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

”Допущено до захисту”
Завідувач кафедри кібербезпеки
та програмного забезпечення
д.т.н., професор
_____ Олексій СМІРНОВ
“____” _____ 2023 р.

ВИПУСКНА КВАЛІФІКАЦІЙНА РОБОТА
за першим (бакалаврським) рівнем вищої освіти
на тему
“Програмне забезпечення системи кібербезпеки захищеного обміну інформацією у мережі під керуванням ОС Linux”

Виконав здобувач вищої освіти
IV курсу, групи КБ-19
ОПП «Кібербезпека»
спеціальності 125 «Кібербезпека»
_____ Головко Д.С.
«____» _____ 2023 р.

Керівник проекту
доктор технічних наук, професор
_____ Смірнов О.А.
«____» _____ 2023 р.
Рецензент _____
_____

м. Кропивницький

Центральноукраїнський національний технічний університет

Факультет *Механіко-технологічний*

Кафедра *Кібербезпеки та програмного забезпечення*

Освітній ступінь *бакалавр*

Галузь знань ._12 _____*“*Інформаційні технології*”*_____

Спеціальність *125 “*Кібербезпека*”*

Освітньо-професійна (освітньо-наукова) програма _*“*Кібербезпека*”*_

ЗАТВЕРДЖУЮ
Завідувач кафедри
д.т.н., проф.
_____ Олексій СМІРНОВ
« 17 » січня 2023 року

ЗАВДАННЯ НА ВИПУСКНУ КВАЛІФІКАЦІЙНУ РОБОТУ ЗА ПЕРШИМ (БАКАЛАВРСЬКИМ) РІВНЕМ ВИЩОЇ ОСВІТИ ЗДОБУВАЧА ВИЩОЇ ОСВІТИ

Головку Даниїлу Сергійовичу
(прізвище, ім’я, по батькові)

1. Тема роботи *Програмне забезпечення системи кібербезпеки захищеного обміну інформацією у мережі під керуванням ОС Linux*

2. Керівник роботи *Смірнов Олексій Анатолійович, докт. техн. наук., професор*
(прізвище, ім’я, по батькові, науковий ступінь, вчене звання)

затверджені наказом вищого навчального закладу № 12-02 від 5.01.2023 року

3. Строк подання студентом роботи до захисту *23.05.2023 р.*

4. Мета та завдання випускної кваліфікаційної роботи*: Метою роботи є розробка програмного забезпечення системи кібербезпеки захищеного обміну інформацією у мережі під керуванням ОС Linux*

5. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити)

1. Призначення та область використання.
2.Перегляд аналогічних існуючих систем.
3. Опис і обґрунтування проектних рішень.
4. Етапи програмування системи.
5. Впровадження системи кібербезпеки в промислову експлуатацію.
6. Висновки

6. Перелік графічного матеріалу (з точним зазначенням обов’язкових креслень)

Структурна схема системи кібербезпеки	*1 аркуш*
Функціональна схема системи кібербезпеки	*1 аркуш*
Діаграма процесів	*1 аркуш*
Блок-схема алгоритму роботи додатку	*2 аркуша*

7. Дата видачі завдання « 17 » січня 2023 р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів випускної кваліфікаційної роботи за першим (бакалаврським) рівнем вищої освіти	Строк виконання етапів випускної кваліфікаційної роботи за першим (бакалаврським) рівнем вищої освіти	Примітка
1.	Аналіз існуючих систем	10.03.2023 р.	
2.	Постановка задачі, оформлення ТЗ	15.03.2023 р.	
3.	Розробка моделі компонента	20.03.2023 р.	
4.	Розробка структур даних	25.03.2023 р.	
5.	Розробка алгоритмів зв’язку та відображення	30.03.2023 р.	
6.	Програмування алгоритмів	10.04.2023 р.	
7.	Оформлення ПЗ	17.04.2023 р.	
8.	Попередній захист роботи	23.05.2023 р.	

Дата видачі завдання
« 17 » січня 2023 р.

Підпис керівника
_____ Смірнов О.А.
(прізвище та ініціали)

Завдання прийнято до виконання
« 17 » січня 2023 р.

Підпис здобувача
_____ Головко Д.С.
(прізвище та ініціали)

АНОТАЦІЯ

Головко Д.С. Програмне забезпечення системи кібербезпеки захищеного обміну інформацією у мережі під керуванням ОС Linux. 125 Кібербезпека. Центральноукраїнський національний технічний університет. Кропивницький. 2023.

В даній випускній кваліфікаційній роботі за першим (бакалаврським) рівнем вищої освіти розроблено програмне забезпечення, яке призначено для системи кібербезпеки захищеного обміну інформацією у мережі під керуванням ОС Linux.

Метою розробки є програмне забезпечення системи кібербезпеки захищеного обміну інформацією у мережі під керуванням ОС Linux.

Результат роботи – програмна реалізація системи кібербезпеки захищеного обміну інформацією у мережі під керуванням ОС Linux.

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

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

Програма може використовуватися на ПЕОМ архітектури IBM PC з ОС Linux.

Програму розроблено в середовищі Visual C++.

Ключові слова: кібербезпека, захищений обмін інформацією, Linux

ABSTRACT

Holovko D.S. Linux-based cyber security system software for secure information sharing on a network. 125 Cyber security. Central Ukrainian National Technical University. Kropyvnytskyi. 2023.

In this final qualification work for the first (bachelor) level of higher education, software is developed, which is intended for a cyber security system of secure information exchange in a network running the Linux OS.

The purpose of the development is the software of the cyber security system of the protected information exchange in the network under the control of the Linux OS.

The result of the work is a software implementation of the cyber security system of protected information exchange in the network under the control of the Linux OS.

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

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

The program can be used on a personal computer of IBM PC architecture with Linux OS.

The program was developed in the Visual C++ environment.

Keywords: cybersecurity, secure information exchange, Linux

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, ОДИНИЦЬ І ТЕРМІНІВ

ОС	–	операційна система
ІМ	–	миттєві повідомлення
IMS	–	Instant Messaging Service – служба миттєвих повідомлень
IP	–	інтернет протокол
MD5	–	алгоритм хешування

ВСТУП

Актуальність теми. Однією, з найбільш часто виконуємих дій у мережі є обмін інформацією. Для цієї цілі підходить програма миттєвого обміну повідомленнями – програма для обміну повідомленнями через Інтернет у реальному часі через служби миттєвих повідомлень (Instant Messaging Service, IMS). Можуть передаватися текстові повідомлення, звукові сигнали, зображення, відео, а також провадитися такі дії, як спільне малювання або ігри. Багато які з таких програм можуть застосовуватися для організації групових текстових чатів або відеоконференцій.

Для цього виду комунікації необхідна клієнтська програма, так званий мессенджер. Відмінність від електронної пошти тут у тому, що обмін повідомленнями йде в реальному часі. Більшість IM-клієнтів дозволяє бачити, чи підключені в цей момент абоненти, занесені в список контактів. У ранніх версіях програм все, що друкував користувач, відразу передавалося. Якщо він робив помилку й виправляв її, це теж було видно. У такому режимі спілкування нагадувало телефонну розмову. У сучасних програмах повідомлення з'являються на моніторі співрозмовника вже після закінчення редагування й відправлення повідомлення.

Як правило, мессенджери не працюють самостійно, а підключаються до центрального комп'ютера мережі обміну повідомленнями, називаного сервером. Тому мессенджери й називають клієнтами (клієнтськими програмами). Термін є поняттям із клієнт-серверних технологій.

Широкому колу користувачів відома деяка кількість популярних мереж обміну повідомленнями. Кожна із цих мереж розроблена окремою групою розроблювачів, має окремий сервер і протоколи, відрізняється своїми правилами й особливостями. Між різними мережами звичайно немає ніякого взаємозв'язку.

Таким чином, користувач мережі ICQ не може зв'язатися з користувачем мережі MSN. Однак ніщо не заважає бути одночасно користувачем декількох мереж.

Майже для кожної з мереж є свій мессенджер, розроблений тією ж командою розроблювачів. Так, для користування вищевказаними мережами розроблювачами пропонуються програми з однойменними назвами: ICQ, MSN Messenger, Yahoo! Messenger. Таким чином, якщо один з адресатів користується тільки мережею ICQ, а іншої – тільки мережею MSN, то можна спілкуватися з ними одночасно, установивши на своєму комп'ютері й ICQ, і MSN Messenger, і зареєструвавшись в обох мережах.

У якості альтернативного мессенджера можна вибрати програму стороннього виробника: як комерційну, так і безкоштовну. Популярними альтернативними програмами для спілкування в мережі ICQ є QIP, Trillian, Miranda IM, Pidgin. Також вони дозволяють підключатися одночасно до декількох мереж, що рятує від необхідності встановлювати окремий мессенджер для кожної мережі й дозволяє спілкуватися з усіма адресатами єдиним образом незалежно від мережі.

Більшість IM-мереж використовують закриті протоколи, тому альтернативні клієнти теоретично можуть мати меншу кількість базових функцій, ніж офіційні, хоча на практиці частіше буває навпаки. Однак при змінах протоколу на стороні сервера мережі альтернативні клієнти можуть раптово перестати працювати (наприклад, подібне явище спостерігалося для «нефірмових» клієнтів сервісу ICQ).

Як альтернатива пропрієтарним протоколам для IM був розроблений відкритий протокол XMPP (також відомий, як Jabber), використовуваний у таких сервісах, як Google Talk і ін. Цей протокол часто використовується для організації спілкування в корпоративній і іншій локальній мережах.

Мета й завдання дослідження. Метою роботи є програмне забезпечення системи кібербезпеки захищеного обміну інформацією у мережі під керуванням ОС Linux.

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

– Огляд існуючих систем захищеного обміну інформацією у мережі під керуванням ОС Linux.

– Дослідження системи кібербезпеки захищеного обміну інформацією у мережі під керуванням ОС Linux.

– Програмна реалізація системи кібербезпеки захищеного обміну інформацією у мережі під керуванням ОС Linux.

Практична цінність отриманих результатів полягає в тому, що розроблені алгоритми дозволяють успішно вирішувати задачі захищеного обміну інформацією у мережі під керуванням ОС Linux.

Таким чином, виходячи з вищеперерахованого, програмне забезпечення системи кібербезпеки захищеного обміну інформацією у мережі під керуванням ОС Linux, є актуальною задачею, яка потребує вирішення у даній випускній кваліфікаційній роботі за першим (бакалаврським) рівнем вищої освіти.

1 ПРИЗНАЧЕННЯ ТА ОБЛАСТЬ ВИКОРИСТАННЯ

1.1 Призначення системи

Призначенням системи є обмін інформацією в мережі під керуванням ОС Linux. Для установки програмного забезпечення обміну інформацією в мережі під керуванням ОС Linux, досить розставити посилання в необхідних частинах сайту. Посилання повинні генеруватися мовою програмування на стороні сервера. Так як сервіс спілкування не вимагає реєстрації користувачів, а використовує дані із сайту, необхідно у всіх посиланнях на програмне забезпечення обміну інформацією в мережі, передавати мінімально необхідну інформацію про користувачів: id користувача, никнув або ім'я й стать. Тоді ці дані будуть доступні користувачам у списку контактів.

Для захисту від підміни посилань використовуються MD5-хеш рядки, що містять пароль із налаштувань програмного забезпечення обміну інформацією в мережі, IP-адресу користувача й передані в посиланні дані. Ви обчислюєте MD5-ключ на своєму сервері й перенаправляєте користувача по посиланню із цим ключем. На сервері ключ обчислюється по тим же правилам і порівнюється з відправленим. При збігу ключів запускається програмне забезпечення обміну інформацією в мережі. Пароль повинен бути досить складним для підбора, а код, де він буде прописаний, надійно захищений на сервері від перегляду. Якщо пароль буде украдений, то зловмисники зможуть відкривати чужу переписку, читати й посилати повідомлення від імені власника. У випадку виявлення витоку пароля необхідно терміново його поміняти.

Для оповіщення користувачів про нові повідомлення додайте картинку з адресою після посилання на програмне забезпечення обміну інформацією в мережі в меню користувача. У картинці буде відображатися кількість нових повідомлень.

Для зручності рекомендуємо створити окремий файл-скрипт на сервері, у якому буде відбуватися редирект на сторінку програмного забезпечення обміну інформацією в мережі з усіма необхідними параметрами. При такому варіанті всі посилання виглядають простими й локальними, ведучими на основний домен сайту, а вся логіка по генерації посилань виноситься в окремий файл.

1.2 Область застосування

Областю застосування програмного продукту є системи, у яких необхідно провадити обмін інформацією в мережі під керуванням ОС Linux. У вікні програмного забезпечення обміну інформацією в мережі є два верхніх фрейми, у яких буде завантажуватися інформація про користувачів-співрозмовників. Ліворуч – поточний користувач програмного забезпечення обміну інформацією в мережі, праворуч – активний користувач зі списку контактів, з яким іде переписка. Адреса для фреймів вводиться в налаштуваннях програмного забезпечення обміну інформацією в мережі й повинна указувати на скрипт сайту. Скрипту будуть передаватися в параметрах ID користувача з контактів і ID поточного користувача програмного забезпечення обміну інформацією в мережі. Таким чином, можливо, наприклад, виводити фотографію користувача або інші дані. Висота фреймів 80 пікселів. Якщо адресу не вказати в налаштуваннях, то фрейми не будуть виводитися.

Також можливо в налаштуваннях програмного забезпечення обміну інформацією в мережі вказати адресу скрипта на сервері, через який будуть завантажуватися картинки користувачів у списку контактів для кращого сприйняття списку. Скрипт буде одержувати ID користувача з контактів і повинен виводити відповідному цьому користувачеві картинку або фото розміром 28x28 пикселів у форматі jpg, gif або png. Якщо ця адреса не буде зазначена у налаштуваннях, то контакти будуть виводитися без картинок у більш компактному виді.

Сервіс спілкування можна використовувати не тільки для користувачів сайту, але й для служби підтримки або консультантів. Для цього необхідно завести програмне забезпечення обміну інформацією в мережі в нашій системі й прописати посилання на вашім сайті для одного або декількох працівників. ID для працівників придумайте такі, щоб вони не могли збігтися з ID користувачів. Якщо хочете, щоб відвідувачі сайту могли писати повідомлення без реєстрації, генеруйте щораз унікальний ID для відвідувачів-гостей і зберігайте його в сесії або куках на випадок, якщо відвідувач закриє вікно спілкування й захоче знову в нього зайти.

Для полегшення завдання по підключенню програмного забезпечення обміну інформацією в мережі, як служба підтримки, існує простий спосіб, що включається в налаштуваннях програмного забезпечення обміну інформацією в мережі прапорцем "Використовувати програмне забезпечення обміну інформацією в мережі для служби підтримки". Даний код можна підключити до будь-якого сайту навіть на безкоштовному хостинзі – він не вимагає програмування. Після розміщення коду відвідувачі зможуть задавати питання прямо із сайту й одержувати відповіді в реальному часі або заходячи на сайт пізніше. У цьому варіанті всі користувачі будуть звертатися анонімно, тобто без реєстрації на сайті. ID для користувачів буде вибиратися унікальним у діапазоні від 10000000. Після того як користувач один раз відкриє вікно, його ID збережеться в куках, і пізніше він завжди буде звертатися в службу підтримки під цим ID і бачити відповіді й попередню переписку. У якості ID користувача для служби підтримки в цьому режимі використовується число 99999999. Для входу у вікно програмного забезпечення обміну інформацією в мережі під цим користувачем можливо використовувати посилання "Вхід" зі сторінки керування на цьому сайті, або розмістити посилання на своєму сайті в недоступному для відвідувачів місці.

Можливо автоматично розсилати повідомлення зі свого сервера від одного користувача іншому. Наприклад, повідомлення від адміністратора або різні

повідомлення про події на сайті. Також можливо посилати одне повідомлення відразу всім користувачам.

У програмному забезпеченні обміну інформацією в мережі є функція пошуку нових користувачів і додавання їх у список контактів. Для можливості шукати користувачів по базі на сайті вкажіть у налаштуваннях адресу для скрипта на сервері, що буде повертати дані користувача по уведеному в рядку пошуку ім'я або ніку. Якщо скрипт не вказати, то пошук буде здійснюватися по нашій базі серед користувачів, що, хоча б один раз, зайшли в програмне забезпечення обміну інформацією в мережі.

Для повідомлення користувачів про нові повідомлення можливо створити спеціальний скрипт на сервері й указати його адресу в налаштуваннях програмного забезпечення обміну інформацією в мережі. Наша система періодично перевіряє базу повідомлень користувачів, збирає інформацію про нові повідомлення й передає ці дані на скрипт. Завдання скрипту – одержати для кожного користувача контакти з бази, наприклад, адресу e-mail, і послати по них повідомлення про нові повідомлення. Таким чином, розсилання повідомлень буде відбуватися із сервера, і немає необхідності передавати контакти користувачів у систему. Якщо ж не вкажете в налаштуваннях адреса скрипта, то повідомлення будуть розсилатися сервером тільки для користувачів, у яких зазначений e-mail. Адресу e-mail користувачі можуть уводити самі в налаштуваннях, або її можна передавати в посиланнях на відкриття програмного забезпечення обміну інформацією в мережі, 4-м параметром у даних про користувача.

Таким чином, виходячи з вищеперерахованого, програмне забезпечення системи кібербезпеки захищеного обміну інформацією у мережі під керуванням ОС Linux, є актуальною задачею, яка потребує вирішення у даній випускній кваліфікаційній роботі за першим (бакалаврським) рівнем вищої освіти.

2 ПЕРЕГЛЯД АНАЛОГІЧНИХ ІСНУЮЧИХ СИСТЕМ

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

До найбільш розповсюджених месседжерів працюючих під ОС Linux відносяться наступні:

– Pidgin (Gaim).

– Kopete.

– LICQ.

Відповідно до завдань, які поставлені в ході виконання бакалаврського проектування, проведемо дослідження перерахованих вище мессенджерів.

Pidgin (Gaim)

Pidgin – модульний клієнт миттєвого обміну повідомленнями. Підтримує найбільш популярні протоколи. Поширюється на умовах GNU General Public License. Дозволяє зберігати коментарі до користувачів з листа-контакт-аркуша. Може поєднувати кілька контактів в один метаконтакт. Pidgin використовує бібліотеку GTK+ і є кроссплатформеним програмним забезпеченням.

Вікно бесіди Pidgin 2.3.0 у середовищі GNOME 2.20.1:

– Метаконтакти.

– Запис протоколу подій.

– Підтримка вкладок у вікні розмови.

– Підключення до декількох аккаунтів одночасно.

– Модульна структура.

– Установка аватарів.

– Спостереження за користувачами.

– Інтеграція з GNOME.

– Обмін файлами.

– Кроссплатформеність.

– У версії 2.3.0 додалася можливість утримання відкритим чату при закритому вікні.

Підтримувані протоколи:

– Bonjour iChat.

– Gadu-Gadu.

– XMPP (Google Talk, LJ Talk, Gizmo5, ...).

– ICQ.

– Internet Relay Chat (IRC).

– .NET Messenger Service (MSN).

– Novell GroupWise.

– OpenNAP.

– OSCAR (AIM/ICQ).

– SILC.

– Yahoo! Messenger.

– Zephyr; підтримуються версією 2.0.0beta4 і вище.

– Lotus Sametime.

– QQ.

– Session Initiation Protocol (SIP); підтримуються версією 2.2.0 і вище.

– MySpaceIM.

Сьогодні появою нового безкоштовного додатка для обміну короткими текстовими повідомленнями через інтернет складно когось здивувати, їх налічуються вже десятки. Однак програму Pidgin все-таки варто виділити із загальної маси хоча б за те, що вона підтримує, ні багато, ні мало, 16 протоколів, серед яких як загальновідомі ICQ, AIM, IRC, Google Talk, XMPP, так і більше рідкі Gadu-Gadu, Bonjour, QQ і безліч інших. Це дає користувачам величезні можливості вільного спілкування без необхідності встановлювати для цього декількох додатків, Pidgin упорається з усім один.

Рисунок 2.1 – Інтерфейс користувача Pidgin

Крім величезного переліку використовуваних протоколів, програма Pidgin відрізняється тим, що поширюється в рамках угоди General Public License, тобто як вільне програмне забезпечення, яке можна за своїм розсудом копіювати й роздавати друзям і навіть вносити в нього зміни. Якщо ви маєте навички програмування, то можете використовувати для модернізації програми вихідні коди, які викладені на її офіційному сайті. Ще одна гордість розроблювачів Pidgin це багатоплатформеність, клієнт може працювати під керуванням усіх найпоширеніших операційних систем: Windows, Mac OS, Linux і BSD.

Процес установки Pidgin нічим не відрізняється від більшості інших програм, нам пропонується на одному з етапів вибрати необхідні компоненти, які знадобляться для роботи. У програмі реалізований механізм перевірки правопису. Його робота дуже сильно нагадує те, як оцінює правильність написання слів усім відомий текстовий редактор Microsoft Word, слова в Pidgin перевіряються прямо в діалоговому вікні.

Якщо в програми виникли якісь сумніви, то вона вам підкреслить слово червоною рисою й запропонує кілька варіантів правильного написання.

Непогано реалізована в Pidgin робота зі статусами, можливо поставити собі основний підпис і до неї зробити додатковий, як у ICQ або QIP. У програмі є можливість вести історію повідомлень, набудовувати прийом і передачу файлів, визначати автоматичне виставляння певних статусів, загалом, тим, хто любить покопатися в основних налаштуваннях, є, чим зайнятися.

Якщо ви любите іноді одержувати про співрозмовника трохи більше інформації, чим дає звичайний ICQ-клієнт, то спеціально для вас реалізований набір функцій під загальною назвою «Спостереження за співрозмовниками». Тут можна створити для певного контакту з вашого списку правила, або відзначити події, про походження яких клієнт буде посилати вам повідомлення. Ну а для випадків, коли ви не хочете одержувати від когось ніяких повідомлень, є функція внесення в список ігнорованих.

Настроюються в Pidgin і мультимедійні можливості, можливо міняти звуковий супровід подій, теми для смайликів, набудовувати журнал і так далі. Є функція прикріплення до контактів з вашого абонентського списку власного коментарю, що нечасто доводиться зустрічати в програмах подібного роду. Цікава також функція об'єднання декількох ваших контакт у єдине ціле, свого роду метаконтакт.

Kopete

Kopete – це гнучка й розширювана мультипротокольна програма з відкритим вихідним кодом для інтерактивного обміну повідомленнями через безліч протоколів, побудована за принципом систем, що підключаються. Вона є частиною інтегрованого робочого оточення KDE. Всі протоколи є підключаються модулями, що, і допускають помодульну установку, налаштування й використання без якого-небудь переналаштування основного додатка для знову завантажених модулів. Метою системи Kopete є забезпечення користувачів простим інтерфейсом для всіх систем інтерактивного обміну повідомленнями й у

той же час забезпечити розроблювачів простим засобом для розробки модулів для нових протоколів. Основні розроблювачі системи Kopete надають всім користувачам невеликий набір модулів, що підключаються, і додатково ряд шаблонів для нових розробок.

