МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
НАЦІОНАЛЬНИЙ АВІАЦІЙНИЙ УНІВЕРСИТЕТ
ФАКУЛЬТЕТ КІБЕРБЕЗПЕКИ, КОМП'ЮТЕРНОЇ ТА ПРОГРАМНОЇ ІНЖЕНЕРІЇ
КАФЕДРА КОМП'ЮТЕРИЗОВАНИХ СИСТЕМ УПРАВЛІННЯ

ДОПУСТИТИ ДО ЗАХИСТУ
Завідувач кафедри

_____ Литвиненко О.Є.
«_____»_____2021 р.

ДИПЛОМНИЙ ПРОЄКТ
(ПОЯСНЮВАЛЬНА ЗАПИСКА)

ЗДОБУВАЧА ОСВІТНЬОГО СТУПЕНЯ
«БАКАЛАВР»

Тема: «Програмний модуль моніторингу мережевої активності»

Виконавець: Чкана С.В.

Керівник: Марченко Н.Б.

Нормоконтролер: Тупота Є.В.

Київ 2021

НАЦІОНАЛЬНИЙ АВІАЦІЙНИЙ УНІВЕРСИТЕТ

Факультет Кібербезпеки, комп'ютерної та програмної інженерії
Кафедра комп'ютеризованих систем управління
Спеціальність 123 «Комп'ютерна інженерія»
(шифр, найменування)

Освітньо-професійна програма «Системне програмування»
Форма навчання денна

ЗАТВЕРДЖУЮ
Завідувач кафедри

Литвиненко О.Є.
«_____»_____2021 р.

ЗАВДАННЯ
на виконання дипломної роботи (проєкту)
Чкані Сергію Володимировичу
(прізвище, ім'я, по батькові випускника в родовому відмінку)

1. Тема дипломної роботи (проекту) «Програмний модуль моніторингу мережевої активності»

затверджена наказом ректора від « 04 » лютого 2021 р. № 135/ст.

2. Термін виконання роботи (проєкту): з 17 травня 2021 р. по 20 червня 2021 р.

3. Вихідні дані до роботи (проєкту): мова програмування *Python3*, бібліотека *PyQt5*, операційна система *Linux*, середовище для розробки графічних інтерфейсів *qt5-designer*, *pyuic5*, вимоги до оформлення дипломних проектів та ДСТУ 3008-95

4. Зміст пояснювальної записки:

аналіз принципів побудови існуючих рішень для моніторингу мереж;

обґрунтування виборів методів реалізації;

розробка компонентів системи;

аналіз роботи програмного продукту моніторингу мережевої активності.

5. Перелік обов'язкового графічного (ілюстративного) матеріалу:

1) блок-схема центрального і дочірнього вузлів системи;

2) програмний модуль моніторингу мережевої активності. Схема структурна

3) модуль розбору пакетів. Схема алгоритму.

6. Календарний план-графік

№ пор.	Завдання	Термін виконання	Відмітка про виконання
1.	Провести огляд літератури за темою дипломного проекту та аналіз існуючих систем.	17.05.2021-18.05.2021	
2.	Зробити вибір середовища програмування і компонентів програмного комплексу	18.05.2021-20.05.2021	
3.	Розробити структуру програмних засобів системи	21.05.2021-22.05.2021	
4.	Розробити програмні засоби. Провести відладку програмних засобів	23.05.2021-05.06.2021	
5.	Написати пояснювальну записку	06.06.2021-10.06.2021	
6.	Підготувати графічний та ілюстративний матеріал	11.06.2021-12.06.2021	
7.	Підготувати доповідь та презентацію	12.06.2021-20.06.2021	

7. Дата видачі завдання: «_17_»___травня____ 2021 р.

Керівник дипломної роботи (проєкту) _____ Марченко Н.Б.
(підпис керівника) (П.І.Б.)

Завдання прийняв до виконання _____ Чкана С.В.

РЕФЕРАТ

Пояснювальна записка до дипломного проекту «Програмний модуль моніторингу мережевої активності»: 60 сторінок, 27 рисунків, 1 таблиця, 23 джерела.

Ключові слова: МОНІТОРИНГ, ОБСЛУГОВУВАННЯ, *PYTHON3, PYQT5, LINUX, QT5-DESIGNER, PYUIC5*.

Об’єкт дослідження – система моніторингу мережі як в активному, так і в пасивному режимі.

Предмет дослідження – надійність функціонування веб-монітору при різних реалізаціях.

Мета дипломного проекту – розробити програмний модуль моніторингу мережевої активності.

Методи проектування – мова програмування *Python3*, бібліотека *PyQt5*, операційна система *Linux*, середовище для розробки графічних інтерфейсів *qt5-designer*, *pyuic5*.

Прогнозні припущення щодо розвитку об’єкта дослідження – використання робочого зразка програмного модуля з інтуїтивно-зрозумілим інтерфейсом, який дозволяє забезпечити моніторинг мережі як в активному, так і в пасивному режимі.

Розроблена система моніторингу дозволятиме:

– в реальному часі отримувати пакети, що проходять через комп'ютер;

– розбирати пакети на складові для зручного надання даних користувачеві;

– збирати і відображати інформацію для моніторингу, таку як пропускна спроможність, кількість пакетів за одиницю часу, час життя пакетів відображений у вигляді графіку, кількість втрачених пакетів.

Результати проектування – програмний модуль для моніторингу мережі як в активному, так і в пасивному режимі.

ЗМІСТ

ПЕРЕЛІК СКОРОЧЕНЬ, УМОВНИХ ПОЗНАЧЕНЬ, ТЕРМІНІВ 6
ВСТУП……..7
РОЗДІЛ 1 АНАЛІЗ ІСНУЮЧИХ РІШЕНЬ, ОБҐРУНТУВАННЯ ТЕМИ ДИПЛОМУ 9
1.1. Моніторинг мережі..... 9
1.2. Управління якістю передачі інформації..... 15
РОЗДІЛ 2 РОЗГЛЯД СИСТЕМ МОНІТОРИНГУ ДЛЯ ОПЕРАТОРІВ ЗВ’ЯЗКУ . 21
2.1. Системи моніторингу..... 21
2.2. Типи та топологія систем моніторингу..... 22
2.3. Архітектура систем моніторингу та механізми моніторингу 23
2.4. Порівняльні характеристики систем моніторингу 26
2.5. Обґрунтування вибору методів реалізації 28
2.6. Мова програмування..... 31
2.7. Система моніторингу з використанням *Raspberry Pi*, датчиками та протоколом *ZIGBEE* 36
РОЗДІЛ 3 РОЗРОБКА КОМПОНЕНТІВ СИСТЕМИ 39
3.1. Загальна структура системи..... 39
3.2. Розробка графічного інтерфейсу. Використання фреймворку *PyQt5* .....**Ошибка! Закладка не определена.**
3.3. Розробка перехоплювача та розбору пакетів на їх складові. ..... **Ошибка! Закладка не определена.**
3.4. Аналіз роботи програмного продукту..... 61
ВИСНОВКИ 64
СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ 65
ДОДАТОК А 67

ПЕРЕЛІК СКОРОЧЕНЬ, УМОВНИХ ПОЗНАЧЕНЬ, ТЕРМІНІВ

ДП – диференційовані послуги

ІП – інтегровані послуги

КЯО – контроль якості обслуговування

СММ – система моніторингу мережі

ЦП – центральний процесор

DSCP – differentiated services code coint (кодова точка диференційованих послуг)

IP – internet protocol (міжмережевий протокол)

LAN – local area network. (локальна мережа)

MIB – management information base (база управляючої інформації)

MOS – mean opinion score (середній бал)

MPLS – multiprotocol label switching (багатопротокольна комутація за мітками)

NAT – network address translation

NMS – network monitoring systems (система моніторингу мережі)

PHB – per-hop behaviour (політика покрокового обслуговування)

PPS – packets per second

QOS – quality of service (якість обслуговування)

RSVP – resource reservation protocol (протокол резервування ресурсів)

SD-WAN – software-defined networking in a wide area network

SLA – service level agreement (угода про рівень обслуговування)

SNMP – simple network management protocol

TTL – time to live (час життя пакету)

URI – uniform resource identifier

WMI – windows management instrumentation

ВСТУП

Моніторинг та управління якістю передачі в комп'ютерній мережі завжди були не простим викликом навіть для професіоналів цієї справи. Для правильної роботи таких систем потрібні платформи з визначеним набором інструментів для коректної роботи. На сьогоднішній день з'являється все більше ресурсів, що наближаються до універсальних і дозволяють їх використовувати в самих різних мережевих системах.

Моніторинг мережі включає в себе декілька методів, які використовуються для підтримки безпеки та цілісності внутрішньої мережі. Внутрішня мережа також називається локальною мережею *LAN* (*Local Area Network*), а моніторинг охоплює контроль апаратного забезпечення, програмного забезпечення, наявність вірусів, шпигунських програм, вразливостей, такі як бекдор і отвори безпеки, а також інші аспекти, які можуть порушити цілісність мережі.

Існує широкий спектр методів моніторингу мережі, які реалізуються IT-фахівцями. Методи розгортаються за допомогою мережевих моніторингових рішень, які автоматично виявляють і реагують на проблеми безпеки і продуктивності. Розглянемо найбільш розповсюджені з них, до яких відносяться: виявлення вторгнень, перехоплення пакетів, сканування вразливостей, брандмауер моніторингу та тестування проникнення.

Такі платформи мають автоматизовані алгоритми для виявлення більшості мережевих компонентів та автоматично відображають інформацію їх корисного використання у текстовому, графічному та інших видах, для зручності користування.

Розуміючи всі аспекти проблеми якості обслуговування мережі необхідно розробити рішення для моніторингу пакетів даних, які передаються через робочу станцію, що дозволить забезпечити збір необхідної інформації для надання її користувачу.

Якість обслуговування *QoS* відноситься до будь- якої технології, яка управляє трафіком даних для зменшення втрати пакетів та затримки в мережі. Якість обслуговування контролює та управляє мережевими ресурсами, встановлюючи пріоритети для конкретних типів даних у мережі.

Корпоративні мережі повинні забезпечувати передбачувані та вимірювані послуги програмам для передачі голосу, відео та даних, що чутливі до затримки. Організації використовують контроль якості обслуговування для задоволення вимог до трафіку чутливих додатків, таких як передача голосових та відео потоків в реальному часі, а також для запобігання погіршенню якості, викликаної втратою пакетів, затримкою та помилками.

Область інструментів моніторингу мережі, програмного забезпечення та постачальників дуже велика. Нове програмне забезпечення, інструменти та утиліти запускаються майже щороку, щоб конкурувати на постійно мінливому ринку IT-моніторингу та моніторингу серверів.

Деякі з функцій, які присутні у інструментів моніторингу – це індикатори *Uptime*/*Downtime*, а також надійні та ретельні системи оповіщення (через електронну пошту/*SMS*), спеціальні шаблони та пороги, інтеграція з *SNMP*, функціональність автоматичного виявлення, зіставлення топології мережі та багато іншого.

РОЗДІЛ 1
АНАЛІЗ ІСНУЮЧИХ РІШЕНЬ, ОБҐРУНТУВАННЯ ТЕМИ ДИПЛОМУ

1.1. Моніторинг мережі

Моніторинг мережі включає в себе декілька методів, які використовуються для підтримки безпеки та цілісності внутрішньої мережі. Внутрішня мережа також називається локальною мережею *LAN* (*Local Area Network*), а моніторинг охоплює контроль апаратного забезпечення, програмного забезпечення, наявність вірусів, шпигунських програм, вразливостей, такі як бекдор і отвори безпеки, а також інші аспекти, які можуть порушити цілісність мережі.

1.1.1. Методи моніторингу мережі

Існує широкий спектр методів моніторингу мережі, які реалізуються IT-фахівцями. Методи розгортаються за допомогою мережевих моніторингових рішень, які автоматично виявляють і реагують на проблеми безпеки і продуктивності. Розглянемо найбільш розповсюджені з них, до яких відносяться: виявлення вторгнень, перехоплення пакетів, сканування вразливостей, брандмауер моніторингу та тестування проникнення.

1) Виявлення вторгнень: виявлення вторгнень контролює локальні мережі для несанкціонованого доступу хакерів. Цей метод може бути реалізований вручну, проте більшість IT-фахівців вважають за краще використовувати програму виявлення вторгнень, яка автоматично виявляє віруси та шкідливе програмне забезпечення, вразливості мережі, такі як бекдори, логічні бомби та інші загрози безпеці, окремі комп'ютери, які підключені до мережі та налаштування файлів.

2) Програми виявлення вторгнень генерують звіти після перевірки системи, тому будь-які проблеми можна вирішити.

3) Перехоплення пакетів: сніфер пакетів – програма, яка перевіряє кожен пакет даних, який проходить через мережу. Метою перегляду пакетів є виявлення неавторизованого програмного забезпечення для моніторингу мережі, яке може бути встановлено хакерами для шпигунства за діловою активністю та інформаційним процесом.

4) Сканування вразливостей: сканер вразливостей періодично сканує комп'ютерну мережу та перевіряє на наявність вразливостей і слабких сторін, які відкривають потенціал для експлуатації. Цей метод відрізняється від виявлення вторгнення, оскільки він виявляє слабкість до того, як атака відбулася. Виявлення вторгнень виявляє несанкціонований доступ після того, як хакер порушив роботу мережі.

5) Брандмауер моніторингу: брандмауери контролюють трафік, що надходить в мережу і виходить з неї. Моніторинг брандмауера відстежує діяльність брандмауера, для забезпечення належного і безпечного процесу перевірки вхідних і вихідних з'єднань.

6) Тестування проникнення: тестування проникнення здійснюється IT-спеціалістами за допомогою методів, які хакери використовують для порушення цілісності мережі. Метою цього процесу є прийняття мережевої безпеки на інший рівень шляхом виявлення вразливостей, про які можуть знати хакери, але які ще не були виявлені за допомогою інших методів моніторингу.

1.1.2. Модель *OSI*

Модель *OSI* стандартизує ключові функції мережі, використовуючи мережеві протоколи. Це дозволяє різним типам пристроїв від різних постачальників спілкуватися один з одним через мережу.

У моделі *OSI* мережеві комунікації згруповані в сім логічних рівнів. Два пристрої використовують стандартизовані протоколи *OSI* на кожному рівні. В

таблиці 1.1 наведені ці рівні та визначено функції, які вони реалізують [1].

Таблиця 1.1

Рівні моделі *OSI*

Рівень	Функція
Рівень 7: Прикладні сервіси	Взаємодіє з програмними додатками, які реалізують комунікаційний компонент
Рівень 6: Представлення даних	Перетворює вхідні та вихідні дані з одного формату презентації в інший (шифрування даних, стиснення тексту)
Рівень 5: Сесійний	Контролює з'єднання між комп'ютерами. Встановлює і припиняє з'єднання, підтримує активним канал протягом сесії
Рівень 4: Транспортний	Забезпечує передачу даних від джерела до хоста призначення через одну або кілька мереж
Рівень 3: Мережний	Маршрутизує пакети даних між двома вузлами в мережі, використовуючи *IP*-адресу
Рівень 2: Передачі даних	Забезпечує надійне з'єднання між двома зв’язаними вузлами шляхом виявлення помилок на фізичному рівні, та передачу даних
Рівень 1: Фізичний	Передає бітовий потік через фізичне середовище, наприклад, через коаксіальний або волоконний кабель

1.1.3. Функції системи моніторингу мережі

Системи моніторингу мережі забезпечують п'ять основних функцій:

- управління конфігурацією мережі;
- обробка помилок;
- аналіз продуктивності;
- управління безпекою;
- облік роботи мережі.

Моніторинг мережі починається з процесу виявлення її складових. Простіше кажучи, якщо не відомо, які модулі підключено до комунікаційного середовища та топологія зв’язків, неможливо контролювати мережу. Системи моніторингу мережі (СММ) виявляють пристрої в мережі – маршрутизатори, комутатори, брандмауери, сервери, принтери та багато іншого.

СММ включають бібліотеку шаблонів моніторингу, яка визначає, як слід контролювати пристрій. У програмі *WhatsUp Gold* ці шаблони називаються ролями пристроїв.

Ролі пристроїв є типовими та специфічними для постачальника. Наприклад, те, що контролюється на маршрутизаторі *Cisco*, відрізнятиметься від того, що контролюється на сервері *Dell*.

Коли система моніторингу мережі конкурує з процесом виявлення, вона автоматично призначає відповідну роль кожному виявленому пристрою.

СММ відрізняються можливостями виявлення. Всі СММ виявляють пристрої в мережі. Однак не всі виявляють, як пристрої підключені до мережі. Наприклад, СММ може ідентифікувати сервер у мережі, але він не матиме інформації про те, до якого вузла він підключений.

Системи моніторингу мережі генерують карти мережі. Мережеві карти є потужним інструментом першої відповіді, який дозволяє адміністраторам мережі візуалізувати свої мережі. Вони забезпечують чисте і впорядковане подання шафи електропроводки. Мережа відображає пристрої відображення та оновлення статусу.

Багато СММ вимагають значного обсягу ручної обробки для створення карти мережі. Деякі лише надають інструмент для її проектування і використовують знання адміністратора мережі для відображення мережі [2].

Системи моніторингу мережі сповіщають адміністраторів мережі про некоректність функціонування. Вони надсилають сповіщення у текстовому повідомленні електронною поштою.

СММ налаштована на оповіщення, коли використання процесора на маршрутизаторі перевищує 80%. Це дає змогу адміністратору мережі активно досліджувати та реагувати, перш ніж маршрутизатор повністю вийде з ладу.

Для збору, та дослідження даних, потрібних для моніторингу комп'ютерної мережі існують окремі протоколи, такі як *Simple Network Management Protocol* (*SNMP*) та *Windows Management Instrumentation* (*WMI*). Вони мають різні реалізації, але обидва мають місце для застосування.

1.1.4. Протокол *SNMP*

SNMP – це стандартний протокол, який збирає дані практично з будь-якого пристрою, підключеного до мережі, зокрема: маршрутизаторів, комутаторів, контролерів бездротової локальної мережі, бездротові точки доступу, сервери, принтери та багато іншого.

SNMP працює, запитуючи «Об'єкти», під якими розуміють модулі, про які СММ збирає інформацію. Наприклад, використання центрального процесору (ЦП) є об'єктом *SNMP*. Запити на об'єкт використання ЦП повертають значення, яке система моніторингу мережі (*Network Monitoring System, NMS*) використовує для оповіщення та звітування.

Об'єкти, запитувані протоколом *SNMP*, зберігаються в базі управляючої інформації (*Management Information Base, MIB*), що визначає всю інформацію, яка піддається впливу за допомогою керованого пристрою. Наприклад, MIB для маршрутизатора *Cisco* буде містити всі об'єкти, визначені *Cisco*, які можуть бути використані для моніторингу такого маршрутизатора, як використання процесора,

використання пам'яті та стан інтерфейсу [3].

Об'єкти в *MIB* каталогізовані з використанням стандартизованої системи числення. Кожен об'єкт має свій унікальний ідентифікатор об'єкта або *OID*.

Деякі *NMS* надають браузер *MIB,* який дозволяє мережевим адміністраторам здійснювати навігацію через *MIB,* для знаходження додаткових об'єктів, які вони хочуть контролювати на мережному пристрої.

1.1.5. Протокол *WMI*

WMI – це протокол для моніторингу серверів і додатків на основі Microsoft Windows. *WMI* є специфічним для Windows і не контролює мережні пристрої або сервери, що не належать до Microsoft.

WMI має велику бібліотеку з тисячами лічильників продуктивності. *WMI* може використовуватися для моніторингу майже всього на сервері Windows, який можна контролювати за допомогою *SNMP*.

Недоліком *WMI*, є більша ресурсомісткість для *NMS*, що споживає більше процесора і пам'яті для обробки, ніж *SNMP*.

Підприємства використовують різні бізнес-додатки, які встановлюються на серверах корпоративної мережі або центрі обробки даних для надання послуг хостам всередині організації. Є також додаткове керування мережею та користувачами, наприклад, *DNS, Active Directory, DHCP* тощо, які надаються з серверів. Крім того, користувачам або клієнтам в організації також потрібна операційна система. Серед численних варіантів, доступних для операційної системи, найчастіше використовуються ОС на базі *Windows*, як для серверних, так і для клієнтських хост-вимог на підприємстві.

Наявність бізнес-додатків на серверах вимагає постійного контролю використання ресурсів, таких як пам'ять, дисковий простір, кеш, процесор і багато іншого. Моніторинг також допомагає визначити можливі проблеми, які впливають на продуктивність сервера. Окрім серверів, клієнтські пристрої також потребують моніторингу для забезпечення безпроблемного користування мережею кінцевому

користувачу.

Системи на базі *Windows* можуть надавати дані системам моніторингу, які потім обробляють і використовують дані для звітування про продуктивність і корисну роботу серверів і робочих станцій. Дані, які використовуються для моніторингу, можуть бути зібрані в операційній системі Windows за допомогою будь-яких доступних методів, описаних нижче.

1.2. Управління якістю передачі інформації

Якість обслуговування *QoS* (*Quality of service*) відноситься до будь-якої технології, яка управляє трафіком даних для зменшення втрати пакетів та затримки в мережі. Якість обслуговування контролює та управляє мережевими ресурсами, встановлюючи пріоритети для конкретних типів даних у мережі.

Корпоративні мережі повинні забезпечувати передбачувані та вимірювані послуги програмам для передачі голосу, відео та даних, що чутливі до затримки. Організації використовують контроль якості обслуговування (КЯО) для задоволення вимог до трафіку чутливих додатків, таких як передача голосових та відео потоків в реальному часі, а також для запобігання погіршенню якості, викликаної втратою пакетів, затримкою та помилками.

Організації для забезпечення КЯО, використовують певні інструменти та методи, такі як формування буфера різниці затримки пакетів. Для багатьох організацій КЯО включено до угоди про рівень обслуговування *SLA* (*Service Level Agreement*) з постачальником послуг мережі, щоб гарантувати певний рівень продуктивності.

1.2.1. Параметри якості обслуговування

Організації можуть кількісно вимірювати *QoS*, використовуючи декілька параметрів, включаючи наступні:

Втрата пакетів відбувається, коли мережні пристрої стають

перевантаженими, а маршрутизатори та комутатори починають відкидати пакети. Коли пакети втрачаються під час спілкування в реальному часі, наприклад, у голосових або відеодзвінках, ці сеанси можуть відчувати різницю в затримці пакетів і прогалини в мові;

– різниця в затримці є результатом перевантаження мережі, дрейфу часу та змін маршруту. Занадто багато різниці в затримці може погіршити якість голосового та відеозв'язку;

– час затримки – це час, протягом якого пакет повинен переміщуватися від джерела до місця призначення. Затримка має бути максимально близькою до нуля. Якщо голосовий виклик через *IP* має високу затримку, він може відчувати відлуння та перекривання звуку;

– пропускна спроможність – це здатність мережевої комунікаційної лінії передавати максимальну кількість даних від однієї точки до іншої за певний проміжок часу. КЯО оптимізує мережу, керуючи пропускною спроможністю та встановлюючи пріоритети для додатків, які потребують більшої швидкості, ніж інші;

– середній бал (*Mean opinion score, MOS*) – це показник якості голосу, який використовує п'ятибальну шкалу, де п'ять вказує на найвищу якість.

