Міністерство освіти і науки України
Національний технічний університет України
“Київський політехнічний інститут імені Ігоря Сікорського”
Факультет інформатики та обчислювальної техніки
Кафедра інформаційних систем та технологій

Лабораторна робота №9

Технології розроблення програмного забезпечення

РІЗНІ ВИДИ ВЗАЄМОДІЇ ДОДАТКІВ: CLIENT-SERVER, PEER-TO-PEER, SERVICE-ORIENTED ARCHITECTURE

Варіант №14

Виконав:

студент групи ІА-14,

Міщук Максим Дмитрович

Перевірив:

Мягкий М.Ю.

Київ 2023

Тема роботи: Різні види взаємодії додатків: client-server, peer-to-peer, service-oriented architecture.

Вхідні дані:

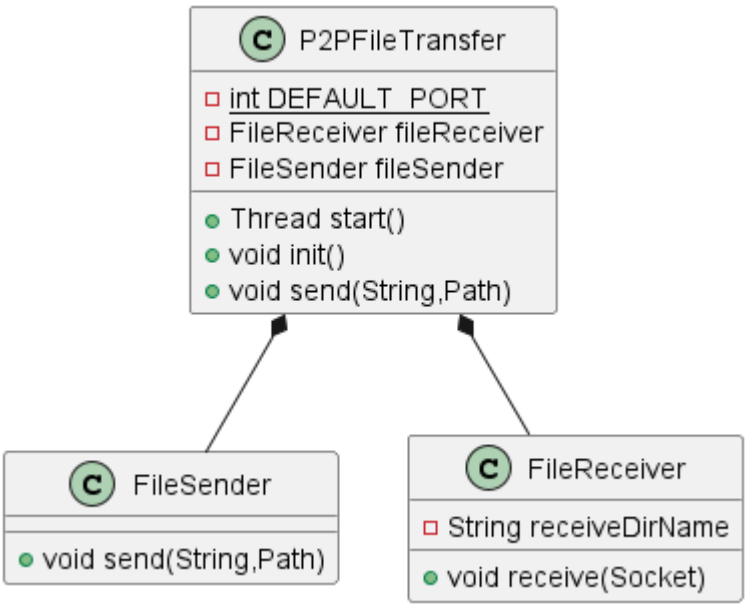
..14 Архіватор (strategy, adapter, factory method, facade, visitor, p2p)

Архіватор повинен являти собою візуальний додаток з можливістю створення і редагування архівів різного типу (.tar.gz, .zip, .rar, .ace) - додавання/ видалення файлів / папок, редагування метаданих (по можливості), перевірка checksum архівів, тестування архівів на наявність пошкоджень, розбиття архівів на частини.

Хід роботи:

Архітектура peer-to-peer (p2p) - це модель комп'ютерної мережі, в якій кожен комп'ютер чи пристрій, що приєднаний до мережі, може виконувати роль і сервера, і клієнта одночасно. У цьому типі мережі відсутні централізовані сервери, котрі контролюють та керують комунікацією між вузлами. Замість цього, вузли взаємодіють напряму один з одним.

На основі даної архітектури було розроблено файлообмінювач, що встановлюється на декілька комп’ютерів та резервує під себе один порт й дає змогу за IP-адресою встановлювати з’єднання та відправляти файли на інші клієнти. Кожин користувач може як приймати та надсилати файле, головне знати адресу за якою відправляти. Також користувачі можуть самі обирати коли вмикати й вимикати їх, за для потреби й економії ресурсів системи.



Діаграма класів

Метод *init()* відповідає за запуск серверної частини клієнта. Коли з'єднання з клієнтом встановлено, дані передаються класу FileReceiver, що вже обробляє запит. Дані будуть зберігатися в папці отриманих повідомлень. Метод *send()* – за клієнтську частину, першим параметром поступає адреса, а другим файл для передачі. Дані так само передаються відповідному класу *FileSender*, що за заданим файлом дістане його мета-дані й основні дані, та надсилає їх на серверну частину іншого клієнта.

