Sem. I 2025-2026

Proiect PATR

Sistem monitorizare și gestionare a traficului într-un tunel

Echipa: Team Awsome

Data: 7 ianuarie 2026

Componența echipei și contribuția individuală (în %)

Membrii	A	C	D	PR	CB	e-mail
Badea Cătălin-Gabriel	54%	12%	47.3%	20%	33.3%	catalin.badea1703@stud.acs.upb.ro
Buterez Daniela-Georgiana	20%	88%	5.3%	20%	33.3%	daniela.buterez@stud.acs.upb.ro
Ionașcu Vlad-Mihai	26%	0%	47.3%	60%	33.3%	vlad_mihai.ionascu@stud.acs.upb.ro
	100%	100%	100%	100%	100%	

A-analiză problemă și concepere soluție, C- implementare cod, D-editare documentație, PR-"proofreading", CB - contribuție individuală totală ([%])

Membrii echipei declară că lucrarea respectă toate regulile privind onestitatea academică. În caz de nerespectare a acestora tema va fi notată cu ***0(zero) puncte***

Cuprins

1 Introducere. Definire problemă

Problema propusă pentru implementare se referă la modelarea traficului printr-un tunel rutier. Mașinile pot circula în ambele sensuri, fiecare sens permițând prezența a maximum N = 10 vehicule simultan. Tunelul este prevăzut cu limite de siguranță privind concentrațiile de fum (F) și gaze naturale (G), care nu trebuie depășite pentru a asigura condiții optime de circulație.

Condițiile pentru accesul mașinilor în tunel sunt următoarele:

- atunci când numărul de mașini din tunel (pe un sens) este mai mare decât N, circulația este blocată pe acel sens;
- dacă se detectează un procent de fum mai mare decât F, accesul în tunel este oprit;
- dacă se detectează un procent al concentrației de gaze naturale mai mare decât G, accesul în tunel este oprit;
- există un operator care poate bloca sau debloca una sau mai multe intrări sau ieșiri;
- dacă este apăsat un buton de panică în interiorul tunelului, sunt blocate intrările în tunel;
- sistemul este inițial pe mod automat, iar când operatorul intervine cu o comandă, sistemul trece pe mod manual până când acesta se deconectează.

Motivație

Motivația noastră de a realiza Tema "Un sistem de monitorizare a traficului dintr-un tunel" a pornit de la dorința de a eficientiza traficul din locuri mai greu de gestionat, acceptand-o ca pe o provocare. Datorita facultații pe care o urmăm (ACS), am învățat să lucrăm cu microprocesoare capabile sa simuleze situații reale din viața de zi cu zi.

Pe lângă asta, ne-au inspirat colegi mai mari, dar și ingineri pricepuți care activează în acest domeniu și care au reușit să optimizeze traficul din aceste locuri.

Astfel, prin realizarea acestei teme, ne dorim sa învățăm și să exersăm conceptele programării aplicațiilor în timp real, alături de cunoștințele necesare pentru a putea contribui în viitor la dezvoltarea societății in care trăim.

2 Analiza problemei

Obiectivul principal al implementării este simularea gestionării dinamice a accesului vehiculelor, bazată pe o serie de senzori și condiții de urgență. Sistemul trebuie să calculeze și să mențină în permanență numărul curent de mașini aflate în interiorul tunelului, utilizând date de la două perechi de senzori intrare-ieșire. Acești senzori rulează continuu, iar diferența dintre totalul mașinilor intrate și cel al mașinilor ieșite determină starea de ocupare a tunelului.

Pe lângă monitorizarea traficului, sistemul este responsabil cu detectarea rapidă a incidentelor interne. Tunelul este echipat cu senzori de gaze naturale și senzori de fum, care, la activare, semnalează imediat o stare de pericol, cum ar fi o scurgere de gaze sau un potențial incendiu. Funcționalitatea critică a aplicației constă în mecanismul de blocare automată a accesului la intrarea în tunel. Intrarea vehiculelor trebuie oprită instantaneu dacă este îndeplinită oricare dintre următoarele condiții: fie numărul de mașini atinge o capacitate maximă prestabilită, fie se detectează un incident de mediu (gaze sau fum), fie este activat un buton de panică/semnal de alarmă.

În plus față de logica automată, aplicația trebuie să simuleze și posibilitatea unei intervenții externe. Un operator uman (cum ar fi serviciul de urgență) trebuie să aibă posibilitatea de a bloca manual intrarea și/sau ieșirea din tunel, anulând temporar controlul automat al traficului. Astfel, sistemul simulat trebuie să combine monitorizarea continuă a senzorilor cu o logică de decizie care prioritizează siguranța și respectă regulile de capacitate, permițând în același timp controlul de urgență.

2.1 Analiza secvențelor de execuție

În continuare vom prezenta analiza unor secvențe de execuție, pentru a ilustra rularea corespunzătoare a logicii de control a tunelului (pe oricare dintre sensuri):

- Secvența 1 - Flux normal de trafic fără pericole:
  - O mașină se apropie de intrarea pe un sens și intră în tunel (se apasă butonul aferent intrării pe sensul respectiv; task-ul de gestionare a liniei incrementează contorul de mașini; serverul verifică condițiile și menține bariera deschisă / LED-ul stins).
  - O altă mașină intră pe același sens (se apasă din nou butonul de intrare; contorul de mașini crește cu o unitate).
  - Prima mașină părăsește tunelul (se apasă butonul corespunzător ieșirii de pe sens; task-ul decrementează contorul de mașini).
  - Periodic se citesc senzorii, iar task-ul de afișare raportează starea curentă (pe moni-

torul serial apar valorile actualizate ale traficului și nivelul zero pentru gaz și fum).

