Національний технічний університет України
«Київський політехнічний інститут імені Ігоря Сікорського»
Факультет інформатики та обчислювальної техніки
Кафедра інформаційних систем та технологій

ЛАБОРАТОРНА РОБОТА №9

з дисципліни «Технології розроблення програмного забезпечення»

Тема: «Взаємодія компонентів системи»

Виконав:
студент групи ІА-34
Сухоручкін Гліб

Перевірив:
Асистент кафедри ІСТ
Мягкий М.Ю.

Київ – 2025

Зміст:

Вступ

Метою цієї роботи є вивчити види взаємодії додатків (Client-Server, Peer-to-Peer, Serviceoriented Architecture), та реалізувати в проєктованій системі одну із архітектур.

Проєкторваною системою цієї лабороторної роботи є FTP-server (використовуючи state, builder, memento, template method, visitor, client-server).

FTP-сервер повинен вміти коректно обробляти і відправляти відповіді по протоколу FTP, з можливістю створення користувачів (з паролями) і доступних їм папок, розподілу прав за стандартною схемою (rwe), ведення статистики з'єднань, обмеження максимальної кількості підключень і максимальної швидкості поширення глобально і окремо для кожного облікового запису.

Теоретичні відомості

Клієнт-серверні додатки являють собою найпростіший варіант розподілених додатків, де виділяється два види додатків: клієнти (представляють додаток користувачеві) і сервери (використовується для зберігання і обробки даних). Розрізняють тонкі клієнти і товсті клієнти.

Тонкий клієнт – клієнт, який повністю всі операції (або більшість, пов'язаних з логікою роботи програми) передає для обробки на сервер, а сам зберігає лише візуальне уявлення одержуваних від сервера відповідей. Грубо кажучи, тонкий клієнт – набір форм відображення і канал зв'язку з сервером. Прикладом тонкого клієнта є класичні Web-застосунки.

У такому варіанті використання майже все навантаження лягає на сервер або групу серверів.

Перевагою таких моделей є простота розгортання, тому що оновлювати потрібно лише сервери і в результаті клієнти з наступними запитами автоматично будуть працювати з оновленою системою.

Товстий клієнт – антипод тонкого клієнта, більшість логіки обробки даних містить на стороні клієнта. Це сильно розвантажує сервер. Сервер в таких випадках зазвичай працює лише як точка доступу до деякого іншого ресурсу (наприклад, бази даних) або сполучна ланка з іншими клієнтськими комп'ютерами. Перевагою такого підходу є менші вимоги до серверної частини. Також перевагою, при певному підході до реалізації, є можливість працювати клієнтам без тимчасового доступу до серверу. Прикладом товстого клієнта можна назвати мобільні застосунки, або десктоп застосунки. Наприклад, Evernote, Viber, MS Outlook, комп’ютерні антивіруси, ігри, що потребують інсталяції (The Sims, GTA, …) та інші.

Проміжним варіантом можна назвати SPA (Single Page Application) – це товсті Web-клієнти, які при старті кожен раз завантажуються з сервера, а надалі працюють з сервером через web-API. З одного боку більшу частину логіки вони відпрацьовують на клієнтський стороні, за рахунок чого зменшується серверне навантаження. Також оновлення простіше ніж для товстих клієнтів. Але такі застосунки не працюють, якщо сервер не доступний.

Клієнт-серверна взаємодія, як правило, організовується за допомогою 3-х рівневої структури: клієнтська частина, загальна частина, серверна частина.

Оскільки велика частина даних загальна (класи, використовувані системою), їх прийнято виносити в загальну частину (middleware) системи.

Клієнтська частина містить візуальне відображення і логіку обробки дії користувача; код для встановлення сеансу зв'язку з сервером і виконання відповідних викликів.

Серверна частина містить основну логіку роботи програми (бізнес-логіку) або ту її частину, яка відповідає зберіганню або обміну даними між клієнтом і сервером або клієнтами.