Код реалізації:

```
public class P2PFileTransfer {
    private static final Logger logger = LogManager.getLogger(P2PFileTransfer.class.getSimpleName());
    protected static final ResourceManager resourceManager = ResourceManager.INSTANCE;
    private static final int DEFAULT_PORT = 44305;
    private final FileReceiver fileReceiver = new FileReceiver("received");
    private final FileSender fileSender = new FileSender();

    private static class FileSender {
        public void send(String address, Path path) throws IOException {
            try (
                    InputStream is = Files.newInputStream(path);
                    Socket connection = new Socket(address, DEFAULT_PORT);
                    DataOutputStream dos = new DataOutputStream(connection.getOutputStream())
            ) {
                String fileName = path.getFileName().toString();

                BasicFileAttributes bfa = Files.readAttributes(path, BasicFileAttributes.class,
LinkOption.NOFOLLOW_LINKS);
                long creationTime = bfa.creationTime().toMillis();
                long lastModifiedTime = bfa.lastModifiedTime().toMillis();

                dos.writeUTF(fileName);
                dos.writeLong(creationTime);
                dos.writeLong(lastModifiedTime);
                dos.writeLong(Files.size(path));

                IOUtils.copy(is, dos, 4096);
            }
        }
    }

    private record FileReceiver(String receiveDirName) {

        public void receive(Socket clientSocket) throws IOException {
            try (
                    DataInputStream dis = new DataInputStream(clientSocket.getInputStream())
            ) {
                Path receiveDir = initReceiveDir();
                receiveDir = initTodayDir(receiveDir);

                String fileName = dis.readUTF();
                Path filePath = initReceivedFile(dis, receiveDir, fileName);
                FileTime lastModifiedTime = FileTime.fromMillis(dis.readLong());

                long fileSize = dis.readLong();
                byte[] buffer = new byte[4096];
                try (OutputStream os = Files.newOutputStream(filePath, StandardOpenOption.WRITE)) {
                    int bytes;
                    while (fileSize > 0 && (bytes = dis.read(buffer, 0, (int) Math.min(buffer.length,
fileSize))) != -1) {
                        os.write(buffer, 0, bytes);
                        fileSize -= bytes;
                    }
                }
                Files.setAttribute(filePath, "basic:lastModifiedTime", lastModifiedTime,
LinkOption.NOFOLLOW_LINKS);
                logger.info(String.format(resourceManager.getString("serverSaved"), fileName,
filePath.toAbsolutePath()));
            }
        }

        private Path initReceiveDir() throws IOException {
            return Files.createDirectories(Path.of(receiveDirName));
        }

        private Path initTodayDir(Path receiveDir) throws IOException {
```

```
            SimpleDateFormat s = new SimpleDateFormat("yyyy-MM-dd");
            String dirName = s.format(new Date());
            Path todayDir = receiveDir.resolve(dirName);
            return Files.createDirectories(todayDir);
        }

        private Path initReceivedFile(DataInputStream dis, Path receiveDir, String fileName) throws
IOException {
            Path filePath = receiveDir.resolve(fileName);
            filePath = FileUtils.getFreePath(filePath);

            FileTime creationTime = FileTime.fromMillis(dis.readLong());


            Path result = Files.createFile(filePath);
            Files.setAttribute(filePath, "basic:creationTime", creationTime, LinkOption.NOFOLLOW_LINKS);

            return result;
        }

    }

    public Thread start() {
        Thread serverThread = new Thread(() -> {
            logger.info(resourceManager.getString("serverStarted"));
            while (!Thread.interrupted()) {
                try {
                    init();
                } catch (IOException e) {
                    throw new RuntimeException(e);
                }
            }
            logger.info(resourceManager.getString("serverStopped"));
        });
        serverThread.start();
        return serverThread;
    }

    public void init() throws IOException {
        try (ServerSocket serverSocket = new ServerSocket(DEFAULT_PORT)) {
            logger.info(resourceManager.getString("serverWaiting"));
            serverSocket.setSoTimeout(15_000);
            try (Socket client = serverSocket.accept()) {
                logger.info(resourceManager.getString("serverConnected"));
                fileReceiver.receive(client);
            } catch (SocketTimeoutException ex) {
                logger.info(resourceManager.getString("serverTimeout"));
            }
        }
    }

    public void send(String address, Path path) throws IOException {
        logger.info(String.format(resourceManager.getString("clientSend"), path.getFileName()));
        fileSender.send(address, path);
    }
}
```

P2PFileTransfer.java

Як вже було сказано, метод *init()* використовується для початку прослуховування порту, можливість встановлення з'єднання. Під час цього етапу, потік виконання програми блокується, що унеможливлює роботу в цей момент з іншими компонентами застосунку. Для того щоб вивести потік з цього стану, встановлюється змінна *soTimeout* , що вказує час через який закінчиться прослуховування порту та викинеться виключення *SocketTimeoutException*, за відсутності наявних підключень.

Окрім вище описаного методу, є ще один - *start()*. Він створює й повертає посилання на окремий потік, в середині якого запускає метод *init()*. Після проходження часу *soTimeout*, або обробки отриманого запиту, сервер знову переходить в стан очікування нових з'єднань. Система буде перебувати в цьому стані до тих пір, доки виконання потоку не буде перервано (реалізовано механізм переривання).

Висновок:

Отриманий в результаті застосунок, може перекидати файли між користувачами напряму, без проміжного серверу. Також сумісне використання файлообмінювача та архіватора дозволить надсилати великі об'єми файлів з стиснутій формі, що зменшить навантаження на мережу в цілому.

Посилання на репозиторій: [Maxim-Mishchuk/trpz-archiver at develop (github.com)]