Міністерство освіти і науки України
Сумський державний університет

5467 Методичні вказівки
до виконання лабораторних робіт
із дисципліни «**Мережеві операційні системи**»
для студентів спеціальності
172 *«Телекомунікації та радіотехніка»*
денної форми навчання

Частина 3

Суми
Сумський державний університет
2022

Методичні вказівки до виконання лабораторних робіт із дисципліни «Мережеві операційні системи» / укладачі: В. В. Гриненко, О. В. Д’яченко. – Суми : Сумський державний університет, 2022. – 44 с.

Кафедра електроніки і комп’ютерної техніки

Лабораторна робота 7

Програмування на мові bash. Написання скриптів

Мета – оволодіння практичними навичками професійної роботи з командною оболонкою shell – використання змінних і створення командних файлів.

Завдання для самостійної підготовки

1. Вивчити:
   - організацію умовного виконання командного рядка, угруповання команд у командному рядку;
   - використання змінних shell;
   - організацію командних файлів;
   - арифметичні обчислення в shell;
   - використання псевдонімів.

2. Детально ознайомитися з довідкової системи ***man*** з такими командами UNIX*: let, echo, read, env, set, sh*. Звернути увагу на метасимволи *, ?, /, […], $ і на правила інтерпретації їх під час використання одинарних і подвійних лапок ‘…’ та "…".

Теоретичні відомості

Shell-скрипт – це звичайний текстовий файл, в який послідовно записані команди, які користувач може зазвичай вводити в командному рядку. Файл виконується командним інтерпретатором – шеллом (shell). У Linux- і Unix-системах для того, щоб бінарний файл або скрипт змогли бути запущені на виконання для користувача, який запускає файл, повинні бути встановлені відповідні права на виконання. Це можна зробити за допомогою команди *chmod u + x імя_скрипта*. У першому рядку скрипту зазначається шлях до інтерпретатора *#!/bin/bash*.

Для створення скрипту можна скористатися текстовим редактором *nano* або *vi*, набравши ім’я редактора в командному рядку.

Розглянемо основні правила програмування на мові *bash*.

Коментарі. Рядки, що починаються з символу # (за винятком комбінації #!), є коментарями. Коментарі можуть також розміщуватися і в кінці рядка з виконуваним кодом.

Особливості роботи з рядками. Одиничні лапки (‘ ‘), що обмежують рядки з обох сторін, використовують для запобігання інтерпретації спеціальних символів, які можуть перебувати в рядку.

Подвійні лапки (" ") запобігають інтерпретації спеціальних символів, за винятком $, `(зворотна лапка) і \ (escape – зворотний слеш). Бажано використовувати подвійні лапки під час звернення до змінних. За необхідності вивести спеціальний символ можна також, використовуючи екранування: символ \ запобігає інтерпретацію наступного за ним символу.

Пробіли та перенесення рядків. Інтерпретатори *sh* і *bash* чутливі до пробілів і переносів рядків. Окремі команди повинні починатися з нового рядка. Якщо є необхідність написати ще одну команду в тому самому рядку, що і попередня – можна поставити крапку з комою в кінці попередньої команди. Пробіл зазвичай розділяє назву команди та параметри, які їй передаються, а також параметри між собою.

Змінні. Ім’я змінної аналогічно традиційному уявленню про ідентифікатор, тобто ім’ям може бути послідовність літер, цифр і підкреслень, що починається з букви або підкреслення. Коли інтерпретатор натрапляє в тексті сценарію на ім’я змінної, то він замість нього підставляє значення цієї змінної. Тому посилання на змінні називаються підстановкою змінних. Якщо variable1 – це ім’я змінної, то $ variable1 – це посилання на її значення. «Чисті» імена змінних, без префіксу $, можуть використовуватися лише під час оголошення змінної або під час присвоєння змінної деякого значення. На відміну від більшості інших мов програмування, Bash не виробляє поділу змінних за типами. По суті, змінні Bash є строковими змінними, але залежно від контексту Bash допускає цілочисленну арифметику зі змінними. Визначальним фактором тут є вміст змінних.

Оператор присвоювання "=". Під час використання оператора присвоювання не можна ставити пробіли ліворуч і

праворуч від знаку рівності. Якщо в процесі привласнення потрібно виконати арифметичні операції, то перед записом арифметичного виразу використовують оператор ***let,*** наприклад:

*let a=2*2* (оператор множення є спеціальним символом і повинен бути екранований).

Якщо потрібно привласнити значення, що містить пробіл, потрібно використовувати лапки:

NAME="Ivan Ivanov".

Арифметичні оператори:

"+" Додавання.

"–" Віднімання.

"*" Множення.

"/" Ділення (цілочисельне).

"**" Зведення в ступінь.

"%" Залишок від ділення.

Спеціальні змінні. Для Bash існує низка зарезервованих імен змінних, які зберігають певні значення.

Позиційні параметри. Аргументи, що передаються скрипту з командного рядка, зберігаються в зарезервованих змінних $0, $1, $2, $3 ..., де $0 – це назва файлу сценарію; $1 – це перший аргумент; $2 – другий; $3 – третій тощо. Аргументи, які йдуть за $9, повинні бути в фігурних дужках, наприклад: $ {10}, $ {11}, $ {12}.

Передача параметрів скрипту відбувається у вигляді перерахування цих параметрів після імені скрипта через пробіл в момент його запуску.

Інші зарезервовані змінні:

$ DIRSTACK – вміст вершини стека каталогів;

$ EUID – ефективний H_UID;

$ UID – ... містить реальний ідентифікатор, який встановлюється лише з логіном;

$ GROUPS – масив груп, до яких належить поточний користувач;

$ HOME – домашній каталог користувача;

$ HOSTNAME – hostname комп'ютера;

$ HOSTTYPE – архітектура машини;

$ PWD – робочий каталог;

$ OSTYPE – тип ОС;

$ PATH – шлях пошуку програм;

$ PPID – ідентифікатор батьківського процесу;

$ SECONDS – час роботи скрипта (в секундах);

$ # – загальна кількість параметрів, переданих скрипту;

$ * – всі аргументи, передані скрипту (виводяться в рядок);

$ @ – те саме, що і попередній, але параметри виводяться в стовпчик;

$! – PID останнього запущеного в тілі процесу;

$$ – PID самого скрипта.

Приклади використання

history

$HISTSIZE=5

history

set PATH=$PATH:/home/student/scripts

set PATH=$PATH:$HOME/scripts

PS1=”`uname`:”

Код завершення. Команда *exit* може використовуватися для завершення роботи сценарію. Крім того, вона може повертати деяке значення, яке може бути проаналізовано процесом, що викликає скрипт. Команді exit можна явно зазначити код повернення у вигляді exit nnn, де nnn – це код повернення (число в діапазоні 0–255).

Оператор виведення. *echo змінні_або_рядки.*

Оператор введення. *read ім’я_змінної.* Одна команда *read* може прочитати (привласнити) значення відразу для декількох змінних. Якщо змінних у *read* більше, ніж їх введено (через пробіли), тим що залишилися присвоюється порожній рядок. Якщо переданих значень більше, ніж змінних в команді *read*, то зайві ігноруються.

$ {<var>} <текст> відділення змінних від тексту

let a=3+5

echo ${a}text.

Для маніпулювання рядками застосовуються спеціальні підстановки:

1. ${змінна:=значення} Значення присвоюється змінній, якщо вона не визначена або є пустим рядком.

var1=”abc”
${var1:=”cde”}
${var2:=”fgh”}
echo $var1
echo $var2

2. ${змінна:+значення} Якщо змінна ініціалізована (визначена),замість неї використовується зазначене в конструкції значення. (*)

var1=${a:+”text”}
echo $var1

3. ${змінна:-значення} Якщо змінна визначена і не є пустим рядком, підставляється її значення, інакше підставляється значення, зазначене в конструкції. (*)

var1=${a:-”text”}
echo $var1

4. ${#змінна} кількість символів у значенні змінної

a=”abc”
echo ${#a}

Під час підстановки команд потрібно використовувати зворотні одинарні лапки (вони розміщені під символом лише на клавіатурі). Підставляти можна не лише одну команду, а цілі списки команд:

USERS=`who | wd –l`
UP=`date; uptime`
I=`whoami`

Команда *test* перевіряє виконання деякої умови. З використанням цієї (вбудованої) команди формуються оператори вибору та циклу мови shell.