Хід роботи

Діаграма класів

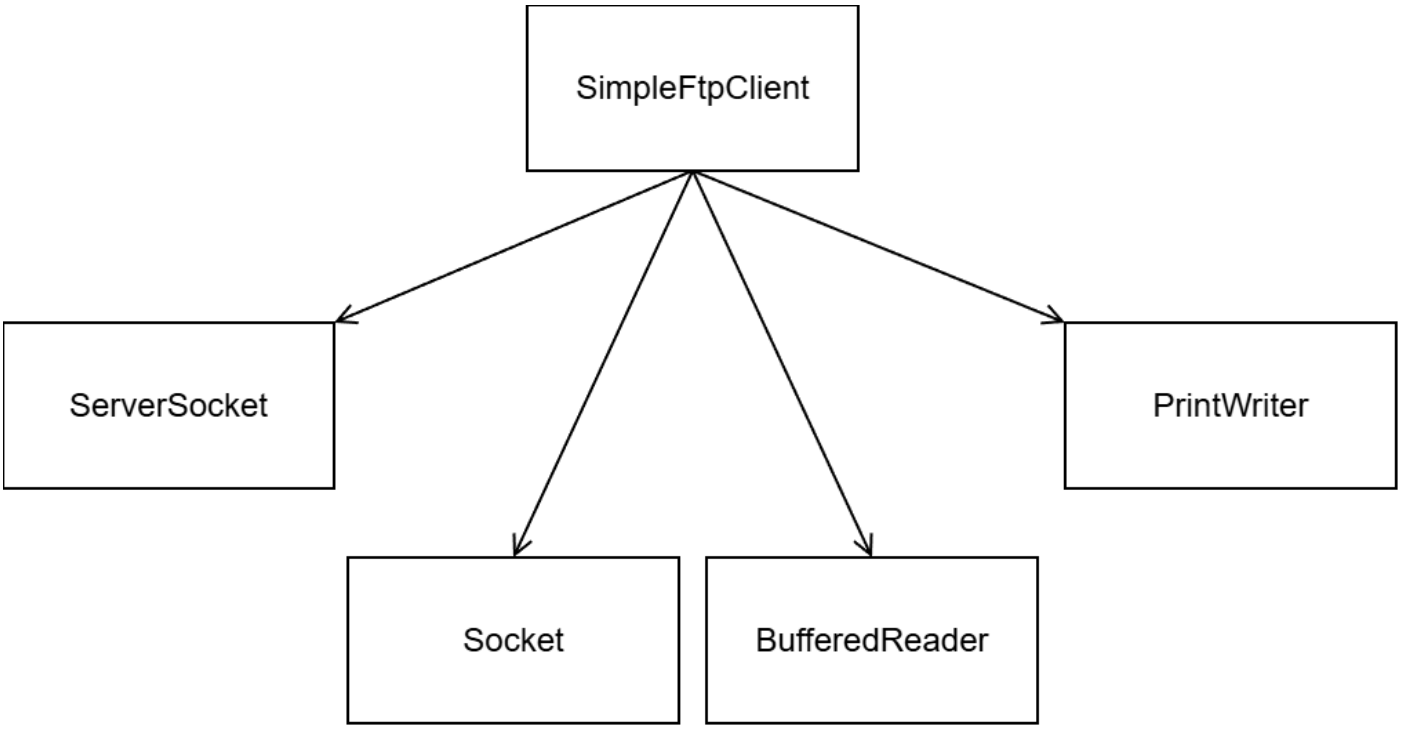

Рис 1. Діаграма класів FTP-сервера.

Пояснення діаграми классів

FtpServer – це головний клас сервера, який піднімає FTP-службу. Він створює ServerSocket на потрібному порту й запускає цикл очікування підключень. Коли з’являється новий клієнт, сервер приймає його через accept() і отримує окремий об’єкт Socket, що відповідає одному TCP-з’єднанню.

Socket дає два потоки: вхідний і вихідний. На основі вхідного потоку сервер створює BufferedReader, щоб зручно читати текстові команди FTP построчно методом readLine(). На основі вихідного потоку він створює BufferedWriter, щоб відправляти клієнту коди й текстові відповіді (наприклад, 220 ..., 331 ..., 230 ...) як звичайні рядки.

Таким чином, FtpServer керує всім життєвим циклом мережевої взаємодії: ServerSocket слухає порт і створює Socket, Socket забезпечує канал зв’язку з конкретним клієнтом, а BufferedReader/BufferedWriter перетворюють байтові потоки в зручний текстовий обмін командами й відповідями FTP.