Kopete дозволяє користувачам використовувати наступні протоколи:

– AIM.

– Bonjour.

– Facebook.

– Gadu-Gadu.

– ICQ.

– XMPP.

– MSN.

– .NET Messenger Service.

– Novell GroupWise.

– QQ.

– SMS.

– Skype (з використанням офіційного клієнта).

– Xfire.

– Yahoo! Messenger.

Можливості:

– Угруповання вікон з повідомленнями через вкладки для легкого перемикання бесід.

– Псевдоніми для контактів.

– Метаконтакти.

– Повідомлення, включаючи спливаючі вікна й звуки.

– Інтеграція з KAddressBook і KMail.

– Смайлики.

– Підтримка веб-камер.

– Перевірка орфографії.

Однією з популярних операційних систем, установлених у користувачів усього миру, є сімейство OS Linux. Їхні переваги перераховувати можна довго, втім, як і їхні недоліки. Але мова не про це, поговорити хотілося б про найпоширенішому ICQ-клієнта для Linix, про програму Kopete. Цей мессенджер працює за принципом систем, що підключаються, і легко вбудовується в робітниче середовище KDE. Використання принципу систем, що підключаються, означає, що дистрибутив Kopete установлюється не як щось єдине й неподільне, а у вигляді окремих модулів. Це значно спрощує роботу користувача із програмою. Установивши основний модуль програми можна при необхідності займатися її добудуванням, причому без переустановки вже завантажених модулів. Кожна частина Kopete окремо настроюється під ваші потреби, знову ж не зачіпаючи всього іншого. Ціль розроблювачі Kopete тут переслідували двояку, з однієї сторони вони орієнтувалися на те, щоб користувач міг максимально оптимізувати програму під свої потреби, а з іншої не забули й про невтомних програмістів, які також одержали зручний інструмент для розробки нових протоколів.

Мессенджер Kopete підтримує практично всі розповсюджені сьогодні протоколи: ICQ, AIM, IRC, Jabber, MSN та інші. Тому складно сказати, що аматори Linux тут себе чимсь обмежують, скоріше навпаки, вони мають право сказати так про тих, хто усе ще воліє користуватися «старими добрими» ICQ. Тут знову ж простежується робота із принципу систем, що підключаються, кожний з потрібних вам протоколів установлюється за допомогою окремого модуля. А виходить, немає необхідності завантажувати жорсткий диск, підключаючи всі й відразу.

Принцип прийому й відправлення повідомлень в Kopete такий же, як і в переважної більшості інтернет-пейджерів, на ньому зупинятися не будемо. Хотілося б відзначити цікаву й, безсумнівно, корисну функцію, що часто зустрічається в мессенджерів для Linux, і в тому числі в Kopete, це створення метаконтакта. Виглядає дана процедура в такий спосіб: якщо той самий контакт із вашого списку користується декількома службами IM, те всі вони будуть

зберігатися в одній папці з його ім'ям. Дуже зручна функція, так як сьогодні дійсно у своєму прагненні бути завжди на зв'язку багато хто користуються декількома обліковими записами в різних протоколах передачі текстових повідомлень, а об'єднучий метаконтакт дозволяє деяким чином забрати плутанину, що з'явилася у зв'язку із цим.

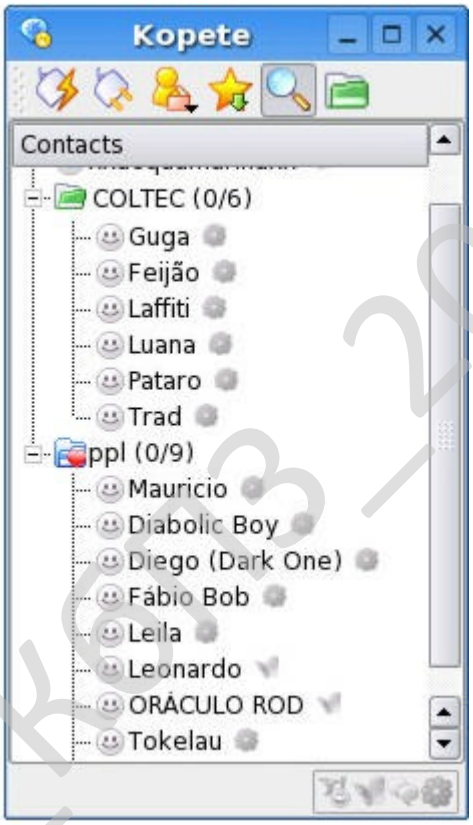

Рисунок 2.2 – Інтерфейс користувача Kopete

Унікальним образом реалізована робота зі списком контактів. Можливо в Kopete кожному абонентові з вашого списку привласнити свій власний значок. Наприклад, для тих, з ким ви спілкуєтеся найбільше, можна виставити ярлички, що виділяються із загальної маси, і навіть настроїти індивідуальні повідомлення. Використовуючи модулі, що підключаються, можна значно розширити функціонал програми. Так, є можливість ставити фільтри на повідомлення, що з'являються в діалоговому вікні й установлювати автоматичну заміну слів. Якщо

є необхідність, то підключенням спеціального модуля ваша переписка буде шифруватися.

Крім текстового спілкування, в Kopete також реалізовані голосовий зв'язок, і навіть можливість проводити відеоконференції. Для реалізації останніх вам знадобиться веб-камера, але навіть якщо спілкуватися за допомогою відеозображення вас не приваблює, камеру можна використовувати для іншої мети. А саме, у ролі своєрідного датчика руху, що буде відслідковувати, коли ви залишаєте робоче місце або вертаєтеся й у цьому зв'язку міняти статуси вашого клієнта. Дуже цікава функція, що змушує ще раз переконатися у всіх перевагах софту, розповсюджуваного з відкритим кодом.

Розроблювачі Kopete, знаючи про деяку специфіку програми, що випускається ними, передбачили два варіанти роботи з нею на різних рівнях складності. Починаючим користувачам краще познайомитися з версією дистрибутива, не потребуючих особливих втручань, а тим, хто вже має досвід, надається повний варіант із величезним вибором індивідуальних налаштувань.

Серед недоліків Kopete відзначається погана робота зі стандартним кодуванням Unicode, що може викликати деякі утруднення при спілкуванні з користувачами традиційних ICQ-клієнтів.

У підсумку хочеться відзначити, що Kopete не зрячи став самим популярним клієнтом серед користувачів Linux. Отут важливу роль зіграло модульна «багатоповерхова» налаштування програми по вашім розсуді й можливість підключати додаткові плагіни, виконані сторонніми розроблювачами. Звичайно, для впевненого користування програмою потрібний деякий досвід, але багатий функціонал додатка перекриває всі замічені недоліки.

LICQ

LICQ – програма для миттєвого обміну повідомленнями в інтернет за протоколом ICQ і XMPP, що працює на Linux і інших Unix-подібних системах. До достоїнств цього клієнта можна віднести можливість шифрування повідомлень на основі SSL за умови, що в приймаючої сторони також

установлений LICQ, mICQ або SIM. За допомогою додаткових модулів можна підключити шифрування на основі GPG.

Програма є вільною й розповсюджується під ліцензією GNU GPL. Програма являє собою демон, до якого за допомогою модулів можна підключати різні інтерфейси: Qt, консольний.

Однією з вічних проблем, що встають перед тими, хто вирішив працювати на Linux, є необхідність заміни звичних усім додатків. Це відноситься до текстових редакторів, різних медіаплеєрів, ну й, звичайно ж, до програм для обміну короткими текстовими повідомленнями через інтернет. Тому розроблювачі створюють додатки, які найчастіше є клонами традиційних програм для Windows, серед них можна назвати мессенджер LICQ.

Почав свій розвиток LICQ в 1998 році, практично повністю скопіювавши зовнішній вигляд і функціонала тодішньої версії програми ICQ. З тих пор LICQ стала з'являтися практично у всіх складаннях операційних систем Linux як інтернет-мессенджер, установлений за замовчуванням. Втім, стосовно до операційної системи Linux складно сказати, що додаток є обов'язковим і «вшито» у код, можливо, користуючись відкритістю оболонки, його вільно модифікувати й замінити на те, що вам підійде більше.

Одним з головних питань для тих, хто переходить із одних інтернет-пейджерів на інші, є перенос на нову платформу свого списку контактів. LICQ із цим завданням справляється, ваш список контактів легко імпортується сюди із серверів ICQ.

Незважаючи на те, що на перший погляд програма здається трохи обділеною з погляду зовнішнього вигляду, функціонал у неї в повному порядку. Присутні всі необхідні можливості, до яких звикли користувачі ICQ. Можливо набудовувати свій статус залежно від зайнятості або настрою, користуватися повноцінною історією повідомлень, модифікувати контакт-аркуш по вашім розсуді. Підтримується в LICQ не тільки передача текстових повідомлень, але й будь-яких типів файлів. Для полегшення й прискорення роботи із програмою можливо використовувати гарячі клавіші.

Рисунок 2.3 – Інтерфейс користувача LICQ

Після більше докладного знайомства з мессенджером LICQ стає зрозуміло, що й зовнішній вигляд його не так вуж безнадійний. Для аматорів усього нова й незвичайного тут передбачена зміна скінів, тому можна змінити стандартний і стиль, що знудив, оформлення, а також відповідно йому й іконки, на ваші персональні. Для багатьох додатків, використовуваних у середовищі Linux, причому не тільки для інтернет-мессенджерів, часто встає проблема невідповідності кодувань при контакті з іншими операційними системами. В LICQ такої проблеми немає, якщо з'явилися якісь проблеми із трансляцією тексту, то досить лише в налаштуваннях поміняти кодування, і проблема зважиться.

Таблиця 2.1 – Операційні системи, у яких мессенджери працюють без емуляції

Мессенджер	Microsoft Windows	Linux	Мессенджер	Microsoft Windows	Linux
AOL Instant Messenger	Так	Так	MindSpring	Так	Немає
aMSN	Так	Так	MySpaceIM	Так	Немає
BitlBee	Так	Так	Naim	Так	Так
BitWise IM	Так	Так	OpenWengo	Так	Так
Centericq	Так	Так	Paltalk	Так	Немає
Coccinella	Так	Так	Pandion	Так	Немає
Digsby (англ.)	Так	Так	Pidgin	Так	Так
Empathy	Немає	Так	Proteus	Немає	Немає
Exodus	Так	Немає	Psi	Так	Так
Fire	Так	Немає	QIP	Так	Немає
Gajim	Так	Так	QIP Infium	Так	Немає
GCN	Так	Немає	QIP	Так	Немає
GOIM	Так	Так	Qnext	Так	Так
Goofey	Так	Так	QQ	Так	Немає
Google Talk	Так	Так	QutIM	Так	Так
iChat	Так	Немає	R&Q	Так	Немає
ICQ	Так	Немає	SIM	Так	Так
IMVU	Так	Немає	Skype	Так	Так
Instantbird	Так	Так	talk	Так [4]	Так
Kadu	Так	Так	Tkabber	Так	Так
Kopete	Немає	Так	Trillian	Так	Немає
Trillian Astra Pro	Так	Немає	LICQ	Немає	Так
Windows Messenger	Так	Немає	mICQ	Так	Так
Winpopup LAN Messenger	Так	Немає	Meetro	Так	Немає
Windows Live Messenger	Так	Немає	Y!M	Так	Так
Microsoft Messenger для Mac	Немає	Немає	Zephyr	Так	Так

Серед не самих ходових, але теж дуже корисних функцій програми LICQ треба відзначити можливість відправлення SMS-повідомлень, використання

механізму авторизації й утаювання своєї IP-адреси. Пошук нових контактів можна здійснювати по досить широкому списку параметрів, що спрощує завдання, якщо інформація вам повністю не відома, але вийти на зв'язок необхідно.

Як і багато додатків для операційних систем Linux, мессенджер LICQ має модульну структуру й тому легко поповнює свій арсенал новими можливостями при установці відповідних плагінів. Також серед переваг цього клієнта часто згадують розширені можливості для шифрування повідомлень. Є два варіанти шифрування: перший на основі SSL, він буде коректно працювати, якщо в того, хто ваше повідомлення приймає, установлений клієнт, що підтримує цей механізм. Якщо вас не влаштовує перший варіант обробки, використовуючи додаткові модулі, можливо настроїти роботу за допомогою шифрування методом GPG.

Від використання LICQ залишається враження, що мессенджер цілком живучий, так як всі необхідні для спілкування функції присутні. Однак він не найкращим.

2.2 Обґрунтування вибору засобів для побудови системи та мови програмування

У зв'язку з тим, що сьогодні рівень складності програмного забезпечення дуже високий, розробка додатків Linux з використанням тільки якої-небудь мови програмування значно утрудняється. Програміст повинен затратити масу часу на рішення стандартних завдань по створенню багатовіконного інтерфейсу. Реалізація технології зв'язування й вбудовування об'єктів – OLE – зажадає від програміста ще більш складної роботи. Щоб полегшити роботу програміста практично всі сучасні компілятори з мови C++ містять спеціальні бібліотеки класів. Такі бібліотеки містять у собі практично весь програмний інтерфейс Linux і дозволяють користуватися при програмуванні засобами більш високого рівня, чим звичайні виклики функцій. За рахунок цього значно спрощується розробка

додатків, що мають складний інтерфейс користувача, полегшується підтримка технології OLE і взаємодія з базами даних. Сучасні інтегровані засоби розробки додатків Windows дозволяють автоматизувати процес створення додатка. Для цього використовуються генератори додатків. Програміст відповідає на питання генератора додатків і визначає властивості додатка – чи підтримує воно багатовіконний режим, технологію OLE, тривимірні органи керування, довідкову систему. Генератор додатків, створить додаток, що відповідає вимогам, і надасть вихідні тексти. Користуючись їм як шаблоном, програміст зможе швидко розробляти свої додатки. Подібні засоби автоматизованого створення додатків включені в компілятор Microsoft Visual C++ і називаються MFC AppWizard. Заповнивши кілька діалогових панелей, можна вказати характеристики додатка й одержати його тексти, постачені великими коментарями. MFC AppWizard дозволяє створювати одновіконні й багатовіконні додатки, а також додатки, що не мають головного вікна, – замість нього використовується діалогова панель. Можна також включити підтримку технології OLE, баз даних, довідкової системи. Звичайно, MFC AppWizard не всесильний. Прикладну частину додатка програмістові прийдеться розробляти самостійно. Вихідний текст додатка, створений MFC AppWizard, стане тільки основою, до якої потрібно підключити інше. Але працюючий шаблон додатка – це вже половина всієї роботи. Вихідні тексти додатків, автоматично отриманих від MFC AppWizard, можуть становити сотні рядків тексту. Набір його вручну був би дуже стомлюючий. Потрібно відзначити, що MFC AppWizard створює тексти додатків тільки з використанням бібліотеки класів MFC (Microsoft Foundation Class library). Тому тільки вивчивши мову C++ і бібліотеку MFC, можна користуватися засобами автоматизованої розробки й створювати свої додатки в найкоротший термін. Як уже згадувався, MFC – це базовий набір (бібліотека) класів, написаних мовою C++ і призначених для спрощення й прискорення процесу програмування для Windows. Бібліотека містить багаторівневу ієрархію класів, що нараховує близько 200 членів. Вони дають можливість створювати Windows-додатки на базі об’єктно-орієнтованного підходу. З погляду програміста, MFC являє собою каркас, на основі якого можна писати програми для Windows. Бібліотека MFC розроблялася для спрощення

завдань, що стоять перед програмістом. Як відомо, традиційний метод програмування під Windows вимагає написання досить довгих і складних програм, що мають ряд специфічних особливостей. Зокрема, для створення тільки каркаса програми таким методом знадобиться близько 75 рядків коду. У міру ж збільшення складності програми її код може досягати воістину неймовірних розмірів. Однак та ж сама програма, написана з використанням MFC, буде приблизно в три рази менше, оскільки більшість приватних деталей приховано від програміста.

Одною з основних переваг роботи з MFC є можливість багаторазового використання того самого коду. В зв'язку з тим, що бібліотека містить багато елементів, загальних для всіх Windows-додатків, немає необхідності щораз писати їх заново. Замість цього їх можна просто успадковувати (говорячи мовою об'єктно-орієнтованного програмування). Крім того, інтерфейс, забезпечуваний бібліотекою, практично незалежний від конкретних деталей, його що реалізують. Тому програми, написані на основі MFC, можуть бути легко адаптовані до нових версій Linux (на відміну від більшості програм, написаних звичайними методами). Ще однією істотною перевагою MFC є спрощення взаємодії із прикладним програмним інтерфейсом (API) Windows. Будь-який додаток взаємодіє з Linux через API, що містить кілька сотень функцій. Значний розмір API утрудняє спроби зрозуміти й вивчити його цілком. Найчастіше навіть складно простежити, як окремі частини API зв'язані один з одним! Але оскільки бібліотека MFC поєднує (шляхом інкапсуляції) функції API у логічно організовану безліч класів, інтерфейсом стає значно легше управляти.

Оскільки MFC являє собою набір класів, написаних мовою C++, тому програми, написані з використанням MFC, повинна бути в той же час програмами на C++. Для цього необхідно володіти відповідними знаннями. Для початку необхідно вміти створювати власні класи, розуміти принципи спадкування й уміти перевизначати віртуальні функції. Хоча програми, що використовують бібліотеку MFC, звичайно не містять занадто специфічних елементів з арсеналу C++, для їхнього написання проте потрібні солідні знання в даній області.

2.3 Розгорнута постановка завдання

Згідно з технічним завданням на випускну кваліфікаційну роботу за першим (бакалаврським) рівнем вищої освіти, реалізації підлягає програмне забезпечення, яке призначено для системи кібербезпеки захищеного обміну інформацією у мережі під керуванням ОС Linux.

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

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

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

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

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

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

е) провести розрахунки по визначенню економічної ефективності розробленої системи;

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

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

3 ОПИС І ОБҐРУНТУВАННЯ ПРОЕКТНИХ РІШЕНЬ

3.1 Опис функціонування системи

Нехай є локальна TCP/IP – мережа під керівництвом ОС Linux. Потрібно написати програму для системи обміну повідомленнями між комп'ютерами цієї мережі. Програма повинна мати функціональність, близьку до відомої системи ICQ, що забезпечує миттєвий обмін текстовою інформацією між різними користувачами в мережі.

Аналіз аналогічних проектів з відкритими кодами в інтернеті показав, що вони мають незручний інтерфейс. Відсутність проектної документації робить практично неможливою їхню модифікацію.

У локальних мережах використовуються комп'ютери під керуванням різних операційних систем. Тому програма розробляється для операційної системи (ОС) Linux.

При цьому програма під ОС Linux проектується із двох частин – сервера (демона, реалізованого у вигляді кінцевого автомата) і клієнта (графічного інтерфейсу), написаного традиційним шляхом.

На відміну від традиційних клієнт-серверних додатків у даній роботі взаємодія сервера й клієнта відбувається в рамках однієї машини. При взаємодії між програмами по мережі обидві сторони є рівноправними.

Для реалізації сервера використовується SWITCH-технологія [1]. Сервер і клієнт реалізовані мовою Visual C++ [2, 3].

Опис програми

Дана програма реалізована із двох частин: сервера, що є демоном (daemon) – резидентною програмою в Linux-системах, не зв'язанj. з жодним користувальницьким сеансом [4], і клієнта – графічного інтерфейсу користувача.

Демонами в Linux традиційно називаються процеси, які не взаємодіють із користувачем прямо. У процесу-демона немає керуючого термінала й немає, відповідно, користувальницького інтерфейсу. Для керування демонами доводиться використовувати інші програми. Сама назва «демони» виникла завдяки тому, що багато процесів цього типу більшу частину часу проводять чекаючи якоїсь події. Коли ця подія наступає, демон активізується, виконує свою роботу й знову засипає чекаючи події. Слід зазначити, що багато демонів, такі як, наприклад, Web-сервер або сервер баз даних, можуть відбирати на себе практично весь процесорний час і інші ресурси системи. Такі демони набагато більше працюють, чим сплять. Область застосування демонів – створення таких додатків, які можуть, і повинні, виконуватися без участі користувача. Звичайно це різного роду сервери. Проте, демони задіють багато важливих елементів системи й розуміння принципів роботи демонів сприяє розумінню принципів роботи Linux у цілому.

Демон, реалізований у бакалаврському проекті, виконує наступні функції:

– прийом і обробку повідомлень із мережі з наступним відправленням клієнтові;

– прийом і обробку повідомлень від клієнта з наступним відправленням у мережу;

– забезпечення роботи додатків при завантаженому трафіку.

Розглянемо докладніше схему роботи демона. У ньому для зв'язку із зовнішнім миром використовуються два сокету – мережний Ipv4 сокет (AF_INET) і локальний сокет (AF_Linux).

Мережний сокет призначений для зв'язку з додатками, що працюють на інших хостах. Через нього здійснюється відправлення повідомлень із комп'ютера, на якому запущений демон, у мережу. Передбачається, що одержувач і відправник перебувають в одній мережі. Цей же сокет забезпечує прийом повідомлень, призначених для клієнта, запущеного на даному хості. Нехай сокет, наприклад, пов'язаний з портом 3538.

Локальний сокет призначений для зв'язку із клієнтським додатком. Через нього здійснюється відправлення й премо повідомлень від клієнта. Нехай сокет пов'язаний з файлом /tmp/schat3538.

При одержанні повідомлення демон робить його перетворення в зовнішній/внутрішній формат для наступного відправлення через відповідний сокет. Таке поводження демона досить зручно для клієнтського додатка, так як дозволяє йому не піклуватися про мережну складову при взаємодії з віддаленим додатком.

Клієнт є другою складовою частиною програми. Він забезпечує безпосередню взаємодію з користувачем. Клієнт є сполучною ланкою між користувачем і серверною частиною, що реалізує мережну взаємодію.

Серверна частина

Основою демона є кінцевий автомат. Його граф переходів наведений на рисунку 3.1. У ньому використовуються наступні позначення.

Стани:

– Очікування.

– Прийом локального повідомлення.

– Перетворення в мережний формат.

– Відправлення повідомлення в мережу.

– Прийом повідомлення з мережі.

– Перетворення в локальний формат.

– Відправлення локального повідомлення.

– Вихід.

Події:

– e0 – у локальному сокеті з'явилося повідомлення;

– e1 – у мережному сокеті з'явилося повідомлення;

– e2 – у локальному сокеті повідомлень не залишилося;

– e3 – у мережному сокеті повідомлень не залишилося;

– e4 – перевищений ліміт по кількості повідомлень, отриманих підряд з локального сокету;

– e5 – перевищений ліміт по кількості повідомлень, отриманих підряд з мережного сокету;

– e6 – завершення роботи клієнта або аварійна ситуація;

– e7 – успішний прийом локального повідомлення;

– e8 – успішне перетворення в мережний формат;

– e9 – успішне відправлення повідомлення в мережу;

– e10 – успішний прийом повідомлення з мережі;

– e11 – успішне перетворення в локальний формат;

– e12 – успішне відправлення локального повідомлення.

