

Міністерство освіти і науки України
Державний заклад
«Луганський національний університет імені Тараса Шевченка»

Навчально-науковий інститут математики та інформаційних технологій
Кафедра інформаційних технологій та систем

Ян Кекай

АНАЛІЗ ТЕХНОЛОГІЇ МУЛЬТИКАСТ З ВИКОРИСТАННЯМ JAVAFX

**кваліфікаційна робота
здобувача вищої освіти другого (магістерського) рівня
освітньої програми «Комп'ютерні мережі»
за спеціальністю 123 Комп'ютерна інженерія**

Особистий підпис _____ Ян Кекай

Науковий керівник _____ Геннадій МОГИЛЬНИЙ,
кандидат технічних наук, доцент
кафедри інформаційних технологій
та систем

Завідувач кафедри _____ Микола СЕМЕНОВ,
кандидат педагогічних наук, доцент
кафедри інформаційних технологій
та систем

Полтава – 2025

АНОТАЦІЯ

Ян Кекай.

Тема: Аналіз технології мультикаст з використанням JAVAFX.

Спеціальність: 123 «Комп'ютерна інженерія»

Установа: ЛНУ імені Тараса Шевченка, 2025р.

Магістерська робота містить: загальна кількість сторінок – 99, з них 82 – пояснювальна записка, 17 – додатки, 19 рисунків, перелік літератури – 22 джерел.

Об'єкт дослідження – сучасні технології обміну повідомленнями в мережі.

Предмет дослідження – реалізація технологій для організації обміну повідомленнями у мові програмування Java.

Мета роботи – комплексний аналіз особливостей реалізації мережевих і мультимедійних засобів Java і розробка системи обміну повідомленнями.

Результати роботи. Проведено аналіз особливостей використання мультимедійних технологій Java/JavaFX. Досліджено особливості архітектури мереженої системи обміну повідомленнями з урахуванням можливостей реалізації у Java. На засадах попереднього аналізу розроблені класи для роботи сервісу автентифікації та контролю користувачів, а також класи клієнтського програмного забезпечення. Наведено опис процесу моделювання та тестування. Розроблена універсальна мультимедійна заставка для запуску програми клієнта, з використанням технології JavaFX.

Висновок. В результаті роботи було проведено аналіз мережених технологій Java, досліджено можливості Java при обробці мультимедійної інформації та отримано програмний комплекс для системи обміну повідомленнями з використанням Multicast.

Ключові слова. КЛАС, МОДЕЛЮВАННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ, КЛІЄНТ, СЕРВЕР, JAVAFX, MULTICAST, ОБМІН ПОВІДОМЛЕННЯМИ, ІНТЕРФЕЙС КОРИСТУВАЧА.

ABSTRACT

Yan Kekai

Theme: Analysis of Multicast Technology Using JAVAFX..

Speciality: 123 "Computer Engineering"

Institution: Luhansk Taras Shevchenko National University (LTSNU), 2025.

The masters work contains: total number of pages – 99, of which 82 – explanatory note, 17 – enclosures, 19 - pictures, bibliography – 22 source.

A research object is modern technologies of exchanging messages in the network.

The article of research is implementation of technologies for the organization of messaging in the Java programming language.

An objective of the work is a comprehensive analysis of the features of the implementation of Java network and multimedia tools and the development of the messaging system.

Job performances. The analysis of peculiarities of using multimedia technologies Java / JavaFX is carried out. The features of the architecture of the network messaging system are considered with consideration for the implementation possibilities in Java. Based on the preliminary analysis, classes have been developed for the operation of the authentication and user control service, as well as classes of client software. The description of the modeling and testing process is given. A universal multimedia screen saver for launching the client program using JavaFX technology has been developed..

Conclusions. As a result of the work, the analysis of Java networking technologies was conducted, the possibilities of Java for processing multimedia information were explored and a software package for the messaging exchanging system using Multicast was got.

Keywords. CLASS, SOFTWARE MODELLING, CLIENT, SERVER, JAVAFX, MULTICAST, EXCHANGE OF MESSAGES, USER INTERFACE.

Міністерство освіти і науки України
Державний заклад «Луганський національний університет
імені Тараса Шевченка»
Факультет (інститут) Інститут математики та інформаційних технологій
(повна назва)
Кафедра Інформаційних технологій та систем
(повна назва)
Галузь знань 12, Інформаційні технології
(код, назва)
Напрямок підготовки (спеціальність) 123 «Комп'ютерна інженерія»
(код, назва)

ЗАВДАННЯ
на кваліфікаційну роботу освітньо-кваліфікаційного рівня
« магістр »
(назва рівня)

Студенту Ян Кекай
(прізвище, ім'я, по батькові)

Керівник кваліфікаційної роботи Могильний Геннадій Анатолійович
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)

1. Тема роботи Аналіз технології мультикаст з використанням JAVAFX

затверджена наказом по університету _____

2. Термін подання студентом закінченої роботи на кафедру _____

3. Вихідні дані до роботи У результаті виконання роботи необхідно
дослідити можливості мережених технологій Java; можливості підтримки
засобів обробки мультимедійної інформації у JavaFX;
розробити систему обміну повідомленнями з використанням технології Multicast
(визначаються кількісні або (та) якісні показники, яким повинен відповідати об'єкт розробки)

4. Зміст пояснювальної записки (перелік питань, що їх належить розробити)
Вибір архітектури на засадах мережених технологій Java , вибір технології
обробки та доставки мультимедійних повідомлень, вибір системи
аутентифікації, розробка сервера, розробка клієнта,. розробка системи обміну
повідомленнями. розробка мультимедійної заставки для запуску клієнта.
(визначаються назви розділів або (та) перелік питань, які повинні увійти до тексту ПЗ)

5. Індивідуальний план виконання кваліфікаційної роботи

| № | Заходи | Термін виконання |
|----|---|-----------------------------------|
| 1. | Вибір теми роботи, вивчення наукової літератури, затвердження теми та керівника. | До 30 жовтня 2023 |
| 2. | Аналіз літературних джерел за темою роботи. Розробка ТЗ. Розробка та апробація методики дослідно-експериментальної роботи. Подання структури теоретичної частини роботи (пояснювальної записки) та плану експериментальних досліджень. | Другий тиждень жовтня 2024 |
| 3. | Робота над теоретичною частиною. Подання теоретичної частини роботи для першого читання керівником. Розробка методики тестування | До 1 грудня 2024 |
| 4. | Усунення зауважень, урахування рекомендацій керівника. Аналіз структури програмного забезпечення. | Перший тиждень грудня 2024 |
| 5. | Поетапний аналіз та обговорення результатів. Перевірка стану виконання роботи. | Перший тиждень грудня 2024 |
| 6. | Урахування рекомендацій керівника, усунення недоліків, підготовка варіанта роботи до передзахисту. Оформлення документації до проекту. | До 15 грудня 2024 |
| 7. | Попередній захист роботи на кафедрі. | За місяць до державної атестації |
| 8. | Доопрацювання роботи з урахуванням рекомендацій після передзахисту. Розробка презентації. Підготовка графічних матеріалів. Перевірка на плагіат. Подання роботи науковому керівникові та рецензентові на підготовку відгуку та рецензії | За 10 днів до державної атестації |
| 9. | Подання на кафедру остаточного варіанта роботи, з відгуком керівника і рецензена. | За 3 дні до державної атестації |

ЗМІСТ

| | |
|--|----|
| Перелік умовних позначень | 7 |
| Вступ..... | 8 |
| Розділ 1. Мультимедійні можливості Java..... | 11 |
| 1.1 API платформи Java | 11 |
| 1.2 Огляд sampled пакету API Java Sound..... | 13 |
| 1.3 Додаткова бібліотека JMF | 22 |
| 1.4 Класи обробки мультимедіа у пакетах JavaFX..... | 28 |
| Розділ 2. Види мережевих технологій локальних мереж..... | 31 |
| 2.1 Мережева технологія | 31 |
| 2.2 Протокол TCP | 35 |
| 2.3 Протокол UDP | 40 |
| 2.4 Прикладний рівень протоколів | 43 |
| Розділ 3. Використання специфічних адресів, методів та моделей | 47 |
| 3.1 Метод передавання бродкаст | 47 |
| 3.2 Метод передавання мультикаст..... | 50 |
| 3.3 Модель передавання клієнт-сервер..... | 53 |
| Розділ 4.Розробка системи обміну повідомленнями | 56 |
| 4.1 Розробка та дослідження архітектури системи..... | 56 |
| 4.2 Сервер – основні принципи роботи..... | 62 |
| 4.3 Клієнт – основні принципи роботи | 68 |
| Висновки | 76 |
| Список літературних джерел | 79 |
| Додатки..... | 82 |
| Додаток А..... | 82 |
| Додаток Б | 84 |

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

| | |
|------|---|
| ПК | Програмний комплекс |
| ОС | Операційна система |
| JVM | Java Viertual Michine (Віртуальна Java машина) |
| IDE | Integrated Development Environment (Інтегроване середовище розробки) |
| JIT | Just-in-time («На лету») |
| ARM | Advanced RISC Machines |
| RMI | Remote Method Invocation (відділений виклик методу) |
| HTTP | HyperText Transfer Protocol (Протокол передавання гіпертексту) |
| JDBC | Java DataBase Connectivity (З'єднання з базами даних) |
| RIA | Rich Internet Application (Насичений інтернет додаток) |
| CSS | Cascading Style Sheets (Каскадні таблиці стилів) |
| MSN | Microsoft Network |
| IPTV | Internet Protocol Television (Телебачення по протоколу Інтернету) |
| IGMP | Internet Group Management Protocol (Протокол керування групами Інтернету) |
| IP | Internet Protocol |
| TCP | Transmission Control Protocol (TCP, протокол керування передаванням) |
| UDP | (англ. User Datagram Protocol — протокол пользовательских дейтаграм) |

ВСТУП

Сьогодні однією з проблем сучасних програмних засобів є відсутність способу передачі текстових, звукових і відео- повідомлень в мережі в реальному часі. Існуючі сторонні програмні засоби (наприклад, Skype) працюють по Інтернет-технології, вимагають високошвидкісного підключення до Інтернету, і викликають цілий комплекс проблем для організації обміну конфіденційними повідомленнями в локальній мережі компанії. Операційна система (ОС), наприклад, Windows не надає можливості передачі повідомлень між користувачами в сучасному графічному інтерфейсі.

В даний час в ОС Windows і в багатьох інших ОС надана можливість передати повідомлення за допомогою консольної команди, аналогічної NET SEND. Загалом, можна говорити, що в даний момент у багатьох ОС не існує стандартного програмного забезпечення для передачі як тестових, так і мультимедійних повідомлень. Попередній аналіз показав, що існуючі ОС, взагалі не передбачають обміну мультимедійними повідомленнями.

Одна з проблем – великий мережевий трафік. Для вирішення даної проблеми необхідно використовувати інші способи доставки повідомлень, наприклад технологію Broadcast або Multicast, які дозволяють відправити одне повідомлення відразу всім користувачам. Інші технології передачі повідомлень припускають відправку повідомлення персонально кожному користувачу. Однак, дані технології не передбачають методів аутентифікації користувача. Тому на першому етапі створення універсальної системи обміну повідомленнями різного типу (текстові, звукові, відео) в реальному часі стоїть завдання –вирішення проблеми створення програмного забезпечення з вбудованими засобами аутентифікації, шифрування, яке працює в будь-якій ОС.

Один із шляхів, вирішення даної проблеми є створення програмного комплексу на мові програмування Java, яка є універсально для багатьох ОС. Таким чином, на даному етапі, рішення цієї проблеми поки не знайдено, але

існує ряд нових технологій, на основі яких можна впритул наблизитися до її вирішення. Тому в даній роботі можливо висловити наступне..

Гіпотеза дослідження – використання спеціальних технологій передачі інформації (Multicast) дозволить створити систему обміну мультимедійними повідомленнями між користувачами в локальній мережі.

Об'єкт дослідження – сучасні технології обміну повідомленнями в мережі.

Предмет дослідження – реалізація технологій для організації обміну повідомленнями в мові програмування Java.

Мета роботи – комплексний аналіз особливостей реалізації мережевих і мультимедійних засобів Java і розробка системи обміну повідомленнями.

Інноваційна новизна – знайдені особливості використання різних технологій передачі в мережі TCP/IP і особливості використання мультимедійних компонентів Java, які дозволяють провести комплексну оцінку створення системи передачі мультимедійних повідомлень в реальному часі для будь-якої операційної системи.

Досягнення цієї мети передбачає вирішення таких основних завдань:

- Провести аналіз переваг і недоліків технології Java.
- Дослідити можливості різних мережевих технологій в сучасній Java.
- Провести аналіз засобів обробки мультимедіа, реалізованих в різних компонентах Java.
- Провести моделювання та аналіз програмного забезпечення для системи обміну повідомленнями.
- Розробити та реалізувати систему обміну повідомленнями.

У першому розділі проведено аналіз можливості обробки звукової та відео інформації, реалізованих в ядрі Java і додаткових бібліотеках сторонніх розробників. Описано їх переваги та недоліки, особливості роботи.

У другому розділі розглянуті види мережевих технологій і протоколів для локальних мереж.

Третій розділ присвячений опису використання специфічних адрес мережі, методів і моделей передачі даних. Описано підходи і способи підтримки різних методів передачі інформації по мережі, що реалізовані в Java.

Четвертий розділ присвячений опису роботи програмного комплексу - системи обміну мультимедійними повідомленнями на основі Multicast. Описано процеси моделювання його архітектури. Запропоновано метод реєстрації та контролю користувачів на основі додаткового компонента – сервера аутентифікації. Описано методи та процес моделювання компонентів програмного комплексу. Показаний приклад реалізації мультимедійних компонентів на основі JavaFX.

РОЗДІЛ 1. МУЛЬТИМЕДІЙНІ МОЖЛИВОСТІ JAVA

1.1 API платформи Java

API Java Sound - це низькорівневий API для здійснення та керування входом і виведенням звукових носіїв, включаючи як аудіо, так і музичний інструмент цифрового інтерфейсу (MIDI). API Java Sound забезпечує явний контроль над можливостями, які зазвичай потрібні для введення та виведення звуку, в рамках, що сприяє розширенню та гнучкості.

Оскільки звук настільки фундаментальний, API Java Sound задовольняє потреби широкого кола розробників програм. Потенційні сфери застосування включають:

- Фреймворки комунікації, такі як конференц-зв'язок та телефонія
- Системи доставки контенту кінцевого користувача, такі як мультимедійні плеєри та музика за допомогою потокового вмісту
- Інтерактивні прикладні програми, такі як ігри та веб-сайти, які використовують динамічний вміст
- Створення та редагування вмісту
- Інструменти, інструментарій та утиліти як аудіо, так і мультимедійний цифровий інтерфейс (MIDI).

API Java Sound забезпечує явний контроль над можливостями, які зазвичай потрібні для введення та виведення звуку, в рамках, що сприяє розширенню та гнучкості.

API Java Sound забезпечує найнижчий рівень підтримки звуку на платформі Java. Це забезпечує прикладні програми з великою кількістю контролю над звуковими операціями, і це розширюване. Наприклад, Java Sound API поставляє механізми для встановлення, доступу та керування системними ресурсами, такими як аудіо мікшери, MIDI-синтезатори, інші аудіо- та MIDI-пристрої, читачі файлів та письменники, а також перетворювачі

форматів звуку. API Java Sound не включає складних редакторів звуку або графічних інструментів, але надає можливості для створення таких програм. Це підкреслює низький рівень контролю за межами того, що зазвичай очікує кінцевий користувач.

Є інші API платформи Java, які мають елементи, пов'язані із звуком. **Java Media Framework (JMF)** - це API вищого рівня, який наразі доступний як стандартний розширення для платформи Java. JMF визначає уніфіковану архітектуру, протокол обміну повідомленнями та інтерфейс програмування для захоплення та відтворення медіа на основі часу. JMF забезпечує просте рішення для базових прикладних програм медіапрогравачів та дозволяє синхронізувати різні типи носіїв, наприклад, аудіо та відео.

З іншого боку, програми, що фокусуються на звучанні, можуть скористатися API Java Sound, особливо якщо вони потребують більш просунутих функцій, таких як можливість ретельно контролювати відтворення буферизованого звуку або безпосередньо управляти MIDI-синтезатором. Інші API Java із звуковими аспектами включають Java 3D та API для телефонії та мови. Реалізація будь-якого з цих API може використовувати внутрішню програму Java Sound API, але не вимагає цього.

Пакети

API Java Sound включає підтримку як цифрового аудіо, так і MIDI-даних. Ці два основних модуля функціональності надаються в окремих пакунках:

- `javax.sound.sampled` Цей пакет вказує інтерфейси для захоплення, змішування та відтворення цифрового (вибіркового) звуку.
- `javax.sound.midi` Цей пакет забезпечує інтерфейси для синтезу MIDI, послідовності та транспортування подій.

Два інших пакети дозволяють постачальникам послуг (на відміну від розробників додатків) створювати власні програмні компоненти, які розширюють можливості реалізації API Java Sound [1]:

- `javax.sound.sampled.spi`

- javax.sound.midi.spi

1.2 Огляд `samplerd` пакету API Java Sound

Пакет `javax.sound.samplerd` пов'язаний із аудіо транспортом - іншими словами, Java Sound API фокусується на відтворенні та захопленні аудіо. Центральне завдання, яке стосується адрес Java-звуку API, полягає в тому, як перемістити байт форматованих аудіо-даних у систему та за його межами. Це завдання передбачає відкриття аудіовхідних та вихідних пристроїв та керування буферами, які наповнюються звуковими даними у реальному часі. Це також може включати в себе змішування декількох потоків аудіо в один потік (будь то для введення або виведення). Передача звуку в систему або її виймання повинна правильно оброблятися, коли користувач запитує, щоб потік звуку був запущений, призупинений, відновлений або зупинений.

Щоб підтримати цю основну увагу на базовому вхідному та вихідному звукових даних, API Java Sound пропонує методи конвертації між різними форматами звукових даних, а також для читання та запису звичайних типів звукових файлів. Однак він не намагається бути всеосяжним інструментарієм звукових файлів. Певна реалізація API Java Sound не повинна підтримувати великий набір типів файлів або перетворення формату даних. Сторонні постачальники послуг можуть надавати модулі, які "підключаються" до існуючої реалізації для підтримки додаткових типів файлів і конверсій.

Буферизоване та небуферне керування звуком

API Java Sound може обробляти аудіо транспорту як в потоковому режимі, так і в буфері, і в пам'яті, без буферу. "Потокове передавання" тут використовується в загальному сенсі для позначення обробки аудіо-байтів у режимі реального часу; це не стосується конкретного, добре відомого випадку надсилання аудіо через Інтернет в певному форматі. Іншими словами, потік аудіо - це просто безліч аудіо-байтів, які прибувають більш-менш з тією ж швидкістю, до якої вони обробляються (відтворюються, записуються тощо).

Операції над байтами починаються до того, як всі дані надійдуть. У потоковому режимі, особливо у випадку аудіо введення, а не аудіо виходу, користувачу не обов'язково знати заздалегідь, якої довжини звук і коли він закінчиться. Ви просто обробляєте один буфер аудіоданих одночасно, доки операція не буде зупинена. У випадку виведення звуку (відтворення), також потрібно буде буферизація даних, якщо звук, який ви хочете відтворити, занадто великий, щоб він міг уміститися повністю в пам'яті. Іншими словами, ви передасте аудіо-байти до звукового движка кусочками, і він дбає про відтворення кожного зразка в потрібний час. Існують механізми, які дозволяють легко зрозуміти, скільки даних потрібно доставляти в кожному фрагменті.

API Java Sound також дозволяє небуферизоване транспортування лише у випадку відтворення, якщо ви вже маєте всі аудіодані, які знаходяться під рукою, і не є надто великими, щоб вписати їх у пам'ять. У цій ситуації додаткова програма не потребує буферизації звуку, хоча буферний підхід до реального часу все ще доступний за бажанням. Замість цього весь звук може бути попередньо завантажений в пам'ять для подальшого відтворення. Оскільки всі звукові дані завантажуються заздалегідь, відтворення може розпочатися негайно, наприклад, як тільки користувач натискає кнопку «Пуск». Це може бути перевагою в порівнянні з буферною моделлю, де відтворення повинно очікувати першого заповнення буфера. Крім того, вбудована модель, що зберігається в пам'яті, дозволяє легко звучати з циклом (циклічно) або встановлювати довільні позиції в даних.

Щоб відтворити або зафіксувати звук за допомогою API Java Sound, потрібно як мінімум три речі: відформатовані аудіодані, змішувач та рядок. Кожен з них пояснюється нижче.

Формати даних

Формат даних розповідає вам про те, як інтерпретувати серію байтів "сирих" вибіркових аудіоданих, таких як зразки, які вже були прочитані із

звукового файлу, або зразки, зняті з входу мікрофона. Можливо, вам знадобиться знати, наприклад, скільки бітів складають один зразок (представлення найкоротшого моменту звуку), і, подібно, вам може знадобитися знати частоту дискретизації звуку (наскільки швидко зразки повинні слідувати один одному). Під час налаштування для відтворення або зйомки ви задаєте формат даних звуку, який ви захоплюєте чи відтворюєте.

У API Java Sound формат даних представлений об'єктом `AudioFormat`, який включає в себе наступні атрибути:

- Технологія кодування, як правило, модуляція імпульсного коду (PCM)
- Кількість каналів (1 для моно, 2 для стерео та ін.)
- Коефіцієнт вибірки (кількість зразків в секунду на канал)
- Кількість бітів на вибірку (на канал)
- Частота кадрів
- Розмір кадру в байтах
- Порядок байтів (big-endian або small-endian)

PCM - це один з видів кодування звукової форми сигналу. API Java Sound включає в себе два кодування PCM, які використовують лінійне квантування амплітуди, а також цілі значення, що підписані або не підписані. Лінійне квантування означає, що кількість, що зберігається в кожному зразку, прямо пропорційна (за винятком будь-яких спотворень) до вихідного звукового тиску в той момент і аналогічно пропорційно зміщенню гучномовця або барабанної перетинки, яка вібрує звуком в той момент. Нелінійне кодування відображає амплітуду вихідного звуку до збереженого значення, використовуючи нелінійну функцію, яка може бути розроблена для отримання більшої амплітудної роздільної здатності для тихого звучання, ніж для гучних звуків.

Змішувач

Багато програмних інтерфейсів програмного забезпечення (API) для звуку використовують поняття звукового пристрою. Пристрій часто є програмним інтерфейсом для фізичного пристрою введення / виведення. Наприклад, пристрій звукового введення може представляти можливості входу звукової карти, включаючи мікрофонний вхід, аналоговий вхід на рівні лінії та, можливо, цифровий аудіовхід.

У Java Sound API пристрої представлені об'єктами Mixer. Метою змішувача є обробка одного або декількох потоків аудіовходу та одного або декількох потоків аудіовиходу. У типовому випадку він фактично об'єднує декілька вхідних потоків у один вихідний потік. Об'єкт Mixer може представляти можливості звукового змішування на фізичному пристрої, наприклад звуковій картці, яка може потребувати змішувати звук, який надходить до комп'ютера з різних входів, або звучання, що надходить із програм, і відбувається на виході.

Іншими словами, об'єкт Mixer може представляти можливості звукового змішування, які повністю реалізовані в програмному забезпеченні без будь-якого внутрішнього інтерфейсу для фізичних пристроїв.

У Java Sound API є такий компонент, як вхід мікрофона на звуковій картці, сам не вважається пристроєм, тобто змішувачем, а не портом в змішувач або з нього. Порт, як правило, забезпечує один потік аудіо в міксері або з нього (хоча потік може бути багатоканальним, наприклад, стерео). Змішувач може мати кілька таких портів. Наприклад, змішувач, який представляє вихідні можливості звукової карти, може об'єднати декілька потоків аудіо разом, а потім відправити змішаний сигнал на будь-який або всі різні вихідні порти, підключені до змішувача. Ці вихідні порти можуть бути, наприклад, роз'ємом для навушників, вбудованим динаміком або виходом на рівні лінії.