Два можливих формати команди:
test умова
або
[[Умова]].

У shell використовуються умови різних типів (табл. 1, 2, 3). Наприклад, якщо ім'я існуючого файлу *abc*, то умова *[-f abc]* була істинна, команда *echo $?* видасть код завершення 0; умова *[-cl abc]* – помилкова, код завершення 1.

Таблиця 7.1 – Умови перевірки файлів

Умова	**Значення**
-a file	Істинне, якщо файл із іменем file існує
-f file	Файл file є звичайним файлом
-d file	Файл file – каталог
-c file	Файл file – спеціальний файл
-r file	Існує дозвіл на читання файлу file
-w file	Існує дозвіл на запис у файл file
-s file	Файл file не пустий

Таблиця 7.2 – Умови перевіркм рядків

Умова	**Значення**
strl = str2	Рядки str1 і str2 збігаються
strl != str2	Рядки str1 і str2 не збігаються
-n strl	Рядок str1 існує
-z str1	Рядок str1 не існує

Таблиця 7.3 – Умови порівняння цілих чисел

Умова	**Значення**
x -eq y	х дорівнює у
x -ne y	х не дорівнює у
X -gt y	х більше у
X -ge y	х більше або дорівнює у
x -lt y	х менше у
X -le y	х менше або дорівнює у

Умовний оператор

if команда1
then команда2
[elif умова then команда3]
[else команда4]
fl

У цьому разі квадратні дужки використані для позначення на необов'язковість конструкцій, **elif** – скорочений варіант від else **if**, може бути використаний поряд із повним, тобто допускається вкладення довільної кількості операторів **if**.

Якщо команда1 повернула після виконання значення «істина», то виконується команда2 після then. Якщо є необхідність порівнювати значення змінних і/або констант, після if використовується спеціальна команда

[Вираз]. Обов'язково ставити пробіли між виразом і дужками, наприклад:

if ["$a" -eq "$b"]; then echo "a = b"; fi

Оператор вибору **case** має структуру:

case рядок in
шаблон) список команд;;
шаблон) список команд;;
esac.

case і **esac** – службові слова, «рядок» (це може бути і один символ) порівнюється з «шаблоном». Потім виконується «список команд» обраного рядка, службове слово esac необхідне для завершення структури.

Цикл for. Оператор циклу **for** може мати два способи завдання.

Стандартний –

*for змінна **in** спісок_значеннь;*
do команди;
done.

2. С-подібний

for ((i = 0; c <= 3; i ++))
do
echo $ i
done

Часто використовується форма **for i in ***, що означає «для всіх файлів поточного каталогу».

Цикл while. Структура **while** краще тоді, коли невідомий заздалегідь точний список значень параметрів або цей список повинен бути отриманий унаслідок обчислень у циклі. Оператор циклу **while** має таку структуру:

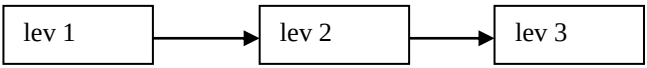
while умова do
список команд done,

де **while** – службове слово, яке визначає тип циклу з істинною умовою. Список команд у тілі циклу (між **do** і **done**) повторюється до того часу, поки зберігається істинність умови (тобто код завершення останньої команди в тілі циклу дорівнює 0) або цикл не буде перерваний зсередини спеціальними командами (**break, continue** або **exit**). Під час першого входу в цикл умова повинна виконуватися.

Для змінних установлений рівень видимості. Змінні можуть бути глобальними та локальними.

Локальні – ті, які видно на поточному рівні роботи процесора.

Глобальні видно на всіх нижніх рівнях, тобто під процесорами



Команди для управління змінними

env – перегляд глобальних змінних;
set – змінні певного рівня;
unset – розіменування змінних;
export – зробити змінну глобальною;

Підстановки
\$ $ для всіх, # – root
\d – поточна дата
\t – час
\s – діючий shell
\u – login
\w – поточний каталог
\h – ім’я хосту

Використання різних кольорів
PS1=”\033[0;32m \u@\h: \033[0m”
0;30 black 0; 33 brown
0;34 blue 0; 35 purple
0;32 green 0; 37 light gray
0;36 eyan 1; 30 dark gray
0; 31 red 1; 37 white
echo –e “\033[0;34m <text>”

Формування псевдонімів

Для простих часто використовуваних команд необов’язково створювати скрипт – достатньо зробити «псевдонім» (alias).

Для визначення псевдонімів використовується оператор alias. Наприклад:

alias ll=‘ls –l’

Отримали нову команду *ll*. Перевіряємо її роботу:

ll.

Аналогічно визначенню змінних, навколо символу "=" не повинно бути пробілів, а якщо в «значенні» псевдоніму є пробіли або спецсимволи, то їх потрібно «екранувати».

Подивитися список псевдонімів можна командою alias без параметрів.

Створення скриптів

Команди, які вводяться в командному рядку можна помістити у файл. Команди виконуватимуться по черзі, одна за одною.

Етапи створення

1. У каталозі (наприклад, /home/student/my_scripts) створити звичайний текстовий файл (назвемо його ech). Це можна зробити за допомогою команди touch ech або будь-якого текстового редактора (vim, gedit, і XW).

2. Помістити в створений файл команди. Це можна зробити перенаправленням виведення (>) або в текстовому редакторі. Виберемо mc → F4. Вводимо echo Hello.

3. Запустити файл до виконання за допомогою спеціальних команд:

. ech

sh ech.

4. Можна зробити створений файл виконуваним.

chmod u+ x ech.

Якщо файл розміщений у каталозі, прописаному в змінній PATH, або ви додали шлях до нього в змінну PATH, то запустити скрипт можна без використання додаткових команд

Echo.

Порядок виконання роботи

Написати програми скриптів і переконатися в правильності їх роботи. Перше завдання спільне для всіх варіантів. Наступні завдання індивідуальні для кожного варіанта. За потреби створіть необхідні умови (створіть каталоги, файли з певним змістом, нових користувачів тощо) та розпишіть всі ці дії під час оформлення звіту з лабораторної роботи. Під час написання скриптів кожну команду супроводжуйте коментарями. Під час написання скриптів передбачити можливість некоректного введення параметрів скрипту з відповідним інформуванням користувача про ці невідповідності.

Завдання загальне для всіх варіантів

1. Створити директорію myscripts, в якій зберігатимуться виконувані файли.
2. У myscripts створити скрипт, що видає таке повідомлення: (використовувати різні кольори). «У моєму домашньому каталозі <n> підкаталогів: <підкаталоги>».
3. Запустити цей скрипт на виконання за допомогою команд інтерпретації.
4. Створити скрипт, що прочитує з екрану деяке слово, і що виводить кількість символів у цьому слові.
5. Зробити цей скрипт виконуваною командою. Запустити її на виконання.
6. Створити свій profile, що виконує такі функції:

− виводить рядок вітання з зазначенням поточної дати;

− дозволяє виконати програми, що зберігаються в my_scripts як звичайні команди;

− дозволяє діставати доступ до файлів і підкаталогів myscripts без завдання додаткового шляху;

− встановлює кількість історії введених команд = 25;

− виводить рядок запрошення, що містить ім’я користувача, ім’я комп’ютера та поточний каталог.

Завдання за варіантами

1. Написати скрипт, що розміщений у заданому каталозі та всіх його підкаталогах, усі файли, власником яких є заданий користувач. Ім'я власника та каталог задаються користувачем як перший і другий аргументи командного рядка. Скрипт виводить результати в файл (третій аргумент командного рядка) у вигляді «повний шлях, ім'я файлу, його розмір». На консоль виводиться загальна кількість переглянутих файлів.

2. Написати скрипт, що підраховує сумарний розмір файлів у заданому каталозі та всіх його підкаталогах (ім'я каталогу задається користувачем як перший аргумент командного рядка). Скрипт виводить результати підрахунку в файл (другий аргумент командного рядка) у вигляді «каталог (повний шлях), сумарний розмір файлів, кількість переглянутих файлів».