1.2.2. Реалізація контролю якості обслуговування

Існують три моделі для реалізації КЯО: негарантована доставка, гарантована доставка (інтегровані послуги, далі ІП) та диференційовані послуги (ДП).

Негарантована доставка (*Best Effort*) – це модель *QoS*, де всі пакети отримують однаковий пріоритет і немає гарантованої доставки пакетів. *Best Effort* застосовується, коли мережі не налаштовували політики КЯО або коли інфраструктура не підтримує КЯО.

Гарантована доставка (ІП) – це модель КЯО, яка резервує пропускну спроможність по певному маршруту в мережі. Програми запитують мережу на резервування ресурсів, а мережеві пристрої стежать за потоком пакетів, щоб

переконатися, що поточне використання мережевих ресурсів дозволяє приймати пакети.

Реалізація ІП вимагає маршрутизаторів, що працюють з ІП і використовує протокол резервування ресурсів *RSVP* (*Resource Reservation Protocol*) для резервування мережевих ресурсів. ІП має обмежену масштабованість і високе споживання мережевих ресурсів.

Диференційовані послуги (ДП) – це модель КЯО, де мережеві елементи, такі як маршрутизатори та комутатори, налаштовані на обслуговування декількох класів трафіку з різними пріоритетами. Мережевий трафік повинен бути розділений на класи на основі вимог компанії.

Наприклад, голосовий трафік може бути визначений більш високим пріоритетом, ніж інші типи трафіку. Пакетам призначаються пріоритети за допомогою кодової точки диференційованих послуг *DSCP* (*Differentiated Services Code Point*) для класифікації. ДП також використовують визначення політики покрокового обслуговування *PHB* (*Per-Hop Behaviour*) для пакета, щоб застосувати методи КЯО, такі як чергування і визначення пріоритетів, до пакетів.

Мережева архітектура також впливає на те, як організація реалізує *QoS*. Багатопротокольна комутація за мітками *MPLS* (*Multiprotocol Label Switching*) включає в себе приватне посилання, яке пропонує комплексний КЯО по одному шляху. *SLA* для *MPLS* визначають смугу пропускання, *QoS*, затримку та тривалість роботи. Однак, *MPLS* може бути дорогим для організацій.

Програмне забезпечення *Software-Defined Wide Area Network* (*SD-WAN*) використовує кілька типів підключень, включаючи *MPLS* і широкосмуговий. *SD-WAN* відстежує стан поточних мережевих з'єднань для вирішення проблем із продуктивністю. Наприклад, якщо втрата пакетів перевищує певний рівень на одному з'єднанні, інструменти *SD-WAN* будуть шукати альтернативне з'єднання.

1.2.3. Механізми контролю якості обслуговування

Деякі механізми КЯО можуть керувати якістю трафіку даних і підтримувати вимоги *QoS*, зазначені в угодах *SLA*. Механізми *QoS* підпадають під конкретні категорії залежно від ролей, які вони відіграють у керуванні мережею. Розглянемо головні інструменти *QoS*.

– інструменти класифікації та маркування розрізняють програми та сортують пакети в різні типи трафіку. Маркування позначить кожен пакет як частину мережевого класу, що дозволяє пристроям у мережі розпізнавати клас пакетів. Класифікація та маркування реалізовані на мережевих пристроях, таких як маршрутизатори, комутатори та точки доступу;

– інструменти керування перевантаженнями використовують класифікацію пакетів і маркування, щоб визначити, в яку чергу записувати пакети.

1.2.4. Засоби моніторингу

Область інструментів моніторингу мережі, програмного забезпечення та постачальників дуже велика. Нове програмне забезпечення, інструменти та утиліти запускаються майже щороку, щоб конкурувати на постійно мінливому ринку IT-моніторингу та моніторингу серверів.

Деякі з функцій, які присутні у інструментів моніторингу – це індикатори *Uptime*/*Downtime*, а також надійні та ретельні системи оповіщення (через електронну пошту/*SMS*), спеціальні шаблони та пороги, інтеграція з *SNMP*, функціональність автоматичного виявлення, зіставлення топології мережі та багато іншого.

Монітор продуктивності мережі SolarWinds – це простий у налаштуванні інструмент, що автоматично виявляє мережеві пристрої і розгортається протягом години. Простий підхід до нагляду за цілою

мережею робить його одним з найпростіших у використанні та найбільш інтуїтивним для користувача.

Продукт легко настроюється і інтерфейс легко змінювати під свої потреби. Можна налаштувати інформаційні панелі, діаграми та перегляди на веб-основі, розробити топологію з урахуванням всієї мережевої інфраструктури, а також створити індивідуальні інтелектуальні сповіщення.

Програмне забезпечення *PRTG Network Monitor* широко відоме своїми передовими можливостями управління інфраструктурою. Всі пристрої, системи, трафік і програми у мережі можуть бути легко відображені в ієрархічному вигляді, який узагальнює продуктивність і оповіщення. *PRTG* відстежує IT-інфраструктуру, використовуючи такі технології, як *SNMP, WMI, SSH*, Потоки/Перехоплення пакетів, *HTTP*-запити, *REST API, Pings*, *SQL* та багато іншого.

Це один з найкращих варіантів для організацій з низьким досвідом моніторингу мережі. Інтерфейс користувача дійсно потужний і дуже простий у використанні.

Особливістю *PRTG* є можливість контролювати пристрої у центрі обробки даних за допомогою мобільного додатка. *QR*-код, що відповідає датчику, роздрукований і приєднаний до фізичного обладнання. Для сканування коду використовується мобільний додаток, а на екрані мобільного телефону відображається коротка інформація про пристрій.

ManageEngine OpManager використовується для управління інфраструктурою, моніторингу мережі та управління продуктивністю додатків *APM* програмного забезпечення.

Продукт добре збалансований, коли мова йде про функції моніторингу та аналізу.

Рішення може керувати мережею, серверами, конфігурацією мережі, несправністю та продуктивністю; Він також може аналізувати мережевий трафік. Для запуску *Manage Engine OpManager* він повинен бути встановлений на робочій станції.

Головною відмінністю цього продукту є те, що він поставляється з попередньо налаштованими шаблонами пристроїв моніторингу мережі. Вони містять попередньо визначені параметри моніторингу та інтервали для конкретних

типів пристроїв.

WhatsUp Gold (*WUG*) – це програмне забезпечення для моніторингу мережі від *Ipswitch*. Це один з найпростіших у використанні і висококонфігурованих інструментів на ринку.

Для щоденного управління IT, *WhatsUp Gold* є функціональним, збалансованим інструментом моніторингу мережі. Він також повністю настроюється. Панелі інструментів можна налаштувати для відображення IT-інфраструктури та оповіщень відповідно до вимог [4].

Таким чином, існуючі засоби моніторингу комп'ютерної мережі дають широкий спектр можливостей адміністраторам та звичайним користувачам. Але для отримання максимального розуміння роботи певної мережі, механізмів, параметрів, потрібні знання протоколів передачі та моніторингу. Інакше все, що на дисплеї – просто числа.

РОЗДІЛ 2
РОЗГЛЯД СИСТЕМ МОНІТОРИНГУ ДЛЯ ОПЕРАТОРІВ ЗВ’ЯЗКУ

2.1. Системи моніторингу

Система моніторингу – група пристроїв та програмне забезпечення, що забезпечує систематичний збір і обробку інформації, яка може бути використана для поліпшення процесу прийняття рішення, а також, побічно, для інформування громадськості або прямо як інструмент зворотного зв'язку з метою здійснення проектів, оцінки програм або вироблення політики. Вона несе одну або більше з трьох організаційних функцій:

– виявляє стан критичних або знаходяться в стані зміни явищ навколишнього середовища, щодо яких буде вироблений курс дій на майбутнє;

– встановлює відносини зі своїм оточенням, забезпечуючи зворотний зв'язок, щодо попередніх успіхів і невдач певної політики або програм;

– встановлює відповідності правилам і контрактним зобов'язанням.

Засоби контролю (моніторингу) дозволяють стежити за процесами, що відбуваються в системі. При цьому можливі два підходи: спостереження в реальному режимі часу або контроль з записом результатів у спеціальному протокольному файлі. Перший підхід зазвичай використовують при вишукуванні шляхів для оптимізації роботи обчислювальної системи та підвищення її ефективності. Другий підхід використовують, коли моніторинг виконується автоматично і (або) дистанційно, про останньому випадку результати моніторингу можна передати віддаленої службі технічної підтримки для встановлення причин конфліктів в роботі програмного і апаратного забезпечення [6].

Моніторинг інженерної інфраструктури ведеться по трьом напрямкам:

– по автономним датчикам (протікання, температурні, руху і т.п.): датчики протікання потрібні завжди, особливо якщо в дата-центрі використовується система охолодження з рідким теплоносієм або фреонова зі зволоженням;

– температурні датчики встановлюємо в холодних і гарячих коридорах машинних залів, в приміщеннях з інженерною інфраструктурою;

– датчики руху, відкриття і закриття дверей стійок.

1) Моніторинг обладнання (кондиціонери, ДБЖ, камери та ін.):

– чи працює обладнання в штатному режимі;

– які помилки виникають в роботі;

– значення окремих параметрів (напруга в ДБЖ, сила струму, температура на вході і виході кондиціонерів та ін.).

2) Моніторинг системи в цілому.

2.2. Типи та топологія систем моніторингу

Система моніторингу може бути визначена як елемент, який реалізується в мережі. Система моніторингу періодично перевіряє доступність і стан кожного вузла і елемента мережі. У разі якщо є якісь проблеми або якщо деякі елементи недоступні то про це автоматично повідомляється відповідальній особі. У деяких випадках можливо активно управляти мережею за допомогою системи моніторингу. Також можливо визначити способи, які будуть використовуватися у випадку якщо критичний вузол недоступний, але це залежить від типу системи моніторингу – в цілому їх можна розділити на три типи. Можуть використовуватися системи моніторингу як додаток на сервері (випадок порівнюваних систем) або як індивідуальний пристрій.

Виділимо наступні типи систем моніторингу:

1) Базові системи моніторингу. Базові системи моніторингу зазвичай працюють з протоколом *ICMP*. Ці системи періодично перевіряють тільки стан

елементів, і вони можуть надавати інформацію про його доступності тільки на доступному/недоступному рівні або вони додають інформацію про час відповіді. Цей тип системи моніторингу підходить тільки для невеликих локальних мереж або для мереж, які не можуть надати більше інформації.

2) Розширені системи моніторингу.

Цей тип моніторингу зазвичай працює з великою кількістю протоколів таких як SNMP, CDP, SSH і так далі. Цей факт дозволяє системам моніторити практично всю інформацію про пристрої в мережі як стан запущених служб, використання системних ресурсів, фактичний потік даних і так далі. З серверами ці системи зазвичай використовують локальні оператори.

3) Системи моніторингу з активним контролем.

Системи моніторингу з активним контролем більш-менш розвинені і може керувати мережевими пристроями. Ці системи дозволяють адміністратору реалізувати автоматичний сценарій, який реагує на зумовлені події і підходять для центрів обробки даних, великих мереж, високодоступних кластерів і так далі.

2.3. Архітектура систем моніторингу та механізми моніторингу

Незважаючи на те, що кожна мережа даних по суті унікальна існує загальна ідея, яка широко застосовується в мережі передачі даних. Архітектури щодо надійності, швидкості і надійності передачі даних. Цю архітектуру найкраще характеризує і описує компанія *Cisco* і її *Enterprise Campus* 3.0 [7]. Принцип цієї архітектури – трирівнева ієрархічна мережа з ядром, розподілом і рівнями доступу, яка дозволяє і полегшує майбутнє зростання мереж і значно полегшує маршрутизацію, адресацію і автономію окремих компонентів і блоків мережі. З точки зору систем моніторингу, одним з найбільш фундаментальних питань є питання їхнього розташування в контрольованої мережі. На думку автора, логічним місцем для його розташування є основний шар, який повинен забезпечити максимальну доступність і в той же час система моніторингу – доступ до всіх шарів елементів мережі.

Коли ви використовуєте систему моніторингу, ви повинні розуміти розмір контрольованої мережі і відповідно до неї повинні вибрати відповідну цьому архітектуру. Взагалі кажучи, на даний час ми використовуємо дві системи систем моніторингу – централізований і об’єднаний.

1) Централізована система моніторингу.

Основна архітектура та впровадження системи моніторингу підходять для невеликих мереж без зовнішніх галузей. Система моніторингу використовується там лише з одним сервером, який контролює всю мережу. Якщо необхідно також контролювати зовнішню гілку, у більшості випадків використовується *VPN*-сайт. Якщо використовується ця архітектура, надзвичайно важливо вибрати відповідне місце системи моніторингу. Логічно, що найкраще місце для його з'єднання в центрі мережі (ядро мережі).

Перевагою цієї архітектури є простота та швидкість реалізації. Найбільш ризикованою і найбільш складною точкою її реалізації є чітко визначена концепція системи моніторингу. Архітектура з централізованою системою моніторингу зображена на рисунку 2.1.

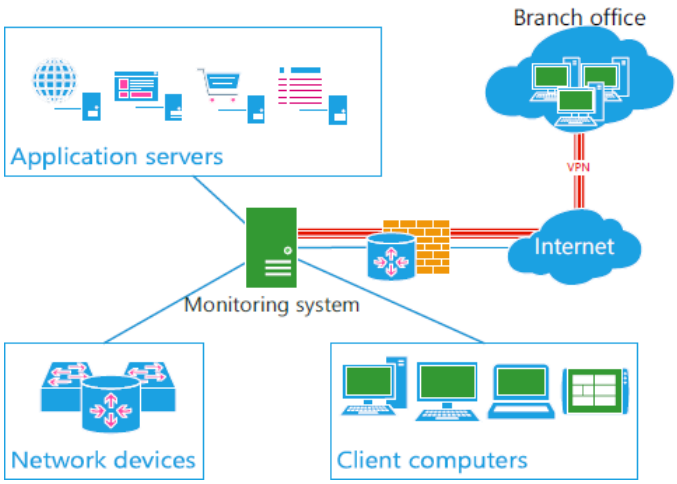

Рис. 2.1. Архітектура з централізованою системою моніторингу

2) Система федеративного моніторингу.

Ця архітектура базується на сегментації відстежуваної мережі до менших частин, які контролюються індивідуальною системою моніторингу. Ці менші

сервери повідомляють повну інформацію про мережу з їх точки зору на центральний сервер моніторингу. Центральний моніторинг на основі інформації від галузевих систем може точно звітувати про постраждалий сегмент мережі. У разі виходу з ладу основної системи моніторингу все ще можливо отримати дані та інформацію від галузевих систем. Цей тип архітектури підходить особливо для розгалужених мереж, або, з іншого боку, для постачальників, які можуть використовувати його для спостереження за своїми клієнтами мереж та виходів з основної системи моніторингу, а потім представляти їх в центр моніторингу. Архітектура з федеративною системою моніторингу зображена на рисунку 2.2.

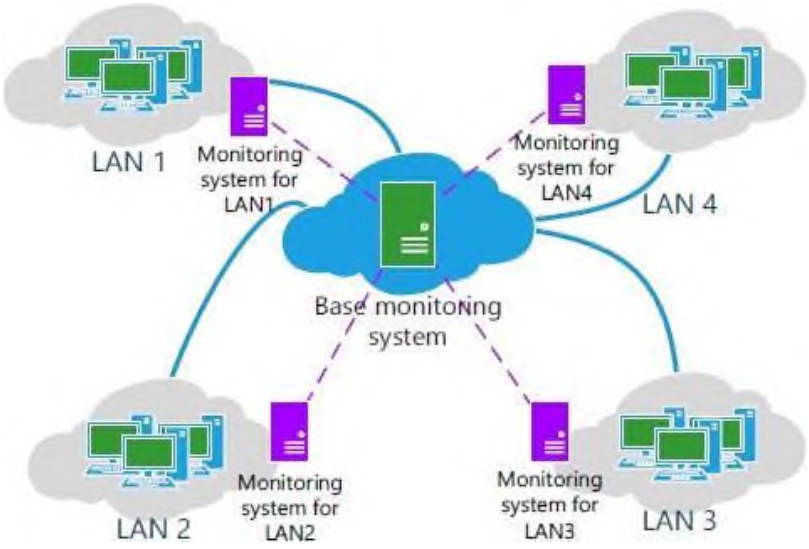

Рис. 2.2. Архітектура з федеративною системою моніторингу

Системи моніторингу для збору інформації з контрольованих пристроїв використовують три методи. Перший метод перевіряє пристроїв з використанням стандартних протоколів, таких як *ICMP, TCP, SNMP* і так далі. Такий спосіб перевірки в більшості випадків не потребує будь-якого великого втручання до конфігурації за винятком дозволу виключень в брандмауері. Це також універсальний метод отримання інформації незалежно від виробників та типу пристрою. Недоліком є певне узагальнення та недоступність деяких спеціалізованих функцій.

Другий варіант – використання *SNMP*-повідомлення під назвою *SNMP*

пастка. На противагу першому варіанту немає періодичної перевірки з боку системи моніторингу, але контрольований елемент в випадку аварії (помилка, збій порту, *RAM* заповнений, і так далі) відправить звіт в систему моніторингу, на основі отриманої інформації. Цей спосіб перевірки у більшості випадків поєднуються з періодичної перевіркою [8].

Третій варіант, як отримати дані системи моніторингу – використання спеціального агента для моніторингу пристрою. Агент працює в цільовій системі як додаток, і він взаємодіє з системою моніторингу за принципом клієнт-сервер. Перевага в тому, що ви не повинні використовувати спеціальний мережевий протокол, оскільки ці дані переважно передаються через *TCP/IP*. Головною перевагою є можливість отримати детальну інформацію про моніторингову систему, а в разі активної системи моніторингу також є можливість керувати станцією, що підлягає моніторингу. Сервери зазвичай контролюються таким способом.

2.4. Порівняльні характеристики систем моніторингу

Якщо ви хочете об'єктивно протестувати та порівняти обрані системи моніторингу, необхідно встановити критерії оцінки та методи оцінки даних. У реальному застосуванні, весь процес повинен складатися з кількох етапів: визначення вимог, дослідження ринку для потенційно придатних систем, застосування методології, реалізація переможця.

Виділимо наступні критерії оцінки:

– ціна. У той момент, коли ми будемо приймати рішення про те, яка система моніторингу повинна бути використана, її ціна завжди буде одним із найважливіших критеріїв. На жаль, ціна занадто завищена, і переважна більшість компаній не зможе правильно аналізувати цей параметр. Ціну слід завжди аналізувати пропорційно до того, що система принесе нам, і де це дозволить заощадити нашу трудомісткість або оптимізувати процеси;

– системні вимоги. Критерій, який оцінює мінімальні вимоги до запуску

системи як з точки зору навантаження, так і додаткових програм;

– користувацький інтерфейс. Ймовірно, найменший предикативний критерій. В основному це пов'язано з тим, що інтерфейс вигляду та роботи з користувачами настільки суб'єктивний, що практично неможливо отримати об'єктивну оцінку;

– труднощі виконання. Критерій оцінки складності установки та базової конфігурації системи моніторингу;

– швидкість відповіді на аварію. Критерій, який оцінює здатність системи реагувати на провал будь-якого елемента в мережі;

– можливість ідентифікації постраждалого сегмента. Оцінка особливо з точки зору можливості визначити ієрархію моніторингової мережі та наслідком здатність виявляти уражений сегмент у разі невдачі;

– автоматичний пошук. Функція, яка автоматично сканує область моніторингу та виконує пошук вузлів мережі для відстеження. Системи моніторингу для цього відстеження використовують протоколи *ICMP* та *SNMP*. Деякі системи потім використовують також утиліту *traceroute* для виявлення шляху передачі знайдених вузлів. З точки зору цього критерію оцінюється, зокрема, якість сканування, швидкість і визнання фактичної топології мережі;

– методи повідомлення. Оцінка з точки зору інформації про ситуацію в мережі надсилається адміністратору. Коли ми забиваємо кожну точку, надану системі, це наявна технологія. Системні типи віддають перевагу електронній пошті або *SMS*. Деякі з них також додають підтримку різних мереж зв'язку, таких як *Jabber* і так далі;

– додаткові функції. Критерій оцінки додаткових функцій, які не є частиною стандарту, на всіх випробуваних системах моніторингу;

– співпраця з іншими системами. Це вивчення сумісності між системами моніторингу та методом, якщо такі існують, як можна підключити системи або як їх об'єднати;

– глибина моніторингу. Вона оцінює вивчену систему з точки зору наявної

інформації про пристрої, що підлягають моніторингу.

2.5. Обґрунтування вибору методів реалізації

Сирі метрики, коли мова йде про моніторинг, це лише цифри, які нічого не повідомляють. Деяке число може здатися високим, але не призводить до виникнення будь-яких проблем, які можуть бути видимі для кінцевих користувачів. Інше число може здатися низьким, але не настільки низьким, як вимагають SLA. Єдиний спосіб зробити значущі метрики – розмістити їх у контексті того, як використовується програма та що очікують користувачі.

Всі ці фактори означають, що не можна дивитися на статистику ефективності з чисто технічної точки зору. Єдиний спосіб зрозуміти продуктивність – подивитися показники продуктивності з точки зору користувача. Це вірно навіть тоді, коли у вас є видимість в мережі та програмі. Мета моніторингу не повинна полягати в підвищенні показників за метриками.

У даному проекті обрано наступні основні метрики моніторингу мережі з боку користувача:

– пропускна спроможність (*bandwidth*) – метрика, що напряму показує швидкість з якою в даній мережі можуть відправлятися та отримуватися дані (в мегабайтах за секунду);

– пакетна пропускна спроможність (*PPS*) – метрика, що показує швидкість з якою пакети можуть проходити через мережу. Виміряється в пакетах за секунду;

– показник втрати пакетів (*frame loss*) – якщо неможливо відновити фрейм, або він не відповідає контрольній сумі, то він може бути відкинутий. Показник показує кількість втрачених пакетів та відсоток їх кількості;

– час життя пакету *TTL* (*Time To Live*) – через скільки маршрутизаторів пакет може бути доставлений з однієї точки до іншої;

– заголовки пакетів – потрібні для ручного контролю та дозволяють досвідченим користувачам отримувати повну інформації про пакети, що проходять через комп’ютер.