Дії:

– z0 – очікування приходу повідомлення в один із сокетів. При завершенні процедури z0 залежно від того, у який сокет повідомлення прийшло, генерується подія e0 або e1;

– z1 – одержання повідомлення з локального сокету. Якщо повідомлення в сокеті не виявилося, то генерується подія e2;

– z2 – перетворення локального повідомлення для наступного відправлення його в мережу;

– z3 – відправлення повідомлення в мережу. Якщо кількість повідомлень, відправлених підряд, перевищує встановлене число, то генерується подія e4;

– z4 – одержання повідомлення з мережного сокету. Якщо повідомлення в сокеті не виявилося, то генерується подія e3;

– z5 – перетворення мережного повідомлення для наступного відправлення його локальному клієнтові;

– z6 – відправлення повідомлення локальному клієнтові. Якщо кількість повідомлень, відправлених підряд, перевищує встановлене число, то генерується подія e5.

– z7 – завершення роботи демона.

Автомат очікує приходу повідомлення. Після його приходу, автомат робить перетворення повідомлення в зовнішній/внутрішній формат і відправлення його по відповідний сокету. Залежно від того, звідки прийшло повідомлення, генерується одне з подій e0 або e1. Відзначимо, що після відправлення повідомлення автомат не переходить у режим очікування. Він знову намагається одержати повідомлення з того ж джерела, і лише переконавшись у його відсутності, переходить у режим очікування.

Автомат також забезпечує захист демона від перевантажень при надмірному трафіку. У випадку одержання з мережі більше N повідомлень підряд, генерується подія e5, по якому автомат примусово переходить на прийом повідомлень із локального сокету. Аналогічно, у випадку одержання підряд більше M повідомлень із локального сокету, генерується подія e4. По цій події автомат примусово переходить на прийом повідомлень із мережі. Події e2 і e3 генеруються у випадку відсутності повідомлень у локальному й мережному сокеті відповідно.

На рисунку 3.1 наведений граф переходів, що описує роботу демона.

Приклад лога демона

```
Демон Запущений
Чекаємо з'єднання із клієнтом...  готове
A0 перейшов у стан 0
Очікування повідомлення
Подія el
A0 перейшов у стан 4
У мережному сокеті з'явилося повідомлення Одержання повідомлення з мережного сокету
Подія el0
A0 перейшов у стан 5
Успішний прийом повідомлення з мережі Конвертування повідомлення у внутрішній формат
Подія ell
A0 перейшов у стан 6 повідомлення =  'Привіт!!!'
Успішне перетворення в локальний формат Відправлення повідомлення через локальний сокет
адреса відправника    192.168.0.72
Подія e12
```

```
        A0 перейшов у стан 4
        Успішне відправлення локального повідомлення Одержання повідомлення з
мережного сокету
        Подія e3
        A0 перейшов у стан 0
        У мережному сокеті Очікування повідомлення
        повідомлень не залишилося
        Подія e0
        A0 перейшов у стан 1
        У локальному сокеті з'явилося повідомлення Одержання повідомлення з
локального сокету
        Подія e7
        A0 перейшов у стан 2
        Успішний прийом локального повідомлення Конвертування повідомлення в
мережний формат
        Подія e8 :
        A0 перейшов у стан 3      :
        повідомлення =  'Привіт. Як справи?'
        адреса призначення = 192.168.0.72
        Успішне перетворення в мережний формат Відправлення повідомлення через
мережний сокет
        Подія e9
        A0 перейшов у стан 1
        Успішне відправлення повідомлення в мережу Одержання повідомлення з
локального сокету
        Подія e2
        A0 перейшов у стан 0
        У локальному сокеті повідомлень не залишилося Очікування повідомлення
        Подія e1
        A0 перейшов у стан 4
        У мережному сокеті з'явилося повідомлення Одержання повідомлення з
мережного сокету
        Подія e10
        A0 перейшов у стан 5
        Успішний прийом повідомлення з мережі Конвертування повідомлення у
внутрішній формат
        Успішне перетворення в локальний формат Відправлення повідомлення через
локальний сокет спілкування =  'Зустрінемося в 5 годин. Ти зможеш?'
        Подія e11
        A0 перейшов у стан 6
        адреса відправника    192.168.0.72
        Подія e12
```

```
        A0 перейшов у стан 4
        Успішне відправлення локального повідомлення Одержання повідомлення з
мережного сокету
        Подія e3
        A0 перейшов у стан 0
        У мережному сокеті повідомлень не залишилося Очікування повідомлення
        Подія e0
        :       У локальному сокеті з'явилося повідомлення
        A0 перейшов у стан 1       :
        Одержання повідомлення з локального сокету
        Подія e7
        A0 перейшов у стан 2
        Успішний прийом локального повідомлення Конвертування повідомлення в
мережний формат
        Подія e8 :
        A0 перейшов у стан 3       : повідомлення =  'Так, звичайно!'
        Успішне перетворення в мережний формат Відправлення повідомлення через
мережний сокет
        адреса призначення
        192.168.0.72
        Подія e9
        A0 перейшов у стан 1
        Успішне відправлення повідомлення в мережу Одержання повідомлення з
локального сокету
        Подія e2
        A0 перейшов у стан 0
        У локальному сокеті Очікування повідомлення повідомлень не залишилося
        Подія e1
        A0 перейшов у стан 4
        У мережному сокеті з'явилося повідомлення Одержання повідомлення з
мережного сокету
        Подія e10
        A0 перейшов у стан 5
        Успішний прийом повідомлення з мережі Конвертування повідомлення у
внутрішній формат
        Подія ell
        A0 перейшов у стан 6
        Успішне
        перетворення в локальний формат
        повідомлення =  'ok. До зустрічі... адреса відправника    192.168.0.72
        Відправлення повідомлення через локальний сокет
        !
```

```
        Подія e12
        A0 перейшов у стан 4
        Успішне відправлення локального повідомлення Одержання повідомлення з
мережного сокету
        Подія e3
        A0 перейшов у стан 0
        У мережному сокеті Очікування повідомлення
        повідомлень не залишилося
        Подія e0
        A0 перейшов у стан 1
        У локальному сокеті з'явилося повідомлення Одержання повідомлення з
локального сокету
        Подія e7
        A0 перейшов у стан 2
        Успішний прийом локального повідомлення Конвертування повідомлення в
мережний формат
        Подія e8 :
        A0 перейшов у стан 3
        повідомлення =  'Пока'
        адреса призначення = 192.168.0.72
        Успішне перетворення в мережний формат Відправлення повідомлення через
мережний сокет
        Подія e9
        A0 перейшов у стан 1
        Успішне відправлення повідомлення в мережу Одержання повідомлення з
локального сокету
        Подія e2
        A0 перейшов у стан 0
        У локальному сокеті повідомлень не залишилося Очікування повідомлення
        Подія e6 :    Завершення роботи клієнта
        Демон закінчив роботу
```

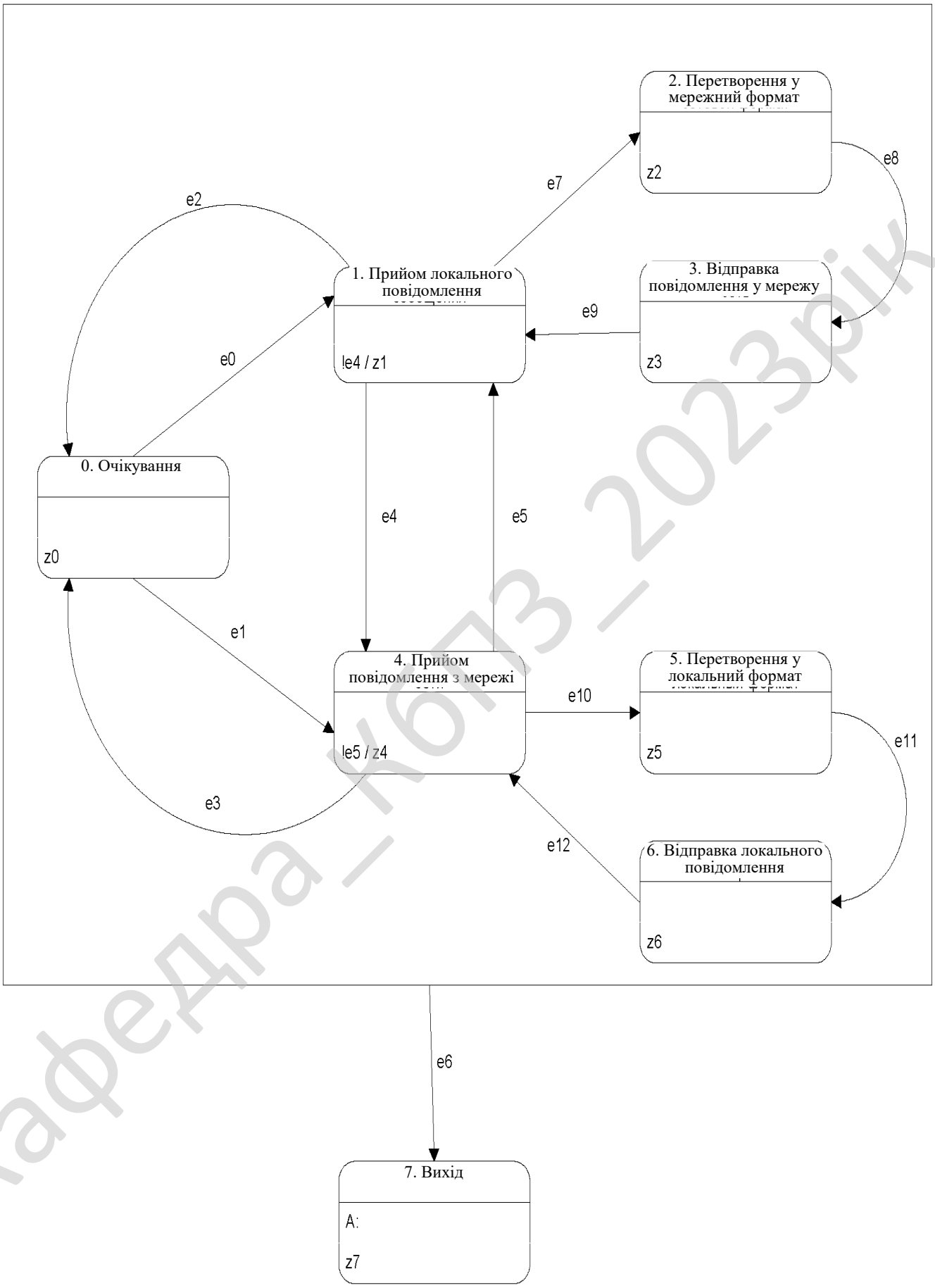

Рисунок 3.1 – Граф переходів, що описує роботу демона

Клієнтська частина

Клієнтська частина Linux-додатку демонструє роботу демона і є його логічним доповненням. Вона дозволяє користувачеві посилати й приймати повідомлення з мережі, зберігати список контактів, призначати імена користувачів. Користувальницький інтерфейс програми написаний із застосуванням бібліотеки Qt. Він досить схожий з популярним ICQ клієнтом під Linux LICQ. Тому освоєння даної програми не повинне скласти утруднень для Linux-користувачів. Інтерфейс програми інтуїтивно зрозумілий і простий у використанні.

Qt – крос-платформний інструментарій розробки ПЗ мовою програмування C++. Є також «прив'язки» до багатьох інших мов програмування: Python – PyQt, PySide; Ruby – QtRuby; Java – Qt Jambi; PHP – PHP-Qt і інші. Дозволяє запускати написане з його допомогою ПЗ у більшості сучасних операційних систем шляхом простої компіляції програми для кожної ОС без зміни вихідного коду. Містить у собі всі основні класи, які можуть знадобитися при розробці прикладного програмного забезпечення, починаючи від елементів графічного інтерфейсу й закінчуючи класами для роботи з мережею, базами даних і XML. Qt є повністю об’єктно-орієнтованим, легко розширюваним і підтримуючийся технікою компонентного програмування.

Відмінна риса Qt від інших бібліотек – використання Meta Object Compiler (MOC) – попередньої системи обробки вихідного коду (у загальному-то, Qt – це бібліотека не для чистого C++, а для його особливого варіанту, з якого й «переводить» MOC для наступної компіляції будь-яким стандартним C++ компілятором). MOC дозволяє в багато разів збільшити міць бібліотек, уводячи такі поняття, як слоти й сигнали. Крім того, це дозволяє зробити код більше лаконічним. Утиліта MOC шукає в заголовних файлах на C++ описи класів, що містять макрос Q_OBJECT, і створює додатковий вихідний файл на C++, що містить мета-об'єктний код.

Qt дозволяє створювати власні плагіни й розміщати їх безпосередньо в панелі візуального редактора. Також існує можливість розширення звичної функціональності віджетів, пов'язаної з розміщенням їх на екрані, відображенням, перемальовуванням при зміні розмірів вікна.

Qt комплектується візуальним середовищем розробки графічного інтерфейсу «Qt Designer», що дозволяє створювати діалоги й форми «мишею» (у режимі WYSIWYG). У поставці Qt є «Qt Linguist» – графічна утиліта, що дозволяє спростити локалізацію й переклад вашої програми на багато мов; і «Qt Assistant» – довідкова система Qt, що спрощує роботу з документацією по бібліотеці, а також дозволяє створювати крос-платформну довідку для розроблювального на основі Qt ПЗ. Починаючи з версії 4.5.0 у комплект Qt включене середовище розробки «Qt Creator», що містить у собі редактор коду, довідку, графічні засоби «Qt Designer» і можливість налагодження додатків. «Qt Creator» може використовувати GCC або Microsoft VC++ як компілятор і GDB у якості відладника. Для Windows версій бібліотека комплектується компілятором, заголовними й об'єктними файлами MinGW.

Бібліотека розділена на кілька модулів, для четвертої версії бібліотеки це:

– QtCore – класи ядра бібліотеки, використовувані іншими модулями.

– QtGui – компоненти графічного інтерфейсу.

– QtNetwork – набір класів для мережного програмування. Підтримка різних високорівневих протоколів може мінятися від версії до версії. У версії 4.2.x присутні класи для роботи із протоколами FTP і HTTP. Для роботи із протоколами TCP/IP призначені такі класи, як QTcpServer, QTcpSocket для TCP і QUdpSocket для UDP.

– QtOpenGL – набір класів для роботи з OpenGL.

– QtSql – набір класів для роботи з базами даних з використанням мови структурованих запитів SQL. Основні класи даного модуля у версії 4.2.x: QSqlDatabase – клас для надання з'єднання з базою, для роботи з якою-небудь конкретною базою даних вимагає об'єкт, успадкований від класу QSqlDriver –

абстрактного класу, що реалізується для конкретної бази даних і може вимагати для компіляції SDK бази даних. Наприклад, для складання драйвера під базу даних FireBird/InterBase потребує .h файли й бібліотеки статичної лінковки, що входять у комплект поставки даної БД.

– QtScript – класи для роботи з Qt Scripts.

– QtSvg – класи для відображення й роботи з даними Scalable Vector Graphics(SVG).

– QtXml – модуль для роботи з XML, підтримується SAX і DOM моделі роботи;

– QtDesigner – класи створення розширень QtDesigner'а для своїх власних віджетів.

– QtUiTools – класи для обробки в додатку форм Qt Designer.

– QtAssistant – довідкова система.

– Qt3Support – модуль із класами, необхідними для сумісності з бібліотекою Qt версії 3.x.x.

– QtTest – модуль для роботи з UNIT тестами.

– QtWebKit – модуль WebKit, інтегрований в Qt і доступний через її класи;

– QtXmlPatterns – модуль для підтримки XQuery 1.0 і XPath 2.0.

– Phonon – модуль для підтримки відтворення й запису відео й аудіо, як локально, так і з пристроїв і по мережі.

– QtCLucene – модуль для підтримки повнотекстового пошуку, застосовується в новій версії Assistant в Qt 4.4.

– ActiveQt – модуль для роботи з Active і COM технологіями для Qt-розроблювачів під Windows.

– QtDeclarative – модуль, що надає декларативний фреймворк для створення динамічних, що налаштовуються користувальницьких інтерфейсів.

Також реалізована технологія WoC – widgets on canvas, за допомогою якої реалізована Plasma в KDE 4.1, Буде можливим використовувати віджети

бібліотеки Qt прямо в аплетах. Забезпечує розташування віджетів на QGraphicsView з можливістю масштабування й різних графічних ефектів.

Бібліотека використовує власний формат проекту, іменований .pro файлом, у якому зібрана інформація про те, які файли будуть скомпільовані, по яких шляхах шукати заголовні файли й багато іншої інформації. Згодом за допомогою утиліти qmake з них виходять makefile для make-утиліти компілятора. Також є можливість роботи за допомогою інтеграторів з Microsoft Visual Studio. Зовсім недавно стала доступна інтеграція в Eclipse для версії бібліотеки 4.x.x.

3.2 Розробка структурної схеми

Структурна схема системи зображена на рисунку 3.2. З неї ми бачимо, що програмне забезпечення системи кібербезпеки захищеного обміну інформацією у мережі під керуванням ОС Linux, складається з наступних структурних блоків:

– Двигун програмного забезпечення системи кібербезпеки захищеного обміну інформацією у мережі під керуванням ОС Linux.

– Структурний блок підтримки протоколів обміну інформацією. До них відносяться наступні:

- AIM;
- ICQ;
- Windows Live Messenger (раніше MSN Messenger);
- Yahoo! Messenger;
- IRC;
- XMPP (Google Talk й інші XMPP-сумістні служби);
- Bonjour (раніше Rendezvous);
- Novell GroupWise Messenger;
- Lotus Sametime;
- Gadu-Gadu;

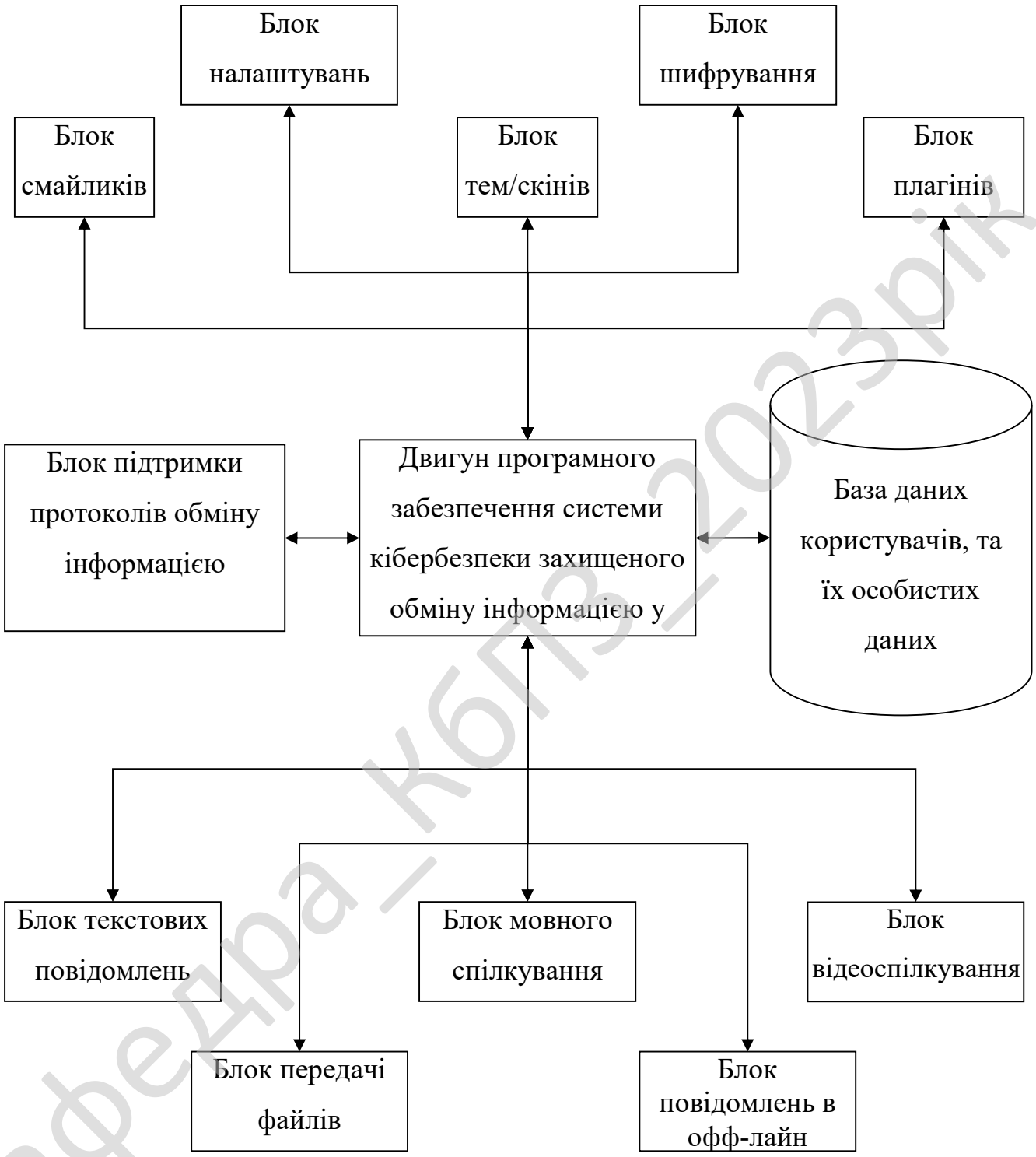

Рисунок 3.2 – Структурна схема системи

– Skype;

– QQ.

– База даних користувачів, та їх особистих даних.

– Блок налаштувань.

– Блок шифрування.

– Блок передачі файлів.

– Блок смайликів.

– Блок тем/скінів.

– Блок плагінів.

– Блок текстових повідомлень

– Блок мовного спілкування.

– Блок відеоспілкування.

– Блок повідомлень в офф-лайн.

3.3 Розробка функціональної схеми

На рисунку 3.3 зображена функціональна схема системи. Нижче розглянемо її більш докладно.

Функціональна схема програмного продукту, розробленого, у результаті виконання бакалаврського проектування, складається із взаємодії наступних функціональних блоків:

– Блок інтерфейсу користувача.

– Блок поділу контактів по групах.

– Блок різних тем та скинів для програми.

– Блок повного налаштування інтерфейсу: колір, шрифт, іконки й інше.

– Блок списку контактів, що має повну функціональність. Відображення он-лайн/офф-лайн користувачів зі значками й всією необхідною інформацією.

– Блок відправлення й прийому повідомлень он-лайн/офф-лайн між користувачами через сервер.

