Київський національний торговельно-економічний університет
Кафедра інженерії програмного забезпечення та кібербезпеки

ВИПУСКНИЙ КВАЛІФІКАЦІЙНИЙ ПРОЄКТ

на тему:

«Модель клієнт-серверного програмного забезпечення супроводу(адміністрування) відеоконференцій»

Студента 2м курсу, 2м групи, спеціальності 121 «Інженерія програмного забезпечення» спеціалізації «Інженерія програмного забезпечення» _____ підпис студента — Шевченко Дмитра Романовича

Науковий керівник кандидат технічних наук, доцент, кафедри інженерії програмного забезпечення та кібербезпеки _____ підпис керівника — Харченко Олександр Анатолійович

Гарант освітньої програми доктор економічних наук, професор кафедри інженерії програмного забезпечення та кібербезпеки _____ підпис гаранта — Токар Володимир Володимирович

КИЇВ – 2021

Київський національний торговельно-економічний університет

Факультет інформаційних технологій
Кафедра інженерії програмного забезпечення та кібербезпеки
Освітній ступінь магістр
Спеціальність 121 «Інженерія програмного забезпечення»

Затверджую
Зав. кафедри інженерії програмного забезпечення та кібербезпеки
Криворучко О. В.
«10» листопада 2020 р.

Завдання

на випускний кваліфікаційний проєкт студентові

Шевченко Дмитру Романовичу
(прізвище, ім'я, по батькові)

1. Тема випускного кваліфікаційного проєкту «Модель клієнт-серверного програмного забезпечення супроводу (адміністрування) відеоконференцій»
Затверджена наказом ректора від «30» листопада 2020 р. № 3224
2. Строк здачі студентом закінченого проєкту 25 листопада 2021
3. Цільова установка та вихідні дані до проєкту

Мета проєкту – розробка програмного забезпечення для проведення відеоконференцій

Об'єкт дослідження - модель клієнт-серверного програмного забезпечення супроводу (адміністрування) відеоконференцій

Предмет дослідження - розробка програмного забезпечення для проведення відеоконференцій

4. Консультанти проєкту із зазначенням розділів, які консультують:

Розділ	Консультант (прізвище, ініціали)	Підпис, дата	
		Завдання видав	Завдання прийняв
1			
2			
3			

5. Зміст випускного кваліфікаційної роботи (перелік питань за кожним розділом)

ВСТУП

РОЗДІЛ 1. АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ ЗАСТОСУВАННЯ ВІДЕОЗВ'ЯЗКУ

1.1 Теоретичне дослідження відеоконференцзв'язку

1.2 Обладнання для відеоконференцзв'язку

1.3 Протоколи для відеопередачі

Висновки до розділу 1

РОЗДІЛ 2. ВИБІР ПРОГРАМНИХ ЗАСОБІВ ТА ПРОЕКТУВАННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

2.1 Архітектура мережевої взаємодії

2.2 Опис протоколу передачі даних

2.3 Структура програми

Висновки до розділу 2

РОЗДІЛ 3. РЕАЛІЗАЦІЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ ДЛЯ ПРОВЕДЕННЯ ВІДЕОКОНФЕРЕНЦІЙ

3.1 Вимоги до системи та інструкція до застосування

3.2 Опис програмних модулів проекту

3.3 Опис інтерфейсу користувача

Висновки до розділу 3

ВИСНОВКИ ТА ПРОПОЗИЦІЇ

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

ТЕХНІЧНЕ ЗАВДАННЯ

ДОДАТКИ

6. Календарний план виконання проєкту

№ пор.	Назва етапів випускного кваліфікаційного проєкту	Строк виконання етапів проєкту	
		за планом	фактично
1	2	3	4
1.	*Вибір теми випускного кваліфікаційного проєкту*	*21.09.2020*	*21.09.2020*
2.	*Розробка та затвердження завдання на проєкт магістра (стац/заоч)*	*06.11.2020*	*22.12.2020*
3.	*Вступ та перелік літературних джерел*	*27.02.2021*	*27.02.2021*
4.	*Розробка технічного завдання*	*20.03.2021*	*20.03.2021*
5.	*Розділ 1. Аналіз предметної області застосування відеозв'язку*	*16.04.2021*	*16.04.2021*
6.	*Розділ 2. Вибір програмних засобів та проектування програмного забезпечення*	*24.05.2021*	*24.05.2021*
7.	*Розділ 3. Реалізація програмного забезпечення для проведення відеоконференцій*	*21.06.2021*	*21.06.2021*
8.	*Розробка програми та методики тестування*	*18.10.2021*	*18.10.2021*
9.	*Написання наукової статті*	*22.05.2021*	*22.05.2021*
10.	*Керівництво користувача*	*21.10.2021*	*21.10.2021*
11.	*Висновки та пропозиції*	*01.11.2021*	*01.11.2021*
12.	*Здача випускного кваліфікаційного проєкту на кафедру (перша перевірка)*	*03.11.2021*	*03.11.2021*
13.	*Підготовка автореферату та презентації доповіді*	*03.11.2021*	*03.11.2021*
14.	*Попередній захист випускного кваліфікаційного проєкту*	*22.11.2021 – 25.11.2021*	*22.11.2021*
15.	*Здача зброшурованої випускного кваліфікаційного проєкту*	*25.11.2021*	*25.11.2021*
16.	*Зовнішнє рецензування випускного кваліфікаційного проєкту*	*26.11.2021*	*26.11.2021*
17.	*Підготовка до публічного захисту випускного кваліфікаційного проєкту*		

7. Дата видачі завдання «30» листопада 2020 р.

8. Науковий керівник випускного кваліфікаційного проєкту Харченко О. А.
(прізвище, ініціали, підпис)

Гарант освітньої програми Токар В.В.
(прізвище, ініціали, підпис)

9. Завдання прийняв до виконання студент Шевченко Д.Р.
(прізвище, ініціали, підпис)

10. Відгук керівника випускного кваліфікаційного проєкту

Науковий керівник випускного кваліфікаційного проекту_____

(підпис, дата)

Відмітка про попередній захист Харченко О.А.

(ПІБ, підпис, дата)

Висновок про випускний кваліфікаційний проект

Випускний кваліфікаційний проект студента Шевченко Д.Р.

(прізвище, ініціали)

може бути допущена до захисту екзаменаційній комісії.

Гарант освітньої програми Токар В.В.

(прізвище, ініціали, підпис)

Завідувач кафедри Криворучко О.В.

(підпис, прізвище, ініціали)

«_____»_____20 ___ р.

АНОТАЦІЯ

Відповідно до мети дослідження робота присвячена розробці програмного забезпечення для проведення та адміністрування відеоконференцій.

В результаті порівняльного аналізу аналогічних рішень визначено що на ринку існують схожі продукти для проведення відеоконференцій, які застосують клієнт - серверну архитектуру такі як «Skype», «Zoom», «Google Meets».

Розробка серверної та клієнтської частини виконана у середовищі Microsoft Visual Studio при використанні мови C#.

Практичне значення одержаних результатів роботи полягає в розробці клієнт-серверного програмного продукту для забезпечення відеоконференції у таких областях, як менеджмент віддалених команд розробки та у якості соціальної мережі.

Ключові слова: відеоконференцзв’язок, адміністрування ВКЗ, передача відео у реальному часі.

ABSTRACT

According to the purpose of the research, the work is devoted to the development of software for conducting and administering video conferences.

As a result of comparative analysis of similar solutions, it was determined that there are similar products on the market for video conferencing, which use client-server architecture such as "Skype", "Zoom", "Google Meets".

The development of the server and client part was performed in the Microsoft Visual Studio environment using the C # language.

The practical significance of the obtained results is the development of a client-server software product to provide video conferencing in such areas as the management of remote development teams and as a social network.

Keywords: video conferencing, administering video conferences, real-time video transmission.

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, ОДИНИЦЬ, СКОРОЧЕНЬ І ТЕРМІНІВ

TCP (Transmission Control Protocol) – орієнтований з'єднання протокол, спрямований забезпечення надійної передачі між процесами, виконуваними одному чи різних комп'ютерах[2].

UDP (User Datagram Protocol) – простий орієнтований на дейтаграм протокол без організації з'єднання, що надає швидке, але необов'язково надійне транспортне обслуговування[2].

Клієнт – комп'ютер, що має віддалений доступ до інформації та програм, що зберігаються на сервері [1].

Сервер – комп'ютер, де зберігається інформація, надана по віддаленому доступу клієнтам[1].

Кодек (англ. codec, від coder/decoder - кодувальник/декодувальник або compressor/decompressor) - пристрій або програма, здатна виконувати перетворення даних або сигналу.

					КНТЕУ 121– 02-16.МР			
Зм.	*Аркуш*	*№ докум.*	*Підпис*	*Дата*				
Зав. каф.		Криворучко О.В.		27.02.202	Модель клієнт-серверного програмного супроводу (адміністрування) відеоконференцій	*Стадія*	*Аркуш*	*Аркушів*
Керівник		Харченко О.А.		27.02.202		*ПС*	*4*	*42*
Гарант		Токар В.В.		27.02.202		*Факультет інформаційних технологій*		
Розробив		Шевченко Д.Р.		27.02.202	Перелік умовних скорочень	*2 курс, 2м група*		

ЗМІСТ

ВСТУП

Актуальність теми. У компаніях та командах будь-якого розміру є наради для обговорення робочих моментів. Але не завжди і не у всіх співробітників є можливість бути присутніми фізично, тоді бізнесу доводиться організовувати віддалені зустрічі. У цих випадках необхідно налаштувати відеоконференції, які дозволяють підвищити ефективність внутрішніх комунікацій.

Особливо актуальною ця тема стала під час пандемії, коли багато компаній були змушені перейти на віддалену роботу, через що різко зросла потреба в якісному відеозв'язку не тільки для внутрішнього спілкування, а й для взаємодії з клієнтами.

Відеоконференція – це область інформаційної технології, що забезпечує одночасно двосторонню передачу, обробку, перетворення та подання інтерактивної інформації на відстань у реальному режимі часу за допомогою апаратно-програмних засобів обчислювальної техніки. Взаємодія в режимі відеоконференцій також називають сеансом відеоконференц-зв'язку.

Відео-конференцзв'язок (скорочена назва ВКЗ) - це телекомунікаційна технологія інтерактивної взаємодії двох і більше віддалених абонентів, при якій між ними можливий обмін аудіо- та відео- інформацією у реальному масштабі часу з урахуванням передачі даних, що управляють.

					КНТЕУ 121– 02-16.МР			
Зм.	*Аркуш*	*№ докум.*	*Підпис*	*Дата*				
Зав. каф.		Криворучко О.В.		27.02.202	Модель клієнт-серверного програмного супроводу (адміністрування) відеоконференцій	*Стадія*	*Аркуш*	*Аркушів*
Керівник		Харченко О.А.		27.02.202		*В*	*6*	*42*
Гарант		Токар В.В.		27.02.202		*Факультет інформаційних технологій 2 курс, 2м група*		
Розробив		Шевченко Д.Р.		27.02.202	ВСТУП			

Використання відеоконференцзв’язку має в останні роки широке поширення, так як на відміну від передачі тексту та голосових повідомлень, він є інтуїтивним і легко сприймається засобом комунікації, внаслідок того, що людина більшу частину інформації сприймає візуально. Відеоконференцзв’язок застосовується для особистого та ділового спілкування, проведення нарад, конференцій та інших важливих заходів, у яких важлива присутність тих суб’єктів спілкування, які з тих чи інших причин не можуть перебувати у місці проведення заходу особисто.

У такому разі завдяки сучасним засобам передачі даних та високопотужним комп’ютерам є можливість організувати спілкування у віртуальному режимі, максимально наближеному до реального спілкування – у вигляді відеоконференції.

Мета та завдання дослідження.

Мета цього дослідження полягає у розробці програмного забезпечення для проведення відеоконференцій.

Відповідно до мети було сформульовано такі **завдання дослідження**:

1. Здійснити теоретичний аналіз предметної області застосування відеоконференцзв’язку.
2. Вибрати програмні засоби та спроектувати програмне забезпечення.
3. Розробити програмне забезпечення для проведення відеоконференцій.

Об’єктом даного дослідження є розробка програмного забезпечення з супроводу (адміністрування відеоконференцій).

Предмет дослідження – модель клієнт-серверного програмного забезпечення супроводу(адміністрування) відеоконференцій.

Методи дослідження.

Під час роботи над теоретичною частиною дипломного проєкту та при обробці отриманих результатів практичного дослідження застосування відеоконференцзв'язку, було використано комплекс взаємодоповнюючих методів дослідження: методи аналізу та синтезу, конкретизації, системного підходу, описовий метод, метод узагальнення та структуризації, методи тестування програмного забезпечення.

Теоретичне і практичне значення отриманих результатів полягає у тому, що була проаналізована та узагальнена інформація щодо застосування відеоконференцзв'язку та розроблено програмне забезпечення з супроводу (адміністрування) відеоконференцій.

РОЗДІЛ 1

АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ ЗАСТОСУВАННЯ ВІДЕОЗВ'ЯЗКУ

1.1 Теоретичне дослідження відеоконференцзв'язку

Відеоконференція (від англ. Videoconference) - область інформаційної технології, що забезпечує одночасно двосторонню передачу, обробку, перетворення та представлення інтерактивної інформації на відстань в режимі реального часу за допомогою апаратно-програмних засобів обчислювальної техніки. Взаємодію в режимі відеоконференцій також називають сеансом відеоконференцзв'язку.

Відеоконференцзв'язок - це телекомунікаційна технологія інтерактивної взаємодії двох і більше віддалених абонентів, при якій між ними можливий обмін аудіо і відеоінформацією в реальному часі, з урахуванням передачі керуючих даних.

Мета, яку ставить перед собою система проведення відеоконференцій – це забезпечення за допомогою програмно-апаратних засобів спілкування, максимально наближеного до живого. Відеоконференція також забезпечує спільну роботу з різними даними.

Дистанційна діагностика людини, устаткування, віддалене навчання - ще один цікавий напрям застосування засобів відеоконференцій. Навіть перебуваючи в сотнях кілометрів від пацієнта, лікар може правильно продіагностувати хворого, вдаючись до «віртуальної» консультації висококласних фахівців, присутність яких в даному місці не представляється можливим.

