Tema de casa PATR
TEMA 9 - Sistem de monitorizare a unei statii de incarcare a masinilor electrice

Echipa: 1
Data: 12.12.2023

Componenta echipei si contributia individuală (în %):

Membrii	A	C	D	PR	CB	e-mail
Galiceanu Valentin	25%	25%	25%	25%	25%	valentin.galiceanu@stud.acs.upb.ro
Patru Claudiu	25%	25%	25%	25%	25%	claudiu.patru0412@stud.acs.upb.ro
Petrica Claudiu	25%	25%	25%	25%	25%	claudiu.petrica@stud.acs.upb.ro
Tanase Denis	25%	25%	25%	25%	25%	denis.tanase@stud.acs.upb.ro
	100%	100%	100%	100%	100%	

A-analiză problemă si concepere solutie, C- implementare cod, D-editare documentatie, PR-”proofreading", CB- contributie individuală totală ([%])

Cuprins

1. Introducere. Definire problemă

Problema propusă pentru gestionarea consumului de energie într-o stație de încărcare a mașinilor electrice

Clienții pot alimenta cu electricitate la o stație de încărcare în manieră self-service. Stația are P pompe și C clienți care încearcă să alimenteze. Pentru această temă, am considerat doar 3 locuri de incarcare: două locuri lente și un loc de încărcare rapidă.

Pe lângă conectarea la rețeaua electrică, stația este direct conectată la o serie de panouri solare, fiind echipată cu un număr de baterii capabile să stocheze o anumită cantitate de energie.

1.1 Condiții de gestionare a curentului din stații:

• Se va utiliza mai întâi curentul din bateriile conectate la panourile solare; ulterior, se va trece la curentul din rețeaua electrică, până la reincărcarea completă a bateriilor.

1.2 Condițiile pentru organizarea accesului la locurile de incarcare sunt următoarele:

• Clienții noi se alătură cozii doar în situația în care nu există locuri de încărcare libere.

• În cazul în care coada de așteptare este plină, alți clienți nu mai pot să se adauge.

• Clientul așteaptă ca locul de încărcare să finalizeze alimentarea și să afișeze suma de plată.

• După validarea plății, locul de încărcare devine disponibil, iar clientul poate pleca.

• La finalul fiecărei zile, se desfășoară procedura de închidere a sistemului, punând capăt celorlalte activități până la reluarea acestora. În situația în care există activități în curs în momentul încheierii zilei, cum ar fi alimentarea sau efectuarea unei plăți, acestea își încheie execuția înainte de a fi oprite.

2. Analiza problemei:

Implementarea vizează simularea procesului de alimentare cu electricitate la o stație echipată cu 3 locuri de încărcare și o coadă de așteptare limitată la maxim 6 clienți. La începutul zilei fiscale, primii 3 clienți care ajung sunt direcționați succesiv către cele 3 locuri de încărcare disponibile. În cazul în care un nou client sosește înainte ca un loc de încărcare să fie eliberat, acesta este adăugat la coada de așteptare. Totuși, dacă un loc de încărcare se eliberează înaintea sosirii noului client, acesta poate ocupa imediat locul fără a fi necesar să aștepte în coadă.

Unitățile de încărcare funcționează în mod continuu, iar disponibilitatea acestora este monitorizată constant pentru a distribui noii clienți care ajung la stație. Dacă o unitate de încărcare este disponibilă, se semnalează acest lucru, iar aceasta va fi alocată unui client. Clientul poate fi fie următorul în coadă, fie unul nou în cazul în care coada este goală.

Clientul are posibilitatea de a vizualiza suma de plată afișată pe ecranul terminalului de plată POS, iar achitarea poate fi efectuată simultan pentru toate cele 3 unități de încărcare. După confirmarea plății, clientul poate pleca, iar unitatea de încărcare se eliberează pentru a permite reîncărcarea bateriilor.

Task-ul de stație semnalează atât deschiderea, cât și închiderea periodică a zilei fiscale, garantând astfel un management eficient al operațiunilor. În plus, stația monitorizează constant nivelul de încărcare al bateriilor pentru a asigura funcționarea optimă a unităților de încărcare. Dacă nivelul de încărcare este scăzut, stația poate comuta automat între alimentarea prin baterii și cea prin rețeaua electrică, menținând astfel un flux de energie continuu și prevenind eventualele întreruperi în serviciile oferite clienților. Acest aspect contribuie la asigurarea fiabilității și disponibilității sistemului în orice moment.

Examinarea unor secvențe corecte de execuție este esențială pentru a valida buna funcționare a aplicației.

Secvența 1:

• Un client sosește la stație.

• Clientul selectează una dintre unitățile de încărcare și începe procesul de alimentare.

• Se efectuează verificarea procentajului bateriilor; în cazul în care acesta este intre 0 – 20%, alimentarea se realizează automat din rețeaua electrică..

In cazul in care in timpul incarcarii procentajul scade la 0% se trece automat la retea.

• Unitatea de încărcare încheie alimentarea și afișează suma de plată pe terminalul destinat clientului, care așteaptă validarea sumei.

• Terminalul POS verifică plata și confirmă finalizarea procesului.

• Clientul părăsește stația, iar unitatea de încărcare se eliberează.

• Se efectuează o nouă verificare a procentajului rămas de curent în baterii.

• Se așteaptă un interval de 10 minute; în absența unui client în acest timp, se inițiază procesul de reîncărcare a bateriilor utilizând panourile solare.

Secvența 2:

• Clienții c1, c2 și c3 sosesc simultan la statie.

• Fiecare client selectează una dintre unitățile de încărcare, respectiv u1, u2 și u3.

• Se verifică procentajul bateriilor; dacă este sub 20%, alimentarea se realizează automat din rețeaua electrică.

• Unitatile (u1, u2, u3) încheie procesul de alimentare, iar clienții așteaptă verificarea plății.

• U1 comunică suma de plată către clientul c1 prin intermediul terminalului POS, iar acesta o validează

• C1 pleacă, iar u1 se eliberează pentru următorul client

• Se verifică procentul rămas de curent din baterii.

