Центральноукраїнський національний технічний університет
Механіко-технологічний факультет
Кафедра кібербезпеки та програмного забезпечення

”Допущено до захисту”
Завідувач кафедри кібербезпеки
та програмного забезпечення
д.т.н., професор
_____ Олексій СМІРНОВ
“____” _____ 2025 р.

ВИПУСКНА КВАЛІФІКАЦІЙНА РОБОТА
за другим (магістерським) рівнем вищої освіти
на тему
“Дослідження та програмна реалізація системи контейнерної віртуалізації мережевої інфраструктури для хостингу”

Виконав здобувач вищої освіти
ІІ курсу, групи КІ-24М
ОПП «Комп’ютерна інженерія»
спеціальності 123 «Комп’ютерна інженерія»
_____ Павлюхін В.Л.
«____» _____ 2025 р.

Керівник проекту
доктор технічних наук, професор
_____ Смірнов О.А.
«____» _____ 2025 р.
Рецензент _____
_____

м. Кропивницький

АНОТАЦІЯ

Павлюхін В.Л. Дослідження та програмна реалізація системи контейнерної віртуалізації мережевої інфраструктури для хостингу. 123 Комп'ютерна інженерія. Центральноукраїнський національний технічний університет. Кропивницький. 2025.

В даній випускній кваліфікаційній роботі за другим (магістерським) рівнем вищої освіти розроблено програмне забезпечення, яке призначено для системи контейнерної віртуалізації мережевої інфраструктури для хостингу.

Метою розробки є дослідження та програмна реалізація системи контейнерної віртуалізації мережевої інфраструктури для хостингу.

Об'єктом дослідження є процес контейнерної віртуалізації мережевої інфраструктури для хостингу.

Предметом дослідження є методи контейнерної віртуалізації мережевої інфраструктури для хостингу.

Методи дослідження базуються на методах побудови комп'ютерних мереж, методах математичної статистики, методах розробки програмного забезпечення.

Результат роботи – програмна реалізація системи контейнерної віртуалізації мережевої інфраструктури для хостингу.

В процесі роботи над програмною моделлю виконано аналіз існуючих апаратних та програмних засобів. В повній мірі описані всі компоненти розробленого програмного забезпечення.

Розроблено зручний інтерфейс користувача. Наведені інструкції по роботі з програмними засобами.

Програма може використовуватися на ПЕОМ з ОС Windows 10/11.

Програму розроблено в середовищі Python.

Ключові слова: комп'ютерна інженерія, контейнерна віртуалізація, мережева інфраструктура, хостинг

ABSTRACT

Pavliukhin V.L. Research and software implementation of a container virtualization system for network infrastructure for hosting. 123 Computer Engineering. Central Ukrainian National Technical University. Kropyvnytskyi. 2025.

In this final qualification work for the second (master's) level of higher education, software has been developed, which is intended for a container virtualization system for network infrastructure for hosting.

The purpose of the development is the research and software implementation of a container virtualization system for network infrastructure for hosting.

The object of the research is the process of container virtualization of network infrastructure for hosting.

The subject of the research is methods of container virtualization of network infrastructure for hosting.

The research methods are based on methods of building computer networks, methods of mathematical statistics, methods of software development.

The result of the work is a software implementation of a container virtualization system for network infrastructure for hosting.

In the process of working on the software model, an analysis of existing hardware and software was performed. All components of the developed software are fully described.

A convenient user interface has been developed. Instructions for working with software are provided.

The program can be used on a PC with Windows 10/11.

The program was developed in the Python environment.

Keywords: computer engineering, container virtualization, network infrastructure, hosting

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, ОДИНИЦЬ І ТЕРМІНІВ

DMA – Direct Memory Access;

DOS – дискова операційна система;

RS-232C – Recommended Standart 232 Version C, ревізія – EIA-232D;

USRT (Universal Synchronous Receiver/Transmitter) – універсальний синхронний прийомопередавач;

АЛП– арифметико–логічний пристрій;

АЦП – аналогово–цифровий перетворювач;

ВДТ – відеодісплейні термінали;

ЕЛТ – електронно–люмінісцентна трубка;

ЕОМ – електронна обчислювальна машина;

ЗПР– запит переривання;

ОС – операційна система;

ПДП – теж саме що і DMA;

ПЗП – постійний запам’ятовуючий пристрій;

ПЗП – постійний запом’ятовуючий пристрій;

ПК – персональний комп’ютер;

ППЗП – перепрограмувальний постійний запом’ятовуючий пристрій;

ПЧ – перетворювач частоти;

РА – регістр адреси;

РК – регістр команд;

РМП – регістр маски переривань;

РП – регістр пріоритетів;

ТД – технічна документація;

ТЗ – технічна задача;

ЦАП – цифро–аналоговий перетворювач;

ВСТУП

Актуальність теми. З появою контейнеризації 10 років тому ми побачили компактний, зручний та портативний спосіб запуску програм безпосередньо одночасно з віртуалізацією. Основна відмінність полягає в архітектурі. Контейнери мають те саме ядро, що й гостьова система, і тому не віртуалізують низькорівневі компоненти, такі як центральний процесор (ЦП).

З одного боку, вони легші та гнучкіші, ніж віртуальні машини (ВМ). З іншого боку, ВМ можуть точніше задовольняти потреби низькорівневих компонентів і є повністю автономними системами. Яка архітектура найкраще підходить для розробки програми сьогодні?

У даній магістерській роботі ми вивчимо два основні віртуальні методи розгортання. Ми порівняємо обидва методи за кількома критеріями: сумісність на основі користувацького досвіду та простота встановлення/розгортання, масштабованість на основі автоматичної еластичності робочого навантаження та енергоефективність з точки зору енергії та комп'ютерних ресурсів. Після тестів ми розробимо програмну реалізацію системи контейнерної віртуалізації мережевої інфраструктури для хостингу.

Мета й завдання дослідження. Метою роботи є дослідження та програмна реалізація системи контейнерної віртуалізації мережевої інфраструктури для хостингу.

Для досягнення поставленої мети визначена програма дослідження, що складається з наступних завдань:

– Огляд існуючих систем контейнерної віртуалізації мережевої інфраструктури для хостингу.

– Дослідження системи контейнерної віртуалізації мережевої інфраструктури для хостингу.

– Програмна реалізація системи контейнерної віртуалізації мережевої інфраструктури для хостингу.

Об’єктом дослідження є процес контейнерної віртуалізації мережевої інфраструктури для хостингу.

Предметом дослідження є методи контейнерної віртуалізації мережевої інфраструктури для хостингу.

Методи дослідження базуються на методах побудови комп’ютерних мереж, методах математичної статистики, методах розробки програмного забезпечення.

Наукова новизна отриманих результатів. У процесі рішення завдань, обумовлених цілями дослідження, отримані наступні результати:

– Удосконалено метод контейнерної віртуалізації мережевої інфраструктури для хостингу.

– Розроблено вітчизняний продукт контейнерної віртуалізації мережевої інфраструктури для хостингу, який має більш широкі можливості, на відміну від існуючих аналогів.

Практична цінність отриманих результатів полягає в тому, що розроблені алгоритми дозволяють успішно вирішувати задачі контейнерної віртуалізації мережевої інфраструктури для хостингу.

Достовірність наукових результатів підтверджена теоретичними викладеннями, даними комп'ютерного моделювання, коректними дослідженнями параметрів на функціонуючій обчислювальній мережі, а також відповідністю отриманих результатів окремим результатам, наведеним у науковій літературі.

Робота апробована на LVII Науково-технічній конференції здобувачів вищої освіти LV науково-технічної конференції «Наука в ЦНТУ: основні досягнення та перспективи розвитку» (2025 р.), основні положення випускної кваліфікаційної роботи за другим (магістерським) рівнем вищої освіти надруковані у статті збірника праць молодих науковців ЦНТУ, випуск №15.

Таким чином, виходячи з вищеперерахованого, дослідження та програмна реалізація системи контейнерної віртуалізації мережевої інфраструктури для хостингу, є актуальною задачею, яка потребує вирішення у даній випускній кваліфікаційній роботі за другим (магістерським) рівнем вищої освіти.

1 ПРИЗНАЧЕННЯ ТА ОБЛАСТЬ ВИКОРИСТАННЯ

1.1 Призначення системи

Контейнеризація та віртуалізація – це технології, які дозволяють запускати кілька програм або робочих навантажень на одній фізичній машині або хості.

Контейнери та віртуальні машини – це два найпопулярніші підходи до налаштування програмної інфраструктури для вашої організації.

Контейнери зараз є важливим гравцем у хмарній розробці. У поєднанні з віртуальними машинами (ВМ) або окремо вони пропонують незліченні переваги для вашої IT-системи.

Багатьом компаніям важко зрозуміти різницю між контейнеризацією та віртуалізацією. Вони відрізняються можливостями, але схожі в деяких аспектах. Обидва підвищують ефективність, впроваджують гнучкість, забезпечують масштабованість, допомагають DevOps та оптимізують життєвий цикл розробки.

Ви можете використовувати контейнеризацію та віртуалізацію разом, щоб підвищити ефективність вашої IT-команди та задовольнити потреби бізнесу. Однак вони також можуть заплутати людей, які тільки починають працювати з інструментами віртуалізації.

Давайте розглянемо деякі основні факти про контейнери та віртуальні машини, як вони працюють, пов'язані з ними проблеми та чи підходять вони для вашого бізнесу.

1.2 Область застосування

Віртуальна машина (ВМ) – це технологія для стимуляції фізичного комп'ютера. Вона містить ті самі компоненти, операційну систему (ОС), мережевий інтерфейс та програми. Однак вона знаходиться в ізоляції всередині

фізичного комп'ютера. Це означає, що на одному комп'ютері може працювати кілька віртуальних машин та їхніх ізольованих компонентів. Їх можна використовувати для розробки, підготовки та створення коду програми. За допомогою віртуальних машин можна створювати віртуалізовані обчислювальні середовища, які вважаються першим поколінням хмарних обчислень.

Віртуальна машина не може працювати без гіпервізора. Ці легкі програмні шари розділяють віртуальні машини та розподіляють процесори, пам'ять і сховище. Вони, по суті, є моніторами машин, які дозволяють кільком операційним системам працювати одночасно.

Таким чином, виходячи з вищеперерахованого, дослідження та програмна реалізація системи контейнерної віртуалізації мережевої інфраструктури для хостингу, є актуальною задачею, яка потребує вирішення у даній випускній кваліфікаційній роботі за другим (магістерським) рівнем вищої освіти.

2 ПЕРЕГЛЯД АНАЛОГІЧНИХ ІСНУЮЧИХ СИСТЕМ

2.1 Огляд існуючих систем, технологій, архітектур, програмних рішень за профілем теми випускної кваліфікаційної роботи за другим (магістерським) рівнем вищої освіти

Контейнеризація – це новий спосіб розгортання програм на віддаленому сервері. Традиційно ми копіювали вихідний код програми та виконували його на віддаленому сервері.

Однак, технологія контейнеризації дозволяє нам об'єднати код програми та всі її необхідні залежності, бібліотеки, файли конфігурації та бінарні файли в єдиний ізольований блок, який називається контейнером. Цей образ контейнера є самодостатнім пакетом, який може стабільно працювати в різних обчислювальних середовищах, від ноутбука розробника до виробничих серверів у хмарі або локальних центрів обробки даних.

У Linux контейнеризація використовує функції віртуалізації на рівні операційної системи, такі як простори імен та групи керування (cgroups). На відміну від традиційних віртуальних машин (ВМ), які потребують повноцінної гостьової операційної системи для кожного екземпляра, контейнери використовують ядро ОС хост-системи. Це робить контейнери дуже легкими, швидшими для запуску та менш ресурсоємними, ніж віртуальні машини. Це дозволяє розробникам розгортати кілька контейнерів на одній ВМ для вищої щільності та ефективнішого використання базових апаратних ресурсів.

Найкращі альтернативи Docker для контейнеризації

Docker – це, безумовно, найпопулярніше середовище виконання контейнерів. Настільки, що багато людей використовують терміни «Docker» та «контейнери» як взаємозамінні, але це не єдина технологія контейнеризації.

Давайте розглянемо альтернативні середовища виконання контейнерів, які

можна використовувати замість Docker:

1. Podman https://podman.io/

2. Linux Containers https://linuxcontainers.org/

3. Red Hat OpenShift https://www.redhat.com/en/technologies/cloud-computing/openshift

4. Apptainer/Singularity https://apptainer.org/

5. Containerd https://containerd.io/

6. Cri-o https://cri-o.io/

7. Mirantis Container Runtime https://www.mirantis.com/software/mirantis-container-runtime/

Podman

Podman – це корисний інструмент для всіх, хто працює з програмними контейнерами. Він дозволяє легко знаходити, завантажувати, запускати, збирати та ділитися контейнерами за допомогою простих команд, таких як пошук, витягування, виконання, збірка та надсилання. Особливістю Podman є його здатність групувати пов'язані контейнери в «поди», що спрощує керування програмами, де різні частини повинні тісно співпрацювати, подібно до того, як працюють більші системи, такі як Kubernetes.

Якщо ви віддаєте перевагу візуальному інтерфейсу замість введення команд, ви можете скористатися застосунком Podman Desktop у Windows, macOS та Linux. Цей застосунок надає вам єдиний екран для керування контейнерами, навіть якщо вони були створені за допомогою інших інструментів, таких як Docker.

Podman Desktop спрощує створення нових образів контейнерів, а також отримання образів з онлайн-репозиторіїв, групування контейнерів у поди та перевірку журналів. Він навіть допомагає вам підготувати та перенести ваші контейнерні програми для роботи на Kubernetes.

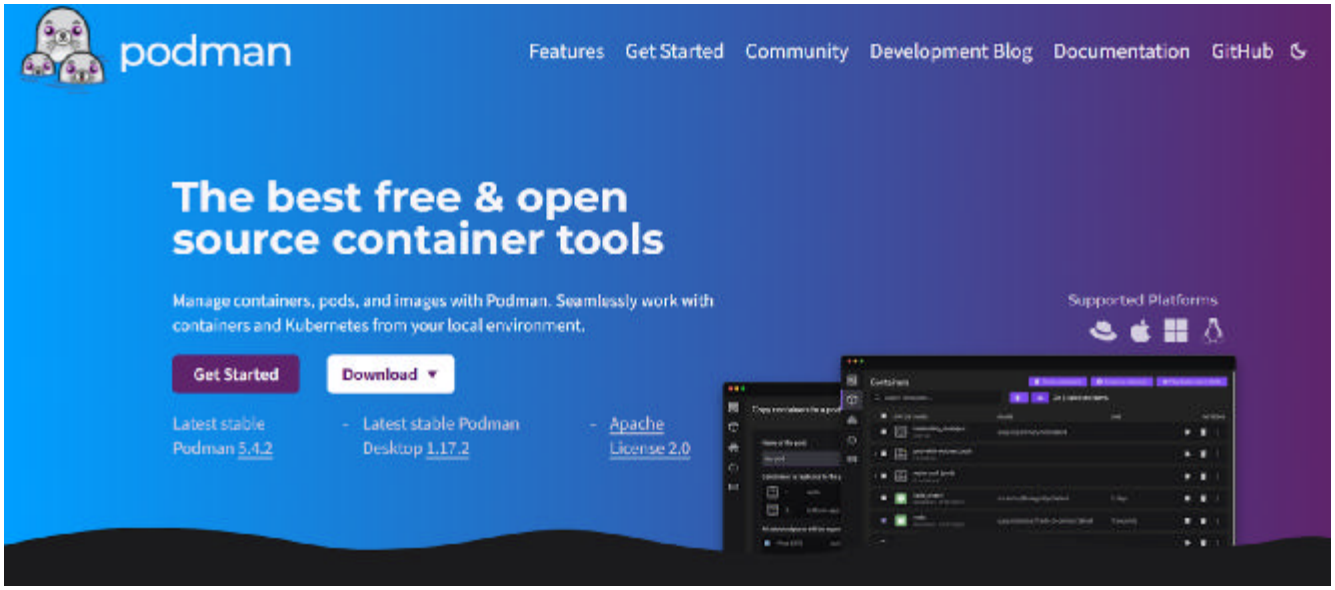

Рисунок 2.1 – Інтерфейс користувача Podman

Linux Containers

Linux Containers, часто скорочено LXC, є одним із найстаріших варіантів, побудованих безпосередньо на ядрі Linux, і включають такі функції, як простори імен і групи. LXC прагне створювати середовища, близькі до стандартної інсталяції Linux, але без окремого ядра. Це дещо відрізняється від Docker, який зазвичай зосереджується на упаковці однієї програми та її залежностей.

LXC більше орієнтований на запуск «системного контейнера» – легкої віртуальної машини, яка може запускати кілька служб або повноцінну систему ініціалізації всередині. Такий підхід «системного контейнера» може не бути повноцінною заміною вашого робочого процесу Docker і може вимагати налаштування способу розгортання програм. LXC використовує потужні функції Linux та пропонує інструменти керування й бібліотеки (наприклад, liblxc). Він імітує повноцінне середовище ОС, що може бути більше, ніж потрібно більшості людей для простої ізоляції програм. LXC може бути чудовим варіантом, якщо вам потрібно відтворити традиційну конфігурацію сервера в контейнері, але якщо ви просто хочете запускати свої програми, це може призвести до зайвої складності.

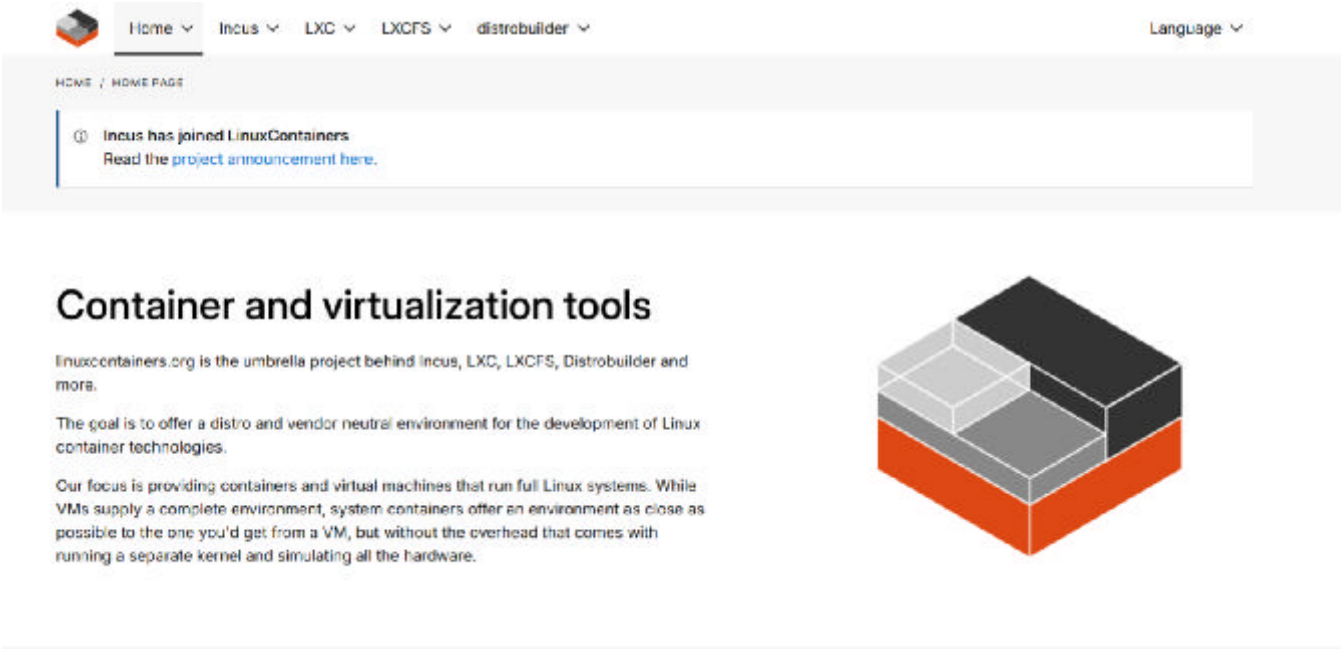

Рисунок 2.2 – Інтерфейс користувача Linux Containers

Red Hat OpenShift

Red Hat OpenShift – це набагато більше, ніж просто середовище виконання контейнера. Це повноцінна платформа додатків, побудована на Kubernetes, призначена для обробки всього життєвого циклу додатків, від розробки та збірки до розгортання та управління в масштабі, навіть у різних хмарних середовищах або на ваших власних серверах.

Якщо ви просто хочете створювати та запускати контейнери, це може бути не той інструмент, який вам потрібен. Red Hat OpenShift розроблений для забезпечення узгодженого середовища з інтегрованими інструментами для створення, автоматизації розгортання (наприклад, конвеєрів CI/CD) та керування додатками, а не лише для базової оркестрації контейнерів.

Платформа пропонує різні способи використання: як керований сервіс у хмарах, таких як AWS або Azure, де Red Hat керує базовою інфраструктурою, або як самостійно керований сервіс для більшого контролю. Він також має вбудовані засоби безпеки, інструменти для розробників та можливість керувати віртуальними машинами разом із контейнерами.

Хоча Red Hat OpenShift – потужний інструмент, особливо для великих команд або складних програм, яким потрібне таке надійне керування та безпека, він також складніший, ніж просто використання Docker. Тому вам слід зважити, чи виправдовує комплексний функціонал потенційну криву навчання та операційні накладні витрати для ваших потреб.

