу чи ні.

TCP/IP – це «стек» протоколів, набір мережевих протоколів, розроблених для безперебійної спільної роботи. Він включає кілька компонентів, кожен з яких визначається стандартним документом RFC або серією RFC:

- **IP**, Інтернет-протокол, який маршрутизує пакети даних від одного комп'ютера до іншого (RFC791);
- **ICMP**, протокол керуючих повідомлень Інтернету, який визначає різні види низькорівневої підтримки IP, включаючи повідомлення про помилки, допомогу в маршрутизації та налагодженні (RFC792);
- **ARP**, протокол визначення адрес, який перетворює IP-адреси на апаратні адреси (RFC826);
- **UDP**, протокол користувацьких дейтаграм, який реалізує неперевірену односторонню доставку даних (RFC768);
- **TCP**, протокол управління передачею, який реалізує надійні, повнодуплексні, керовані потоком, з корекцією помилок передачі даних (RFC793).

Ці протоколи розташовані в ієрархії або «стеку», де протоколи вищих рівнів використовують протоколи нижчих рівнів. TCP/IP традиційно описується як п'ятирівнева система (рис. 5.1), але фактично протоколи TCP/IP займають лише три з цих рівнів.

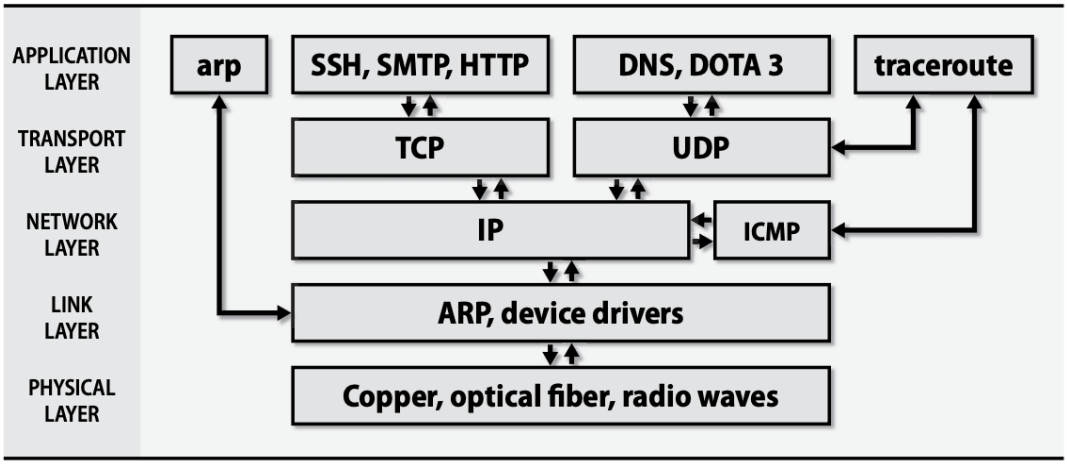


Рисунок 5.1 – Модель рівнів TCP/IP

5.1.1. Пакети та інкапсуляція

TCP/IP підтримує різноманітні фізичні мережі та транспортні системи, зокрема Ethernet, Token Ring, MPLS (багатопротокольна комутація за мітками), бездротовий Ethernet та системи на основі послідовних ліній. Управління апаратним забезпеченням здійснюється на канальному рівні архітектури TCP/IP, а протоколи вищих рівнів не знають і не цікавляться конкретним апаратним забезпеченням, що використовується.

Дані переміщуються мережею у вигляді пакетів, згустків даних з максимальною довжиною, встановленою канальним рівнем. Кожен пакет складається із заголовка та корисного навантаження. Заголовок повідомляє, звідки прийшов пакет і куди він прямує. Він також може містити контрольні суми, специфічну для протоколу інформацію або інші інструкції з обробки.

Корисне навантаження – це дані, які потрібно передати.

Назва примітивної одиниці даних залежить від рівня протоколу. На канальному рівні вона називається кадром, на IP-рівні – пакетом, а на TCP-рівні – сегментом. У цій лекції ми використовуємо «пакет» як загальний термін, який охоплює всі ці різні випадки.

Коли пакет рухається вниз стеком протоколів (від транспортного TCP або UDP до IP, Ethernet і фізичного дроту), готуючись до відправлення, кожен протокол додає власну інформацію до заголовка. Готовий пакет кожного протоколу стає частиною корисного навантаження в пакеті, згенерованому наступним протоколом. Таке вкладання називається інкапсуляцією. На приймаючій машині інкапсуляція змінюється на протилежну, коли пакет рухається назад по стеку протоколів.

Наприклад, UDP-пакет, що передається через Ethernet, містить три різні обгортки або конверти. На лінії Ethernet він обрамляється простим заголовком, який містить апаратні адреси джерела та наступного вузла призначення, довжину кадру та контрольну суму кадру (CRC). Корисним навантаженням кадру Ethernet є IP-пакет, корисним навантаженням IP-пакету є UDP-пакет, а корисним навантаженням UDP-пакету є дані, що передаються. На рис. 5.2 показані компоненти такого кадру.

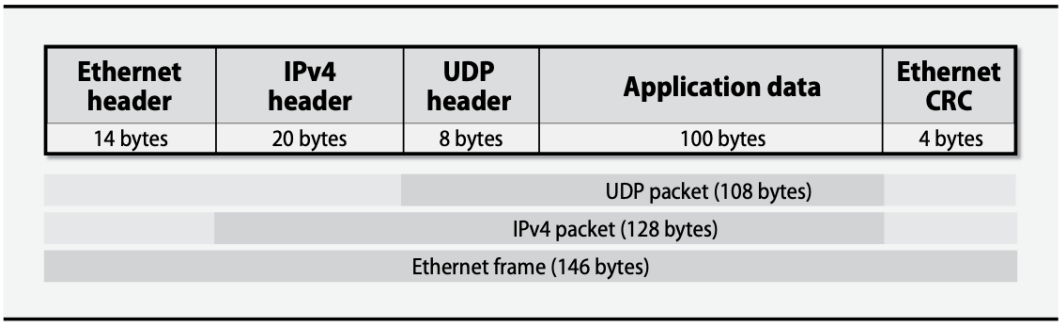


Рисунок 5.2 – Типовий мережевий пакет

5.1.2. Кадрування Ethernet

Одне з основних завдань канального рівня – додавання заголовків до пакетів і розділювачів між ними. Заголовки містять інформацію про адресацію на канальному рівні та контрольні суми кожного пакета, а роздільники гарантують, що приймачі можуть визначити, де закінчується один пакет і починається наступний. Процес додавання цих додаткових бітів відомий як кадрування.

Канальний рівень поділяється на дві частини: MAC, підрівень керування доступом до середовища, і LLC, підрівень керування логічним з'єднанням. Підрівень MAC працює з середовищем передачі даних і передає пакети по кабелю. Підрівень LLC відповідає за формування кадрів.

На сьогоднішній день існує єдиний стандарт для кадрування Ethernet: DIX Ethernet II. У далекому минулому також використовувалося кілька дещо відмінних стандартів, заснованих на IEEE 802.2. Ви все ще можете натрапити на рудиментарні посилання на вибір кадру в мережевій документації, але зараз ви можете ігнорувати цю проблему.

5.2 Адресація пакетів

Подібно до листів і повідомлень електронної пошти, мережеві пакети можуть досягти пункту призначення тільки за наявності правильної адреси. У системі TCP/IP використовується поєднання кількох схем адресації.

- Адреси MAC (media access control) для використання в мережевому обладнанні.
- Мережеві адреси протоколів IPv4 і IPv6 для використання в програмному забезпеченні.
- Імена комп'ютерів для використання користувачами.

5.2.1. Апаратна адресація (MAC)

Кожен мережевий інтерфейс комп'ютера має одну MAC-адресу канального рівня, яка відрізняє його від інших комп'ютерів у фізичній мережі, а також одну або кілька IP-адрес, що ідентифікують інтерфейс у глобальній мережі Інтернету. Останнє твердження варто повторити: IP-адреса ідентифікує мережеві інтерфейси, а не машини. (Для користувачів ця відмінність не має значення, але адміністратори повинні про це знати.)

Найнижчий рівень адресації задається мережевими апаратними засобами. Наприклад, Ethernet-пристроям у процесі виготовлення призначаються унікальні шестибайтові апаратні адреси. Ці адреси традиційно записуються у вигляді низки двоцифрових шістнадцяткових байтів, розділених двокрапками, наприклад 00:50:8D:9A:3B:DF.

Мережеві плати Token Ring також мають шестибайтові адреси. У деяких мережах із двоточковим з'єднанням (наприклад, у PPP-мережах) апаратні адреси взагалі не потрібні: адресу пункту призначення вказують безпосередньо під час установлення з'єднання.

Шестибайтова Ethernet-адреса розбивається на дві частини: перші три байти визначають виробника пристрою, а останні три – виступають як унікальний серійний номер, який призначає виробник. Системні адміністратори можуть з'ясувати марку пристрою, що спричиняє проблеми в мережі, пошукавши трибайтовий ідентифікатор відповідних пакетів у таблиці ідентифікаторів виробників. Поточна таблиця доступна за адресою

`http://www.iana.org/assignments/ethernet-numbers`

Трибайтові коди насправді являють собою ідентифікатори OUI (Organizationally Unique Identifier – унікальний ідентифікатор організації), що присвоюються організацією IEEE, тому їх можна знайти безпосередньо в базі даних IEEE за адресою

`http://standards.ieee.org/regauth/oui`

Зрозуміло, відносини між виробниками мікросхем, компонентів і систем мають складний характер, тому ідентифікатор виробника, закодований у MAC-адресі, може ввести користувача в оману.

Теоретично, апаратні адреси Ethernet повинні призначатися на постійній основі і залишатися незмінними. На жаль, деякі мережеві плати допускають програмне налаштування апаратних адрес. Це зручно під час заміни зіпсованих комп'ютерів або мережевих карт, MAC-адресу яких змінювати з тих чи інших причин небажано (наприклад, якщо її фільтрують усі ваші комутатори, якщо ваш DHCP-сервер видає адреси на основі MAC-адрес або MAC-адресу було використано як ліцензійний ключ для програмного забезпечення). Фальсифіковані MAC-адреси можуть також виявитися корисними, якщо вам необхідно проникнути в бездротову мережу, що використовує механізм управління доступом на основі MAC-адрес. Однак, щоб не ускладнювати ситуацію, рекомендується зберігати унікальність MAC-адрес.

5.2.2. IP-адресація

На наступному, вищому, рівні використовується Інтернет-адресація (частіше її називають IP-адресацією). IP-адреси глобально унікальні й апаратно незалежні.

Відповідність між IP-адресами та апаратними адресами встановлюється на канальному рівні моделі TCP/IP. У мережах, що підтримують широкомовний режим (тобто в мережах, що дають змогу адресувати пакети «усім комп'ютерам цього фізичного сегмента»), протокол ARP забезпечує автоматичну прив'язку адрес без втручання системного адміністратора. У протоколі IPv6 MAC-адреси інтерфейсів можна використовувати як частину IP-адрес, завдяки чому перетворення IP-адрес в апаратні адреси стає практично автоматичним.

5.2.3. «Адресація» імен машин

Оскільки IP-адреси являють собою довгі, на перший погляд, випадкові числа, запам'ятовувати їх важко. Операційні системи дають змогу закріплювати за IP-адресою одне або кілька текстових імен, щоб замість 128.9.160.27 користувач міг ввести «rfc-editor.org». У системах UNIX і Linux це відображення можна здійснити за допомогою статичного файлу (/etc/hosts), бази даних LDAP і, нарешті, DNS (Domain Name System) – глобальної системи доменних імен. Слід пам'ятати про те, що ім'я комп'ютера – це лише скорочений спосіб запису IP-адреси, і він відноситься до мережевого інтерфейсу, а не комп'ютера.

5.2.4. Порти

IP-адреси ідентифікують мережеві інтерфейси комп'ютера, але вони недостатньо конкретні для ідентифікації окремих процесів і служб, багато з яких можуть активно використовуватися в мережі одночасно. Протоколи TCP і UDP розширюють концепцію IP-адрес, вводячи поняття порту. Порт являє собою 16-розрядне число, що додається до IP-адреси і вказує на конкретний канал взаємодії. Усім стандартним службам, зокрема електронній пошті, FTP і HTTP, призначаються «добре відомі» порти, які визначено у файлі /etc/services. Для того щоб запобігти спробам сторонніх процесів замаскуватися під стандартні служби, системи UNIX надають доступ до портів із номерами до 1024 тільки процесам користувача root. (Взаємодіяти із сервером через порти з невеликими номерами може будь-хто; обмеження поширюється лише на програми, що прослуховують цей порт).

Типи адрес

У протоколі IP підтримується кілька типів адрес, деякі з яких мають еквіваленти на канальному рівні.

- Спрямовані (unicast) - адреси, які позначають окремий мережевий інтерфейс.
- Групові (multicast) - адреси, що ідентифікують групу вузлів.
- Широкомовні (broadcast) - адреси, що позначають усі вузли локальної мережі.
- Альтернативні (anycast) - адреси, що позначають будь-який із групи вузлів.

Режим групового мовлення використовується, наприклад, у відеоконференціях, де одна й та сама послідовність пакетів надсилається всім учасникам конференції. Протокол IGMP (Internet Group Management Protocol – протокол управління групами вузлів Інтернету) відповідає за управління сукупностями вузлів, що ідентифікуються як один узагальнений адресат.

Групові адреси в даний час в Інтернеті практично не використовуються. Проте вони були використані в протоколі IPv6, в якому широкомовні адреси, по суті, являють собою спеціалізовану форму групової адресації.

Альтернативні адреси забезпечують балансування навантаження на канальний рівень мережі, дозволяючи доставляти пакети в найближчий із кількох пунктів призначення в сенсі мережевої маршрутизації. Можна було очікувати, що ці адреси будуть нагадувати групові, але фактично вони більше схожі на спрямовані адреси.

Більшість деталей механізму альтернативних адрес приховано на рівні маршрутизації, а не на рівні протоколу IP. Завдяки альтернативній адресації відбулося реальне послаблення традиційних вимог, щоб IP-адреси однозначно ідентифікували пункт призначення. З формальної точки зору, альтернативна реалізація призначена

для протоколу IPv6, але аналогічний трюк можна здійснити і в протоколі IPv4, наприклад, так, як це зроблено для кореневих серверів імен DNS.

5.3. Приватні адреси та система NAT

Тимчасове розв'язання проблеми скорочення адресного простору протоколу IPv4 полягає у використанні приватних областей IP-адрес, описаних у документі RFC1918. Приватні адреси використовуються тільки всередині сайту і ніколи не демонструються в Інтернеті (принаймні ненавмисно). Перетворення приватних адрес на адреси, виділені інтернет-провайдером, здійснюється пограничним маршрутизатором.

Документ RFC1918 визначає, що одна мережа класу A, 16 мереж класу B і 256 мереж класу C резервуються для приватного використання і ніколи не виділяються глобально. У табл. 5.1 показано діапазони приватних адрес (кожен діапазон представлено в коротшій нотації протоколу CIDR). Із перелічених діапазонів організації можуть вибирати для себе мережі потрібного розміру.

Таблиця 5.1 – IP-адреси, зарезервовані для приватного використання

IP class	From	To	CIDR range
Class A	10.0.0.0	10.255.255.255	10.0.0.0/8
Class B	172.16.0.0	172.31.255.255	172.16.0.0/12
Class C	192.168.0.0	192.168.255.255	192.168.0.0/16

Спочатку ідея полягала в тому, щоб вузли могли самостійно вибирати клас адрес із зазначених можливостей, щоб правильно визначити свій розмір. Однак наразі протокол CIDR і підмережі стали універсальним інструментом, тому для всіх нових приватних мереж найдоцільніше використовувати адреси класу A (звісно, з підмережами).

Для того щоб вузли, які використовують приватні адреси, могли отримувати доступ до Інтернету, на прикордонному маршрутизаторі організації має виконуватися система NAT (Network Address Translation – трансляція мережевих адрес). Ця система перехоплює пакети і замінює в них адресу відправника реальною зовнішньою IP-адресою. Може також відбуватися заміна номера вихідного порту. Система зберігає таблицю перетворень між внутрішніми і зовнішніми адресами/портами, щоб відповідні пакети надходили потрібному адресату.

Завдяки нумерації портів з'являється можливість під'єднати кілька вихідних з'єднань до загальної IP-адреси, щоб внутрішні вузли спільно використовували однакову зовнішню IP-адресу. Іноді в організації достатньо мати одну «справжню» зовнішню адресу. Наприклад, цю конфігурацію за замовчуванням встановлено в більшості популярних маршрутизаторів, що використовують кабель і модеми DSL.

Вузол, що використовує систему NAT, запитує адреси у провайдера, але більшість адрес тепер використовують для внутрішньосистемних прив'язок і не призначають окремим комп'ютерам. Якщо вузол пізніше захоче змінити провайдера, буде потрібно лише змінити конфігурацію прикордонного маршрутизатора і його системи NAT, але не самих комп'ютерів.

Існує можливість змусити операційні системи UNIX або Linux виконувати функції NAT, хоча в багатьох організаціях воліють делегувати ці обов'язки маршрутизаторам або пристроям підключення до мережі.

Неправильна конфігурація системи NAT може призвести до того, що пакети з приватними адресами почнуть проникати в Інтернет. Вони можуть досягти вузла призначення, але відповідні пакети не будуть отримані. Організація CAIDA, що заміряє трафік магістральних мереж, повідомляє про те, що 0,1-0,2% пакетів, що проходять магістраллю, мають або приватні адреси, або неправильні контрольні суми. На перший погляд показник здається незначним, але насправді це призводить до того, що кожної хвилини в мережі циркулюють мільйони зайвих пакетів. Інформацію щодо статистики Інтернету та засобів вимірювання продуктивності глобальної мережі можна отримати на вебвузлі www.caida.org.

Однією з особливостей системи NAT є те, що вузол Інтернету не може на пряму підключитися до внутрішніх машин організації. Для того щоб подолати це обмеження, у деяких реалізаціях системи NAT дозволяється створювати тунелі, які підтримують прямі з'єднання з обраними вузлами.

Ще одна проблема полягає в тому, що деякі додатки вбудовують IP-адреси в інформаційну частину пакетів. Такі додатки не можуть нормально працювати спільно з NAT. До них належать певні протоколи маршрутизації, програми потокової доставки даних і низка FTP-команд. Система NAT іноді також відключає віртуальні приватні мережі (VPN – virtual private network).

Система NAT приховує внутрішню структуру мережі. Це може здатися перевагою з погляду безпеки, але фахівці вважають, що насправді система NAT не забезпечує належну безпеку і вже в усякому разі не усуває потребу в брандмауері. Крім того, вона перешкоджає спробам оцінити розміри і топологію Інтернету.

5.4. Питання безпеки

Темі безпеки буде присвячено лекцію 8, але низка питань, що стосуються IP-мереж, заслуговує окремої згадки. У цій лекції розглянемо мережеві механізми, які традиційно спричиняють проблеми безпеки, і опишемо шляхи розв'язання цих проблем.

5.4.1. Перенаправлення IP-пакетів

Якщо в UNIX- або Linux-системі дозволено перенаправлення IP-пакетів, то комп'ютер може виступати як маршрутизатор. Інакше кажучи, він може приймати пакети від третьої сторони, що надходять на його мережевий інтерфейс, порівнювати їх зі шлюзом або пунктом призначення і передавати мережею.

Якщо ваша система не має кілька мережевих інтерфейсів і не призначена для функціонування в ролі маршрутизатора, цю функцію рекомендується відключити. Вузли, що перенаправляють пакети, часто виявляються залученими в атаки на системи безпеки, будучи змушеними видавати зовнішні пакети за свої власні. Цей трюк дає змогу пакетам зловмисників обходити мережеві сканери та фільтри.

Найкраще, щоб вузол використовував кілька мережевих інтерфейсів для свого власного трафіку і не пересилав сторонні пакети.

5.4.2. Директиви переадресації протоколу ICMP

За допомогою переадресовуючих ICMP-пакетів можна зловмисно змінювати напрямок трафіку і редагувати таблиці маршрутизації. Більшість операційних систем за замовчуванням приймає ці пакети і дотримується інструкцій, що містяться в них. Але, погодьтеся, навряд чи можна назвати прийнятною ситуацію, коли на кілька годин весь трафік організації переспрямовують конкуренту, особливо якщо в цей час виконують резервне копіювання. Ми рекомендуємо так конфігурувати маршрутизатори (або системи, що відіграють роль маршрутизаторів), щоб переадресовуючі ICMP-пакети ігнорувалися і, можливо, фіксувалися в журнальному файлі.

5.4.3. Маршрутизація «від джерела»

Механізм маршрутизації «від джерела» в протоколі IP дає змогу відправнику явно вказати послідовність шлюзів, через які має пройти пакет на шляху до одержувача. При цьому відключається алгоритм пошуку наступного переходу, що виконується на кожному шлюзі для визначення того, куди потрібно послати пакет.

Маршрутизація «від джерела» була частиною вихідної специфікації протоколу IP і слугувала для цілей тестування. Але вона створює проблему з точки зору безпеки, адже пакети часто фільтруються залежно від того, звідки вони прибули. Зловмисник може так підібрати маршрут, що пакет здаватиметься таким, що прибув із внутрішньої мережі, а не з Інтернету, тому брандмауер його пропустить. Рекомендується не приймати і не перенаправляти доставлені в такий спосіб пакети.

5.4.4. Широкомовні ICMP-пакети та інші види спрямованих широкомовних повідомлень

Пакети команди ping, що несуть у собі широкомовну адресу мережі (а не конкретного вузла), зазвичай доставляються всім вузлам мережі. Такі пакети застосовуються в атаках типу «відмова від обслуговування», наприклад, у так званих атаках «smurf» (за назвою програми, у якій їх уперше було застосовано).

Широкомовні ICMP-пакети є «спрямованими» в тому сенсі, що вони надсилаються за широкомовною адресою конкретної віддаленої мережі. Стандартні підходи до обробки таких пакетів поступово змінюються. Наприклад, в операційній системі Cisco IOS 11.x і більш ранніх версій за замовчуванням здійснювалася переадресація спрямованих широкомовних пакетів, але у версіях 12.0 і вище цього вже немає. Зазвичай можна налаштувати стек TCP/IP так, щоб широкомовні пакети, які надходять з інших мереж, ігнорувалися, але, оскільки це потрібно зробити для кожного мережевого інтерфейсу, подібне завдання є досить нетривіальним у великій організації.

5.4.5. Підміна IP-адрес

Початкова адреса IP-пакета зазвичай заповнюється функціями мережевих бібліотек і являє собою адресу вузла, з якого було надіслано пакет. Але якщо програма, що створює пакет, працює з низькорівневим IP-сокетом, вона може підставити будь-яку вихідну адресу, яку забажає. Це називається підміною IP-адрес і зазвичай асоціюється зі спробами злому мережі. У цій схемі жертвою часто стає комп'ютер, ідентифікований за підробленою IP-адресою (якщо він дійсно існує).

Повідомлення про помилки і відповідні пакети можуть переповнити і вивести з ладу мережу жертви.

Підроблені IP-адреси слід забороняти на прикордонному маршрутизаторі, блокуючи надіслані пакети, вихідні адреси яких не перебувають у підконтрольній діапазоні. Це особливо важливо в університетських мережах, де студенти люблять експериментувати і часто подібним чином зривають свою злість.

Водночас, якщо в локальній мережі використовуються адреси з приватного діапазону, то фільтрації можуть піддаватися пакети з приватними адресами, які намагаються «проскочити» в Інтернет. Відповідь на такі пакети ніколи не буде отримано через відсутність відповідних маршрутів у магістральній мережі. Їхня поява свідчить про те, що в системі є внутрішня помилка конфігурації.

Потрібно також захищатися від хакерів, які підробляють вихідні адреси зовнішніх пакетів, унаслідок чого брандмауер починає вважати, ніби вони надійшли з локальної мережі. Допомогти цьому може евристичний метод, названий як «одноадресна передача зворотним шляхом» (unicast reverse path forwarding – uRPF). У цьому разі IP-шлюзи відкидатимуть усі пакети, що задовольняють такій умові: інтерфейс, через який прийшов пакет, не збігається з тим інтерфейсом, через який пакет піде, якщо його вихідна адреса дорівнюватиме цільовій адресі. Для перевірки джерела мережевих пакетів цей метод використовує звичайну таблицю IP-маршрутизації. Метод uRPF застосовується не тільки в спеціалізованих маршрутизаторах, а й у ядрі системи Linux, у якій цей режим увімкнено за замовчуванням.

Якщо ваш сайт має кілька виходів в Інтернет, доцільно розділити зовнішні маршрути на вихідні та вхідні. У цьому разі слід вимкнути режим uRPF, щоб протокол маршрутизації працював правильно. Якщо ж вихід в Інтернет тільки один, безпечніше ввімкнути режим uRFF.