• Se așteaptă 10 minute; dacă nu sosește un client în acest interval, se începe reîncărcarea bateriilor de la panourile solare.

• U2 comunică suma de plată către client prin intermediul terminalului POS, iar acesta o validează.

• C2 pleacă, iar U2 se eliberează.

• Se verifică procentul rămas de curent din baterii.

• Se așteaptă 10 minute; dacă nu sosește un client în acest interval, se începe reîncărcarea bateriilor de la panourile solare.

• U3 comunică suma de plată către client prin intermediul terminalului POS, iar acesta o validează.

• C3 pleacă, iar p3 se eliberează.

• Se verifică procentul rămas de curent din baterii.

• Se așteaptă 10 minute; dacă nu sosește un client în acest interval, se începe reîncărcarea bateriilor de la panourile solare.

Secvența 3:

• Sosesc 6 clienți (c1, ..., c6).

• C1, c2, c3 intră la unitățile de încărcare (U1, U2, U3) și încep procesul de alimentare; simultan, c4, c5 și c6 intră în coadă.

• Unitățile de încărcare încheie procesul de alimentare, iar clienții așteaptă verificarea plății.

• U1 comunică suma de plată către clientul c1 prin intermediul terminalului POS, iar acesta o validează.

• C1 pleacă, iar U1 se eliberează.

• C4 intră la U1 și se verifică procentajul bateriilor; în caz de insuficiență pentru alimentarea completă, se trece la încărcarea din rețeaua electrică.

• U2 comunică suma de plată către clientul c2 prin intermediul terminalului POS, iar acesta o validează.

• C2 pleacă, iar U2 se eliberează.

• C5 intră la U2 și se verifică procentajul bateriilor; în caz de insuficiență pentru alimentarea completă, se trece la încărcarea din rețeaua electrică.

• U1 încheie alimentarea, iar c4 așteaptă verificarea plății.

• U3 comunică suma de plată către clientul c3 prin intermediul terminalului POS, iar acesta o validează.

• C3 pleacă, iar U3 se eliberează.

• Se efectuează verificarea procentajului rămas de curent în baterii.

• Se așteaptă 10 minute; în absența unui nou client, se începe reîncărcarea bateriilor de la panourile solare.

• U1 comunică suma de plată către clientul c4 prin intermediul terminalului POS, iar acesta o validează.

• C4 pleacă, iar U1 se eliberează.

• C6 intră la U1 și se verifică procentajul bateriilor; în caz de insuficiență pentru alimentarea completă, se trece la încărcarea din rețeaua electrică.

• U2 încheie alimentarea, iar c5 așteaptă verificarea plății.

• U2 comunică suma de plată către clientul c5 prin intermediul terminalului POS, iar acesta o validează.

• C5 pleacă, iar U2 se eliberează.

• Se efectuează verificarea procentajului rămas de curent în baterii.

• Se așteaptă 10 minute; în lipsa unui nou client, se inițiază procesul de reîncărcare a bateriilor de la panourile solare.

• U1 încheie alimentarea, iar c6 așteaptă verificarea plății.

• U3 comunică suma de plată către clientul c6 prin intermediul terminalului POS, iar acesta o validează.

• C6 pleacă, iar U3 se eliberează.

• Se efectuează verificarea procentajului rămas de curent în baterii.

• Se așteaptă 10 minute; în lipsa unui nou client, se inițiază procesul de reîncărcare a bateriilor de la panourile solare.

.

Evaluarea unor secvențe greșite de execuție:

• Secvența 1 - Sosesc 4 clienți, dar coada de așteptare este limitată la 3. Clienții c1, c2, și c3 intră la unitățile de încărcare, iar c4 rămâne în afara coadei. Însă, coada ar trebui să accepte toți clienții sosind și să îi gestioneze corect.

• Secvența 2 - Clienții c1 și c2 încep alimentarea simultan la aceeași unitate de încărcare U1. Aceasta poate duce la conflicte și erori în funcționare, deoarece o unitate de încărcare ar trebui să fie ocupată de un singur client la un moment dat.

• Secvența 3 - C5 intră la U2 și, în loc să verifice procentajul bateriilor, începe direct alimentarea. Acest lucru poate duce la erori în cazul în care bateria nu este suficient de încărcată pentru a susține întregul proces de alimentare.

• Secventa 4 - În timp ce clienții c1 și c2 așteaptă verificarea plății, începe procesul de reîncărcare a bateriilor de la panourile solare. Acest lucru poate afecta disponibilitatea de energie și poate duce la întreruperi în funcționare.

• Secventa 5 - U1 comunică o sumă de plată greșită către clientul c3 prin intermediul terminalului POS. Acest lucru poate duce la neînțelegeri și la probleme în procesul de plată.

3. Aplicația. Structura și soluția de implementare propusă.

3.1. Definirea structurii aplicatiei

În acest capitol se vor identifica task-urile care compun aplicația și mecanismele de sincronizare ce vor fi fost utilizate în modelarea și implementarea comportamentului dorit al aplicației.

Task-ul principal

Constă în inițializarea semafoarelor și mutex-urilor necesare, precum și crearea thread-urilor asociate unităților de încărcare (analog cu Pompelor), Terminalului POS și Stației. După aceste proceduri inițiale, programul așteaptă introducerea comenzilor de la tastatură. Caracterul 'c' semnifică sosirea unui nou client, iar orice alt caracter determină ieșirea din bucla infinită și încheierea programului.

La sosirea unui client, se efectuează verificări pentru a determina modul de gestionare a acestuia. În primul rând, se examinează dacă există unități de încărcare disponibile. În caz afirmativ, noul client este considerat a intra direct la una dintre unitățile de încărcare. Dacă nu există unități de încărcare libere, se verifică numărul de clienți în coadă. Dacă coada nu este plină, noul client este adăugat la coadă. În situația în care coada este plină, comanda dată de utilizator este ignorată, iar un nou thread Client nu este creat.