2.5.1. Обґрунтування пропускної спроможності

Пропускна спроможність – це максимальна швидкість, з якою можна отримувати, та відправляти дані в Інтернеті. Чим вона більша, тим більше даних можна отримати за деякий проміжок часу. В повсякденності вона виміряється в мегабітах за секунду. Слідує звернути увагу на те, що один мегабайт це вісім мегабіт. При швидкості в один мегабіт на секунду для завантаження файлу розміром 1 мегабайт потрібно вісім секунд.

2.5.2. Обґрунтування пакетної пропускної спроможності

У доповнення до пропускної спроможності метрика, така як пакетна пропускна спроможність, також вельми важлива для повного розуміння продуктивності мережі. Інтерфейси мережевого пристрою працюють з лінійною швидкістю, коли пристрій здатний пересилати пакети незалежно від розміру. Таким чином, навіть для самих маленьких пакетів (найвища швидкість передачі пакетів) мережевий пристрій буде виконувати свої функції. Наприклад, оскільки один з новітніх маршрутизаторів *Cisco ASR*, здатний пересилати пакети зі швидкістю до 16 *Мбіт/с* з включеними послугами, він може підтримувати обробку еквівалента трафіку 10 *Гбіт/с* з лінійною швидкістю, навіть для невеликих пакетів [5].
Для маршрутизаторів пакетна пропускна спроможність є одним з найбільш важливих параметрів, але для інших пристроїв необхідно враховувати і інші метрики, такі як брандмауери, балансування навантаження, системи запобігання вторгнень, пристрої трансляції *NAT* (*Network Address Translation*).

2.5.3. Обґрунтування показника втрати пакетів

Втрата пакету – це збій передачі одного або декількох пакетів в місці призначення. Ця подія може викликати помітні зміни у всіх типах цифрових

комунікацій.

Наслідки втрати пакетів:

– часткова втрата даних, що призводить до помилок у завантажених файлах;

– у відеоконференції, це може створити тремтіння картинки, через те, що для відтворення відео використовуються дані про зміни картинки, відносно попереднього кадру. Приклад на рисунку 2.1;

– у аудіозв’язку, це може викликати тремтіння і часті пропуски в прийнятій мові;

– у гірших випадках втрата пакета може привести до серйозного пошкодження отриманих даних, втрати більшої частини зображень, нерозбірливої мови або навіть повної відсутності прийнятого сигналу.

Рис. 2.1. Наслідок втрати пакету з даними для відтворення відео

Причини втрати пакетів включають в себе недостатню потужність сигналу в пункті призначення, природні або штучні перешкоди, надмірний шум системи, апаратний збій, пошкодження програмного забезпечення або перевантажені вузли мережі. Часто більш ніж один з цих факторів впливає на появу втрати пакетів.

У разі, коли причина не може бути усунена, може використовуватися маскування втрати пакета, щоб мінімізувати вплив втрачених пакетів.

2.5.4. Обґрунтування часу життя пакету (*TTL*)

Час життя (*TTL*) – це, в основному, кількість маршрутизаторів, які проходить пакет, перш ніж він буде відкинутий. Певні номера *TTL* вказують максимальний діапазон який може пройти пакет.

Початкове значення *TTL* задається при відправці хостом як поле з восьми двійкових цифр заголовка пакета. Поле *TTL* встановлюється відправником дейтаграми і зменшується кожним маршрутизатором на маршруті аж до місця призначення. При пересиланні *IP*-пакетів маршрутизатор зменшує значення *TTL* мінімум на 1. Коли значення *TTL* пакета досягає 0, маршрутизатор відкидає його і відправляє повідомлення з використанням протоколу *ICMP* назад на хост.

2.6. Мова програмування

Правильно вибрати мову програмування для конкретної задачі – це важлива частина роботи. Для реалізації даних можливостей була обрана мова *Python*. Вона дозволяє досить легко та елегантно створювати об’єктно орієнтовані програми та має ряд корисних внутрішніх модулів, що підходять для виконання поставленої задачі.

Модуль *socket*.

Сокетне програмування – це спосіб з’єднання двох вузлів в мережі один з одним. Один сокет (вузол) слухає конкретний порт з *IP*-адреси, а інший сокет звертається до іншого, щоб сформувати з'єднання. Сервер формує сокет слухача, коли клієнт звертається до сервера. Вони являють собою реальні основи для перегляду веб-сторінок. Говорячи коротко є сервер і клієнт.

Програмування сокета починається шляхом імпортування бібліотеки сокетів і створення простого сокета:

s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)

Тут створюється екземпляр сокету та в нього передаються два параметри. Перший – *AF_INET*, відноситься до сімейства адрес. Другий – *SOCK_STREAM*

значить орієнтований на з'єднання протокол *TCP*.

Тепер для прикладу можна підключитися до серверу за допомогою цього сокету. Треба звернути увагу, що якщо при створенні сокету виникає яка-небудь помилка – то генерується помилка *socket.error* та нам не вдасться підключитися до серверу. Для виправлення помилки нам потрібно буде знайти *IP*:

```
ip = socket.gethostbutename('www.google.com')
```

Приклад підключення до *Google*:

```
import socket
import sys
try:
        s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
        print «Сокет успішно створений»
except socket.error as err:
        print «Створення сокету завершилось з помилкою %s» %(err)
port=80
try:
        host_ip = socket.gethostbutename('www.google.com')
except socket.gaierror:
        print «Помилка хоста»
        sys.exit()
#Підключення до серверу
s.connect((host_ip, port))
print «Сокет успішно приєднано до Google»
on port == %s" %(host_ip)
```

Також модуль *socket* дозволяє створювати програми типу клієнт- сервер. Сервер має метод *bind*(), що зв'язує його з *IP* та портом, щоб він міг прослуховувати вхідні запроси. Також метод *listen*(), що переводить сервер в режим прослуховування. Також методи *accept*() для з'єднання з клієнтом та *close*() для закриття з'єднання.

Модуль *socket* в даному проекті потрібен для:

– передачі даних між процесами;

– отримання потоку даних, що проходять через мережу.

Модуль *PyQt5*.

PyQt5 це бібліотека, що дозволяє використовувати каркаси *Qt GUI* у мові програмування *Python*. Так як *Qt* написаний на мові *C++*, то швидкість створеного додатку буде достатньо високою, а написання мовою *Python* легше та швидше ніж на *C++*.

Модуль дозволяє створювати графічний інтерфейс з певними віджетами. Усі елементи, що бачить користувач у вікні є віджетами. Вони можуть бути вкладеними.

Найбільш популярні віджети:

– *QLabel* – мітка з текстом;

– *QComboBox* – для вибору одного з випадаючого списку (рис. 2.2);

Рис. 2.2. *QcomboBox*

– *QCheckBox* – для встановлення «галочки» на пункті (рис. 2.3);

Рис. 2.3. *QCheckBox*

– *QRadioButton* – вибір одного варіанту з декількох (рис. 2.4);

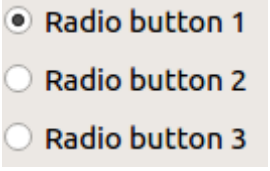

Рис. 2.4. *QRadioButton*

– *QPushButton* – кнопка для натискання (рис. 2.5);

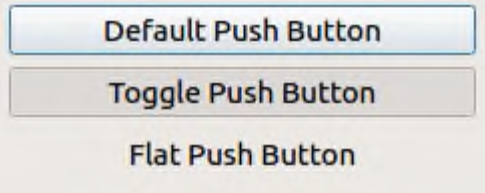

Рис. 2.5. *QPushButton*

– *QTableWidget* – таблиця зі значеннями (рис. 2.6);

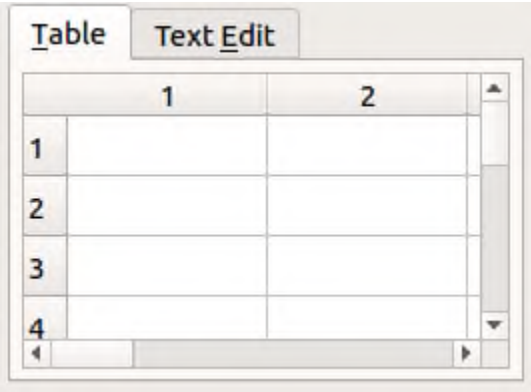

Рис. 2.6. *QTableWidget*

– *QLineEdit* – для вводу значень з клавіатури користувачем (рис. 2.7);

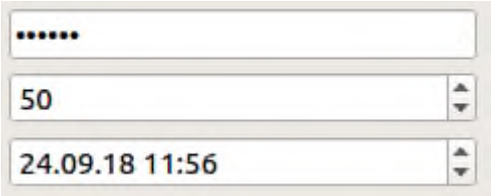

Рис. 2.7 *QLineEdit*

– *QSlider* – вибір приблизного значення переміщенням повзунка (рис. 2.8);
– *QProgressBar* – для виводу прогресу виконання (рис. 2.9).

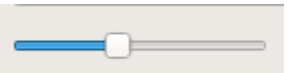

Рис. 2.8. *QSlider*

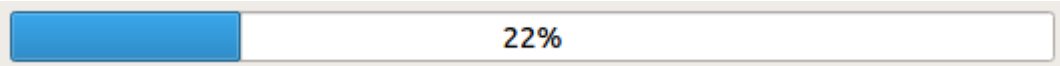

Рис. 2.9. *QProgressBar*

Одна з сильних сторін *Qt* це підтримка стилів написаних творцем програми. Для використання специфічного стилю, треба використовувати *setStyle*(...). Доступні стилі залежать від платформи, найпопулярніші це «*Fusion*», «*Windows*», «*WindowsVista*»(тільки для *Windows*) і «*Macintosh*»(тільки для *Mac*).

Сигнали та слоти.

Qt використовує механізми, що називаються сигналами, щоб дати можливість реагувати на події, такі як натискання кнопки, вибір пункту меню, встановлення «галочки» та інші.

Для прив’язки віджету до функції використовується метод *connect*(), що викликається для подій, що зарезервовані для кожного елементу.

Приклад прив’язки натискання кнопки до функції:

```
app = QApplication([])
button = QPushButton('Click')
def on_button_clicked():
    alert = QMessageBox()
    alert.setText(«Ви натиснули кнопку!»)
    alert.exex_()
button.clicked.connect(on_button_clicked)
button.show()
app.exec_()
```

Button.clicked це так званий сигнал, *a.connect()* дозволяє встановити слот, що є простою функцією, що викликається при появі сигналу. Результат можна бачити на рис. 2.10.

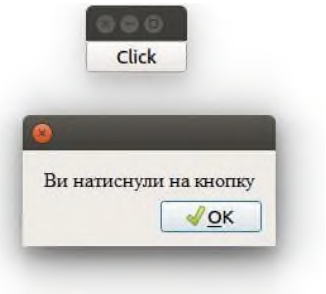

Рис. 2.10. Результат прив’язки функції до кнопки

2.7. Система моніторингу з використанням *Raspberry Pi*, датчиками та протоколом *ZIGBEE*

Сучасні системи моніторингу, що використовують одноплатні комп’ютери в якості базових вузлів, використовують протокол *ZigBee*. Кожен сенсор мережі під’єднується до мікроконтролера, що підключається до модуля зв’язку *XBee*. Центральний вузол системи – це одноплатний комп’ютер, обладнаний аналогічним модулем. Живлення кожного вузла відбувається за рахунок акумулятора. На центральному вузлі встановлюється веб-додаток, головним завданням якого є обробка даних, отриманих від вузлів з сенсорами та представлення даних кінцевому користувачеві.

Протокол *ZigBee* передбачає наявність головного вузла, що управляє топологією мережі, його функції бере на себе одноплатний комп’ютер.

Архітектура мережі зображена на рисунку 3.2.

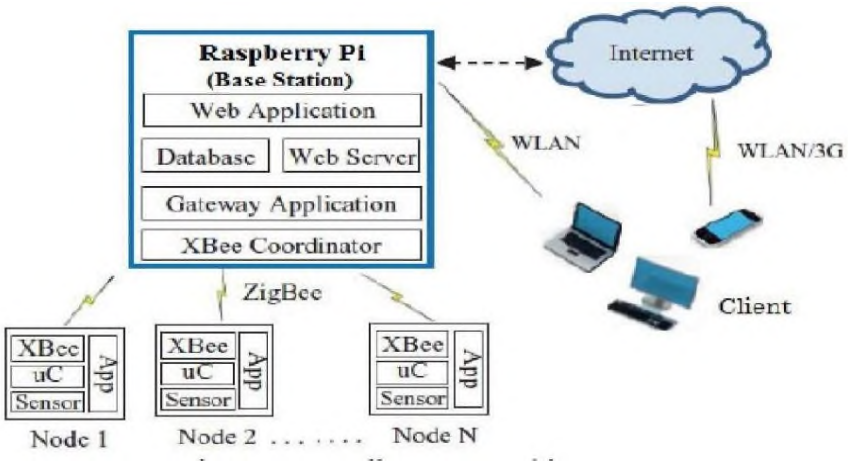

Рис. 3.2. Архітектура системи моніторингу на основі одноплатних комп’ютерів

У якості дочірніх вузлів мережі використовуються мікроконтролери компанії *Arduino* – торгова марка апаратно-програмних засобів для побудови простих систем автоматики і робототехніки, орієнтована на непрофесійних користувачів. Програмна частина складається з безкоштовної програмної оболонки (*IDE*) для написання програм, їх компіляції та програмування апаратури. Апаратна частина являє собою набір змонтованих друкованих плат, що продаються як офіційним виробником, так і сторонніми виробниками[12]. Повністю відкрита архітектура системи дозволяє вільно копіювати або доповнювати лінійку продукції *Arduino*. Блок-схема центрального і дочірнього вузлів системи зображена на рисунку 3.3.

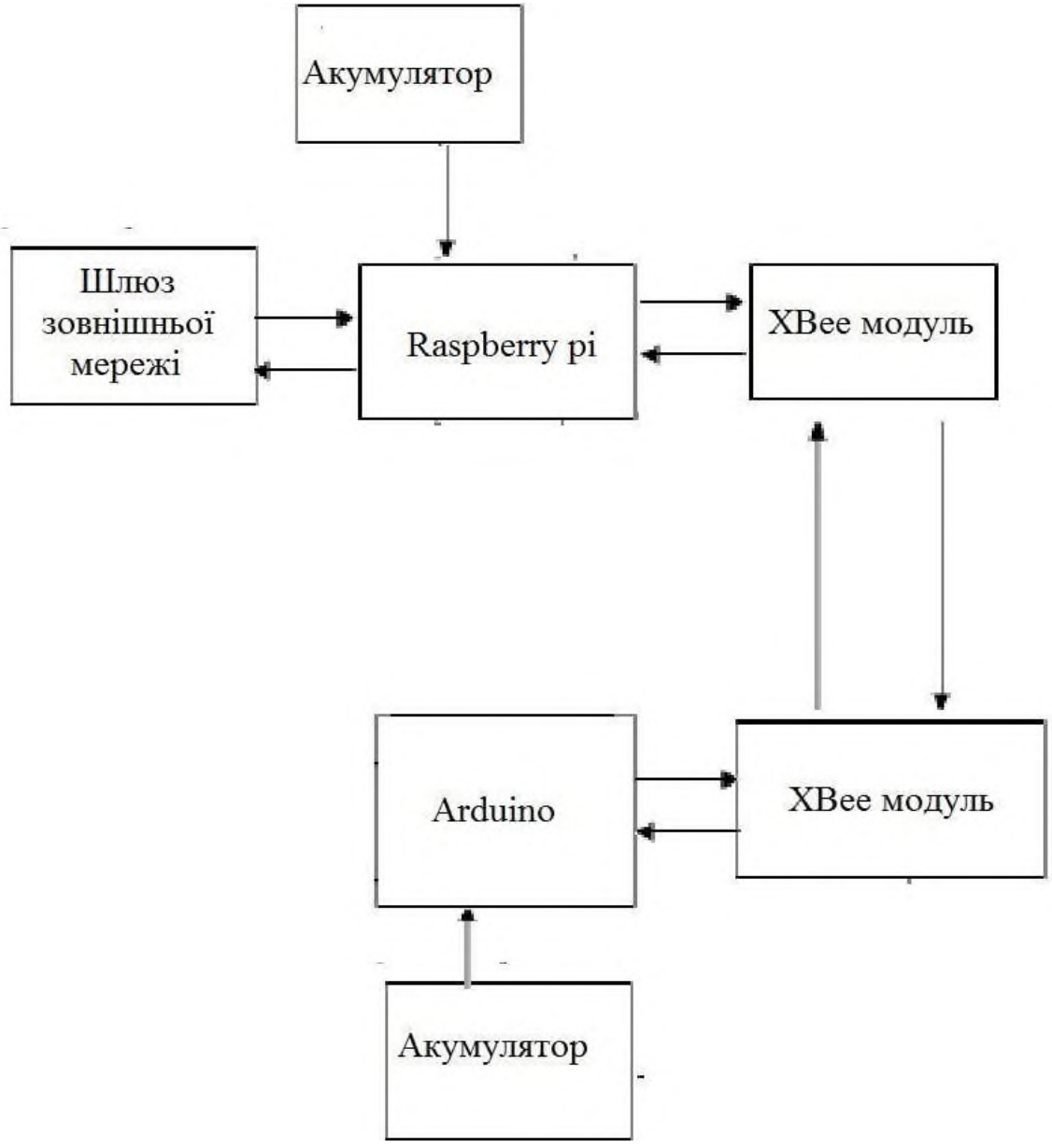

Рис. 3.3 Блок-схема центрального і дочірнього вузлів системи

Переваги:

– низькі енергетичні затрати;

– попередня обробка даних;

– можливість використання різних мереж передачі даних до кінцевого користувача;

– простота зміни конфігурацій топології системи;

– низька вартість системи.

Недоліки:

– низька швидкість передачі даних;

– необхідність періодичної заміни акумуляторів;

– необхідність використання додаткового обладнання (модулі *XBee*).

РОЗДІЛ 3
РОЗРОБКА КОМПОНЕНТІВ СИСТЕМИ

3.1. Загальна структура системи

Створений програмний засіб має виконувати розрахунки лабораторної роботи на тему «Рух тіла по похилій площині», а саме – знаходити значення прискорення тіла. Згідно технічного завдання, потрібно реалізувати можливість відображення проміжних результатів розрахунків та наявність теоретичного матеріалу щодо виконання лабораторної роботи. Інтерфейс програми має бути простим та максимально зрозумілим. Також планується розробити можливість ведення історії розрахунків за сеанс.

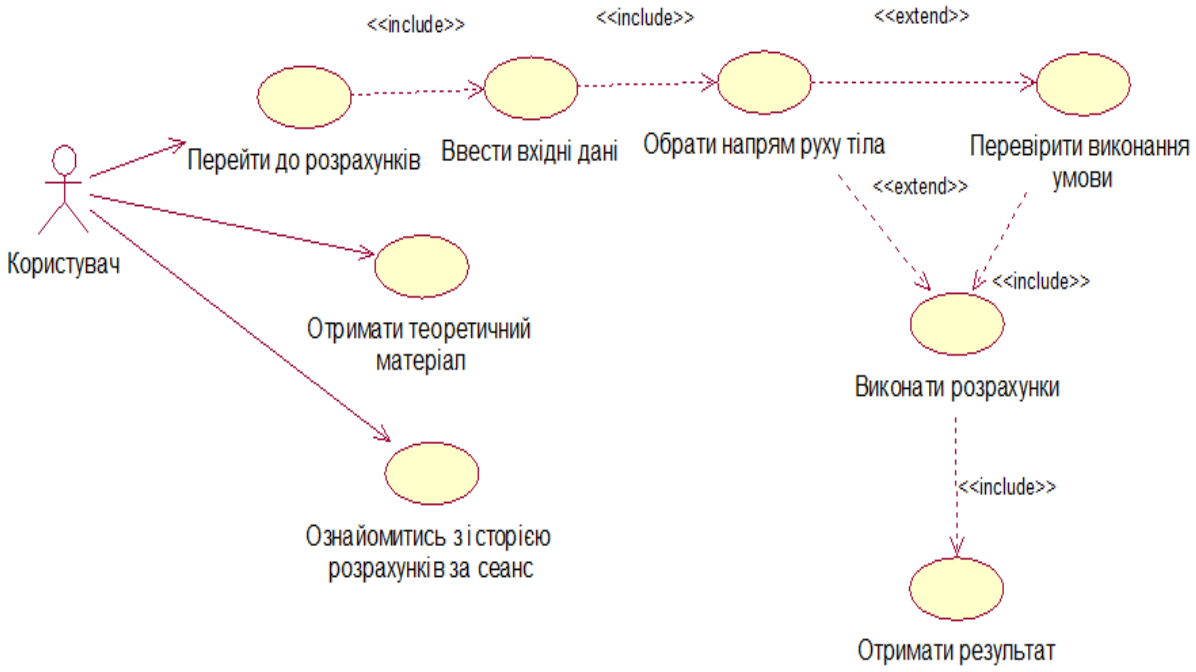

Рисунок3.1 – Діаграма варіантів використання

Розроблена програма може рахувати прискорення при русі тіла по похилій площині вниз та вгору. Якщо тіло рухається вниз, то умови для перевірки не потрібно, адже за умовою сила тяги не може завадити тілу рухатись вниз. Алгоритм розрахунку прискорення наведено в додатку А.

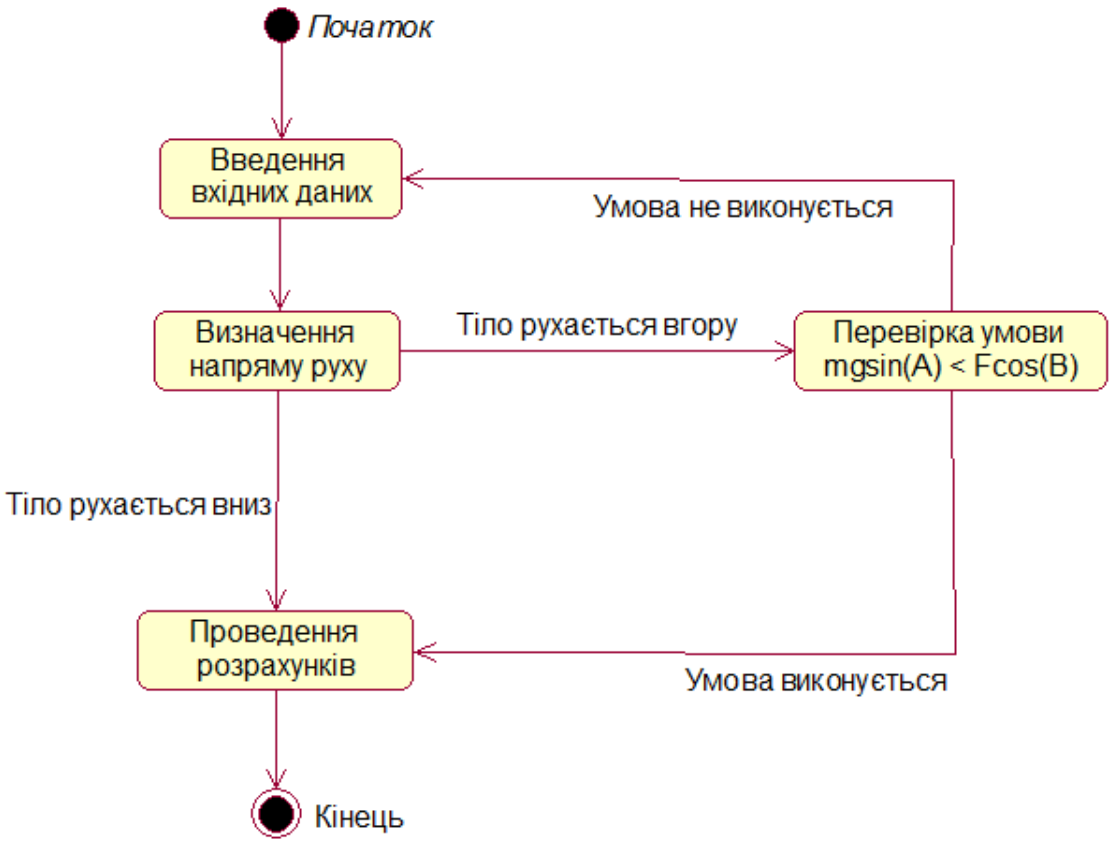

Рисунок 3.2 – Діаграма станів

Згідно рис. 2.2, розрахунок прискорення при русі тіла вгору не буде виконано, доки користувач не введе коректні дані. Також слід враховувати, що користувач може ввести значення, які будуть більше або менше ніж можливі. Тому перед початком розрахунків програма повинна перевірити правильність даних і лише тоді розпочати обчислення, якщо виконуються необхідні для розрахунків умови.