5.4.6. Вбудовані брандмауери

Як правило, з'єднання вашої локальної мережі із зовнішнім світом і управління трафіком відповідно до правил сайту, здійснюють мережеві фільтри пакетів або брандмауери (firewalls). На жаль, компанія Microsoft перекрутила уявлення про те, як має працювати брандмауер, на прикладі своїх систем Windows, сумнозвісних своєю вразливістю. Кілька останніх випусків системи Windows містили свої власні брандмауери і «голосно обурювалися», коли користувач намагався їх відключити.

Усі системи, які ми використовуємо як приклади, містять програмне забезпечення для фільтрації пакетів, але звідси не випливає, що кожній UNIX- або Linux-машині потрібен окремий брандмауер. Це не так. Механізми фільтрації пакетів, вбудовані в ці системи, дають змогу цим машинам виконувати функції мережевих шлюзів.

Однак не рекомендується використовувати робочу станцію як брандмауер. Навіть найдосконаліша операційна система занадто складна, щоб бути надійною. Спеціальне програмне забезпечення більш передбачуване і більш надійне, навіть якщо воно таємно використовує систему Linux.

Навіть складне програмне забезпечення, пропоноване, наприклад, компанією Check Point (чиї програми виконуються на вузлах під управлінням операційних

систем UNIX, Linux і Windows), поступається в надійності міжмережевим екранам серії Adaptive Security Appliance компанії Cisco, хоча має майже таку саму ціну.

5.4.7. Віртуальні приватні мережі

Багатьом організаціям, які мають офіси в різних частинах світу, хотілося б, щоб усі ці офіси були з'єднані однією великою приватною мережею. На жаль, вартість оренди трансконтинентальних і навіть транснаціональних ліній зв'язку робить це нереальним. Таким організаціям доводиться використовувати Інтернет як «приватний» канал, організовуючи серію захищених, зашифрованих «тунелів» між офісами. Мережеві конгломерати такого роду називаються віртуальними приватними мережами (virtual private network – VPN).

Можливості віртуальних приватних мереж необхідні також співробітникам, які повинні з'єднуватися з офісом з дому або перебувають у поїздці. Система VPN не розв'язує всіх питань, пов'язаних із безпекою і цим спеціальним з'єднанням, але в багатьох відношеннях вона цілком безпечна.

У низці приватних мереж використовується протокол IPsec, який у 1998 році був стандартизований організацією IETF як низькорівневий додаток до протоколу IP. В інших мережах, таких як OpenVPN, система безпеки VPN реалізується на основі протоколу TCP за допомогою криптографічного протоколу Transport Layer Security (TSL), який раніше був відомий під назвою Secure Sockets Layer (SSL).

Існує маса запатентованих реалізацій систем VPN. Ці системи не взаємодіють ні одна з одною, ні зі стандартизованими системами VPN, але це не можна вважати недоліком, якщо всі кінцеві точки мережі перебувають під контролем.

З цієї точки зору системи VPN, засновані на протоколі TLS, є лідерами. Вони також безпечні, як і протокол IPsec, але набагато простіші. Вільна реалізація у вигляді OpenVN також не завдає жодної шкоди. (На жаль, ця система поки що не працює під управлінням операційних систем HP-UX і AIX.)

Для підтримки користувачів, які працюють удома або в поїздці, стало загальноприйнятим використання невеликого компонента Java або ActiveX, завантажуваного через їхні веббраузери. Ці компоненти встановлюють з'єднання VPN з мережею підприємства. Цей механізм зручний для користувачів, але слід врахувати, що браузери сильно відрізняються один від одного: деякі з них реалізують службу VPN за допомогою псевдомережевого інтерфейсу, а інші використовують тільки спеціальні порти. Утім, веб-браузери останньої категорії набагато нечисленніші, ніж знамениті веб-проксі.

Слід перевірити, що ви правильно розумієте технологію, на якій ґрунтуються рішення, що застосовуються. Справжня служба VPN (тобто повноцінне IP-з'єднання через мережевий інтерфейс) вимагає наявності адміністративних привілеїв та інсталяції програмного забезпечення на клієнтській машині, незалежно від того, яку операційну систему на ній встановлено: Windows або UNIX. Крім того, слід перевірити сумісність браузера, оскільки механізм, застосовуваний під час реалізації систем VPN за допомогою одного браузера, часто не підтримується іншим.

ЛЕКЦІЯ 6. Маршрутизація.

6.1. Основні протоколи маршрутизації

Управління потоками даних – непросте завдання. У попередній лекції коротко розповідалося про перенаправлення IP-пакетів. У цій лекції ми докладніше вивчимо цей процес і познайомимося з різними мережевими протоколами, завдяки яким маршрутизатори автоматично знаходять найефективніші маршрути. Ці протоколи беруть на себе основне навантаження з управління таблицями маршрутизації, істотно спрощуючи завдання адміністраторам. Вони також дають змогу швидко перенаправляти трафік у разі виходу з ладу маршрутизатора або мережевого сегмента.

Важливо відрізняти реальне переспрямування IP-пакетів від управління таблицею маршрутизації, хоча в обох випадках вживають один і той самий термін – «маршрутизація». Перенаправлення пакетів – проста процедура, тоді як обчислення маршрутів відбувається за досить заплутаним алгоритмом. Ми познайомимося тільки з одноадресною маршрутизацією, тому що за багатоадресної (коли пакети спрямовують групі підписувачів) виникає низка нових проблем, розглянути які, з огляду на обсяг лекції, не видається можливим.

Якщо мережева інфраструктура вже налагоджена, можна просто задати єдиний статичний маршрут, і все готово – у вас є інформація про весь Інтернет. Якщо ж ви прагнете впоратися з мережею, що має складну топологію, або використовуєте операційні системи UNIX чи Linux як частину своєї мережевої інфраструктури, тоді ця лекція про протоколи та інструменти динамічної маршрутизації буде корисною для вас.

В основі маршрутизації лежить принцип знаходження «наступного переходу». У будь-якій конкретній точці потрібно визначити тільки наступний вузол або маршрутизатор, куди буде відправлено пакет на шляху до пункту призначення. Це докорінно відрізняється від багатьох старих протоколів, де перед відправленням пакета в мережу потрібно було задати точний його маршрут (принцип маршрутизації «від джерела»).

6.1.1. Детальніше про маршрутизацію пакетів

Перш ніж приступати до вивчення алгоритмів маршрутизації, розглянемо детальніше, як використовуються таблиці маршрутизації. Припустимо, мережа має топологію, зображену на рис. 6.1.

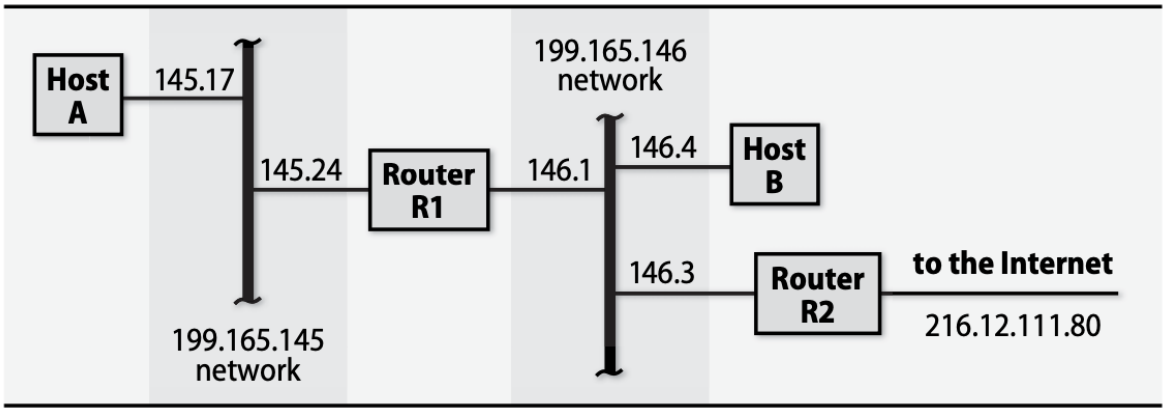


Рисунок 6.1 – Демонстрційна мережа

Маршрутизатор R1 з'єднує дві мережі, а маршрутизатор R2 – одну з цих мереж із зовнішнім світом. Поки будемо вважати, що R1 і R2 – універсальні комп'ютери, а не спеціалізовані маршрутизатори. (У всіх наведених прикладах передбачається використання операційної системи Linux і протоколу IPv4, але в системі UNIX і в протоколі IPv6 використовуються схожі команди і принципи). Подивимося, який вигляд мають таблиці маршрутизації та деякі конкретні сценарії перенаправлення пакетів. Таблиця вузла A має такий вигляд.

```
A% netstat -rn
Kernel IP routing table
Destination     Gateway         Genmask         Flags MSS Window irtt Iface
199.165.145.0   0.0.0.0         255.255.255.0   U     0   0       0    eth0
127.0.0.0       0.0.0.0         255.0.0.0       U     0   0       0      lo
0.0.0.0         199.165.145.24  0.0.0.0         UG    0   0       0    eth0
```

Вузол A має найпростішу конфігурацію серед усіх чотирьох комп'ютерів. Перші два маршрути описують власні мережеві інтерфейси вузла. Вони необхідні, щоб пакети, які спрямовуються безпосередньо в під'єднані мережі, не маршрутизувалися особливим чином. Пристрій eth0 – це Ethernet-інтерфейс вузла A, а lo – інтерфейс зворотного зв'язку (віртуальний мережевий інтерфейс, емульований ядром). Зазвичай такі записи автоматично додаються командою ifconfig під час конфігурування мережевого інтерфейсу.

Деякі системи інтерпретують «маршрут зворотного зв'язку» як вузловий марш рут до конкретної IP-адреси, а не до всієї мережі. Оскільки 127.0.0.0.1 - це єдина IP-адреса, що існує в мережі зворотного зв'язку, по суті, неважливо, як саме її визначено. Єдині зміни, які можна помітити в таблиці маршрутизації, полягають у тому, що в стовпчику `Destination` замість адреси 127.0.0.0 замість адреси 127.0.0.0 буде стояти адреса 127.0.0.1, а в стовпчику Flags – буква H.

Між вузловим і мережевим маршрутами немає принципової різниці. Коли ядро операційної системи переглядає таблицю маршрутизації, вони інтерпретуються абсолютно однаково; єдина відмінність полягає в тому, що вони мають різну довжину неявної маски. Стандартний маршрут вузла A задає переспрямування всіх пакетів, не адресованих інтерфейсу зворотного зв'язку або мережі 195.165.145, на маршрутизатор R1, адреса якого в даній мережі – 199.165.145.24. Прапор G вказує на те, що даний маршрут веде до шлюза, а не до одного з локальних інтерфейсів вузла A. Шлюзи повинні знаходитися на відстані одного переходу.

Припустимо тепер, що вузол A надсилає пакет вузлу B з адресою 199.165.146.4. IP-підсистема шукає маршрут до мережі 199.165.146 і, не знайшовши такого, відправляє пакет за стандартним маршрутом, тобто маршрутизатору R1. На рис. 6.2 наведено структуру пакета, що проходить мережею Ethernet (у заголовку Ethernet зазначено MAC-адреси відповідних інтерфейсів комп'ютерів A і R1 у мережі 145).

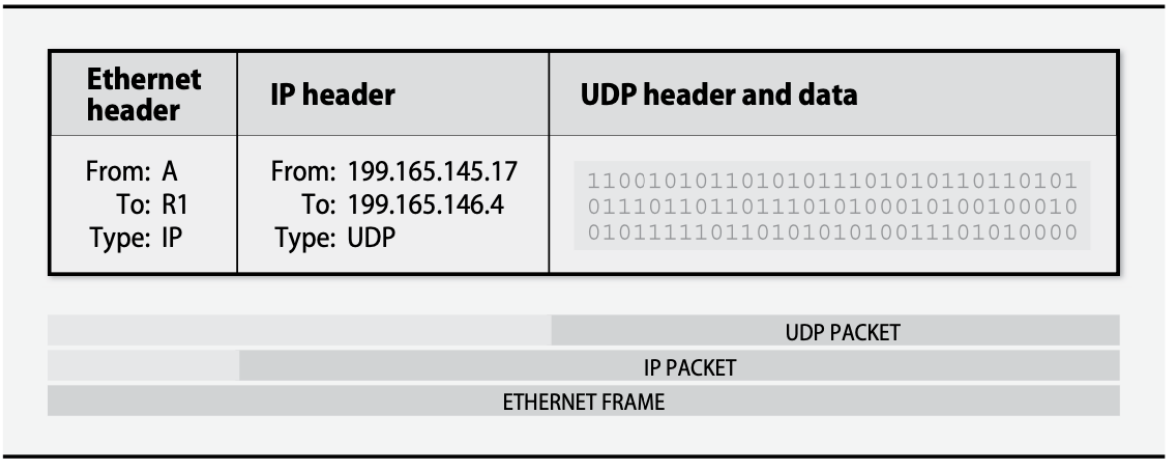


Рисунок 6.2 – Пакет Ethernet

Цільова апаратна Ethernet-адреса відповідає маршрутизатору R1, але IP-пакет, прихований в Ethernet-фреймі, не містить жодних згадок про маршрутизатор. Коли маршрутизатор переглядає пакет, що надійшов, він виявляє, що не є його одержувачем. Тоді він звертається до власної таблиці маршрутизації, щоб дізнатися, як переслати пакет вузлу B, не переписуючи його IP-заголовок (необхідно, щоб відправником пакета залишався вузол A).
Таблиця маршрутизації вузла R1 має такий вигляд.

```
R1% netstat -rn
Kernel IP routing table
Destination     Gateway         Genmask         Flags  MSS Window  irtt  Iface
127.0.0.0       0.0.0.0         255.0.0.0       U        0 0          0  lo
199.165.145.0   0.0.0.0         255.255.255.0   U        0 0          0  eth0
199.165.146.0   0.0.0.0         255.255.255.0   U        0 0          0  eth1
0.0.0.0         199.165.146.3   0.0.0.0         UG       0 0          0  eth1
```

Вона майже аналогічна таблиці вузла A, за винятком того, що тут присутні два фізичні мережеві інтерфейси. Стандартний маршрут у цьому випадку веде до вузла R2, оскільки через нього здійснюється вихід в Інтернет. Пакети, адресовані будь-якій із мереж 199.165, доставляються безпосередньо.
Подібно до вузла A, вузол B має один реальний мережевий інтерфейс. Але для коректного функціонування йому необхідний додатковий маршрут, оскільки вузол має пряме з'єднання відразу з двома маршрутизаторами. Трафік мережі 199.165.145 має проходити через вузол R1, а весь інший трафік – спрямовуватися в Інтернет через вузол R2.

```
B% netstat -rn
Kernel IP routing table
Destination     Gateway         Genmask         Flags MSS Window irtt   Iface
127.0.0.0       0.0.0.0         255.0.0.0       U       0 0          0   lo
199.165.145.0   199.165.146.1   255.255.255.0   U       0 0          0   eth0
199.165.146.0   0.0.0.0         255.255.255.0   U       0 0          0   eth0
0.0.0.0         199.165.146.3   0.0.0.0         UG      0 0          0   eth0
```

Теоретично, ви можете налаштувати хост B, маючи початкові знання лише про один шлюз і покладаючись на допомогу перенаправлень ICMP для усунення зайвих переходів. Наприклад, ось одна з можливих початкових конфігурацій для хоста B:

```
B% netstat -rn
Kernel IP routing table
Destination     Gateway         Genmask         Flags MSS Window irtt    Iface
127.0.0.0       0.0.0.0         255.0.0.0       U       0 0          0    lo
199.165.146.0   0.0.0.0         255.255.255.0   U       0 0          0    eth0
0.0.0.0         199.165.146.3   0.0.0.0         UG      0 0          0    eth0
```

Тепер, якщо вузол B надсилає пакет вузлу A (199.165.145.17), прямий маршрут знайдено не буде, і пакет відправиться на вузол R2. Оскільки вузол R2 є маршрутизатором, він має повну інформацію про стан мережі і, отже, «знає» про роль вузла R1, куди і буде надіслано пакет. Крім того, маршрутизатор R2 виявить, що вузли R1 і B перебувають в одній мережі, тому він додатково надішле вузлу B ICMP-повідомлення, відповідно до якого вузол B додасть у свою таблицю маршрутизації прямий маршрут до вузла A.

```
199.165.145.17 199.165.146.1 255.255.255.255 UGHD 0 0 0 eth0
```

Завдяки цьому маршруту весь трафік, адресований вузлу A, почне йти безпосередньо через маршрутизатор R1. Однак ця зміна не зачіпає трафік до інших вузлів у мережі 145. Для них потрібно отримати окремі директиви від маршрутизатора R2.

Деякі адміністратори обирають для своїх систем директиви переадресації протоколу ICMP як основний «протокол» маршрутизації, думаючи, що такий підхід забезпечить більшу динамічність. На жаль, системи і маршрутизатори здійснюють перенаправлення по-різному. Деякі зберігають ці директиви постійно. Інші видаляють їх із таблиці маршрутизації через відносно короткий час (5-15 хвилин). Треті взагалі ігнорують їх, що, ймовірно, правильно з погляду безпеки.

6.1.2. Демони і протоколи маршрутизації

У найпростіших мережах, подібно до тієї, яка представлена на рис. 6.1, доцільно налаштовувати маршрутизацію вручну. Однак з певного моменту мережі стають занадто складними для такого адміністрування. Замість того щоб явно розповідати кожному комп'ютеру, як знаходити інші комп'ютери і мережі, краще змусити самі комп'ютери шукати цю інформацію. Це завдання покладається на протоколи маршрутизації та демони, які їх реалізують

Основна перевага протоколів маршрутизації над системами статичної маршрутизації полягає в тому, що вони дають змогу швидко адаптуватися до змін у топології мережі. Коли пропадає канал зв'язку, демони швидко знаходять альтернативні маршрути, якщо вони існують, і повідомляють про них у мережі, пов'язані з цим каналом.

Демони маршрутизації збирають інформацію з трьох джерел: конфігураційних файлів, наявних таблиць маршрутизації та «споріднених» демонів інших систем. Зібрані дані об'єднуються, і обчислюють оптимальний набір маршрутів, після чого нові маршрути записують у системну таблицю (і, за потреби, надсилають іншим системам за допомогою протоколів маршрутизації). Стан мережі час від часу

змінюється, тому демони мають періодично опитувати один одного, щоб переконатися в актуальності наявної в них інформації.

Конкретний алгоритм обчислення маршрутів залежить від протоколів. Останні бувають двох типів: дистанційно-векторні та топологічні.

6.1.3. Дистанційно-векторні протоколи

В основі дистанційно-векторних протоколів лежить така ідея: якщо маршрутизатор X розташований за п'ять переходів від мережі Y і є моїм сусідом, то я знаходжусь за шість переходів від цієї мережі. Демон, що працює за таким протоколом, оголошує про те, як далеко, за його розрахунками, розташовані відомі йому мережі. Якщо сусідні демони не «знають» коротших маршрутів до цих мереж, вони позначають цей комп'ютер як оптимальний шлюз. В іншому разі вони просто ігнорують цей анонс. Передбачається, що з часом таблиці маршрутизації прийдуть у стабільний стан.

Це досить гарна ідея, і якби все працювало так, як задумано, маршрутизація істотно спростилася б. На жаль, описаний алгоритм не найкращим чином справляється зі змінами топології. Іноді стабілізація таблиць взагалі не настає внаслідок виникнення нескінченних циклів (наприклад, маршрутизатор X отримує інформацію від маршрутизатора Y і надсилає її маршрутизаторові Z, який повертає її маршрутизатору Y). На практиці доводиться вводити складні евристичні правила або задавати обмеження. Приміром, у протоколі RIP (Routing Information Protocol – протокол маршрутної інформації) вважають, що будь-яка мережа, що перебуває на відстані понад п'ятнадцять переходів, недоступна.

Навіть у звичайних ситуаціях може знадобитися занадто багато циклів оновлень, перш ніж усі маршрутизатори перейдуть у стабільний стан. Отже, щоб не перевищити допустимі межі, необхідно зробити періоди оновлень короткими, а це, своєю чергою, призводить до того, що дистанційно-векторні протоколи виявляються занадто «балакучими». Наприклад, протокол RIP вимагає, щоб маршрутизатори здійснювали широкомовну розсилку всієї наявної у них інформації кожні 30 секунд. У протоколі EIGRP оновлення анонсуються кожні 90 секунд.

На противагу цьому в протоколі BGP (Border Gateway Protocol – протокол прикордонного шлюзу) вся таблиця надсилається один раз, після чого зміни розпространюються в міру виникнення. Така оптимізація дає змогу істотно знизити «переговорний» трафік (здебільшого непотрібний).

Name	Long name	Application
RIP	Routing Information Protocol	Internal LANs (if that)
RIPng	Routing Information Protocol, next generation	IPv6 LANs
EIGRP [a]	Enhanced Interior Gateway Routing Protocol	WANs, corporate LANs
BGP	Border Gateway Protocol	Internet backbone routing

Рисунок 6.3 – Поширені дистанційно-векторні протоколи маршрутизації

6.1.4. Топологічні протоколи

У топологічних протоколах, або протоколах стану каналу, інформація розсилається у відносно необробленому вигляді. Записи виглядають приблизно так: «Маршрутизатор X є суміжним щодо маршрутизатора Y, і канал функціонує». Повний набір таких записів утворює карту мережевих зв'язків, на підставі якої кожен маршрутизатор може сформувати власну таблицю. Основна перевага топологічних протоколів, порівняно з дистанційно-векторними, полягає у здатності швидко стабілізувати таблиці маршрутизації в разі непередбаченого збою. До недоліків недоліків належить необхідність зберігання повної карти з'єднань на кожному вузлі, для чого потрібні додаткові процесорні потужності та пам'ять.

Оскільки процес спілкування між маршрутизаторами не є частиною власне алгоритму обчислення маршрутів, з'являється можливість реалізувати топологічні протоколи так, щоб не виникало циклів. Зміни в топологічній базі даних поширюються мережею дуже швидко, не перевантажуючи ні канал, ні центральний процесор.

Топологічні протоколи складніші за дистанційно-векторні, зате вони дають змогу реалізувати такі технології, як маршрутизація на підставі запитуваного типу обслуговування (поле TOS IP-пакета) і підтримка кількох маршрутів до одного адресата.

Поширення набули тільки два топологічних протоколи: OSPF і IS-IS. Незважаючи на те що протокол IS-IS було реалізовано в широких масштабах, він не так часто використовується, й не рекомендується для використання в нових мережах.

6.1.5. Метрики вартості

Для того щоб протокол маршрутизації міг визначити, який шлях до заданої мережі є найкоротшим, необхідно насамперед з'ясувати, що означає «найкоротший». Це шлях із найменшим числом переходів? З найменшою затримкою? З найменшими фінансовими витратами?

Для цілей маршрутизації якість каналу зв'язку визначається числом, званим метрикою вартості. Шляхом додавання метрик окремих відрізків шляху обчислюється загальна вартість маршруту. У найпростіших системах кожному каналу призначається вартість 1, і в результаті метрикою маршруту стає число переходів. Але будь-який з перерахованих вище критеріїв може бути метрикою вартості.

Експерти в галузі мереж довго і наполегливо працювали над тим, щоб визначення такого поняття, як метрика вартості, було максимально гнучким, а деякі сучасні протоколи навіть дають змогу використовувати різні метрики для різних видів мережевого трафіку. Проте в 99% випадків на все це можна не звертати уваги. Для більшості систем цілком підійдуть стандартні метрики.

Бувають ситуації, коли найкоротший фізичний маршрут до адресата не повинен бути обраний за замовчуванням з адміністративних міркувань. У таких випадках можна штучно завищити метрики критичних каналів. Всю іншу роботу надайте демонам.

6.1.6. Внутрішні та зовнішні протоколи

Автономна система – це група мереж, що перебувають під адміністративним або політичним контролем однієї юридичної особи. Таке визначення досить

розпливчасте. Реальні автономні системи можуть являти собою як глобальні корпоративні мережі, так і мережі університетів або навіть окремих факультетів. Усе залежить від того, як здійснюється маршрутизація. Зараз спостерігається тенденція до укрупнення автономних систем. Це спрощує адміністрування і підвищує ефективність маршрутизації.

Маршрутизація всередині автономної системи відрізняється від маршрутизації між такими системами. Протоколи другого типу (зовнішні, або протоколи зовнішніх шлюзів) повинні керувати безліччю маршрутів до різних мереж і враховувати той факт, що сусідні маршрутизатори перебувають під контролем інших людей. Зовнішні протоколи не розкривають топологію автономної системи, тому в певному сенсі їх можна розглядати як другий рівень маршрутизації, на якому з'єднуються групи мереж, а не окремі комп'ютери або кабелі.