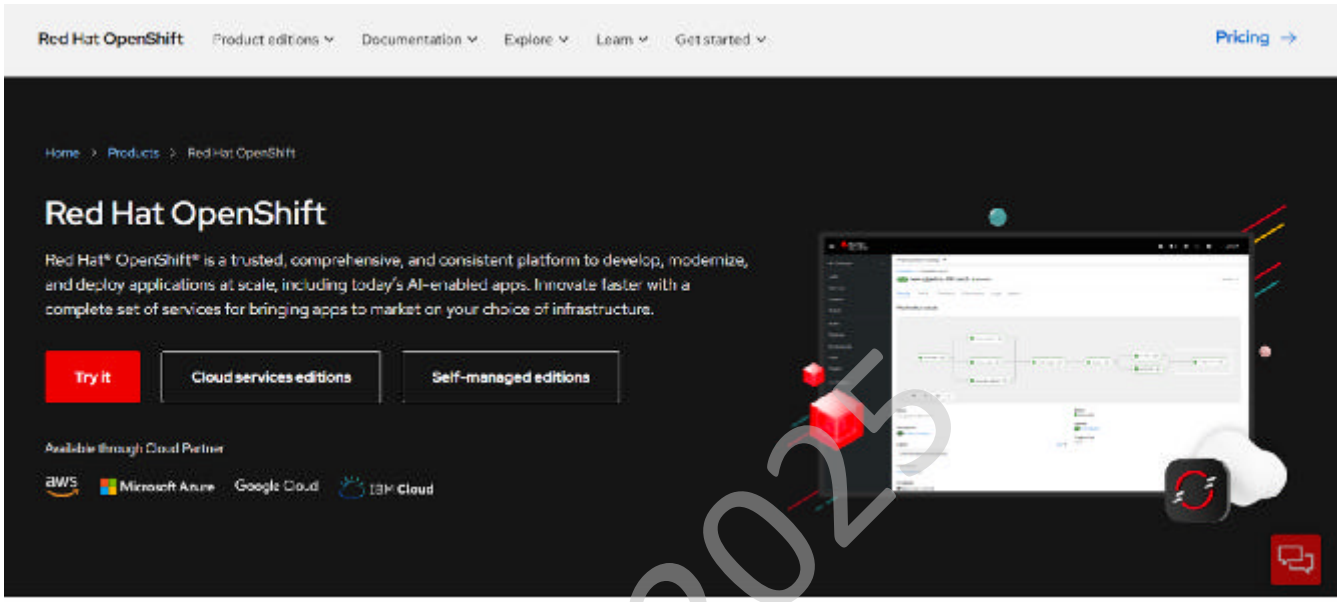

Рисунок 2.3 – Інтерфейс користувача Red Hat OpenShift

Apptainer

Apptainer, який раніше називався Singularity, – це ще один інструмент для пакування та запуску програмного забезпечення всередині контейнерів, подібно до того, як працює Docker. Це програмне забезпечення з відкритим кодом, яке зараз є частиною Linux Foundation і розроблене для простоти, швидкості та безпеки. Apptainer особливо популярний у середовищах, де багато людей використовують одні й ті ж комп'ютерні системи, наприклад, в університетських обчислювальних кластерах або дослідницьких лабораторіях, а також для запуску програмного забезпечення, яке потребує великої обчислювальної потужності.

Він в першу чергу зосереджений на ресурсомістких застосунках, що зазвичай зустрічаються у високопродуктивних обчисленнях (HPC), наукових дослідженнях та робочих навантаженнях штучного інтелекту/машинного

навчання. Він розроблений для середовищ, де дуже важливими є максимізація обчислювальної продуктивності, керування складними програмними стеками, забезпечення відтворюваності та робота зі спеціалізованим обладнанням, таким як графічні процесори.

Apptainer відрізняється тим, що обробляє контейнери та взаємодіє з комп'ютером, на якому він працює. Він упаковує все в один файл, що спрощує копіювання, переміщення контейнера між комп'ютерами або обмін ним з іншими. Apptainer також дозволяє програмному забезпеченню всередині контейнера легко використовувати спеціальне обладнання на хост-машині, таке як потужні відеокарти (GPU) або швидкі мережеві з'єднання, що важливо для наукових обчислень.

Його підхід до безпеки також досить простий: за замовчуванням у вас є ті ж самі дозволи всередині контейнера, що й зовні, що допомагає запобігти випадковому отриманню користувачами додаткових привілеїв у системі.

Containerds

Containerds – це середовище виконання контейнерів, орієнтоване на управління повним життєвим циклом контейнера. Це включає такі завдання, як передача та зберігання зображень, виконання та контроль контейнера, низькорівневе сховище та мережеві підключення.

Вас може здивувати, що Docker використовує containerd під капотом (або компоненти, похідні від нього), а це означає, що containerd не обов'язково є заміною всього досвіду розробника Docker, а радше компонентом рушія, який виконує важку роботу.

Однак, вона набагато нижчого рівня, ніж команда Docker run, оскільки надає розробникам чіткі етапи створення та виконання контейнера. Замість того, щоб надавати розробникам простий, універсальний інтерфейс командного рядка, вона розроблена радше для інтеграції у більші системи або для користувачів, яким потрібен детальний контроль.

Він має добре налагоджену документацію, API та клієнтські бібліотеки,

зокрема для клієнта Go для програмного керування. Ви можете легко використовувати цей клієнт для підключення до демона, отримання образів, створення специфікацій OCI, керування знімками (контейнерними файловими системами) та багато іншого.

Хоча containerd є ключовою частиною екосистеми контейнерів та реалізації стандартного інтерфейсу виконання (CRI) для Kubernetes, він не замінює безпосередньо інструмент командного рядка Docker, орієнтований на користувача, та пов'язані з ним функції збірки/композиції. Натомість він замінює частину середовища виконання, якою традиційно керував Docker.

Якщо вам потрібен лише компонент середовища виконання, containerd – найкращий варіант. Однак для реплікації повного робочого процесу розробника Docker потрібні інші інструменти (такі як nerdctl для Docker-сумісного CLI або інструменти збірки, такі як BuildKit).

CRI-O

Якщо ви працюєте з Kubernetes, ви, можливо, вже знайомі з CRI-O. Це легка реалізація інтерфейсу виконання контейнерів (CRI) Kubernetes. Це означає, що його основна мета не бути універсальним контейнерним рушієм, як Docker, а радше надавати саме те, що потрібно Kubernetes для ефективного та надійного управління життєвими циклами контейнерів (подами), використовуючи стандартні середовища виконання, сумісні з OCI, такі як runc.

CRI-O виступає посередником між кублетом Kubernetes та низькорівневими операціями з контейнерами. Він обробляє вилучення образів з будь-якого реєстру, сумісного з OCI, керує сховищем контейнерів, генерує специфікацію середовища виконання OCI, запускає власне середовище виконання (наприклад, runc), налаштовує мережу через CNI та використовує conmon для моніторингу. Зосереджуючись лише на цих важливих для Kubernetes завданнях, він прагне бути більш стабільним та ресурсоефективним у кластері, ніж більш багатофункціональний демон, як-от Docker.

Якщо ви думаєте про заміну Docker, ви можете використовувати CRI-O

для заміни компонента середовища виконання на вузлах Kubernetes. Однак, слід зазначити, що він не пропонує прямої заміни інтерфейсу командного рядка Docker або інструментів, таких як Docker Compose, для локальних робочих процесів розробки.

Mirantis Container Runtime

Mirantis Container Runtime (MCR) схожий на «Docker Engine – Enterprise». Він розроблений для сумісності з основним API Docker та командами, які ви, можливо, вже знаєте. Головна мета MCR – забезпечити цю знайому функціональність Docker Engine, але спеціально адаптовану для потреб підприємства, разом із комерційною підтримкою (наприклад, цілодобовими опціями) та розширеними функціями безпеки, які можуть бути необхідними, якщо ваша організація має суворіші вимоги, ніж ті, що пропонує стандартний Docker з відкритим кодом.

MCR робить великий акцент на аспектах безпеки, які часто вимагаються великими організаціями або регульованими галузями. Наприклад, він використовує криптографію, перевірену стандартом FIPS 140-2, має безпечні конфігурації за замовчуванням і пропонує такі можливості, як забезпечення використання цифрових підписаних образів для захисту ланцюжка постачання програмного забезпечення.

Він має широкі можливості, оскільки підтримує контейнери як Linux, так і Windows. Він може працювати в автономному режимі, як частина розгортання Kubernetes або в кластерах Docker Swarm. Це означає, що ви можете використовувати його в різних налаштуваннях інфраструктури, не вимагаючи повного перегляду стратегій оркестрації.

Підсумки

Вибір правильного середовища виконання контейнера – це критично важливе рішення, яке значною мірою залежить від потреб вашої команди, вашої інфраструктури та типу програм, які ви створюєте. Хоча Docker роками був вибором за замовчуванням, очевидно, що ландшафт контейнерів у 2025 році

пропонує потужні альтернативи, такі як Podman, LXC, containerds, CRI-O та OpenShift, кожна з яких має унікальні сильні сторони.

Якщо ви керуєте великими кластерами Kubernetes, CRI-O або containerd можуть мати сенс. Якщо вам потрібне рішення без демонів із надійними принципами безпеки, Podman – переконливий варіант. А якщо ви працюєте у високорегульованому корпоративному середовищі, Mirantis Container Runtime забезпечує сумісність з Docker із посиленими функціями безпеки.

Однак, хоча вибір правильної платформи контейнеризації є критично важливим, ефективне управління серверами та розгортаннями не менш важливе – і саме тут на допомогу приходить RunCloud.

RunCloud спрощує керування сервером для розробників та команд, які працюють з контейнерними або традиційними PHP-додатками. Незалежно від того, чи використовуєте ви Docker, containerd чи інше середовище виконання, RunCloud допоможе вам:

– Швидше розгортання веб-застосунків.

– Налаштуйте автоматичне резервне копіювання.

– Керуйте безпекою сервера за допомогою впроваджених найкращих практик.

– Відстежуйте продуктивність з єдиної панелі інструментів.

– Легко масштабуйте проекти в міру зростання вашої інфраструктури.

Замість того, щоб турбуватися про глибинні складнощі серверів та розгортань, ви можете повністю зосередитися на створенні та постачанні кращого програмного забезпечення.

2.2 Обґрунтування вибору засобів для побудови системи та мови програмування

Як мова програмування обрана Python. Python – високорівнева мова програмування загального призначення з акцентом на продуктивність

розроблювача й читаність коду. Синтаксис ядра Python мінімалістичний. У той же час стандартна бібліотека включає великий обсяг корисних функцій.

Python підтримує кілька парадигм програмування, у тому числі структурне, об'єктно-орієнтоване, функціональне, імперативне й аспектно-орієнтоване. Основні архітектурні риси – динамічна типізація, автоматичне керування пам'яттю, повна інтроспекція, механізм обробки виключень, підтримка багатопоточні обчислень і зручні високорівневі структури даних. Код у Python організовується у функції й класи, які можуть поєднуватися в модулі (які у свою чергу можуть бути об'єднані в пакети).

Еталонною реалізацією Python є інтерпретатор CPython, що підтримує більшість активно використовуваних платформ. Він поширюється вільно під дуже ліберальною ліцензією, що дозволяє використовувати його без обмежень у будь-яких застосунках, включаючи пропрієтарні. Є реалізації інтерпретаторів для JVM (з можливістю компіляції), MSIL (з можливістю компіляції), LLVM і інших. Проект PyPy пропонує реалізацію Python на самому Python, що зменшує витрати на зміни мови й постановку експериментів над новими можливостями.

Python – мова програмування, що активно розвивається, нові версії (з додаванням/зміною язикових властивостей) виходять приблизно раз у два з половиною року. Внаслідок цього й деяких інших причин на Python відсутні ANSI, ISO або інші офіційні стандарти, їхня роль виконує CPython.

Python портований і працює майже на всіх відомих платформах – від КПК до мейнфреймів. Існують порти під Microsoft Windows, практично всі варіанти UNIX (включаючи FreeBSD і Linux), Plan 9, Mac OS і Mac OS X, iPhone OS 2.0 і вище, Palm OS, OS/2, Amiga, AS/400 і навіть OS/390, Symbian і Android.

При цьому, на відміну від багатьох портуємих систем, для всіх основних платформ Python має підтримку характерних для даної платформи технологій (наприклад, Microsoft COM/DCOM). Більше того, існує спеціальна версія Python для віртуальної машини Java – Jython, що дозволяє інтерпретаторові виконуватися на будь-якій системі, що підтримує Java, при цьому класи Java

можуть безпосередньо використовуватися з Python й навіть бути написаними на Python. Також кілька проектів забезпечують інтеграцію із платформою Microsoft.NET, основні з яких – IronPython і Python.Net.

Python підтримує динамічну типізацію, тобто тип змінної визначається тільки під час виконання. Тому замість «присвоювання значення змінної» краще говорити про «зв'язування значення з деяким ім'ям». У Python є убудовані типи: бульові, рядки, Unicode-рядки, цілі числа довільної точності, числа із плаваючої коми, комплексні числа й деякі інші. З колекцій Python підтримує кортежі (tuples), списки, словники (асоціативні масиви) і, починаючи з версії 2.4, безлічі. Всі значення в Python є об'єктами, у тому числі функції, методи, модулі, класи.

Додати новий тип можна або написавши клас (class), або визначивши новий тип у модулі розширення (наприклад, написаному мовою С). Система класів підтримує спадкування (одиночне й множинне) і метапрограмування. Можливе спадкування від більшості убудованих типів і типів розширень.

Всі об'єкти діляться на посилальні й атомарні. До атомарного ставляться int, long, complex і деякі інші. При присвоюванні атомарних об'єктів копіюється їхнє значення, у той час як для посилальних копіюється тільки покажчик на об'єкт, таким чином, обидві змінні після присвоювання використовують те саме значення. Посилальні об'єкти бувають змінювані й незмінні. Наприклад, рядки й кортежі є незмінними, а списки, словники й багато інших об'єктів – змінюваними. Кортеж у Python є, по суті, незмінним списком. У багатьох випадках кортежі працюють швидше списків, тому якщо ви не плануєте змінювати послідовність, то краще використовувати саме їх. Мова має чіткий і послідовний синтаксис, продуману модульність й масштабованість, завдяки чому вихідний код написаних на Python програм легко читаємий. Python – стабільна й розповсюджена мова. Він використовується в багатьох проектах і в різних якостях: як основна мова програмування або для створення розширень і інтеграції застосунків. На Python реалізоване велика кількість проектів, також він активно використовується для створення прототипів майбутніх програм. Python використовується в багатьох великих компаніях.

2.3 Розгорнута постановка завдання

Згідно з технічним завданням на випускну кваліфікаційну роботу за другим (магістерським) рівнем вищої освіти, реалізації підлягає програмне забезпечення, яке призначено для системи контейнерної віртуалізації мережевої інфраструктури для хостингу.

В процесі розробки випускної кваліфікаційної роботи за другим (магістерським) рівнем вищої освіти необхідно виконати наступний обсяг роботи:

а) провести аналіз існуючих систем-аналогів для виявлення їх позитивних і негативних якостей. Результати аналізу врахувати в подальших розробках;

б) вибрати та обґрунтувати методику побудови системи контролю роботи технологічного обладнання на виробництві в автоматизованому режимі. Розробити функціональну та структурну схеми системи;

в) розробити програмне забезпечення системи, що дозволить реалізувати поставлену технічним завданням задачу. Побудувати блок-схеми алгоритмів програми та підпрограми;

г) організувати інтерфейс користувача з метою формування та виводу на екран ЕОМ повідомлень про некоректні дії користувача та нестандартні ситуації в роботі технологічного обладнання;

д) розробити рекомендації по організаційних та методичних заходах, які забезпечать впровадження системи в промислову експлуатацію та її подальшу успішну експлуатацію;

е) провести розрахунки по визначенню економічної ефективності розробленої системи;

ж) розробити заходи по охороні праці при впровадженні та експлуатації системи, а також розробити заходи з цивільного захисту;

з) сформувати висновки про виконаний обсяг робіт та одержані результати.

3 ОПИС І ОБҐРУНТУВАННЯ ПРОЕКТНИХ РІШЕНЬ

3.1 Опис функціонування системи

Віртуалізація – це процес використання програмного забезпечення для створення віртуального ресурсу, який працює на окремому рівні від фізичного обладнання. Найпоширенішим випадком використання віртуалізації є хмарні обчислення.

Ви можете запускати кілька віртуальних машин на одному комп'ютері за допомогою віртуалізації. Ці віртуальні машини є незалежними системами, але мають спільну фізичну IT-інфраструктуру та керуються гіпервізором.

Віртуалізація набула величезного значення в останній галузі програмного забезпечення. Очікується, що до 2026 року світовий ринок віртуалізації додатків становитиме 5,76 мільярда доларів. Це пояснюється тим, що віртуалізація дозволяє користувачам отримувати доступ до додатків і функцій без їх встановлення на комп'ютер.

Ця хмарна технологія економить гроші, час і місце для зберігання, пропонуючи всі можливості хмарних обчислень. Від неї вигоду отримують як великі підприємства, так і малий бізнес. Деякі з переваг віртуалізації:

- Доступність усіх ресурсів ОС для програм.
- Добре налагоджена функціональність.
- Кращі інструменти та засоби контролю безпеки.
- Надійні інструменти управління.
- Економія коштів та висока ефективність.
- Централізоване робоче навантаження без накладних витрат.

VirtualBox, VMware Workstation Player та Microsoft Hyper-V є найпопулярнішими постачальниками віртуальних машин.

Обмеження віртуалізації

Віртуалізація – це потужна технологія, яка має багато переваг, але також має кілька обмежень, про які слід знати. Деякі з основних обмежень віртуалізації включають:

- Для роботи кількох операційних систем та віртуальної копії обладнання потрібні значні ресурси оперативної пам'яті та процесора.
- Перехід між приватними та публічними хмарами та центрами обробки даних ускладнює життєвий цикл розробки програмного забезпечення.
- Він монолітний і запускає програми як окремі великі файли.
- Додає обчислювальні ресурси та дуже швидко циклічно перезавантажується/завантажується.
- Він не запускає деякі програми належним чином.
- Деякі старіші або спеціалізовані програми можуть бути несумісними з програмним забезпеченням для віртуалізації або можуть вимагати додаткового налаштування для належної роботи.
- Віртуалізовані середовища можуть бути не такими простими в масштабуванні, як фізичні, особливо коли йдеться про додавання додаткових апаратних ресурсів.

Контейнери – це засіб ізоляції програми від її оточення шляхом інкапсуляції її залежностей та конфігурацій в одному блоці. Після цього цей блок можна перенести в інші середовища, такі як приватні хмари, публічні хмари та центри обробки даних.

Контейнери є легшими та гнучкішими, коли справа доходить до віртуалізації вашого середовища без гіпервізора. Вони дозволяють DevOps зосередитися на розробці та розгортанні коду, що забезпечує швидше надання ресурсів. Контейнеризований додаток поводиться послідовно в середовищах розробки, тестування та виробництва.

Як згадувалося раніше, контейнеризація – це процес пакування кожного компонента, необхідного для запуску програми або мікросервісу, включаючи

пов'язані бібліотеки. Кожен контейнер складається з коду, залежностей та самої ОС. Це дозволяє програмам працювати однаково на різних платформах.

Контейнеризація – це форма віртуалізації ОС, яка використовує можливості операційної системи для ізоляції процесів та контролю їх доступу до пам'яті, дискового простору та процесорів.

Поширення контейнеризації розпочалося з Docker, платформи з відкритим кодом для створення, розгортання та управління контейнерними додатками. З її появою у 2013 році технологія та екосистема контейнерів зазнали значного розвитку.

Деякі переваги контейнеризації:

- Зменшення завантаженості ресурсів управління IT.
- Вимоги до меншого розміру.
- Швидше завантаження та спрощені оновлення безпеки.
- Менше коду для міграції, перенесення або завантаження робочих навантажень.
- Швидша доставка.
- Легше управління.

Контейнеризація працює шляхом спільного використання ядра хост-ОС з іншими контейнерами як ресурсу лише для читання. Ви можете розгорнути кілька контейнерів на одному сервері або віртуальній машині, оскільки вони легкі та масштабовані.

Таким чином, ви підтримуєте лише одну ОС і не присвячуєте цілий сервер одній програмі. Контейнеризація є відповіддю на кілька проблем DevOps. Саме тому багато підприємств застосовують цей підхід для міграції керованих сервісів у хмару.

Контейнери дозволяють розбивати програми на найменші компоненти або мікросервіси. Ці сервіси розробляються та розгортаються незалежно, що усуває монолітність.

Наприклад, якщо ви підтримуєте кілька кнопок дій на своєму веб-сайті,

збій однієї з них не впливає на продуктивність інших. Це зменшує час простою, навантаження на обслуговування та залежність.

Обмеження контейнеризації

Як і віртуальні машини, контейнери також мають деякі обмеження.

- Усі контейнери повинні працювати на схожих операційних системах.
- Якщо контейнери базуються на іншій ОС, їм потрібен інший хост.
- Вони можуть створювати вразливості безпеки в ядрі ОС, оскільки всі контейнери на хост-машині використовують цю ОС.
- Це рішення все ще розробляється та вдосконалюється, тому його впровадження може бути складнішим.

3.2 Розробка структурної схеми

Контейнеризація та віртуалізація мають свої сильні та слабкі сторони. Вони використовуються незалежно для задоволення потреб вашого бізнесу. Їх також можна використовувати разом для створення ефективної IT-інфраструктури для DevOps.

1. Ізоляція

Контейнери та віртуальні машини забезпечують різні ступені ізоляції. Контейнерній системі потрібна базова ОС для надання базових послуг усім контейнеризованим програмам. З іншого боку, гіпервізор запускає віртуальні машини зі своєю ОС, яка використовує апаратну підтримку.

В результаті, контейнерні системи мають менші накладні витрати, ніж віртуальні машини, і зазвичай орієнтовані на середовища з тисячами контейнерів. Вони забезпечують ізоляцію сервісів між контейнерами, тоді як віртуальна машина забезпечує повністю ізольоване середовище. Це призводить до обмеженого доступу до ресурсів для контейнерних сервісів, таких як файлові системи.

Щоб поєднати можливості контейнеризації та віртуалізації, можна обрати паравіртуалізацію. Вона ізолює програми завдяки підтримці віртуальної пам'яті та вимагає спеціальних драйверів пристроїв у віртуальній машині, підключеній до ОС через гіпервізор.

