SmartParkingAssistant

Grozea Tatiana 2B1

December 2023

1 Introducere

Proiectul va fi scris în C și va ajuta șoferii să găsească un loc disponibil de parcare în timp real. Parcarea ar fi una de tip închis(cu mai multe intrări), fiecare loc de parcare va avea un anumit statut (liber, ocupat, bronat) iar datele acestea vor fi furnizate de camere de supraveghere și senzori simulați(șoferii fiind liberi să plece când doresc). Când șoferul apare la una din intrări verifică dacă există locuri libere, dacă nu sunt poate alege să plece.

- ***Serverul*** va oferi informațiile clienților și va folosi principiul conectării concurente pentru a putea primi mai mulți clienți, se va conecta la un anumit port și va aștepta clienții să se conecteze pe acel port.

- ***Clienții*** pentru a putea accesa detaliile asupra locurilor de parcare de la server se conectează la acesta, iar fiecare client va avea firul său nou de execuție care nu va interfera cu cele ale celorlalți clienți.

2 Tehnologii Aplicate

Pentru a avea un proiect cât mai atractiv și bine structurat se vor utiliza diferite tehnologii potrivite temei:

2.1 Protocolul de Comunicare: TCP/IP

Motivație: Utilizarea TCP/IP (Transmission Control Protocol/ Internet Protocol)asigură o comunicație fiabilă între client și server, asigurându-se că nici o dată nu este pierdută, lucru ce nu ar fi posibil dacă ar fi implementat cu UDP(User Datagram Protocol) care este mai avantajos din perspectiva latenței, dar anume fiind un proiect de tipul Smart Parking, unde datele despre disponibilitatea locurilor de parcare trebuie să ajungă fără pierderi, TCP este o alegere potrivită.

2.2 Socket-uri în C

Motivație: Folosirea funcțiilor și structurilor din biblioteca de socket-uri în limbajul C permite inițierea și gestionarea conexiunilor de rețea. Astfel, serverul poate asculta pe un anumit port și să accepte conexiuni de la mai mulți clienți(șoferii care ar căuta un loc de parcare), astfel reușind să implementăm TCP-ul concurent mai ușor.

2.3 Comunicarea Concurentă: Thread-uri în C

Motivație: Utilizarea thread-urilor în limbajul C permite gestionarea concurentă a mai multor clienți simultan(deoarece pot fi mai mulți șoferi ce au nevoie de un loc de parcare). Fiecare client poate fi tratat într-un fir de execuție separat pentru a gestiona comunicarea asincronă și pentru a evita blocarea întregului server.

2.4 Bază de Date: SQLite

Motivație: SQLite este o bază de date ușoară, încorporată și adecvată pentru proiecte mici sau mijlocii. Deoarece proiectul Smart Parking nu necesită o bază de date complexă, SQLite oferă o soluție eficientă pentru stocarea și interogarea datelor care vor fi actualizate în funcție de locurile de parcare disponibile, oferite de camerele de supraveghere sau de senzori sau informația de la clienți.

3 Structura Aplicației

Pentru a dezvolta o structură coerentă a aplicației SmartParkingAssistant, este important să avem bine structurate conceptele de bază folosite în modelare:

Server:

- Implementat un socket care ascultă, așteaptă și primește conexiuni de la clienți(șoferii) la un anumit port (de preferat peste 1024 pentru a nu exista conflicte sau a apărea blocajul ca portul să fie deja ocupat de funcțiile sistemului);
- Găzduiește logica principală a aplicației;
- Comunică cu baza de date pentru a obține informații despre disponibilitatea locurilor de parcare;

Client:

- Implementat cu un socket care se va conecta la server printr-un port;
- Interfața prin care utilizatorii interacționează cu aplicația, tastează comenzile și așteaptă răspunsurile;
- Solicită informații despre disponibilitatea locurilor de parcare și primește actualizări în timp real;
- Poate alege un loc de parcare propus, sau poate alege să plece.

Socket-uri:

- Permite inițierea și gestionarea conexiunilor de rețea;
- Serverul poate asculta pe un anumit port și să accepte conexiuni de la mai mulți clienți.

