Smart Parking Assistant

Buzdea Ștefan

Facultatea de Informatică Iași, Universitatea Alexandru Ioan Cuza, România
https://www.info.uaic.ro/

1 Introducere

Smart Parking Assistant are rolul, așa cum sugerează și numele, de a ajuta șoferii să găsească locuri de parcare libere în momentul căutării. Dezvoltarea aplicației client/server are obiectivul de găsi o soluție rapidă într-o parcare greu de parcurs. Prin colectarea datelor de la senzorii locurilor de parcare, manipulăm informațiile afișate șoferilor, care pot ocupa orice loc liber, fapt după care se va actualiza harta parcării în funcție de disponibilitate.

Această aplicație este dedicată deținătorilor de permis auto ce caută frecvent locuri de parcare.

Proiectul este dezvoltat pentru o singură parcare, dar poate fi extins pe mai multe. Șoferul intră în aplicație și i se arată harta parcării cu locurile libere și ocupate, fiecare marcate într-un anume fel. După care, acesta parchează într-un loc liber (fără a interacționa cu aplicația, ca să nu piardă timp), iar senzorul locului pe care s-a pus semnalează ocuparea. Drept urmare, harta se va actualiza pentru toți șoferii ce sunt în aplicație în momentul respectiv. Același lucru se întâmplă și când anumiți șoferi părăsesc parcarea.

Astfel, eficiența crește din punctul de vedere al timpului. În plus, cu această abordare, se optimizează consumul psihic al conducătorilor auto, reducându-se stresul, frustrarea sau starea de nervi.

2 Tehnologii Aplicate

Se va folosi o metodă de comunicare cu câte un *thread* individual pentru fiecare client. Serverul va trimite harta fiecărui șofer care intră în aplicație. Întrucât nu este permisă pierderea datelor în contextul locurilor libere de parcare, iar acestea trebuie transmise în ordinea primirii de către server, mai departe, se va folosi protocolul de transport TCP.

În cazul în care am fi folosit protocolul UDP, am fi riscat să pierdem date importante, care ar fi putut induce șoferii în eroare, provocând astfel stări de frustrare, în loc de a le îndepărta (obiectivul aplicației).

Același tip de protocol va fi util și în comunicarea dintre senzori și server, deoarece este necesară acțiunea de confirmare a ocupării locurilor (de la senzor) pentru a asigura actualizarea corectă a disponibilității în timp real.

Deci, în ambele comunicări ale serverului (cea cu șoferii și cea cu senzorii), va fi folosit TCP.