- Secvența 2 - Gestionarea capacității maxime și a aglomerării pentru o anumită linie:
  - Tunelul este aproape plin, mai este un singur loc liber (variabila internă care ține numărul mașinilor este cu o unitate mai mică decât limita maximă admisă N).
  - O nouă mașină intră în tunel (se apasă butonul de intrare; contorul atinge limita maximă; task-ul activează indicatorul care semnalează că tunelul este plin).
  - Sistemul decide blocarea accesului pentru a preveni supraaglomerarea (serverul citește starea de plin, transmite comanda către task-ul de închidere barieră; led-ul roșu de la intrare se aprinde).
  - O mașină încearcă să intre în timp ce bariera este coborâtă. (Apăsarea butonului de intrare este detectată, dar logica programului refuză incrementarea contorului deoarece s-a atins deja valoarea maximă N. Astfel, limita de capacitate este protejată software).
  - O mașină părăsește tunelul (se apasă butonul de ieșire; contorul scade sub N; condiția de tunel plin dispare).
  - Sistemul redeschide accesul (serverul observă că s-a eliberat un loc și nu există alte pericole, transmite comanda către task-ul de deschidere barieră; led-ul roșu se stinge).
  - Periodic se citesc senzorii, iar task-ul de afișare raportează starea curentă.
- Secvența 3 - Detecție Pericol Mediu cauzat de gaze:
  - Nivelul de gaze din tunel crește peste limita admisă (se rotește potențiometrul asociat senzorului de gaz peste pragul stabilit).
  - Senzorul citește valoarea și semnalează pericolul (task-ul de citire convertește valoarea, iar cel de verificare activează flag-ul global de pericol gaz).
  - Serverul prioritizează siguranța și blochează intrările (serverul detectează pericolul și aprinde led-urile de la intrări; ieșirile rămân deschise pentru evacuare).
  - Nivelul de gaz scade sub limită (se rotește potențiometrul înapoi sub pragul de declanșare).
  - Sistemul revine la normal (task-ul de verificare dezactivează indicatorul de pericol; serverul transmite comanda către task-ul de deschidere barieră; led-ul roșu se stinge).
  - Periodic se citesc senzorii, iar task-ul de afișare raportează starea curentă.
- Secvența 4 - Modul Panică și Resetare:
  - Operatorul observă un incident grav și declanșează alarma (se apasă butonul fizic de panică; rutina de întrerupere activează semaforul pentru task-ul dedicat).
  - Sistemul intră instantaneu în modul de protecție totală (task-ul Panică inversează starea variabilei globale; serverul comandă închiderea tuturor barierelor, atât la intrări

cât și la ieșiri).

- Traficul și senzorii sunt ignorați temporar (indiferent de poziția potențiometrelor sau scăderea numărului de mașini, barierele rămân închise cât timp panica este activă).
- Operatorul dezactivează alarma (se apasă din nou butonul de panică).
- Sistemul revine la starea de funcționare anterioară (indicatorul de panică este oprit; serverul verifică din nou senzorii și traficul, ridicând barierele doar dacă condițiile sunt sigure).

- Secvența 5 - Suprapunere de condiții de închidere (Trafic urmat de Gaz):
  - Tunelul se aglomerează până la atingerea capacității maxime (se apasă butonul de intrare până când contorul de mașini atinge limita setată).
  - Sistemul blochează automat intrările din cauza lipsei de spațiu (serverul detectează că tunelul este plin și aprinde led-ul roșu de la intrare).
  - În timp ce barierele sunt deja coborâte, apare o acumulare de gaze nocive (se rotește potențiometrul de gaz peste pragul de alertă; flag-ul de pericol devine activ).
  - Câteva mașini părăsesc tunelul, eliberând locuri (se apasă butonul de ieșire; contorul de mașini scade sub limita maximă).
  - Barierele rămân închise, deși acum există spațiu fizic, deoarece pericolul de gaz persistă (serverul verifică condițiile în ordine prioritară și vede că, deși restricția de trafic a dispărut, cea de mediu este activă).
  - Nivelul gazului revine la normal după ventilare (se rotește potențiometrul înapoi sub pragul de declanșare).
  - Abia în acest moment sistemul redeschide intrările (serverul constată că nu mai există nicio restricție activă, nici de trafic, nici de gaz, și stinge led-ul).

- Secvența 6 - Control Manual prin terminalul serial:
  - Operatorul preia controlul asupra unei bariere (se introduce un caracter '1'-'4' în terminalul serial).
  - Sistemul trece în mod manual și ignoră automatizările (task-ul manual setează variabila `modManual` și schimbă starea barierei selectate; serverul execută strict comanda operatorului).
  - Condițiile de mediu sunt perfecte, dar bariera rămâne închisă (deși potențiometrele sunt la minim și tunelul e gol, led-ul rămâne aprins datorită forțării manuale).
  - Operatorul redă controlul sistemului automat (se introduce caracterul '0' de resetare în terminal).
  - Sistemul re-evaluează situația și revine la comportamentul autonom (serverul vede că modul manual este dezactivat, verifică senzorii și acționează barierele corespunzător).