Această structură permite gestionarea eficientă a intrărilor de clienți, evitând supraaglomerarea coziilor sau crearea excesivă de thread-uri în cazul în care unitățile de încărcare sunt ocupate. Programul rămâne în așteptarea comenzilor ulterioare de la utilizator, asigurând funcționarea corectă și eficientă a întregului sistem.

Task de tip Client

Atunci când un thread Client începe execuția, acesta generează o valoare reprezentând cantitatea de electricitate dorită și verifică disponibilitatea unităților de încărcare . Dacă una dintre unități de încărcare este liberă, clientul o ocupă. În cazul în care toate unitățile de încărcare sunt ocupate, thread-ul Client așteaptă eliberarea uneia dintre ele.

După ce clientul ocupă o unitate de încărcare, introduce valoarea generată anterior în bufferul corespunzător și așteaptă semnalul de încheiere a operației de alimentare. Odată finalizată alimentarea, clientul așteaptă confirmarea plății de la terminalul POS. După primirea confirmării, clientul poate pleca, iar thread-ul asociat său se încheie.

Task de tip Unitate de incarcare

Fiecare unitate de încărcare are alocat un fir de execuție, creat la pornirea aplicației și activ în mod continuu până la oprirea acesteia. Unitățile de încărcare efectuează verificări periodice pentru a identifica cererile de alimentare din partea clienților, în timpul funcționării stației. La finalul zilei, unitățile de încărcare încetează să mai efectueze verificări pentru potențiali clienți noi.

Atunci când o unitate de încărcare detectează un client, aceasta citește cantitatea de electricitate solicitată. În situația în care această cantitate nu este disponibilă în baterii, alimentarea se realizează din rețeaua de curent. Unitatea de încărcare calculează suma totală ce trebuie plătită și începe operația de alimentare. La finalizarea acestei operații, unitatea de încărcare notifică clientul și transmite suma totală de plată către terminalul POS. După confirmarea plății de către terminal, unitatea de încărcare se eliberează.

În situația în care ziua se încheie în timpul procesului de alimentare, unitatea de încărcare își finalizează operațiunea curentă înainte de a intra în starea de pauză. Această adaptare asigură o descriere coerentă și detaliată a modului de funcționare a unităților de încărcare în contextul sistemului menționat.

Task de tip Stație

Ciclul de activitate al acestui task include semnalarea periodică a încheierii zilei. Astfel, task-urile Unitate de Încărcare își opresc operațiile în curs după finalizarea acestora, iar firul principal nu mai acceptă input-uri de la utilizator. După expirarea unei perioade prestabilite de timp, firul principal indică redeschiderea stației, iar celelalte task-uri își reiau activitatea. Similar cu Unitățile de Încărcare , task-ul Stație rulează continuu, asigurând funcționarea constantă a sistemului

La intervale regulate, acest task monitorizează necesitatea reîncărcării bateriilor. Dacă procentajul este intre 0-20% sau s-au scurs cele 10 minute de așteptare pentru un nou client, în momentul în care apare un client nou, procesul se oprește temporar până la finalizarea operațiunilor de alimentare

3.2. Definirea solutiei in vederea implementarii

3.2.1. Linux-Xenomai / POSIX

Implementarea soluției propuse pentru stația de încărcare a mașinilor electrice se bazează pe sistemul de operare Linux-Xenomai și interfața C/POSIX. Această alegere permite beneficierea de caracteristicile timpului real oferite de Xenomai și de mecanismele de sincronizare și comunicare standardizate prin POSIX.

Pentru a asigura un consum eficient al energiei regenerabile, au fost implementate următoarele mecanisme de sincronizare și comunicare:

Structuri de Date:

Client

- int id: Identificatorul unic al clientului.
- int requested_current: Cantitatea de curent solicitată de către client.

Pump

- int battery_capacity: Capacitatea maximă a bateriei pompei.
- int current_battery_level: Nivelul curent al bateriei pompei.
- sem_t pump_sem: Semafor pentru a controla accesul la pompa pentru încărcare.
- sem_t payment_sem: Semafor pentru a controla accesul la efectuarea plății la pompă.
- sem_t charging_sem: Semafor pentru a restricționa încărcarea de la panouri solare în timpul alimentării unui client.

Semafoare Globale

- sem_t station_sem: Semafor pentru a controla accesul la stație și a limita numărul de clienți simultani.
- sem_t queue_sem: Semafor pentru a controla accesul la coadă și a limita numărul de clienți care pot intra în coadă simultan.
- sem_t queue_mutex: Semafor pentru a asigura accesul exclusiv la variabila current_queue_size.

Variabile Globale

- int current_queue_size: Numărul curent de clienți în coadă.

- int current_client_id: Ultimul ID atribuit unui client.

Funcții

initialize()

- Inițializează structurile de date și semafoarele.
- Creează firele de execuție pentru fiecare pompă.

station_action(void *arg)

- Acțiunea executată de fiecare pompă.
- Afișează starea pompei și permite accesul la aceasta la intervale regulate.

client_action(void *arg)