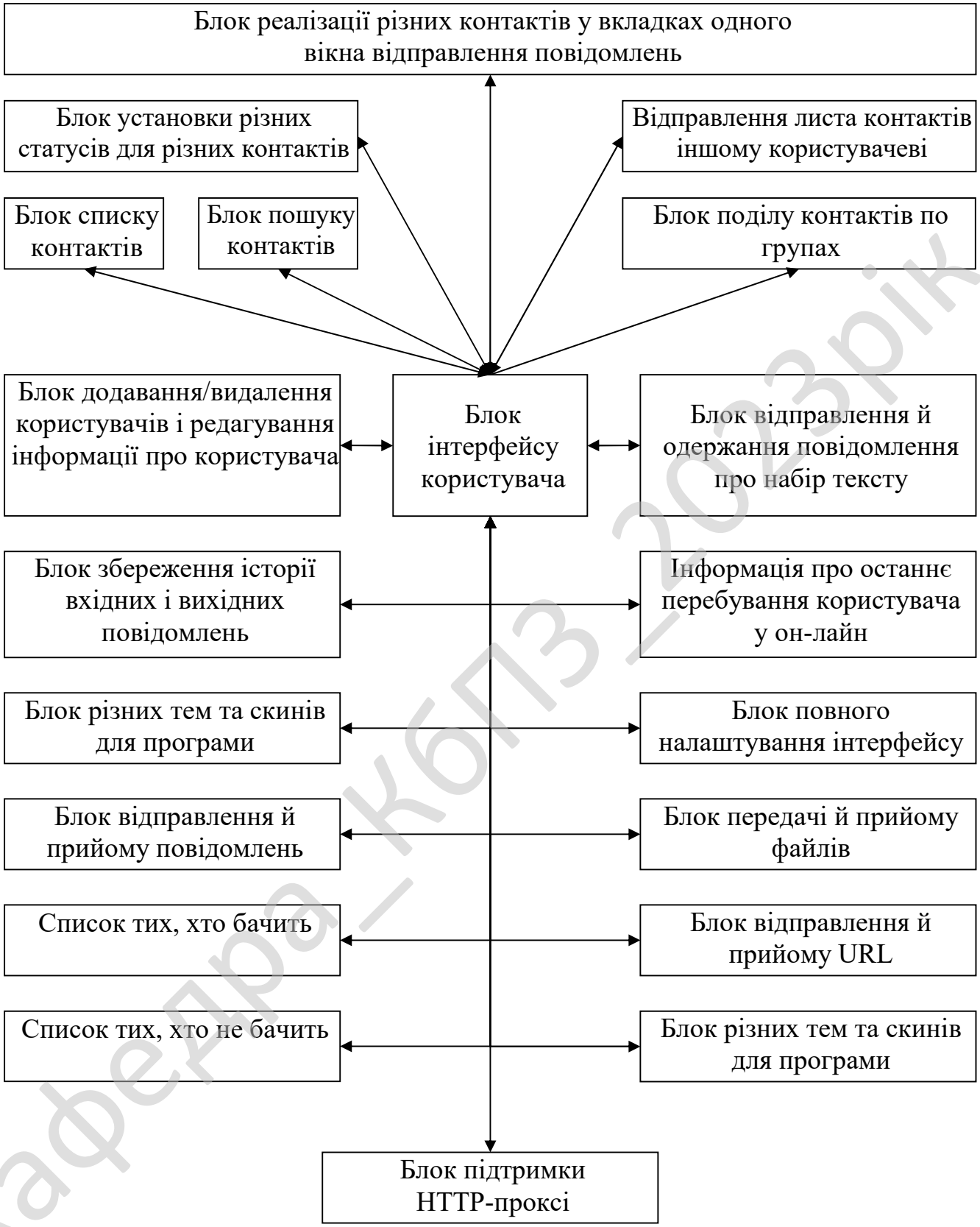

Рисунок 3.3 – Функціональна схема системи

– Блок реалізації різних контактів у вкладках одного вікна відправлення повідомлень.

– Блок передачі й прийому файлів від інших користувачів, у тому числі й мультимедійних.

– Блок відправлення й прийому URL.

– Блок установки різних статусів для різних контактів.

– Блок збереження історії вхідних і вихідних повідомлень.

– Блок додавання/видалення користувачів і редагування інформації про користувача.

– Список тих, хто бачить – будьте видимі тільки для деяких користувачів.

– Список тих, хто не бачить – будьте не видимі тільки для деяких користувачів.

– Інформація про останнє перебування користувача у он-лайн.

– Відправлення листа контактів іншому користувачеві.

– Блок пошуку контактів по e-mail/uin/alias/інформації про користувача.

– Блок відправлення й одержання повідомлення про набір тексту.

– Блок підтримки HTTP-проксі.

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

3.4 Розробка діаграми процесів

Діаграма взаємодії процесів системи, розробленої у результаті виконання бакалаврського проектування, наведена на рисунку 3.4. Першим процесом, який запускається у системі, є процес відкриття стартового вікна. Цей процес взаємодіє з наступними процесами:

– Процес реєстрації нового користувача.

– Процес авторизації користувача.

Рисунок 3.4 – Діаграма взаємодії процесів

Процес авторизації користувача взаємодіє з процесом підключення до серверу обміну повідомленями. Цей процес взаємодіє з процесом відкриття основного вікна програми.

Процес відкриття основного вікна програми взаємодіє з наступними процесами:

– Процес входу під іншим обліковим записом.

– Процес зміни статусу.

– Процес завантаження списку контактів.

– Процес відключення від серверу обміну повідомленями, який є завершальним.

Процес завантаження списку контактів взаємодіє з наступними процесами:

– Процес редагування списку контактів.

– Процес натиснення на одному з контактів.

Процес редагування списку контактів взаємодіє з наступними процесами:

– Процес видалення контактів.

– Процес додавання нових контактів.

– Процес пошуку нових контактів.

– Процес створення груп та занесення у них контактів.

Процес натиснення на одному з контактів взаємодіє з процесом відкриття вікна повідомлень.

Цей процес взаємодіє з наступними процесами:

– Процес перегляду історії повідомлень.

– Процес обміну текстовими повідомленями.

Останій процес взаємодіє з наступними процесами:

– Процес створення повідомлень.

– Процес читання надісланих повідомлень.

Процес створення повідомлень взаємодіє з наступними процесами:

– Процес додаваня смаглів та цитування.

– Процес відправки повідомлень.

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

4 РЕАЛІЗАЦІЯ РОБОТИ. РОЗРАХУНКИ І ЕКСПЕРИМЕНТАЛЬНІ ДАНІ, ЩО ПІДТВЕРДЖУЮТЬ ВІРНІСТЬ ПРОЕКТНИХ ТА ПРОГРАМНИХ РІШЕНЬ

4.1 Блок-схеми та опис алгоритмів функціонування системи

На рисунку 4.1 наведено блок-схему основної програми. Її робота складається з виконання наступних кроків.

Спершу відбувається виведення основного вікна програми. Після цього користувач обирає чи треба зробити нову реєстрацію.

Якщо так, то користувач реєструє новий обліковий запис та вводить логін й пароль.

Якщо авторизація пройшла успішно, то відбувається виведення основного вікна програми.

У іншому випадку відбувається виведення повідомлення про помилку та повідомлення про можливість реєстрації або відновлення пароля.

Після виведення головного вікна програми відбувається заповнення анкети користувача.

Якщо потібно змінити статус, то відбувається встановлення/зміна статусу користувача й виведення списку контактів, та їх статусів.

Наступною операцією, яку може зробити користувач, є редагування списку контактів.

Якщо він хоче редагувати список, тоді відбувається пошук, додавання, видалення контактів та створення груп й занесення в них контактів.

Якщо користувач натискує на контакт тоді запускається підпрограма обміну текстовими повідомленями.

Якщо користувач завершує діалог, то він визначає, чи буде далі розмовляти по інтернету, чи ні.

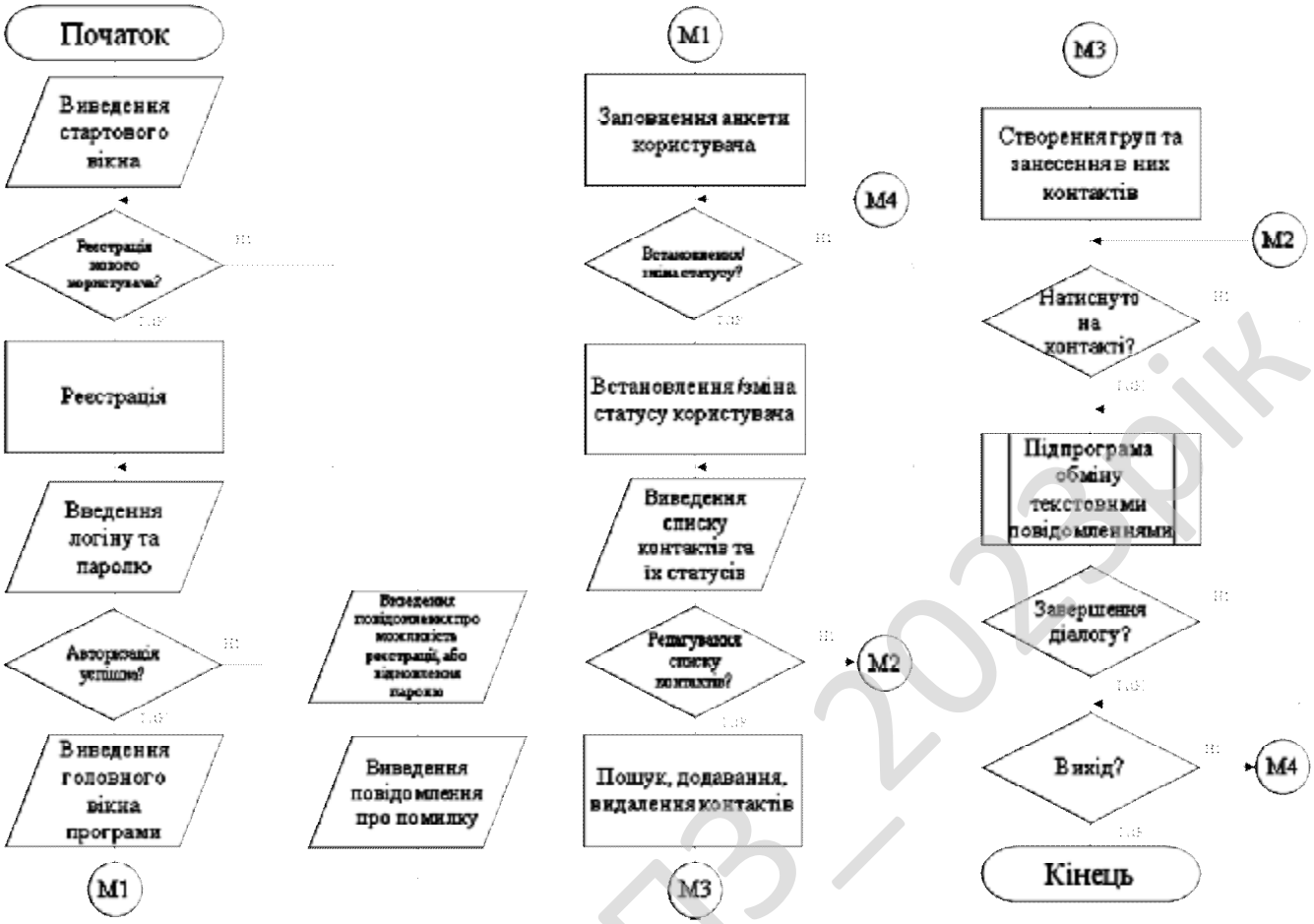

Рисунок 4.1 – Блок-схема роботи основної програми

На рисунку 4.2 зображено блок-схему роботи підпрограми обміну текстовими повідомленнями. Вона працює наступним чином.

Спершу відбувається виведення вікна повідомлень.

Після цього користувач обирає обмін тестовими повідомленнями.

Наступним кроком є перевірка надісланих повідомлень. Якщо вони є, тоді відбувається виведення цих повідомлень.

Якщо користувач обирає створювання повідомлення, то він його набирає на клавіатурі, при цьому на екран виводиться тест повідомлення.

Крім того існує функція цитування усього повідомлення, або його частини.

Якщо він обирає цитування усього повідомлення, тоді відбуваються наступні дії:

– Виділяється частина тесту в історії повідомлень.

– Відбувається цитування виділеного фрагменту тексту з історії повідомлень.

Якщо при наборі тесту користувач обирає функцію додавання смайлу, тоді відбувається послідовне виконання наступних дій:

– Вибір смайлу зі списку.

– Додавання в текст повідомлення вибраного файлу.

Якщо користувач бажає переглянути історію повідомлень, тоді відбувається виведення історії повідомлень.

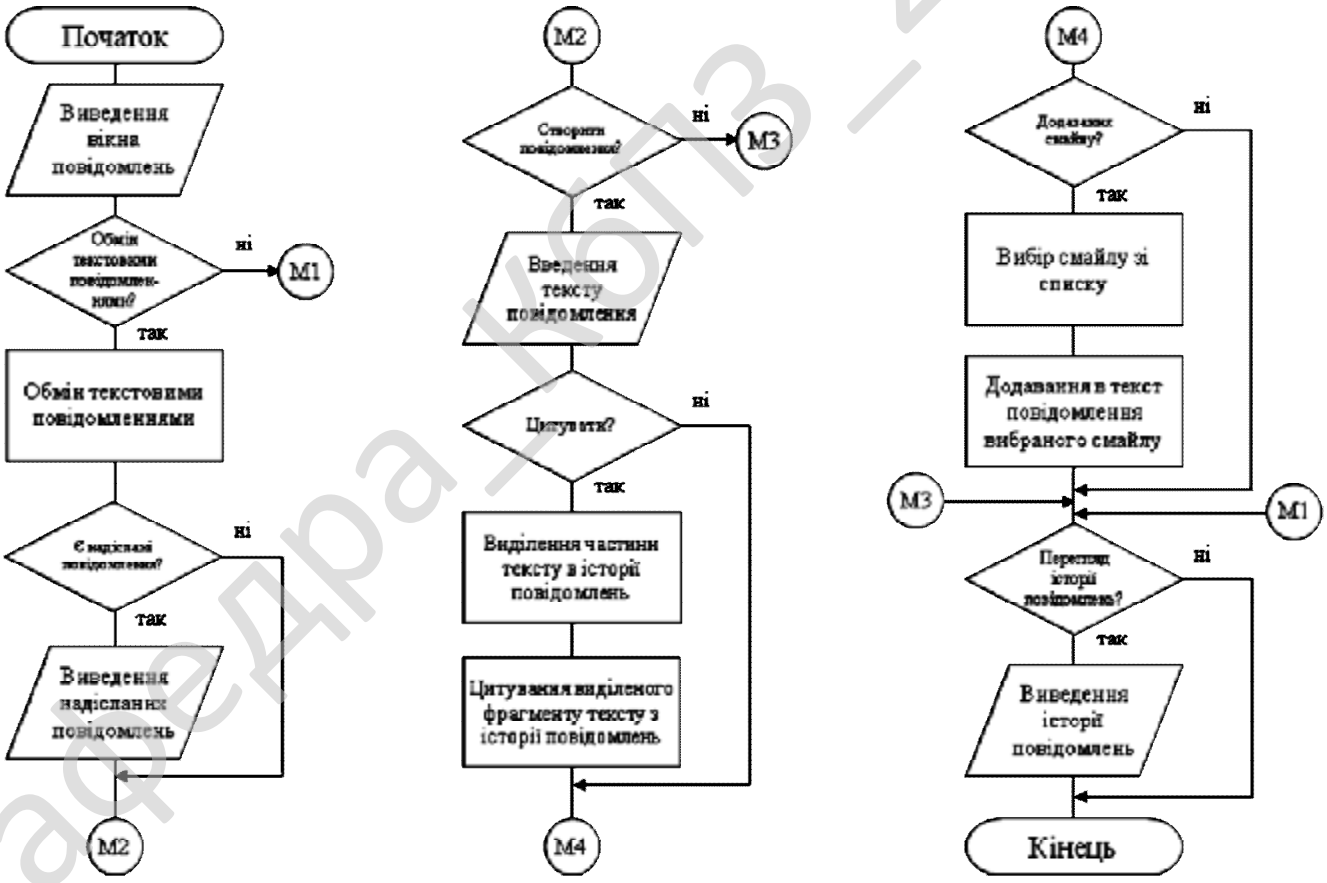

Рисунок 4.2 – Блок-схема роботи підпрограми обміну текстовими повідомленнями

Приведемо код демона, який реалізує мережний сервер системи кібербезпеки захищеного обміну інформацією у мережі під керуванням ОС Linux, здатний приймати запити клієнтів і повертати відповіді. Опишемо вихідний код (функції main() нашого демона).

```
volatile sig_atomic_t   gGracefulShutdown=0;
volatile sig_atomic_t   gCaughtHupSignal=0;
int               gLockFileDesc=-1;
int               gMasterSocket=-1;
const int         gserver_peredachi_informaciiPort=30333;
const           char           *const           gLockFilePath           =
"/var/run/server_peredachi_informacii.pid"; // pid-файл серверу
int main(int argc,char *argv[])
{
      int                                   result;
      pid_t                                 daemonPID;
      if (argc > 1) {
            int fd, len;
            pid_t pid;
            char pid_buf[16];
            if ((fd = open(gLockFilePath, O_RDONLY)) < 0)
            {
                  perror("Заблокований файл не знайдений. Може бути сервер
не запущений?");
                  exit(fd);
            }
            len = read(fd, pid_buf, 16);
            pid_buf[len] = 0;
            pid = atoi(pid_buf);
            if(!strcmp(argv[1], "stop")) {
                  kill(pid, SIGUSR1);
                  exit(EXIT_SUCCESS);
            }
            if(!strcmp(argv[1], "restart")) {
                  kill(pid, SIGHUP);
                  exit(EXIT_SUCCESS);
            }
            printf  ("використовується%s  [завершення|перезавантаження]\n",
argv[0]);
            exit (EXIT_FAILURE);
      }
```

```
            if((result            =            BecomeDaemonProcess(gLockFilePath,
"server_peredachi_informacii",
                  LOG_DEBUG, &gLockFileDesc, &daemonPID))<0) {
                        perror("Процес запуску демона зазнав невдачі");
                        exit(result);
                  }
            if((result = ConfigureSignalHandlers())<0) {
                  syslog(LOG_LOCAL0|LOG_INFO, "ConfigureSignalHandlers failed,
                        errno=%d", errno);
                  unlink(gLockFilePath);
                  exit(result);
            }
            if((result              =              BindPassiveSocket(INADDR_ANY,
gserver_peredachi_informaciiPort,
                        &gMasterSocket))<0) {
                  syslog(LOG_LOCAL0|LOG_INFO, "BindPassiveSocket failed,
                        errno=%d", errno);
                  unlink(gLockFilePath);
                  exit(result);
            }
            do {
                  if(AcceptConnections(gMasterSocket)<0) {
                        syslog(LOG_LOCAL0|LOG_INFO,"AcceptConnections не працює,
                              errno=%d",errno);
                        unlink(gLockFilePath);
                        exit(result);
                  }
                  if((gGracefulShutdown==1)&&(gCaughtHupSignal==0))
                        break;
                  gGracefulShutdown=gCaughtHupSignal=0;
            } while(1);
            TidyUp();
            return 0;
      }
```

Зараз ми пропустимо блок операторів if (argc > 1) {...} (ми повернемося до нього пізніше) і розглянемо основні етапи роботи демона. Функція BecomeDaemonProcess() перетворює звичайний консольний процес Linux у процес-демон. Функція ConfigureSignalHandlers() налаштовує оброблювачі сигналів процесу-демона, а функція BindPassiveSocket() відкриває визначенний порт TCP/IP для прослуховування вхідних запитів. Далі йде цикл, у якому сервер

обробляє запити. Багато мережних серверів, одержавши запит, створюють дочірній процес для його обробки. У такий спосіб досягається можливість паралельної обробки запитів. Деякі сервери використовують для паралельної обробки запитів потоки. Що стосується нашого сервера, то з міркувань простоти він обробляє запити в послідовному (блокуючому) режимі.

Нормальний вихід із циклу обробки запитів відбувається при одержанні процесом сигналу SIGUSER1. Після виходу із циклу процес викликає функцію TidyUp() і завершує роботу. Ми, безумовно, можемо завершити процес-демон, пославши йому сигнал SIGKILL (SIGTERM і деякі інші), але користувальницький сигнал SIGUSER1 гарантує ввічливе завершення нашого демона. «Ввічливе завершення» означає, що сервер відповість на поточний запит перед тим, як завершитися й видалить свій pid-файл.

Розглянемо тепер докладніше функцію BecomeDaemonProcess(), завдяки якій звичайний процес Linux стає демоном. Ми не будемо передруковувати тексти функцій цілком, так як інакше бакалаврська робота буде складатися виняткова з лістингів.

У рядку: chdir("/") ми робимо кореневу директорію поточною директорією процесу-демона. Будучи запущений, наш демон може працювати аж до перезавантаження системи, тому його поточна директорія повинна належати файловій системі, що не може бути розмонтована. Далі йде виклик:

```
lockFD = open(lockFileName, O_RDWR|O_CREAT|O_EXCL, 0644);
```

Кожний процес-демон створює так званий pid-файл (або файл блокування). Цей файл звичайно втримується в директорії /var/run і має ім'я daemon.pid, де “daemon” відповідає імені демона. Файл блокування містить значення PID процесу демона. Цей файл важливий по двох причинах. По-перше, його наявність дозволяє встановити, що в системі вже запущений один екземпляр демона. Більшість демонів, включаючи наш, повинні виконуватися не більш ніж в одному екземплярі (це логічно, якщо врахувати, що демони часто звертаються до неподілюваних ресурсів, таких, як мережні порти). Завершуючись, процес-демон видаляє pid-файл, указуючи тим самим, що можна запустити інший

екземпляр процесу. Однак, робота демона не завжди завершується нормально, і тоді на диску залишається pid-файл неіснуючого процесу. Це, здавалося б, може стати непереборною перешкодою для повторного запуску демона, але насправді, демони успішно справляються з такими ситуаціями. У процесі запуску демон перевіряє наявність на диску pid-файлу з відповідним ім'ям. Якщо такий файл існує, демон зчитує з нього значення PID і за допомогою функції kill(2) перевіряє, чи існує в системі процес із зазначеним PID. Якщо процес існує, виходить, користувач намагається запустити демон повторно. У цьому випадку програма виводить відповідне повідомлення й завершується. Якщо процесу із зазначеним PID у системі ні, значить pid-файл належав аварійно завершеному демонові. У цій ситуації програма звичайно радить користувачеві видалити pid-файл (відповідальність у таких справах завжди краще перекласти на користувача) і спробувати запустити її ще раз. Може, звичайно, трапитися й так, що після аварійного завершення демона на диску залишиться його pid-файл, а потім якийсь інший процес одержить той же самий PID, що був у демона. У цій ситуації для демона, що запускається знову, усе буде виглядати так, начебто його копія вже працює в системі, і запустити демон повторно ви не зможете. На щастя, описана ситуація вкрай малоймовірна.

Друга причина, по якій файл блокування вважається корисним, полягає в тому, що за допомогою цього файлу ми можемо швидко з'ясувати PID демона, не прибігаючи до команди ps.

Далі наш демон викликає функцію fork(3), що створює копію його процесу. Батьківський процес при цьому завершується:

```
curPID=fork();
switch(curPID) {
        case 0: /* ми в дочірньому процесі */
                break;
        case -1: /* помилка fork() – трапилося щось страшне */
                fprintf(stderr,"Помилка: Ініціалізуючий fork не працює:%s\n",
                        strerror(errno));
                return -1;
                break;
```

```
        default: /* ми в батьківському процесі, завершуємо його */
                exit(0);
                break;
}
```

Робиться це для того щоб процес-демон відключився від керуючого термінала. З кожним терміналом Linux зв'язаний набір груп процесів, іменований сесією. У кожний момент часу тільки одна із груп процесів, що входять у сесію, має доступ до термінала (тобто, може виконувати введення/вивід за допомогою термінала). Ця група йменується foreground (пріоритетної). У кожній сесії є процес-родоначальник, що називається лідером сесії. Якщо процес-демон запущений з консолі, він, природно, стає частиною пріоритетної групи процесів, що входять у сесію відповідного термінала.

