Міністерство освіти і науки України

Національний університет «Києво-Могилянська академія»

Факультет інформатики

Кафедра інформатики

Курсова робота

освітній ступінь – бакалавр

на тему: «**Автоматизація крос-дистрибутивного тестування модулів ядра ОС Linux**»

Виконав: студент 3-го року навчання,

Спеціальності

121 «Інженерія Програмного Забезпечення»

Студента Запорожця Дмитра

Керівник Бабич Т.А.

Старий викладач

«1» травня 2025 р.

Київ – 2025

Національний університет «Києво-Могилянська академія»

Факультет інформатики

Кафедра інформатики

Освітній ступінь бакалавр

Спеціальність 121 «Інженерія Програмного Забезпечення»

Освітня програма бакалавр

ЗАТВЕРДЖУЮ

Завідувач кафедри інформатики

Гороховський С. С.

_____

ЗАВДАННЯ

ДЛЯ КУРСОВОЇ РОБОТИ СТУДЕНТУ

Запорожцю Дмитру

1. Тема роботи **«Автоматизація крос-дистрибутивного тестування модулів ядра ОС Linux»**, керівник роботи Бабич Трохим Анатолійович, магістр комп'ютерних наук, асистент

2. Строк подання студентом роботи 2 травня 2025

3. План роботи

Анотація

Вступ

Розділ 1. Дослідження та аналіз предметної області
1.1. Типи віртуалізації та їх застосування для тестування модулів ядра
1.1.1. Порівняння повного підходу та підходу з апаратною підтримкою віртуалізації
1.1.2. Оцінка застосовності обох підходів під час розробки Netfilter-модулів

1.2. Підготовка та конфігурація гостевих віртуальних машин
1.2.1. Автоматизоване створення золотих образів для дистрибутивів Debian та Ubuntu

1.2.2. Використання ключових мережевих схем у тестовому середовищі — NAT, Host-Only, Bridge, переадресація портів та комбіновані конфігурації

1.2.3. Послідовність відновлення системи за допомогою снапшотів: етапи GoldenClean, PreTest та BeforeInsmod

1.2.4. Надійне автоматичне налаштування віддаленого доступу через SSH із обмеженим рівнем sudo, брандмауером UFW, перенаправленням SSH-агента та контролем цілісності

1.3. Bash-пайплайни для збірки та виконання тестів модуля фільтрації ядра

1.3.1. Вимоги до оформлення скриптів та налаштування суворого режиму виконання із застосуванням логування й обробки сигналів

1.3.2. Оновлення Makefile із динамічним визначенням каталогу вихідних файлів ядра, перевіркою конфігурації ядра та активацією DEBUG-режиму

1.3.3. Концептуальна модель життєвого циклу модуля: послідовність складання, розгортання, завантаження й вивантаження та виконання юніт-тестів із дотриманням атомарності, установки тайм-аутів і принципу ідемпотентності

1.3.4. Теоретичні основи створення мінімального Netfilter-модуля: реалізація точок входу, обробка заголовків IP і TCP, безпечний механізм виведення printk та забезпечення сумісності з nf_tables і XDP

1.4. Управління VBoxManage через SSH

1.4.1. Організація зберігання й використання ED25519-ключів у VM Lifecycle Orchestrator із SSH-агентом і перенаправленням агентів, інтеграція з менеджером секретів і керування відомими хостами

1.4.2. Оптимізація передачі файлів: використання ControlMaster, rsync, архівація tar через SSH та утиліта autossh для прискорення SCP і паралельних копіювань

1.4.3. Повний цикл керування гостьовою машиною за допомогою VBoxManage — виявлення віртуальних машин, отримання детальної інформації в машино-зчитуваному форматі, запуск безголової сесії, відновлення стану через снапшоти, керування живленням через ACPI та експорт образів у формат OVA

1.4.4. Розробка Java-обгортки для VBoxManage із надійним API для процесів, аналізом машино-зчитуваного виводу, атомарними файловими операціями та структурованим підходом до логування

Розділ 2. Проєктування та розробка системи

2.1. Архітектура застосунку та технологічний стек

– Main — модуль збору інформації про віртуальні машини, вибору снапшоту та запуску оркестрування

– VMInfo — структура для зберігання назви, статусу, типу ОС, версії ядра та списку снапшотів

– ScriptGenerator — компонент генерації Bash-скриптів за шаблонами

– VmIpDetector та SmartVmIpDetector — двоступеневий механізм отримання IP-адреси гостя із повторними спробами

– Виклики зовнішніх утиліт через ProcessBuilder із підтримкою VBoxManage, SSH, SCP та sshpass

– Apache Maven для побудови проєкту, Java 17+, CLI VirtualBox 7.x, OpenSSH і DKMS у гостевій ОС

2.2. Ручний прототип та перші скрипти

– Створення тестового середовища: WSL з Ubuntu з Microsoft Store і розгортання в VMware Workstation із ISO-образу

– Розробка C-коду Netfilter-модуля та Makefile із перевіркою версії ядра, цілями all, clean, load і unload

– Початкові Bash-скрипти етапів від запуску ВМ до перевірки й тестування модуля

– Копіювання артефактів і SSH-ключів, збірка через make, insmod і rmmod із подальшим тестуванням

– Аналіз виводу системних журналів dmesg, перевірка завантажених модулів lsmod, перевірка фільтрів iptables та реалізація механізму відкату

2.3. Міграція середовища на Ubuntu 24.04 LTS

– Встановлення VirtualBox і доповнень гостевої ОС через офіційні репозиторії, підключення модулів мережевого фільтра та адаптера, пакет virtualbox-guest-dkms

– Перехід із VMware через проблеми з графічним драйвером і спільними папками

– Налаштування Host-Only мережевого інтерфейсу, підключення модулів ядра та реалізація опитування властивостей гостя з контрольованими інтервалами

2.4. Генерація та виконання скриптів за шаблонами

– Організація шаблонів у каталозі src/main/scripts із маркерами для назви ВМ, IP-адреси, імені снапшоту та SSH-ключа

– Підстановка фактичних значень у методі generateScript з використанням заміни рядків і встановленням прав на виконання

– Виконання скриптів через ProcessBuilder із передачею потоків вводу/виводу, обробкою кодів завершення та винятків

2.5. Тестування, обробка помилок та аналіз результатів

– Порівняння базового механізму виявлення IP-адреси гостя та розширеного підходу SmartVmIpDetector із повторними спробами

– Оцінка часу підготовки середовища при ручному налаштуванні та автоматизованому оркеструванні

– Формування структурованих журналів успішних операцій та помилок, очищення тимчасових файлів, відкат стану через снапшоти та видалення резервних артефактів

Висновки

Список використаних джерел

Додатки

ГРАФІК ПІДГОТОВКИ КУРСОВОЇ РОБОТИ ДО ЗАХИСТУ

№ з/п	ПЕРЕЛІК РОБІТ	Термін виконання	Дата ознайомлення наукового керівника	Підпис наукового керівника	Примітки
1.	Визначення теми та її затвердження на кафедрі, призначення наукового керівника; узгодження календарного плану й ознайомлення з критеріями оцінювання роботи.	5 листопада 2024			
2.	Пошук і аналіз джерел: наукової літератури, архівних матеріалів, періодичних видань; збір і систематизація фактів і даних.	6 листопада 2024 – 2 грудня 2024			
3.	Складання докладного плану курсової роботи та його затвердження з науко	6 листопада 2024			
4.	Написання основної частини курсової: підготовка тексту розділів відповідно до затвердженого плану.	6 листопада 2024 – 1 квітня 2025			
5.	Проведення проміжного контролю виконання роботи.	13 січня 2025			
6.	Завершення тексту та передача першого варіанта керівнику для рецензії; врахування зауважень і доопрацювання	2 січня 2025 – 3 квітня 2025			
	Частина 1 (теоретична частина)	22 січня 2025			
	Частина 2 (аналітично-дослідницька частина)	07 березня 2025			
	Частиа 3 (проектно-рекомендаційна частина)	28 березня 2025			
7.	Остаточне оформлення курсової згідно з вимогами та подання на відгук науковому керівнику.	31 березня 2025 – 02 травня 2025			
8.	Подання роботи для перевірки на відповідність нормам академічної доброчесності.	05 травня 2025			
9.	Публічний захист курсової роботи перед екзаменаційною комісією (згідно з розкладом ЕК).	згідно з розкладом роботи ЕК			

Графік узгоджено 5 листопада 2024 р.

Науковий керівник Бабич Трохим Анатолійович

Виконавець курсової роботи Запорожець Дмитро

ЗМІСТ

Анотація

У цій курсовій роботі представлено розробку та впровадження автоматизованого інструменту для розгортання й налаштування віртуальних машин на базі VirtualBox — кросплатформеного рішення з відкритим вихідним кодом для створення й управління віртуальними середовищами, що підтримує різноманітні операційні системи.

Для керування VirtualBox застосовано VirtualBox CLI (Command Line Interface) — інтерфейс командного рядка, який забезпечує виконання операцій через набір консольних команд без використання графічного інтерфейсу.

Спочатку досліджено існуючі підходи до віртуалізації в середовищах Windows (WSL + VMware Workstation) та Linux, визначено переваги й обмеження кожного рішення, після чого обрано VirtualBox як гнучку та безкоштовну платформу. Розроблено Java-застосунок, що на основі шаблонів генерує й виконує Bash-скрипти для:

1. зупинки або відновлення ВМ зі снапшоту;
2. запуску віртуальної машини;
3. копіювання файлів та ключів SSH;
4. збірки та завантаження модулів фільтрації ядра;
5. перевірки стану модуля й результатів тестування.

Застосунок здійснює автоматичне визначення IP-адреси гостьової системи — унікального мережевого ідентифікатора, призначеного операційній системі віртуальної машини в обраній мережевій конфігурації — та ведення журналу подій. Робота демонструє ефективність поєднання Java

та VirtualBox CLI для автоматизації складних операцій у розробці системного ПЗ.

Вступ

Постановка завдання

Розробити автоматизовану систему складання модулів фільтрації ядра

Linux типу Netfilter та стандартизованого середовища для їх розгортання і тестування. Модулі виконують моніторинг мережевого трафіку, виявляючи задані патерни та блокуючи шкідливі пакети, а за потреби — ініціюють виконання зовнішніх утиліт (наприклад, перевірку файлів за допомогою YARA — інструмента для виявлення та класифікації зразків шкідливого ПЗ на основі набору правил) для підтвердження відсутності небажаних об’єктів у системі. Модулі забезпечують оперативний механізм реагування на інциденти до оновлення основних сигнатурних баз.

Мета роботи

Створення інструменту на базі Java та CLI VirtualBox, який забезпечить автоматизовану підготовку віртуального середовища для розробки, збірки та тестування фільраційних модулів.

Завдання дослідження

1. **Аналіз існуючих підходів**
   – Огляд процесів розгортання віртуальних машин у Windows (WSL + VMware Workstation) та Linux.

2. **Вивчення платформи VirtualBox**
   – Дослідження можливостей і обмежень використання VirtualBox CLI для автоматизації життєвого циклу ВМ.

3. **Проектування архітектури Java-застосунку**
   – Розробка модулів для збору метаданих про ВМ, генерації шаблонних Bash-скриптів і керування їх виконанням.

4. **Реалізація функціоналу автоматизації**

   – Автоматичне копіювання вихідних кодів і **SSH-ключів** (криптографічних ключів для безпарольної аутентифікації за протоколом SSH) у ВМ, збірка модулів фільтрації ядра із вихідних кодів мови програмування С під різні версії ядра, завантаження модулів у систему та перевірка їх стану.

Ключові результати

Отримані дані демонструють, що поєднання Java-застосунку з CLI VirtualBox:

- Автоматизує процес підготовки віртуального середовища;
- забезпечує послідовність і стабільність процедур збірки та розгортання.

Розділ 1. Дослідження та аналіз предметної області

1.1. Типи віртуалізації і їхня придатність для тестування модулів ядра

1.1.1 Порівняння повного підходу та підходу з апаратною підтримкою віртуалізації

Гіпервізори діляться на два базові класи. Type 1 («bare-metal») працюють безпосередньо на апаратурі й зазвичай встановлюються на сервери (KVM, ESXi, Hyper-V Server). Type 2 («hosted») запускаються як звичайні застосунки поверх настільної ОС. VirtualBox 7 та VMware Workstation 17 належать саме до Type 2, але спираються на розширення процесорів Intel VT-x чи AMD-V. Це означає, що гостьове ядро Linux отримує потрібні інструкції привілейованого режиму без складної бінарної трансляції (динамічний переклад привілейованих інструкцій з образу гостя в інструкції, зрозумілі апаратурі хоста), а різниця з гіпервізором типу 1 майже непомітна.

Для розробки модулів фільтрації ядра це критично — ми маємо справжнє ядро, можемо вільно завантажувати (insmod, утиліта для вставки модуля ядра .ko) і вилучати (rmmod, утиліта для видалення модуля ядра) будь-які .ko, отримувати kernel panic (критична помилка ядра, яка призводить до зупинки системи та виведення діагностики) і відтворювати граничні помилки (сценарії, що виникають в крайових умовах навантаження або рідкісних конфігураціях), не боячись втратити життєздатність

1.1.2 Практична придатність до розробки Netfilter-модулів

У типовому циклі «компіляція → insmod → fuzzing → rmmod → відкат» найважливіше — швидко повернути чистий стан, тобто відновити гостьову ОС до попереднього знімка з вихідним образом файлової системи, порожньою оперативною пам’яттю та без жодних завантажених

модулів ядра, і при цьому зберегти можливість підвантажувати нові тестові ядра після оновлень.

VirtualBox забезпечує це завдяки відкритому вихідному коду, який дозволяє без змін у самому гіпервізорі інтегрувати модулі під будь-яку версію ядра, автоматичному DKMS-перебудуванню (Dynamic Kernel Module Support — система, що при встановленні або оновленні ядра автоматично перекомпілює та встановлює всі зареєстровані модулі .ko під поточну версію), і механізму guestproperty (вбудований у VirtualBox key–value сховок, який дає змогу без входу в гостьову систему зчитувати та встановлювати параметри — наприклад, IP-адресу чи будь-які інші конфігураційні значення — без додаткових агентів у гостьовій ОС).

VMware натомість пропонує ширшу GPU-інтеграцію, що корисно для тестування прискорених графічних сценаріїв, та зручний автоматичний захист снапшотів, але іноді при стрибках версій Linux-ядра його закриті модулі потребують ручного перебудовування, що ускладнює роботу безперервного VM Lifecycle Orchestrator.

1.2. Підготовка та конфігурація гостьових віртуальних машин

Перш ніж шукати помилки у власних Netfilter-модулях, необхідно бути впевненим, що тестова ОС кожного разу завантажується з абсолютно однаковими параметрами: від конфігурації ядра та версії драйверів до значень sysctl і порядку завантаження сервісів. Це гарантує відтворюваність результатів – якщо під час фузінгу буде виявлено збій або нетипову поведінку, ви точно знаєте, що причина в самому модулі, а не в «плаваючому» середовищі тестового стенду. Нижче описано повний життєвий цикл створення «золотих» образів, вибудовування мережевих топологій, стратегію snapshot-відкату та налаштування безпечного віддаленого доступу.

1.2.1 Автоматизоване створення золотих образів для дистрибутивів Debian та Ubuntu

Концепція «золотий образ» — еталонна ВМ із мінімально необхідним програмним забезпеченням. Знімок зберігається як VDI/VMDK та контрольна точка.