Лінія

Лінія це метафора фізичної консолі змішування також корисна для розуміння концепції лінії Java Sound API.

Лінія є елементом цифрового аудіо "трубопроводу", тобто шлях для переміщення аудіо в систему або виходу з неї. Зазвичай ця лінія являє собою шлях до міксера або з неї (хоча в технічному сенсі сам мікшер теж є своєрідною лінією).

Вхідні і вихідні аудіосигнали - це лінії. Вони аналогічні мікрофонам і динамікам, підключеним до фізичної консолі змішування. Інший вид лінії - це шлях до даних, через який програма може отримати вхідний аудіосигнал або відправити вихідний аудіо сигнал у змішувач. Ці шляхи даних аналогічні до трас мультитрейкового записуючого пристрою, підключеного до фізичної консолі змішування.

Одна різниця між лініями в API Java Sound і функціями фізичного змішувача полягає в тому, що аудіодані, що проходять через лінію в API Java Sound, можуть бути моно- або багатоканальними (наприклад, стерео). На відміну від цього, кожен з входів і виходів фізичного мікшера, як правило, є єдиним каналом звуку. Щоб отримати два або більше каналів виведення з фізичного змішувача, зазвичай використовуються два або більше фізичних виходів (принаймні, у випадку аналогового звуку, цифровий вихідний роз'єм часто багатоканальний). У Java Sound API кількість каналів у рядку визначається за допомогою AudioFormat даних, які поточно проходять через рядок. [2]

AudioSystem Class

Клас AudioSystem виступає в якості клірингової установи для аудіокомпонентів, включаючи вбудовані служби та окремо встановлені сервіси від сторонніх постачальників. AudioSystem слугує точкою входу додатка для доступу до цих встановлених зразкових аудіо ресурсів. Ви можете задати запит AudioSystem, щоб дізнатись, які різновиди ресурсів було

встановлено, а потім ви можете отримати доступ до них. Наприклад, програма може починатися з запрошення у класі AudioSystem, чи існує міксер, який має певну конфігурацію, наприклад, один з конфігурацій вводу чи виводу, ілюстрований раніше під час обговорення ліній. З мікшера програма потім отримає лінії даних і так далі.

Ось деякі з ресурсів, які прикладна програма може отримати від AudioSystem:

- Змішувачі У системі, як правило, встановлено кілька змішувачів. Зазвичай принаймні один - для вхідного аудіо та один - для виведення звуку. Також можуть бути змішувачі, які не мають портів вводу-виводу, але замість цього приймають аудіо з прикладної програми та передають змішане звучання назад у програму. Клас AudioSystem надає список всіх встановлених мікшери.
- Лінії Незважаючи на те, що кожна лінія пов'язана з мікшером, програма може отримати лінію безпосередньо з AudioSystem, не маючи чіткого зв'язку з змішувачами.
- Форматування конверсій Програма може використовувати перетворення формату для перекладу аудіоданих з одного формату в інший.

Файли та потоки Клас AudioSystem забезпечує методи перекладу між аудіофайлами та аудіо потоками. Він також може повідомляти про формат файлу звукового файлу і може записувати файли у різних форматах.

Кілька класів у API Java Sound надають корисну інформацію про пов'язані інтерфейси. Наприклад, Mixer.Info містить відомості про встановлений мікшер, наприклад постачальника мікшери, назву, опис та версію. Line.Info отримує клас певної лінії. Підкласи Line.Info включають Port.Info та DataLine.Info, які отримують інформацію, що стосується конкретного порту та лінії передачі даних, відповідно. Кожен з цих класів

описаний далі в відповідному розділі нижче. Важливо не плутати об'єкт Info із змішувачем чи об'єктом лінії, який він описує.

Отримання змішувача

Зазвичай, одна з перших речей, яку потрібно виконати для програми, яка використовує Java Sound API, - отримати мікшер або принаймні один рядок змішувача, щоб ви могли отримувати звук на комп'ютері або виходити з нього. Для програми може знадобитися конкретний змішувач, або ви, можливо, необхідно відобразити список всіх доступних змішувачів, щоб користувач міг вибрати його. У будь-якому випадку ви повинні дізнатись, які типи змішувачів встановлені. AudioSystem забезпечує наступний метод:

```
static Mixer.Info [] getMixerInfo ()
```

Кожний об'єкт Mixer.Info, який повертається за допомогою цього методу, визначає один тип встановленого змішувача. (Зазвичай у системі є не більше одного змішувача даного типу. Якщо трапляється, що це більше, ніж один з даного типу, то повернутому масиву до цих пір є тільки один Mixer.Info для цього типу). Програма може робити ітерацію над об'єктами Mixer .Inf, щоб знайти відповідний, відповідно до його потреб. Mixer.Info містить наступні рядки, щоб визначити вид змішувача:

- Ім'я
- Версія
- Постачальник
- Опис

Це довільні рядки, тому прикладна програма, яка потребує певного змішувача, повинна знати, чого очікувати і як порівнювати рядки з. Компанія, яка надає змішувач, повинна включити цю інформацію в свою документацію. Як альтернатива, і, можливо, більш типова, прикладна програма відобразить всі рядки об'єктів Mixer.Info користувача та дозволить користувачеві вибрати відповідний мікшер.

Коли знайдено відповідний мікшер, прикладна програма викличе наступний метод AudioSystem для отримання бажаного міксера:

```
static Mixer getMixer(Mixer.Info info)
```

Отримання лінії бажаного типу

Існує два способи отримати лінію:

- Безпосередньо з об'єкта `AudioSystem`
- З мікшера, який ви вже отримали від об'єкта `AudioSystem`.

В першому випадку, безпосередньо з `AudioSystem`, якщо змішувач не був отримано, і програма є простою, який дійсно потребує певної лінії, можна використовувати метод `AudioSystem`:

```
static Line getLine(Line.Info info),
```

що аналогічно методу `getMixer`, описаному вище.

На відміну від `Mixer.Info`, `Line.Info`, використаний як аргумент, не зберігає текстову інформацію, щоб вказати потрібну лінію. Замість цього зберігається інформація про клас потрібної лінії.

`Line.Info` є абстрактним класом, тому треба використовувати один з його підкласів (`Port.Info` або `DataLine.Info`), щоб отримати лінію. Наступний витяг коду використовує підклас `DataLine.Info`, щоб отримати та відкрити цільову рядок даних:

```
TargetDataLine line;
DataLine.Info info = new DataLine.Info(TargetDataLine.class, format);
// format is an AudioFormat object
if (!AudioSystem.isLineSupported(info)) {
    // Handle the error.
}
// Obtain and open the line.
try {
    line = (TargetDataLine) AudioSystem.getLine(info);
    line.open(format);
} catch (LineUnavailableException ex) {
    // Handle the error.
    //...
}
```

Цей код отримує об'єкт `TargetDataLine`, не вказуючи жодних атрибутів крім його класу та його аудіоформату. Ви можете використовувати аналогічний код для отримання інших видів ліній. Для `SourceDataLine` або `Clip`

просто замініть цей клас для `TargetDataLine` як класу змінної лінії, а також у першому аргументі конструктора `DataLine.Info`.

Для порту можна використовувати статичні екземпляри `Port.Info` у коді, подібному до наступного:

```
if (AudioSystem.isLineSupported(Port.Info.MICROPHONE)) {  
    try {  
        line = (Port) AudioSystem.getLine(  
            Port.Info.MICROPHONE);  
    }  
}
```

Другий варіант отримання лінії з мікзера полягає у наступному.

Інтерфейс `Mixer` включає варіації методів доступу `AudioSystem` для вихідних та цільових ліній, описаних вище. До таких методів міксерів відносяться ті, що приймають аргументи `Line.Info`, як це роблять методи `AudioSystem`. Тим не менш, міксер також включає ці варіанти, які не приймають жодних аргументів:

```
Line.Info[] getSourceLineInfo()  
Line.Info[] getTargetLineInfo()
```

Ці методи повертають масиви всіх об'єктів `Line.Info` для конкретного мікзера. Після того, як ви отримаєте масиви, ви можете виконати перебір над ними, викликаючи метод `getLine` `Mixer` для отримання кожного рядка, а потім відкритим методом `Line` для резервування використання кожного рядка для вашої програми.

Дозвіл на використання аудіо ресурсів

API `Java Sound` включає в себе клас `AudioPermission`, який вказує, які види доступу до аплету (або програми, запущеної менеджером безпеки) можуть мати у системі `sampled-audio`. Дозвіл на запис звуку регулюється окремо. Ці дозволи слід надавати обережно, щоб запобігти небезпеці безпеки, такому як несанкціоноване підслуховування. За замовчуванням аплети та додатки отримують дозвіл наступним чином:

- Аплет, що працює з менеджером безпеки аплетів, може відтворювати, але не записувати звук.
- Програма, що працює без менеджера безпеки, може як грати, так і записувати звук.
- Програма, запущена за допомогою менеджера безпеки за замовчуванням, може відтворювати, але не записувати аудіо.

Загалом, аплети запускаються під контролем менеджера безпеки і не дозволяється запис звуку. З іншого боку, програми не автоматично встановлюють менеджера безпеки і можуть записувати звук. (Однак, якщо менеджер безпеки за замовчуванням викликається явно для програми, програмі не дозволяється записувати звук).

Обидва аплети та програми можуть записувати звук навіть під час роботи з диспетчером безпеки, якщо їм надано явне дозвіл на це. [3]

1.3 Додаткова бібліотека JMF

Java Media Framework (JMF) - це бібліотека Java, яка дозволяє отримувати доступ до аудіо, відео та інші медіа на основі часу на Java-додатках та аплетах. Цей додатковий пакет, який може захоплювати, відтворювати, транслювати та перекодувати декілька медіаформатів, розширює платформу Java, Standard Edition (Java SE) і дозволяє розробляти крос-платформні мультимедійні програми.

JMF абстрактні медії працюють в об'єкті DataSource (для медіа, що читаються в JMF) та об'єкти DataSink (для експорту даних). Розробникам не надано значного доступу до відомостей про будь-який конкретний формат. ЗМІ представлені як джерела (які отримуються з URL-адрес), які можна читати, відтворювати, обробляти та експортувати (хоча не всі кодеки підтримують обробку та перекодування). Клас Manager пропонує статичні методи, які є основним точковим контактом з JMF для додатків.

Є три етапи обробки даних у архітектурі JMF: вхід, обробка та вихід.

Вхід включає роботу з пристроями захоплення, читання даних з файлів на жорсткому диску та всілякі введення даних в мережу. Під час етапу введення дані зчитуються з джерела та передаються в буфери до стадії обробки. Стадія введення може включати в себе читання даних з локального пристрою захоплення (наприклад, веб-камери або картки захоплення телевізора), файлу на диску або потоку з мережі.

Обробка складається з ряду кодеків та ефектів, призначених для зміни потоку даних до одного, придатного для виводу. Ці кодеки можуть виконувати такі функції, як стиснення або розпакування аудіо в інший формат, додавання водяного знаку, очищення шуму або застосування ефекту до потоку (наприклад, відлуння звуку). Після того, як етап обробки застосує перетворення до потоку, він передає інформацію на виході

Вихід включає в себе використання відео рендеринга, збереження виводу на диск та збереження виводу в мережу. Вихідний етап може приймати потік і передавати його на файл на диску, вивести його на місцевий відеовихід або передавати його по мережі. Наприклад, система JMF може читати вхід з карти пам'яті телевізора з локальної системи, яка записує вхід із відеомагнітофона на етапі введення. Потім він може передати його на етапі обробки, щоб додати водяний знак у кутку кожного кадра, а потім транслювати його по локальній внутрішній мережі на виході. [4]

Архітектура компонентів JMF

JMF побудований на основі архітектури компонентів. Компоненти поділяються на ці основні категорії: засоби обробки медіа, джерела даних, кодеки та ефекти, рендери та mux / demux.

Обробники медіа реєструються для кожного типу файлів, які JMF повинна мати можливість обробляти. Щоб підтримувати нові формати файлів, можна створити новий MediaHandler.

Обробник джерела даних управляє джерельними потоками з різних входів. Це може бути для мережевих протоколів, таких як http або ftp, або для простого введення з дисків.

Кодеки та ефекти - це компоненти, які приймають вхідний потік, застосовують до неї перетворення та виводять його. Кодеки можуть мати різні формати введення та виведення, а ефекти - це прості перетворення одного вхідного формату у вихідний потік того самого формату.

Відтворювач подібний до кодека, але остаточний випуск - це десь інший, ніж інший потік. VideoRenderer виводить кінцеві дані на екран, але інший вид рендеринга може виводити на інше обладнання, наприклад, на вивідну картку телевізора.

Mux / demuxer (що розшифровуються мультиплексорами та Demultiplexers) використовуються для об'єднання декількох потоків у єдиний потік або навпаки, відповідно. Вони корисні для створення та читання пакета аудіо та відео для збереження на диску як одного файлу або передачі через мережу [5].

```
import javax.swing.JDialog;
import javax.swing.JFrame;
import javax.swing.UIManager;
import javax.swing.UnsupportedLookAndFeelException;
import javax.swing.plaf.metal.MetalLookAndFeel;
public class ExampleJMF{
public static void main(String[] args){
JFrame.setDefaultLookAndFeelDecorated(true);
JDialog.setDefaultLookAndFeelDecorated(true);
try {
UIManager.setLookAndFeel(new MetalLookAndFeel());
} catch (UnsupportedLookAndFeelException e) {
e.printStackTrace();
}
new exampleFrame();
}
}

//-----
import java.awt.Toolkit;
import java.awt.event.WindowAdapter;
import java.awt.event.WindowEvent;
import javax.swing.JFrame;
```



```

public class exampleFrame extends JFrame {
private static final long serialVersionUID = 1L;
public exampleFrame()
{
super("JMF - Example...");
setSize(400, 300);
setLocation((Toolkit.getDefaultToolkit().getScreenSize().width -
getWidth())/2,
(Toolkit.getDefaultToolkit().getScreenSize().height -
getHeight())/2);
addWindowListener(new WindowAdapter() {
public void windowClosing(WindowEvent evt) {
System.exit(0);
}});
setContentPane(new examplePanel());
setVisible(true);
}}
//-----;
import java.awt.Component;
import java.awt.Graphics;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.io.IOException;
import java.net.MalformedURLException;
import java.net.URL;
import javax.media.ControllerEvent;
import javax.media.ControllerListener;
import javax.media.Manager;
import javax.media.NoPlayerException;
import javax.media.Player;
import javax.media.RealizeCompleteEvent;
import javax.swing.JPanel;
public class examplePanel extends JPanel implements ActionListener,
ControllerListener {
private static final long serialVersionUID = 1L;
private Component visualComponent;
private Player player;
public examplePanel() {

try
{
player = Manager.createPlayer(new URL("file:///tmp/a.mpg"));
player.addControllerListener(this);
player.start();
} catch(NoPlayerException e) {
e.printStackTrace();
}
catch(MalformedURLException e)
{ e.printStackTrace();
} catch(IOException e)
{
e.printStackTrace();
} }
}

```

```

public void paintComponent(Graphics g) {
    super.paintComponent(g);
}
public void actionPerformed(ActionEvent e) { }
public void controllerUpdate(ControllerEvent c) {
    if(player == null)
        return;
    if(c instanceof RealizeCompleteEvent) {
        if((visualComponent = player.getVisualComponent()) != null)
            add(visualComponent);
    } } }

```

Протокол RTP у складі JMF

Протокол транспортування в режимі реального часу (RTP) - це мережевий протокол для передачі звуку та відео через IP-мережі. Широко використовується RTP в системах зв'язку та розваг, що включають потокове передавання інформації, такі як телефон, додатки для відеоконференцзв'язку, включаючи WebRTC, телевізійні послуги та веб-функції "Натисни і говори".

RTP зазвичай проходить через протокол обробки даних користувача (UDP). RTP використовується разом із протоколом керування RTP (RTCP). Хоча RTP здійснює мультимедійні потоки (наприклад, аудіо та відео), RTCP використовується для моніторингу статистики передачі та якості обслуговування (QoS) і допомагає синхронізувати декілька потоків. RTP є однією з технічних основ Voice over IP, і в цьому контексті часто використовується разом із протоколом сигналізації, таким як протокол ініціації сеансу (SIP), який встановлює зв'язки по всій мережі.

RTP був розроблений Робочою групою аудіовізуального транспорту Спеціальної цільової групи Internet Engineering (IETF) і вперше опублікований у 1996 році як RFC 1889, який був заміщений RFC 3550 у 2003 році.

RTP призначений для передачі потокового носія в режимі реального часу. Протокол передбачає можливості компенсації джиттера та виявлення втрат пакетів та постачання поза замовленням, які є загальними під час передачі в IP-мережі. RTP дозволяє передавати дані до кількох адресатів через IP-мультивещання. RTP розглядається як основний стандарт для аудіо / відео

транспорту в IP-мережах і використовується з відповідним форматом та корисним завантаженням. Дизайн RTP заснований на архітектурному принципі, який називається рамкою прикладних рівнів, де функції протоколу реалізуються в додатку, а не в стек протоколу операційної системи.

Програми потокового мультимедіа в реальному часі вимагають своєчасної доставки інформації, і часто вони можуть допустити втрату пакетів для досягнення цієї мети. Наприклад, втрата пакету в аудіоприкладній програмі може призвести до втрати частки секунди аудіоданих, яку можна зробити непомітним за допомогою відповідних алгоритмів приховування помилок. Протокол керування передачею (TCP), хоча стандартизований для використання RTP, як правило, не використовується в програмах RTP, оскільки TCP надає перевагу надійності над операціями. Замість цього більшість реалізацій RTP побудовані за протоколом Datagram (UDP). Інші транспортні протоколи, спеціально розроблені для мультимедійних сеансів, є SCTP та DCCP, хоча з 2012 року вони не широко використовуються.

RTP був розроблений робочою групою Audio / Video Transport організації стандартів IETF. RTP використовується спільно з іншими протоколами, такими як H.323 і RTSP. Стандарт RTP визначає пару протоколів: RTP і RTCP. RTP використовується для передачі мультимедійних даних, а RTCP використовується для періодичного надсилання контрольної інформації та параметрів QoS.

Специфікація RTP описує два субпротоколу, RTP і RTCP. Протокол передачі даних, RTP, полегшує передачу даних у реальному часі. Інформація, надана цим протоколом, включає мітки часу (для синхронізації), порядкові номери (для втрати пакетів та виявлення переупорядкування) та формат корисної завантаженості, який вказує кодований формат даних [9]. Протокол управління, RTCP, використовується для якості обслуговування (QoS) зворотного зв'язку та синхронізації між мультимедійними потоками.

Пропускна спроможність трафіку RTCP порівняно з RTP невелика, як правило, близько 5%.

Для кожного мультимедійного потоку встановлюється сеанс RTP. Сеанс складається з IP-адреси з парою портів для RTP і RTCP. Наприклад, аудіо- та відеопотоки використовують окремі сеанси RTP, дозволяючи приймач вибірково отримувати компоненти певного потоку.

У специфікації рекомендується, щоб номери портів RTP були вибрані рівномірно, і що кожен з пов'язаних RTCP порт був наступним вищим непарним номером: 68. Однак для одного і того ж порту вибирається RTP та RTCP у програмах, які мультиплектують протоколи RTP та RTCP зазвичай використовують непривілейовані UDP-порти (1024 до 65535), але також можуть використовувати інші транспортні протоколи, зокрема SCTP та DCCP, оскільки протокол не є транспортним.

У деяких засобах наведено загальну схему використання RTP (рис. 1.1), але дуже мало опису процесу його використання та реально цей пакет вже давно не оновлювався та працює не стабільно. Більш ґрунтовний аналіз показав, що у ядрі Java підтримка цього протоколу не включена. [6]

1.4 Класи обробки мультимедіа у пакетах JavaFX

Аудіокліп ідеально підходить для відтворення коротких звуків у відповідь на дії користувача в грі або іншому мультимедійному додатку. Екземпляр AudioClip будується, передаючи конструктору простий рядок URI як єдиний параметр. Цей URI може вказувати на ресурс в Інтернеті, локальній файловій системі або в файлі jar, використовуючи, відповідно, схеми http :, file: і jar. Після побудови ви можете налаштувати кілька властивостей, які впливають на відтворення кліпу, такі як гучність, швидкість відтворення, панорамування та баланс. Виклик методу відтворення розпочне відтворення аудіокліпу. Ви можете повторно називати метод відтворення, щоб розпочати кілька повторюваних відтворень даного кліпу.

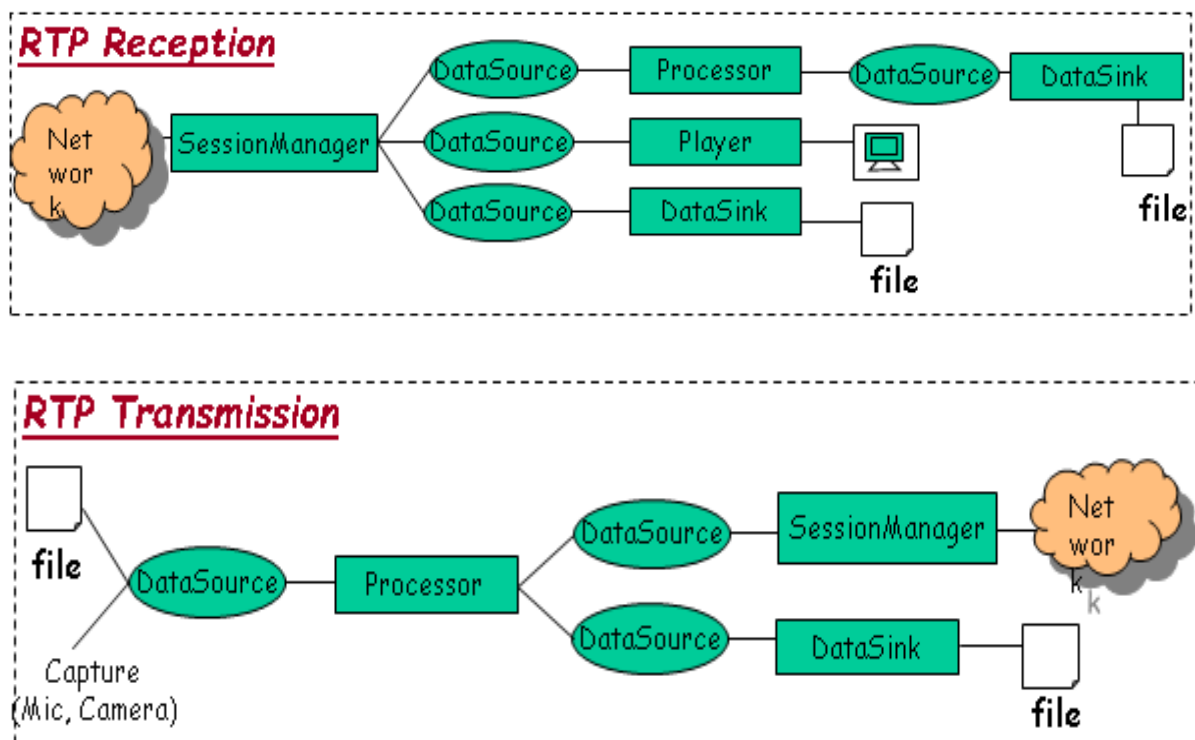


Рис. 1.1. Загальні принципи RTP

Насправді в класі `AudioClip` існує три перевантажені версії методу відтворення. Виклик методу без аргументів використовуватиме поточні властивості екземпляра `AudioClip` для відтворення. Існує також варіант методу відтворення, який приймає об'ємний аргумент, який дозволяє змінювати гучність кліпу тільки для цього відтворення. Остаточний варіант методу відтворення дозволяє вказати гучність, баланс, швидкість, панораму та пріоритет цього відтворення. Визначення цих параметрів як аргументів методів відтворення не призводить до їх постійного збереження; вони є лише однократним заміщенням даних екземпляра `AudioClip`. Існує також метод зупинки, який припиняє відтворення аудіо запису. У прикладі показано основні методи використання цього класу [7]

```
import java.net.URL;
import javafx.application.Application;
import javafx.event.ActionEvent;
import javafx.event.EventHandler;
import javafx.geometry.Insets;
import javafx.scene.Scene;
```

```

import javafx.scene.control.Button;
import javafx.scene.layout.StackPane;
import javafx.scene.media.AudioClip;
import javafx.stage.Stage;

public class Main extends Application {
    public static void main(String[] args) {
        launch(args);
    }
    @Override
    public void start(Stage primaryStage) {
        final URL resource = getClass().getResource("resources/beep.wav");
        final AudioClip clip = new AudioClip(resource.toString());

        final Button button = new Button("Bing Zzzzt!");
        button.setOnAction(new EventHandler<ActionEvent>() {

            @Override
            public void handle(ActionEvent event) {
                clip.play(1.0);
            }
        });

        final StackPane stackPane = new StackPane();
        stackPane.setPadding(new Insets(10));
        stackPane.getChildren().add(button);

        final Scene scene = new Scene(stackPane, 200, 200);
        final URL stylesheet = getClass().getResource("media.css");
        scene.getStylesheets().add(stylesheet.toString());

        primaryStage.setTitle("Basic AudioClip Example");
        primaryStage.setScene(scene);
        primaryStage.show(); } }

```

РОЗДІЛ 2. ВИДИ МЕРЕЖЕВИХ ТЕХНОЛОГІЙ ЛОКАЛЬНИХ МЕРЕЖ

2.1 Мережева технологія

Мережева технологія складається з двох компонентів: мережевих протоколів і апаратури, що забезпечує роботу цих протоколів. Протоколом в свою чергу є набір «правил», за допомогою яких комп'ютери, що знаходяться в мережі, можуть з'єднуватися один з одним, а також обмінюватися інформацією. За допомогою мережевих технологій у нас є Інтернет, є локальна зв'язок між комп'ютерами, що стоять у вас вдома. Ще мережеві технології називають базовими, але також мають ще одне красиве назву – мережеві архітектури.