- Secvența 7 - Tratarea erorilor la ieșire (Tunel Gol):
  - Tunelul este gol (contorul de mașini este zero).
  - Senzorul de ieșire detectează o mașină în mod eronat sau o declanșare falsă (se apasă butonul de ieșire).
  - Protecție software: Logica task-ului de ieșire include o verificare explicită (`if nr > 0`) înainte de decrementare. Astfel, sistemul ignoră comanda de scădere, prevenind apariția unui număr negativ de mașini, ceea ce asigură integritatea datelor în cazul erorilor de senzor.
- Secvența 8 - Prioritate și persistența stării (Panică suprapusă cu Manual):
  - Sistemul se află în starea de Panică (butonul a fost apăsat anterior; toate barierele sunt închise, led-urile roșii sunt aprinse).
  - Operatorul decide să intervină manual pentru a permite accesul unui vehicul de intervenție (se introduce comanda '234' în terminal).
  - Modul Manual devine activ și suprascrie starea de panică (deoarece Manual are prioritate absolută în cod, serverul execută comanda operatorului și închide barierele 2, 3, respectiv 4, 1 rămânând deschisă și ignorând temporar variabila de panică).
  - Vehiculul de intervenție trece, iar operatorul dezactivează modul manual (se introduce comanda '0' - Reset).
  - Sistemul revine în modul automat, dar detectează că starea de panică nu a fost anulată (variabila globală `panica` este încă `true`).
  - Barierele se închid instantaneu la loc (Serverul evaluează prioritățile: Manual e oprit → Verifică Panică → Găsește Panică activă → Închide tot).
  - Operatorul dezactivează oficial alarma (se apasă din nou butonul fizic de panică).
  - Abia acum sistemul revine la funcționarea normală, bazată pe trafic și senzori.
- Secvența 9 - Anomalie logică permisă (Incrementare contor în timpul Panicii):
  - Sistemul este în mod Panică (toate barierele sunt coborâte, accesul este interzis).
  - Se acționează butonul de intrare (simulând trecerea forțată a unui vehicul sau o declanșare eronată a senzorului).
  - Contorul de mașini este incrementat (Task-ul de trafic preia semnalul și actualizează variabila globală, deoarece logica de citire a senzorilor nu este condiționată de starea barierelor sau de modul Panică).
  - Rezultat: Sistemul raportează o mașină în plus în tunel. Deși fizic acest scenariu este improbabil fără distrugerea barierei, codul permite înregistrarea evenimentului, asigurând astfel monitorizarea continuă a intrusilor chiar și atunci când accesul este teoretic blocat.

3 Aplicația. Structura și soluția de implementare propusă

3.1 Definirea structurii aplicației

Aplicația este divizată într-un set de task-uri distincte, gestionate de sistemul de operare în timp real (FreeRTOS), corespunzătoare entităților care interacționează în sistem sau execută operații specifice. Acestea sunt: **Citire-Senzor** (pentru achiziția datelor brute), **Verificare Nivel Senzor** (pentru interpretarea stării de pericol), **Gestionare Trafic** (Intrare/Ieșire pe fiecare sens), **Monitorizare Panică**, **Control Manual**, task-ul coordonator **Server**, setul de task-uri pentru acționare **Închidere/Deschidere Barieră** și, în final, task-ul de **Afișare**.

În această implementare, fiecare funcționalitate corespunde unuia sau mai multor fire de execuție (thread-uri), iar sincronizarea resurselor partajate se realizează prin Mutex-uri și Semafoare Binare. De asemenea, pentru detectarea evenimentelor externe (apăsarea butoanelor), s-au utilizat rutine de tratare a întreruperilor (ISR).

Task-urile de tip Citire Senzor

Acest tip de task are rolul de a prelua informația analogică de la pini (în cazul nostru, un senzor de gaz și un senzor de fum, simulați prin potențiometre). Valoarea citită ($0-1023$) este convertită într-un procent ($0-100\%$) și stocată într-o variabilă globală (ex. `nivel_gaze`), accesul la aceasta fiind protejat printr-un Mutex dedicat (`Mutex_Gaze` respectiv `Mutex_Fum`).

Această operație rulează ciclic pe tot parcursul funcționării programului, cu o frecvență specifică (un delay de 200ms), asigurând actualizarea constantă a datelor de mediu.

Fiind vorba doar de o interpolare simplă, am ales să prelucrăm datele (conversia în procente) direct în acest task pentru a simplifica implementarea, deși pentru o rigurozitate maximă procesarea ar fi putut fi decuplată complet de achiziția propriu-zisă.

Task-urile de tip Verificare Nivel Senzor

Aceste task-uri funcționează în paralel cu cele de citire și au rolul de a monitoriza dacă valorile achiziționate depășesc pragurile critice prestabilite (`prag_gaz`, `prag_fum`).

Task-ul preia valoarea curentă (utilizând Mutex-ul senzorului respectiv) și o compară cu pragul de alertă. În cazul în care se detectează o depășire, task-ul va semnala starea de pericol prin actualizarea unei variabile globale de tip flag (`pericol_gaze` sau `pericol_fum`), protejată de un

Mutex general de Pericol. Această informație este vitală pentru task-ul Server, care va declanșa procedura de evacuare.

Separarea logicii de verificare de cea de citire reprezintă o bună practică de programare, permițând modificarea pragurilor sau a logicii de detecție fără a afecta rutina de achiziție a datelor.

ISR-ul și Mecanismul de Debounce

Pentru a semnala apăsarea butoanelor (Intrare/Ieșire sensuri, Panică), se utilizează Rutine de Tratare a Întreruperilor (ISR). Deoarece apăsarea unui buton este un eveniment asincron, ISR-ul permite microcontrolerului să reacționeze instantaneu.

Totuși, pentru a menține execuția ISR-ului cât mai scurtă și a nu bloca procesorul, am adoptat o strategie de procesare amânată a evenimentului. Concret, ISR-ul execută o singură instrucțiune rapidă: eliberează un semafor binar (`xSemaphoreGiveFromISR`) pentru a semnala producerea evenimentului.