3. Написати скрипт із використанням циклу for, що виводить на консоль розміри та права доступу для всіх файлів у заданому каталозі та всіх його підкаталогах (ім'я каталогу задається користувачем як перший аргумент командного рядка).

4. Написати скрипт для пошуку файлів заданого розміру в заданому каталозі (ім'я каталогу задається користувачем як третій аргумент командного рядка). Діапазон (min–max) розмірів файлів задається користувачем як перший і другий аргументи командного рядка.

5. Написати скрипт для пошуку заданого користувачем рядка у всіх файлах заданого каталогу і всіх його підкаталогах (рядок та ім'я каталогу задаються користувачем як перший і другий аргументи командного рядка). На консоль виводяться повний шлях та імена файлів, у яких є заданий рядок і їх розмір. Якщо до якогось каталогу немає доступу, необхідно вивести відповідне повідомлення та продовжити виконання.

6. Написати скрипт, що розміщений у заданому каталозі та всіх його підкаталогах усі файли заданого розміру та належать певному користувачеві. Діапазон (min–max) розмірів файлів задається користувачем як перший і другий аргументи командного рядка. Ім'я власника та каталог задаються користувачем як третій і четвертий аргументи командного

рядка. Скрипт виводить результати пошуку в файл (п’ятий аргумент командного рядка) у вигляді «повний шлях, ім’я файлу, його розмір». На консоль виводиться загальна кількість переглянутих файлів.

Загальне завдання для всіх варіантів

Завдання 1

Створити директорію myscripts, в якій зберігатимуться виконувані файли.

1. Створіть сценарій, який використовує такі ключі:

Ключ -a відображає IP-адреси та імена всіх хостів у поточній підмережі.

Ключ -t відображає список відкритих у системі портів TCP.

Якщо виконувати запуск скрипта без параметрів, то потрібно відобразити “help” – список можливих ключів та їх опис.

Водночас код окремої підзадачі повинен бути поміщений в окрему функцію.

Завдання 2

Створіть скрипт резервного копіювання даних, який матиме такі дані як параметри:

1. Шлях до каталогу синхронізації.
2. Шлях до каталогу, де будуть зберігатися копії файлів.
3. Забезпечити можливість зберігання копії файлів як архіву tar.
4. Забезпечити створення та заповнення (логування) журналу, в який скрипт повинен додати до нього відповідні записи виконаних дій та їх результату з зазначенням часу, типу операції (перевірка, додавання, видалення, архівування) та імені файлу.

Зміст звіту

1. Початкові дані та поставлення задачі.
2. Текст програми.
3. Висновки.

Лабораторна робота 8

Моніторинг та управління процесами

Мета – одержання практичних навичок створення, моніторингу, зміни пріоритетів процесів і завдань і надсилання їм сигналів у оболонці *bash* .

Вивчаються команди: *ps, top, pstree, bg, fg, nice, renice, kill, killall, tee.*

Теоретичні відомості

Процеси в Linux

ОС сімейства UNIX є багатозадачною операційною системою. Це означає, що одночасно може бути запущена більше ніж одна програма. Кожна програма, що працює в деякий момент часу, називається процесом. Кожна команда, яку ви запускаєте, породжує хоча б один процес. Є кілька системних запущених процесів, які увесь час підтримують функціональність системи.

У кожного процесу є унікальний номер, названий process ID, або PID, як і у файлів, у кожного процесу є власник і група. Інформація про власника та групу процесу використовується для визначення того, які файли та пристрої можуть бути відкриті процесом з урахуванням прав на файли. Також у більшості процесів є батьківський процес. Наприклад, під час запуску команд з оболонки, оболонка є процесом і будь-яка запущена команда також є процесом. Для кожного запущеного таким шляхом процесу оболонка буде батьківським процесом. Винятком із цього правила є спеціальний процес, названий init. init завжди перший процес, його PID завжди 1. init запускається автоматично ядром під час завантаження ОС.

Ідентифікатор процесу (PID)

Кожен процес має унікальний ідентифікатор PID, що дозволяє ядру системи розрізняти процеси. Коли створюється новий процес, ядро присвоює йому наступний вільний (тобто не асоційований ні з яким процесом) ідентифікатор. Присвоєння ідентифікатора зазвичай відбувається за зростанням, тобто

ідентифікатор нового процесу більший, ніж ідентифікатор процесу, створеного перед ним. Якщо ідентифікатор досягає максимального значення, наступний процес отримає мінімальний вільний PID і цикл повторюється. Коли процес завершує роботу, ядро звільняє ідентифікатор, що використовувався ним.

Ідентифікатор батьківського процесу (PPID)

Ідентифікатор процесу, що породив цей процес. Усі процеси в системі, крім системних процесів і процесу init, що були першими серед усіх інших процесів, породжені одним з існуючих або існуючих раніше процесів.

Термінальна лінія (TTY)

Термінал або псевдотермінал, пов'язаний із процесом. Із цим терміналом за умовчанням пов'язані стандартні потоки: вхідний, вихідний і потік повідомлень про помилки. Потоки (програмні канали) є стандартним засобом взаємодії між процесами в ОС UNIX.

Реальний (UID) і ефективний (EUID) ідентифікатори користувача

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

Реальний (GID) і ефективний (EGID) ідентифікатори групи

Реальний ідентифікатор групи дорівнює ідентифікатору основної або поточної групи користувача, що запустив процес. Ефективний ідентифікатор необхідний для визначення прав доступу до системних ресурсів від імені групи. Зазвичай ефективний ідентифікатор групи збігається з реальним. Але

якщо для виконуваного файлу встановлений біт SGID, такий файл виконується з| ефективним ідентифікатором групи-власника.

Кожному процесу в ОС Linux призначаються числові ідентифікатори (особисті номери) діапазоном від 1 до 65535 (PID – Process Identifier) і ідентифікатори батьківського процесу (PPID – Parent Process Identifier). PID є ім'ям процесу, за яким ми можемо ідентифікувати процес в операційній системі прід час використання різних засобів перегляду та управління процесами. PPID визначає родинні стосунки між процесами, які значною мірою визначають його властивості та можливості. Інші параметри, які необхідні для роботи програми, називають «оточення процесу». Одним із таких параметрів є керуючий термінал, його мають далеко не всі процеси.

Процеси, імена яких поміщено в квадратні дужки, наприклад "[keventd]" – це процеси ядра. Ці процеси керують роботою системи, а точніше такими її частинами, як менеджер пам'яті, планувальник часу процесора, менеджери зовнішніх пристроїв тощо.

Решта процесів є процесами користувача, запущеними або з командної оболонки (терміналу), або з графічної оболонки, або під час ініціалізації системи.

Інтерактивний і фоновий режими виконання процесів

Під час звичайного запуску в оболонці bash процес працює на передньому плані, тобто процес «прив'язується» до терміналу, з якого він запущений, зчитуючи введення з цього терміналу і здійснюючи на нього виведення. Такий режим роботи процесів називається інтерактивним.

Процес може виконуватися у фоновому режимі (не будучи пов'язаним із терміналом, тобто не виконуючи операції читання – записування на термінал), якщо під час його створення зазначити амперсант:

[root@localhost ~]# sleep 2000 &

[1] 1181

Наприклад, розглянемо unix-утиліта sleep, яка виконує затримання на зазначений час, а після вичерпання цього часу завершується.

Під час створення фонового процесу в оболонці bash у квадратних дужках зазначається номер завдання в системі, а поряд – його PID.

Системні фонові процеси в Linux називаються демонами. Зазвичай вони містять літеру d в кінці своєї назви (systemd, keventd).

Для виведення процесу з фонового режиму використовується команда fg і процес отримує фокус керування (володіє потоками введення – виведення терміналу):

[root@localhost ~]# fg
sleep 2000

Для призупинення інтерактивних процесів використовується комбінація клавіш Ctrl+Z – процесу надсилається сигнал SIGSTOP:

^Z
[1]+ Stopped sleep 2000

Водночас у терміналі відображається запис про зупинене завдання: в квадратних дужках його номер, статус (призупинено), відповідна команда (без амперсанта, оскільки процес був інтерактивним).