- Практичний рецепт
- Packer або cloud-init: автоматичне розгортання ISO Ubuntu 24.04, встановлення openssh-server, build-essential, linux-headers-$(uname -r), dkms, git, curl; додавання користувача builder з SSH-ключем та sudo NOPASSWD.
- Harden-скрипти після першого reboot: вимкнення зайвих сервісів, очищення журналів, заборона root-login по паролю, скидання /etc/machine-id.
- Консервація образу: вимкнення ВМ, створення snapshot GoldenClean через VBoxManage або vmrun, завантаження дисків у git-LFS чи Nexus для подальшого використання VM Lifecycle Orchestrator.

1.2.2 Мережеві топології для тестового стенду

Kernel-модуль Netfilter взаємодіє з пакунками; важливо моделювати різні режими трафіку.

NAT: гість виходить у WAN через NAT хоста. *Пояснення*: для apt update не треба налаштовувати DNS або маршрути — гість використовує налаштування хоста автоматично, що пришвидшує підключення до зовнішніх репозиторіїв.

Host-only: віртуальний свіч між гостем(ами) та хостом. *Пояснення*: ідеально для ручних експериментів з Scapy чи інших фреймворків, оскільки трафік обмежено приватною підмережею без доступу в інтернет.

Bridged: гість отримує IP в тій самій фізичній мережі, що й хост. *Пояснення*: поведінка максимально наближена до фізичної машини — ви бачите ті ж маршрути, правила маршрутизації та міжмережеві пристрої.

Port-forward: перенаправлення портів у NAT-режимі, наприклад VBoxManage modifyvm <vm> --natpf1 "ssh,tcp,,2222,,22". *Пояснення*: дозволяє тестувати зовнішній доступ до сервісів гостя, імітуючи публічний порт у DMZ.

Комбіновані схеми: дві NIC — NAT для оновлень і Host-only для контрольованого трафіку (наприклад, між кількома гостями).

1.2.3 Схема використання шнапшоту як механізму відновлення

Під час фузінгу модуль може «збити» ядро (викликати **kernel panic** або критичний збій). Щоб не втрачати час на ручне відновлення, налаштовуємо ланцюжок знімків.

- **GoldenClean**: точка одразу після hardening, без модулів.

- **PreTest**: клон GoldenClean перед кожною тестовою сесією з додатковими залежностями поточної гілки.
- **BeforeInsmod**: (опційно) знімок безпосередньо перед insmod.
  Пояснення «збиває ядро»: якщо під час insmod або фузінгу ядро викидає panic або втрачає працездатність — це і є «збій». Використовуючи цей знімок, ми відкотимося безпосередньо до стану перед встановленням модуля.
- **Автоматичний відкат**: Bash-скрипт завершує тест, виконує VBoxManage snapshot <vm> restore PreTest, вимикає ВМ (acpipowerbutton або poweroff), а потім знову запускає чистий клон. Відновлення знімка займає кілька секунд.

1.2.4 Автоматичне та безпечне налаштування віддаленого доступу

Безпечний SSH-доступ до VM потрібен для віддаленого керування та інтеграції в VM Lifecycle Orchestrator:

- **Генерація ключів**: Packer створює SSH-пару під час збірки, публічний ключ додається до /home/builder/.ssh/authorized_keys.
- **Sudo без пароля**: у /etc/sudoers.d/builder прописано builder ALL=(ALL) NOPASSWD: /usr/bin/insmod, /usr/bin/rmmod, /usr/bin/make. Інші команди вимагають пароль.
- **Базовий firewall (ufw)**:
  - ufw default deny incoming — відхилити всі вхідні за замовчуванням.
  - ufw allow 22/tcp — дозволити SSH.
  - ufw allow proto all from <test-host-IP> to any comment 'nf_test_control' — правило для керованого трафіку Netfilter тестів.
    Пояснення: це обмежує доступ до гостьової ОС лише з відомих джерел та лише по потрібних портах.
- **ssh-agent-forward** у VM Lifecycle Orchestrator: приватний ключ не зберігається на диску гостя — він пробросується через агента, що мінімізує ризик витоку.
- **Перевірка цілісності при старті**: системний systemd-таймер виконує скрипт, який хешує збірку модуля (sha256sum module.ko), порівнює з еталонним хешем, і якщо файл змінився — автоматично створює новий snapshot BeforeInsmod для безпечного відкату.

1.3. Bash-пайплайни збірки та прогону тестів модулів фільтрації ядра

1.3.1 Стандарти оформлення Bash-скриптів

Розробляючи сценарії для інфраструктури, де кожен збій може призвести до зависання ядра, важливо завжди вибирати політику **«fail fast»** — підхід, за якого скрипт або сервіс при найперших ознаках будь-якої помилки негайно припиняє свою роботу замість того, щоб продовжувати з непередбачуваним станом. Це дає змогу одразу виявити джерело проблеми й ізолювати його, не витрачаючи час на ланцюгові наслідки початкового збою.

Конструкція set -euo pipefail(пояснити) — збирає три опції Bash для забезпечення суворого режиму виконання:

- Інструкція -e гарантує, що процес зупиниться на першому ненульовому статусі,
- -u — що звернення до необ’явленої змінної не конвертується мовчки у порожній рядок,
- а pipefail — що помилка в «ранній» ланці конвеєра не буде прихована успіхом останньої.

Такий «режим підвищеної строгості» бажано доповнювати однаковими правилами логування. Лог має бути машинно-чита́ним (ISO-8601 timestamps, рівні INFO/WARN/ERROR) і водночас зручним для людини. Для цього у функцію-обгортку, умовно log(), закладають префікс із часовою міткою і, за потреби, ідентифікатором конкретного запуску. Відповідно, всі події перетворюються на потік повідомлень одного формату, який легко фільтрувати, агрегувати й переглядати.

Ще одна обов’язкова практика — єдине місце обробки фатальних помилок. Використання конструкції trap 'handler $LINENO' ERR, тобто ми

гарантуємо, що при будь-якій помилці (сигнал ERR) виконається функція handler, якій передається номер рядка $LINENO, де сталося збою. У ній можна вивести детальну діагностику — яка саме команда не спрацювала і на якому рядку. Це пришвидшує як локальне, так і віддалене (VM Lifecycle Orchestrator) налагодження.

1.3.2 Актуалізація Makefile та підстановки, прив'язані до uname –r

Makefile — це текстовий файл, який використовується утилітою make для автоматизації процесу збірки програмного забезпечення. У ньому визначаються цілі (targets), залежності між файлами та правила для перетворення вихідних файлів (наприклад, .c) на кінцеві артефакти (наприклад, .ko).

У розробці модулів ядра одним із головних ризиків є неочікувана невідповідність між конфігурацією збірки поточного ядра та шляхами або символами, які використовує Makefile. Навіть зміна одного прапора, наприклад CONFIG_NETFILTER=y→n, або інша версія компілятора GCC може зробити згенерований файл .ko несумісним. Тому Makefile повинен бути «самовідновним» і щоразу перед побудовою перевіряти, що всі шляхи й параметри справді відповідають поточній версії ядра.

В основі підстановок лежить стандартна конструкція

```
KDIR := /lib/modules/$(shell uname -r)/build
```

Змінна `KDIR` визначає шлях до каталогу з вихідними файлами та заголовками поточного ядра, які необхідні для побудови модуля. Зазвичай це символьне посилання `/lib/modules/$(uname -r)/build`, що вказує на директорію, де знаходяться файли конфігурації та заголовки версії ядра, запущеної в системі.

де uname -r повертає точно той рядок, під яким ядро було зібрано (наприклад, 5.18.0-rc6-custom), а директива shell запускає цю команду в оболонці. В результаті змінна KDIR завжди вказує на каталог, де лежать заголовки та файли побудови поточного ядра.

Перед тим як викликати make -C $(KDIR) M=$(PWD) modules, Makefile може включати блок валідації:

```
ifeq ($(wildcard $(KDIR)/.config),)
$(error Заголовки для ядра $(shell uname -r) не знайдено в $(KDIR).build)
endif
```

Тут wildcard перевіряє наявність файлу .config у каталозі заголовків; якщо його немає, збірка зупиняється з чітким повідомленням про помилку. Така перевірка гарантує, що каталог $(KDIR) коректно вказаний і містить ту саму конфігурацію, яка була використана під час компіляції ядра.

Далі важливо переконатися в активності ключових опцій конфігурації, наприклад:

```
NETFILTER_ENABLED := $(shell grep -E '^CONFIG_NETFILTER=y'
$(KDIR)/.config || true)
ifeq ($(NETFILTER_ENABLED),)
$(error У поточному .config не знайдено CONFIG_NETFILTER=y —
неможливо збирати Netfilter-модуль)
endif
```

Така конструкція виконує команду grep над файлом .config і у разі відсутності відповідного рядка призупиняє збірку з описом проблеми. Альтернативно, якщо вимірювання часу дозволяє, Makefile може

самостійно підхопити збережений файл old.config, виконати make oldconfig у каталозі KDIR і згенерувати коректні заголовки.

Ще один аспект — підтримка режиму налагодження без потреби ручного редагування Makefile. Замість того, щоб писати в коді

```
EXTRA_CFLAGS := -DDEBUG
```

необхідно проксирувати зовнішню змінну:

```
ifdef DEBUG
   EXTRA_CFLAGS += -DDEBUG
endif
```

Тоді виклик make DEBUG=1 автоматично додає до прапорів компілятора -DDEBUG, а звичайна команда make залишиться в production-режимі без дебаг-символів. Такий підхід уніфікує збірку для відлагоджувальних та релізних варіантів без зміни вихідного файлу.

Перевага описаних механізмів полягає в тому, що джерелом істини залишається тільки поточне ядро. Makefile щоразу використовує значення uname -r для підстановки шляху до заголовків, перевіряє наявність налаштувань і лише потім починає збірку. Цей «самопристосовний» підхід запобігає помилкам несумісності між різними версіями ядра, знімає потребу у жорсткому прописуванні шляхів і зменшує кількість некоректних модулів у VM Lifecycle Orchestrator.

1.3.3 Ланцюг «build → deploy → insmod/rmmod → unit-tests» як концептуальна модель

Повний процес роботи з модулем ядра можна уявити як детермінований автомат із чотирма фазами. Кожна фаза має чітко визначені **вхід** і **вихід**:

- **Build**
    - **Вхід:** вихідні файли .c і актуалізований Makefile
    - **Вихід:** скомпільований модуль .ko, ідентичний при повторних збірках (ідемпотентність)

Ідемпотентність означає, що незалежно від кількості запусків, за умови незмінних вхідних даних, результат (байт-у-байт вміст ELF-модуля або його хеш) залишається незмінним. Це дозволяє використовувати кешування артефактів та одразу виявляти невідповідності в середовищі збірки.

- **Deploy**
    - **Вхід:** скомпільований модуль .ko
    - **Вихід:** модуль «атомарно» скопійований у файлову систему гостьової ОС

Атомарна копія гарантує: або файл повністю доставлено, або він не з’явився взагалі. В разі механічного переривання копіювання уникнути появи «половинчастого» .ko, яке може призвести до невизначеної поведінки при завантаженні.

- **insmod / rmmod**
    - **Вхід:** присутній на диску гостьової ОС модуль .ko
    - **Вихід:** модуль завантажено в ядро або безпечно вилучено з нього
- Перед викликом insmod слід перевірити, що модуль із таким іменем ще не завантажений і немає конфлікту символів. При видаленні заборонено маскувати помилки командою типу sudo rmmod

modulename || true оскільки це приховає ситуацію, коли модуль перебуває в стані **in-use** (заблокований ресурсами), що ускладнить діагностику. Натомість необхідно зловити код помилки та зафіксувати її в логах.

- **Unit-tests**
    - **Вхід:** завантажений у ядро модуль
    - **Вихід:** детальний звіт, що містить інформацію про проходження чи відмову кожного тесту

Принцип «один тест — одна точка відмови» означає, що кожен скрипт перевіряє лише одну властивість модуля (наприклад, правильність обробки певного netfilter-ланцюга). Це дозволяє одразу ідентифікувати причину невдачі. Крім того, для уникнення «зависань» в межах ядра кожному тесту задається жорсткий таймаут: якщо модуль не відповідає протягом відведеного часу (наприклад, 30 секунд), тест фіксує помилку і завершується. Інакше «зависле» ядро паралізує весь пайплайн.

Після завершення тестів автоматично виконується rmmod, щоб повернути ядро в чистий стан: кожен прогін починається з нуля, без залишків попередніх структур у пам’яті. Це гарантує, що від однієї ітерації до наступної немає «рецидивів» у вигляді забутого залишкового стану модуля.

1.3.4 Теоретичні засади написання мінімального Netfilter-модуля

Модуль підключає два набори заголовків – <linux/*.h> для базових утиліт ядра (реєстрація, атрибути, Netfilter API, доступ до struct sk_buff) і <net/*.h> для внутрішніх мережевих хелперів(network helpers) — набору вбудованих у ядро функцій для розбору мережевих заголовків (IP, TCP), розрахунку контрольних сум, підтримки фрагментації й маршрутизації без прямого маніпулювання сирими байтами в sk_buff. Завдяки хелперам розробник

уникає помилок із виходом за межі буферу (OOB) та отримує готові до використання структури заголовків. Перевірка версії ядра не потрібна, оскільки використовується стабільний Netfilter API.

Суть отримання сповіщення на обробку пакета описує структура nf_hook_ops, де вказується функція зворотного виклику, точка вставки (наприклад, рання стадія PREROUTING для IPv4), сімейство протоколів (PF_INET) і пріоритет (щоб опрацювати пакет перед стандартними таблицями).

Логіка функції-обробника виглядає так:

- Перевірка, що skb не NULL.
- Фільтрація лише TCP-пакетів (перевірка поля protocol).
- Використання ip_hdr() і tcp_hdr(), які самі перевіряють межі буферу.
- За потреби — трасувальне логування (printk(KERN_INFO…), безпечний формат %pI4).
- Повернення NF_ACCEPT для мінімального втручання; уникання необґрунтованого NF_DROP.

Життєвий цикл модуля:

- При ініціалізації реєструється обробник (успіх викликає завантаження, за помилки — відмова).
- При видаленні —обробник знімається; жодних “сплячих” блокувань у контексті RCU.

Увага до сумісності: нові можливості зосереджені в nf_tables (Netlink-інтерфейс), але nf_register_net_hook() залишається підтримуваним; для контейнерів варто прив’язуватися до конкретного network namespace; у разі

XDP-прискорення Netfilter PREROUTING може спрацювати після XDP-хуків.

Отже, навіть найпростішому модулю потрібно поєднати інфраструктурний підхід (детерміноване збирання, тести, Makefile) із чітким розумінням точки входу в мережевий стек, гарантій виконання API та запобігання гонкам, витокам і OOB-доступам.

1.4 Керування VBoxManage через SSH

1.4.1 Управління ключами SSH у сценаріях VM Lifecycle Orchestrator

У VM Lifecycle Orchestrator парольна автентифікація майже непридатна: автоматизувати MFA важко, а зберігати паролі в репозиторії небезпечно — адже кодова база може бути скомпрометована або репозиторій отримати непередбачений доступ, а історія комітів назавжди зберігає ці секрети, що значно ускладнює їх своєчасне оновлення та відкликання. Тому асиметричні ключі SSH стали «де-факто» стандартом: вони легко генеруються, зручно відкликаються, добре логуються й не потребують синхронізації сторін. Нижче подано практичні рекомендації, що забезпечують безпечний і відтворюваний доступ до віртуальних машин-агентів протягом усього життєвого циклу збірки.