Для того щоб відключитися від термінала, демон повинен почати нову сесію, не пов'язану з яким-небудь терміналом. Для того щоб демон міг почати нову сесію, він сам не повинен бути лідером якої-небудь іншої сесії. Виклик fork() створює дочірній процес, що свідомо не є лідером сесії. Далі дочірній процес, отриманий за допомогою fork(), починає нову сесію за допомогою виклику функції setsid(2). При цьому процес стає лідером (і єдиним учасником) нової сесії.

```
if(setsid()<0)
        return -1;
```

Отже, на даному етапі ми маємо процес, не пов'язаний з яким-небудь терміналом. Далі в деяких керівництвах рекомендується викликати fork() ще раз, щоб новий процес перестав бути лідером нової сесії (в System V лідер сесії може автоматично одержати керуючий термінал при деяких умовах). В Linux у повторному виклику fork() немає необхідності й ми його робити не будемо.

Варто відзначити, що тепер наш демон одержав новий PID, що ми знову повинні записати в pid-файл демона. Ми записуємо значення PID у файл у строковому виді (а не як змінну типу pid_t). Робиться це для зручності користувача, щоб значення PID з pid-файлу можна було прочитати за допомогою cat.

Наприклад:

```
kill 'cat /var/run/server_peredachi_informacii.pid '
```

Нашому демонові вдалося розірвати зв'язок з терміналом, з якого він був запущений, але він усе ще може бути пов'язаний з іншими процесами й файловими системами через файлові дескриптори, успадковані від батьківських процесів. Для того щоб розірвати й цей зв'язок, ми закриваємо всі файлові дескриптори, відкриті в нашім процесі:

```
numFiles = sysconf(_SC_OPEN_MAX);
for(i = numFiles-1; i >= 0; ---ii) {
      if(i != lockFD)
      close(i);
}
```

Функція sysconf() з параметром _SC_OPEN_MAX повертає максимально можливу кількість дескрипторів, які може відкрити наша програма. Ми викликаємо функцію close() для кожного дескриптора (незалежно від того, відкритий він чи ні), за винятком дескриптора pid– файлу, що повинен залишатися відкритим.

Під час роботи демона дескриптори стандартних потоків уведення, виводу й помилок також повинні бути відкриті, оскільки вони необхідні багатьом функціям стандартної бібліотеки. У той же час, ці дескпритори не повинні вказувати на які-небудь реальні потоки введення/виводу. Для того, щоб вирішити це завдання, ми закриваємо перші три дескриптори, а потім знову відкриваємо їх, указуючи як ім'я файлу /dev/null:

```
stdioFD = open("/dev/null", O_RDWR);
dup(stdioFD);
dup(stdioFD);
```

Тепер ми можемо бути впевнені, що демон не одержить доступу до якого-небудь терміналу. Проте, у демона повинна бути можливість виводити кудись повідомлення про свою роботу. Традиційно для цього використовуються файли журналів (log-файли). Якщо в роботі демона відбувся якийсь збій, користувач може проаналізувати файл журналу й установити причину збою. Ніщо не заважає нашому демонові відкрити свій власний файл журналу, але це не дуже зручно.

Більшість демонів користуються послугами утиліти syslog, що веде журнали безлічі системних подій. Ми відкриваємо доступ до журналу syslog за допомогою функції openlog(3):

```
openlog(logPrefix, LOG_PID|LOG_CONS|LOG_NDELAY|LOG_NOWAIT, LOG_LOCAL0);
(void) setlogmask(LOG_UPTO(logLevel));
```

Перший параметр функції openlog() – префікс, що буде додаватися до кожного запису в системному журналі. Далі випливають різні опції syslog. Функція setlogmask(3) дозволяє встановити рівень пріоритету повідомлень, які записуються в журнал подій. При виклику функції BecomeDaemonProcess() ми передаємо в параметрі logLevel значення LOG_DEBUG. У сполученні з макросом LOG_UPTO це означає, що в журнал будуть записуватися всі повідомлення із пріоритетом, починаючи з найвищого й закінчуючи LOG_DEBUG.

Останнє, що нам потрібно зробити для «демонизації» процесу – викликати функцію setpgrp().

Цей виклик створює нову групу процесів, ідентифікатором якої є ідентифікатор поточного процесу. На цьому робота функції BecomeDaemonProcess() завершується, так як тепер наш процес став справжнім демоном.

Функція ConfigureSignalHandlers() налаштовує оброблювачі сигналів. Сигнали, які одержить наш демон, можна розділити на три групи: ігноровані, «фатальні» і оброблювані. Викликаючи функцію signal(SIGUSR2, SIG_IGN), ми вказуємо, що наш демон повинен ігнорувати сигнал SIGUSR2. Аналогічно ми поступаємо із сигналами SIGPIPE, SIGALRM, SIGTSTP SIGPROF, SIGCHLD. Сигнали SIGQUIT, SIGILL, SIGTRAP, SIGABRT, SIGIOT, SIGBUS, SIGFPE, SIGSEGV, SIGSTKFLT, SIGCONT, SIGPWR і SIGSYS відносяться до категорії «фатальних». Ми не можемо їх ігнорувати, але й продовжувати виконання процесу-демона після одержання одного із цих сигналів небажано. Ми призначаємо всім цим сигналам оброблювач FatalSigHandler, наприклад:

```
signal(SIGQUIT, FatalSigHandler);
```

Функція-оброблювач FatalSigHandler() записує в журнал подій інформацію про отриманий сигнал, і потім завершує процес, викликавши перед цим функції closelog() і TidyUp(), які вивільняють всі зайняті процесом ресурси:

```
void FatalSigHandler(int sig) {
#ifdef _GNU_SOURCE
      syslog(LOG_LOCAL0|LOG_INFO,"отриманий сигнал:%s – вихід",strsignal(sig));
#else
      syslog(LOG_LOCAL0|LOG_INFO," отриманий сигнал:%d – вихід",sig);
#endif
      closelog();
      TidyUp();
      _exit(0);
}
```

На ті три сигнали, які відносяться до категорії оброблюваних, – SIGTERM, SIGUSR1 і SIGHUP, демон реагує по-різному:

```
sigtermSA.sa_handler = TermHandler;
sigemptyset(&sigtermSA.sa_mask);
sigtermSA.sa_flags = 0;
sigaction(SIGTERM, &sigtermSA, NULL);
sigusr1SA.sa_handler = Usr1Handler;
sigemptyset(&sigusr1SA.sa_mask);
sigusr1SA.sa_flags = 0;
sigaction(SIGUSR1, &sigusr1SA, NULL);
sighupSA.sa_handler = HupHandler;
sigemptyset(&sighupSA.sa_mask);
sighupSA.sa_flags = 0;
sigaction(SIGHUP, &sighupSA, NULL);
```

Оброблювач TermHandler() викликає функцію TidyUp() і завершує процес. Оброблювач Usr1Handler() робить у системному журналі запис про ввічливе завершення процесу й привласнює змінній gGracefulShutdown значення 1 (що, як ви пам'ятаєте, приводить до виходу із циклу обробки запитів, коли цикл буде готів до цього). Оброблювач сигналу HupHandler() також робить запис у системному журналі, після чого привласнює значення 1 змінним gGracefulShutdown і gCaughtHupSignal. У реальному житті одержання сигналу SIGHUP приводить до перезапуску демона, що супроводжується повторним

прочитанням файлу конфігурації (який звичайно читається демоном саме під час запуску) і переустановкою значень записаних у ньому параметрів. Саме необхідність прочитати повторно файл конфігурації є найбільш частою причиною перезапуску демонів. У нашого демона файлу конфігурації ні, так що в процесі перезапуску робити йому особливо нема чого.

Зверніть увагу на тип змінних gGracefulShutdown і gCaughtHupSignal. З типом sig_atomic_t ми раніше не зустрічалися. Застосування цього типу гарантує, що читання й запис даних у змінні gGracefulShutdown і gCaughtHupSignal буде виконуватися атомарно, однією інструкцією процесора, що не може бути перервана. Атомарність при роботі зі змінними gGracefulShutdown і gCaughtHupSignal важлива тому, що до них можуть одночасно одержати доступ і оброблювачі сигналів, і головна функція програми. По цій же причині ми позначаємо зазначені змінні ключовим словом volatile.

Функція BindPassiveSocket() відкриває для прослуховування порт сервера (у нашім випадку це порт 30333) на всіх доступних мережних інтерфейсах і повертає відповідний сокет:

```
int BindPassiveSocket(const int portNum,
        int *const boundSocket)
{
      struct sockaddr_in sin;
      int   newsock, optval;
      size_t optlen;
      memset(&sin.sin_zero, 0, 8);
      sin.sin_port = htons(portNum);
      sin.sin_family = AF_INET;
      sin.sin_addr.s_addr = htonl(INADDR_ANY);
      if((newsock= socket(PF_INET, SOCK_STREAM, 0))<0)
      return -1;
      optval = 1;
      optlen = sizeof(int);
      setsockopt(newsock, SOL_SOCKET, SO_REUSEADDR, &optval, optlen);
      if(bind(newsock,     (struct     sockaddr*)     &sin,     sizeof(struct
sockaddr_in))<0)
      return -1;
      if(listen(newsock,SOMAXCONN)<0)
```

```
        return -1;
        *boundSocket = newsock;
        return 0;
}
```

Відзначимо одну цікаву деталь. Якщо попередній, уже закритий, сокет, пов'язаний з даним портом, перебуває в стані TIME_WAIT, між закриттям старого й відкриттям нового сокету може відбутися затримка, рівним двом періодам життя сегмента (затримка може становити до двох мінут). Для того, щоб при повторному запуску демона нам не довелося чекати, ми використовуємо функцію setsockopt() з параметром SO_REUSEADDR.

Функція AcceptConnections() обробляє запити послідовно, використовуючи виклик, що блокує, accept():

```
int AcceptConnections(const int master)
{
        int   proceed = 1, slave, retval = 0;
        struct sockaddr_in      client;
        socklen_t   clilen;
        while((proceed==1)&&(gGracefulShutdown==0)) {
                clilen = sizeof(client);
                slave = accept(master,(struct sockaddr *)&client,&clilen);
                if(slave<0) {       /* помилка accept() */
                        if(errno == EINTR)
                                continue;
                        syslog(LOG_LOCAL0|LOG_INFO,"accept() failed:%m\n");
                        proceed = 0;
                        retval = -1;
                }
                else
                {
                        retval = HandleConnection(slave);
                        if(retval)
                                proceed = 0;
                }
                close(slave);
        }
        return retval;
}
```

Змінна proceed, разом зі змінною gGracefulShutdown, указує, чи належна програма продовжувати обробляти запити. Якщо черговий виклик connect() або HandleConnection() повернув повідомлення про помилку, цієї змінної привласнюється 0 і обробка запитів припиняється. Новий сокет, отриманий у результаті виклики accept(), передається функції HandleConnection().

```
int HandleConnection(const int slave)
{
	char readbuf[1025];
	size_t	bytesRead;
	const size_t	buflen=1024;
	int	retval;
	retval = ReadLine(slave, readbuf, buflen, &bytesRead);
	if(retval==0)
		WriteToSocket(slave, readbuf, bytesRead);
	return retval;
}
```

Функція HandleConnection() зчитує переданий клієнтом рядок і відразу повертає його клієнтові. Потім функція AcceptConnections() закриває з'єднання, відкрите в результаті виклику accept(). Функції ReadLine() і WriteToSocket() тривіальні, і розглядати їх ми не будемо. Якщо десь у ланцюжку викликів AcceptConnections(), HandleConnection(), ReadLine() і WriteToSocket() виникла помилка, інформація про помилку буде передаватися нагору по ланцюжку доти, поки не досягне функції main(). У функції main() ця інформація приведе до негайного завершення роботи демона з відповідним записом у журнал системних повідомлень.

Розглянемо, нарешті, функцію TidyUp(), до якої звертаються багато функцій сервера перед тим, як завершити його роботу.

```
void TidyUp(void)
{
	if(gLockFileDesc!=-1) {
		close(gLockFileDesc);
		unlink(gLockFilePath);
		gLockFileDesc=-1;
	}
	if(gMasterSocket!=-1) {
```

```
            close(gMasterSocket);
            gMasterSocket=-1;
        }
    }
```

Завдання функції TidyUp() – «прибрати сміття» за процесом-демоном. У принципі, без цієї функції можна обійтися, так як після завершення процесу система сама закриє всі його дескриптори, але правила гарного тону вимагають явного вивільнення всіх ресурсів, виділених явно.

Якщо ви скомпілюєте програму-демон за допомогою команди:

gcc server_peredachi_informacii.c -o server_peredachi_informacii

То зможете запустити демон командою:

server_peredachi_informacii

Оскільки демон має потребу в доступі до директорії /var/run, запускати його потрібно в режимі root. Відразу після запуску програми ви знову побачите запрошення командного рядка, що для демонів зовсім нормально. Якби сервер server_peredachi_informacii виконував що-небудь корисне, команду його запуску можна було б прописати в одному зі сценаріїв запуску системи в директорії /etc/init.d, але ми цього робити не будемо. Після того як сервер запущений, ви можете дати команду:

telnet 127.0.0.1 30333

У результаті буде встановлене з'єднання із сервером. Надрукуйте яку-небудь послідовність символів у консолі telnet і натисніть «Ввод». Як відповідь сервер поверне надрукований рядок і закриє з'єднання.

Повернемося тепер до початкових рядків функції main(). Хоча ми можемо одержати PID демона з його pid-файлу й управляти демоном за допомогою команди kill, такий варіант не можна назвати дуже зручним. Часто для керування демоном використовується сам здійсненний файл демона, що запускається зі спеціальними аргументами командного рядка. Наш демон розуміє дві команди: stop (завершення роботи демона) і restart (перезапуск). Подивимося, як поведеться демон, запущений з аргументами командного рядка. У цьому випадку на початку програми демон намагається вважати значення PID зі свого pid-файлу.

Якщо відкрити pid-файл не вдається, виходить, швидше за все, демон не запущений, і керуючому режиму просто нема чого робити. Якщо значення PID отримане, процес, керуючий демоном, посилає демонові відповідний сигнал за допомогою функції kill().

Демони не розраховані на те, щоб отримувати яку-небудь інформацію від користувача. Власну інформацію вони передають іншим програмам або записують у журнали системних подій.

4.2 Захист розробленого програмного забезпечення

Дані в програмі захищаються за допомогою використання алгоритму Md5. Він отримує на вході повідомлення довільної довжини і створює на виході дайджест повідомлення довжиною 128 біт. Алгоритм складається з наступних кроків:

1. Додавання недостаючих біт. Повідомлення доповнюється так, щоб його довжина стала рівна 448 по модулю 512 (довжина 448 mod 512). Це означає, що довжина доданого повідомлення на 64 біта менше, ніж число, кратне 512. Додавання проводиться завжди, навіть якщо повідомлення має потрібну довжину. Наприклад, якщо довжина повідомлення 448 біт, воно доповнюється 512 бітами до 960 біт. Таким чином, число біт, що додаються, знаходиться в діапазоні від 1 до 512.

Додавання складається з одиниці, за якою слідує необхідна кількість нулів.

2. Додавання довжини. 64-бітове представлення довжини початкового (до додавання) повідомлення в бітах приєднується до результату першого кроку. Якщо первинна довжина більша, ніж 264, то використовуються тільки останні 64 біта. Таким чином, поле містить довжину початкового повідомлення по модулю 264.

В результаті перших двох кроків створюється повідомлення, довжина якого кратна 512 бітам. Це розширене повідомлення представляється як послідовність 512-бітових блоків Y_0, Y_1,..., Y_{l-1}, при цьому загальна довжина

розширеного повідомлення рівна *L**512 бітам. Таким чином, довжина отриманого розширеного повідомлення кратна шістнадцяти 32-бітовим словам.

3. Ініціалізація MD-буфера. У алгоритмі Md5 використовується 128-бітовий буфер для зберігання проміжних і остаточних результатів хеш-функції. Буфер може бути представлений як чотири 32-бітові регістри (*A*, *B*, *C*, *D*). Ці регістри ініціалізувалися наступними шістнадцятковими числами:

$$A = 01234567$$

$$B = 89abcdef$$

$$C = Fedcba98$$

$$D = 76543210$$

4. Обробка послідовності 512-бітових (16-словних) блоків. Основою алгоритму Md5 є модуль, що складається з чотирьох циклічних обробок, позначений як Hmd5. Чотири цикли мають схожу структуру, але кожен цикл використовує свою елементарну логічну функцію, *ff*, що позначається, *fg*, *fh* і *fi* відповідно.

Кожен цикл приймає на вхід поточний 512-бітовий блок Y_q, що обробляється в даний момент, і 128-бітове значення буфера ABCD, яке є проміжним значенням дайджесту, і змінює вміст цього буфера. Кожен цикл також використовує четвертую частина 64-елементної таблиці *T*[1 . 64], побудованої на основі функції sin. *i*-ий елемент *T*, *T*[i], що позначається, має значення, рівне цілій частині від 232 * abs (sin (*i*)), і задане в радіанах. Оскільки abs (sin (*i*)) є числом між 0 і 1, кожен елемент *T* є цілим, яке може бути представлене 32 бітами. Таблиця забезпечує “випадковий” набір 32-бітових значень, які повинні ліквідовувати будь-яку регулярність у вхідних даних.

Для отримання *Mdq*+1 вихід чотирьох циклів складається по модулю 232 з *Mdq*. Складання виконується незалежно для кожного з чотирьох слів в буфері.

5) Вихід Md5. Після обробки всіх *L* 512-бітових блоків виходом *L*-ої стадії є 128-бітовий дайджест повідомлення.

Детальніше логіку кожного з чотирьох циклів виконання одного 512-бітового блоку розглянуто нижче. Кожен цикл складається з 16 кроків, що оперують з буфером *ABCD*.

$$A \quad B + \text{Clss}\,(A + f\,(B, C, D) + X\,[k] + T\,[i]),$$

A, *B*, *C*, *D* – чотири слова буфера; після виконання кожного окремого кроку відбувається циклічне зрушення вліво на одне слово.

f – одна з елементарних функцій *ff*, *fg*, *fh*, *fi*.

CLSs – циклічне зрушення вліво на s біт 32-бітового аргументу.

X [*k*] – *M* [*q* * 16 + *k*] – кодує 32-бітове слово в *q*-ому 512 блоці повідомлення.

T [*i*] – *i*-е 32-бітове слово в матриці T.

+ – складання по модулю 232.

На кожному з чотирьох циклів алгоритму використовується одна з чотирьох елементарних логічних функцій. Кожна елементарна функція отримує три 32-бітові слова на вході і на виході створює одне 32-бітове слово. Кожна функція є безліччю побітових логічних операцій, тобто *n*-ий біт виходу є функцією від n-ого біта трьох входів. Елементарні функції наступні:

$$ff = (B \,\&\, C) \quad (\text{not } B \,\&\, D)$$

$$fg = (B \,\&\, D) \text{ V } (C \,\&\, \text{not } D)$$

$$fh = B \quad C \quad D$$

$$fi = C \quad (B \,\&\, \text{not } D)$$

Масив з 32-бітових слів *X* [0..15] містить значення поточного 512-бітового вхідного блоку, який обробляється зараз. Кожен цикл виконується 16 разів, а оскільки кожен блок вхідного повідомлення обробляється в чотирьох циклах, то кожен блок вхідного повідомлення обробляється по схемі 64 рази. Якщо представити вхідний 512-бітовий блок у вигляді шістнадцяти 32-бітових слів, то кожне вхідне 32-бітове слово використовується чотири рази, по одному разу в кожному циклі, і кожен елемент таблиці T, що складається з 64 32-бітових слів, використовується тільки один раз. Після кожного кроку циклу відбувається циклічне зрушення вліво чотирьох слів *A*, *B*, *C* і *D*. На кожному кроці змінюється тільки одне з чотирьох слів буфера *ABCD*. Отже, кожне слово буфера змінюється 16 разів, і потім 17-й раз в кінці для отримання остаточного виходу даного блоку.

5 МЕТОДИКА ВПРОВАДЖЕННЯ СИСТЕМИ КІБЕРБЕЗПЕКИ В ПРОМИСЛОВУ ЕКСПЛУАТАЦІЮ

Методика впровадження системи в промислову експлуатацію реалізується наступним чином.

Установка розробленого програмного забезпечення, складається з виконання наступних кроків:

– Розпакуйте архів в окремий каталог.

– У терміналі переходимо в каталог з розпакованої архівом (тарболом): cd /шлях_до_папки. Для зручності користуйтеся кнопкою Tab, що автоматично вставляє ім'я каталогу при його наборі або дає список всіх каталогів.

– Конфігуруємо пакет: набираємо: ./configure.

– Створюємо пакет: make.

– Установлюємо пакет: make install.

Спеціальні вимоги розробленого програмного забезпечення:

– Підтримка останнього випуску POSIX thread (pthreads). Більшість систем підтримують pthreads за замовчуванням.

– Thread-safe X11 libraries. Знову ж більшість систем підтримують за замовчуванням.

– Компілятор C++, libstdc++ і c++ headers.

– GNU Make. Входить у комплект BSD і Linux.

– Boost C++ Libraries не нижче версії 1.31.1.

– Qt3 не нижче версії 3.0.0. Необхідний для Qt плагінів.

– Qt4 не нижче версії 4.3.2. Необхідний для Qt4 плагінів.

– CMake 2.4.2 або вище. Необхідний для Qt4 плагінів.

– Вибірково:

– OpenSSL для підтримки безпечного з'єднання.

– GpgMe для підтримки GPG.