Toată logica consumatoare de timp (cum ar fi așteptarea pentru filtrarea zgomotului mecanic (debounce) și verificarea stării pinului) este delegată către task-ul asociat, care așteaptă deblocarea acestui semafor. Astfel, întreruperea rămâne liberă să detecteze noi evenimente imediat, în timp ce logica complexă este gestionată ordonat de către planificatorul de task-uri.

Task-urile de tip Gestionare Trafic (Intrare/Ieșire)

Există 4 instanțe ale acestui tip de task, corespunzătoare celor două sensuri de mers (Intrare/Ieșire, Sens 1 și Sens 2). Aceste task-uri sunt, în marea majoritate a timpului, în stare *Blocked*, așteptând semaforul de la ISR-ul asociat.

Fluxul de execuție la apăsarea unui buton este următorul:

1. ISR-ul detectează frontul crescător și eliberează semaforul.
2. Task-ul se deblochează și execută o așteptare scurtă (50ms - software debounce) pentru a filtra oscilațiile mecanice ale butonului.
3. Se verifică din nou starea fizică a pinului. Dacă butonul este încă apăsat, se consideră o comandă validă.
4. Se actualizează contorul de mașini (`nrMSens1` sau `nrMSens2`) într-o zonă critică protejată de `Mutex_Trafic`.

Logica de incrementare/decrementare ține cont de limitele tunelului (N mașini). Acest task nu ia decizia de a închide bariera, ci doar actualizează starea traficului. Decizia de închidere se ia centralizat în task-ul Server, pe baza numărului de mașini raportat aici.

Task-ul Panică

Acest task este responsabil de gestionarea butonului de panică, utilizând același mecanism de sincronizare cu ISR-ul descris anterior. Concret, după ce ISR-ul semnalează evenimentul, task-ul preia execuția, aplică filtrul de debounce și accesează variabila globală `panica` într-o zonă critică protejată de `Mutex_Panica`.

Acțiunea executată este de inversare a stării curente (logică de tip *toggle*):

- Dacă sistemul este în stare normală, o apăsare activează modul Panică.
- Dacă sistemul este deja în Panică, o nouă apăsare îl dezactivează (resetare).

Pentru a asigura stabilitatea comenzii și a evita oscilațiile multiple la o singură interacțiune, task-ul include la final o perioadă de așteptare suplimentară ("timp mort") de 500ms înainte de a fi pregătit pentru o nouă citire.

Task-ul Manual

Acest task are rolul de a prelua comenzile operatorului uman prin intermediul interfeței seriale și de a gestiona tranziția între regimurile de funcționare ale sistemului.

Din punct de vedere arhitectural, s-a optat pentru un model bazat pe stări partajate. Concret, responsabilitatea comutării între modurile **AUTOMAT** și **MANUAL** este gestionată direct de acest task, care actualizează variabila globală `modManual` (protejată prin Mutex). Această strategie asigură o decuplare eficientă: task-ul acesta se ocupă exclusiv de interpretarea asincronă a comenzilor text (operație lentă), în timp ce Serverul rămâne concentrat pe bucla critică de control, reacționând la schimbarea de stare în următorul său ciclu de execuție.

Comenzile implementate sunt:

- **'0'** - Comandă de Resetare: Trecerea sistemului în modul **AUTOMAT**. Toate forțările manuale sunt șterse, iar variabila `modManual` devine `false`.
- **'1', '2', '3', '4'** - Comandă de Intervenție: Orice tastă din acest set activează modul **MANUAL** (`modManual` devine `true`). Simultan, se inversează (toggle) starea dorită pentru bariera asociată tastei (In1, Out1, In2, respectiv Out2).

Un detaliu important al logicii de control este starea inițială: la momentul comutării în modul manual, toate barierele sunt considerate implicit „deschise" (variabilele de stare sunt inițializate cu 0). Astfel, operatorul acționează selectiv doar asupra barierelor pe care dorește să le închidă, sistemul pornind dintr-o stare neutră. O posibilă direcție de îmbunătățire pentru dezvoltări ulterioare ar fi ca starea de inițializare a modulului manual să reflecte în mod fidel starea curentă a barierelor.

Task-ul actualizează flag-urile de stare manuală (`manualIn1`, etc.) sub protecția `Mutex_Manual`, transmițând intenția operatorului către Server, care deține autoritatea decizională finală asupra execuției comenzilor.

Task-ul Server

Acesta este nucleul logic al aplicației, rulând într-o buclă infinită pentru a coordona activitatea tunelului. Serverul centralizează informațiile provenite de la task-urile de monitorizare și interfață, luând decizii de acționare a barierelor pe baza unei ierarhii stricte de priorități.

Ordinea de evaluare a condițiilor și acțiunile aferente sunt următoarele:

1. **MOD MANUAL (Prioritate Absolută):** În prima instanță, Serverul verifică dacă operatorul a activat modul manual. Această stare are prioritate superioară oricărei alte condiții. Logica este construită astfel pentru a permite intervenția umană în situații excepționale (de exemplu, defectarea senzorilor sau necesitatea ridicării barierelor pentru accesul vehiculelor de urgență în timpul unui incendiu). Cât timp modul manual este activ, Serverul ignoră automatizările și execută strict comenzile operatorului.

2. **PANICĂ:** Dacă sistemul nu se află în mod manual, se verifică starea globală de panică. Dacă variabila `panica` este activă, Serverul comandă închiderea imediată a tuturor barierelor (intrări și ieșiri) pentru izolarea tunelului. Este important de notat că activarea modului manual nu resetează variabila de panică, ci doar îi suspendă efectul temporar; la revenirea în modul automat, dacă panica nu a fost dezactivată explicit, sistemul va reveni în starea de izolare.