Чому саме ED25519

OpenSSH підтримує механізм ED25519 із версії 6.5. Він дає коротший відкритий ключ, ніж RSA-2048, не спирається на SHA-1 і швидше виконує підпис на слабших CPU. Для VM Lifecycle Orchestrator середовища це означає менше мережевих витрат і швидший handshake.

```
ssh-keygen -t ed25519 -a 100 \
  -C "orchestrator-${ORCHESTRATOR_RUN_ID}:${VM_NAME}" \
  -N "" \
  -f "${HOME}/Cursach/${VM_NAME}_key"
```

• ‘-a 100’ підвищує скадність перебору фрази-пароля, навіть якщо ключ збережеться на диск.
• Коментар у форматі «build-#PIPELINE:repo» відразу прив’язує ключ до конкретного запуску, що спрощує аудит та прибирання непотрібних записів в authorized_keys.

Зберігання та використання приватного ключа

Використовуємо зовнішній Secrets Manager (HashiCorp Vault, AWS Secrets Manager або GitLab Variables), в якому ключі зберігаються з прапорами masked і protected. Opchestrator під час виконання отримує їх через API, тому вміст ніколи не потрапляє в логи й не розкривається при форках проєктів.

Opchestrator завантажує приватний ключ у робоче середовище тільки перед виконанням сценаріїв (runScript). Якщо додати його на стадії побудови образу або збірки контейнера, ключ потрапить у шар кешу й залишиться там назавжди.

У папці ~/.ssh opchestrator викладає персональний known_hosts із фіксованими fingerprint кожної ВМ. Це гарантує захист від MITM-атак навіть якщо runner/оркестратор обслуговує кілька проєктів.

SSH-agent і agent forwarding

У контексті CI-пайплайнів, визначеної послідовності автоматизованих кроків, які виконуються щоразу, коли в репозиторій надходить новий код,

у VM Lifecycle Orchestrator використовують фоновий демон **SSH agent**, що зберігає в оперативній пам’яті приватні ключі замість того, щоб тримати їх у файлах на диску. Приватний ключ (файл ssh_key) — це згенерований за допомогою ssh-keygen набір біт, який співвідноситься з публічним ключем у authorized_keys гостьової ОС.

При старті runner виконує:

```
eval "$(ssh-agent -s)"
```

```
ssh-add /path/to/ssh_key
```

— це завантажує приватний ключ у агент і запам’ятовує його в пам’яті процесу **ssh-agent**.

Під час підключення до VM додають опцію

```
-o ForwardAgent=yes
```

або ставлять у конфігурації SSH ForwardAgent yes. Це означає, що коли гостева машина потребує автентифікації, SSH-клієнт у хості перенаправляє запит на підпис даних безпосередньо до локального SSH agent, замість того щоб копіювати приватний ключ у середовище VM.

Після завершення необхідних операцій (наприклад, git clone або scp) слід очистити агент:

```
ssh-add -D   # видалити всі ідентифікатори
```

```
pkill ssh-agent
```

Таким чином приватний ключ існує лише протягом життя пайплайна, а не на диску VM, що значно звужує поверхню атаки. Однак **agent forwarding** краще вмикати лише для короткочасних команд, оскільки довільний код у гостьовій ОС може скористатися агентом і підписати будь-які дані.

Моделі розподілу ключів

Модель	Переваги	Недоліки	Де доречна
Один ключ → одна ВМ	Легко відкликати, мінімальний blast-radius	Багато ключів	Постійні production-агенти
Один ключ → стадія	Менше метаданих, швидший деплой	Ширший blast-radius	Ефемерні build-хости

У практиці курсової застосовано змішаний підхід. Базовий «золотий» образ ubuntu-24.04-golden має довгоживучий master-ключ. Перед кожним тестом VM Lifecycle Orchestrator знімає snapshot, генерує тимчасову пару, додає її лише до клонованої ВМ і видаляє під час rollback-у snapshot-у.

1.4.2 Оптимізація передачі файлів: використання ControlMaster, rsync, архівація tar через SSH та утиліта autossh для прискорення SCP і паралельних копіювань

Порт-форвардинг по SSH дозволяє проксувати трафік через захищений канал, створений між локальною машиною і віддаленим хостом. Це забезпечує шифрування всього переданого трафіку та контроль доступу за допомогою тих самих механізмів аутентифікації, що й для звичайного SSH-підключення. Існує три основні режими порт-форвардингу:

1. **Local Forwarding (−L)**

   У цьому режимі певний порт на локальній машині зв’язується через

SSH-канал із портом на віддаленій машині (тут — з сервісом, що працює всередині ВМ). Наприклад, якщо за замовчуванням на ВМ відкритий HTTP-сервіс на порті 80, то, використавши local forwarding, ми можемо спрямувати запити з локального 8080 безпосередньо в гостьову систему, попередньо захистивши їх SSH-шифруванням.

2. **Remote Forwarding (−R)**
   У цьому випадку порт на віддаленому хості (від VM) прокситься в мережу локальної машини. Це корисно, коли необхідно надати доступ до локальних сервісів із гостьової ОС без прямого відкриття портів у брандмауері. Такий підхід застосовується, наприклад, для організації безпечного доступу до внутрішніх інструментів розробника через VM.

3. **Dynamic Forwarding (−D)**
   Дозволяє налаштувати SOCKS-проксі, який динамічно встановлює з’єднання з будь-яким кінцевим сервером за запитами клієнта. Під час роботи всі вихідні з’єднання на локальний SOCKS-порт розшифровуються і переадресовуються через SSH до віддаленої сторони. Таким чином можна захистити будь-який клієнтський трафік, не задаючи вручну кожен окремий порт.

Для забезпечення **стійкості** SSH-тунелів тобто зашифрованих TCP-каналів, через які пробросуються порти (Local, Remote або Dynamic Forwarding) усередині одного SSH-з’єднання, доцільно використовувати утиліту autossh. Вона відстежує стан SSH-каналу й автоматично перезапускає тунель при обриві. Ключові моменти налаштування:

- **Моніторинг каналу:** autossh використовує внутрішній моніторинговий порт, щоб перевіряти активність SSH-тунелю. При виявленні збоїв утиліта перезапускає тунель.
- **KeepAlive та таймаути:** рекомендується додати параметри ServerAliveInterval і ServerAliveCountMax у SSH-конфігурацію, щоб швидше виявляти недоступність каналу.
- **Фоновий режим:** autossh зазвичай запускається з опціями −f (фон) і −N (не відкривати shell), що відповідає задачі лише підтримки тунелю.
- **Повторні спроби:** можна задати інтервал повторного підключення через опцію −M (моніторинг) або шляхом конфігурації сервера за допомогою параметрів Autoreconnect у SSH-конфігурації.

Таким чином, поєднання SSH-форвардингу і autossh гарантує безпечний і надійний канал для доступу до сервісів всередині VM, навіть якщо мережеві з’єднання нестабільні або мають періодичні збої.

1.4.3 Повний життєвий цикл ВМ через VBoxManage

У більшості реалізацій VM Lifecycle Orchestrator саме VirtualBox виконує роль «легковагого хмарного провайдера», тобто виконує практично ті самі базові функції, що й публічні хмарні сервіси (типу AWS EC2 або Google Compute Engine) — надає API для динамічного створення, запуску, зупинки та скасування віртуальних машин, але в локальному середовищі, без усієї великої інфраструктури, мережевих надбудов і масштабованості справжньої хмари, а VBoxManage — його CLI-шлюз. Повний цикл обслуговування машини включає інвентаризацію, запуск у headless-режимі,

виконання робочого навантаження, відкат до початкового стану та очищення диска.

Інвентаризація здійснюється через VBoxManage list vms, що повертає пари ім’я–UUID; далі за допомогою showvminfo --machinereadable отримують поля у форматі key=value для перетворення в JSON або відповідну структуру.

Запуск машини в headless-режимі (startvm <uuid> --type headless) поєднується з очікуванням появи IP-адреси через guestproperty wait <uuid> /VirtualBox/GuestInfo/Net/0/V4/IP, що усуває необхідність в довільних затримках.

Після встановлення SSH-доступу виконуються тестові сценарії, збирають логи й метрики (час завантаження, CPU-навантаження, I/O) і передають результати в систему моніторингу.

Відкат до чистого стану реалізується снапшотом (snapshot take clean_state), відновлення – snapshot restore clean_state. Для уникнення фрагментації та накопичення дельта-дисків періодично виконується клонування образу з опцією clonevm.

Коректне завершення роботи гостя здійснюється через сигнал ACPI – відкритий стандарт для керування живленням та конфігурацією апаратного забезпечення, який визначає, як операційна система обмінюється з BIOS або віртуалізатором подіями живлення. Сигнал ACPI Power Button імітує натискання фізичної кнопки живлення на корпусі комп’ютера. Коли VirtualBox виконує команду

VBoxManage controlvm <uuid> acpipowerbutton

він надсилає у віртуальну машину подію «Power Button Pressed» через ACPI-шину. Гостьова ОС, отримавши цю подію, запускає звичайну процедуру коректного вимкнення (виконання скриптів завершення роботи, синхронізація файлових систем та очищення кешів). Таким чином, замість примусового відключення (poweroff) система гасне без ризику пошкодження даних і файлових систем.

Очищення накопичених змін за розкладом передбачає експорт базових образів у OVA - єдиний архівний файл (зазвичай у форматі TAR), що містить усі компоненти віртуальної машини у стандарті OVF (Open Virtualization Format), видалення старих дисків і імпорт нового чистого віртуального середовища. У проєктах великого масштабу додатково використовують внутрішні позначки стану (guestproperty set), які дозволяють зберігати диски для невдалих циклів і скидати їх після успішних.

1.4.4 Java-обгортка для VBoxManage

У рамках уніфікації управління віртуальними машинами в VM Lifecycle Orchestrator ми переносимо виклики CLI VBoxManage із shell-скриптів у Java-бібліотеку. Це дозволяє застосувати потужні інструменти платформи JVM для надійного виконання зовнішніх команд, обробки результатів і керування артефактами без прямих вставок «system calls» у конвеєр. Для запуску зовнішніх команд використовується стандартний клас запуску процесів, що приймає список аргументів та гарантує захист від командної ін’єкції. Об’єднання потоків помилок і стандартного виводу і подальше покрокове читання результатів надає централізований контроль над таймаутами та кодами завершення зовнішніх викликів. Результат опції “machinereadable” представляється як послідовність рядків “ключ=значення”. Використання шаблонів пошуку та витягнення пар

ключ–значення забезпечує гнучкий та стійкий до змін форматів спосіб перетворення сирого тексту у словникову структуру властивостей віртуальної машини.

Усі маніпуляції із ключами та тимчасовими скриптами здійснюються через високорівневий API для роботи з файлами й каталогами. Це дозволяє гарантувати атомарність перейменування застарілих файлів, створення необхідних директорій та запис нових артефактів із належними правами виконання, водночас мінімізуючи ризик залишити чутливі дані на диску. Надійність отримання IP-адреси гостя забезпечується подвійною перевіркою: спочатку читається відповідна властивість, а у разі відсутності відповіді ініціюється циклічний опит із затримкою. Такий підхід дозволяє врахувати варіативність часу старту ОС гостя без втрати стабільності конвеєра.

Структурований «debug-логи» про ключові кроки виконання та проміжні значення змінних формуються за допомогою простого механізму виводу в консоль із зазначенням імен кроків і контексту. Це дає змогу швидко виявляти слабкі місця та інтегруватися в загальну систему моніторингу VM Lifecycle Orchestrator.

Основні характеристики стеку технологій

- **Портативність**: вся логіка упаковується в один Java-архів, незалежно від середовища виконання.
- **Тестованість**: окремі компоненти (запуск процесів, парсинг, файлові операції) легко покривати юніт-тестами та мокувати.
- **Безпека**: аргументи зовнішніх команд передаються як окремі елементи списку, а доступ до чутливих ресурсів виключно через пам’ять або ізольовані директорії.

- **Масштабованість і моніторинг**: архітектура дає можливість безболісно додавати підтримку нових команд, а точки інтеграції з бібліотеками метрик спрощують побудову дашбордів та тривожних сповіщень.

Завдяки глибокому застосуванню Java-технологій (платформа для запуску процесів, високорівневий файловий API, шаблонний парсинг, структуровані DTO) ми отримуємо надійну, безпечну та керовану обгортку над VBoxManage, що відповідає вимогам сучасної DevSecOps-культури.

Розділ 2. Проектування та розробка системи

У цьому розділі докладно розглянуто архітектуру розробленого Java-застосунку, його ключові модулі та алгоритми, а також спосіб інтеграції з VirtualBox CLI та Bash-скриптами. Описано перехід від ручних прототипів до повністю шаблонізованого автоматизованого workflow, наведено приклади коду та механізми обробки помилок.

2.1. Архітектура застосунку та технологічний стек

У рамках розробки системи керування віртуальними машинами застосовано модульно-орієнтовану архітектуру, що складається з п’яти основних компонентів. Головним елементом є клас Main, який виступає точкою входу в додаток та послідовно виконує такі операції: збір списку доступних віртуальних машин і їхнього стану; вибір цільової машини та відповідного снапшоту; генерація та виконання скриптів step1…step4; очищення тимчасових файлів по завершенні всіх кроків.

Для формалізованого зберігання властивостей віртуальних машин запроваджено запис VMInfo (record), що містить поля name, status, osType, kernelVersion та snapshots. Генерація Bash-скриптів реалізована в класі

ScriptGenerator, який використовує шаблони з підстановкою параметрів, записує файли на диск і встановлює для них атрибут виконуваності.

Підсистема визначення IP-адреси гостьової ОС побудована за двоступеневим принципом: базовий VmIpDetector здійснює виклик

VBoxManage guestproperty get <VM_NAME> /VirtualBox/GuestInfo/Net/0/V4/IP

та парсить рядок Value: <IP>; за необхідності його розширює SmartVmIpDetector, який очікує завершення ініціалізації мережевого стеку. У разі невдачі обидва забезпечують піднімання винятку з діагностичним повідомленням.

Запуск зовнішніх процесів — VBoxManage, ssh, scp, sshpass та інші власні сценарії — здійснюється через стандартний клас ProcessBuilder (Java SE), що гарантує коректне перенаправлення stdout/stderr та обробку кодів завершення.

Технологічний стек включає Java 17+ (LTS) для реалізації бізнес-логіки та ProcessBuilder із Java SE для безпечного виклику зовнішніх команд; Apache Maven використовується для побудови проєкту й управління залежностями. Взаємодія з гіпервізором здійснюється через Oracle VirtualBox 7.x та його CLI-інтерфейс VBoxManage, а шаблонні скрипти генеруються на Bash із динамічною підстановкою параметрів. Віддалений доступ і копіювання файлів реалізовано за допомогою OpenSSH (ssh, scp, sshpass) із парно-ключовою ED25519-аутентифікацією, яку генерує ssh-keygen. Для автоматичного складання та поновлення модулів у гостьовій ОС застосовується DKMS у поєднанні з необхідними заголовками ядра.

2.2. Ручний прототип і перші скрипти

На Windows 10 увімкнення підсистеми WSL виконується через «Програми та засоби» → «Увімкнення або вимкнення засобів Windows» → Windows Subsystem for Linux, після чого система перезавантажується. У Microsoft Store обирається дистрибутив Ubuntu (наприклад, 20.04 LTS) і встановлюється пакет. Для створення віртуальної машини в VMware Workstation завантажується ISO-образ Ubuntu з офіційного сайту, створюється нова ВМ із призначеним образом, налаштовуються параметри процесора, пам’яті та дискового простору, після чого запускається інсталяція в графічному режимі.