Простіше кажучи, віртуальні машини, якими керують гіпервізори, використовують обладнання віртуальної машини, тоді як контейнерні системи надають служби ОС від базового хоста для ізоляції програм за допомогою обладнання віртуальної пам'яті.

2. Операційна система

Контейнери та віртуальні машини суттєво відрізняються з точки зору автономності ОС. Контейнери дотримуються віртуалізації ОС, що означає, що вони використовують ресурси хост-ОС. З іншого боку, кожен екземпляр у віртуальній машині є повноцінною гостьовою ОС сам по собі.

Таким чином, віртуальна машина містить гостьову ОС, віртуальну копію обладнання для її запуску, програму, її бібліотеки та залежності. Вона може запускати різні операційні системи на одному фізичному сервері. Оскільки контейнери віртуалізують ОС, вони містять лише програму з її бібліотеками та залежностями.

Оскільки екземпляр віртуальної машини віртуалізує всю ОС, він зрештою додає компоненти, не пов'язані з вашою програмою. Однак, незалежно від типу операційної системи, контейнери ізолюють лише ресурси, від яких залежить ваша програма. Тому вони мають більшу гнучкість ОС, ніж віртуальні машини.

Віртуалізація ОС дозволяє перенести вашу програму на іншу систему, не впливаючи на її розробку та розгортання. Це відповідає потребам багатохмарних та гібридних рішень, зменшуючи ризики прив'язки до постачальника та забезпечуючи плавний перехід з незначними накладними витратами.

3. Сумісність з гостями

Віртуальні машини сумісні майже з усіма операційними системами на хості, тоді як контейнери сумісні лише з версією операційної системи, подібною

до версії хоста. Хоча віртуальні машини з'явилися раніше та широко використовувалися протягом багатьох років, контейнери зараз є найкращим вибором для DevOps у багатьох відношеннях.

Нові технології та інновації зазвичай створюються для контейнерних середовищ. Крім того, доступно багато образів контейнерів з відкритим кодом, попередньо зібраних та налаштовуваних. Попередньо зібрані образи для віртуальних машин також доступні, але їх важко налаштувати та забезпечити сумісність.

4. Контейнеризація проти віртуалізації: розгортання

У цьому відношенні контейнеризація також пропонує більшу гнучкість. Існує лише кілька кроків для розгортання програми за допомогою віртуальної машини.

1. Збірка та налаштування віртуальної машини.
2. Встановлення програми на основі цільової ОС.
3. Опублікуйте додаток на платформі на ваш вибір.

Але цей процес тривалий та ресурсомісткий.

Щоб розгорнути контейнеризований застосунок, ви можете отримати образ та розгорнути його на будь-якій сумісній платформі. Це швидкий процес, який не такий ресурсоємний, як віртуальна машина.

5. Постійне зберігання

Контейнери та віртуальні машини відрізняються тим, як їм потрібна інфраструктура зберігання даних та як вона використовується. Це впливає на проектування вашої IT-інфраструктури та допомагає максимізувати цінність середовища застосунків.

Контейнери є тимчасовими, тобто вони запускаються та зупиняються автоматично. Це не стосується віртуальних машин. Однак, обом потрібен доступ до постійного сховища у виробничому середовищі. Без цього доступу контейнер завершує роботу, і дані стають недоступними.

Якщо ви використовуєте контейнеризацію корпоративного рівня, вам потрібно налаштувати постійне середовище зберігання, яке підтримує контейнери.

6. Балансування навантаження

Балансування навантаження віртуальних машин передбачає переміщення запущених віртуальних машин на інші вузли для відмовостійкого кластера. З іншого боку, контейнери можуть автоматично запускатися або зупинятися на вузлах кластера залежно від змін навантаження.

У цьому випадку змінюється лише доступність, але самі контейнери не переміщуються.

7. Нетворкінг

Контейнери використовують ізольований вигляд віртуального мережевого адаптера для легкої віртуалізації, оскільки вони спільно використовують брандмауер хоста. Віртуальні машини використовують повну віртуалізацію з VNA.

Контейнеризація проти віртуалізації: що підходить саме вам?

Як контейнери, так і віртуальні машини (ВМ) мають свої переваги. Вам слід обрати контейнеризацію, щоб максимізувати кількість програм на мінімальних серверах. Це також правильний вибір для розгортання хмарних програм, упаковки мікросервісів та переміщення масштабованих програм між IT-середовищами з однією й тією ж ОС.

Як варіант, віртуальні машини можуть виконувати більше операцій і найкраще підходять для монолітних робочих навантажень, що вимагають повної функціональності ОС. Вибирайте віртуальні машини, якщо ваш застосунок не потребує портативності, і ви хочете розміщувати застарілі застосунки, ізолювати ризиковані середовища розробки та виділити IT-ресурси, такі як сервери, сховища та мережі.

Ви також можете використовувати контейнери та віртуальні машини разом для оптимізації ємності та використання сервера. Інтеграція контейнерів у

віртуальні машини дозволяє командам DevOps підвищити ефективність фізичного сервера та зменшити кількість збоїв.

Гіпервізор працює в такий спосіб: операційна система хоста емулює апаратне забезпечення, поверх якого вже запускаються гостьові операційні системи. Це означає, що взаємозв'язок між гостьовою й хостовой операційними системами треба «залізній» парадигмі: усе, що «уміє» робити встаткування, повинне бути доступно гостьовий ОС із боку хостовой. Навпроти, контейнери – це віртуалізація на рівні операційної системи, а не встаткування, тобто кожна гостьова ОС використовує те ж саме ядро (а в деяких випадках – і інші частини ОС), що й хостова. Це дає контейнерам велику перевага: вони менше й компактніше гіпервізорних гостьових середовищ, оскільки в них з хостом набагато більше загального.

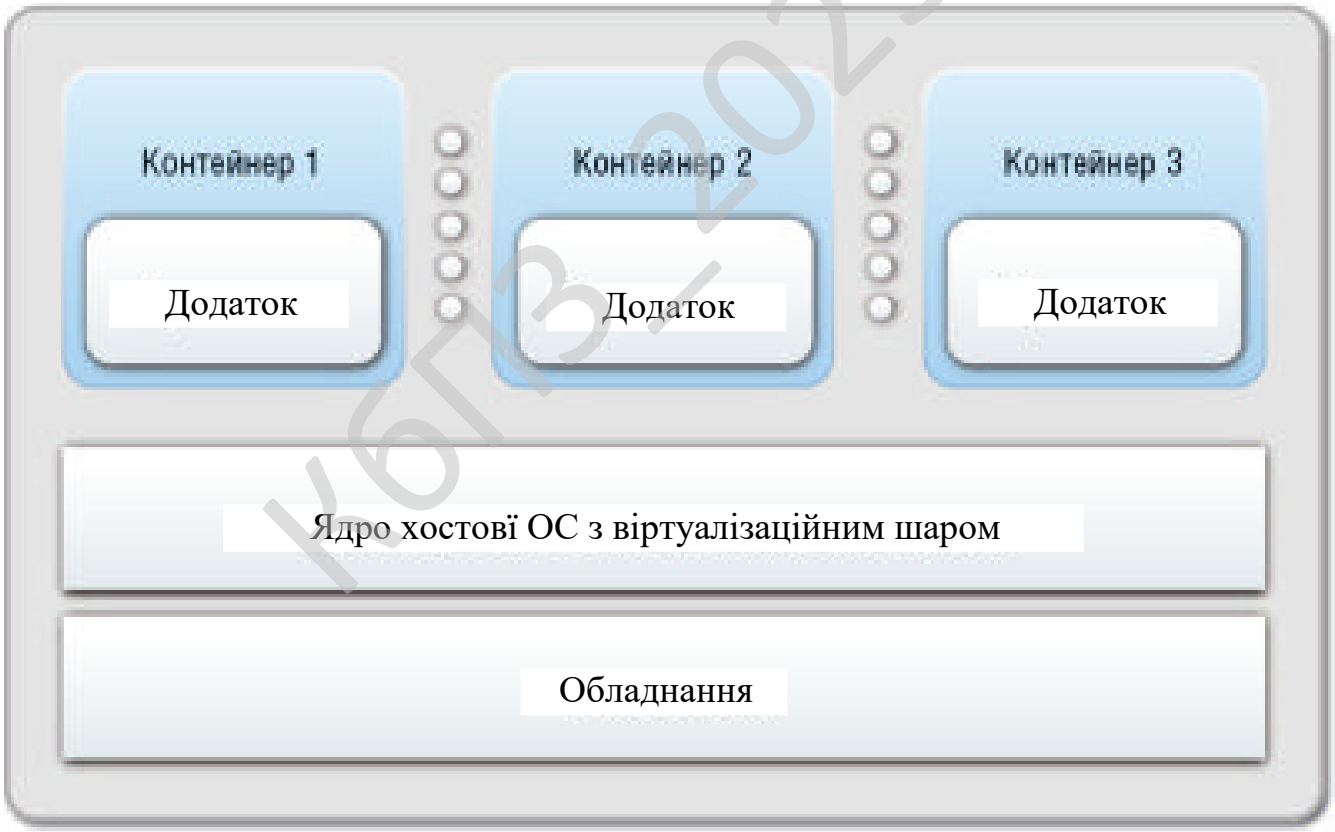

Рисунок 3.1 – Структурна схема системи

3.3 Розробка функціональної схеми

На рисунку 3.2 зображена функціональна схема системи. Нижче розглянемо її більш докладно.

Віртуалізація – це, безумовно, революційна технологія. У даній роботі застосовується ефективний підхід до віртуалізації апаратних ресурсів процесора Cell Broadband Engine, що називається контейнерної віртуалізацією (або віртуалізацією операційної системи). Обговоримо елементи, необхідні для реалізації віртуалізації процесора Cell/B.E. програмними методами.

Поняття OpenVZ і процесора Cell/B.E.

OpenVZ – це програмна реалізація підтримки контейнерної віртуалізації в Linux з відкритим вихідним кодом. OpenVZ складається із двох компонентів:

– Ядро Linux OpenVZ.

– Інструменти простору користувача OpenVZ.

Основна мета проекту OpenVZ складається в створенні ізольованих контейнерів, які працюють під керуванням віртуальних екземплярів операційної системи Linux. У всіх контейнерів є доступ до єдиного віртуалізованому ядру. Базова система, що реалізує це ядро, обслуговує так звану основну четвірку віртуалізованих пристроїв: центральний процесор, оперативну пам'ять, дискові накопичувачі й мережні пристрої. Також можна передати один із пристроїв прямо контейнеру, після чого воно стає недоступним базовій системі й всьому іншому контейнеру, за винятком контейнера, що володіє цим пристроєм (виділення пристрою в монопольне володіння).

Ви вже, імовірно, знайомі із процесором Cell/B.E., що застосовується в різних продуктах сторонніх виробників – від гібридних суперкомп'ютерів до спеціалізованих високозахищених апаратних прискорювачів і ігрової приставки Sony Playstation 3. Процесор Cell містить у собі ядро загального призначення з архітектурою Power Architecture™ і оптимізовані сопроцесорні елементи (streamlined co-processing elements, SPU), які значно прискорюють мультимедійні й векторні додатки. Процесор Cell/B.E. має продуктивність операцій із

плаваючою точкою одиничної точності в 25,6 гігафлопс на SPU, за що його часто називають суперкомп'ютером у мікросхемі.

Базова архітектура складається із процесорного елемента Power (Power Processing Element, PPE), що являє собою ядро PowerPC зі звичайною підсистемою пам'яті, і восьми синергічних процесорних елементів (Synergistic Processing Elements, SPE), з'єднаних високошвидкісним внутрішнім каналом, називаним шиною з'єднання елементів (Element Interconnect Bus, EIB). Кожний SPE складається із синергічного процесора (Synergistic Processing Unit, SPU), локальної пам'яті (LS) обсягом 256 КБ, у якій зберігаються інструкції й дані SPE, а також контролера потоків пам'яті (MFC), що управляє передачею даних по DMA між основною пам'яттю системи й LS елемента SPE. PPE служить винятково для потреб операційної системи. На сьогоднішній день Linux є єдиною операційною системою, що підтримує архітектуру Cell/B.E.

Контейнерна віртуалізація й процесор Cell/B.E

Контейнерам надається доступ до виділеним фізичних SPU, наявним у системі. SPU, доступні в контейнері, недоступні в інших контейнерах і в базовій системі на час виділення. Кожний контейнер використовує тільки ті SPU, якими він володіє. Щоб реалізувати це, необхідно виконати чотири дії:

– Віртуалізувати файлову систему SPU (spuf). spuf повинна бути доступна усередині контейнерів, але кожний контейнер повинен бачити тільки потоки SPE, створені їм самим.

– Скорегувати віртуальну файлову систему, реалізовану в ядрі Linux 2.6 (sysfs). В /sys/devices/system/spu в sysfs усередині контейнера повинні втримуватися правильні записи каталогів для кожного SPU, виділеного йому. libspe2 використовує ці записи каталогів для підрахунку SPU, доступних усередині контейнера.

– Змінити планувальник SPU. Необхідно змінити планування SPU таким чином, щоб потоки SPE, створені усередині контейнера, працювали тільки на SPU, виділених цьому контейнеру.

– Змінити інструменти OpenVZ. Інструмент vzctl з пакета інструментів OpenVZ повинен підтримувати виділення SPU контейнерам і навпаки, звільняти SPU від контейнерів. Виділення й звільнення повинні функціонувати під час роботи контейнера.

Нам потрібний механізм, що буде контролювати, якою частиною пристрою володіє контейнер або протягом якого часу весь пристрій буде доступно контейнеру. Оскільки SPU неможливо розділити на частині фізично, контейнери використовують SPU з поділом за часом доступу.

У рамках нашої реалізації повинні виконуватися наступні чотири дії:

– Віртуалізувати spufs. spufs повинна бути доступна усередині контейнерів, але кожний контейнер повинен бачити тільки потоки SPE, створені їм самим.

– Скорегувати sysfs. У папці /sys/devices/system/spu в sysfs усередині контейнера повинні втримуватися правильні записи каталогів. Кращим рішенням буде наявність контейнерів, у яких є доступ до всім SPU, установленим у системі, тому в директорії /sys/devices/system/spu будуть однакові записи як на базовій системі, так і у всіх контейнерах.

– Змінити планувальник SPU. Планувальник SPU необхідно змінити таким чином, щоб потоки SPE, створені усередині контейнера, мали доступ тільки до SPU, доступним у системі на певний момент часу. Для потоків SPE може бути розроблений алгоритм планування, схожий із дворівневим рівномірним планувальником центрального процесора, що реалізувала команда OpenVZ для звичайних процесів. Можна впровадити навіть повністю новий алгоритм планування.

– Змінити інструменти OpenVZ. Інструмент vzctl з пакета інструментів OpenVZ повинен підтримувати установку часу виконання SPU для кожного контейнера. Цей параметр повинен бути змінюємо під час роботи контейнера.

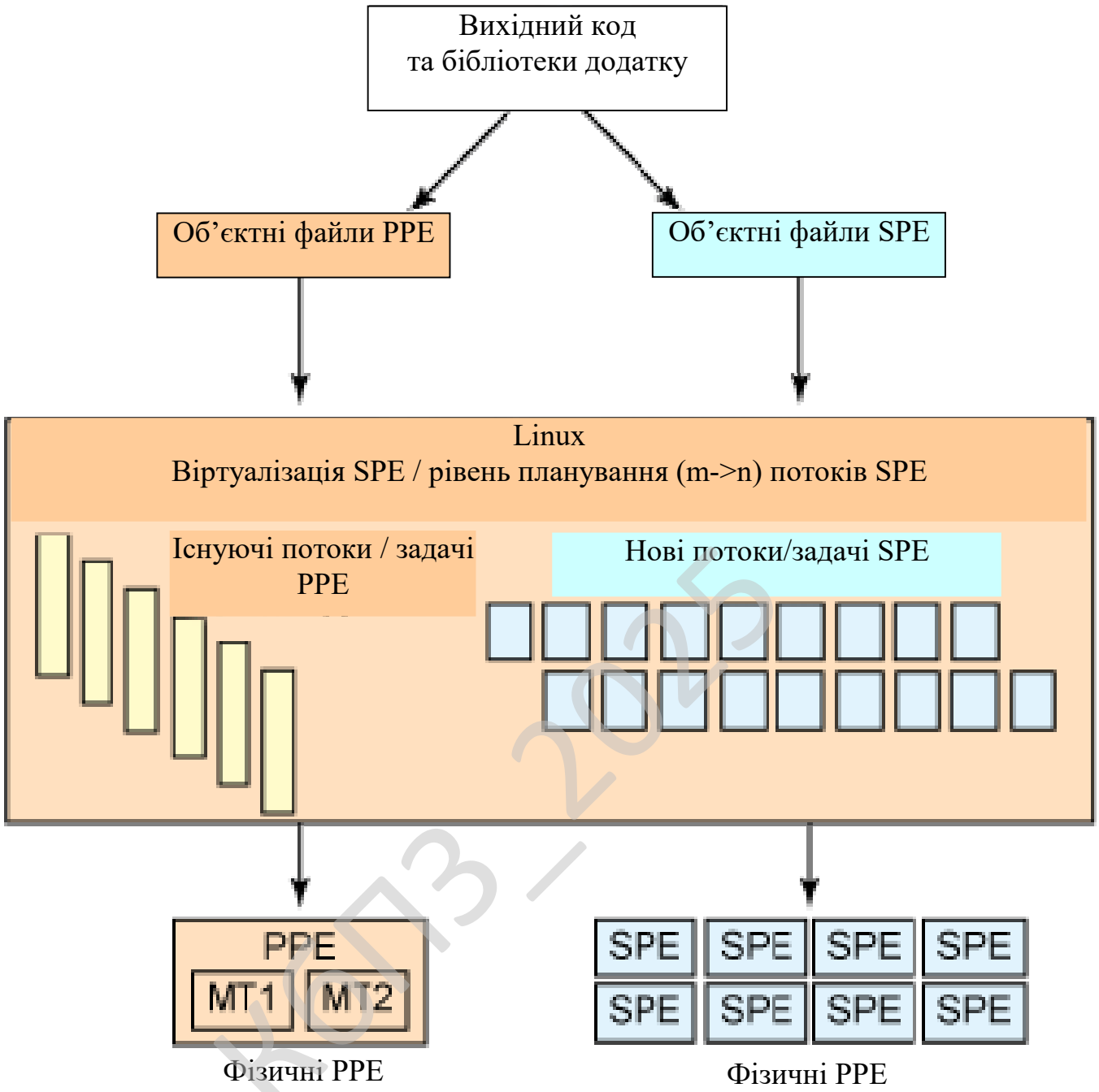

Рисунок 3.2 – Функціональна схема системи

Тепер трохи про spufs.

Використання spufs

Spufs сумісна з інший добре відомою віртуальною файловою системою (VFS) procfs. Procfs була першою абстрактною файловою системою в Linux, що

була покликана максимально спростити подання процесів. У той час як procfs представляє процеси, що працюють на центральному процесорі, spufs представляє потоки, що працюють на SPU процесора Cell/B.E.

Spufs є спеціальною файловою системою. Це віртуальна файлова система розроблена для керування SPU на процесорі Cell/B.E. Кожна директорія spufs позначає логічний контекст SPE. Такий логічний контекст SPE розглядається так само, як фізичний SPE. Властивості контексту представлені у вигляді файлів у каталозі. При доступі до контексту SPE відбувається маніпуляція або реальним SPE, або його станом, збереженим в оперативній пам'яті. Для запуску програми SPU скопіюйте виконується файл, ЩО, ELF SPU в LS SPE і виконаєте системний виклик `spu_run`.

Якщо ви хочете виконати в Linux код, спеціально призначений для Cell/B.E., наприклад, програму, що працює на SPU процесора Cell/B.E., вам доведеться використовувати spufs. Іншого способу запуску на Linux процесів, що використовують SPU процесора Cell/B.E, не існує.

Як ви можете побачити в лістингу 1, перед виконанням команди `ls` на двох потоках SPE був запущений приклад програми швидкого перетворення Фур'є з Cell SDK. Точка монтування spufs – /spu.

Лістинг 1. Виконання приклада швидкого перетворення Фур'є з Cell SDK із двома потоками SPE

```
[root@c02b 12-0 ~]# ls /spu
spethread-20914-25296904 spethread-20914-25297464

[root@c02b 12-0 ~]# ls /spu/*
/spu/ spethread-20914-25296904:
cntl decr_status event_mask fpcr ibox_info lslr mbox_info mem mss
object-id proxydma_info regs signal1_type signal2_type wbox wbox_stat
decr dma_info event_status ibox ibox_stat mbox mbox_stat mfc npc physid
psmap signal1 signal2 srr0 wbox_info

/spu/ spethread-20914-25297464:
cntl decr_status event_mask fpcr ibox_info lslr mbox_info mem mss
object-id proxydma_info regs signal1_type signal2_type wbox wbox_stat
```

```
decr dma_info event_status ibox ibox_stat mbox mbox_stat mfc npc physid
psmap signal1 signal2 srr0 wbox_info
```

Результат роботи першої команди показує, що для кожного потоку SPE визначається одна директорія (kobject), у назві якої вказується ідентифікатор процесу (PID) і ідентифікатор потоку SPE. Результат роботи другої команди показує файли, що втримуються в обох директоріях.

Файл, що виконується, SPE у вигляді об'єктного файлу SPE прив’язується до потоку SPE, що виконується на фізичному SPE. Спеціальний планувальник SPE вирішує, коли і який потік SPE буде виконуватися на фізичному SPE. Об'єктні файли PPE обробляються так само, як і в системах стандартної архітектури PowerPC®, наприклад, на Blade-серверах JS21. Об'єктні файли PPE прив’язуються до стандартних потоків Linux, за керування якими відповідає планувальник Linux, реалізований у ядрі. Тому нічого нового в додатках, що працюють на PPE, немає.