Аналогічно група експертів може провести діагностування обладнання, перебуваючи в офісі і не витрачаючи час на нескінченні перельоти. Отримана останнім часом розвитком практика поступового впровадження засобів відеоконференцій у сферу навчання дозволить не просто прослухати і побачити лекцію відомого викладача, що знаходиться в іншій півкулі, але здійснювати інтерактивне спілкування за допомогою відеоконференцій.

Перші спроби з виробництва таких систем зробила 1964 р. компанія AT&T, що розробила аудіовізуальну систему електронного взаємодії. Наприкінці 1970 року. З'явилися перші системи відеоконференцзв'язку (ВКЗ), які сьогодні найбільш наближають інтерактивне спілкування реального.

Технологія відеозв'язку в усьому світі визнана як ефективний засіб для оптимізації бізнес-процесів, що дозволяє заощадити час та скоротити витрати на ділові поїздки, зменшуючи їхню кількість. Крім цього, вона знаходить своє застосування в різних галузях і сферах діяльності, таких як дистанційна освіта, медицина, судова система, державна управління та інше. Може бути застосована для вирішення різних завдань, де необхідне сприйняття саме зорової інформації.

Традиційно відеоконференції характеризувалися як комбінація спеціалізованого звуку і відео, а також технології роботи з мережами зв'язку для взаємодії в реальному масштабі часу і часто використовувалися робочими групами, які збиралися в спеціалізованому місці (зазвичай це був зал засідань, оснащений ВК обладнанням), щоб зв'язатися з іншими групами людей.

Історично склалося так, що проведення відеоконференцій можна розділити не тільки за технічними характеристиками і принципам відповідності різним стандартам, а й на настільні (індивідуальні), групові та студійні. Кожен з цих варіантів відеоконференцій чітко орієнтований на

рішення своїх завдань. Найбільш поширені завдяки відносно невисокій вартості і швидкості окупності витрат сьогодні настільні засоби проведення відеоконференцій.

Враховуючи функції та цілі застосування, обладнання відеоконференцзв'язку систематизується на категорії та класи.

Індивідуальні системи забезпечують можливість індивідуального відеоспілкування користувача в режимі реального часу, не залишаючи свого робочого місця. Для використання такої системи потрібні лише відносно дешеві програмно-аппаратні засоби. Для передачі відеосигналу може використовуватись низькошвидкісне з'єднання, як що відсутні вимоги до високої якості зображення. Такий тип відеозв'язку використовується для неформального спілкування між двома особами, обміну інтерактивною інформацією та файлами, при низьких витратах часу та фінансів.

Групові системи – завдяки розвитку алгоритмів та технологій обробки та передачі данних у сучасних системах з'явилася можливість проводити відеоконференції, що дозволяють спілкуватись групі користувачів одночасно. Груповий відеозв'язок став доступним для звичайних користувачів, теперь не обов'язково встановлювати дороге обладнання, достатньо лише встановити на комп'ютер оснащений веб-камерою програмне рішення. Як приклад, програмне рішення «Skype». У деяких рішеннях прийняти участь у конференції можна через інтернет-браузер, лише відкривши сайт. Приклад такого рішення – програмне забезпечення «Zoom». Такі рішення призначені та найблішь часто використовуються для проведення групових сеансів відеоконференцзв'язку у переговорних (дорадчих) кімнатах.

Галузеві системи – застосовуються у певної галузі. Наприклад, у медичній галузі дуже часто застосовують системи для проведення операцій (телемедицина), у судовій системі — для проведення дистанційних

касаційних та наглядових судових процесів, у нафтогазовій, енергетичній, будівельній галузі для оперативності подання інформації.

Мобільні системи дозволяють за короткий час організувати сеанс відеоконференцзв'язку в нестандартних умовах. Ці системи зазвичай використовуються державними органами, що приймають оперативні рішення.

Адміністративні системи - це сукупність апаратно-програмних засобів адміністрування/управління багатоточковими сеансами відеоконференцзв'язку з використанням різного кінцевого обладнання.

Сьогодні більшість компаній шукають способи використання цієї нової технології, щоб зберегти конкурентоспроможність на своєму сегменті ринку. Основна проблема з якістю відео полягає в тому, що наявні технології дозволяють здійснювати відносно низьку швидкість передачі кадру (фрейму).

Однак ця проблема може бути вирішена, якщо система буде використовувати гарну відеофіксацію та ефективну реалізацію стиснення зображення без істотної втрати якості.

Значно простіше вирішення проблем з якістю аудіо. Незважаючи на те, що середнє людське вухо в стані сприймати коливання від 20 Гц до 20 кГц, коливання, викликані людським голосом, лежать в значно більш вузькій смузі. Це дозволяє істотно зменшити витрати мережевого трафіку на передачу аудіоінформації. Ось чому багато постачальників систем настільних відеоконференцій воліють класти в основу своїх продуктів гарну якість аудіо і розвинені засоби групової обробки інформації [17].

1.2 Обладнання для відеоконференцзв'язку

Обладнання для відеоконференцій поділяється на три основні види:

1. Персональні системи відеоконференцій, що використовують можливості персонального комп'ютера та веб-камери;
2. Системи відеоконференцій для переговорних кімнат, призначені для встановлення у конференц-залах малих та середніх розмірів;
3. Системи відеоконференцій для залів засідання, які використовуються у великих конференц-залах та підтримують максимально можливу кількість додаткових сервісів.

Для організації відеоконференцій використовуються такі пристрої [3]:
Термінали абонентів із підтримкою аудіо та відеозв’язку — індивідуальні або групові відеосистеми або IP-телефони.

Термінал складається з наступних компонентів:

- Кодек (основний обробний пристрій);
- Камера;
- Мікрофон;
- Відображальний пристрій.
- Сервери багатоточкового зв’язку (MCU). MCU H.323 поєднує в собі обов'язковий багатоточковий контролер, що управляє з’єднаннями, і один або кілька опціональних мультимедійних процесорів, призначення яких - мікшування аудіо та відеосигналів, що надходять від багатьох учасників.
- Шлюзи (gateways) з’єднують комутовані ISDN-мережі з пакетними IP-мережами.
- Функції шлюзу включають перетворення форматів передачі даних та комунікаційних процедур (H.225/H.221 та H.245/H.242).
- Контролери зони (gatekeepers) - це програмні модулі, які авторизують підключення, транслюють імена терміналів і шлюзів, що використовуються в системі, в IP-адреси,

маршрутизують запити через шлюзи. Крім того, контролери зони надають додаткові послуги, такі як керування шириною смуги, переадресація виклику, підтримка служби каталогів, статистичні звіти для білінгових систем.

- Мережеві екрани та проксі-сервери (firewalls та proxies) запобігають несанкціонованому доступу до конференції у разі зв'язку через Інтернет.

Структура мережі відеоконференцзв'язку представлена: (рис.1.1)

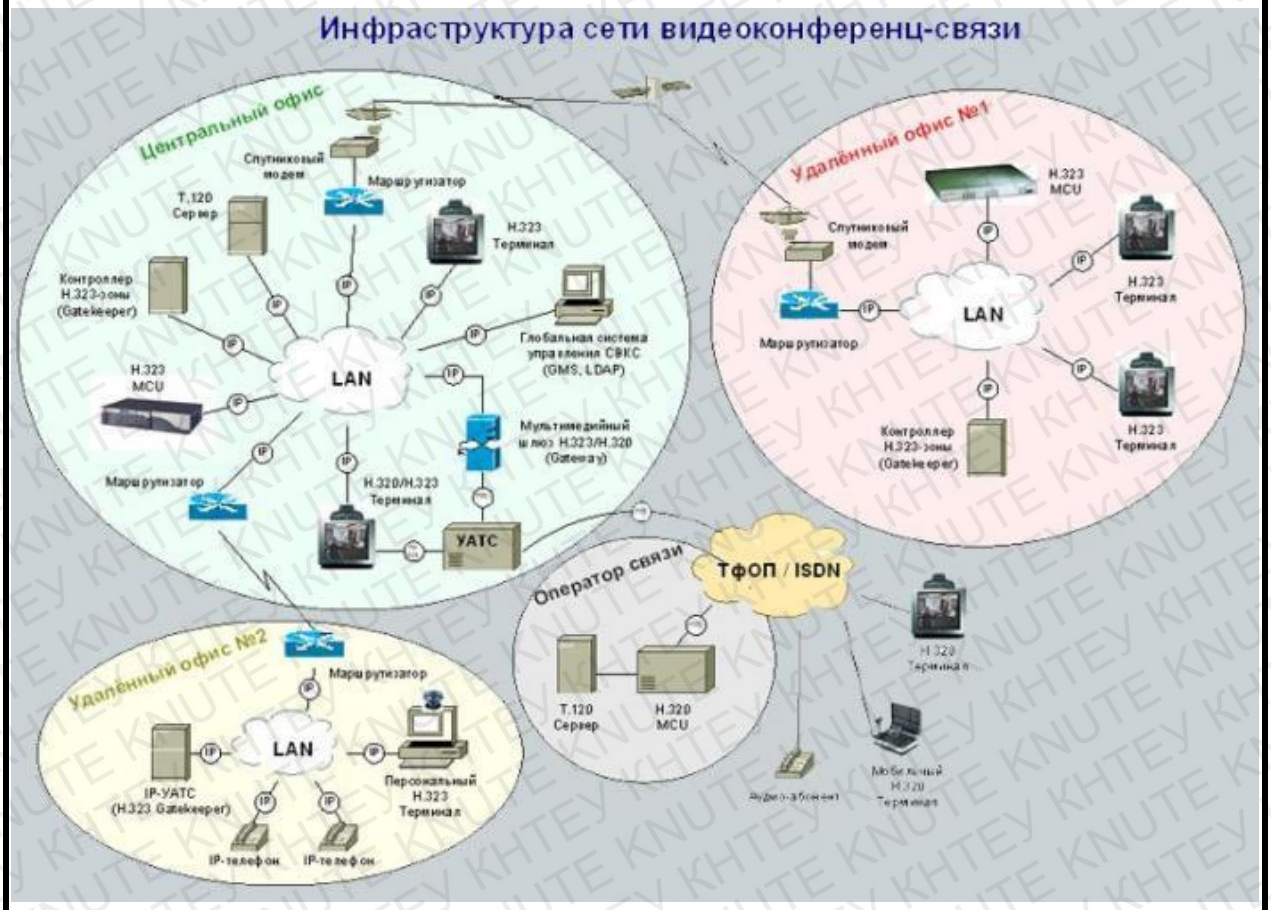


Рис. 1.1. Інфраструктура мережі відеоконференцзв’язку

Залежно від конфігурації терміналів та MCU, розрізняються такі моделі обміну даними [1]:

- Централізована – кожен термінал посилає свої аудіо- та відеосигнали на MCU, у двоточковому режимі (див. рисунок 1.2).
- MCU визначає можливості кожного з терміналів, та формує медіапотоки відповідно до цих можливостей та розсилає їх.

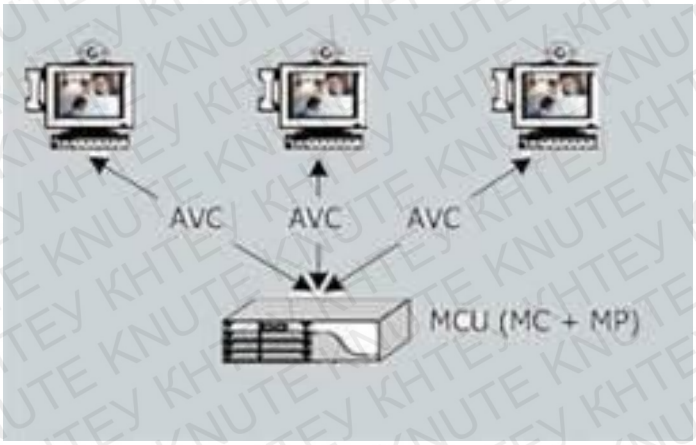


Рис. 1.2. Централізована модель обміну даними ВКЗ

- Децентралізована – термінали розсилають свої аудіо- та відеосигнали всім іншим терміналам у режимі Multicast (див. малюнок 1.3.). Мікшування прийнятого аудіосигналу та вибір відеосигналу для відображення виконується терміналами. Керуюча інформація та дані, як і раніше, передається між контролером, що входить до складу MCU, і терміналами в двоточковому режимі.
- Multicast дозволяє використовувати мережеві ресурси ефективніше, але збільшує обчислювальне навантаження на термінали; Крім того, маршрутизатори та комутатори, які використовуються в мережі, також повинні підтримувати Multicast.

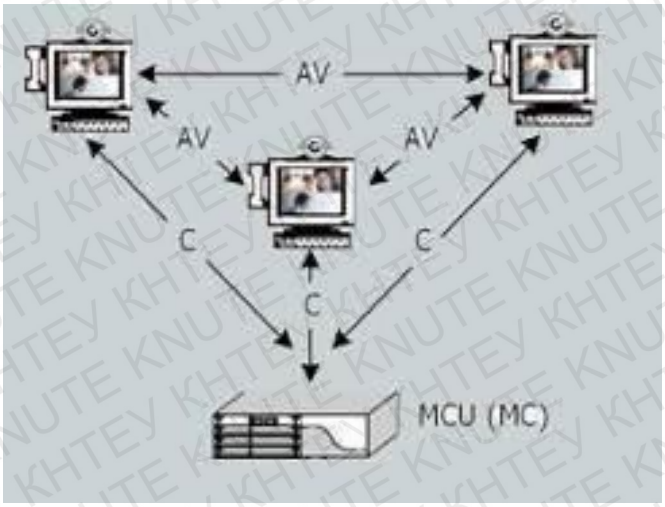


Рис. 1.3. Децентралізована модель обміну даними ВКЗ

Каскадна — найдешевша модель для великих конференцій із багатьма учасниками. Взаємодіють два або більше MCU, кожен із яких керує групою терміналів.

У деяких ситуаціях, коли в конференції задіяно багато терміналів, можлива така конфігурація системи, при якій тільки деякі абоненти беруть у сеансі активну участь, а інші підключаються тільки для перегляду. При цьому можлива динамічна зміна статусу учасника за необхідності.

1.3 Протоколи для відеопередачі

Стрімке зростання передачі мультимедіа-інформації висуває нові вимоги до швидкості та обсягів передачі даних. І для того, щоб задовольнити всі ці запити, одного збільшення ємності мережі недостатньо, необхідні розумні та ефективні методи управління трафіком та контролем завантаженості ліній передачі.