Структура С-коду та Makefile для Netfilter

• netfilter.c: реалізує hook у точці NF_INET_PRE_ROUTING, перевіряє наявність IPv4/TCP-пакета, виводить debug-повідомлення через printk(KERN_INFO) і завжди повертає NF_ACCEPT; ініціалізація модулю здійснюється через module_init, деініціалізація — через module_exit.
• Makefile: перевіряє, що ОС — Linux, приймає змінну MODULE_NAME; будує модуль командою
make -C /lib/modules/$(uname -r)/build M=$(PWD) modules
Після компіляції переміщує файл .ko у відповідні каталоги build-production або build-debug; містить цілі all (збірка), clean (очищення артефактів), load (insmod), unload (rmmod).

Bash-скрипти (step1_start_vm.sh…step4_check_and_test.sh)

1. step1_start_vm.sh
   – Використовує строгий режим: set -euo pipefail
   – Перевіряє стан ВМ (running, paused, stuck) і, за потреби, виконує VBoxManage controlvm <VM_NAME> poweroff у циклі очікування з повторною перевіркою статусу

– Відновлює снапшот VBoxManage snapshot restore та запускає ВМ у GUI-режимі через VBoxManage startvm

2. step2_copy_files.sh

   – Налаштовує SSH-ключі: якщо поточний публічний ключ співпадає з новим — використовує sshpass для передачі ключа за паролем; інакше — копіює через scp із вказаним приватним ключем

   – Створює каталог у гостьовій ОС та копіює вміст локальної директорії до віддаленої через scp

3. step3_build_and_load.sh

   – Підключається по SSH, встановлює пакети build-essential та заголовки ядра

   – Виконує make у каталозі netfilter, переносить згенерований модуль .ko у build-директорію

   – Завантажує модуль за допомогою insmod; у разі потреби видаляє попередню версію через rmmod; перевіряє завантаження через lsmod

4. step4_check_and_test.sh

   – Перевіряє статус модуля за допомогою lsmod

   – Аналізує останні рядки журналу dmesg для виявлення можливих помилок

   – Виконує базовий тест iptables -L для підтвердження відсутності конфліктів із існуючими правилами firewall

2.3. Міграція середовища

Оскільки систему будуть використовувати системні адміністратори з різним досвідом роботи під Linux, було прийнято рішення перенести весь функціонал застосунку на платформу Ubuntu 24.04 LTS. Це забезпечує уніфіковане середовище, зручне управління через пакети DEB,

довгострокову підтримку ядра й стандартизований набір інструментів адміністрування. Первинна спроба розгорнути середовище на VMware виявилася проблемною через несумісність модулів vmwgfx із ядром нової версії Ubuntu та нестабільну інтеграцію субсистеми спільних папок, тому було ухвалено рішення перейти на VirtualBox, який у поточній LTS-конфігурації забезпечив стабільну роботу й необхідний набір можливостей.

1. Налагодження VirtualBox у Linux
   – Для встановлення використовувався офіційний репозиторій Ubuntu:
   sudo apt update && sudo apt install virtualbox virtualbox-guest-dkms virtualbox-guest-utils
   – Після інсталяції перевірено версію за допомогою
   VBoxManage --version
   – Увімкнено служби vboxdrv, vboxautostart та перевірено їхній статус через systemctl.
   – Для доступу до USB-пристроїв і спільних папок учасника проекту додано до групи vboxusers:
   sudo usermod -aG vboxusers $USER
   Після цього користувач повинен був вийти та зайти в систему повторно, щоб набули чинності групові права.
2. Проблеми з мережевим інтерфейсом та гостьовими властивостями
   – У якості основи мережевих підключень обрано режим Host-only network, оскільки він дозволяє напряму взаємодіяти з гостьовою ОС без залежності від зовнішньої інфраструктури.
   – Завантажено й налаштовано модулі ядра:
   sudo modprobe vboxnetflt
   sudo modprobe vboxnetadp
   Додано до /etc/modules для автоматичного підключення після

перезавантаження.

– Для коректного зчитування властивостей гостьової ОС (IP-адреса, інформація про спільні папки) використовується механізм гостевих властивостей VirtualBox:

VBoxManage guestproperty set <VM_NAME>
/VirtualBox/GuestInfo/Net/0/V4/IP <value>
VBoxManage guestproperty set <VM_NAME>
/VirtualBox/GuestInfo/SharedFolders/Mount <mount-point>

– У деяких випадках властивості з’являються із затримкою через інерцію завантаження гостевих модулів. Щоб усунути цю проблему, у Java-модулі реалізовано цикл опитування зі статичним інтервалом 1 s і загальним таймаутом 60 s, який періодично викликає

VBoxManage guestproperty get …

і перевіряє наявність валідного значення; у разі перевищення таймауту піднімається виняток із діагностичною інформацією для подальшого аналізу.

2.4 З’єднання віртуальної машини з хост-системою

Для забезпечення взаємодії між хост-системою та гостьовими дистрибутивами Ubuntu та Debian, віртуальна машина має бути належним чином підключена до мережі, а також повинні бути встановлені додаткові компоненти, які забезпечують повноцінну інтеграцію з хостом.

2.4.1 Налаштування мережевого інтерфейсу

За замовчуванням VirtualBox створює інтерфейс NAT, що дозволяє гостьовій системі виходити в інтернет, але ускладнює з’єднання з нею з боку хоста. Для забезпечення двостороннього з’єднання доцільно використовувати один із двох підходів:

- **Bridged Adapter** — гостьова система отримує IP-адресу в тій же підмережі, що й хост.
- **Host-Only Adapter** — створюється окрема внутрішня мережа, доступна лише між хостом і гостьовими системами.

У даній роботі застосовується конфігурація з **Host-Only Adapter**, що дозволяє забезпечити прямий доступ до гостьової системи через IP-адресу, незалежно від зовнішнього мережевого середовища.

2.4.2 Встановлення SSH-з’єднання між хостом та ВМ

Для налаштування безпечного каналу зв’язку між хостом і ВМ використовується протокол SSH. З боку хост-системи генерується пара ключів:

```
mkdir -p "$HOME/Cursach"
```

```
ssh-keygen -t ed25519 -f "$HOME/Cursach/<VM_NAME>_key" -N "" -C "<VM_NAME>@$(hostname)"
```

Після встановлення openssh-server у гостьовій системі та налаштування конфігураційного файлу /etc/ssh/sshd_config, сервер перезапускається:

```
sudo apt update && sudo apt install openssh-server -y
```

```
sudo sed -Ei 's/^#?PasswordAuthentication.*/PasswordAuthentication no/' /etc/ssh/sshd_config
```

```
sudo systemctl restart ssh
```

Публічний ключ з хост-системи передається на ВМ:

```
ssh <VM_USER>@<VM_IP> "mkdir -p ~/.ssh && chmod 700 ~/.ssh"
```

```
scp "$HOME/Cursach/<VM_NAME>_key.pub"
<VM_USER>@<VM_IP>:~/.ssh/authorized_keys

ssh <VM_USER>@<VM_IP> "chmod 600 ~/.ssh/authorized_keys"
```

Перевірка безпарольного входу здійснюється командою:

```
ssh -i "$HOME/Cursach/<VM_NAME>_key" <VM_USER>@<VM_IP>
```

2.4.3 Встановлення VirtualBox Guest Additions

Для покращення інтеграції між гостьовою та хост-системою (спільний буфер обміну, синхронізація часу, автопідгонка роздільної здатності) у гостьовій машині встановлюється компонент **VirtualBox Guest Additions**. На системах Ubuntu ISO-файл вже наявний за шляхом /usr/share/virtualbox/VBoxGuestAdditions.iso. Його слід змонтувати у ВМ:

```
sudo mkdir -p /mnt/vbox && sudo mount /dev/sr0 /mnt/vbox

sudo /mnt/vbox/VBoxLinuxAdditions.run

sudo reboot
```

Після перезавантаження сервіс VBoxService має з’явитися серед активних процесів, що свідчить про коректну інтеграцію.

2.4.4 Перевірка з'єднання та доступності ВМ

Фінальним етапом є перевірка можливості з'єднання з гостьовою машиною по IP-адресі, наданій Host-Only Adapter. Також перевіряється робота ssh, наявність Guest Additions (ps aux | grep VBoxService) і функціональність обміну файлами за потреби.

Таким чином, реалізується повноцінне середовище взаємодії між хост-системою та віртуальною машиною, що дозволяє автоматизувати процеси

тестування, розгортання та налагодження без потреби в ручному втручанні через інтерфейс VirtualBox.

2.5. Генерація та виконання скриптів із шаблонів

Для автоматизації кроків взаємодії з віртуальними машинами використано підхід із шаблонами Bash-скриптів, що дозволяє мінімізувати дублювання коду та легко підтримувати послідовність дій.

1. Формат шаблонів

   Шаблони розміщені в каталозі src/main/scripts і мають назви типу stepN_*.sh, де N відповідає порядковому кроку. Кожен файл містить змінні-плейсхолдери в подвійних фігурних дужках:

   • {{VM_NAME}} — ім’я цільової віртуальної машини;
   • {{VM_IP}} — IP-адреса гостьової ОС;
   • {{SNAPSHOT_NAME}} — назва снапшоту для відновлення;
   • {{SRC_DIR}} і {{DEST_DIR}} — локальна та віддалена директорії для копіювання файлів;
   • {{SSH_KEY}} — шлях до приватного SSH-ключа;
   • додаткові плейсхолдери за потреби (наприклад {{TIMEOUT}}, {{RETRY_COUNT}}).

Шаблонний підхід забезпечує чітку відокремленість логіки формування команд від конкретних значень, що підставляються в рантаймі.

2. Підстановка значень і запис файлу

   У класі ScriptGenerator реалізовано метод generateScript(), який приймає параметри контексту (значення для всіх плейсхолдерів) та ім’я кроку. Спочатку виконується зчитування вмісту шаблона як рядка, після чого для кожного ключа здійснюється заміна

String.replace("{{KEY}}", value). Отриманий текст записується у тимчасовий файл у каталозі /tmp із назвою stepN.sh. Після запису файлу застосовуються файлові атрибути: outPath.toFile().setExecutable(true) для надання права виконання.

3. Запуск через ProcessBuilder

   Для виконання згенерованого скрипта використовується стандартний клас ProcessBuilder. У методі runScript() формується новий ProcessBuilder із єдиним аргументом — шляхом до тимчасового скрипта. Виклик inheritIO() забезпечує перенаправлення вихідних потоків дочірнього процесу в потоки JVM, що дозволяє відображати прогрес і помилки без додаткової обробки. Після start() виконується waitFor(), результатом якого є код завершення процесу.

У межах Java-модуля передбачено обробку виняткових ситуацій: якщо процес повертає ненульовий код або кидається IOException/InterruptedException, виконується фіксація стек-трейсу та підняття власного виключення з діагностичною інформацією про невдалий крок, щоб забезпечити зрозуміле повідомлення про помилку та можливість відлагодження.

2.6 Приклад роботи застосунку

У цьому розділі демонструються ключові етапи виконання нашого Java-клієнта та bash-скриптів для керування VirtualBox VM. Кожен крок ілюструється скріншотом із коротким поясненням.

1) Рисунок 1. Меню вибору віртуальних машин

Відображає список доступних VM із їхніми іменами, станом (poweroff), ОС, версіями ядра та снапшотами. Користувач вибирає номер машини для подальшої роботи.

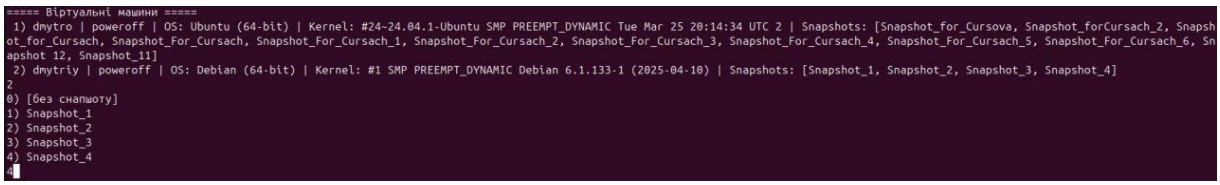

Рисунок 1

2) Рисунок 2. Відновлення і старт VM

Скрипт step1_start_vm.sh перевіряє стан машини — якщо вона вже вимкнена, пропускає зупинку, потім відновлює снапшот Snapshot_4 із прогресом 0…100 % і запускає VM у GUI. Після успішного старту отримує IP через VBoxManage guestproperty.

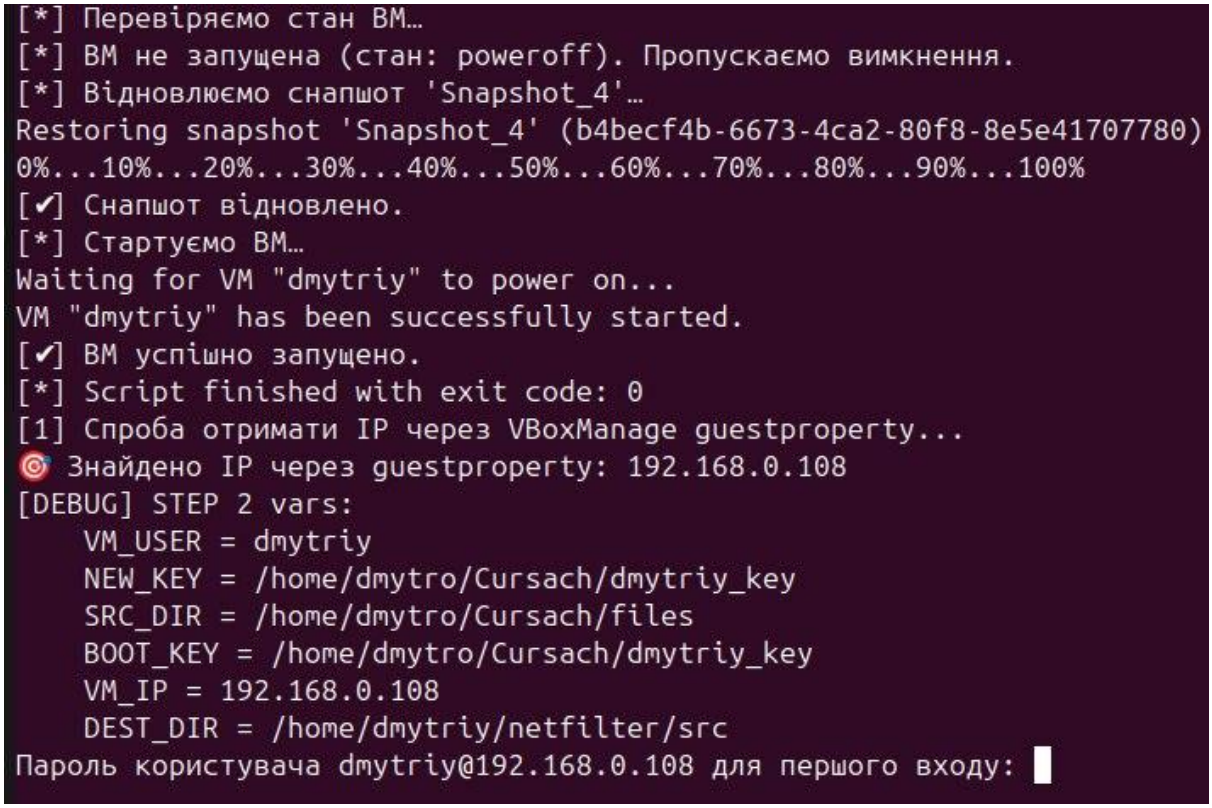

Рисунок 2

3) Рисунок 3.2. Відновлення і старт VM

Скрипт step2_copy_files.sh копіює новий публічний SSH-ключ на VM (через пароль), створює директорію /home/dmytriy/netfilter/src та рекурсивно копіює вміст локальної папки Cursach/files в цю директорію.