На практиці невеликим і середнього розміру організаціям рідко потрібен зовнішній протокол, якщо тільки вони не підключені до кількох провайдерів одночасно. За наявності декількох провайдерів традиційний поділ на локальний домен і домен Інтернету порушується, оскільки маршрутизаторам доводиться визначати, який маршрут в Інтернеті найкраще підходить для конкретної адреси. (Це не означає, що кожен маршрутизатор має знати всю необхідну інформацію. Більшість вузлів може спрямовувати свої пакети внутрішньому шлюзу, який зберігає необхідні відомості).

6.1.7. Протоколи RIP і RIPng

RIP (Routing Information Protocol – протокол маршрутної інформації) – це старий протокол компанії Xerox, адаптований для IP-мереж. Його IP-версію було описано приблизно 1988 року в документі RFC1058. Існує три версії цього протоколу: RIPv1, RIPv2 і RIPng тільки для протоколу IPv6 (ng (next generation) означає «наступне покоління»).

Усі версії цього протоколу являють собою прості дистанційно-векторні протоколи, метрикою вартості в яких є кількість переходів. Оскільки протокол RIP розробляли в ті часи, коли окремі комп'ютери були дорогими, а мережі – маленькими, у версії RIPv1 припускають, що всі вузли, які знаходяться на відстані п'ятнадцяти і більше переходів, недоступні. У пізніших версіях це обмеження не було знято, щоб стимулювати адміністраторів складних мереж переходити на складніші протоколи маршрутизації.

Протокол RIPv2 – це поліпшена версія протоколу RIP, у якій разом з адресою наступного переходу передається мережева маска. Це спрощує управління мережами, де є підмережі і застосовується протокол CIDR, порівняно з протоколом RIPv1. У ньому було також зроблено невиразну спробу посилити безпеку протоколу RIP.

Протокол RIPv2 можна виконувати в режимі сумісності. Це дає змогу зберегти більшість його нових функціональних можливостей, не відмовляючись від одержувачів, які використовують простий протокол RIP. У багатьох аспектах протокол RIPv2 ідентичний вихідному протоколу і віддавати перевагу слід саме йому.

Протокол RIPng являє собою переформулювання протоколу RIP у термінах протоколу IPv6. Він може використовуватися тільки в рамках протоколу IPv6, тоді як протокол RIPv2 - тільки в рамках протоколу IPv4. Якщо ви хочете використовувати

як протокол IPv4, так і протокол IPv6 разом із протоколом RIP, то RIP і RIPng необхідно виконувати як окремі протоколи.

Незважаючи на те що протокол RIP відомий своїм марнотратним використанням широкомовного режиму, він вельми ефективний при частих змінах мережі, а також у тих випадках, коли топологія віддалених мереж невідома. Однак після збою каналу він може уповільнити стабілізацію системи.

Спочатку дослідники були впевнені, що поява складніших протоколів маршрутизації, таких як OSPF, зробить протокол RIP застарілим. Проте протокол RIP продовжує використовуватися, тому що він простий, легкий у реалізації і не вимагає складного конфігурування. Таким чином, чутки про смерть протоколу RIP виявилися занадто перебільшеними.

Протокол RIP широко використовується на платформах, що не використовують операційну систему UNIX. Багато пристроїв, включно з мережевими принтерами та мережевими керованими SNMP-компонентами, здатні приймати RIP-повідомлення, дізнаючись про можливі мережеві шлюзи. Крім того, майже у всіх версіях систем UNIX і Linux у тій чи іншій формі існує клієнт протоколу RIP. Таким чином, протокол RIP вважається «найменшим спільним знаменником» протоколів маршрутизації. Як правило, його застосовують для маршрутизації в межах локальної мережі, тоді як глобальну маршрутизацію здійснюють потужніші протоколи.

6.1.8. Протокол OSPF

OSPF (Shortest Path First Open – відкритий протокол першочергового виявлення найкоротших маршрутів) є найпопулярнішим топологічним протоколом. Термін «першочергове виявлення найкоротших маршрутів» (shortest path first) означає спеціальний математичний алгоритм, за яким обчислюються маршрути; термін «відкритий» (open) – синонім слова «непатентований». Основна версія протоколу OSPF (версія 2) визначена в документі RFC2328, а розширена версія протоколу OSPF, що підтримує протокол IPv6 (версія 3), – в документі RFC5340. Перша версія протоколу OSPF застаріла і зараз не використовується.

OSPF – протокол промислового рівня, який ефективно функціонує у великих мережах зі складною топологією. Порівняно з протоколом RIP, він має низку переваг, включно з можливістю керування кількома маршрутами, що ведуть до одного адресата, і можливістю поділу мережі на сегменти («області»), які ділитимуться між собою тільки високорівневими даними маршрутизації. Сам про токол дуже складний, тому має сенс використовувати його тільки у великих системах, де важлива ефективність маршрутизації. Для ефективного використання протоколу OSPF необхідно, щоб ваша схема адресації мала ієрархічний характер.

У специфікації протоколу OSPF не нав'язується конкретна метрика вартості. За замовчуванням у реалізації цього протоколу компанією Cisco як метрика використовують пропускну здатність мережі.

6.1.9. Протокол EIGRP

EIGRP (Enhanced Interior Gateway Routing Protocol – протокол маршрутизації внутрішніх шлюзів) – це запатентований протокол маршрутизації, який використовують тільки маршрутизатори компанії Cisco. Протокол IGRP було розроблено для усунення деяких недоліків протоколу RIP ще в ті часи, коли не було

такого надійного стандарту, як протокол OSPF. Протокол IGRP було відхилено на користь протоколу EIGRP, який допускає довільні мережеві маски CIDR. Протоколи IGRP і EIGPR конфігуруються однаково, незважаючи на відмінності в їхній організації.

Протокол EIGRP підтримує протокол IPv6, але в ньому, як і у всіх інших протоколах маршрутизації, адресні простори IPv6 і IPv4 конфігурують окремо та існують як паралельні домени маршрутизації.

Протокол EIGRP є дистанційно-векторним, але його спроєктовано так, щоб уникнути проблем зациклення і повільної стабілізації, властивих іншим протоколам цього класу. У цьому сенсі протокол EIGRP вважається зразком. Для більшості застосувань протоколи EIGRP і OSPF забезпечують рівні функціональні можливості.

6.1.10. IS-IS: протокол маршрутизації між проміжними системами

Протокол IS-IS (Intra-domain Intermediate System to Intermediate System Routing Pro tocol) є відповіддю на протокол OSPF з боку організації ISO. Спочатку він призначався для маршрутизації в рамках мережевих протоколів OSI, але згодом його було розширено для підтримки IP-маршрутизації.

Обидва протоколи – IS-IS і OSPF – створювали на початку 90-х років, коли протоколи організації ISO навмисно зберігали в таємниці. Завдяки зусиллям з боку організації IETF, протокол IS-IS отримав видимість законності, але з часом став дедалі сильніше поступатися в популярності протоколу OSPF і сьогодні використовується рідко. Зараз через безліч непотрібних функціональних особливостей, закладених у нього організацією ISO, краще його уникати.

6.1.11. Протоколи RDP і NDP

Протокол RDP (Router Discovery Protocol – протокол виявлення маршрутизаторів) використовує ICMP-повідомлення, що надсилаються по груповій IP-адресі 224.0.0.1, для отримання інформації про інші маршрутизатори в мережі. На жаль, не всі маршрутизатори наразі розсилають такі повідомлення, і не всі комп'ютери можуть їх приймати. Залишається сподіватися, що коли-небудь цей протокол стане більш популярним.

Протокол NDP (Neighbor Discovery Protocol – протокол виявлення сусіднього вузла), заснований на протоколі IPv6, об'єднує функціональні можливості протоколів RDP і ARP (Address Resolution Protocol – протокол визначення адреси), які використовують для відображення адрес IPv4 в адреси апаратних пристроїв у локальних мережах. Оскільки цей протокол є основним компонентом протоколу IPv6, він використовується там, де використовується протокол IPv6, і протоколи маршрутизації в рамках протоколу IPv6 засновані саме на ньому.

6.1.12. Протокол BGP

Протокол BGP (Border Gateway Protocol – протокол пограничної маршрутизації) є протоколом зовнішньої маршрутизації, тобто він керує трафіком між автономними системами, а не між окремими мережами. Існувало кілька популярних протоколів зовнішньої маршрутизації, але протокол BGP пережив їх усіх.

BGP зараз є стандартним протоколом, що використовується для магістральної маршрутизації в Інтернеті. Станом на середину 2017 року таблиця маршрутизації в

Інтернеті містить близько 660 000 префіксів. З цього числа має бути зрозуміло, що вимоги до масштабування магістральної маршрутизації дуже відрізняються від вимог до локальної маршрутизації.

6.2. Вибір стратегії маршрутизації

Існує чотири рівні складності, що характеризують процес управління маршрутизацією в мережі:

- відсутність маршрутизації як такої;
- тільки статична маршрутизація;
- переважно статична маршрутизація, але клієнти приймають RIP- оновлення;
- динамічна маршрутизація.

Загальна топологія мережі істотно впливає на маршрутизацію кожного конкретного сегмента. У різних мережах можуть вимагатися абсолютно різні рівні підтримки маршрутизації. Під час вибору стратегії слід керуватися переліченими нижче емпіричними правилами.

- Автономна мережа не потребує маршрутизації.
- Якщо з мережі є тільки один вихід у зовнішній світ, клієнти мережі (нешлюзові вузли) повинні мати стандартний статичний маршрут до цього шлюзу. Жодної іншої конфігурації не потрібно, крім як на самому шлюзі.
- Можна задати явний статичний маршрут до шлюзу, що веде в невелику групу мереж, і стандартний маршрут до шлюзу, що веде в зовнішній світ. Однак динамічна маршрутизація краща, якщо до потрібної мережі можна дістатися різними маршрутами.
- Якщо мережі перетинають державні або адміністративні кордони, слід використовувати динамічну маршрутизацію, навіть якщо складність мереж цього не вимагає.
- Протокол RIP відмінно працює і широко використовується. Не відмовляйтеся від нього просто тому, що він має репутацію застарілого протоколу.
- Проблема протоколу RIP полягає в тому, що його неможливо масштабувати до нескінченності. Розширення мережі рано чи пізно призведе до відмови від нього. Через це протокол RIP вважається проміжним протоколом із вузькою областю застосування. Ця область обмежена, з одного боку, мережами, які є занадто простими, щоб застосовувати в них будь-який протокол маршрутизації, а з іншого боку - мережами, які є занадто складними для протоколу RIP. Якщо ви плануєте розширювати свою мережу, то було б доцільно ігнорувати протокол RIP взагалі.
- Навіть якщо протокол RIP не відповідає вашій стратегії глобальної маршрутизації, він залишається хорошим способом розподілу маршрутів до кінцевих вузлів. Однак його не слід застосовувати без особливої потреби: системи в мережі, що має тільки один шлюз, ніколи не потребують динамічного оновлення.
- Протоколи EIGRP і OSPF мають однакові функціональні можливості, але протокол EIGRP є власністю компанії Cisco. Компанія Cisco робить чудові й оптимальні за співвідношенням «ціна-якість» маршрутизатори; проте стандартизація протоколу EIGRP обмежує ваші можливості майбутнього розширення.

Маршрутизатори, підключені до магістралі Інтернету, повинні використовувати протокол BGP. Зазвичай більшість маршрутизаторів має тільки один вхід і, отже, для них достатньо задати простий статичний стандартний маршрут.

Для організації середнього розміру з відносно стабільною локальною структурою, де є вихід в іншу мережу, підійде поєднання статичної та динамічної маршрутизацій. Маршрутизатори локальної мережі, які не є шлюзами в зовнішні мережі, можуть використовувати статичну маршрутизацію, спрямовуючи всі невідомі пакети стандартній машині, здатній спілкуватися із зовнішнім світом і виконувати динамічну маршрутизацію.

Мережа, керувати якою за такою схемою було б занадто складно, повинна ґрунтуватися на динамічній маршрутизації. У кінцевих мережах (leaf nets), як і раніше, можна використовувати стандартні статичні маршрути, але комп'ютери в мережах із більш ніж одним маршрутизатором повинні запускати демон routed у пасивному режимі.

6.3. Демони маршрутизації

Не рекомендується використовувати системи UNIX і Linux як маршрутизатори у виробничих мережах. Спеціалізовані маршрутизатори простіші, надійніші, безпечніші та більш швидкодіючі (навіть якщо вони приховано запускають ядро системи Linux). Інакше кажучи, дуже добре мати можливість організувати нову підмережу, використовуючи всього лише мережеву карту вартістю 600 грн і комутатор за 1500 грн. Це зовсім не розумний підхід до організації розріджених, тестових і допоміжних мереж.

Системи, що діють як шлюзи в таких підмережах, не потребують додаткових інструментів для управління їхніми власними таблицями маршрутизації. Статичні маршрути цілком адекватні як для машин, що використовуються як шлюзи, так і для машин, що являють собою вузли самої підмережі. Однак якщо ви хочете, щоб ваша підмережа була доступною для інших мереж у вашій організації, вам необхідно повідомити про її існування та ідентифікувати маршрутизатор, до якого будуть прив'язані пакети, які надсилаються цій підмережі. Зазвичай для цього на шлюзі запускається демон маршрутизації.

Системи UNIX і Linux можуть брати участь у більшості протоколів маршрутизації за допомогою різних демонів маршрутизації. Важливим винятком із цього правила є протокол EIGRP, який, наскільки відомо, не має широко доступної реалізації в системах UNIX і Linux.

Оскільки демони маршрутизації рідко реалізуються у виробничих системах, не будемо докладно описувати їхнє використання і конфігурацію.

6.3.1. Демон route: застаріла реалізація в протоколі RIP

Тривалий час демон routed був стандартним демоном маршрутизації, і його досі включають у дистрибутиви деяких систем. Демон routed розуміє тільки протокол RIP і при цьому погано: навіть підтримка версії RIPv2 не виправила ситуацію. Демон routed не розуміє протокол RIPng, реалізація якого заснована на сучасних демонах, таких як Quagga і ramd у системі HP-UX.

Демон routed доцільно використовувати в пасивному режимі (-q), у якому він приймає повідомлення про оновлення маршрутів, але не здійснює широкомовної розсилки власних повідомлень. Крім вказівки опцій у командному рядку, демон routed не потребує конфігурації. Він являє собою дешевий і простий спосіб отримання повідомлень про оновлення маршрутів без складного конфігурування.

Демон заносить виявлені маршрути в таблицю ядра. Усі маршрути мають анонсуватися, як мінімум, кожні чотири хвилини, інакше їх буде видалено. Щоправда, демон route пам'ятає, які маршрути він додавав, і не видаляє статичні маршрути, задані за допомогою команди route.

6.3.2. Демон gated: перший багатопротокольний демон маршрутизації

Демон gated – елегантна і легко доступна програма маршрутизації, що дає змогу одночасно використовувати кілька протоколів маршрутизації. Він надає адміністратору повний контроль над анонсованими маршрутами, широкосмуговими адресами, правилами безпеки та метриками вартості. Демон gated дозволяє кільком протоколам використовувати одні й ті самі маршрути, даючи змогу в такий спосіб створювати шлюзи між автономними областями, у яких прийнято різні системи маршрутизації. Нарешті, демон gated має один із найзручніших командних інтерфейсів і форматів файлів конфігурації серед усього адміністративного програмного забезпечення.

На жаль, демон gated мертвий (або, принаймні, при смерті), хоча пам'ять про нього ще живе у випусках систем HP-UX 3.5.9 і AIX 6.0, які взагалі повільно реагують на зміни, що відбуваються.

Демон gated являє собою повчальний приклад ризику, з яким пов'язані спроби розробляти відкрите програмне забезпечення. Він замислювався як вільно розповсюджувана програма, але в 1992 році його приватизував і переробив консорціум із розроблення програм; унаслідок цього його оновлення стали доступними тільки членам цього консорціуму. Зрештою консорціум було розформовано, і права на комерційну версію демона gated кілька разів «змінювали» власників. Тим часом відкриті проєкти Zebra і Quagga витіснили демон gated і самі стали відігравати провідні ролі в галузі відкритих пакетів маршрутизації. Наразі демон gated згас і як комерційний продукт, і як відкритий проєкт. Сумний кінець корисного і добре спроєктованого пакета.

6.3.3. Пакет Quagga: основний демон маршрутизації

Quagga (quagga.net) – це дослідно-конструкторський підрозділ проєкту Zebra, що виконується в рамках проєкту GNU. Проєкт Zebra запустили Куніхіро Ішигуро (Kunihiro Ishiguro) і Йошинарі Йошикава (Yoshinari Yoshikawa) для реалізації багатопротокольної маршрутизації за допомогою колекції незалежних демонів, а не одного монолітного застосунку. У реальному житті квагга (quagga) - це зниклий підвид зебри, який було востаннє сфотографовано в 1870 році. Але його цифрова ре інкарнація Quagga вижила, а проєкт Zebra закрився.

Наразі пакет Quagga реалізує всі протоколи RIP (усі версії), OSPF (версії 2 і 3), BGP і IS-IS. Він виконується в системах Linux, Solaris і різних варіантах системи BSD. У системі Solaris і версіях системи Linux, наявних у нашому розпорядженні, пакет Quagga або інстальовано за замовчуванням, або він доступний як допоміжний пакет, який можна завантажити зі стандартного системного репозиторію програмного забезпечення.

У пакеті Quagga демон zebra відіграє роль інформаційного центру для маршрутизації. Він керує взаємодією між таблицею маршрутизації ядра і демонами, що відповідають окремим протоколам маршрутизації (ripd, ripngd, ospfd, ospf6d, bgpd

і isisd). Крім того, він керує потоками інформації про маршрути, якими обмінюються протоколи. Кожен демон має свій власний конфігураційний файл у каталозі /etc/quagga.

Для того щоб надіслати запит або змінити конфігурацію, можна з'єднатися з будь-яким із демонів Quagga за допомогою інтерфейсу командного рядка (vtysh у системі Linux і quaggaadm у системі Solaris). Командну мову розроблено так, щоб вона була зрозуміла користувачам операційної системи IOS компанії Cisco. Як і в системі IOS, для того, щоб увійти в режим «суперкористувача», необхідно виконати команду `enable`, для того, щоб увести команди конфігурації, необхідно виконати команду `config term`, а для того, щоб зберегти зміни конфігурації у файлі конфігурації демона, необхідно виконати команду `write`.

Офіційна документація доступна на сайті quagga.net у вигляді файлів у формах HTML і PDF. Незважаючи на свою повноту, ця документація містить, в основному, зручний каталог опцій, а не загальний огляд системи. Більш практичну документацію можна знайти на сайті wiki.quagga.net. Тут містяться детально прокоментовані приклади файлів конфігурації, відповіді на поширені запитання та поради.

Незважаючи на те що файли конфігурації мають простий формат, необхідно знати протокол, конфігурацію якого ви налаштовуєте, а також розуміти, що означають ті чи інші опції.

6.4. Маршрутизатори Cisco.

Маршрутизатори, що випускаються компанією Cisco Systems, Inc., сьогодні є стандартом де-факто на ринку інтернет-маршрутизаторів. Захопивши понад 60% ринку, компанія Cisco домоглася широкої популярності своїх продуктів, до того ж є багато фахівців, які вміють працювати з цими пристроями. Раніше як маршрутизатори часто доводилося використовувати UNIX-системи з кількома мережевими інтерфейсами. Сьогодні спеціалізовані маршрутизатори розміщені в комутаційних шафах і над підвісними стелями, де сходяться мережеві кабелі.

Більшість маршрутизаторів Cisco працює під управлінням операційної системи Cisco IOS, яка є власністю компанії і не має відношення до системи UNIX. У неї досить великий набір команд: повна паперова документація займає на полиці майже півтора метра. Ми ніколи не змогли б повністю описати тут цю операційну систему, але все ж постараємося дати про неї основні відомості.

За замовчуванням у системі IOS визначено два рівні доступу (призначений для користувача і привілейований), кожен з яких містить механізм парольного захисту. За замовчуванням до маршрутизатора Cisco можна отримати доступ на користувацькому рівні, користуючись утилітою telnet.

Наведемо кілька порад, що стосуються ефективної роботи з маршрутизаторами Cisco.

- Надайте маршрутизатору ім'я за допомогою команди `hostname`. Це дасть змогу уникнути нещасних випадків, пов'язаних зі зміною конфігурації не того маршрутизатора. Задане ім'я завжди відображатиметься в командному рядку.
- Завжди зберігайте резервну копію конфігураційного файлу маршрутизатора. За допомогою команд `scp` або `tftp` можна щоночі пересилати поточну конфігурацію в іншу систему на зберігання.

- Найчастіше існує можливість зберігати копію конфігурації в енергонезалежній пам'яті NVRAM або на переносному флеш-накопичувачі. Не нехтуйте цим!

- Сконфігурувавши маршрутизатор для SSH-доступу, вимкніть протокол Telnet зовсім.

- Контролюйте доступ до командного рядка маршрутизатора, створюючи списки доступу для кожного порту VTY (аналогічний порту PTY в UNIX-системі). Це не дозволить стороннім користувачам «зламати» маршрутизатор.

- Керуйте трафіком між мережами (за можливості, і в зовнішній світ теж), створюючи списки доступу для кожного інтерфейсу.

- Фізично обмежуйте доступ до маршрутизаторів. Адже якщо у зловмисника буде фізичний доступ до пристрою, він легко зможе змінити пароль під час вілегованого режиму.

Детальніше роботу з маршрутизаторами Cisco розглядатимемо під час вивчення дисциплін «Комп’ютерні мережі», «Безпека комп’ютерних мереж» та «Управління мережевою безпекою».

ЛЕКЦІЯ 7. Мережеві апаратні засоби

Чи користуєтеся ви пошуковою машиною Google за допомогою свого мобільного телефону, чи виконуєте інтерактивні банківські операції, чи отримуєте відеодзвінки за допомогою програми Skype від своїх кумів із Полтави, практично все у світі нині обробляється в цифровій формі. Переміщення даних з одного місця в інше на думці майже у всіх. В основі всього цього божевілля лежить фантастичне мережеве обладнання і, як ви вже здогадалися, безліч різноманітних програм, створених у надрах системи UNIX. Великомасштабна передача пакетів даних на основі технології UNIX увійшла в життя дуже багатьох людей.

Багато мережевих технологій просувалися протягом довгих років, але в результаті з'явився очевидний переможець: Ethernet. Сьогодні технологію Ethernet можна зустріти всюди: від ігрових приставок до холодильників. Глибоке розуміння принципів роботи цієї системи надзвичайно важливе для успішної роботи системного адміністратора.

Цілком очевидно, що швидкодія і надійність мереж безпосередньо впливають на результати діяльності компаній. Однак нині мережеві технології настільки всюдисущі, що стан мережі може вплинути на можливість взаємодії між людьми, наприклад на можливість робити телефонні дзвінки. Погана організація мережі – це особиста і професійна невдача, яка може мати катастрофічні соціальні наслідки. Крім того, усунення цих недоліків часом обходиться дуже дорого.

7.1. Тестування та налагодження мереж

Успішне створення мережі залежить від принаймні чотирьох найважливіших чинників, а саме:

- розробки розумної структури мережі;
- вибору високоякісного обладнання;
- правильної інсталяції та документування;
- компетентної експлуатації та супроводу.

У цій лекції розглядаються принципи, інсталяція та функціонування мереж Ethernet. Ми також коротко опишемо такі застарілі технології, як DSL (Digital Subscriber Line), які зазвичай постають перед кінцевими користувачами у вигляді (сюрприз!) технології Ethernet.

7.1.1. Технологія Ethernet: мережева панацея

Захопивши понад 95% світового ринку локальних мереж (LAN – Local Area Network), технологія Ethernet у найрізноманітніших формах проявляється майже всюди. Розробку стандарту Ethernet розпочав Боб Меткалф (Bob Metcalfe) з Массачусетського технологічного інституту в рамках своєї кандидатської дисертації, але наразі вона описана в багатьох стандартах IEEE.

У початковій специфікації Ethernet було визначено швидкість передавання даних 3 Мбіт/с (мегабіт за секунду), але майже відразу ж вона зросла до 10 Мбіт/с. Щойно в 1994 році було закінчено роботу над стандартом, який передбачав швидкість 100 Мбіт/с, стало зрозуміло, що технологія Ethernet буде лише еволюціонувати, а не витіснятися новою технологією. Це спричинило перегони технологій, під час яких виробники намагалися створити все більш швидкодіючу версію Ethernet, і це

змагання ще не завершено. Основні етапи еволюції різних стандартів Ethernet наведено на рис. 7.1.