3. **PERICOL (Gaz/Fum):** În absența intervenției manuale și a stării de panică, se analizează datele de la senzorii de mediu. Dacă se detectează depășirea pragurilor admise pentru gaze sau fum, sistemul intră în procedura de evacuare: intrările se închid pentru a opri accesul, iar ieșirile sunt forțate pe deschis.

4. **TRAFIC NORMAL (Automat):** Aceasta este starea implicită de funcționare. Dacă nu există alerte active, Serverul reglează fluxul de trafic exclusiv pe baza contoarelor de mașini, menținând numărul de vehicule sub capacitatea maximă N prin închiderea și deschiderea dinamică a intrărilor.

Pentru execuția comenzilor, Serverul nu trimite direct semnale către pini, ci deblochează semafoarele task-urilor dedicate de barieră (ex. `xSemaphoreGive(semInchideIn1)`). De asemenea, serverul menține o memorie locală a stării barierelor pentru a nu trimite comenzi redundante (ex. nu încearcă să închidă o barieră deja închisă).

Task-urile de tip Acționare Barieră (Închidere/Deschidere)

Deoarece acționarea unei bariere reale implică timp și logică specifică, s-au creat 8 task-uri dedicate (câte o pereche *Închide/Deschide* pentru fiecare din cele 4 puncte de acces).

Aceste task-uri sunt extrem de simple în simulare: așteaptă un semafor binar de la Server și, la primirea acestuia, modifică starea pinului asociat (aprinderea LED-ului pentru închis, stingerea pentru deschis). Această arhitectură decuplată permite ca, într-o implementare fizică, înlocuirea LED-ului cu un motor pas-cu-pas sau un servomotor să se facă doar modificând aceste task-uri, fără a altera logica complexă a Serverului.

Task-ul Afișare

Pentru monitorizarea sistemului, task-ul de afișare scrie periodic (la fiecare 500ms) starea completă a sistemului pe portul serial. Acesta preia valorile variabilelor globale folosind mutex-urile aferente și afișează:

- Nivelul de gaz și fum (procentual) și statusul de pericol.
- Numărul de mașini din fiecare sens.
- Starea modului Panică.
- Modul de operare curent (Automat/Manual) și starea switch-urilor virtuale în cazul modului manual.

3.1.1 Observații și Concluzii

O îmbunătățire posibilă pentru o versiune viitoare ar fi înlocuirea semafoarelor de comandă pentru bariere cu cozi de mesaje (*Queues*) sau notificări directe către task (*Task Notifications*). Deși semafoarele binare funcționează corect în acest context, o coadă ar permite stocarea unei secvențe de comenzi în cazul în care bariera are un timp de acționare lung (hardware real), evitând pierderea unor comenzi rapide succesive.

Apăsarea butoanelor specifică senzorilor de detectare nu ține cont de starea barierelor, deoarece nu ar avea sens condiționarea detecției de permisiunea de trecere. Astfel, în implementarea noastră, vehiculele pot intra și ieși prin acționarea butoanelor chiar și când barierele sunt închise. Deși în realitate acest lucru este fizic imposibil, am decis să nu blocăm logica senzorilor pentru a putea monitoriza eventualele cazuri excepționale care ar necesita atenție sporită din partea controlului. Totuși, am implementat o limitare strictă la numărul maxim de mașini în tunel, care pentru analiza acestor anomali ar trebui eliminată. În orice caz, această investigație fost eliminată din obiectivele actuale ale acestui proiect.

Utilizarea unui RTOS permite obținerea unui comportament determinist al sistemului, chiar și în condiții de încărcare ridicată. Prin planificarea predictibilă a task-urilor și mecanismele de sincronizare oferite, sistemul răspunde prompt la evenimentele critice, menținând un nivel ridicat de robustețe și fiabilitate. Această abordare asigură respectarea cerințelor temporale specifice aplicațiilor embedded critice.

3.2 Definirea soluției în vederea implementării

Soluția a fost implementată pe o placuță Arduino Mega, folosind FreeRTOS.

Pentru a putea satisface condițiile de funcționare corectă a sistemului nostru de tuneluri, am ales să utilizăm următoarele mecanisme:

- 4 butoane de control al traficului (asociate intrărilor și ieșirilor pe cele două sensuri), având rolul de a incrementa/decrementa numărul curent de vehicule din tunel și de a simula senzorii de prezență;
- 1 buton de panică, configurat cu prioritate critică, care declanșează o rutină de urgență pentru blocarea instantanee a tuturor intrărilor, indiferent de starea senzorilor sau a modului de lucru (automat/manual);
- 2 potențiometre pentru simularea senzorilor de mediu, citite periodic printr-un task dedicat, utilizate pentru a stabili nivelul curent de fum și concentrația de gaze naturale;
- 4 LED-uri de semaforizare, controlate de logica centrală a sistemului, care oferă feedback vizual imediat șoferilor (Pornit/Oprit) în funcție de capacitatea tunelului și de parametrii de siguranță monitorizați.

Pentru a asigura coerența datelor și execuția deterministă a sarcinilor, soluția implementată utilizează două mecanisme principale de sincronizare oferite de FreeRTOS: Semafoare Binare și Mutex-uri.