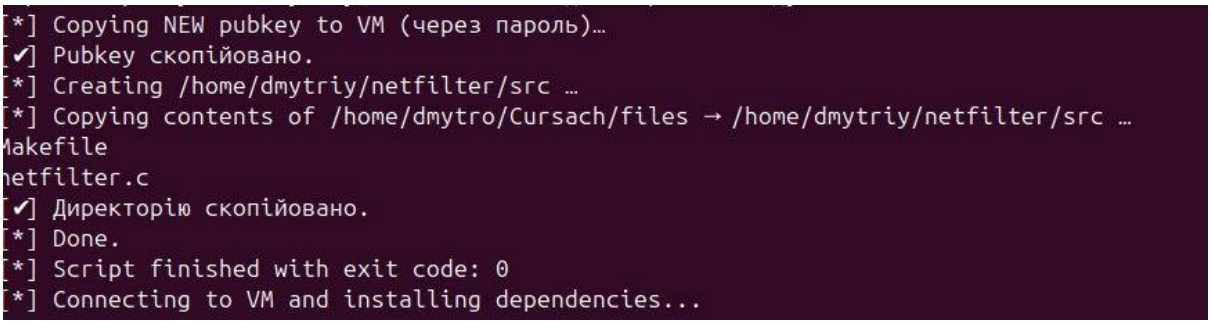

Рисунок 3

4) Рисунок 4. Збірка та завантаження модуля netfilter

Скрипт step3_build_and_load.sh підключається до VM, встановлює build-essential і linux-headers, збирає модуль за допомогою make, а потім завантажує його в ядро (insmod test_ko.ko).

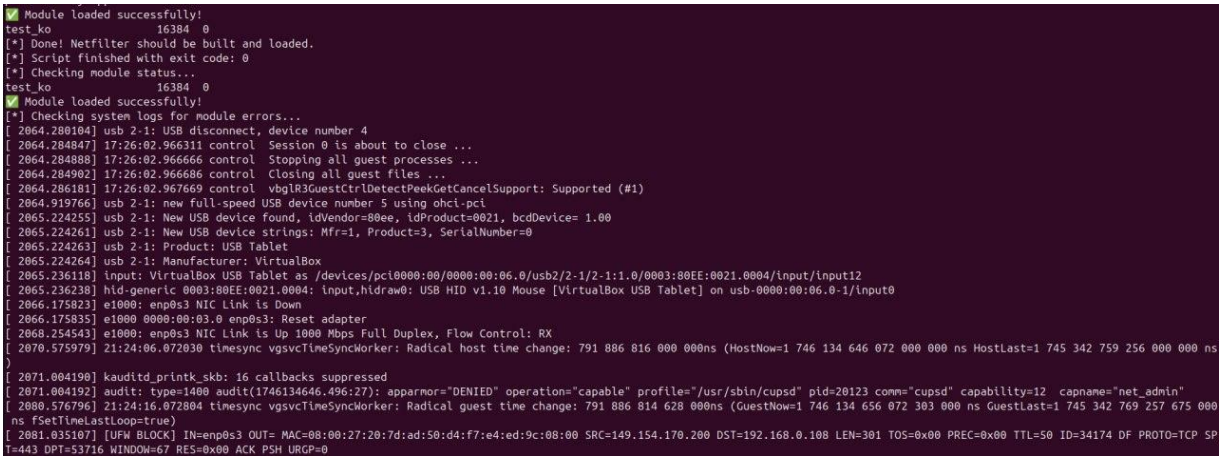

Рисунок 4

5) Рисунок 5. Перевірка статусу модуля та лог-файли

Скрипт step4_check_and_test.sh перевіряє, чи модуль завантажився (lsmod), виводить останні рядки dmesg для діагностики та перевіряє правила UFW. Після завершення відображається повідомлення ✅ Workflow завершено.

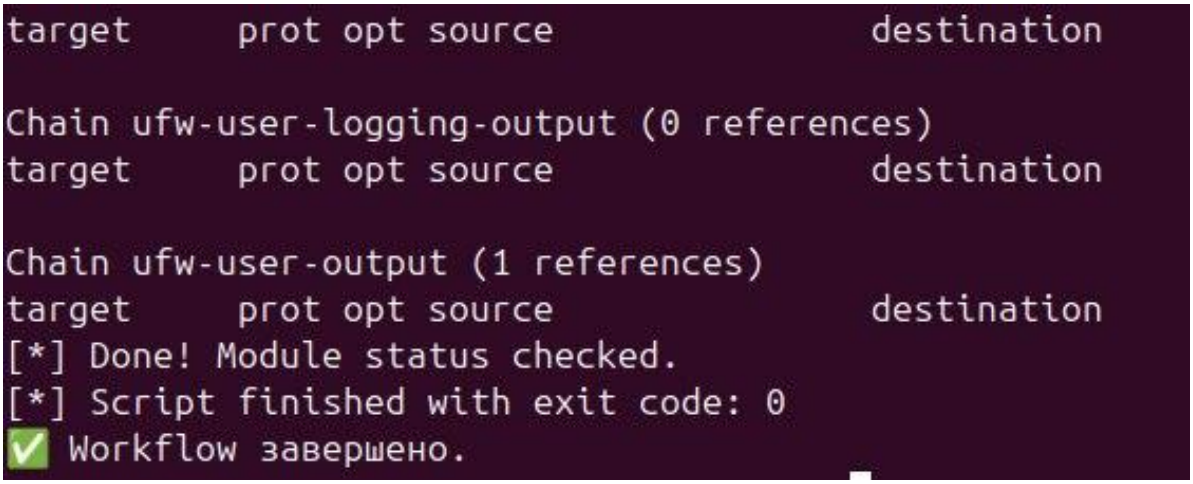

Рисунок 2 1

Висновки

У ході виконання курсової роботи було спроектовано та реалізовано комплексну систему автоматизації розгортання та налагодження середовища VirtualBox із використанням поєднання Java-модулів, Bash-скриптів і C-модуля для Netfilter. Основні результати та досягнення:

1. **Модульна архітектура**
   – Розділення функціональності на чотири Java-модулі (Main, ScriptGenerator, VmIpDetector, SmartVmIpDetector) забезпечило чітку відповідальність кожного компонента, спростило підтримку і потенційне розширення система.
   – Централізований механізм запуску зовнішніх процесів (ProcessBuilder) і шаблонізації скриптів знизив повторюваність коду й мінімізував ризик помилок при формуванні команд.

2. **Перехід від ручного до автоматизованого workflow**
   – Ранні ручні Bash-скрипти (step1_start_vm.sh…step4_check_and_test.sh) успішно зарекомендували себе як прототип, показавши послідовність необхідних дій і контрольні точки.
   – Автоматизація цих кроків через Java-обгортку кардинально скоротила час підготовки віртуального середовища, адже не треба вручну вписувати параметри {{VM_NAME}}, {{VM_IP}}, {{SNAPSHOT_NAME}} , {{SRC_DIR}}, {{DEST_DIR}} в декілька файлів, при цьому зберігши прозорість логів і можливість швидкого відлагодження.

3. **Надійність і діагностика**
   – Використання «strict mode» у Bash-скриптах (set -euo pipefail), детальні лог-повідомлення з префіксами [✓], [✗], [DEBUG] та перевірки вихідних кодів процесів гарантують своєчасне виявлення та локалізацію збоїв.
   – Функціонал перейменування старих ключів SSH у .bak і автоматичне очищення тимчасових файлів запобігають накопиченню застарілих артефактів у робочому каталозі.

4. **Інтеграція з ядром Linux**
   – Розроблений C-модуль Netfilter зі стандартним hook-ом для TCP-трафіку продемонстрував схему розширення ядра Linux для аналізу мережевих пакетів.
   – Makefile із підтримкою Release/Debug-збірок та перевіркою платформи гарантують коректне складання і швидке тестування в різних конфігураціях.

Список використаних джерел

1. @Shrikant Havale, Stack Overflow. VirtualBox Java API - Get IP Address of VM in VirtualBox. *Stack Overflow*. 25.03.2015. URL: https://stackoverflow.com/questions/29259698/virtualbox-java-api-get-ip-address-of-vm-in-virtualbox.

2. @cooow. Install VBOX guest additions on Debian Testing. *Stack Overflow*. 17.11.2014. URL: https://stackoverflow.com/questions/26969183/install-vbox-guest-additions-on-debian-testing.

3. Ask Ubuntu. Kernel driver not installed (rc=-1908) verr_vm_driver_not_installed [duplicate]. *Ask Ubuntu*. 26.11.2017. URL: https://askubuntu.com/questions/980378/kernel-driver-not-installed-rc-1908-verr-vm-driver-not-installed.

4. Ask Ubuntu. How to enable or disable GUI in Ubuntu. *Ask Ubuntu*. 23.05.2020. URL: https://askubuntu.com/questions/1242965/how-to-enable-or-disable-gui-in-ubuntu.

5. Ask Ubuntu. Skipping BTF generation xxx. due to unavailability of vmlinux on Ubuntu 21.04. *Ask Ubuntu*. 25.06.2021. URL: https://askubuntu.com/questions/1348250/skipping-btf-generation-xxx-due-to-unavailability-of-vmlinux-on-ubuntu-21-04.

6. Ask Ubuntu. Can’t install Ubuntu 24.04 on VirtualBox. *Ask Ubuntu*. 01.05.2024. URL: https://askubuntu.com/questions/1512459/cant-install-ubuntu-24-04-on-virtualbox.

7. Ask Ubuntu, user611888. How to Silence the read command. *Ask Ubuntu*. 15.01.2018. URL: https://askubuntu.com/questions/996370/how-to-silence-the-read-command.

8. Awryte Mate, Stack Overflow. ssh-copy-id fails when run from within a remote session. *Stack Overflow*. 27.08.2020. URL: https://stackoverflow.com/questions/63614352/ssh-copy-id-fails-when-run-from-within-a-remote-session.

9. Buzdar K., Greenwebpage Community. How to Install VirtualBox Guest Additions on Debian 12: An Easy Approach. *Greenwebpage Community*. 06.10.2024. URL: https://greenwebpage.com/community/how-to-install-virtualbox-guest-additions-on-debian-12/.

10. CloudSpinx. Install and Configure SSH Server on Debian 12/11/10. *CloudSpinx*. 24.02.2025. URL: https://cloudspinx.com/install-and-configure-ssh-server-on-debian/.

11. Debian Forums User. nftables vs ufw. *Debian User Forums*. URL: https://forums.debian.net/viewtopic.php?t=159355.

12. Debian Project. SSH. *Debian Wiki*. 01.03.2023. URL: https://wiki.debian.org/SSH.

13. Debian Project. Uncomplicated Firewall (ufw). *Debian Wiki*. 15.10.2021. URL: https://wiki.debian.org/Uncomplicated_Firewall_%28ufw%29.

14. Debian Project. SourcesList. *Debian Wiki*. 20.04.2025. URL: https://wiki.debian.org/SourcesList.

15. Debian Project. VirtualBox. *Debian Wiki*. 18.12.2023. URL: https://wiki.debian.org/VirtualBox.

16. Debian project. Debian 12 (Bookworm). *Debian*. 10.06.2023. URL: https://www.debian.org/releases/bookworm/.

17. Debian project. Debian 12 (Bookworm). *Debian*. 10.06.2023. URL: https://www.debian.org/releases/bookworm/.

18. Free Software Foundation. Bash Reference Manual. *GNU*. 19.09.2022. URL: https://www.gnu.org/software/bash/manual/bash.html.

19. GitHub. jsch-agent-proxy. *GitHub*. URL: https://github.com/ymnk/jsch-agent-proxy.

20. LinuxCapable, Joshua James. How to Install UFW on Debian 12, 11 or 10. *LinuxCapable*. 05.08.2024. URL: https://linuxcapable.com/how-to-install-ufw-on-debian-linux/.

21. Linuxiac. How to Install VirtualBox on Debian 12 (Bookworm). *Linuxiac*. 19.07.2023. URL: https://linuxiac.com/how-to-install-virtualbox-on-debian-12-bookworm/.

22. Machtelt G., Garrels BVBA. Bash Guide for Beginners. *The Linux Documentation Project*. 27.12.2008. URL: https://tldp.org/LDP/Bash-Beginners-Guide/html/.

23. Mendel Cooper, The Linux Documentation Project. Advanced Bash-Scripting Guide. *The Linux Documentation Project*. 10.03.2014. URL: https://tldp.org/LDP/abs/html/.

24. Microsoft. Advanced settings configuration in WSL. *Microsoft Learn*. 17.03.2025. URL: https://learn.microsoft.com/en-us/windows/wsl/wsl-config.

25. OpenSSH Project. ssh-keygen(1) Manual Page. *OpenSSH Manual*. URL: https://www.openssh.com/manual.html#ssh-keygen.

26. OpenSSH project. ssh-agent(1) Manual Page. *man7.org*. 10.08.2023. URL: https://www.man7.org/linux/man-pages/man1/ssh-agent.1.html.

27. Oracle. Controlling VirtualBox from the Command Line. *Oracle*. 13.06.2014. URL: https://www.oracle.com/technical-resources/articles/it-infrastructure/admin-manage-vbox-cli.html.

28. Oracle. Oracle VirtualBox: User Guide for Release 7.1. *VirtualBox*. 12.04.2025. URL: https://www.virtualbox.org/manual/.

29. Oracle. ProcessBuilder (Java Platform SE 8). *Oracle Docs*. URL: https://docs.oracle.com/javase/8/docs/api/java/lang/ProcessBuilder.html.

30. Oracle. Packaging Programs in JAR Files. *Oracle Java Tutorials*. URL: https://docs.oracle.com/javase/tutorial/deployment/jar/.

31. Oracle. *Oracle Java Tutorials*. URL: https://docs.oracle.com/javase/tutorial/deployment/jar/basicsindex.html.

32. Reddit, animefan6121. Sometimes get “Insufficient system resources exist to complete the required service” error when loading WSL2 (Ubuntu 20.04.4 LTS). *Reddit*. 17.11.2022. URL: https://www.reddit.com/r/bashonubuntuonwindows/comments/yxbi26/sometimes_get_insufficient_system_resources_exist/.

33. Stack Overflow. VirtualBox: VERR_VM_DRIVER_NOT_INSTALLED. *Stack Overflow*. 24.12.2018. URL: https://stackoverflow.com/questions/53911534/virtualbox-verr-vm-driver-not-installed.

34. Stack Overflow, @NigoroJr. Can’t ssh from external network. *Stack Overflow*. 21.08.2013. URL: https://stackoverflow.com/questions/18365282/cant-ssh-from-external-network.

35. Stack Overflow, @Sae1962. How to get the ipaddress of a virtual box running on local machine. *Stack Overflow*. 22.03.2017. URL:

https://stackoverflow.com/questions/42953785/how-to-get-the-ipaddress-of-a-virtual-box-running-on-local-machine.

36. Stack Overflow, @conectionist. Cannot get VirtualBox machine's IP using the VBox API. *Stack Overflow*. 08.09.2016. URL: https://stackoverflow.com/questions/39392775/cannot-get-virtualbox-machines-ip-using-the-vbox-api.

37. Stack Overflow, @dev59. Why ssh-copy-id copies public key when adding it to authorized_keys seems enough?. *Stack Overflow*. 24.12.2019. URL: https://stackoverflow.com/questions/59470721/why-ssh-copy-id-copies-public-key-when-adding-it-to-authorized-keys-seems-enough.