Для визначення дій користувача при русі тіла вниз було побудовано наступну діаграму послідовності:

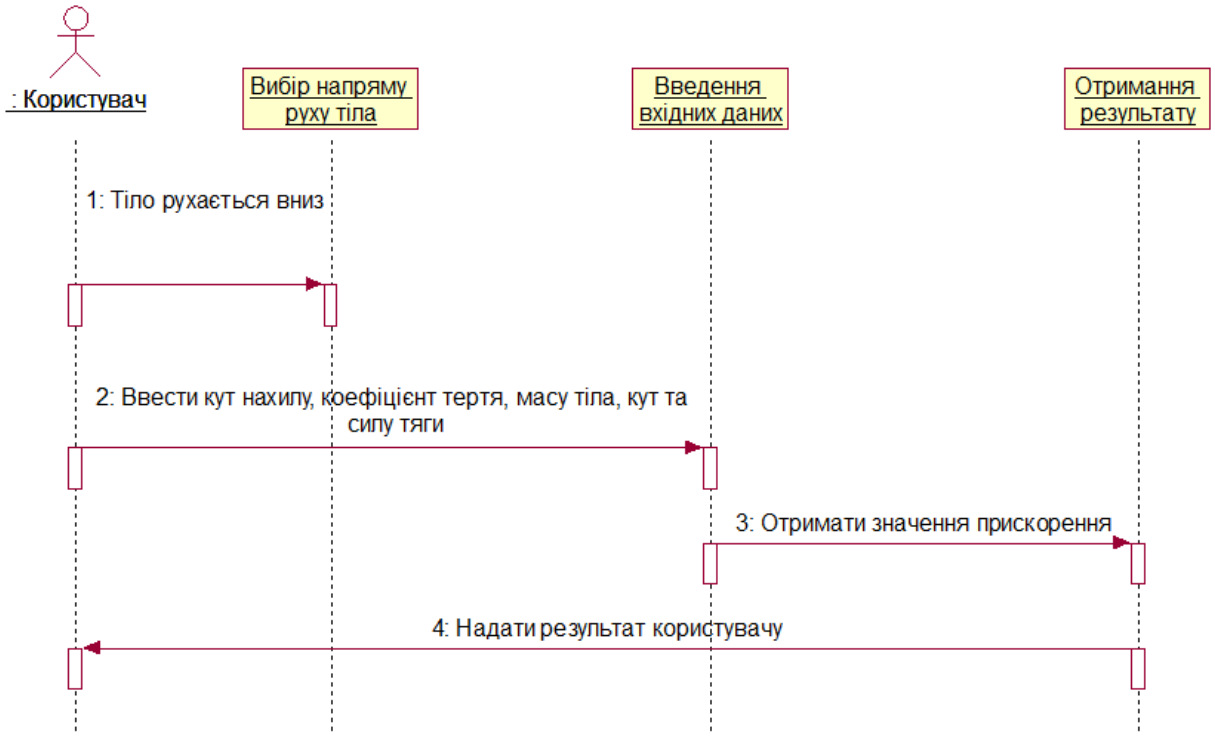

Рисунок 3.3 – Діаграма послідовності

При русі тіла вгору спочатку потрібно перевірити чи вистачить сили тяги для того, щоб тіло рухалось в потрібному напрямі. Якщо вхідні дані задовольняють умову, то діаграма послідовності буде виглядати так:

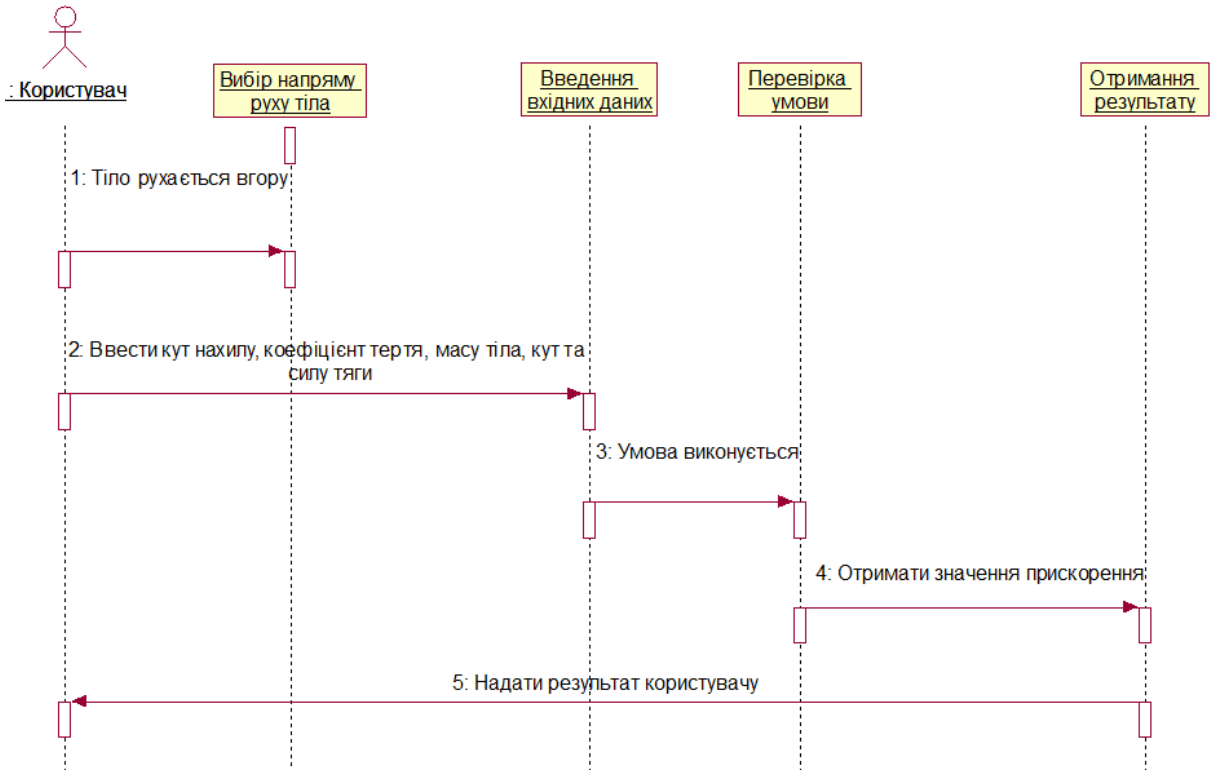

Рисунок 3.4 – Діаграма послідовності

3.2 Опис засобів реалізації

Перед початком розрахунків мають бути перевірені всі вхідні значення, що знаходяться в елементах керування NumericUpDown. Якщо значення не задовольняє таку властивість, як Minimum або Maximum, то це значення повинно бути встановлено рівним мінімальному або максимальному відповідно:

```
if (angleUpDown.Value > angleUpDown.Maximum)
    angleUpDown.Value = angleUpDown.Maximum;
else if (angleUpDown.Value < angleUpDown.Minimum)
    angleUpDown.Value = angleUpDown.Minimum;
```

Перевірка була виконана за допомогою використання умовних операторів «if-else». Окрім операторів «if-else» в програмі також застосовано конструкції «switch». За їх допомогою відбувається перемикання між кроками, щоб користувач міг отримувати покрокові розрахунки.

Для виводу повідомлень про помилки використовується метод Show(String, String, MessageBoxButtons, MessageBoxIcon) класу MessageBox, який виводить вікно повідомлення з заданим текстом, заголовком, кнопками і значком:

```
MessageBox.Show("Прискорення дорівнює нулю, рух вгору не відбувається.\nДля розрахунку необхідно зменшити коефіцієнт тертя μ або збільшити силу тяги F", "Повідомлення", MessageBoxButtons.OK, MessageBoxIcon.Information);
```

Для виконання різних математичних операцій в бібліотеці класів .NET призначений клас Math. Він є статичним, тому всі його методи також є статичними. Для його підключення потрібно прописати на початку using System;

Обчислення синуса і косинуса відбувається в радіанах, тому у випадку коли користувачу потрібні значення в градусах, потрібно зробити конвертацію:

```
radianA = Convert.ToDouble(angleUpDown.Value);
degreeA = radianA * Math.PI / 180 ;
radianB = Convert.ToDouble(BetaUpDown.Value);
degreeB = radianB * Math.PI / 180;
```

Тут використовуються методи класу Convert та константа PI класу Math. Клас Convert потрібен для того, щоб перетворювати значення одного базового типу даних до іншого базового типу даних (наприклад ToString(Double) перетворює

значення заданого числа з плаваючою крапкою подвійної точності в еквівалентне строкове представлення).

Окрім константи PI в програмі було використано методи Sin, Cos, Round класу Math. Всі три методи використано при знаходженні прискорення тіла при русі вниз та вгору:

```
a = Math.Round((-Convert.ToDouble(massUpDown.Value) * g * (Math.Sin(degreeA) - Convert.ToDouble(coeffUpDown.Value) * Math.Cos(degreeA)) - Convert.ToDouble(tracForce.Value) * (Convert.ToDouble(coeffUpDown.Value) * Math.Sin(degreeB) + Math.Cos(degreeB))) / Convert.ToDouble(massUpDown.Value), 2);
```

```
a = Math.Round((-Convert.ToDouble(massUpDown.Value) * g * (Math.Sin(degreeA) + Convert.ToDouble(coeffUpDown.Value) * Math.Cos(degreeA)) + Convert.ToDouble(tracForce.Value) * (Convert.ToDouble(coeffUpDown.Value) * Math.Sin(degreeB) + Math.Cos(degreeB))) / Convert.ToDouble(massUpDown.Value), 2);
```

У вищенаведеному коді всі значення, що не мали формат з плаваючою крапкою Double, спочатку конвертуються до нього за допомогою методу ToDouble(value) класу Convert, далі відбуваються арифметичні операції з використанням методів Sin та Cos класу Math, після чого отримане значення округлюється до двох знаків після коми за допомогою методу Round(double value, int digits) класу Math.

Замість звичних елементів керування TextBox було використано елементи керування RichTextBox, які призначені для редагування тексту з додатковими можливостями форматування. Зміна формату забезпечується властивостями Selection. Оскільки TextBox не підтримує верхні та нижні індекси, довелося використати RichTextBox. Наприклад, щоб вивести речення, де є позначення сили тертя $F_{тр}$ з нижнім індексом, було використано приблизно такий код:

```
resultsBox.SelectedText = "\nКрок 1. Запишемо рівняння динаміки руху тіла:\nmg + N + F + F";
resultsBox.SelectionCharOffset = -5;
resultsBox.SelectedText = "тр";
resultsBox.SelectionCharOffset = 0;
resultsBox.SelectedText = "= ma (1)";
```

Тут в RichTextBox під назвою resultsBox спочатку записується частина, що не містить нижнього індекса сили тертя, далі змінюється властивість SelectionCharOffset на від’ємне число (від значення залежить розташування індексу, якщо значення було б додатнім, то індекс був би верхнім), записується індекс, після чого значення SelectionCharOffset повертається на звичне і дописується решта тексту.

Для ведення історії розрахунків була реалізована робота з файлами. Файл – це набір даних, який зберігається на зовнішньому запам’ятовуючому пристрої (наприклад на жорсткому диску). Файл має ім'я і розширення. Розширення дозволяє ідентифікувати, які дані і в якому форматі зберігаються у файлі. В C# є простір імен System.IO, в якому реалізовані всі необхідні класи для роботи з файлами. Щоб підключити цей простір імен, необхідно на початку програми додати рядок using System.IO.

Для створення пустого файлу, в класі File є метод Create(). Він приймає один аргумент – шлях. Наприклад:

```
File.Create("D:\\new_file.txt");
```

Якщо файл з таким ім'ям вже існує, то він буде переписаний на новий порожній файл.

Метод WriteAllText() створює новий файл (якщо такого немає) або відкриває існуючий і записує текст, замінюючи все, що було в файлі:

```
File. WriteAllText("D:\\new_file.txt", "текст");
```

Метод AppendAllText () працює, як і метод WriteAllText () за винятком того, що новий текст дописується в кінець файлу, а не переписує все що було в файлі:

```
File.AppendAllText ("D: \\ new_file.txt", "текст методу AppendAllText()");
```

Метод Delete() видаляє файл за вказаним шляхом:

```
File.Delete ("d: \\ test.txt");
```

Крім того, щоб читати або записувати дані в файл можна використовувати потоки.

В програмі використані методи WriteAllText та AppendAllText класу File. Спочатку за допомогою методу WriteAllText() очищається файл(якщо він існує) чи

створюється(якщо файл не знайдено), після чого за допомогою властивості DateTime.Today визначається та записується дата. Це реалізовано в користувацькому методі ClearFile():

```
void ClearFile()
    {
        DateTime date = DateTime.Today;
        int day = date.Day;
        int month = date.Month;
        int year = date.Year;
        string data = day.ToString("00") + "." + month.ToString("00") + "." + year.ToString("00");
        File.WriteAllText(Environment.CurrentDirectory + @"\History.txt", data + "\n");
    }
```

Запис в файл відбувається за допомогою методу AppendAllText() кожен раз при натисканні на кнопку початку розрахунків. Приклад додавання до історії задачі на рух тіла вниз:

```
File.AppendAllText(Environment.CurrentDirectory + @"\History.txt", Environment.NewLine + Environment.NewLine + counter + ".Тіло масою " + massUpDown.Value.ToString() + " кг рухається вниз. Кут нахилу дорівнює " + angleUpDown.Value.ToString() + "°. Коефіцієнт тертя між тілом та похилою площиною дорівнює " + coeffUpDown.Value.ToString() + ". Також на тіло діє сила тяги " + tracForce.Value.ToString() + " Н під кутом " + BetaUpDown.Value.ToString() + "°.\n Прискорення дорівнює " + a + " м/с");
```

3.3 Порівняльний аналіз реалізованого програмного засобу та програм-аналогів

Створений програмний засіб має обмаль програм-аналогів на ринку. Найближчі з них – «Наклонная плоскость 1.0», «MSC Adams», «Comparison of forces».

«Наклонная плоскость 1.0» – це віртуальна 3D лабораторна робота з фізики, що імітує виконання лабораторної роботи з фізики на тему «Рух тіла по похилій площині», вона допомагає визначити сили, що діють на похилій площині. Тобто

користувач може побачити залежність руху тіла від сили тяги та кута нахилу площини.

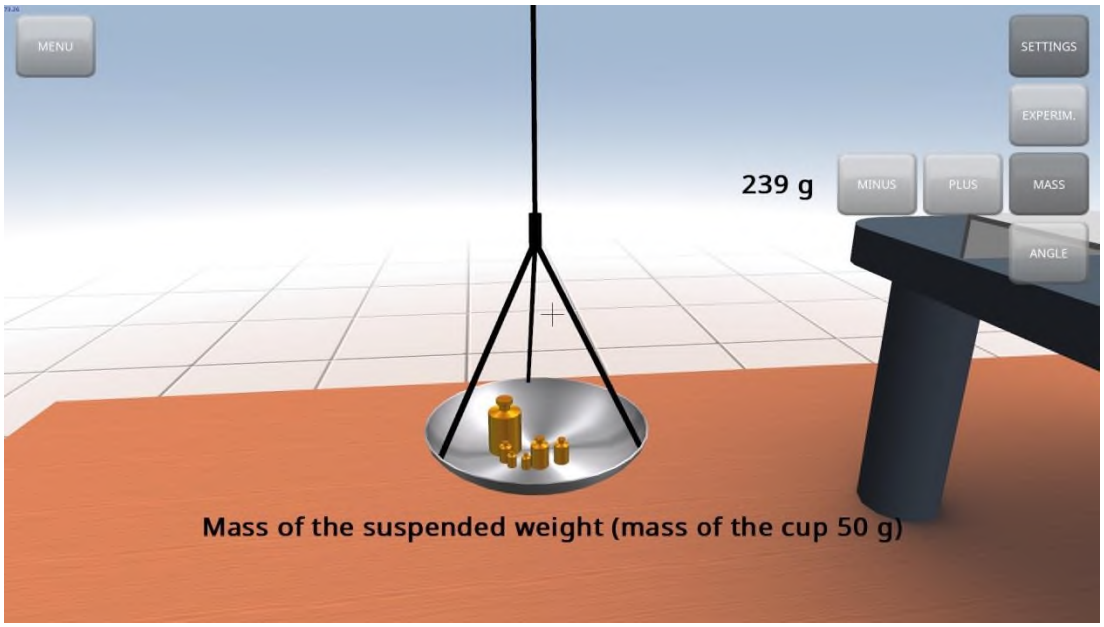

Рисунок 3.5 – Зміна значення сили тяги

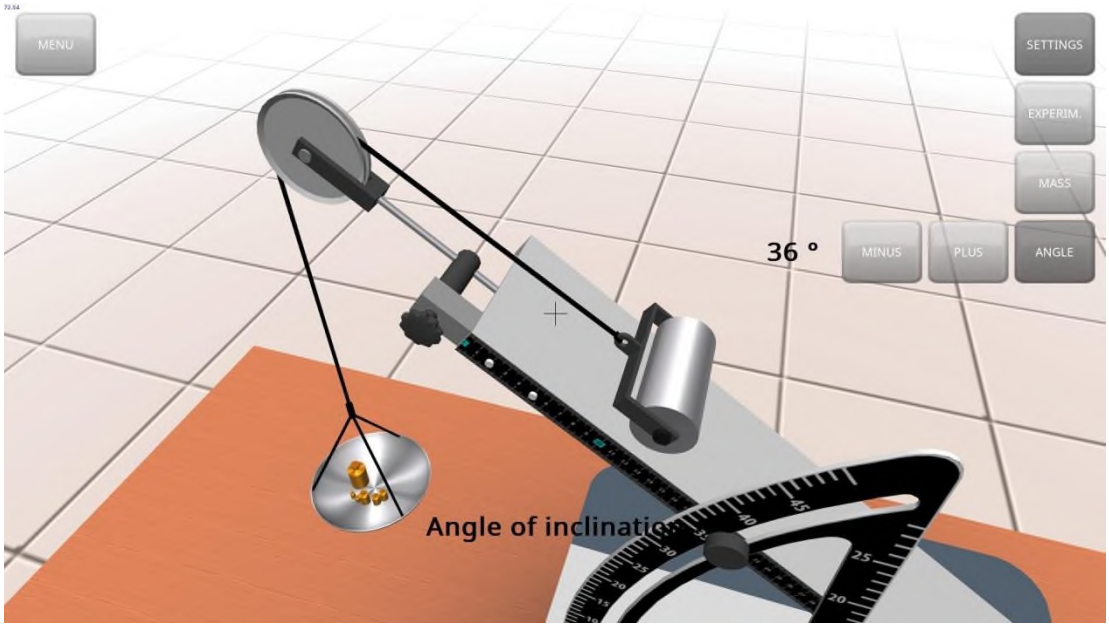

Рисунок 3.6 – Зміна кута нахилу

З рис.3.5 та рис.3.6 видно, що користувач може змінювати лише кут нахилу та значення сили тяги, що в даному випадку залежить від маси тягарців, але користувач не може змінити такі параметри як кут сили тяги, масу тіла та коефіцієнт тертя. Також цей програмний продукт не дає змогу виконати розрахунки, в ньому відсутні теоретичний матеріал та переклад.

«MSC Adams» має більший функціонал, ніж «Наклонная плоскость 1.0», але дозволяє виконувати розрахунки для руху тіла вгору, тобто – не враховує силу тяги. В ньому можна отримати деякі остаточні результати розрахунків, без можливості отримання проміжних результатів та теоретичного матеріалу.

