НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО»

ФАКУЛЬТЕТ ПРИКЛАДНОЇ МАТЕМАТИКИ

Кафедра системного програмування і спеціалізованих комп‘ютерних систем

«До захисту допущено»

Завідувач кафедри

_____ Романкевич О.В.
(підпис)

“___” червня 2020 р.

Дипломний проєкт

на здобуття ступеня бакалавра

зі спеціальності **123 «Комп'ютерна інженерія»**

на тему: Багатопотоковий клієнт-серверний додаток.

Виконав: студент IV курсу, групи КВ-61
Снопок Артем Ігорович

(підпис)

Керівник доц.каф. СПІСКС, к.т.н. Павловський В.І.

(підпис)

Консультант з нормоконтролю, доц.каф.СПСКС, к.т.н. Клятченко Я.М.

(підпис)

Рецензент_____

_____
(посада, науковий ступінь, вчене звання, науковий ступінь, прізвище та ініціали) (підпис)

Засвідчую, що у цьому дипломному проєкті немає запозичень з праць інших авторів без відповідних посилань.
Студент _____
(підпис)

Київ – 2020 року

АНОТАЦІЯ

Мета дипломного проєкту дослідити і структурувати теоретичні відомості необхідні створення багатопотокового додатку, з архітектурою клієнт-сервер і на основі отриманих знань, розробити власний додаток.

В курсовому проєкті наведено основні теоретичні відомості щодо поняття архітектури клієнт-сервер, її особливостей та модифікацій. Зокрема, особлива увага в проєкті звертається на опис різних рівнів клієнт-серверних архітектур.

У результаті роботі було розроблено багатопотоковий додаток, з використанням оптимальних наборів та даних для його створення.

Всі компоненти програми розроблені мовою програмування java. Даний вибір середовища розробки додатку, дозволяє створювати програми та модулі, сумісні з більшістю сучасних операційних систем. Також результати дипломної роботи можуть бути використані для розробки нових архітектур, або вдосконалення цієї.

Ключові слова: АРХІТЕКТУРА КЛІЄНТ-СЕРВЕР, МОДЕЛЬ OSI, ТОНКИЙ КЛІЄНТ, СОКЕТ, ТОВСТИЙ КЛІЄНТ, ПОРТ, IP АДРЕСА, JAVA, БАГАТОПОТОКОВІСТЬ.

ANNOTATION

The purpose of the thesis project is to explore and structure the theoretical information needed to create a multithreaded application, with a client-server architecture and based on the knowledge gained, to develop your own application.

The course project provides basic theoretical information on the concept of client-server architecture, its features and modifications. In particular, special attention in the project is paid to the description of different levels of client-server architectures.

As a result, a multithreaded application was developed, using optimal sets and data to create it.

All components of the program are developed in the java programming language. This choice of application development environment allows you to create programs and modules that are compatible with most modern operating systems. Also, the results of the thesis can be used to develop new architectures, or improve this one.

Keywords: CLIENT-SERVER ARCHITECTURE, OSI MODEL, THIN CLIENT, SOCKET, THICK CLIENT, PORT, IP ADDRESS, JAVA, MULTITHREADING.

Поз.	Формат	ПОЗНАЧЕННЯ	НАЙМЕНУВАННЯ	Кількість аркушів	№ прим.	Примітки
	А4	ІАЛЦ. 045430.004 ПЗ	Багатопотоковий	50		
			клієнт-серверний додаток			
			Пояснювальна записка			
	А4	ІАЛЦ. 045430.005 Д1	Взаємодія модулів в	1		
			архітектурі			
			Схема структурна			
	А4	ІАЛЦ.045430.006 Д2	Взаємодія модулів в	1		
			архітектурі клієнт-сервер			
			Схема структурна			
	А4	ІАЛЦ.045430.007 Д3	Схема підключення	1		
			Клієнту до серверу			
			Схема алгоритму			

Змін.	Арк.	№ докум.	Підпис	Дата	ІАЛЦ.467500.003 ТП		
Розробив		Снопок А.І.			*“Багатопотоковий клієнт-серверний додаток “*	Літ.	Аркуш 1 / Аркушів 2
Перевірив		Павловський В.І.					
Консульт.					**Відомість технічного проекту**	КПІ ім. Ігоря Сікорського, ФПМ КВ-51	
Н. контроль		Клятченко Я.М.					
Зав. каф.		Романкевич В.О.					

Поз.	Формат	ПОЗНАЧЕННЯ	НАЙМЕНУВАННЯ	Кількість аркушів	№ прим.	Примітки
	А4	ІАЛЦ.045430.008 Д4	Взаємодія клієнту та	1		
			серверу			
			Схема алгоритму			

Змін.	Арк.	№ докум.	Підпис	Дата	ІАЛЦ.467500.003 ТП	Арк. 2

ЗМІСТ

1. НАЙМЕНУВАННЯ ТА ГАЛУЗЬ РОЗРОБКИ 2

2. ПІДСТАВА ДЛЯ РОЗРОБКИ 2

3. МЕТА І ПРИЗНАЧЕННЯ РОБОТИ 2

4. ДЖЕРЕЛА РОЗРОБКИ 2

5. ТЕХНІЧНІ ВИМОГИ 2

5.1 Вимоги до програмного продукту, що розробляється 2

5.2 Вимоги до апаратного забезпечення 3

5.3 Вимоги до програмного та апаратного забезпечення користувача 3

6. ЕТАПИ РОЗРОБКИ 4

1. НАЙМЕНУВАННЯ ТА ГАЛУЗЬ РОЗРОБКИ

Назва розробки: «Багатопотоковий клієнт-серверний додаток ».

Галузь застосування: створення додатку з архітектурою клієнт-сервер.

2. ПІДСТАВА ДЛЯ РОЗРОБКИ

Підставою для розробки є завдання на виконання роботи першого (бакалаврського) рівня вищої освіти, затверджене кафедрою системного програмування і спеціалізованих комп'ютерних систем Національного технічного університету України «Київський політехнічний інститут імені Ігоря Сікорського».

3. МЕТА І ПРИЗНАЧЕННЯ РОБОТИ

Метою даного проєкту проєкту дослідження і структуруваня теоретичних відомостей необхідних для створення багатопотокового додатку, з архітектурою клієнт-сервер і на основі отриманих знань, розробити власний додаток, мовою високого рівня java.

4. ДЖЕРЕЛА РОЗРОБКИ

Джерелом інформації є технічна та науково-технічна література, технічна документація, публікації в періодичних виданнях та електронні статті у мережі Інтернет.

5. ТЕХНІЧНІ ВИМОГИ

5.1 Вимоги до програмного продукту, що розробляється

- сумісність з популярними операційними системами;
- можливість підключення клієнта до сервера;
- можливість роботи сервера без клієнта;

- можливість отримання інформації від сервера;
- можливість припинення роботи як клієнта так і сервера;
- прив'язка ip адреси та порта до сокету;
- зберігання встановленого зв’язку клієнту та сервера.

5.2 Вимоги до апаратного забезпечення

- Об'єм оперативної пам'яті: 2 Gb або більше;
- Процесор другого покоління і вище;
- Жорсткий диск SATA 2 або SSD диск..

5.3 Вимоги до програмного та апаратного забезпечення користувача

- Процесор другого покоління і вище;
- Об'єм оперативної пам'яті: 2 Gb або більше;

6. ЕТАПИ РОЗРОБКИ

№ з/п	Назва етапів виконання дипломного проекту	Термін виконання етапів
1.	Видача завдання на дипломне проектування	04.11.2019
2.	Вивчення літератури за тематикою роботи	25.11.2019
3.	Розроблення та узгодження технічного завдання	18.12.2019
4.	Розроблення структури додатку	23.01.2020
5.	Розроблення дизайну та графічних елементів	16.02.2020
6.	Програмна реалізація додатку	26.03.2020
7.	Тестування додатку	15.04.2020
8.	Підготовка матеріалів текстової частини проекту	24.04.2020
9.	Підготовка матеріалів графічної частини проекту	10.05.2020
10.	Оформлення технічної документації проекту	18.05.2020

Поз.	Формат	ПОЗНАЧЕННЯ	НАЙМЕНУВАННЯ	Кількість аркушів	№ прим.	Примітки
	А4	ІАЛЦ. 045430.002 ТЗ	Багатопотоковий	4		
			клієнт-серверний додаток			
			Технічне завдання			
	А4	ІАЛЦ. 045430.003 ТП	Багатопотоковий	2		
			клієнт-серверний додаток			
			Відомість технічного			
			проекту			
	А4	ІАЛЦ. 045430.004 ПЗ	Багатопотоковий	50		
			клієнт-серверний додаток			
			Пояснювальна записка			
	А4	ІАЛЦ. 045430.005 Д1	Взаємодія модулів в	1		
			архітектурі			
			Схема структурна			

Змін.	Арк.	№ докум.	Підпис	Дата	ІАЛЦ.467500.003 ТП		
Розробив		Снопок А.І.			*“Багатопотоковий клієнт-серверний додаток “* **Відомість технічного проекту**	Літ.	Аркуш 1 / Аркушів 2
Перевірив		Павловський В.І.					
Консульт.						КПІ ім. Ігоря Сікорського, ФПМ КВ-51	
Н. контроль		Клятченко Я.М.					
Зав. каф.		Романкевич В.О.					

Поз.	Формат	ПОЗНАЧЕННЯ	НАЙМЕНУВАННЯ	Кількість аркушів	№ прим.	Примітки
	А4	ІАЛЦ.045430.006 Д2	Взаємодія модулів в	1		
			архітектурі клієнт-сервер			
			Схема структурна			
	А4	ІАЛЦ.045430.007 Д3	Схема підключення	1		
			Клієнту до серверу			
			Схема алгоритму			
	А4	ІАЛЦ.045430.008 Д4	Взаємодія клієнту та	1		
			серверу			
			Схема алгоритму			

Змін.	Арк.	№ докум.	Підпис	Дата	ІАЛЦ.467500.003 ТП	Арк. 2

ЗМІСТ

ПЕРЕЛІК ТЕРМІНІВ, СКОРОЧЕНЬ ТА ПОЗНАЧЕНЬ 2
ВСТУП 3
1. КЛІЄНТ-СЕРВЕРНА АРХІТЕКТУРА 5
1.1 Опис архітектури 5
1.2 Принцип роботи 7
2. РІЗНОВИД КЛІЄНТ-СЕРВЕРНОЇ АРХІТЕКТУРИ 11
2.1 Однорівнева архітектура 11
2.2 Архітектура з виділеним сервером 12
2.3 Дворівнева архітектура 14
2.4 Багаторівнева архітектура 17
3. ТЕХНОЛОГІЇ КЛІЄНТ-СЕРВЕР 20
3.1 АНАЛІЗ ІСНУЮЧИХ ТЕХНОЛОГІЙ 20
3.2 Тонкий клієнт 22
3.3 Товстий клієнт 26
3.4 Модель клієнт-сервер в мережі 28
3.5 Багатопотоковість 31
4. ОПИС МОДУЛІВ 35
4.1 Модуль реалізації зв’язку 35
4.2 Модуль сервера 41
4.3 Модуль клієнта 43
5 ТЕСТУВАННЯ ПРОГРАМИ ТА АНАЛІЗ РЕЗУЛЬТАТІВ 45
5.1 Тестування роботи 45
ВИСНОВКИ 48
СПИСОК ВИКОРИСТАНОЇ ЛІТЕРАТУРИ 49

ПЕРЕЛІК ТЕРМІНІВ, СКОРОЧЕНЬ ТА ПОЗНАЧЕНЬ

TCP (Transmission Control Protocol) – протокол транспортного рівня передачі даних в моделі OSI.

БД – база даних.

ПЗ – програмне забезпечення.

HTTP (Hyper Text Transfer Protocol) – протокол прикладного рівня передачі гіпертекстових документів в моделі OSI.

XML (Extensible Markup Language) –мова розмітки для обміну між різними застосунками.

SQL (structured query language) – мова структурованих запитів, який використовується для управління даними в реляційних базах даних.

ВСТУП

Область застосування інформаційних систем постійно збільшується , а самі вони стають все більш складнішими. З часом, ресурсів одного комп'ютера почало не вистачати на реалізації ідей і механізмів які почали збільшуватися та розростатися в геометричній прогресії.

Саме для того щоб забезпечити максимальну користь та продуктивність інформаційних систем, була придумана архітектурна схема клієнт-сервер, що допомогла вирішити велику кількість проблем на той період.

Для становлення в сучасну архітектуру, клієнт-серверна взаємодія пройшла довгий шлях, починаючи з централізованої системи архітектури додатків, які були популярні в 70-х роках минулого століття. Час йшов і таки системи перейшли на новий рівень, рівень персональних електронних обчислювальних машин, та локальні завдання на цих машинах. З розвитком персональних машин, перейшли на новий ступінь, локальні обчислювальні мережі з розробленою архітектурою файл-сервер. Так і з'явились перші однорангові мережі, а з розвитком цих мереж почало перше розділення обчислювальних машин на клієнтів і сервери. І з розвитком технологій та їх вдосконаленням була створена архітектура клієнт-сервер яка займає високе положення і в теперешній час.

Але в теперішній час, такі системи зростають і ускладняються настільки, що набувають глобального характеру, і від їх правильного і надійного функціонування починає залежати діяльність великої кількості людей. Такі системи часто мають дуже складну архітектуру, яка складається з великого набору компонентів, кожен з яких виконується на окремому пристрої.

Оскільки число таких систем постійно зростає, вимоги до них, теж зростають. Складність проектування і розробки таких систем вимагає великої

кількості часу та сил, а методи і засоби, що застосовуються при реалізації таких проектів, відмінні від розробки типових систем.