Для переведення призупиненого процесу в фоновий режим та його відновлення використовується команда bg:

[root@localhost ~]# bg
[1]+ sleep 2000 &

Моніторинг процесів і завдань

Команда jobs

Для перегляду списку фонових завдань використовується команда jobs:

```
bee@home:~$ jobs
[1]   Running                 sleep 2000 &
[2]+  Stopped                 sleep 3000
[3]   Running                 sleep 4000 &
[4]-  Running                 sleep 5000 &
```

У її виведенні знаком «+» позначено процес, з якими працювали останнім, «–» – процес, з яким працювали передостаннім.

Опція –l дозволить переглянути відповідні PID фонових завдань.

Команда ps

Для перегляду списку процесів в Linux призначена команда **ps:**

ps [PID] options.

Стилі опцій команди

Команда має декілька стилів подання опцій:

GNU-версія цієї програми, що входить до складу Linux, підтримує опції в стилі трьох різних типів:

у стилі Unix98: ps [-опції] (використовується дефіс);

у стилі BSD: ps [опції] (без дефіса);

у стилі GNU-версії: ps [-- довге ім’я опції [--довге ім’я опції] ...] (використовується два дефіса).

Без параметрів **ps** показує всі процеси, які були запущені впродовж поточної сесії та асоційовані з даними терміналом, за винятком демонів. Виводиться ідентифікатор процесу (PID), ідентифікатор терміналу (TTY), витрачений до цього моменту час ЦП (TIME) та ім’я команди (CMD).

```
bee@bee-VirtualBox:~$ ps
  PID TTY          TIME CMD
 2228 pts/0    00:00:00 bash
 2241 pts/0    00:00:00 ps
```

Якщо потрібна інша інформація, необхідно користуватися відповідними опціями. Перелік опцій і пояснень до них можна прочитати в довіднику з команди, скориставшись утилітою *man*: *man ps.*

Опції, які найчастіше використовуються, наведено в таблиці 8.1.

```
bee@bee-VirtualBox:~$ ps -l
F S   UID   PID  PPID  C PRI  NI ADDR SZ WCHAN  TTY          TIME CMD
0 S  1000  2228  2220  0  80   0 -  7054 wait   pts/0    00:00:00 bash
0 S  1000  3088  2228  0  80   0 -  3211 restar pts/0    00:00:00 sleep
0 R  1000  3120  2228  0  80   0 -  3916 -      pts/0    00:00:00 ps
```

Під час зазначення опції -f утиліта ps намагається визначити ім’я команди та аргументи, з якими був створений процес. Якщо це не вдається, ім’я процесу виводиться так само, як і під час відсутності опції -f, тільки в квадратних дужках (процеси ядра не мають виконуваних файлів, їх назва подається саме так).

```
bee@bee-VirtualBox:~$ ps -f
UID          PID  PPID  C STIME TTY          TIME CMD
bee         2228  2220  0 11:25 pts/0    00:00:00 bash
bee         3088  2228  0 12:23 pts/0    00:00:00 sleep 2000
bee         3122  2228  0 12:40 pts/0    00:00:00 ps -f
```

Таблиця 8.1 – Деякі опції команди **ps**

Опції	**Значення**
-e	Вивести інформацію про всі процеси
-a	Вивести інформацію про всі процеси, пов'язані з терміналом
-t список_терміналів	Видавати інформацію лише про процеси, асоційовані з терміналами із заданого списку
-p список_PID_процесів	Видавати інформацію лише про процеси із зазначеними pid
-u список_UID_користувачів	Видавати інформацію лише про процеси зазначених користувачів
-C	Вивести інформацію про процеси, які породжені певною командою (для всіх користувачів і терміналів)
-f	Генерувати повний лістинг (full – повний), процеси ядра позначаються в квадратних дужках
-l	Генерувати лістинг у довгому форматі (long – розширене виведення)
-o	Вмістити в лістинг певні поля
-H	Відобразити ієрархію процесів
--sort	Сортувати виведення за певними полями
ps -ax | grep httpd	Вивести інформацію про конкретний процес

У таблиці 8.2 наводять заголовки колонок видачі ps і їх зміст.

Таблиця 8.2 – Заголовки колонок видачі команди **ps**

Колонка	**Зміст**
1	2
UID	Ідентифікатор власника процесу
PID	Ідентифікатор процесу
PPID	Ідентифікатор батьківського процесу
%CPU	Частка часу центрального процесора (у відсотках), виділеного цьому процесу
%MEM	Частка реальної пам'яті (у відсотках), яка використовується цим процесом
VSZ	Віртуальний розмір процесу
TTY	Термінал, з якого запущено процес
STAT	Статус процесу: D – Очікує на введення / виведення (зазвичай очікує запису на диск); S – Сплячий: очікує завершення події; R – Виконуваний або Готовий: стоїть у черзі на виконання; Z – Стан «зомбі»: процес завершений, але батьківський процес не чекає цього; T – Трасований: процес зупинений сигналом, так як батьківський процес трасує його; X – Зростаючий: процес очікує отримання більшого обсягу основної пам'яті
	Біля покажчика статусу можуть стояти додаткові символи з такого набору (BSD формат): W – процес не має резидентних сторінок; < – Високо-пріоритетний процес; N – низько-пріоритетний процес; L – процес має сторінки, заблоковані в пам'яті; s – є лідером сеансу; l – багатопоточний; + Знаходиться в групі процесів переднього плану
START	Час запуску процесу

1	2
TIME	Час виконання на процесорі
PRI (l)	Пріоритет процесу; більше число означає менший пріоритет
NI (l)	Поправка до пріоритету
ADDR (l)	Адреса процесу в пам'яті
SZ (l)	Розмір (у блоках по 512 байт) образу процесу в пам'яті, який він потребує
VSZ	Розмір ВАП процесу у Кб (VIRT у top)
RSS	Розмір пам'яті процесу, який він займає на цей час (RES у top)
WCHAN (l)	Адреса події, яку очікує процес – системний виклик, який привів до блокування процесу. В активного процесу ця колонка порожня
TTY	Керуючий термінал (зазвичай – термінал, з якого був запущений процес). Якщо такого немає, видається символ «?»
TIME	Витрачений процесом час ЦП
TIME+	Сумарний витрачений процесом час ЦП
COMMAND	Ім'я програми; якщо зазначена опція -f, то виводиться повне ім'я команди та її аргументи

Букви *l* або *f* в дужках означають, що ця колонка з'являється відповідно під час довгого або повного формату видачі; відсутність букв означає, що ця колонка виводиться завжди. Відзначимо, що опції *-l* і *-f* впливають лише на формат видачі, а не на список процесів, інформацію про яких буде надано.

Процес, який закінчив виконання своєї програми і має батьківський процес, який ще не дочекався завершення, як ім'я програми отримує <*defunct*>.

Команда *top*

На відміну від ps, команда top відображає стан процесів і їх активність «у реальному режимі часу».

Команда top має кілька підкоманд, найбільш корисні з яких подані в таблиці 8.3.

Таблиця 8.3 – Підкоманди команди *top*

Підкоманда	**Значення**
h	help – виведення довідкової інформації
q	quit – завершення роботи команди top
f	field – додавання або видалення видимих полів
o	Сортування порядку виведення інформації
F	Вибір полів, за якими виконується сортування
n	number – кількість процесів у виведенні
s	Час оновлення виведення
c	command – відображення імені команди та аргументів
r	renice – зміна пріоритету обраного процесу
k	kill – надсилання сигналу обраному процесу

Виведення команди top містить 2 блоки: статистика роботи системи та статистика роботи процесів.

Перший блок має 5 рядків:

1-й рядок – загальна інформація (top);

2-й рядок – статистика процесів (task);

3-й рядок – статистика використання центрального процесора;

4-й і 5-й рядки – статистика використання пам’яті (memory usage).

```
top - 10:12:07 up 20 min,  1 user,  load average: 0,27, 0,17, 0,17
Tasks: 196 total,   1 running, 195 sleeping,   0 stopped,   0 zombie
%Cpu(s):  3,7 us,  1,0 sy,  0,0 ni, 95,0 id,  0,3 wa,  0,0 hi,  0,0 si,  0,0 st
KiB Mem :  2048456 total,   466556 free,   779544 used,   802356 buff/cache
KiB Swap:   784380 total,   784380 free,        0 used.  1070136 avail Mem
```

Нижче наведено опис перших 3 рядків результатів команди *top*, які описують стан системи.