Мережеві архітектури визначають кілька параметрів мережі, про які необхідно мати невелике уявлення, щоб розібратися в пристрої локальної мережі:

1) Швидкість передачі даних. Визначає, скільки інформації, яка зазвичай вимірюється в бітах, може бути передана через мережу за певний час.

2) Формат мережевих кадрів. Інформація, передана через мережу, існує у вигляді так званих «кадрів» - пакетів інформації. Мережеві кадри в різних мережевих технологіях мають різні формати переданих пакетів інформації.

3) Тип кодування сигналів. Визначає яким чином за допомогою електричних імпульсів, інформація кодується в мережі.

4) Середовище передачі. Це матеріал (зазвичай кабель), через який проходить потік інформації – тієї самої, яка в підсумку виводиться на екрани наших моніторів.

5) Топологія мережі. Це схема мережі, в якій є «ребра», що представляють собою кабелю і «вершини» - комп'ютери, до яких ці кабелю тягнуться. Поширені три основні види схем мереж: кільце, шина і зірка.

6) Метод доступу до середовища передачі даних. Використовується три методу доступу до мережевому середовищі: детермінований метод,

випадковий метод доступу і пріоритетна передача. Найбільш поширений детермінований метод, при якому за допомогою спеціального алгоритму, час використання передавальної середовища ділиться між усіма комп'ютерами знаходяться в середовищі. У разі випадкового методу доступу до мережі комп'ютери змагаються в доступі мережі. Такий метод має ряд недоліків. Одним з таких недоліків є втрата частини переданої інформації через зіткнення пакетів інформації в мережі. Пріоритетний доступ забезпечує відповідно найбільший обсяг інформації до встановленої пріоритетною станції.

Набір цих параметрів визначає мережеву технологію [8].

У локальних мережах, як правило, використовується поділюване середовище передачі даних (моноканал) і основна роль відводиться протоколами фізичного і канального рівнів, так як ці рівні найбільшою мірою відображають специфіку локальних мереж.

«Мережева технологія – це погоджений набір стандартних протоколів і що реалізують їх програмно-апаратних засобів, достатній для побудови локальної обчислювальної мережі. Мережеві технології називають базовими технологіями або мережевими архітектурами локальних мереж» [8].

Мережева технологія або архітектура визначає топологію і метод доступу до середовища передачі даних, кабельну систему або середовище передачі даних, формат мережових кадрів тип кодування сигналів, швидкість передачі в локальній мережі. У сучасних локальних обчислювальних мережах широкого поширення набули такі технології або мережеві архітектури, як: Ethernet, Token-Ring, ArcNet, FDDI.

Мережеві технології локальних мереж Ethernet

В даний час ця мережева технологія найбільш популярна в світі. Популярність забезпечується простими, надійними і недорогими технологіями. У класичній локальній мережі Ethernet застосовується стандартний коаксіальний кабель двох видів (товстий і тонкий).

«Однак все більшого поширення набула версія Ethernet, що використовує в якості середовища передачі виті пари, так як монтаж і обслуговування їх набагато простіше.»[10] У локальних мережах Ethernet застосовуються топології типу «шина» і типу «пасивна зірка», а метод доступу CSMA / CD.

«Стандарт IEEE802.3 в залежності від типу середовища передачі даних має модифікації:

- 10BASE5 (товстий коаксіальний кабель) – забезпечує швидкість передачі даних 10 Мбіт / с і довжину сегменту до 500м.
- 10BASE2 (тонкий коаксіальний кабель) – забезпечує швидкість передачі даних 10 Мбіт / с і довжину сегменту до 200м.
- 10BASE-T (неекранована кручена пара) – дозволяє створювати мережу по зоряній топології. Відстань від концентратора до кінцевого вузла до 100м. Загальна кількість вузлів не повинно перевищувати 1024.
- 10BASE-F (оптоволоконний кабель) – дозволяє створювати мережу по зоряній топології. Відстань від концентратора до кінцевого вузла до 2000м.

У розвиток мережевої технології Ethernet створені високошвидкісні варіанти: IEEE802.3u / Fast Ethernet і IEEE802.3z / Gigabit Ethernet. Основна топологія, яка використовується в локальних мережах Fast Ethernet і Gigabit Ethernet, пасивна зірка»[8].

Мережева технологія Fast Ethernet забезпечує швидкість передачі 100 Мбіт / с і має три модифікації:

- 100BASE-T4 – використовується неекранована кручена пара (зчетверена кручена пара). Відстань від концентратора до кінцевого вузла до 100м.

- 100BASE-TX – використовуються дві кручені пари (неекранована і екранована). Відстань від концентратора до кінцевого вузла до 100м.
- 100BASE-FX – використовується оптоволоконний кабель (два волокна в кабелі). Відстань від концентратора до кінцевого вузла до 2000м.

Мережева технологія локальних мереж Gigabit Ethernet – забезпечує швидкість передачі 1000 Мбіт / с. – це найбільш популярний стандарт сучасних мереж.

Існують наступні модифікації стандарту:

- 1000BASE-SX – застосовується оптоволоконний кабель з довжиною хвилі світлового сигналу 850 нм.
- 1000BASE-LX – використовується оптоволоконний кабель з довжиною хвилі світлового сигналу 1300 нм.
- 1000BASE-CX – використовується екранована кручена пара.
- 1000BASE-T – застосовується зчетверена неекранована кручена пара.

Локальні мережі Fast Ethernet і Gigabit Ethernet сумісні з локальними мережами, виконаними за технологією (стандарту) Ethernet, тому легко і просто з'єднувати сегменти Ethernet, Fast Ethernet і Gigabit Ethernet в єдину обчислювальну мережу [9].

Мережеві технології локальних мереж IEEE802.5 / Token-Ring

Мережа Token-Ring передбачає використання середовища передачі даних, яка утворюється об'єднанням всіх вузлів в кільце.

Мережа Token-Ring має зоряно-кільцеву топологію (основна кільцева і зоряна додаткова топологія). Для доступу до середовища передачі даних використовується маркерний метод (детермінований маркерний метод).

Стандарт підтримує виту пару (екрановану і неекрановану) і оптоволоконний кабель. Максимальне число вузлів на кільці – 260,

максимальна довжина кільця – 4000 м. Швидкість передачі даних до 16 Мбіт/с. В цей час цей стандарт рідко використовується. [10]

Мережеві технології локальних мереж IEEE802.4 / ArcNet

Як топології локальна мережа ArcNet використовує «шину» і «пасивну зірку». Підтримує екрановану і неекрановану виту пару і оптоволоконний кабель.

У мережі ArcNet для доступу до середовища передачі даних використовується метод передачі повноважень. Локальна мережа ArcNet – це одна з найстаріших мереж і користувалася великою популярністю. Серед основних переваг локальної мережі ArcNet можна назвати високу надійність, низьку вартість адаптерів і гнучкість.

Основним недоліків мережі є низька швидкість передачі інформації (2,5 Мбіт / с). Максимальна кількість абонентів – 255. Максимальна довжина мережі – 6000 метрів.

Мережеві технології локальних мережі FDDI (Fiber Distributed Data Interface)

FDDI- стандартизована специфікація для мережевої архітектури високошвидкісної передачі даних по оптоволоконних лініях. Швидкість передачі – 100 Мбіт / с. Ця технологія багато в чому базується на архітектурі Token-Ring і використовується детермінований маркерний доступ до середовища передачі даних.

Максимальна довжина кільця мережі – 100 км. Максимальна кількість абонентів мережі – 500. Мережа FDDI – це дуже високонадійна мережу, яка створюється на основі двох оптоволоконних кілець, що утворюють основний і резервний шляхи передачі даних між вузлами [11].

2.2 Протокол TCP

Протокол TCP (Transmission Control Protocol, Протокол контролю передачі) забезпечує наскрізну доставку даних між прикладними процесами,

запущеними на вузлах, взаємодіючих по мережі. Стандартний опис TCP міститься в RFC-793.

TCP – надійний байт-орієнтований (byte-stream) протокол із установами з'єднання. TCP знаходиться на транспортному рівні стека TCP / IP, між протоколом IP і власне програмою. Протокол IP займається пересиланням дейтаграм по мережі, що не гарантуючи доставку, цілісність, порядок прибуття інформації і готовність одержувача до прийому даних; всі ці завдання покладені на протокол TCP.

При отриманні дейтаграми, в поле Protocol якої зазначений код протоколу TCP (6), модуль IP передає дані цієї дейтаграми модулю TCP. Ці дані представляють собою TCP-сегмент, що містить TCP-заголовок і дані користувача (прикладного процесу). Модуль TCP аналізує службову інформацію заголовка, визначає, яким саме процесу призначені дані користувача, перевіряє цілісність і порядок приходу даних і підтверджує їх прийом іншій стороні. У міру отримання правильної послідовності неспотворених даних користувача вони передаються прикладному процесу.

Нижче основні функції протоколу TCP і їх реалізація розглянуті більш докладно. [12]

1. Базова передача даних.

Модуль TCP виконує передачу безперервних потоків даних між своїми клієнтами в обох напрямках. Клієнтами TCP є прикладні процеси, що викликають модуль TCP при необхідності отримати або відправити дані процесу-клієнта на іншому вузлі. Цей протокол розглядає дані клієнта як безперервний неінтерпретіруемый потік октетів. TCP поділяє цей потік на частини для пересилання на інший вузол в TCP-сегментах деякого розміру. Для відправлення або одержання сегмента модуль TCP викликає модуль IP. Негайне відправлення даних може бути затребувано процесом-клієнтом від TCP-модуля за допомогою спеціальної функції PUSH, інакше TCP сам буде

вирішувати, як накопичувати і коли відправляти дані клієнта або коли передавати клієнту отримані дані.

2. Забезпечення достовірності.

Модуль TCP забезпечує захист від пошкодження, втрати, дублювання і порушення черговості отримання даних. Для виконання цих завдань всі октети в потоці даних наскрізним чином пронумеровані в порядку зростання. Тема кожного сегмента містить число октетів даних в сегменті і порядковий номер першого октету тієї частини потоку даних, яка пересилається в даному сегменті. Наприклад, якщо в сегменті пересилаються октети з номерами від 2001 до 3000, то номер першого октету в даному сегменті дорівнює 2001, а число октетів дорівнює 1000. Номер першого байта в потоці визначається на етапі встановлення з'єднання і позначається $ISN + 1$. Наприклад, $ISN + 1 = 1$. Також для кожного сегмента обчислюється контрольна сума, що дозволяє виявити пошкодження даних. При вдалому прийомі октету даних модуль посилає відправнику підтвердження про прийом - номер вдало прийнятого октету. Якщо протягом деякого часу відправник не отримає підтвердження, вважається, що октет не дійшов або був пошкоджений, і він посилається знову. Цей механізм контролю надійності називається PAR (Positive Acknowledgment with Retransmission). Насправді підтвердження надсилається не для одного октету, а для деякого числа послідовних октетів. Нумерація октетів використовується також для впорядкування даних в порядку черговості і виявлення дублікатів (які можуть бути послані через велику затримку при передачі підтвердження або втрати підтвердження) [13].

3. Поділ каналів.

Протокол TCP забезпечує роботу одночасно декількох з'єднань. Кожен прикладний процес ідентифікується номером порту. Темою TCP-сегмента є номери портів процесу-відправника та процесу-одержувача. При отриманні сегмента модуль TCP аналізує номер порту одержувача і відправляє дані відповідному прикладному процесу. Всі поширені сервіси Інтернет мають

стандартизовані номери портів. Наприклад, номер порту сервера електронної пошти - 25, сервера FTP - 21. Список стандартних номерів портів можна знайти в файлі / etc / services (Unix). Сукупність IP-адреси і номера порту називається сокетом. Сокет унікально ідентифікує прикладний процес в Інтернет. Наприклад, сокет сервера електронної пошти на хості 194.84.124.4 позначається як 194.84.124.25; часто номер порту відділяється двокрапкою.

4. Управління з'єднаннями.

З'єднання - це сукупність інформації про стан потоку даних, що включає сокети, номери надісланих, прийнятих і підтверджених октетів, розміри вікон. Кожне з'єднання унікально ідентифікується в Інтернет парою сокетів. З'єднання характеризується для клієнта ім'ям, яке є показником на структуру TCB (Transmission Control Block), що містить інформацію про з'єднання. Відкриття з'єднання клієнтом здійснюється викликом функції OPEN, якій передається сокет, з яким потрібно встановити з'єднання. Функція повертає ім'я з'єднання. Розрізняють два типи відкриття з'єднання: активне і пасивне. При активному відкритті TCP-модуль починає процедуру встановлення з'єднання з зазначеним сокетом, при пасивному - очікує, що віддалений TCP-модуль почне процедуру встановлення з'єднання з зазначеного сокета. Вказівка 0.0.0.0:0 як сокета при пасивному відкритті означає, що очікується з'єднання з будь-якого сокета. Такий спосіб застосовується в демонів - серверах Інтернет, які чекають встановлення з'єднання від клієнта. Клієнт же застосовує процедуру активного відкриття; сокет при цьому формується з IP-адреси сервера і стандартного номера порту для даного сервісу. Закриття з'єднання клієнтом проводиться за допомогою функції CLOSE, якій передається ім'я з'єднання.

5. Управління потоком.

Для прискорення і оптимізації процесу передачі великих обсягів даних протокол TCP визначає метод управління потоком, званий методом ковзного

вікна, який дозволяє відправнику посилати черговий сегмент, не чекаючи підтвердження про отримання в пункті призначення попереднього сегмента.

Протокол TCP формує підтвердження не для кожного конкретного успішно отриманого пакета, а для всіх даних від початку посилки до деякого порядкового номера ACK SN (Acknowledge Sequence Number) виключно. В якості підтвердження успішного прийому, наприклад, перших 2000 байт, висилається ACK SN = 2001: це означає, що всі дані в байтовому потоці під номерами від $ISN + 1 = 1$ до даного ACK SN -1 (2000) успішно отримані.

Разом з посилкою відправнику ACK SN одержувач оголошує також "розмір вікна", наприклад - 6000. Це означає, що відправник може посилати дані з порядковими номерами від поточного ACK SN = 2001 до $(ACK SN + \text{розмір вікна} - 1) = 8000$, не чекаючи підтвердження з боку одержувача.

Для тимчасової зупинки посилки даних досить оголосити нульове вікно. Але навіть і в цьому випадку через певні проміжки часу будуть відправлятися сегменти з одним октетом даних. Це робиться для того, щоб відправник гарантовано дізнався про те, що одержувач знову оголосив нульове вікно, оскільки одержувач зобов'язаний підтвердити отримання "пробних" сегментів, а в цих доказах він вкаже також і поточний розмір свого вікна.

Модуль TCP може оптимізувати максимальний розмір сегмента виходячи з значень MTU на різних ділянках маршруту та інших характеристик з'єднання.

Модуль TCP може використовувати алгоритм "повільного старту", формуючи при встановленні з'єднання вікно перевантаження, розмір якого спочатку дорівнює розміру одного сегмента. Це вікно показує, скільки сегментів TCP-модуль, з його власної точки зору, може відправити без отримання підтвердження. Ковзне ж вікно, розглянуте вище, показує, який обсяг непідтверджених даних модулю дозволено відправити з точки зору віддаленого модуля, одержувача його даних.

Після приходу підтвердження від одержувача вікно перевантаження збільшується на 1 сегмент, і відправник може вислати вже два сегмента, не чекаючи підтвердження. Такий підхід дозволяє поступово збільшувати навантаження на мережу. Якщо вікно перевантаження стає більше ковзного вікна, що оголошується одержувачем, обмеження на передачу непідтверджених даних встановлює вже ковзне вікно одержувача.

У разі, якщо ніякі дані програмами не передаються, а з'єднання відкрито, модуль TCP може періодично посилати сегменти-зонди для з'ясування того, чи не відключилася чи інша сторона без попередження партнера (наприклад, в результаті обриву лінії або іншим некоректним чином). Таке зондування проводиться приблизно кожні дві години неактивності [14].

2.3 Протокол UDP

UDP (англ. User Datagram Protocol - протокол призначених для користувача датаграм) - один з ключових елементів TCP / IP, набору мережевих протоколів для Інтернету. З UDP комп'ютерні програми можуть надсилати повідомлення (в даному випадку звані датаграму) іншим хостам по IP-мережі без необхідності попереднього повідомлення для установки спеціальних каналів передачі або шляхів даних. Протокол був розроблений Девідом П. Рідом в 1980 році і офіційно визначений в RFC 768.

UDP надає ненадійний сервіс, і датаграми можуть прийти не один за одним, дублюватися або зовсім зникнути без сліду. UDP має на увазі, що перевірка помилок і виправлення або не потрібні, або повинні виконуватися в додатку. Чутливі до часу додатки часто використовують UDP, так як краще скинути пакети, ніж чекати затрималися пакети, що може виявитися неможливим в системах реального часу. При необхідності виправлення помилок на мережевому рівні інтерфейсу додаток може задіяти TCP або SCTP, розроблені для цієї мети.

Природа UDP як протоколу без збереження стану також корисна для серверів, що відповідають на невеликі запити від величезного числа клієнтів, наприклад DNS і потокові мультимедійні додатки на зразок IPTV, Voice over IP, протоколи тунелювання IP і багато онлайн-ігри.

Службові порти UDP

UDP-програми використовують датаграмні сокети для установки з'єднання між хостами. Додаток пов'язує сокет з його кінцевою точкою передачі даних, яка є комбінацією IP-адреси і порту служби. Порт - це програмна структура, яка визначається номером порту - 16-бітовим цілочисельним значенням (тобто від 0 до 65535). Порт 0 зарезервований, хоча і є допустимим значенням порту джерела в разі, якщо процес-відправник не очікує відповідати на них.

IANA розбила номери портів на три групи.

- Порти з номерами від 0 до 1023 використовуються для звичайних, добре відомих служб. В Unix-подібних операційних системах для використання таких портів необхідний дозвіл суперкористувача.
- Порти з номерами від 1024 до 49151 призначені для зареєстрованих IANA служб.
- Порти з 49152 по 65535 можуть бути використані для будь-яких цілей, оскільки офіційно не розроблені для якоїсь певної служби. Вони також використовуються як динамічні (тимчасові) порти, які запущене на хості програмне забезпечення може випадковим чином вибрати для самовизначення. По суті, вони використовуються як тимчасові порти в основному клієнтами при зв'язку з серверами.

Структура пакета UDP

«UDP - мінімальний орієнтований на обробку повідомлень протокол транспортного рівня, задокументований у RFC 768.

UDP не надає жодних гарантій доставки повідомлення для вищого протоколу і не зберігає стану відправлених повідомлень. З цієї причини UDP іноді називають Unreliable Datagram Protocol (англ. - ненадійний протокол датаграм)» [9].

UDP забезпечує багатоканальну передачу (за допомогою номерів портів) і перевірку цілісності (за допомогою контрольних сум) заголовка і істотних даних. Надійна передача в разі необхідності повинна реалізовуватися призначеним для користувача програмою.

- Порт відправника. У цьому полі вказується номер порту відправника. Передбачається, що це значення задає порт, на який при необхідності буде надсилатися відповідь. В іншому ж випадку, значення повинно бути рівним 0. Якщо хостом-джерелом є клієнт, то номер порту буде, швидше за все, динамічним. Якщо джерелом є сервер, то його порт буде одним з «добре відомих».
- Порт одержувача. Це поле є обов'язковим і містить порт одержувача. Аналогічно порту відправника, якщо хостом-одержувачем є клієнт, то номер порту динамічний, якщо одержувач - сервер, то це буде «добре відомий» порт.
- Довжина датаграми. Поле, що задає довжину всієї датаграми (заголовок і даних) в байтах. Мінімальна довжина дорівнює довжині заголовка - 8 байт. Теоретично, максимальний розмір поля - 65535 байт для UDP-датаграми (8 байт на заголовок і 65527 на дані). Фактичний межа для довжини даних при використанні IPv4 - 65507 (крім 8 байт на UDP-заголовок потрібно ще 20 на IP-заголовок).

На практиці також слід враховувати, що якщо довжина IPv4 пакета з UDP буде перевищувати MTU, то відправка такого пакета може викликати його фрагментацію, що може привести до того, що він взагалі не зможе бути доставлений, якщо проміжні маршрутизатори або кінцевий хост не

підтримуватимуть фрагментовані IP пакети. Також в RFC 791 вказується мінімальна довжина IP пакета 576 байт, яку повинні підтримувати всі учасники IPv4, і рекомендується відправляти IP пакети більшого розміру тільки в тому випадку якщо ви впевнені, що приймаюча сторона може прийняти пакети такого розміру. Отже, щоб уникнути фрагментації UDP пакетів (і можливої їх втрати), розмір даних в UDP не повинен перевищувати: $MTU - (Max\ IP\ Header\ Size) - (UDP\ Header\ Size) = 1500 - 60 - 8 =$ тисячі чотиреста тридцять-два байт. Для того щоб бути впевненим, що пакет буде прийнятий будь-яким хостом, розмір даних в UDP не повинен перевищувати: $(мінімальна\ довжина\ IP\ пакета) - (Max\ IP\ Header\ Size) - (UDP\ Header\ Size) = 576 - 60 - 8 = 508$ байт.

- Контрольна сума. Поле контрольної суми використовується для перевірки заголовка і даних на помилки. Якщо сума не згенерована передавачем, то поле заповнюється нулями. Поле не є обов'язковим для IPv4 [15].

2.4 Прикладний рівень протоколів

Прикладний рівень - абстрактний рівень (протоколу або моделі), який визначає спільні комунікаційні протоколи та методи інтерфейсу, що використовуються хостами в мережі зв'язку. Абстракція прикладного шару використовується в обох стандартних моделях комп'ютерних мереж: Internet Protocol Suite (TCP / IP) та модель OSI. Хоча обидві моделі використовують один і той самий термін для свого найвищого рівня, детальні визначення та цілі відрізняються. На наступному рисунку 2.1 показано структурне співвідношення моделі OSI та TCP

Більш детальний аналіз показав, що не існує загального підходу до виділення протоколів цього рівня. Є один важливий критерій – це протоколи, які використовуються при вирішенні конкретних завдань керування мережею, приладами, обміном інформацією і таке інше.