38. Stack Overflow, Alex Bravo. VBoxManage cant get vm ip address. *Stack Overflow*. 23.06.2020. URL: https://stackoverflow.com/questions/62530575/vboxmanage-cant-get-vm-ip-address.

39. Stack Overflow, DanHeidel. What, exactly, does ssh-copy-id do?. *Stack Overflow*. 27.03.2014. URL: https://stackoverflow.com/questions/22700818/what-exactly-does-ssh-copy-id-do.

40. Stack Overflow, Kedarnath C. Get IP address of android on virtual box. *Stack Overflow*. 21.07.2014. URL: https://stackoverflow.com/questions/24867948/get-ip-address-of-android-on-virtual-box.

41. Stack Overflow, Praful B. How to sshpass ssh-copy-id?. *Stack Overflow*. 09.08.2015. URL: https://stackoverflow.com/questions/31908788/how-to-sshpass-ssh-copy-id.

42. Super Use, John M. Enable Remote Access Linux. *Super User*. 28.03.2013. URL: https://superuser.com/questions/573910/enable-remote-access-linux.

43. Super User. Installing VirtualBox on Debian 12 Bookworm With Errors. *Super User*. 08.03.2024. URL: https://superuser.com/questions/1834320/installing-virtualbox-on-debian-12-bookworm-with-errors.

44. Super User. Installing VirtualBox on Debian 12 Testing. *Super User*. 03.03.2024. URL: https://superuser.com/questions/1833610/installing-virtualbox-on-debian-12-testing.

45. Super User, @TimD1. How do I access Virtualbox internal IP from host machine?. *Super User*. 11.11.2014. URL: https://superuser.com/questions/838411/how-do-i-access-virtualbox-internal-ip-from-host-machine.

46. The Apache Software Foundation. Apache Maven Shade Plugin. *Maven Plugins*. 28.05.2024. URL: https://maven.apache.org/plugins/maven-shade-plugin/.

47. Ubuntu Community. OpenSSH Server. *Ubuntu Community Help Wiki*. 27.02.2015. URL: https://help.ubuntu.com/community/SSH/OpenSSHServer.

48. Ubuntu Community. UFW. *Ubuntu Community Help Wiki*. 27.09.2023. URL: https://help.ubuntu.com/community/UFW.

49. Unix & Linux Stack Exchange. Issues running VirtualBox after installing on Linux debian (bookworm). *Unix & Linux Stack Exchange*. 22.12.2025. URL: https://unix.stackexchange.com/questions/788556/issues-running-virtualbox-after-installing-on-linux-debian-bookworm.

50. Unix & Linux Stack Exchange, @nik. ufw not working on debian. *Unix & Linux Stack Exchange*. 13.02.2020. URL: https://unix.stackexchange.com/questions/567330/ufw-not-working-on-debian.

51. Unix & Linux Stack Exchange, @whairst. Debian 12 won't accept remote ssh connections. *Unix & Linux Stack Exchange*. 06.12.2024. URL: https://unix.stackexchange.com/questions/787718/debian-12-wont-accept-remote-ssh-connections.

52. Virtualbox. *Virtualbox.org*. *Forums*. 04.03.2024. URL: https://forums.virtualbox.org/viewtopic.php?t=111198.

53. docs.kernel.org. Linux Kernel Makefiles. *The Linux Kernel documentation*. URL: https://docs.kernel.org/kbuild/makefiles.html.

54. javadoc.io. JSch Javadoc (com.jcraft:jsch). *javadoc.io*. URL: https://javadoc.io/doc/com.jcraft/jsch/latest/index.html.

55. phoenixNAP, Kaplarevic V. How to Enable SSH on Debian 12. *phoenixNAP Knowledge Base*. 20.03.2025. URL: https://phoenixnap.com/kb/how-to-enable-ssh-on-debian.

56. user3331975, user3331975. ssh-copy-id no identities found error. *Stack Overflow*. 20.03.2014. URL: https://stackoverflow.com/questions/22530886/ssh-copy-id-no-identities-found-error.

Додатки

Додаток А

Характеристика	**VirtualBox**	**VMware Workstation**

Перебудова драйверів DKMS після оновлення ядра хоста	Автоматична (модуль vboxdrv компілюється без втручання)	Потрібні сумісні vmmon і vmnet; при зміні API часто треба патчі
CLI-керування	VBoxManage (повний набір, зокрема guestproperty)	vmrun + vmware-cmd (повний набір, але без guestproperty)
Інтеграційний пакет усередині гостя	Guest Additions: спільний буфер, синхронізація часу, OpenGL, guestproperty get для IP	VMware Tools: спільний буфер, синхронізація часу, ширший 3D-функціонал
3D-прискорення хост-GPU	OpenGL ≤ 4.1 (достатньо для Mesa-тестів)	DirectX 11 і OpenGL 4.3 у Windows-гостях, зручне для графічних бенчмарків
Снапшоти та відкат	Звичайні та ланцюгові snapshot-и; керуються вручну через VBoxManage snapshot …	Snapshot-и + режим autoprotect (фонові інкрементальні копії, миттєвий rollback)
Стійкість після apt upgrade ядра хоста	Висока: DKMS перебудовує модуль,	Середня: поки не випустять патч,

	гіпервізор стартує без додаткових дій	гіпервізор може не завантажитися
Підтримка nested virtualization	Є, але з помітним оверхедом	Є, з дещо кращою продуктивністю

Додаток Б

Головні функціональні блоки з класу Main.java

1. Shell-helper:

```
private static List<String> run(String... cmd) throws Exception {
    Process p = new ProcessBuilder(cmd).redirectErrorStream(true).start();
    try (BufferedReader br = new BufferedReader(
            new InputStreamReader(p.getInputStream()))) {
        List<String> out = br.lines().collect(Collectors.toList());
        p.waitFor();
        return out;
    }
}
```

2. DTO для інформації про VM:

```
/* ——— VBox → DTO ——— */
private record VMInfo(String name,String state,String os,String
kernel,List<String> snaps){
    @Override public String toString(){
        return "%s | %s | OS: %s | Kernel: %s | Snapshots: %s"
                .formatted(name,state,os,kernel,snaps.isEmpty()?"-":snaps);
    }
}
```

3. Зчитування властивостей VM

```
private static Map<String,String> vmInfoMap(String vm) throws Exception {
    Pattern kv = Pattern.compile("^([^=]+)=\"?(.*?)\"?$");
    Map<String,String> m = new HashMap<>();
```

```
    for (String l : run("VBoxManage","showvminfo",vm,"--machinereadable"))
{
        Matcher mm = kv.matcher(l);
        if (mm.find()) m.put(mm.group(1), mm.group(2));
    }
    return m;
}
```

4. Отримання повної інформації про VM:

```
private static VMInfo getVmInfo(String vm) throws Exception {
    Map<String,String> kv = vmInfoMap(vm);
    String st = kv.getOrDefault("VMState","-");
    String os = kv.getOrDefault("ostype","-");
    List<String> snaps = kv.entrySet().stream()
            .filter(e->e.getKey().startsWith("SnapshotName"))
            .sorted(Map.Entry.comparingByKey())
            .map(Map.Entry::getValue).toList();
    String k = guestProp(vm,"/VirtualBox/GuestInfo/OS/Version");
    boolean headless = false;
    if(k==null){
        if(!"running".equals(st)){ run("VBoxManage","startvm",vm,"--
type","headless"); headless=true; }
        k = waitKernel(vm);
        if(headless) run("VBoxManage","controlvm",vm,"poweroff");
    }
    return new VMInfo(vm,st,os,k,snaps);
}
```

Додаток В

Ключові фрагменти з ScriptGenerator.java – класу, який перетворює шаблони Bash скриптів в Bash скрипті, які виконуються програмою

1. Генерація скрипту з шаблону

```
public static Path generateScript(String templateName, Map<String, String>
vars) {
    // Тепер без папки "scripts/"
    String resourcePath = templateName;
    String content;
    try (InputStream in = ScriptGenerator.class
            .getClassLoader()
            .getResourceAsStream(resourcePath)) {
        if (in == null) {
            throw new IllegalArgumentException("Resource not found: " +
resourcePath);
        }
        content = new String(in.readAllBytes(), StandardCharsets.UTF_8);
    } catch (IOException e) {
        throw new UncheckedIOException(e);
    }

    // Підстановка {{KEY}} → VALUE
    for (var entry : vars.entrySet()) {
        content = content.replace("{{" + entry.getKey() + "}}",
entry.getValue());
    }

    try {
        // Запис у тимчасовий файл
        Path tmp = Files.createTempFile("cursach_", "_" + templateName);
        Files.writeString(tmp, content, StandardCharsets.UTF_8);
        tmp.toFile().setExecutable(true);
        return tmp;
    } catch (IOException e) {
        throw new UncheckedIOException(e);
    }
```

2. Запуск уже згенерованого скрипту

```
public static void runScript(String scriptPath) throws IOException,
InterruptedException {
    ProcessBuilder pb = new ProcessBuilder(scriptPath);
    pb.inheritIO(); // Передає stdout/stderr у консоль
    Process process = pb.start();
    int exitCode = process.waitFor();
    System.out.println("[*] Script finished with exit code: " + exitCode);
}
```

Додаток Г

Фрагменти класів для визначення IP-адреси VM

1. SmartVmIpDetector.detectVmIp

```
public static String detectVmIp(String vmName) throws Exception {
    // 1. GuestProperty (через VBoxManage)
    try {
        System.out.println("[1] Спроба отримати IP через VBoxManage
guestproperty...");
        String ip = VmIpDetector.getVmIp(vmName);
        System.out.println("🎯 Знайдено IP через guestproperty: " + ip);
        return ip;
    } catch (Exception e) {
        System.out.println("➡️ Guestproperty не спрацювало: " +
e.getMessage());
    }
    throw new RuntimeException("❌ Не вдалося визначити IP-адресу для VM:
" + vmName);
}
```

2. VmIpDetector.getVmIp

```
public static String getVmIp(String vmName) throws IOException,
InterruptedException {
    // Формуємо команду для отримання властивості гостя з IP
    ProcessBuilder pb = new ProcessBuilder(VBOXMANAGE, "guestproperty",
"get", vmName, "/VirtualBox/GuestInfo/Net/0/V4/IP");
    pb.redirectErrorStream(true);
    Process process = pb.start();

    BufferedReader reader = new BufferedReader(new
InputStreamReader(process.getInputStream()));
    String line;
    String ip = null;
    while ((line = reader.readLine()) != null) {
        if (line.startsWith("Value: ")) {
            ip = line.substring("Value: ".length()).trim();
            break;
        }
    }
    int exitCode = process.waitFor();
    if (exitCode != 0 || ip == null || ip.isEmpty() || ip.equals("No value
set!")) {
        throw new RuntimeException("Не вдалося визначити IP для віртуальної
машини '" + vmName + "'. Отримано: " + ip);
    }
    return ip;
}
```

Додаток Ґ

Набір шалонів до Bash скриптів для розгортання віртуальної машини, передачі за збірки файлів netfilter і запуску тесту.

1. step1_start_vm.sh — перевірка стану, відновлення снапшоту та старт VM

```
#!/usr/bin/env bash
# Увімкнути строгий режим Bash: вихід при будь-якій помилці, незаданих
змінних або помилці конвеєру
set -euo pipefail

# Знаходимо повний шлях до VBoxManage у $PATH
VBM="$(command -v VBoxManage)"
# Ім'я віртуальної машини та снапшоту задається через змінні оточення або
підстановку шаблону
VM_NAME="${VM_NAME}"
SNAPSHOT="${SNAPSHOT_NAME}"

# ── 1. Перевірка стану ВМ та коректне вимкнення
──────────────────────────────
echo "[*] Перевіряємо стан ВМ…"
# Отримуємо поточний стан віртуальної машини з властивості VMState
VM_STATE="$("$VBM" showvminfo "$VM_NAME" --machinereadable | awk -F'\"'
'/^VMState=/{print $2}')"

# Якщо ВМ запущена, призупинена або зависла — виконуємо коректне вимикання
if [[ "$VM_STATE" =~ ^(running|paused|stuck)$ ]]; then
    echo "[*] ВМ у стані '$VM_STATE'. Вимикаємо…"
    # Виконуємо команду poweroff, щоб завершити роботу ОС у ВМ
    "$VBM" controlvm "$VM_NAME" poweroff

    # Цикл очікування переходу в стан 'poweroff'
    while true; do
        sleep 2
        VM_STATE="$("$VBM" showvminfo "$VM_NAME" --machinereadable | awk -
F'\"' '/^VMState=/{print $2}')"
        # Виходимо з циклу, коли ВМ повністю вимкнена
        [[ "$VM_STATE" == "poweroff" ]] && break
        echo "    …очікуємо (поточний стан: $VM_STATE)"
    done
    echo "[✓] ВМ вимкнено."
else
    # Якщо ВМ вже вимкнена або не існує — пропускаємо крок вимкнення
    echo "[*] ВМ не запущена (стан: $VM_STATE). Пропускаємо вимкнення."
fi

# ── 2. Відновлення зі снапшоту
──────────────────────────────────────
echo "[*] Відновлюємо снапшот '$SNAPSHOT'…"
# Команда restore повертає ВМ до стану, зафіксованого в указаному снапшоті
"$VBM" snapshot "$VM_NAME" restore "$SNAPSHOT"
echo "[✓] Снапшот відновлено."
```

```
# ─── 3. Запуск ВМ
──────────────────────────────────────────────────────────────

echo "[*] Стартуємо ВМ…"
# Запускаємо ВМ у графічному режимі (GUI)
if "$VBM" startvm "$VM_NAME" --type gui; then
    echo "[✓] ВМ успішно запущено."
else
    # Якщо команда повернула помилку — виводимо повідомлення та виходимо з
кодом 1
    echo "❌ Не вдалося запустити ВМ!"
    exit 1
fi
```

2. step2_copy_files.sh — копіювання SSH-ключів та файлів

```
#!/bin/bash
# Вихід при будь-якій помилці
set -e

# Користувач та IP-адреса віддаленої ВМ
VM_USER="${VM_USER}"
VM_IP="${VM_IP}"
# Шляхи до локальної директорії з файлами та цільової директорії на ВМ
SRC_DIR="${SRC_DIR}"      # наприклад: /home/username/netfilter/src/files
DEST_DIR="${DEST_DIR}"    # наприклад: /home/username/netfilter/src
# Файли SSH-ключів: ключ для початкового доступу та новий ключ для
подальших підключень
BOOT_KEY="${BOOT_KEY}"
NEW_KEY="${NEW_KEY}"

# Якщо початковий ключ і новий збігаються, виконуємо копіювання
використовуючи пароль
if [[ "$BOOT_KEY" == "$NEW_KEY" ]]; then
  # Запит паролю користувача для першого підключення
  echo -n "Пароль користувача $VM_USER@$VM_IP для першого входу: "
  read -rs VM_PASS
  echo

  # Створюємо папку .ssh та встановлюємо права доступу
  echo "[*] Copying NEW pubkey to VM (через пароль)…"
  sshpass -p "$VM_PASS" ssh -o StrictHostKeyChecking=no \
      "$VM_USER@$VM_IP" \
      'mkdir -p ~/.ssh && chmod 700 ~/.ssh'

  # Копіюємо новий публічний ключ у authorized_keys на ВМ
  sshpass -p "$VM_PASS" scp -o StrictHostKeyChecking=no \
      "$NEW_KEY.pub" \
      "$VM_USER@$VM_IP:~/.ssh/authorized_keys"
else
  # Інакше копіюємо новий ключ за допомогою старого приватного ключа
  echo "[*] Copying NEW pubkey to VM (старим ключем)…"
  scp -i "$BOOT_KEY" -o StrictHostKeyChecking=no \
      "$NEW_KEY.pub" \
      "$VM_USER@$VM_IP:~/.ssh/authorized_keys"
fi
# Підтвердження успішності операції
```

```
echo "[✓] Pubkey скопійовано."

# Створюємо цільову директорію на ВМ і встановлюємо права 700
echo "[*] Creating $DEST_DIR …"
ssh -i "$NEW_KEY" -o StrictHostKeyChecking=no \
    "$VM_USER@$VM_IP" "mkdir -p '$DEST_DIR' && chmod 700 '$DEST_DIR'"

# Рекурсивно копіюємо вміст SRC_DIR до DEST_DIR з використанням нового
ключа
echo "[*] Copying contents of $SRC_DIR → $DEST_DIR …"
scp -r -i "$NEW_KEY" -o StrictHostKeyChecking=no \
    "${SRC_DIR}/." "$VM_USER@$VM_IP:$DEST_DIR/"
# Повідомлення про завершення копіювання
echo "[✓] Директорію скопійовано."

# Завершальне повідомлення
echo "[*] Done."
```