1. Semafoare Binare pentru Sincronizarea Evenimentelor (Signaling)

Semafoarele binare au fost utilizate pentru a semnala apariția unui eveniment și pentru a decupla momentul detecției de momentul procesării efective. În aplicație, acestea deservesc două scenarii distincte:

- **Gestionarea Întreruperilor (ISR to Task):** Pentru butoanele fizice (Intrare/Ieșire și Panică), am utilizat semafoare (`semIn1`, `semOut1`, `semPanica`, etc.) pentru a implementa mecanismul de „Deferred Interrupt Processing". Rutina de întrerupere (ISR) este foarte scurtă și doar eliberează semaforul (`xSemaphoreGiveFromISR`), în timp ce task-ul asociat, care așteaptă semaforul, preia execuția complexă (debounce software și actualizarea logicii).
- **Comanda Barierelor (Task to Task):** Pentru a separa logica de decizie a Serverului de acționarea fizică a pinilor, am folosit un set de semafoare dedicate fiecărei acțiuni (`semInchideIn1`, `semDeschideIn1`, etc.). Serverul „deblochează" task-ul de barieră specific doar când este necesară o schimbare de stare, eficientizând utilizarea procesorului.

2. Mutex-uri pentru Protecția Resurselor Partajate (Mutual Exclusion)

Deoarece variabilele globale sunt accesate concurent de multiple task-uri (de exemplu, task-ul de citire scrie valoarea, iar cel de afișare o citește), am utilizat Mutex-uri pentru a proteja secțiunile critice și a preveni coruperea datelor („Race Conditions").

S-au definit următoarele zone critice protejate:

- **Trafic (`Mutex_Trafic`):** Protejează variabilele `nrMSens1` și `nrMSens2`. Acestea sunt modificate de task-urile de intrare/ieșire și citite de Server și de task-ul de Afișare.
- **Mediu (`Mutex_Gaze`, `Mutex_Fum`):** Asigură accesul exclusiv la variabilele `nivel_gaze` și `nivel_fum` în timpul scrierii de către task-urile de citire și al verificării de către task-urile de nivel.
- **Stare Pericol (`Mutex_Pericol`):** Protejează flag-urile `pericol_gaze` și `pericol_fum`, asigurând că decizia de evacuare luată de Server se bazează pe o stare consistentă.
- **Control Sistem (`Mutex_Panica`, `Mutex_Manual`):** Aceste mutex-uri protejează variabilele de stare care schimbă fundamental modul de operare al sistemului, prevenind conflictele între comenzile operatorului și logica automată.

Utilizarea acestor mecanisme garantează că nicio variabilă nu este citită în timp ce este modificată și că evenimentele externe critice (precum apăsarea butonului de panică) sunt tratate cu prioritate prin deblocarea imediată a task-urilor responsabile.