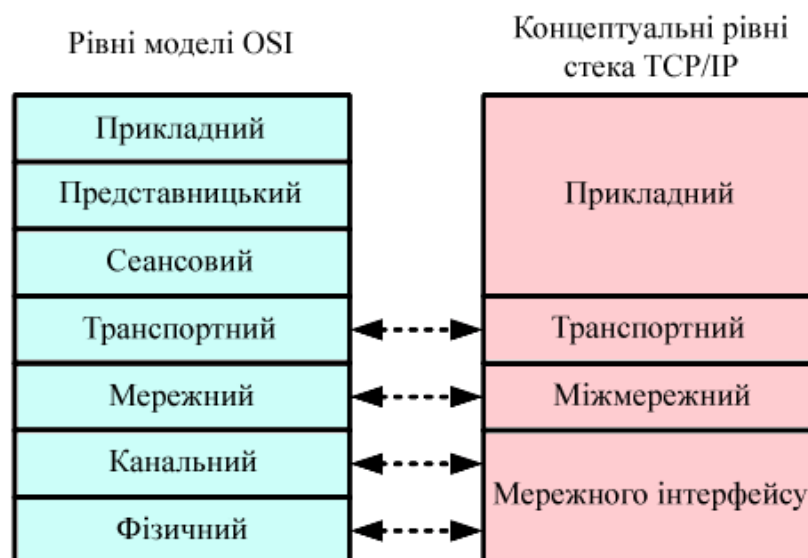


Рис. 2.1. Чотирирівнева модель стека TCP/IP і її взаємозв'язок з моделлю OSI [22]

Ці протоколи створюються прикладними програмістами при вирішенні конкретних програмних завдань. Всі вони працюють поверх транспортних протоколів (TCP, UDP, IPX). Таким чином, можна виділити декілька груп протоколів, цього рівня:

- прикладної асоціації (Application Association);
- прикладного рівня OSI (File Transfer, Access and Management, Common Management Information Protocol, Virtual Terminal Protocol, Message Handling Systems, Directory Services)
- прикладного рівня NetWare (Netware Core Protocol, NetBIOS, NetWare MHS);
- прикладного рівня TCP/IP (Telnet, SSH, FTP, HTTP, IPP, SNMP, DNS, SMTP, POP3, IMAP, IRC, NNTP, LDAP).

У TCP / IP прикладний рівень містить протоколи зв'язку та методи інтерфейсу, що використовуються в процесі обробки повідомлень через комп'ютерну мережу Internet Protocol (IP). Прикладний шар лише стандартизує зв'язок і залежить від базових протоколів транспортного рівня для встановлення каналів передачі даних від хоста до хосту та керування обміном даними в клієнт-сервері або в одноранговій мережі. Незважаючи на те, що

прикладний рівень TCP / IP не описує конкретні правила чи формати даних, які програми повинні враховувати під час спілкування, оригінальна специфікація (в RFC 1123) спирається на принцип надійності та рекомендує його для дизайну додатків [16].

У моделі OSI визначення наповнювача є більш вузьким. Модель OSI визначає шар програми як інтерфейс користувача, відповідальний за відображення отриманої інформації користувачеві. На відміну від цього, Internet Protocol Suite не стосується таких деталей. OSI також чітко виділяє додаткові функції нижче рівня застосування, але над транспортним рівнем на двох додаткових рівнях: рівні сеансу та рівні подання. OSI вказує строгий модульний розподіл функціональності на цих шарах і забезпечує реалізацію протоколів для кожного шару.

Слід відзначити, що велика кількість протоколів прикладного рівня не реалізована в багатьох мовах програмування і може бути підтримана за рахунок додаткових бібліотек та розробок. У мові програмування Java значна кількість протоколів вирішена тільки частково. Так встановлено, що:

- Протокол HTTP може використано тільки з клієнтської сторони (URLConnection) – не має HTTP сервера.
- Протокол FTP може використано тільки з клієнтської сторони (URLConnection) – не має FTP сервера.
- Протоколи пошти можуть бути використано тільки з клієнтської сторони (бібліотека java mail Oracle) – не має поштового серверу.
- Протокол SNMP може використано тільки частково з використанням InetAddress.getByName(host).isReachable(timeOut) – працює не стабільно.
- Протокол NNTP – тільки за допомогою сторонніх бібліотек Apache.org.
- Протокол RTP – може використано тільки з клієнтської сторони за допомогою бібліотек JMF.

- Протокол LDAP – може використано тільки з клієнтської сторони за допомогою бібліотек Novell (Oracle).
- Протокол DNS – може використано тільки з клієнтської сторони за допомогою класу InetAddress.

РОЗДІЛ 3. ВИКОРИСТАННЯ СПЕЦИФІЧНИХ АДРЕСІВ, МЕТОДІВ ТА МОДЕЛЕЙ

Згідно протоколу TCP/IP кожен хост мережі має свою адресу та відправляє повідомлення на іншу адресу. Тому, при вирішенні завдань спрямованих на комунікацію багатьох хостів в режимі реального часу стає задача використання специфічних адресів та методів відправки повідомлень, які підтримуються цим протоколом.

3.1 Метод передавання бродкаст

У комп'ютерних мережах, теорії телекомунікацій та інформації, бродкаст - це спосіб передачі повідомлення одразу всім одержувачам. Бродкаст може виконуватися як операція високого рівня в програмі, наприклад, трансляція інтерфейсу передачі повідомлень, або це може бути операція мережевого зв'язку низького рівня, наприклад, трансляція на Ethernet (рис.3.1).

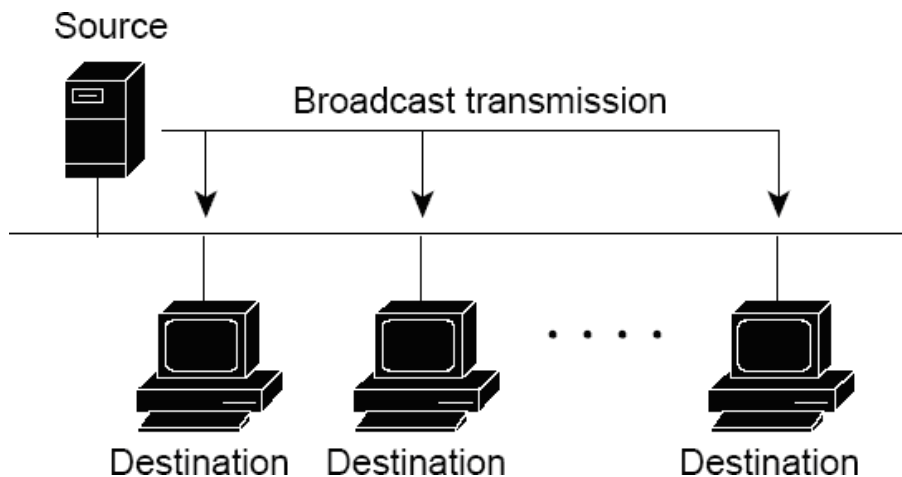


Рис. 3.1. Broadcast трафик

All-to-all-комунікація – це спосіб комп'ютерного зв'язку, в якому кожен відправник передає повідомлення всім одержувачам всередині групи. Це контрастує з методом "точка-точка", в якому кожен відправник спілкується з одним приймачем.

У комп'ютерних мережах бродкастинг означає передачу пакета, який буде отримано кожним пристроєм у мережі. На практиці обсяг передач обмежується широкомовним доменом. Бродкаст повідомлення на відміну від одноадресної адресації, в якій хост передає даними іншому одному хосту, ідентифікованому унікальною IP-адресою.

Бродкаст - це найпоширеніший спосіб зв'язку, а також найбільш інтенсивний в тому сенсі, що вимагається велика кількість повідомлень.

Бродкаст може виконуватися як весь розкид, в якому кожен відправник виконує власний розподіл, в якому повідомлення різні для кожного приймача, або всі трансляції, в яких вони однакові.

Не всі мережеві технології підтримують адрес бродкасту. Бродкаст в значній мірі обмежується технологіями локальної мережі (LAN), зокрема Ethernet і токенним кільцем, де ефект від впливу радіомовлення не настільки великий, як у широкосмуговій мережі.

Наступник Інтернет-протоколу Version 4 (IPv4), IPv6 також не реалізує метод мовлення, щоб запобігти завадженню всіх вузлів у мережі, коли лише деякі з них можуть бути зацікавлені в певній послугі. Натомість він спирається на мультикаст - концептуально подібна методологія маршрутизації "один-до-багатьох". Проте багатоадресність обмежує пул приймачів тим, хто приєднується до певної групи багатоадресних приймачів.

Як Ethernet, так і IPv4 використовують трансляційну адресу для всіх, щоб вказати пакет широкомовної передачі. Token Ring використовує спеціальне значення у полі керування IEEE 802.2 [17].

Відповідно до протоколу IPv4, широкомовна адреса є логічною адресою, на які пристрої, підключені до мережі, увімкнуті для прийому пакетів. У прикладі ми використовуємо певну IP-адресу, 255.255.255.255, яка є адресок широкомовної локальної мережі.

За визначенням, маршрутизатори, що з'єднують локальну мережу з іншими мережами, не пересилають пакети, надіслані на цю широкомовну адресу.

По-перше, демонструємо, як транслювати повідомлення. Потрібно викликати метод `setBroadcast ()` у сокеті, щоб повідомити йому про те, що пакет повинен транслюватися:

```
public class BroadcastingClient {
    private static DatagramSocket socket = null;

    public static void main((String[] args)) throws IOException {
        broadcast("Hello", InetAddress.getByName("255.255.255.255"));
    }

    public static void broadcast(
        String broadcastMessage, InetAddress address) throws IOException {
        socket = new DatagramSocket();
        socket.setBroadcast(true);
        byte[] buffer = broadcastMessage.getBytes();
        DatagramPacket packet = new DatagramPacket(buffer, buffer.length, address,
            4445);
        socket.send(packet);
        socket.close();
    } }
```

Наступний фрагмент показує, як ітерації через всі `NetworkInterfaces`, щоб знайти свою трансляційну адресу:

```
List<InetAddress> listAllBroadcastAddresses() throws SocketException {
    List<InetAddress> broadcastList = new ArrayList<>();
    Enumeration<NetworkInterface> interfaces =
        NetworkInterface.getNetworkInterfaces();

    while (interfaces.hasMoreElements()) {
        NetworkInterface networkInterface = interfaces.nextElement();

        if (networkInterface.isLoopback() || !networkInterface.isUp()) {
            continue; }

        networkInterface.getInterfaceAddresses().stream().map(a ->
            a.getBroadcast()).filter(Objects::nonNull).forEach(broadcastList::add); }
    return broadcastList;
}
```

Після того, як ми маємо список адрес мовлення, ми можемо виконати код у методі Broadcast (), який показано вище для кожної з цих адрес.

На приймаючій стороні немає спеціального коду, щоб отримувати трансляційне повідомлення. Ми можемо повторно використовувати той самий код, який отримує звичайну дейтаграму UDP. [18]

3.2 Метод передавання мультикаст

У комп'ютерних мережах multicast - це групова комунікація, де передача даних одночасно адресована групі комп'ютерів призначення. Мультикаст може бути розподілом "один до багатьох" або "багато-багато" [2]. Багатостадійне повідомлення не слід плутати з фізичним рівнем зв'язку між точками та кількома точками (рис. 3.2).

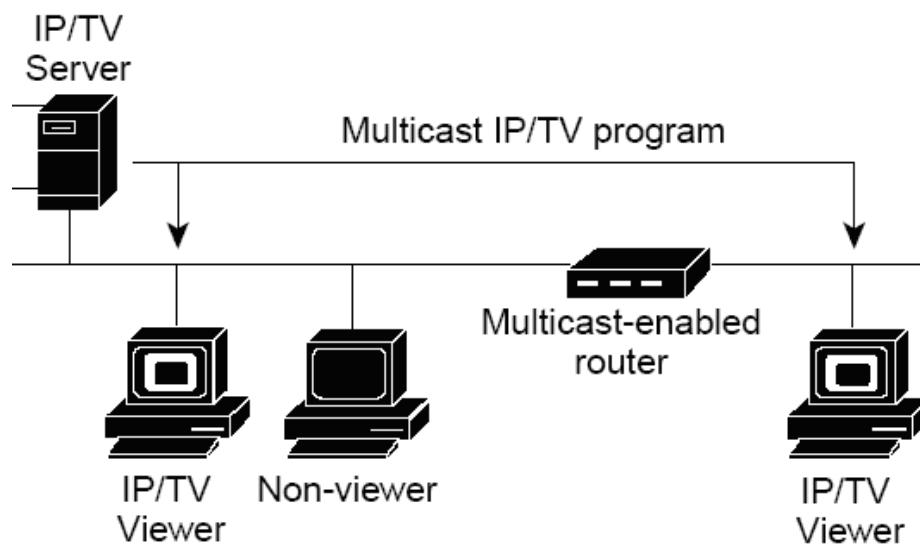


Рис. 3.2. Multicast трафік

Групова комунікація може бути або багатоадресною програмою або багатоадресною мережею, де останній дає змогу джерелу ефективно надсилати групі в одній передачі. Копії автоматично створюються в інших елементах мережі, таких як маршрутизатори, комутатори та стільникові базові станції, але лише сегментам мережі, які наразі містять членів групи. Многоадресна передача через мережу може бути реалізована на рівні каналу передачі даних, використовуючи адресну та комутаційну одиночну адресу,

таку як адреси багатоадресного Ethernet, режим асинхронного перенесення (ATM), віртуальні схеми з точністю до багатьох точок (P2MP) або багатоадресні Infiniband. Мережна асоціативна багатоадресна передача може також бути реалізована на Інтернет-рівні за допомогою IP-багатоадресної передачі. У багатоадресній передачі IP реалізація концепції багатоадресної передачі відбувається на рівні маршрутизації IP, де маршрутизатори створюють оптимальні шляхи розповсюдження для дейтаграм, відправлених на адресу призначення для багатоадресної передачі.

Мультимовлення часто застосовується в програмах Інтернет-протоколу (IP) потоковому медіа, таких як IPTV та багатоточкові відео конференції.

Мультикаст IP – це методика комунікації через IP-мережі. Цільові пункти передають протокол керування Інтернет-групуванням приєднуються та залишають повідомлення, наприклад, у випадку IP-телебачення, коли користувач змінює один телеканал на інший. Мультикаст IP для більшого числа користувачів отримувачів, не вимагаючи попереднього знання про те, хто і скільки є одержувачів. Мультикаст ефективно використовує мережеву інфраструктуру, вимагаючи від джерела надсилати пакет лише один раз, навіть якщо це потрібно доставити до великої кількості приймачів. Вузли в мережі забезпечують реплікацію пакета для охоплення кількох приймачів лише за необхідності.

Найпоширеніший протокол транспортного рівня для керування багатоадресною адресою - це протокол обробки даних користувача (UDP). За своєю природою, UDP не є надійним, повідомлення можуть бути втрачені або доставлені поза порядку. Додавши механізми детектування та перерозподілу втрат, надійна багатоадресна передача була впроваджена поверх UDP або IP різними продуктами проміжного програмного забезпечення, наприклад, ті, що реалізують протокол Real-Time-Publish-Subscribe (RTPS) у стандарті Service Distribution Service (DDS) Групи керування об'єктами (OMG), а також

спеціальними транспортними протоколами, такими як Pragmatic General Multicast (PGM). [19]

У IPv4 будь-яка адреса між 224.0.0.0 та 239.255.255.255 може бути використана як адреса багатоадресної передачі. Лише ті вузли, які підписалися на групу, отримують пакети, передані групі.

У Java, MulticastSocket використовується для отримання пакетів, відправлених на багатоадресну IP-адресу. Наступний приклад демонструє використання MulticastSocket:

```
public class MulticastReceiver extends Thread {
    protected MulticastSocket socket = null;
    protected byte[] buf = new byte[256];
    public void run() {
        socket = new MulticastSocket(4446);
        InetAddress group = InetAddress.getByName("230.0.0.0");
        socket.joinGroup(group);
        while (true) {
            DatagramPacket packet = new DatagramPacket(buf, buf.length);
            socket.receive(packet);
            String received = new String(
                packet.getData(), 0, packet.getLength());
            if ("end".equals(received)) {
                break;
            }
        }
        socket.leaveGroup(group);
        socket.close();
    }
}
```

Після скріплення MulticastSocket до порту ми називаємо метод joinGroup (), який використовує багатоадресний IP як аргумент. Це необхідно для отримання пакетів, опублікованих для цієї групи. Метод leaveGroup () може бути використаний для виходу з групи.

У наведеному нижче прикладі показано, як публікувати в багатоадресному IP [18]

```
public class MulticastPublisher {
    private DatagramSocket socket;
    private InetAddress group;
    private byte[] buf;
    public void multicast(
        String multicastMessage) throws IOException {
        socket = new DatagramSocket();
        group = InetAddress.getByName("230.0.0.0");
        buf = multicastMessage.getBytes();
    }
}
```

```
DatagramPacket packet  
= new DatagramPacket(buf, buf.length, group, 4446);  
socket.send(packet);  
socket.close();  
} }
```

3.3 Модель передавання клієнт-сервер

Модель клієнт-сервер - це розподілена структура додатків, яка розділяє завдання або навантаження між постачальниками ресурсу або сервісу, називаються серверами та запитувальниками служб, які називаються клієнтами. Часто клієнти та сервери спілкуються через комп'ютерну мережу на окремому апаратному забезпеченні, але як клієнт, так і сервер можуть перебувати в тій самій системі. Хост сервера запускає одну або декілька серверних програм, які діляться ресурсами з клієнтами. Клієнт не надає ніякого ресурсу, але запитує вміст сервера або службову функцію. Тому клієнти починають спілкування з серверами, які чекають на вхідні запити. Прикладами комп'ютерних програм, що використовують модель клієнт-сервер, є Електронна пошта, мережевий друк та Всесвітня павутина.

Характеристика клієнт-сервер описує взаємозв'язок програм, що співпрацюють у додатку. Серверний компонент надає функцію або послугу одному або багатьом клієнтам, які ініціюють запити на такі послуги. Сервери класифікуються за наданими ними службами. Наприклад, веб-сервер обслуговує веб-сторінки, а файловий сервер обслуговує комп'ютерні файли. Спільний ресурс може бути будь-яким програмним забезпеченням та електронними компонентами серверного комп'ютера, від програм і даних до процесорів і пристроїв зберігання даних. Обмін ресурсами сервера становить сервіс.

Чи комп'ютер - це клієнт, сервер чи обидва комп'ютера, залежить від характеру програми, яка потребує сервісних функцій. Наприклад, один комп'ютер може запускати веб-сервер та програмне забезпечення файлового сервера одночасно, щоб обслуговувати різні дані для клієнтів, що здійснюють

різні типи запитів. Клієнтське програмне забезпечення також може спілкуватися з серверним програмним забезпеченням на одному комп'ютері. Зв'язок між серверами, наприклад, для синхронізації даних, іноді називається міжсерверним або сервером-сервером зв'язку.

Загалом, сервіс є абстракцією ресурсів комп'ютера, і клієнту не потрібно турбуватися про те, як сервер виконує під час виконання запиту та доставки відповіді. Клієнт повинен лише зрозуміти відповідь на основі відомого протоколу додатків, тобто вмісту та форматування даних для запитуваної служби. [20]

Клієнти та сервери обмінюються повідомленнями в режимі повідомлення-запиту-відповіді. Клієнт надсилає запит, і сервер повертає відповідь. Цей обмін повідомленнями є прикладом міжпроцесного зв'язку. Щоб спілкуватися, комп'ютери повинні мати загальну мову, і вони повинні дотримуватися правил, щоб як клієнт, так і сервер знали, чого чекати. Мова та правила зв'язку визначаються в комунікаційному протоколі. Усі клієнт-серверні протоколи працюють на рівні додатків. Протокол прикладного рівня визначає основні схеми діалогу. Щоб формалізувати обмін даними ще далі, сервер може реалізувати інтерфейс прикладного програмування (API). API - це абстрактний рівень для доступу до сервісу. Обмежуючи зв'язок із певним форматом вмісту, він полегшує розбір. Абстрагуючи доступ, він полегшує крос-платформний обмін даними.

Сервер може одержувати запити від багатьох окремих клієнтів за короткий проміжок часу. Комп'ютер може виконувати обмежену кількість завдань у будь-який момент і спирається на систему планування, щоб пріоритети вхідних запитів від клієнтів для їх розміщення. Щоб запобігти зловживанням та максимально збільшити доступність, серверне програмне забезпечення може обмежити доступність для клієнтів. Напад на відмову в обслуговуванні призначений для того, щоб використати зобов'язання сервера обробляти запити, перевантажуючи їх з надмірними ставками запиту [21].

Ця модель має значні недоліки у випадку коли в мережі є багато клієнтів та серверів, які можуть передавати одночасно. У випадку використання її для системи передавання повідомлень у реальному часі необхідно використовувати (на рівні мови Java та протоколу TCP) серверні та клієнтські сокети. В загалом, цю модель використовують для систем, які мають короткі повідомлення та значний час їх обробки. Для систем обміну повідомленнями це не важливо. Ці системи навпаки вимагають значних обсягів повідомлень, що передаються та не значного часу на їх обробку.

РОЗДІЛ 4. РОЗРОБКА СИСТЕМИ ОБМІНУ ПОВІДОМЛЕННЯМИ

4.1 Розробка та дослідження архітектури системи

Призначення даної системи – це створення програмного комплексу (ПК) обміну текстовими та мультимедійними повідомленнями на основі сучасних технологій Java.

В ході написання роботи були розглянуті технології, методи і способи передачі повідомлень в мережі на основі мови програмування Java (див. розділи 2,3). Встановлено, що найпростішим, але одним з найбільш витратних способів є встановлення з'єднання кожного клієнта з іншими без використання сервера (рис. 4.1).

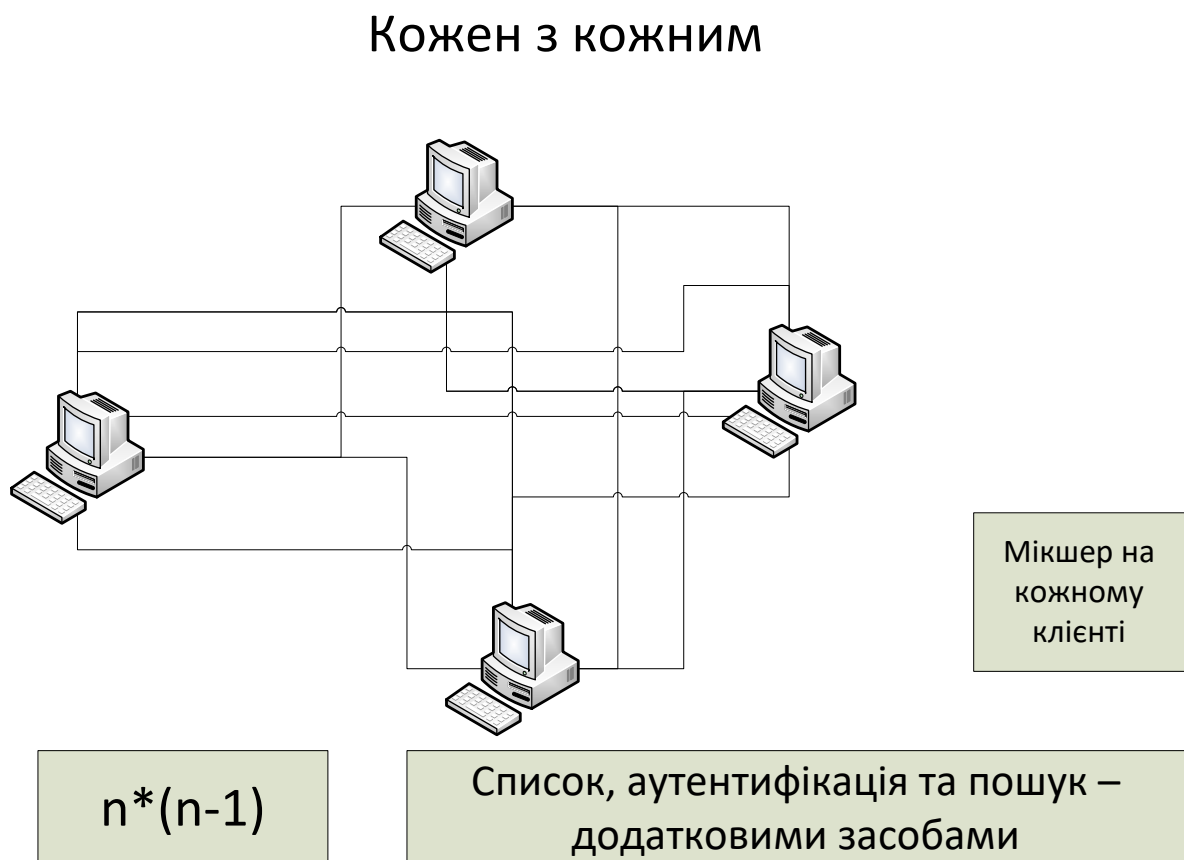


Рис. 4.1. Архітектура кожен з кожним

До істотних недоліків даної архітектури відноситься наступне:

- Мікшування має виконуватися кожним клієнтом самостійно.
- Аутентифікація і пошук інших клієнтів повинні бути розроблені іншими програмними засобами, які вкрай складно реалізувати в такій повністю децентралізованій системі.
- Кількість з'єднань дорівнює кількості клієнтів помножене на кількість клієнтів мінус один. Цей фактор призведе до істотного зниження продуктивності і перевантаження каналів зв'язку

При такій архітектурі може виникнути ситуація, коли вся мережа буде зайнята процесами передачі повідомлень, що призведе до зупинки інших мережесих ресурсів.