Year	Speed	Common name	IEEE#	Dist	Media[a]
1973	3 Mb/s	Xerox Ethernet	–	?	Coax
1976	10 Mb/s	Ethernet 1	–	500m	RG-11 coax
1989	10 Mb/s	10BASE-T	802.3	100m	Cat 3 UTP copper
1994	100 Mb/s	100BASE-TX	802.3u	100m	Cat 5 UTP copper
1999	1 Gb/s	1000BASE-T (“gigabit”)	802.3ab	100m	Cat 5e, 6 UTP copper
2006	10 Gb/s	10GBASE-T (“10 gig”)	802.3an	100m	Cat 6a, 7, 7a UTP
2009	40 Gb/s	40GBASE-CR4	P802.3ba	10m	UTP copper
		40GBASE-SR4		100m	MM fiber
2009	100 Gb/s	100GBASE-CR10	P802.3ba	10m	UTP copper
		100GBASE-SR10		100m	MM fiber
2018[b]	200 Gb/s	200GBASE-FR4	802.3bs[c]	2km	CWDM fiber
		200GBASE-LR4		10km	CWDM fiber
2018[b]	400 Gb/s	400GBASE-SR16	802.3bs	100m	MM fiber (16 strand)
		400GBASE-DR4		500m	MM fiber (4 strand)
		400GBASE-FR8		2km	CWDM fiber
		400GBASE-LR8		10km	CWDM fiber
2020[b]	1 Tb/s	TbE	TBD	TBD	TBD

Рисунок 7.1 – Основні етапи еволюції різних стандартів Ethernet

7.1.2. Як працює Ethernet

Технологію Ethernet можна уявити у вигляді великосвітського рауту, на якому гості (комп'ютери) не перебивають один одного, а чекають паузи в розмові (відсутності трафіку в мережевому кабелі), щоб заговорити. Якщо два гості починають говорити одне тимчасово (тобто виникає конфлікт), обидва вони зупиняються, вибачаються одне перед одним, чекають трохи, а потім один із них починає говорити знову.

У технічній термінології така схема називається CSMA/CD (Carrier Sense Multiple Access with Collision Detection – множинний доступ із контролем несучої та виявленням конфліктів). Сенс цієї назви полягає в такому:

- контроль несучої – можна визначити, чи зайнятий канал;
- множинний доступ – хто завгодно може передавати повідомлення;
- виявлення конфліктів – передавальна система «знає», коли вона «перебиває» кого-небудь.

Фактична затримка при виявленні конфліктів є випадковою. Це дає змогу уникнути такого розвитку подій, за якого два комп'ютери одночасно передають повідомлення в мережу, виявляють колізію, чекають певний час, а потім синхронно відновлюють передачу, переповнюючи, таким чином, мережу конфліктами.

Нині важливість угод CSMA/CD усвідомили навіть прихильники комутаторів, які зазвичай обмежують кількість вузлів у домені, в якому відбуваються колізії, до двох. (Якщо продовжити аналогію з великосвітським раутом, можна описати цей варіант як ситуацію, у якій двоє співрозмовників, як у старому кіно, манірно сидять на протилежних кінцях довгого обіднього столу).

7.1.3. Топологія Ethernet

З точки зору топології, мережа Ethernet являє собою розгалужену шину, але без петель. У пакета є тільки один шлях проходження між будь-якими двома вузлами, розташованими в одній мережі. У мережі Ethernet можуть передаватися пакети трьох типів: односпрямовані, групові та широкомовні. Пакети першого типу адресовані одному вузлу, другого – групі вузлів, третього – усім вузлам сегмента.

Широкомовний домен – це сукупність вузлів, які приймають пакети, що направляються за апаратною широкомовною адресою. У кожному логічному сегменті мережі Ethernet існує тільки один широкомовний домен. У ранніх стандартах Ethernet і засобах передавання (наприклад 10Base5) поняття фізичного та логічного сегментів були тотожними, оскільки всі пакети передавали по одному великому кабелю, у який встромляли мережеві інтерфейси комп'ютерів3.

З появою сучасних комутаторів логічні сегменти стали містити в собі безліч (десятки і навіть сотні) фізичних сегментів, до яких під'єднано лише два пристрої: порт комутатора і комп'ютер. Комутатори відповідають за доставку групових і односпрямованих пакетів у фізичний сегмент, де розташований потрібний адресат (адресати); широкомовні пакети спрямовують в усі мережеві порти логічного сегмента.

Логічний сегмент може складатися з фізичних сегментів, що мають різну швидкість передачі даних (10 Мбіт/с, 100 Мбіт/с, 1 Гбіт/с або 10 Гбіт/с). Отже, комутатори повинні мати засоби буферизації та синхронізації для запобігання можливих конфліктів.

7.1.4. Неекранована кручена пара

Неекранована кручена пара (UTP) – найпопулярніше середовище передавання даних у мережах Ethernet. Воно засноване на зіркоподібній топології та має низку переваг порівняно з іншими середовищами:

- застосовується недорогий, недефіцитний мідний кабель (іноді можна використовувати готову телефонну проводку);
- з'єднання на основі UTP набагато простіше змонтувати і налагодити, ніж з'єднання на основі коаксіального кабелю або оптичного волокна, до того ж легко підібрати потрібну довжину кабелю;
- використовуються дешеві, надійні та зручні в експлуатації роз'єми RJ-45;
- усі з'єднання функціонують незалежно одне від одного, тому дефект кабельної системи в одному з'єднанні не вплине на інші комп'ютери мережі.

Загальна схема мережі на основі UTP зображена на рис. 7.2.

Дроти, що використовуються в сучасних локальних обчислювальних мережах на основі неекранованої крученої пари, зазвичай поділяють на вісім категорій. Цю систему оцінювання параметрів вперше запровадила компанія Anixter, великий постачальник кабельної продукції, і згодом стандартизувала організація TIA (Telecommunications Industry Association – асоціація телекомунікаційної промисловості). Сьогодні виділяють категорії 1-7 і проміжні категорії 5e і 6a.

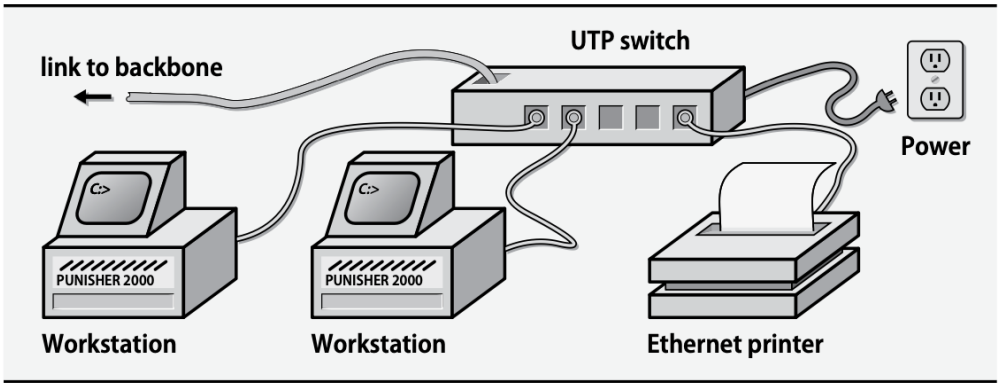


Рисунок 7.2 – Загальна схема мережі на основі UTP

Організація ISO (International Organization for Standartization – Міжнародна організація зі стандартизації) теж долучилася до процесу стандартизації кабелів і запропонувала власну класифікацію, яка майже в точності повторює класифікацію TIA. Наприклад, кабель категорії 5 у системі TIA еквівалентний кабелю класу D у системі ISO. Для наочності на рис. 6.3 підсумовано ключові відмінності між основними сучасними стандартами кабелів.

Parameter	Units	Cat 5[b] Class D	Cat 5e	Cat 6 Class E	Cat 6a Class EA	Cat 7 Class F	Cat 7a Class FA	Cat 8 Class I
Frequency range	MHz	100	100	250	500	600	1000	2000
Attenuation	dB	24	24	21.7	18.4	20.8	60	50
NEXT[a]	dB	27.1	30.1	39.9	59	62.1	60.4	36.5
ELFEXT[a]	dB	17	17.4	23.2	43.1	46.0	35.1	–
Return loss	dB	8	10	12	32	14.1	61.93	8
Propagation delay	ns	548	548	548	548	504	534	548

a. NEXT = Near-end crosstalk, ELFEXT = Equal level far-end crosstalk
b. Includes additional TIA and ISO requirements TSB95 and FDAM 2, respectively

Рисунок 7.3 – Основні характеристики сучасних кабелів UTP

На практиці кабелі категорій 1 і 2 придатні тільки для передавання звукових сигналів. Кабель категорії 3 є стандартом для найповільніших локальних мереж: 10BaseT зі швидкістю передачі 10 Мбіт/с. Кабель категорії 4 ні для чого конкретного не призначений.

Кабель категорії 5 підтримує роботу на швидкості 100 Мбіт/с. Кабелі категорій 5е, 6 і 6а підтримують швидкість передавання 1 Гбіт/с і наразі використовуються як стандарт. Кабель категорії 6а найкраще підходить для організації нових мереж, оскільки він особливо стійкий до завад, що виникають через використання старих стандартів передачі сигналів (наприклад, 10BaseT). Під час прокладання кабелів категорій 5 і 5е було зафіксовано певні проблеми. Кабелі категорій 7 і 7а призначені для передавання даних зі швидкістю 10 Гбіт/с.

Для з'єднань 10BaseT потрібні дві пари дротів категорії 3, причому довжина кожної лінії передачі обмежена 100 метрами. У з'єднаннях 100BaseTX гранична гранична довжина та сама, але використовуються дві пари проводів категорії 5.

З'єднання 1000BaseTX вимагає чотирьох пар дротів категорій 5e або 6/6a. Аналогічно з'єднання 10GBaseTX вимагає чотирьох пар проводів категорій 6a, 7 або 7a.

Стандарти 1000BaseTX і 100BaseTX призначені для передавання даних багатьма парами дротів. Вони використовують багатожильні кабелі для передавання даних лінією зв'язку, які виконують її швидше, ніж це може здійснити будь-яка окрема пара дротів.

Існують дроти з полівінілхлоридною і тефлоновою ізоляцією. Вибір ізоляції диктується середовищем, у якому будуть прокладені кабелі. У замкнутих приміщеннях, пов'язаних із вентиляційною системою будівлі, зазвичай потрібна тефлонова ізоляція. Полівінілхлоридна ізоляція дешевша і простіша в експлуатації.

7.1.5. Оптичне волокно

Оптичне волокно використовується в тих ситуаціях, коли використання мідного кабелю з тих чи інших причин є неприйнятним. Оптичне волокно передає сигнал швидше, ніж мідний дріт. Крім того, воно є більш стійким до електричних завад, що в деяких додатках дуже важливо. Там, де оптичне волокно не є абсолютно необхідним, зазвичай обирають мідний кабель, оскільки він дешевший і з ним легше працювати.

Оптичне волокно буває «багатомодовим» і «одномодовим». Багатомодове оптичне волокно зазвичай використовується в будівлях або комплексах будівель. Воно товстіше, ніж одномодове, і може проводити кілька променів світла; ця властивість дає змогу використовувати менш дорогу електроніку (наприклад, як джерело світла можна використовувати світлодіоди).

Одномодове оптичне волокно часто використовується в магістральних застосунках, наприклад, для прокладання ліній зв'язку між містами і регіонами. Воно може проводити тільки один світловий промінь і вимагає дорогої прецизійної електроніки в кінцевих точках.

На кінцях оптичних волокон використовуються роз'єми більш ніж 30 типів, і немає ні чітких правил, ні принципів, що регламентують їх вибір. У кожній конкретній ситуації на вибір того чи іншого типу роз'єму впливають постачальники обладнання або параметри оптичного волокна, вже прокладеного всередині будівлі.

7.1.6. З'єднання та розширення мереж Ethernet

Мережі Ethernet можна з'єднувати за допомогою пристроїв декількох типів. На вибір пристроїв, описаних нижче, впливає їхня вартість, причому дешевші пристрої описано насамперед. Що складніші логічні правила, за якими пристрої переміщують біти з однієї мережі в іншу, то більше апаратного та вбудованого програмного забезпечення необхідне і то дорожчою стає мережа.

7.1.7. Концентратори

Концентратори (hub) іноді ще називають повторювачами (repeators). Це активні пристрої, що використовуються для з'єднання сегментів мереж Ethernet на фізичному рівні. Їм потрібне зовнішнє джерело живлення.

Виступаючи як повторювач, концентратор ретранслює Ethernet-фрейми, але ніяк не інтерпретує їх. Він «не має уявлення» ні про те, куди прямують пакети, ні про те, який протокол вони використовують. За винятком екзотичних ситуацій,

концентратори більше не повинні використовуватися в промислових мережах, і не слід їх використовувати навіть у домашніх мережах. (Чому? Тому що комутатори (switches) значно ефективніше використовують смугу пропускання частот у мережі та в теперішній час коштують недорого).

7.1.8. Комутатори

Комутатор з'єднує мережі Ethernet на канальному рівні. Його призначення – об'єднати дві фізичні мережі так, щоб вони виглядали як одна велика фізична мережа. Нині комутатори є промисловим стандартом для з'єднання пристроїв Ethernet.

Комутатори приймають, регенерують і ретранслюють пакети на апаратному рівні. Вони використовують алгоритм динамічного навчання. Вони запам'ятовують, які вихідні адреси надходять з одного порту, а які – з іншого. Пакет переходить з одного порту в інший тільки за необхідності. Спочатку пересилаються всі пакети, але через кілька секунд, коли комутатор вивчить розташування більшості вузлів мережі, запускається механізм фільтрації.

Оскільки між мережами пересилаються не всі пакети, кожен сегмент кабелю менш завантажений, ніж у разі, коли всі комп'ютери під'єднані до одного кабелю. А якщо врахувати, що основний трафік має тенденцію до локалізації, то збільшення реальної пропускної здатності може виявитися помітним. Крім того, комутатор не впливає на логічну модель мережі, тому його встановлення потребує лише незначного втручання з боку адміністратора.

Якщо мережа має петлі, комутатор може безнадійно заплутатися, тому що пакети, які надсилаються одним комп'ютером, опиняться відразу на двох (або більше) портах комутатора. В одній мережі Ethernet петлі не буває, але після об'єднання декількох таких мереж за допомогою маршрутизаторів і комутаторів топологія зміниться, внаслідок чого може утворитися кілька шляхів до одного вузла. Деякі комутатори вирішують цю проблему шляхом резервування альтернативних маршрутів на той випадок, якщо основний маршрут стане недоступним. Вони спрощують топологію видимої ними мережі, відсікаючи дублюючі шляхи доти, доки в сегментах, що залишилися, не опиниться тільки по одному маршруту до кожного вузла мережі. Інші комутатори створюють між мережами подвійні канали і перемикають трафік за циклічним принципом.

Комутатори повинні переглядати кожен пакет, визначаючи, чи потрібно його пересилати в інший сегмент. Продуктивність цих пристроїв зазвичай вимірюють як швидкість огляду пакетів, так і швидкість їх пересилання. Багато постачальників не вказують у діаграмах продуктивності комутаторів розміри протестованих пакетів, тому реальна продуктивність може бути нижчою за оголошену.

Незважаючи на те що швидкодія комутаторів Ethernet увесь час зростає, ефективно використовувати їх можна за умови об'єднання в один логічний сегмент не більше сотні комп'ютерів. У великих комутованих мережах часто виникають проблеми на кшталт «широкомовних штормів», оскільки широкомовний трафік має проходити через усі порти. Для вирішення цієї проблеми потрібно ізолювати широкомовний трафік між комутованими сегментами за допомогою маршрутизатора (створюючи тим самим більше одного логічного Ethernet-сегмента).

Вибір комутатора може становити певні труднощі. У цьому сегменті ринку дуже висока конкуренція, наслідком якої є численні рекламні заяви, які не завжди

підтверджуються на практиці. Тому не варто особливо довіряти даним, які наводять постачальники; краще прислухатися до порад незалежних експертів (перегляньте тести, наведені в журналах). Останніми роками нерідко траплялося так, що чийсь продукт виявлявся «найкращим» упродовж кількох місяців, а потім, після спроб внесення поліпшень, його продуктивність або надійність падала нижче критичної позначки.

У будь-якому разі переконайтеся, що швидкість об'єднувальної панелі комутатора є достатньою. У добре спроєктованого комутатора ця швидкість має перевищувати суму швидкостей усіх його портів.

7.1.9. Маршрутизатори

Маршрутизатори (відомі також як «комутатори третього рівня») направляють трафік на третьому мережевому рівні моделі OSI. Маршрутизатори доставляють пакети адресатам на підставі інформації, що зберігається в TCP/IP-заголовках. Крім простого переміщення пакетів, маршрутизатори можуть також виконувати низку особливих функцій, наприклад фільтрацію пакетів (згідно з правилами безпеки), розподіл трафіку за пріоритетами (відповідно до заданої якості обслуговування) і виявлення загальної мережевої топології.

Конфігурація маршрутизаторів буває фіксованою або модульною. Пристрої першого типу містять мережеві інтерфейси, встановлені в заводських умовах. Вони зазвичай підходять для спеціалізованих застосувань. Наприклад, маршрутизатор з інтерфейсами T1 і Ethernet може виявитися зручним, коли потрібно підключити невелику компанію до Інтернету.

Модульні маршрутизатори мають слотову або шинну архітектуру, а інтерфейси до них додаються користувачами. Як правило, це дорожчі пристрої, але зате вони гнучкіші в експлуатації.

Залежно від необхідної надійності та очікуваного трафіку, спеціалізований маршрутизатор може виявитися як дорожчим, так і дешевшим за систему UNIX або Linux, сконфігуровану як маршрутизатор. Однак спеціалізований пристрій, як правило, демонструє вищу продуктивність і надійність. Це та область мережевого проєктування, де краще заздалегідь вкласти трохи більше грошей, ніж потім мати головний біль.

7.1.10. Автоузгодження

З появою різних стандартів Ethernet виникла необхідність, щоб пристрої могли ідентифікувати конфігурацію своїх сусідів і узгоджувати з ними свої налаштування. Наприклад, мережа не працюватиме, якщо на одній стороні з'єднання вона працює зі швидкістю 1 Гбіт/с, а на іншій – зі швидкістю 10 Гбіт/с. Для виявлення і розв'язання цієї проблеми організацією IEEE було розроблено стандарт автоузгодження Ethernet. В одних випадках він працює, а в інших застосовується неправильно і лише посилює проблему.

Слід запам'ятати два золотих правила автоузгодження.

- Ви зобов'язані використовувати автоузгодження всіх інтерфейсів, що працюють на швидкості 1 Гбіт/с і вище. Цього вимагає стандарт.

- Якщо інтерфейси обмежені швидкостями 100 Мбіт/с і нижче, необхідно або конфігурувати обидва кінці з'єднання, або вручну налаштувати швидкість і дуплекс

(половинний або повний) обох сторін. Якщо в режимі автоузгодження налаштувати тільки одну сторону з'єднання, то здебільшого вона не зможе з'ясувати, яку конфігурацію має інша сторона. У результаті конфігурація стане неузгодженою і продуктивність знизиться.

7.1.11. Передавання електроживлення мережами Ethernet

Технологія передавання живлення мережами Ethernet (PoE – Power on Ethernet) ґрунтується на передаванні електроживлення тією самою неекранованою крученою парою (UTP Ethernet), якою передається сигнал Ethernet. Ця технологія регламентується стандартом IEEE 802.3af. Це особливо зручно для систем зв'язку, що забезпечують передавання мовного сигналу мережею Інтернет (VoIP – Voice over IP), або пунктів доступу до системи бездротового зв'язку (ми навели тільки два приклади, але список можна продовжити), у яких потрібна як сила струму, так і мережеве з'єднання.

За потужністю, системи PoE поділяються на чотири класи в діапазоні від 3,84 до 12,95 Вт. Промисловість, яка ніколи не зупиняється на досягнутому, вже працює над новим стандартом (802.3at), що передбачає вищу потужність (понад 60 Вт).

Технологія PoE породжує дві обставини, про які має знати системний адміністратор.

- Ви маєте знати про існування пристроїв PoE у вашій інфраструктурі, щоб правильно спланувати доступ до портів комутаторів, що підтримують технологію PoE. Ці порти дорожчі, ніж порти, що не підтримують технологію PoE.

- Обчислюючи витрати електроенергії на обслуговування комунікаційних шаф, що містять комутатори PoE, слід враховувати потужність пристроїв PoE. Зверніть увагу на те, що ви не повинні планувати ту саму витрату електроенергії на додаткове охолодження комунікаційних шаф, оскільки більша частина тепла, що виділяється через споживання потужності PoE, розсіюється за межами шафи (зазвичай по офісу).

7.1.12. Гігантські пакети

Технологія Ethernet стандартизована для типового пакета розміром 1 500 байт (разом із фреймом – 1 518 байт). Це значення було вибрано давно, коли мережі були повільними і пам'ять для буферів була дефіцитною. Нині пакети розміром 1 500 байт виглядають крихітними в контексті гігабітних мереж Ethernet.

Оскільки з кожним пакетом пов'язані накладні витрати і певний час затримки, продуктивність мережі можна підвищити, якщо допустити більші розміри пакетів.

На жаль, стандарти IEEE для різних типів мереж Ethernet забороняють використання великих пакетів з міркувань сумісності мереж. Однак оскільки швидкість магістрального трафіку часто у багато разів перевищує встановлену межу, не стандартні великі пакети Ethernet у сучасних мережах перестали бути рідкістю. Підбурювані нетерплячими споживачами, виробники мережевого обладнання негласно бойкотують стандарт IEEE і забезпечують підтримку великих фреймів у своїй гігабітній продукції.

Для використання так званих гігантських пакетів (jumbo frames) необхідно лише підвищити максимально можливий розмір пакета (MTU – maximal transmission unit) в інтерфейсах мережі. Підвищення продуктивності залежить від виду трафіку, але найбільший виграш досягається для великомасштабних переміщень за протоколом

TCP (наприклад, у файлових службах NFSv4 або CIFS). Очікується, що помірне, але помітне підвищення продуктивності має скласти приблизно 10%.

Проте слід зазначити таке.

- Підтримувати і використовувати гігантські пакети має все мережеве обладнання в підмережах, включно з комутаторами і маршрутизаторами. Їх не можна змішувати і підганяти.
- Оскільки гігантські пакети є нестандартними, зазвичай їх необхідно дозволяти явно. Пристрої можуть приймати гігантські пакети за замовчуванням, але, найімовірніше, вони не будуть їх генерувати.
- Оскільки гігантські пакети являють собою незаконне явище, не існує консенсусу, наскільки великими вони можуть або повинні бути. Типовою величиною є 9 000 біт або 9 018 разом із фреймом. Необхідно перевірити, який максимальний розмір пакета може прийняти ваш пристрій. Пакети розміром понад 9 Кбайт іноді називають надгігантськими, але ця екзотична назва вас лякати не повинна. Що більший розмір, то краще, принаймні в діапазоні до 64 Кбайт.
- Гігантські пакети можуть існувати тільки у внутрішніх мережах. Інтернет не передає такі пакети.

Зазвичай схвалюється використання гігантських пакетів у гігабітних мережах Ethernet, але тільки там, де це легко і безпечно (наприклад, у складних промислових середовищах їх використовувати навряд чи варто). Будьте готові до додаткового налагодження, якщо щось піде не так, як треба. Найкраще розгорнути нову мережу, задавши максимально можливий розмір пакета за замовчуванням, а пізніше, коли надійність мережі буде перевірено, змінити ці налаштування і дозволити гігантські пакети.

7.1.13. Бездротовий стандарт: локальна мережа для кочівників

Бездротові мережі складаються з бездротових точок доступу (WAP – Wireless Access Points) і клієнтів бездротової мережі. Точки WAP можуть з'єднуватися традиційними дротовими мережами (звичайна конфігурація) або з іншими точками WAP без використання дротів (конфігурація відома під назвою «бездротова мережа»).

Точки WAP зазвичай оснащують спеціальним обладнанням, що складається з одного або кількох радіоприймачів і вбудованої операційної системи, яка часто являє собою усічений варіант системи Linux. Одна точка WAP може забезпечити доступ для багатьох клієнтів, але їхня кількість обмежена. Зазвичай одна точка промислової мережі одночасно обслуговує не більше восьми клієнтів. Будь-який пристрій, що діє за допомогою бездротового стандарту, підтримується точкою WAP як клієнт.

Найпоширенішими стандартами бездротового зв'язку сьогодні є IEEE 802.11g, 802.11n та 802.11ac. 802.11g працює в діапазоні частот 2,4 ГГц і забезпечує доступ, подібний до локальної мережі, зі швидкістю до 54 Мбіт/с. Робочий діапазон варіюється від 100 метрів до 40 кілометрів, залежно від обладнання та рельєфу місцевості.

Стандарт 802.11n забезпечує пропускну здатність до 600 Мбіт/с і може використовувати як діапазон 5 ГГц, так і 2,4 ГГц (хоча для розгортання рекомендується використовувати діапазон 5 ГГц). Типовий робочий діапазон приблизно вдвічі більший, ніж у 802.11g. 802.11ac є розширенням стандарту 802.11n з підтримкою пропускної здатності до 1 Гбіт/с між декількома станціями.