3. step3_build_and_load.sh — збірка та завантаження модуля netfilter

```
#!/bin/bash
# Вихід при будь-якій помилці
set -e

# Змінні для підключення до ВМ та параметрів збірки
VM_USER="${VM_USER}"         # Ім'я користувача на віддаленій ВМ
VM_IP="${VM_IP}"             # IP-адреса віддаленої ВМ
NETFILTER_DIR="${NETFILTER_DIR}"  # Директорія з вихідним кодом netfilter
(на ВМ)
BUILD_DIR="${BUILD_DIR}"     # Директорія для збереження скомпільованого
модуля (на ВМ)
MODULE_NAME="${MODULE_NAME}" # Ім'я модуля без розширення (.ko)
SSH_KEY="${SSH_KEY}"         # Шлях до приватного SSH-ключа для підключення

# ─── 1. Встановлення залежностей на ВМ
──────────────────────────────────
echo "[*] Connecting to VM and installing dependencies..."
ssh -i "$SSH_KEY" "$VM_USER@$VM_IP" << 'EOF'
    # Оновлюємо список пакетів та встановлюємо інструменти для компіляції
    sudo apt update -y
    sudo apt install -y build-essential linux-headers-$(uname -r)
EOF

# ─── 2. Збірка netfilter на ВМ
──────────────────────────────────────────
echo "[*] Connecting to VM and building Netfilter..."
ssh -i "$SSH_KEY" "$VM_USER@$VM_IP" << 'EOF'
    # Перевіряємо наявність необхідних файлів в каталозі з вихідниками
    if [ ! -f "$NETFILTER_DIR/netfilter.c" ] || [ ! -f
"$NETFILTER_DIR/Makefile" ]; then
        echo "❌ Error: Missing source files!"
        exit 1
    fi
```

```
    # Очищаємо попередню збірку та компілюємо модуль у каталозі ядра
    make -C /lib/modules/$(uname -r)/build M=$NETFILTER_DIR clean
    make -C /lib/modules/$(uname -r)/build M=$NETFILTER_DIR modules

    # Переміщуємо зібраний .ko файл до директорії BUILD_DIR
    mkdir -p $BUILD_DIR
    mv $NETFILTER_DIR/$MODULE_NAME.ko $BUILD_DIR
EOF

# ─── 3. Завантаження/перезавантаження модуля ──────────────────
echo "[*] Connecting to VM and loading module..."
ssh -i "$SSH_KEY" "$VM_USER@$VM_IP" << 'EOF'
    # Якщо модуль вже завантажений, видаляємо його для свіжого завантаження
    if lsmod | grep -q "$MODULE_NAME"; then
        sudo rmmod $MODULE_NAME
    fi

    # Спроба завантажити зібраний модуль та вивід результату
    sudo insmod $BUILD DIR/$MODULE NAME.ko && \
        echo "✅ Module loaded successfully!" \
        || echo "❌ Error: Failed to load module!"

    # Перевіряємо, чи модуль справді завантажився
    lsmod | grep "$MODULE_NAME"
EOF

# ─── Завершення ──────────────────────────────────────────────────
echo "[*] Done! Netfilter should be built and loaded."
```

4. step4_check_and_test.sh — перевірка статусу модуля та логів

```
#!/bin/bash
# Вихід при будь-якій помилці (опційно можна додати 'set -e' для жорсткіших перевірок)

# ─── Налаштування параметрів підключення ──────────────────────
VM_USER="${VM_USER}"         # Ім'я користувача на віддаленій ВМ
VM_IP="${VM_IP}"             # IP-адреса віддаленої ВМ
MODULE_NAME="${MODULE_NAME}" # Ім'я перевірюваного модуля без розширення .ko
SSH_KEY="${SSH_KEY}"         # Шлях до приватного SSH-ключа для підключення

# ─── 1. Перевірка статусу модуля ──────────────────────────────
echo "[*] Checking module status..."
ssh -i "$SSH_KEY" "$VM_USER@$VM_IP" << 'EOF'
    # Виконуємо lsmod і шукаємо рядок з назвою модуля
    if lsmod | grep -q "$MODULE_NAME"; then
        echo "✅ Module loaded successfully!"
    else
        echo "❌ Module not found!"
    fi
EOF
```

```
# ——— 2. Перевірка системних логів на помилки модуля
# ————————————————————
echo "[*] Checking system logs for module errors..."
ssh -i "$SSH_KEY" "$VM_USER@$VM_IP" << 'EOF'
    # Показуємо останні 20 рядків з dmesg для пошуку повідомлень
пов'язаних з модулем
    sudo dmesg | tail -n 20
EOF

# ——— 3. Виконання базових тестів (iptables)
# ——————————————————————————
echo "[*] Running tests..."
ssh -i "$SSH_KEY" "$VM_USER@$VM_IP" << 'EOF'
    # Перевіряємо поточний список правил netfilter через iptables
    sudo iptables -L
EOF

# ——— 4. Завершальне повідомлення
# ——————————————————————————————————
echo "[*] Done! Module status checked."
```

Додаток Д

Makefile та фрагмент коду модуля netfilter

1. Makefile

```
# Linux only!
UNAME := $(shell uname)
ifneq ($(UNAME), Linux)
$(error [!] Please, build and test the kernel module on Linux! Make sure
the Linux version, kernel, and the architecture match your target)
endif

# Set module name (may/should be redefined!)
MODULE_NAME ?= test_ko

# List objects to build into the module
$(MODULE_NAME)-objs := netfilter.o

# Set the kernel module name
obj-m := $(MODULE_NAME).o

# Build customization
CUSTOM=
ifdef DEBUG
    CUSTOM += -DDEBUG=1
    BUILD_DIR=debug
else
    BUILD_DIR=production
endif
EXTRA_CFLAGS += $(CUSTOM)

# Get the current dir
PWD := $(shell pwd)

# Current kernel headers are here:
KDIR := /lib/modules/$(shell uname -r)/build

# Compile the module
all:
    @echo
    @echo '[*] Building the $(BUILD_DIR) module, make arguments:
MODULE_NAME=$(MODULE_NAME) $(CUSTOM) ...'
    $(MAKE) -C $(KDIR) M=$(PWD) modules
    @echo
    @echo '[*] Moving the module to ../build-$(BUILD_DIR)...'
    mkdir -p ../build-$(BUILD_DIR)
    mv ./$(MODULE_NAME).ko ../build-$(BUILD_DIR)

# Clean it up
clean:
    @echo
    @echo '[*] Cleaning up previous $(BUILD_DIR) build...'
    $(MAKE) -C $(KDIR) M=$(PWD) clean
    rm -f ../build-$(BUILD_DIR)/$(MODULE_NAME).ko

# Load the module
load:
```

```
    @echo
    @echo '[*] Loading the $(BUILD_DIR) module (root required)...'
    sudo insmod ../build-$(BUILD_DIR)/$(MODULE_NAME).ko

# Unload the module
unload:
    @echo
    @echo '[*] Unloading the module (root required)...'
    sudo rmmod $(MODULE_NAME)
```

2. Фрагмент модуля netfilter.c

```
/* Includes */
#include <linux/kernel.h>
#include <linux/module.h>
#include <linux/netfilter.h>
#include <linux/netfilter_ipv4.h>
#include <linux/ip.h>
#include <linux/tcp.h>
#include <linux/skbuff.h>
#include <linux/in.h>
#include <net/ip.h>
#include <net/tcp.h>

/* The netfilter struct (we need it to register the hook) */
static struct nf_hook_ops netfilter_hook;

/* Netfilter callback */
static unsigned int hook_func(void *priv, struct sk_buff *skb, const struct
nf_hook_state *state) {
    #ifdef DEBUG
    printk(KERN_INFO "[Netfilter] Entering the filter hook...\n");
    #endif

    /* Nothing to check? Accept, let it go */
    if (!skb) {
        #ifdef DEBUG
        printk(KERN_INFO "[Netfilter] The socket is null, let it go\n");
        #endif
        return NF_ACCEPT;
    }

    /* No IP header or not IP? Accept, let it go */
    struct iphdr *ip_header = ip_hdr(skb);
    if (!ip_header || ip_header->protocol != IPPROTO_TCP) {
        #ifdef DEBUG
        printk(KERN_INFO "[Netfilter] Not an IP header or not TCP, let it
go\n");
        #endif
        return NF_ACCEPT;
    }

    /* Report success */
    struct tcphdr *tcp_header = tcp_hdr(skb);
    #ifdef DEBUG
    printk(KERN_INFO "[Netfilter] Got the packet from [%pI4:%d]!\n",
           &ip_header->saddr, ntohs(tcp_header->source));
```

```
    #endif
    return NF_ACCEPT;
}

/* Load the module */
static int __init nf_init(void) {
    /* Hook information: pre-routing, TCP, high priority hook */
    netfilter_hook.hook = hook_func;
    netfilter_hook.hooknum = NF_INET_PRE_ROUTING;
    netfilter_hook.pf = PF_INET;
    netfilter_hook.priority = NF_IP_PRI_FIRST;

    #ifdef DEBUG
    printk(KERN_INFO "[Netfilter] Loading module...\n");
    #endif

    /* Register the hook */
    nf_register_net_hook(&init_net, &netfilter_hook);

    #ifdef DEBUG
    printk(KERN_INFO "[Netfilter] Loaded. Hook registered.\n");
    #endif

    return 0;
}

/*  Unload the module */
static void __exit nf_exit(void) {
    #ifdef DEBUG
    printk(KERN_INFO "[Netfilter] Unloading module...\n");
    #endif

    nf_unregister_net_hook(&init_net, &netfilter_hook);

    #ifdef DEBUG
    printk(KERN_INFO "[Netfilter] Unloaded. Hook unregistered.\n");
    #endif
}

/* Module information */
MODULE_LICENSE("GPL");
MODULE_AUTHOR("DK");
MODULE_DESCRIPTION("A test Kernel Object");

/* Load/unload module hooks */
module_init(nf_init);
module_exit(nf_exit);
```

Додаток Е

Середовище розгортання й залежності

Операційна система та ядро

- Ubuntu 24.04 LTS
- Linux kernel 6.6.x +
- Необхідні пакети

VBoxManage 6.1.38
sshpass 1.06
gcc 12.2.0
make 4.3
Java (OpenJDK) 17.0.8
Maven 3.8.8

Приклад встановення:

```
dmytro@dmytro-Vostro-15-3510:~$ sudo apt update
sudo apt install -y \
  virtualbox \
  sshpass \
  build-essential \
  linux-headers-$(uname -r) \
  openjdk-17-jdk \
  maven
```

Приклад перевірки переліку встановлених пакетів

```
dmytro@dmytro-Vostro-15-3510:~$ dpkg -l | grep -E 'virtualbox|sshpass|openjdk|maven'
ii  libmaven-parent-java                      35-1                    all          Maven metadata for Apache Maven itself
ii  libmaven-resolver-java                    1.6.3-1                 all          Library to handle Java artifact repositories
ii  libmaven-shared-utils-java                3.3.4-1                 all          Replacement for plexus-utils in Maven
ii  libmaven3-core-java                       3.8.7-2                 all          Core libraries for Maven 3
ii  maven                                     3.8.7-2                 all          Java software project management and comprehension tool
ii  openjdk-21-jdk:amd64                      21.0.6+7-1~24.04.1      amd64        OpenJDK Development Kit (JDK)
ii  openjdk-21-jdk-headless:amd64             21.0.6+7-1~24.04.1      amd64        OpenJDK Development Kit (JDK) (headless)
ii  openjdk-21-jre:amd64                      21.0.6+7-1~24.04.1      amd64        OpenJDK Java runtime, using Hotspot JIT
ii  openjdk-21-jre-headless:amd64             21.0.6+7-1~24.04.1      amd64        OpenJDK Java runtime, using Hotspot JIT (headless)
ii  sshpass                                   1.09-1                  amd64        Non-interactive ssh password authentication
ii  virtualbox                                7.0.16-dfsg-2ubuntu1.1  amd64        x86 virtualization solution - base binaries
ii  virtualbox-dkms                           7.0.16-dfsg-2ubuntu1.1  amd64        x86 virtualization solution - kernel module sources for dkms
ii  virtualbox-ext-pack                       7.0.16-1                all          extra capabilities for VirtualBox, downloader.
ii  virtualbox-qt                             7.0.16-dfsg-2ubuntu1.1  amd64        x86 virtualization solution - Qt based user interface
```