З іншого боку, це найпростіша в реалізації архітектура. Кожен клієнт встановлює з кожним клієнтом пряме сокетне підключення, яке працює просто і надійно. Крім того, дана архітектура має найвищу ступінь надійності. Вихід з ладу будь-якого компонента системи не призведе до руйнування системи в цілому.

Другим способом передачі повідомлень є використання сервера. Даною архітектурою користується більшість існуючих систем передачі мультимедійних повідомлень.

У цьому випадку сервер кожен клієнт встановлює одне пряме сокетне з'єднання з сервером. Сервер, в свою чергу, встановлює з'єднання між усіма клієнтами, отримує повідомлення, мікшує їх і посилає назад (рис. 4.2).

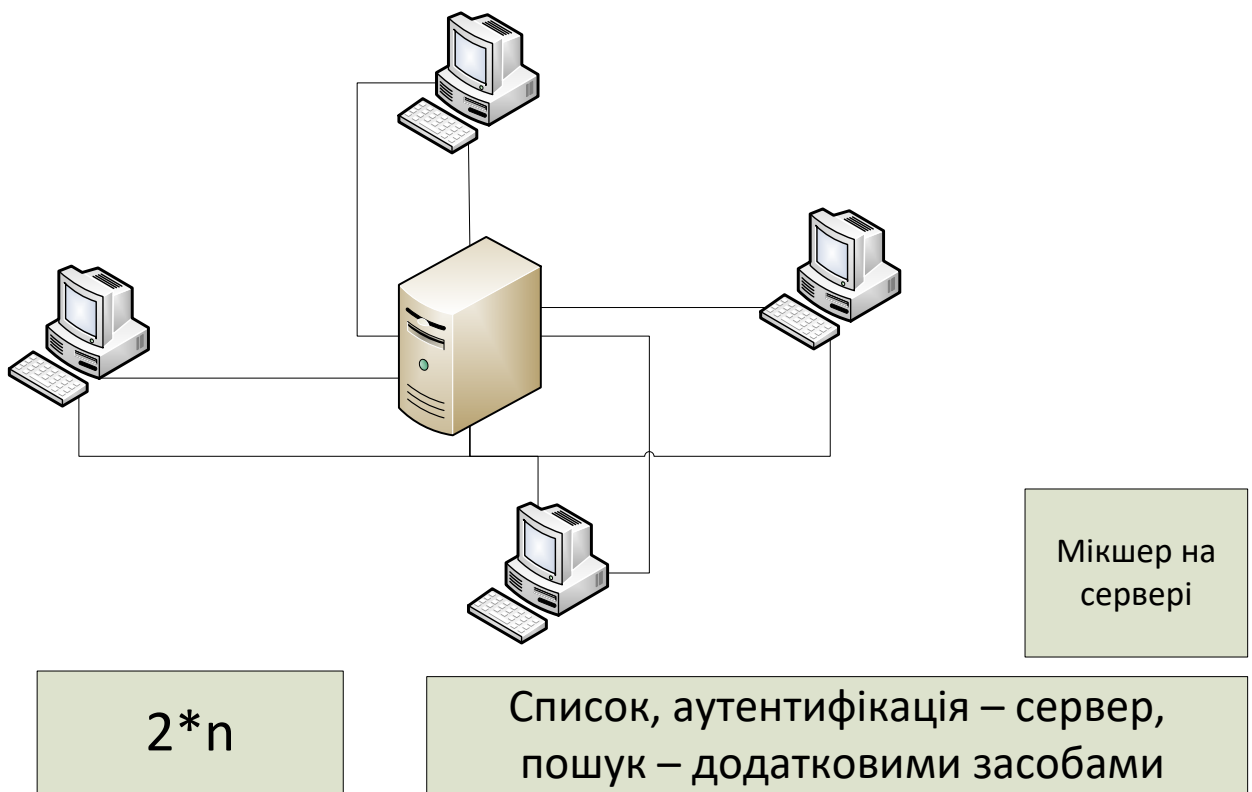


Рис. 4.2. Архітектура кожен з сервером

В даному випадку аутентифікація і розсилка службових повідомлень (списки активних користувачів) може виконуватися на сервері.

До недоліків даної архітектури можна віднести наступне:

- Необхідна розробка додаткових заходів (програмних або / та організаційних) щодо визначення місця розташування сервера в мережевій архітектурі. Клієнти повинні знати, де знаходиться сервер, знати адресу сервера.
- Низька надійність системи. Вихід з ладу сервера призведе до повної зупинки всієї системи.

Наступним можливим рішенням відправки повідомлень є використання технології бродкаст (рис 4.3).

В даному випадку сервер не використовується. Мікшування відбувається на клієнті і відправляється одне повідомлення для всіх. Працює дана технологія тільки в рамках локальної мережі.

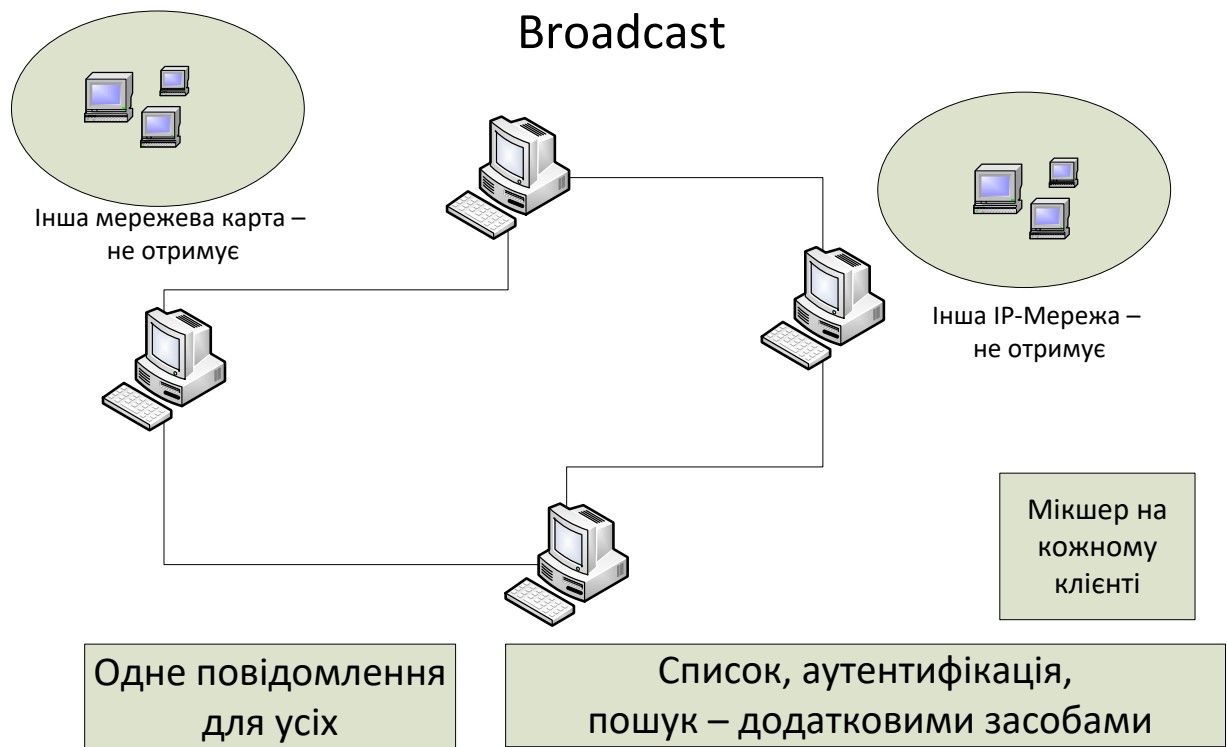


Рис. 4.3. Архітектура з використання бродкасту

Дана архітектура подібна архітектурі кожним з кожним. До істотних недоліків даної архітектури відноситься наступне:

- У даній архітектурі мікшування має виконуватися кожним клієнтом самостійно.
- Аутентифікація і пошук інших клієнтів повинні бути розроблена іншими програмними засобами, які вкрай складно реалізувати в такий повністю децентралізованою системі.
- Клієнти, розташованих в іншій підмережі не можуть отримати повідомлення.

Проте, найважливішими перевагами її є:

- Низьке завантаження каналів зв'язку
- Високий ступінь надійності.

Особливу увагу в магістерській роботі приділено дослідженню архітектури на основі технології мультикаст. Ця архітектура має ряд переваг в порівнянні з архітектурою, заснованої на технології бродкаст і за попередніми даними з літературних джерел і підручників є одним з перспективних рішень.

У даній архітектурі всі клієнти з'єднані між собою, мікшують повідомлення і по мультикаст відправляють один одному. Даний варіант працює тільки в локальній мережі (рис. 4.4).

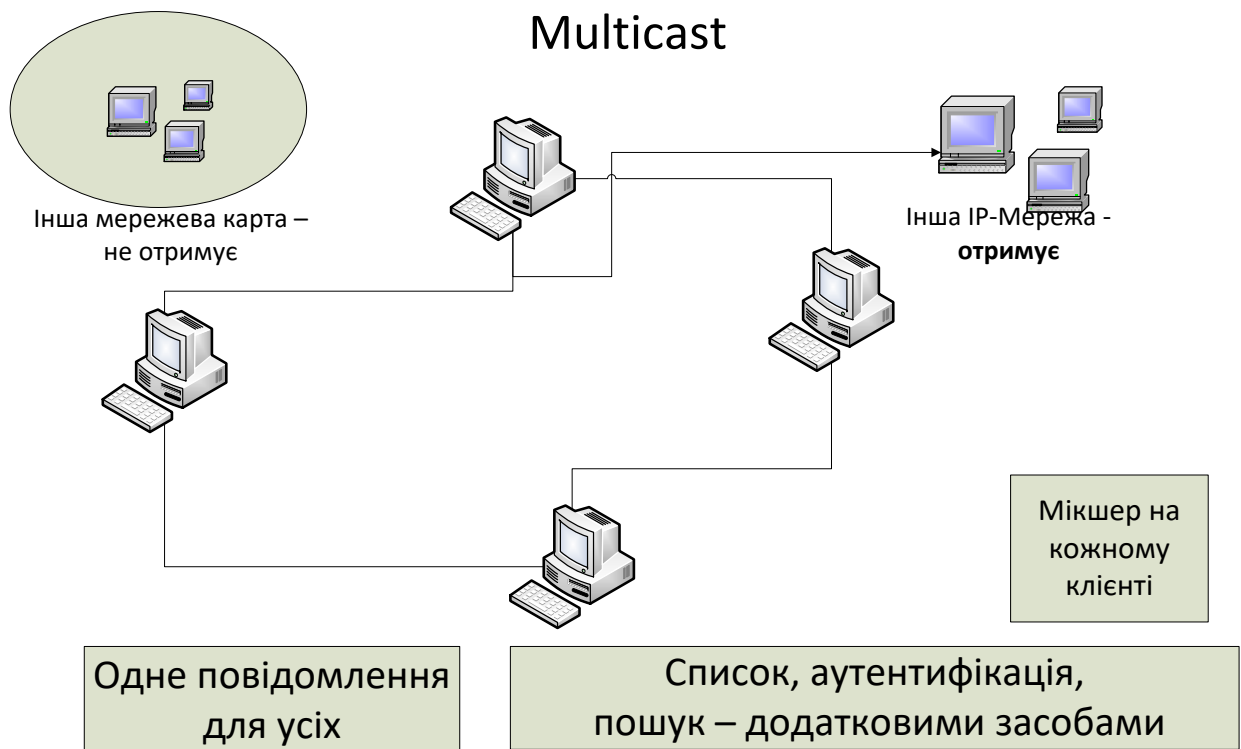


Рис. 4.4. Архітектура з використання мультикасту

Дана архітектура подібна архітектурі бродкаст і має менше недоліків. До недоліків даної архітектури відноситься наступне:

- Мікшування має виконуватися кожним клієнтом самостійно.
- Аутентифікація і пошук інших клієнтів повинні бути розроблені іншими додатковими програмними засобами, які вкрай складно реалізувати в такий повністю децентралізованою системі.

При цьому дана архітектура не навантажує канали зв'язку і має високий ступінь надійності.

В результаті аналізу накопиченого практичного досвіду для вирішення проблеми відсутності можливості аутентифікації пропонується ввести в цю архітектуру спеціальний додатковий сервер, який не задіяний в основному процесі роботи системи.

Таким чином, пропонується остаточний варіант архітектури, зображений на рисунку 4.5

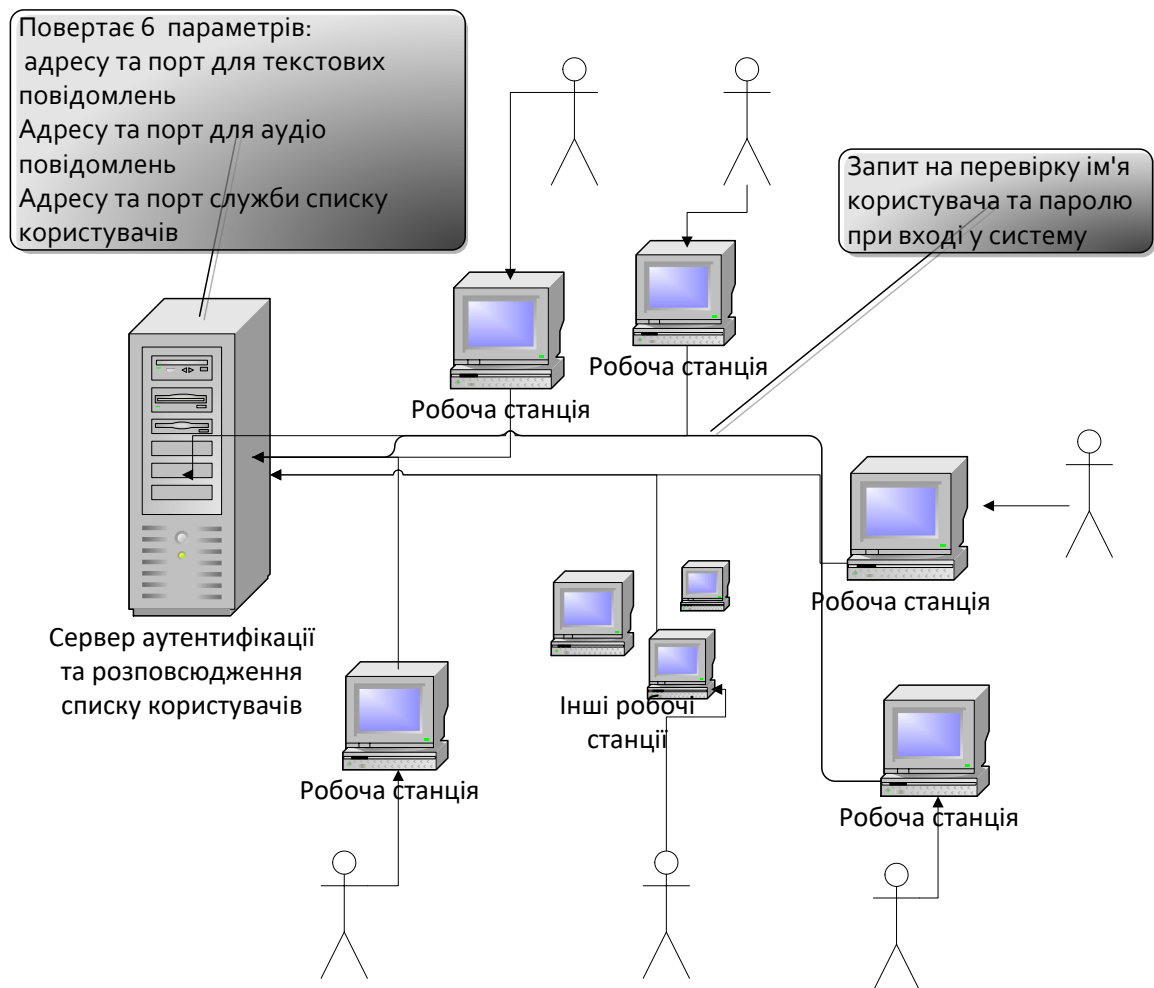


Рис. 4.5. Загальна архітектура експериментального додатку «Система обміну мультимедійними повідомленнями»

Даний ПК працює ОС Windows 7/10/11 і повинен забезпечувати незалежність від особливостей ОС. Щоб запустити його на персональному комп'ютері повинна бути встановлена Java-машина версії 1.8. ПК має зручний

інтерфейс. Також до складу ПК входить універсальна мультимедійна заставка, виконана засобами JavaFX. ПК підтримує обмін повідомлень, використовуючи технологію Multicast.

Користувачі не зможуть обмінюватися повідомленнями, якщо сервер не запущений, тому перед входом користувачі повинні переконатися, що сервер запущений. Якщо у користувача немає логіна і пароля, він повинен звернутися до адміністратора, який зобов'язаний створити нову пару логін-пароль і записати в спеціальний файл.

З іншого боку, відсутність сервера в системі призведе до того, що користувачі не зможуть увійти в систему, проте вже зареєстровані користувачі будуть продовжувати обмінюватися інформацією. Даний підхід - «сервер тільки для входу» істотно підвищує надійність системи в цілому.

В результаті встановлено, що основні функції клієнта:

- Зареєструватися на сервері і отримати параметри передачі повідомлень
- Відправляти повідомлення
- Отримувати повідомлення

Тоді, узагальнена схема основних завдань програмного комплексу, що моделюється, представлена в Додатку Б на рис. Б.2.

Безумовно, що для організації поставлених завдань необхідно використовувати розподілену структуру програмного комплексу. В результаті отримана діаграма розміщення основних компонентів програмного комплексу (див. Додаток А, рис А.2).

4.2 Сервер – основні принципи роботи

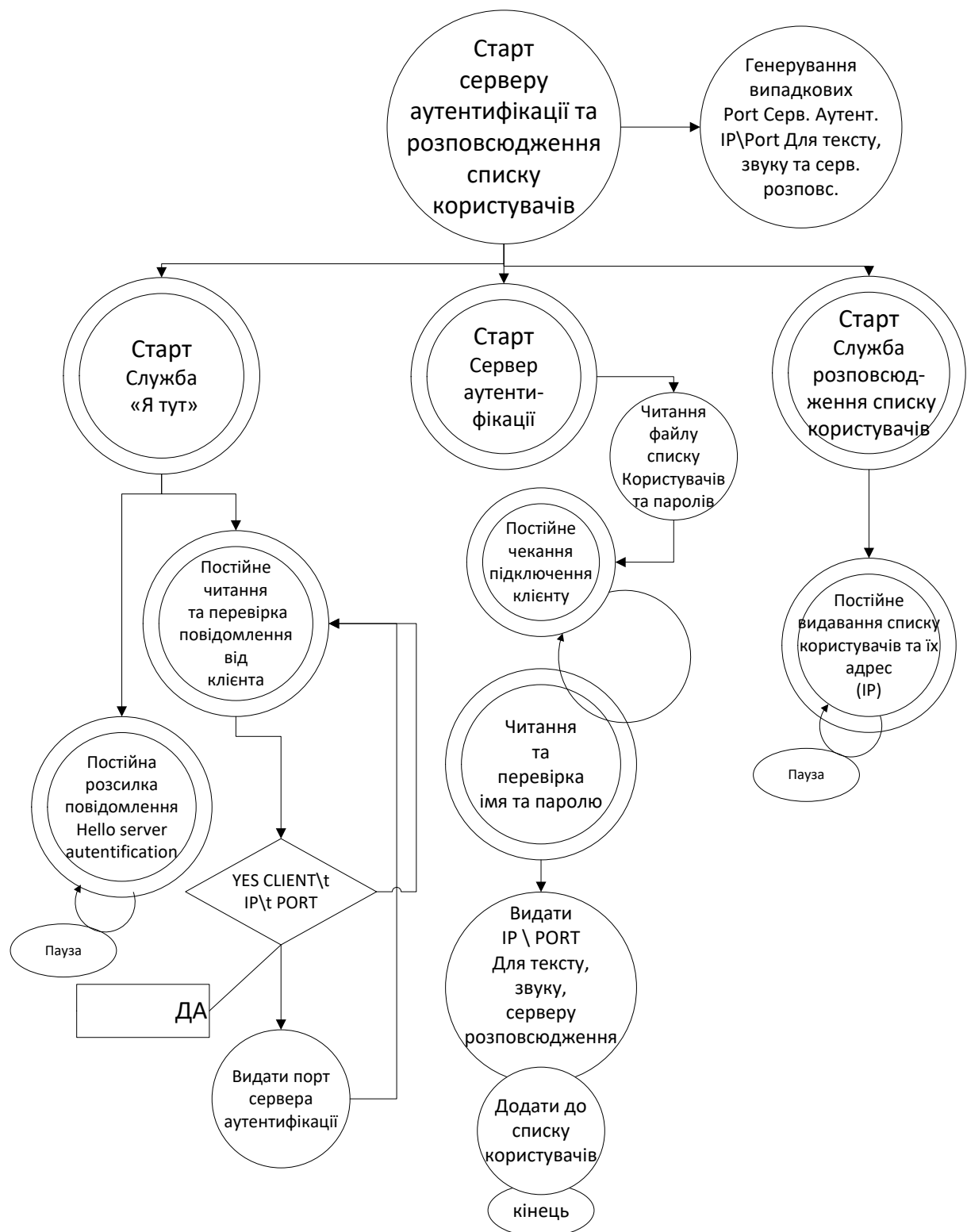
В рамках магістерської роботи зупинимося тільки на основних процесах пропонованої системи.

З аналізу даної архітектури м виявлено, що сервер в даній системі має додаткову функцію. Однак процес її реалізації має складний програмний код. Тому зупинимося лише на описі основних фонових (багатозадачних) процесах, які виконуються на сервері. Загальна схема процесів сервера представлена на рисунку 4.6. При старті сервер генерує цілий ряд випадкових чисел в певному діапазоні і запускає фонові процеси.

В результаті використання даної архітектури встановлено, що сервер може працювати на будь-якій машині користувача і повинен виконувати наступні функції (див. рис. 4.7):

- Повідомити клієнту про своє місцезнаходження в мережі (організувати службу «Я тут»)
- Провести аутентифікацію користувача - перевірити ім'я користувача і пароль (сервер аутентифікації). Видати аутентифікованим користувача параметри, необхідні для обміну повідомленнями.
- Організувати передачу користувачеві інформацію про всіх користувачів, що беруть участь в обміні повідомленнями в мережі в рамках даного сервера (сервер розсилки списку активних користувачів).