У додатках реального часу відправник генерує потік даних із постійною швидкістю, а одержувач (або одержувачі) повинен надавати ці дані додатку з тією самою швидкістю. Такі програми включають, наприклад,

аудіо- та відеоконференції, живе відео, віддалену діагностику в медицині, комп’ютерну телефонію, розподілене інтерактивне моделювання, ігри, моніторинг у реальному часі[7].

Найбільш широко використовуваний протокол транспортного рівня це TCP. Незважаючи на те, що TCP дозволяє підтримувати безліч різноманітних розподілених програм, він не підходить для програм реального часу. TCP забезпечує гарантовану доставку пакетів та перевірку їх цілісності та відмінно підходить для завдань, пов'язаних із передачею важливої інформації, не прив’язаної до часу[7].

Протокол TCP здійснює перевірку доставки пакета та у разі невдалої доставки здійснює повторну передачу. Такий механізм неприпустимий для передачі у режимі реального часу, оскільки у разі втрати пакета та його повторної передачі решта пакетів буде змушена чекати доставки загубленого пакета, що призведе до суттєвого розриву в часі передачі та прийому пакетів – що є неприпустимим для передачі інформації в реальному часі[7].

Це завдання вирішує дейтаграмний протокол UDP, завдання якого – швидка доставка дейтаграм без встановлення з’єднання, повторної передачі та гарантії доставки пакета. Незважаючи на ці недоліки, протокол UDP кращий для передачі даних реального часу, оскільки забезпечує більш швидку передачу даних.

У середовищі реального часу відправник генерує пакети з постійною швидкістю. Вони відправляються через однакові інтервали часу, проходять через мережу та приймаються одержувачем, який відтворює дані в реальному часі після їх отримання.

Однак через зміну часу затримки при передачі пакетів по мережі вони можуть прибувати через нерегулярні інтервали часу. Для компенсації цього ефекту пакети, що надходять, буферизуються і потім надаються з постійною швидкістю програмного забезпечення, що генерує висновок.

1.4 Висновки до розділу 1

Отже, за теоретичною частиною ми з'ясували, що відеокоференцзв'язок – це інтерактивний інструмент, який включає в себе аудіо, відео, комп'ютерні та комунікаційні технології для здійснення зв'язку віддалених територіально співрозмовників «обличчям до обличчя» в реальному часі, а також має можливість поділу всіх типів інформації, включаючи дані, звук, зображення, документи і т.п. По суті, відеозв'язок дозволяє подолати бар'єр відстані для здійснення ефективної комунікації.

Висока якість звуку і повноекранне відео, можливість оперативного обміну даними і документами, які підтримуються розвиненими протоколами – роблять відеозв'язок потужним інструментом з найширшим спектром практичного застосування.

РОЗДІЛ 2

ВИБІР ПРОГРАМНИХ ЗАСОБІВ ТА ПРОЕКТУВАННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

2.1 Архітектура мережевої взаємодії

Для мережевої взаємодії застосовується гібридна архітектура. З одного боку, необхідно забезпечити гарантовану передачу даних між клієнтами та сервером, тому була реалізована архітектура «клієнт-сервер», схема якої представлена на рис. 2.1.

Рис. 2.1. Архітектура «клієнт-сервер»

Вона забезпечує централізоване управління клієнтами на сервері та ефективну гарантовану взаємодію клієнтів між собою через сервер. З іншого боку, необхідно забезпечити швидку та ефективну взаємодію між клієнтами для передачі мультимедіа. Тому було реалізовано архітектуру «точка-точка», схема якої представлена рис. 2.2.

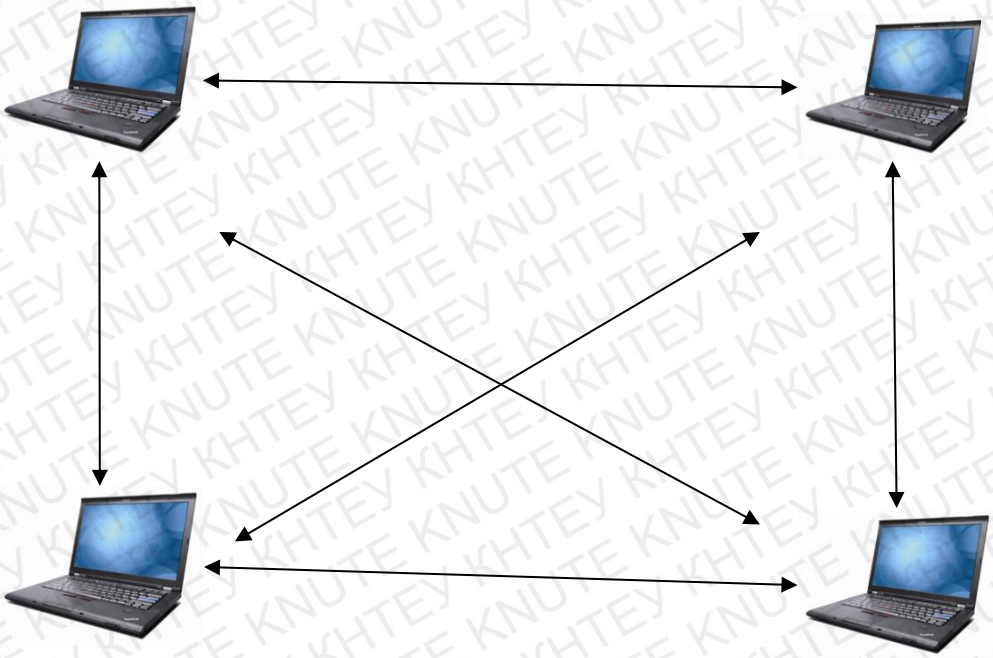


Рис. 2.2. Архітектура «крапка-крапка»

За такої архітектури всі учасники процесу рівноправні. Взаємодія відбувається між клієнтами, які одночасно є серверами. Окремо виділеного сервера немає, тому навантаження на сервер відсутня, що дозволяє масштабувати систему. Такий підхід є досить вигідним при передачі аудіо- та відеоданих.

Таким чином, у системі використовується гібридна архітектура, яка використовує клієнт-серверну взаємодію для здійснення управління в системі та взаємодію «точка-точка» для передачі даних між клієнтами.

2.2 Опис протоколу передачі данних

Для здійснення передачі даних між клієнтом і сервером використовується протокол, заснований на транспортному протоколі TCP, передача даних між клієнтами здійснюється за протоколом, який використовує як основу дейтаграмний протокол UDP.

Взаємодія клієнта з сервером вимагає високого ступеня надійності, тому потрібна гарантована доставка повідомлень. Для цього використовувався протокол, що базується на протоколі TCP. Розглянемо його основні аспекти.

При підключенні клієнта встановлюється з'єднання із сервером. Сервер, отримуючи запит на з'єднання, виділяє потік взаємодії з користувачем і відправляє йому підтвердження.

Після цього клієнт надсилає серверу повідомлення Authorization, яке супроводжує ім'я та пароль користувача, який намагається підключитися до сервера. Сервер аналізує отримані дані – перевіряє, чи є користувач з такими обліковими даними у списку зареєстрованих користувачів, чи не знаходиться користувач з таким ім'ям у мережі, і надсилає клієнту повідомлення AuthorizationSuccess або AuthorizationFailed залежно від результату перевірки.

У разі реєстрації користувача клієнт надсилає повідомлення Registration, супроводжуючи його обліковими даними. Сервер перевіряє, чи немає у списку зареєстрованих користувачів користувача з такою назвою, і надсилає клієнту повідомлення RegistrationSuccess чи RegistrationFailed залежно від результатів перевірки.

Клієнт, отримавши повідомлення про успішну реєстрацію або авторизацію, відправляє на сервер повідомлення ClientInfo для отримання інформації про користувачів, що знаходяться в мережі. Сервер при отриманні

такого повідомлення надсилає клієнту інформацію про всіх активних користувачів. Таким чином, кожен клієнт має інформацію про всіх активних користувачів і, у разі потреби, зможе організувати передачу даних з іншим клієнтом.

При необхідності припинити взаємодію клієнт надсилає серверу повідомлення Disconnect. Сервер, отримавши це повідомлення, видаляє всю інформацію про користувача.

Схема взаємодії представлена рис. 2.3.

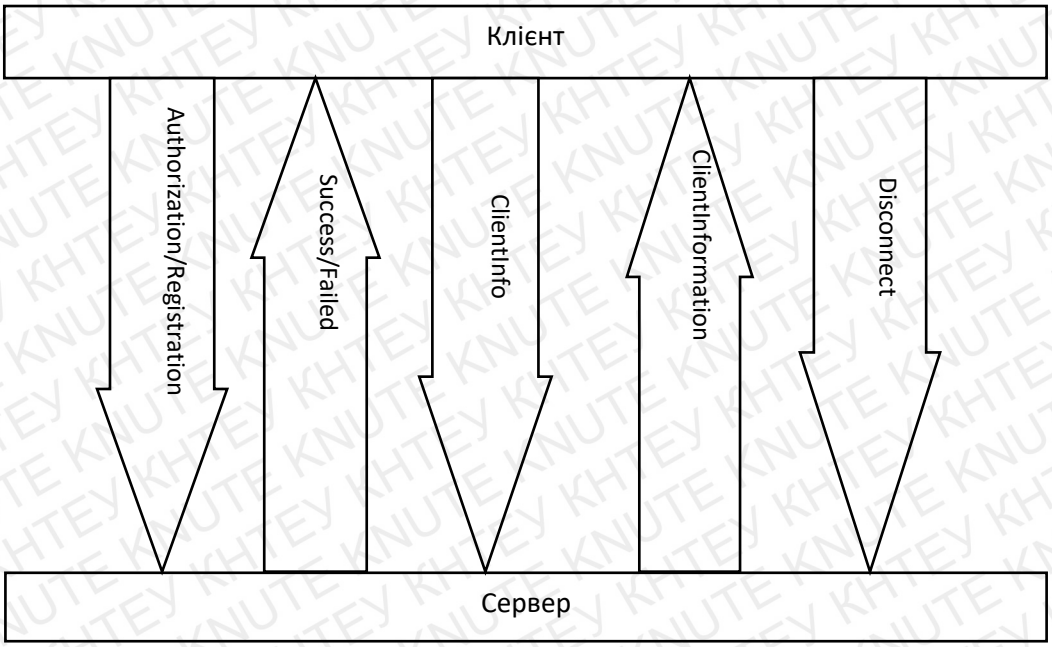


Рис. 2.3. Схема взаємодії між клієнтом та сервером

Взаємодія між клієнтами здійснюється як за протоколом TCP, і за протоколом UDP.

За протоколом TCP здійснюється надсилання запиту на передачу медіаданих, підтвердження або відмова від ініціювання передачі, а також повідомлення про відключення користувача від процесу відеопередачі.

За протоколом UDP здійснюється безпосередньо передача відео- та аудіоданих.

Під час ініціювання запиту на передачу даних Клієнт 1 надсилає повідомлення TransferRequest Клієнту 2, з яким він хоче з'єднатися.

Клієнт 2, отримавши повідомлення, погоджується або відмовляється від з'єднання та надсилає повідомлення TransferSuccess або TransferFailed відповідно. Якщо з'єднання встановлено, то починається передача відео- та аудіоданих за протоколом UDP.

Якщо Клієнт 1 хоче припинити передачу даних – він надсилає Клієнту 2 повідомлення TransferEnd, після чого обидва клієнти припиняють передачу медіаданих.

Схема взаємодії між клієнтами представлена рис. 2.4.

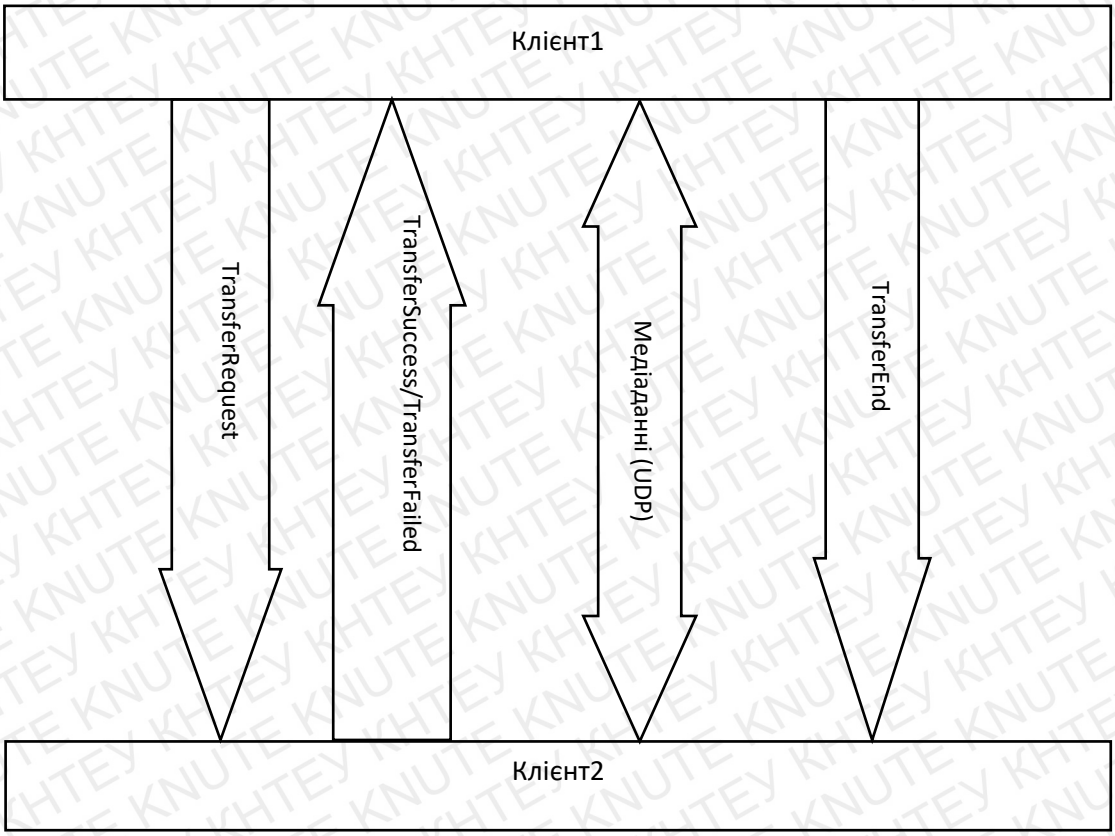


Рис. 2.4. Взаємодія між клієнтами

2.3 Структура програми

Серверний додаток виконує функцію, що управляє, функцію координатора клієнтів, а також веде збір статистики.