1. КЛІЄНТ-СЕРВЕРНА АРХІТЕКТУРА

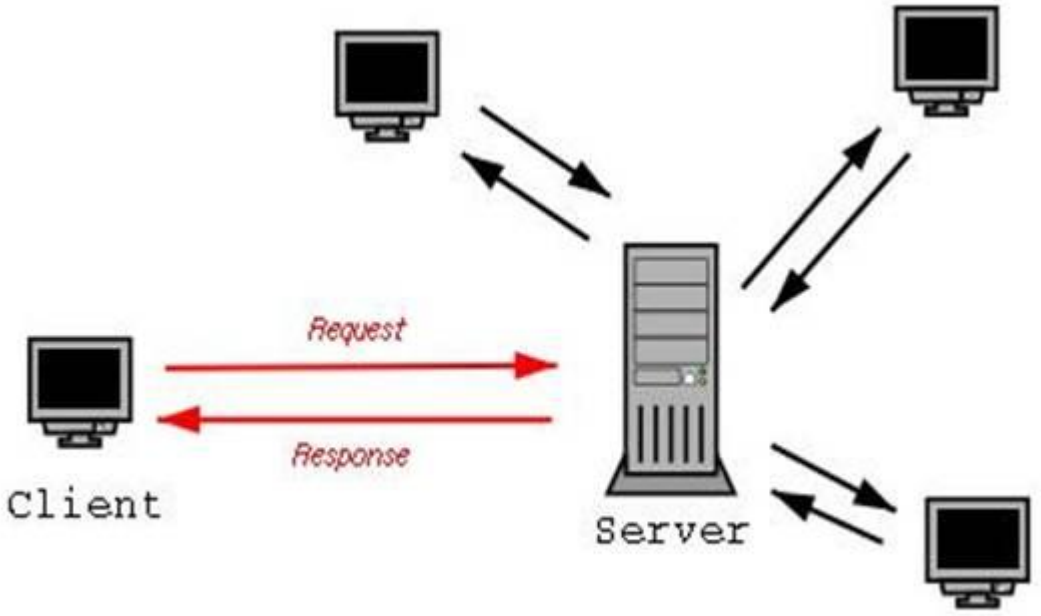
1.1 Опис архітектури

Термін "клієнт-сервер" означає такий програмний комплекс архітектури, в якій його функціональні частини взаємодіють за простою схемою, клієнт дає запит, сервер дає відповідь. Якщо розглянути кожну взаємодіючу частину цього комплексу, то одна з них, а саме клієнт, виконує активну роботу, тобто формує деякі запити, а інша, сервер, відповідає на них у пасивному режимі. По мірі розвитку інформаційних систем, ці ролі можуть змінюватися, наприклад буде можливість розробки блоку, буде одночасно виконувати як функції серверу, так і функції клієнту по відношенню до інших блоків.

Зауважимо, для будь-якої інформаційної системи необхіддно мати щонайменше три основні функціональні блоки, це модулі зберігання даних, модулі їх обробки та відповідного інтерфейсу для діалогу з користувачем. Можна реалізувати будь яку з цих частин незважаючи на дві інші частини. Можна навести приклад, що при сталому коду програм, що використовуються для обробки та збереження даних, маємо можливість зміни інтерфейсу з клієнтом таким чином, що одна й та сама інформація, буде відображатися у вигляді інших представленням даних, наприклад у вигляді графіків, у вигляді таблиць або в гістограммах. Не змінюючи програм представлення даних та їх збереження, можна змінювати програми для обробки даних, наприклад змінивши алгоритм роботи програми, наприклад для пошуку тексту. І нарешті, не змінюючи програм представлення і обробки даних, можна змінити програмне забезпечення для зберігання даних, перейшовши, наприклад, на іншу файлову систему.

У класичній архітектурі клієнт-сервер доводиться розподіляти три основні частини програми по двом фізичним модулів. Зазвичай за зберігання

даних розташовується на інформаційному сервері, наприклад деякий сервер БД, інтерфейс з користувачем - на стороні клієнта, а обробку даних розподіляється між клієнтськими і серверними частинами. (рис. 1.1).



“Рис. 1.1 - Взаємодія клієнт-сервер”

Для того щоб розбити алгоритми даних необхідно синхронізовізувати поведінку всіх частин в системі. У всіх розробників повинна бути повна інформація про нові зміни, які внесені в систему, щоб зрозуміти їхв подальшому. При нехтуванні цих операцій можна наштовхнутися на великі труднощі при розробці клієнт-серверних систем, при їх встановленні або супроводі. Оскільки потрібно брати до уваги роботу дій різних груп розробників. Коли розробники не доходять до фінального результату, виникають суперечності, які гальмують роботу та розвиток системи. Це змушує їх змінювати вже готові та перевірені елементи системи.

1.2 Принцип роботи

Модель клієнт-сервер була предметом безлічі дебатів і суперечок. Один з головних питань полягав у тому, як точно розділити клієнта і сервера. Розглядаючи безліч додатків типу клієнт-сервер, призначених для організації доступу користувачів до баз даних, і в рамках багаторівневого подання систем для обчислення, можна виділити три групи функцій, орієнтованих на рішення різних підзадач: [5]

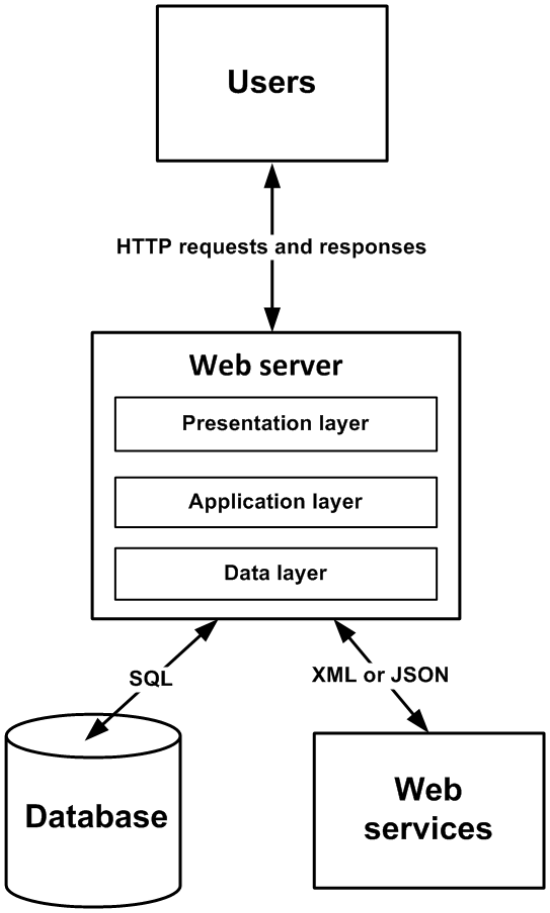
- функції введення і відображення даних, які забезпечують взаємодію з користувачем;
- прикладні функції, бізнес логіка;
- функції управління ресурсами.

Виконання цих функцій в основному забезпечується програмними засобами, які можна представити у вигляді взаємопов'язаних компонентів (рис. 1.2), де:

- компонент представлення відповідає за користувальницький інтерфейс (presentation layer);
- прикладний компонент реалізовує алгоритм вирішення конкретного поставленого завдання (application layer);
- компонент управління ресурсами, забезпечує доступ до необхідних даних та ресурсів (data layer).

Автономна система, тобто комп'ютер, який не підключений до мережі, представляє всі ці компоненти як на різних рівнях, так і на рівні додатків. Так само і мережу

вона представляє всі ці компоненти, але, в загальному випадку, розподілені між вузлами. Завдання зводиться до забезпечення мережевої взаємодії між цими компонентами.[4]



“ Рис. 1.2 - Компоненти клієнт-серверної архітектури”

Архітектура клієнт-сервер визначає загальні принципи роботи та взаємодії вузлів в мережі, де існують сервери, вузли які використовуються для доставки деяких специфічних функцій і клієнти, які споживають ці функції.

Рівень користувальницького інтерфейсу зазвичай реалізується на клієнтських пристроях. На цьому рівні знаходяться програми, з використанням яких, власне користувач може мати змогу комунікувати з додатком. Досить різною може бути складність додатків, які входять до користувальського інтерфейсу.

Найлегший варіант програми, яка входить до користульського інтерфейсу, коли вона не містить нічого окрім дисплея на якому виводиться результат. Такі інтерфейси зазвичай використовуються при роботі з великими серверами. У тому випадку, коли цей сервер може контролювати кожну

взаємодію, включаючи роботу клієнта з клавіатурою, а також монітором, але в цьому випадку ми навряд чи зможемо говорити про модель взаємодії клієнт-сервер. Але все ж, у багатьох випадках, термінали клієнтів, можуть виробляти деяку обробку даних, локально, здійснюючи, наприклад, віддалений друк рядків які друкуються, або надаючи інтерфейси деяких форм, в яких можна зробити маніпуляції, ще до до їх відправлення на сервер.

Бізнес-логіка - це сукупність правил, принципів і залежностей поведінки об'єктів предметної області системи. Синонімом даного поняття є логіка предметної області. Бізнес-логіка - це реалізація предметної області в інформаційній системі, наприклад, бухгалтерського обліку, навчання студентів, тощо. У багаторівневих інформаційних системах цей рівень взаємодіє з нижнім рівнем інфраструктурних сервісів, наприклад, інтерфейсом до бази даних або файлової системи і вищерозміщеним рівнем сервісів додатки який вже, в свою чергу, взаємодіє з рівнем призначеного для користувача інтерфейсу (User Interface Layer) або зовнішніми системами.

У фазі бізнес-моделювання та розробки вимог бізнес-логіка може описуватися у вигляді тексту, концептуальних аналітичних моделей предметної області, бізнес-правил, різноманітних алгоритмів, діаграм діяльності, графів і діаграм переходу станів, моделей бізнес-процесів. У фазі аналізу і проектування системи бізнес-логіка втілюється в класах і методах класів, в разі використання об'єктно-орієнтованих мов програмування, або процедур і функцій, в разі застосування процедурних мов.

Рівень інформації даних у моделі клієнт-серверу включає в себе програми, за допомогою який можна обробляти дані в додатках . Своєрідною вимогою для цього рівня є збереження даних. Але, якщо програма не буде працювати, інформація буде збережена у визначеному для неї місці з розрахунком на подальше її користування.

У простішій варіації, рівень даних стане файловою системою, але нерідко для їх реалізації використовується повномасштабна база даних. У

макеті клієнт-сервер рівень інформації в основному знаходиться на стороні сервера.Крім простого зберігання інформації рівень даних звичайно відповідає за підтримка цілісності даних для різних додатків. Для бази даних підтримання специфічні метадані додатків, також зберігаються на цьому рівні.

Зазвичай в діловому середовищі рівень даних організовується у формі реляційної бази даних. Ключовим тут є незалежність даних. Дані організовуються незалежно від додатків так, щоб зміни в організації даних не впливали на додатки, а додатки не чинили впливу на організацію даних. Використання реляційних баз даних у моделі клієнт-сервер допомагає нам відокремити рівень обробки від рівня даних, розглядаючи обробку і дані незалежно один від одного.[12]Проте існує великий клас програм, для яких реляційні бази даних не є найкращим вибором. Характерною рисою таких програм є робота зі складними типами даних, які простіше моделювати в поняттях об'єктів, а не відносин. Також і мультимедійних систем значно простіше працювати з відео і аудіо потоками, використовуючи специфічні для них операції, ніж з моделями цих потоків у вигляді реляційних таблиць. У тих випадках, коли операції з даними значно простіше виразити в поняттях роботи з об'єктами, має сенс реалізувати рівень даних засобами об'єктно орієнтованих баз даних. Подібні бази даних не тільки підтримують організацію складних даних у формі об'єктів, але і зберігають реалізації операцій над цими об'єктами. Таким чином, частина функціональності, припадала на рівень обробки, мігрує в цьому випадку на рівень даних.

Функції які виконуються в середовищі клієнт-сервер

Клієнт	Сервер
Керує користувальським інтерфейсом	Приймає і оброблює запити до бази даних зі сторони клієнтів
Приймає і перевіряє запити введені користувачем	Перевіряє повноваження користувачів
Генерує запит до бази даних і відправляє його серверу	Виконує запити і відправляє результат клієнту
Відображає отримані дані користувачу	Підтримка системного каталогу Керування відновленням даних
Виконує додаток	Гарантує цілісніть

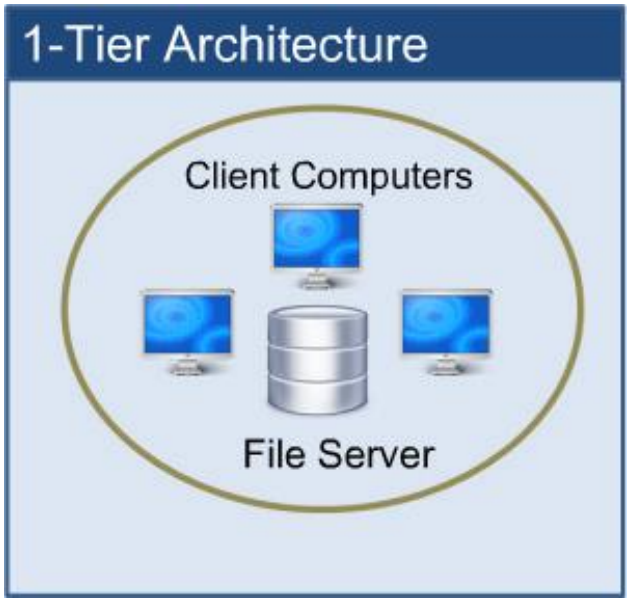
“Таблиця 1 Функціональність між клієнтом та сервером”

2. РІЗНОВИД КЛІЄНТ-СЕРВЕРНОЇ АРХІТЕКТУРИ

2.1 Однорівнева архітектура

Різниця між однорівневої, дворівневої і трирівневої архітектури залежить від того, яким чином поділяються на частини ці компоненти. В однорівневій архітектурі вони всі є частинами однієї програми. У дворівневій архітектурі ці компоненти розділені на дві окремі частини. У трирівневої архітектурі компоненти розділені на три окремі частини.