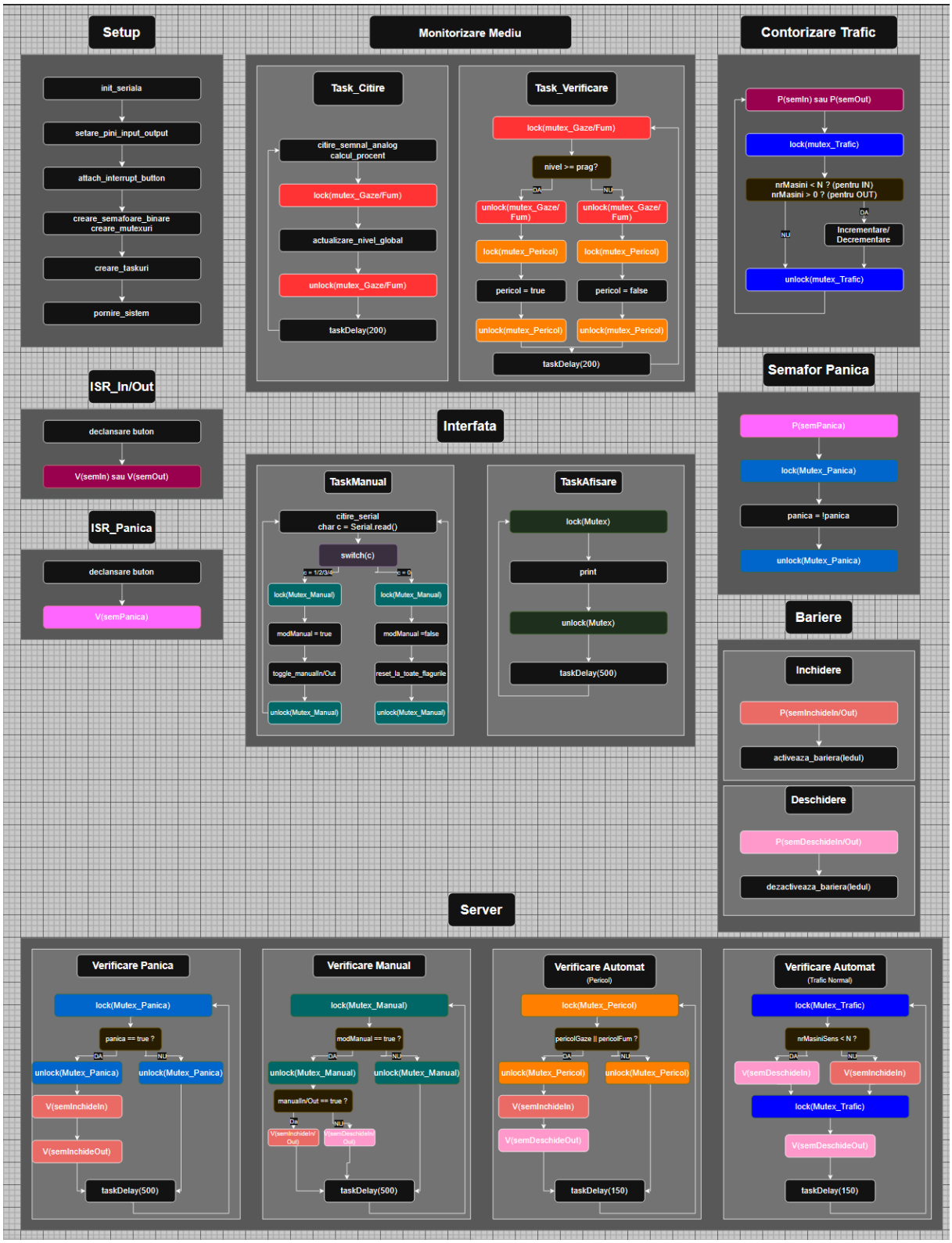

Figura 3.1: Organigrama taskurilor

3.3 Implementarea soluției

```
#include <Arduino_FreeRTOS.h>
#include <semphr.h>

// --- DEFINIŢII PINI ---
#define PIN_LED_In1   13
#define PIN_LED_Out1  5
#define PIN_LED_In2   12
#define PIN_LED_Out2  4
#define PIN_LED_Panica  6

#define PIN_GAZE  A14
#define PIN_FUM   A15

// Butoane (Intreruperi)
#define PIN_In1   2
#define PIN_Out1  18
#define PIN_In2   3
#define PIN_Out2  19
#define PIN_Panica  20

// --- VARIABILE GLOBALE ---
int nrMSens1 = 0;
int nrMSens2 = 0;

float nivel_gaze = 0;
float nivel_fum = 0;
bool pericol_fum = false;
bool pericol_gaze = false;
bool panica = false;

// Variabile control Manual
bool modManual = false;
bool manualIn1 = false, manualOut1 = false;
bool manualIn2 = false, manualOut2 = false;


// --- SEMAFOARE & MUTEX ---
SemaphoreHandle_t semIn1, semOut1, semIn2, semOut2, semPanica;

// Semafoare pentru bariere
SemaphoreHandle_t semInchideIn1, semDeschideIn1;
SemaphoreHandle_t semInchideOut1, semDeschideOut1;
SemaphoreHandle_t semInchideIn2, semDeschideIn2;
SemaphoreHandle_t semInchideOut2, semDeschideOut2;

SemaphoreHandle_t Mutex_Trafic;
SemaphoreHandle_t Mutex_Gaze, Mutex_Fum;
SemaphoreHandle_t Mutex_Pericol;
SemaphoreHandle_t Mutex_Panica;
SemaphoreHandle_t Mutex_Manual;

const int N = 10; // maxim numar masini pe sens
const int prag_gaz = 60;
const int prag_fum = 70;
const unsigned long debounce_delay = 250;

// --- ISR-uri (Doar dau semaforul) ---
// Variabilele "last_interrupt" nu mai sunt critice daca facem debounce in Task,
```

```
// dar le pastram pentru siguranţă.
volatile unsigned long last_time_In1 = 0;
void ISR_In1() {
  if(millis() - last_time_In1 > debounce_delay){
    xSemaphoreGiveFromISR(semIn1, NULL);
    last_time_In1 = millis();
  }
}

volatile unsigned long last_time_Out1 = 0;
void ISR_Out1() {
  if(millis() - last_time_Out1 > debounce_delay){
    xSemaphoreGiveFromISR(semOut1, NULL);
    last_time_Out1 = millis();
  }
}

volatile unsigned long last_time_In2 = 0;
void ISR_In2() {
  if(millis() - last_time_In2 > debounce_delay){
    xSemaphoreGiveFromISR(semIn2, NULL);
    last_time_In2 = millis();
  }
}

volatile unsigned long last_time_Out2 = 0;
void ISR_Out2() {
  if(millis() - last_time_Out2 > debounce_delay){
    xSemaphoreGiveFromISR(semOut2, NULL);
    last_time_Out2 = millis();
  }
}

volatile unsigned long last_time_Panic = 0;
void ISR_Panica() {
  if(millis() - last_time_Panic > debounce_delay){
    xSemaphoreGiveFromISR(semPanica, NULL);
    last_time_Panic = millis();
  }
}

// --- PROTOTIPURI TASK-URI ---
void TaskCitireGaze( void *pvParameters );
void TaskCitireFum( void *pvParameters );
void TaskVerificareNivelGaze( void *pvParameters );
void TaskVerificareNivelFum( void *pvParameters );
void TaskInSens1( void *pvParameters );
void TaskOutSens1( void *pvParameters );
void TaskInSens2( void *pvParameters );
void TaskOutSens2( void *pvParameters );
void TaskPanica( void *pvParameters );
void TaskManual( void *pvParameters );
void TaskServer( void *pvParameters );
void TaskAfisare( void *pvParameters );
// Cele 8 Task-uri specifice pentru bariere:
void TaskInchideBarieraIn1( void *pvParameters );
void TaskInchideBarieraIn2( void *pvParameters );
void TaskInchideBarieraOut1( void *pvParameters );
void TaskInchideBarieraOut2( void *pvParameters );
void TaskDeschideBarieraIn1( void *pvParameters );
void TaskDeschideBarieraIn2( void *pvParameters );
```

```
120 void TaskDeschideBarieraOut1( void *pvParameters );
121 void TaskDeschideBarieraOut2( void *pvParameters );
122
123
124 void setup() {
125   Serial.begin(115200);
126   while(!Serial);
127
128   // Configurare Pini
129   pinMode(PIN_LED_In1, OUTPUT);
130   pinMode(PIN_LED_Out1, OUTPUT);
131   pinMode(PIN_LED_In2, OUTPUT);
132   pinMode(PIN_LED_Out2, OUTPUT);
133   pinMode(PIN_LED_Panica, OUTPUT);
134
135   // BUTOANE: INPUT simplu (cu rezistente externe Pull-Down)