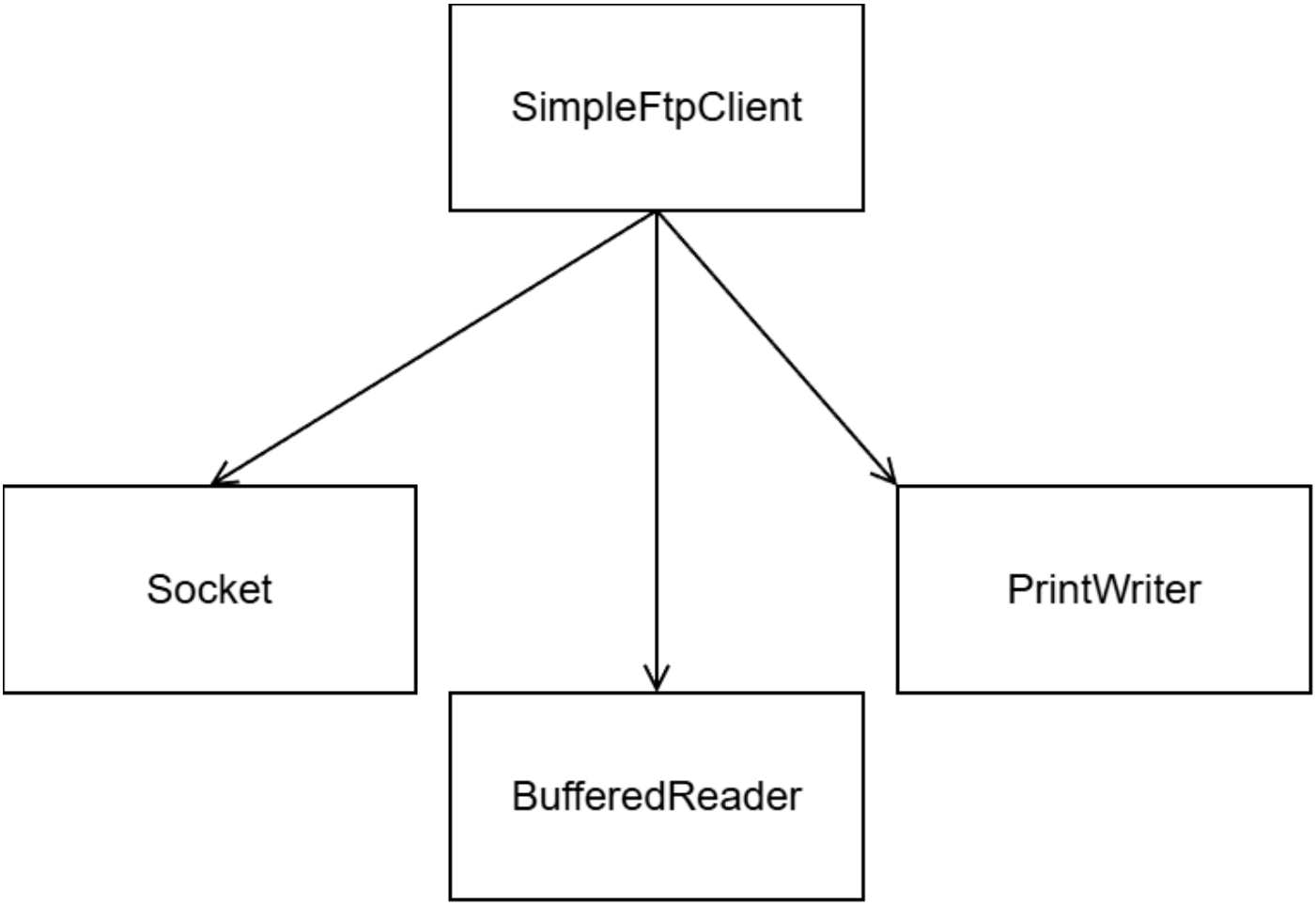

Рис 2. Діаграма класів FTP-клієнта.

Пояснення діаграми классів

SimpleFtpClient – це клас простого консольного FTP-клієнта. Він запускається з main(...), отримує від користувача хост і порт (або бере значення за замовчуванням) і відповідає за встановлення з'єднання з FTP-сервером. Усі мережеві об'єкти (Socket, BufferedReader, PrintWriter) створюються й використовуються саме всередині цього класу.

Socket – це TCP-канал зв'язку між клієнтом і FTP-сервером. SimpleFtpClient створює його через new Socket(host, port), після чого через цей сокет надсилає команди й отримує відповіді. Поки сокет відкритий, клієнт вважається підключеним до сервера; при виході або помилці SimpleFtpClient його закриває.