Перший рядок виводить дані за порядком:

– поточний час (10:12:07);

– час роботи системи (20 min);

– кількість відкритих сесій користувачів (1 користувач);

– середнє завантаження системи (середнє навантаження: 0,27, 0,17, 0,17), три значення відповідають завантаженню в останню хвилину, п’ять хвилин і п’ятнадцять хвилин відповідно.

Другий рядок виводить такі дані:

– загальна кількість процесів у системі (196 total);

– кількість працюючих у даний момент процесів (1 running);

– кількість процесів, які очікують подій (195 sleeping);

– кількість зупинених процесів (0 stopped);

– кількість процесів, які очікують батьківський процес для передачі статусу завершення (0 zombie).

Третій рядок виводить такі дані:

– % використання CPU процесами користувача (3,7 % us);

– % використання CPU системними процесами (1,0 % sy);

– % використання CPU процесами з пріоритетом, підвищеним за допомогою nice (0,0 % ni);

– % часу, коли CPU не використовується (95,0 % id);

– % використання CPU процесами, які очікували завершення операцій введення – виведення (0,3 % wa);

– % використання центрального процесора обробниками апаратних переривань (0,0 % hi – Hardware IRQ);

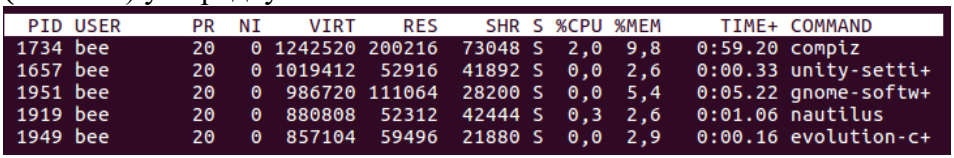
– % використання центрального процесора обробниками програмних переривань (0,0% si – Software Interrupts);

– кількість ресурсів CPU «запозичених» у віртуальної машини гіпервізором для інших завдань; це значення дорівнюватиме нулю на настільних комп’ютерах і серверах, які не використовують віртуальні машини (0,0 % st – Steal Time (запозичений час)).

Повну інформацію про команду *top* можна знайти на *man*-сторінках.

Другий блок виведення команди *top* стосується статистики роботи процесів. Далі наведено приклад, в якому процеси

відсортовано за кількістю використаної віртуальної пам'яті (% MEM) у порядку зменшення:



PID	USER	PR	NI	VIRT	RES	SHR	S	%CPU	%MEM	TIME+	COMMAND
1734	bee	20	0	1242520	200216	73048	S	2,0	9,8	0:59.20	compiz
1657	bee	20	0	1019412	52916	41892	S	0,0	2,6	0:00.33	unity-setti+
1951	bee	20	0	986720	111064	28200	S	0,0	5,4	0:05.22	gnome-softw+
1919	bee	20	0	880808	52312	42444	S	0,3	2,6	0:01.06	nautilus
1949	bee	20	0	857104	59496	21880	S	0,0	2,9	0:00.16	evolution-c+

Перемикати режими відображення (сортувати) можна за допомогою таких гарячих клавіш:

<Shift> + <N> – за PID; **<Shift> + <A>** – за віком;

<Shift> + <P> – за використанням ЦП; **<Shift> + <M>** – за використанням пам'яті; **<Shift> + <T>** – за часом виконання.

Запам'ятати комбінації клавіш зручно за відповідними словами: *N* – number, *A* – age, *P* – processor, *M* – memory, *T* – time.

Керування пріоритетом процесів

У кожного процесу існує параметр, який називається пріоритетом (PRI). Планувальник ОС визначає, який із запущених процесів виконуватиметься першим, який буде наступним тощо на основі значення пріоритету процесу. Значення поправки до пріоритету (NI – *niceness, «люб'язність»*) дозволяє процесам підвищувати або зменшувати власний пріоритет. Завданням з великим чисельним значенням пріоритету дістається менше процесорного часу. Тому працюють вони зазвичай повільніше і споживають набагато менше системних ресурсів. Пріоритет нового процесу дорівнює пріоритету процесу-батька.

В операційних системах Linux використовується система пріоритетів (NI), яка має 40 рівнів, починаючи з –20 (найвищий пріоритет) і закінчуючи 19 (нижчий пріоритет).

Процеси, запущені звичайними користувачами, зазвичай мають пріоритет 0.

Команда ps може показати пріоритет процесу (наприклад, значення nice або NI) за допомогою опції –l.

Команда nice показує пріоритет за замовчуванням.

Запуск процесу з певним пріоритетом

Команда *nice -n число command* дозволяє зазначати пріоритет, з яким виконуватиметься процес після запуску. Від'ємне значення пріоритету може задавати лише суперкористувач (root):

sudo nice -n 10 sleep 1000.

```
bee@home:~/mount/NTFS/OS$ sleep 2000 &
[2] 2600
bee@home:~/mount/NTFS/OS$ ps -f| grep sleep
bee        2600  2000  0 14:28 pts/18   00:00:00 sleep 2000
bee        2610  2000  0 14:29 pts/18   00:00:00 grep --color=auto sleep
```

Зміна пріоритету процесу

Команда *renice -n число PID* дозволяє зменшувати пріоритет виконуваного процесу, збільшуючи його поправку до пріоритету. Змінювати пріоритет процесу дозволено лише користувачу, який запустив процес адміністратору.

Якщо потрібно пришвидшити виконання процесу, потрібно зазначити від'ємну поправку до пріоритету. Виконати таку зміну можна лише від імені адміністратора (root):

sudo renice -n -10 PID.

Надсилання сигналів процесам

Під час роботи процесу ядро контролює його стан і в разі виникнення непередбаченої ситуації керує процесом за допомогою надсилання йому сигналу. Користувач теж може надсилати сигнали, але лише своїм процесам. Під час надходження сигналу процес може скористатися дією за замовчуванням, або, якщо у нього є обробник сигналу, він може перехопити або ігнорувати сигнал. Сигнали SIGKILL і SIGSTOP неможливо ні перехопити, ні ігнорувати, оскільки вони адресовані не процесу, а планувальнику ОС.

Команда *kill -SIGNAL pid* надсилає *SIGNAL* процесу з ідентифікатором *pid*. Якщо сигнал не зазначений, команда посилає процесу SIGTERM.

Команда *killall -s SIGNAL процес* надсилає сигнал усім процесам з іменем *процес*. Якщо сигнал не зазначений, надсилається SIGTERM.

Сигнали для цих команд необхідно зазначати без SIG . Для отримання відповідності цифрового вигляду та імені сигналу використовується опція –l команди *kill*. У таблиці 4 наведено опис деяких сигналів, що існують в ОС Ubuntu:

Звичайні користувачі можуть надсилати сигнали лише тим процесам, для яких вони є власниками. Привілейований користувач root може посилати сигнали будь-яким процесам.

Якщо в команді *kill* скористатися ідентифікатором процесу (PID) таким, що дорівнює *–1*, то зазначений у команді сигнал буде надісланий усім процесам цього користувача. Коли привілейований користувач надсилає сигнал із PID=–1, він розсилається всім процесам, за винятком системних. Якщо цим сигналом буде *SIGKILL*, то у простих користувачів будуть втрачені всі відкриті ними, але не збережені файли даних.