Опис SDK Cell/B.E. і libspe

На сьогоднішній день випущена версія 3.0 комплекту для розробки програмного забезпечення (SDK) для Cell/B.E. (останню версію можна одержати в розділі завантаження центра ресурсів Cell). В SDK Cell/B.E. входять всі інструменти, необхідні для розробки додатків, що використовують процесори Cell/B.E. Крім усього іншого, в SDK входять різні пакети компіляторів і компонувальників, бібліотеки, що спрощують розробку (наприклад, libspe), а також приклади програм SPU, MFC, LS і т.д., які демонструють використання можливостей архітектури Cell/B.E.

На сьогоднішній день libspe (версій libspe1 і libspe2) є єдиним користувачем spufs. Це середовище, що спрощує розробку коду для SPE архітектури Cell/B.E. Вона копіює виконуються файли, ЩО, ELF у локальну пам'ять SPE і виконує системний виклик `spu_run`, тому вам не потрібно піклуватися про внутрішні процеси запуску файлів, що виконуються, SPE.

На рисунку 6 показана ієрархія розширень, реалізованих в Linux для використання процесора Cell/B.E.

На рисунку показана бібліотека керування SPE у реальному часі (libspe), що працює в просторі користувача, що використовує файлову систему spufs, реалізовану в просторі ядра.

Опис sysfs

Споконвічно sysfs була реалізована в ядрі Linux як driverfs, її метою було надання загального огляду всіх пристроїв і драйверів, відомих ядру. Вона була розроблена як більше простій, чим procfs, спосіб доступу до пристроїв і драйверам. sysfs показує ієрархію структур даних kobject (представляючи кожну директорією) і безліч атрибутів (структури kobject), які є файлами, що звичайно містять одне значення, закодоване в текстовий рядок.

Наприклад, у лістингу 2 наведений лістинг директорії /sys/devices/system/spu на blade-сервері QS21 Cell/B.E.

Listing 2. One directory for each SPU

```
[root@c02b 12-0 ~]# ls /sys/devices/system/spu/
spu0  spu1  spu10  spu11  spu12  spu13  spu14  spu15  spu2  spu3  spu4
spu5  spu6
spu7  spu8  spu9
[root@c02b 12-0 ~]#
```

В blade-сервері QS21 Cell/B.E. використовується два процесори Cell/B.E. с вісьма SPU у кожному, таким чином, усього в сервері є 16 SPU. Як ви можете бачити, у директорії /sys/devices/system/spu утримується по одній директорії для кожного SPU. libspe2 по цих записах директорій підраховує кількість фізичних SPU, доступних у системі.

Опис ядра OpenVZ

Ядро OpenVZ – це модифіковане ядро Linux, у якому реалізована можливість створення середовищ ізольованих контейнерів операційної системи. Крім того, воно виконує керування ресурсами й копіювання пам'яті в контрольних точках для контейнерів. Всі контейнери й навіть базова система використовують те саме загальне віртуалізоване ядро.

У кожному контейнері є власна безліч ресурсів, надаваних ядром, у число яких входять:

– Файли: системні бібліотеки й додатки.

– Віртуалізовані файлові системи: procfs або sysfs.

– Користувачі й групи: у кожному контейнері є власний користувач root, так само як і інших користувачах і групи.

– Дерево процесів: з контейнера видно тільки власна безліч процесів з віртуалізованими ідентифікаторами (PID процесу init дорівнює 1).

– Мережа: віртуальні мережні пристрої із власними адресами IP і правилами фільтрації й маршрутизації.

– Пристрою: віртуалізуються деякі пристрої. При необхідності будь-якому контейнеру може бути наданий доступ до реального (не віртуалізованого) пристрою.

– Об'єкти IPC: семафори, повідомлення й загальна пам'ять.

Керування ресурсами

Керування ресурсами виконується по різних типах ресурсів:

– Дискова квота: В OpenVZ реалізується дворівневе дискове квотування, що дозволяє обмежити дисковий простір для контейнера, а також дає контейнеру можливість установлювати квоти у власному середовищі.

– Планувальник центрального процесора: Рівномірний планувальник центрального процесора також використовує дворівневий механізм. Перший рівень цього планувальника, що набудовується – рівень контейнерів, на якому ви можете визначити, скільки процесорного часу буде виділено тому або іншому контейнеру. На другому рівні планувальник працює із плануванням процесів у середовищі усередині контейнерів.

– Лічильники beancounters користувачів: набір лічильників, обмежень і гарантій для ресурсів контейнерів. Існує порядку 20 параметрів пам'яті й різних об'єктів ядра, наприклад, сегментів загальної пам'яті IPC і мережних буферів.

Копіювання пам'яті в контрольних точках

Копіювання пам'яті в контрольних точках (checkpointing) – ще одна важлива функція системи. Копіювання й відновлення пам'яті в контрольних

точках необхідно для міграції в реальному часі. Копіювання пам'яті в контрольних точках – це процес заморожування контейнера й наступного повного збереження його стану в дисковий файл. Відновлення являє собою зворотний процес. Міграція контейнера в реальному часі – це процес копіювання пам'яті контейнера в контрольній точці на одній базовій системі і її відновлення на іншій базовій системі.

Міграція в реальному часі – це одна дія, тоді як копіювання й відновлення пам'яті в контрольних точках – це дві різних дії, для виконання яких потрібне додатковий пристрій зберігання, доступне з обох апаратних вузлів.

Щоб продемонструвати реалізацію системи, у цій статті будуть освітлені наступні питання:

– Віртуалізація spufs: запобігання використання одного root-inode для всіх точок монтування.

– Коректування sysfs: запису необхідно коректувати в реальному часі.

– Модифікація планувальника SPU: додавання кроку реалізації віртуалізації.

– Модифікація інструментів OpenVZ: використання нових функцій.

Віртуалізація spufs

За замовчуванням spufs використовує той самий root-`inode` при кожному монтуванні. Наприклад, у вас може бути два середовища `chroot` (A й B), і spufs обох буде змонтована по шляху /spu. Якщо ви створите потоки SPE у середовищі A, після чого виведете вміст /spu, ви побачите тільки що створені потоки SPE. Однак у середовищі B ви побачите той же лістинг /spu, оскільки ви звертаєтеся до того же `root-inode`, що й у середовищі A.

Щоб змінити таке поводження, тобто щоб в обох середовищах можна було побачити тільки створені в них потоки SPE, необхідно змінити той факт, що spufs завжди використовує той самий root-`inode` для всіх точок монтування. При монтуванні spufs вона викликає функцію ядра, що повертає системний блок. Звичайно використовується функція `get_sb_single()`, що завжди повертає

той самий системний блок. Функція `get_sb_nodev()` завжди повертає різні системні блоки, що приводить до потрібному нам результату.

OpenVZ дозволяє змонтувати усередині контейнера тільки кілька файлових систем. Кожна система реалізує екземпляр `struct file_system_type`, що визначає аспекти, пов'язані з файлами. OpenVZ розширює цю структуру (поряд з іншими), доповнюючи її компонентом, що визначає, чи може бути змонтована файлова система. Таким чином, компоненту.`fs_flags` повинно бути встановлене значення `FS_VIRTUALIZED`.

У процесу, створеного усередині контейнера, є два PID. У кожного контейнера є власна безліч ресурсів, надаваних ядром. У дерева процесів була одна безліч ресурсів, а контейнер міг бачити тільки власну безліч процесів з віртуалізованими PID. Усередині контейнера є віртуалізований PID, а в базовій системі той же процес ідентифікується іншим PID, що називається глобальним PID. Як уже згадувалося, назва директорій spufs задається у вигляді `spethread-<PID>-< thread-ID>`, де `<PID>` – ідентифікатор процесу потоку SPE, а `< thread-ID>>` відповідний ідентифікатор потоку. Оскільки в середовищі контейнера використовуються віртуалізовані PID, лістинг spufs усередині контейнера повинен показувати віртуалізовані, а не глобальні PID. На щастя, такий алгоритм уже реалізований в OpenVZ.

Зміните файл arch/powerpc/platforms/cell/spufs/inode.c у ядрі Linux®.

Коректування sysfs

Sysfs уже віртуалізована, тому її можна використовувати усередині контейнерів. Вона є не копією sysfs, видимої в базовій системі, а її частиною. Ціль полягає в тому, щоб виділяти SPU контейнерам і звільняти їх у реальному часі. Це означає, що запису sysfs також необхідно адаптувати в реальному часі. Перед цим створіть директорію, у якій перераховані SPU.

Це директорія /sys/devices/system/spu. За замовчуванням в sysfs у середовищі контейнера немає директорій /sys/devices/system/spu, /sys/devices/system і /sys/devices. Ці три записи необхідно створити під час

ініціалізації (запуску) контейнера. Піддиректорії /sys/devices/system/spu можуть мати вигляд від spu0 до spu<N>, де <N> – кількість доступних SPU у системі мінус 1. Директорії необхідно створювати, коли SPU призначається контейнеру, і видаляти, коли вони відкріплюються від контейнера.

Кожної директорії в лістингу sysfs повинен відповідати екземпляр kobject у просторі ядра. kobject – це структура, що визначає назву директорії і її батьківський kobject (інакше кажучи, що відповідає батьківську директорію). Наприклад, в kobject, що відображається як директорія /sys/devices/system/spu/spu3, є два параметри, які потрібно змінити:

– Назва spu3.

– Батько (покажчик на kobject, що представляє /sys/devices/system/spu).

Після реєстрації kobject в sysfs за допомогою виклику `subsystem_register()`, він стає видимим із простору користувача. Зворотна дія складається у видаленні директорії з sysfs. Для цього можна викликати функцію `subsystem_unregister()`, указавши kobject, якому необхідно видалити.

В kobject з назвою spu3 є покажчик на батьківський kobject з назвою spu.

Основні зміни необхідно зробити у файлі ядра kernel/ve/vecalls.c. Це файл OpenVZ, у якому реалізована більшість функцій, які викликаються під час ініціалізації й настроювання параметрів контейнерів.

Модифікація планувальника SPU

Кожний фізичний SPU системи представлений екземпляром структури `spu` у просторі ядра. У цій структурі зберігається кілька параметрів, у число яких входять:

– Ідентифікатор (номер) SPU.

– До якого вузла Cell/B.E. він належить.

– Покажчик на LS SPU.

Нова змінна зберігає власника SPU у вигляді ідентифікатора контейнера. Планувальник SPU реалізує функцію `spu_alloc()`, що шукає вільний SPU, на

якому буде виконуватися потік SPE. Таким чином, він шукає список доступних у системі SPU (на які в цей момент не виконується потоків SPE).

Звичайно береться перший SPU у списку й на ньому запускається потік SPE. Щоб реалізувати віртуалізацію, функція повинна перевірити, чи збігається ідентифікатор контейнера вільного SPU з ідентифікатором контейнера потоку SPE, що передбачається на ньому виконати.

Якщо ця додаткова перевірка дає негативний результат, функція перевіряє наступний елемент у списку вільних SPU. Якщо вільних SPU для контейнера, що запускає потік SPE, ні, планувальник SPU поводиться так само, як якби список був порожній, і чекає звільнення SPU.

Функція `spu_alloc()` реалізована у вихідному файлі ядра Linux arch/powerpc/platforms/cell/spu_base.c.

Модифікація інструментів OpenVZ

Більша частина потрібної функціональності вже реалізована, але для того, щоб використовувати нові функції, необхідно модифікувати інструменти OpenVZ. Інструмент vzctl управляє виділенням SPU у реальному часі. Це основний інструмент для настроювання параметрів в OpenVZ. Новий параметр для установки кількості SPU, виділених контейнеру, – `---spus <nr_spus>`.

Значення `<nr_spus>` представляє кількість SPU, виділених контейнеру. Це абсолютне значення, тому якщо контейнеру з вісьма SPU привласнюється значення 6, то від процесора відкріплюються 2 SPU, а не додаються 6 SPU (8 – 6 = 2).

Наприклад, от результат роботи команди, коли контейнер з ID, рівним 101, одержує вісім SPU:

```
[root@c02b 12-0 ~]# vzctl set 101 ---ispus 8
Setting SPUs: 8
Configure meminfo: 1024000
WARNING: Settings were not saved and will be reset to original
 values on next start (use ---isave flag)
[root@c02b 12-0 ~]#
```

Для реалізації такого алгоритму інструмент vzctl повинен перейти бар'єр простору користувача й виконати ряд операцій керування в просторі ядра. Інструмент повинен знайти SPU, які не використовуються іншими контейнерами. Інструмент vzctl виконує пошук за списком доступних SPU і перевіряє ідентифікатор контейнера в структурі нового spu (описана в розділі модифікації планувальника SPU). Якщо це значення дорівнює 0, SPU може бути призначений розглянутому контейнеру. Значення 0 використовується тому, що значення ідентифікатора контейнера повинне бути більше 0, тобто значення 0 указує на те, що SPU не призначено жодному з контейнерів. Якщо функція не може знайти потрібного для виконання запиту кількості вільних SPU, процедура завершується й не призначає контейнеру жодного SPU. Якщо кількість SPU, уже призначених контейнеру, більше запитаного кількості SPU, різниця відкріплюється.

Щоб перебороти бар'єр між простором користувача й простором ядра, можна використовувати різні моделі реалізації. Найбільш простий спосіб складається в реалізації нового системного виклику, що накладає параметри `<containerID>` і `<nr_spus>` на параметри системного виклику.

Функції, які виконують настроювання параметрів SPU контейнера, повинні реалізовуватися в тій частині ядра, що може бути реалізована в модулі ядра. Це представляє більшу проблему. Якщо модуль ядра не завантажений, функція обробки системного виклику в просторі ядра нічого не зробить. Однак якщо модуль завантажений, він викличе функцію, реалізовану в модулі. Це нетривіальне завдання, оскільки таблиця системних викликів (де зберігаються покажчики функції на функції обробки системних викликів) є частиною статичного складання ядра.

Модуль не є частиною цієї статичної функції, і тому статична убудована функція обробки системного виклику не може викликати функцію, що є частиною модуля. Рішення складається в реалізації оболонки функції, що копіює покажчик на функції модуля в змінну убудованій статичній функції обробки системного виклику таким чином, щоб убудований статичний оброблювач

системного виклику міг викликати функцію модуля. Ця оболонка функції викликається під час ініціалізації й очищення модуля.

Ви бачите, як покажчик функції, реалізованої в модулі, копіюється в убудовану статичну функцію простору ядра. Пунктирні стрілки показують, як додатка простору користувача викликають функцію модуля, передаючи убудовану статичну функцію оброблювача системного виклику.

Необхідно змінити вихідний код ядра в наступних файлах:

– include/ asm-powerpc/systbl.h.

– include/ asm-powerpc/unistd.h.

– include/linux/syscalls.h.

– kernel/sys.c.

– kernel/sys_ni.c.

– kernel/ve/vecalls.c.

Крім того, обновляються файли вихідного коду vzctl OpenVZ:

– include/res.h.

– include/vzctl_param.h.

– include/vzsyscalls.h.

– src/lib/config.c.

– src/lib/res.c.

– src/vzctl.c.

У систему складання OpenVZ додається два нових файли:

– include/spu.h.

– src/lib/spu.c.

Розглянувши усі блоки функціональної схеми перейдемо до розгляду діаграми взаємодії процесів, які відбуваються у системі.

3.4 Розробка діаграми процесів

Діаграма процесів розробленої системи зображена на рисунку 3.3. При детальному її розгляді можна побачити як саме проходе взаємодія у розробленій

системі. Використовується модель проектування, графічне представлення «потоків» даних в інформаційній системі.

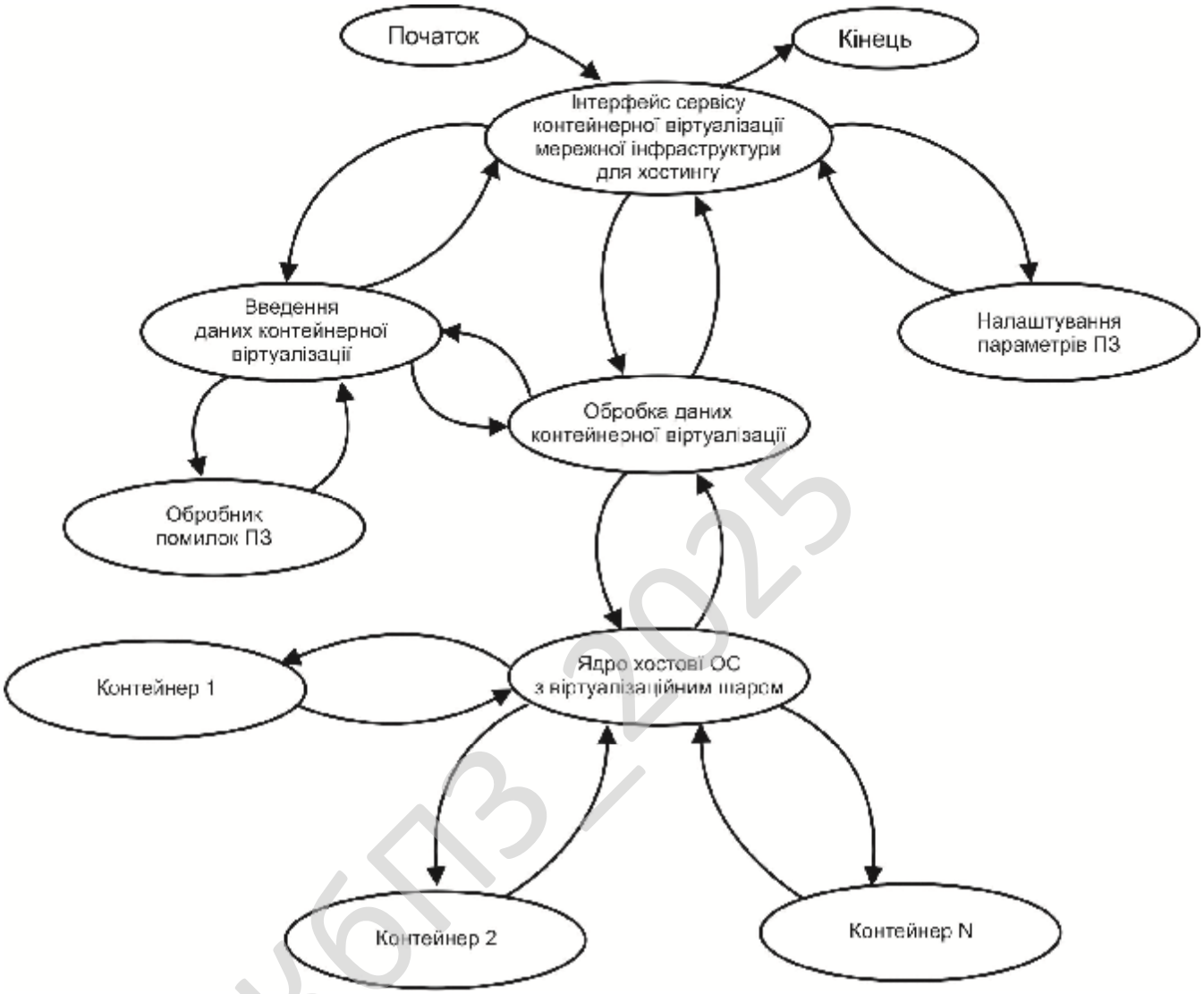

Рисунок 3.3 – Діаграма взаємодії процесів

Діаграма взаємодії процесів використовується для візуалізації процесів обробки даних (структурне проектування). Для розробника вважається звичним спочатку креслити діаграму взаємодії процесів даних рівня контексту, завдяки чому буде показано взаємодію системи. Ця діаграма в подальшому підлягає уточненню шляхом деталізації процесів та потоків даних з метою показати систему що розробляється.

Діаграми потоків даних містять чотири типи елементів:

– Процеси які являють собою трансформацію даних в рамках описуваної системи.

– Сховища даних (репозиторії).

– Зовнішні по відношенню до системи сутності.

– Потоки даних між елементами трьох попередніх типів.

Таким чином, розглянувши опис системи, структурну, функціональну схеми системи, та діаграму взаємодії процесів перейдемо до опису блок-схем основної програми, та підпрограм, які використовуються, для реалізації системи.

4 РЕАЛІЗАЦІЯ РОБОТИ. РОЗРАХУНКИ І ЕКСПЕРИМЕНТАЛЬНІ ДАНІ, ЩО ПІДТВЕРДЖУЮТЬ ПРАВИЛЬНІСТЬ ПРОЕКТНИХ РІШЕНЬ

4.1 Блок-схеми та опис алгоритмів функціонування системи

Блок-схеми є першоджерелами стратегії розвитку ПЗ. Тому від точності і детальної блок-схеми залежить результат всієї програми.

При виборі початкової точки відліку при побудові схем було враховано, що виходячи з вибору мови програмування і інших технічних засобів, програма буде об’єктно-орієнтована що вимагає оптимізації програми високого рівня, також те, що при розробці програми слід надати особливу увагу модулю сервісу контейнерної віртуалізації мережної інфраструктури для хостингу.

Під час роботи над магістерською роботою було створено блок-схеми. Перед їх розглядом необхідно провести роз’яснення який саме тип блок-схем використовується.

Блок-схема це представлення задачі для її аналізу або розв'язування за допомогою спеціальних символів (геометричних образів), які позначають такі елементи, як операції, потік, дані тощо. Блок вхідних та вихідних даних прийнято позначати паралелограмом, блок обчислень (обробки) даних – прямокутником, блок прийняття рішень – ромбом, еліпсом – початок та кінець алгоритму.

У інформаційних технологіях функціональна схема складається з функціональних блоків, які являють собою конструктивно відособлені частини (елементи або пристрої) автоматичних систем, які виконують певні функції. Функціональні блоки на схемі позначають прямокутниками, всередині яких надписують їх найменування відповідно до функцій, що виконуються. Зв'язки між функціональними блоками (внутрішні впливи) позначаються лініями зі стрілками, які вказують напрям впливів.