Найбільш простим і вперше використаним на практиці варіантом клієнт-серверної архітектури є так звана однорівнева архітектура (рис. 2.1). У даній архітектурі клієнт здійснює тільки відображення інформації, що надається сервером, який несе всю обчислювальну навантаження.



“Рис. 2.1 - Однорівнева архітектура клієнт-сервер”

Як приклад даної архітектури можна розглядати доступ з терміналу до віддаленого сервера або віддалений робочий стіл. Централізовану архітектуру можна розглядати як окремий випадок однорівневої клієнт-серверної архітектури.

2.2 Архітектура з виділеним сервером

Якщо до мережі підключено більше 10 користувачів, то однорангова мережа, де комп'ютери виступають у ролі і клієнтів, і серверів, може виявитися недостатньо продуктивною. Тому більшість мереж використовує виділені сервери. Виділених називається такий комп'ютер, який функціонує тільки як сервер. Вони спеціально оптимізовані для швидкої обробки запитів від мережних клієнтів і для керування файлами і каталогами. Мережі на основі серверів в даний час є промисловим стандартом.

Ці мережі спираються на спеціалізовані комп'ютери, що називаються серверами, що представляють собою централізовані сховища мережевих ресурсів і об'єднуючими централізоване забезпечення безпеки і управління доступом. На відміну від мереж з виділеним сервером, однорангові мережі не

мають централізованого забезпечення безпеки і управління. Сервер являє собою поєднання спеціалізованого програмного забезпечення та обладнання, яке надає служби в мережі для інших клієнтських комп'ютерів або інших процесів.[11]

Щоб реалізувати мережу з виділеним сервером, який буде включати в себе централізоване управління мережевими ресурсами використовуючи мережевої безпеки та управління завдяки установці і настройці сервера. Якщо брати до уваги обладнання, то у серверних комп'ютерів, як правило, мають більший центральний процесор, великі жорсткі диски та додаткові периферійні пристрої, більший об’єм пам’яті. Наприклад: приводи компакт-дисків та накопичувачі на магнітній стрічці, порівнюючи з клієнтськими машинами.

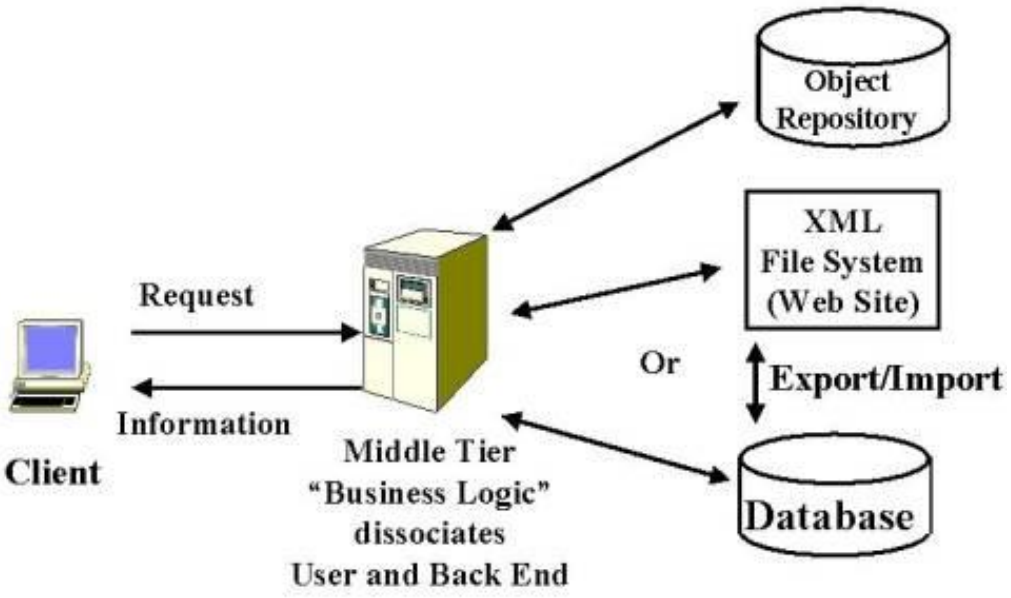
У тому числі сервери теж орієнтовані на обробку численних запитів на колективні ресурси ефективно ті швидко. Зазвичай вони використовуються для обслуговування мережних запитів клієнтів. До того ж, фізична безпека - доступ до самої машини і є основною складовою безпеки мережі. Тому важливо, щоб розташування серверів в спеціальному приміщенні можливо було контролювати у відокремленому приміщенні з загальним доступом.

Кожна мережа з виділеним сервером, також надають для перевірки, облікові записи своїх клієнтів, а також і їх паролів. Наприклад, одна з операційних систем віндовс, а саме NT, використовує задумку, через доменну реалізацію, для управління клієнтами, групами та машинами і для контролю над доступом до мережевих ресурсів. До того моменту, як користувач зможе мати змогу зайти до мережі інтернет, він повинен повідомити серверу, свій пароль та ім’я, для перевірки збігу імен і паролів облікових записів ,контролеру домену, який має перевірити їх у базі даних. Доступ до певних мережевих ресурсів буде наданий тільки у тому випадку, при збігу паролю та імені реєстрації, тоді контролер домену надасть права для цього. Керувати якоюсь зміною у цій процедурі, може тільки адміністратор цієї мережі. Цей

підхід набув популярності в мережах компаній, так як він забезпечує централізовану безпеку, також має можливість зміни доступу контролю і управління, в залежності від необхідності та важливості дій.

2.3 Дворівнева архітектура

У будь-якій сучасній мережі, яка використовує сучасні технології, присутні елементи клієнт-серверного взаємодії. З однорівневої архітектури, вона стає дворівневою, через те що між клієнтом та сервером необхідний бути розподіл трьох основних компонентів, взаємодію з користувачем, бізнес логіку та управління ресурсами(рис. 2.2).



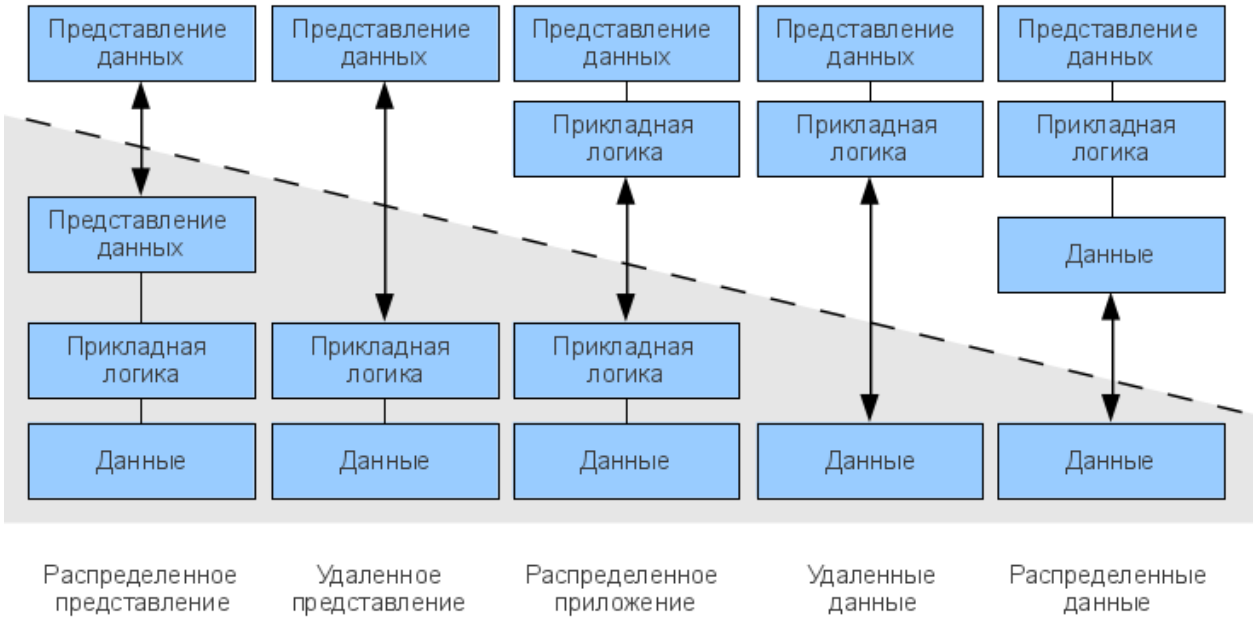
"Рис. 2.2 - Дворівнева архітектура клієнт-сервер"

Дворівнева архітектура використовується в клієнт-серверних додатках, де сервер відповідає на клієнтські запити безпосередньо і в повному обсязі, при цьому використовуючи тільки власні ресурси. Тобто сервер не викликає сторонні мережеві програми, але не звертається до сторонніх ресурсів для виконання будь-якої частини запиту. [13]

Розподілення компонентів з боку клієнта, або ж зі сторони сервера, визначає наступні моделі для їхньої взаємодії (Рис. 2.3) при додатку дворівневої архітектури:

- сервер для додатків – додаток з віддаленим доступом;
- сервер для терміналів - представлення даних в розподіленому вигляді;
- сервер БД - віддалене представлення даних;
- файл-сервер - доступ до віддаленої БД та віддалених файлових ресурсів.

Однією з перших моделей, з'явилася модель для розподіленого представлення інформації яка має назву, модель сервер-терміналів. Ця модель була реалізована на універсальній електронно обчислювальній машині і виступала в ролі сервера, і мала в своїй архітектурі спеціальні терміналами з інтерфейсом на основі цифрового відображення . На електронно обчислювальній машині виконувалась обробка даних та їх результат, завдяки введенню з клавіатури терміналу включаючи формування результату. І в результаті, повертала користувачеві на екран, правильно виконану роботу.



“Рис. 2.3 - Моделі клієнт-серверної взаємодії”

З появою ПК і невеликих мереж, була реалізована модель файлового сервера, який представляв доступ до файлових ресурсів і до віддаленої бази даних. У цьому випадку виділений вузол мережі є файловим сервером, на якому розміщені файли БД.[4] На клієнтських пристроях виконуються програми, в яких поєднані компонент представлення і прикладний компонент які використовують підключену базу даних, як файл який знаходить на тому самому пристрої. Протоколи для обміну інформацією при цьому предстають у вигляді низькорівневих викликів системи файлів.

На жаль, така модель була неефективною, через те, виникає велике навантаження на мережу, при активній роботі з таблицями баз даних. Рішенням, хоч і не повним, стала підтримка реплікації, розподілу таблиць і запитівза вузлами. У такому випадку, наприклад при зміні деяких даних, йде оновлення не всієї таблиці, а тільки частини, яка була модифікована.

З появою модифікованих систем управління БД, з'явилася можливість для реалізації ще однієї моделі, це модель доступу до віддалених БД, так звана модель сервера БД. При розгляді цього випадку, ядро такої БД функціонує на сервері, клієнтська програма на клієнті, а спеціальний протокол для обміну даними, який забезпечується з використанням SQL, мови запитів до баз даних,. Такий вдосконалений підхід, на відмінну від файлового сервера, призводить до значного зменшення навантаження в мережі та одноманітності інтерфейсу клієнт-серверної архітектури. Однак, навіть з цим, спостерігається значне підвищення трафіку у мережі, крім того, залишилася проблема для адміністрування таких додатків, оскільки в одному додатку, поєднуються велика кількість функції.

В рамках дворівневої архітектури, велику роль грає розташування компонентів, як на стороні клієнта, так і на стороні сервера, що і визначає основні моделі їх взаємодії. З вдосконаленням і впровадженням на рівні серверу БД, замість механізму збережених процедур з'явилася нова концепція,

яка називається активний сервер БД. У цій реалізації, частина функціональної реалізації прикладного компонента, виконана у вигляді збережених процедур, які виконуються на серверній стороні. Інша прикладна логіка виконується на стороні клієнта. Протокол взаємодії між ними — відповідний діалог мови SQL.

Переваги такого підходу:

- можливість централізованого адміністрування прикладного програмного інтерфейсу;
- зниження вартості системи за рахунок взяття сервера в тимчасовий обіг, а не його купівлі;
- значне зниження трафіку в мережеі, так як передаються виклики збережених процедур.

Недоліки такого підходу:

- адміністрування даної системи вимагає кваліфікованого професіонала;
- непрацездатність сервера може зробити непрацездатною всю обчислювальну мережу;
- бізнес логіка додатків залишається в клієнтському ПЗ;
- висока вартість обладнання.

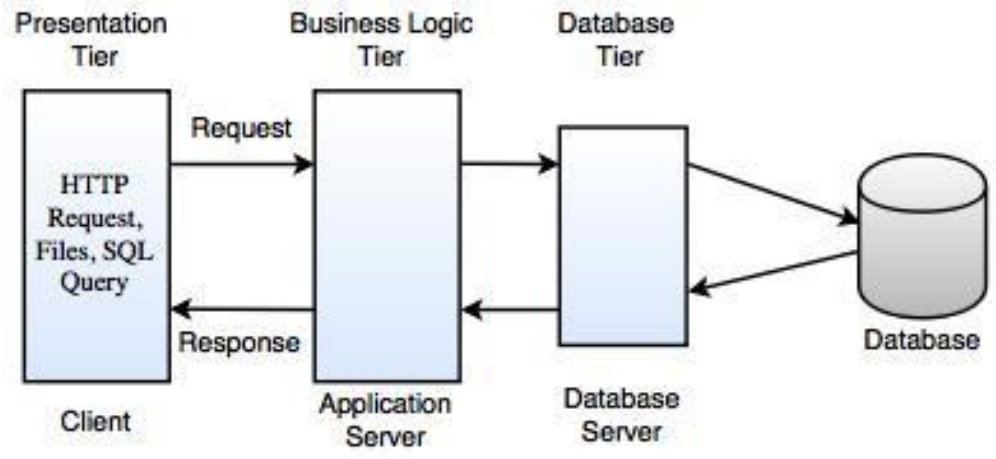
Так як реалізація прикладного компонента виконана на стороні сервера, це дає наступну модель, модель серверу додатків. Реалізація прикладного компонента на сервері, значно знижує вимоги до клієнтського ПЗ та спрощує його адміністрування, але вимагає підвищенних вимог до продуктивності та надійності сервера.