Таблиця 8.4 – Деякі сигнали в ОС Ubuntu

№	Назва	Значення
2	SIGINT	Сигнал надсилається ядром усім процесам під час натиснення клавіші переривання ***CTRL+C***
9	SIGKILL	Сигнал, під час отримання якого виконання процесу припиняється
15	SIGTERM	Сигнал зазвичай має певне попередження, що процес незабаром буде знищений. Цей сигнал дозволяє процесу відповідним чином «підготуватися до смерті» – видалити тимчасові файли, завершити необхідні транзакції і так далі. Команда kill за замовчуванням відправляє саме цей сигнал
18	SIGCONT	Сигнал надсилається для продовження призупиненого процесу
19	SIGSTOP	Отримання сигналу викликає зупинення виконання процесу
20	SIGTSTP	Сигнал надсилається всім процесам поточної групи під час натиснення клавіш **CTRL+Z** Отримання сигналу викликає зупинення виконання процесу

Порядок виконання роботи

1. Спостереження за процесами та побудова ієрархії процесів:

а) створити нового користувача, зайти від нього в систему. Написати скрипт, що складається з команди ***sleep,*** а параметр для цієї команди задати як параметр скрипта. Запустити декілька процесів (команду ***sleep*** і створений скрипт);

б) переглянути список процесів, які є в системі, і записати їх у файл procList;

в) через поля PID і PPID простежити всю ієрархію процесів поточної оболонки;

г) побудувати дерево процесів, які визначені в попередньому пункті. Результат виконання вивести на екран і дописати в файл resultN;

ґ) переглянути список процесів вашого користувача;

д) вивести список процесів вашого користувача у вигляді дерева (команда pstree);

е) вивести на екран та у файл результатів resultN інформацію про процеси вашого користувача (поля S, PID, UID, параметр сортування, CMD), сортовані відповідно до завдання (параметр сортування взяти з табл. 8.5 відповідно з останньою цифрою номера варіанта).

Таблиця 8.5 – Параметри сортування для команди ps

№ варіанта, n	Параметр	Сортування
1	pcpu	За зростанням
2	c	За спаданням
3	pmem	За зростанням
4	vsz	За спаданням
5	time+	За зростанням
6	time+	За спаданням

2. Керування фоновими процесами та наданням процесам пріоритету:

а) запустити у фоні виконання 3 завдання із зазначеними пріоритетами:

- yes fg (пріоритет 10);
- yes bg (пріоритет 10);
- yes nice (пріоритет 0);

б) змінити пріоритети завдань і вивести інформацію про них у resultN:

- yes fg – пріоритет 10 + n;
- yes bg – пріоритет – (10 + n).

3. Команда top: моніторинг процесів, надсилання сигналів, зміна пріоритетів:

а) налаштувати команду top і визначити процеси, які використали найбільше часу ЦП;

б) оновлення результатів кожні 5 секунд;

в) кількість процесів у виведенні – n + 5;

г) команда повинна бути записана в повному форматі;

ґ) стовпчики для відображення – PID, USER, PRI, NI, S, %CPU, %MEM, TIME+, COMMAND;

д) процеси сортовано за TIME+ ;

е) знищити процес “yes bg”, який займає найбільше процесорного часу;

ж) змінити пріоритет процесу “yes nice” на 10 + n;

з) вивести на екран та у файл результатів resultN інформацію про процеси, породжені за допомогою команди yes.

4. Надсилання сигналів:

а) зупинити процес “yes fg”, надіславши йому сигнал SIGSTOP (19);

б) знищити процес “yes fg”, надіславши йому сигнал SIGKILL (9);

в) знищити процес “yes nice”, скориставшись номером завдання;

г) за допомогою клавіш Ctrl + D завершити дочірній bash.

5. Підготовка файлів звіту resultN.txt, procN.txt.

Файли результатів resultN, procN необхідно відкрити в текстовому редакторі gedit і зберегти у форматі txt у кодуванні utf-8 із закінченням рядків як у Windows.

Завдання за варіантами

Варіант 1

1) Згенерувати інформацію – повний лістинг про всі процеси системи.

2) Завершити виконання двох процесів, власником яких є поточний користувач. Перший процес завершити за допомогою сигналу SIGTERM, задавши його ім'я, другий – за допомогою сигналу SIGKILL, задавши його номер.

3) Визначити ідентифікатори процесів, власником яких не є root.

4) У звіті надати всі кроки ваших дій шляхом копіювання з консолі. Коротко пояснити результати виконання всіх команд.

Варіант 2

1) Отримати таку інформацію про процеси поточного користувача: ідентифікатор та ім'я власника процесу, статус і пріоритет процесу.

2) Завершити виконання двох процесів, власником яких є поточний користувач. Перший процес завершити за допомогою сигналу SIGINT, задавши його ім'я, другий – за допомогою сигналу SIGQUIT, задавши його номер.

3) Визначити ідентифікатори та імена процесів, ідентифікатор групи яких не дорівнює кодом поточного користувача.

4) У звіті надайте всі кроки ваших дій шляхом копіювання з консолі. Коротко поясніть результати виконання всіх команд.

Варіант 3

1) Згенерувати таку інформацію – повний лістинг у довгому форматі про процеси поточного користувача: PID, PPID, виділений час ЦП, час запуску, розмір образу.

2) За допомогою сигналу SIGTSTP (використовуючи комбінацію клавіш і команду kill) призупинити виконання процесу, власником якого є поточний користувач. Через кілька секунд відновити виконання процесу.

3) Визначити ідентифікатор та ім’я процесу, створеного останнім користувачем root.

4) У звіті надайте всі кроки ваших дій шляхом копіювання з консолі. Коротко поясніть результати виконання всіх команд.

Варіант 4

1) Показати інформацію про процеси зазначеного користувача у вигляді ієрархії, виведення впорядкувати за значенням PID.

2) За допомогою сигналу SIGSTOP призупинити виконання процесу, власником якого є поточний користувач. Через кілька секунд відновити виконання процесу.

3) Визначити ідентифікатори та імена процесів, які пов’язані з зазначеним терміналом.

4) У звіті надати всі кроки ваших дій шляхом копіювання з консолі. Коротко пояснити результати виконання всіх команд.

Варіант 5

1) Зберегти в файл миттєвий стан процесів у системи зазначеного користувача.

2) Відіслати сигнал SIGINT (за іменем і за номером сигналу) всім процесам, запущеним командою vi.

3) Змінити на 3 одиниці пріоритети процесів, власником яких є поточний користувач.

4) У звіті надати всі кроки ваших дій шляхом копіювання з консолі. Коротко пояснити результати виконання всіх команд.

Варіант 6

1) Згенерувати таку інформацію про m (m > 2) процесах системи, що мають значення ідентифікатора більше заданого n: прапор – відомості про процес, статус, PID, PPID, пріоритет, використане час та ім'я програми.

2) Завершити виконання двох процесів, власником яких є поточний користувач. Перший процес завершити за допомогою сигналу SIGKILL, задавши його ім'я, другий – за допомогою сигналу SIGINT, задавши його номер.

3) Вивести ідентифікатори процесів, для яких батьківським процесом є командний інтерпретатор.

4) У звіті надати всі кроки ваших дій шляхом копіювання з консолі. Коротко пояснити результати виконання всіх команд.

Питання для самоперевірки

1. Поняття процесу. Типи процесів.
2. В яких станах може перебувати процес.
3. Поняття фонового режиму. Знищення процесу.
4. Поняття пріоритету процесу. Типи пріоритетів. Зміна пріоритету процесу, запущеного користувачем.
5. Які команди призначені для управління завданнями?
6. Яка команда дозволяє переглянути список процесів?
7. Яка команда дозволяє переглянути динамічну інформацію про процеси?
8. Як дізнатися та змінити пріоритет виконання процесів?
9. Які команди призначені для надсилання сигналів процесам?

Список електронних джерел

https://www.ibm.com/developerworks/ru/library/l-lpic1-v3-103-5/
https://www.ibm.com/developerworks/ru/library/l-lpic1-v3-103-6/

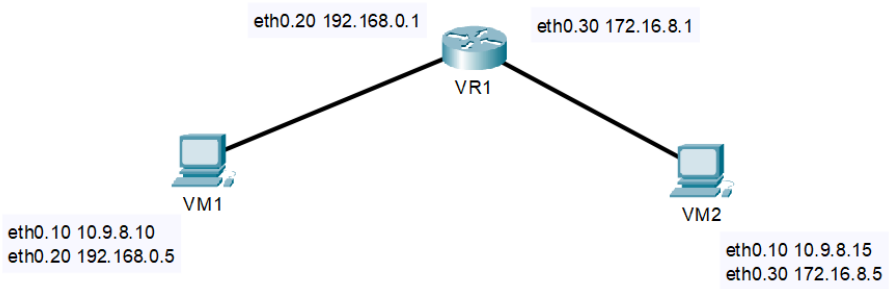
Лабораторна робота 9

Мережеві засоби моніторингу операційної системи Linux

Мета – одержання практичних навичок щодо налаштування та тестування роботи мережі.