Всі ці стандарти охоплюються загальним терміном «Wi-Fi». Теоретично, ярлик Wi-Fi обмежується реалізаціями Ethernet з сімейства IEEE 802.11. Однак це єдиний тип бездротового обладнання Ethernet, який ви можете купити, тому весь бездротовий Ethernet - це Wi-Fi.

Сьогодні поширені стандарти 802.11g і 802.11n. Трансивери недорогі і вбудовані в більшість ноутбуків. Для настільних комп'ютерів також широко і дешево доступні додаткові плати.

7.1.14. Бездротовий клієнтський доступ

Ви можете налаштувати UNIX або Linux для підключення до бездротової мережі як клієнт, якщо у вас є відповідне обладнання та драйвери. Оскільки більшість бездротових карт для ПК розроблено для Microsoft Windows, вони можуть не постачатися з заводу з драйверами для FreeBSD або Linux.

При спробі додати бездротове підключення до системи FreeBSD або Linux вам, ймовірно, знадобляться ці команди:

- `ifconfig` - налаштування бездротового мережевого інтерфейсу
- `iwlist` - створити список доступних точок бездротового доступу
- `iwconfig` - налаштування параметрів бездротового з'єднання
- `wpa_supplicant` - автентифікація до бездротової (або дротової 802.1x) мережі

На жаль, шалена боротьба за продаж дешевого обладнання часто призводить до того, що налагодження коректної роботи бездротового адаптера під UNIX або Linux може зайняти години спроб і помилок. Плануйте заздалегідь або купуйте такий самий адаптер, з яким комусь пощастило працювати в Інтернеті під тією ж версією ОС, що і у вас.

7.1.15. Бездротова інфраструктура та точки доступу

Всі хочуть мати бездротове з'єднання скрізь і всюди, і для забезпечення бездротового зв'язку пропонується широкий спектр продуктів. Але, як і у випадку з багатьма іншими речами, ви отримуєте те, за що платите. Недорогі пристрої часто задовольняють потреби домашніх користувачів, але погано масштабуються в корпоративному середовищі.

Точки доступу – це, як правило, спеціальні пристрої, які складаються з одного або декількох радіоприймачів і певної форми вбудованої мережевої операційної системи, часто урізаної версії Linux. Одна точка доступу може забезпечити точку з'єднання для кількох клієнтів, але не для необмеженої кількості клієнтів. Хорошим емпіричним правилом є обслуговування не більше сорока одночасних клієнтів з однієї точки доступу корпоративного класу. Клієнтом може бути будь-який пристрій, який підтримує стандарт бездротового зв'язку, що підтримується вашою точкою доступу.

Точки доступу WAP налаштовані на оголошення одного або декількох «ідентифікаторів набору послуг», також відомих як SSID. SSID діє як ім'я бездротової локальної мережі і має бути унікальним у межах певної області. Коли клієнт хоче підключитися до бездротової мережі, він прослуховує, які SSID оголошуються, і дозволяє користувачеві вибрати одну з цих мереж.

Ви можете назвати свій SSID чимось корисним і легким для запам'ятовування, наприклад, «Chnut», «KBMM_Pentest», або ж підійти до справи творчо – «dosyt sverdlyty u vyhydny».

Однак, небагато аспектів бездротової мережі є дійсно простими. Що робити, якщо ваш будинок або будівля занадто великі, щоб обслуговуватися однією точкою доступу? Або якщо вам потрібно надати різні мережі різним групам користувачів (наприклад, працівникам і гостям)? Для цих випадків вам потрібно стратегічно структурувати вашу бездротову мережу.

Ви можете використовувати декілька SSID, щоб розділити групи користувачів або функції. Зазвичай, ви зіставляєте їх з окремими VLAN, які потім можете маршрутизувати або фільтрувати за бажанням, так само, як і в дротових мережах.

Частотний спектр, виділений для бездротового зв'язку 802.11, розбитий на смуги, які називаються каналами. Наданий сам собі, WAP вибирає тихий радіоканал для оголошення SSID. Клієнти і точка доступу потім використовують цей канал для зв'язку, утворюючи єдиний домен трансляції. Точки доступу, що знаходяться поруч, швидше за все, виберуть інші канали, щоб максимізувати доступну пропускну здатність і мінімізувати завади.

Теорія полягає в тому, що коли клієнти переміщуються середовищем, вони від'єднуються від однієї точки доступу, коли її сигнал слабшає, і підключаються до ближчої точки доступу з сильнішим сигналом. Однак теорія і реальність часто не співпрацюють. Багато клієнтів мертвою хваткою тримаються за слабкий сигнал точки доступу та ігнорують кращі варіанти.

У більшості ситуацій ви повинні дозволити WAP автоматично вибирати улюблені канали. Якщо ви повинні втручатися в цей процес вручну і використовуєте 802.11b/g/n, виберіть канал 1, 6 або 11. Спектр, виділений для цих каналів не перекривається, тому комбінації цих каналів створюють найбільшу ймовірність широко відкритої бездротової магістралі. Канали за замовчуванням для 802.11a/ac взагалі не перетинаються, тому просто виберіть свій улюблений номер.

Деякі точки доступу мають кілька антен і використовують технологію множинного входу і множинного виходу (MIMO). Ця практика може збільшити доступну смугу пропускання за рахунок використання декількох передавачів і приймачів для використання переваг зміщення сигналу, що виникає внаслідок затримки розповсюдження. У деяких ситуаціях ця технологія може забезпечити незначне поліпшення продуктивності, хоча, ймовірно, не настільки значне, як можна було б очікувати через вражаюче збільшення кількості антен.

Якщо вам потрібна фізично більша зона покриття, розгорніть кілька точок доступу. Якщо територія повністю відкрита, ви можете розгорнути їх у вигляді сітки. Якщо на території є стіни та інші перешкоди, ви можете інвестувати в професійне обстеження бездротового зв'язку. Дослідження визначить найкращі варіанти розміщення точок доступу, враховуючи фізичні характеристики вашого простору.

7.1.16. Безпека бездротових мереж

Безпека бездротових мереж традиційно була низькою. Wired Equivalent Privacy (WEP) - це протокол, який використовується в застарілих мережах 802.11b для шифрування пакетів, що передаються через ефір. На жаль, цей стандарт містить фатальний недолік, який робить його не більше, ніж обмежувачем швидкості для

шпигунів. Хтось, хто сидить за межами вашої будівлі або будинку, може отримати доступ до вашої мережі безпосередньо і непомітно, як правило, менш ніж за хвилину.

Нещодавно стандарти безпеки Wi-Fi Protected Access (WPA) породили нову впевненість у безпеці бездротових мереж. Сьогодні WPA (зокрема, WPA2) слід використовувати замість WEP у всіх нових інсталяціях. Без WPA2 бездротові мережі слід вважати абсолютно незахищеними, і їх не можна використовувати в корпоративних брандмауерах. Не використовуйте WEP навіть вдома!

Щоб пам'ятати, що саме WEP є небезпечним, а WPA – безпечним, просто запам'ятайте, що WEP розшифровується як Wired Equivalent Privacy (дротовий еквівалент конфіденційності). Назва точна: WEP дає вам такий самий захист, як якщо б ви дозволили комусь підключитися безпосередньо до вашої дротової мережі. (Тобто, взагалі ніякого захисту – принаймні на рівні IP-адреси).

7.1.17. SDN: програмно-визначена мережа

Як і у випадку з віртуалізацією серверів, відокремлення фізичного мережевого обладнання від функціональної архітектури мережі може значно підвищити гнучкість і керованість. Найкраще на цьому шляху просувається рух програмно-визначених мереж (SDN).

Основна ідея SDN полягає в тому, що компоненти, які керують мережею (площина управління), фізично відокремлені від компонентів, які пересилають пакети (площина даних). Площина даних програмується через площину управління, тому ви можете точно налаштувати або динамічно конфігурувати шляхи передачі даних для досягнення цілей продуктивності, безпеки та доступності.

Як і багато іншого в нашій промисловості, SDN для корпоративних мереж стала чимось на зразок маркетингового трюку. Початковою метою була стандартизація способів реконфігурації мережевих компонентів, що залежать від постачальника. Хоча частина цього ідеалізму була реалізована, багато постачальників зараз пропонують власні корпоративні продукти SDN, які дещо суперечать початковій меті SDN. Якщо ви досліджуєте простір корпоративної SDN, обирайте продукти, які відповідають відкритим стандартам і сумісні з продуктами інших виробників.

7.1.18. DSL і кабельні модеми: «остання миля»

Для великих компаній не складає особливих труднощів переміщати великі обсяги даних. Такі технології, як T1, T3, SONET, MPLS і Frame Relay, реалізують достатньо прості канали обміну даними. Але вони не підходять для підключення до мережі в домашніх умовах. Їхня вартість занадто висока, та й не завжди доступне відповідне обладнання.

У технології DSL (Digital Subscriber Line – цифрова абонентська лінія) використовується звичайний мідний телефонний дріт, яким передають дані зі швидкістю до 24 Мбіт/с (щоправда, для типового DSL-з'єднання цей показник перебуває в діапазоні від 256 Кбіт/с до 5 Мбіт/с). У більшості будинків є телефонна проводка, що робить цю технологію дуже зручною для телефонних компаній. З боку користувача DSL-лінія закінчується в пристрої, що працює подібно до маршрутизатора TCP/IP. До нього підключаються локальні Ethernet-пристрої.

На відміну від звичайних телефонних ліній і ISDN-з'єднань, що вимагають «додзвонюватися» до абонента, DSL – це виділена лінія, в якій постійно є зв'язок. Це

робить технологію DSL ще більш привабливою, оскільки відсутні затримки, пов'язані з початковим конфігуруванням і дозвоном.

Існує кілька різновидів технології DSL, тому назву технології часто наводять у вигляді xDSL, де x – префікс різновиду, наприклад: A (асиметрична), S (симетрична), H (високошвидкісна), RA (з адаптованою швидкістю) і I (DSL-на-ISDN). Останній варіант особливо корисний для віддалених від офісу робочих місць, якщо швидкість передавання даних має бути вищою, ніж у звичайній технології DSL. Конкретний варіант реалізації та доступна швидкість передачі залежать від обладнання, встановленого в центральному офісі або в інтернет-провайдера.

Підключення до мережі сотень мільйонів домашніх користувачів – заповітна мрія багатьох компаній. Тут замішані великі гроші. Технологія DSL ґрунтується на наявній інфраструктурі телефонних ліній, інвестиції в яку дали змогу операторам місцевого зв'язку (ILEC – Incumbent Local Exchanges Carriers) отримувати казкові прибутки, доки повз них проносилася мережева революція 80-х-90-х років.

Компанії кабельного телебачення, що розвивають власну оптоволоконну інфраструктуру, пропонують високошвидкісні (хоч і асиметричні) рішення для користувачів. В індустрії кабельних модемів не так давно розпочався процес стандартизації протоколів передавання даних, і сьогодні рекламують стандарт DOCSIS (Data Over Cable Service Interface Specification – специфікація інтерфейсу передавання даних кабельними лініями). Він визначає технічні характеристики як кабельних модемів, так і обладнання компанії, що надає відповідні послуги, і дозволяє взаємодіяти пристроям різних виробників.

Якщо порівнювати DSL і кабельну технологію, то виграш у конкурентній боротьбі дістанеться тому, що забезпечить вищу частоту передавання даних у конкретну точку за нижчою ціною. Хороші новини для споживачів у тому, що це змушує компанії інвестувати кошти в інфраструктуру для обслуговування житлових районів.

7.1.19. Тестування та налагодження мереж

Основною причиною широкомасштабного переходу до стандарту Ethernet (і до інших технологій, заснованих на використанні неекранованої крученої пари) стала простота налагодження мережі. Оскільки ці мережі можна перевіряти посегментно, апаратні проблеми часто вирішують за лічені секунди.

Ключ до налагодження мережі – її розбивка на сегменти і тестування кожного з них доти, доки не буде виявлено несправність. Загадкові лампочки на комутаторах і концентраторах (що позначають, наприклад, стан каналу і наявність трафіку пакетів) допомагають швидко виявити джерело проблеми. Для того щоб ці індикатори працювали так, як ви хочете, слід керуватися першокласною документацією.

Як завжди, важливо мати під рукою потрібні інструменти, щоб виконати роботу правильно і без зволікань. На ринку пропонуються засоби мережевого налагодження двох типів (щоправда, спостерігається тенденція до їх об'єднання).

Пристрій першого типу – ручний кабельний тестер. Він вимірює електричні характеристики кабелю, включно з його довжиною (для цього застосовується особлива технологія, яка називається рефлектометрією в часовій області). Такі пристрої здатні виявляти найпростіші проблеми, наприклад розрив або неправильну розводку кабелю. Нашим улюбленим інструментом тестування локальних мереж є

пристрій Fluke LanMeter. Це універсальний аналізатор, здатний навіть посилати ехо-пакети протоколу ICMP. Професійні варіанти цього обладнання описані на спеціальному сайті. Для телекомунікаційних мереж WAN найкраще підходить тестер T-BERD, що випускається компанією JDSU (jdsu.com).

Засоби налагодження другого типу – це аналізатори мережевих пакетів. Вони переглядають мережеві пакети на предмет наявності помилок протоколів, неправильної конфігурації та іншого безладу. Ці аналізатори працюють на рівні каналів, а не на електричному рівні, тому вони не можуть розпізнавати проблеми, пов'язані з фізичними пошкодженнями кабелів або електроживленням.

Існують професійні аналізатори мережевих пакетів, але є вільно розповсюджувана програма Wireshark (wireshark.org), яка може виконуватися на повнофункціональному ноутбуці. Саме її можна вважати найкращим вибором.

7.2. Обслуговування та документування мереж

Досвід показує, що зручність обслуговування мережі безпосередньо залежить від якості документації на неї. Точна, повна документація, що своєчасно коригується, абсолютно необхідна.

Кабелі слід маркувати у всіх точках підключення. Рекомендуємо вкладати копії місцевих монтажних схем у комутаційні шафи, щоб у разі всіх змін ці екземпляри можна було відкоригувати на місці. Кожні кілька тижнів необхідно переносити всі коригування в електронну базу даних.

У термінах системного адміністрування, документування означає ведення записів про те, де що знаходиться, пояснення того, що і як виконується, і забезпечення доступності корисної інформації для користувачів. Зазвичай системні адміністратори не люблять писати документацію: часу ледь вистачає на поточні завдання, навіщо ще писати документацію? Причина в тому, що документація багато в чому допомагає і її нестача знижує можливості системних адміністраторів ефективно працювати.

Буває важко вирішити, що потрібно документувати. Потрібно виходити з власних потреб: застосовуйте документацію як засіб для полегшення вашої роботи. Служба підтримки постійно завалена одними й тими самими запитаннями? Створіть документацію, щоб користувачі могли самі собі допомогти. Є завдання, які ви не любите? Внесіть їх у документацію, щоб було простіше передати їхнє вирішення кому-небудь, наприклад менш досвідченому системному адміністратору. Вам важко забути про роботу, перебуваючи у відпустці? Ви взагалі не можете дозволити собі піти у відпустку? Задокументуйте процеси, які можете виконувати тільки ви. Ви цілими днями виправляєте помилки, появі яких можна було б запобігти? Створіть інструкції та пам'ятки, щоб поліпшити відтворюваність.

Документація – це спосіб створення «пам'яті» організації, яка дає змогу команді системних адміністраторів підвищувати свій рівень знань і навичок. Вважайте документацію RAID-масивом для системних адміністраторів: вона забезпечує надмірність групи. Деякі системні адміністратори бояться, що документація зробить їх менш незамінними, і тому відмовляються документувати те, що роблять. У результаті вони зазвичай опиняються у власній пастці, загнані в кут і нездатні покинути своє робоче місце. Істина полягає в тому, що завдяки одночасному зберіганню та поширенню знань документація дає змогу організації розвивати своїх людей.

7.2.1. Що документувати

Найважливіші для документування процеси найчастіше є або складними та неприємними, або тими, які ви постійно пояснюєте комусь. Іноді предмет документування належить до обох категорій, наприклад доступ до корпоративної мережі в поїздках. Ми використовуємо характеристику «складний і неприємний» стосовно як самого процесу, так і наслідків у разі помилки. Гарним прикладом є процес підготовки робочого місця для нового співробітника: налаштування комп'ютера, створення потрібних облікових записів, зокрема необхідних у конкретному підрозділі, повідомлення потрібних людей тощо.

Таким чином, якщо процес має безліч етапів, що вимагають точного порядку виконання, особливо якщо в разі помилки вам доводиться кликати на допомогу свого керівника, є сенс задокументувати його якомога швидше. Ви позбавите кого-небудь від безлічі труднощів, і цим «ким-небудь» можете бути ви самі.

Документування нелюбимих вами завдань спрощує пошук інших людей для їх виконання. Часто найскладнішим елементом є власне розробка процесу, а потім його виконання стає простішим. При отриманні дозволу включити ще одного системного адміністратора в нашу групу нам часто хочеться найняти когось із такими ж навичками, як у нас, щоб порівну розділити роботу. Але може бути важко знайти когось, настільки ж досвідченого. Однак, якщо ми задокументували завдання, виконувати які не любимо, ми можемо найняти для їхнього виконання менш досвідчену людину, з часом навчаючи і просуваючи її. Менш досвідчений співробітник обходиться дешевше і врешті-решт стає людиною, яка знає, як працює компанія і ваш IT-підрозділ, і володіє переданими вами навичками.

Посадові інструкції (значно спрощено) зазвичай складаються з двох частин – списків обов'язків і необхідних навичок. Створіть список обов'язків, перерахувавши процеси, які ви не любите виконувати і які були задокументовані. Створіть список необхідних навичок, розглянувши кожен документ і визначивши навички та технології, з якими системний адміністратор має бути знайомий, щоб розуміти документацію. У принципі, всі посадові інструкції пишуться самі собою.

7.2.2. Простий шаблон для початку

Найскладніший етап створення документа – це початок. Ось простий принцип: визначте чотири основні елементи документа – назву, метадані, що і як. Створіть шаблон або план документа і заповніть його розділи з початку до кінця.

1. **Назва**: проста назва, зрозуміла іншим.

2. **Метадані**: контактна інформація автора документа – зазвичай ваша, дата останньої редакції або історія змін. Люди, які читають документ, зможуть звернутися до автора із запитаннями, а коли ви отримаєте підвищення, ваші наступники пам'ятатимуть вас як людину, що дала їм цей документ. Дата останньої редакції та історія змін допоможуть людям зрозуміти, чи є документ усе ще актуальним.

3. **Що**: речення, що описує зміст документа або мету, якої людина має досягти, дотримуючись вказівок. Достатньо одного-двох речень.

4. **Як**: кроки, що робляться для виконання завдання. Для кожного кроку, який може бути незрозумілим або заплутаним, ви можете додати чому, наприклад «Перемотати стрічку (Чому? Тому що ми виявили, що в цьому випадку під час

повного резервного копіювання помилки запису трапляються рідше, навіть із новими стрічками»).

Потім ретельно перевірте рівень якості документа. Точність має дуже важливе значення. Помилка або пропущений крок може призвести до того, що людина створить більше проблем, ніж передбачалося вирішити за допомогою документа.

Виконайте кроки, зазначені в документі, самостійно вводячи кожну команду. Потім попросіть кого-небудь іншого виконати завдання, використовуючи документ, і надати вам звіт, де він зустрів якісь труднощі. Дайте людині паперовий примірник, щоб вона зробила позначки, які ви потім зможете подивитися.

Хоча можна зробити це інтерактивно, спостерігаючи і записуючи за людиною її зауваження, дуже легко перетворити сеанс на бесіду, допомагаючи людині в процесі та запам'ятовуючи те, що треба буде записати потім, коли ви повернетеся за свій стіл. Будьте стриманим: якщо ви не дозволяєте людині зробити роботу самій, вона не тестує ваші інструкції. Якщо ваша присутність не викликає в людини невдоволення, сядьте поза її полем зору і спостерігайте, записуючи запитання, що викликають труднощі.

Після успішного використання цієї документації кількома співробітниками створіть на її основі більш ємне «коротке керівництво», що узагальнює кроки. Цей посібник просто допоможе досвідченим системним адміністраторам нічого не забути. У системних адміністраторів має бути можливість вирізати і вставляти командні рядки з цього документа в консоль, щоб прискорити процес.

Один зі способів спрощення ваших завдань із документування – зробити позначки під час наступного виконання завдання. Навіть якщо це сповільнить процес, це буде простіше, ніж написання документа з пам'яті.

7.2.3. Збереження скріншотів

Навчіться користуватися засобом для зняття скріншотів. Коли ви наступного разу робитимете щось, що хочете описати в документації, знімайте скріншоти під час виконання кожного кроку. Якщо ви створите документ зі скріншотів, усе, що вам залишиться, це додати один-два рядки тексту до кожного зображення, що описують, що виконується на цьому етапі. Таким чином ви отримаєте деталізований і наочний мультимедійний документ. Такий наочний посібник є чудовим для додаткової перевірки правильності, оскільки будь-хто, хто скористається документом, може на кожному етапі порівнювати скріншот із зображенням на моніторі з метою переконатися, що все має такий вигляд, як передбачалося, перш ніж іти далі.

7.2.4. Збереження вмісту командного рядка

Якщо ви частіше працюєте з командним рядком, ніж із графічним інтерфейсом, копіюйте і вставляйте в документ вміст вікна терміналу або консолі. Деякі програми терміналів або консолей містять функцію збереження у файл, UNIX-команда `script` зберігає у файл увесь сеанс.

UNIX-команда `history` повертає список останніх виконаних команд. Ці збережені команди при перетворенні в автоматизований скрипт можуть бути початковою точкою документації процесу. Документація є першим кроком до автоматизації: якщо ви не знаєте точно, як ви щось робите, ви не зможете це автоматизувати.

Комбінація команд `script` та `history` є потужним засобом. Команда `script` зберігає те, що вводите ви, і те, що виводить комп'ютер, однак може заплутати незрозумілими символами, якщо ви користуєтеся різними засобами редагування командного рядка. Команда `history` точно показує використані команди і може застосовуватися для перевірки того, що вони були записані.

Незалежно від того, чи використовуєте ви буфер обміну, чи автоматично зберігаєте результати, це краще, ніж заново набирати текст, тому що такі способи вимагають менше зусиль і відрізняються більшою точністю.

7.2.5. Зберігання документації

Створення місця для зберігання документів, або сховища документів, є важливим кроком у процесі документування. Централізоване розміщення забезпечує початкову точку для організації та оновлення документів. Зазвичай у кожного системного адміністратора є свої копії документів та особисті записи, наявність центрального сховища полегшує людям пошук останньої версії документа.

Створення сховища документів також є чудовим способом виявлення документації, про існування якої ви не знали. Якщо існує центральне сховище, люди приноситимуть різні документи, написані ними. У більшості системних адміністраторів є одна або більше директорій із довідковими документами, і вони з радістю поділяться ними, якщо з'явиться місце, де їх можна розмістити.

Найпростішим методом створення сховища документів є створення директорії з документацією на диску зі спільним доступом. Почніть її заповнення з файлу README, що описує всі правила і політики, яких необхідно дотримуватися при створенні документації для включення в дане сховище. Створіть кілька початкових піддиректорій із назвами потрібних тем, наприклад робочі станції, принтери або Linux.

У міру створення системними адміністраторами документів вносьте їх у відповідну піддиректорію, створюючи її за потребою, і використовуйте інформативні імена для файлів із документацією. Людина, яка шукає щось конкретне, може просто вивести список вмісту піддиректорії і знайти об'єкти, які її цікавлять. Наприклад, людина, яка шукає документ про додавання принтерів, може вивести список вмісту піддиректорії printers або провести у всіх піддиректоріях пошук із фільтром за ім'ям файлу, що містить слово print.

У сховищі документів корисно запровадити контроль вихідного коду, що допомагає під час пошуку більш ранніх версій документів, якщо хтось внесе зміни, які виявляться неправильними, або файл буде випадково видалено. Зазвичай набагато простіше перевірити наявність дубліката в сховищі вихідного коду, ніж відновлювати копію з резервного носія. Продукти з відкритим вихідним кодом, наприклад SubVersion, забезпечують легку організацію простого сховища.

Вебсайт може бути чудовим сховищем документів, але підтримка таблиці змісту вручну та інші завдання можуть вимагати більше роботи, ніж ви заощадите за рахунок наявності документів. Одне з рішень – налаштувати вебсервер, щоб він виводив вміст директорій. Стандартні програми виведення вмісту директорій багатьох вебсерверів підтримують імена файлів тільки певної довжини, тому довгі описові імена можуть бути обрізані. На інших вебсерверах легко керувати довжиною імені файлу, що відображається у вмісті.