Структура серверного додатка представлена рис.2.5.

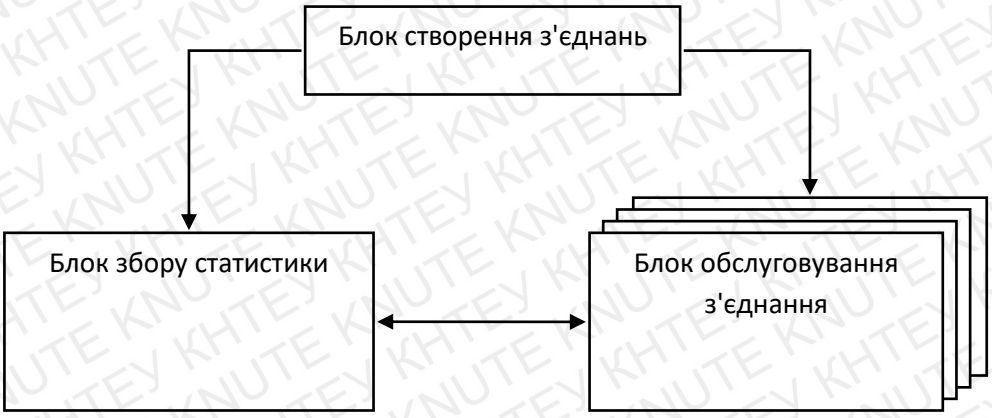


Рис. 2.5. Структура серверної програми

Блок створення з’єднань – здійснює прослуховування порту та при отриманні запиту на з’єднання створює об’єкт класу Connection, виділяючи йому потік. Ініціює об’єкт класу StatisticBlock. Потім передає керування блоку обслуговування з’єднань.

Блок обслуговування з’єднань – здійснює отримання та обробку запитів від клієнтів, а також надсилання повідомлень клієнтам.

Алгоритм роботи цього блоку показаний рис. 2.6.

У нескінченному циклі блок читає пакети, що надходять від клієнта, з яким встановлено з’єднання. Потім відбувається аналіз типу отриманого пакета. Якщо тип пакета = Registration, відбувається реєстрація нового користувача на сервері. Якщо тип пакета = Authorization – авторизація користувача на сервері.

При отриманні пакета з типом ClientInfo відбувається передача інформації про всіх клієнтів цього клієнта. Отримавши пакет з типом Disconnect - видаляє користувача зі списку активних користувачів і відсилає всім клієнтам повідомлення про користувача, що відключився.

Блок збору статистики - здійснює збір інформації про всі дії, зроблені на сервері. Отримані дані записує файл на жорсткому диску.

Блок створення з'єднання створюється безпосередньо після запуску сервера та функціонує до виходу із серверної програми. У ньому створюються блок збору статистики, який також функціонує протягом усього часу роботи сервера, та блоки обслуговування підключень, кожен із яких створюється при з'єднанні з новим клієнтом і знищується при відключенні клієнта.

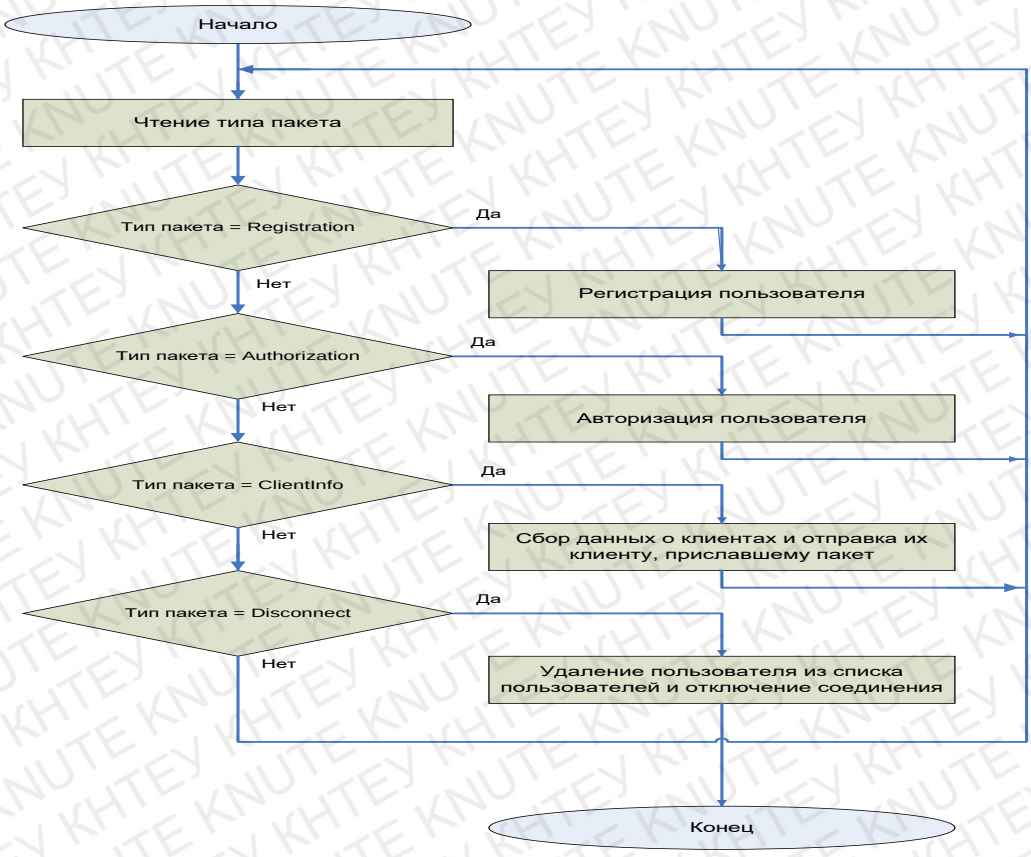


Рис. 2.6. Блок схема алгоритму роботи блоку обслуговування з'єднання

Завданнями **клієнтської програми** є взаємодія з користувачем з одного боку та з сервером з іншого боку.

Структура клієнтського додатка представлена рис. 2.7.

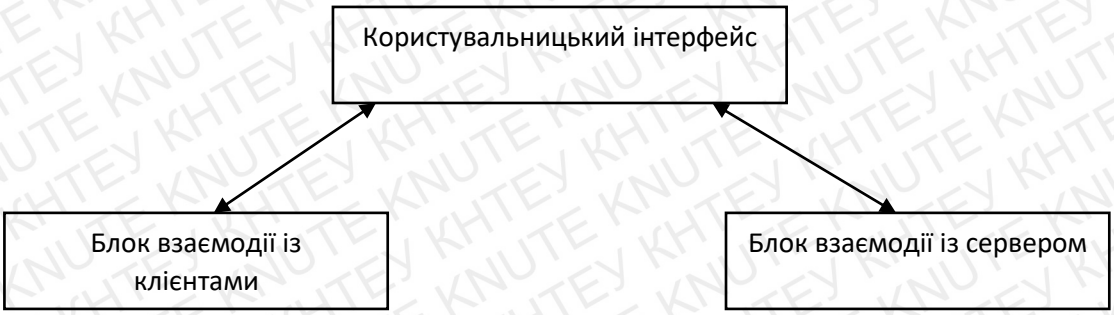


Рис. 2.7. Структура клієнтської програми

Інтерфейс користувача – виконує функцію взаємодії користувача з програмним комплексом. З його допомогою на екран виводяться дані, отримані від сервера (список користувачів), і дані отримані з інших клієнтів (відео і звук).

Блок взаємодії із сервером – здійснює прийом і передачі пакетів, і координацію дій клієнта залежно від даних. У цьому блоці здійснюється реєстрація та авторизація користувачів, отримання списку доступних користувачів, комутація з іншими клієнтами для початку та припинення передачі даних.

Алгоритм роботи блоку представлений на малюнку 2.8.

Після надсилання клієнтом серверу запиту на підключення клієнт може отримати одне з наступних повідомлень: Registration/AuthorizationSuccess, Registrtion/AuthrizationFailed, ClientAllreadyConnect. У разі повідомлень першого типу – клієнт успішно зареєстровано/авторизовано на сервері, він надсилає запит на отримання списку доступних користувачів.

У другому випадку реєстрація/авторизація була невдалою і на екран виводиться повідомлення з причиною невдачі. У разі третього повідомлення – на екрані відображається повідомлення про те, що користувач з таким ім’ям вже в мережі.

При підключенні до сервера нового клієнта всім іншим клієнтам надсилається повідомлення ClientConnected, отримавши яке клієнт запитує інформацію про нового користувача і додає її до списку доступних користувачів.

При відключенні клієнта від сервера всім іншим клієнтам надсилається повідомлення ClientDisconnected, отримавши яке клієнт видаляє інформацію про користувача зі списку доступних користувачів. Під час ініціювання передачі відео клієнт надсилає серверу пакет для іншого клієнта з типом TransferRequest, що містить інформацію про користувача.

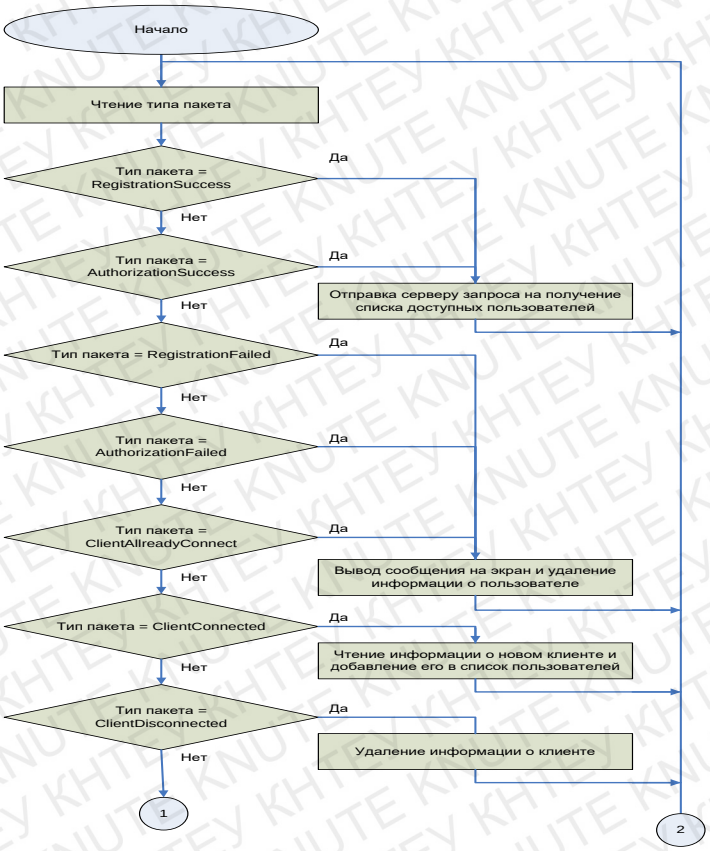


Рис. 2.8. Блок-схема алгоритму роботи блоку взаємодії із сервером

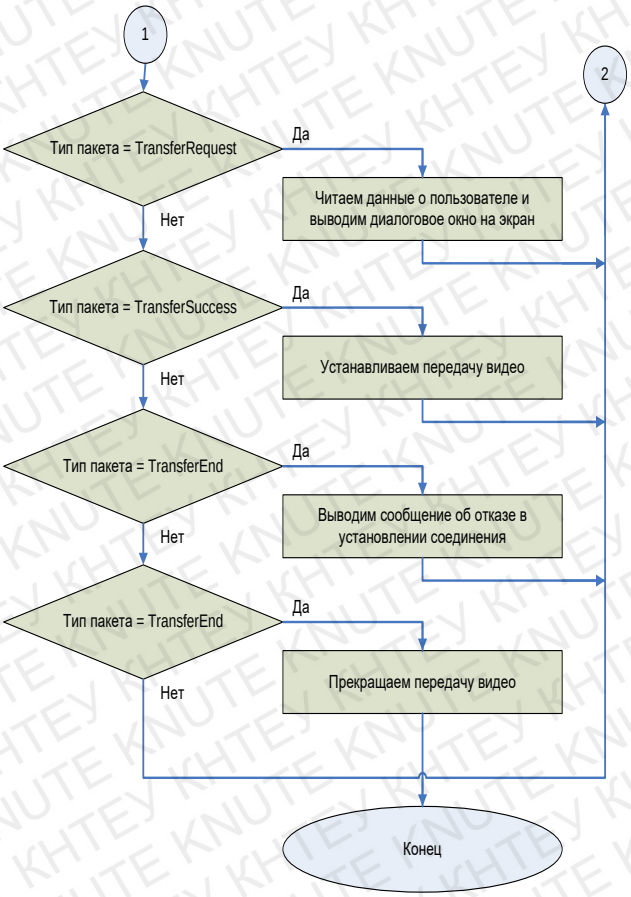


Рис. 2.8. Блок-схема алгоритму роботи блоку взаємодії із сервером

Клієнт, який отримав таке повідомлення, читає дані про користувача та виводить повідомлення на екран із запитом на з'єднання. Якщо користувач готовий розпочати відеопередачу, він надсилає клієнту пакет типу TransferSuccess, який містить інформацію про даного користувача.

Отримавши такий пакет, клієнт читає інформацію про користувача та ініціює відеопередачу.

Якщо користувач не готовий або не бажає розпочати відеопередачу - він відправляє пакет з типом TransferFailed, отримавши який інший клієнт повідомляє користувача про відмову в з'єднанні.

При завершенні відео, клієнт отримує повідомлення TransferEnd і припиняє передачу даних.

Блок взаємодії з клієнтами здійснює передачу та прийом медіаданих від інших клієнтів, а також організацію взаємодії між клієнтами (запити на початок та припинення передачі даних).

Відеоінформація, що отримується з веб-камери, перетворюється на послідовність кадрів, яка передається за протоколом UDP. Для захоплення відео використовуються засоби бібліотеки DirectShow. Захоплення звуку здійснюється також за допомогою бібліотеки DirectShow та записується в буфер фрагментами, які згодом передаються також за протоколом UDP.