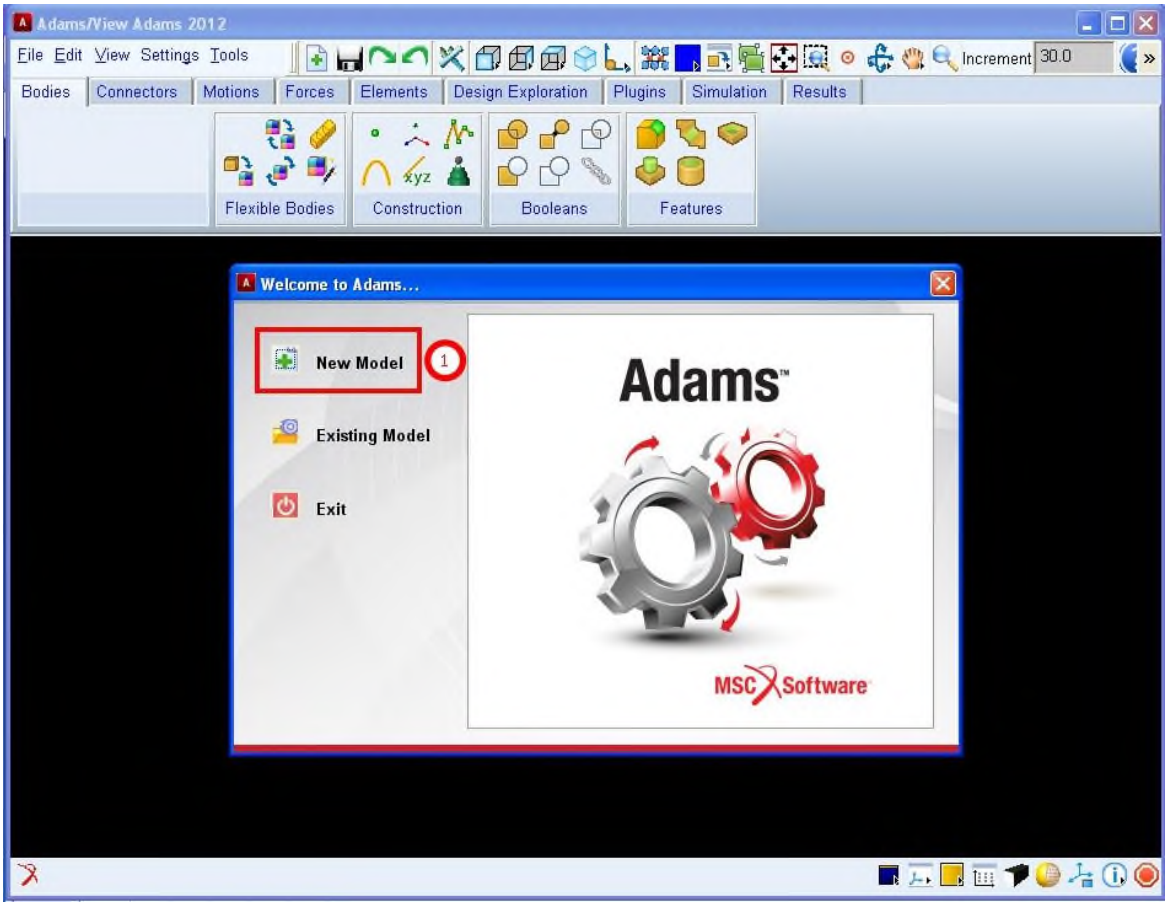

Рисунок3.7 – Вхід в «MSC Adams»

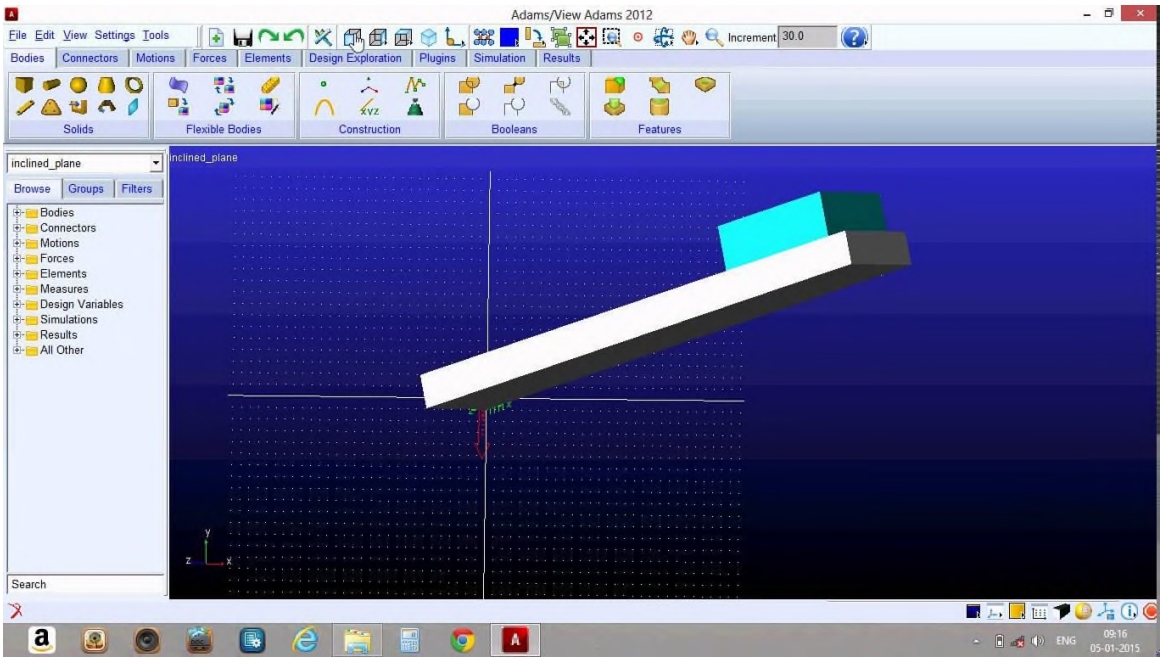

Рисунок 3.8 – Інтерфейс «MSC Adams»

«Comparison of forces» дозволяє порівняти силу тяжіння та силу тяги при русі тіла вниз, при цьому користувач може змінити лише кут нахилу та коефіцієнт тертя. Цей програмний засіб не дозволяє отримати проміжні результати розрахунків, але на відміну від попередніх програм, дозволяє отримати спрощену теоретичну довідку.

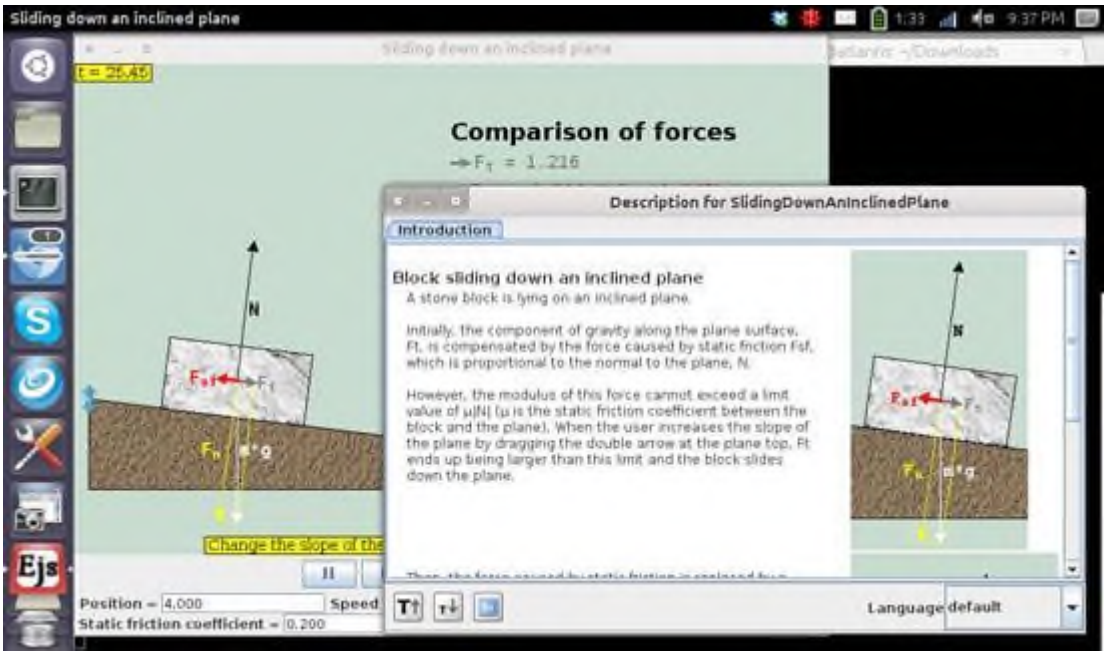

Рисунок 3.9 – Інтерфейс «Comparison of forces»

Порівнявши створений програмний засіб з програмами-аналогами можна виділити наступні переваги:

– можливість визначення більшої кількості вхідних даних: користувач може змінювати такі параметри, як кут нахилу α, коефіцієнт тертя μ, масу тіла m, силу тяги F, кут сили тяги β та обирати напрям руху тіла;

– контроль вхідних даних: в результаті виконання програми користувач отримає значення прискорення руху тіла якщо виконується необхідна умова. Якщо ж умова не виконується, то користувач отримає повідомлення про помилку, що захищає користувача від роботи з помилковими значеннями;

– наявність історії розрахунків за сеанс: вся історія розрахунків за сеанс зберігається в окремому текстовому файлі, тож користувач може відкрити його для ознайомлення та зберегти за необхідністю;

– простий та інтуїтивно зрозумілий інтерфейс;

– наявність спливаючих підказок;

– наявність теоретичного матеріалу: користувач може ознайомитись з усіма означеннями, формулами та таблицями, що необхідні для виконання лабораторних робіт.

3.4 Інструкція роботи користувача

3.4.1 Запуск програми

Для запуску програми потрібно натиснути на піктограму «Рух тіла по похилій площині».

Рисунок 3.10 – Піктограма «Рух тіла по похилій площині»

Після цього з’явиться вікно, в якому потрібно обрати напрям руху тіла та встановити необхідні значення вхідних даних.

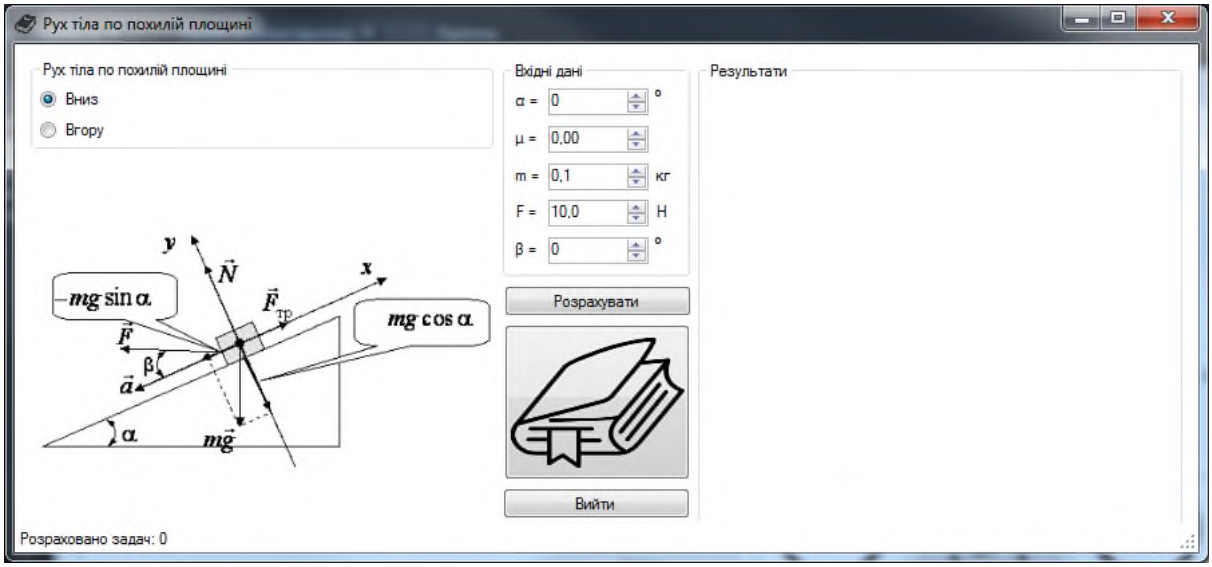

Рисунок 3.11 – Вікно програми

3.4.2 Визначення напряму руху тіла

Для визначення напряму руху тіла по похилій площині необхідно обрати відповідне значення перемикача в лівому верхньому куті вікна.

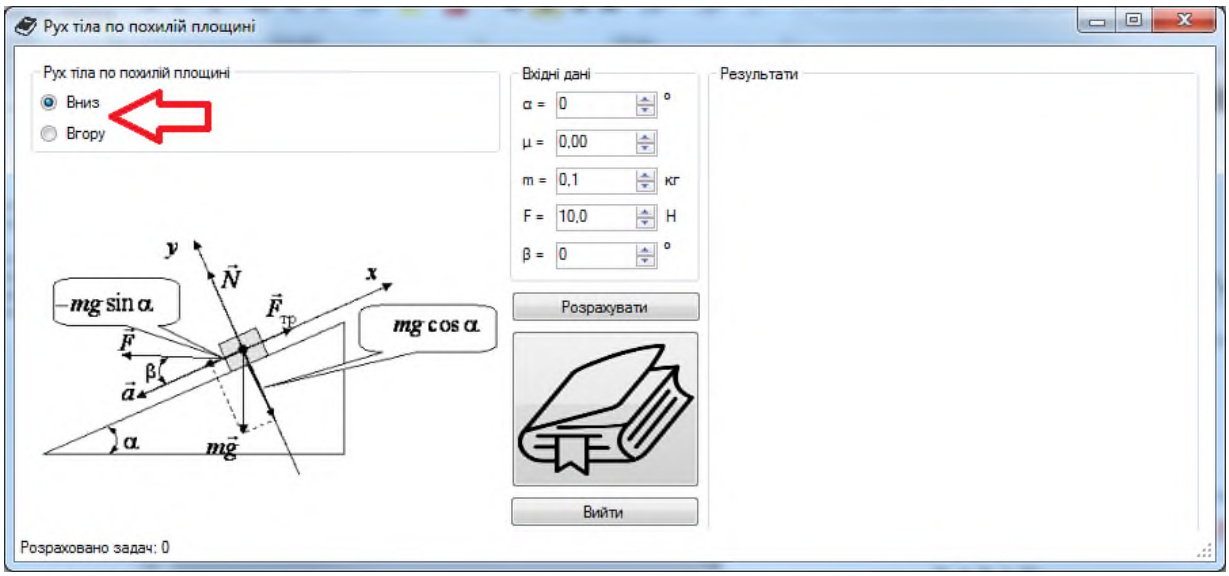

Рисунок 3.12 – Вікно програми (визначення напряму руху тіла)

3.4.3 Визначення вхідних даних

Для визначення вхідних даних необхідно встановити потрібні значення полів, що знаходяться у верхній частині вікна.

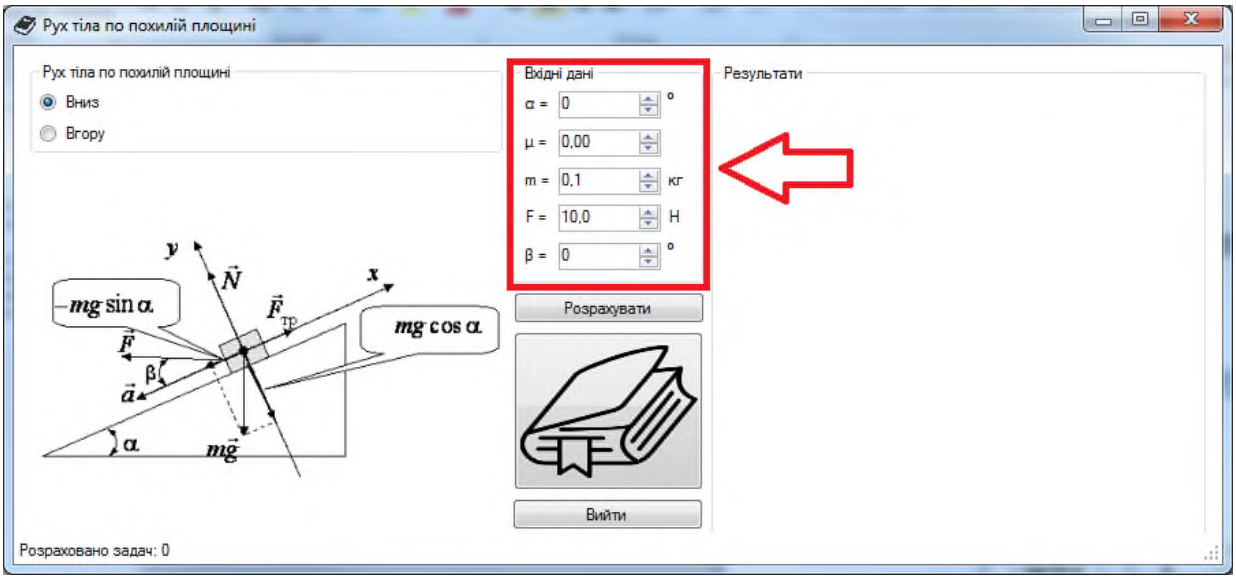

Рисунок 3.13 – Вікно програми (визначення вхідних даних)

3.4.4 Перехід до розрахунків

Щоб перейти до розрахунків необхідно обрати напрям руху та ввести вхідні дані, після чого – натиснути на кнопку «Розрахувати».

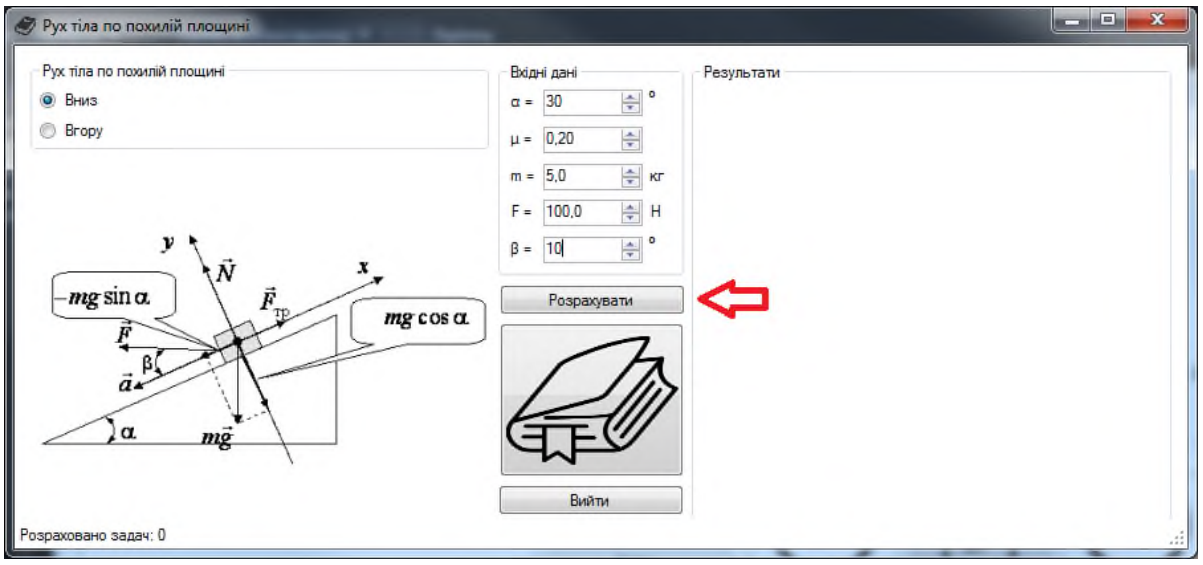

Рисунок 3.14 – Вікно програми (перехід до розрахунків)

Після цього користувач перейде до розрахунків, якщо виконується необхідна умова.

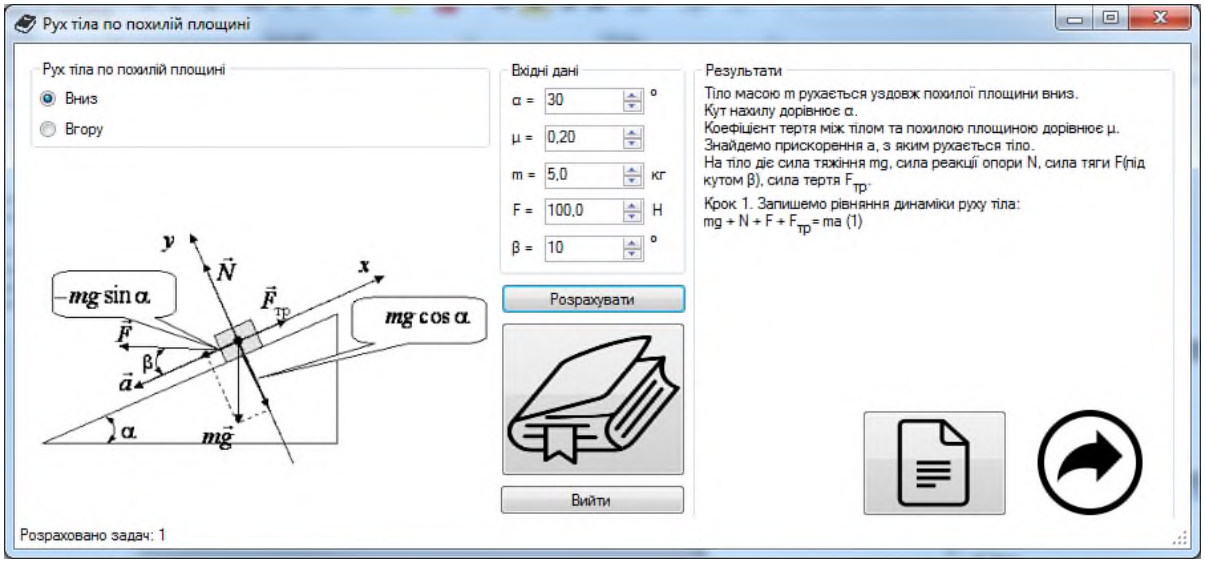

Рисунок 3.15 – Вікно програми (перехід до розрахунків)

Якщо умова не виконується, то користувач отримає повідомлення про помилку і йому доведеться змінити вхідні дані.

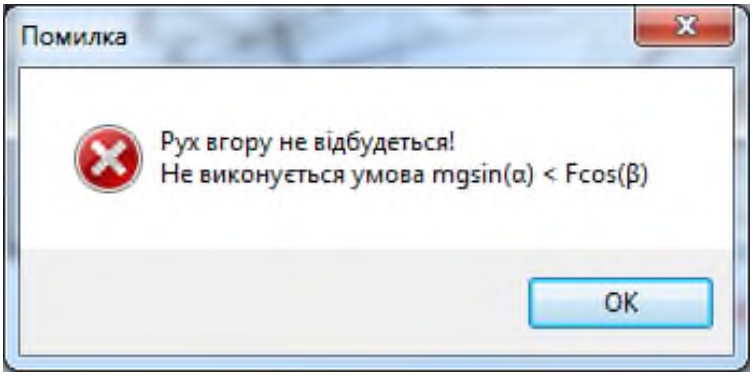

Рисунок 3.16 – Повідомлення про помилку

3.4.5 Навігація в розрахунках

Для того, щоб перейти до наступних кроків в розрахунках потрібно в правому нижньому куті натиснути кнопку «Далі», що зображена у вигляді стрілки вперед.