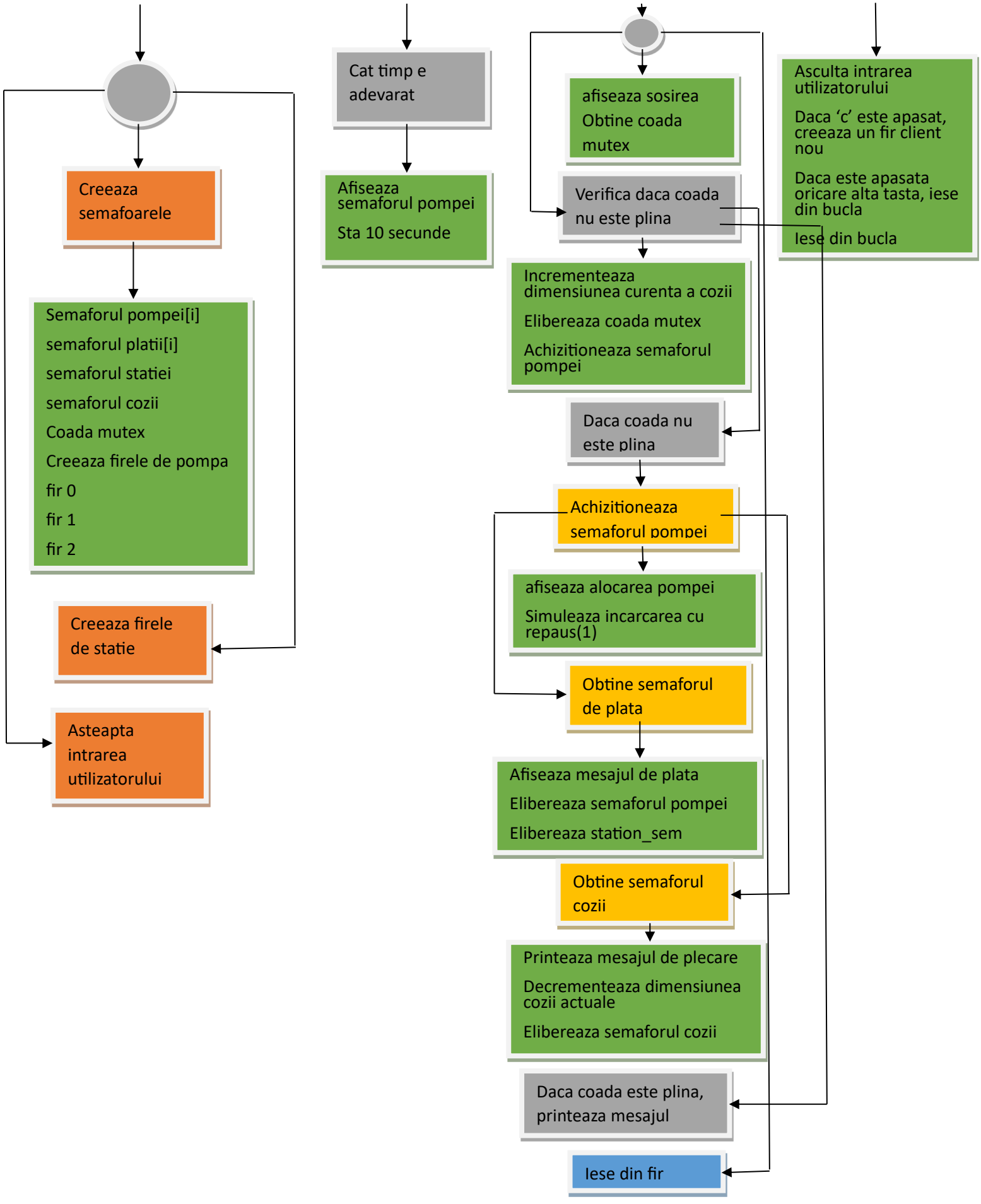
- Acțiunea executată de fiecare client.
- Simulează intrarea clientului în stație, solicitarea de încărcare și părăsirea stației.

wait_for_user_input()

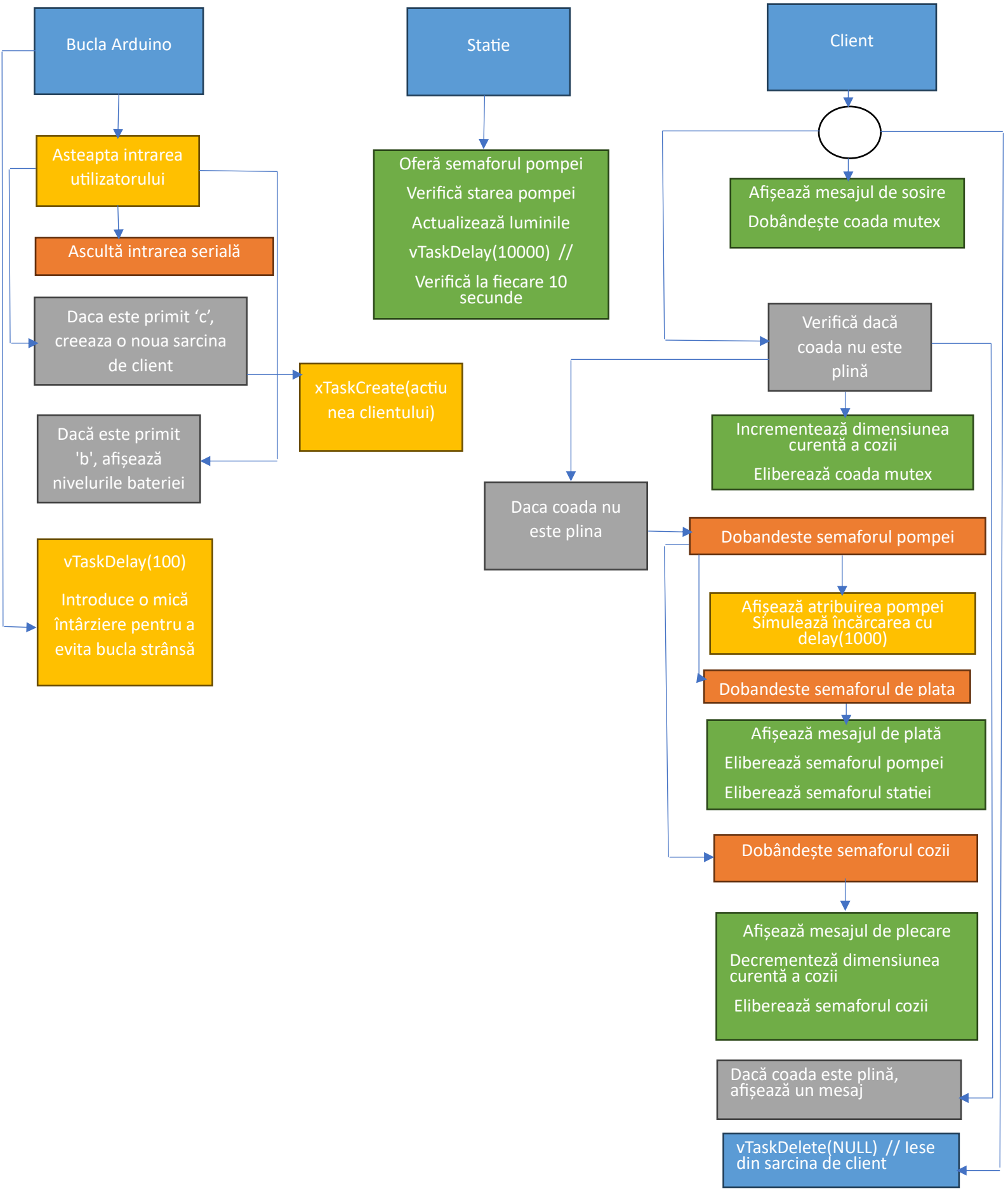
- Așteaptă introducerea utilizatorului și acționează în consecință.
- Dacă se apasă tasta 'c', un nou client este creat și adăugat la coadă.
- Dacă se apasă tasta 'b', afișează nivelurile bateriilor pentru fiecare pompă.
- Orice altă tastă încheie bucla de așteptare a introducerii.