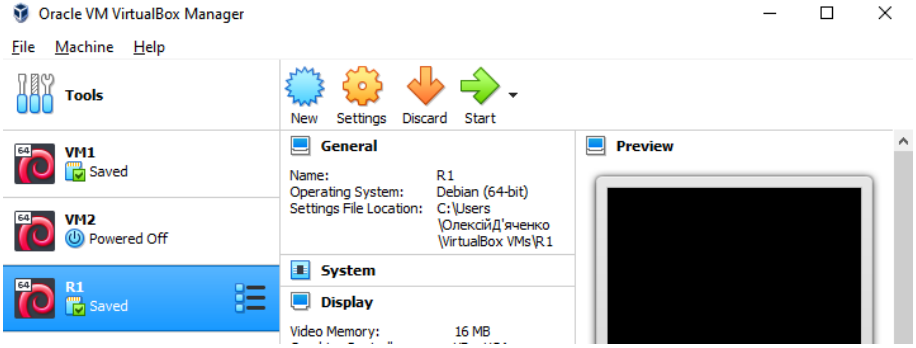
Порядок виконання роботи

Топологія мережі*:

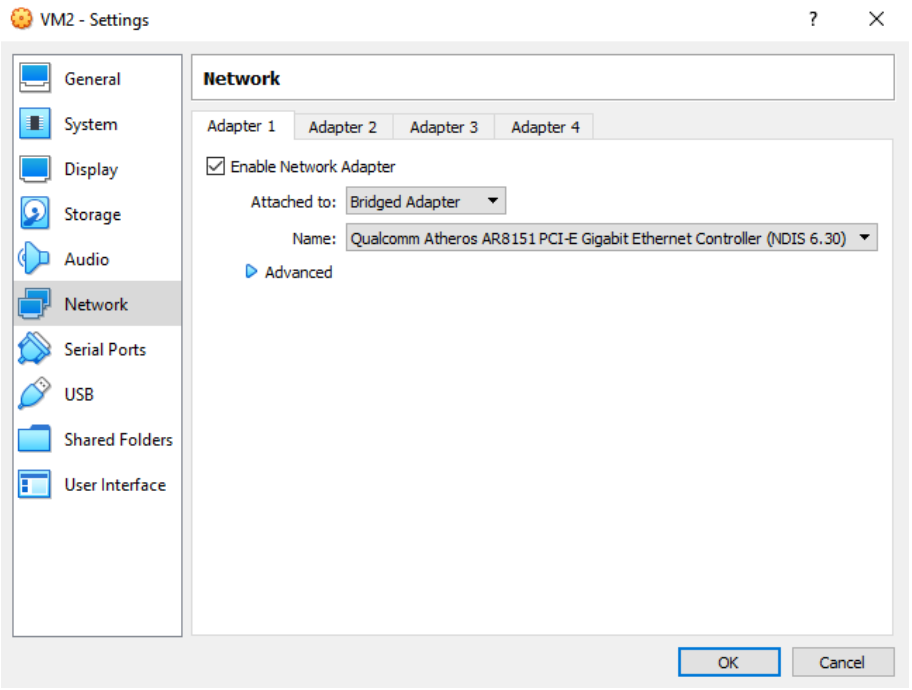


**ip адреси та назви інтерфейсів можуть відрізнятися залежно від варіанта.*

1. Створити віртуальні машини VM1, VM2 та VR1.



2. Налаштування для віртуальних машин вибрати як «Bridged Adapter» на будь який інтерфейс:



3. Запустити віртуальні машини VM1 та VM2.
4. Переглянути існуючі інтерфейси в системі. (*Результат/скріншот додати в звіт. Прокоментувати.*).

Будуємо мережу між VM1 і VM2 в одному VLAN:

5. На VM1 створити новий підінтерфейс інтерфейсу **eth0** з назвою **eth0.10** зі значенням **vlan id = 10**.
6. Призначити йому ip адресу відповідно до топології та варіанта. ***10.9.8.(3 · x)***, *x* – номер варіанта.
7. Перевірити результат. (*Результат/скріншот додати в звіт. Прокоментувати.*)
8. Підняти щойно створений підінтерфейс **eth0.10.** (*Результат/скріншот додати в звіт. Прокоментувати.*).
9. Провести аналогічні налаштування для VM2. ***10.9.8.(10 + 3 · x)***, *x* – номер варіанта.
10. Перевірити зв'язок між віртуальними машинами відповідно до варіанта.

парні номери варіантів: VM1 -> VM2;

непарні номери варіантів: VM2 -> VM1.

(Результат/скріншот додати в звіт. Прокоментувати.).

11. Зробити висновок щодо зв'язку між VM.

Розбиваємо VM1 і VM2 в різні VLAN:

12. На VM1 створити та налаштувати підінтерфейс відповідно до варіанта. (*Результат/скріншот додати в звіт. Прокоментувати.*)

vlan id = (10 + 5 · x), *x* – номер варіанта. ***192.168.x.(3 · x)***, *x* – номер варіанта.

13. На VM2 створити та налаштувати підінтерфейс відповідно до варіанта. (*Результат/скріншот додати в звіт. Прокоментувати*)

vlan id = (20 + 5 · x), *x* – номер варіанта. ***172.16.x.(3 · x)***, *x* – номер варіанта.

14. Запустити VR1.

15. На VR1 потрібно створити та налаштувати підінтерфейси під vlan, що використані в пп.12, 13 відповідно до топології – цей підінтерфейс буде шлюзом у створених мережах. Використовувати першу доступну адресу в мережі. (*Результат/скріншот додати в звіт. Прокоментувати*).

16. Перевірити доступність VM1 і VM2 (за адресами в пп. 12, 13) для VR1. (*Результат/скріншот додати в звіт. Прокоментувати*).

17. Дозволити маршрутизацію для VR1:

Для зв'язку віртуальних машин через VR1 потрібно дозволити йому пересилати пакети між інтерфейсами (маршрутизація). Для цього необхідно в файлі */etc/sysctl.conf* дозволити/розкоментувати директиву ***net.ipv4.ip_forward = 1*** і зберегти зміни.

(*Результат/скріншот додати в звіт. Прокоментувати*).

18. Переглянути список маршрутів на VM1. (*Результат/скріншот додати в звіт. Прокоментувати*).

19. Перевірити доступність VM2 (за адресами в пп. 12, 13) з VM1. (*Результат/скріншот додати в звіт. Прокоментувати*).

20. На VM1 додати маршрут до мережі 172.16.0.0/24 через шлюз - ip адресу інтерфейсу VR1. (*Результат/скріншот додати в звіт. Прокоментувати*).

21. Перевірити доступність шлюзу VR1.

22. Перевірити доступність VM2 (за адресами в пп. 12, 13). (*Результат/скріншот додати в звіт. Прокоментувати пп.21,22*).

23. Переглянути список маршрутів на VM2. (*Результат/скріншот додати в звіт. Прокоментувати*).

24. На VM2 додати маршрут за замовчанням через VR1, зазначивши адресу та маску (0.0.0.0/0 – тобто під неї потрапляють абсолютно всі адреси) або використовуючи ключове слово ***default***.

25. Переглянути список маршрутів на VM2. (*Результат/скріншот додати в звіт. Прокоментувати*).

26. Перевірити зв'язок між віртуальними машинами відповідно до варіанта. *(Результат/скріншот додати в звіт. Прокоментувати)*

парні номери варіантів: VM2 -> VM1;

непарні номери варіантів: VM1 -> VM2.

27. Зобразити топологію мережі та зробити висновки роботи. Додати в звіт.

Питання для самоперевірки

1. Поняття інтерфейсу. Типи інтерфейсів.
2. В яких станах може перебувати інтерфейс.
3. Поняття vlan. Його параметри.
4. Поняття маршрутизації. Типи маршрутизації.
5. Які команди призначені для перевірки інтерфейсів?
6. Яка команда дозволяє переглянути таблицю маршрутизації?
7. Яка команда дозволяє налаштувати маршрут за замовчанням?