Функціональні схеми можуть виконуватися в укрупненому і розгорненому вигляді. У першому випадку на схемі зображають найважливіші блоки системи і зв'язки між ними.

У другому варіанті схема відображається більш детально, що полегшує її читання та ілюструє принцип роботи.

Основні елементи схем алгоритму це термінатор, процес, рішення, зумовлений процес (підпрограма), дані та з'єднувач.

Термінатор це елемент відображає вхід із зовнішнього середовища або вихід з неї (найчастіше застосування – початок і кінець програми). Всередині фігури записується відповідна дія.

Процес це виконання однієї або кількох операцій, обробка даних будь-якого виду (зміна значення даних, форми подання, розташування). Всередині фігури записують безпосередньо самі операції.

Рішення це показує рішення або функцію перемикального типу з одним входом і двома або більше альтернативними виходами, з яких тільки один може бути обраний після обчислення умов, визначених всередині цього елементу. Вхід в елемент позначається лінією, що входить зазвичай у верхню вершину елементу. Якщо виходів два чи три то зазвичай кожен вихід позначається лінією, що виходить з решти вершин (бічних і нижній). Якщо виходів більше трьох, то їх слід показувати однією лінією, що виходить з вершини (частіше нижній) елемента, яка потім розгалужується. Відповідні результати обчислень можуть записуватися поруч з лініями, що відображають ці шляхи.

Зумовлений процес (підпрограма) це символ відображає виконання процесу, що складається з однієї або кількох операцій, що визначені в іншому місці програми (у підпрограмі, модулі). Всередині символу записується назва процесу і передані в нього дані.

Дані це перетворення у форму, придатну для обробки (введення) або відображення результатів обробки (виведення). Цей символ не визначає носія даних (для вказівки типу носія даних використовуються специфічні символи).

З'єднувач це символ відображає вихід в частину схеми і вхід з іншої частини цієї схеми. Використовується для обриву лінії та продовження її в іншому місці (приклад: поділ блок-схеми, що не поміщається на листі). Відповідні сполучні символи повинні мати одне (при тому унікальне) позначення.

На рисунку 4.1 зображена основна блок-схема програми, на рисунку 4.2 зображено роботу підпрограми.

З яких видно що робота основної програми складається з початкових етапів ініціалізації ПЗ, перевірки наявності ресурсів системи, блоку початку основного циклу з чеканням запиту від користувача в якому відбувається виклик підпрограми та останньої стадії – перевірка поточного стану з завершенням роботи розробленого ПЗ. При роботі підпрограми виконується основний функціонал системи з циклічними послідовностями, перевіркою поточного стану та поверненням в основну програму прапорів стану виконання.

Було використано підходи з використанням UML, це уніфікована мова моделювання, використовується у парадигмі об'єктно-орієнтованого програмування. Є невід'ємною частиною уніфікованого процесу розробки програмного забезпечення. UML є мовою широкого профілю, це відкритий стандарт, що використовує графічні позначення для створення абстрактної моделі системи, називаної UML-моделлю.

UML був створений для визначення, візуалізації, проектування й документування в основному програмних систем. UML не є мовою програмування, але в засобах виконання UML-моделей як інтерпретованого коду можлива кодогенерація.

UML може бути застосовано на всіх етапах життєвого циклу аналізу бізнес-систем і розробки прикладних програм. Різні види діаграм які підтримуються UML, і найбагатший набір можливостей представлення певних аспектів системи робить UML універсальним засобом опису як програмних, так і ділових систем.

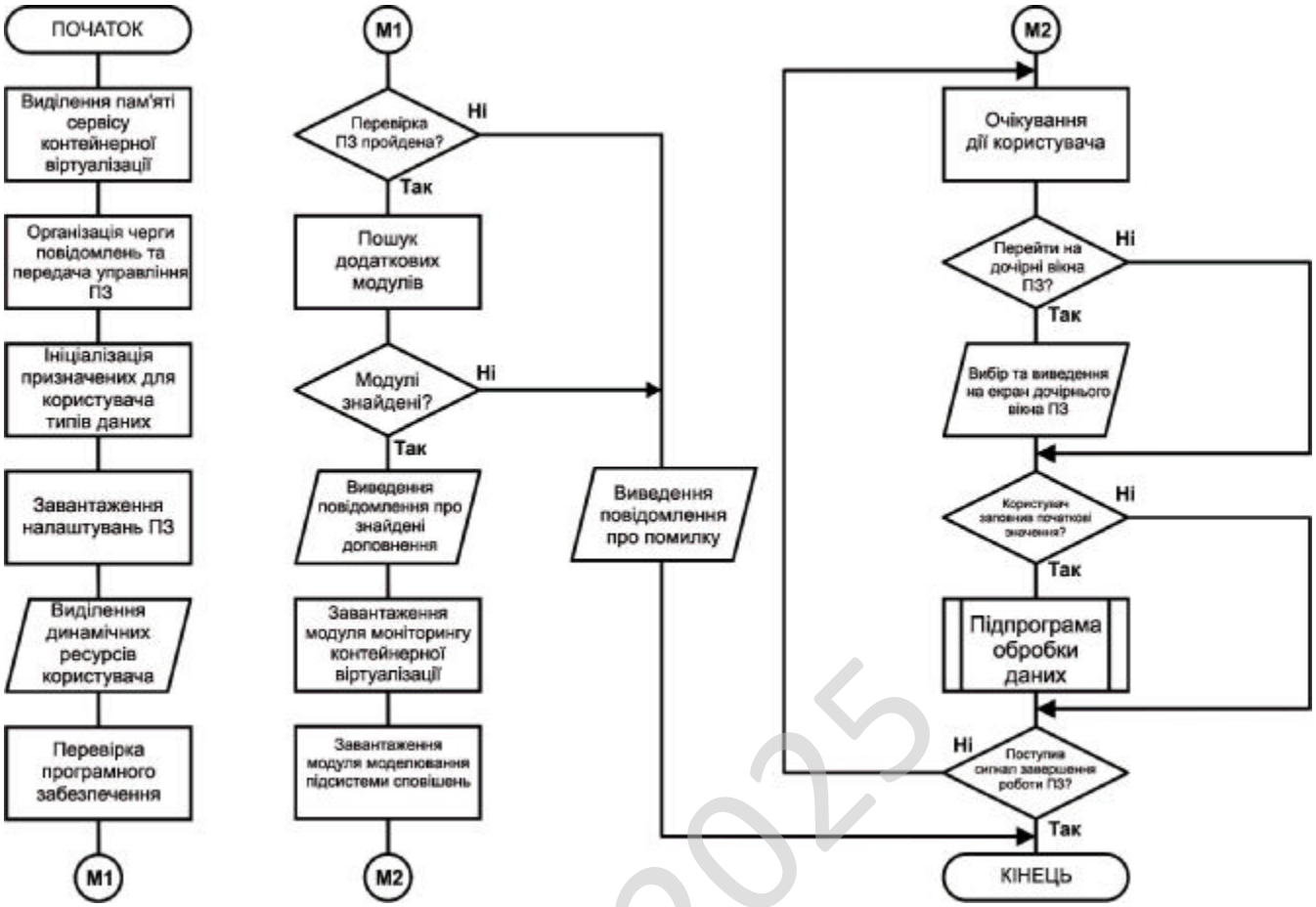

Рисунок 4.1 – Блок-схема основної програми

Діаграми дають можливість представити систему (як ділову, так і програмну) у такому вигляді, щоб її можна було легко перевести в програмний код. Основною причиною використання мови UML є спілкування розробників між собою.

Крім того, UML спеціально створювалася для оптимізації процесу розробки програмних систем, що дозволяє збільшити ефективність їх реалізації у кілька разів і помітно поліпшити якість кінцевого продукту.

UML прекрасно зарекомендувала себе в багатьох успішних програмних проектах. Засоби автоматичної генерації кодів дозволяють перетворювати моделі мовою UML у вихідний код об’єктно-орієнтованих мов програмування, що ще більш прискорює процес розробки. Практично усі CASE-засоби (програми автоматизації процесу аналізу і проектування) мають підтримку UML.

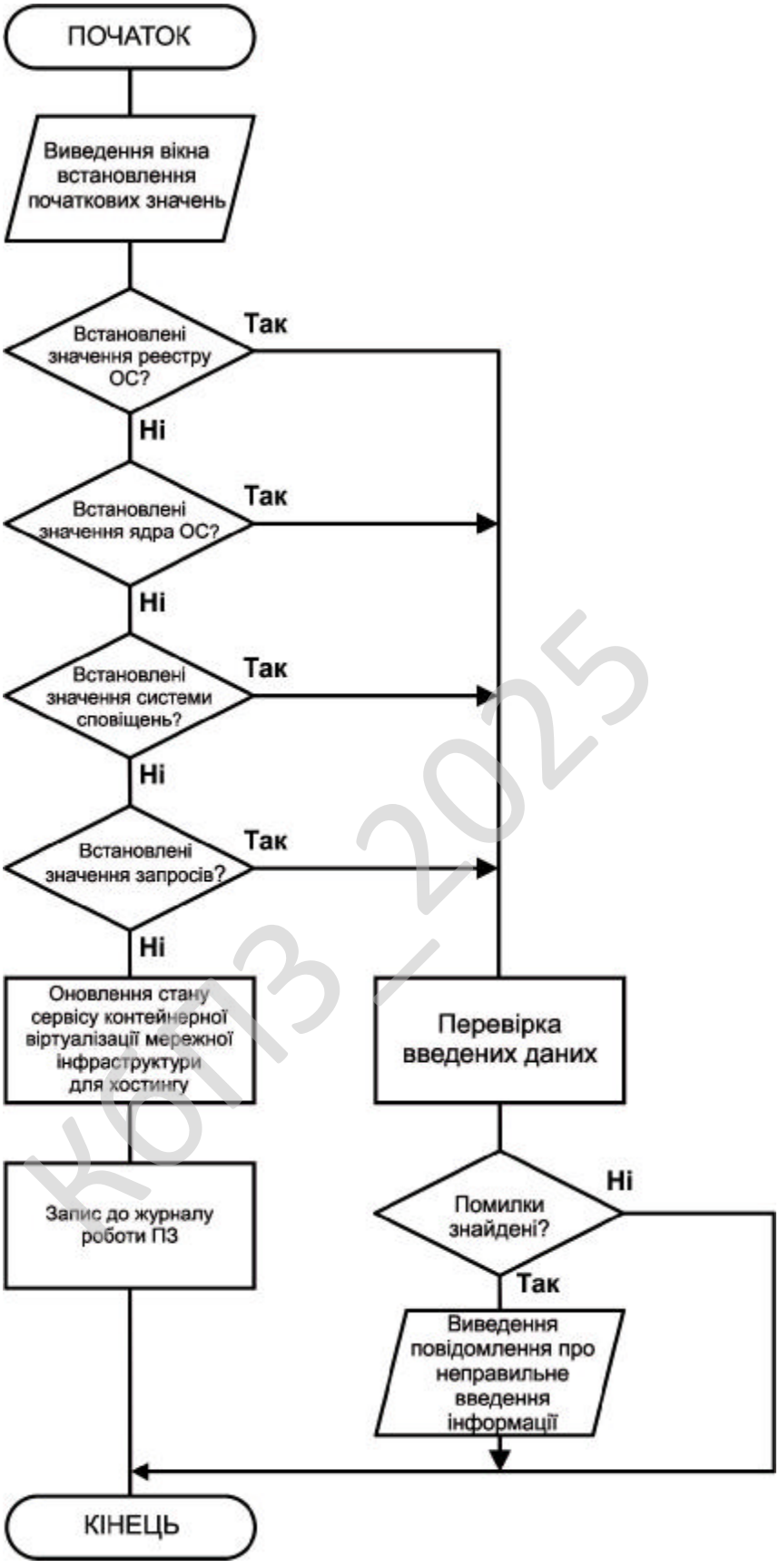

Рисунок 4.2 – Блок-схема роботи підпрограми

Моделі розроблені в UML, дозволяють значно спростити процес кодування і направити зусилля програмістів безпосередньо на реалізацію системи.

Діаграми підвищують супроводжуваність проекту і полегшують розробку документації.

UML необхідний:

– Керівникам проектів, які керують розподілом завдань і контролем за проектом.

– Проектувальникам інформаційних систем які розробляють технічні завдання для програмістів.

– Бізнес-аналітикам, які досліджують реальну систему і здійснюють інжиніринг і реінжиніринг бізнесу компанії.

– Програмістам які реалізовують модулі інформаційної системи.

При модифікації системи об'єктний підхід дозволяє легко включати в систему нові об'єкти і виключати застарілі без істотної зміни її життєздатності. Використання побудованої моделі при модифікаціях системи дає можливість усунути небажані наслідки змін, оскільки вони не ламають структури системи, а тільки змінюють поведінку об'єктів.

Також при розробці магістерської роботи було використано наступні підходи UML: Діаграма класів; Діаграма компонент; Діаграма об'єктів; Діаграма розгортання.

Діаграма класів це статичне представлення структури моделі. Відображає статичні (декларативні) елементи, такі як: класи, типи даних, їх зміст та відношення.

Діаграма класів, також, може містити позначення для пакетів та може містити позначення для вкладених пакетів. Також, діаграма класів може містити позначення деяких елементів поведінки, однак їх динаміка розкривається в інших типах діаграм.

Діаграма класів (class diagram) служить для представлення статичної структури моделі системи в термінології класів об'єктно-орієнтованого

програмування. На цій діаграмі показують класи, інтерфейси, об'єкти й кооперації, а також їхні відносини.

В UML існують наступні типи зв'язків які використовуються у діаграмі класів: Асоціації; Агрегація; Композиція.

Асоціації це якщо між двома класами визначена асоціація, то можна переміщатися від об'єктів одного класу до об'єктів іншого. Цілком припустимі випадки, коли обидва кінці асоціації відносяться до одного і того ж класу. Це означає, що з об'єктом деякого класу дозволено зв'язати інші об'єкти з того ж класу. Асоціація, що зв'язує два класи, називається бінарної. Можна, хоча це рідко буває необхідним, створювати асоціації, що зв'язують відразу кілька класів. Графічно асоціація зображується у вигляді лінії, що з'єднує клас сам з собою або з іншими класами.

Асоціації може бути присвоєно ім'я, яке описує природу відносини. Зазвичай ім'я асоціації не вказується, якщо тільки ви не хочете явно задати для неї рольові імена або у вашій моделі настільки багато асоціацій, що виникає необхідність посилатися на них і відрізняти один від одного. Ім'я буде особливо корисним, якщо між одними і тими ж класами існує кілька різних асоціацій.

Клас, що бере участь в асоціації, грає в ній деяку роль. По суті, це "обличчя", яким клас, що знаходиться на одній стороні асоціації, звернений до класу з іншого її боку. Можна явно позначити роль, яку клас грає в асоціації.

Часто при моделюванні буває важливо вказати, скільки об'єктів може бути пов'язано допомогою одного примірника асоціації. Це число називається кратністю (Multiplicity) ролі асоціації та записується або як вираз, значенням якого є діапазон значень, або в явному вигляді.

Вказуючи кратність на одному кінці асоціації, ви тим самим говорите, що на цьому кінці саме стільки об'єктів повинно відповідати кожному об'єкту на протилежному кінці. Кратність можна задати рівною одиниці (1), можна вказати діапазон: "нуль або одиниця" (0..1), "багато" (0.. *), "одиниця або більше" (1.. *). Дозволяється також вказувати певне число (наприклад, 3). За допомогою списку

можна задати і більш складні кратності, наприклад 0.. 1, 3..4, 6.. *, що означає "будь-яке число об'єктів, крім 2 і 5".

Агрегація це проста асоціація між двома класами відображає структурний відношення між рівноправними сутностями, коли обидва класу знаходяться на одному концептуальному рівні і ні один не є більш важливим, ніж інший. Але іноді доводиться моделювати відношення типу «частина/ціле», в якому один з класів має більш високий ранг (ціле) і складається з декількох менших за рангом (частин).

Ставлення такого типу називають агрегацією; воно зараховане до відносин типу «має» (з урахуванням того, що об'єкт-ціле має кілька об'єктів-частин). Агрегація є окремим випадком асоціації і зображується у вигляді простої асоціації з незафарбованим ромбом з боку «цілого». Графічно агрегація представляється порожнім ромбом на блоці класу, і лінією, яка від цього ромба до міститься класу.

Композиція це більш суворий варіант агрегації. Відома також як агрегація за значенням.

Композиція має жорстку залежність часу існування екземплярів класу контейнера та примірників містяться класів. Якщо контейнер буде знищений, то весь його вміст буде також знищено. Графічно представляється як і агрегація, але з зафарбовані ромбиком.

Діаграма компонент в UML це діаграма, на якій відображаються компоненти, залежності та зв'язки між ними.

Діаграма компонент відображає залежності між компонентами програмного забезпечення, включаючи компоненти вихідних кодів, бінарні компоненти, та компоненти, що можуть виконуватись.

Модуль програмного забезпечення може бути представлено в якості компоненти. Деякі компоненти існують під час компіляції, деякі – під час компонування, а деякі під час роботи програми.

Діаграма компонент відображає лише структурні характеристики, для відображення окремих екземплярів компонент слід використовувати діаграму розгортування.

Компоненти об'єднуються разом використовуючи структурні зв'язки (assembly connector) щоб об'єднати інтерфейси двох компонент. Це ілюструє зв'язок типу «клієнт-сервер».

Структурна взаємодія – «зв'язок двох компонент, який передбачає, що один з них надає послуги, потрібні іншому компоненту».

При використанні діаграми компонент щоб показати внутрішню структуру компонента, клієнтські та серверні інтерфейси можуть утворювати пряме з'єднання з внутрішніми. Таке з'єднання називається з'єднанням делегації.

Діаграма об'єктів в UML це діаграма, що відображає об'єкти та їх зв'язки в певний момент часу.

Діаграма об'єктів може розглядатись як окремий випадок діаграми класів, на якій можуть бути представлені як класи, так і екземпляри (об'єкти) класів. Схожою за змістом є діаграма взаємодії (collaboration diagram).

Діаграми об'єктів не мають власної нотації. Оскільки діаграми класів можуть відображати об'єкти, то діаграма класів, на якій відображено лише об'єкти, та не відображено класи, може вважатись діаграмою об'єктів.

Діаграма об'єктів відображає об'єкти та зв'язки в певний момент роботи програми. Об'єкти можуть містити інформацію про власні значення а не про описання. Для відображення загальних шаблонів об'єктів та зв'язків, що можуть багаторазово створюватись під час роботи програми, слід використовувати діаграму взаємодії, яка може відображати характеристики об'єктів та зв'язків. Екземпляр діаграми взаємодії створює діаграму об'єктів.

Діаграма об'єктів не відображає еволюцію системи під час роботи. Натомість, слід використовувати діаграми взаємодії з повідомленнями, або діаграми послідовності.

Діаграма розгортання (deployment diagram) це діаграма в UML, на якій

відображаються обчислювальні вузли під час роботи програми, компоненти, та об'єкти, що виконуються на цих вузлах. Компоненти відповідають представленню робочих екземплярів одиниць коду.

Компоненти, що не мають представлення під час роботи програми на таких діаграмах не відображаються; натомість, їх можна відобразити на діаграмах компонент.

Діаграма розгортання відображає робочі екземпляри компонент, а діаграма компонент, натомість, відображає зв'язки між типами компонент.

У роботі використовувались дві мови програмування Builder C++ та Delphi XE8. Що дало змогу підвищити надійність та стабільність реалізації сервісу контейнерної віртуалізації мережної інфраструктури для хостингу. Одне з першечергових завдань віртуалізації мережної інфраструктури – забезпечення системи надійним алгоритмом шифрування внутрішніх модифікованих даних.

Методи захисту даних на персональних комп’ютерах дуже різноманітні як по кінцевій цілі, так і по технічному втіленню.

Нижче наведено розроблений вихідний код, методу шифрування внутрішніх модифікованих даних.

Опис компонентів системи

Система контейнерної віртуалізації мережевої інфраструктури для хостингу у магістерській роботі виконує роль навчального програмного комплексу.

Система моделює роботу платформи контейнеризації яка розміщує вебсайти і супутні сервіси на наборі хостів. Вона не прив’язується до конкретної реалізації типу Docker або Kubernetes і працює як логічний емулятор для демонстрації принципів розподілу навантаження, побудови віртуальних сегментів мережі, керування ресурсами та політиками безпеки.

Уся логіка системи реалізується на мові Python. Структура коду відповідає модульному підходу. У межах одного вихідного файлу створюються класи які відображають основні сутності інфраструктури. Додатково реалізується

консольний інтерфейс адміністратора який демонструє типові сценарії використання платформи для хостингу.

Система працює на основі внутрішньої моделі інфраструктури. Ця модель містить вузли розміщення контейнерів, специфікації сервісів, екземпляри контейнерів, логічні мережеві сегменти та правила міжмережевої взаємодії. Для збереження стану використовується файл у форматі JSON тому адміністратор може зберігати конфігурацію та відновлювати її при наступному запуску.

Модель вузлів інфраструктури

У коді створюється клас HostNode. Він моделює фізичний або віртуальний сервер на якому розміщуються контейнери.

Для кожного вузла зберігаються ідентифікатор, кількість обчислювальних ядер, обсяг оперативної пам’яті, місткість дискової підсистеми та пропускна здатність мережевого інтерфейсу.

Клас також зберігає поточне використання цих ресурсів та перелік контейнерів які працюють на вузлі.

Клас HostNode має метод який перевіряє можливість розміщення нового контейнера з певним профілем ресурсів. Якщо доступні ресурси вузла достатні для покриття вимог контейнера тоді вузол вважається придатним для розміщення. Окремий метод виконує фактичне розміщення контейнера. Він оновлює лічильники використання ресурсів та повертає об’єкт екземпляра контейнера.