main()

- Inițializează sistemul.
- Creează firul de execuție pentru acțiunile de stație.
- Așteaptă introducerea utilizatorului.


Creeaza semafoarele
Semaforul pompei[i]
semaforul platii[i]
semaforul statiei
semaforul cozii
Coada mutex
Creeaza firele de pompa
fir 0
fir 1
fir 2
Creeaza firele de statie
Asteapta intrarea utilizatorului
Cat timp e adevarat
Afiseaza semaforul pompei
Sta 10 secunde
afiseaza sosirea
Obtine coada mutex
Verifica daca coada nu este plina
Incrementeaza dimensiunea curenta a cozii
Elibereaza coada mutex
Achizitioneaza semaforul pompei
Daca coada nu este plina
Achizitioneaza semaforul pompei
afiseaza alocarea pompei
Simuleaza incarcarea cu repaus(1)
Obtine semaforul de plata
Afiseaza mesajul de plata
Elibereaza semaforul pompei
Elibereaza station_sem
Obtine semaforul cozii
Printeaza mesajul de plecare
Decrementeaza dimensiunea cozii actuale
Elibereaza semaforul cozii
Daca coada este plina, printeaza mesajul
Iese din fir
Asculta intrarea utilizatorului
Daca ‘c’ este apasat, creeaza un fir client nou
Daca este apasata oricare alta tasta, iese din bucla
Iese din bucla

Arduino.


Bucla Arduino
Asteapta intrarea utilizatorului
Ascultă intrarea serială
Daca este primit ‘c’, creeaza o noua sarcina de client
xTaskCreate(actiunea clientului)
Dacă este primit 'b', afișează nivelurile bateriei
vTaskDelay(100)
Introduce o mică întârziere pentru a evita bucla strânsă
Statie
Oferă semaforul pompei
Verifică starea pompei
Actualizează luminile
vTaskDelay(10000) //
Verifică la fiecare 10 secunde
Client
Afișează mesajul de sosire
Dobândește coada mutex
Verifică dacă coada nu este plină
Incrementează dimensiunea curentă a cozii
Eliberează coada mutex
Daca coada nu este plina
Dobandeste semaforul pompei
Afișează atribuirea pompei
Simulează încărcarea cu delay(1000)
Dobandeste semaforul de plata
Afișează mesajul de plată
Eliberează semaforul pompei
Eliberează semaforul statiei
Dobândește semaforul cozii
Afișează mesajul de plecare
Decrementeză dimensiunea curentă a cozii
Eliberează semaforul cozii
Dacă coada este plină, afișează un mesaj
vTaskDelete(NULL) // Iese din sarcina de client