2.4 Багаторівнева архітектура

Багаторівнева архітектура різновид клієнт-серверної архітектури, в якої функція обробки даних виконується на одному або декількох окремих серверах. Це дозволяє здійснити розподіл функції для зберігання, обробки і показу даних для більш ефективного використання ресурсів, як серверів, так клієнтів.

Також, ще одна з популярних речей в клієнт-серверних технологіях, зв'язана з все більшим використанням систем для розподілених обчислень. Їх реалізація виконується на частині сервера, де додаток який знаходиться у мережі, розподілений на декілька або більше розділів, кожний з яких може реалізовуватись на окремих вузлах.

Серед багаторівневої архітектури клієнт-сервер найбільш поширена трирівнева архітектура. Виділені частини додатків взаємодіють один з одним, обмінюючись повідомленнями та даними в форматі, який реалізований заздалегідь. В цьому випадку дворівнева клієнт-серверна архітектура стає трирівневою (Рис. 2.4).



“Рис. 2.4 - Трирівнева архітектура клієнт-сервер”

Як правило, третім пунктом в трирівневій архітектурі виконує роль саме сервер додатків, тоді компоненти можуть розподіляються за наступною схемою :

- представлення даних виконується повністю на стороні клієнта;

- прикладний компонент виконується на виділеному сервері додатків ;
- управління ресурсами виконується на сервері БД, і надає необхідні дані для відповіді.

Трирівнева архітектура дозволяє ще більше стабілізувати навантаження на різні вузли та мережу, а також сприяє спеціалізації інструментів для розробки додатків і усуває недоліки дворівневої моделі клієнт-сервер. Централізація логіки додатка спрощує адміністрування і підтримку. Йде чітке розділення на платформи та інструменти для реалізації інтерфейсу і прикладної логіки, що дозволяє з максимальною ефективністю реалізовувати їх фахівцям вузького профілю. Масштабованість і розширення функціональності в трирівневій системі також не викликають особливих проблем - встановлення додаткового сервера додатків вирішує завдання для забезпечення нових функціональних вимог до системи. Таким чином, багаторівнева архітектура розподілених додатків дозволяє підвищити ефективність роботи корпоративної інформаційної системи та оптимізувати розподіл її програмно-апаратних ресурсів.

Однак вузьким місцем, як і в дворівневої клієнт-серверній архітектурі, залишаються підвищені вимоги до пропускної здатності мережі, що в свою чергу накладає жорсткі обмеження на використання таких систем в мережах з нестійкою зв'язком і малою пропускною здатністю.

Переваги такого підходу:

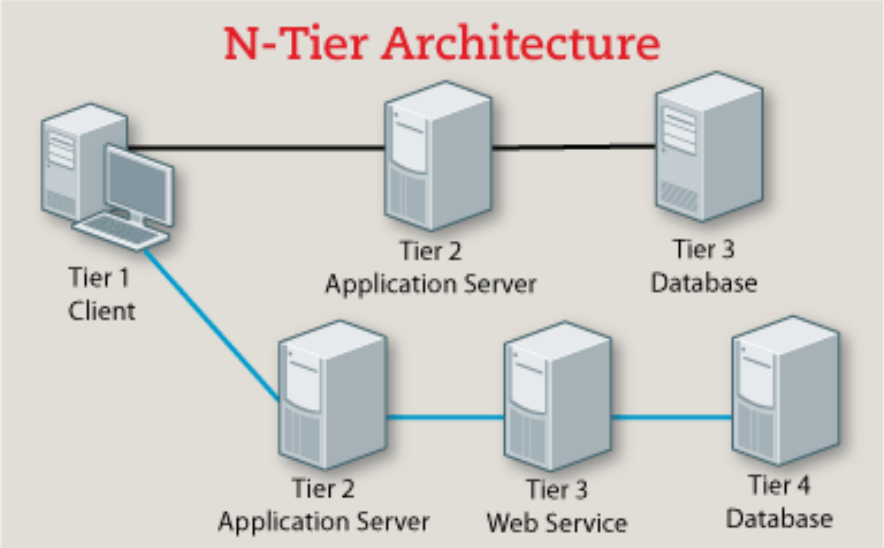
- високий ступінь масштабованості та розподіленості;
- високу безпеку, так як можна встановити свій приниципп захисту на кожному з рівнів;
- високу продуктивність, так як усі завдання розподіляються між різними серверами.

Недоліки такого підходу:

- Високі вимоги до швидкості каналу;

- Більш висока складність створення додатків;
- Більш висока складність в розгортанні та адмініструванні;
- Високі вимоги до продуктивності серверів додатків та сервера бази даних, як наслідок, висока вартість серверного обладнання;

Трирівнева архітектура може бути розширена до багаторівневої, шляхом виділення додаткових серверів для кожного рівня, кожен з яких представлятиме власні сервіси і надасть змогу користуватися послугами інших серверів різного рівня. [3] Абстрактний приклад багаторівневої моделі клієнт-сервер наведено на рисунку (Рис. 2.5).



“Рис. 2.5 - Багаторівнева архітектура клієнт-сервер”

3. ТЕХНОЛОГІЇ КЛІЄНТ-СЕРВЕР

3.1 Аналіз існуючих технологій

У сучасному світі, архітектура клієнт-сервер застосовується у великій кількості мережевих додатках та технологіях, що використовуються для

доступу до різних сервісів та додатках у мережі. Для доступу до будь-яких мережевих додатків чи сервісів, використовуються такі клієнти, можливості яких характеризуються таким поняттям як товщина. Воно визначає конфігурацію і набір певного програмного забезпечення і обладнаяння, що є у наявності. [2]

Щоб завадити різним проблемах елементів архітектури, потрібно виконувати обробку та синтез даних, на якійсь одній строні, на стороні сервера, тоді виникає поняття як "тонкий" клієнт, або ж на стороні клієнта, тоді виникає поняття як "товстий" клієнт. Кожен з підходів має свої переваги та недоліки. Якщо ми розглядаємо товстого клієнта, це призводить до надлишкової роботи мережі, внаслідок чого вона перевантажується, оскільки в ній передаються необроблена інформація, тобто надлишкові дані. Крім того, значно ускладнюється підтримка такої системи, та її можлива заміна, так як зміна алгоритму обчислень даних чи виправлення помилок, вимагають повної заміни всього інтерфейсного обладнання, в інакшому випадку, можуть виникнути серйозні помилки або неузгодженість даних, що може призвести до невідворотніх проблем. Якщо ж ми використовуємо тонкого клієнта, тобто вся інформації оброблюється на сервері, то виникають проблеми з описом процедур що вбудовані, та їх налагодження. Так як мова реалізації вбудованих процедур, зазвичай є декларативною і як висновок, не допускає покрокового налагодження. Крім того, великою проблемою є перенесення системи з обробкою інформації на сервері на іншу платформу, що є серйозним недоліком такої системи.

Більшість сучасних засобів швидкої розробки додатків (RAD), які працюють з різними базами даних, реалізує першу стратегію, тобто "товстий" клієнт, він забезпечує інтерфейс з сервером бази даних через вбудований SQL. Такий варіант реалізації системи з "товстим" клієнтом, крім перерахованих вище недоліків, зазвичай забезпечує неприпустимо низький рівень безпеки. Крім того, дану систему майже неможливо перевести на Web-технологію,

оскільки для доступу до сервера бази даних використовується спеціалізоване клієнтське ПЗ. [3]

3.2 Тонкий клієнт

Тонкі клієнти - це компактні пристрої без жорстких дисків, обчислювальна потужність якого і обсяг пам'яті визначаються завданнями користувача. Вони підключаються до серверів для виконання обчислювальних ролей та запускають протоколи віддаленого відображення для доступу до жорстких дисків у захищених центрах обробки даних.

Технологія тонкий клієнт-сервер базується на трьох основних складових:

- стовідсоткове виконання прикладних задач на термінальному сервері;
- розрахована на багато користувачів операційна система;
- технологія розподіленого відображення призначеного для користувача інтерфейсу додатків.

Клієнти мають можливість одночасно заходити в систему і запускати програми на сервері в різних, захищених один від одного сесіях сервера. В системі з використанням тонкого клієнта по мережі або при віддаленому підключенні на сервер передаються сигнали, які відображають натискання на ту чи іншу клавішу або той чи інший рух миші, клієнтом. А сервер виконує відповідні дії і формує відповідь, та зміни для екрану користувача і передає ці зміни тонкому клієнту (Рис. 3.1).

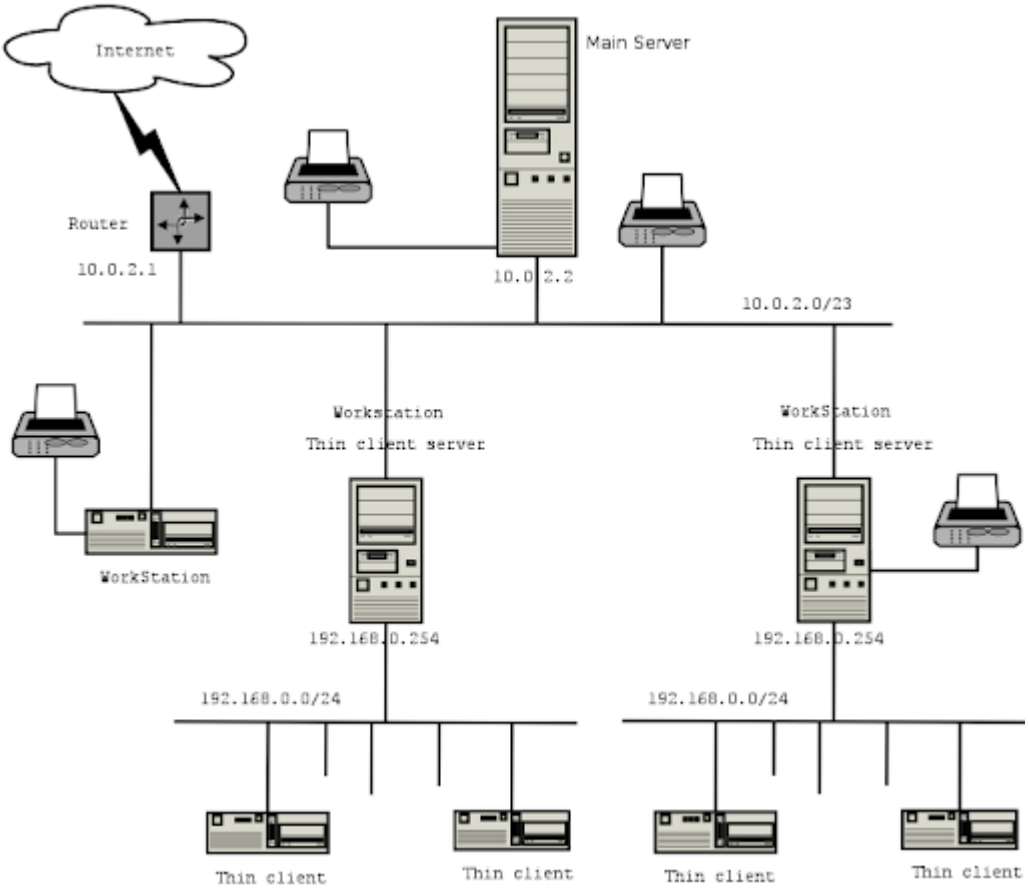
Для початку це ефективна технологія заміни ПК, яка сприяє негайному доступу до віртуальних настільних ПК та додатків та пропонує централізовані можливості обчислювальної техніки. Оновлення програмного забезпечення та програмного забезпечення та зміни програм можна легко здійснити в центрі обробки даних. Продуктивність робочого місця зростає, оскільки IT-командам не потрібно вирішувати проблеми на місці робочого столу кінцевого

користувача. Крім того, централізоване резервне копіювання настільних ПК та пристроїв доступу клієнтів спрощує завантаження адміністратора.

Користувачі не можуть копіювати інформацію на диск або зберігати її в інші місця, крім сервера. Несанкціоноване програмне забезпечення або програми не можуть бути встановлені на особистих пристроях, що захищає компанії від атак зловмисного програмного забезпечення та спроб порушення даних. Все це захищає інтелектуальну власність та забезпечує конфіденційність даних. В ідеалі необхідно використовувати пристрої, які мають за основу операційну систему на базі Linux і автоматично отримують рекомендовані оновлення безпеки. Так як вони не можуть бути легко підроблені, оскільки Linux є однією з найбільш орієнтованих на безпеку операційних систем на ринку.

Ці тонкі клієнти представляють найбільш доступні та екологічно стійкі рішення на сьогоднішній день. Як встановлено, ця технологія є обчислювальною моделлю, заснованою на сервері, що знижує рівень вартості обладнання. Тут програми запущені на віддаленому сервері та відображаються на настільних пристроях. Користувачі можуть отримати доступ до наборів програм з будь-якого пристрою, підключеного до сервера, не вимагаючи, щоб ІТ встановлював додатки на окремих пристроях.

Крім того, маючи обмежені деталі, що рухаються, мало пам'яті та мікропроцесорні пристрої, цей пристрій споживає приблизно 8-20 Вт порівняно з ПК який може працювати як на 170 Вт, так і на 800 Вт. Це різко знижує викиди вуглецю, і компанії можуть реінвестувати економію витрат на електроенергію в інших місцях.



