Proiect - SmartParkingAssistant

Mihail Popovici

Facultate de Informatică, Universitatea ,,Alexandru Ioan Cuza" din Iași

1 Introducere

,,SmartParkingAssistant" reprezintă o soluție în contextul problemei tot mai acute a gestionării eficiente a parcărilor, fiind conceput pentru a simplifica și optimiza procesul de identificare a locurilor de parcare în timp real. Astfel, se vor crea două aplicații (client/server); serverul propune simularea unei configurații inițiale a parcării, bazată pe senzori și camere inteligente, date ce vor fi furnizate ulterior aplicației client.

Printre obiectivele proiectului se numără simularea dezvoltarea unei interfețe simple și ușor de utilizat de către client, dar și crearea unui mediu dinamic, în care valabilitatea locurilor de parcare se actualizează constant și automat pe baza intrărilor și ieșirilor din parcare

2 Tehnologii Aplicate

În dezvoltarea acestui proiect s-a utilizat un server TCP concurent, ce creează un nou thread pentru fiecare client conectat. Motivația din spatele alegerii protocolului TCP se bazează pe integritatea datelor pe parcusul transmiterii acestora între aplicații. Astfel protocolul TCP se asigură ca toate pachetele ajung la destinație și niciun pachet nu este pierdut. În contextul de față, ne dorim ca statusul parcării și locul de parcare dorit de către client să fie transmise în mod corect, pentru a evita eventualele confuzii ce pot apărea dacă se corup datele pe drum.

3 Structura Aplicației

Implementarea aplicației se împarte în două părți complementare:

1) **Clientul**: poate fi la rândul său de două tipuri:

Client de tipul 1: acesta vine în parcare și transmite serverului dorința de a parca, furnizând totădată și numărul de înmatriculare al mașinii, ce va fi verificat pentru a se încadra în standarde. Ulterior, clientului îi este prezentat răspunsul de la server (imposibilitatea de a parca dacă parcarea este plină/număr de înmatriculare invalid/locul de parcare alocat, împreună cu cheia de identificare folosită pentru a-și scoate mașina din parcare).

Client de tipul 2: acesta transmite serverului intenția de a ieși din parcare și cheia unică de identificare primită inițial de la server, ce trebuie să fie în

conformitate cu structura generală a cheilor înainte de a fi trimisă către server. Dacă în urma verificărilor făcute de server cheia corespunde unei mașini din parcare, clientului îi este furnizat accesul la mașină, cât și locul de parcare pe care se găsește aceasta.

2) **Serverul**: primește de la client tipul acestuia și numărul de înmatriculare/cheia de identificare.

În cazul clientului de tip 1, serverul se asigură de faptul că numărul furnizat nu aparține deja unei mașini aflate în parcare (deși este conceptual imposibil ca 2 mașini să aibă același număr de înmatriculare, se iau măsuri de siguranță împotriva introducerii greșite a acestuia, cât și a dorințelor rău intenționate ale unor utilizatori de a strica logica aplicației). Ulterior, după ce trece acest test, se verifică statusul curent al parcării și i se alocă un loc, dacă este posibil, urmând a se trimite înapoi către client rezultatul acestor operații.

În ceea ce privește clienții de tip 2, se verifică cheia furnizată pentru a fi a unei mașini din parcare, în caz afirmativ urmând a se oferi accesul clientului la mașină. Răspunsul este trimis către client, alături de locul de parcare unde îi este situată mașina.

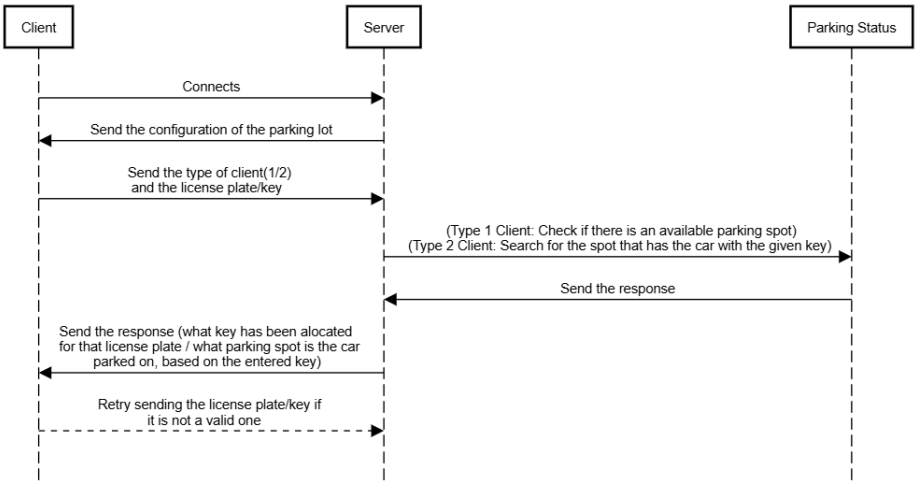

Fig. 1. Diagrama aplicației

4 Aspecte de implementare

Inițial, serverul va trimite clientului configurația curentă a parcării, a cărei stocare se face cu ajutorul unui vector, unde valoarea 0 semnifică faptul că locul respectiv liber, iar valoarea 1 că acel loc este ocupat. Extragerea ei se face folosind funcția din **Fig. 2**

```c
void get_parking_status(char* buffer){
    strcat(buffer, "Current parking status:\n");
    for(int i=0;i<NR_OF_PARKING_SPOTS;i++){
        char text[50];
        if(parking_spots[i]==0){
            snprintf(text, sizeof(text), GREEN "Spot %d: available\n" RESET, i+1);
        }else{
            snprintf(text, sizeof(text), RED "Spot %d: NOT available\n" RESET, i+1);
        }
        strcat(buffer, text);
    }
}
```