Розглянемо кожну функцію детальніше.



Перелік основних процесів серверу

Рис. 4.6. Загальна структуру процесів серверу

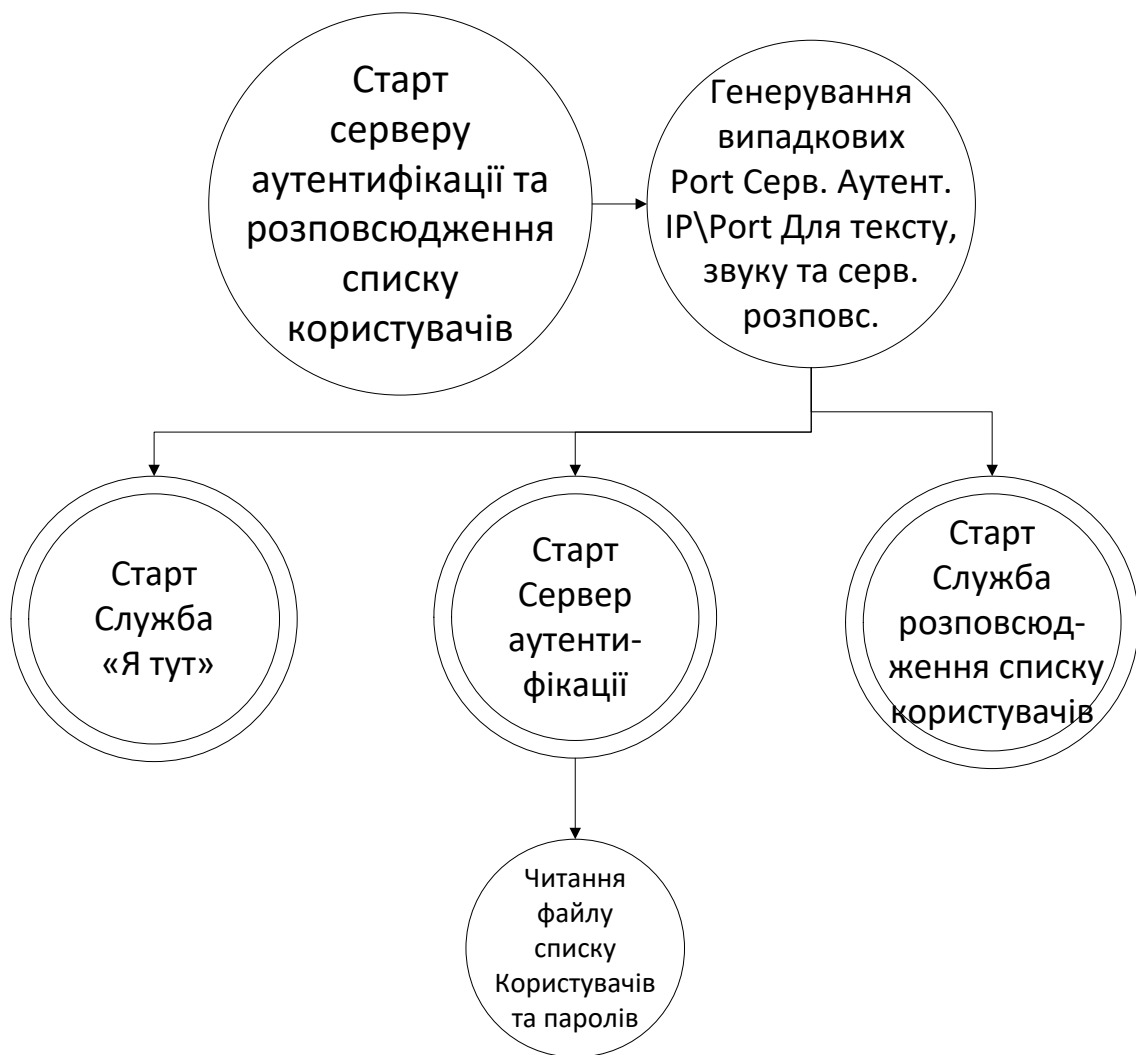


Рис. 4.7. Основні компоненти (процеси) серверу

Служба «Я тут»

Служба «Я тут» (рис. 4.8) призначена для повідомлення всіх клієнтів системи про місцезнаходження (IP адресу) самого сервера і передачі порту сервера аутентифікації. Дана служба працює в нескінченному циклі в двох фонових процесах.

Загальний алгоритм наступний:

- За технологією мультикаст постійно (в фоновому процесі), з інтервалом в 3 сек, розсилається спеціальне повідомлення UDP (Hello server autentification). Клієнти шукають дане повідомлення в мережі, і якщо знаходять, то повинні вислати у відповідь інше спеціальне повідомлення UDP (Yes client свій IP порт).

- За технологією мультикаст UDP постійно очікується повідомлення від можливих клієнтів. Якщо воно приходить, то перевіряється його структура, відповідність адрес, «вирізується» з повідомлення порт клієнта.
- Якщо все вірно, то на порт клієнта відсилається порт «сервера аутентифікації» - випадкове число.

На цьому робота даної служби закінчена

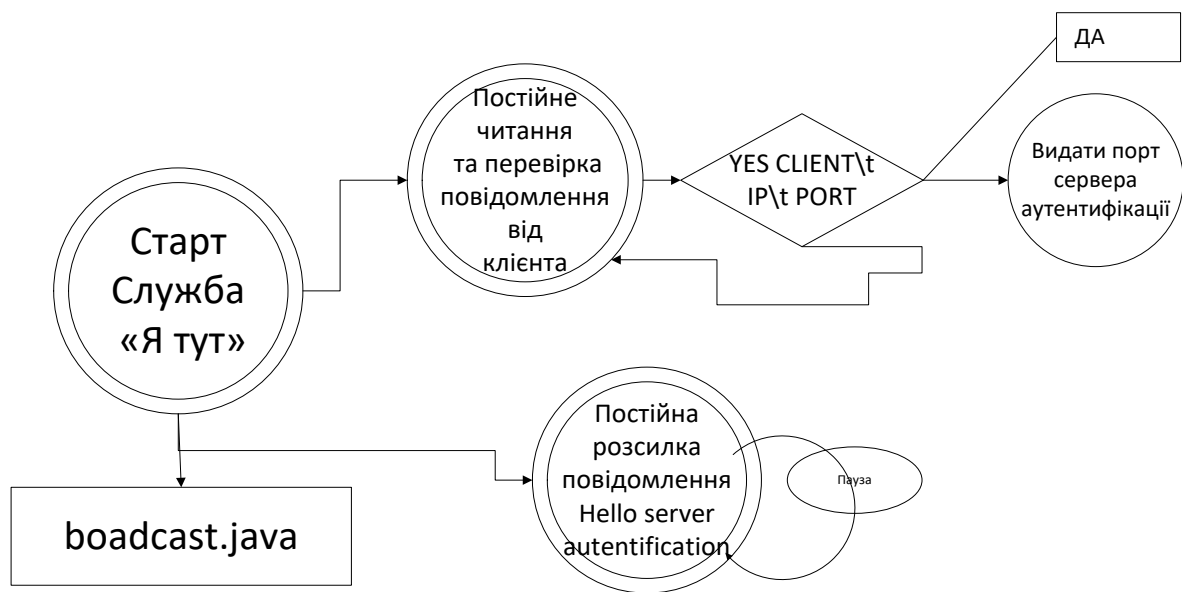


Рис. 4.8. Загальна структуру процесів серверної служби «Я ТУТ»

Сервер аутентифікації

Основне завдання сервера аутентифікації (рис. 4.9) перевірити ім'я користувача, пароль і видати параметри роботи інформаційної системи.

Дана служба працює в нескінченному циклі в декількох фонових процесах. Алгоритм його роботи наступний:

- При старті читає службовий текстовий файл зі списком користувачів і паролів.
- По моделі «клієнт - сервер» (сокет) стартує нескінченний фоновий процес очікування підключення клієнта на порту (випадкове число, яке було передано клієнтові службою «Я тут»)

- Якщо клієнт підключився, то запускається окремий фоновий процес (для кожного клієнта). У цьому процесі виконується наступне:
 - Читається ім'я користувача і пароль.
 - Перевіряється на збіг.
 - Якщо вірно, то виводиться (передається користувачеві)
 - мультікатову адресу і порт для прийому / передачі текстових повідомлень,
 - мультікатовку адресу і порт для прийому / передачі звукових повідомлень
 - параметри роботи служби розсилки користувачів - мультікатовку адресу і порт.
- Ім'я та параметри (IP адреса) даного користувача буде додано до списку активних користувачів.

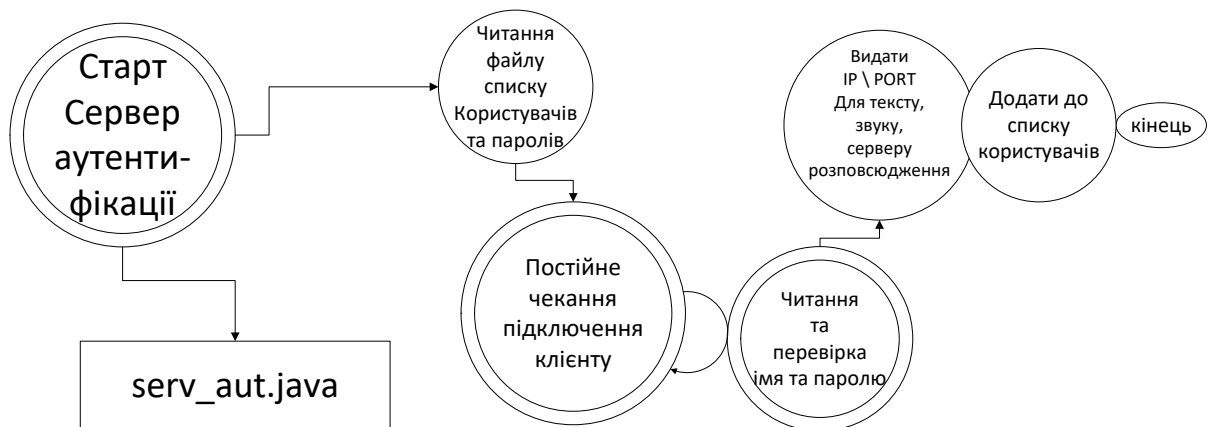


Рис. 4.9. Загальна структуру процесів серверної служби «Сервер аутентифікації»

Служба розсилки списку користувачів

Основне завдання служби розсилки списку користувачів (рис. 4.10) повідомити всіх клієнтів системи про те, які користувачі працюють в даний момент часу.

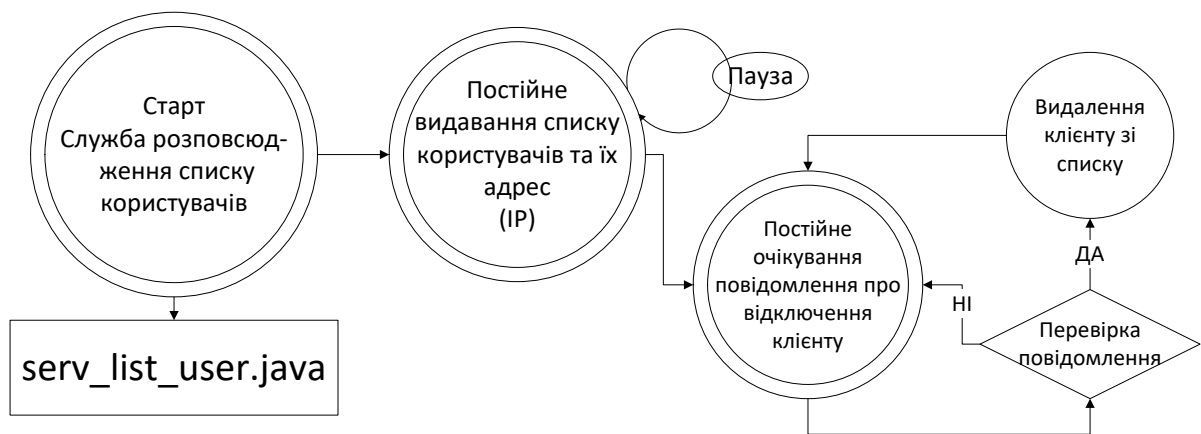


Рис. 4.10. Загальна структура процесів серверної служби
«Розповсюдження списку користувачів»

Дана служба працює в нескінченному циклі фоновому процесі. Алгоритм його роботи наступний:

- Запускається фоновий нескінченний процес, який за технологією мультикаст UDP з інтервалом в 5 сек. Виводить список активних користувачів.
- За технологією мультикаст UDP постійно очікується спеціальне повідомлення від клієнта про завершення роботи (BYE client IP адреса). Далі:
 - Перевіряється структура повідомлення,
 - перевіряється наявність даного клієнта в списку активних користувачів,
 - порівнюються IP адреси в повідомленні і в списку.
- Якщо все вірно, то віддаляється даний клієнт зі списку активних користувачів

4.3 Клієнт – основні принципи роботи

З аналізу запропонованої архітектури ПК слід, що клієнтське програмне забезпечення (див. Загальну схему процесів на рис. 4.11) має виконувати основний обсяг завдань з пересилання / прийому і відображенню повідомлень різного типу.

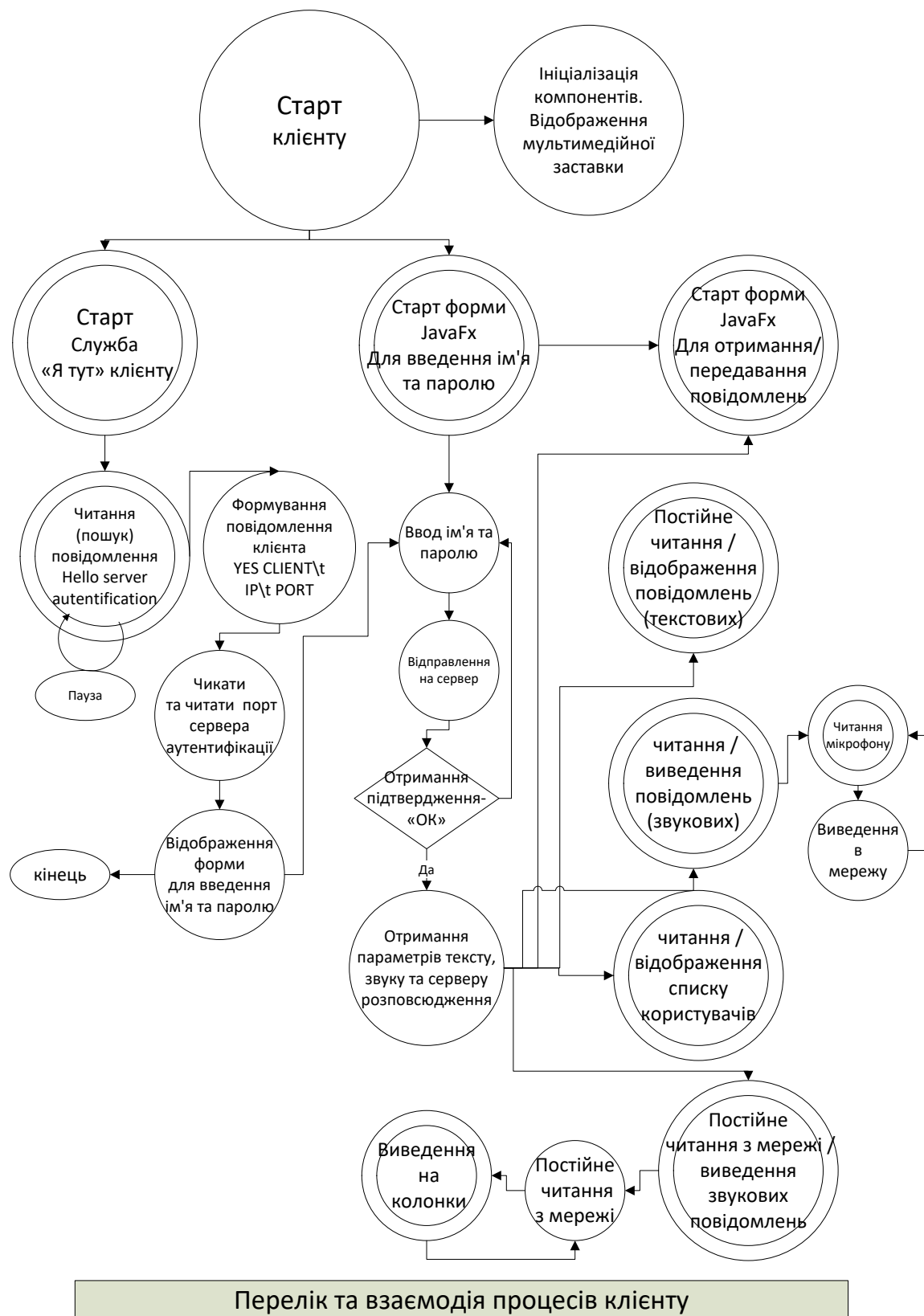


Рис. 4.11. Загальна структуру процесів клієнтського додатку

В даному випадку завдання було обмежене тільки текстовими та звуковими повідомленнями.

Для вирішення поставленого завдання використовувався інтерфейс стандартної бібліотеки JavaFx. Це дозволило знайти і застосувати нові сучасні методи візуалізації і відображення різної інформації користувачів.

В роботі клієнта можна виділити два основних етапи:

- Ініціалізація інтерфейсу, вхід в систему і отримання параметрів (мультікастовських), необхідних для організації процесу обміну повідомленнями.
- Ініціалізація і організація процесів обміну текстовими і звуковими повідомленнями.

Процес старту клієнтської програми починається з відображення рекламної заставки, в якій були реалізовані сучасні мультимедійні можливості JavaFX (див. рис. 4.12).

В результаті додаткового аналізу бібліотек Java, і на основі накопиченого досвіду вдалося розробити окремий універсальний мультимедійний програмний компонент, який може використовуватися в рекламних і дизайнерських цілях при старті будь-якого програмного забезпечення на Java.

Після відображення заставки відбувається перший етап завантажування програми.

Ініціалізація інтерфейсу

Додаток частково ініціалізує інтерфейс і працює за наступним алгоритмом:

- На основі мультікаст запускається процес пошуку сервера (шукає повідомлення Hello server autetification).

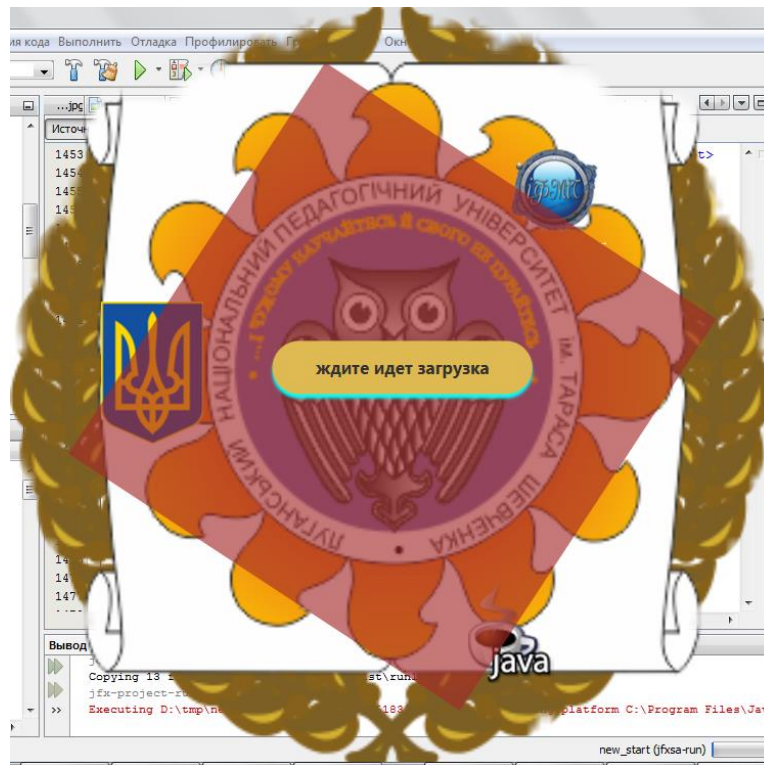


Рис. 4.12. Мультимедійна заставка клієнтського додатку.

- Якщо сервер знайдений, то визначається IP адреса сервера і формується відповідь: Yes client IP-адресу порт і приймає від сервера число - порт сервера аутентифікації.
- Показує на формі діалог введення імені користувача і пароля.
- Відсилає на сервер інформацію і, в разі успіху, приймає:
 - адресу і порт для передачі текстових повідомлень;
 - адресу і порт для передачі звукових повідомлень;
 - адресу і порт для отримання списку активних користувачів.
- Показує остаточну форму для передачі і прийому повідомлень.

На цьому перший етап роботи клієнтської програми закінчено. Надалі сервер необхідний тільки для оновлення списку активних користувачів і не бере участі в процесі обміну повідомленнями.

Другий етап роботи програми у вигляді діаграми основних фонових процесів представлений на рисунку 4.13.

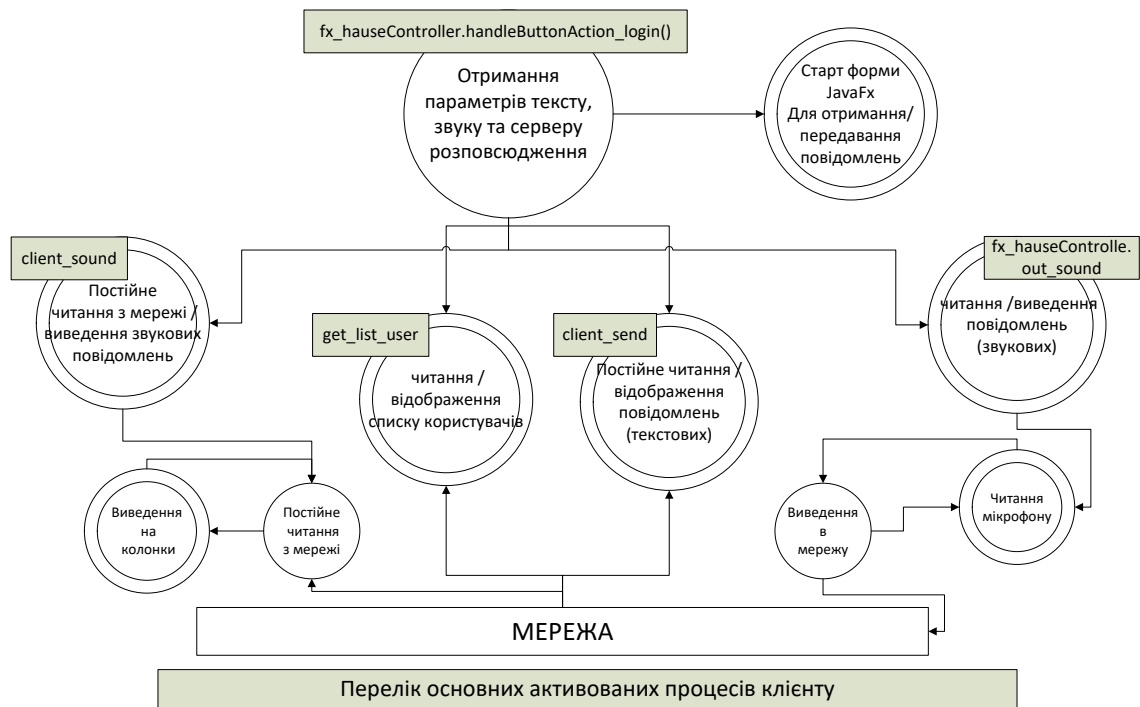


Рис. 4.13. Основні процеси клієнтського додатку

Ініціалізація і організація процесів обміну повідомленнями

Основний етап роботи програми полягає в ініціалізації чотирьох складових фонових процесів (див. рис. 4.13):

- На основі технології мультикаст запускається фоновий нескінченний процес (client_sound) читання звукових повідомлень з мережі, який синхронно працює з додатковим фоновим процесом - висновом на колонки.
- На основі технології мультикаст запускається фоновий нескінченний процес читання інформації про список активних користувачів і відображення на формі.
- На основі технології мультикаст запускається фоновий нескінченний процес читання текстових повідомлень і відображення їх на формі.
- Запускається фоновий процес читання інформації з мікрофона, який синхронно працює з додатковим процесом виведення в мережу (мультикаст).