“Рис. 3.1 - Принцип роботи тонкого клієнту”

Тонкі клієнти ефективні, якщо хочеться, щоб рівномірне зображення робочого столу працювало на їхньому підприємстві. Багатьом організаціям може бути важко придбати однакові конфігурації, навіть якщо постачальник задовольняє їх попит на кілька одиниць. Уявіть сценарій, коли 30 різних зображень дозволяють підтримувати 355 настільних комп’ютерів та 78 серверів, скажімо, чотирьох постачальників. Тонкі клієнти спрощують цю ситуацію, надаючи універсальне зображення на робочому столі через мережу. Це також вирішує процес вирішення окремих варіантів, і якщо користувач працює з відносно унікальним жорстким диском або графічним адаптером.[7]

Розглянемо фінансову установу, яка будує нову будівлю, для якої IT-адміністраторам потрібно розгорнути кілька робочих столів. Використання звичайних ПК означає, що вони не змогли б скласти всі свої ПК та скласти карту своїх накопичувачів на сто або більше робочих столів. Технологія

тонкого клієнта - це життєздатний варіант, оскільки має мінімальну конфігурацію і оснащена функціями миттєвого доступу до ресурсів. Компанія може додатково отримати значні заощадження за рахунок зниження рівня підтримки IT-служб, низького споживання енергії, обмежених рухомих компонентів, вентиляторних систем та високої доступності.

Ці кінцеві точки ідеально підходять для використання як основи індивідуального рішення для віддаленої роботи підприємства. Вони представляють ефективні, віддалено керовані пристрої, які звільняють IT-підрозділи для визначення пріоритетності IT-інновацій щодо конкурентних переваг. Такі функції, як прості налаштування профілів з централізованим програмним забезпеченням віддаленого управління та управління знімають тягар управління IT-системами. Пристрої дозволяють гнучко розгортатися, пропонуючи як версії програмного забезпечення Linux, так і Windows, а також протоколи віддаленого відображення. Крім того, сучасні рішення розроблені для роботи з будь-яким провідним постачальником термінальних послуг. Що стосується плюсів і мінусів тонких клієнтів, ця технологія має не суттєві недоліки, враховуючи широкий спектр переваг, які вона пропонує.

Переваги тонких клієнтів:

- Уніфікація пристроїв — всі клієнти мають однаковий набір програмного забезпечення;
- Зниження початкових витрат на придбання внаслідок мінімальних вимог до конфігурації;
- Значне зниження енергоспоживання — типовий тонкий клієнт має споживану потужність
- Простота реалізації завдань — немає необхідності налаштовування кожного комп'ютер окремо, оскільки здійснюється централізоване управління клієнтами;

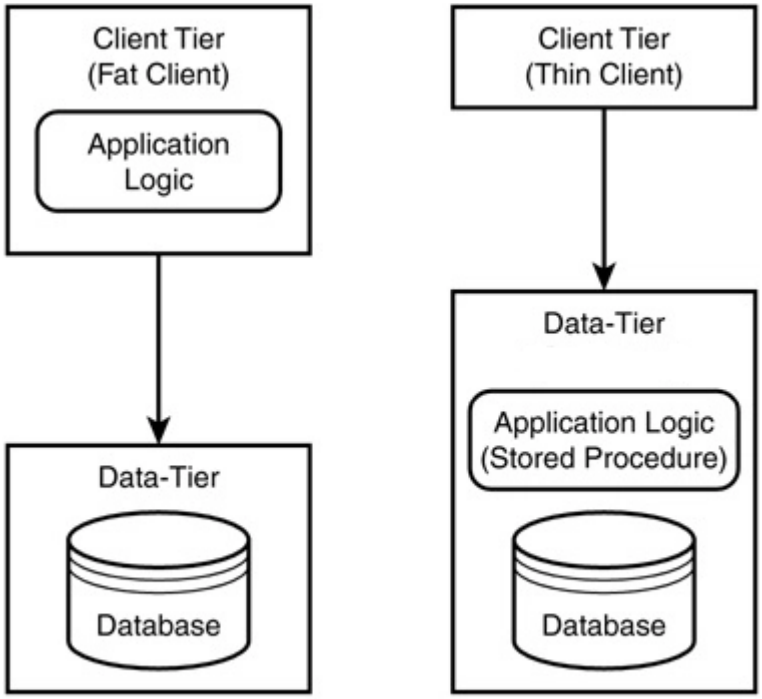
- Економія часу системного адміністратора, який обслуговує однакові комп'ютери, вірогідність поломок яких зведена до мінімуму, а всі програми встановлені на сервері;
- Захист від витоків інформації ;
- Масштабованість — створений один раз образ системи для роботи всієї групи користувачів дозволяє підтримувати легко масштабовану мережу.

Недоліки технології тонкий клієнт в основному пов'язані з високими вимогами до сервера і високою вартістю серверного обладнання:

- Висока вартість термінального сервера;
- Неможливість роботу без доступу до інтернету;
- Високі вимоги до швидкості мережевого каналу.

3.3 Товстий клієнт

На противагу цьому, товстий клієнт - це той, хто виконуватиме основну частину обробки в програмах клієнт-сервер. З товстими клієнтами немає необхідності в постійній комунікації з сервером, оскільки це в основному передача серверу інформації про архівне зберігання. Як і у випадку з тонким клієнтом, термін часто використовується для позначення програмного забезпечення, але знову ж використовується і для опису самого мережевого комп'ютера. Якщо вашим програмам потрібні мультимедійні компоненти або інтенсивна пропускна здатність, ви також хочете розглянути можливість роботи з товстими клієнтами. Одна з найбільших переваг товстих клієнтів полягає в тому, що деякі операційні системи та програмне забезпечення не можуть працювати на тонких клієнтах. Товсті клієнти можуть впоратися з цим, оскільки у нього є власні ресурси (Рис. 3.2).



“ Рис. 3.2 - Приклад роботи тонкого і товстого клієнту на рівні логіки.”

Також, зараз все частіше використовують ще один термін, так званий rich – клієнт. Розробники rich клієнту, знайшли компроміс між товстим і тонким клієнтом. Як і тонкий клієнт, rich-клієнт також представляє графічний інтерфейс, що описується вже засобами XML і включає деяку функціональність товстих клієнтів, наприклад, інтерфейс, вкладки, множинні вікна, що випадають меню. [6]

Прикладна логіка таких клієнтів, як і на тонких клієнтах, також реалізована на сервері. Обмін усіх даних відбувається в стандартному форматі, на основі того ж XML через основі протоколи, і потім показуються на клієнтському обладнанні.

Переваги товстих клієнтів:

- Більш багата функціональність у порівнянні з тонкими клієнтами;
- Багатокористувальська робота;
- Можливість роботу навіть без доступу до інтернету;
- Швидка робота інтерфейсу.

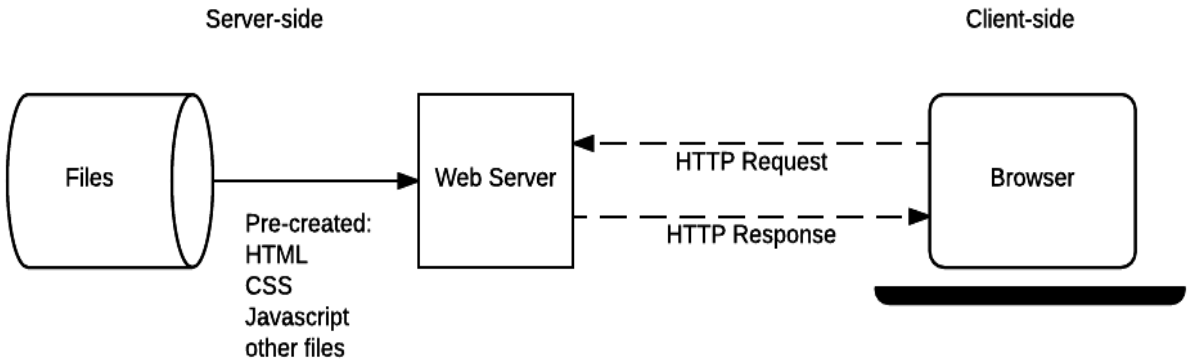
Недоліки товстих клієнтів:

- Великий розмір дистрибутиву;
- Складність встановлення і налаштування;
- Складність оновлення системи на стороні клієнта;
- Дані можуть бути неактуальні до оновлення.

3.4 Модель клієнт-сервер в мережі

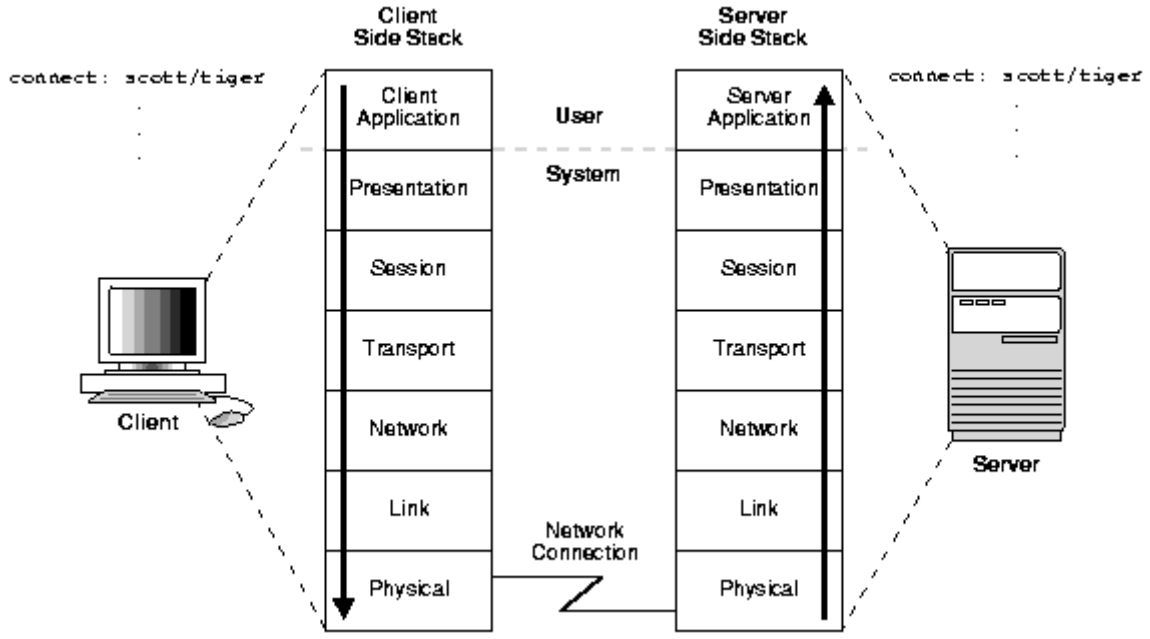
Служби інтернет, подібно серверу баз даних, такі як FTP, протокол передачі файлів в мережі, або Web, надають послуги з подібністю до моделі клієнт-сервер. Клієнт і сервер взаємодіють один з одним по різних протоколах - SMTP, FTP, HTTP, та інші, але алгоритм взаємодії за різними протоколами однаковий: клієнт надсилає запит серверу, сервер отримує його, обробляє, і якщо потрібно, відправляє відповідь клієнту. Буває, що цей процес повторюється кілька разів, а буває, що він виконується одноразово.

Для взаємодії клієнта і сервера по протоколу HTTP, на комп'ютері-сервері повинен бути встановлений веб-сервер, а на комп'ютері користувача – клієнт, наприклад web-браузер. Все починається з введення адреси сайту в адресному рядку браузера. Спочатку користувач вводить URL адресу сторінки сайту, після натискання кнопки Enter, браузер спочатку перевіряє, чи існує домен з таким ім'ям. Якщо домен існує, браузер отримує IP адресу сервера, який пов'язаний з доменом, і відправляє HTTP запит на цей сервер (Рис. 3.3). У ньому браузер вказує домен, кукі, свою назву, бажані формати даних, і ін. Сервер отримує ці дані, і на їх основі формує відповідь. У відповіді від сервера міститися заголовки відповіді, і тіло відповіді у Json файлі. У тілі відповіді міститься сама веб-сторінка. Клієнт отримує цю інформацію, і виводить її на монітор користувача.



“Рис. 3.3 - Взаємодія клієнта і сервера по протоколу HTTP”

Найважливішою особливістю обчислювальної моделі клієнт-сервер є розподіл прикладних завдань між клієнтами і серверами. Ілюстрація загального випадку наведена на рис. . Як на клієнті, так і на сервері базовим програмним забезпеченням є, зрозуміло, операційна система. Апаратні платформи і операційні системи клієнтів та серверів можуть відрізнятися. У самому справі, в єдиному оточенні можуть використовуватися різні типи клієнтських і серверних платформ і операційних систем. Проте ці відмінності не мають значення, якщо сервер і клієнт використовують одні і ті ж комунікаційні протоколи і підтримують однакові додатка. Взаємодія клієнта і сервера забезпечується комунікаційним програмним забезпеченням. Прикладами такого програмного забезпечення є набір протоколів TCP/IP - протоколи OSI, а також різні фірмові архітектури (Рис. 3.4).



“Рис. 3.4 - Клієнт-сервер в моделі OSI”

Зрозуміло, призначення всього цього програмного забезпечення підтримки протоколів і операційної системи, полягає в наданні бази для розподілених додатків. В ідеальному випадку виконується додатком функція повинна бути розподілена між клієнтом і сервером таким чином, щоб обчислювальні та мережеві ресурси використовувалися оптимально, а користувачі отримали оптимальні можливості для виконання різних завдань і спільної роботи. В деяких випадках для цього може бути необхідно, щоб більша частина програмного забезпечення виконувалася на сервері, тоді як в інших випадках велика частина логіки може бути реалізована на клієнті.

