1. Introducere

Scopul acestui proiect este implementarea unui sistem client–server concurent pentru vânzarea de bilete la spectacole, care demonstrează utilizarea corectă a mecanismelor de programare paralelă și distribuită, precum: thread pool, futures, sincronizare, task-uri periodice și verificarea consistenței datelor.

2. Arhitectura Sistemului

Aplicația este organizată pe o arhitectură client–server, structurată pe trei niveluri:

Nivel de prezentare

Serverul și clienții, care comunică prin socket-uri TCP/IP.

Nivel de logică de business

Componenta ConcertHall, responsabilă de gestionarea rezervărilor și a vânzărilor.

Nivel de persistență

DatabaseManager, bazat pe SQLite, care asigură stocarea persistentă a datelor.

Comunicarea se realizează prin serializarea obiectelor Java (Request și Response) pe portul 8080.

Serverul utilizează:

un **ExecutorService** cu thread pool fix pentru procesarea cererilor clienților;

un **ScheduledExecutorService** pentru task-uri periodice (verificare consistență, curățare rezervări expirate și shutdown automat).

3. Fluxul de Funcţionare

La pornire, serverul inițializează baza de date, creează cele trei spectacole și încarcă starea existentă în memorie. Clienții se conectează la server și trimit cereri de cumpărare la intervale regulate.

Procesarea unei cereri are două etape:

Rezervarea temporară a locurilor, dacă acestea sunt disponibile.

Confirmarea plății, cu verificarea expirării rezervării.

Rezervările neconfirmate sunt invalidate automat după 10 secunde. Răspunsul fiecărei cereri este transmis clientului, care afișează rezultatul și continuă execuția.

4. Decizii de Sincronizare

Sincronizarea este realizată printr-un **ReentrantLock** global în clasa ConcertHall. Acesta protejează toate operațiile critice care modifică starea sistemului, garantând:

atomicitatea operațiilor complexe;

consistența dintre memorie și baza de date;

eliminarea posibilității de deadlock.

Variabila care stochează **soldul total** este declarată **volatile**, asigurând vizibilitatea modificărilor între thread-uri. Structurile de date principale sunt implementate cu **ConcurrentHashMap**, permițând acces concurent sigur și iterații fără erori.

5. Utilizarea Thread Pool și Futures

Serverul folosește un **thread pool fix** pentru procesarea cererilor clienților, eliminând overhead-ul creării dinamice de thread-uri și limitând consumul de resurse.

Pe partea de **client**, cererile sunt executate asincron folosind **Future**, cu timeout explicit. Acest mecanism previne blocarea clienților în cazul unor întârzieri sau erori ale serverului și crește robustețea sistemului.

Task-urile periodice (verificare consistență, cleanup și shutdown) sunt implementate cu **ScheduledExecutorService**.

6. Verificarea Consistenței

La intervale de 5 sau 10 secunde, sistemul verifică:

- concordanța locurilor vândute între memorie și baza de date;
- corectitudinea soldului total;
- consistența matematică a vânzărilor.

Toate verificările efectuate au raportat status CORECT, demonstrând absența race condition-urilor și sincronizarea completă între componente.

7. Analiza Performanței

RULARE: ./run_tests_cerinta.sh

Metricile colectate includ:

- timpul mediu de răspuns;
- throughput-ul (cereri pe secundă);
- rata de succes a cererilor.

Rezultatele obținute:

timp mediu de răspuns: aproximativ 125–128 ms
throughput: aproximativ 8–9 cereri/secundă
rată de succes: 85–87%

Performanța este limitată în principal de lock-ul global și de scrierile serializate în SQLite.