Implementarea solutiei

```
#include <stdio.h>
#include <pthread.h>
#include <semaphore.h>
#include <unistd.h>
#include <stdlib.h>

#define MAX_PUMPS 3
#define MAX_QUEUE_SIZE 6

typedef struct {
  int id;
  int requested_current;
} Client;

typedef struct {
  int battery_capacity;
  int current_battery_level;
  sem_t pump_sem;
  sem_t payment_sem;
  sem_t charging_sem; // Semafor pentru a restricționa încărcarea de la panouri solare în timpul alimentării unui client
} Pump;

Pump pumps[MAX_PUMPS];

sem_t station_sem;
sem_t queue_sem;
sem_t queue_mutex;
int current_queue_size = 0;
int current_client_id = 1;

void *station_action(void *arg);

void *client_action(void *arg) {
  Client *client = (Client *)arg;

  // Generare aleatoare a cantității de curent cerute de client (între 0 și 10)
  client->requested_current = rand() % 11;

  printf("Client %d arrived. Requested current: %dA.\n", client->id, client->requested_current);

  sem_wait(&queue_mutex);

  if (current_queue_size < MAX_QUEUE_SIZE) {
    current_queue_size++;
    sem_post(&queue_mutex);
```

```
    Pump *selected_pump = &pumps[client->id % MAX_PUMPS];

    sem_wait(&selected_pump->pump_sem);

    if (selected_pump->current_battery_level >= 0 && selected_pump->current_battery_level <= 20) {
      printf("Client %d at pump %d charging from the grid. Battery level: %d/%dA.\n", client->id, client->id %
MAX_PUMPS, selected_pump->current_battery_level, selected_pump->battery_capacity);
      sleep(1); // Simulare încărcare din rețea

      // Încarcă bateria de la panourile solare simultan doar dacă pompa este liberă
      sem_wait(&selected_pump->charging_sem);
      if (selected_pump->current_battery_level < 20) {
        printf("Battery charging from solar panels.\n");
        if (selected_pump->current_battery_level > 90) {
      int remaining_capacity = selected_pump->battery_capacity - selected_pump->current_battery_level;
      selected_pump->current_battery_level += selected_pump->battery_capacity - selected_pump-
>current_battery_level;
      printf("Charged only with remaining capacity: %dA. Total battery level: %d/%dA.\n", remaining_capacity,
selected_pump->current_battery_level, selected_pump->battery_capacity);
    } else {
      selected_pump->current_battery_level += 10; // Exemplu de încărcare de la panouri solare
      printf("Client %d paid at pump %d with battery level: %d/%dA.\n", client->id, client->id % MAX_PUMPS,
selected_pump->current_battery_level, selected_pump->battery_capacity);
    }

        if (selected_pump->current_battery_level > selected_pump->battery_capacity) {
          selected_pump->current_battery_level = selected_pump->battery_capacity;
        }
      } else {
        printf("Battery charging from solar panels restricted due to low battery level.\n");
      }
      sem_post(&selected_pump->charging_sem);
    } else {
      printf("Client %d at pump %d with battery level: %d/%dA.\n", client->id, client->id % MAX_PUMPS,
selected_pump->current_battery_level, selected_pump->battery_capacity);
      sleep(1); // Simulează încărcare
    }

    // Scade cantitatea de curent cerută din nivelul curent al bateriei
    selected_pump->current_battery_level -= client->requested_current;

    // Verificare pentru a nu depăși 100 la încărcarea bateriei


    sem_post(&selected_pump->pump_sem);

    sem_post(&station_sem);
```

```
        sem_wait(&queue_sem);
        printf("Client %d leaves.\n", client->id);
        current_queue_size--;
        sem_post(&queue_sem);
    } else {
        sem_post(&queue_mutex);
        printf("Queue is full. Client %d leaves.\n", client->id);
    }
    free(client);  // Eliberăm memoria alocată pentru client
    pthread_exit(NULL);
}

void initialize() {
    for (int i = 0; i < MAX_PUMPS; i++) {
        pumps[i].battery_capacity = 100; // Exemplu de capacitate a bateriei
        pumps[i].current_battery_level = 100; // Exemplu de nivel curent al bateriei
        sem_init(&pumps[i].pump_sem, 0, 1);
        sem_init(&pumps[i].payment_sem, 0, 1);
        sem_init(&pumps[i].charging_sem, 0, 1);
    }

    sem_init(&station_sem, 0, 1);
    sem_init(&queue_sem, 0, 1);
    sem_init(&queue_mutex, 0, 1);

    pthread_t pumps_threads[MAX_PUMPS];
    for (int i = 0; i < MAX_PUMPS; i++) {
        pthread_create(&pumps_threads[i], NULL, station_action, (void *)&pumps[i]);
    }
}

void *station_action(void *arg) {
    Pump *current_pump = (Pump *)arg;

    while (1) {
        sem_post(&current_pump->pump_sem); // Pompa este disponibilă
        sleep(10); // Verifică la fiecare 10 secunde
    }
}

void wait_for_user_input() {
    char input;
    while (1) {
        scanf(" %c", &input);
        if (input == 'c') {
            Client *new_client = malloc(sizeof(Client));
            new_client->id = current_client_id++;
            pthread_t client_thread;
            pthread_create(&client_thread, NULL, client_action, (void *)new_client);
```

```
    } else if (input == 'b') {
      printf("Battery levels:\n");
      for (int i = 0; i < MAX_PUMPS; i++) {
        printf("Pump %d: %d/%dA\n", i, pumps[i].current_battery_level, pumps[i].battery_capacity);
      }
    } else {
      break;
    }
  }
}

int main() {
  initialize();

  pthread_t station_thread;
  pthread_create(&station_thread, NULL, station_action, (void *)&pumps[0]);

  wait_for_user_input();

  return 0;
}
```