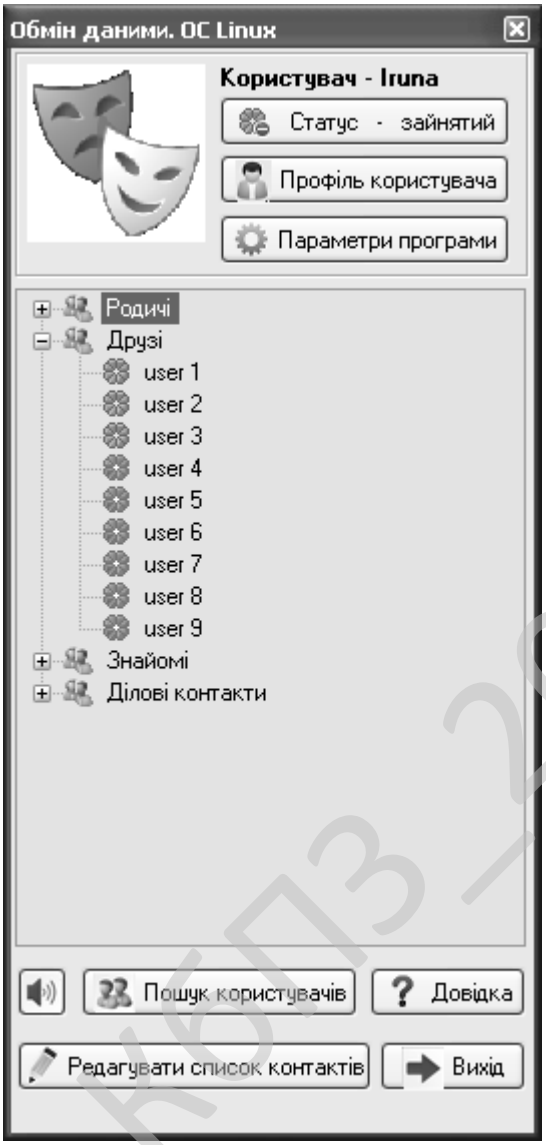

Рисунок 5.1 – Головне вікно

На рисунку 5.1 – 5.3 зображені скрін-шоти роботи програми, які у повному обсязі показують усі функції розробленого програмного забезпечення.

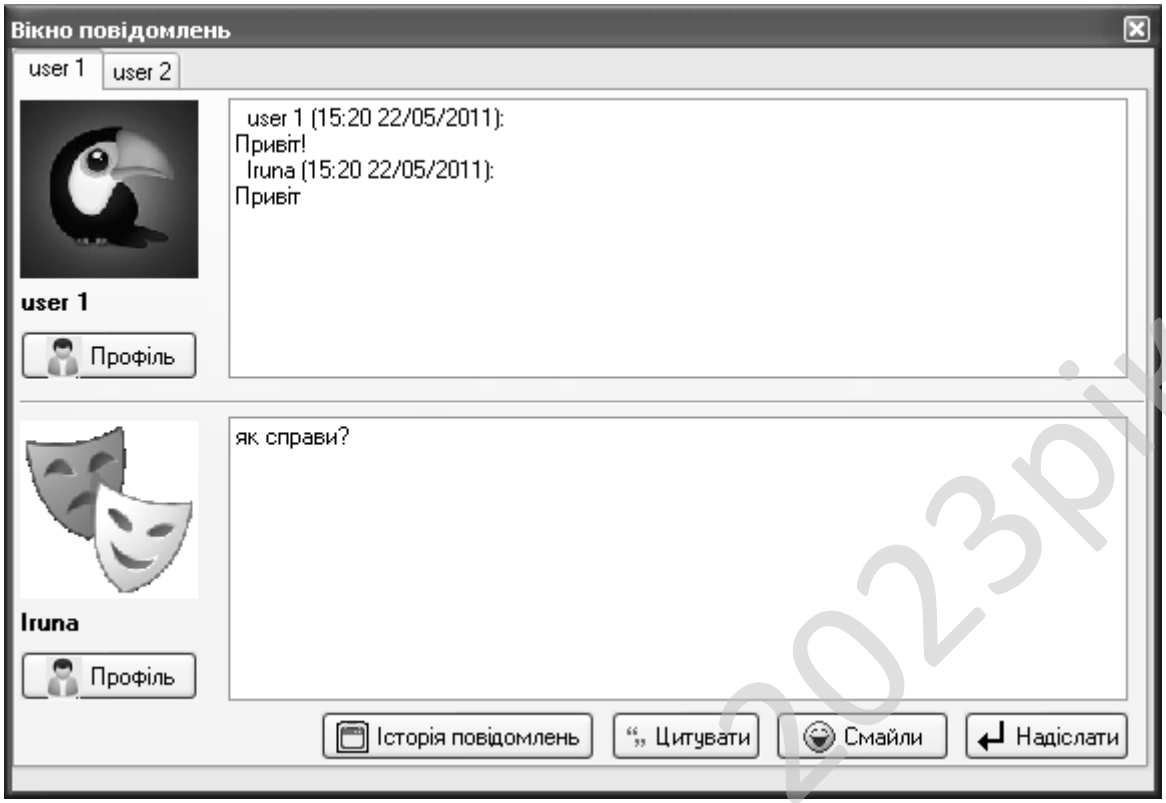

Рисунок 5.2 – Вікно повідомлень

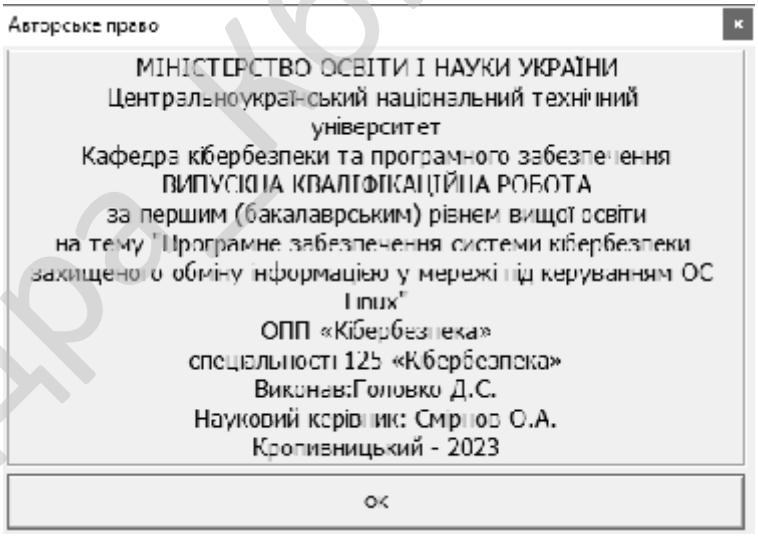

Рисунок 5.3 – Довідка

6 ОСНОВНІ ВИСНОВКИ

Програмне забезпечення, створене в результаті виконання випускної кваліфікаційної роботи за першим (бакалаврським) рівнем вищої освіти, призначено для системи кібербезпеки захищеного обміну інформацією у мережі під керуванням ОС Linux.

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

Рішення завдання полягало у вирішенні наступних задач:

– Був проведений огляд існуючих систем захищеного обміну інформацією у мережі під керуванням ОС Linux.

– Досліджена система захищеного обміну інформацією у мережі під керуванням ОС Linux.

– На основі отриманих результатів досліджень створена програмна реалізація системи кібербезпеки захищеного обміну інформацією у мережі під керуванням ОС Linux.

Розроблені під час виконання випускної кваліфікаційної роботи за першим (бакалаврським) рівнем вищої освіти алгоритми дозволяють успішно вирішувати завдання захищеного обміну інформацією у мережі під керуванням ОС Linux.

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

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

Програма реалізована на мові високого рівня Visual C++. Дана мова програмування дозволяє найбільш ефективно обробляти дані призначені для системи кібербезпеки захищеного обміну інформацією у мережі під керуванням

ОС Linux. Це дозволило мінімізувати строк розробки програмного забезпечення, і, як слід, зменшити витрати на його розробку. Запропоноване програмне забезпечення ділиться на загальне програмне забезпечення, що поставляється із засобами обчислювальної техніки й спеціальне програмне забезпечення, що спеціально розроблене для даної конкретної системи кібербезпеки й включає програми, що реалізують її функції.

Програма призначена для виконання під управлінням багатозадачної операційної системи Linux.

Даються необхідні рекомендації з установки розробленого програмного забезпечення.

Для підвищення рівня безпеки запропоновано застосовувати алгоритм Md5.

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

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Бертсекас Д. Сети передачи данных: пер. с англ. / Д. Бертсекас, Р. Галлагер; под ред. Б.С. Цыбакова. – М.: Мир, 1989. – 544 с.

2. Вегешна Ш. Качество обслуживания в сетях IP / Ш. Вегешна. – М.: Вильямс, 2003. – 368 с.

3. Вишневский В.М. Теоретические основы проектирования компьютерных сетей / В.М. Вишневский. – М.: Техносфера, 2003. – 512 с.

4. Галкин В.А. Телекоммуникации и сети / В.А. Галкин, Ю.А.Григорьев. – М.: МГТУ имени Н.Э. Баумана, 2003. – 608 с.

5. Гмурман В.Е. Теория вероятностей и математическая статистика / В. Е. Гмурман. – М.: Высшая школа, 2003. – 479 с.

6. ГОСТ Р ИСО/МЭК 13335-1-2006. Информационная технология. Методы и средства обеспечения безопасности. Часть 1. Концепция и модели менеджмента безопасности информационных и телекоммуникационных технологий [Електронний ресурс]. – Режим доступу до ресурсу: http://www.rfcmd.ru/sphider/docs/InfoSec/GOST-R_ISO_IEC_13335-1- 2006.htm

7. ГОСТ Р ИСО/МЭК 27033-1-2011 Информационная технология. Методы и средства обеспечения безопасности. Безопасность сетей. Часть 1. Обзор и концепции [Електронний ресурс]. – Режим доступу до ресурсу: http://protect.gost.ru/document.aspx?control=7&id=179072

8. ДСТУ В 3265 – 95. Зв'язок військовий. Терміни та визначення. – К.: УкрНДІССІ, 1995. – 23 с.

9. ДСТУ ISO 9000:2007 Системи управління якістю. Основні положення та словник термінів [Електронний ресурс]. – Режим доступу до ресурсу: http://document.ua/docs/tdoc14237.php

10. Дымарский Я.С. Управление сетями связи: Принципы, протоколы, прикладные задачи / Я.С. Дымарский, Н.П. Крутякова, Г.Г. Яновский. –М.: ИТЦ «Мобильные коммуникации», 2003. – 384 с.

11. Ершов В.А. Мультисервисные телекоммуникационные сети / В.А. Ершов, Н.А. Кузнецов – М.: Изд. МГТУ им. Н.Э. Баумана, 2003. – 432 с.

12. Защита информации в автоматизированных системах обработки информации [Електронний ресурс]. – Режим доступу до ресурсу: http://www.xserver.ru/computer/raznoe/bezopasn/2/

13. Информационное сообщение об утверждении требований к средствам антивирусной защиты от 30 июля 2012 г. № 240/24/3095 [Електронний ресурс]. – Режим доступу до ресурсу: http://fstec.ru/component/attachments/download/402

14. Касперский Е. Компьютерное зловредство / Е. Касперский. – СПб.: Питер, 2007. – 208 с.

15. Касперски К. Техника сетевых атак. [Електронний ресурс]. – Режим доступу до ресурсу: http://rghost.ru/download/43730077/8e48b6263ce45c7dc2a65a7453383dc33b22486d/Крис%20Касперски%20-%20Техника%20сетевых%20атак.pdf

16. Касперски К. Техника и философия хакерских атак / К. Касперски. – М.: Солон-Пресс, 2004 – 272 с.

17. Касперски К. Записки исследователя компьютерных вирусов / К. Касперски. – СПб.: Питер, 2006. – 316 с.

18. Касперски К. Компьютерные вирусы изнутри и снаружи. / К. Касперски. – СПб.: Питер, 2006. – 526 с.

19. Кингман Дж. Пуассоновские процессы / Дж. Кингман. – М.:МЦНМО, 2007. – 136 с.

20. Корченко А.Г. Построение систем защиты информации на нечетких множествах. Теория и практические решения / А.Г. Корченко. – К.: «МК-Пресс», 2006. – С. 207-214.

21. Конахович Г.Ф. Сети передачи пакетных данных / Г.Ф. Конахович, В.М.Чуприн. – К.: МК-Пресс, 2006. – 272 с.

22. Кучук Г.А. Управление ресурсами инфотелекоммуникаций / Г.А. Кучук, Р.П. Гахов, А.А. Пашнев. – М.: Физматлит, 2006. – 220 с.

23. Лавренков Ю.Н. Исследование и разработка комбинированных нейросетевых технологий для повышения эффективности безопасной маршрутизации информации в сетях связи: диссертация ... кандидата технических наук: 05.13.17 / Лавренков Юрий Николаевич, 2014.– 208 с.

24. Лазарев И. А. Композиционное проектирование сложных агрегативных систем / И.А. Лазарев. – М.: Радио и связь, 1986. – 312 с.

25. Лукацкий А.В. Обнаружение атак / А.В. Лукацкий. – С.Пб.: BHV, 2001. – 624 с.

26. Майника Э. Алгоритмы оптимизации на сетях и графах: пер. с англ. / Э. Майника; под ред. Е.К. Масловского. – М.: Мир, 1981. – 321 с.

27. МСЭ-Т Рекомендация G.101. Международные телефонные соединения и цепи – Общие определения //11/2003. [Електронний ресурс]. – Режим доступу до ресурсу: http://www.telecom61.ru/SharedFiles/ Download.aspx? …pageid=106

28. Назаров А.Н. Модели и методы расчета структурно-сетевых параметров АТМ сетей / Алексей Николаевич Назаров. – М.: Горячая линия - Телеком, 2002. – 256 с.

29. Одом Ш. Коммутаторы CISCO / Ш. Одом, Х. Ноттингем – М.: "Кудиц-Образ", 2003. – 528 с.

30. Олифер В.Г. Компьютерные сети. Принципы, технологии, протоколы: учебник для вузов / В.Г. Олифер, Н.А. Олифер. – 2-е изд. – СПб.: Питер, 2007. – 958 с.

31. Партыка С.А. Метод ускоренной коррекции SPT с использованием динамических алгоритмов [Електронний ресурс]. – Режим доступу до ресурсу: http://openarchive.nure.ua/bitstream/123456789/936/1/ ASU_158_2012%20%2842-47%29.pdf

32. Руководство по технологиям объединенных сетей. 4-е изд. / пер.с англ. и ред. А.Н. Крикуна – М.: Изд. дом «Вильямс», 2005. – 1040 с.

33. Семенов С.Г. Математическая модель мультисервисного канала связи на основе экспоненциальной GERT-сети / С.Г. Семенов, Є.В. Мелешко, Я.В. Ілюшко // Системи озброєння і військова техніка. – Х.:ХУ ПС. – 2011. –Вип. 3(27). – С. 64-67.

34. Семенов С.Г. Математична модель системи криптографічного захисту електронних повідомлень на основі GERT-мережі / С.Г. Семенов, О.О. Сур // Системи управління, навігації та зв'язку. – К.: ЦНДІ навігації і управління. – 2012. – Том 1. Вип. 1(21). – С. 131-137

35. Семенов С.Г. Исследования вероятностно-временных характеристик мультисервисного канала связи с использованием математического аппарата GERT-сети / С.Г. Семенов, В.В. Босько, І.А. Березюк // Системи обробки інформації. – Х.: ХУ ПС, 2012. – Т. 1., Вип. 3(101). – С. 139-142.

36. Семенов С.Г. Моделирование защищенного канала связи с использованием экспоненциальной GERT-сети / С.Г. Семенов, А.А. Можаев // Информатика, математическое моделирование, экономика. – Смоленськ.: Смоленский филиал АНО ВПО ЦС РФ "Российский университет кооперации". – 2012. – Том.1. – С. 152-160.

37. Семенов С.Г. Методика математического моделирования защищенной ИТС на основе многослойной GERT-сети / С.Г. Семенов // Вісник Національного технічного університету «Харківський політехнічний інститут». – Х.:НТУ «ХПІ». – 2012. –№62 (968). – С 173-181.

38. Семенов С.Г. Защита данных в компьютеризированных управляющих системах / С.Г. Семенов, В.В. Давыдов, С.Ю. Гавриленко. – LAP Lambert Academic Publishing GmbH & Co. KG (Саарбрюккен, Германия), 2014. – 236 с.

39. Смирнов А.А. Анализ и сравнительное исследование перспективных направлений развития цифровых телекоммуникационных систем и сетей / А.А.Смирнов, В.В.Босько, Е.В.Мелешко // Системи обробки інформації. – Х.: ХУ ПС, 2008. – Вип.7(74). – С.120-123.

40. Смирнов А.А. Усовершенствование метода управления очередями в многопротокольных узлах телекоммуникационной сети / А.А.Смирнов, Е.В.Мелешко // Збірник тез та доповідей другої всеукраїнської науково-практичної конференції «Системний аналіз. Інформатика. Управління». Запоріжжя. Тези доповідей. Запоріжжя: КПУ, 2011.

41. Смирнов С.А. Метод безопасной маршрутизации метаданных в облачные антивирусные системы / А.К. Дидык, С.А. Смирнов // Информационные технологии в управлении, образовании, науке и промышленности: монография / Под редакцией профессора В.С. Пономаренко. – Х.: Видавець Рожко С.Г., 2016. – 566 с.

42. Смирнов С. А. Сравнительные исследования математических моделей технологии распространения компьютерных вирусов в информационно-телекоммуникационных сетях / Мохамад Абу Таам Гани, А. А. Смирнов, А. В. Коваленко, С. А. Смирнов // Системи обробки інформації: зб. наук. праць. – Х.: ХУПС, 2014. – Вип. 9(125). – 105-110.

43. Смирнов С. А. Математическая модель интеллектуального узла коммутации с обслуживанием информационных пакетов различного приоритета / Мохамад Абу Таам Гани, А. А. Смирнов, Н. С. Якименко, С. А. Смирнов // Збірник наукових праць Харківського університету Повітряних Сил. – Харків: ХУПС, 2014. – Вип. 4 (41). – С. 48-52.

44. Смирнов С. А. Исследование показателей качества функционирования интеллектуальных узлов коммутации в телекоммуникационных системах и сетях / Мохамад Абу Таам Гани, А. А. Смирнов, Н. С. Якименко, С. А. Смирнов // Наука і техніка Повітряних Сил Збройних Сил України: наук. журн. –Х.: ХУПС, 2014. – № 4(17). – С. 90-95.

45. Смирнов С. А. Усовершенствованный алгоритм управления доступом к «облачным» телекоммуникационным ресурсам / Мохамад Абу Таам Гани, А. А. Смирнов, Н. С. Якименко, С. А. Смирнов // Системи обробки інформації: зб. наук. праць. – Х.: ХУПС, 2015. –Вип. 1(126). – С. 150-153.

46. Smirnov S.A. Method of controlling access to intellectual switching nodes of telecommunication networks and systems / A.A. Smirnov, Mohamad Abou Taam, S.A. Smirnov // International Journal of Computational Engineering Research (IJCER). – Volume 5, Issue 5. – India. Delhi. – 2015. – P. 1-7.

47. Смирнов С. А. Анализ и исследование методов управления сетевыми ресурсами для обеспечения антивирусной защиты даннях / Мохамад Абу Таам Гани, А. А. Смирнов, С. А. Смирнов // Системи озброєння і військова техніка: наук. журн. – Х.: ХУПС, 2015. – № 3(43). – С. 100-107.

48. Смирнов С. А. Исследование эффективности метода управления доступом к облачным антивирусным телекоммуникационным ресурсам / Мохамад Абу Таам Гани, А. А. Смирнов, С. А. Смирнов // Наука і техніка Повітряних Сил Збройних Сил України: наук. журн. –Х.: ХУПС, 2015. – № 3(20). – С. 134-141.

49. Смирнов С. А. Комплекс gert-моделей технологии облачной антивирусной защиты телекоммуникационной системы / А. А. Смирнов, А. К. Дидык, А. Н. Дреев, С. А. Смирнов // Безпека інформації: наук. - практ. журн. – К.: НАУ, 2015. – Т. 21, № 3. – С. 251-262.

50. Смирнов С. А. Метод безопасной маршрутизации метаданных в облачные антивирусные системы / А. А. Смирнов, А. К. Дидык, С. А. Смирнов //

Системи озброєння і військова техніка: наук. журн. – Х.: ХУПС, 2016. – № 2 (46). – С. 146-149.

51. Смирнов С. А. Модели системы нейросетевых экспертов безопасной маршрутизации в облачных антивирусных системах / А. А. Смирнов, А. К. Дидык, А. Н. Дреев, С. А. Смирнов // Системи обробки інформації: зб. наук. праць. – Х.: ХУПС, 2016. – Вип. 3 (140). – С. 36-39.

52. Смирнов С. А. Метод безопасной маршрутизации на базовом множестве путей передачи метаданных в облачные антивирусные системы / В. Л. Бурячок, С. А. Смирнов // Системи управління, навігації та зв'язку. – Полтава, 2016. – Вип. 4(40). – С. 57-62.

53. Смирнов С. А. Способ контроля линий связи телекоммуникационной системы облачного антивируса / А. А. Смирнов, А. К. Дидык, А. Н. Дреев, С. А. Смирнов // Збірник наукових праць Харківського університету Повітряних Сил. – Харків: ХУПС, 2016. – № 2 (47). – С. 148-152.

54. Смирнов С. А. Дослідження та реалізація GERT-моделі технології розповсюдження комп'ютерних вірусів для захисту телекомунікаційних систем / В. Л. Бурячок, Мохамад Абу Таам Гани, С. А. Смирнов // Інформаційні технології та комп'ютерна інженерія: зб. тез доп. наук.-практ. конф., м. Кіровоград, 4 грудня 2014 р. – Кіровоград: КНТУ, 2014. – С. 168.

55. Смирнов С. А. Исследование математических моделей технологии распространения компьютерных виру сов / А. А. Смирнов, Мохамад Абу Таам Гани, С. А. Смирнов // Актуальні питання забезпечення кібернетичної безпеки та захисту інформації: зб. наук. праць міжнар. наук.-практ. конф., м. Київ, 25-28 лютого 2015 р. – К.: Європейський університет, 2015. – С. 90-91.

56. Смирнов С. А. Метод управления доступом к «облачным» ресурсам для защиты телекоммуникационных систем / Мохамад Абу Таам Гани, А. А. Смирнов, С. А. Смирнов // Всеукраїнська науково-практична конференція «Інформаційна безпека держави, суспільства та особистості», м. Кіровоград, 16 квітня 2015 р.: зб. тез доп. – Кіровоград: КНТУ, 2015. – С. 50-52.

57. Смирнов С. А. Разработка метода управления доступом в интеллектуальных узлах коммутации / А. А. Смирнов, Мохамад Абу Таам Гани, С. А. Смирнов // Проблеми і перспективи розвитку ІТ-індустрії: зб. тез VII міжнар. наук.-практ. конф., м. Харків, 17-18 квітня 2015 р. – Х.: ХНЕУ, 2015. – С. 14.

58. Смирнов С.А. Реализация метода управления доступом в интеллектуальных узлах коммутации / А.А. Смирнов, Мохамад Абу Таам Гани, С.А. Смирнов // Збірник тез XVII міжнародного науково-практичного семінару «Комбінаторні конфігурації та їх застосування». м. Кіровоград. 17-18 квітня 2015 р. – Кіровоград: КНТУ. – 2015. – С. 91-92.

59. Смирнов С. А. технология передачи сигнатур в облачные антивирусные системы для обеспечения защищенности телекоммуникационных сетей / А. А. Смирнов, С. А. Смирнов // Збірник тез V міжнародної науково-технічної конференції «ITSEC», Київ, 19-22 травня 2015 р. – К.: НАУ 2015. – С. 12-13.

60. Смирнов С. А. Реализация математической модели интеллектуального узла коммутации для обеспечения защищенности телекоммуникационной сети / Мохамад Абу Таам Гани, А. А. Смирнов, С. А. Смирнов // Інформаційна та економічна безпека (INFECO-2015): зб. тез II Міжнар. наук.-практ. Інтернет-конф., м. Харків, 21-22 травня 2015 р. – Х.: ХІБС УБС НБУ, 2015. – С. 20-24.