Таким чином у пояснювальній записці можна наочно показати як Python код моделює рішення задачі про розподіл контейнерів між серверами з урахуванням обмежень за процесором, пам’яттю, диском та мережею.

Специфікація сервісів та екземпляри контейнерів

Для опису параметрів сервісу використовується клас ContainerSpec. Він зберігає назву сервісу, опис образу, вимоги до ресурсів, список відкритих портів та тип сервісу.

Клас також може містити кількість реплік за замовчуванням. Ця інформація потрібна оркестратору для розгортання сервісів.

Клас ContainerInstance представляє один запущений контейнер. Він посилається на об’єкт специфікації, містить ідентифікатор екземпляра, інформацію про вузол на якому контейнер працює, поточний стан та прості показники навантаження.

Система не взаємодіє з реальними контейнерами тому показники навантаження моделюються у програмі.

На основі цих двох класів система демонструє два рівні моделі. Перший рівень описує те як саме повинен працювати сервіс. Другий рівень описує конкретні запущені екземпляри на конкретних вузлах.

Модель мережевої інфраструктури

Для опису віртуальної мережевої інфраструктури у програмі використовується клас NetworkSegment. Він зберігає назву сегменту, адресу підмережі, ідентифікатор VLAN та список прив’язаних контейнерів.

Таке представлення дозволяє моделювати поділ хостинг оточення на фронтенд, бекенд та службові мережі. Кожен контейнер при створенні прив’язується до певного сегменту.

Для моделювання політик безпеки створюється клас FirewallRule. Він зберігає назви сегментів джерела і призначення, список дозволених портів та дію правила. У поточній реалізації використовується проста модель де дією є дозвіл або заборона.

Оркестратор при спробі з’єднання між контейнерами може звернутися до переліку правил та визначити чи дозволяється взаємодія.

Така модель дає можливість у пояснювальній записці показати як контейнерна віртуалізація поєднується з логічною сегментацією мережі та політиками доступу.

Стан інфраструктури та сховище конфігурацій

Для зручності керування усі об’єкти інфраструктури зберігаються у класі InfrastructureState.

Він містить словники вузлів, специфікацій сервісів, екземплярів контейнерів та сегментів мережі. Клас надає методи додавання та пошуку об’єктів.

Окремі методи виконують перетворення стану у структури придатні для серіалізації у JSON і навпаки.

На підставі цих методів створюється клас StateRepository який читає файли конфігурацій та записує у них поточний стан системи.

Таким чином студент демонструє у магістерській роботі вміння зберігати конфігурації інфраструктури у вигляді машинно опрацьовуваних моделей.

Планувальник розміщення контейнерів

Логіку вибору вузла для розміщення контейнера реалізує клас Scheduler. Він працює за жадібним алгоритмом. Планувальник перебирає усі доступні вузли та знаходить ті що здатні розмістити контейнер.

Для кожного такого вузла обчислюється коефіцієнт завантаження. Далі вибирається вузол з найменшим показником завантаження.

Такий підхід демонструє просту реалізацію балансування навантаження без складних алгоритмів оптимізації. У коментарях до коду пояснюється як обчислюється завантаження за процесором та пам’яттю.

Оркестратор контейнерної платформи

Основну логіку керування хостинг середовищем реалізує клас Orchestrator. Він приймає об’єкт стану інфраструктури, планувальник та сховище конфігурацій.

Оркестратор надає методи реєстрації нових вузлів, створення нових сегментів, додавання опису сервісу та розгортання сервісу.

Метод розгортання приймає назву сервісу та бажану кількість реплік. Далі він знаходить відповідну специфікацію, викликає планувальник для кожної репліки, створює екземпляри контейнерів, прив’язує їх до вузлів і сегментів та зберігає у стані інфраструктури. Після виконання операції оркестратор може зберегти оновлений стан у файл.

Окремі методи оркестратора зупиняють сервіс, видаляють контейнери, змінюють кількість реплік та обчислюють сумарне використання ресурсів. Завдяки цьому адмін може побачити навантаження на інфраструктуру від конкретного хостинг проекту.

Підсистема моніторингу та журналювання

Щоб надати студенту приклад реалізації елементарного моніторингу, у програмі створюється клас Monitor. Він генерує умовні показники для кожного контейнера.

Для моделювання використовується модуль random. Монітор оновлює поле з навантаженням у об’єкті контейнера та формує події журналу. Журнал зберігає список подій з відміткою часу, назвою контейнера та коротким описом дії.

Клас Monitor працює у синхронному режимі і викликається з консолі адміністратора. Така реалізація спрощує код та при цьому дозволяє показати концепцію моніторингу стану контейнерів у пояснювальній записці.

Консольний інтерфейс адміністратора

Окремий клас ConsoleUI надає текстовий інтерфейс для взаємодії з системою. Він приймає оркестратор та монітор. У головному циклі реалізується просте меню.

Адміністратор може додати вузол, додати сегмент, зареєструвати сервіс, розгорнути сервіс, відобразити стан, виконати моніторинг, зберегти конфігурацію та завантажити її.

Для стану інфраструктури створюється функція яка виводить таблицю з вузлами, їх ресурсами та списком контейнерів. Також виводиться перелік сервісів та кількість екземплярів кожного. У пояснювальній записці демонструється типовий сценарій використання такого меню для розміщення хостинг сервісу.

```
import json
import time
import uuid
import random
from typing import Dict, List, Optional
```

```
        # Клас HostNode моделює вузол інфраструктури для розміщення контейнерів
        class HostNode:
            def __init__(
                self,
                node_id: str,
                cpu_cores: int,
                ram_mb: int,
                storage_gb: int,
                network_mbps: int,
            ) -> None:
                self.node_id = node_id
                self.cpu_cores = cpu_cores
                self.ram_mb = ram_mb
                self.storage_gb = storage_gb
                self.network_mbps = network_mbps
                self.used_cpu_cores = 0.0
                self.used_ram_mb = 0
                self.used_storage_gb = 0
                self.used_network_mbps = 0
                self.containers: List["ContainerInstance"] = []
            def can_host(self, spec: "ContainerSpec") -> bool:
                cpu_ok = self.used_cpu_cores + spec.cpu_cores <= self.cpu_cores
                ram_ok = self.used_ram_mb + spec.ram_mb <= self.ram_mb
                storage_ok = self.used_storage_gb + spec.storage_gb <=
self.storage_gb
                network_ok = self.used_network_mbps + spec.network_mbps <=
self.network_mbps
                return cpu_ok and ram_ok and storage_ok and network_ok

            def allocate(self, spec: "ContainerSpec") -> "ContainerInstance":
                self.used_cpu_cores += spec.cpu_cores
                self.used_ram_mb += spec.ram_mb
                self.used_storage_gb += spec.storage_gb
                self.used_network_mbps += spec.network_mbps
                instance_id = f"{spec.name}-{uuid.uuid4().hex[:8]}"
                instance = ContainerInstance(
                    instance_id=instance_id,
                    spec=spec,
                    host_id=self.node_id,
                )
                self.containers.append(instance)
```

```
        return instance

    def release(self, instance: "ContainerInstance") -> None:
        if instance in self.containers:
            self.used_cpu_cores -= instance.spec.cpu_cores
            self.used_ram_mb -= instance.spec.ram_mb
            self.used_storage_gb -= instance.spec.storage_gb
            self.used_network_mbps -= instance.spec.network_mbps
            self.containers.remove(instance)
    def utilization(self) -> Dict[str, float]:
        cpu_util = 0.0
        ram_util = 0.0
        if self.cpu_cores > 0:
            cpu_util = self.used_cpu_cores / float(self.cpu_cores)
        if self.ram_mb > 0:
            ram_util = self.used_ram_mb / float(self.ram_mb)
        return {
            "cpu": cpu_util,
            "ram": ram_util,
        }

    def to_dict(self) -> Dict:
        return {
            "node_id": self.node_id,
            "cpu_cores": self.cpu_cores,
            "ram_mb": self.ram_mb,
            "storage_gb": self.storage_gb,
            "network_mbps": self.network_mbps,
        }

# Клас ContainerSpec описує вимоги сервісу який розгортається у контейнерах
class ContainerSpec:
    def __init__(
        self,
        name: str,
        image: str,
        cpu_cores: float,
        ram_mb: int,
        storage_gb: int,
        network_mbps: int,
        ports: Optional[List[int]] = None,
        default_replicas: int = 1,
```

```
            segment_name: str = "frontend",
        ) -> None:
            self.name = name
            self.image = image
            self.cpu_cores = cpu_cores
            self.ram_mb = ram_mb
            self.storage_gb = storage_gb
            self.network_mbps = network_mbps
            self.ports = ports or []
            self.default_replicas = default_replicas
            self.segment_name = segment_name

        def to_dict(self) -> Dict:
            return {
                "name": self.name,
                "image": self.image,
                "cpu_cores": self.cpu_cores,
                "ram_mb": self.ram_mb,
                "storage_gb": self.storage_gb,
                "network_mbps": self.network_mbps,
                "ports": list(self.ports),
                "default_replicas": self.default_replicas,
                "segment_name": self.segment_name,
            }

    # Клас ContainerInstance моделює окремий екземпляр контейнера
    class ContainerInstance:
        def __init__(
            self,
            instance_id: str,
            spec: ContainerSpec,
            host_id: str,
        ) -> None:
            self.instance_id = instance_id
            self.spec = spec
            self.host_id = host_id
            self.status = "running"
            self.cpu_load_percent = 0.0
            self.ram_load_percent = 0.0
        def to_dict(self) -> Dict:
            return {
                "instance_id": self.instance_id,
```

```
                "spec_name": self.spec.name,
                "host_id": self.host_id,
                "status": self.status,
                "cpu_load_percent": self.cpu_load_percent,
                "ram_load_percent": self.ram_load_percent,
            }

    # Клас NetworkSegment описує віртуальний мережевий сегмент
    class NetworkSegment:
        def __init__(
            self,
            name: str,
            cidr: str,
            vlan_id: int,
        ) -> None:
            self.name = name
            self.cidr = cidr
            self.vlan_id = vlan_id
            self.containers: List[str] = []

        def to_dict(self) -> Dict:
            return {
                "name": self.name,
                "cidr": self.cidr,
                "vlan_id": self.vlan_id,
                "containers": list(self.containers),
            }
    # Клас FirewallRule моделює просте правило міжмережевої взаємодії між
сегментами
    class FirewallRule:
        def __init__(
            self,
            src_segment: str,
            dst_segment: str,
            allowed_ports: Optional[List[int]] = None,
            action: str = "allow",
        ) -> None:
            self.src_segment = src_segment
            self.dst_segment = dst_segment
            self.allowed_ports = allowed_ports or []
            self.action = action
```

4.2 Захист розробленого програмного забезпечення

Для захисту розробленого програмного забезпечення запропоновано використовувати алгоритм SEED – у криптографії симетричний блоковий криптоалгоритм на основі Мережі Фейстеля, розроблений Корейським агентством інформаційної безпеки (Korean Information Security Agency, KISA) в 1998 році. В алгоритмі використовується 128-бітний блок і ключ довжиною 128 біт.

Алгоритм одержав широке поширення й використовується фінансовими й банківськими структурами, виробничими підприємствами й бюджетними установами Південної Кореї, оскільки 40-бітний SSL не забезпечує на даний момент мінімально необхідного рівня безпеки. Агентством по захисту інформації специфіковане використання шифру SEED у протоколах TLS і S/MIME.

У той же час, алгоритм SEED не реалізований у більшості сучасних браузерів і інтернет-додатків, що утрудняє його використання в даній сфері поза межами Південної Кореї.

SEED являє собою мережу Фейстеля з 16 раундами, 128-бітовими блоками й 128-бітовим ключем. Алгоритм використовує дві 8×8 таблиці підстановки, які, як такі з Safer, виведені з дискретного зведення в ступінь (у цьому випадку, x^{247} і x^{251} – плюс деякі «несумісні операції»).

Це є деякою подібністю с MISTY1 у рекурсивності його структури: 128-бітовий повний шифр – мережа Фейстеля з F-функцією, що впливає на 64-бітові половини, у той час як сама F-функція – Мережа Фейстеля, складена з G-функції, що впливає на 32-розрядні половини.

Однак рекурсія не простягнеться далі, тому що G-функція – не Мережа Фейстеля. В G-функції 32-розрядне слово розглядають як чотири 8-бітових байта, кожний з яких проходить через одну або іншу таблицю підстановки, потім поєднується в помірковано комплексному наборі булевих функцій таким чином, що кожний біт виводу залежить від 3 з 4 вхідних байтів.

SEED має складний ключовий розклад, генеруючи тридцять два 32-розрядних додаткових символу, використовуючи G-функції на серіях обертань вихідного неопрацьованого ключа, комбінованого зі спеціальними раундовими константами (як в TEA) від «Золотого співвідношення» (англ. Golden ratio).

Згідно з дослідженнями KISA, алгоритм SEED «надійно протистоїть відомим атакам».

5 МЕТОДИКА ВПРОВАДЖЕННЯ СИСТЕМИ В ПРОМИСЛОВУ ЕКСПЛУАТАЦІЮ

На рисунку 5.1 зображено інтерфейс програмного забезпечення, розробленого у результаті виконання магістерської роботи. Розроблене програмне забезпечення сервісу контейнерної віртуалізації мережної інфраструктури для хостингу складається з наступних функціональних блоків:

– Навігаційне меню: довідкова інформація контейнеру; Контейнери; Виконання дії; Довідка.

– Блоку налаштування параметрів сервісу контейнерної віртуалізації мережної інфраструктури для хостингу.

– Вікно виведення результату роботи системи.

– Навігаційного меню яке викликається натисканням правої клавіші маніпулятора миші та функціональні кнопки ПЗ.

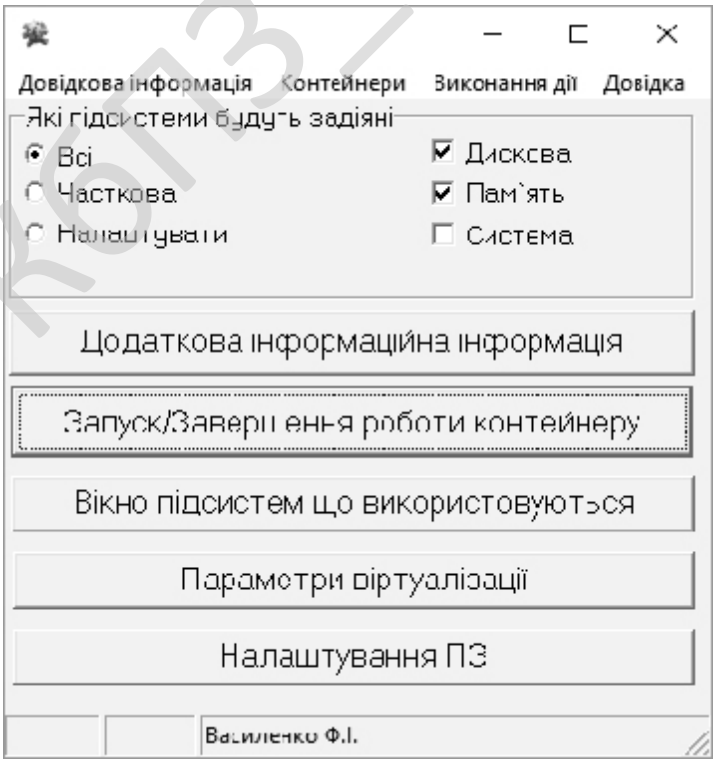

Рисунок 5.1 – Головне вікно розробленого ПЗ

Для перегляду короткої довідки про програму слід натиснути на основному вікні кнопку авторського права, після чого на екрані з’явиться вікно показане на рисунку 5.2.

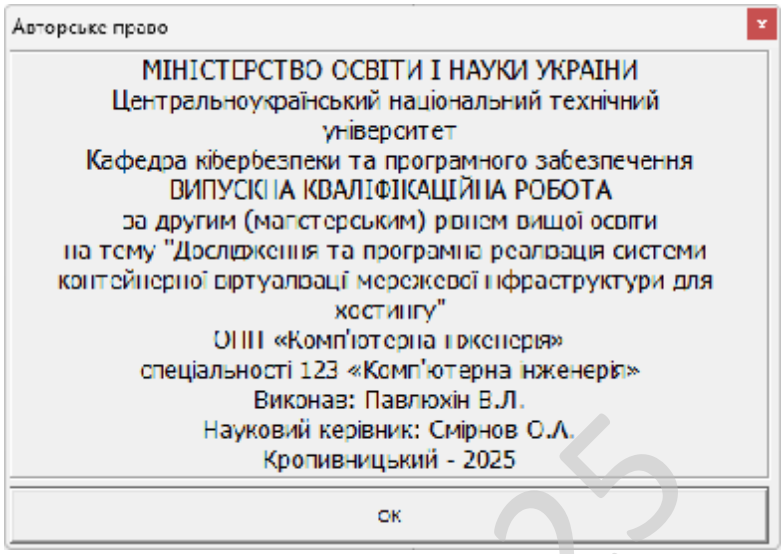

Рисунок 5.2 – Вікно розробника ПЗ

Під час роботи над програмою було проведено тестування програмного забезпечення, тобто технічне дослідження, призначене для виявлення інформації про якість продукту відносно контексту, в якому воно має використовуватись.

Тестування включає як процес пошуку помилок або інших дефектів, так і випробування програмних складових з метою їх оцінки.

Проводилась оцінка:

– відповідності поставленим вимогам;

– правильна відповідь для усіх можливих вхідних даних;

– виконання функцій за прийнятний час;

– практичність;

– сумісність з ОС та стороннім ПЗ.

Оскільки число можливих тестів для програмних компонент практично нескінченне, тому стратегія тестування полягала в тому, щоб провести всі можливі тести з урахуванням наявного часу та ресурсів.

Як результат ПЗ тестувалось стандартним виконанням програми з метою виявлення помилок або інших дефектів.

Проводилось тестування форматом білої скриньки та чорної скриньки.

Тестування форматом білої скриньки засноване на аналізі керуючої структури програми. Програма вважається повністю перевіреною, якщо проведено вичерпне тестування маршрутів (шляхів) її графа управління.

У цьому випадку формуються тестові варіанти, в яких:

– Гарантується перевірка всіх незалежних маршрутів програми.

– Знаходяться гілки True, False для всіх логічних рішень.

– Виконуються всі цикли (у межах їхніх кордонів та діапазонів).

– Аналізується правильність внутрішніх структур даних.

Недоліки тестування "білої скриньки":

– Кількість незалежних маршрутів може бути дуже велика.

– Повне тестування маршрутів не гарантує відповідності програми вихідним вимогам до неї.

– У програмі можуть бути пропущені деякі маршрути.

– Не можна виявити помилки, поява яких залежить від даних.

Переваги тестування "білої скриньки" пов'язані з тим, що принцип «білої скриньки» дозволяє врахувати особливості програмних помилок:

– Кількість помилок мінімально в «центрі» і максимально на «периферії» програми.

– Попередні припущення про ймовірність потоку керування або даних у програмі часто бувають некоректними. У результаті типовим може стати маршрут, модель обчислень за яким опрацьована слабо.

– При записі алгоритму програмного забезпечення у вигляді тексту на мові програмування можливе внесення типових помилок трансляції (синтаксичних та семантичних).

– Деякі результати в програмі залежать не від вихідних даних, а від внутрішніх станів програми.

Проводилось тестування чорної скриньки.

Основне місце програми тестів «чорної скриньки» – інтерфейс ПЗ. Відомі: функції програми. Досліджується: робота кожної функції на всій області визначення.

Ці тести демонструють:

– Як виконуються функції програми.

– Як приймаються вихідні дані.

– Як виробляються результати.

– Як зберігається цілісність зовнішньої інформації.

При тестуванні «чорної скриньки» розглядаються системні характеристики програм, ігнорується їхня внутрішня логічна структура. Вичерпне тестування, як правило, неможливе.

Наприклад, якщо в програмі 10 вхідних величин і кожна приймає по 10 значень, то кількість тестових варіантів становитиме 10^10. Тестування «чорної скриньки» не реагує на багато особливостей програмних помилок.

Тестування «чорної скриньки» (функціональне тестування) дозволяє отримати комбінації вхідних даних, які забезпечують повну перевірку всіх функціональних вимог до програми.

Програмний виріб тут розглядається як «чорна скринька», чию поведінку можна визначити тільки дослідженням його входів та відповідних виходів. При такому підході бажано мати:

– Набір, утворений такими вхідними даними, які призводять до аномалій у поведінці програми (назвемо його ІТс).

– Набір, утворений такими вхідними даними, які демонструють дефекти програми (назвемо його ОТ).

Будь-який спосіб тестування «чорної скриньки» повинен:

– Виявити такі вхідні дані, які з високою ймовірністю належать набору ІТс;

– Сформулювати такі очікувані результати, які з високою імовірністю є елементами набору ОТ.

Принцип «чорної скриньки» не альтернативний принципу «білої скриньки». Скоріше це доповнює підхід, який виявляє інший клас помилок.

Тестування «чорної скриньки» забезпечує пошук наступних категорій помилок:

– Некоректних чи відсутніх функцій;

– Помилок інтерфейсу;

– Помилок у зовнішніх структурах даних або в доступі до зовнішньої бази даних;

– Помилок характеристик (необхідна ємність пам'яті і т.д.);

– Помилок ініціалізації та завершення.

Обрано умови розповсюдження – Shareware.

Під умовно-безплатним програмним забезпеченням можна розуміти спосіб або метод розповсюдження комерційного ПЗ на ринку (тобто на шляху до кінцевого користувача), при якому випробувачеві пропонується обмежена за можливостями (не повнофункціональна або демонстраційна версія), терміном дії (тріал версія) або версія з вбудованим набридливим нагадуванням про необхідність оплати використання програми.

В угоді про використання (ліцензії для кінцевого користувача, EULA) також може бути обумовлена заборона на комерційне або професійне (не тестове) її використання.