Інтерфейс користувача взаємодіє як з блоком взаємодії з сервером, так і з блоком взаємодії з клієнтами.

2.4 Висновки до розділу 2

У другому розділі ми розглянули архітектуру мережевої взаємодії. У системі використовується гібридна архітектура, яка використовує клієнт-серверну взаємодію для здійснення управління в системі та взаємодію «точка-точка» для передачі даних між клієнтами.

З’ясували, що для здійснення передачі даних між клієнтом і сервером використовується протокол, заснований на транспортному протоколі TCP, передача даних між клієнтами здійснюється за протоколом, який використовує як основу дейтаграмний протокол UDP.

РОЗДІЛ 3

РЕАЛІЗАЦИЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ ДЛЯ ПРОВЕДЕННЯ ВІДЕОКОНФЕРЕНЦІЙ

3.1 Вимоги до системи та інструкція до застосування

Для нормального функціонування програмного комплексу система повинна задовольняти такі **вимоги**:

- операційна система Windows XP, Windows Vista, Windows 7, Windows 10;
- встановлений .NET Framework 3.0;
- бібліотеки DirectShow та Voice;
- наявність веб-камери (вбудованої чи зовнішньої);
- наявність мікрофона (вбудованого чи зовнішнього).

Для **встановлення програмного забезпечення** необхідно скопіювати папку на жорсткий диск та дотримуватися інструкцій у файлі INSTALL.txt

Як **середовище програмування** було обрано середовище Microsoft Visual Studio, тому що воно має такі переваги:

- зручність розробки програмних продуктів;
- велика різноманітність класів, у тому числі для розробки мережевих програм;
- простота реалізації інтерфейсу користувача;
- наявність засобів для комфортного налагодження мережевих програм.

Як мову програмування було обрано мову C#, оскільки він дозволяє легко розробляти складні кроссплатформенні програми, має потужну бібліотеку класів і зручні методи розробки додатків.

Для захоплення відео та аудіо використовувалися бібліотека DirectShow, які дозволяють побудувати систему кодування, стиснення, передачі та декодування медіаданих.

3.2 Опис програмних модулів проекту

Серверний додаток складається з трьох модулів, які відображають суть блоків, описаних у пункті 2.3.

Модуль створення з'єднань – основний модуль серверної програми. Він представлений класом Server, листинг якого відображено у додатку (А1).

Клас містить інформацію про сервері – адресу та порт, сокет для прийому та передачі даних клієнтам. Також у класі міститься головний потік серверної програми, який приймає запити на підключення клієнтів, список встановлених з'єднань – об'єктів класу Connection. У сервері зберігається список зареєстрованих та активних користувачів. Тут ініціалізується об'єкт LogWriter, який відповідає за ведення журналу подій.

Цей модуль здійснює очікування нового підключення, а коли підключення ініційовано – створює нове з'єднання передачі даних.

Модуль обслуговування з'єднань – модуль, завданням якого є здійснення зв'язку між клієнтом та сервером. Він представлений класом Connection, листинг якого відображено у додатку (А2).

Цей клас містить інформацію про конкретне з'єднання – сокет передачі даних, ім'я клієнта і потік для організації паралельної обробки сервером запитів від клієнтів. Завдання модуля – отримання, аналіз та відправлення пакетів клієнтам.

Модуль збору статистики необхідний для здійснення контролю дій, що відбуваються на сервері. Модуль представлений класом LogWriter, листинг якого представлений у додатку (А3).

Завдання модуля – здійснювати аналіз подій і записувати інформацію у лог-файл для подальшого аналізу станів сервера. Модуль тісно пов'язаний з двома вищеописаними методами, оскільки виконується поруч із роботою інших модулів клієнтського докладання.

Клієнтська програма складається з трьох модулів, які відображають сутність блоків, описаних у пункті 2.3.

Інтерфейс користувача здійснює взаємодію користувача з програмою. Інтерфейс користувача представлений головною формою програми, яка містить вікно для виведення відео, поле, що містить список користувачів і набір кнопок для управління процесом відеопередачі. Детальний опис інтерфейсу користувача можна знайти в пункті 3.3.

Модуль взаємодії з сервером призначений для надсилання та отримання повідомлень від сервера. Представлений класом Client, листинг якого наведено у додатку (Б1).

У цьому класі зберігається інформація про поточне з'єднання з сервером – ім'я, пароль, сокет, потік обробки повідомлень від сервера, а також список підключених користувачів. Завдання модуля – здійснити авторизацію користувача на сервері, отримати список доступних користувачів і далі чекати на повідомлення від сервера та здійснювати їх обробку. Даний модуль взаємодіє з інтерфейсом користувача шляхом генерації та обробки подій.

Модуль взаємодії між клієнтами – модуль, що реалізує передачу відео з клієнтами, а також його додаткову обробку. Модуль реалізує основну функціональність клієнтської програми, а саме захоплення, кодування, передачу, отримання, декодування та виведення на екран відеоінформації.

Модуль передачі та отримання зображення представлений двома класами VideoSender та VideoReceiver.

Основу відеозахоплення представляє клас Capture. Він побудований на основі бібліотеки DirectShow і здійснює захоплення зображення з веб-камери. Лістинг класу наведено у додатку (Б2).

Клас VideoSender здійснює передачу зображення, захопленого за допомогою об'єкта класу Capture. У цьому класі відбувається отримання зображення з веб-камери, запис його в потік і надсилання потоку UDP-протоколу.

Клас VideoReceiver займається отриманням та виведенням на екран зображення. Він читає надіслане зображення у потік, потім створює об'єкт класу Image з потоку та виводить зображення на екран.

Докладніший опис класу Capture наведено нижче.

Клас Capture побудований на основі інтерфейсу ISampleGrabberCB, який дозволяє отримати буфер. Завдання класу – отримати зображення з камери, представлене у вигляді покажчика Bitmap.

Для цього будується граф, який послідовно додаються пристрій відеозахоплення, фільтр захоплення, визначається розмір, тип зображення. Виконується установка параметрів відео: ширини і висоти зображення, частоти кадрів, якості зображення, що передається. Потім відбувається отримання зображення, копіювання його до буфера і передача покажчика на буфер об'єкту класу VideoSender, який потім передає зображення іншому клієнту.

3.3 Опис інтерфейсу користувача

Основу користувальницького інтерфейсу становить головне вікно клієнтської програми. Його вид представлений рис. 3.1.

Рис. 3.1. Головне вікно клієнтської програми

У центрі головного вікна знаходиться поле для виведення відео, отриманого від іншого користувача. У правому нижньому куті – поле-ескіз для виведення зображення, отриманого з власної веб-камери.

У лівому верхньому куті розміщується меню. Вкладка «Файл» має підпункти «Підключення» та «Відключення» (див. рис. 3.2.).

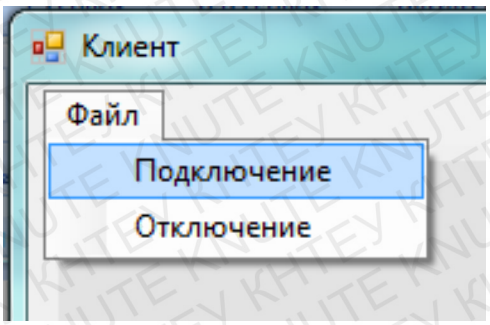
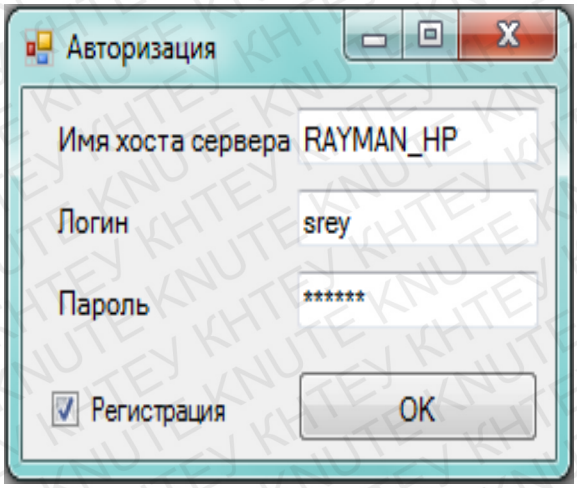


Рис. 3.2. Меню «Файл»

При натисканні на пункт «Підключення» з’являється діалогове вікно «Авторизація» (див. рис. 3.3), за допомогою якого можна здійснити авторизацію або реєстрацію на сервері. У цьому діалоговому вікні

розташовано поле для введення імені комп'ютера, на якому знаходиться сервер, поля для введення імені та пароль користувача. Також у вікні знаходиться прапор «Реєстрація», встановивши який можна зареєструватися на сервері.



Рису. 3.3. Диалоговое окно «Авторизация»

Після успішної авторизації/реєстрації в нижньому лівому куті з'являється напис «<ім'я користувача> в мережі» (див. рис. 3.4.).

Рис. 3.4. Поле статусу

Також клієнт отримує від сервера інформацію про підключених користувачів та виводить їх у поле «Користувачі в мережі» (див. рис. 3.5).

Рис. 3.5. Поле Користувачі в мережі

Під цим полем знаходиться кнопка «Оновити», натиснувши яку, можна отримати оновлену інформацію про підключених користувачів.

При натисканні правою кнопкою миші по полю з’являється спливаюче меню, яке містить підпункти «Почати передачу» та «Закінчити передачу» (див. Рис. 3.6). Натиснувши «Почати передачу», користувач ініціює відеопередачу з користувачем зі списку. Натиснувши на пункт «Припинити передачу», користувач завершує відео.

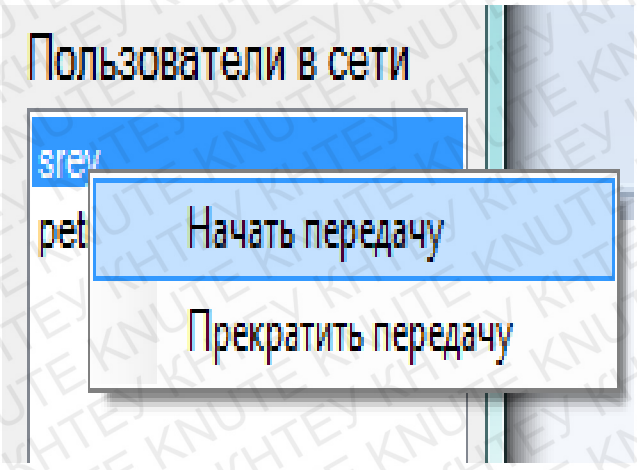


Рис. 3.6. Меню передачі відео

При передачі даних у центрі вікна знаходиться зображення, отримане від іншого користувача, в правому нижньому куті – зображення користувача, що отримує відео (див. рис.3.7).

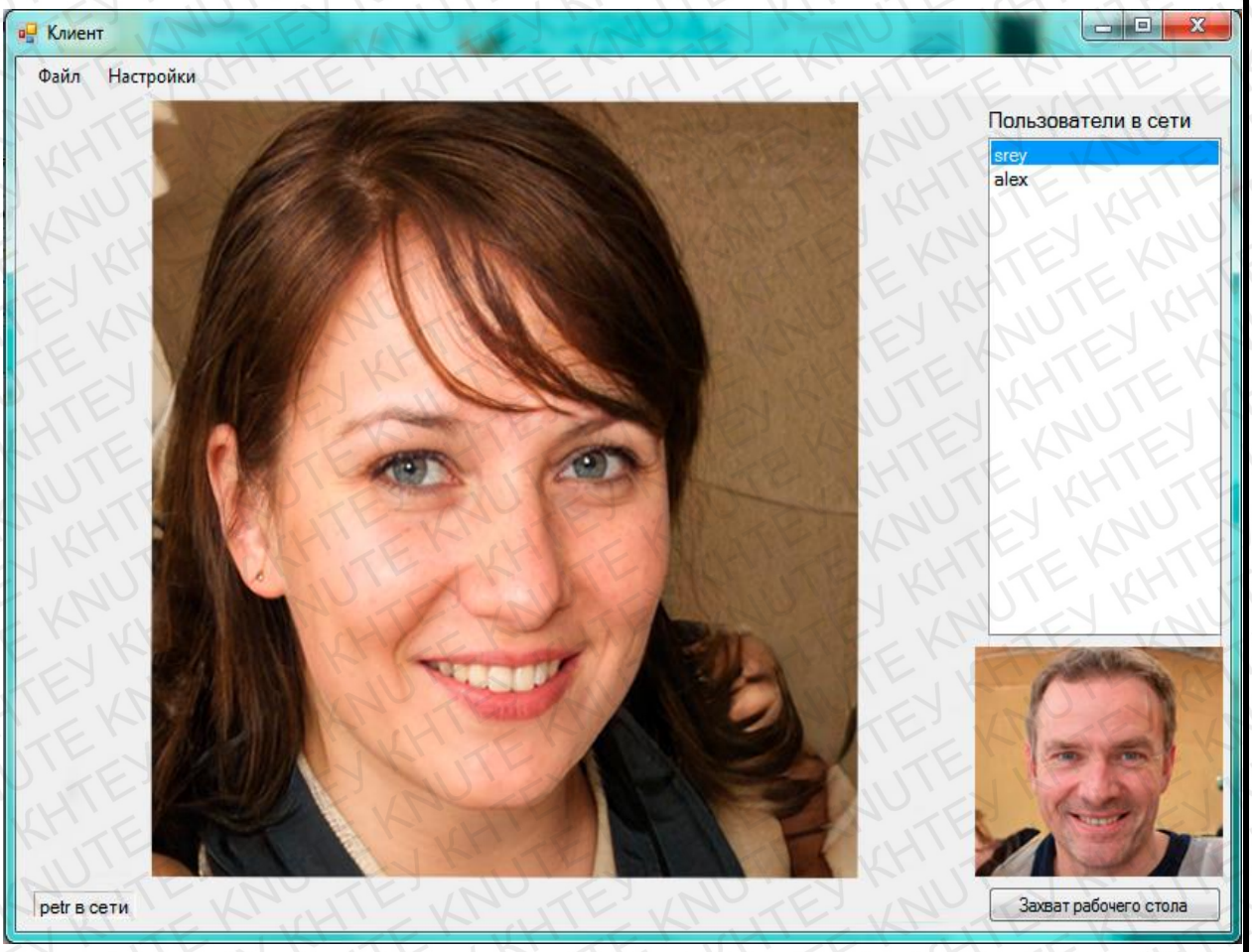


Рис. 3.7. Головне вікно клієнтської програми під час конференції

Додатковою опцією є можливість захоплення зображення з робочого столу та передача його користувачеві, з яким встановлено з'єднання (див. рис. 3.8).