Крім цих процесів працюють фонові процеси JavaFX, які обслуговують дії користувача при натисканні клавіатури і взаємодії з маніпулятором «миша».



Рис. 4.14. Результати дослідження процесів клієнтського додатку

З практичного боку, при проектуванні і дослідженні даного програмного комплексу, були додатково розроблені (додаток Б):

- «Інструкція користувача»
- «Інструкція адміністратора»
- «Програма і методика тестування»

У процесі налагодження даного клієнтського додатка встановлені основні особливості його роботи (див. рис. 4.14). Ці особливості істотно впливають на процес роботи всього програми і можуть бути враховані іншими програмістами при розробці архітектури подібних мультимедійних систем.

1. Виявлено втрата пакетів в мережі. Отже, необхідно вводити спеціальний мову передачі повідомлень з обов'язковою нумерацією посланих пакетів і аналізом при їх отриманні.

2. Виявлена суттєва залежність від розміру буфера прийнятих і переданих повідомлень.

а. Чим більше буфер, тим стабільніше працює (немає розривів, переривання звуку).

б. Чим менше буфер, тим простіше виконувати з'єднання звукового потоку від декількох користувачів.

5. Для стабільної роботи необхідно створювати спеціальний програмний мікшер повідомлень, а в структуру мови повідомлення вводити ім'я користувача.

6. Будь-яка неправильна (або часто повторюється) ініціалізація мікрофона і колонок призводить до постійного спотворення звукового потоку (рипіння і дзижчання). Тому уникайте ініціалізації цих пристроїв.

7. Необхідно виконувати оптимізацію фонових процесів і давати максимальний пріоритет процесам обробки звуку, а решта примусово зменшувати.

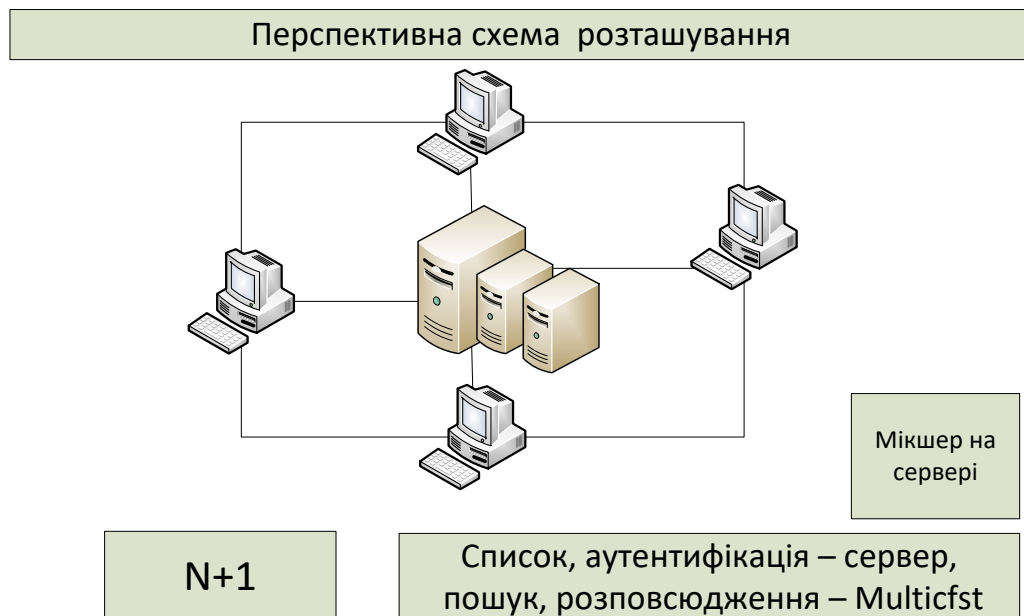


Рис. 4.15. Перспективна архітектура додатку
«Система обміну мультимедійними повідомленнями»

Аналіз отриманих недоліків даного програмного комплексу свідчить про те, що можливо необхідно застосовувати іншу архітектуру. У новій архітектурі необхідно позбутися від процесу мікшування звуку на клієнтських машинах (див. рис. 4.15).

Тому пропонується розглянути і провести експеримент архітектуру з виділеним сервером. Дана архітектура володіє більш низькою надійністю, але можливо дозволить поліпшити якість переданих звукових (відео) повідомлень.

ВИСНОВКИ

Більшість сучасних ОС, наприклад, Windows не надають можливості передачі повідомлень між користувачами в сучасному графічному інтерфейсі. Не зважаючи на те, що існує значна кількість додаткових програмних засобів, які здатні передавати мультимедійні повідомленнями, практичне їх застосування (при значній кількості користувачів) викликає цілий комплекс проблем для організації обміну конфіденційними повідомленнями в локальній мережі компанії. Тому завдання створення аналогічних програмних комплексів є своєчасним та актуальним.

В межах запропонованої магістерської роботи проведено комплексний аналіз особливостей реалізації мережевих і мультимедійних засобів Java і виконано розробку системи обміну повідомленнями.

Для досягнення поставленої мети було проведено аналіз можливості обробки звукової та відео інформації, реалізованих в ядрі Java і додаткових бібліотеках сторонніх розробників. Описано їх переваги та недоліки, особливості роботи.

Встановлено, що до ядра Java входить компонент API SOUND, який має в своєму складі значний комплекс засобів з обробки звуку. Однак повністю відсутні засоби обробки відео. З іншого боку, додаткова бібліотека JMF має компоненти з обробки відео, але ця бібліотека давно не оновлювалась і працює не стабільно. Крім того до складу ядра Java входить комплекс пакетів JavaFX. В цьому комплексі присутні засоби обробки аудіо- та відеоінформації, але вони спрямовані на організацію інтерфейсу користувача і не можуть бути пристосовані до отримання потокової інформації з мережі. На засадах цього було додатково створено рекламне, мультимедійне програмне забезпечення «Завантажувач», на яке отримано «Право на твір» [23].

В процесі роботи над магістерським завданням проведено аналіз сучасних мережевих технологій передавання інформації, особливостей та

структур різноманітних протоколів, методів та моделей передавання інформації.

Проведено аналіз підтримки цих методів і моделей у мові програмування Java на основі якого встановлено, що вирішення цих питань цілком можливо за винятком деяких аспектів та особливостей цих протоколів. Це стосується підтримки всіх засобів HTML, FTP, RTP, поштових протоколів і таке інше.

Серед цікавих методів слід відзначити метод бродкаст та мультікаст, які дозволяють значно зменшити навантаження на мережу.

На засадах проведеного аналізу у четвертому розділі було проведено дослідження можливих архітектурних рішень, які можуть використатися при створенні системи обміну мультимедійними повідомленнями. За основу було запропоновано задіяти архітектуру мультікаст з додатковим сервером аутентифікації.

В процесі реалізації цієї архітектури було сплановано та розроблено розподілену програмну систему, проведено аналіз складових фонових процесів необхідних для вирішення питань роботи серверного та клієнтського програмного забезпечення. Розроблено всі компоненти запропонованої системи. Основним критерієм є можливість стабільної роботи при одночасній участі багатьох користувачів (конференція у реальному часі)

В процесі тестування та налагодження цього програмного комплексу виявлені особливості в підходах до розробки аналогічних систем:

1. Значна залежність від розмірів буферів приймання та передавання.
2. Необхідність створення спеціальної структури повідомлення, до складу якого необхідно включати номер повідомлення та ім'я користувача.
3. В край обережне ставлення до процесів ініціалізації мікрофону та колонок.

4. Необхідність додаткової оптимізації усіх фонових процесів з урахуванням конкретних особливостей програмної розробки.
5. Необхідність розробки додаткового програмного забезпечення для мікширування звукових повідомлень під час відеоконференцій.

В загалі, на засадах всіх знайдених недоліків пропонується розглянути та дослідити модель з виділеним сервером, який працює по технології мультикаст.

СПИСОК ЛІТЕРАТУРНИХ ДЖЕРЕЛ

1. Oracle Helper Center [Електроний ресурс]. – Режим доступу: https://docs.oracle.com/javase/8/docs/technotes/guides/sound/programmer_guide/chapter1.html - Chapter 1: Introduction to the Java Sound API
2. Oracle Helper Center [Електроний ресурс]. – Режим доступу: https://docs.oracle.com/javase/8/docs/technotes/guides/sound/programmer_guide/chapter2.html - Chapter 2: Overview of the Sampled Package
3. Oracle Helper Center [Електроний ресурс]. – Режим доступу: https://docs.oracle.com/javase/8/docs/technotes/guides/sound/programmer_guide/chapter3.html - Chapter 3: Accessing Audio System Resources
4. Wikipedia - the free encyclopedia [Електроний ресурс]. – Режим доступу: https://en.wikipedia.org/wiki/Java_Media_Framework - Java Media Framework
5. TechRepublic [Електроний ресурс]. – Режим доступу: <https://www.techrepublic.com/blog/software-engineer/process-multimedia-with-the-java-media-framework-api/> - Process multimedia with the Java Media Framework API
6. Wikipedia - the free encyclopedia [Електроний ресурс]. – Режим доступу: https://en.wikipedia.org/wiki/Real-time_Transport_Protocol - Real-time Transport Protocol
7. .What-when-how [Електроний ресурс]. - Режим доступу: <http://what-when-how.com/javafx-2/working-with-audio-clips-using-the-media-classes-javafx-2/> - Working with Audio Clips (Using the Media Classes) (JavaFX 2)
8. Вибір мережевої технології [Електроний ресурс]. – Режим доступу: <https://studfile.net/preview/9906498/page:3/>
9. ПРОТОКОЛИ TCP / IP I UDP. МЕРЕЖІ TCP / IP [Електроний ресурс]. – Режим доступу: https://stud.com.ua/84321/ekonomika/protokoli_merezhi

10. Мережа та мережева технологія. Мережні інформаційні технології [Електроний ресурс]. – Режим доступу: <http://hi-news.pp.ua/tehnika-tehnologyi/2369-merezha-ta-merezheva-tehnologya-merezhn-nformacyn-tehnologyi.html>
11. Windows, Linux, macOS: порівняння та особливості [Електроний ресурс]. – Режим доступу: <https://www.globallogic.com/ua/insights/blogs/basics-of-operating-systems/>
12. Мережні технології [Електроний ресурс]. – Режим доступу: https://uk.wikipedia.org/wiki/%D0%9C%D0%B5%D1%80%D0%B5%D0%B6%D0%BD%D1%96_%D1%82%D0%B5%D1%85%D0%BD%D0%BE%D0%BB%D0%BE%D0%B3%D1%96%D1%97
13. Unicast vs. Multicast vs. Broadcast: What's the Difference? [Електроний ресурс]. – Режим доступу <https://castr.com/blog/unicast-vs-multicast-vs-broadcast/nologyi.html>
14. Difference between Unicast, Broadcast and Multicast in Computer Network [Електроний ресурс]. – Режим доступу: <https://www.geeksforgeeks.org/difference-between-unicast-broadcast-and-multicast-in-computer-network/>
15. Wikipedia - the free encyclopedia [Електроний ресурс]. – Режим доступу: <https://ru.wikipedia.org/wiki/UDP> - UDP
16. Wikipedia - the free encyclopedia [Електроний ресурс]. – Режим доступу: https://en.wikipedia.org/wiki/Application_layer - Application layer
17. Wikipedia - the free encyclopedia [Електроний ресурс]. – Режим доступу: [https://en.wikipedia.org/wiki/Broadcasting_\(networking\)](https://en.wikipedia.org/wiki/Broadcasting_(networking)) - Broadcasting (networking)
18. Baeldung [Електроний ресурс]. – Режим доступу: <http://www.baeldung.com/java-broadcast-multicast> – Broadcasting and Multicasting in Java

19. Wikipedia - the free encyclopedia [Електроний ресурс]. – Режим доступу: <https://en.wikipedia.org/wiki/Multicast> - Multicast
20. Wikipedia - the free encyclopedia [Електроний ресурс]. – Режим доступу: https://en.wikipedia.org/wiki/Client%E2%80%93server_model - Client–server model
21. Шеховцов В. А. Операційні системи / В. А. Шеховцов. – К.: Видавнича група ВНУ, 2005. – 576 с.
22. Стек протоколів TCP/IP [Електроний ресурс]. – Режим доступу: <http://www.znanius.com/3608.html>

ДОДАТКИ

Додаток А

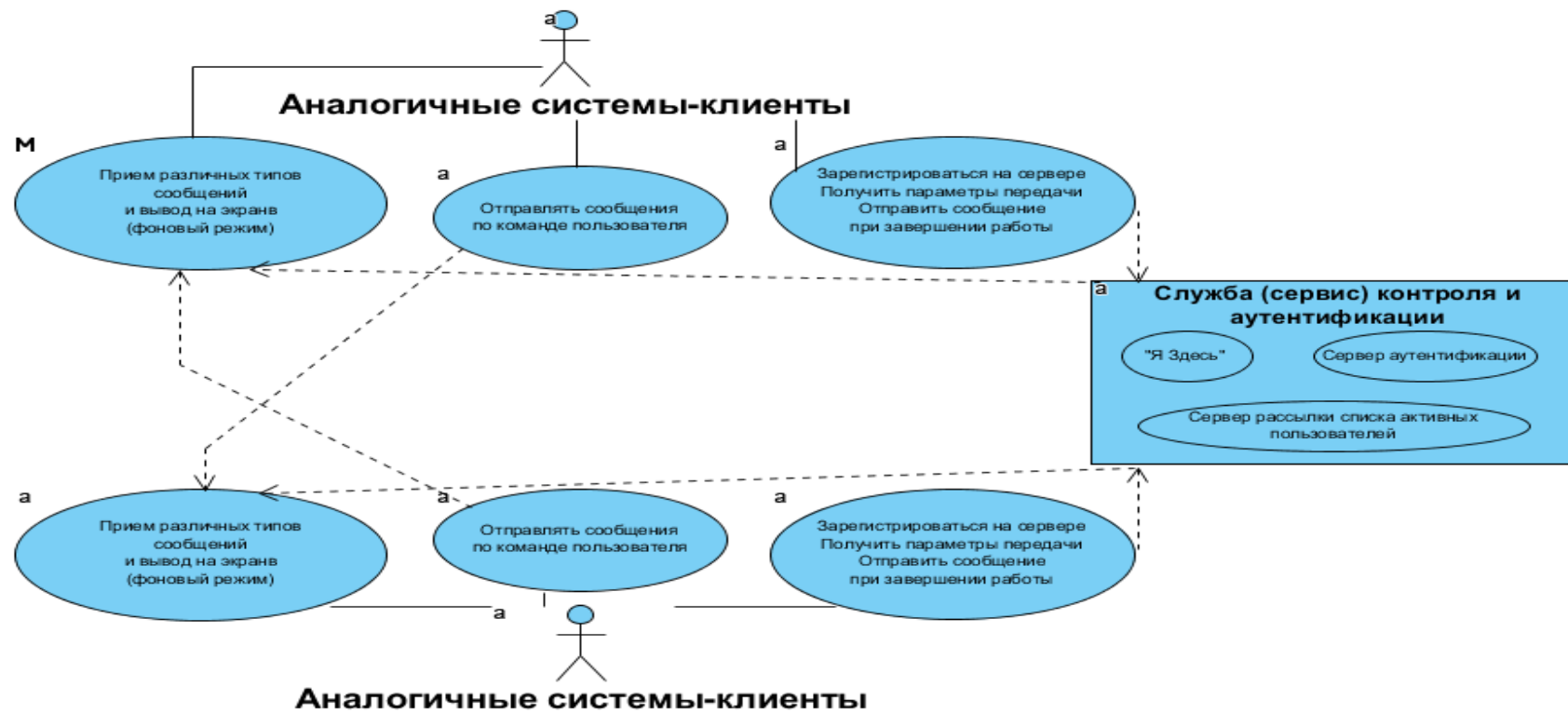


Рис А.1. Обобщенная укрупненная схема взаимодействия.

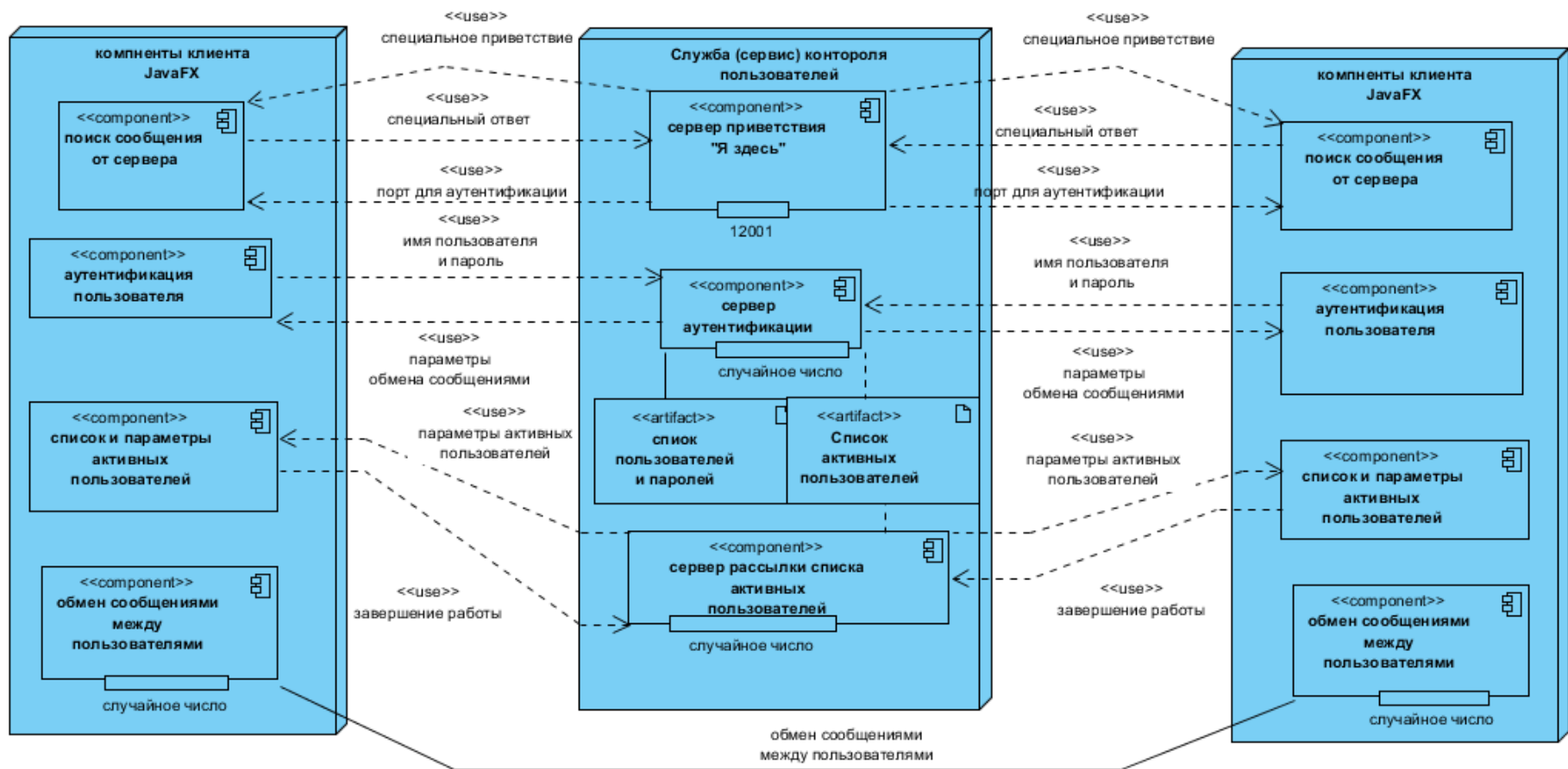


Рис.А.2 Размещение компонентов

Додаток Б

Міністерство освіти і науки України
Державний заклад «Луганський національний університет
імені Тараса Шевченка»

КЕРІВНИЦТВО АДМІНІСТРАТОРА на виконання програмної розробки (ПР) : " Аналіз технології мультикаст з використанням JAVAFX "

Полтава 2025

Содержание

| | |
|---|----|
| 1. Структура программного комплекса | 86 |
| 2. Уставка программного коиплекса..... | 86 |
| 3. Работа с сервером | 87 |
| 4. Работа с клиентом | 88 |
| 5. Файл с логином и паролем..... | 89 |

1. СТРУКТУРА ПРОГРАММНОГО КОМПЛЕКСА

Программный комплекс поставляется в виде 4-х файлов

- new_mogil.jar – классы программного комплекса в бинарном виде;
- client.bat файл для запуска комплекса в режиме клиента;
- server.bat файл для запуска комплекса в режиме сервера;
- user_paswd.txt текстовый файл со списком имен пользователей и паролей.

Дополнительно может поставляться каталог javadoc – описание классов и компонентов программного комплекса в виде HTML файлов.

2. УСТАНОВКА ПРОГРАММНОГО КОМПЛЕКСА

Программный комплекс не имеет специального инсталлятора и размещается на компьютерах путем обычного копирования файлов.

Однако для корректной работы и размещения комплекса по рабочим станциям необходимо выполнить следующее:

1. Включить на всех роутерах поддержку multicast. В ряде случаев она заблокирована поставщиком.
2. Выбрать компьютер – сервер, на котором программный комплекс будет работать в режиме сервера. В качестве такого компьютера может выступать любая машина с ограниченным доступом (например, компьютер администратора, выделенный сервер), постоянно работающая в период времени использования программного комплекса.
3. Установить на всех компьютерах виртуальную Java-машину версии 1.8 или более позднюю.
4. На компьютере-сервере скопировать в отдельный каталог все файлы программного комплекса.

5. На компьютерах-клиентах скопируйте в отдельный каталог ТОЛЬКО ДВА ФАЙЛА: new_mogil.jar и client.bat.
6. Создайте на рабочем столе ярлыки для запуска client.bat.
7. На компьютере-сервере создайте ярлык для запуска server.bat и разместите его в автозагрузке операционной системе.

Примечание. Файл user_paswd.txt содержит списки имен пользователей и паролей, поэтому ограничьте к нему доступ обычных пользователей.

Важно.

1. При возникновении в сети второго серверного процесса возникнут конфликты в программном комплексе.
2. Программный комплекс случайным образом использует адреса в диапазоне Multicast 224.0.0.1- 239.255.255.255.
3. Для работы службы «Я ЗДЕСЬ», которая входит в состав программного комплекса используется адрес 239.1.2.3 и порты 12000 и 12001 – ПРОВЕРЬТЕ, чтобы не было конфликтов с существующим программным обеспечением. Служба «РАССЫЛКИ СПИСКА АКТИВНЫХ ПОЛЬЗОВАТЕЛЕЙ» выбирает случайный адрес в диапазоне Multicast и порт 14000-14999.
4. Сервер аутентификации случайным образом использует порт в диапазоне 13000-13999.

3. РАБОТА С СЕРВЕРОМ

Для запуска сервера необходимо обеспечить запуск файла server.bat (можно через ярлык в автозагрузке), кликнув два раза на нем. Появится окно консоли. Когда в окно консоли выведутся сообщения (рис Б.1) можно начинать работу с клиентом.

```

START NETstart
START reactivex_font
START send_fon
START NETstart
START reactivex_font
START send_fon_for list

```

Рис. Б.1. Запуск сервера.

4. РАБОТА С КЛИЕНТОМ

Для запуска клиента необходимо запустить файл client.bat, щелкнув на нем два раза. Запуститься универсальная мультимедийная заставка (рис Б.2).