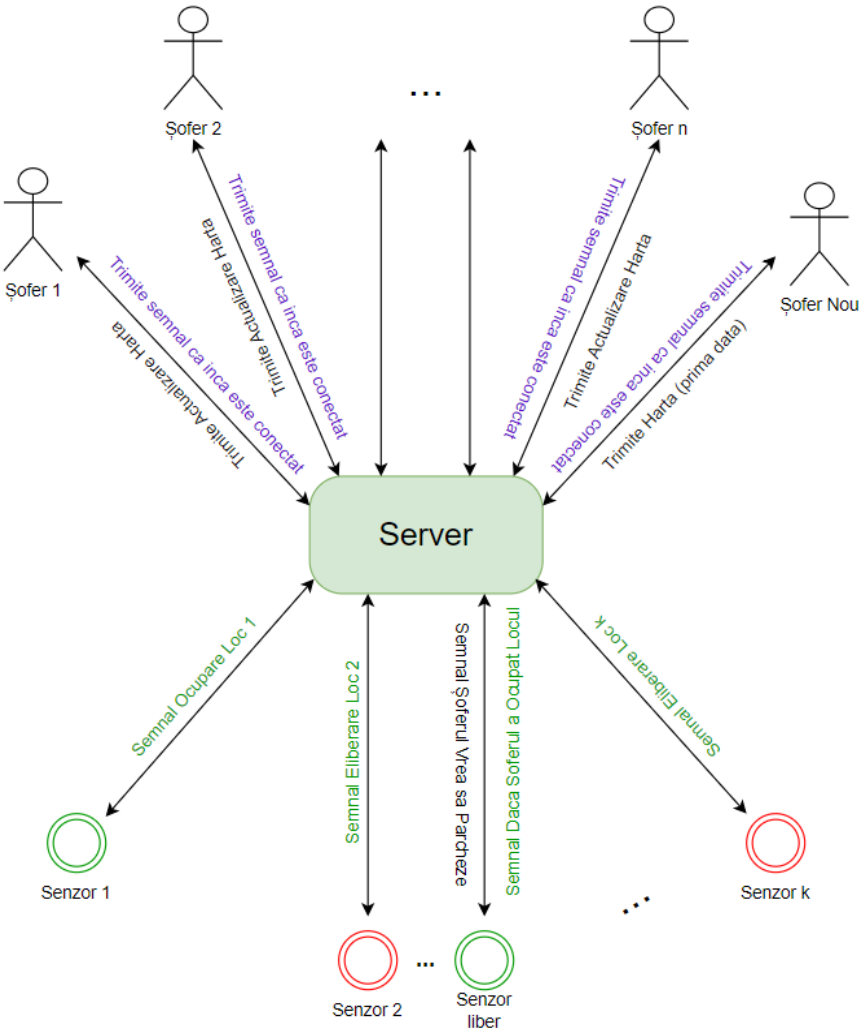
3 Structura Aplicației

Conceptul de client/server este folosit în două direcții în această aplicație.

Primul caz constă în comunicarea serverului cu șoferul. La prima vedere, conducătorul auto pare că nu interacționează deloc cu serverul, însă, în spate, clientul trimite mesaje constant către server. Aceste mesaje au rolul esențial de a-i spune serverului că șoferul încă este conectat la aplicație. Serverul va trimite noului intrat în aplicație harta pentru a se orienta spre un loc liber. Când există o modificare în parcare, serverul actualizează organizarea locurilor și trimite noua hartă tuturor clienților care sunt în aplicație. Această actualizare va fi indicată de către un mesaj transmis de la un senzor, când acesta semnalează o schimbare.

Al doilea caz apare în comunicarea dintre senzori și server, în momentul în care senzorii transmit disponibilitatea locurilor. Orice loc de parcare are senzorul său propriu. Fiecare senzor îi spune serverului când se ocupă sau când se eliberează locul atribuit lui, iar serverul primește semnalul schimbării și actualizează parcarea.

Conceptul va fi reprezentat mai amplu în diagrama de pe următoarea pagină.



Conceptul de client/server în aplicație

Interpretare Diagramă:

Senzorii colorați cu roșu sunt marcați ca ocupați, iar cei colorați cu verde sunt marcați ca fiind liberi.

Acțiunile de pe săgeți sunt colorate cu negru, verde sau mov. Cele cu negru sunt transmise de server, cele cu verde sunt transmise de senzori, iar cele cu mov de șoferi.

Dacă un șofer (nou) intră în aplicație, după conectare i se va transmite harta parcării. Așa cum am menționat și mai sus, șoferii care sunt în aplicație de ceva timp vor primi, pe lângă harta de început, actualizări ale hărții pe parcursul timpului. Dacă nu are loc nicio modificare în parcare, ei nu vor mai primi nimic pe moment.

În cazul Server-Senzori, comunicarea este bidirecțională. Senzorii transmit serverului când un loc se ocupă sau se eliberează pentru a ajuta serverul să actualizeze harta și să o transmită clienților din aplicație. Când un șofer intră în aplicație, serverul trimite semnal senzorilor liberi pentru a-i pregăti în cazul în care șoferul vrea să parcheze. După parcare, senzorul atribuit locului ales de șofer (în cazul nostru, unul liber, ales în ordinea crescătoare a indicilor locurilor) se va ocupa. În cazul în care totuși șoferul intră în aplicație dar nu mai parchează, senzorul nu trimite niciun semnal întrucât nimic nu s-a schimbat.

Trebuie să avem în vedere faptul că unii șoferi intră în parcare fără a intra în aplicație și își caută fără ajutorul ei locul de parcare. Acest caz este acoperit de senzori, care vor marca ocuparea la un anumit interval de timp (random).