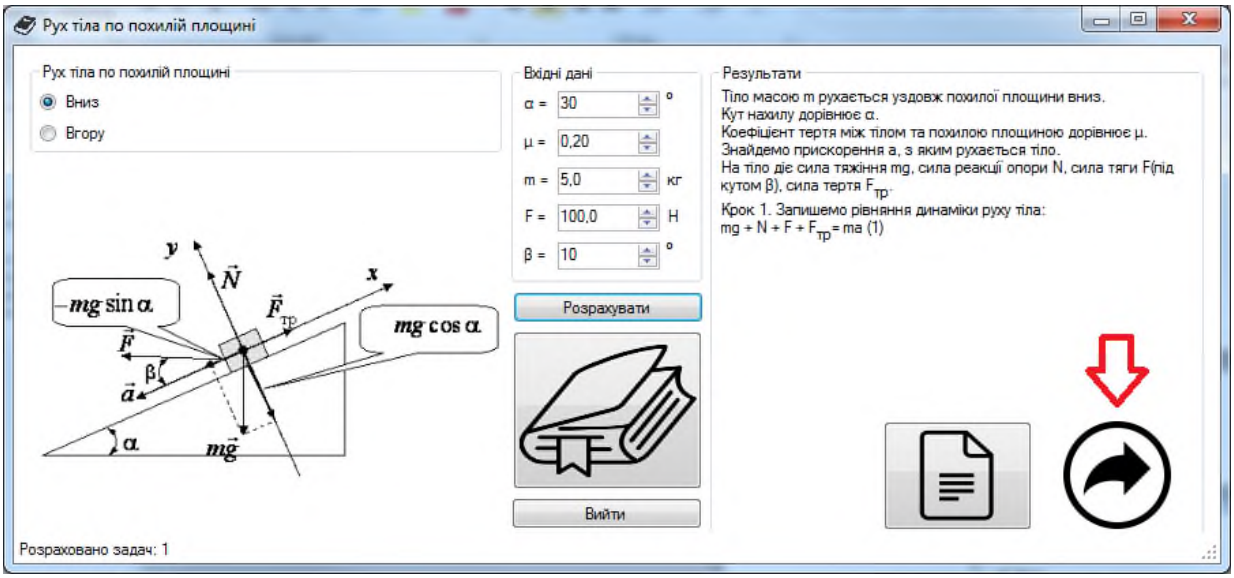

Рисунок 2.17 – Вікно програми (перехід до наступних кроків)

Для того, щоб повернутись до попередніх кроків в розрахунках потрібно в правому нижньому куті натиснути кнопку «Назад», що зображена у вигляді стрілки назад.

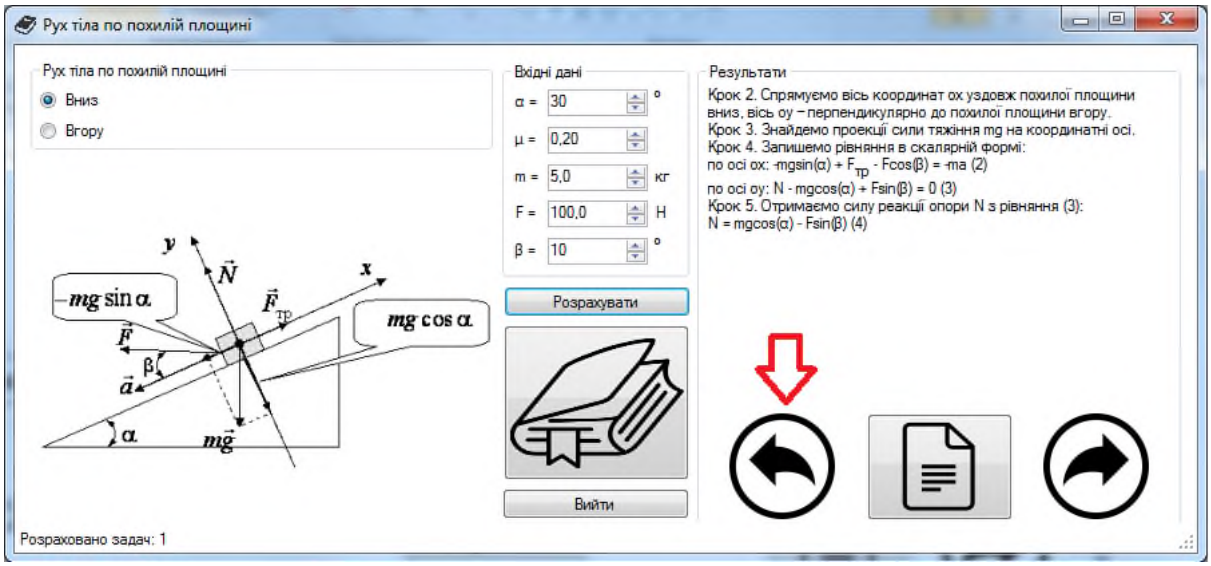

Рисунок 2.18 – Вікно програми (перехід до попередніх кроків)

2.4.6 Перехід до теорії

Щоб перейти до теорії потрібно у вікні програми натиснути на кнопку «Перейти до теорії», що зображена у вигляді книжки.

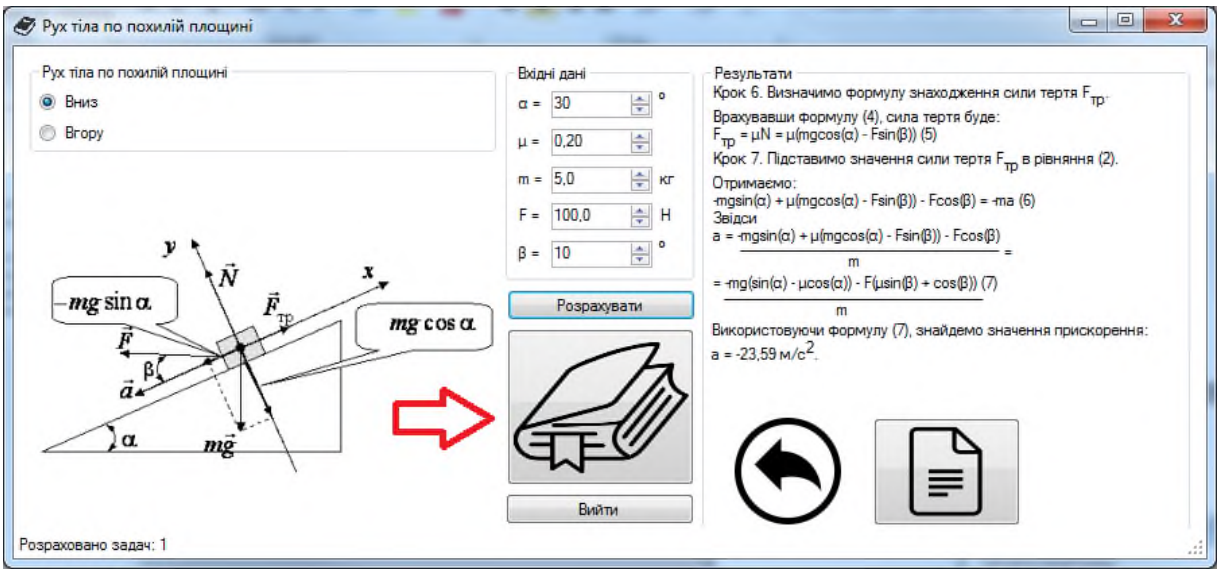

Рисунок 2.19 – Вікно програми (перехід до теорії)

Після цього користувач отримає на екрані вікно з теорією, де зможе обрати необхідний матеріал або таблиці, використовуючи меню на панелі змісту.

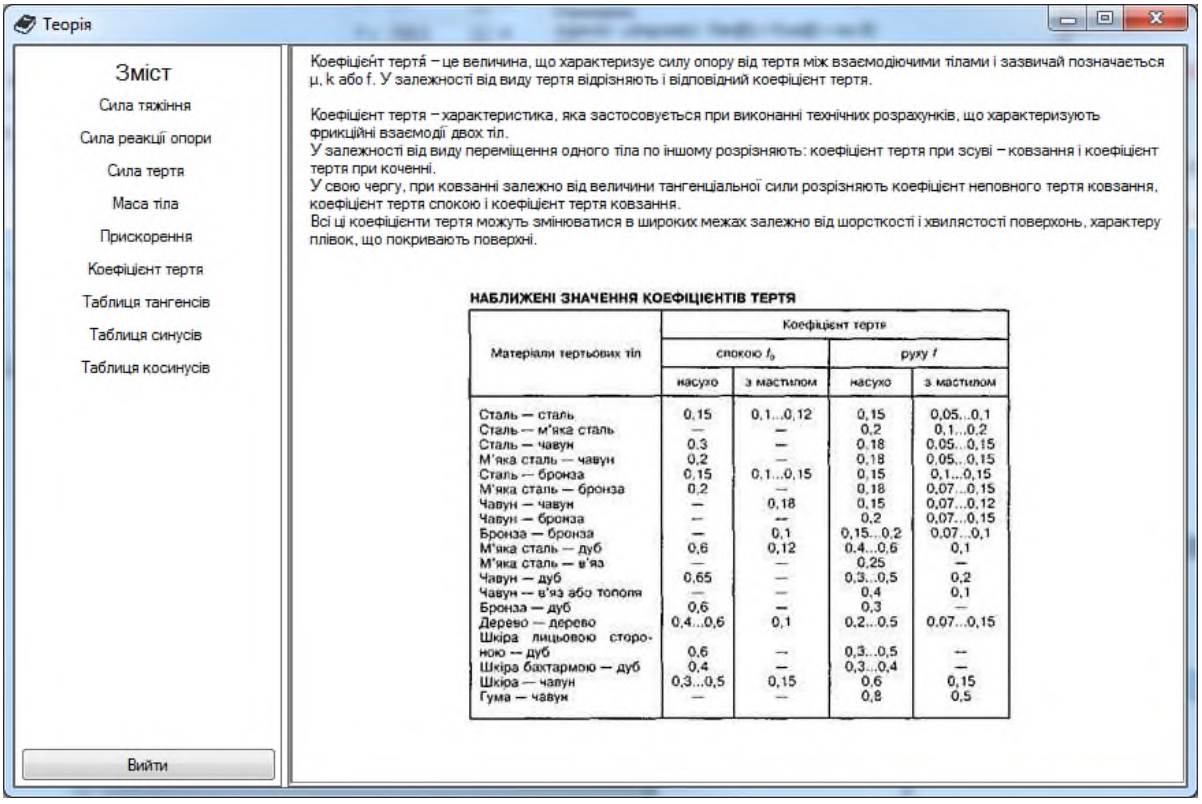

Рисунок 2.20 – Вікно теорії

Для виходу з теорії необхідно натиснути кнопку «Вийти» у вікні теорії.

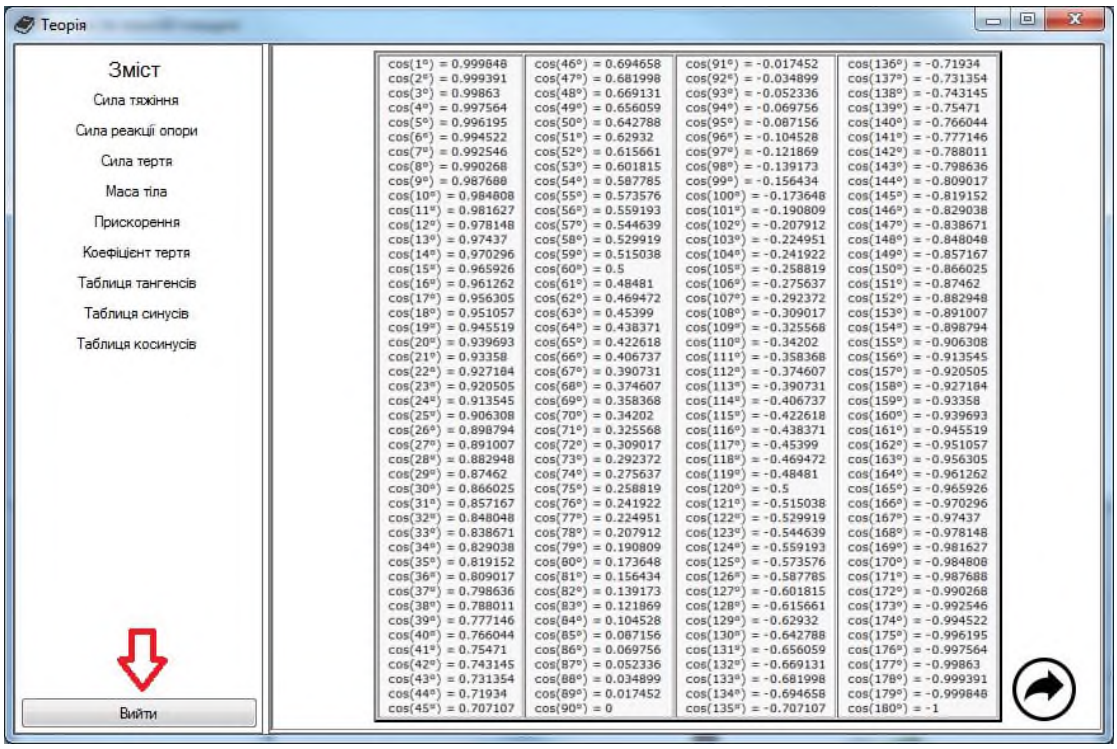

Рисунок 2.21 – Вікно теорії (вихід)

2.4.7 Перегляд історії розрахунків

Програма зберігає історію розрахунків за сеанс в окремому файлі. Для перегляду цього файлу потрібно натиснути кнопку «Перейти до історії».

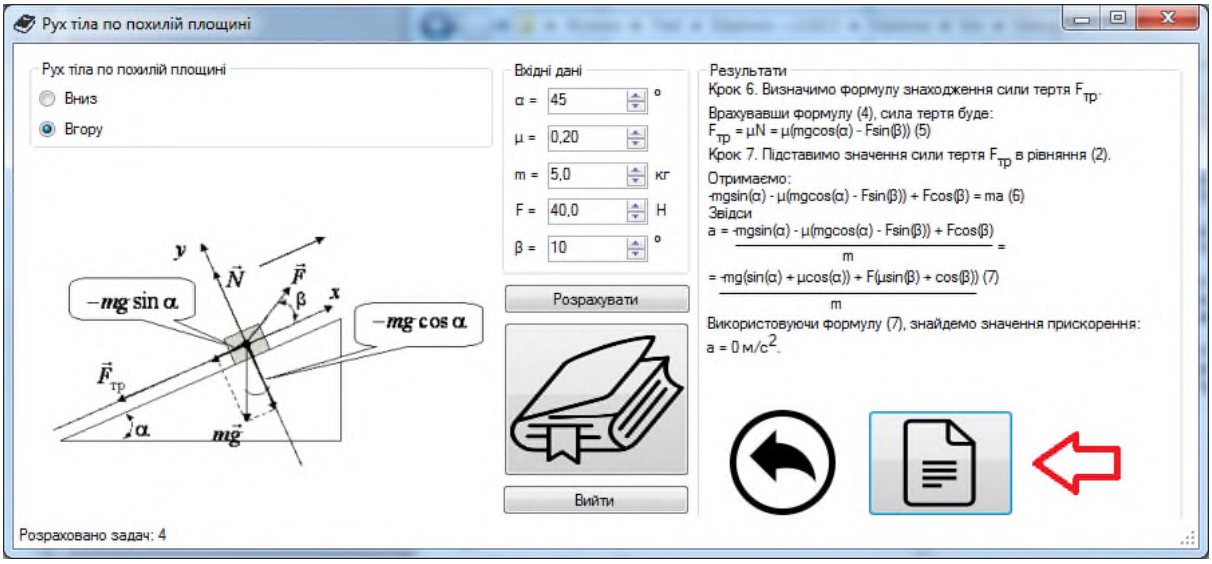

Рисунок 2.22 – Вікно програми (відкриття історії)

Кількість розрахованих задач ви також можете переглянути в нижньому лівому куті програми.

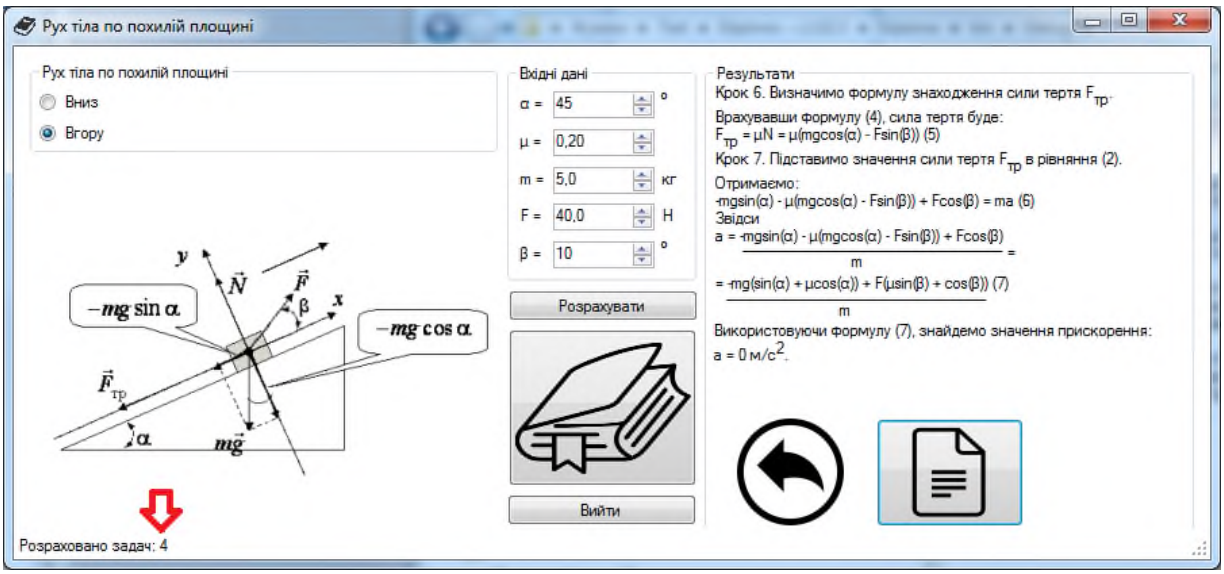

Рисунок 2.23 – Вікно програми (кількість розрахованих задач)

Файл з історією розрахунків містить дату розрахунків, вхідні дані та результат, він може виглядати наступним чином:

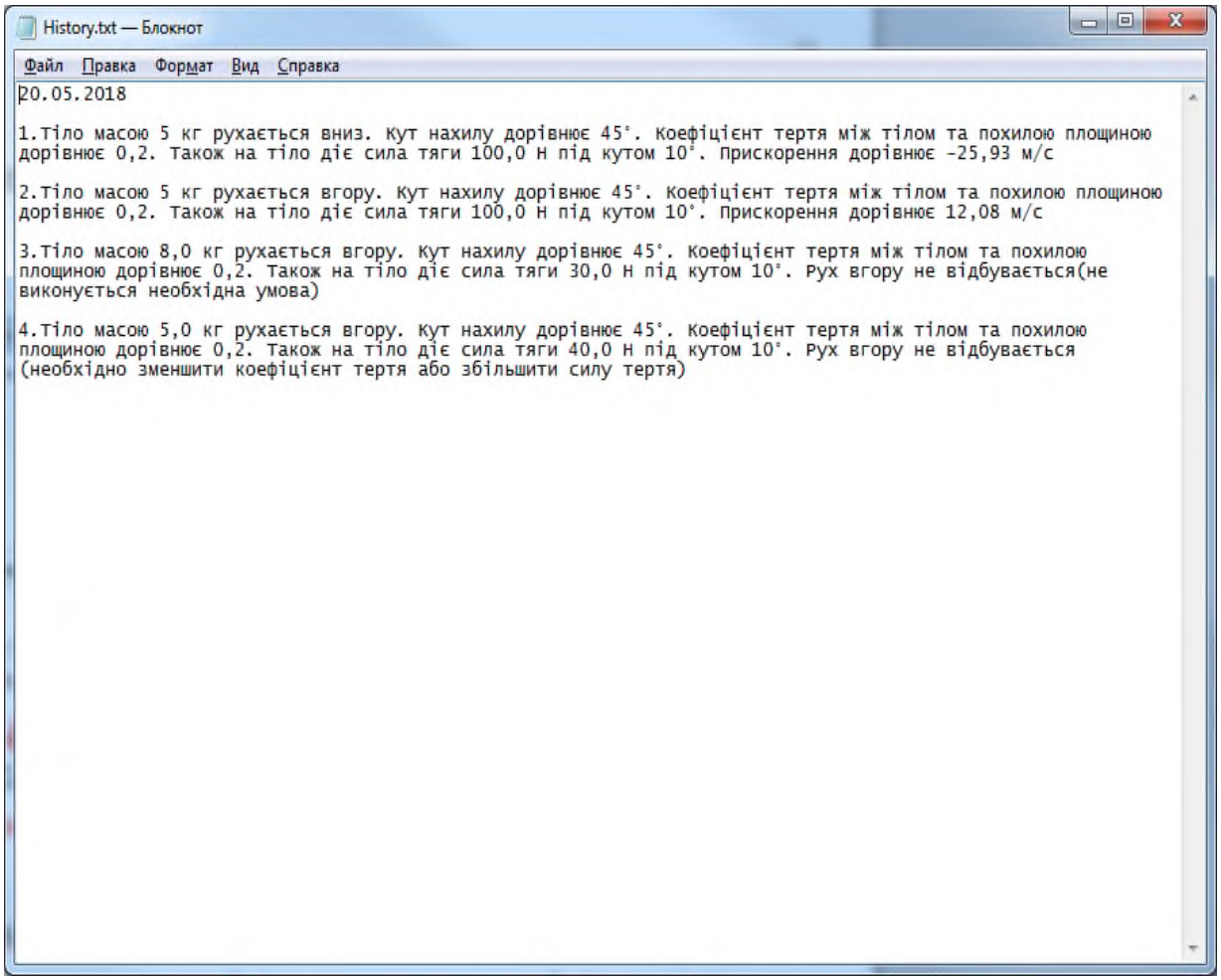

Рисунок 2.24 – Історія розрахунків

2.4.8 Вихід з програми

Для виходу з програми необхідно натиснути кнопку «Вийти» у вікні програми.