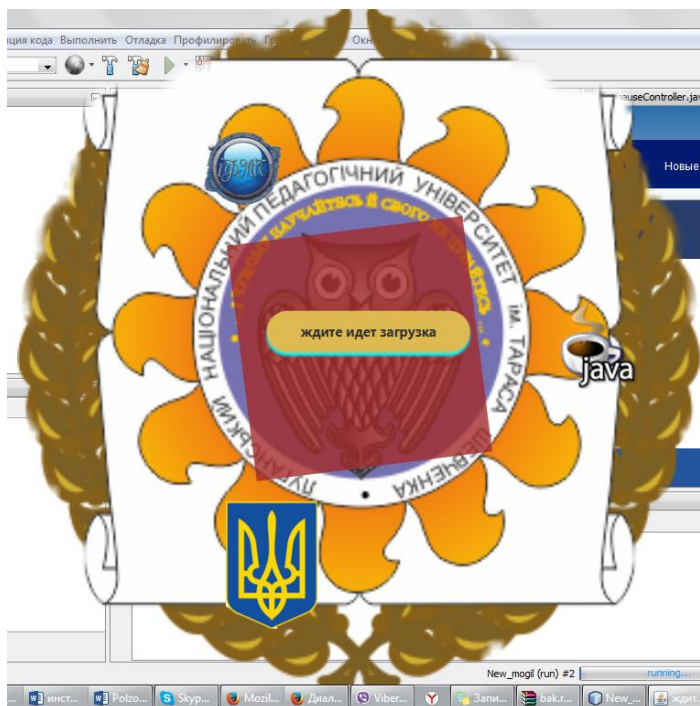


Рис. Б.2. Универсальная мультимедийная заставка перед запуском клиента

Более детальную информацию по использованию см. документ ІТС.КІ4.0516-04-КК «КЕРІВНИЦТВО КОРИСТУВАЧА».

5. ФАЙЛ С ЛОГИНОМ И ПАРОЛЕМ

Предварительно администратору нужно выдать пользователям логины и пароли и соответственно отредактировать данный файл.

Если будет не хватать логинов и паролей, но необходимо добавить их, изменив файл `user_paswd.txt`. В первой строке пишется логин, во второй пароль (рис Б.3).

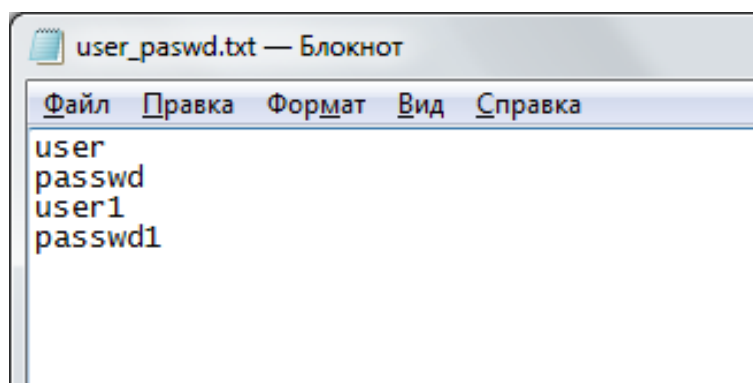


Рис. Б.3. Пример окна с логинами и паролями

Міністерство освіти і науки України
Державний заклад «Луганський національний університет
імені Тараса Шевченка»

Програма та методика тестування
на виконання програмної розробки (ПР):
" Аналіз технології мультикаст з використанням JAVAFX "

Полтава 2025

Содержание

| | |
|---|----|
| 1. Объект испытаний | 92 |
| 2. Цель тестирования | 92 |
| 3. Методы тестирования..... | 92 |
| 3.1. Тестирование заставки..... | 92 |
| 3.2. Тестирование входа в систему | 93 |
| 3.3. Тестирование обмена сообщениями..... | 94 |
| 3.4. Выход из программы..... | 97 |
| 3.5. Тестирование сервера..... | 99 |
| 3.6. Запуск клиента без сервера..... | 99 |

1. ОБЪЕКТ ИСПЫТАНИЙ

Разрабатываемый программный комплекс должен передавать сообщения, с использованием технологии Multicast.

Для измерения качества разрабатываемого программного обеспечения выполнено тестирование приложения.

Программный комплекс должен:

- Давать возможность отправлять и получать сообщений, отображая их в окне чата.
- Добавлять и удалять участников.
- использовать логины и пароли, сохраненные в отдельном файле.
- Отображать список активных пользователей.
- Давать возможность сделать переадресацию.
- Запускать универсальную мультимедийную заставку, перед входом в чат.
- Запрашивать логин и пароль для входа в чат.

2. ЦЕЛЬ ТЕСТИРОВАНИЯ

Целью тестирования является процесс технического исследования о качестве продукта относительно контекста, в котором он должен использоваться. К этому процессу входит выполнение программы с целью обнаружения ошибок.

3. МЕТОДЫ ТЕСТИРОВАНИЯ

3.1. Тестирование заставки

Корректная работа заставки:

1. Если пользователь ничего не делает заставка завершает свою работу через 20 секунд сама и начинается поиск сервера.

2. Пользователь нажимает на кнопку, заставка завершает свою работу и идет поиск сервера.

Запущена заставка и ничего не сделано. Через двадцать секунд после начала работы, заставка закрывается и начинается поиск сервера. Отработано корректно.

Запущена заставка еще раз и нажата кнопка. Заставка закрылась и начался поиск сервера. Отработано корректно.

Вывод: в результате испытаний первая часть клиентской части отрабатывает верно.

3.2. Тестирование входа в систему

1. Введено правильно имя пользователя и пароль. Отработано корректно. Окно изменилось – появилось сообщение:

«вы вошли в с систему. Ваше имя -USER»(рис Б.4).

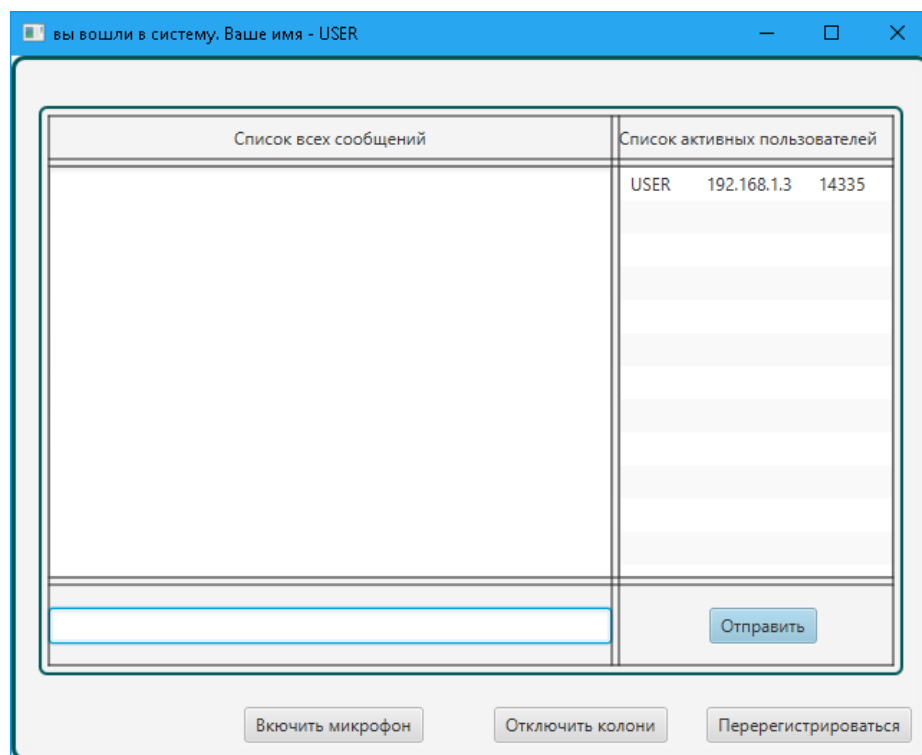


Рис. Б.4. Результат ввода правильных данных

2. Введено правильно имя пользователя, но неправильно пароль. Выдало ошибку «Системная ошибка - нет пользователя с данным паролем». В систему не запущено (рис Б.5).

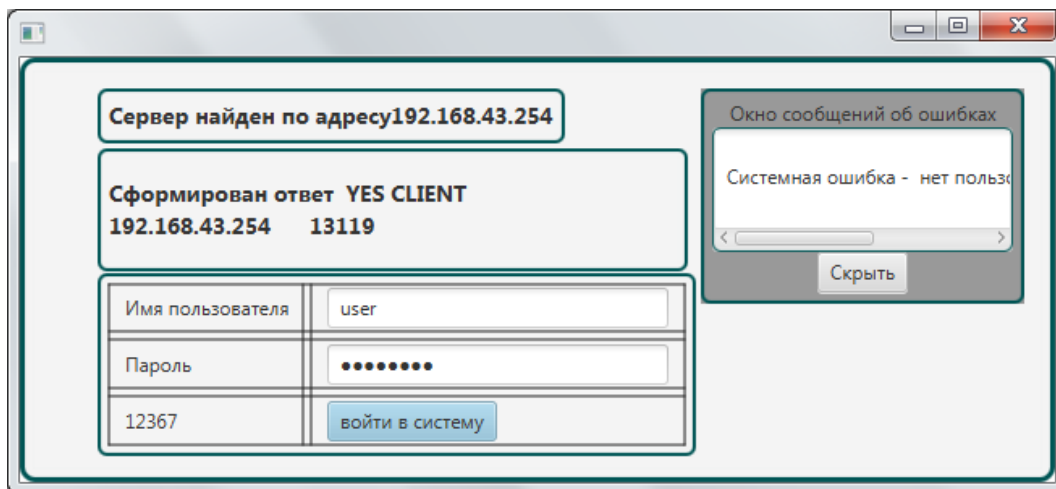


Рис. Б.5. Ввод неправильного пароля

3. Введено неправильно имя пользователя, правильно пароль. Выдана системная ошибка. В систему не запущено (рис Б.5.).

4. Введено неправильное имя пользователя и пароль. Выдана системная ошибка. В систему не запущено (рис Б.5).

Вывод: в результате испытаний вторая часть клиентской части отрабатывает верно.

3.3. Тестирование обмена сообщениями

Запущено два пользователя User и User1. Начала обмена сообщениями начинает User. Сообщение появилось в обоих окнах (рис Б.6, рис Б.7). Также видно, что в списке пользователей видно два пользователя. Программа на данном этапе работает корректно.

Далее продолжает User1. После отправления сообщения сообщение также видно в обоих окнах (рис Б.8, рис Б.9). Программа на данном этапе работает корректно.

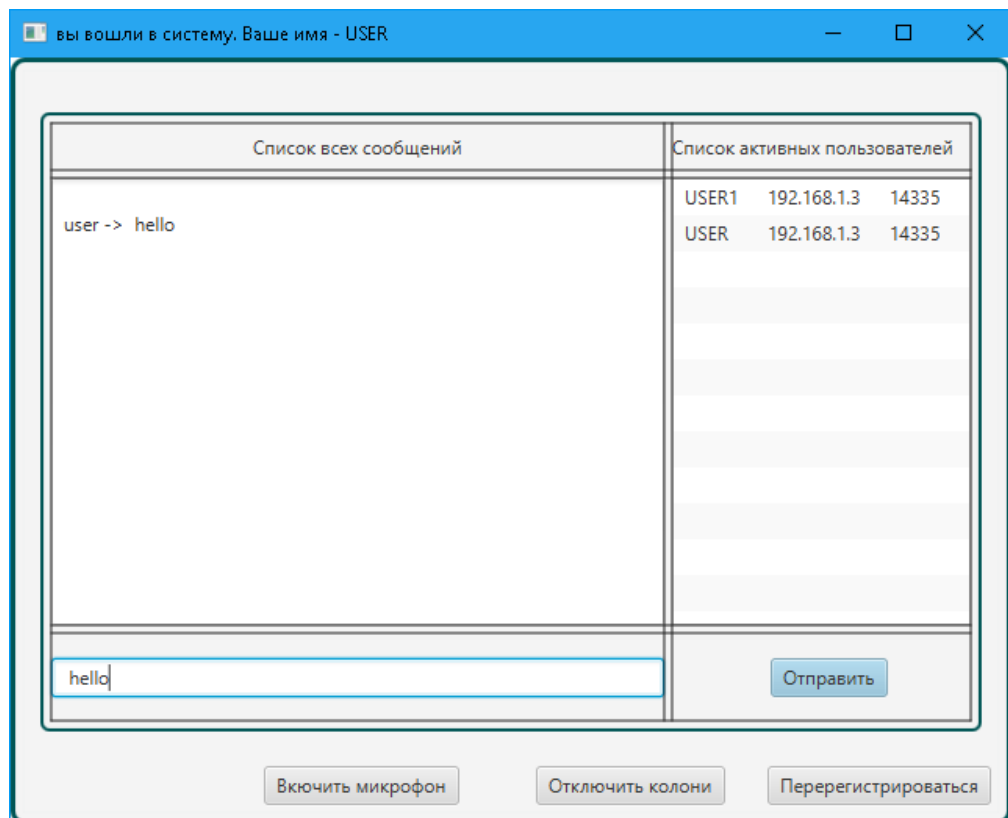


Рис. Б.6. Сообщение в первом окне

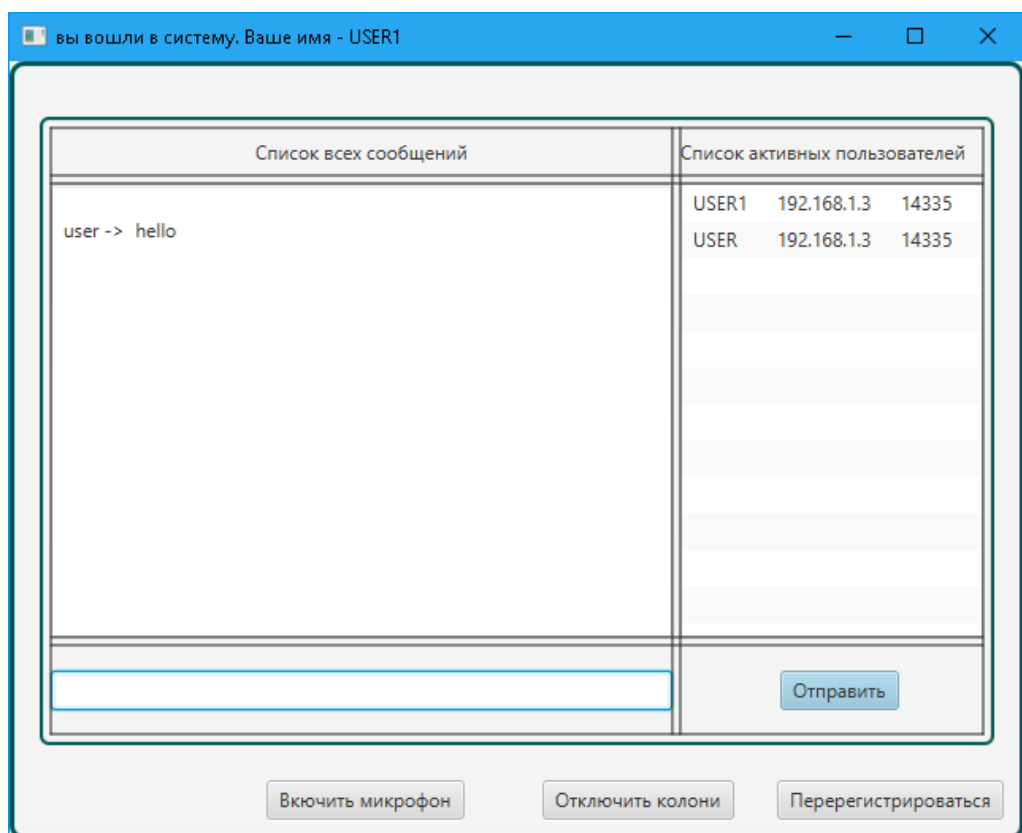


Рис Б.7. Сообщение во втором окне

Вывод: в результате испытаний третья часть клиентской части работает корректно.

3.4. Перерегистрация

Для регистрации нажимается кнопка «Перерегистрироваться», которая находится в левом нижнем углу.

По нажатию кнопки должно открыться окно ввода в систему. Программа выполняется корректно (рис В.10).

| | |
|--|--|
| Сервер найден по адресу 192.168.43.254 | |
| Сформирован ответ YES CLIENT 192.168.43.254 13785 | |
| Имя пользователя | <input type="text" value="user"/> |
| Пароль | <input type="password" value="....."/> |
| 12771 | <input type="button" value="войти в систему"/> |

Рис. Б.10. Окно для входа в систему

Необходимо удалить старые данные и ввести новые. Перерегистрация выполнена успешно (рис В.10), осуществлен вход под другим именем пользователя.

3.5. Выход из программы

Для выхода из программы нажат крестик в верхнем правом углу User1. Окно пользователя пропало. В окне User обновился список пользователей (рис Б.11- Б.12).

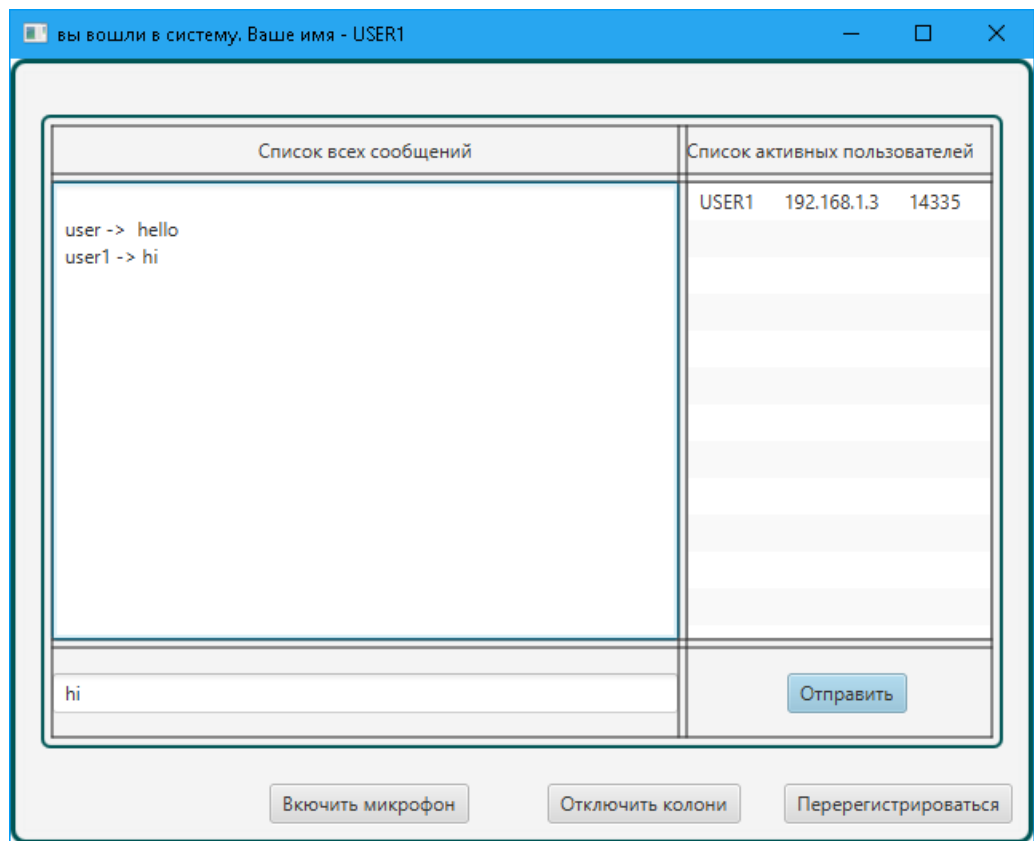


Рис. Б.11. Перегистрация

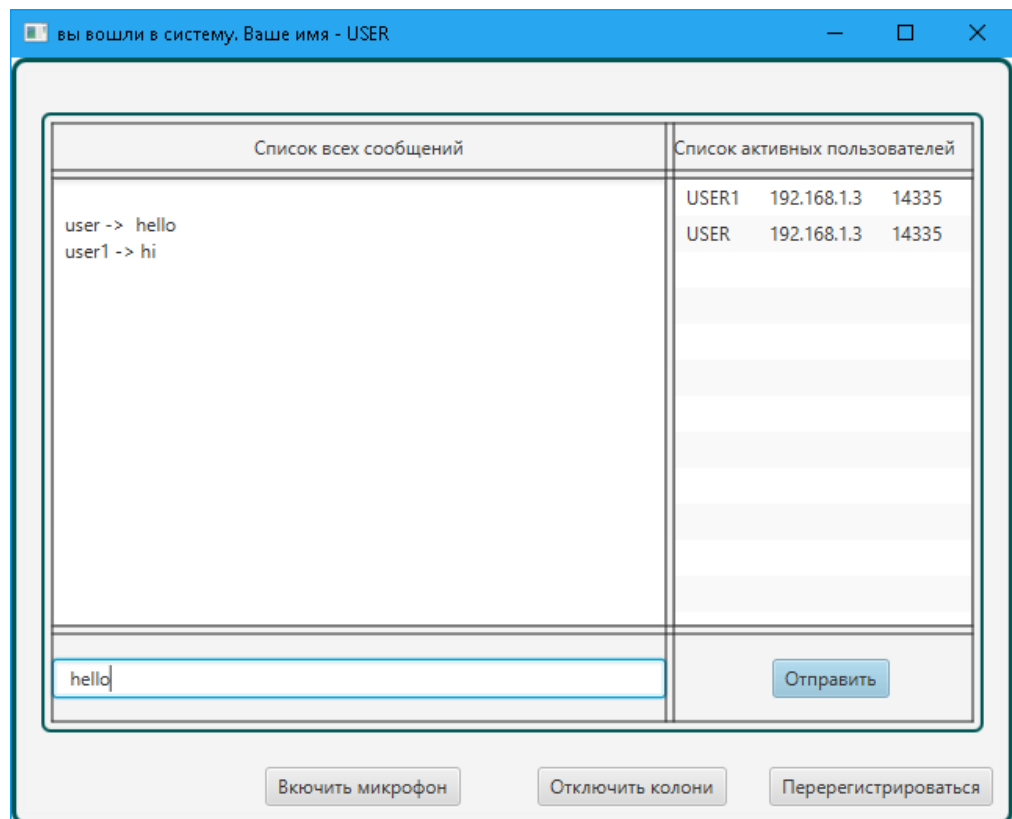


Рис. Б.12. Обновленная версия программы после выхода второго пользователя

Вывод: в результате испытаний четвертая часть клиентской части работает корректно.

3.6. Тестирование сервера

После запуска сервера в командную строку выводятся сообщения о запуске главных методов для работы сервера. Программа работает корректно (рис Б.13).

```
STERT NETstart  
STERT NETstart  
STERT NETstart  
STERT ressiv_font  
STERT ressiv_font
```

Рис. Б.13. Работа сервера

3.7. Запуск клиента без сервера

Запущен клиент без запуска сервера. Заставка запустилась корректно. Начался поиск сервера. Сервер не может быть найден (рис Б.14).

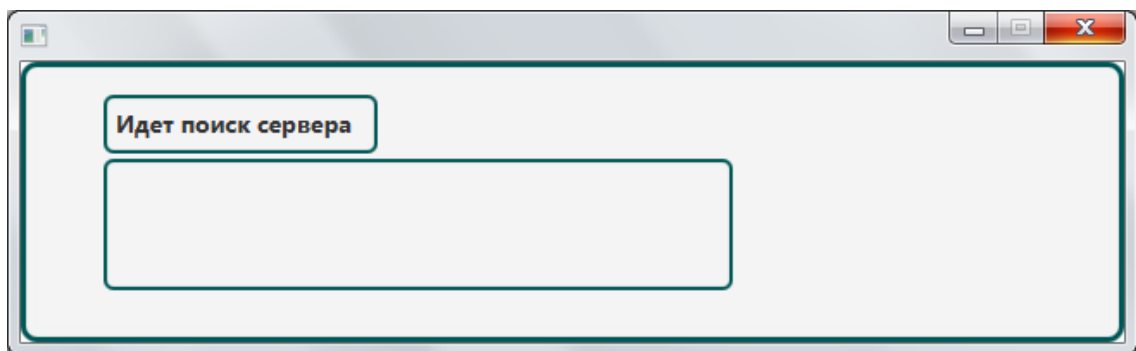


Рис. Б.14. Сервер не найден

Чтобы избежать подобной ошибки пользователи должны быть уверены, что сервер запущен, иначе обмен сообщениями будет невозможен.