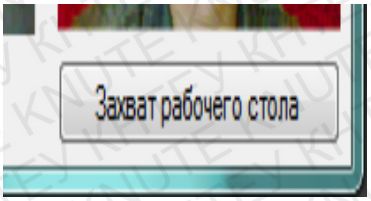


Рис. 3.8. Додаткові опції

Натиснувши кнопку «Захоплення робочого столу», користувач ініціює передачу зображення зі свого робочого стола співрозмовнику (див. рис. 3.9.). Щоб припинити захоплення робочого столу, необхідно натиснути кнопку «Припинити захоплення».

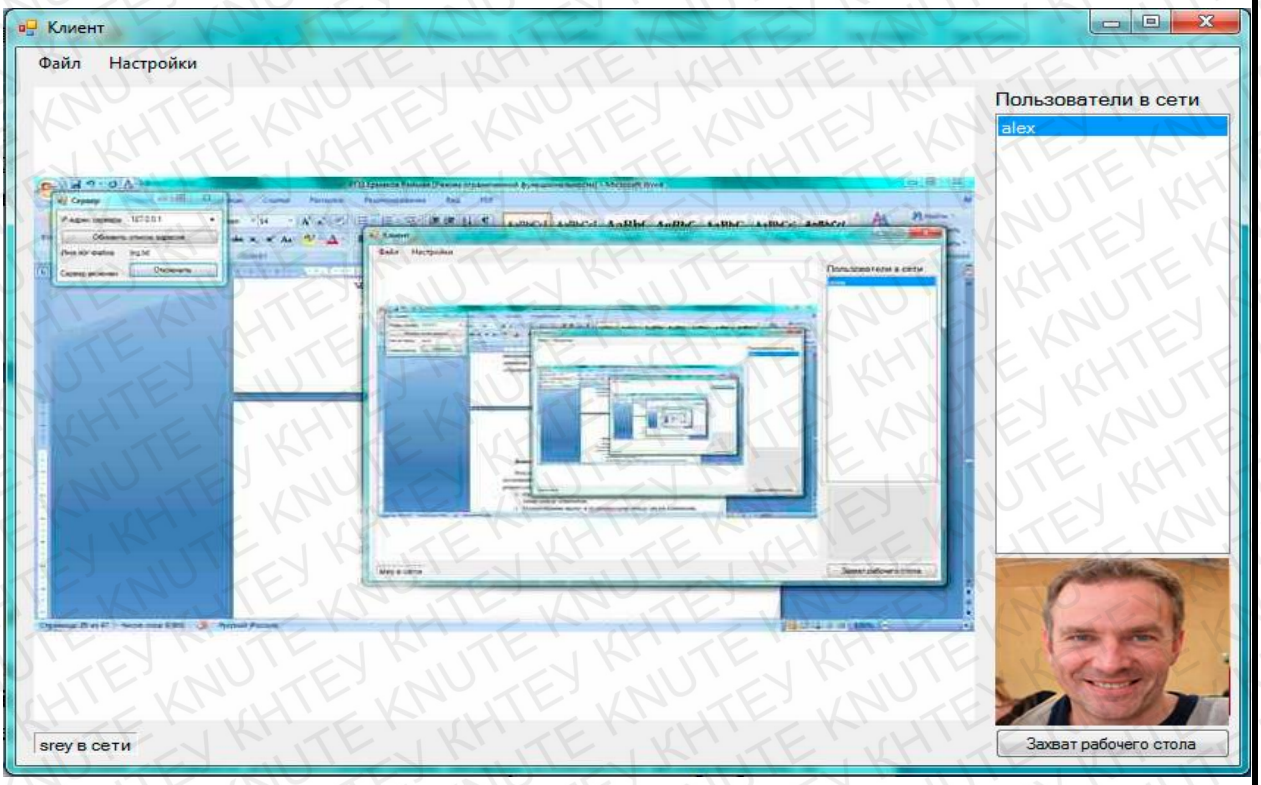


Рис. 3.9 Демонстрація зображення екрану

3.4 Висновки до розділу 3

У третьому розділі ми визначили, що для нормального функціонування програмного комплексу система повинна задовольняти такі вимоги: операційна система Windows XP, Windows Vista, Windows 7, Windows 10; встановлений .NET Framework 3.0; бібліотеки DirectShow та Voice; наявність веб-камери (вбудованої чи зовнішньої); наявність мікрофона (вбудованого чи зовнішнього) та зробили детальний опис інтерфейсу користувача.

ВИСНОВКИ ТА ПРОПОЗИЦІЇ

У ході моєї кваліфікаційної роботи було досліджено сферу застосування відеоконференцзв'язку, виявлено переваги та недоліки існуючих продуктів у цій сфері.

Ми провели теоретичне дослідження відеоконференцзв'язку та з'ясували, що це телекомунікаційна технологія інтерактивної взаємодії двох і більше віддалених абонентів, при якій між ними можливий обмін аудіо і відеоінформацією в реальному часі, з урахуванням передачі керуючих даних.

Враховуючи функції та цілі застосування, обладнання відеоконференцзв'язку систематизуються на категорії та класи. Такі, як індивідуальні, групові, галузеві, мобільні та адміністративні системи.

З'ясували, що для здійснення передачі даних між клієнтом і сервером використовується протокол, заснований на транспортному протоколі TCP, передача даних між клієнтами здійснюється за протоколом, який використовує як основу дейтаграмний протокол UDP.

Виконано проектування моделі клієнт – серверного додатку для проведення відеоконференцій.

На основі результатів аналізу було розроблено демо-версію мережевого додатка, що здійснює відеозв'язок між користувачами. Були виконані такі вимоги:

- Підтримка інтерфейсу користувача;
- Здійснення з'єднання локальної мережі між кількома користувачами;
- Організація відеозв'язку один на один (Peer-To-Peer);

					КНТЕУ 121– 02-16.МР			
Зм.	*Аркуш*	*№ докум.*	*Підпис*	*Дата*				
Зав. каф.		Криворучко О.В.		27.10.21	Модель клієнт-серверного програмного супроводу (адміністрування) відеоконференцій	*Стадія*	*Аркуш*	*Аркушів*
Керівник		Харченко О.А.		27.10.21		*ВП*	*39*	*42*
Гарант		Токар В.В.		27.10.21		*Факультет інформаційних технологій 2 курс, 2м група*		
Розробив		Шевченко Д.Р.		27.10.21	Висновки та пропозиції			

- Здійснення віддаленого доступу до робочого столу учасників відеоконференції.

Недоліками демо-версіямї є відсутність можливості відеозв’язку один до багатьох і відеозв’язку багато до багатьох. Ці недоліки легко усунути під час реалізації повнофункціональної версії програмного комплексу.

Устаткування, що використовується в таких мережах відеоконференцзв‘язку, зокрема, криптомаршрутизатори, повинні забезпечувати достатню смугу пропущення і підтримувати режим Qo (Quality of Service).

Відеоконференцзв’язок знаходить собі гідне застосування скрізь, де необхідні: оперативність в аналізі ситуації й ухваленні рішення; консультація фахівця або спільна робота в режимі віддаленого доступу над проектами і рішеннями.

Для подальшого розвитку програмного забезпечення рекомендується поліпшити інтерфейс, що зробить продукт більш зручним і привабливим для використання кінцевим користувачем.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Computer Networks/ Andrew S. Tanenbaum; Prentice-Hall 1996. – 813 pages.
2. IBM Documentation [Електронний ресурс]. - Режим доступу: https://www.ibm.com/docs/ru/aix/7.2?topic=management-transmission-control-protocolinternet-protocol – відкритий
3. Microsoft C# Documentation [Електронний ресурс]. - Режим доступу: https://docs.microsoft.com/en-us/dotnet/csharp/ – відкритий
4. Microsoft DirectShow Documentation[Електронний ресурс]. - Режим доступу: https://docs.microsoft.com/en-us/windows/win32/directshow/using-directshow – відкритий
5. Microsoft TCP/UDP Documentation [Електронний ресурс]. - Режим доступу: https://docs.microsoft.com/en-us/dotnet/framework/network-programming/tcp-udp – відкритий
6. Цифрова обробка аудіо та відеоінформації у мультимедійних системах: Навчальний посібник /О.В. Дробик, В.В. Кідалов, В.В. Коваль,Б.Я. Костік, В.С. Лазебний, Г.М. Розорінов, Г.О. Сукач. – К.: Науковадумка, 2008. – 144 с.
7. Протоколи ATM [Електронний ресурс]. – Режим доступу: https:// wiki.cuspu.edu.ua/index.php/Протоколи_ATM – відкритий
8. Newman S./ Building microservices: designing fine-grained systems. – " O'Reilly Media, Inc.", 2015.

					КНТЕУ 121– 02-16.МР			
Зм.	*Аркуш*	*№ докум.*	*Підпис*	*Дата*				
Зав. каф.		Криворучко О.В.		27.02.202	Модель клієнт-серверного програмного супроводу (адміністрування) відеоконференцій	*Стадія*	*Аркуш*	*Аркушів*
Керівник		Харченко О.А.		27.02.202		*СВД*	*41*	*42*
Гарант		Токар В.В.		27.02.202		*Факультет інформаційних технологій 2 курс, 2м група*		
Розробив		Шевченко Д.Р.		27.02.202	*Список використаних джерел*			

9. Furukawa, Y. Web-based control application using WebSocket. / Y. Furukawa, [Електронний Ресурс] European Synchrotron Radiation Facility ESRF, 38 Grenoble (France); 1423 p;ISSN 2226-0358; Worldcat; 2012; p. 695-697. Режим доступу до ресурсу : http://www.iaea.org/INIS/contacts – відкритий.
10. RFC-6455. – Fette, I. and A. Melnikov, "The WebSocket Protocol", RFC 6455,December 2011. Режим доступу до ресурсу: https://tools.ietf.org/html/rfc6455 – відкритий.
11. Berson, A. Client - server architecture. / Berson, Alex. / New York, NY : McGraw-Hill, 1992. - 452 p.
12. Loreto S. Known Issues and Best Practices for the Use of Long Polling and Streaming in Bidirectional HTTP. [Електронний Ресурс]. / G. Wilkins, S. Salsano, P. Saint-Andre. Режим доступу до ресурсу: Available: https://tools.ietf.org/html/rfc6202#page-16 – відкритий.
13. RFC-6455. – Fette, I. and A. Melnikov, "The WebSocket Protocol", RFC 6455, December 2011. Режим доступу до ресурсу: https://tools.ietf.org/html/rfc6455 – відкритий.
14. Апаратне забезпечення персонального комп'ютера [Електронний ресурс]. – Режим доступу: https://kafinfo.org.ua/files/Informatyka_10_11/Glava_2_5.pdf – відкритий.
15. История развития устройств ввода и вывода [Електронний ресурс]. - Режим доступу: http://vvod-vivod.blogspot.com/p/blog-page_3386.html – відкритий.
16. Системи відеоконференцзв'язку: призначення та загальна характеристика. URL: http://hrytsakangelina.blogspot.com/2012/12/blog-post.html?m=1

ТЕХНІЧНЕ ЗАВДАННЯ

У ході дослідження предметної галузі та існуючих продуктів для реалізації відеоконференції було поставлено такі завдання для реалізації проекту:

- Підтримка інтерфейсу користувача;
- Здійснення з'єднання локальної мережі між кількома користувачами;
- Організація відеозв'язку один на один (Peer-To-Peer);
- Організація відеомовлення (відеозв'язок одним-багатьом);
- Організація відеоконференції багато-багатьом;
- Здійснення віддаленого доступу до робочого столу учасників відеоконференції;
- Можливість запису отриманого відеопотоку у файл.
- Було виявлено такі вимоги, що накладаються на програмний комплекс:
- Архітектура програмного комплексу повинна відповідати гібридній архітектурі, що з'єднує архітектури «клієнт-сервер» та «крапка-крапка»;
- Мережна взаємодія повинна здійснюватися за протоколом, що базується на TCP (клієнт-серверна взаємодія) та на UDP;
- Для захоплення відео та аудіо необхідно використовувати бібліотеку DirectShow;
- Структура програмного комплексу повинна відповідати описаним у пункті 2.3

					КНТЕУ 121– 02-16.МР			
Зм.	*Аркуш*	*№ докум.*	*Підпис*	*Дата*				
Зав. каф.		Криворучко О.В.		20.03.202	Модель клієнт-серверного програмного супроводу (адміністрування) відеоконференцій	*Стадія*	*Аркуш*	*Аркушів*
Керівник		Харченко О.А.		20.03.202		*ТЗ*	*43*	*42*
Гарант		Токар В.В.		20.03.202		*Факультет інформаційних технологій 2 курс, 2м група*		
Розробив		Шевченко Д.Р.		20.03.202	*Технічне завдання*			