61. Смирнов С. А. Разработка математической модели технологии распространения компьютерных вирусов в информационно-телекоммуникационных сетях / Мохамад Абу Таам Гани, А. А. Смирнов, С. А. Смирнов // Сборник тезисов XI международной конференции «Стратегия качества в промышленности и образовании», г. Варна, Болгария, 01-06 июня 2015 г. – Варна: ТУВ, 2015. – С. 488-491.

62. Смирнов С. А. Метод управления доступом к облачным телекоммуни-кационным ресурсам для обеспечения защиты даннях / Мохамад

Абу Таам Гани, А. А. Смирнов, С. А. Смирнов // Комп'ютерні технології та інформаційна безпека: зб. тез доп. міжнар. наук.-практ. конф., м. Кіровоград, 2-3 липня 2015 р. – Кіровоград: КНТУ, 2015. – С. 4-5.

63. Смирнов С. А. Имитационная модель системы управления доступом к облачным антивирусным телекоммуникационным ресурсам / Мохамад Абу Таам Гани, А. А. Смирнов, С. А. Смирнов // Збірник тез першої всеукраїнської науково-практичної конференції «Перспективні напрями захисту інформації» (м. Затока, 7-9 вересня 2015 р.). – Одеса: ОНАЗ, 2015. – С. 90-94.

64. Смирнов С. А. Разработка комплекса gert-моделей технологии облачной антивирусной защиты телекоммуникационной системы / А. А. Смирнов, С. А. Смирнов, А. К. Дидык // Інформаційні технології та взаємодії» (IT & I): зб. тез II міжнар. наук.-практ. конф., м. Київ, 3-5 листопада 2015 р. – К.: КНУ ім. Тараса Шевченка, 2015. – С. 65-67.

65. Смирнов С. А. Разработка моделей телекоммуникационной системы формирования и обработки метаданных в облачных антивирусных системах / А. А. Смирнов, С. А. Смирнов, А. К. Дидык // Информационные и телекоммуникационные технологии: образование, наука, практика: сб. тезисов II междунар. научно-практ. конф., г. Алматы, Казахстан, 3-4 декабря 2015 г. – Алматы: КазНИТУ им. К.И. Сатпаева, 2015. – С. 309-313.

66. Смирнов С. А. gert-модели технологии облачной антивирусной защиты / А. А. Смирнов, С. А. Смирнов, А. К. Дидык // Безпека українського суспільства в концепції вступу в постіндустріальне суспільство ЄС: зб. тез Круглого столу, м. Київ, 16 грудня 2015 р. – К.: Європейський університет, 2015. – С.41-43.

67. Смирнов С. А. Алгоритмы формирования множества маршрутов передачи метаданных в облачные антивирусные системы / А. А. Смирнов, С. А. Смирнов, А. К. Дидык // Актуальні питання забезпечення кібернетичної безпеки та захисту інформації: зб. наук. праць II Міжнар. наук.-

практ. конф., м. Київ, 24-27 лютого 2016 р. – К.: Європейський університет, 2016. – С. 140-142.

68. Смирнов С. А. Разработка и реализация метода безопасной маршрутизации метаданных в облачные антивирусные системы / А. А. Смирнов, С. А. Смирнов, А. К. Дидык // Securitea informationala 2015-2016: Conferenta internationala (editia a XII-a), Chisinau, Moldova, 3 martie 2016. – Chisinau: ADSEM, 2016. – С. 90-96.

69. Смирнов С. А. Алгоритм формирования базового множества маршрутов передачи метаданных в облачные антивирусные системы / А. А. Смирнов, С. А. Смирнов, А. К. Дидык // Інформатика та системні науки (ІСН-2016): зб. тез VII всеукр. наук.-практ. конф., м. Полтава, 10-12 березня 2016 р. – Полтава: ПУЕТ, 2016. – С. 261-263.

70. Смирнов С. А. Система обработки и формирования начального состояния маршрутизации метаданных в облачные антивирусные системы / А. А. Смирнов, С. А. Смирнов, А. К. Дидык // Проблеми кібербезпеки інформаційно-телекомунікаційних систем: зб. тез наук.-практ. конф., м. Київ, 10-11 березня 2016 р. – К.: КНУ ім. Тараса Шевченка, 2016. – С. 81-82.

71. Смирнов С. А. Алгоритм безопасной маршрутизации на базовом множестве путей передачи метаданных в программный сервер облачной антивирусной системы / А. А. Смирнов, С. А. Смирнов, А. К. Дидык // Інформаційна безпека та комп'ютерні технології (IS&CT): зб. тез міжнар. наук.-практ. конф., м. Кіровоград, 24-25 березня 2016 р. – Кіровоград: КНТУ, 2016. – С. 73.

72. Смирнов С. А. Исследование способа контроля линий связи телекоммуникационной системы для облачных антивирусов / А. А. Смирнов, С. А. Смирнов, А. К. Дидык // Збірник тез першої міжнародної науково-практичної конференції «Проблеми науково-технічного та правового забезпечення кібербезпеки у сучасному світі» (ПНПЗК-2016), м. Харків, 30 березня - 1 квітня 2016 р. – Х.: НТУ «ХПІ», 2016. – С. 14.

73. Смирнов С. А. Разработка способа контроля линий связи телекоммуникационной системы для облачных антивирусов / А. А. Смирнов, С. А. Смирнов, А. К. Дидык // Матеріали XVIII міжнародного науково-практичного семінару «Комбінаторні конфігурації та їх застосування» (м. Кіровоград, 15-16 квітня 2016 р.). –Кіровоград: КНТУ, 2016. – С. 182-186.

74. Смирнов С. А. Разработка и исследование способа контроля линий связи телекоммуникационных сетей для облачных антивирусных систем / А. А. Смирнов, С. А. Смирнов, А. К. Дидык // Проблеми і перспективи розвитку ІТ-індустрії: VIII міжнар. наук.-практ. конф., м. Харків, 28-29 квітня 2016 р.: зб. тез. – Х.: ХНЕУ, 2016. – С. 48.

75. Смирнов С. А. Модель системы нейросетевых экспертов безопасной маршрутизации для облачных антивирусных систем / А. А. Смирнов, С. А. Смирнов, А. К. Дидык // Інформаційна та економічна безпека (INFECO-2016): зб. тез III міжнар. наук.-практ. конф., м. Харків, 28-30 кві. 2016 р. – Х.: ХННІ ДВНЗ «УБС», 2016. – С. 178-182.

76. Смирнов С. А. Метод безопасной маршрутизации метаданных в облачные антивирусные системы / А. А. Смирнов, С. А. Смирнов, А. К. Дидык // Сборник тезисов XII международной конференции «Стратегия качества в промышленности и образовании» (г. Варна, Болгария, 30 мая - 02 июня 2016 г.). – Варна: ТУВ, 2016. – С. 581-585.

77. Смирнов С. А. Оценка эффективности метода безопасной маршрутизации метаданных в облачные антивирусные системы / А. А. Смирнов, С. А. Смирнов, А. С. Коваленко // РадіоЕлектроніка та ІнфоКомунікації: зб. тез першої наук. - техн. конф., м. Київ, 11-16 вересня 2016 р. – К.: НТУУ «КПІ», 2016. – С. 17.

Додаток А
(обов'язковий)

Технічне завдання

Зміст

1 Найменування та область застосування

Це технічне завдання розповсюджується на розробку системи кібербезпеки захищеного обміну інформацією у мережі під керуванням ОС Linux.

2 Підстава для розробки

Підставою для розробки служить завдання на випускну кваліфікаційну роботу за першим (бакалаврським) рівнем вищої освіти, видане на кафедрі кібербезпеки та програмного забезпечення (нак. № 12-02 від 5.01.2023 року).

3 Мета та призначення розробки

Метою випускної кваліфікаційної роботи за першим (бакалаврським) рівнем вищої освіти є розробка програмного забезпечення системи кібербезпеки захищеного обміну інформацією у мережі під керуванням ОС Linux.

4 Джерела розробки

Джерелом цієї випускної кваліфікаційної роботи за першим (бакалаврським) рівнем вищої освіти є стосовна до теми література і існуючі аналоги.

5 Технічні вимоги

5.1 Склад продукції

Складниками розробки є:

– вибір і обґрунтування методів реалізації проекту;

– розробка програмної частин системи, а також розробка взаємодії системи кібербезпеки з ОС та з користувачем;

– розробка програми, що реалізує спроектовані алгоритми роботи системи.

5.2 Показники призначення

Система повинна забезпечувати:

– системи кібербезпеки захищеного обміну інформацією у мережі під керуванням ОС Linux;

– цілісність даних у процесі роботи та при зберіганні;

– простий, інтуїтивно зрозумілий інтерфейс.

5.3 Вимоги до функціональних характеристик

Розроблене програмне забезпечення не повинно мати обмежень на версію драйверів та операційної системи.

5.4 Вимоги до архітектури

Компонент, що розробляється повинен використовувати системні засоби та апаратні засоби, що на даному етапі розвитку обчислювальної техніки найбільше поширені.

5.5 Вимоги до надійності

Програмні модулі написані по всім правилам, які стосуються стандартних викликів процедур, функцій, методів і форм, визначених технічною документацією на середовище розробки.

5.6 Умови експлуатації

Робочі місця користувачів ПЗ повинні задовольняти наступним умовам експлуатації:

– температура повітря: 19-20 град. по Цельсію;

– відносна вологість повітря до 80%;

– атмосферний тиск 107 кПа.

5.7 Вимоги до складу та параметрів технічних засобів

Програмне забезпечення повинно бути реалізоване на ПЕОМ архітектури IBM PC, працювати в ОС Linux і з сумісними з цією платформою пристроями і прикладним програмним забезпечення.

5.8 Вимоги до інформаційної і програмної сумісності

Переносність програмного забезпечення повинна бути забезпечена за рахунок його реалізації стандартного інтерфейсу взаємодії з ОС, що працюють під управлінням ОС Linux.

5.8.1 Обладнання

Комп’ютер Intel® Celeron/8 Mb/1.2 Gb/SVGA 14” 1Mb або сумісні з ним.

5.8.2 Мова програмування

Середовище Visual C++.

5.8.3 Вхідні дані

Опис алгоритму роботи запропонованої системи.

5.8.4 Вихідні дані

Робоча програма.

6 Вимоги до програмної документації

Програмна продукція повинна бути представлена у виді опису структури даних, схем та опису алгоритму, а також текстів вихідних модулів програмного забезпечення згідно ЄСПД .

7 Перелік документів, що розробляються

– Структурна схема системи – 1 аркуш.
– Функціональна схема системи – 1 аркуш.
– Діаграма процесів – 1 аркуш.
– Блок-схема алгоритму роботи програми – 2 аркуша.
– Пояснювальна записка – 77 аркушів.

8 Етапи розробки

8.1 Збір і обробка інформації по темі випускної кваліфікаційної роботи за першим (бакалаврським) рівнем вищої освіти. Постановка задачі на виконання випускної кваліфікаційної роботи за першим (бакалаврським) рівнем вищої освіти (складання ТЗ).

8.2 Проведення досліджень або експериментальних робіт для уточнення основних положень випускної кваліфікаційної роботи за першим (бакалаврським) рівнем вищої освіти.

8.3 Розробка функціональних схем, блок схем алгоритмів роботи програмного забезпечення.

8.4 Побудова схем взаємодії даних.

8.5 Створення прототипу ПЗ.

8.6 Віднаходження ПЗ, аналіз отриманих результатів.

8.7 Оформлення пояснювальної записки і виконання робіт по графічній частині.

11 Порядок контролю та приймання

11.1 Подання випускної кваліфікаційної роботи за першим (бакалаврським) рівнем вищої освіти на попередній захист 23.05.2023 р.

11.2 Подання випускної кваліфікаційної роботи за першим (бакалаврським) рівнем вищої освіти на захист 2.06.2023 р.

Додаток Б

(обов’язковий)

Міністерство освіти і науки України

Центральноукраїнський національний технічний університет

ЗАТВЕРДЖУЮ

Керівник випускної кваліфікаційної роботи за

першим (бакалаврським) рівнем вищої освіти

_____ Смірнов О.А.

Програмне забезпечення системи кібербезпеки захищеного обміну інформацією у мережі під керуванням ОС Linux

Лістинг програми

Код документу 12

Носій: CD/DVD-диск / USB-флеш-накопичувач

Загальна кількість аркушів: 26

Літера: РП

Кропивницький – 2023 року

Вихідний код Linux - клієнту

currentchat.cpp

```
** Форма створюється з читання ui файлу  './currentchat.ui'
**

#include "currentchat.h"
#include "linuxsocket.h"
#include <qvariant.h>
#include <qpushbutton.h>
#include <qtextedit.h>
#include <qlayout.h>
#include <qtooltip.h>
#include <qwhatsthis.h>
/*
*  Конструктори CurrentChat які є потомками батька, з
*  іменем  'name'  й флаг віджета встановлений у  'f'.
*/
CurrentChat::CurrentChat( LinuxSocket *def_linux_socket, QWidget* parent, const
char* name, WFlags fl,
    QHostAddress *def_ip_address, QString *def_alias ) : QWidget( parent, name,
    fl )
    if(   !name )   {
        setName(  «Поточний чат»  );
    }
    resize( 364,  344  );
    setCaption( trUtf8(  ""  )   );

    message = new QTextEdit( this,   «Повідомлення»  );
    message->setGeometry( QRect( 10,   170,  366,   121 )   );
    message->setTextFormat( Qt::PlainText );

    chat = new QTextEdit( this, «Чат» );
    chat->setGeometry( QRect( 10,   10,  366,   151 )   );
    chat->setReadOnly( TRUE );
    chat->setTextFormat( Qt::PlainText );


    button_send = new QPushButton( this, «Кнопка відправлення» );
    button_send->setGeometry( QRect( 225,  300,  70,  26 )   );
    button_send->setText( trUtf8(  «Відправити»  )   );

    button_close = new QPushButton( this, «Кнопка завершення» )
    button_close->setGeometry( QRect( 305,  300,   70,  26 )   )
    button_close->setText( trUtf8( «Закрити» ) );

    button_clear = new QPushButton( this,   «Кнопка очищення»  )
    button_clear->setGeometry( QRect( 10, 300, 70, 26 ) );
    button_clear->setText( trUtf8( «Очистити» ) );

    if( def_ip_address )  {
       ip_address = def_ip_address;

    }
    if( def_alias )
    {
       alias = def_alias;

    }
    linux_socket = def_linux_socket;

    connect( button_send,  SIGNAL( released()   ),  this,  SLOT(
    sendButtonReleased()) );

    connect( button_clear, SIGNAL( released() ), this, SLOT(
clearButtonReleased()   )   );
```

```
    connect( button_close, SIGNAL( released() ), this, SLOT(
closeButtonReleased()   )   );

}
/*
*    Знищення об’єктів, та вивільнення ресурсів
*/
CurrentChat::~CurrentChat()
{
    // немає необхідності видаляти віджети потомків,  Qt робить це автоматично
}

#include "currentchat.moc" void CurrentChat::sendButtonReleased()
{
    chat->append( QString(  "Ви писали  :" )   );
    chat->append( message->text()   );

    linux_socket->Send( &message->text(),  ip_address );
    message->clear();

}

void CurrentChat::clearButtonReleased() {
    chat->clear();

}
void CurrentChat::closeButtonReleased()
{
    chat->clear();

}
```

Заголовочний файл для поточного чату

currentchat.h

```
** Інтерфейс форми створюється з читання ui файлу  './currentchat.ui'
**

#ifndef CURRENTCHAT_H

#define CURRENTCHAT_H

#include <qvariant.h>
#include <qwidget.h>
#include <qhostaddress.h>
#include <qstring.h>

class QVBoxLayout;
class QHBoxLayout;
class QGridLayout;
class QPushButton;
class QTextEdit;
class LinuxSocket;
class CurrentChat  : public QWidget

{
Q_OBJECT public:
    CurrentChat( LinuxSocket *def_linux_socket, QWidget* parent = 0, const char*
name = 0, WFlags fl = 0,
          QHostAddress *def_ip_address = 0,  QString *def_alias = 0 );

    ~CurrentChat();

    QPushButton* button_send;
    QPushButton* button_close;
    QPushButton* button_clear;
    QTextEdit* chat;
    QTextEdit* message;

QHostAddress *ip_address;
QString *alias;
protected:
    LinuxSocket *linux_socket;

Загальні слоти:  // Загальні слоти
  void sendButtonReleased();
  void clearButtonReleased();
  void closeButtonReleased();

};


#endif // CURRENTCHAT_H
```

Підпрограмма формування нового вікна обміну повідомленями та мультимедійною інформацією

newcontactdialog.cpp

```
#include "newcontactdialog.h"
#include "schatcontactlist.h"

#include <qvariant.h>
#include <qlabel.h>
#include <qlineedit.h>
#include <qpushbutton.h>
#include <qmime.h>
#include <qdragobject.h>
#include <qlayout.h>
#include <qtooltip.h>
#include <qwhatsthis.h>
#include <qimage.h>
#include <qpixmap.h>
#include <qmessagebox.h>
static QPixmap uic_load_pixmap_NewContactDialog( const QString &name )
{
    const QMimeSource *m = QMimeSourceFactory::defaultFactory()->data( name );

    if ( !m )
      return QPixmap();
      QPixmap pix;

    QImageDrag::decode( m,  pix );
    return pix;

}
/*
*  Конструктори NewContactDialog які є потомками батька, з
*  name  'name'  й флаг віджета встановлений у  'f'. *
*/
NewContactDialog::NewContactDialog( QWidget* parent,    const char* name,  bool
modal, WFlags fl )
    :  QDialog( parent,  name, modal,   fl )
{
    if  (   !name )
       setName(  "NewContactDialog"  );

    setEnabled( TRUE );

    resize( 250,   133 );

setSizePolicy( QSizePolicy(   (QSizePolicy::SizeType)5,
(QSizePolicy::SizeType)5, 0, 0, sizePolicy().hasHeightForWidth()  )  );
setCaption( trUtf8( "New Contact" ) );

    text_label = new QLabel( this,   "text_label"  );

    text_label->setGeometry( QRect( 70, 10, 120, 30 ) );
    text_label->setText( trUtf8( "Enter IP address" ) );

    ip_edit_field = new QLineEdit( this,   "ip_edit_field"  );

    ip_edit_field->setGeometry( QRect( 20, 40, 211, 31 ) );

    ip_edit_field->setFrameShape( QLineEdit::LineEditPanel );
    ip_edit_field->setFrameShadow( QLineEdit::Sunken );

    button_ok = new QPushButton( this,   "button_ok"  );
    button_cancel = new QPushButton( this, "button_cancel" );
    button_cancel->setGeometry( QRect(  150,   80,   81,  31  )   );
```

```
    button_ok->setGeometry( QRect( 20, 80, 81, 31 ) );
    button_ok->setText( trUtf8( "OK" ) );

    // сигнали та слоти з'єднання
    connect( button_cancel, SIGNAL( released()   ),  this, SLOT(
buttonCancelReleased()   )   );

    connect( button_ok, SIGNAL( released()  ), this, SLOT( buttonOkReleased()  )
);

    connect( ip_edit_field, SIGNAL( returnPressed()  ), this, SLOT(
buttonOkReleased()   )   );

    my_parent=( SchatContactList* )parent;

    init();

}


    Знищення об'єктів, та вивільнення ресурсів

NewContactDialog::~NewContactDialog() {
      // немає необхідності видаляти віджети потомків,  Qt робить це автоматично
}

void NewContactDialog::init()
{
// }

void NewContactDialog::buttonCancelReleased()
{
    this->hide();

}

void NewContactDialog::buttonOkReleased()
{
    QString address;

    QHostAddress *ipAddress = new QHostAddress();
    address = ip_edit_field->text();

    if( !ipAddress->setAddress( address )   ) {
       QMessageBox mb( "Просто чат", "Запис IP адрес", QMessageBox::Information,
       QMessageBox::Ok, QMessageBox::NoButton, QMessageBox::NoButton );
       mb.exec();
       return;

    }
    my_parent->newIPAddress( ipAddress );
    this->hide();
```

Заголовочний файл для формування нового вікна обміну повідомленями та мультимедійною інформацією

newcontactdialog.h

```
#ifndef NEWCONTACTDIALOG_H
#define NEWCONTACTDIALOG_H


#include <qvariant.h>
#include <qdialog.h>
#include <qhostaddress.h>
#include <qstring.h>

class QVBoxLayout;
class QHBoxLayout;
class QGridLayout;
class QLabel;
class QLineEdit;
class QPushButton;

class NewContactDialog :  public QDialog
{
    Q_OBJECT

public:
    NewContactDialog( QWidget* parent = 0,  const char* name = 0,  bool modal =
FALSE, WFlags fl = 0 );

    ~NewContactDialog();

    QLabel* text_label;
    QPushButton* button_cancel;
    QLineEdit* ip_edit_field;
    QPushButton* button_ok;


Загальні слоти:
    virtual void init();

    virtual void buttonCancelReleased();
    virtual void buttonOkReleased();


protected:
    friend class SchatContactList;
    SchatContactList *my_parent;

};


#endif // NEWCONTACTDIALOG_H
```