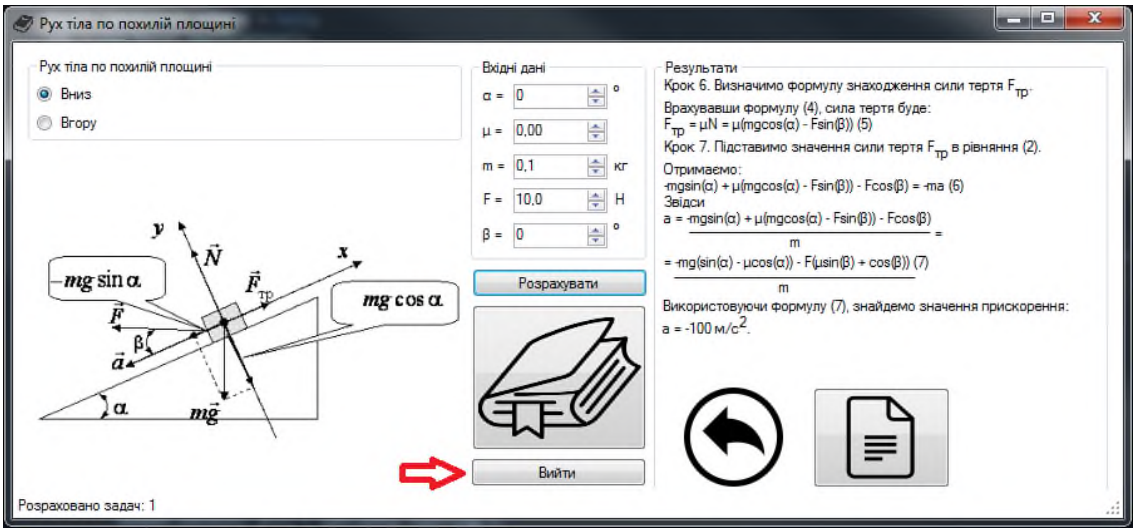

Рисунок 2.25 – Вікно програми (вихід з програми)

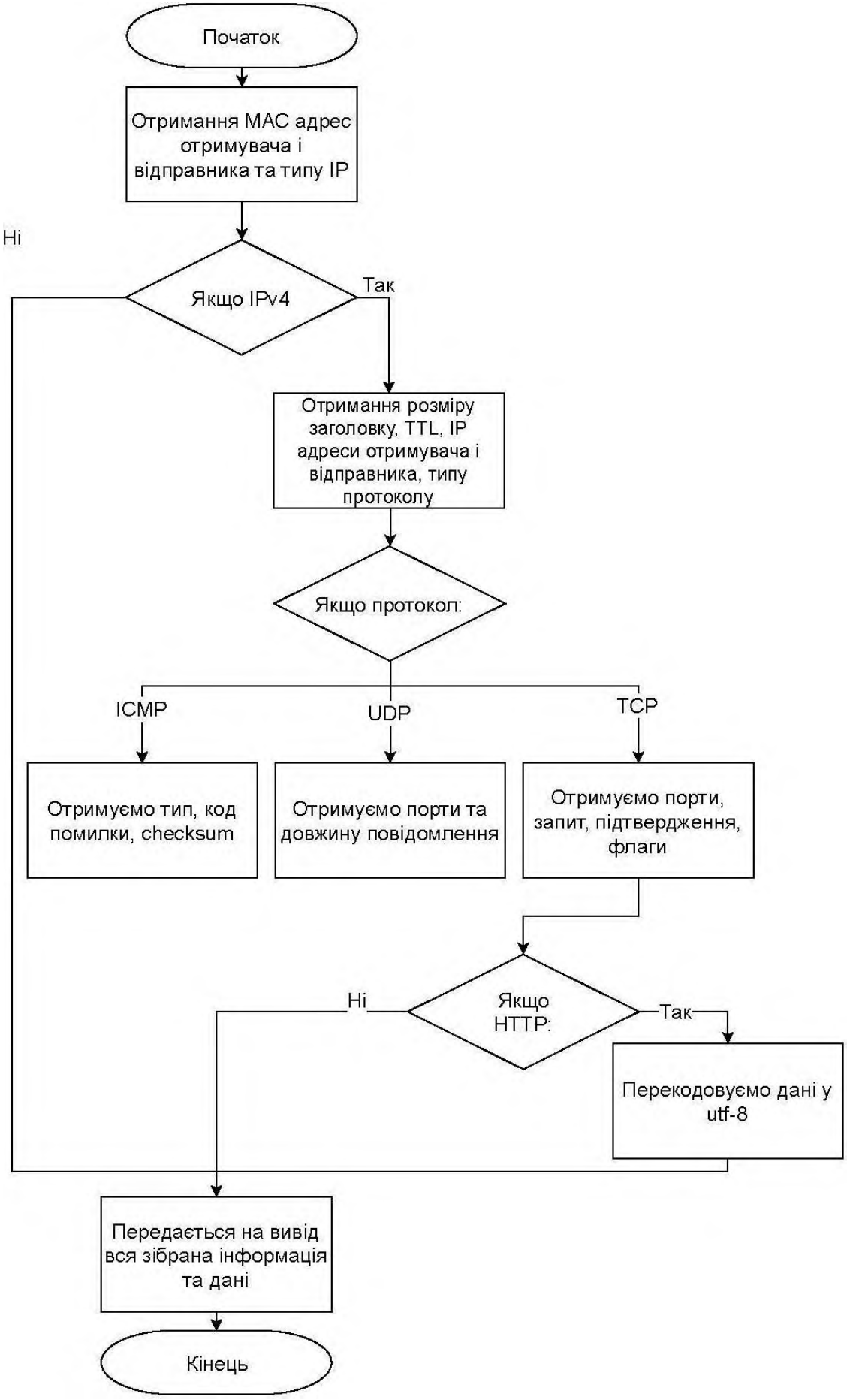

Рис. 3.2. Алгоритм модуля розбору пакетів

2) Створюємо екземпляр класу *sniffer.networking.ethernet*. *Ethernet*, що отримує MAC адреси отримувача і відправника та поле *prototype*(з двох байтів поля *TYPE*, *Ethernet* фрейму).

3) Якщо отриманий *prototype* дорівнює 8 – то отриманий пакет версії *IPv4*. Тоді маємо *IPv4* заголовок зображений на рис. 3.3.

<table>
<tr><td>Версія</td><td>Довжина заголовку</td><td>Тип сервісу</td><td colspan="2">Загальна довжина</td></tr>
<tr><td colspan="3">Ідентифікатор пакету</td><td>IP флаги
x D M</td><td>Зміщення фрагменту</td></tr>
<tr><td>Час життя пакету</td><td colspan="2">Протокол верхнього рівня</td><td colspan="2">Контрольна сума</td></tr>
<tr><td colspan="5">IP адреса відправника 32 біти</td></tr>
<tr><td colspan="5">IP адреса отримувача 32 біти</td></tr>
<tr><td colspan="5">Опції і вирівнювання (не обов’язково)</td></tr>
</table>

Рис. 3.3. Структура *IPv4* заголовку

4) Створюємо екземпляр класу sniffer.networking.ipv4.IPv4, що отримує розмір заголовку, *TTL*, *IP* адреси отримувача і відправника та поле протоколу.

5) Якщо поле протоколу дорівнює 6 то пакет переданий за протоколом *TCP* та маємо його структуру зображену на рис. 3.4. Тоді створюємо екземпляр класу *sniffer.networking. tcp.TCP*, що отримує порти отримувача та відправника, порядковий номер, номер підтвердження та флаги *URG*, *ACK*, *PSH*, *RST*, *SYN*, *FIN*.

Якщо порт відправника, або порт отримувача дорівнює 80 то пакет *HTTP* прикладного рівня. Тоді декодуємо дані у формати виводу *utf*-8 для перегляду в виводі детальних даних про пакет.

Якщо поле протоколу дорівнює 17 то пакет переданий за протоколом *UDP* та маємо структуру зображену на рис. 3.5. Тоді створюємо екземпляр класу *sniffer.networking.udp*. *UDP*, що отримує порти отримувача та відправника та

довжину датаграми.

Якщо поле протоколу дорівнює 1 то пакет переданий за протоколом *ICMP* та маємо структуру зображену на рис. 3.6. Тоді створюємо екземпляр класу *sniffer.networking.udp.ICMP*, що отримує тип, код та контрольну суму.

6) Після всього розбору на вивід передається уся зібрана інформація.

<table>
<tr><td colspan="3">Порт відправника</td><td>Порт отримувача</td></tr>
<tr><td colspan="4">Порядковий номер</td></tr>
<tr><td colspan="4">Номер підтвердження</td></tr>
<tr><td>Довжина
Заголовку</td><td>Резерв</td><td>Флаги</td><td>Розмір вікна</td></tr>
<tr><td colspan="3">Контрольна сума</td><td>Вказівник на термінові дані</td></tr>
<tr><td colspan="4">TCP параметри</td></tr>
</table>

Рис. 3.4. Структура *TCP*

Порт відправника	Порт отримувача
Довжина датаграми	Контрольна сума

Рис. 3.5. Структура *UDP*

Тип (8 або 0)	Код(0)	Контрольна сума
Ідентифікатор		Номер по порядку

Рис. 3.6. Структура *ICMP*

Розробка модуля збору інформації для моніторингу якості мережі.

В даному модулі для відображення необхідної інформації, а це пропускна спроможність за одиницю часу, максимальна пропускна спроможність, кількість пакетів за одиницю часу, максимальна кількість пакетів, кількість втрачених фреймів та відсоток втрачених фреймів, потрібно в класі головного вікна створити

такі змінні:

– *bandwidthOut* – пропускна спроможність відправки;

– *bandwidthIn* – пропускна спроможність отримання;

– *maxBandwidthOut* – максимальна пропускна спроможність відправки;

– *maxBandwidthIn* – максимальна пропускна спроможність отримання;

– *tx1, rx1* – кількість байт відправки та отримання в попередню секунду;

– *tx2, rx2* – кількість байт відправки та отримання в дану секунду;

– *packetsNow* – спільна змінна для модулю перехоплювача та модулю збору інформації. Модуль перехоплювача збільшує змінну на одиницю, коли отримує пакет;

– *packetsPerSecond* – кількість пакетів, що були зібрані за останню секунду;

– *maxPacketsPerSecond* – максимальна кількість пакетів, що були зібрані за останню секунду;

– *bandwidthTimer* – таймер, що кожну секунду отримує значення показників;

– *allPacketsCount* – спільна змінна для модулю перехоплювача та модулю збору інформації. Модуль перехоплювача збільшує змінну на одиницю, коли отримує пакет;

– *lossPacketsCount* – спільна змінна для модулю перехоплювача та модулю збору інформації. Модуль перехоплювача збільшує змінну на одиницю, коли пакет було втрачено.

Значення для змінних *tx1* та *tx2* будуть отримуватися із системного файлу операційної системи *Linux/sys/class/net/your_adapter/statistics/tx_bytes/*. Для змінних *rx1* та *rx2* з файлу */sys/class/net/your_adapter/statistics/ rx_bytes/*.

Завдяки таймеру кожну секунду з файлів буде зчитуватися кількість байт у змінні *tx2* та *rx2*, після чого різниця між *tx2* та *tx1* буде записуватись у змінну *bandwidthOut*, а різниця між *rx2* та rx1 у змінну *bandwidthIn*, а змінні *tx2* та *rx2* будуть приймати значення *tx1* та *rx1* відповідно. Якщо змінна *bandwidthOut* стане більшою ніж *maxBandwidthOut*, то друга візьме значення першої. Аналогічно для

bandwidthIn та *maxBandwidthIn*.

Також кожну секунду змінна *packetsPerSecond* приймає значення змінної *packetsNow*, а друга присвоюється нулю. Виводиться у вікно змінна *lossPacketsCount* та *lossPacketsCount*/*allPacketsCount* * 100 + '%', для відображення відсотку втрачених пакетів.

3.4. Аналіз роботи програмного продукту

Запуск програми.

Для запуску програми необхідно встановити *python* 3.7.3, завантаживши його з офіційного сайту. Також встановити такі пакети:

pip3 install --user pyqt5

sudo apt-get install python3-pyqt5

sudo apt-get install pyqt5-dev-tools

sudo apt-get install qttolls5-dev-tools

Файл для запуску – *getPackets.py.*

Тестування роботи.

Просте тестування та моніторинг мережі. Запускаємо файл getPackets.py та бачимо головне меню зображене на рис. 4.1.

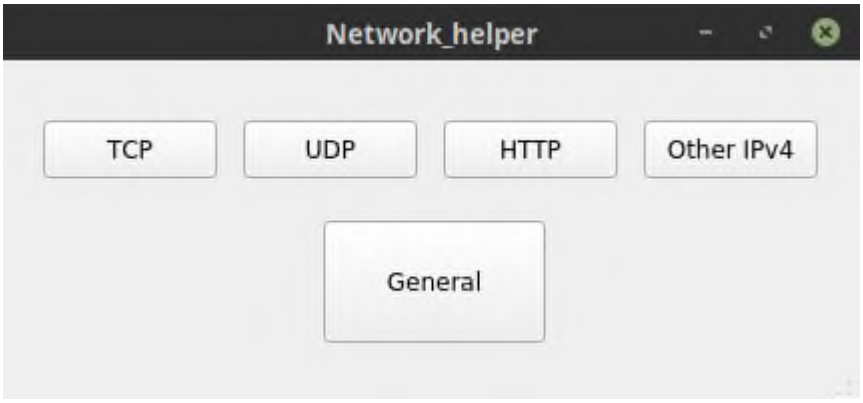

Рис. 4.1. Головне меню додатку.

Якщо натиснути на кнопку *General* відкриється вікно рис. 4.2 та процес розпочнеться. На кожній сторінці присутні кнопки управління для початку або

зупинки роботи модулів перехоплювача та розбору, а також кнопка *Clear*, що видаляє всі збережені в програмі до цього дані.

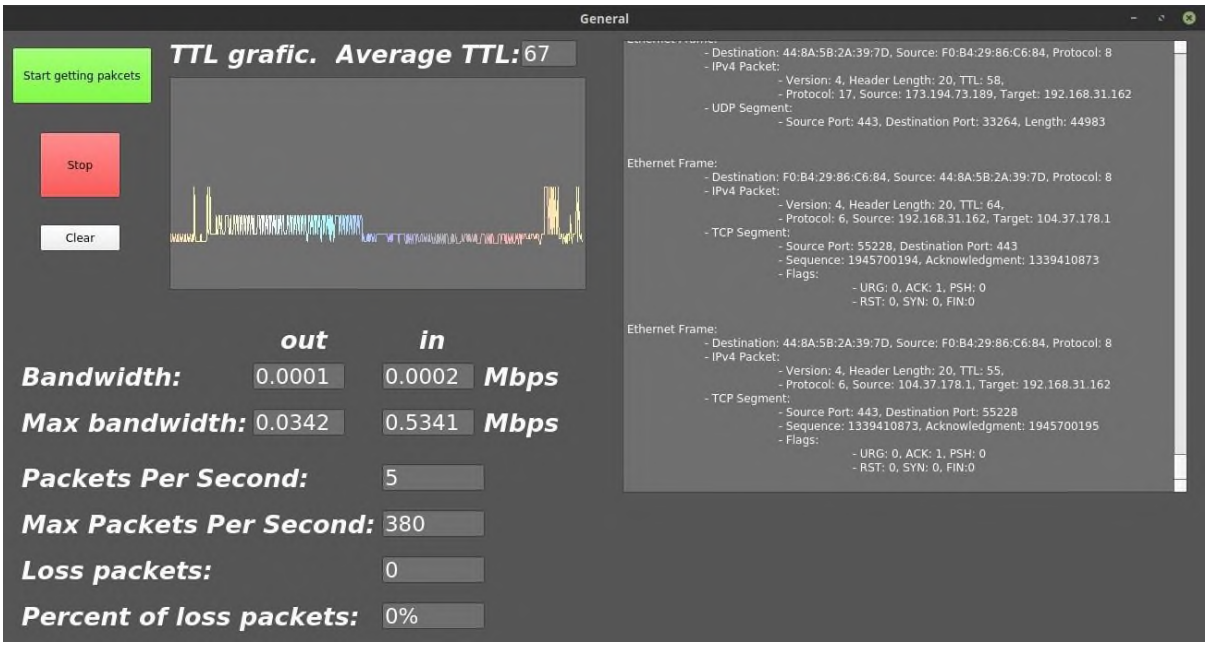

Рис. 4.2. Вікно *General*

Спробуємо загрузити мережу по максимуму (рис. 4.3).

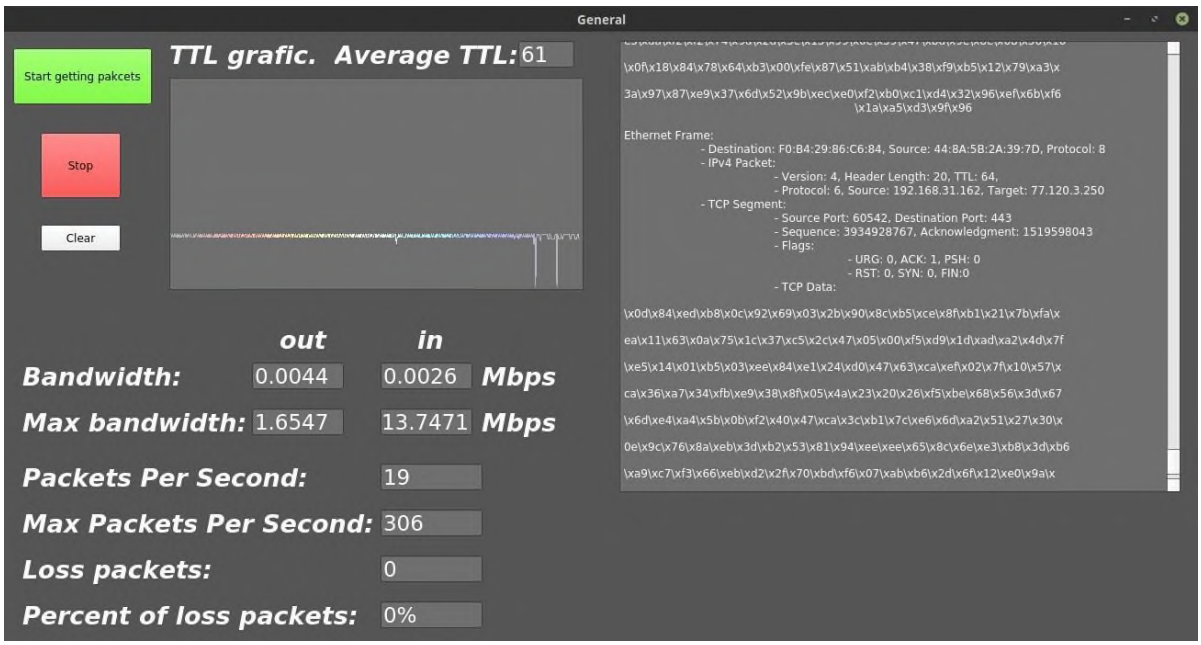

Рис. 4.3. Тест максимальної загрузки. Вікно *General*

Рис. 4.7. Вікно *Other*

ВИСНОВКИ

В ході виконання даного дипломного проекту було проаналізовано методи та інструменти для моніторингу мережевої активності.

Аналіз існуючих рішень в першому розділі показав, що проблема моніторингу комп'ютерної мережі була та є актуальною, як для великих підприємств, так і для звичайного користувача. Були розглянуті методи для моніторингу та управління якістю, а також існуючі засоби. Було вивчено модель *OSI*, протоколи для моніторингу та параметри якості обслуговування.

У другому розділі були розглянуті інструменти для реалізації, обґрунтування їх вибору та описані основні метрики для моніторингу якості обслуговування комп'ютерної мережі. Якщо не стежити за станом якості обслуговування, в майбутньому, можуть виникнути проблеми, що описані в розділі. Було проведено аналіз та порівняння існуючих систем моніторингу.

Система моніторингу – група пристроїв та програмне забезпечення, що забезпечує систематичний збір і обробку інформації, яка може бути використана для поліпшення процесу прийняття рішення, а також, побічно, для інформування громадськості або прямо як інструмент зворотного зв'язку з метою здійснення проектів, оцінки програм або вироблення політики.

У третьому розділі продемонстровано структуру проекта, розбиту на модулі. Описано алгоритми, класи та функції, задіяні в програмі та процес їх розробки. Цей розділ демонструє процес запуску та використання додатку та представляє собою коротке керівництво користувача.

Розроблений додаток має потенціал до розширення функціоналу, так як велика частина одержуваної інформації виводиться у вигляді тексту, але її можна обробляти, сортувати та виводити додаткову інформацію та статистику для моніторингу та управління якістю комп'ютерної мережі.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1) *Inc. staff* (2010), *How to Choose a Data Center*, *retrieved* 2012-07-21

2) Центры обработки данных [Електронний ресурс] – Режим доступу до ресурсу: *http://www.tadviser.ru/index.php*

3) *Greenberg A. What Goes Into a Data Center*/*A. Greenberg*, *D. Maltz*.

4) *A. Greenberg et al*., "*VL2: A scalable and flexible data center network*" *ACM SIGCOMM Comput. Commun. Rev.*, *vol*. 39, *no*. 4, *pp*. 51–62, *Oct*. 2009.

5) *C. Guo, H. Wu, K. Tan, L. Shi, Y. Zhang, and S. Lu,* "*DCell: A scalable and fault- tolerant network structure for data centers*" *ACM SIGCOMM Comput. Commun. Rev.*, *vol*. 38, *no*. 4, *pp*. 75–86, *Oct*. 2008.

6) Hung LeHong, Jackie Fenn. Key Trends to Watch in Gartner 2012 *Emerging Technologies Hype Cycle*

7) *Cisco Systems, Inc*., "*Enterprise Campus 3.0 Architecture: Overview and Framework*" 2008. [Електронний ресурс] – Режим доступу до ресурсу: *http://www.cisco.com/en/US/docs/solutions/Enterprise/Campus/campover.html*

8) *J. Case, M. Fedor, M. Schoffstall and J. Davin*, "*RFC1157*: *A Simple Network Management Protocol* (*SNMP*)," 5 1990. [Електронний ресурс] – Режим доступу до ресурсу: *http://www.ietf.org/rfc/rfc1157.txt*.

9) Офіційний сайт *APC* [Електронний ресурс] – Режим доступу до ресурсу: *http://www.apc.com/ua/ru/*

10) Офіційний сайт *Vutlan* [Електронний ресурс] – Режим доступу до ресурсу: *http://www.vutlan.com/ru/*

11) *Smith H. Data Center Storage/Hubbert Smith*..

12) *Kochlan, M.; Hodon, M.; Cechovic, L.; Kapitulik, J.; Jurecka, M., "WSN for traffic monitoring using Raspberry Pi board," Computer Science and Information Systems (FedCSIS), 2014 Federated Conference on, vol., no., pp.*1023,1026.

13) *Alpha boards in manufacture Raspberry Pi Foundation* [Електронний ресурс]

– Режим доступу до ресурсу: *https://www.raspberrypi.org/blog/alpha-boards-in-manufacture/*

14) *Nagios*. (*n.d*.). [Електронний ресурс] – Режим доступу до ресурсу: *http://nagios.org*

15) *Sophon Mongkolluksamee, Panita Pongpaibool, Chavee Issariyapat, "Strengths and Limitations of Nagios as a Network Monitoring Solution" Proceedings of the 7th International Joint Conference on Computer Science and Software Engineering* (*JCSSE* 2010) *Vol*. 1, *pp*. 96-101, *Bangkok, Thailand, May* 2010

16) *Ahmed D. Kora and Moussa Moindze Soidridine*, "*Nagios based enhanced IT management system*" *Interntional Journal of Engineering Science and Technology, vol*. 4, *no*. 3, *pp*. 818–822, 2012.

17) *SpringGraph Flex Component*. (*n.d*). [Електронний ресурс] – Режим доступу до ресурсу: *http://markshepherd.com/SpringGraph/*

18) *NSClient++ for Windows, Secure monitoring daemon, Retrieved December 2012*. [Електронний ресурс] – Режим доступу до ресурсу: *http://www.nsclient.org/nscp/wiki/doc/about/0.4.x*

19) *D. Doug, B. James R., M. High, "Best of open source networking software,"infoworld.com, Aug* 31, 2009.

20) Сервер *HP ProLiant DL120 G5* [Електронний ресурс] – Режим доступу до ресурсу: *https://support.hpe.com/hpsc/doc/public*

21) Сервер *Dell PowerEdge 2950* [Електронний ресурс] – Режим доступу до ресурсу: *https://s4u.com.ua/dell-poweredge-29502.html*

22) ДСТУ 3008-95 Документація. Звіти у сфері науки і техніки. Структура і правила оформлення.