Thread-uri:

- Folosite pentru a gestiona comunicarea concurentă cu mai mulți clienți simultan;
- Fiecare client poate fi tratat într-un fir de execuție separat.

Bază de Date SQLite:

- Stochează informații despre locurile de parcare, starea lor și alte detalii relevante;
- Folosită pentru a salva și accesa datele în mod eficient.

Senzori și Camere Inteligente (Simulate):

- Generează date despre starea locurilor de parcare (ocupate/libere/bronate).

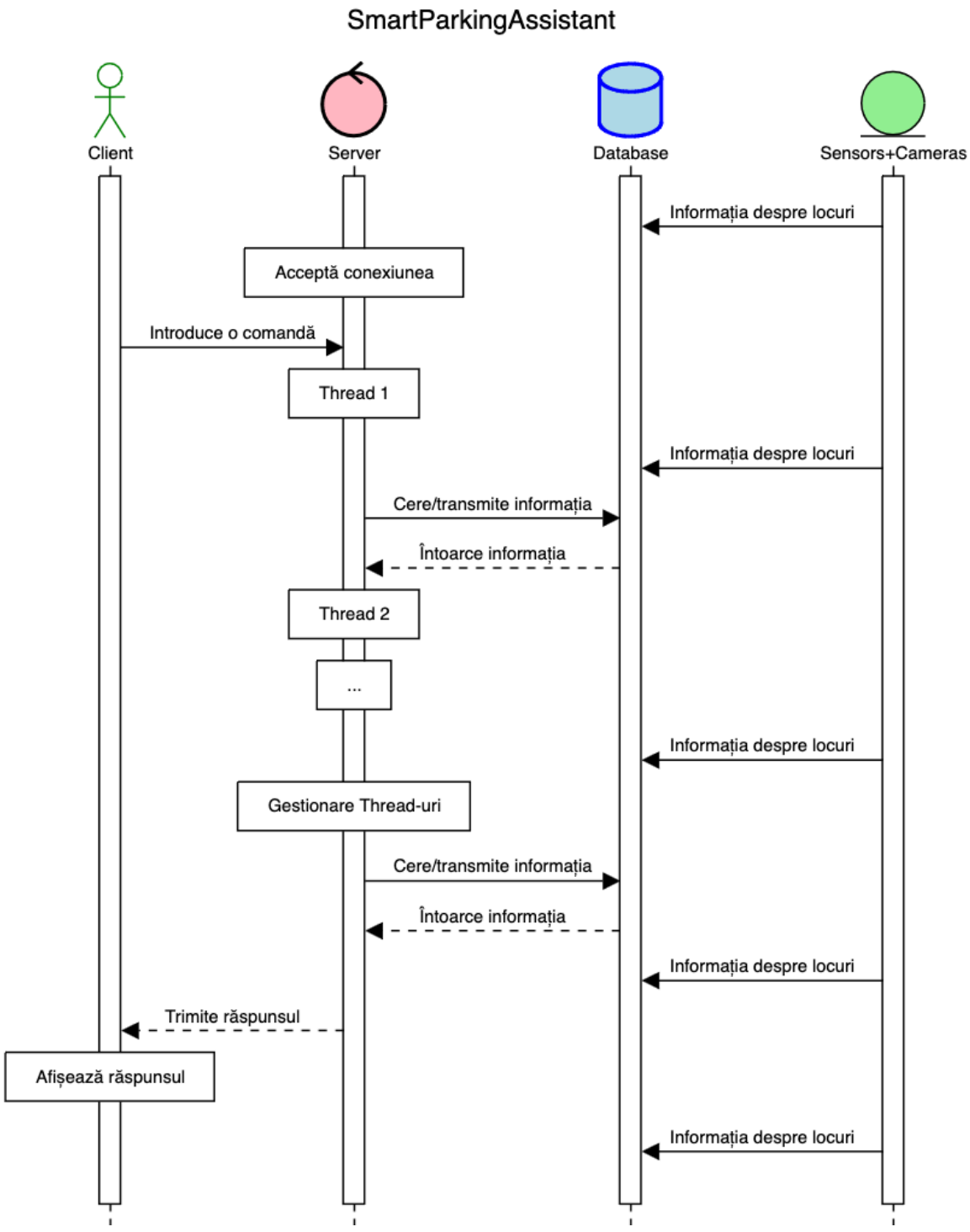

Figure 1: Un model de diagramă a aplicației

4 Aspecte de Implementare

Protocolul utilizat la nivelul aplicației este TCP/IP concurent(cu thread-uri):

Cerințe de conexiune:

- Clientul se conectează la server pentru a primi informații despre disponibilitatea locurilor de parcare.