Документація надає користувачам потрібну інформацію, тому вони рідше турбують системних адміністраторів, що призводить до економії часу останніх. Документація дає змогу системним адміністраторам повторювати процеси без помилок і спрощувати їх, щоб було легше передавати підтримку процесів іншим людям.

Документацію простіше створити, застосовуючи шаблон, з яким ви можете використовувати скріншоти, збережені сеанси роботи з терміналом, архіви електронної пошти та системи заявок для допомоги у створенні вмісту документів. Контрольні аркуші є гарним способом документації багатокрокових процедур, що допоможе вам повторювати їх в одній і тій самій послідовності, документувати вимоги інших груп до вас або забезпечити молодшим системним адміністраторам спосіб відмітити виконані завдання і передати цей звіт керівнику.

Процес документування є складним. Однак, щойно ви виконаєте важку роботу зі створення процесу, документацією зможуть скористатися менш поінформовані люди. Таким чином, виконання процесу стане простіше доручити комусь іншому. Наявність документації щодо процедур спрощує складання посадових інструкцій під час найму нових співробітників.

Документи мають перебувати у сховищі, щоб їх можна було підтримувати та використовувати спільно. Дуже зручною системою для створення сховищ є wiki, тому що вони спрощують створення й оновлення документів і не вимагають знання HTML. За допомогою доповнень wiki можуть надавати додаткові служби.

Людей може бути важко залучити до використання сховища документації. Надання допомоги, навчання та шаблонів знижує цей бар'єр.

Сховища можуть бути зручними не тільки для процедур. Вони можуть стати службами самостійної допомоги і містити документи HOWTO, списки FAQ, довідкову документацію та інвентарні списки.

Групи системних адміністраторів із напруженим графіком роботи завжди вітають документування процесів і поширення інформації. Гарне сховище може сприяти цьому. Документація економить час вам і всім іншим, а також дає змогу використовувати знання всіх співробітників для поліпшення системи.

7.3. Управління мережею

Якщо необхідно забезпечити нормальну роботу мережі, одні функції управління слід централізувати, інші – розподілити, а треті – залишити на локальному рівні. Потрібно сформулювати й узгодити обґрунтовані «правила поведінки добропорядних громадян».

Типове великомасштабне середовище містить у собі:

- магістральну мережу, що з'єднує будівлі;
- мережі підрозділів, під'єднані до магістралі;
- підмережі робочих груп у межах підрозділу;
- з'єднання із зовнішнім світом (наприклад, з Інтернетом або периферійними філіалами).

Під час проєктування та реалізації мереж слід передбачати централізовані контроль, відповідальність, супровід і фінансування. Оскільки підрозділи, як правило, прагнуть звести до мінімуму власні витрати, швидко зростає кількість мереж

із централізованою оплатою кожного з'єднання. Ось основні об'єкти централізованого управління:

- структура мережі, зокрема принципи використання підмереж, маршрутизаторів, комутаторів тощо;
- магістральний кабель, зокрема підключення до нього;
- IP-адреси та імена комп'ютерів, доменні імена;
- використовувані протоколи (потрібно забезпечити їхню взаємодію);
- правила доступу в Інтернет.

Імена доменів, IP-адреси та мережеві імена комп'ютерів у певному сенсі вже перебувають під централізованим контролем таких організацій, як ARIN (American Registry for Internet Numbers) та ICANN, але координація використання цих елементів на локальному рівні також необхідна.

Центральний орган управління має загальне уявлення про мережу, її структуру, продуктивність і перспективи зростання. Він може дозволити собі мати власне контрольне обладнання (і персонал, що його обслуговує) і стежити за нормальною роботою магістральної мережі. Центральний орган може наполягти на правильному виборі структури мережі, навіть якщо для цього доведеться змусити підрозділ купити маршрутизатор і створити підмережу для підключення до магістралі. Таке рішення іноді необхідне для того, щоб нове з'єднання не зашкодило роботі наявної мережі.

Стики між великими системами у вигляді комутаторів або маршрутизаторів можуть спростити налагодження, оскільки дають змогу ізолювати частини мережі та налагоджувати їх окремо. Корисно також розмежовувати адміністративні області.

Якщо в мережі працюють різнорідні комп'ютери, операційні системи та протоколи, обов'язково потрібно мати «високоінтелектуальний» маршрутизатор (наприклад, компанії Cisco), який слугуватиме шлюзом між мережами.

7.4. Система доменних імен

До Інтернету підключено безліч комп'ютерів. Як керувати ними, якщо вони розташовані в різних країнах і належать різним мережам та адміністративним групам? Цій меті служать два елементи глобальної мережевої інфраструктури: система доменних імен (Domain Name System, DNS), що відстежує інформацію про імена та адреси комп'ютерів, і система маршрутизації в Інтернеті, що контролює з'єднання між комп'ютерами.

Система доменних імен призначена для кількох цілей, але основна з них – перетворення імен комп'ютерів на IP-адреси і навпаки. Користувачам і програмам користувацького рівня зручніше посилатися на комп'ютери за іменами, але низькорівневе мережеве програмне забезпечення розуміє тільки числові адреси. Система DNS відіграє роль проміжної сполучної ланки. Вона також важлива для маршрутизації електронної пошти та організації доступу до веб-серверів.

DNS – це розподілена база даних. У цій моделі один сайт зберігає дані про відомі йому комп'ютери, інший сайт зберігає дані про свій власний набір комп'ютерів, а потім сайти співпрацюють і обмінюються даними, коли одному з них потрібно знайти дані іншого. З адміністративної точки зору, DNS-сервери, які ви налаштували для свого домену, відповідають на запити із зовнішнього світу про імена у вашому домені; вони також запитують сервери інших доменів від імені ваших користувачів.

7.4.1. Провайдери послуг DNS

Багато років тому одним з основних завдань кожного системного адміністратора було налаштування та підтримка DNS-сервера для своєї організації. Сьогодні ситуація змінилася. Якщо організація взагалі підтримує DNS-сервер, то часто він призначений лише для внутрішнього використання.

Кожна організація, як і раніше, потребує зовнішнього DNS-сервера, але зараз прийнято використовувати для цієї функції один з багатьох комерційних «керованих» DNS-провайдерів. Ці сервіси пропонують графічний інтерфейс управління та високодоступну, безпечну DNS-інфраструктуру всього за копійки (або долари) на день. Amazon Route 53, CloudFlare, GoDaddy, DNS Made Easy і Rackspace - це лише деякі з основних провайдерів.

Звичайно, ви можете налаштувати і підтримувати власний DNS-сервер (внутрішній або зовнішній), якщо бажаєте. У вас є десятки реалізацій DNS на вибір, але система Berkeley Internet Name Domain (BIND) все ще домінує в Інтернеті. Понад 75% DNS-серверів використовують ту чи іншу її форму.

Незалежно від того, який шлях ви оберете, як системний адміністратор ви повинні розуміти основні концепції та архітектуру DNS.

7.4.2. DNS для пошуку

Незалежно від того, чи використовуєте ви власний сервер імен, чи користуєтеся керованою службою DNS, чи хтось інший надає вам послуги DNS, ви, безумовно, захочете налаштувати всі свої системи на пошук імен в DNS.

Для цього потрібно зробити два кроки. По-перше, ви налаштовуєте свої системи як клієнти DNS. По-друге, ви вказуєте системам, коли використовувати DNS, а не інші методи пошуку імен, такі як статичний файл /etc/hosts.

7.4.3. resolv.conf: конфігурація клієнтського перетворювача

Кожен хост у мережі повинен бути клієнтом DNS. Ви налаштовуєте клієнтський перетворювач у файлі /etc/resolv.conf. Цей файл містить список серверів імен, до яких хост може надсилати запити.

Якщо ваш хост отримує IP-адресу та мережеві параметри від DHCP-сервера, файл /etc/resolv.conf зазвичай налаштовується автоматично. В іншому випадку ви повинні відредагувати файл вручну. Він має наступний формат

```
search domainname …
nameserver ipaddr
```

Можна вказати до трьох серверів імен. Ось повний приклад:

```
search atrust.com booklab.atrust.com
nameserver 63.173.189.1          ; ns1
nameserver 174.129.219.225       ; ns2
```

Рядок пошуку містить список доменів, до яких слід звертатися, якщо ім'я хоста не є повністю визначеним. Наприклад, якщо користувач вводить команду ssh coraline, перетворювач доповнює ім'я першим доменом у списку пошуку і шукає coraline.atrust.com. Якщо такого домену не існує, він спробує також coraline.booklab.atrust.com. Кількість доменів, які можна вказати в пошуковій директиві, залежить від конкретного вирішувача; більшість з них допускають від шести до восьми доменів з обмеженням у 256 символів.

Сервери імен, перелічені у файлі resolv.conf, мають бути налаштовані так, щоб ваш хост міг надсилати запити. Вони також повинні бути рекурсивними, тобто відповідати на запити якнайкраще, а не намагатися перенаправляти вас на інші сервери імен.

Звернення до DNS-серверів виконується в певному порядку. Поки перший продовжує відповідати на запити, інші ігноруються. Якщо виникає проблема, запит врешті-решт завершується, і відбувається спроба звернутися до наступного сервера імен. Кожен сервер пробується по черзі, до чотирьох разів. Інтервал таймауту збільшується з кожною невдачею. За замовчуванням інтервал таймауту становить п'ять секунд, що здається вічністю нетерплячим користувачам.

7.4.4. nsswitch.conf: у кого запитувати ім'я?

Як у FreeBSD, так і у Linux використовується файл-перемикач /etc/nsswitch.conf, який визначає, як слід виконувати зіставлення імені хоста з IP-адресою, і чи слід намагатися використовувати DNS першим, останнім або взагалі не використовувати його. Якщо файл перемикача відсутній, поведінка за замовчуванням буде такою

```
hosts: dns [!UNAVAIL=return] files
```

Рядок !UNAVAIL означає, що якщо DNS доступний, але ім'я в ньому не знайдено, спроба пошуку повинна завершитися невдачею, а не переходити до наступного запису (в даному випадку до файлу /etc/hosts). Якщо сервер імен не запущено (що може статися під час завантаження), процес пошуку звертатиметься до файла hosts.

Усі наші приклади дистрибутивів мають такий типовий запис у nsswitch.conf:

```
hosts: files dns
```

Ця конфігурація надає перевагу файлу /etc/hosts, який завжди перевіряється. DNS звертається тільки для імен, які неможливо вирішити за допомогою файлу /etc/hosts.

Насправді не існує найкращого способу налаштування пошуку - це залежить від того, як управляється ваш сайт. Загалом, ми вважаємо за краще зберігати якомога більше інформації про хост в DNS, але завжди зберігаємо можливість повернутися до статичного файлу hosts під час завантаження, якщо це необхідно.

Якщо служба імен надається вам сторонньою організацією, ви можете закінчити налаштування DNS після налаштування resolv.conf і nsswitch.conf.

7.4.5. Простір імен DNS

Простір імен DNS організовано у вигляді дерева, яке містить як прямі, так і зворотні відображення. Прямі відображення відображають імена хостів на IP-адреси (та інші записи), а зворотні відображення відображають IP-адреси на імена хостів. Кожне повне ім'я хоста (наприклад, nubark.atrust.com) є вузлом у прямій гілці дерева, а (теоретично) кожна IP-адреса є вузлом у зворотній гілці (рис.7.4).

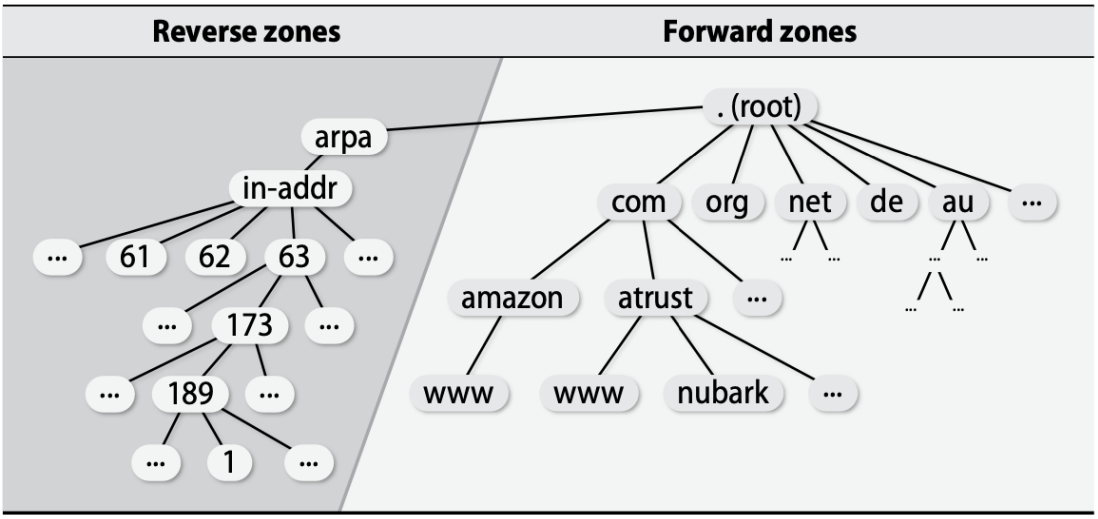


Рисунок 7.4 – Загальний вигляд дерева зон DNS

Для того, щоб одна і та ж система DNS могла керувати як іменами (які мають найбільш значущу інформацію праворуч), так і IP-адресами (які мають найбільш значущу частину ліворуч), IP-гілка простору імен інвертується шляхом перерахування октетів IP-адреси у зворотному порядку. Наприклад, якщо хост nubark.atrust.com має IP-адресу 63.173.189.1, то відповідний вузол прямої гілки дерева імен буде «nubark.atrust.com.», а вузол зворотної гілки - «1.189.173.63.in-addr.arpa.»

Обидві ці назви закінчуються крапкою, так само, як повні імена файлів завжди починаються з косої риски. Це робить їх «повними доменними іменами» або скорочено FQDN.

Поза контекстом DNS такі імена, як nubark.atrust.com (без кінцевої крапки), іноді називають «повноцінними іменами хостів», але це розмовний термін. У самій системі DNS наявність або відсутність кінцевої крапки має вирішальне значення.

Існує два типи доменів верхнього рівня: домени з кодом країни (ccTLD) і загальні домени верхнього рівня (gTLD). ICANN, Інтернет-корпорація з присвоєння імен і номерів, акредитує різні агентства для участі в проєкті спільного реєстру для реєстрації імен в gTLD, таких як com, net і org. Щоб зареєструвати ім'я ccTLD, відвідайте веб-сторінку IANA (Орган з присвоєння номерів в Інтернеті) iana.org/cctld, щоб знайти реєстр, відповідальний за реєстрацію в конкретній країні.

7.4.6. Реєстрація доменного імені

Щоб отримати доменне ім'я другого рівня (наприклад, blazedgoat.com), ви повинні подати заявку до реєстратора на відповідний домен верхнього рівня. Щоб заповнити форму реєстрації домену, ви повинні вибрати ім'я, яке ще не зайняте, і визначити технічну контактну особу, адміністративну контактну особу та принаймні два хости, які будуть серверами імен для вашого домену. Плата за реєстрацію залежить від реєстратора, але на сьогоднішній день вона, як правило, досить недорога.

7.4.7. Створення власних субдоменів

Процедура створення субдомену схожа на процедуру створення домену другого рівня, за винятком того, що центральний орган тепер локальний (або, точніше, в межах вашої організації). Конкретно, кроки виглядають наступним чином:

- Виберіть ім'я, яке є унікальним у місцевому контексті.

- Визначте два або більше хостів, які стануть серверами для вашого нового домену.

- Узгодьте це з адміністратором батьківського домену.

Батьківський домен повинен переконатися, що сервери імен дочірнього домену працюють перед виконанням делегування. Якщо сервери не працюють, це призведе до «невдалого делегування», і ви можете отримати неприємний електронний лист із проханням виправити ваші дії з DNS.

7.4.8. Як працює DNS

Сервери імен по всьому світу працюють разом, щоб відповідати на запити. Зазвичай вони поширюють інформацію, яку зберігає той адміністратор, що знаходиться найближче до місця запиту. Розуміння ролей і взаємозв'язків між серверами імен є важливим як для повсякденної роботи, так і для налагодження.

7.4.9. Сервери імен

Сервер імен виконує кілька завдань:

- Відповідає на запити про імена хостів та IP-адреси вашого сайту.

- Він запитує про локальні та віддалені хости від імені ваших користувачів.

- Він кешує відповіді на запити, щоб наступного разу відповісти швидше.

- Він взаємодіє з іншими локальними серверами імен для синхронізації даних DNS.

Сервери імен працюють із «зонами», де зона – це, по суті, домен за вирахуванням його субдоменів. Навіть у цій лекції ви часто зустрінете термін «домен», коли мається на увазі зона, хоча насправді мається на увазі зона.

Сервери імен можуть працювати в декількох різних режимах. Відмінності між ними проходять по декількох осях, тому остаточна класифікація часто не є чіткою. Щоб зробити речі ще більш заплутаними, один і той самий сервер може відігравати різні ролі щодо різних зон. На рис. 7.5 наведено деякі з прикметників, що використовуються для опису серверів імен.

Type of server	Description
authoritative	Officially represents a zone
master	The master server for a zone; gets its data from a disk file
primary	Another name for the master server
slave	Copies its data from the master
secondary	Another name for a slave server
stub	Like a slave, but copies only name server data (not host data)
distribution	A server advertised only within a domain (aka "stealth server")
nonauthoritative [a]	Answers a query from cache; doesn't know if the data is still valid
caching	Caches data from previous queries; usually has no local zones
forwarder	Performs queries on behalf of many clients; builds a large cache
recursive	Queries on your behalf until it returns either an answer or an error
nonrecursive	Refers you to another server if it can't answer a query

a. Strictly speaking, "nonauthoritative" is an attribute of a DNS query response, not a server.

Рисунок 7.5 – Класифікація серверів DNS

Ці класифікації залежать від джерела даних сервера імен (авторитетний, кешуючий, головний, підлеглий), типу збережених даних (заглушка), шляху запиту (прямий), повноти відповідей (рекурсивний, нерекурсивний) і, нарешті, видимості сервера (дистрибутивність).

7.4.10. Авторитетні сервери та сервери, що працюють лише з кешуванням

Головні, підлеглі та сервери, що лише кешують, розрізняють за двома характеристиками: звідки надходять дані, і чи є сервер авторитетним для домену. У кожній зоні зазвичай є один головний сервер імен. Головний сервер зберігає на диску офіційну копію даних зони. Системний адміністратор змінює дані зони, редагуючи файли даних головного сервера.

Підлеглий сервер отримує свої дані від головного сервера за допомогою операції «передавання зони». Зона може мати кілька підлеглих серверів імен, але принаймні один з них обов'язково повинен бути. Заглушка – це особливий тип підлеглого сервера, який завантажує тільки записи NS (сервер імен) з головного сервера. Один і той самий комп'ютер може бути як головним сервером для одних зон, так і підлеглим для інших.

Сервер імен із кешуванням завантажує адреси серверів кореневого домену зі стартового файлу, а решту даних накопичує, кешуючи відповіді на запити, які він обробляє. Сервер імен, що лише кешує, не має власних даних і не є авторитетним для жодної зони (за винятком, можливо, зони localhost).

Авторитетна відповідь від сервера імен «гарантовано» є точною; неавторитетна відповідь може бути застарілою. Однак, дуже високий відсоток неавторитетних відповідей є абсолютно правильними. Головний і підлеглий сервери є авторитетними для своїх зон, але не для інформації, яку вони можуть зберігати в кеші про інші домени. По правді кажучи, навіть авторитетні відповіді можуть бути неточними, якщо системний адміністратор змінює дані головного сервера, але забуває поширити ці зміни (наприклад, не змінює серійний номер зони).

Для кожної зони потрібен принаймні один підлеглий сервер. В ідеалі має бути щонайменше два підлеглих сервери, один з яких знаходиться в місці, що не має спільної інфраструктури з головним сервером. Локальні підлеглі повинні працювати в різних мережах і від різних джерел живлення. Коли служба імен припиняє роботу, припиняється і весь звичайний доступ до мережі.

7.4.11. Рекурсивні та нерекурсивні сервери

Сервери імен можуть бути рекурсивними або нерекурсивними. Якщо нерекурсивний сервер має відповідь на запит, закешовану з попередньої транзакції, або є авторитетним для домену, до якого відноситься запит, він надає відповідну відповідь. В іншому випадку, замість реальної відповіді, він повертає перенаправлення на авторитетні сервери іншого домену, які з більшою ймовірністю знають відповідь. Клієнт нерекурсивного сервера повинен бути готовий приймати реферали і діяти за ними.

Хоча нерекурсивні сервери можуть здатися лінивими, вони зазвичай мають вагомі причини не брати на себе зайву роботу. Нерекурсивними є лише авторитетні сервери (наприклад, кореневі сервери та сервери доменів верхнього рівня), але

оскільки вони можуть обробляти десятки тисяч запитів на секунду, ми можемо пробачити їм деяку ощадливість.

Рекурсивний сервер повертає лише реальні відповіді та повідомлення про помилки. Він сам відстежує реферали, звільняючи клієнтів від цього обов'язку. В інших аспектах базова процедура обробки запиту практично не відрізняється.

З міркувань безпеки, зовнішні сервери імен організації завжди повинні бути нерекурсивними. Рекурсивні сервери імен, видимі для всього світу, можуть бути вразливими до атак отруєння кешу.

Зверніть увагу: бібліотеки перетворювачів не розуміють перенаправлення. Будь-який локальний сервер імен, перелічений у клієнтському файлі resolv.conf, має бути рекурсивним.

7.4.12. Записи ресурсів

Кожен сайт підтримує одну або кілька частин розподіленої бази даних, яка складає всесвітню систему DNS. Ваша частина бази даних складається з текстових файлів, які містять записи для кожного з ваших хостів; вони відомі як «записи ресурсів». Кожен запис - це один рядок, що складається з імені (зазвичай імені хоста), типу запису і деяких значень даних. Поле імені можна опустити, якщо його значення збігається зі значенням попереднього рядка.

Наприклад, рядки

```
nubark              IN     A        63.173.189.1
                    IN     MX       10 mailserver.atrust.com.
```

у файлі «forward» (який називається atrust.com), і рядок

```
1                   IN     PTR      nubark.atrust.com.
```

у «зворотному» файлі (під назвою 63.173.189.rev) пов'язує nubark.atrust.com з IP-адресою 63.173.189.1. Запис MX перенаправляє електронну пошту, адресовану цьому комп'ютеру, на хост mailserver.atrust.com.

Поля IN позначають класи записів. На практиці це поле завжди має значення IN для Internet.

Записи ресурсів є мовою DNS і не залежать від конфігураційних файлів, які керують роботою конкретної реалізації DNS-сервера. Вони також є фрагментами даних, які циркулюють по системі DNS і кешуються в різних місцях.

7.4.13. Делегування

Усі сервери імен зчитують ідентифікатори кореневих серверів з локального конфігураційного файлу або вбудовують їх у код. Кореневі сервери знають сервери імен для com, net, edu, fi, de та інших доменів верхнього рівня. Далі по ланцюжку edu знає про colorado.edu, berkeley.edu і так далі. Кожен домен може делегувати повноваження для своїх субдоменів іншим серверам.

Давайте розглянемо реальний приклад. Припустимо, ми хочемо дізнатися адресу машини vangogh.cs.berkeley.edu з машини lair.cs.colorado.edu. Хост lair запитує свій локальний сервер імен ns.cs.colorado.edu, щоб дізнатися відповідь. На наступній ілюстрації (рис. 7.6) показано подальші події.

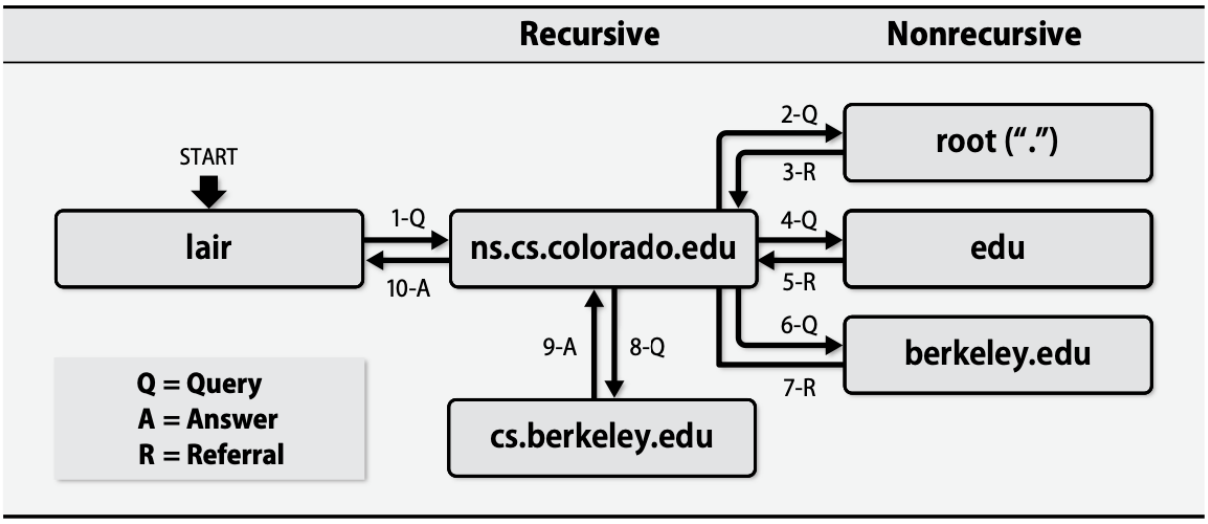


Рисунок 7.6 – Процес DNS-запиту для vangogh.cs.berkeley.edu

Числа на стрілках між серверами показують порядок подій, а буква позначає тип транзакції (запит, перенаправлення або відповідь). Ми припускаємо, що ніяка необхідна інформація не була закешована перед запитом, окрім імен та IP-адрес серверів кореневого домену.