Лабораторна робота 10

Створення та налаштування мережі в Linux. Фільтрування мережного трафіку

Мета – одержання практичних навичок щодо створення, налаштування та тестування роботи мережі.

Порядок виконання роботи

Практична частина завдання передбачає створення засобами Virtual Box мережі, що показана на рисунку 10.1.

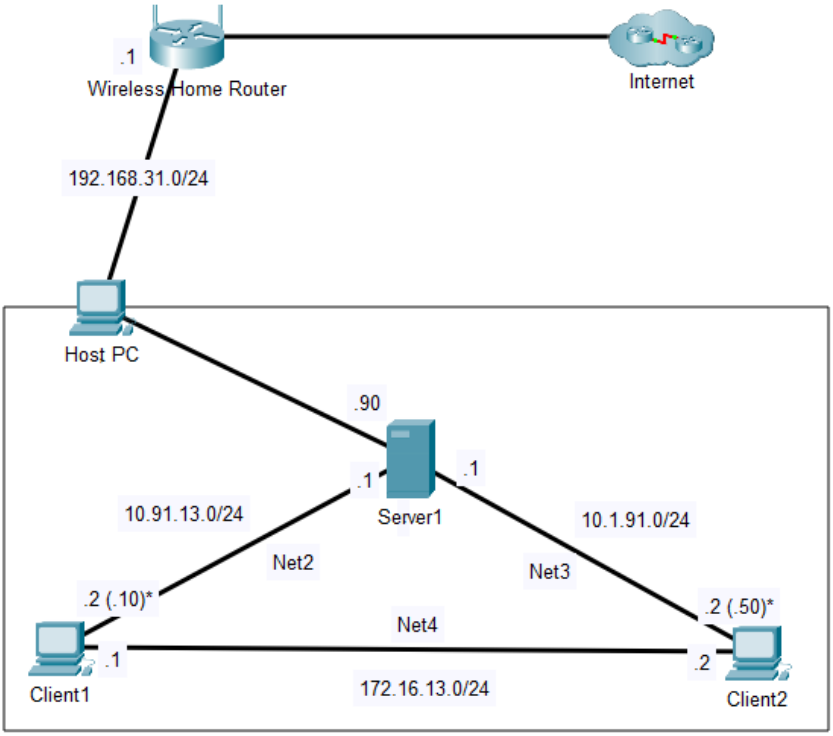


Рисунок 10.1 – Приклад топології мережі до завдання

Host PC – це комп’ютер, на якому запущений Virtual Box.

Server1 – Віртуальна машина, на якій розгорнуто ОС Linux. Int1 цієї машини в режимі «Мережевий міст» під'єднаний до мережі Net1, тобто розміщений в адресному просторі домашньої мережі.

IP-адреса Int1 встановлюється статично відповідно до адресного простору, наприклад 192.168.1.200/24.

Інтерфейси Int2 та Int3 відповідно під'єднано в режимі «Внутрішня мережа» до мереж Net2 та Net3.

Client1 та Client2 – Віртуальні машини, на яких розгорнуто ОС Linux (**Увага!!!** ***різні дистрибутиви***, наприклад, Ubuntu та CentOS). Інтерфейси під'єднані в режимі «Внутрішня мережа» до мереж Net2, Net3 та Net4, як показано на рисунку 1.

1. Створити топологію мережі, як показано на рисунку 1 та провести налаштування Virtual Box згідно з показниками вище.
2. Зробити таблицю адресації, в якій відобразити пристрої, інтерфейси, IP-адреси, маску мережі, шлюз за замовчанням.

Адреса мережі Net2 – 10.Y.D.0/24, де Y – дві останні цифри з вашого року народження, D – дата народження.

Адреса мережі Net3 – 10.M.Y.0/24, де M – номер місяця народження.

Адреса мережі Net4 – 172.16.D.0/24.

Увага! Якщо адресний простір Net2, Net3 або Net4 перетинається з адресним простором Net1 – відповідну адресу можна змінити на власний розсуд.

Увага!!! Подальші налаштування повинні бути постійними, тобто зберігатися в разі перезавантаження пристроїв.

3. На Server1 налаштувати статичні адреси на всіх інтерфейсах. *Зробити відповідні скріншоти.*
4. На Server1 налаштувати DHCP сервіс, який буде конфігурувати адреси Int1 Client1 та Client2. *Зробити відповідні скріншоти.*
5. За допомогою команд *ping* та *traceroute* перевірити зв'язок між віртуальними машинами. *Зробити відповідні скріншоти. Результат прокоментувати*.

Увага! Для того щоб із Client1 і Client2 проходили пакети в мережу Internet (точніше, щоб поверталися з Internet на Client1 та Client2) на Wi-Fi Router необхідно налаштувати статичні маршрути для мереж Net2 та Net3.

6. На віртуальному інтерфейсі ***lo*** Client1 призначити дві IP-адреси за таким правилом: 172.17.D+10.1/24 та 172.17.D+20.1/24. (**Увага!!!** Їх також необхідно додати у таблицю адресації). Налаштувати маршрутизацію так, щоб трафік із Client2 до 172.17.D+10.1 проходив через Server1, а до 172.17.D+20.1 через Net4. Для перевірки використати *traceroute*. *Зробити відповідні скріншоти.*

7. Розрахувати спільну адресу та маску адрес 172.17.D+10.1 та 172.17.D+20.1, при чому маска повинна бути максимальною з можливих. Видалити маршрути, встановлені на попередньому кроці та замінити їх об'єднаним маршрутом, який повинен проходити через Server1. *Зробити відповідні скріншоти.*

8. Налаштувати SSH-сервіс так, щоб Client1 і Client2 могли під'єднуватися до Server1 та один до одного. *Зробити відповідні скріншоти.*

9. Налаштуйте на Server1 firewall так:

– дозволено під'єднатися через SSH із Client1 та заборонено з Client2;

– з Client2 на 172.17.D+10.1 *ping* проходив, а на 172.17.D+20.1 не проходив.

Відповідні команди налаштувань і результат роботи (скріншоти) відобразити в звіті.

10. Якщо в п. 5 була налаштована маршрутизація для доступу Client1 і Client2 до мережі Інтернет – видалити відповідні записи. На Server1 налаштувати NAT сервіс так, щоб із Client1 і Client2 проходив *ping* у мережу Інтернет.

Зміст звіту

1. Топологію мережі згідно з завданням.
2. Таблицю адресації.
3. Результат виконання кожного з кроків лабораторної роботи. За необхідності з коментарями.
4. Висновки до роботи, де описати складнощі, які у вас виникли, коментарі та пропозиції щодо лабораторної роботи.

Список літератури

1. Організація комп'ютерних мереж: підручник для студ. спеціальності 121 «Інженерія програмного забезпечення» та 122 «Комп'ютерні науки» / КПІ ім. Ігоря Сікорського; Ю. А. Тарнавський, І. М. Кузьменко. – Київ : КПІ ім. Ігоря Сікорського, 2018. – 259 с.

2. Операційні системи : навч. посібник / Б. І. Погребняк, М. В. Булаєнко ; Харків. нац. ун-т міськ. госп-ва ім. О. М. Бекетова. – Харків : ХНУМГ ім. О. М. Бекетова, 2018. – 104 с.

3. Cisco Networking Academy. Cisco Packet Tracer, Tutorials (Getting Started, Logical Workspace, Configuring Devices, Realtime and Simulation Modes) [Електронний ресурс]. – Режим доступу : http://tutorials.ptnetacad.net/tutorials70.htm.

Електронне навчальне видання

Методичні вказівки
до виконання лабораторних робіт
із дисципліни «**Мережеві операційні системи**»
для студентів спеціальності
172 *«Телекомунікації та радіотехніка»*
денної форми навчання

Частина 3

Відповідальний за випуск А. С. Опанасюк
Редактор Н. М. Мажуга
Комп'ютерне верстання О. В. Д'яченка

Формат 60x84/16. Ум. друк. арк. 2,56. Обл.-вид. арк. 2,34.

Видавець і виготовлювач
Сумський державний університет,
вул. Римського-Корсакова, 2, м. Суми, 40007
Свідоцтво суб'єкта видавничої справи ДК № 3062 від 17.12.2007.