Додатки з архітектурою клієнт-сервер, такі як Web-сервери чи браузери, використовують для обміну даних стек протоколів TCP/IP. Ці інтернет додатки виконують обмін даними в TCP/IP мережах з використанням інтерфейсу сокетів. Інтерфейс сокетів реалізований із бібліотеки стандартних програм, котрі розробник додатку може використати для розробки додатків, здатних реалізувати обмін даними з іншими додатками через інтернет.[1]

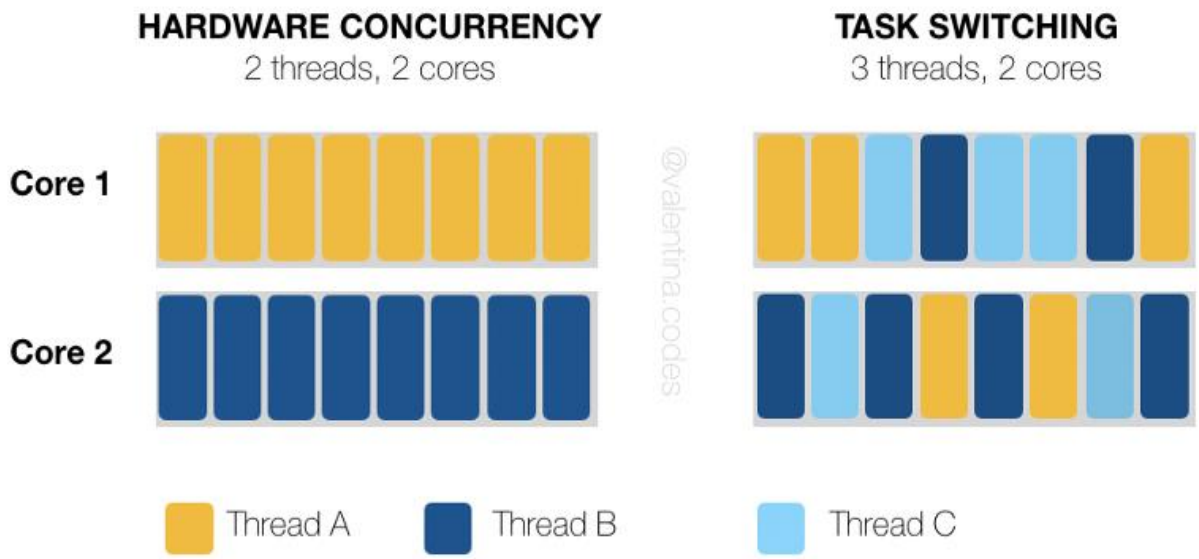
Обмін даними, за допомогою стеку протоколів TCP/IP, додаток клієнт-сервер виконує такі наступні дії:

- Створення сокету. Якщо сокет існує, то цей крок пропускаємо.
- Прив’зка IP – адресу і порта до сокету.
- Пошук запитів на з’єднання, якщо додаток є сервером, з використанням потокового сокету.
- Встановлення з’єднання, якщо додаток є клієнтом, з використанням потокового сокету.
- Обмін даними.
- Закриття сокету після того, як обмін даними був завершений.

3.5 Багатопотоковість

У комп'ютерній архітектурі багатопотоковість - здатність центрального процесора або одного ядра в багатоядерному процесорі одночасно виконувати кілька процесів або потоків, відповідним чином підтримуваних операційною системою. Цей підхід відрізняється від багатопроцесорності, так як багатопотоковість процесів і потоків спільно використовує ресурси одного або декількох ядер: обчислювальних блоків, кеш-пам'яті або буфера перекладу з перетворенням (Рис. 3.5).

У тих випадках, коли багатопроцесорні системи включають в себе кілька повних блоків обробки, багатопотоковість спрямована на максимізацію використання ресурсів одного ядра процесора, використовуючи паралелізм на рівні потоків, а також на рівні інструкцій. Оскільки ці два методи є взаємодоповнюючими, їх іноді об'єднують в системах з декількома багатопотоковими процесорами і в процесорами з декількома багатопотоковими ядрами.



“Рис. 3.5 - Приклад роботи багатопотоковості”

Сутність багатопотоковості це квазібагатозадачність на рівні одного процесу який виконується. З цього випливає те, що усі потоки процесу крім загального простору адресів, мають і загальні дескриптори файлів, також, для коректної роботи, виконуваний процес щонайменше один головний потік.

На будь-якому простому процесорі, управління потоками здійснюється її ж операційною системою. Потік буде виповнюватись до тих пір, доти не відбудеться апаратне переривання процесу, будь-який системний виклик або поки не закінчиться відведений для цього потоку час. Після того як закінчився потік, процесор перемикається на код операційної системи, який зберігає стан потоку, або перемикається на стан другого потоку, якому теж виділяється деякий час. При такій багатопотоковості, більша кількість тактів процесора витрачається на код операційної системи, який перемикає контексти. Однак, якщо підтримку потоків реалізувати апаратно, то процесор сам зможе перемикатися між потоками або навіть виконувати кілька потоків одночасно за кожен такт. Розрізняють дві форми апаратної реалізації багатопоточності: тимчасова багатопотоковість та одночасна багатопотоковість.

У багатопотоковому середовищі часто виникають проблеми, які зв’язані з використанням потоків які виконуються паралельно один з одим, одних і тих же пристроїв або деяких даних. Для вирішення подібних проблем використовуються такі методи взаємодії потоків, як м'ютекси, семафори, критичні секції і події.

М'ютекс це такий об'єкт синхронізації, який встановлюється в особливий стан сигналу, коли він не зайнятий деяким потоком. Тільки один потік володіє цим об'єктом в будь-який момент часу, одночасний доступ до загального ресурсу виключається. Після всіх необхідних дій м'ютекс звільняється, надаючи іншим потокам доступ до загального ресурсу. Об'єкт може підтримувати рекурсивний захоплення вдруге тим же потоком, збільшуючи лічильник, що не блокуючи потік, і вимагаючи потім багаторазового звільнення. Проте, є і такі реалізації, які не підтримують таке і призводять до взаємної блокування потоку при спробі рекурсивного захоплення. [8]

Семафори це такі об’єкти, які можуть бути використані деяким числом потоків в один і той же період часу, аж доти список таких ресурсів не стане порожнім. Тоді всі інші додаткові потоки змушені будуть чекати, доти необхідна кількість ресурсів не звільниться і не буде доступна. Семафори дуже ефективний спосіб реалізації багатопотоковості, оскільки вони мають змогу надати одночасний доступ до ресурсів.

Критична секція Critical section, об'єкт синхронізації потоків дозволяє запобігти одночасному виконанню деякого набору операцій, яків основному пов'язаних з доступом до даних, декількома потоками. Критичні секції працюють за принципом взаємного виключення, коли потік знаходиться в критичній секції, вхід будь-яких інших потоків не допускається. [9] Критична секція виконує ті ж завдання, що і м'ютекс. Подібно до роботи м'ютексів, критична секція, що представляється ресурсом, може бути використана лише

одним конкретним потоком, саме в конкретний момент часу, що показує їх значну перевагу, для розмежованого доступу, до ресурсів.

Найважливішою областю є планувальник потоків, який повинен швидко вибрати зі списку готових до запуску потоків для виконання наступного, а також підтримувати готові до запуску і зупинені списки потоків. Важливою підтемою є різні схеми пріоритетів потоків, які можуть використовуватися планувальником. Планувальник потоків може бути повністю реалізований в програмному забезпеченні, повністю на апаратному рівні або у вигляді апаратно-програмної комбінації.

Ще однією областю досліджень є те, які типи подій повинні викликати перемикання потоків: втрати в кеш-пам'яті, багатопотоковия зв'язок, тощо. Якщо багатопотокова схема копіює все програмне забезпечення, включаючи привілейовані регістри управління то вона дозволяє створювати віртуальні машини для кожного потоку. Це дозволяє кожному потоку запускати свою власну операційну систему на одному процесорі. З іншого боку, якщо зберігається тільки стан призначеного для користувача режиму, потрібна менша кількість апаратного забезпечення, що дозволило б одночасно активізувати кілька потоків для однієї і тієї ж області кристала.

Перваги:

- полегшення програми за допомогою використання загального адресного простору;
- менші витрати на створення потоку в порівнянні з процесами;
- підвищення продуктивності роботи процесу за рахунок розпаралелювання обчислень процесораа;
- якщо потік часто втрачає кеш, інші потоки можуть продовжувати використовувати невикористані обчислювальні ресурси.

Недоліки:

- кілька потоків можуть втручатися один в одного при спільному використанні апаратних ресурсів;
- з програмною точки зору апаратна підтримка багатопоточності більш трудомістка для програмного забезпечення;
- проблема планування потоків;
- специфіка використання, яка може призвести до погіршеної продуктивність через конкуренцію за загальні ресурси.

4. ОПИС МОДУЛІВ

4.1 Модуль реалізації зв’язку

Кожен комп'ютер в мережі має IP-адресу. IP-адреса (Internet Protocol Address) - це унікальний мережевий ідентифікатор, який присвоюється кожному учаснику локальної або глобальної комп'ютерної мережі. Це може бути як Всесвітня павутина, так і приватна мережа підприємства. Головне - вона повинна бути заснована на протоколі TCP / IP.

Незалежно від типу мережі, IP-адреси в її межах не повинні повторюватися. Завдяки можливості привласнення унікального ідентифікатора кожному користувачеві з'являється можливість розмежування дій. Система вміє розпізнавати користувачів, отже, кожному з них можна давати той чи інший рівень доступу, відстежити дії або зовсім заблокувати.

Сокет схожий на тип програмного забезпечення, який діє як кінцева точка, яка функціонує при встановленні двонаправленого з'єднання з мережею між кінцевою точкою сервера і програмою-одержувачем клієнта. Її також часто називають однією кінцевою точкою в каналі двостороннього зв'язку. Ці сокети виготовляються і мобілізуються разом з набором програмних запитів, які ідентифікуються як виклики функцій, які технічно називаються

інтерфейсом прикладного програмування (API). Сокет може спростити роботу програми, оскільки тепер програмістам залишається тільки турбуватися про те, як управляти функціями сокета, і це дозволяє їм покладатися на операційну систему для коректної передачі повідомлень по мережі.

Мережеві додатки використовують сокети для передачі даних в мережі, в TCP/IP мережах. Сокет (socket) – це абстракція, яка надає кінцеву точку з'єднання. Так як сокет являється двонаправним об'єктом, з його допомогою можна як і отримувати дані, так і відправлять.

Сокет має три атрибута:

- Мережевий адрес (IP-адрес) системи;
- Номер порту, який індетифікує процес, який виконує обмін даними через сокет;
- Тип сокету, потоковий чи дейтаграмний, який індетифікує протокол обміну даних.

IP-адрес ідентифікує мережевий вузол, номер порту – процес у вузлі, а тип сокету – спосіб обміну даними, за допомогою протоколу встановленням зв'язку чи без нього.

Для формування маршруту обміну даних необхідно мати два сокети. Коли які-небудь два процеси обмінюються даними, вони використовують модель клієнт-сервер для встановлення зв'язку. Серверний додаток контролює конкретний порт системи, сервер повністю виявляється IP-адресом системи, в який він працює, та номером порту, на якому очікує з'єднання. Клієнт ініціює з'єднання з будь-якого доступного порту та намагається підключитись до серверу, з конкретним IP-адресом та номером порту. Як тільки зв'язок буде встановлений, клієнт і сервер зможуть здійснювати обмін даними у відповідність з їх власними протоколами.

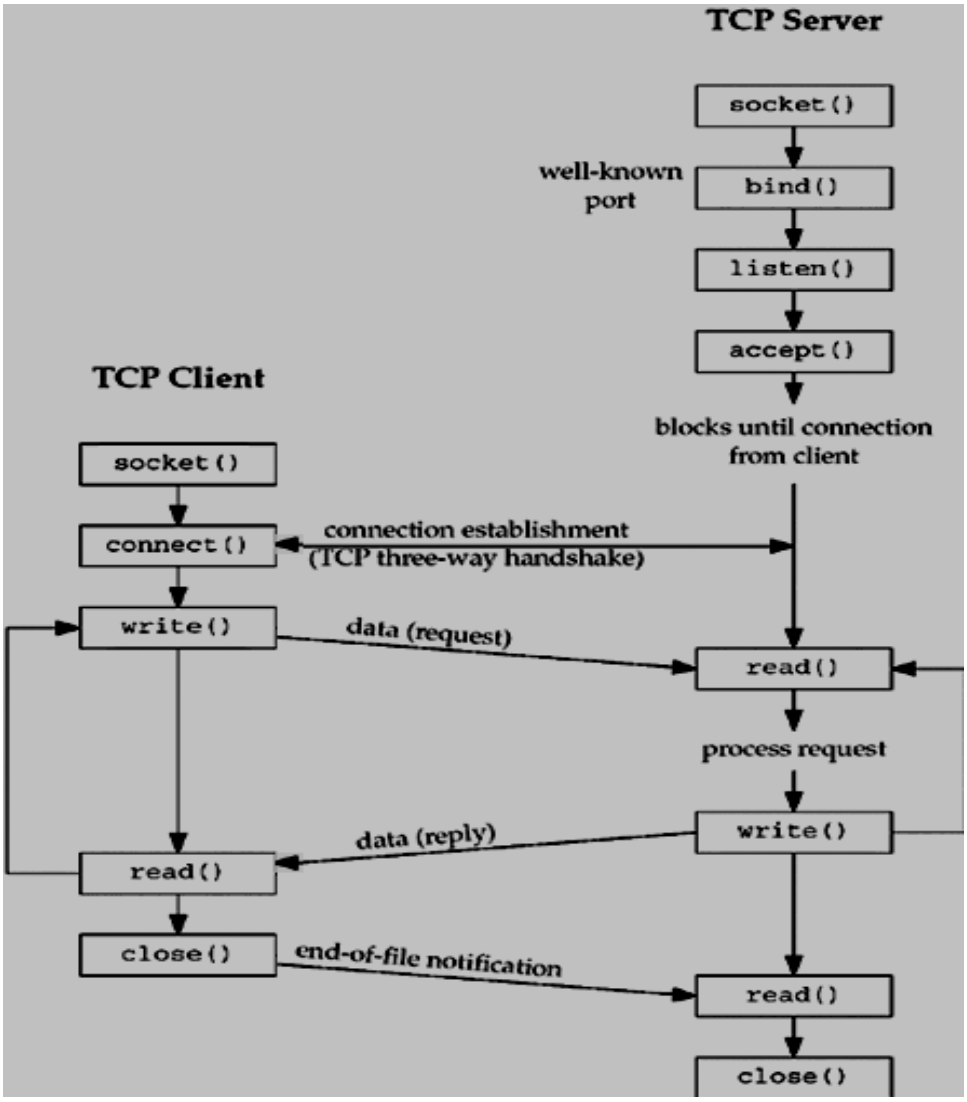
Послідовність дій в процесі обміну даними з використанням сокетів, залежить від того, орієнтована передача на встановлення зв’язку по TCP, чи не потребує встановленням зв’язку по UDP.