Локальний сервер не знає адреси vangogh. Насправді, він нічого не знає про cs.berkeley.edu, berkeley.edu чи навіть edu. Однак він знає сервери кореневого домену, тому він запитує кореневий сервер про vangogh.cs.berkeley.edu і отримує перенаправлення на сервери для edu.

Локальний сервер імен є рекурсивним сервером. Коли відповідь на запит складається з перенаправлення на інший сервер, локальний сервер повторно надсилає запит на новий сервер. Він продовжує слідувати за перенаправленнями, поки не знайде сервер, на якому є дані, які він шукає.

У цьому випадку локальний сервер імен надсилає свій запит на сервер домену edu (запитуючи, як завжди, про vangogh.cs.berkeley.edu) і отримує назад перенаправлення на сервери для berkeley.edu. Потім локальний сервер імен повторює той самий запит на сервері berkeley.edu. Якщо на сервері Берклі немає кешованої відповіді, він повертає перенаправлення на сервери для cs.berkeley.edu. Сервер cs.berkeley.edu є авторитетним для запитуваної інформації, шукає відповідь у своїх зональних файлах і повертає адресу vangogh.

Коли пил вляжеться, ns.cs.colorado.edu зберігає адресу vangogh'a в кеші. Він також кешує дані на серверах edu, berkeley.edu і cs.berkeley.edu.

Ви можете детально переглянути процес виконання запиту за допомогою `dig +trace` або `drill -T`.

7.4.14. Кешування та ефективність

Кешування підвищує ефективність пошуку: кешована відповідь майже безкоштовна і, як правило, правильна, оскільки зіставлення імен хостів з адресами змінюються рідко. Відповідь зберігається протягом періоду часу, який називається «час життя» (TTL) і визначається власником відповідного запису даних.

Більшість запитів стосуються локальних хостів і можуть бути вирішені швидко. Користувачі також ненавмисно допомагають підвищити ефективність, оскільки вони

повторюють багато запитів; після першого запиту повторні запити більш-менш безкоштовні.

За нормальних умов, записи про ресурси вашого сайту повинні використовувати TTL, який становить десь від години до доби. Чим довший TTL, тим менше мережевого трафіку буде спожито інтернет-клієнтами, які отримують свіжі копії записів.

Якщо у вас є певний сервіс, який розподіляє навантаження між логічними підмережами (часто його називають «глобальним балансуванням навантаження на сервер»), ваш постачальник послуг з балансування навантаження може вимагати, щоб ви вибрали короткий TTL, наприклад, 10 секунд або 1 хвилину. Короткий TTL дозволяє балансувальнику навантаження

балансувальнику навантаження швидко реагувати на непрацюючі сервери та атаки на відмову в обслуговуванні. Система все ще працює коректно з коротким TTL, але вашим серверам імен доводиться багато працювати.

У наведеному вище прикладі vangogh TTL становив 42 дні для коренів, 2 дні для edu, 2 дні для berkeley.edu і 1 день для vangogh.cs.berkeley.edu. Це розумні показники. Якщо ви плануєте масову перенумерацію, змініть TTL на коротше значення задовго до початку.

DNS-сервери також реалізують негативне кешування. Це означає, що вони запам'ятовують, коли запит зазнав невдачі, і не повторюють його, доки не закінчиться значення TTL негативного кешування. Негативне кешування потенційно може зберігати відповіді наступних типів:

- Жоден хост або домен не відповідає запитуваному імені.
- Запитуваний тип даних не існує для цього хоста.
- Сервер не відповідає.
- Сервер недоступний через мережеві проблеми.

Реалізація BIND кешує перші два типи негативних даних і дозволяє налаштовувати час кешування негативних даних.

Загалом тема DNS досить обширна й вимагає вивчення значного обсягу спеціалізованої літератури, тому для нашої дисципліни (для ознайомлення з DNS) поки що зупинимось на досягнутому.

ЛЕКЦІЯ 8. Безпека

Комп'ютерна безпека перебуває в жалюгідному стані. На відміну від прогресу, який спостерігається практично в кожній іншій сфері комп'ютерних технологій, недоліки безпеки стають дедалі серйознішими, а наслідки недостатнього рівня безпеки – більш тяжкими. Проблеми комп'ютерної безпеки безпосередньо впливають і загрожують суспільствам у всьому світі.

Нагадаємо вам кілька подій у сфері комп'ютерної безпеки, що сталися з часу останнім часом:

- Витончений хробак Stuxnet, виявлений у 2010 році, атакував ядерну програму Ірану, пошкодивши центрифуги на заводі зі збагачення урану.

- У 2013 році Едвард Сноуден викрив масивну машину стеження АНБ, показавши, що деякі великі інтернет-компанії були співучасниками того, що дозволяли уряду шпигувати за громадянами США.

- Приблизно в 2013 році з'явився новий тип атак, відомий як програми-вимагачі. Зловмисники компрометують цільову систему і шифрують її дані, тримаючи її в заручниках. Жертви повинні заплатити викуп за їх відновлення. Вони часто так і роблять.

- У 2015 році було зламано систему Управління кадрового менеджменту США, що призвело до витоку конфіденційних і приватних даних понад 21 мільйона американських громадян, багато з яких мали допуск до державної таємниці.

- У 2016 році російські державні хакери, як стверджується, організували кампанію з метою вплинути на результат президентських виборів у США.

- У 2017 році безпрецедентна атака вірусу-здирника захопила системи Windows у понад 150 країнах. Атака використовувала експлойт, розроблений АНБ.

Ставки ще ніколи не були такими високими. Ми вважаємо, що ситуація ще погіршиться, перш ніж покращиться.

Частково проблема полягає в тому, що проблеми безпеки не є суто технічними. Ви не можете вирішити їх, купивши певний продукт або послугу у третьої сторони. Досягнення прийнятного рівня безпеки вимагає терпіння, пильності, знань і наполегливості – не лише від вас та інших системних адміністраторів, але й від усіх користувачів та керівництва.

Як системний адміністратор ви несете важкий тягар. Ви повинні просувати порядок денний, який забезпечить безпеку систем і мереж вашої організації, гарантувати їх пильний моніторинг та належне навчання користувачів і персоналу. Ознайомтеся з сучасними технологіями безпеки та співпрацюйте з експертами для виявлення та усунення вразливостей на вашому сайті. Міркування безпеки повинні бути частиною кожного рішення.

Дотримуйтесь балансу між безпекою та зручністю використання. Пам'ятайте, що

$$\text{Security} = \frac{1}{(1.072)(\text{Convenience})}$$

Чим більше заходів безпеки ви впроваджуєте, тим більше ви та ваші користувачі будете обмежені. Впроваджуйте заходи безпеки, запропоновані у цій главі, лише після ретельного обмірковування наслідків для ваших користувачів.

Чи безпечна UNIX? Звичайно, ні. UNIX і Linux не є безпечними, як і будь-яка інша операційна система, яка взаємодіє у мережі. Якщо вам потрібна абсолютна, повна, недосяжна безпека, то вам потрібен вимірюваний повітряний проміжок між вашим комп'ютером і будь-яким іншим пристроєм. Деякі люди стверджують, що вам також потрібно закрити комп'ютер у спеціальному приміщенні, яке блокує електромагнітне випромінювання (пошукайте в інтернеті «клітка Фарадея»). Хіба це весело?

У цій лекції розглядається складна сфера комп'ютерної безпеки: джерела загроз, основні способи захисту систем, інструменти професії та джерела додаткової інформації.

8.1. Слабкі місця у системі захисту

Далі розглядатимуться деякі найпоширеніші проблеми, пов'язані з безпекою, і стандартні контрзаходи, що дають змогу уникнути цих проблем. Але перш ніж зануритися в деталі, необхідно поглянути на ситуацію загалом і визначити основні джерела неприємностей.

8.1.1. Людський фактор

Користувачі (і адміністратори) системи часто є її найслабшою ланкою. Навіть зараз, коли освіченість людей у сфері безпеки підвищилася, безтурботні користувачі легко поширюють конфіденційну інформацію. Жодна технологія не може захистити мережу від необережних користувачів, тому системний адміністратор зобов'язаний інформувати користувачів про наявні загрози, і ця обставина стане частиною системи безпеки.

Ця проблема може проявлятися в різних формах. Хакери телефонують своїм жертвам і представляються легальними користувачами, які потрапили в скрутне становище, намагаючись проникнути в мережу. Іноді самі системні адміністратори необережно розміщують конфіденційну інформацію на відкритих форумах, описуючи вирішення складних проблем. Фізичні зломи виникають, наприклад, коли зовні легальні співробітники відділу технічного обслуговування проникають у телефонну монтажну шафу.

Терміном «фішинг» (phishing) називають спробу збору інформації від користувачів за допомогою обманних електронних повідомлень, миттєвих повідомлень і навіть повідомлень SMS. Від фішингу особливо важко захиститися, оскільки повідомлення часто містять інформацію, яка свідчить про знайомство з жертвою та створює видимість автентичності.

Використання людського фактора залишається потужним хакерським прийомом, якому важко протистояти. Ваша стратегія забезпечення безпеки має передбачати навчання нових співробітників. Ефективно також поширювати серед співробітників інформацію про телефонні атаки, фізичні проблеми, фішинг і правильний вибір паролів.

Може здатися, що для того, щоб протистояти спробам використати людський фактор, можна спробувати самому спровокувати таку атаку. Однак спочатку необхідно отримати дозвіл від менеджера. Якщо у вас не буде такого дозволу, такі дії виглядатимуть підозріло. Крім того, вони можуть викликати «шпигуноманію» й образу.

Багато організацій вважають за необхідне повідомити своїм користувачам, що адміністратори за жодних обставин не мають права запитувати в них паролі ні через електронні повідомлення, ні через миттєві повідомлення, ні телефоном. Про будь-яку спробу дізнатися пароль за допомогою цих засобів слід повідомляти у відділ інформаційних технологій.

8.1.2. Помилки в програмах

За багато років у програмному забезпеченні (включно зі сторонніми програмами, як комерційними, так і безплатними) було виявлено незліченну кількість помилок, пов'язаних із безпекою. Використовуючи непомітні програмістські прорахунки або контекстні залежності, хакерам вдавалося маніпулювати системою на свій розсуд.

Основною програмною помилкою, однією з найнебезпечніших за своїми наслідками, є переповнення буфера. Для зберігання інформації розробники часто виділяють заздалегідь визначений обсяг тимчасової пам'яті, який називається буфером. Якщо програма не стежить за розмірами даних і контейнера, у якому вони мають зберігатися, то пам'ять, суміжна з виділеною областю, ризикує виявитися перезаписаною. Умілі хакери можуть ввести ретельно скомпоновані дані так, щоб вони призвели до краху програми або, у найгіршому разі, виконали якийсь заданий код.

На щастя, величезна кількість випадків переповнення буферів, яка спостерігалася останніми роками, навчила програмістів боротися з цим. Незважаючи на те що переповнення буфера залишається складною проблемою, її швидко розпізнають і виправляють, особливо в додатках із відкритим кодом. Новітні програмні системи, такі як Java і .Net, містять механізми, що автоматично перевіряють розміри даних і запобігають переповненню буфера (щоправда, не завжди).

Переповнення буфера є окремим випадком ширшого класу проблем, пов'язаних із безпекою програмного забезпечення, що має назву вразливість перевірки значень, що вводяться (input validation vulnerability). Майже всі програми приймають від користувачів дані, що вводяться (наприклад, аргументи командного рядка або форми HTML). Якщо програма обробляє ці дані без суворої перевірки відповідного формату та змісту, то можуть виникнути неприємності. Розглянемо такий простий приклад.

```
#!/usr/bin/perl
# Example user input validation error

open(HTMLFILE,"/var/www/html/$ARGV[0]") or die "trying\n";
while(<HTMLFILE>) {print; }
close HTMLFILE;
```

Імовірно, цей код має виводити на друк вміст якогось HTML-файлу, що міститься в каталозі /var/www/html, який за замовчуванням є кореневим каталогом документів для веб-сервера Apache серед серверів системи Red Hat. Цей код отримує від користувача ім'я файлу і включає його як частину аргументу команди open. Однак зловмисник ввів як аргумент рядок ../../../../etc/passwd і отримав як відповідь вміст цього файлу!

Як адміністратор може запобігти такому типу атак? Майже ніяк, принаймні, доки не буде знайдено помилку і створено відповідну заплатку. З цієї причини випуск патчів і бюлетенів, присвячених питанням безпеки, є важливою частиною роботи системного адміністратора. Більшість дистрибутивних пакетів системи Linux містить автоматизовані утиліти для створення патчів, як-от yum у системі Red Hat і apt-get у системі Ubuntu. Система OpenSolaris також має автоматизовані (і надійні) модифікації, реалізовані за допомогою команди pkg image-update. Скористайтеся перевагами цих утиліт, щоб уберегти ваш сайт від небезпек.

8.1.3. Помилки конфігурації

Багато компонентів програмного забезпечення можна конфігурувати в режимі повної або часткової безпеки. На жаль, оскільки програмне забезпечення має бути корисним і не повинно дратувати користувачів, за замовчуванням найчастіше прийнято другий варіант. Хакери вдираються в системи, єзуїтськи експлуатуючи функціональні можливості, які надані розробниками для зручності користувачів: облікові записи без паролів, глобальний спільний доступ до жорстких дисків тощо.

Один із типових прикладів уразливості вузлів є стандартна практика, коли системи сімейства Linux завантажуються, не вимагаючи введення пароля завантажувача. Завантажувач операційної системи GRUB можна налаштувати так, щоб він під час інсталяції запитував пароль, але адміністратори часто відмовляються від цієї можливості. Це упущення відкриває можливість для фізичної атаки системи. Однак цей же захід демонструє необхідність балансування між безпекою та зручністю. Запит пароля означає, що в разі мимовільного перезавантаження системи (наприклад, у разі збою електроживлення) адміністратор має бути фізично присутнім під час повторного ввімкнення комп'ютера.

Одне з найважливіших завдань, пов'язаних із забезпеченням безпеки системи, - переконатися в тому, що, дбаючи про благополуччя користувачів, ви випадково не відкрили двері для хакерів. Такі проблеми простіше виявити й усунути, ніж інші, хоча їх може бути дуже багато і не завжди очевидно, що саме слід перевіряти. Сканування портів і вразливих місць може допомогти зацікавленому адміністратору виявити проблему ще до її виникнення.

8.2. Ключові аспекти безпеки

У цій лекції розглядається безліч аспектів комп'ютерної безпеки. В ідеалі їх усі потрібно враховувати. Більшість операційних систем не постачається з уже готовим захистом. Крім того, налаштування, яке здійснюється як під час інсталяції, так і після неї, змінює профіль безпеки нових систем. Адміністратори повинні зміцнювати нові системи, інтегрувати в локальне середовище і планувати їх довготривалу безпечну експлуатацію.

Коли до вас прийде перевірка, ви маєте довести, що дотримувалися стандартної методології, особливо якщо ця методологія відповідає загальноприйнятим рекомендаціям і передовим досягненням у вашій галузі промисловості.

Для захисту нових систем ми використовуємо технологічну карту. Системний адміністратор застосовує стандартні заходи зі зміцнення безпеки системи, а

адміністратор з питань безпеки перевіряє, щоб ці заходи було вжито правильно, і веде системний журнал.

8.2.1. Програмні «латки»

Оснащення операційної системи новорозробленими програмними «латками» є найпершим обов'язком системного адміністратора. Більшість систем конфігурується так, щоб мати зв'язок із репозиторієм постачальника. У цьому випадку встановлення програмної «латки» зводиться всього лише до виконання кількох команд. У великих мережах можуть існувати локальні сховища, які є дзеркалами репозиторію постачальника.

Під час встановлення програмних «латок» слід враховувати таке.

- Потрібно скласти розклад інсталяції програмних «латок» і суворо його дотримуватися. Під час складання такого розкладу необхідно враховувати його вплив на користувачів. Зазвичай достатньо виконувати щомісячні оновлення; регулярність важливіша за терміновість. Не можна зосереджуватися тільки на захисті від шкідливих програм та ігнорувати інші оновлення.

- Необхідно розробити план змін, який враховував би вплив кожного набору «латок», передбачав відповідні етапи тестування після ін сталяції системи та описував можливості відмови від внесених оновлень у разі виникнення проблем. Цей план слід обговорити з усіма зацікавленими сторонами.

- Необхідно розуміти, які патчі підходять до даного оточення. Системні адміністратори повинні підписуватися на спеціальні списки розсилки і блоки з питань безпеки, що створюються постачальниками, а також брати участь у роботі спеціалізованих форумів з питань безпеки.

8.2.2. Непотрібні служби

Більшість систем постачається з кількома службами, які запускаються за замовчуванням. Вимкніть (і за можливості видаліть) усі непотрібні служби, особливо якщо вони реалізовані у вигляді мережевих демонів. Для того щоб виявити виконувані служби, можна використовувати команду netstat.

Для виявлення служб, що використовують невідомий порт, існує кілька прийомів. У більшості операційних систем це завдання можна вирішити за допомогою команд lsof або fuser. У системі Linux ці команди можуть розпізнати ідентифікатор PID процесу, що використовує цей порт.

З'ясувавши ідентифікатор PID, за допомогою команди ps можна ідентифікувати конкретний процес. Якщо ця служба не потрібна, зупиніть її і переконайтеся, що вона не буде заново стартувати під час завантаження системи.

На жаль, доступність команд lsof і fuser залежить від системи, і їхні реалізації сильно відрізняються одна від одної. Багатьом версіям цих інструментів бракує підтримки мережевих сокетів.

Якщо утиліти lsof і fuser недоступні (або марні), можна або перевірити «добре відомі» порти для служб у файлі /etc/service, або запустити утиліту netstat без параметра -n, щоб вона виконала цю перевірку за вас.

Деяким мережевим протоколам притаманний ризик, якому піддається безпека системи. У цьому разі вона залишається майже весь час вразливою. Наприклад, програми FTP, Telnet і BSD «r» (rcp, rlogin і rsh) використовують ненадійну

автентифікацію і ненадійні методи передачі даних. Їх потрібно вимкнути в усіх системах і замінити безпечнішими програмами, наприклад програмою SSH.

8.2.3. Віддалена реєстрація подій

Механізм syslog надсилає реєстраційну інформацію у файли, списки користувачів та інші вузли мережі. Спробуйте налаштувати вузол, який відповідає за безпеку, так, щоб він діяв як центральна реєстраційна машина, яка аналізує отримані події та виконує відповідні дії. Єдиний централізований реєстратор подій може отримувати повідомлення від різних пристроїв і попереджати адміністраторів про важливі події. Віддалена реєстрація також запобігає перезапису або очищенню реєстраційних файлів у зламаних системах.

Більшість систем постачають сконфігурованими так, що механізм syslog увімкнено за замовчуванням, але для налаштування віддаленої реєстрації необхідно виконати додаткове конфігурування.

8.2.4. Резервні копії

Регулярне резервне копіювання є важливою частиною системи безпеки будь-якої мережі. Воно входить до так званої тріади КЦД і належить до категорії «доступність». Переконайтеся, що всі частини системи регулярно копіюються і що ця інформація зберігається за межами сайту. Якщо станеться серйозний інцидент, пов'язаний із безпекою мережі, ви зможете скористатися «надійним» джерелом інформації та відновити роботу.

Резервне копіювання саме по собі також може створювати загрозу безпеці. Вкрадена колекція магнітних стрічок може знищити всю систему безпеки. Для зберігання магнітних стрічок використовуйте вогнетривкі сейфи, які можуть протистояти злому, і застосовуйте шифрування. Розглядаючи можливість укладення контракту на зберігання інформації, попросіть дозволу здійснити екскурсію й особисто переконатися в надійності сховища.

8.2.5. Віруси та хробаки

Системи UNIX і Linux мають імунітет від вірусів. Існує всього кілька вірусів (більшість із них має суто академічний характер), які можуть заразити системи UNIX і Linux, але жоден із них не здатний завдати серйозної шкоди, на відміну від середовища Windows, де це стало частим явищем. Проте цей факт не знижує стурбованості постачальників антивірусних програм – зрозуміло, якщо ви купуєте їхній продукт за спеціальною ціною.

Точна причина відсутності шкідливих програм неясна. Деякі стверджують, що система UNIX просто займає занадто незначну частку ринку настільних систем і тому не цікавить авторів вірусів. Інші наполягають на тому, що середовище з контролем доступу, яке є в системі UNIX, обмежує розмір шкоди від хробаків, що саморозповсюджуються, або вірусів. Останній аргумент заслуговує на увагу. Оскільки система UNIX обмежує доступ виконуваних модулів на файловому рівні, непривілейовані користувачі не можуть інфікувати решту середовища. Якщо код вірусу було запущено не з кореня, його шкідливість буде значно обмежена. Отже, не потрібно використовувати кореневий обліковий запис для повсякденної діяльності.

Як не дивно, одна з причин впровадження антивірусного програмного забезпечення на серверах UNIX – прагнення захистити комп'ютери, що працюють під управлінням Windows, які існують у вашій мережі, від Windows-орієнтованих вірусів. Поштовий сервер може сканувати вхідну електронну пошту на віруси, а файловий сервер у пошуках інфекції може сканувати загальні файли. Однак це рішення має бути додатковим щодо антивірусного захисту настільних систем, а не основним.

8.2.6. Троянські програми

Це програми, які не мають вигляду програм. Прикладом є програма turkey, що поширювалася в мережі Usenet багато років тому. Ця програма стверджувала, що здатна намалювати індичку на екрані комп'ютера, але насправді видаляла файли з робочого каталогу.

Фрагменти троянських програм зустрічаються в основних пакетах програмного забезпечення й досі. Автори програм sendmail, tcpdump, OpenSSH і nterBase навіть опублікували поради щодо шкідливих програм, впроваджених у їхнє програмне забезпечення. Ці троянські програми зазвичай впроваджують шкідливий код, який дозволяє хакерам проникати в систему. На щастя, більшість постачальників регулярно виправляють свої помилки і публікують відповідні рекомендації раз на один-два тижні. Слідкуйте за списками розсилок з питань безпеки, присвяченими всім мережевим програмним пакетам, які виконуються на ваших вузлах.

Незважаючи на велику кількість проблем, що виникли в галузі безпеки систем UNIX за останні кілька років, слід зазначити малу кількість інцидентів, пов'язаних із троянськими програмами. Здебільшого це пояснюється швидкістю комунікацій через Інтернет. Очевидні проблеми з безпекою швидко виявляються і широко обговорюються. Шкідливі пакети не затримуються надовго на добре відомих інтернет-серверах.

Ви можете бути впевнені, що будь-яке виявлене шкідливе програмне забезпечення викличе в Інтернеті широке хвилювання. Пошукайте в системі Google ім'я програмного пакета, перш ніж його інсталювати, і подивіться, чи немає на першій сторінці неприємних повідомлень про нього.

8.2.7. Руткіти

Умілі хакери намагаються приховувати свої сліди та уникати викриття. Часто вони розраховують продовжити використання вашої системи для нелегального розповсюдження програмного забезпечення, зондування інших мереж або запуску атаки проти інших мереж. Для того щоб залишатися непоміченими, вони часто використовують «Руткіти» («rootkits»). Троянські програми, впроваджені у файли, створені фірмою Sony, використовують можливості руткітів, щоб приховати себе від користувачів.

Руткіти – це програми та латки, що приховують важливу системну інформацію, наприклад процес, диск або мережеву активність. Вони мають багато облич і різний ступінь складності – від простих замін додатка (як зламані версії утиліт ls і ps) до модулів ядра, які практично неможливо виявити.

Для ефективного моніторингу системи та виявлення впроваджених руткітів існує спеціальне програмне забезпечення, наприклад OSSEC. Крім того, розроблено

сценарії розпізнавання руткітів (такі, як chkrootkit, chkrootkit.org), що сканують систему в пошуках відомих руткітів.