Основний принцип умовно-безплатного ПЗ – «спробуй, перш ніж купити» (try before you buy). ПЗ що поширюється як умовно-безплатний, надається користувачам безоплатно.

Звичайно користувач платить тільки за час завантаження файлів через Інтернет або за носій (CD диск, флешку, ключ).

Протягом певного терміну, що становить зазвичай тридцять днів, він може користуватися програмою, тестувати її, освоювати її можливості.

6 НАУКОВА НОВИЗНА

У випускній кваліфікаційній роботі за другим (магістерським) рівнем вищої освіти розроблено програмне забезпечення, яке призначено для системи контейнерної віртуалізації мережевої інфраструктури для хостингу.

Метою розробки є дослідження та програмна реалізація системи контейнерної віртуалізації мережевої інфраструктури для хостингу.

Об’єктом дослідження є процес контейнерної віртуалізації мережевої інфраструктури для хостингу.

Предметом дослідження є методи контейнерної віртуалізації мережевої інфраструктури для хостингу.

Методи дослідження базуються на методах побудови комп’ютерних мереж, методах математичної статистики, методах розробки програмного забезпечення.

Наукова новизна отриманих результатів. У процесі рішення завдань, обумовлених цілями дослідження, отримані наступні результати:

– Удосконалено метод контейнерної віртуалізації мережевої інфраструктури для хостингу.

– Розроблено вітчизняний продукт контейнерної віртуалізації мережевої інфраструктури для хостингу, який має більш широкі можливості, на відміну від існуючих аналогів.

7 МАРКЕТИНГОВЕ ТА ЕКОНОМІЧНЕ ОБГРУНТУВАННЯ ІТ-ПРОЄКТУ

7.1 Визначення цільової аудиторії кінцевого готового продукту

Результати дослідження та програмної реалізації контейнерної віртуалізації мережевої інфраструктури для хостингу можуть бути цікавими для керівників хостингових компаній, оскільки ця технологія допомагає оптимізувати витрати на сервери та збільшити ефективність роботи. Вони можуть зацікавитися результатами, оскільки контейнеризація дозволяє значно знизити витрати на фізичне обладнання та зменшити складність управління інфраструктурою.

ІТ-фахівці, які займаються адмініструванням мережевої інфраструктури, також знайдуть цінність у результатах дослідження, оскільки контейнерна віртуалізація дозволяє значно полегшити процеси масштабування, впровадження оновлень та автоматизації роботи серверів. Крім того, консультанти та системні інтегратори, які займаються оптимізацією процесів у сфері хостингу, можуть використовувати ці результати для розробки рекомендацій для клієнтів, допомагаючи їм оптимізувати витрати та покращити продуктивність їх інфраструктури. Оскільки віртуалізація через контейнери є актуальною і для великих підприємств, менеджери з розвитку продукту та бізнес-аналітики також можуть бути зацікавлені, оскільки використання цієї технології дозволяє забезпечити гнучкість і швидкість у роботі з хмарними інфраструктурами.

7.2 Оцінка привабливості шляхом застосування методів експертних оцінок

Для оцінки привабливості програмної реалізації системи контейнерної віртуалізації мережевої інфраструктури можна використати метод експертних

оцінок. Одним із найбільш поширених підходів є застосування методу парних порівнянь. У цьому випадку група експертів оцінює різні аспекти системи, наприклад, ефективність використання ресурсів, легкість масштабування, зниження витрат на обслуговування та простота інтеграції з існуючими рішеннями. Кожен експерт порівнює два аспекти, оцінюючи, який з них є більш важливим або корисним для реалізації системи. Згодом результати оцінок усіх експертів комбінуються для визначення загальної привабливості системи. Наприклад, якщо експерти оцінюють масштабованість системи вище, ніж витрати на обслуговування, це вказує на те, що система може бути більш вигідною для великих компаній, які потребують високої гнучкості. Такі оцінки допомагають прийняти обґрунтовані рішення при виборі між різними технологічними рішеннями для хостингу.

7.3 Вибір методу оцінки вартості ПЗ

Для оцінки вартості програмної реалізації системи контейнерної віртуалізації мережевої інфраструктури доцільно використовувати метод ціноутворення на основі вартості життєвого циклу (LCC, Life Cycle Costing). Цей метод дозволяє оцінити загальні витрати на систему з урахуванням не тільки початкових витрат на впровадження, але й поточних витрат на обслуговування, оновлення, а також витрат на навчання персоналу та підтримку системи.

Врахування всіх витрат на протязі життєвого циклу дозволяє точно оцінити повну вартість володіння (TCO) і прийняти обґрунтоване рішення щодо інвестицій. Оскільки контейнеризація має низькі витрати на обладнання і дозволяє ефективно використовувати наявні ресурси, застосування цього методу дозволяє не тільки зрозуміти потенційні витрати на запуск, але й отримати точну картину витрат на обслуговування та оновлення системи на роки вперед.

7.4 Розрахунок економічної ефективності від впровадження реалізованого ПЗ як фактору його привабливості

Хостинг-провайдер або корпоративний дата-центр використовує традиційну модель розміщення сервісів: кожен сервіс працює на окремому фізичному сервері або у важковагомих віртуальних машинах. Це призводить до: низького середнього рівня завантаження ресурсів (CPU/RAM); перевитрат на апаратне забезпечення, енергоспоживання та охолодження; ускладнення масштабування та розгортання нових сервісів; збільшення часу простою під час оновлень і міграцій; високих трудових витрат на адміністрування розподіленої інфраструктури.

Впровадження системи контейнерної віртуалізації мережевої інфраструктури для хостингу (оркестрація контейнерів, автоматизований деплой, мережеві політики, сервіс-меш тощо) дозволяє: консолідувати сервіси на меншій кількості фізичних серверів; зменшити витрати на обладнання та енергоспоживання; скоротити час розгортання й оновлення сервісів; підвищити відмовостійкість та зменшити кількість критичних простоїв; оптимізувати роботу IT-персоналу за рахунок автоматизації. Вхідні дані зафіксовано в таблиці 7.1.

Розрахунок економічного ефекту демонструє наступне: економія на апаратному забезпеченні – 1 500 000 грн, економія на енергоспоживанні та охолодженні – 500 000 грн, зменшення втрат від простоїв – 1 120 000 грн, економія трудових витрат на адміністрування, загальна щорічна економія – 3 525 000 грн/рік, чистий економічний ефект у перший рік – 725 000 грн, термін окупності ≈ 0,79 року (близько 9–10 місяців), рентабельність інвестицій (ROI) – ≈ 126 %.

Таблиця 7.1 – Вихідні дані для розрахунку

Показник	До впровадження	Після впровадження	Економічний ефект
Кількість фізичних серверів, задіяних під мережеві сервіси	40	24	–16 серверів
Середній рівень завантаження CPU	25%	60%	+35 п.п.
Щорічні витрати на апаратне забезпечення (амортизація, сервісні контракти, оновлення), грн	4 000 000	2 500 000	–1 500 000
Щорічні витрати на енергоспоживання та охолодження, грн	1 200 000	700 000	–500 000
Кількість критичних простоїв мережевих сервісів на рік	10	4	–60%
Середня тривалість одного простою, год	2	1	–1 год
Вартість простою (1 год), грн	70 000	70 000	—
Витрати часу адміністраторів на підтримку інфраструктури, людино-год/рік	2 000	1 100	–900
Вартість 1 людино-години адміністратора, грн	450	450	—
Початкові інвестиції у впровадження системи контейнерної віртуалізації (розробка ПЗ, ліцензії, інтеграція), грн	—	—	2 800 000

Додаткові нефінансові переваги: підвищення надійності мережевої інфраструктури – контейнери ізольовані, оновлення та перезапуск окремих сервісів не «падають» усією системою, швидке розгортання нових сервісів – час деплою скорочується з годин до хвилин, що зменшує time-to-market нових хостинг-продуктів, гнучке масштабування – можна динамічно збільшувати/зменшувати кількість контейнерів під навантаження, не закуповуючи додаткові фізичні сервери, автоматизація та стандартизація – використання інструментів оркестрації (наприклад, Kubernetes, Docker Swarm) знижує залежність від людського фактору та помилок конфігурації, покращена спостережуваність (observability) – централізований моніторинг, логування й трасування мережевих сервісів дозволяють швидше виявляти та усувати проблеми, привабливість для клієнтів хостингу – можливість надання контейнерних рішень (PaaS, managed containers), гнучкі тарифні плани, SLA з меншим рівнем простоїв.

Впровадження системи контейнерної віртуалізації мережевої інфраструктури для хостингу забезпечує щорічну економію понад 3,5 млн грн, окупність проєкту – менш ніж за рік, а рентабельність інвестицій перевищує 120 %. Окрім суттєвої фінансової вигоди, система підвищує надійність, масштабованість і гнучкість хостинг-платформи, що робить реалізоване ПЗ конкурентоспроможним та привабливим рішенням для практичного впровадження.

7.5 Пропозиція алгоритму просування проєкту розробки ПЗ

Алгоритм просування проєкту програмної реалізації системи контейнерної віртуалізації мережевої інфраструктури має починатися з аналізу цільового ринку та визначення потенційних клієнтів. На першому етапі важливо зрозуміти потреби ринку, зокрема компаній, що використовують хмарні технології або мають складні інфраструктури. Наступним кроком є створення ціннісної

пропозиції, яка повинна акцентувати переваги контейнерної віртуалізації – економія витрат, зручність масштабування та простота інтеграції. Далі необхідно підготувати маркетингові матеріали, такі як презентації, відео-огляди, технічні описання, а також провести публікації в спеціалізованих ЗМІ. На етапі просування важливо не забути про підтримку клієнтів: надавати безкоштовні консультації, демонстрації продукту, організовувати онлайн-зустрічі з технічними фахівцями. В кінці варто запустити рекламні кампанії в Google Ads, соцмережах та інших онлайн-ресурсах, орієнтуючись на спеціалізовані платформи для IT-рішень.

7.6 Оптимізація каналів збуту та шляхів реалізації ПЗ

Для оптимізації каналів збуту можна створити партнерські мережі з іншими постачальниками IT-рішень, які пропонують послуги хостингу або інфраструктурні послуги, щоб розширити охоплення потенційних клієнтів. Також варто розглянути можливість постійного контент-маркетингу, публікуючи в блозі компанії кейс-стаді, технічні статті, а також вебінари та онлайн-курси, що дозволяють демонструвати ефективність контейнерної віртуалізації та її переваги для бізнесу.

Для каналів збуту можна використати цифрові платформи, зокрема маркетплейси для IT-рішень, де можна продати рішення як готовий продукт. Крім того, для великих компаній може бути корисною послуга індивідуальних консультацій та кастомізації системи, що дозволить залучити більше клієнтів із різних галузей.

7.7 Визначення ключових факторів успіху конкретного проєкту

Ключовими факторами успіху проєкту є гнучкість і масштабованість системи, оскільки ці характеристики дозволяють компаніям адаптувати рішення

під різні потреби та швидко реагувати на зміну вимог. Важливим є також якість обслуговування та підтримки, оскільки клієнти повинні бути впевнені, що система працює стабільно, а будь-які технічні питання вирішуються оперативно. Захист даних і безпека є критично важливими факторами, оскільки кожен хостинговий сервіс повинен забезпечувати максимальний рівень захисту для своїх клієнтів. Нарешті, відгуки користувачів та позитивний досвід впровадження системи також є важливими складовими успіху, оскільки рекомендації від задоволених клієнтів допомагають залучати нових споживачів та підвищувати лояльність існуючих.

8 ЗАХОДИ З ОХОРОНИ ПРАЦІ ТА ТЕХНІКИ БЕЗПЕКИ

8.1 Вступ

Охорона праці – система збереження життя і здоров'я працівників у процесі трудової діяльності, що включає правові, соціально-економічні, організаційні, технічні, санітарно-гігієнічні, лікувально-профілактичні, реабілітаційні та інші заходи. Загальні положення державної політики, щодо галузі охорони праці зазначені у Законі України “Про охорону праці”. Цей Закон визначає основні положення щодо реалізації конституційного права працівників на охорону їх життя і здоров'я у процесі трудової діяльності, на належні, безпечні і здорові умови праці, регулює за участю відповідних органів державної влади відносини між роботодавцем і працівником з питань безпеки, гігієни праці та виробничого середовища і встановлює єдиний порядок організації охорони праці в Україні [20]. Законодавство про працю містить норми і вимоги з техніки безпеки і виробничої санітарії, норми, що регулюють робочий час і час відпочинку, звільнення та переведення на іншу роботу, норми праці щодо жінок, молоді, гігієнічні норми і правила тощо. Загальний нагляд за додержанням норм охорони праці покладено на прокуратуру, спеціальний – на професійні спілки. Контроль за безпекою праці здійснюють також, державні й відомчі спеціалізовані інспекції. Науково-технічний прогрес вніс серйозні зміни в умови виробничої діяльності робітників розумової діяльності. Їх праця стала більш інтенсивною, напруженою і вимагає значних витрат розумової, емоційної і фізичної енергії. Це призвело до необхідності у знаходженні комплексного рішення проблем ергономіки, гігієни і організації праці, регламентації режимів праці та відпочинку. Охорона здоров'я робітників, забезпечення безпеки умов праці, ліквідація та профілактика професійних захворювань і виробничого травматизму складає одну з головних турбот людського суспільства.

8.2 Аналіз умов праці на робочому місці ІТ-фахівця

На робочому місці ІТ-фахівця (або програміста) виникають небезпечні та шкідливі для безпечної життєдіяльності фактори:

– підвищений рівень шуму;

– несприятливі мікрокліматичні умови;

– недостатній рівень освітленості;

– шкідливі речовини;

– підвищений рівень електромагнітних випромінювань радіочастот;

– висока напруга електричної мережі;

– статична електрика та інші.

Робота програміста супроводжується також підвищеним ступенем напруженості трудового процесу. При систематичному впливі виробничих факторів, які не відповідають нормативним показникам, зростає рівень професійно зумовленої захворюваності працюючих та можуть виникнути професійні захворювання органів зору, руху, нервової системи. Таким чином, вивчення умов праці на робочому місці програміста є необхідною умовою запобігання негативних наслідків впливу небезпечних та шкідливих факторів. Робоче місце, добре пристосоване до трудової діяльності інженера, правильно і доцільно організоване, щодо простору, форми, розміру забезпечує йому зручне положення при роботі і високу продуктивність праці при найменшому фізичному і психічному напруженні.

Нормування параметрів проводиться в залежності від періоду року та категорії важкості виконуваних робіт. Для постійних робочих місць, якими є робочі місця ІТ-фахівців, встановлені оптимальні параметри мікроклімату, а за неможливості їх дотримання використовують допустимі параметри. Робота ІТ-фахівця за важкістю відноситься до Іа (роботи, що виконуються сидячи і не потребують фізичного напруження) та Іб (роботи, що виконуються сидячи, стоячи або пов'язані з ходінням та супроводжуються деяким фізичним

напруженням) категорій. В таблиці 8.1. наведені оптимальні параметри мікроклімату в приміщеннях.

Таблиця 8.1 – Параметри мікроклімату для приміщень з ПК

Період року	Параметр мікроклімату	Величина
Холодний	Температура повітря в приміщенні; відносна вологість; швидкість руху повітря	22...24 °С; 40... 60%; до 0,1 м/с
Теплий	Температура повітря в приміщенні; відносна вологість; швидкість руху повітря	23...25 °С; 40...60%; до 0,1...0,2 м/с

Виміряні за допомогою приладів температура та вологість у приміщеннях праці ІТ-фахівців повинні відповідати зазначеним у таблиці для теплого періоду року. Слід зазначити, що для нормалізації параметрів мікроклімату слід використовувати у приміщеннях кондиціювання повітря, або забезпечити подачу свіжого повітря системами вентиляції. Норми подачі свіжого повітря наведені у таблиці 8.2.

Таблиця 8.2 – Норми подачі свіжого повітря в приміщення

Характеристика приміщення	Об'ємна витрата свіжого повітря, що подається в приміщення, $м^3$ на одну людину в годину
Об'єм до 20 $м^3$ на людину	Не менше 30
20... 40 $м^3$ на людину	Не менше 20
Більше 40 $м^3$на людину	Може біти використана природна вентиляція

Створення сприятливих умов праці і правильне естетичне оформлення робочих місць на виробництві має велике значення як для полегшення праці, так і

для підвищення його привабливості, позитивно впливає на продуктивність праці. Забарвлення приміщень і меблів повинні сприяти створенню сприятливих умов для зорового сприйняття, гарного настрою. У службових приміщеннях, у яких виконується одноманітна розумова робота, що вимагає значної нервової напруги і великого зосередження, забарвлення повинно бути спокійних тонів – малонасичені відтінки холодного зеленого або блакитного кольорів.

При розробці оптимальних умов праці програміста необхідно враховувати освітленість. Раціональне освітлення робочого місця є одним з найважливіших факторів, що впливають на ефективність трудової діяльності людини, що попереджають травматизм і професійні захворювання. Правильно організоване освітлення створює сприятливі умови праці, підвищує працездатність і продуктивність праці. Освітлення на робочому місці програміста повинно бути таким, щоб працівник міг без напруги зору виконувати свою роботу. Стомлюваність органів зору залежить від ряду причин: недостатність освітленості; надмірна освітленість; неправильний напрям світла. Недостатність освітлення приводить до напруги зору, ослабляє увагу, приводить до настання передчасної стомленості. Надмірно яскраве освітлення викликає засліплення, роздратування і різь в очах. Неправильний напрямок світла на робочому місці може створювати різкі тіні, відблиски, дезорієнтувати працюючого. Всі ці причини можуть призвести до нещасного випадку або профзахворювань. [21]

8.3 Пропозиції щодо підвищення працездатності ІТ-фахівця

Практичне значення заходів щодо підвищення працездатності випливає із закономірностей її динаміки і зводиться ось до чого:

– збільшення фази стійкого стану у фонді робочого часу;
– прискорення процесу працювання;
– віддалення фази розвитку втоми;
– забезпечення високої продуктивності праці за нормальних фізіологічних

затрат.

Комплекс заходів щодо підвищення і збереження працездатності працівників на оптимальному рівні реалізується на техніко-організаційному, соціально-економічному, санітарно-гігієнічному, медико-біологічному, психологічному напрямках.

Вагомим фактором високої працездатності і продуктивності праці є оптимізація трудових навантажень на основі механізації і автоматизації виробничих процесів, удосконалення технології, скорочення і ліквідації важкої ручної праці. Доведено, що при правильній організації праці на легких роботах спостерігається найбільша тривалість фази стійкого стану, а на важких роботах вона нетривала.

Високий рівень працездатності безпосередньо залежить від умов праці, оскільки поліпшення їх супроводжується зменшенням енергетичних затрат організму на подолання несприятливого впливу факторів виробничого середовища.

Важливим напрямком підвищення працездатності працюючих є ритмізація трудових процесів, оптимізація темпу роботи, а також раціоналізація трудових рухів на фізіологічній основі, що сприяє формуванню і закріпленню робочих динамічних стереотипів, а отже зменшенню м’язових і вольових зусиль. Ритмічна робота підвищує функціональні можливості організму, сприяє його тренованості і забезпечує економізацію енергетичних затрат. [1]

Багатьом програмістам постійно доводиться працювати з великою кількістю програм одночасно. Часте перемикання туди-сюди між IDE та довідкою суттєво зменшує продуктивність фахівця. Однак вирішення цієї проблеми досить просте та очевидне: встановлення більшої кількості моніторів.

Оптимальним варіантом є два монітори. Все ж таки це найпростіший з апаратної точки зору варіант. Крім того, якби їх було більше, то ними було б важче керувати, та й столі просто не вистачить місця на ще один монітор. Але тут ще залежить розміру моніторів. Є системи із 4 або 6 відносно невеликими

екранами, які кріпляться на кронштейні. Але оптимальнім є два 27-дюймові монітори, на яких все добре видно, особливо коли працювати доводиться в основному з текстом [3].

8.4 Пожежна безпека

Вимоги до пожежної безпеки на підприємстві неухильно повинен дотримуватися кожен співробітник, а організаційна складова при цьому покладається на посадових осіб за відповідним рішенням керівництва і прописується в посадових інструкціях і положеннях по структурним підрозділам.

Зокрема, вказуються конкретні території, ділянки, зони, об’єкти, цілі будівлі і їх частини, поверхи, на яких відповідального співробітника повинне проводити такі організаційні роботи.

Відповідальні особи зобов’язуються розробити, впровадити та підтримувати в певному інструкцією і положенням на ввірених їм об’єктах протипожежний режим і інструкції відповідно до вимог, викладених в нормативних актах.

Передбачено також створення підрозділу добровільної пожежної охорони та пожежно-рятувальної команди в його складі.

Встановлений режим включає порядки з описом місць спеціального призначення та правила їх користування та утримання, наприклад:

– евакуаційних шляхів;

– так званих «курилок»;

– місць складування продукції та сировини;

– стоянки транспорту.

Також встановлюється порядок роботи та технічного обслуговування:

– вентиляційного устаткування;

– засобів пожежогасіння і захисту від загорянь;

– нагрівальних приладів;

– електрообладнання.

Розробляються і впроваджуються правила роботи з відкритим вогнем і горючими матеріалами. Створюються графіки проходження інструктажів з пожежної безпеки співробітників, а також порядок і терміни перевірок знань пожежно-технічного мінімуму, в тому числі, тих працівників, які відповідальні за цю ділянку роботи на підприємстві. При цьому можуть передбачатися внутрішні лекції, семінари, тренінги та практичні заняття на підприємстві, а також зовнішні – на базі спеціалізованих навчальних центрів з професійними викладачами.

Важливою складовою протипожежного режиму на будь-якому об’єкті є розробка і впровадження порядку дій при виникненні пожежі. Неодмінно має бути план евакуації, описано, як повинні відключатися електроустановки, що і в якій послідовності необхідно робити співробітникам.