Тип сокету вказує на те, який протокол використовується для обміну даними через цей сокет. Потік з встановленням зв’язку схожий на звичайну розмову по телефону. Коли ви хочете поговорити, ви повинні набрати номер телефону й встановити з’єднання, перед тим як почати діалог. Точно так само й обмін даними на основі з’єднання потребує як і від процесу, котрий відправляє дані, так і від процесу який приймає дані, встановити з’єднання перед тим, як можна буде почати обмін деякими даними (Рис 4.1).

В рамках набору протоколів TCP/IP, TCP протокол керуванням передачею, підтримує на основі встановленого зв’язку, передачу даних між двома процесами, які виконуються на двох пристроях в мережі інтернет. Протокол TCP надає надійний двосторонній обмін даними між процесами. IP протокол вирішує задачу доставки пакетів адресанту. Протокол IP не гарантує доставку пакетів даних, він навіть не доставляє пакети в якійсь конкретній послідовності, але також він ефективно доставляє пакети з однієї мережі в іншу.

Протокол TCP відповідає за впрядкованість пакетів в задану послідовність, також за виявлення помилок, які можуть виникнути при передачі та за повсторну передачу пакетув, у разі виникнення помилок.

TCP протокол зазвичай використовується в додатках, які планують обмін великими об’ємами даних, за один сеанс. Крім того, додатки які потребуються надійний обмін даними, як правило, теж використовують TCP протокол. В моделі з сокетами, який використовує протокол TCP, називається потоковим сокетом.



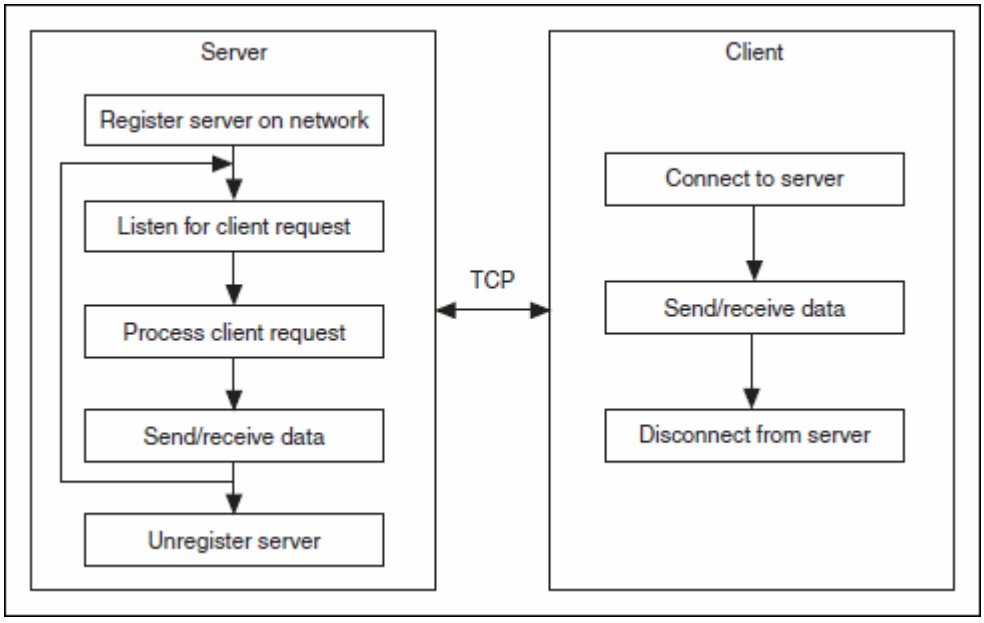
"Рис 4.1 - Робота сокету в стеку TCP/IP"

Також існують протоколи обміну даних, які не орієнтовані на встановлення зв'язку, не потребують того, щоб отримувач і відправник встановлювали явний зв'язок. В рамках набору протоколів TCP/IP, UDP протокол для відправки та отримання даних, так званих дейтаграмм. На відмінну від TCP протокол UDP не гарантує ні того що дейтаграмма дістанеться пункту передачі, ні того що вони будуть відправлені.

Протокол UDP використовується додатками, які здійснюють обмін невеликими кількостями даних за один сеанс зв'язку, або додатками які не мають високих вимог до надійності або порядку доставки необхідних пакетів

даних. В моделі з сокетами, сокет, який використовує протокол UDP називається дейтаграмним сокетом.

Для сокетів без встановлення зв'язку, тобто по протоколу UDP, дії під номером три та чотири, виконувати не потрібно. Не залежно від того, чи є додаток сервером, або ж клієнтом, кожен додаток, перш за все створює сокет з IP адресом локального комп'ютера і номером порту. IP адрес виявляє машину, на якій виконується додаток, а номер порту індетифікує додаток, яке використовує сокет.



"Рис. 4.2 - Модель роботи клієнта та сервера"

Для реалізації зв'язку між модулем клієнта, та модулем сервера був розроблений модуль Shell, який місить всі необхідні функції для правильного та коректного зв'язку. BufferedReader - це клас, який спрощує читання тексту з потоку введення символів. Він буферизує символи, щоб забезпечити ефективне читання текстових даних. Як і більшість класів введення-виведення Java, BufferedReader імплементує шаблон Decorator, це означає, що він очікує Reader в своєму конструкторі. Таким чином, він дозволяє нам гнучко розширювати екземпляр реалізації Reader з функцією буферизації.

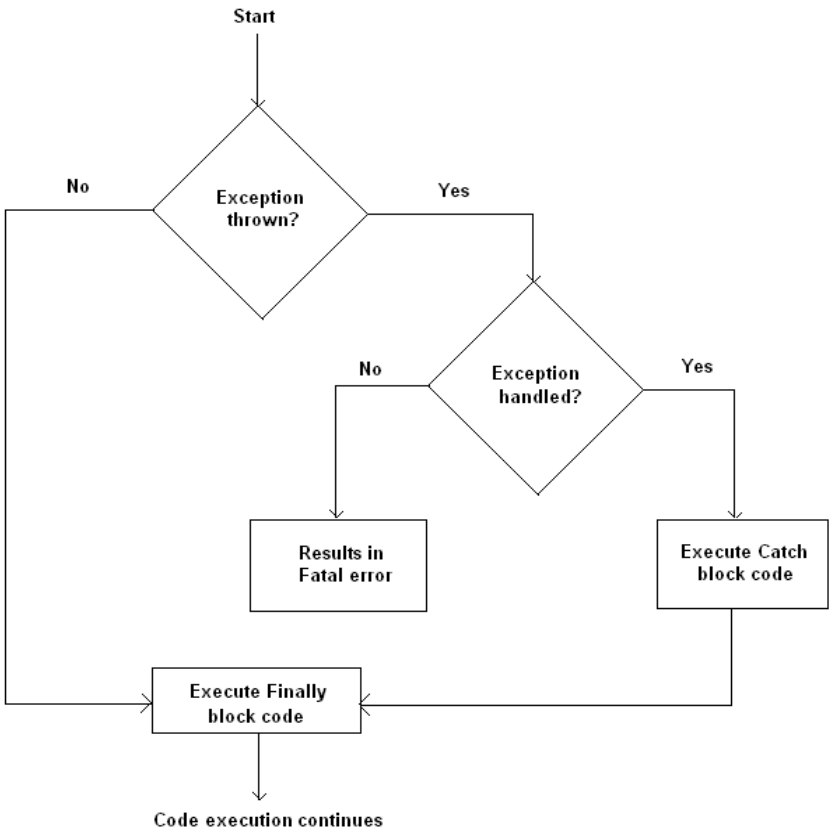
Клас BufferedWriter записує текст в потік, попередньо буферизуючи записуються символи, тим самим знижуючи кількість звернень до фізичного носія для запису даних.

У даному модулі реалізовані конструктори для забезпечення правильного зв’зку та дій як зі сторони клієнта, так і зі сторони сервера, що значно спрощує написання та об’єм коду, для інших модулів. Також з цього модуля був створений Jar архів за допомогою якого можна використовувати і в інших додатках. Також у цьому модулі виконується перевірка типових помілок за допомогою спеціальної конструкції try-catch. Яка працює наступним чином.

- Спочатку виконується код всередині блоку try {...}.
- Якщо в ньому немає помилок, то блок catch (err) ігнорується: виконання доходить до кінця try і потім далі, повністю пропускаючи catch.
- Якщо ж в ньому виникає помилка, то виконання try переривається, і потік управління переходить в початок catch (err). Мінлива err (можна використовувати будь-яке ім'я) містить об'єкт помилки з докладною інформацією про те, що сталося.

Блок try-catch Java використовується для обробки винятків, оголошуючи тип винятку в межах параметра. Заявлений виняток повинен бути винятком батьківського класу, тобто винятком, або згенерованим типом винятку. Однак хороший підхід - оголосити створений тип виключення.

Блок try-catch повинен використовуватися лише після випробувального блоку. Ви можете використовувати кілька блоків лову з одним блоком спробу.



“Рис. 4.3 – Блок-схема роботи try-catch”

Таким чином ми можемо мінімізувати типові помилки які можуть виникнути в нашій архітектурі, щоб запобігти поламок, або інших проблем.

4.2 Модуль сервера

Конструктор класу Socket приймає два параметри - рядок, IP-адреса сервера і ціле число, номер порту на сервері, до якого клієнт хотів би підключитися. 127.0.0.1 - це адреса за замовчуванням локальної системи в комп'ютерних мережах, так званий localhost. У комп'ютерній мережі localhost, відноситься до комп'ютера, на якому запущена програма. Комп'ютер працює як віртуальний сервер. У цьому сенсі комп'ютер - це не фізичний об'єкт, а система, яка працює всередині. Метод getOutputStream() класу Socket повертає об'єкт OutputStream, тут об'єкт є ostream. Це відправна точка всього

спілкування (програми). Тут сокет пов'язаний з потоками. Потоки сприяють передачі даних.

OutputStream є абстрактним класом; він не може бути використаний безпосередньо. У наведеному вище коді він пов'язаний з конкретним класом DataOutputStream. Метод writeBytes () об'єкта DataOutputStream приймає строкове повідомлення і передає його в Socket. Тепер клієнтський сокет відправляється на інший сокет на сервері. Коли робота закінчиться, закрийте потоки і сокет. Він звільняє дескриптори (посилання), пов'язані з системними ресурсами.

Так як надавати програмі більше ресурсів ніж їй необхідно - і витратна і не розумне справу, тому в конструкторі ServerSocket вам пропонують оголосити максимум з'єднань, прийнятих сервером при роботі. Якщо воно не вказано, то замовчуванням це число буде вважатися рівним 50.

Так, можна припустити, що ServerSocket це такий же сокет, тільки для сервера. Але він грає зовсім іншу роль ніж клас Socket. Він потрібен тільки на етапі створення з'єднання. Створивши об'єкт типу ServerSocket необхідно з'ясувати, що з сервером хтось хоче з'єднатися. Тут підключається метод accept (). Шуканий чекає поки хто-небудь не захоче під'єднатися до нього, і коли це відбувається повертає об'єкт типу Socket, тобто відтворений клієнтський сокет. І ось коли сокет клієнта створений на стороні сервера, можна починати двостороннє спілкування. Створити об'єкт типу Socket на стороні клієнта і відтворити його за допомогою ServerSocket на стороні сервера - ось необхідний мінімум для з'єднання.

У комп'ютера є фізичні порти, які можна побачити, наприклад, USB-пристрої. А є і програмні, існуючі в віртуальному електронно-цифровому світі і недоступні для неозброєного ока. Кожна програма повинна бути готова до передачі і прийому електронної інформації. Тому всередині операційної системи для них виділяються порти, для обміну інформацією. Кількість портів обмежена з урахуванням 16-бітної адресації 2^16 = 65536 номерів, початок з

нульовогу порту. Всі порти розділені на три діапазони - загальновідомі або системні порти,(0-1023), зареєстровані або призначені для користувача, (1024-49151), і динамічні або приватні порти, (49152-65535).[14] Раніше передача велася в напівдуплексному режимі, і для з'єднання потрібно було два порти. З прийняттям протоколів TCP і UDP за необхідим залишився лише один порт, і парні номери не застосовуються - цим пояснюється відсутність реєстрації деяких портів з діапазону загальновідомих.

Для нашого випадку, використовуємо один порт з тих, які призначені для користувачів, тобто з номуру (1024-49151). А для IP адреси, візьмемо localhost, 127.0.0.1. Основний метод main приймає запит від клієнта, оброблює, та формує відповідь, яку потім відправляє клієнту. Для коректної роботи, клієнт має вибрати такий самий порт, та IP адресу, інакше запит просто не буде надісланий до сервера.

4.3 Модуль клієнта

Модуль клієнту реалізований досить простим чином. Більшість функцій реалізовано схожим чином, для керування потоками. Клас Socket реалізує ідею сокета. Через його канали введення та виведення спілкується клієнт з сервером, тільки на стороні сервера, ServerSocket. Як вже сказано, цей клас реалізується на стороні клієнта, а сервер відтворює його, отримуючи сигнал при підключенні. Таким чином відбувається спілкування в мережі. Для корекної роботи в мережі, необхідна IP-адреса. Але якщо клас сокету не зміг перетворити його в реальний, існуючий адрес в мережі, то він згенерує виняток UnknownHostException.