3.2.2. Arduino/ FreeRTOS

```
#include <Arduino.h>
#include <FreeRTOS.h>
#include <semphr.h>


#define MAX_PUMPS 3
#define MAX_QUEUE_SIZE 6

struct Client {
  int id;
  int requested_current;
};

struct Pump {
  int battery_capacity;
  int current_battery_level;
  SemaphoreHandle_t pump_sem;
  SemaphoreHandle_t payment_sem;
  SemaphoreHandle_t charging_sem;
};

struct Light {
  int pin;
  bool is_on;
};
```

```
Pump pumps[MAX_PUMPS];
SemaphoreHandle_t station_sem;
SemaphoreHandle_t queue_sem;
SemaphoreHandle_t queue_mutex;
int current_queue_size = 0;
int current_client_id = 1;

Light lights[MAX_PUMPS] = {
  {2, false}, // Pin 2 for Light 0
  {3, false}, // Pin 3 for Light 1
  {4, false}  // Pin 4 for Light 2
};

Client* new_client = nullptr; // Variabila globala

void initialize();
void* station_action(void* pvParameters);
void* client_action(void* pvParameters);
void wait_for_user_input();

void setup() {
  initialize();

  for (int i = 0; i < MAX_PUMPS; i++) {
    pinMode(lights[i].pin, OUTPUT);
  }

  xTaskCreate(station_action, "Station", 10000, NULL, 1, NULL, 0);
  xTaskCreate(client_action, "Client", 10000, NULL, 1, NULL, 1);
}

void loop() {
  wait_for_user_input();
  vTaskDelay(100);
}

void initialize() {
  for (int i = 0; i < MAX_PUMPS; i++) {
    pumps[i].battery_capacity = 100;
    pumps[i].current_battery_level = 100;
    pumps[i].pump_sem = xSemaphoreCreateMutex();
    pumps[i].payment_sem = xSemaphoreCreateMutex();
    pumps[i].charging_sem = xSemaphoreCreateMutex();
  }

  station_sem = xSemaphoreCreateMutex();
  queue_sem = xSemaphoreCreateMutex();
  queue_mutex = xSemaphoreCreateMutex();
```

```
}

void* station_action(void* pvParameters) {
  Pump* current_pump = &pumps[0];
  while (1) {
    xSemaphoreGive(current_pump->pump_sem);

    for (int i = 0; i < MAX_PUMPS; i++) {
      if (xSemaphoreTake(pumps[i].pump_sem, 0) == pdTRUE) {
        lights[i].is_on = true;
        digitalWrite(lights[i].pin, HIGH);
      } else {
        lights[i].is_on = false;
        digitalWrite(lights[i].pin, LOW);
      }
    }

    vTaskDelay(10000); // Verifică la fiecare 10 secunde
  }
}

void* client_action(void* pvParameters) {
  Client* client = (Client*)pvParameters;
  client->requested_current = random(0, 11);

  Serial.print("Client ");
  Serial.print(client->id);
  Serial.print(" arrived. Requested current: ");
  Serial.print(client->requested_current);
  Serial.println("A.");

  xSemaphoreTake(queue_mutex, portMAX_DELAY);

  if (current_queue_size < MAX_QUEUE_SIZE) {
    current_queue_size++;
    xSemaphoreGive(queue_mutex);

    Pump* selected_pump = &pumps[client->id % MAX_PUMPS];

    xSemaphoreTake(selected_pump->pump_sem, portMAX_DELAY);

    if (selected_pump->current_battery_level >= 0 && selected_pump->current_battery_level <= 20) {
      Serial.print("Client ");
      Serial.print(client->id);
      Serial.print(" at pump ");
      Serial.print(client->id % MAX_PUMPS);
      Serial.print(" charging from the grid. Battery level: ");
      Serial.print(selected_pump->current_battery_level);
      Serial.print("/");
```

```
    Serial.print(selected_pump->battery_capacity);
    Serial.println("A.");
    delay(1000);

    xSemaphoreTake(selected_pump->charging_sem, portMAX_DELAY);
    if (selected_pump->current_battery_level > 90) {
      int remaining_capacity = selected_pump->battery_capacity - selected_pump->current_battery_level;
      selected_pump->current_battery_level += remaining_capacity;
      Serial.print("Charged only with remaining capacity: ");
      Serial.print(remaining_capacity);
      Serial.print("A. Total battery level: ");
      Serial.print(selected_pump->current_battery_level);
      Serial.print("/");
      Serial.print(selected_pump->battery_capacity);
      Serial.println("A.");
    } else {
      selected_pump->current_battery_level += 10;
      Serial.print("Client ");
      Serial.print(client->id);
      Serial.print(" paid at pump ");
      Serial.print(client->id % MAX_PUMPS);
      Serial.print(" with battery level: ");
      Serial.print(selected_pump->current_battery_level);
      Serial.print("/");
      Serial.print(selected_pump->battery_capacity);
      Serial.println("A.");
    }

    xSemaphoreGive(selected_pump->charging_sem);
  } else {
    Serial.print("Client ");
    Serial.print(client->id);
    Serial.print(" at pump ");
    Serial.print(client->id % MAX_PUMPS);
    Serial.print(" with battery level: ");
    Serial.print(selected_pump->current_battery_level);
    Serial.print("/");
    Serial.print(selected_pump->battery_capacity);
    Serial.println("A.");
    delay(1000);
  }

  selected_pump->current_battery_level -= client->requested_current;

  if (selected_pump->current_battery_level > 90) {
    int remaining_capacity = selected_pump->battery_capacity - selected_pump->current_battery_level;
    selected_pump->current_battery_level += remaining_capacity;
    Serial.print("Charged only with remaining capacity: ");
    Serial.print(remaining_capacity);
```

```
        Serial.print("A. Total battery level: ");
        Serial.print(selected_pump->current_battery_level);
        Serial.print("/");
        Serial.print(selected_pump->battery_capacity);
        Serial.println("A.");
      } else {
        Serial.print("Client ");
        Serial.print(client->id);
        Serial.print(" paid at pump ");
        Serial.print(client->id % MAX_PUMPS);
        Serial.print(" with battery level: ");
        Serial.print(selected_pump->current_battery_level);
        Serial.print("/");
        Serial.print(selected_pump->battery_capacity);
        Serial.println("A.");
      }

      xSemaphoreGive(selected_pump->pump_sem);

      xSemaphoreGive(station_sem);

      xSemaphoreTake(queue_sem, portMAX_DELAY);
      Serial.print("Client ");
      Serial.print(client->id);
      Serial.println(" leaves.");
      current_queue_size--;
      xSemaphoreGive(queue_sem);
    } else {
      xSemaphoreGive(queue_mutex);
      Serial.print("Queue is full. Client ");
      Serial.print(client->id);
      Serial.println(" leaves.");
    }
    delete client;
    vTaskDelete(NULL);
}

void wait_for_user_input() {
    if (Serial.available() > 0) {
      char input = Serial.read();
      if (input == 'c') {
        new_client = new Client;
        new_client->id = current_client_id++;
        xTaskCreate(client_action, "Client", 10000, (void*)new_client, 1, NULL);
      } else if (input == 'b') {
        Serial.println("Battery levels:");
        for (int i = 0; i < MAX_PUMPS; i++) {
          Serial.print("Pump ");
          Serial.print(i);
```

```
            Serial.print(": ");
            Serial.print(pumps[i].current_battery_level);
            Serial.print("/");
            Serial.print(pumps[i].battery_capacity);
            Serial.println("A");
        }
    }
  }
}
```

4.Testarea aplicatiei si validarea solutiei propuse

Procesul de testare a fost continuu pe durata dezvoltării aplicației. Noile elemente de sincronizare și comunicare, împreună cu toate funcționalitățile adăugate, au fost supuse unor teste individuale inițiale pentru a confirma buna funcționare. Integrarea lor graduală în cadrul sistemului complet a fost, de asemenea, evaluată cu atenție pentru a evita interferențele cu restul aplicației în fiecare etapă.

Pentru validarea soluției, după implementarea tuturor funcționalităților necesare, au fost folosite secvențe de intrare variate, similare celor descrise în secțiunea 2 a documentației. Aceasta a avut rolul de a acoperi o gamă cât mai largă de posibile succesiuni de evenimente. Procesul de validare a constat în introducerea de la tastatură a unui număr variat de variabile în diferite momente de timp din funcționarea aplicației.

În urma observațiilor efectuate în baza acestor teste, aplicația a fost adaptată pentru a obține o comportare cât mai apropiată de cea ideală. La stadiul actual, programul oferă răspunsuri satisfăcătoare pentru orice secvență de intrare introdusă.