Відповідно, для кожного об’єкта, кожного приміщення (крім коридорів, санвузлів, басейнів і подібних приміщень), окремих видів робіт складаються інструкції, за якими повинен працювати персонал, залучений на певних ділянках і в виконанні окремих видів робіт. За інструкціями проводиться навчання (інструктаж) персоналу з подальшим контролем знань.

Детально про те, як розробити протипожежний режим, прописати порядки та інструкції, пояснюють на тематичних курсах і семінарах [24].

8.5 Розрахункова частина

Проведемо розрахунок штучного освітлення за методом коефіцієнта використання світлового потоку для приміщення, ширина якого складає 6 м, довжина – 7 м, висота – 3,4 м.

У зазначеному приміщенні працює 7 людей.

Для того, щоб визначити потрібну кількість світильників, які повинні забезпечити нормований рівень освітленості, визначимо світловий потік, що

падає на робочу поверхню за формулою [1]:

$$F = E \cdot S \cdot K \cdot Z \cdot / n,$$

де

F – світловий потік, що розраховується, Лм;

E – нормована мінімальна освітленість, Лк; E = 300 Лк;

S – площа освітлюваного приміщення (у нашому випадку S = 6 х 7 = 42 м^2);

K – коефіцієнт запасу, що враховує зменшення світлового потоку лампи в результаті забруднення світильників в процесі експлуатації (його значення залежить від типу приміщення і характеру робіт, що проводяться в ньому, в нашому випадку K = 1,5);

Z – відношення середньої освітленості до мінімальної (зазвичай приймається рівним 1.1... 1.2, в нашому випадку Z = 1,1);

n – коефіцієнт використання світлового потоку, (відношення світлового потоку, що падає на розрахункову поверхню, до сумарного потоку всіх ламп і обчислюється в долях одиниці [8]; залежить від характеристик світильника, розмірів приміщення, забарвлення стін і стелі, що характеризуються коефіцієнтами відбиття від стін ($\rho_{стін.}$) і стелі ($\rho_{стелі}$), значення коефіцієнтів дорівнюють $\rho_{стін}$ = 50% і $\rho_{стелі}$ = 50%.

Обчислимо індекс приміщення за формулою:

$$i=S/(h(A+B)),$$

де:

S – площа приміщення, S = 42 м2;

h – розрахункова висота підвісу, h = 3,4 м (співпадає з висотою стелі, т.я. лампи освітлення закріплюються на стелі);

A – ширина приміщення, A = 6 м;

B – довжина приміщення, B = 7 м.

Підставимо всі значення у формулу та визначимо індекс приміщення:

$$i=1,4.$$

Знаючи індекс приміщення, за знаходимо n = 0,23 (з табличних даних коефіцієнтів використання світлового потоку (n) світильників з відповідним типом лампам) [8]. Підставимо всі значення у формулу, визначимо світловий потік: F = 90391 Лм.

Для розрахунку дудемо використовувати світлодіодні стельові панелі Призма-72 6400К, світловий потік яких $F_л$ = 7200 Лм.

Число ламп визначається по формулі:

$$N=F/F_л$$

де

F – світловий потік,

$F_л$ – світловий потік однієї лампи.

Підставимо всі значення у формулу та визначимо кількість світильників:

$$N= 90391 / 7200=12,5 \text{ шт.}$$

Приймаємо необхідну кількість світлодіодних світильників 13 шт.

Висновки до розділу

Дотримання всіх необхідних умов праці не лише сприяє збереженню здоров’я працівників, а також підвищує ефективність виробництва в цілому.

З цих міркувань було здійснено аналіз умов праці, призначеного для праці програмістів, проведено розгляд небезпечних та шкідливих факторів, що негативно впливають на програмістів під час роботи. Виконано розрахунок штучного освітлення, як одного з ключових факторів впливу на працездатність та здоров’я програміста. Розроблено заходи з умов поліпшення охорони праці.

9 ОСНОВНІ ВИСНОВКИ

Програмне забезпечення, створене в результаті виконання випускної кваліфікаційної роботи за другим (магістерським) рівнем вищої освіти, призначено для системи контейнерної віртуалізації мережевої інфраструктури для хостингу.

В межах України в недостатній мірі представлені вітчизняні розробки в цій області.

У випускній кваліфікаційній роботі за другим (магістерським) рівнем вищої освіти наведені теоретичне узагальнення й рішення наукового завдання дослідження методів контейнерної віртуалізації мережевої інфраструктури для хостингу.

Рішення даного завдання полягало у вирішенні наступних задач:

– Був проведений огляд існуючих систем контейнерної віртуалізації мережевої інфраструктури для хостингу.

– Досліджена система контейнерної віртуалізації мережевої інфраструктури для хостингу.

– На основі отриманих результатів досліджень створена програмна реалізація системи контейнерної віртуалізації мережевої інфраструктури для хостингу.

Розроблені під час виконання випускної кваліфікаційної роботи за другим (магістерським) рівнем вищої освіти алгоритми дозволяють успішно вирішувати завдання контейнерної віртуалізації мережевої інфраструктури для хостингу.

Проведено аналіз предметної галузі в ході якого були виявлені об'єкти, взаємодія яких носить істотний характер для функціональної діяльності предметної галузі, і їхні основні характеристики; побудована алгоритм і вибраний середовище розробки.

Розроблене програмне забезпечення має простий, дружній та зручний інтерфейс користувача, що забезпечує легкість у освоєнні роботи програмного продукту, зручність у використанні, і не потребує особливих спеціальних знань.

При створені програмного забезпечення було використано об'єктно-орієнтований підхід, що відповідає сучасним тенденціям у галузі розробки комерційних програмних систем.

Програма реалізована на мові високого рівня Python. Дана мова програмування дозволяє найбільш ефективно обробляти дані. Це дозволило мінімізувати строк розробки програмного забезпечення, і, як слід, зменшити витрати на його розробку. Запропоноване програмне забезпечення ділиться на загальне програмне забезпечення, що поставляється із засобами обчислювальної техніки й спеціальне програмне забезпечення, що спеціально розроблене для даної конкретної системи й включає програми, що реалізують її функції.

Програма призначена для виконання під управлінням багатозадачної операційної системи Windows 10/11.

Даються необхідні рекомендації з установки розробленого програмного забезпечення.

Для підвищення рівня безпеки запропоновано застосовувати алгоритм SEED.

В цілому створене програмне забезпечення підтверджує правильність використаних проектних рішень та повністю відповідає вимогам технічного завдання. Створене програмне забезпечення має потенційну можливість для подальшого вдосконалення і застосування у різних галузях.

Проведено маркетингове та економічне обґрунтування ІТ-проєкту, що дозволило визначити ключові фактори успіху даного проекту.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Павлюхін В.Л. Дослідження та програмна реалізація системи контейнерної віртуалізації мережевої інфраструктури для хостингу // Збірник праць молодих науковців ЦНТУ. – Вип. 15. – Кропивницький: ЦНТУ, 2025. .

2. Ramon Nastase «Computer Networking: The Beginner's guide for Mastering Computer Networking, the Internet and the OSI Model». 2018. – 186 p.

3. Russ White & Ethan Banks «Computer Networking Problems and Solutions: An Innovative Approach to Building Resilient, Modern Networks». 2017. – 832 p.

4. Вінтенко Б., Смірнов О., Миронець І., Смірнова Т., Смірнов С. «Імітаційна модель шляхів вхідних даних комп'ютерної інтелектуальної системи підтримки оператора енергоблоку АЕС». Комбінаторні конфігурації та їхні застосування: Матеріали XXVII Міжнародного науково-практичного семінару, присвяченого 125-річчю Національного університету «Запорізька політехніка» (Запоріжжя-Кропивницький-Київ, 4-6 червня 2025 р.). Запоріжжя: НУ «Запорізька політехніка», 2025. С.82-91.

5. Al-Azzeh, J., Ayyoub, B., Mesleh, A., Smirnova, T., Gnatyuk, S., Drieiev, O., Smirnov, O., Dorenskyi, O. «Cloud-Based Information System for Evaluating Caverns in the Process of Blasting Metal Surfaces of Details». International Review on Modelling and Simulations 18 (1), 2025. pp. 32-42.

6. Смірнова Т.В., Конопліцька-Слободенюк О.К., Буравченко К.О., Смірнов С.А., Кравчук О.В., Козірова Н.Л., Смірнов О.А. «Дослідження технологій забезпечення кібербезпеки хмарних сервісів IaaS, PaaS та SaaS». Кібербезпека: освіта, наука, техніка. 2024. №4(24), С. 6-27.

7. Батрак О., Смірнова Т., Гнатюк В., Одарченко Р., Смірнов О. «Дослідження показників ефективності функціонування та перспектив розвитку систем IP-телефонії». Підводні технології, 2024, № 13, с. 28-35.

8. Kuznetsov, O., Kryvinska, N., Ilchenko, O., Smirnova, T., Ulianovska, Y. «Comparative Analysis of Cryptocurrency Trading Platforms Using the Analytic Hierarchy Process». CEUR Workshop Proceedings, 2023, 3628, pp. 106-115.

9. Al-Mudhafar Aqeel, A.M., Smirnova, T., Buravchenko, K., Smirnov, O. «The method of assessing and improving the user experience of subscribers in software-configured networks based on the use of machine learning». Advanced Information Systems, 2023, 7(2), pp. 49-56.

10. Smirnov, O., Sydorenko, V., Aleksander, M., Zhyharevych, O., Yenchev, S. «Simulation of the cloud IoT-based monitoring system for critical infrastructures». CEUR Workshop Proceedings, Volume 3530, 2023, pp. 256-265.

11. Smirnov, O., Odarchenko, R., Smirnova, T., Bondar, S., Volosheniuk, D. «Optimal Structure Construction of Private 5G Network for the Needs of Enterprises». Lecture Notes on Data Engineering and Communications Technologies, 2023, 178, pp. 208–223.

12. Аль-Мудхафар Акіл Абдулхуссейн М., Смірнова Т.В., Буравченко К.О., Смірнов О.А. «Метод оцінки та підвищення користувальницького досвіду абонентів в програмно-конфігурованих мережах на основі використання машинного навчання». Сучасні інформаційні системи, 2023, том 7, № 2, С. 49-56.

13. Smirnova, T., Gnatyuk, S., Yudin, O., Sydorenko, V., Polozhentsev, A., «The Model for Calculating the Quantitative Criteria for Assessing the Security Level of Information and Telecommunication Systems». CEUR Workshop Proceedings Volume 3156, 2022, Pages 390-399.

14. Смірнова Т.В., Гнатюк С.О., Сидоренко В.М., Юдін О.Ю., Сидоренко С.Ю., «Модель визначення критичності галузевих інформаційно-телекомунікаційних систем». Проблеми інформатизації та управління, № 2(70). 2022. С. 28-37.

15. Смірнов О.А., Смірнова Т.В., Якименко Н.М., Смірнов С.А., Поліщук Л.І., «Дослідження стійкості до диференціального криптоаналізу запропонованої функції гешування удосконаленого модуля криптографічного

захисту в інформаційно-комунікаційних системах» Системи управління, навігації та зв'язку, 2022, № 3(69). С. 93-98.

16. Смірнов О.А., Смірнова Т.В., Якименко Н.М., Поліщук Л.І., Смірнов С.А. «Дослідження статистичної стійкості та швидкісних характеристик запропонованої функції гешування удосконаленого модуля криптографічного захисту в інформаційно-комунікаційних системах» Вісник Хмельницького національного університету. Серія: «Технічні науки», № 2 (307). С. 46-52. 2022.

17. Смірнов О.А., Смірнова Т.В., Константинова Л.В., Смірнов С.А., Якименко Н.М., «Дослідження стійкості до лінійного криптоаналізу запропонованої функції гешування удосконаленого модуля криптографічного захисту в інформаційно-комунікаційних системах» Системи управління, навігації та зв'язку, 2022, № 1(67). С. 84-89.

18. Smirnova T., Gnatyuk S., Berdibayev R., Avkurova Zh., Iavich M. «Cloud-Based Cyber Incidents Response System and Software Tools». Communications in Computer and Information Science, 2021, vol 1486. Springer, Cham. pp 169-184.

19. Smirnov O., Kuznetsov A., Kiian A., Kuznetsova T. «Non-binary constant weight coding technique». CEUR Workshop Proceedings. Volume 2740, 2020, Pages 102-114.

20. Smirnov O., Alimseitova Zh., Adranova A., Akhmetov B., Lakhno V., Zhilkishbayeva G. «Models and algorithms for ensuring functional stability and cybersecurity of virtual cloud resources». Journal of theoretical and applied information technology Vol.98. No 21, 2020, P. 3334-3346.

21. Smirnov O., Kuznetsov A., Kiian A., Cherep A., Kanabekova M., Chepurko I. «Testing of code-based pseudorandom number generators for post-quantum application». 2020 IEEE 11th International Conference on Dependable Systems, Services and Technologies (DESSERT), Ukraine, Kyiv, May 14-18. 2020. P. 172-177.

22. Smirnov O., Kuznetsov A., Pushkar'ov A., Serhiienko R., Babenko V., Kuznetsova T., «Representation of Cascade Codes in the Frequency Domain». In: Radivilova T., Ageyev D., Kryvinska N. (eds) Data-Centric Business and Applications. Lecture Notes on Data Engineering and Communications Technologies, vol 48. Springer, Cham. 2021. pp 557-587.

23. Smirnov, O., Markovets, O. Vovk, N., Turchyn, Y., «Model of informational support for social network administrators' content creation». CEUR Workshop Proceedings Volume 2616, 2020, Pages 125-136.

24. Smirnov, O., Drieieva, H., Drieiev, O., Polishchuk, Y., Brzhanov, R., Aleksander, M. «Method of fractal traffic generation by a model of generator on the graph». CEUR Workshop Proceedings Volume 2616, 2020, Pages 366-379.

25. Smirnov, O., Drieieva, H., Drieiev, O., Simakhin, V., Bondar, S., Odarchenko, R. «Managing multifractal properties of the binary sequence generated with the Markov chains», CEUR Workshop Proceedings Volume 2608, 2020, Pages 633-645.

26. Smirnov O. Kuznetsov A., Zaicenko Yu., Pastukhov M., Oleshko O., Kuznetsova K., «Formation of Discrete Signals with Special Correlation Properties». International Conference on Information and Telecommunication Technologies and Radio Electronics, UkrMiCo 2019; Odessa; Ukraine; 9-13 September 2019. P.22-28.

27. Smirnov, O., Kuznetsov, A., Kolovanova, I., Kuznetsova, T., «Noise immunity of the algebraic geometric codes». International Journal of Computing; 2019, Volume 18, Issue 4 – Research Institute for Intelligent Computer Systems – 2019. – P. 393-407.

28. Smirnov, O., Kuznetsov, A., Reshetniak, O., Ivko, N., Katkova, T., Kuznetsova, T., «Generators of Pseudorandom Sequence with Multilevel Function of Correlation». 2019 IEEE International Scientific-Practical Conference Problems of Infocommunications, Science and Technology (PIC S&T), Kyiv, Ukraine, 8 – 11 October 2019. P.517-522.

29. Smirnov, O., Odarchenko, R., Abakumova, A., Usik, P., Kundyz, M., «QoE optimization technique for media delivery in 5G networks». 2019 IEEE International Scientific-Practical Conference Problems of Infocommunications, Science and Technology (PIC S&T), Kyiv, Ukraine, 8 – 11 October 2019. P.597-601.

30. Smirnov, O., Krasnobayev, V., Yanko, A., Kuznetsova, T. «Methods of nulling numbers in the system of residual classes». CEUR Workshop Proceedings, Vol 2588, P. 90-106, 2019.

31. Smirnov, O., Kuznetsov, A., Kovalchuk, D., Averchev, A., Pastukhov, M., Kuznetsova, K., «Formation of Pseudorandom Sequences with Special Correlation Properties», 2019 3rd International Conference on Advanced Information and Communications Technologies, AICT -2019/ Lviv, Ukraine, 2-6 July, 2019, P. 395-399.

32. Smirnov, O., Kuznetsov, A., Kiian, A., Zamula, A., Rudenko, S., Hryhorenko, V., «Variance Analysis of Networks Traffic for Intrusion Detection in Smart Grids», 2019 IEEE 6th International Conference On Energy Smart Systems (2019 IEEE ESS), Kyiv, Ukraine April 17-19, 2019 P. 353-358.

33. Smirnov, O., Kuznetsov, A., Kavun, S., Babenko, B., Nakisko, O., Kuznetsova, K., «Malware Correlation Monitoring in Computer Networks of Promising Smart Grids», 2019 IEEE 6th International Conference On Energy Smart Systems (2019 IEEE ESS), Kyiv, Ukraine April 17-19, 2019 P. 347-352.

34. Smirnov, O., Kuznetsov, A., Kovalchuk, D., Pastukhov, M., Kuznetsova, K., Prokopovych-Tkachenko, D., «Discrete Signals with Special Correlation Properties», CEUR Workshop Proceedings Volume 2353, CEUR Workshop Proceedings 2019, Pages 618-629.

35. Smirnov A.A., Kuzne tsov A.A., Danilenko D.A., Berezovsky A., «The statistical analysis of a network traffic for the intrusion detection and prevention systems», Telecommunications and Radio Engineering. – Volume 74, Issue 1. – Begel House Inc. – 2015. – P. 61-78.

36. Смірнов О.А., Смірнова Т.В., Буравченко К.О., Кравченко С.С., Горбов В.О., «Хмарна система підтримки прийняття рішень технологічного процесу відновлення поверхонь конструкцій і деталей машин». Сучасні інформаційні системи. 2021. Т. 5, № 4. С. 79-95.

37. Смірнов О.А., Усік П.С., Миронець І.В., Буравченко К.О., Якименко Н.М. «Метод підвищення ефективності розподіленої обробки даних у комп'ютерних системах операторів стільникового зв'язку» Вісник Черкаського державного технологічного університету. Технічні науки. №4. С. 103-110. 2020.

38. О.А.Смірнов, Т.В.Смірнова, Л.І. Поліщук, К.О. Буравченко, А.О.Макевнін, «Дослідження хмарних технологій як сервісів», Кібербезпека: освіта, наука, техніка. № 3(7). С. 43-62. 2020.

39. Смірнов О.А., Конопліцька-Слободенюк О.К., Смірнов С.А., Буравченко К.О., Смірнова Т.В., Поліщук Л.І. Інформаційна безпека в комп'ютерних мережах. Навчальний посібник – Кропивницький: вид. Лисенко В.Ф. 2020. – 294 с. .

40. О.А. Смірнов, П.С. Усік, «Дослідження перспектив використання технологічних рішень в мережах 5G» у Кібербезпека та інформаційні технології: монографія. – Х. : ТОВ «ДІСА ПЛЮС», 2020.С. 122-135.

41. Смірнов О.А., Дрєєва Г.М., Дрєєв О.М., Смірнова Т.В. «Фрактальний аналіз генератора самоподібного трафіку на основі ланцюга Маркова». Центральноукраїнський науковий вісник. Технічні науки. № 2(33). с. 161-172, 2019. .

42. Смірнов О.А., Конопліцька-Слободенюк О.К., Смірнов С.А., Буравченко К.О., Смірнова Т.В. Поліщук Л.І. Проектування комп'ютерних систем та мереж. Навчальний посібник – Кропивницький: вид. Лисенко В.Ф. 2019. – 264 с.

43. Smirnov, O., Kuznetsov, A., Kuznetsova., K. Synthesis of Discrete Signals with Improved Correlation Properties. Монографія: In.: ISCI'2019: Information Security in Critical Infrastructures. Collective monograph. Edited by Ivan

D. Gorbenko and Alexandr A. Kuznetsov, ASC Academic Publishing, USA, 2019, pp. 281-299. – ISBN: 978-0-9989826-8-7 (Hardback), ISBN: 978-0-9989826-9-4 (Ebook).

44. Смірнов О.А., Дрєєва Г.М. Метод генерування фрактального трафіку за допомогою моделі генератора на графі. Монографія: Інформаційна безпека та інформаційні технології : монографія / за заг. ред. В. С. Пономаренка. – Х. : Вид. Рожко С.Г. 2019. С. 123-139.

45. Дрєєва Г.М., Смірнов О.А., Дрєєв О.М. Метод генерування фрактальноподібної числової послідовності на основі скінченного автомату для моделювання трафіку у мережі. Центральноукраїнський науковий вісник. Технічні науки. № 1(32). с. 173-183, 2019.

46. Смірнова Т.В., Солових Є.К., Смірнов О.А., Дрєєв О.М. Побудова хмарних інформаційних технологій оптимізації технологічного процесу відновлення та зміцнення поверхонь деталей. Центральноукраїнський науковий вісник. Технічні науки. № 1(32). с. 184-194, 2019.

47. Смірнов О.А., Смірнов С.А., Поліщук Л.І., Смірнова Т.В., Конопліцька-Слободенюк О.К. Метод формування антивірусного захисту даних з використанням безпечної маршрутизації метаданих. Кібербезпека: освіта, наука, техніка. – Том 3 № 3. – Київ: КУ ім. Бориса Грінченка. – 2019. – С. 63-87.

48. Смірнов О.А., Гнатюк С.О., Кавун С.В., Терейковський І.А., Жмурко Т.О., Смірнов С.А., Коваленко А.С. Основи безпеки в комп'ютерних мережах. Навчальний посібник – Кропивницький: вид. Лисенко В.Ф. 2018. – 177 с.

49. Смірнов О.А., Котелянець В.В. Стійкі до колізій стохастичні моделі функціонування безпроводових сенсорних мереж. Вісник інженерної академії України, №3, с. 145-152, 2018.

50. Смірнов О.А., Смірнов С.А., Дідик А.К., Дрєєв А.М. Алгоритми формування безлічі маршрутів передачі метаданих у антивірусні хмарні системи. Збірник наукових праць "Системи обробки інформації". – Випуск 5 (142). – Х.: ХУПС – 2016. – С. 148-152.