ДОДАТКИ

ДОДАТОК А

Код класу Server

```
class Server
    {
    // клас сервер - містить адресу і порт сервера, сокет прийому даних, головний потік,
    // Список зареєстрованих та активних користувачів, список з'єднань

    private Socket socket;
    private IPAddress address;
    private int port;
    private Thread thread;
    private List<ClientRecord> registered_clients;
    private List<ClientRecord> active_clients;
    private List<Connection> connections;
    LogWriter log_writer;
    private void AcceptConnections()
    {
        while (true)
        {
            // Приймаємо з'єднання

            Socket t_socket = socket.Accept();
            Connection connection = new Connection();
            connection.Socket = t_socket;
            //connection.stream = новий NetworkStream(t_socket);
            log_writer.Write("New connection from " +
            t_socket.RemoteEndPoint.ToString());
```

```
                // Створюємо потік отримання даних

                connection.Thread = New Thread(ProcessConnection);
                connection.Thread.IsBackground = true;
                connection.Thread.Start(connection);
                // Зберігаємо з'єднання

                lock (connections) connections.Add(connection);
        }
}

private void ProcessConnection(object state)
{

        // обробляємо запит у з’єднанні

        Connection connection = (Connection)state;
        byte[] buffer = new byte[255];
        try
        {
                while (true)
                {

                        // читаємо тип повідомлення
                        byte b = NetworkReader.ReadType(connection.Socket);
                        switch (b)
                        {// залежно від типу повідомлення викликаємо відповідну
                        функцію
                                case (byte) Messages.Registration:
                                {
                                Registration(connection);
                                break;
                                }
                                case (byte) Messages.Authorization:
```

```
                                {
                                Authorization(connection);
                                break;
                                }
                                case (byte) Messages.ClientInfo:

                                {
                                ClientInfo(connection);
                                break;
                                }
                                case (byte) Messages.InfoForClient:
                                {
                                RequestToClient(connection);
                                break;
                                }
                                case (byte) Messages.Disconnect:
                                {
                                DisconnectClient(connection);
                                return;
                                }
                        }
                }
        }
        catch (SocketException exc)
        {
                MessageBox.Show(exc.Message);
                log_writer.Write("SocketException:" + exc.Message);
        }
        catch (Exception exc)
        {
                MessageBox.Show(exc.Message);
                log_writer.Write("Exception:" + exc.Message);
        }
        finally
        {
```

```
            DisconnectClient(connection);
        }
        }


        void Registration(Connection conn)
        {
        // Реєстрація нового користувача
            ClientRecord rec = NetworkReader.ReadClientRecord(conn.Socket);
            ClientInfo info = NetworkReader.ReadClientInfo(conn.Socket);
            log_writer.Write("Registration of new user with name <" + rec.Name
            + "> and password <" + rec.Password + ">");
            //перевірка на існуючого користувача
            foreach (ClientRecord record in registered_clients)
            {
                if (record.Name == rec.Name)
                {

                    // отправляем сообщение о неудачной регистрации

                    SendMessage(conn, Messages.RegistrationFailed);
                    log_writer.Write("Registration of new user with name <" +
                    rec.Name + "> and password <" + rec.Password + "> failed");

                    // удаляем соединение

                    conn.Socket.Close();
                    lock (connections) connections.Remove(conn);
                    return;
                }
            }
            //інакше – додаємо інформацію про користувача у список
            registered_clients.Add(rec);
            active_clients.Add(rec);
            // и отправляем сообщение о удачной регистрации
```

```
            SendMessage(conn, Messages.RegistrationSuccess);
            log_writer.Write("Registration of new user with name <" + rec.Name +
            "> and password <" + rec.Password + "> success");
            conn.Name = rec.Name;

            // відсилаємо усім клієнтам інформацію про нового користувача

            foreach (Connection c in connections)
            {
                    if (c != conn)
                    {
            SendMessage(c, Messages.ClientConnected);
            c.SendClientInfo(info);
                    }
            }
    }
    void Authorization(Connection conn)
    {

            // авторизація
            ClientRecord rec = NetworkReader.ReadClientRecord(conn.Socket);
            ClientInfo info = NetworkReader.ReadClientInfo(conn.Socket);
            log_writer.Write("Authorization of new user with name <" + rec.Name +
            "> and password <" + rec.Password + ">");

            // перевірка, чи є користувач онлайн

            foreach (ClientRecord record in active_clients)
            {
                    if (record.Equals(rec) == true)
                    {
                    // відсилаємо повідомлення, про існуючого користувача
                    SendMessage(conn, Messages.ClientAllredyConnect);
                    log_writer.Write("User with name <" + rec.Name + "> and password <" +
            rec.Password + "> allredy on-line");
```

```
                // видаляємо з’єднання
                conn.Socket.Close();
                lock (connections) connections.Remove(conn);
                return;
                }
        }
        // перевірка на користувача з таким іменем
        foreach (ClientRecord record in registered_clients)
        {
                if (record.Equals(rec) == true)
                {
                        // якщо зареєстрований – відправляємо повідомлення про
                успішну авторизацію
                        // та додаємо інформацію про користувача у список
                        active_clients.Add(rec);
                        SendMessage(conn, Messages.AuthorizationSuccess);
                        log_writer.Write("Authorization of new user with name <" +
                rec.Name + "> and password <" + rec.Password + "> success");
                        conn.Name = rec.Name;
                        // відсилаємо усім користувачам інформацію про нового
                клієнта
                        foreach (Connection c in connections)
                        {
                                if (c != conn)
                                {
                                SendMessage(c, Messages.ClientConnected);
                                c.SendClientInfo(info);
                                }
                        }
                        conn.Name = rec.Name;
                        return;
                }
        }

        // інакше – відправляємо інформацію про невдалу авторизацію
```

```
            SendMessage(conn, Messages.AuthorizationFailed);
            log_writer.Write("Authorization of new user with name <" + rec.Name +
            "> and password <" + rec.Password + "> failed");
            conn.Socket.Close();
            lock (connections) connections.Remove(conn);
        }


        void ClientInfo(Connection connection)
        {

            // відправляємо клієнту інформацію щодо кількості зареєстрованих
        користувачів
            NetworkWriter.WriteType(connection.Socket,
            (byte)(connections.Count - 1));
            log_writer.Write("Request from user with name <" +
            connection.Name + "> for information about connected users");
            foreach (Connection conn in connections)
            {
            //if (conn != connection)
            // відправляємо інформацію про кожне з'єднання
            //connection.SendClientInfo(conn);
            }
        }
        void RequestToClient(Connection conn)
        {
            String name = NetworkReader.ReadString(conn.Socket);
            byte type = NetworkReader.ReadType(conn.Socket);
            ClientInfo info = NetworkReader.ReadClientInfo(conn.Socket);
            foreach (Connection c in connections)
            {
                    if (c.Name == name)
                    {
                            NetworkWriter.WriteType(c.Socket, type);
                            NetworkWriter.WriteClientInfo(c.Socket, info);
```

```
                    log_writer.Write("new message from user '" +
                    conn.Name + "' to user '" + c.Name + "'");
                    break;
                }
            }
        }


        void DisconnectClient(Connection conn)
        {
        // відключення клієнта
            // видаляємо користувача із списка
            foreach (ClientRecord rec in active_clients)
            {
                if (rec.Name == conn.Name)
                {
                    active_clients.Remove(rec);
                    log_writer.Write("User with name <" + conn.Name +
                    "> disconnected");
                    break;
                }
            }

            // усім клієнтам відправляємо інформацію про відключення користувача
            foreach (Connection c in connections)
            {
                if (c != conn)
                {
                    SendMessage(c, Messages.ClientDisconnected);
                    IPEndPoint ep = new IPEndPoint(IPAddress.Loopback, 10000);
                    ClientInfo info = new ClientInfo(conn.Name,
                    ep.Address.ToString(), ep.Port);
                    c.SendClientInfo(info);
                }
            }
```

```
            // видаляємо з'єднання
            conn.Socket.Close();
            lock (connections) connections.Remove(conn);
        }

        void SendMessage(Connection conn, Messages mes)
        {
            NetworkWriter.WriteType(conn.Socket, (byte)mes);
        }
}
```

Код класу Connection

```
class Connection
{
        // клас з'єднання – містить ім'я, потік, сокет, потік для передачі данних
        public String Name;
        public Socket Socket;
        public Thread Thread;
        public void SendClientInfo(ClientInfo info)
        {
            // відправка інформації про з'єднання – ім'я, адрес та порт
            NetworkWriter.WriteClientInfo(Socket, info);
        }
        public void SendRequest(byte type, String address, int port)
        {
            NetworkWriter.WriteType(Socket, type);
            NetworkWriter.WriteClientInfo(Socket,
            new ClientInfo(Name, address, port));
        }
}
```

Код класу LogWriter

```
class LogWriter
```

ПРОДОВЖЕННЯ ДОДАТКУ А

```
{
        String filename;
        public LogWriter(String name)
        {
                filename = name;
        }
        public void Write(String s)
        {
                StreamWriter wr = new StreamWriter(filename, true);
                wr.WriteLine(DateTime.Now.ToString() + " : " + s + "\n");
                wr.Close();
        }
}
```

Код класу Client

```
class Client
{
	// клас клієнт – містить ім’я та пароль користувача,
	// сокет TCP та UDP, потік для передачі данних

	String name;
	String password;
	Socket tcp_socket;
	Thread thread;
	VideoReceiver receiver;
	VideoSender sender;
	List<ClientInfo> connected_clients;
	public List<String> ConnectedUsers;
	public EmptyHandler ClientConnectedHandler;
	public EmptyHandler ClientDisconnectedHandler;
	public RequestHandler TransferRequestHandler;
	public MessageHandler TransferFailedHandler;
	public String Name
	{
		get { return name; }
		set { name = value; }
	}
	public String Password
	{
		get { return password; }
		set { password = value; }
	}
	public Client(String host, String n, String p, bool reg,
	PictureBox pb, VideoDescriptor desc)
```

```
{

        // задаємо змінні для початкових данних
        name = n;
        password = p;
        connected_clients = new List<ClientInfo>();
        ConnectedUsers = new List<String>();

        // створюємо сокет та приєднуємся до сервера

        tcp_socket = new Socket(AddressFamily.InterNetwork,
        SocketType.Stream, ProtocolType.Tcp);
        tcp_socket.Connect(host, 11000);

        // ініціалізуємо відправника та отримувача відео
        int port = ((IPEndPoint)tcp_socket.LocalEndPoint).Port - 10000;
        IPAddress addr = ((IPEndPoint)tcp_socket.LocalEndPoint).Address;
        sender = new VideoSender(desc, addr, port);
        receiver = new VideoReceiver(pb, addr, port + 1);

        //викликаємо функцію авторизації

        Authorization(reg);
        //SetConnectedClients();

        // потік для обробки повідомлень
        thread = new Thread(ReadMessage);
        thread.IsBackground = true;
        thread.Start();
}
void Authorization(bool register)
{
// авторизація
        byte b;
```

```
        if (register == true)
            b = (byte)Messages.Registration;
        else
            b = (byte)Messages.Authorization;

        // відправляємо тип повідомлення та інформацію про користувача
        NetworkWriter.WriteType(tcp_socket, b);
        NetworkWriter.WriteClientRecord(tcp_socket,
                                        new ClientRecord(name, password));
        NetworkWriter.WriteClientInfo(tcp_socket, new ClientInfo(name,
            sender.GetEndPoint().Address.ToString(), sender.GetEndPoint().Port));

        // очікуємо відповідь

        b = NetworkReader.ReadType(tcp_socket);
        ProcessMessage(b);
    }

    void ReadMessage()
    {
        while (true)
        {
            // читаємо тип повідомлення та виконуємо відповідні дії
            byte b = NetworkReader.ReadType(tcp_socket);
            ProcessMessage(b);
        }
    }
    void ProcessMessage(byte type)
    {
        switch (type)
        {
        case (byte)Messages.RegistrationFailed:
            {
                throw (new Exception("Користувач з таким іменем вже
            зареєстрований"));
```

```
            }
        case (byte)Messages.RegistrationSuccess:
            {
                break;
            }
        case (byte)Messages.AuthorizationFailed:
            {
            throw (new Exception("Користувач з таким іменем вже зареєстрований
"));
            }
        case (byte)Messages.AuthorizationSuccess:
            {
                break;
            }
        case (byte)Messages.ClientAllredyConnect:
            {
                throw (new Exception("Пользователь с таким именем уже в
            сети"));
            }
        case (byte)Messages.ClientInfo:
            {
                ClientInfo();
                break;
            }
        case (byte)Messages.ClientConnected:
            {
                NewClientConnected();
                break;
            }
        case (byte)Messages.ClientDisconnected:
            {
                ClientDisconnected();
                break;
            }
        case (byte)Messages.TransferRequest:
```

```
                {
                    TransferAnswer();
                    break;
                }
            case (byte)Messages.TransferFailed:
                {
                    TransferFail();
                    break;
                }
            case (byte)Messages.TransferSuccess:
                {
                    BeginVideoTranslation();
                    break;
                }
            case (byte)Messages.TransferEnd:
                {
                    EndVideoTranslation();
                    break;
                }
            }
        }
        void NewClientConnected()
        {

            // підключення нового користувача
            // зчитуємо інформацію про нового клієнта та додаємо її до списків
            ClientInfo info = NetworkReader.ReadClientInfo(tcp_socket);
            ConnectedUsers.Add(info.Name);
            connected_clients.Add(info);
            NetworkWriter.WriteType(tcp_socket, (byte)Messages.InfoForClient);
            NetworkWriter.WriteString(tcp_socket, info.Name);
            NetworkWriter.WriteType(tcp_socket, (byte)Messages.ClientInfo);
            NetworkWriter.WriteClientInfo(tcp_socket, new ClientInfo(name,
            sender.GetEndPoint().Address.ToString(), sender.GetEndPoint().Port));
```

```
        // викликаємо подію, щоб відобразити нового користувача у вікні програми

        ClientConnectedHandler();
    }
    void ClientInfo()
    {
        ClientInfo info = NetworkReader.ReadClientInfo(tcp_socket);
        ConnectedUsers.Add(info.Name);
        connected_clients.Add(info);

        // викликаємо подію, щоб відобразити нового користувача у вікні програми
        ClientConnectedHandler();
    }
    void ClientDisconnected()
    {
        // відключення користувача
        // читаємо інформацію про користувача та видаляємо її зі списків
        ClientInfo info = NetworkReader.ReadClientInfo(tcp_socket);
        ConnectedUsers.Remove(info.Name);
        foreach (ClientInfo inf in connected_clients)
        {
            if (inf.Name == info.Name)
            {
                connected_clients.Remove(info);
                break;
            }
        }
        // викликаємо подію, щоб оновити список користувачів у вікні програми
        ClientDisconnectedHandler();
    }
    void SetConnectedClients()
    {
        // отримання списку підключених користувачів
        List<String> list = new List<String>();
```

```
            // надсилаємо запит
            NetworkWriter.WriteType(tcp_socket, (byte)Messages.ClientInfo);

            // зчитуємо кількість підключених користувачів
            int clients_count = (int)NetworkReader.ReadType(tcp_socket);

            // и информацию о каждом из них
            for (int i = 0; i < clients_count; i++)
            {
                ClientInfo info = NetworkReader.ReadClientInfo(tcp_socket);
                ConnectedUsers.Add(info.Name);
                connected_clients.Add(info);
            }
        }
        public void TransferRequest(String user_name)
        {
            // надсилаємо користувачеві з ім'ям user_name запит на з'єднання
            NetworkWriter.WriteType(tcp_socket, (byte)Messages.InfoForClient);
            NetworkWriter.WriteString(tcp_socket, user_name);
            NetworkWriter.WriteType(tcp_socket, (byte)Messages.TransferRequest);
            NetworkWriter.WriteClientInfo(tcp_socket, new ClientInfo(name,
            sender.GetEndPoint().Address.ToString(), sender.GetEndPoint().Port));
        }
        void TransferAnswer()
        {
            // надсилаємо відповідь користувачу та зчитуємо його дані
            ClientInfo info = NetworkReader.ReadClientInfo(tcp_socket);

            // виводимо на екран діалогове вікно
            if (TransferRequestHandler(info.Name) == true)
            {
            // надсилаємо користувачеві згоду на відеозв'язок
                NetworkWriter.WriteType(tcp_socket, (byte)Messages.InfoForClient);
                NetworkWriter.WriteString(tcp_socket, info.Name);
                NetworkWriter.WriteType(tcp_socket, (byte)Messages.TransferSuccess);
```

```
                NetworkWriter.WriteClientInfo(tcp_socket, new ClientInfo(name,
                receiver.GetEndPoint().Address.ToString(), receiver.GetEndPoint().Port));

                // пов'язуємо сокет із віддаленим сокетом
                //receiver.Connect(info.Point);
                sender.Connect(info.Point);

                // і починаємо передачу відео
                BeginVideoTranslation1();
            }
            else
            {
                // відправляємо відмову у відеозв'язку
                NetworkWriter.WriteType(tcp_socket, (byte)Messages.InfoForClient);
                NetworkWriter.WriteString(tcp_socket, info.Name);
                NetworkWriter.WriteType(tcp_socket, (byte)Messages.TransferFailed);
                NetworkWriter.WriteClientInfo(tcp_socket, new ClientInfo(name,
                sender.GetEndPoint().Address.ToString(), sender.GetEndPoint().Port));
            }
        }
        void TransferFail()
        {
            // отримавши відмову у відеозв'язку виводимо відповідне повідомлення на
            екран
            ClientInfo info = NetworkReader.ReadClientInfo(tcp_socket);
            TransferFailedHandler(info.Name);
        }
        public void TransferEnd(String user_name)
        {
            // надсилаємо користувачеві повідомлення про завершення відеозв'язку
            NetworkWriter.WriteType(tcp_socket, (byte)Messages.InfoForClient);
            NetworkWriter.WriteString(tcp_socket, user_name);
            NetworkWriter.WriteType(tcp_socket, (byte)Messages.TransferEnd);
            NetworkWriter.WriteClientInfo(tcp_socket, new ClientInfo(name,
            sender.GetEndPoint().Address.ToString(), sender.GetEndPoint().Port));
```

```
            // та закінчуємо відеопередачу
            EndVideoTranslation1();
        }

        public void Close()
        {

            // надсилаємо повідомлення про завершення з'єднання
            NetworkWriter.WriteType(tcp_socket, (byte)Messages.Disconnect);

            // та відключаємо сокет від сервера
            tcp_socket.Disconnect(false);
            receiver = null;
            sender = null;
        }
        public void BeginVideoTranslation()
        {
            ClientInfo info = NetworkReader.ReadClientInfo(tcp_socket);
            receiver.Connect(info.Point);
            sender.Connect(info.Point);
            receiver.BeginReceiving();
            sender.BeginSending();
        }
        void BeginVideoTranslation1()
        {
            receiver.BeginReceiving();
            sender.BeginSending();
        }
        public void EndVideoTranslation()
        {
        ClientInfo info = NetworkReader.ReadClientInfo(tcp_socket);
            receiver.EndReceiving();
            receiver = null;
            sender.EndSending();
```

```
            sender = null;
        }
        public void EndVideoTranslation1()
        {
            receiver.EndReceiving();
            receiver = null;
            sender.EndSending();
            sender = null;
        }
}
```