Підпрограмма створення листу контактів

schatcontactlist.cpp

```
#include "newcontactdialog.h"
#include "schatcontactlist.h"
#include "currentchat.h"
#include "linuxsocket.h"

#include "list.h"

#include <qvariant.h>
#include <qlistbox.h>
#include <qtabwidget.h>
#include <qwidget.h>
#include <qmime.h>
#include <qdragobject.h>
#include <qlayout.h>
#include <qtooltip.h>
#include <qwhatsthis.h>

#include <qaction.h>
#include <qmenubar.h>
#include <qpopupmenu.h>
#include <qtoolbar.h>
#include <qimage.h>
#include <qpixmap.h>
#include <qtextedit.h>


/*
*  Конструктори SchatContactList які є потомками батька, з
*  імені 'name' й флаг віджета встановлений у 'f'. *
*/
SchatContactList::SchatContactList( QWidget* parent,    const char* name, WFlags
fl )
    :  QMainWindow( parent,  name,  fl )
{
    if  (   !name )
    setName(  «Контактний лист чату»  );
    resize( 576,   408  );

    setCaption( trUtf8(  "Просто чат"  )   );

    setCentralWidget( new QWidget( this,   "qt_central_widget"  )   );

    tab_widget = new QTabWidget( centralWidget(), "tab_widget" );
    tab_widget->setGeometry( QRect( 180,   12,  391,  365  )   );


    contact_list = new QListBox( centralWidget(), «Лист контактів» );
    contact_list->setGeometry( QRect( 10,   0,   161,  371 )   );

    // дії
    new_connection_dialog_action = new QAction( this,
"new_connection_dialog_action"  );

    new_connection_dialog_action->setText( trUtf8(  "Нове з'єднання"  )   );

    new_connection_dialog_action->setMenuText( trUtf8(  "&Нове з'єднання"  )
    );

    new_connection_dialog_action->setWhatsThis( trUtf8(  "Створення нового
з'єднання"  )   );

    new_connection_dialog_action->setAccel( 4194382  );
```

```
    delete_connection_action = new QAction( this,   "delete_connection_action"
    );
    delete_connection_action->setText( trUtf8(  "Видалення з’єднання"  )   );
    delete_connection_action->setMenuText( trUtf8(  "&Видалення з’єднання"  )
    ) ;
    delete_connection_action->setWhatsThis( trUtf8(  "Видалення поточного
    з’єднання"
)  );

    delete_connection_action->setAccel( 4194372  );

    opions_action = new QAction( this,   "opions_action"  );
    opions_action->setText( trUtf8(  "Опції"  )   );
    opions_action->setMenuText( trUtf8(  "&Опції"  )   );
    opions_action->setWhatsThis( trUtf8( "Меню опцій"  )   );
    opions_action->setAccel( 4194383 );

    exit_action = new QAction( this,   "exit_action" );
    exit_action->setText( trUtf8(  "Вихід"  )   );
    exit_action->setMenuText( trUtf8(  "& Вихід "  )   );
    exit_action->setAccel( 4194385  );


    // menu_bar
    menu_bar = new QMenuBar( this,   "menu_bar"  );

    popup_menu = new QPopupMenu( this );
    new_connection_dialog_action->addTo( popup_menu );
    delete_connection_action->addTo( popup_menu );
    opions_action->addTo( popup_menu );
    exit_action->addTo( popup_menu );

    menu_bar->insertItem( trUtf8(  "Система"  ),  popup_menu );

connect( exit_action,  SIGNAL( activated()   ),  this,  SLOT( close()   )   );
connect( new_connection_dialog_action,  SIGNAL( activated()   ),  this,  SLOT(
showDialog()   )   );

    connect( contact_list, SIGNAL( doubleClicked(QListBoxItem*) ), this, SLOT(
contactListDoubleClicked(QListBoxItem*)   )   );

    connect( contact_list, SIGNAL( returnPressed(QListBoxItem*) ), this, SLOT(
contactListDoubleClicked(QListBoxItem*)   )   );

    connect( this,  SIGNAL( messageReceived( QString*,  QHostAddress*)   ),
this, SLOT( messageHandle( QString*, QHostAddress*)  )  );

    init();

    QThread::start();


/*
*    Знищення об’єктів, та вивільнення ресурсів
*/
SchatContactList::~SchatContactList()
{
      // немає необхідності видаляти віджети потомків,  Qt робить це автоматично
}

void SchatContactList::init()
{
    contact_dialog = new NewContactDialog( this );

    list = new List();

    linux_socket = new LinuxSocket();

}
```

```
void SchatContactList::run()
{
    while( TRUE )
    {
        QString *message;
        QHostAddress *from;

        linux_socket->Receive(  &message,   &from );
        emit messageReceived( message,  from );



void SchatContactList::helpIndex() {


void SchatContactList::helpContents()
{


void SchatContactList::helpAbout()
{
}
void SchatContactList::showDialog()
{
    contact_dialog->show();

}
/** Ця функція додає IP адреси до листа контактів*/
void SchatContactList::newIPAddress( QHostAddress *new_address )
{
    int i;
    int index;


    if(   ( i = list->getIndex( new_address )   )   >= 0 )
    {
       contact_list->setSelected( list->getListBoxItem( i ), TRUE );
       return;

    index = list->add( new_address );

    QString *alias = new QString( new_address->toString()   );

    list->add( alias,  index );
    contact_list->insertItem( *alias );

    contact_list->setSelected( contact_list->count()   - 1,  TRUE );
    list->add( contact_list->item( contact_list->count()   - 1 ),  index );


    contact_list->sort();

}
/** Без опису*/
void SchatContactList::contactListDoubleClicked( QListBoxItem *list_box_item ) {
    int i;

    i = list->getIndex( list_box_item );
    if(  list->getCurrentChat(  i  )   != NULL )
    {
        int index = list->getTabIndex( i );

        tab_widget->setTabEnabled( tab_widget->page( index )   ,  TRUE );

        tab_widget->showPage( tab_widget->page( index )   );

        return;

    }
```

```
    current_chat = new CurrentChat( linux_socket,  this,  NULL,   0,
    list->getIP( i ),  list->getAlias( i )   );
    tab_widget->addTab( current_chat,  list_box_item->text()   );
    tab_widget->setTabEnabled( current_chat,  TRUE );
    tab_widget->showPage( current_chat );


    list->add( current_chat,  i );

    list->setTabIndex( tab_widget->count() - 1, i );
    current_chat->show();

}

/** Без опису*/
void SchatContactList::closeContactList( QListBoxItem *list_box_item )
{
    int i;


    i = list->getIndex( list_box_item );

    if( list->getCurrentChat( i )  != NULL )
    {
        int index = list->getTabIndex( i );

        tab_widget->setTabEnabled( tab_widget->page( index )   ,  TRUE );

        tab_widget->showPage( tab_widget->page( index )   );
        return;


    current_chat = new CurrentChat( linux_socket,  this,  NULL,   0,
    list->getIP( i ), list->getAlias( i )  );
    tab_widget->addTab( current_chat,  list_box_item->text()   );
    tab_widget->setTabEnabled( current_chat,  TRUE );
    tab_widget->showPage( current_chat );


    list->add( current_chat,  i );

    list->setTabIndex( tab_widget->count() - 1, i );
    current_chat->show();

}

/** слот, з'єднується з messageReceived. Ця функція оперує з отриманими
повідомленями, або мультимедійними даними*/
void SchatContactList::messageHandle( QString *message,  QHostAddress *from )
{
    int index;

    index = list->getIndex( from );

    if( index >= 0 ) {
        current_chat = list->getCurrentChat( index );

        current_chat->chat->append( from->toString()  + QString(  " записуємо:"
        )   );
        current_chat->chat->append( *message );
        // Тобто  повертає список контактів;

    }
    newIPAddress( from );

    index = list->getIndex( from );

    current_chat = new CurrentChat( linux_socket,  this,   «Поточний чат»  );
    current_chat = new CurrentChat( linux_socket,  this,  NULL,   0,
```

```
    list->getIP( index ),  list->getAlias( index )   );
    tab_widget->addTab( current_chat,   *list->getAlias( index )   );

    tab_widget->setTabEnabled( current_chat,  TRUE );
    tab_widget->showPage( current_chat );

    list->add( current_chat,  index );

    list->setTabIndex( tab_widget->count()   - 1,  index );
    current_chat->show();


    list->getCurrentChat(     index )->chat->append( from->toString() + QString(
    записуємо:"  )   );

    list->getCurrentChat(     index )->chat->append( *message );

}
```

Заголовочний файл для створення листу контактів

schatcontactlist.h

```
#ifndef SCHATCONTACTLIST_H
#define SCHATCONTACTLIST_H



#include <qvariant.h>
#include <qhostaddress.h>
#include <qmainwindow.h>

#include <qthread.h>
class QVBoxLayout;
class QHBoxLayout;
class QGridLayout;
class QAction;
class QActionGroup;
class QToolBar;
class QPopupMenu;
class QListBox;
class QListBoxItem;
class QTabWidget;
class QWidget;


class SchatContactList  : public QMainWindow,  public QThread {
Q_OBJECT public:
    SchatContactList( QWidget* parent = 0, const char* name = 0, WFlags fl =
WType_TopLevel  );

    ~SchatContactList();

    QTabWidget* tab_widget;
    QListBox* contact_list;
    QMenuBar *menu_bar;
    QPopupMenu *popup_menu;
    QPopupMenu *help_menu;

/*
    QAction* help_contents_action;
    QAction* help_index_action;
    QAction* help_about_action;

*/
    QAction* new_connection_dialog_action;
    QAction* delete_connection_action;
    QAction* opions_action;
    QAction* exit_action;



Загальні слоти:
    virtual void init();
    virtual void helpIndex();
    virtual void helpContents();
    virtual void helpAbout();



private slots:
    virtual void showDialog();

protected:
    void run();

    /** Ця функція додає IP адреси до листа контактів*/
```

```
    virtual void newIPAddress( QHostAddress *new_address );


    friend class NewContactDialog;

    NewContactDialog *contact_dialog;

    friend class CurrentChat;
    CurrentChat *current_chat;

    friend class List;
    List *list;

    friend class LinuxSocket;
    LinuxSocket *linux_socket;

  Загальні слоти: // Загальні слоти /** Без опису*/
  virtual void contactListDoubleClicked( QListBoxItem *list_box_item );
  Загальні слоти: // Загальні слоти
  /** слот, з'єднується з messageReceived. Ця функція оперує з отриманими
  повідомленями, або мультимедійними даними
*/
  void messageHandle( QString *message, QHostAddress *from );
  signals: // сигнали
   /** сигнал, що повідомлення отримане */
  void messageReceived( QString *message, QHostAddress *from );

};


#endif //  SCHATCONTACTLIST_H
```

Підпрограмма роботи з сокетами в Linux

linuxsocket.cpp

```
#include "linuxsocket.h"

#include <unistd.h>
#include <stdio.h>

#include <qstring.h>
#include <qhostaddress.h>

LinuxSocket::LinuxSocket() {
   struct sockaddr_un addr;

   sockfd = socket( AFJLINUX,   SOCK_STREAM,   0  );

   if(   !sockfd ) {
      perror(  "socket"  ) ;

   }
   addr.sunJfamily = AFJLINUX;

   strcpy(  addr.sun_path,   INT_SOCKET_PATH );


   if( connect( sockfd,   ( struct sockaddr*  )&addr,  sizeof( addr )   )   !=
   0 ) {
      perror(  «з'єднання»  );

   }
}
LinuxSocket::~LinuxSocket()
{
    close( sockfd );

}
void LinuxSocket::Receive( QString **message,  QHostAddress **from )
{
    tJmessageJint msg;

    recv( sockfd,   &msg,   sizeof( msg ),   0 );


    *message = new QString( msg.message );
    *from = new QHostAddress( msg.ipaddr );

}
void LinuxSocket::Send( QString *message,  QHostAddress *to )
{
    tJmessageJint msg;

    sprintf( msg.message,   "%s", message->latin1()   );
    msg.ipaddr = to->ip4Addr();

    send( sockfd,   &msg,   strlen( msg.message )  + sizeof( msg.ipaddr )  + 1,
    0 );

}
```

Заголовочний файл для роботи з сокетами в Linux

linuxsocket.h

```
linuxsocket.h

description



#ifndef LINUXSOCKET_H
#define LINUXSOCKET_H

#include <sys/socket.h>
#include <sys/un.h>


#define INT_SOCKET_PATH "/tmp/schat3538"

#define MAX_MESSAGE_SIZE      64*1024

typedef struct {
   u_int32_t ipaddr;

   char message[MAX_MESSAGE_SIZE];

} t_message_int;



class QString;
class QHostAddress;


/**Цей клас забезпечує обслуговування з AF_LINUX сокетами
  */
class LinuxSocket {
private:
    int sockfd;

protected:
    friend class SchatContactList;
    SchatContactList *contact_list;

public:
    LinuxSocket();
    ~LinuxSocket();

    void Receive( QString **message, QHostAddress **from );
    void Send( QString *message, QHostAddress *to );

};


#endif
```

Вихідний код Linux – сервера

Програма демона чату

schat_deamon.c

```
#ifdef HAVE_CONFIG_H
#include <config.h>

#endif


#include <stdio.h>
#include <unistd.h>
#include <stdlib.h>
#include <errno.h>

#include <sys/poll.h>
#include <sys/socket.h>
#include <sys/un.h>
#include <netinet/in.h>


#include "schat_deamon.h"

t_message_int message_int;
t_message_ext message_ext;

int ext_socket, int_socket;

struct pollfd sockets[2];
int pollres;

int e;
int y0;

int n_int_msg, n_ext_msg;


extern int init_int_socket( void )
extern int init_ext_socket( void )


void z0( void );
void z1( void );
void z2( void );
void z3( void );
void z4( void );
void z5( void );
void z6( void );
void z7( void );

void log( void )   {


   if( e  != -1 )   {
      printf(  "Дія  e%d                :      ",  e );

   }
      switch( e )   { case 0:
      printf(  "у локальному сокеті з'явилося повідомлення \n"  );
      break;
      case 1:
      printf(  "у мережному сокеті з'явилося повідомлення \n"  );
      break;
```

```
case 2:
printf(  "у локальному сокеті повідомлень не залишилося^"  );
break;
case 3:
printf(  "у мережному сокеті повідомлень не залишилося^"  );
break;
case 4:
printf(  "перевищений ліміт по кількості повідомлень, отриманих підряд з
локального сокету   (  %d )\n",  MAX_N_INT_MSG );
break;
case 5:
printf(  "перевищений ліміт по кількості повідомлень, отриманих підряд з
мережного сокету
сокета ( %d )\n", MAX_N_EXT_MSG );

break;
case 6:
printf(  "завершення роботи клієнта або аварійна ситуація\и"  );

break;
case 7:
printf(  "успішний прийом локального повідомлення\n"  );
break;
case 8:
printf(  "успішне перетворення в мережний формат"  );
break;
case 9:
printf(  "успішне відправлення повідомлення в мережу^"  );
break;
case 10:
printf(  "успішний прийом повідомлення з мережі^"  ) ;
break;
case 11:
printf(  "успішне перетворення в локальний формат"  ) ;
break;
case 12:
printf(  "успішне відправлення локального повідомлення^"  );

break;



if( y0 != 7 ) {
printf(  "A0 перейшов до стану %d      :      ",  y0 );



switch( y0 )   { case 0  :
printf(  "Очікування приходу повідомлення в один із сокетів^^"  );
 break;
 case 1  :
 printf(  "Одержання повідомлення з мережного сокету^^"  );
 break;
 case 2  :
 printf(  "Перетворення мережного повідомлення для наступного відправлення
 його локальному клієнтові^^"  );
 break;
 case 6  :
   printf(  "Відправлення повідомлення через локальний сокет" );

   printf("        повідомлення= '%s'\n",
                  ( char* )message_int.message );

   printf("        адреса відправника    %d.%d.%d.%d\n\n",
                  ( message_int.ipaddr >> 24 )   % 256, (
                  message_int.ipaddr >> 16 )   % 256, (
                  message_int.ipaddr >> 8 )   % 256, (
                  message_int.ipaddr)   % 256 );
```

```
         break;
         case 4  :
       printf(  "Одержання повідомлення з мережного сокету^^"  );
       break;
       case 5  :
       printf(  "Перетворення повідомленняу внутрішній формат^^"  );
       break;
       case 3  :
         printf(  "Відправлення повідомлення через мережний сокет" );

         printf("        повідомлення= '%s'\n",
                         ( char* )message_ext.message );

         printf("        адрес назначения = %d.%d.%d.%d\n\n",
                         ( message_ext.ipaddr >> 24 )   % 256, (
                        message_ext.ipaddr >> 16 )   % 256, (
                        message_ext.ipaddr >> 8 )   % 256, (
                        message_ext.ipaddr)   % 256 );

         break;
         case 7  :
         printf(  "Демон закінчив роботу\n"  );

         break;
         default  :
         printf(  "Непередбачувана ситуація\n\n"  );

         break;

    }
    fflush(  stdout );

}

void A0( void )
{
   if( e == 6 ) { y0 = 7;
   };
   log();

      switch( y0 )   { case 0:
        z0();

        if( e == 0 ) { y0 = 1;
        break;
        } if( e == 1 ) { y0 = 4;
        break;
        }
        break;
        case 1:



z1();
if( e = if(  e = break;
case 2: z2();
        if( e == 8 ) { y0 = 3;
        break;
        };
        break;

      case 3: z3() ,
      if( e == 9 )   { y0 = 1;
      break;
        };
      break;
      case 4:
        if( e == 5  )   { y0 = 1;
        break;
```

```
          };


   z4();
   if( e if( e
   break;
   case 5:
   z5();

if( e == 11 )   { y0 = 6;
break;
  } break;
case 6:
   z6();

   if( e == 12 ) { y0 = 4;
   break;
   }
   break;
   case 7:
   z7();

break;
default:
   y0 = 7;

   break;



}
void z0( void ) { n_int_msg = 0;
n_ext_msg = 0;

   pollres = poll( сокетами,  2,  -1 );

   if(  sockets[0].revents  &   (  POLLERR  |   POLLHUP  )   )   { perror(
      "Помилка локального сокету"  );

      e = 6;

      return;

   }
   if( sockets[1].revents & ( POLLERR | POLLHUP ) ) {
      perror( "Помилка мережного сокету"  );

      e = 6;

      return;
   }
   if(  sockets[0].revents  &   (  POLLIN  |   POLLPRI  )   )   {
      e = 0;

      return;

   }
   if( sockets[1].revents & ( POLLIN | POLLPRI ) ) { e = 1;


return;


   e = 6;

}
void z1( void )   { int nbytes;
```

```
   errno = 0;

   if( n_int_msg >= MAX_N_INT_MSG )   {
      e = 4;

      n_int_msg = 0;
      return;

   }
   n_int_msg++;

   bzero( message_int.message,   sizeof( message_int.message )   );

   nbytes = recv( int_socket,   &message_int,   sizeof( message_int ),
   MSG_DONTWAIT
);

   if( errno == EAGAIN )  { e = 2;

      return;

   }
   if(    nbytes < 0 )   {
      perror(  "Помилка в отриманому повідомлені з локального сокету"  );

      e = 6;

      return;

   }
   e = 7;

   return;

}
void z2( void )   {
   message_ext.ipaddr = message_int.ipaddr;

   sprintf( message_ext.message,   "%s", message_int.message );

   e = 8;

   return;

}
void z3( void )   {
   struct sockaddr_in clnt_addr;

   int caddrlen = sizeof( struct sockaddr_in );

   clnt_addr.sin_family = AF_INET;

   clnt_addr.sin_port = htons(   ( u_short  )EXT_SOCKET_PORT  );
   clnt_addr.sin_addr.s_addr = htonl( message_ext.ipaddr );

   if( sendto( ext_socket, message_ext.message, strlen( message_ext.message )  +
      1,   0, ( struct sockaddr*  )&clnt_addr,  caddrlen )  < 0 )   { perror(
      "Error in sending message to network"  );

      e = 6;

      return;

   }
   e = 9;

   return;
```

```
void z4( void )   {
   struct sockaddr_in clnt_addr;

   int caddrlen = sizeof( struct sockaddr_in );

   int len;

   errno = 0;

   if( n_ext_msg >= MAX_N_EXT_MSG ) { e = 5;

      n_ext_msg = 0;
      return;

   }
   n_ext_msg++;

   bzero( message_ext.message,  sizeof( message_ext.message )   );

   len = recvfrom( ext_socket, message_ext.message,   sizeof(
   message_ext.message
),
         MSG_DONTWAIT,   ( struct sockaddr*  )&clnt_addr,   &caddrlen );

   if( errno == EAGAIN )  { e = 3;

      return;

   }
   if( len < 0 ) {
      perror( "Помилка в отриманому повідомлені з мережі"  );

      e = 6;

      return;

   }
   message_ext.ipaddr = ntohl( clnt_addr.sin_addr.s_addr );

   e = 10;

   return;

}

void z5( void )   {
   message_int.ipaddr = message_ext.ipaddr;

   sprintf( message_int.message,   "%s", message_ext.message );

   e = 11;

   return;

}
void z6( void )   {
   if( send( int_socket,   &message_int,
  strlen( message_int.message )  + sizeof( message_int.ipaddr )  + 1,   0  )  < 0
)   {
      perror( "Помилка в відправленому  повідомлені через локальний сокет"  );

      e = 6;

      return;

   }
   e = 12;

   return;
```

```
void z7( void )   {
   close( int_socket );
   close( ext_socket );
   unlink(  INT_SOCKET_PATH );

}

int main( void )   {
   printf(  "\nДемон запущений\n\n"  );

   printf(  "Чекаємо з'єднання з клієнтом..."  );

   int_socket = init_int_socket();
   ext_socket = init_ext_socket();

   printf(  " готово\n\n"  );

   e = -1;

   if(   !int_socket  ||   !ext_socket )   { e = 6;

   }
   sockets[0].fd = int_socket;
   sockets[1].fd = ext_socket;

   sockets[0].events = POLLIN | POLLPRI;
   sockets[1].events = POLLIN  |   POLLPRI;

   y0 = 0;

      while( y0   != 7 )   { A0();

   }
   return 0;

}
```

Заголовочний файл для демона чату

schat_deamon.h

```
#ifndef     SCHAT_DEAMON_H

#define  SCHAT_DEAMON_H


#define INT_SOCKET_PATH      "/tmp/schat3538"
#define EXT_SOCKET_PORT      3538
#define MAX_MESSAGE_SIZE    64*1024

#define MAX_N_INT_MSG 2 0

#define MAX_N_EXT_MSG 20
typedef struct  {
   u_int32_t ipaddr;

   char message[MAX_MESSAGE_SIZE];
   }  t_message_int;


typedef struct  {
   u_int32_t ipaddr;

   char message[MAX_MESSAGE_SIZE];
   }  t_message_ext;



#endif /*  SCHAT_DEAMON_H */
```

Підпрограма утиліт демону чату

schat_deamon_utils.c

```
#include <unistd.h>
#include <stdio.h>


#include <sys/socket.h>
#include <sys/un.h>
#include <netinet/in.h>


#include "schat_deamon.h"


int init_int_socket( void )   {
   struct sockaddr_un addr;
   int sockfd,   csockfd;

   unlink(  INT_SOCKET_PATH );

   sockfd = socket( AF_LINUX,   SOCK_STREAM,   0  );

   if( sockfd <= 0 )   {
      perror(  "Помилка при створенні локального сокету"  );
      goto fail;

   }
   addr.sun_family = AF_LINUX;

   strcpy(  addr.sun_path,   INT_SOCKET_PATH );

      if( bind( sockfd,   ( struct sockaddr* )&addr,   sizeof( addr )   )   )
      { perror(  "Error in binding local socket"  );
      goto fail;

   }
   if( listen( sockfd,   1 )   )   {
      perror(  "Помилка при прослуховуванні локального сокету" );
      goto fail;

      }
      csockfd = accept( sockfd, NULL,  NULL );

      if( csockfd <= 0 )  {
      perror(  "Помилка при додаванні локального з'єднання"  );
      goto fail;

      }
      if( close( sockfd ) != 0 )  {
      perror(  "Помилка при закритті локального сокету"  );

      }
      return csockfd;

      fail:
      return 0;

      }

int init_ext_socket( void )   {
   struct sockaddr_in addr;
   int sockfd;
```

```
   sockfd = socket( AF_INET, SOCK_DGRAM, IPPROTO_UDP );
   if( sockfd <= 0 ) {
      perror(  "Помилка при створенні мережного сокету"  );
      goto fail;

   }
   addr.sin_family = AF_INET;

   addr.sin_port = htons( ( u_short )EXT_SOCKET_PORT );
   addr.sin_addr.s_addr = INADDR_ANY;

      if( bind( sockfd,   ( struct sockaddr* )&addr,  sizeof( addr )   )   )  {
      perror(  "Помилка в мережному сокеті"  );
      goto fail;

   }
   return sockfd;

fail:
return 0;
```