Незважаючи на існування програм, що дають змогу системним адміністраторам видаляти руткіти зі зламаних систем, час, який вони змушені витрачати на очищення систем, можна було б заощадити, зберігши дані, переформатувавши диск і почавши роботу з нуля. Більшість сучасних руткітів знають про існування програм для їх видалення і намагаються чинити опір.

8.2.8. Фільтрація пакетів

Якщо система під'єднується до мережі, де є вихід в Інтернет, необхідно, щоб між цією системою і зовнішнім світом стояв брандмауер або маршрутизатор, що фільтрує пакети. Як альтернативу можна ввімкнути фільтрацію пакетів у ядрі. Якою б не була реалізація, фільтр має пропускати через себе трафік тільки найважливіших служб, що виконують «корисну» роботу в мережі.

8.2.9. Паролі

Усі ми любимо прості правила. Ось одне з них: у кожного облікового запису має бути пароль, який важко вгадати. Але навіть найкращі паролі не можна передавати через Інтернет у текстовому вигляді. Тому, якщо в системі допускається віддалена реєстрація, слід застосовувати SSH або який-небудь інший механізм аутентифікації.

8.2.10. Пильність

Для того щоб бути впевненим у безпеці системи, стежте за її станом, мережевими з'єднаннями, таблицею процесів. Робити це потрібно регулярно (бажано щодня). Проблема завжди починається з малого, а потім наростає, як снігова куля, тож що раніше буде виявлено аномалію, то меншим виявиться збиток.

8.2.11. Загальні принципи захисту

Ефективна система безпеки повинна базуватися на здоровому глузді. Вона багато в чому нагадує боротьбу з мишами в будинку. Ось низка правил, яких при цьому слід дотримуватися.

Цими ж правилами (у комп'ютерному варіанті) можна з успіхом керуватися під час організації захисту Linux-систем. Ось як вони звучать стосовно Linux.

- Не залишайте без нагляду файли, які можуть становити інтерес для хакерів і не в міру цікавих товаришів по службі. Комерційні таємниці, персональні досьє, бухгалтерські відомості, результати виборів тощо – за всім цим потрібен нагляд. Набагато надійніше зашифрувати дані, ніж просто намагатися запобігти несанкціонованому доступу до них.
- В організації має існувати порядок роботи з конфіденційною інформацією.
- Намагайтеся, щоб у системі не було місць, де хакери могли б закріпитися. Хакери часто вламуються в одну систему, а потім використовують її як базу для злому інших систем. Іноді хакери використовують вашу мережу для приховування своїх слідів під час атаки реальної мети. Вразливі служби з відкритим доступом, ка талоги,

доступні для запису за протоколом FTP в анонімному режимі, групові облікові записи, облікові записи з погано підібраними паролями – ось основні вразливі місця.

- Встановлюйте пастки для виявлення вторгнень або спроб вторгнення. Такі інструменти, як OSSEC, Bro, Snort і John the Ripper, допоможуть попередити можливі проблеми.
- Слідкуйте за звітами, які генеруються цими утилітами. Незначна проблема, проігнорована у звіті, до моменту отримання наступного звіту може перерости в катастрофу.
- Вчіться захищати системи своїми силами. Застосовуйте традиційні технології, навчайте користувачів, керуйтеся, зрештою, здоровим глуздом. У разі потреби запрошуйте фахівців зі сторони, але при цьому ретельно контролюйте їхню роботу.
- Постійно стежте за тим, чи не з'явилися відхилення від нормального ходу роботи системи. Звертайте увагу на все незвичне, наприклад на незрозумілі журнальні повідомлення або зміну характеру використання будь-якого облікового запису (різке зростання активності, робота в незвичний час, використання облікового запису під час відпустки його власника).

8.3. Паролі та облікові записи користувачів

Дуже часто джерелом неприємностей є погане управління паролями. За замовчуванням у файлах /etc/passwd і /etc/shadow містяться дані про те, хто може входити в систему і що він при цьому має право в ній робити. Ці файли являють собою передову лінію захисту системи від загарбників. Їх потрібно вести з особливою ретельністю, намагаючись не допускати помилок і не захаращувати файли застарілими даними.

Система UNIX дає змогу користувачам обирати свої власні паролі, але, незважаючи на те, що це зручно, через це виникає безліч проблем, пов'язаних із безпекою. Коли ви даєте користувачам їхні реєстраційні імена, маєте також інструктувати їх про правила вибору паролів. Рекомендується обирати паролі не менше ніж із восьми символів, серед яких мають бути цифри, розділові знаки, а також великі та малі літери. Безглузді поєднання символів, складів, перші літери слів фрази, що легко запам'ятовується, – ось найкращі паролі. При цьому фраза, що легко запам'ятовується, не повинна бути однією з широко поширених. Краще придумати свою власну.

Важливо постійно перевіряти (бажано щодня), щоб кожен реєстраційний запис мав пароль. Записи у файлі /etc/shadow, що описують псевдокористувачів, таких як «демон», які володіють файлами, але ніколи не реєструються, мають бути позначені зірочками або знаком оклику в полі зашифрованого пароля. Ці символи не збігаються з жодним паролем і тим самим запобігають використанню цього облікового запису.

В організаціях, що використовують централізовану схему автентифікації, таку як LDAP або Active Directory, використовується та сама логіка. Вони вимагають складних паролів і блокують облікові записи після кількох невдалих спроб реєстрації.

8.3.1. Застарівання паролів

У більшості систем, що використовують тіньові паролі, можна реалізувати механізм так званого застарівання паролів, за якого користувачів змушують

періодично змінювати паролі. На перший погляд, це гарна ідея, однак її практична реалізація тягне за собою певні проблеми. Не всякому користувачеві до душі періодично змінювати пароль, оскільки це пов'язано із запам'ятовуванням нового пароля. Зазвичай для пароля вибирають просте слово, яке легко вводять і запам'ятовують, і коли приходить час заміни, багато користувачів, не бажаючи себе обтяжувати, знову беруть попередній пароль. Таким чином, дискредитується сама ідея. Модулі PAM можуть допомогти вибрати сильні паролі, щоб уникнути описаної вище ситуації.

У системі Linux процесом старіння паролів керує програма chage. З її допомогою адміністратори можуть задавати мінімальну і максимальну кількість змін пароля, дату закінчення терміну дії пароля, кількість днів до настання дати закінчення терміну дії пароля, коли слід завчасно попередити користувача, кількість днів простою, протягом яких облікові записи залишаються заблокованими, та інші параметри. Наступна команда задає мінімальну кількість днів між змінами пароля рівною 2, максимальну кількість змін пароля рівною 90, дату закінчення терміну дії пароля рівною 31 липня 2010 року, а також те, що користувача слід попередити про закінчення терміну дії пароля за 14 днів.

```
$ sudo chage -m 2 -M 90 -E 2010-07-31 -W 14 ben
```

8.3.2. Групові та спільно використовувані облікові записи

Небезпечно, якщо обліковий запис використовується кількома людьми. Групові реєстраційні імена (наприклад, guest або demo) являють собою зручну лазівку для хакерів, тому вони заборонені в багатьох мережах федеральними законами, такими як HIPAA. Не допускайте цього у своїй мережі. Однак технічні засоби не можуть запобігти спільному використанню користувачами паролів, тому в цьому питанні найкраще вести роз'яснювальну роботу.

8.3.3. Користувацькі оболонки

Теоретично можна встановити як оболонку для користувацького облікового запису будь-яку програму, включно з користувацьким сценарієм. На практиці використання оболонок, що відрізняються від стандартних, таких як bash і tcsh, дуже небезпечне. Ще небезпечніші безпарольні реєстраційні імена, оболонкою яких є сценарій. Якщо у вас виникне спокуса створити таке реєстраційне ім'я, застосуйте замість нього пару ключів SSH без пароля.

8.3.4. Привілейовані облікові записи

Єдина відмінна риса користувача root полягає в тому, що його ідентифікатор дорівнює 0. Оскільки у файлі /etc/passwd може бути кілька записів із таким ідентифікатором, існує й кілька способів входу в систему як суперкористувач.

Один зі способів, який хакери, отримавши доступ до інтерпретатора команд суперкористувача, широко застосовують для відкриття «чорного ходу», – редагування файлу /etc/passwd. Оскільки такі команди, як `who` і `w`, працюють з реєстраційним ім'ям, що зберігається у файлі /var/run/utmp, а не з ідентифікатором власника реєстраційного інтерпретатора, вони не в змозі викрити хакера, який

виглядає як пересічний користувач, хоча насправді зареєстрований у системі в якості суперкористувача.

Завдяки програмі `sudo` необхідність реєструватися як суперкористувач, навіть із системної консолі, виникає рідко.

8.4. Інструментальні засоби захисту

Для розв'язання завдань, згаданих у попередніх розділах, можна використовувати вільно розповсюджувані програми. Нижче буде розглянуто найцікавіші з них.

8.4.1. Команда nmap: сканування мережевих портів

Ця команда є сканером мережевих портів. Її основне призначення – перевірка зазначених комп'ютерів на предмет того, які TCP- і UDP-порти на них прослуховуються серверними програмами. За більшістю мережевих служб закріплені «відомі» порти, тому така інформація дає змогу дізнатися, які програми виконуються на комп'ютері.

Запуск команди `nmap` – чудовий спосіб дізнатися, який вигляд має система в очах того, хто намагається її зламати. Ця можливість може виявитися дуже корисною при аналізі стану локальної мережі. На жаль, вона ж є робочим інструментом хакерів, які можуть визначити тип атаки на підставі відомих слабких місць конкретних операційних систем або серверів.

Не забувайте також, що більшість адміністраторів не вітають спроб сканування мережі та з'ясування її слабких місць, якими б шляхетними не були мотиви. Ніколи не запускайте команду nmap у чужій мережі без дозволу мережевого адміністратора.

8.4.2. Nessus: мережевий сканер наступного покоління

У 1998 році Рено Дерезон (Renaud Deraison) випустив цікавий пакет Nessus, у якому реалізовано багато функціональних можливостей сканера. Він використовує понад 31000 надбудов для перевірки локальних і віддалених дефектів. Незважаючи на те що ця програма має закритий код і є комерційним продуктом, вона вільно поширюється і для неї регулярно з'являються нові надбудови. Це найбільш широко розповсюджений і повний із доступних сканерів.

Програма Nessus – це мережевий сканер, який нічому не довіряє. Замість того щоб припускати, наприклад, ніби всі веб-сервери працюють через порт 80, він шукає веб-сервери на будь-якому порту, а потім аналізує їхні слабкі місця. Програма не довіряє навіть номеру версії, про який повідомляє сам сервер, перевіряючи всі відомі слабкі місця, щоб зрозуміти, наскільки вразливий сервер.

Для першого запуску програми Nessus потрібно витратити чимало зусиль (необхідно встановити багато пакетів, які не входять до стандартного дистрибутива), але кінцевий результат того вартий. Система Nessus складається з клієнта і сервера. Сервер відіграє роль бази даних, а клієнт відповідає за графічний користувацький інтерфейс. Сервери і клієнти можуть працювати на платформах UNIX або Windows.

Однією з головних переваг програми Nessus є її модульна структура, що дозволяє стороннім розробникам додавати власні модулі перевірки. Завдяки активності спільноти її користувачів, ця програма зможе бути корисною довгий час.

8.4.3. John the Ripper: засіб для виявлення слабких паролів

Один зі способів присікти використання поганих паролів полягає в тому, що б самостійно зламати пароль і змусити користувача вибрати інший. John the Ripper - це складна програма, розроблена компанією Solar Designer, яка реалізує різні алгоритми злому паролів у вигляді єдиного інструменту.

Незважаючи на те що більшість систем використовує файл тіньових паролів, щоб приховати зашифровані паролі від публіки, доцільно перевіряти, що паролі користувачів є стійкими до злому. Знання пароля користувача може виявитися корисним, тому що люди вважають за краще весь час використовувати один і той самий пароль. Єдиний пароль може забезпечити доступ до іншої системи, зашифрувати файли, що зберігаються в робочому каталозі, і відкрити доступ до фінансових рахунків у мережі веб. (Не варто й казати, що використовувати пароль у такий спосіб дуже нерозумно. Однак ніхто не хоче запам'ятовувати десять різних паролів.)

Незважаючи на свою внутрішню складність, програма John the Ripper досить проста у використанні. Достатньо просто націлити програму john на файл, що підлягає злому, найчастіше /etc/shadow, і станеться диво.

```
$ sudo ./john /etc/shadow
Loaded 25 password hashes with 25 different salts (FreeBSD MD5 [32/32])
password (jsmith)
badpass (tjones)
```

У цьому прикладі з тіньового файлу було зчитано 25 унікальних паролів. Після злому пароля програма John виводить його на екран і зберігає у файл john.pot. Цей вивід містить пароль у лівому стовпчику і реєстраційне ім'я в дужках у правому стовпчику. Для перегляду паролів після завершення роботи програми john слід виконати ту саму команду з аргументом -show.

Як і для більшості інших методів моніторингу безпеки, перед зломом паролів за допомогою програми John the Ripper необхідно отримати схвалення керівництва.

8.4.4. Snort: популярна програмна система для розпізнавання проникнення в мережу

Відкрита система Snort призначена для запобігання і розпізнавання несанкціонованого проникнення в мережу; написана Марті Решем (Marty Roesch) і наразі підтримується комерційною компанією Sourcefire (snort.org). Вона де-факто стала стандартом для кустарних систем NIDS, а також основною для багатьох комерційних реалізацій систем NIDS з «керованими послугами».

Сама по собі система Snort поширюється як пакет із відкритим кодом. Однак компанія Sourcefire стягує плату за підписку на доступ до новітніх правил розпізнавання.

Велика кількість платформ, створених сторонніми виробниками, містять у собі або експортують систему Snort, причому деякі з цих проєктів є програмами з відкритим кодом. Яскравим прикладом є проєкт Aanval (aanval. com), що збирає дані від численних датчиків системи Snort на веб-консоль.

Система Snort перехоплює пакети з кабелю і порівнює їх із наборами правил, які називаються сигнатурами. Коли система Snort розпізнає подію, яка становить

інтерес, вона може попередити системного адміністратора або встановити контакт із мережевим пристроєм і, крім іншого, заблокувати небажаний трафік.

8.5. Особливості брандмауерів у системі Linux

Зазвичай не рекомендується використовувати комп'ютер, що працює під управлінням Linux (а також UNIX або Windows), як брандмауер, оскільки загалом операційна система не забезпечує належний рівень безпеки. Однак краще інсталювати посилену систему Linux, ніж узагалі нічого, якщо йдеться про домашній комп'ютер або організацію, бюджет якої не передбачає купівлю обладнання відповідного рівня. У будь-якому разі локальний фільтр, такий як команда iptables, може виявитися чудовим інструментом для забезпечення безпеки.

Встановлюючи систему Linux як брандмауер, переконайтеся, що вона містить найостанніші оновлення та «латки», що стосуються прогалин у системі захисту.

За детальними рекомендаціями з налаштування брандмауеру слід звернутися до спеціалізованої літератури або довікової системи.

8.6. Віртуальні приватні мережі

У найпростішому вигляді віртуальна приватна мережа (Virtual Private Networks, VPN) – це спеціальний тип мережевого з'єднання, що емулює безпосереднє підключення віддаленої мережі, навіть якщо насправді вона знаходиться за тисячу кілометрів і прихована за безліччю маршрутизаторів. Для підвищення безпеки з'єднання не тільки піддається аутентифікації в тому чи іншому вигляді (зазвичай з використанням спільного секретного ключа, наприклад пароля), але ще й шифрується. Це називається захищеним тунелем.

Наведемо приклад ситуації, в якій мережа VPN виявляється дуже зручною. Припустимо, компанія має офіси в кількох містах: Київ, Чернігів та Жмеринка. Якщо кожен офіс під'єднаний до локального провайдера Інтернету, то за допомогою віртуальних приватних мереж компанія може прозоро (і, головне, досить безпечно) з'єднати всі офіси через таку ненадійну мережу, як Інтернет. Звісно, такого ж результату можна досягти і за рахунок купівлі виділених ліній, але це обійдеться набагато дорожче.

Як ще один приклад можна взяти компанію, службовці якої виходять у мережу з дому. Наявність віртуальних приватних мереж дасть змогу службовцям користуватися перевагами своїх високошвидкісних і недорогих кабельних модемів і водночас працювати так, ніби вони безпосередньо підключені до корпоративної мережі.

Завдяки зручності та популярності такого підходу, багато компаній стали пропонувати рішення, пов'язані з мережею VPN. Функції підтримки VPN можуть бути реалізовані в маршрутизаторі, в операційній системі і навіть у спеціалізованому мережевому пристрої.

8.6.1. Тунелі IPSEC

Ті, кого цікавить надійне і недороге рішення у сфері VPN, повинні звернути увагу на IPSEC (Internet Protocol Security – механізм захисту протоколу IP). Спочатку він був реалізований для протоколу IPv6, але тепер доступний також і для IPv4. IPSEC – це схвалена організацією IETF система аутентифікації та шифрування

з'єднань. Практично всі серйозні постачальники VPN-систем оснащують свої продукти принаймні режимом сумісності з IPSEC. Системи Linux, Solaris, HP-UX і AIX містять підтримку технології IPSec, вбудовану в ядро.

Для аутентифікації та шифрування механізм IPSec використовує сильні криптографічні алгоритми. Аутентифікація гарантує, що пакети надходять від легального відправника і дорогою не були спотворені, а шифрування запобігає несанкціонованому перегляду вмісту пакета.

У тунельному режимі система шифрує заголовок транспортного рівня, що містить номери портів відправника й одержувача. На жаль, цей механізм безпосередньо конфліктує зі способом роботи більшості брандмауерів. З цієї причини в більшості сучасних реалізацій за замовчуванням використовують транспортний режим, у якому шифрується тільки вміст пакетів (тобто передані дані).

Існує одна тонкість, пов'язана з тунелями IPSEC і параметром MTU переданих пакетів. Важливо переконатися в тому, що пакет, зашифрований засобами IPSEC, не піддається фрагментації під час проходження тунелю. Для цього може знадобитися знизити значення MTU для мережевих інтерфейсів, розташованих перед тунелем (зазвичай підходить значення 1400 байт).

8.6.2. Чи так вже потрібні віртуальні приватні мережі

На жаль, у віртуальних приватних мереж є і недоліки. Вони дають змогу організувати досить надійний тунель через ненадійний канал зв'язку між двома кінцевими точками, але вони, як правило, не захищають самі кінцеві вузли. Наприклад, якщо створити мережу VPN між корпоративною магістраллю і домашнім комп'ютером президента компанії, то можна ненароком відкрити зручну лазівку для 15-річного вундеркінда, який ночами навідується в татів кабінет. Добре, якщо він використовує цю можливість лише для ігор по мережі з однокласниками. А якщо ні?

Звідси висновок: з'єднання, що встановлюються через тунелі VPN, слід розглядати як зовнішні і надавати їм додаткові привілеї лише в разі крайньої необхідності, ретельно зваживши всі за і проти. Можна додати до зводу правил безпеки, прийнятого в організації, спеціальний пункт, що стосується роботи в мережі VPN.

8.7. Що потрібно робити у разі атаки на сервер

Насамперед, не панікуйте, найімовірніше, до моменту виявлення злому більшу частину збитків уже завдано. Можливо, хакер «гуляв» системою кілька тижнів або навіть місяців. Імовірність того, що вам вдалося виявити факт злому, що стався всього годину тому, близька до нуля.

У світлі цього мудра сова наказує зробити глибокий вдих і почати ретельно продумувати стратегію усунення злому. Намагайтеся не сполохати зловмисника, привселюдно заявляючи про подію або виконуючи дії, які можуть насторожити того, хто спостерігав за діяльністю організації впродовж кількох тижнів. Підказка: у цей момент непогано виконати резервне копіювання. Сподіваємося, це не здивує порушника?!

Варто також нагадати самому собі про те, що, згідно з дослідженнями, у 60% випадків, пов'язаних із порушенням безпеки, був присутній «внутрішній ворог». Не

вивчивши всіх фактів, намагайтеся не посвячувати в проблему зайвих людей, ким би вони не були.

Наведемо короткий план, який допоможе адміністратору пережити кризу.

Етап 1: не панікуйте. У багатьох випадках проблема виявляється тільки через якийсь час. Ще одна-дві години або дні вже нічого не вирішують. Однак має значення реакція на подію – панічна або раціональна. Часто внаслідок паніки, що виникла, ситуація тільки погіршується знищенням важливої інформації, яка дозволяє виявити порушника.

Етап 2: визначте адекватну реакцію. Нікому не вигідно роздмухувати інцидент. Будьте розсудливі. Вирішіть, хто з персоналу братиме участь у вирішенні проблеми, що виникла, і які ресурси при цьому мають бути задіяні. Решта допоможуть здійснювати «винесення тіла», коли все закінчиться.

Етап 3: зберіть усю можливу інформацію. Перевірте облікові та журнальні файли. Спробуйте визначити, де спочатку стався злом. Виконайте резервне копіювання всіх систем. Переконайтеся в тому, що резервні стрічки фізично захищені від запису, якщо ви вставляєте їх у дисковод для читання.

Етап 4: оцініть завдані збитки. Визначте, яка важлива інформація (якщо така є) «покинула» компанію, і виробіть відповідну стратегію пом'якшення можливих наслідків. Оцініть ступінь майбутнього ризику.

Етап 5: висмикніть шнур. Якщо це необхідно і можливо, відключіть «зламані» комп'ютери від мережі. Перекрийте виявлені «дірки».

Етап 6: розробіть стратегію відновлення. Зберіть тямущих людей і обговоріть план відновлення. Не заносьте його в комп'ютер. Зосередьтеся на тому, щоб загасити «пожежу» і звести збитки до мінімуму. Уникайте звинувачень на чиюсь адресу або нагнітання обстановки. Не забувайте про психологічний удар, який можуть завдавати користувачі.

Етап 7: повідомте про свій план. Розкажіть користувачам і керуючому персоналу про наслідки злому, можливі майбутні проблеми та попередню стратегію відновлення. Будьте чесні та відверті. Інциденти, пов'язані з безпекою, є невід'ємною частиною сучасних мереж. Це не відображення ваших здібностей як системного адміністратора або чогось іншого, чого можна було б соромитися. Відкрите визнання проблеми, що виникла, – 90% перемоги, якщо при цьому ви демонструєте, що маєте план виходу із ситуації.

Етап 8: втіліть план у життя. Ви знаєте свої системи і мережі краще, ніж будь-хто інший. Довіртеся плану та інстинктам. Порадьтеся з колегами з інших організацій, щоб переконатися в правильності обраного напрямку.

Етап 9: повідомте про інцидент у відповідні органи. Якщо в інциденті брав участь «зовнішній гравець», повідомте про цей випадок в організацію CERT. Відповідна адреса `https://cert.gov.ua`. Надайте їм якомога більше інформації.

Якщо є підозра, що вами виявлено раніше невідому проблему в програмному забезпеченні, слід також повідомити про це компанію-розробника.

СПИСОК ВИКОРИСТАНОЇ ЛІТЕРАТУРИ

1. Limoncelli T. A., Hogan C. J., Chalup S. R. Practice of System and Network Administration, The (2nd Edition). 2nd ed. Addison-Wesley Professional, 2007. 1056 p.
2. UNIX and Linux System Administration Handbook (5th Edition) / E. Nemeth et al. Addison-Wesley Professional, 2017. 1232 p.
3. Ровник О. Командна оболонка BASH – поняття, команди та використання. URL: https://freehost.com.ua/ukr/faq/articles/komandnaja-obolochka-bash--ponjatie-komandi-i-ispolzovanie/(дата звернення: 26.08.2023).
4. Ровник О. Мовні конструкції та внутрішні змінні BASH. URL: https://freehost.com.ua/ukr/faq/articles/jazikovie-konstruktsii-i-vnutrennie-peremennie-bash/(дата звернення: 26.08.2023).
5. Ровник О. Організація обробки даних в сценаріях BASH. URL: https://freehost.com.ua/ukr/faq/articles/organizatsija-obrabotki-dannih-v-stsenarijah-bash/(дата звернення: 26.08.2023).
6. Ровник О. Використання циклів та виразів у сценаріях BASH. URL: https://freehost.com.ua/ukr/faq/articles/ispolzovanie-tsiklov-i-virazhenij-v-stsenarijah-bash/(дата звернення: 26.08.2023).
7. Зовнішні команди BASH для роботи з виразами та виконання складних обчислень. URL: https://freehost.com.ua/ukr/faq/articles/vneshnie-komandi-bash-dlja-raboti-s-virazhenijami-i-vipolnenija-slozhnih-vichislenij/(дата звернення: 26.08.2023).
8. Запуск python скрипта в Linux. *Losst*. URL: https://losst.pro/zapusk-python-skripta-v-linux(дата звернення: 26.08.2023).