4.1 Linux-Xenomai

```
c
Client 1 arrived. Requested current: 6A.
Client 1 at pump 1 with battery level: 100/100A.
c
Client 2 arrived. Requested current: 10A.
Client 2 at pump 2 with battery level: 100/100A.
c
Client 3 arrived. Requested current: 6A.
Client 3 at pump 0 with battery level: 100/100A.
c
Client 4 arrived. Requested current: 2A.
Client 4 at pump 1 with battery level: 100/100A.
Client 1 leaves.
cClient 2 leaves.
Client 3 leaves.

Client 5 arrived. Requested current: 1A.
Client 5 at pump 2 with battery level: 90/100A.
Client 4 leaves.
c
Client 6 arrived. Requested current: 4A.
Client 6 at pump 0 with battery level: 94/100A.
Client 5 leaves.
Client 6 leaves.
c
Client 7 arrived. Requested current: 0A.
Client 7 at pump 1 with battery level: 92/100A.
Client 7 leaves.
b
Battery levels:
Pump 0: 90/100A
Pump 1: 92/100A
Pump 2: 89/100A
```

```
cClient 23 leaves.

Client 28 arrived. Requested current: 10A.
Client 28 at pump 1 with battery level: 72/100A.
c
Client 29 arrived. Requested current: 3A.
Client 29 at pump 2 with battery level: 58/100A.
Client 24 leaves.

cClient 25 leaves.

Client 30 arrived. Requested current: 10A.
Client 30 at pump 0 with battery level: 53/100A.
c
Client 31 arrived. Requested current: 10A.
Client 31 at pump 1 with battery level: 68/100A.
Client 26 leaves.
cClient 27 leaves.

Client 32 arrived. Requested current: 7A.
Client 32 at pump 2 with battery level: 52/100A.
c
Client 33 arrived. Requested current: 10A.
Client 33 at pump 0 with battery level: 53/100A.
Client 28 leaves.

cClient 29 leaves.

Client 34 arrived. Requested current: 3A.
Client 34 at pump 1 with battery level: 58/100A.

cClient 30 leaves.

Client 35 arrived. Requested current: 7A.
Client 35 at pump 2 with battery level: 49/100A.
c
Client 36 arrived. Requested current: 9A.
Client 36 at pump 0 with battery level: 43/100A.
Client 31 leaves.
c
Client 37 arrived. Requested current: 7A.
Client 37 at pump 1 with battery level: 48/100A.
```

```
Client 56 arrived. Requested current: 6A.
Client 56 at pump 2 charging from the grid. Battery level: 17/100A.
Client 57 arrived. Requested current: 6A.
Client 57 at pump 0 with battery level: 23/100A.
Client 58 arrived. Requested current: 5A.
Client 58 at pump 1 charging from the grid. Battery level: 14/100A.
Client 59 arrived. Requested current: 5A.
Client 59 at pump 2 charging from the grid. Battery level: 17/100A.
Client 60 arrived. Requested current: 7A.
Client 60 at pump 0 with battery level: 23/100A.
Client 61 arrived. Requested current: 4A.
Client 61 at pump 1 charging from the grid. Battery level: 14/100A.
```

```
Battery charging from solar panels restricted due to low battery level.
Battery charging from solar panels.
Client 61 paid at pump 1 with battery level: 29/100A.
Client 61 leaves.
Client 59 leaves.
b
Battery levels:
Pump 0: 10/100A
Pump 1: 25/100A
Pump 2: 16/100A
```

A doua rulare

```
Client 2 arrived. Requested current: 10A.
Client 2 at pump 2 with battery level: 100/100A.
Client 1 arrived. Requested current: 6A.
Client 1 at pump 1 with battery level: 100/100A.
Client 5 arrived. Requested current: 1A.
Client 5 at pump 2 with battery level: 100/100A.
Client 4 arrived. Requested current: 2A.
Client 4 at pump 1 with battery level: 100/100A.
Client 3 arrived. Requested current: 6A.
Client 3 at pump 0 with battery level: 100/100A.
Client 2 leaves.
Client 1 leaves.
Client 5 leaves.
Client 3 leaves.
Client 4 leaves.
b
Battery levels:
Pump 0: 94/100A
Pump 1: 92/100A
Pump 2: 89/100A
ccc
Client 6 arrived. Requested current: 4A.
Client 6 at pump 0 with battery level: 94/100A.
Client 7 arrived. Requested current: 0A.
Client 7 at pump 1 with battery level: 92/100A.
Client 8 arrived. Requested current: 6A.
Client 8 at pump 2 with battery level: 89/100A.
Client 6 leaves.
Client 7 leaves.
Client 8 leaves.
b
Battery levels:
Pump 0: 90/100A
Pump 1: 92/100A
Pump 2: 83/100A
```

```
Client 12 arrived. Requested current: 7A.
Client 12 at pump 0 with battery level: 90/100A.
Client 14 arrived. Requested current: 5A.
Client 14 at pump 2 with battery level: 83/100A.
Client 13 arrived. Requested current: 3A.
Client 13 at pump 1 with battery level: 92/100A.
Client 15 arrived. Requested current: 7A.
Queue is full. Client 15 leaves.
Client 16 arrived. Requested current: 4A.
Queue is full. Client 16 leaves.
Client 17 arrived. Requested current: 9A.
Queue is full. Client 17 leaves.
Client 18 arrived. Requested current: 10A.
Queue is full. Client 18 leaves.
Client 9 leaves.
Client 10 leaves.
Client 12 leaves.
Client 13 leaves.
Client 14 leaves.
Client 11 leaves.
cc
Client 19 arrived. Requested current: 2A.
Client 19 at pump 1 with battery level: 88/100A.
Client 20 arrived. Requested current: 0A.
Client 20 at pump 2 with battery level: 70/100A.
Client 19 leaves.
Client 20 leaves.
c
Client 21 arrived. Requested current: 10A.
Client 21 at pump 0 with battery level: 80/100A.
Client 21 leaves.
c
Client 22 arrived. Requested current: 8A.
Client 22 at pump 1 with battery level: 86/100A.
Client 22 leaves.
```

4.2. Arduino / FreeRTOS