Fig. 2. Configurația parcării

Așa cum a fost explicat și mai sus, inputul clientului trece printr-o serie de verificări înainte de a fi trimis către server, dar asta nu garantează că serverul nu poate considera anumite inputuri invalide. Astfel, se va analiza și răspunsul dat de server (**Fig 3.**).

```c
//we are splitting the communication with the server based on the type of client
if(choice=='1'){
    int running=1;
    while(running){
        printf("Enter your license plate (Format : LLNNLLL) (L-letter, N-number): ");
        fflush(stdout);
        memset(buffer, 0, sizeof(buffer));
        scanf("%s", buffer);
        while(!(buffer[0]>='A' && buffer[0]<='Z') || !(buffer[1]>='A' && buffer[1]<='Z')
          || !(buffer[4]>='A' && buffer[4]<='Z') || !(buffer[5]>='A' && buffer[5]<='Z')
          || !(buffer[6]>='A' && buffer[6]<='Z') || !(buffer[2]>='0' && buffer[2]<='9')
          || !(buffer[0]>='0' && buffer[3]<='9') || strlen(buffer)!=7){

            printf("Invalid input! Try again : ");
            fflush(stdout);
            scanf("%s", buffer);
        }
        //Sending to the server the type of client and its license plate
        memset(msg_to_server, 0, sizeof(msg_to_server));
        msg_to_server[0]=choice;
        strncpy(msg_to_server+2, buffer, 7);
        msg_to_server[9]='\0';
        if (write (sd, msg_to_server, sizeof(msg_to_server)) <= 0)
        {
          perror ("[client]Eroare la write() spre server.\n");
          return errno;
        }
        //getting and printing the response from the server
        memset(buffer, 0, BUFFER_SIZE);
        if(read(sd, buffer, BUFFER_SIZE)<=0){
            perror("[Client] Error at read() from server\n");
            return errno;
        }
        printf("%s", buffer);
        fflush(stdout);
        if(strstr(buffer, "invalid")==0) running=0;
    }
```

Fig. 3. Client

Generarea cheilor unice de identificare se face cu ajutorul unei funcții cunoscute de hashing, și anume djb2. Astfel, ne asigurăm că fiecarui număr de înmatriculare

îi este atribuit un cod unic, asemenea unui pin, în absența căruia clientul nu poate avea acces la mașină, asigurând o măsură suplimentară de securitate. (**Fig 4.**).

```c
// Function to generate a numeric hash key using djb2
unsigned long generate_numeric_key(const char *license_plate) {
    unsigned long hash = 5381; // Initialize hash value
    int c;

    // Process each character in the license plate
    while ((c = *license_plate++)) {
        hash = ((hash << 5) + hash) + c; // hash * 33 + c
    }

    return hash % 1000000; // Return a 6-digit key
}
```

Fig. 4. Hash function

Modificările aduse asupra locurilor de parcare pot provoca probleme din punct de vedere al concurenței, astfel că protejăm accesul simultan la date cu ajutorul unui mutex(**Fig 5.**)

```c
char* license_plate = msg_from_server+2;
int selected_spot=0;
pthread_mutex_lock(&lock);
selected_spot=get_available_spot();
memset(buffer, 0, BUFFER_SIZE);
if(selected_spot==-1){
    running=0;
    snprintf(buffer, BUFFER_SIZE, RED "No available spots in the parking. Come again later!\n" RESET);
}else{
    unsigned long key = generate_numeric_key(license_plate);
    if(check_for_thesame_key(key)==0){
        running=0;
        spots_keys[selected_spot-1]=key;
        parking_spots[selected_spot-1]=1;//We are making that parking spot NOT available
        snprintf(buffer, BUFFER_SIZE, GREEN "You've been given the spot nr %d\n" RESET YELLOW "Your ke
    }else{
        snprintf(buffer, BUFFER_SIZE, RED "The license plate you've given is invalid!\nThere is alread
    }
}
pthread_mutex_unlock(&lock);
```

Fig. 5. Mutex

5 Concluzii

Aplicația constituie în momentul acesta o soluție stabilă pentru problema adusă de proiect, însă poate beneficia de modificări și funcționalități noi. Una dintre acestea ar fi implementarea unei baze de date unde se va stoca configurația parcării, întrucât momentan valabilitatea locurilor de parcare este stocată local, pe server, aceasta fiind resetată de fiecare dată când este oprit serverul.

6 Referințe bibliografice

1. *Cursul „Rețele de calculatoare". A concurrent server that creates a thread for each connected client.* `https://edu.info.uaic.ro/computer-networks/files/NetEx/S12/ServerConcThread/servTcpConcTh2.c`
2. *Laboratorul materiei „Rețele de calculatoare". Modelul Client/Server iterativ TCP.* `https://profs.info.uaic.ro/andrei.scutelnicu/teaching/ComputerNetworks/Lab6/Lab6.pdf`
3. *Geeks for Geeks. Thread functions in C/C++* `https://www.geeksforgeeks.org/thread-functions-in-c-c/`
4. *Wikipedia. ANSI escape code* `https://en.wikipedia.org/wiki/ANSI_escape_code`
5. *Geeks for Geeks. snprintf() in C* `https://www.geeksforgeeks.org/snprintf-c-library/`