Додаток Є

Приклад повної роботи програми на прикладі віртуальної машини №2(Debian)

```
 ===== Віртуальні машини =====
 1) dmytro | poweroff | OS: Ubuntu (64-bit) | Kernel: #24~24.04.1-Ubuntu
SMP PREEMPT_DYNAMIC Tue Mar 25 20:14:34 UTC 2 | Snapshots:
[Snapshot_for_Cursova, Snapshot_forCursach_2, Snapshot_for_Cursach,
Snapshot_For_Cursach, Snapshot_For_Cursach_1, Snapshot_For_Cursach_2,
Snapshot_For_Cursach_3, Snapshot_For_Cursach_4, Snapshot_For_Cursach_5,
Snapshot_For_Cursach_6, Snapshot 12, Snapshot_11]
 2) dmytriy | poweroff | OS: Debian (64-bit) | Kernel: #1 SMP
PREEMPT_DYNAMIC Debian 6.1.133-1 (2025-04-10) | Snapshots: [Snapshot_1,
Snapshot_2, Snapshot_3, Snapshot_4]
2
0) [без снапшоту]
1) Snapshot_1
2) Snapshot_2
3) Snapshot_3
4) Snapshot_4
4
[*] Перевіряємо стан ВМ…
[*] ВМ не запущена (стан: poweroff). Пропускаємо вимкнення.
[*] Відновлюємо снапшот 'Snapshot_4'…
Restoring snapshot 'Snapshot_4' (b4becf4b-6673-4ca2-80f8-8e5e41707780)
0%...10%...20%...30%...40%...50%...60%...70%...80%...90%...100%
[✓] Снапшот відновлено.
[*] Стартуємо ВМ…
Waiting for VM "dmytriy" to power on...
VM "dmytriy" has been successfully started.
[✓] ВМ успішно запущено.
[*] Script finished with exit code: 0
[1] Спроба отримати IP через VBoxManage guestproperty...
 Знайдено IP через guestproperty: 192.168.0.108
[DEBUG] STEP 2 vars:
    VM_USER = dmytriy
    NEW_KEY = /home/dmytro/Cursach/dmytriy_key
    SRC_DIR = /home/dmytro/Cursach/files
    BOOT_KEY = /home/dmytro/Cursach/dmytriy_key
    VM_IP = 192.168.0.108
    DEST_DIR = /home/dmytriy/netfilter/src
Пароль користувача dmytriy@192.168.0.108 для першого входу:
[*] Copying NEW pubkey to VM (через пароль)…
[✓] Pubkey скопійовано.
[*] Creating /home/dmytriy/netfilter/src …
[*] Copying contents of /home/dmytro/Cursach/files →
/home/dmytriy/netfilter/src …
Makefile
100% 1433     1.2MB/s   00:00
netfilter.c
100% 2426     4.7MB/s   00:00
[✓] Директорію скопійовано.
[*] Done.
[*] Script finished with exit code: 0
[*] Connecting to VM and installing dependencies...
Pseudo-terminal will not be allocated because stdin is not a terminal.
```

```
Linux dmytriy 6.1.0-33-amd64 #1 SMP PREEMPT_DYNAMIC Debian 6.1.133-1 (2025-
04-10) x86_64

The programs included with the Debian GNU/Linux system are free software;
the exact distribution terms for each program are described in the
individual files in /usr/share/doc/*/copyright.

Debian GNU/Linux comes with ABSOLUTELY NO WARRANTY, to the extent
permitted by applicable law.

WARNING: apt does not have a stable CLI interface. Use with caution in
scripts.

Hit:1 http://deb.debian.org/debian bookworm InRelease
Get:2 http://deb.debian.org/debian bookworm-updates InRelease [55.4 kB]
Get:3 http://security.debian.org/debian-security bookworm-security
InRelease [48.0 kB]
Get:4 http://security.debian.org/debian-security bookworm-security/main
Sources [128 kB]
Get:5 http://security.debian.org/debian-security bookworm-security/main
amd64 Packages [257 kB]
Get:6 http://security.debian.org/debian-security bookworm-security/main
Translation-en [154 kB]
Fetched 643 kB in 1s (647 kB/s)
Reading package lists...
Building dependency tree...
Reading state information...
43 packages can be upgraded. Run 'apt list --upgradable' to see them.

WARNING: apt does not have a stable CLI interface. Use with caution in
scripts.

Reading package lists...
Building dependency tree...
Reading state information...
build-essential is already the newest version (12.9).
linux-headers-6.1.0-33-amd64 is already the newest version (6.1.133-1).
linux-headers-6.1.0-33-amd64 set to manually installed.
0 upgraded, 0 newly installed, 0 to remove and 43 not upgraded.
[*] Connecting to VM and building Netfilter...
Pseudo-terminal will not be allocated because stdin is not a terminal.
Linux dmytriy 6.1.0-33-amd64 #1 SMP PREEMPT_DYNAMIC Debian 6.1.133-1 (2025-
04-10) x86_64

The programs included with the Debian GNU/Linux system are free software;
the exact distribution terms for each program are described in the
individual files in /usr/share/doc/*/copyright.

Debian GNU/Linux comes with ABSOLUTELY NO WARRANTY, to the extent
permitted by applicable law.
make: Entering directory '/usr/src/linux-headers-6.1.0-33-amd64'
make: Leaving directory '/usr/src/linux-headers-6.1.0-33-amd64'
make: Entering directory '/usr/src/linux-headers-6.1.0-33-amd64'
  CC [M]  /home/dmytriy/netfilter/src/netfilter.o
/home/dmytriy/netfilter/src/netfilter.c: In function ‘hook_func’:
/home/dmytriy/netfilter/src/netfilter.c:40:20: warning: unused variable
‘tcp_header’ [-Wunused-variable]
   40 |     struct tcphdr *tcp_header = tcp_hdr(skb);
      |                    ^~~~~~~~~~
  LD [M]  /home/dmytriy/netfilter/src/test_ko.o
```

```
  MODPOST /home/dmytriy/netfilter/src/Module.symvers
  CC [M]  /home/dmytriy/netfilter/src/test_ko.mod.o
  LD [M]  /home/dmytriy/netfilter/src/test_ko.ko
  BTF [M] /home/dmytriy/netfilter/src/test_ko.ko
Skipping BTF generation for /home/dmytriy/netfilter/src/test_ko.ko due to
unavailability of vmlinux
make: Leaving directory '/usr/src/linux-headers-6.1.0-33-amd64'
[*] Connecting to VM and loading module...
Pseudo-terminal will not be allocated because stdin is not a terminal.
Linux dmytriy 6.1.0-33-amd64 #1 SMP PREEMPT_DYNAMIC Debian 6.1.133-1 (2025-
04-10) x86_64

The programs included with the Debian GNU/Linux system are free software;
the exact distribution terms for each program are described in the
individual files in /usr/share/doc/*/copyright.

Debian GNU/Linux comes with ABSOLUTELY NO WARRANTY, to the extent
permitted by applicable law.
✅ Module loaded successfully!
test_ko                16384  0
[*] Done! Netfilter should be built and loaded.
[*] Script finished with exit code: 0
[*] Checking module status...
test_ko                16384  0
✅ Module loaded successfully!
[*] Checking system logs for module errors...
[ 2064.280104] usb 2-1: USB disconnect, device number 4
[ 2064.284847] 17:26:02.966311 control  Session 0 is about to close ...
[ 2064.284888] 17:26:02.966666 control  Stopping all guest processes ...
[ 2064.284902] 17:26:02.966686 control  Closing all guest files ...
[ 2064.286181] 17:26:02.967669 control
vbglR3GuestCtrlDetectPeekGetCancelSupport: Supported (#1)
[ 2064.919766] usb 2-1: new full-speed USB device number 5 using ohci-pci
[ 2065.224255] usb 2-1: New USB device found, idVendor=80ee,
idProduct=0021, bcdDevice= 1.00
[ 2065.224261] usb 2-1: New USB device strings: Mfr=1, Product=3,
SerialNumber=0
[ 2065.224263] usb 2-1: Product: USB Tablet
[ 2065.224264] usb 2-1: Manufacturer: VirtualBox
[ 2065.236118] input: VirtualBox USB Tablet as
/devices/pci0000:00/0000:00:06.0/usb2/2-1/2-
1:1.0/0003:80EE:0021.0004/input/input12
[ 2065.236238] hid-generic 0003:80EE:0021.0004: input,hidraw0: USB HID
v1.10 Mouse [VirtualBox USB Tablet] on usb-0000:00:06.0-1/input0
[ 2066.175823] e1000: enp0s3 NIC Link is Down
[ 2066.175835] e1000 0000:00:03.0 enp0s3: Reset adapter
[ 2068.254543] e1000: enp0s3 NIC Link is Up 1000 Mbps Full Duplex, Flow
Control: RX
[ 2070.575979] 21:24:06.072030 timesync vgsvcTimeSyncWorker: Radical host
time change: 791 886 816 000 000ns (HostNow=1 746 134 646 072 000 000 ns
HostLast=1 745 342 759 256 000 000 ns)
[ 2071.004190] kauditd_printk_skb: 16 callbacks suppressed
[ 2071.004192] audit: type=1400 audit(1746134646.496:27): apparmor="DENIED"
operation="capable" profile="/usr/sbin/cupsd" pid=20123 comm="cupsd"
capability=12  capname="net_admin"
[ 2080.576796] 21:24:16.072804 timesync vgsvcTimeSyncWorker: Radical guest
time change: 791 886 814 628 000ns (GuestNow=1 746 134 656 072 303 000 ns
GuestLast=1 745 342 769 257 675 000 ns fSetTimeLastLoop=true)
```

```
[ 2081.035107] [UFW BLOCK] IN=enp0s3 OUT=
MAC=08:00:27:20:7d:ad:50:d4:f7:e4:ed:9c:08:00 SRC=149.154.170.200
DST=192.168.0.108 LEN=301 TOS=0x00 PREC=0x00 TTL=50 ID=34174 DF PROTO=TCP
SPT=443 DPT=53716 WINDOW=67 RES=0x00 ACK PSH URGP=0
[*] Running tests...
Chain INPUT (policy DROP)
target     prot opt source               destination
ufw-before-logging-input  all  --  anywhere             anywhere
ufw-before-input  all  --  anywhere             anywhere
ufw-after-input  all  --  anywhere             anywhere
ufw-after-logging-input  all  --  anywhere             anywhere
ufw-reject-input  all  --  anywhere             anywhere
ufw-track-input  all  --  anywhere             anywhere

Chain FORWARD (policy DROP)
target     prot opt source               destination
ufw-before-logging-forward  all  --  anywhere             anywhere
ufw-before-forward  all  --  anywhere             anywhere
ufw-after-forward  all  --  anywhere             anywhere
ufw-after-logging-forward  all  --  anywhere             anywhere
ufw-reject-forward  all  --  anywhere             anywhere
ufw-track-forward  all  --  anywhere             anywhere

Chain OUTPUT (policy ACCEPT)
target     prot opt source               destination
ufw-before-logging-output  all  --  anywhere             anywhere
ufw-before-output  all  --  anywhere             anywhere
ufw-after-output  all  --  anywhere             anywhere
ufw-after-logging-output  all  --  anywhere             anywhere
ufw-reject-output  all  --  anywhere             anywhere
ufw-track-output  all  --  anywhere             anywhere

Chain ufw-after-forward (1 references)
target     prot opt source               destination

Chain ufw-after-input (1 references)
target     prot opt source               destination
ufw-skip-to-policy-input  udp  --  anywhere             anywhere
udp dpt:netbios-ns
ufw-skip-to-policy-input  udp  --  anywhere             anywhere
udp dpt:netbios-dgm
ufw-skip-to-policy-input  tcp  --  anywhere             anywhere
tcp dpt:netbios-ssn
ufw-skip-to-policy-input  tcp  --  anywhere             anywhere
tcp dpt:microsoft-ds
ufw-skip-to-policy-input  udp  --  anywhere             anywhere
udp dpt:bootps
ufw-skip-to-policy-input  udp  --  anywhere             anywhere
udp dpt:bootpc
ufw-skip-to-policy-input  all  --  anywhere             anywhere
ADDRTYPE match dst-type BROADCAST

Chain ufw-after-logging-forward (1 references)
target     prot opt source               destination
LOG        all  --  anywhere             anywhere             limit: avg
3/min burst 10 LOG level warn prefix "[UFW BLOCK] "

Chain ufw-after-logging-input (1 references)
target     prot opt source               destination
```

```
LOG        all  --  anywhere             anywhere             limit: avg
3/min burst 10 LOG level warn prefix "[UFW BLOCK] "

Chain ufw-after-logging-output (1 references)
target     prot opt source               destination

Chain ufw-after-output (1 references)
target     prot opt source               destination

Chain ufw-before-forward (1 references)
target     prot opt source               destination
ACCEPT     all  --  anywhere             anywhere             ctstate
RELATED,ESTABLISHED
ACCEPT     icmp --  anywhere             anywhere             icmp
destination-unreachable
ACCEPT     icmp --  anywhere             anywhere             icmp time-
exceeded
ACCEPT     icmp --  anywhere             anywhere             icmp
parameter-problem
ACCEPT     icmp --  anywhere             anywhere             icmp echo-
request
ufw-user-forward  all  --  anywhere             anywhere

Chain ufw-before-input (1 references)
target     prot opt source               destination
ACCEPT     all  --  anywhere             anywhere
ACCEPT     all  --  anywhere             anywhere             ctstate
RELATED,ESTABLISHED
ufw-logging-deny  all  --  anywhere             anywhere
ctstate INVALID
DROP       all  --  anywhere             anywhere             ctstate
INVALID
ACCEPT     icmp --  anywhere             anywhere             icmp
destination-unreachable
ACCEPT     icmp --  anywhere             anywhere             icmp time-
exceeded
ACCEPT     icmp --  anywhere             anywhere             icmp
parameter-problem
ACCEPT     icmp --  anywhere             anywhere             icmp echo-
request
ACCEPT     udp  --  anywhere             anywhere             udp
spt:bootps dpt:bootpc
ufw-not-local  all  --  anywhere             anywhere
ACCEPT     udp  --  anywhere             mdns.mcast.net       udp dpt:mdns
ACCEPT     udp  --  anywhere             239.255.255.250      udp dpt:1900
ufw-user-input  all  --  anywhere             anywhere

Chain ufw-before-logging-forward (1 references)
target     prot opt source               destination

Chain ufw-before-logging-input (1 references)
target     prot opt source               destination

Chain ufw-before-logging-output (1 references)
target     prot opt source               destination

Chain ufw-before-output (1 references)
target     prot opt source               destination
ACCEPT     all  --  anywhere             anywhere
```

```
ACCEPT     all  --  anywhere             anywhere             ctstate
RELATED,ESTABLISHED
ufw-user-output  all  --  anywhere             anywhere

Chain ufw-logging-allow (0 references)
target     prot opt source               destination
LOG        all  --  anywhere             anywhere             limit: avg
3/min burst 10 LOG level warn prefix "[UFW ALLOW] "

Chain ufw-logging-deny (2 references)
target     prot opt source               destination
RETURN     all  --  anywhere             anywhere             ctstate
INVALID limit: avg 3/min burst 10
LOG        all  --  anywhere             anywhere             limit: avg
3/min burst 10 LOG level warn prefix "[UFW BLOCK] "

Chain ufw-not-local (1 references)
target     prot opt source               destination
RETURN     all  --  anywhere             anywhere             ADDRTYPE
match dst-type LOCAL
RETURN     all  --  anywhere             anywhere             ADDRTYPE
match dst-type MULTICAST
RETURN     all  --  anywhere             anywhere             ADDRTYPE
match dst-type BROADCAST
ufw-logging-deny  all  --  anywhere             anywhere             limit:
avg 3/min burst 10
DROP       all  --  anywhere             anywhere

Chain ufw-reject-forward (1 references)
target     prot opt source               destination

Chain ufw-reject-input (1 references)
target     prot opt source               destination

Chain ufw-reject-output (1 references)
target     prot opt source               destination

Chain ufw-skip-to-policy-forward (0 references)
target     prot opt source               destination
DROP       all  --  anywhere             anywhere

Chain ufw-skip-to-policy-input (7 references)
target     prot opt source               destination
DROP       all  --  anywhere             anywhere

Chain ufw-skip-to-policy-output (0 references)
target     prot opt source               destination
ACCEPT     all  --  anywhere             anywhere

Chain ufw-track-forward (1 references)
target     prot opt source               destination

Chain ufw-track-input (1 references)
target     prot opt source               destination

Chain ufw-track-output (1 references)
target     prot opt source               destination
ACCEPT     tcp  --  anywhere             anywhere             ctstate NEW
ACCEPT     udp  --  anywhere             anywhere             ctstate NEW
```

```
Chain ufw-user-forward (1 references)
target     prot opt source               destination

Chain ufw-user-input (1 references)
target     prot opt source               destination
ACCEPT     tcp  --  anywhere             anywhere             tcp dpt:ssh

Chain ufw-user-limit (0 references)
target     prot opt source               destination
LOG        all  --  anywhere             anywhere             limit: avg
3/min burst 5 LOG level warn prefix "[UFW LIMIT BLOCK] "
REJECT     all  --  anywhere             anywhere             reject-with
icmp-port-unreachable

Chain ufw-user-limit-accept (0 references)
target     prot opt source               destination
ACCEPT     all  --  anywhere             anywhere

Chain ufw-user-logging-forward (0 references)
target     prot opt source               destination

Chain ufw-user-logging-input (0 references)
target     prot opt source               destination

Chain ufw-user-logging-output (0 references)
target     prot opt source               destination

Chain ufw-user-output (1 references)
target     prot opt source               destination
[*] Done! Module status checked.
[*] Script finished with exit code: 0
✅ Workflow завершено.
```