4 Aspecte de Implementare

În aplicație, pentru a comunica în ambele direcții, șoferii și senzorii așteaptă un mesaj, însă, dacă acesta ar fi transmis prin `read` (blocant), ar întârzia și desincroniza întregul proces. De exemplu, dacă clientul șofer era implementat cu `read` blocant, acesta ar fi oprit serverul din a executa comunicarea cu senzorii în momente esențiale. Astfel, nu se mai trimiteau notificări de actualizare iar șoferul nu mai primea harta actualizată. De aceea, șoferul primește date de la server în mod neblocant. În aceeași manieră primesc mesaje atât și serverul, cât și senzorii în comunicarea lor, pentru a continua procesul și a executa alte comenzi în loc să rămână blocați la una singură, neprioritară.

`Read`-ul neblocant a fost implementat cu ajutorul funcției `select`, așa cum este arătat mai jos.

```
int forRead;

fd_set in;
FD_ZERO(&in);
FD_SET(sd, &in);

struct timeval timeout; // interval de timp de asteptare a mesajului de la server
timeout.tv_sec = 1;     // 1 secunda
timeout.tv_usec = 0;    // + 0 milisecunde

forRead = select(sd + 1, &in, NULL, NULL, &timeout);

// printf("forRead este %d \n", forRead);

if (forRead < 0)
{
    perror("[client]Eroare la select().\n");
    return errno;
}
else if (forRead > 0)
{
    // citim
    if (FD_ISSET(sd, &in))
    {
        int desc_read = read(sd, &alarma_sofer_nou, sizeof(int));

        if (desc_read < 0)
        {
            perror("[client]Eroare la read() de la server.\n");
            return errno;
        } else if(desc_read == 0) {
            printf("[senzor] Serverul a picat, inchidem senzor \n");
            break;
        } else {
        printf("[senzor] Am citit de la server %d\n", alarma_sofer_nou);
        }
    }
}
```

Read neblocant

Un alt aspect special în implementarea proiectului este distribuirea locurilor.

Cum sunt ele atribuite? Când un șofer intră în aplicație, serverul transmite un mesaj, spunând că un client s-a conectat, deci caută un loc de parcare.

Semnalul este trimis către un senzor liber, care simulează marcarea locului.

Cum este ales senzorul? Se iterează prin indicii senzorilor și se alege primul liber (presupunând că intrarea parcării este mai aproape de locurile cu indici mici, deci se încearcă o alegere eficientă a zonei).

```
// DOAR PENTRU SENZORUL LIBER CU UN ANUMIT ID SE FACE WRITE
int k = 0;

for (k = 0; k < NR_SENZORI; k++)
{
  if (senzori[k] == 0)
  {
    ID_SENZOR = k;
    PARK_PLIN = 0;
    break;
  }
}

// printf("S-a ales senzorul %d \n", ID_SENZOR);

if (k == NR_SENZORI)
{ // INSEAMNA CA PARCAREA E FULL DECI NU SE MAI TRIMITE ALARMA LA SENZOR PANA NU SE DEBLOCHEAZA MACAR UNUL
  PARK_PLIN = 1;
}

if (PARK_PLIN == 0) // DACA PARCAREA NU E PLINA, TRIMITEM MESAJ LA SENZOR CA SOFERUL VREA SA PARCHEZE
{
  if (SOFER_NOU == 0)
  {
    for (k = NR_SENZORI; k <= 100; k++)
    {
      if (soferNou[k] == 1)
      {
        SOFER_NOU = 1;
        id_thread_sofer = k;
        soferNou[k] = 0;
        break;
      }
    }
  }
```

```
    if (SOFER_NOU && (ID_SENZOR == tdL.idThread))
    {
      sleep(17); // PAUZA IN CARE SOFERUL SE UITA PE HARTA SI ALEGE UNDE SA PARCHEZE
      // printf("[thread] Avem sofer nou %d care vrea sa se puna pe locul %d \n", k, ID_SENZOR);
      if (write(tdL.cl, &vreau_parcare, sizeof(int)) <= 0)
      {
        printf("[Thread %d] ", tdL.idThread);
        perror("[Thread]Eroare la write() catre client.\n");
      }
      else
      {
        printf("[Thread %d]Alarma de sofer nou a fost transmisa cu succes catre senzor.\n", tdL.idThread);
        SOFER_NOU = 0;
      }
    }
  }
}
```

Dar cum se simulează marcarea locului când șoferul parchează? Se atribuie o șansă de 90% acțiunii de efectuare a parcării de către șofer și o șansă de 10% acțiunii de răzgândire a șoferului (cazul în care nu mai vrea să parcheze).