Код класу Capture

```
class Capture : ISampleGrabberCB, IDisposable
{

// Побудова інтерфейсу графа
private IFilterGraph2 FilterGraph = null;
private IMediaControl mediaCtrl = null;
// Чи можемо ми чекати на закінчення асинхронної роботи
private ManualResetEvent PictureReady = null;
// Встановлення асинхронності для захоплення зображення
private volatile bool bGotOne = false;
// Статус графу
private bool bRunning = false;
// Розміри зображення
private IntPtr handle = IntPtr.Zero;
private int videoWidth;
private int videoHeight;
private int stride;
public int Dropped = 0;
[DllImport("Kernel32.dll", EntryPoint = "RtlMoveMemory")]
private static extern void CopyMemory(IntPtr Destination, IntPtr Source, int Length);
public void Dispose()
```

```
{
        CloseInterfaces();
        if (PictureReady != null)
        {
                PictureReady.Close();
                PictureReady = null;
        }
}

// Деструктор
~Capture()
{
        Dispose();
}
public int Width
{
        get { return videoWidth; }
}
public int Height
{
        get { return videoHeight; }
}
public int Stride
{
        get { return stride; }
}

// Захоплення наступного зображення
public IntPtr GetBitMap()
{
        handle = Marshal.AllocCoTaskMem(stride * videoHeight);
        try
        {
                PictureReady.Reset();
                bGotOne = false;
```

```
                // запускаємо граф
                Start();

                // і починаємо чекати зображення
                if (!PictureReady.WaitOne(5000, false))
                {
                        throw new Exception("Timeout waiting to get picture");
                }
        }
        catch
        {
                Marshal.FreeCoTaskMem(handle);
                throw;
        }

        // повертаємо покажчик на зображення
        return handle;
}

// Початок захоплення
public void Start()
{
        if (!bRunning)
        {
                int hr = mediaCtrl.Run();
                DsError.ThrowExceptionForHR(hr);
                bRunning = true;
        }
}

// Призупинення (пауза) захоплення

public void Pause()
{
```

```
        if (bRunning)
        {
            int hr = mediaCtrl.Pause();
            DsError.ThrowExceptionForHR(hr);
            bRunning = false;
        }
}

// Загальний конструктор класу Capture
public Capture(VideoDescriptor desc)
{
        DsDevice[] capDevices;
        // Отримання всіх пристроїв захоплення
        capDevices = DsDevice.GetDevicesOfCat(FilterCategory.VideoInputDevice);
        if (desc.VideoDevice + 1 > capDevices.Length)
        {
            throw new Exception("Пристрій захоплення не знайдено!");
        }
        try
        {
            // Виклик функції встановлення параметрів захоплення
            SetupGraph(capDevices[desc.VideoDevice], desc.FrameRate, desc.VideoWidth,
            desc.VideoHeight);

            // Виклик CallBack для ігнорування нових зображень
            PictureReady = new ManualResetEvent(false);
            bGotOne = true;
            bRunning = false;
        }
        catch
        {
            Dispose();
            throw;
            }
}
```

ПРОДОВЖЕННЯ ДОДАТКУ Б

```
// Ініціалізація Графа
private void SetupGraph(DsDevice dev, int iFrameRate, int iWidth, int iHeight)
{
    int hr;
    ISampleGrabber sampGrabber = null;
    IBaseFilter capFilter = null;
    ICaptureGraphBuilder2 capGraph = null;

    // Отримання об'єкта graphbuilder
    FilterGraph = (IFilterGraph2)new FilterGraph();
    mediaCtrl = FilterGraph as IMediaControl;
    try
    {

        // Отримання ICaptureGraphBuilder2
        capGraph = (ICaptureGraphBuilder2)new CaptureGraphBuilder2();

        // Отримання інтерфейсу SampleGrabber
        sampGrabber = (ISampleGrabber)new SampleGrabber();

        // Початок побудови Графа
        hr = capGraph.SetFiltergraph(FilterGraph);
        DsError.ThrowExceptionForHR(hr);

        // Додавання в Граф пристрій відеозахоплення
        hr = FilterGraph.AddSourceFilterForMoniker(dev.Mon, null, "Video input", out
        capFilter);
        DsError.ThrowExceptionForHR(hr);
        IBaseFilter baseGrabFlt = (IBaseFilter)sampGrabber;
        ConfigureSampleGrabber(sampGrabber);

        // Додавання до Графа фільтра-захоплення
        hr = FilterGraph.AddFilter(baseGrabFlt, "Ds.NET Grabber");
        DsError.ThrowExceptionForHR(hr);
```

```
            if (iFrameRate + iHeight + iWidth > 0)
            {
                SetConfigParms(capGraph, capFilter, iFrameRate, iWidth, iHeight);
            }
            hr = capGraph.RenderStream(PinCategory.Capture, MediaType.Video, capFilter,
            null, baseGrabFlt);
            DsError.ThrowExceptionForHR(hr);
            SaveSizeInfo(sampGrabber);
        }
        finally
        {
            if (capFilter != null)
            {
                Marshal.ReleaseComObject(capFilter);
                capFilter = null;
            }
            if (sampGrabber != null)
            {
                Marshal.ReleaseComObject(sampGrabber);
                sampGrabber = null;
            }
            if (capGraph != null)
            {
                Marshal.ReleaseComObject(capGraph);
                capGraph = null;
            }
        }
}
private void SaveSizeInfo(ISampleGrabber sampGrabber)
{
        int hr;
        // Отримання типу інформації з SampleGrabber
        AMMediaType media = new AMMediaType();
        hr = sampGrabber.GetConnectedMediaType(media);
        DsError.ThrowExceptionForHR(hr);
```

```
        if ((media.formatType != FormatType.VideoInfo) || (media.formatPtr == IntPtr.Zero))
        {
                throw new NotSupportedException("Unknown Grabber Media Format");
        }

        // Розмір інформації, що захоплюється
        VideoInfoHeader videoInfoHeader =
        (VideoInfoHeader)Marshal.PtrToStructure(media.formatPtr, typeof(VideoInfoHeader));
        videoWidth = videoInfoHeader.BmiHeader.Width;
        videoHeight = videoInfoHeader.BmiHeader.Height;
        stride = videoWidth * (videoInfoHeader.BmiHeader.BitCount / 8);
        DsUtils.FreeAMMediaType(media);
        media = null;
}

private void ConfigureSampleGrabber(ISampleGrabber sampGrabber)
{
        AMMediaType media;
        int hr;
        // встановлюємо тип відео
        media = new AMMediaType();
        media.majorType = MediaType.Video;
        media.subType = MediaSubType.RGB24;
        media.formatType = FormatType.VideoInfo;
        hr = sampGrabber.SetMediaType(media);
        DsError.ThrowExceptionForHR(hr);
        DsUtils.FreeAMMediaType(media);
        media = null;

        // конфігуруємо SampleGrabber
        hr = sampGrabber.SetCallback(this, 1);
        DsError.ThrowExceptionForHR(hr);
}
// Встановлення параметрів відео
```

ПРОДОВЖЕННЯ ДОДАТКУ Б

```
private void SetConfigParms(ICaptureGraphBuilder2 capGraph, IBaseFilter capFilter, int iFrameRate, int iWidth, int iHeight)
{
    int hr;
    object o;
    AMMediaType media;

    // Знаходження потоку конфігурації інтерфейсу
    hr = capGraph.FindInterface(PinCategory.Capture, MediaType.Video, capFilter, typeof(IAMStreamConfig).GUID, out o);
    IAMStreamConfig videoStreamConfig = o as IAMStreamConfig;
    if (videoStreamConfig == null)
    {
        throw new Exception("Failed to get IAMStreamConfig");
    }

    // Отримання блоку чинного формату
    hr = videoStreamConfig.GetFormat(out media);
    DsError.ThrowExceptionForHR(hr);

    // Копіювання з videoinfoheader
    VideoInfoHeader v = new VideoInfoHeader();
    Marshal.PtrToStructure(media.formatPtr, v);

    // Встановлення частоти кадрів
    if (iFrameRate > 0)
    {
        v.AvgTimePerFrame = 10000000 / iFrameRate;
    }

    // Встановлення ширини
    if (iWidth > 0)
    {
        v.BmiHeader.Width = iWidth;
    }
```

```
        // Встановлення висоти
        if (iHeight > 0)
        {
                v.BmiHeader.Height = iHeight;
        }

        //Зворотне копіювання структури інформації
        Marshal.StructureToPtr(v, media.formatPtr, false);

        // Встановлення нового формату
        hr = videoStreamConfig.SetFormat(media);
        DsError.ThrowExceptionForHR(hr);
        DsUtils.FreeAMMediaType(media);
        media = null;
}

private void CloseInterfaces()
{
        int hr;
        try
        {
                if (mediaCtrl != null)
                {
                        // Зупинка захоплення
                        hr = mediaCtrl.Stop();
                        bRunning = false;
                }
        }
        catch (Exception ex)
        {
                Debug.WriteLine(ex);
        }
        if (FilterGraph != null)
        {
```

```
            Marshal.ReleaseComObject(FilterGraph);
            FilterGraph = null;
        }
}

int ISampleGrabberCB.SampleCB(double SampleTime, IMediaSample pSample)
{
        if (!bGotOne)
        {
            // Встановлення bGotOne для запобігання виклику, у той час що, отримуємо
            нове зображення
            bGotOne = true;
            IntPtr pBuffer;
            pSample.GetPointer(out pBuffer);
            int iBufferLen = pSample.GetSize();
            if (pSample.GetSize() > stride * videoHeight)
        {
            throw new Exception("Buffer is wrong size");
        }
        CopyMemory(handle, pBuffer, stride * videoHeight);

        // Зображення готове
        PictureReady.Set();
        }
        Marshal.ReleaseComObject(pSample);
        return 0;
}

int ISampleGrabberCB.BufferCB(double SampleTime, IntPtr pBuffer, int BufferLen)
{
        if (!bGotOne)
        {
            // Перевірка розміру буфера
            if (BufferLen <= stride * videoHeight)
            {
```

ПРОДОВЖЕННЯ ДОДАТКУ Б

```
                    // Копіювання зображення в Буфер
                    CopyMemory(handle, pBuffer, stride * videoHeight);
                }
                else
                {
                    throw new Exception("Buffer is wrong size");
                }
                //Встановлення bGotOne для запобігання виклику поки що отримуємо нове
            зображення
                bGotOne = true;

                // Зображення готове
                PictureReady.Set();
            }
            else
            {
                Dropped++;
            }
            return 0;
            }
}
```