Також другим параметром - є порт. Якщо в якості номера порту буде вказано 0, то система сама виділить вільний порт. Також при втраті з'єднання може статися виняток IOException.

Слід зазначити тип адреси в другому конструкторі - InetAddress. Він приходить на допомогу, наприклад, коли потрібно вказати в якості адреси доменне ім'я. Так само коли під доменом мається на увазі кілька IP-адрес, то за допомогою InetAddress можна отримати їх масив. Проте з IP він працює теж. Так само можна отримати ім'я хоста, масив байт складових IP адресу і т.д. Ми трохи торкнемося його далі, але за повною інформацією доведеться пройти до офіційної документації.

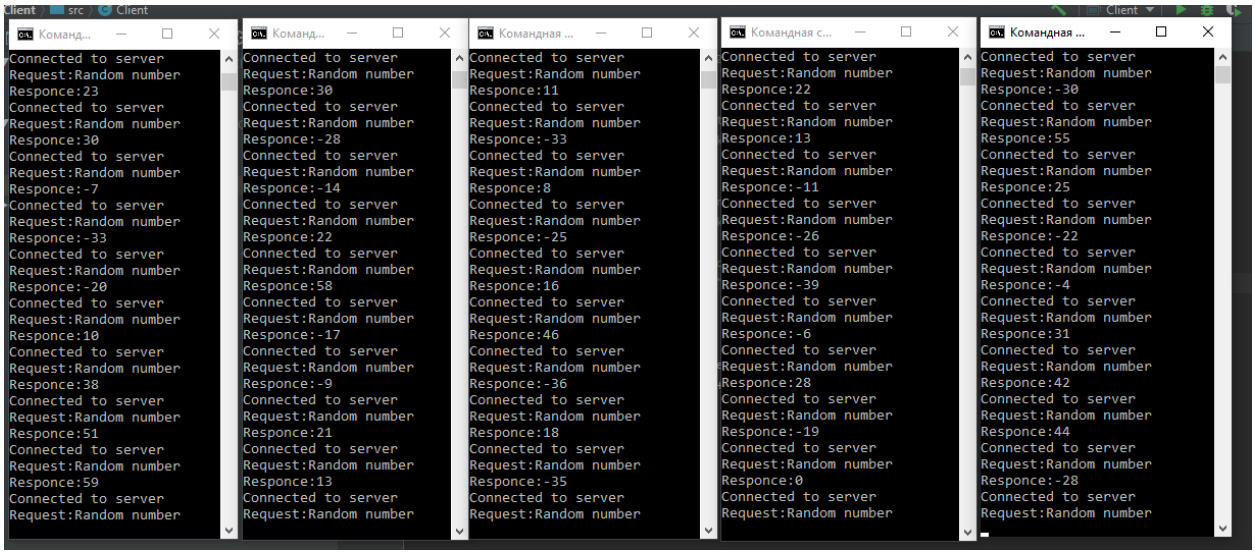
При ініціалізації об'єкта типу Socket, клієнт, якого той належить, оголошує в мережі, що хоче з'єднатися з сервером про певною адресою і номером порту. Класс Socket реалізує інтерфейс AutoCloseable, тому його можна використовувати в конструкції try-with-resources. Проте закрити сокет також можна класичним чином, за допомогою close ().

Також в модулі клієнта, реалізовані основні запити для серверу, для швидкої перевірки якості та швидкодії відповіді серверу на відправлений запит. Для перевірки коректної роботи сервера при його навантаження, а також для коректної роботи багатопотоковості, був створений спеціальний скрипт, який моделює безліч запитів клієнта до сервера.

5 ТЕСТУВАННЯ ПРОГРАМИ ТА АНАЛІЗ РЕЗУЛЬТАТІВ

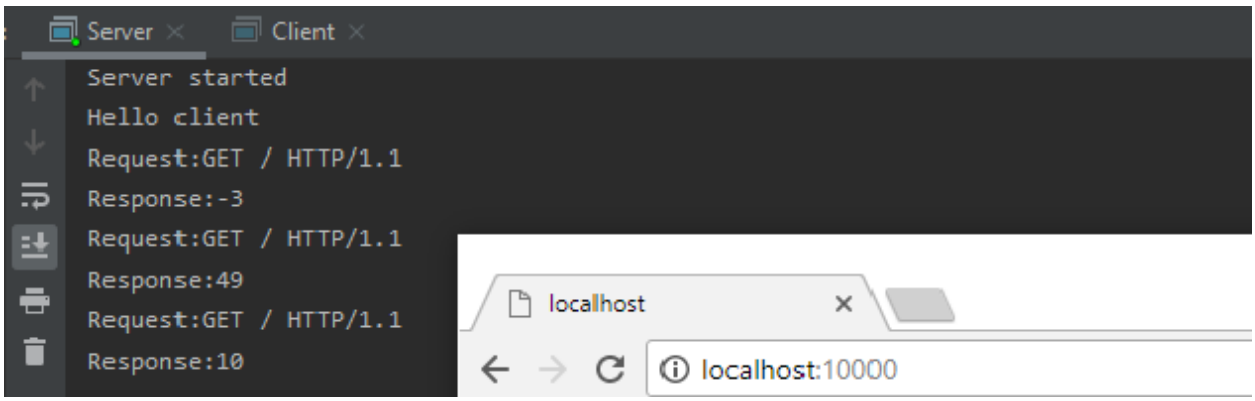
5.1 Тестування роботи

Під час тестування, були перевірені основні модулі системи. Додаток працює як і планувалося. Для перевірки багатопотоковості додатку, було створено скрипт і названий Client.dat, який збирає і запускає файл клієнту. Для перевірки запускаємо 5 клієнтів з часом запиту 4000 мілісекунд, які постійно відправляють запити серверу (Рис 5.1).



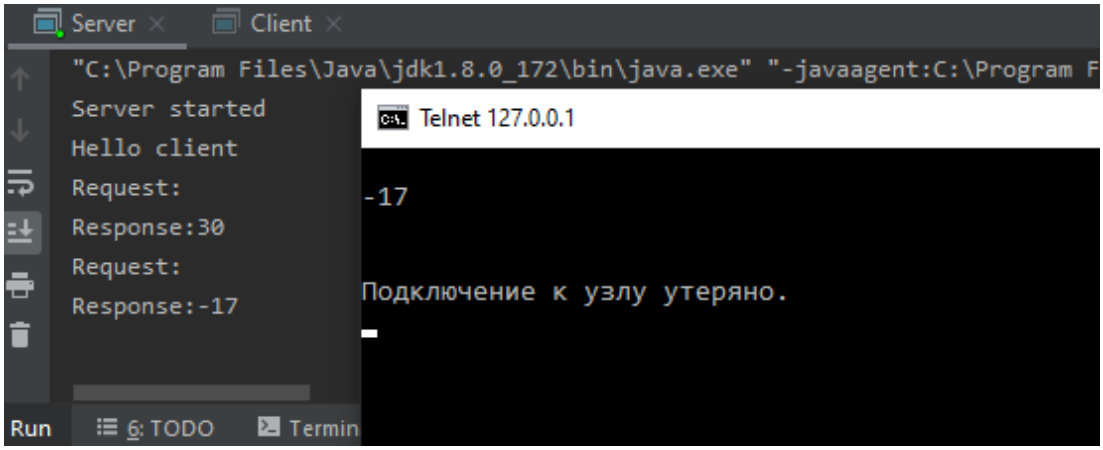
“Рис. 5.1 – Приклад багатопотоковості додатку”

Також був реалізований доступ до серверу через веб браузер, що дає змогу робити і отримувати запити в мережі інтернет (рис.5.2). Для цього вказуємо потрібну IP адресу, в нашому випадку це localhost, та номер порту в нашому випадку, 10000, взятий довільно, в адресі браузера.



“Рис. 5.2 Доступ до додатку через браузер”

Також маємо змогу отримати доступ до серверу, за допомогою протоколу для передачі текстових даних (Рис. 5.3). Також маємо змогу підключитись до сервера віддалено, по цьому ж протоколу, але на жаль, простокол telnet не гарантує надійності передачі, за що й втратив свою популярність, поступившися протоколу віддаленого доступу SSH.

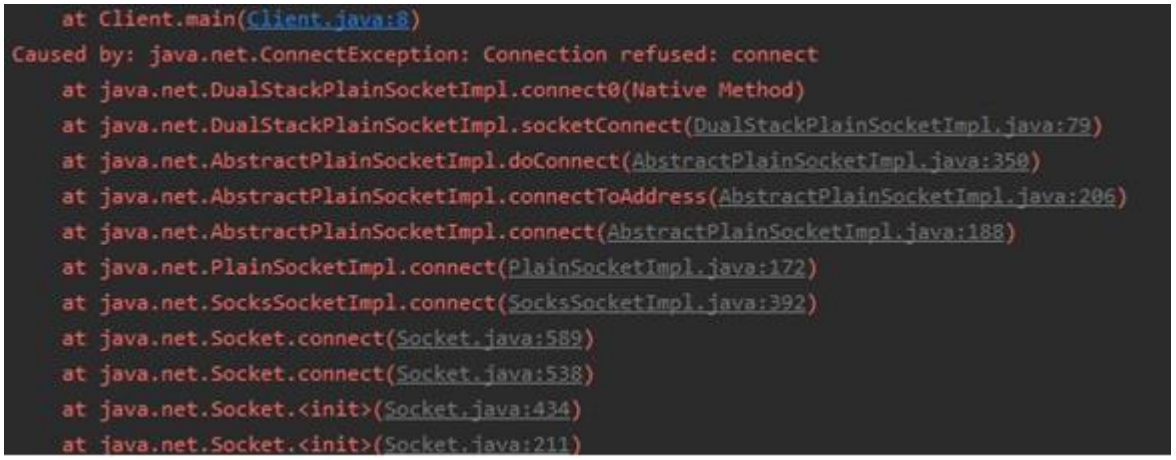


“Рис. 5.3 Доступ до додатку через telnet”

Для коректної роботи програми було створений jar архів, який було створено у вигляді бібліотеки для подолання типових проблем, при правильному доступу ресурсів клієнта, до ресурсу сервера. Також слід

зауважити, що сервер працює в стабільному режимі, безперебійно, що дозволяє у будь який час, без проблем отримати необхідні дані від сервера.

Приклад помилки доступу через сокет, (Рис. 5.3) клієнт намагається з'єднатися з сервером, вказуючи непрацюючий порт, при коректній роботі серверу, отримаємо помилку клієнта.



"Рис. 5.3 Помилка сокету"

ВИСНОВКИ

Головною метою даного дипломного проєкту було створення оптимізованого багатопотокового клієнт-серверного додатку на основі отриманих знань. Система розроблялась за допомогою сучасних технологій клієнт-серверної архітектури, та мови високого рівня Java.

Також, у ході розробки, було реалізовано архітектуру з оптимальним зв'язком, між клієнтом та сервером, що значно зменшив та оптимізував програмну реалізацію як клієнту, так і серверу. У тому числі було створено відповідні файли та файли, за допомогою яких можна з портувати систему на різні пристрої.

Одним із варіантів подальшого розвитку програми, є розвиток архітектури у веб середовище, та розширення його функціональності, як програмної так і апаратної. Також для можливості більшого навантаження на сервер, можна збільшити можливості ПЗ пристрою, чи взяти в оренду вже існуючий сервер.

В цілому, система продемонструвала свою надійність, та безперебійність роботи, що досить важливо в сучасному світі де будь яка затримка від серверу, може коштувати великих затрат, як в часі, так і в грошовому ресурсі.

СПИСОК ВИКОРИСТАНОЇ ЛІТЕРАТУРИ

1. Red Hat Linux. Секреты проффесионала. [Текст] – 2004. Ст. 185- 188. Режим доступу : https://books.google.com.ua. –Дата доступпу: березень 2020.
2. Національна бібліотека ім. М. Баумана. [Електронний ресурс]. -2017.- Режим доступу : https://ru.bmstu.wiki/. –Дата доступпу: квітень 2019.
3. Многоуровневые системы клиент-сервер Валерий Коржов. [Текст]. Режим доступу : https://www.osp.ru/nets/1997/06/142618. /. –Дата доступпу: березень 2019.
4. Компоненты сетевого приложения. [Електронний ресурс]. Режим доступу : http://www.4stud.info/networking. –Дата доступпу: березень 2020.
5. Распределенные объектные технологии. [Текст]. Режим доступу : http://www.hpc-education.ru/files/lectures/radchenko/radchenko/.–Дата доступпу: березень 2020.
6. Desktop vs Browser not just another Thick vs Thin Client debate. [Електронний ресурс] – 2018. - Режим доступу : https://www.msafocus.com/news/desktop-vs-browser. –Дата доступпу: квітень 2020.
7. What is a Thin Client [Електронний ресурс]. Режим доступу : https://www.clearcube.com/posts/what-is-a-thin. –Дата доступпу: квітень 2020.
8. Немного о многопоточности. [Електронний ресурс]. Режим доступу : https://www.kv.by/archive/index2008441107.htm –Дата доступпу: квітень 2020.
9. Критическая секция. Critical section [Електронний ресурс]. Режим доступу : https://www.viva64.com/ru/t/0009/ –Дата доступпу: квітень 2020.

10. Архитектура клиент-сервер [Електронний ресурс]. Режим доступу : https://sergeygavaga.gitbooks.io/–Дата доступу: травень 2020.
11. Сети с выделенным сервером [Електронний ресурс]. Режим доступу : http://mydirectx.ru/seti/ Дата доступпу: квітень 2020.
12. Модель «Клиент-Сервер» [Електронний ресурс]. Режим доступу : http://window.edu.ru/ Дата доступпу: травень 2020.
13. Клиент-сервер. [Електронний ресурс]. Режим доступу : https://developer.mozilla.org/ Дата доступпу: травень 2020.
14. Що таке мережевий порт? [Електронний ресурс]. Режим доступу : https://2ip.ua/ua/blog/network-port Дата доступпу: травень 2020