23) Бойченко С.В., Іванченко О.В. Положення про дипломні роботи (проекти) випускників Національного авіаційного університету. – К.: НАУ, 2017. – 63 с.

ДОДАТОК А

Лістинг коду програмних модулів

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows.Forms;
using System.IO;
namespace Diploma
{
    public partial class Form1 : Form
    {
        int step;
        const double g = 9.8;
        double degreeA, degreeB, radianA, radianB;
        int counter;
        public Form1()
        {
            InitializeComponent();
            counter = 0;
            downBtn.Checked = true;
            pictureBox1.Image = Properties.Resources.Down;
            backBox.Visible = false;
            forwardBox.Visible = false;
```

```
    resultsBox.ReadOnly = true;
    result1box.ReadOnly = true;
    result2box.ReadOnly = true;
    result3box.ReadOnly = true;
    resultsBox.BackColor = Color.White;
    result1box.BackColor = Color.White;
    result2box.BackColor = Color.White;
    result3box.BackColor = Color.White;
    historyBtn.Visible = false;
    Clear();
    ClearFile();
    toolStripStatusLabel1.Text = "Розраховано задач: " + counter;
}
private void downBtn_CheckedChanged(object sender, EventArgs e)
{
    if (downBtn.Checked == true)
    {
        pictureBox1.Image = Properties.Resources.Down;
        valueChanged();
     }
    else
    {
        pictureBox1.Image = Properties.Resources.Up;
        valueChanged();
    }
}
void Clear()
{
    result1box.Visible = false;
    result2box.Visible = false;
```

```
    result3box.Visible = false;
    lineLbl.Visible = false;
    line1Lbl.Visible = false;
    equalLbl.Visible = false;
}
void valueChanged()
{
    resultsBox.Clear();
    backBox.Visible = false;
    forwardBox.Visible = false;
    result1box.Clear();
    result2box.Clear();
    result3box.Clear();
    Clear();
    step = 0;
}
void Check()
{
    if (angleUpDown.Value > angleUpDown.Maximum)
        angleUpDown.Value = angleUpDown.Maximum;
    else if (angleUpDown.Value < angleUpDown.Minimum)
        angleUpDown.Value = angleUpDown.Minimum;
    if (coeffUpDown.Value > coeffUpDown.Maximum)
        coeffUpDown.Value = coeffUpDown.Maximum;
    else if (coeffUpDown.Value < coeffUpDown.Minimum)
        coeffUpDown.Value = coeffUpDown.Minimum;
    if (massUpDown.Value > massUpDown.Maximum)
        massUpDown.Value = massUpDown.Maximum;
    else if (massUpDown.Value < massUpDown.Minimum)
        massUpDown.Value = massUpDown.Minimum;
```

```
    if (tracForce.Value > tracForce.Maximum)
        tracForce.Value = tracForce.Maximum;
    else if (tracForce.Value < tracForce.Minimum)
        tracForce.Value = tracForce.Minimum;
    if (BetaUpDown.Value > BetaUpDown.Maximum)
        angleUpDown.Value = BetaUpDown.Maximum;
    else if (angleUpDown.Value < BetaUpDown.Minimum)
        angleUpDown.Value = BetaUpDown.Minimum;
}
private void startBtn_Click(object sender, EventArgs e)
{
    resultsBox.Clear();
    result1box.Clear();
    result2box.Clear();
    result3box.Clear();
    historyBtn.Visible = true;
    Clear();
    step = 0;
    backBox.Visible = false;
    radianA = Convert.ToDouble(angleUpDown.Value);
    degreeA = radianA * Math.PI / 180 ;
    radianB = Convert.ToDouble(BetaUpDown.Value);
    degreeB = radianB * Math.PI / 180;
    Check();
    if (downBtn.Checked == true)
    {
        forwardBox.Visible = true;
        DownZero();
    }
    else
```

```
        {
            if (Convert.ToDouble(massUpDown.Value) * g * Math.Sin(degreeA) < Convert.ToDouble(tracForce.Value) * Math.Cos(degreeB))
            {
                forwardBox.Visible = true;
                UpZero();
            }
            else
            {
                MessageBox.Show("Рух вгору не відбудеться!\nНе виконується умова mgsin(α) < Fcos(β)", "Помилка", MessageBoxButtons.OK, MessageBoxIcon.Error);
            }
        }
        WriteHistory(downBtn.Checked, ++counter);
    }
    private void exitBtn_Click(object sender, EventArgs e)
    {
        this.Close();
    }
    void ClearFile()
    {
            DateTime date = DateTime.Today;
            int day = date.Day;
            int month = date.Month;
            int year = date.Year;
            string data = day.ToString("00") + "." + month.ToString("00") + "." + year.ToString("00");
            File.WriteAllText(Environment.CurrentDirectory + @"\History.txt", data + "\n");
    }
```

```
void WriteHistory(bool downDirection, int counter)
{
    double a;
    if (downDirection == false && (Convert.ToDouble(massUpDown.Value) * g *
Math.Sin(degreeA) > Convert.ToDouble(tracForce.Value) * Math.Cos(degreeB)))
    {
        File.AppendAllText(Environment.CurrentDirectory + @"\History.txt",
Environment.NewLine + Environment.NewLine + counter + ".Тіло масою " +
massUpDown.Value.ToString() + " кг рухається вгору. Кут нахилу дорівнює " +
angleUpDown.Value.ToString() + "°. Коефіцієнт тертя між тілом та похилою
площиною дорівнює " + coeffUpDown.Value.ToString() + ". Також на тіло діє сила
тяги " + tracForce.Value.ToString() + " Н під кутом " + BetaUpDown.Value.ToString()
+ "°.\n Рух вгору не відбувається(не виконується необхідна умова)");
    }
    else if (downDirection == false && ((-Convert.ToDouble(massUpDown.Value)
* g * (Math.Sin(degreeA) + Convert.ToDouble(coeffUpDown.Value) *
Math.Cos(degreeA)) + Convert.ToDouble(tracForce.Value) *
(Convert.ToDouble(coeffUpDown.Value) * Math.Sin(degreeB)
+ Math.Cos(degreeB))) / Convert.ToDouble(massUpDown.Value)) < 0)
    {
        File.AppendAllText(Environment.CurrentDirectory + @"\History.txt",
Environment.NewLine + Environment.NewLine + counter + ".Тіло масою " +
massUpDown.Value.ToString() + " кг рухається вгору. Кут нахилу дорівнює " +
angleUpDown.Value.ToString() + "°. Коефіцієнт тертя між тілом та похилою
площиною дорівнює " + coeffUpDown.Value.ToString() + ". Також на тіло діє сила
тяги " + tracForce.Value.ToString()
+ " Н під кутом " + BetaUpDown.Value.ToString() + "°.\n Рух вгору не
відбувається(необхідно зменшити коефіцієнт тертя або збільшити силу тертя)");
    }
    else if (downDirection == false)
```

```
        {
            a = Math.Round((-Convert.ToDouble(massUpDown.Value) * g * (Math.Sin(degreeA) + Convert.ToDouble(coeffUpDown.Value) * Math.Cos(degreeA)) + Convert.ToDouble(tracForce.Value) * (Convert.ToDouble(coeffUpDown.Value) * Math.Sin(degreeB) + Math.Cos(degreeB))) / Convert.ToDouble(massUpDown.Value), 2);
            File.AppendAllText(Environment.CurrentDirectory + @"\History.txt", Environment.NewLine + Environment.NewLine + counter + ".Тіло масою " + massUpDown.Value.ToString() + " кг рухається вгору. Кут нахилу дорівнює " + angleUpDown.Value.ToString() + "°. Коефіцієнт тертя між тілом та похилою площиною дорівнює " + coeffUpDown.Value.ToString() + ". Також на тіло діє сила тяги " + tracForce.Value.ToString() + " Н під кутом " + BetaUpDown.Value.ToString() + "°.\n Прискорення дорівнює " + a + " м/с");
        }
        else
        {
            a = Math.Round((-Convert.ToDouble(massUpDown.Value) * g * (Math.Sin(degreeA) - Convert.ToDouble(coeffUpDown.Value) * Math.Cos(degreeA)) - Convert.ToDouble(tracForce.Value) * (Convert.ToDouble(coeffUpDown.Value) * Math.Sin(degreeB) + Math.Cos(degreeB))) /
Convert.ToDouble(massUpDown.Value), 2);
            File.AppendAllText(Environment.CurrentDirectory + @"\History.txt", Environment.NewLine + Environment.NewLine + counter + ".Тіло масою " + massUpDown.Value.ToString() + " кг рухається вниз. Кут нахилу дорівнює " + angleUpDown.Value.ToString() + "°. Коефіцієнт тертя між тілом та похилою площиною дорівнює " + coeffUpDown.Value.ToString() + ". Також на тіло діє сила тяги " + tracForce.Value.ToString() + " Н під кутом " + BetaUpDown.Value.ToString() + "°.\n Прискорення дорівнює " + a + " м/с");
        }
        toolStripStatusLabel1.Text = "Розраховано задач: " + counter;
```

```
    }
    void DownZero()
    {
        resultsBox.Clear();
         resultsBox.SelectedText = "Тіло масою m рухається уздовж похилої площини вниз.\nКут нахилу дорівнює α.\nКоефіцієнт тертя між тілом та похилою площиною дорівнює μ.\nЗнайдемо прискорення a, з яким рухається тіло.\nНа тіло діє сила тяжіння mg, сила реакції опори N, сила тяги F(під кутом β), сила тертя F";
        resultsBox.SelectionCharOffset = -5;
        resultsBox.SelectedText = "тр";
        resultsBox.SelectionCharOffset = 0;
        resultsBox.SelectedText = ".";
        resultsBox.SelectedText = "\nКрок 1. Запишемо рівняння динаміки руху тіла:\nmg + N + F + F";
        resultsBox.SelectionCharOffset = -5;
        resultsBox.SelectedText = "тр";
        resultsBox.SelectionCharOffset = 0;
        resultsBox.SelectedText = "= ma (1)";
    }
    void DownFirst()
    {
        resultsBox.Clear();
        resultsBox.SelectedText = "Крок 2. Спрямуємо вісь координат ox уздовж похилої площини вниз, вісь oy – перпендикулярно до похилої площини вгору.\nКрок 3. Знайдемо проекції сили тяжіння mg на координатні осі.\nКрок 4. Запишемо рівняння в скалярній формі:\nпо осі ox: -mgsin(α) + F";
        resultsBox.SelectionCharOffset = -5;
        resultsBox.SelectedText = "тр";
        resultsBox.SelectionCharOffset = 0;
        resultsBox.SelectedText = " - Fcos(β) = -ma (2)\nпо осі oy: N - mgcos(α) +
```

```
Fsin(β) = 0 (3)\nКрок 5. Отримаємо силу реакції опори N з рівняння (3):\nN = mgcos(α) - Fsin(β) (4)";
    }
    void DownSecond()
    {
        resultsBox.Clear();
        result1box.Visible = true;
        result2box.Visible = true;
        result3box.Visible = true;
        equalLbl.Visible = true;
        result1box.SelectedText = "Крок 6. Визначимо формулу знаходження сили тертя F";
        result1box.SelectionCharOffset = -5;
        result1box.SelectedText = "тр";
        result1box.SelectionCharOffset = 0;
        result1box.SelectedText = ". Врахувавши формулу (4), сила тертя буде:\nF";
        result1box.SelectionCharOffset = -5;
        result1box.SelectedText = "тр";
        result1box.SelectionCharOffset = 0;
        result1box.SelectedText = " = μN = μ(mgcos(α) - Fsin(β)) (5)\nКрок 7. Підставимо значення сили тертя F";
        result1box.SelectionCharOffset = -5;
        result1box.SelectedText = "тр";
        result1box.SelectionCharOffset = 0;
        result1box.SelectedText = " в рівняння (2). Отримаємо:\n-mgsin(α) + μ(mgcos(α) - Fsin(β)) - Fcos(β) = -ma (6)\nЗвідси\na = -mgsin(α) + μ(mgcos(α) - Fsin(β)) - Fcos(β)";
        lineLbl.Visible = true;
        line1Lbl.Visible = true;
        result2box.SelectedText = "= -mg(sin(α) - μcos(α)) - F(μsin(β) + cos(β)) (7)";
```

```
        double a = Math.Round((-Convert.ToDouble(massUpDown.Value) * g * (Math.Sin(degreeA) - Convert.ToDouble(coeffUpDown.Value) * Math.Cos(degreeA)) - Convert.ToDouble(tracForce.Value) * (Convert.ToDouble(coeffUpDown.Value) * Math.Sin(degreeB) + Math.Cos(degreeB))) / Convert.ToDouble(massUpDown.Value), 2);
        result3box.SelectedText = "Використовуючи формулу (7), знайдемо значення прискорення:\na = " + a.ToString() + " м/с";
        result3box.SelectionCharOffset = 5;
        result3box.SelectedText = "2";
        result3box.SelectionCharOffset = 0;
        result3box.SelectedText = ".";
    }
    void UpZero()
    {
        resultsBox.Clear();
        resultsBox.SelectedText = "Тіло масою m рухається уздовж похилої площини вгору.\nКут
нахилу дорівнює α.\nКоефіцієнт тертя між тілом та похилою площиною дорівнює μ.\nЗнайдемо прискорення a, з яким рухається тіло.\nНа тіло діє сила тяжіння mg, сила реакції опори N, сила тяги F(під кутом β), сила тертя F";
        resultsBox.SelectionCharOffset = -5;
        resultsBox.SelectedText = "тр";
        resultsBox.SelectionCharOffset = 0;
        resultsBox.SelectedText = ".";
        resultsBox.SelectedText = "\nКрок 1. Запишемо рівняння динаміки руху тіла:\nmg + N + F + F";
        resultsBox.SelectionCharOffset = -5;
        resultsBox.SelectedText = "тр";
        resultsBox.SelectionCharOffset = 0;
        resultsBox.SelectedText = "= ma (1)";
```

```
    }
    void UpFirst()
    {
        resultsBox.Clear();
        resultsBox.SelectedText = "Крок 2. Спрямуємо вісь координат ox уздовж похилої площини вгору, вісь oy – перпендикулярно до похилої площини вниз.\nКрок 3. Знайдемо проекції сили тяжіння mg на координатні осі.\nКрок 4. Запишемо рівняння в скалярній формі:\nпо осі ox: -mgsin(α) - F";
        resultsBox.SelectionCharOffset = -5;
        resultsBox.SelectedText = "тр";
        resultsBox.SelectionCharOffset = 0;
        resultsBox.SelectedText = " + Fcos(β) = ma (2)\nпо осі oy: N - mgcos(α) + Fsin(β) = 0
 (3)\nКрок 5. Отримаємо силу реакції опори N з рівняння (3):\nN = mgcos(α) - Fsin(β) (4)";
    }
    void UpSecond()
    {
        resultsBox.Clear();
        result1box.Visible = true;
        result2box.Visible = true;
        result3box.Visible = true;
        equalLbl.Visible = true;
        result1box.SelectedText = "Крок 6. Визначимо формулу знаходження сили тертя F";
        result1box.SelectionCharOffset = -5;
        result1box.SelectedText = "тр";
        result1box.SelectionCharOffset = 0;
        result1box.SelectedText = ". Врахувавши формулу (4), сила тертя буде:\nF";
        result1box.SelectionCharOffset = -5;
```

```
result1box.SelectedText = "тр";
result1box.SelectionCharOffset = 0;
result1box.SelectedText = " = μN = μ(mgcos(α) - Fsin(β)) (5)\nКрок 7. Підставимо значення сили тертя F";
result1box.SelectionCharOffset = -5;
result1box.SelectedText = "тр";
result1box.SelectionCharOffset = 0;
result1box.SelectedText = " в рівняння (2). Отримаємо:\n-mgsin(α) - μ(mgcos(α) - Fsin(β)) + Fcos(β) = ma (6)\nЗвідси\na = -mgsin(α) - μ(mgcos(α) - Fsin(β)) + Fcos(β)";
line1Lbl.Visible = true;
lineLbl.Visible = true;
result2box.SelectedText = "= -mg(sin(α) + μcos(α)) + F(μsin(β) + cos(β)) (7)";
double a = Math.Round((-Convert.ToDouble(massUpDown.Value) * g * (Math.Sin(degreeA) + Convert.ToDouble(coeffUpDown.Value) * Math.Cos(degreeA)) + Convert.ToDouble(tracForce.Value) * (Convert.ToDouble(coeffUpDown.Value) * Math.Sin(degreeB) + Math.Cos(degreeB))) / Convert.ToDouble(massUpDown.Value), 2);
if (a < 0)
{
   a = 0;
   MessageBox.Show("Прискорення дорівнює нулю, рух вгору не відбувається.\nДля розрахунку необхідно зменшити коефіцієнт тертя μ або збільшити силу тяги F", "Повідомлення", MessageBoxButtons.OK, MessageBoxIcon.Information);
}
result3box.SelectedText = "Використовуючи формулу (7), знайдемо значення прискорення:\na = " + a.ToString() + " м/с";
result3box.SelectionCharOffset = 5;
result3box.SelectedText = "2";
```

```
    result3box.SelectionCharOffset = 0;
    result3box.SelectedText = ".";
}
private void forwardBox_Click(object sender, EventArgs e)
{
    if (downBtn.Checked == true)
    {
        switch (step)
        {
            case 0:
                {
                    DownFirst();
                    step++;
                    break;
                }
            case 1:
                {
                    DownSecond();
                    step++;
                    forwardBox.Visible = false;
                    break;
                }
        }
        backBox.Visible = true;
    }
    else
    {
        switch (step)
        {
            case 0:
```

```
            {
                UpFirst();
                step++;
                break;
            }
        case 1:
            {
                UpSecond();
                step++;
                forwardBox.Visible = false;
                break;
            }
    }
    backBox.Visible = true;
  }
}
private void angleUpDown_ValueChanged(object sender, EventArgs e)
{
  valueChanged();
}
private void coeffUpDown_ValueChanged(object sender, EventArgs e)
{
  valueChanged();
}
private void massUpDown_ValueChanged(object sender, EventArgs e)
{
  valueChanged();
}
private void angleUpDown_KeyPress(object sender, KeyPressEventArgs e)
{
```

```
        valueChanged();
    }
    private void coeffUpDown_KeyPress(object sender, KeyPressEventArgs e)
    {
        valueChanged();
    }
    private void massUpDown_KeyPress(object sender, KeyPressEventArgs e)
    {
        valueChanged();
    }
    private void tracForce_ValueChanged(object sender, EventArgs e)
    {
        valueChanged();
    }
    private void tracForce_KeyPress(object sender, KeyPressEventArgs e)
    {
        valueChanged();
    }
    private void theoryBtn_Click(object sender, EventArgs e)
    {
        theoryForm thF = new theoryForm();
        thF.ShowDialog();
    }
    private void button1_Click(object sender, EventArgs e)
    {

System.Diagnostics.Process.Start(Path.Combine(Directory.GetCurrentDirectory(),
"History.txt"));
    }
    private void BetaUpDown_ValueChanged(object sender, EventArgs e)
```

```
{
    valueChanged();
}
private void BetaUpDown_KeyPress(object sender, KeyPressEventArgs e)
{
    valueChanged();
}
private void backBox_Click(object sender, EventArgs e)
{
    result1box.Clear();
    result2box.Clear();
    result3box.Clear();
    Clear();
    if (downBtn.Checked == true)
    {
        switch (step)
        {
            case 1:
                {
                    DownZero();
                    step--;
                    backBox.Visible = false;
                    break;
                }
            case 2:
                {
                    DownFirst();
                    step--;
                    forwardBox.Visible = true;
                    break;
```

```
                }
            }
        }
        else
        {
            switch (step)
            {
                case 1:
                {
                    UpZero();
                    step--;
                    backBox.Visible = false;
                    break;
                }
                case 2:
                {
                    UpFirst();
                    step--;
                    forwardBox.Visible = true;
                    break;
                }
            }
        }
    }
}
```

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows.Forms;
namespace Diploma
{
    public partial class theoryForm : Form
    {
        public theoryForm()
        {
            InitializeComponent();
            tabControl1.SelectedTab = tabPage1;
            gravitationBtn.FlatAppearance.BorderSize = 0;
            gravitationBtn.FlatStyle = FlatStyle.Flat;
            reactionForceBtn.FlatAppearance.BorderSize = 0;
            reactionForceBtn.FlatStyle = FlatStyle.Flat;
            frictionForceBtn.FlatAppearance.BorderSize = 0;
            frictionForceBtn.FlatStyle = FlatStyle.Flat;
            massBtn.FlatAppearance.BorderSize = 0;
            accelerationBtn.FlatAppearance.BorderSize = 0;
            accelerationBtn.FlatStyle = FlatStyle.Flat;
            frictionCoefBtn.FlatAppearance.BorderSize = 0;
            frictionCoefBtn.FlatStyle = FlatStyle.Flat;
            coeffTableBtn.FlatAppearance.BorderSize = 0;
            coeffTableBtn.FlatStyle = FlatStyle.Flat;
```

```
    sinTalbleBtn.FlatAppearance.BorderSize = 0;
    sinTalbleBtn.FlatStyle = FlatStyle.Flat;
    cosTableBtn.FlatAppearance.BorderSize = 0;
    cosTableBtn.FlatStyle = FlatStyle.Flat;
}
private void coeffTableBtn_Click(object sender, EventArgs e)
{
    tabControl1.SelectedTab = tabPage7;
    tgPictureBox.Image = Properties.Resources.tangensTable;
    backTgBox.Visible = false;
    forwardTgBox.Visible = true;
}
private void closeBtn_Click(object sender, EventArgs e)
{
    this.Close();
}
private void sinTalbleBtn_Click(object sender, EventArgs e)
{
    tabControl1.SelectedTab = tabPage8;
    sinTableBox.Image = Properties.Resources.sinusTable1;
    backBox.Visible = false;
    forwardBox.Visible = true;
}
private void forwardBox_Click(object sender, EventArgs e)
{
    sinTableBox.Image = Properties.Resources.sinusTable2;
    backBox.Visible = true;
    forwardBox.Visible = false;
}
private void backBox_Click(object sender, EventArgs e)
```

```
        {
            sinTableBox.Image = Properties.Resources.sinusTable1;
            forwardBox.Visible = true;
            backBox.Visible = false;
        }
        private void backTgBox_Click(object sender, EventArgs e)
        {
            tgPictureBox.Image = Properties.Resources.tangensTable;
            forwardTgBox.Visible = true;
            backTgBox.Visible = false;
        }
        private void cosTableBtn_Click(object sender, EventArgs e)
        {
            tabControl1.SelectedTab = tabPage9;
            cosTableBox.Image = Properties.Resources.cosinusTable1;
            backCosBtn.Visible = false;
            forwCosBtn.Visible = true;
        }
    }
}
```