- Serverul acceptă conexiuni multiple de la diferiți clienți.

Comunicare între client și server:

- Clientul solicită informații despre disponibilitatea locurilor de parcare.

- Prin socket se transmite serverului solicitarea clientului.

- Serverul procesează solicitarea și furnizează datele corespunzătoare (ocupate/libere/bronate) (căutând în baza de date asociată).

- La un interval de timp senzorii și camerele reînnoiesc informația din baza de date referitoare la locurile libere de parcare.

- Prin socket se transmite înapoi răspunsul clientului.

Protocolul comenzilor fiind:

- connect
- available parking spots
- choose spot x
- call
- exit

Scenarii reale de utilizare:

- Numerele arată starea fiecărui loc de parcare (liber - 0, ocupat - 1, bronat - 2).

- Pot veni două mașini, doar un loc liber de parcare, prima care a fost mai rapidă ocupă locul, cealaltă va fi anunțată că s-au ocupat toate locurile și va alege să plece.

Un prim aspect de implementare a aplicației ar fi următoarea bucată de cod:

int on = 1;

$setsockopt(sd, SOL\_SOCKET, SO\_REUSEADDR, \&on, sizeof(on))$;

care folosește *setsockopt*, o funcție care permite configurarea opțiunilor pentru un socket, iar utilizarea $SO\_REUSEADDR$ este utilă atunci când se dezvoltă servere, iar serverul trebuie să pornească și să se oprească frecvent(cazul parcării, nu are un flux constant de clienți). Aceasta va permite să reluați utilizarea aceleiași adrese de socket într-un timp mai scurt după închiderea serverului.

O altă bucată de cod specifică pentru aplicația mea ar fi meniul de comenzi ce apare oricărui client conectat pe stdout:

$printf("----------MENU----------")$;
$printf("----------")$;
$printf("connect")$;
$printf("available parking spots")$;
$printf("choose spot x (where x is a number)")$;
$printf("call")$;
$printf("exit")$;

O altă bucată de cod inovativă este afișarea în server a numărului și id-ului thread-ului fiecărui client, astfel distingând mai ușor clienții între ei:

$printf("S: Thread ID for client \%d: \%ld", client, thread\_id)$;

care va utiliza structura ThreadArgs, primul membru indicând numărul clientului, iar cel de-al doilea membru indicând id-ul thread-ului:

struct ThreadArgs
{

int client;
pthread_t thread_id;
};

5 Concluzii

Potențialele îmbunătățiri ale proiectului ar fi:

- Preț pentru fiecare loc de parcare, ce ar fi achitat la intrare pentru timp nelimitat(ar exista câteva premium cu o taxă mai ridicată, câteva medium cu taxă medie și câteva low cu o taxă mai mică);
- Vizualizare grafică(GUI) în timp real a disponibilității locurilor de parcare;
- Posibilitatea de a rezerva online un loc de parcare disponibil;
- Notificări pe E-mail/SMS cu confirmarea alegerii locului de parcare/plății;
- Indicații de navigare către locul de parcare selectat.

6 Referințe Bibliografice

https : *//profs.info.uaic.ro/ computernetworks/files/homework2_ro.txt*
https : *//profs.info.uaic.ro/ computernetworks/cursullaboratorul.php*
https : *//sites.google.com/view/fii − lab − retele − de − calculatoare/laboratoare*
https : *//en.wikipedia.org/wiki/Transmission_Control_Protocol*
https : *//www.youtube.com/watch?v = OTwp3xtd4dg*
https : *//en.wikipedia.org/wiki/Parking_lot*
https : *//sequencediagram.org*