Pe lângă această modalitate de ocupare a parcării, senzorii mai au șansa de a-și schimba statusul (din liber în ocupat sau invers, din ocupat în liber) o dată la 30 de secunde.

Cum? La fiecare 30 de secunde de la pornirea senzorului se verifică dacă statusul se schimbă, dându-i acestuia o șansă de 50% de a se schimba.

De ce facem asta? Pentru a trata și cazul în care șoferii intră în parcare și ocupă locuri fără ca ei să se orienteze cu ajutorul aplicației. De asemenea, eliberarea locului nu se face pe baza aplicației, deci verificăm și cazul când șoferii pleacă din parcare (la 30 de secunde, există șansa de 50% să plece).

```
if (alarma_sofer_nou != 1)
{
    timp_trecut = difftime(prezent, startExec);
    if (timp_trecut >= 30.0)
    {
        if (status != RandomInt())
        {
            status = 1 - status;
            changed_status = 1;
        }
        //resetam timer
        time(&startExec);
    }
}
else
{
    printf("Intra sofer nou ? \n");
    // if probabilitate sa parcheze e mai mica de 0.9, change status
    if (RandomFloat() <= 0.9)
    {
        printf("da \n");
        status = 1;
        changed_status = 1;
    }
    alarma_sofer_nou = 0;
}
```

5 Concluzie

În concluzie, aplicația dezvoltată are rolul de a eficientiza o situație întâlnită în viața de zi cu zi de către orice șofer.

Totuși, mereu există loc pentru îmbunătățiri. Astfel, pentru o mai ușoară interacțiune cu clientul, în loc de un output cu o parcare simulată în linia de

comandă, se poate crea o interfață mai accesibilă pentru șofer, unde este afișată infrastructura parcării cu locuri colorate în roșu (pentru a marca faptul că sunt ocupate) și verde (pentru a marca faptul că sunt libere). Locurile sunt numerotate.

De asemenea, prin interfață am putea afișa pentru șofer câte locuri disponibile mai sunt în parcare, pentru a-l ajuta să își facă o idee despre șansele pe care le are în a găsi un loc liber.

Desigur, se mai pot adăuga și alte opțiuni, de exemplu, reținerea locului de parcare. Când șoferul revine în parcare, se poate uita pe aplicație în cazul în care a uitat locul sau zona în care a parcat. În interfață ar scrie că mașina sa (reținută de aplicație în funcție de numărul matricol) se află pe un anumit loc, fapt ce l-ar ajuta semnificativ să își găsească locul.

Se mai pot adăuga rezervări de locuri (marcate cu portocaliu în interfață) pe un anumit timp sau camere de supraveghere pentru reținerea numerelor de înmatriculare la intrarea în parcare. Se pot reține mai multe detalii despre fiecare șofer în parte într-o bază de date, inclusiv intervalele orare în care au fost în parcare (pentru monitorizare) și multe altele.

References

1. Network Programming (III), `https://profs.info.uaic.ro/~computernetworks/files/7rc_ProgramareaInReteaIII_En.pdf`, last accessed 2023/11/14

2. TCP vs UDP: The Ultimate Comparison, `https://orhanergun.net/tcp-vs-udp-ultimate-comparison`, last accessed 2023/04/13

3. Parallel TCP/IP Socket Server With Multithreading and Multiprocessing in C, `https://dzone.com/articles/parallel-tcpip-socket-server-with-multi-threading`, last accessed 2020/06/16

4. Diagram Maker, `https://app.diagrams.net/`, last accessed 2023/11/30

5. Differences between TCP and UDP, `https://www.geeksforgeeks.org/differences-between-tcp-and-udp/`, last accessed 2023/05/06

6. Detecting TCP Client Disconnect, `https://itecnote.com/tecnote/c-detecting-tcp-client-disconnect/`, last accessed 2023/12/09

7. Server detect closed client, `https://www.linuxquestions.org/questions/programming-9/how-could-server-detect-closed-client-socket-using-tcp-and-c-824615/`, last accessed 2010/06/08

8. LNCS Homepage, `http://www.springer.com/lncs`, last accessed 2023/11/30