На основі потоків сокета клієнт будує дві текстові обгортки: BufferedReader для читання рядків відповіді від сервера та PrintWriter для надсилання текстових FTP-команд. BufferedReader читає вхідний потік socket.getInputStream() построчно й виводить відповіді на екран, а PrintWriter пише у вихідний потік socket.getOutputStream() усі введені користувачем команди (USER, PASS, LIST тощо). Таким чином, SimpleFtpClient перетворює роботу із сирими байтами сокета на зручний текстовий

Вихідний код

https://github.com/GlebTiKsTRPZ/lab9

Відповіді на питання до лабораторної роботи

1. Що таке клієнт-серверна архітектура?

Клієнт-серверна архітектура – це варіант розподілених додатків, де виділяють клієнти (представляють програму користувачу) і сервери (зберігають та обробляють дані).

2. Розкажіть про сервіс-орієнтовану архітектуру.

Сервіс-орієнтована архітектура (SOA) – це модульний підхід, заснований на розподілених, слабо пов'язаних сервісах зі стандартизованими інтерфейсами, які взаємодіють за стандартизованими протоколами (зазвичай веб-сервіси через HTTP з SOAP або REST).

3. Якими принципами керується SOA?

SOA керується принципами слабкого зв'язування розподілених сервісів, стандартизованих інтерфейсів і протоколів, а також реєстрації сервісів у спеціальних службах, щоб їх можна було знаходити й перевикористовувати.

4. Як між собою взаємодіють сервіси в SOA?

Сервіси в SOA взаємодіють між собою шляхом обміну повідомленнями, без створення прямого спільного доступу, наприклад, до однієї бази даних.

5. Як розробники взнають про існуючі сервіси і як робити до них запити?

Розробники дізнаються про сервіси через їх реєстрацію на спеціальних сервісах-реєстрах, де можуть знайти потрібний сервіс і викликати його через стандартизований інтерфейс (часто HTTP+SOAP/REST).

6. У чому полягають переваги та недоліки клієнт-серверної моделі?

Переваги клієнт-серверної моделі: простота розгортання тонких клієнтів (оновлюється лише сервер) і можливість перенесення частини логіки на клієнт у товстих/SPA-варіантах, що зменшує навантаження на сервер. Недоліки: велике навантаження на сервер при тонких клієнтах і неможливість роботи застосунків типу SPA без доступу до сервера.

7. У чому полягають переваги та недоліки однорангової моделі взаємодії?

Переваги однорангової (P2P) моделі: децентралізація, рівноправність вузлів і розподіл ресурсів між учасниками мережі. Недоліки: складність забезпечення безпеки, синхронізації даних і ефективного пошуку ресурсів при зростанні кількості вузлів.

8. Що таке мікро-сервісна архітектура?

Мікросервісна архітектура – це підхід, за якого серверний додаток будується як набір малих служб, кожна з яких виконується у власному процесі, реалізує свою бізнес-логіку в обмеженому контексті та розробляється й розгортається автономно.

9. Які протоколи використовуються для обміну даними в мікросервісній архітектурі?

Для обміну даними в мікросервісній архітектурі використовують протоколи HTTP/HTTPS, WebSockets та AMQP.

10. Чи можна назвати підхід сервіс-орієнтованою архітектурою, коли ми в проєкті між шаром веб-контролерів та шаром доступу до даних реалізуємо шар бізнес-

логіки у вигляді сервісів?

SOA – це розподілена архітектура зі слабо пов’язаними сервісами, що спілкуються через стандартизовані протоколи (веб-служби, HTTP, SOAP/REST). Просте виділення шару бізнес-логіки у вигляді «сервісів» між веб-контролерами та шаром доступу до даних у межах одного застосунку без розподіленої взаємодії не відповідає повному визначенню SOA з конспекту.

Висновок

У цій лабораторній роботі було реалізовано патерн «Visitor» для ієрархії станів FTP-сервера, що дозволило відокремити побудову текстового статусу від структури станів і спростити подальше розширення функціональності без зміни їхнього коду.

Клієнт-серверна архітектура для FTP-сервера є оптимальною, оскільки дає централізований контроль доступу й бізнес-правил, полегшує масштабування та оновлення сервера й таким чином знижує витрати на підтримку системи.