136   pinMode(PIN_In1, INPUT);
137   pinMode(PIN_Out1, INPUT);
138   pinMode(PIN_In2, INPUT);
139   pinMode(PIN_Out2, INPUT);
140   pinMode(PIN_Panica, INPUT);
141
142   // Creare Semafoare
143   semIn1 = xSemaphoreCreateBinary(); semOut1 = xSemaphoreCreateBinary();
144   semIn2 = xSemaphoreCreateBinary(); semOut2 = xSemaphoreCreateBinary();
145   semPanica = xSemaphoreCreateBinary();
146
147   semInchideIn1 = xSemaphoreCreateBinary(); semDeschideIn1 =
        xSemaphoreCreateBinary();
148   semInchideOut1 = xSemaphoreCreateBinary(); semDeschideOut1 =
        xSemaphoreCreateBinary();
149   semInchideIn2 = xSemaphoreCreateBinary(); semDeschideIn2 =
        xSemaphoreCreateBinary();
150   semInchideOut2 = xSemaphoreCreateBinary(); semDeschideOut2 =
        xSemaphoreCreateBinary();
151
152   Mutex_Trafic = xSemaphoreCreateMutex();
153   Mutex_Gaze = xSemaphoreCreateMutex();
154   Mutex_Fum = xSemaphoreCreateMutex();
155   Mutex_Pericol = xSemaphoreCreateMutex();
156   Mutex_Panica = xSemaphoreCreateMutex();
157   Mutex_Manual = xSemaphoreCreateMutex();
158
159   // CREARE TASK-URI
160   // Folosim stack minim (100) la task-urile simple pentru a nu umple memoria
161   xTaskCreate(TaskCitireGaze, "Gaze", 100, NULL, 1, NULL);
162   xTaskCreate(TaskCitireFum, "Fum", 100, NULL, 1, NULL);
163   xTaskCreate(TaskVerificareNivelGaze, "VerifG", 100, NULL, 1, NULL);
164   xTaskCreate(TaskVerificareNivelFum, "VerifF", 100, NULL, 1, NULL);
165
166   xTaskCreate(TaskInSens1, "In1", 128, NULL, 2, NULL);
167   xTaskCreate(TaskOutSens1, "Out1", 128, NULL, 2, NULL);
168   xTaskCreate(TaskInSens2, "In2", 128, NULL, 2, NULL);
169   xTaskCreate(TaskOutSens2, "Out2", 128, NULL, 2, NULL);
170
171   xTaskCreate(TaskPanica, "Panic", 128, NULL, 3, NULL);
172   xTaskCreate(TaskManual, "Man", 128, NULL, 1, NULL);
173   xTaskCreate(TaskServer, "Srv", 200, NULL, 2, NULL); // Serverul e complex, are
        nevoie de stack
174
175   // Cele 8 Task-uri de bariera - Stack MIC (80-100)
```

```
176   xTaskCreate(TaskInchideBarieraIn1,  "C_In1", 85, (void*)PIN_LED_In1, 2, NULL);
177   xTaskCreate(TaskDeschideBarieraIn1, "O_In1", 85, (void*)PIN_LED_In1, 2, NULL);
178   xTaskCreate(TaskInchideBarieraOut1,  "C_Out1", 85, (void*)PIN_LED_Out1,2, NULL
      );
179   xTaskCreate(TaskDeschideBarieraOut1, "O_Out1", 85, (void*)PIN_LED_Out1,2, NULL
      );
180   xTaskCreate(TaskInchideBarieraIn2,  "C_In2", 85, (void*)PIN_LED_In2, 2, NULL);
181   xTaskCreate(TaskDeschideBarieraIn2, "O_In2", 85, (void*)PIN_LED_In2,2, NULL);
182   xTaskCreate(TaskInchideBarieraOut2,  "C_Out2", 85, (void*)PIN_LED_Out2,2, NULL
      );
183   xTaskCreate(TaskDeschideBarieraOut2, "O_Out2", 85, (void*)PIN_LED_Out2,2, NULL
      );
184
185   xTaskCreate(TaskAfisare, "Disp", 256, NULL, 1, NULL);
186
187   // INTRERUPERI LA FINAL (RISING pentru Pull-Down)
188   attachInterrupt(digitalPinToInterrupt(PIN_In1), ISR_In1, RISING);
189   attachInterrupt(digitalPinToInterrupt(PIN_Out1), ISR_Out1, RISING);
190   attachInterrupt(digitalPinToInterrupt(PIN_In2), ISR_In2, RISING);
191   attachInterrupt(digitalPinToInterrupt(PIN_Out2), ISR_Out2, RISING);
192   attachInterrupt(digitalPinToInterrupt(PIN_Panica), ISR_Panica, RISING);
193
194   Serial.println("Sistem initializat.");
195   vTaskStartScheduler();
196 }
197
198 void loop() {}
199
200 // --- IMPLEMENTARE TASK-URI ---
201
202 // Citire Senzori
203 void TaskCitireGaze(void *pvParameters) {
204     for(;;) {
205         int gaze = analogRead(PIN_GAZE);
206         float procent = (gaze / 1023.0) * 100.0;
207         if(xSemaphoreTake(Mutex_Gaze, portMAX_DELAY) == pdTRUE){
208            nivel_gaze = procent;
209            xSemaphoreGive(Mutex_Gaze);
210         }
211         vTaskDelay(200 / portTICK_PERIOD_MS);
212     }
213 }
214
215 void TaskCitireFum(void *pvParameters) {
216     for(;;) {
217         int fum = analogRead(PIN_FUM);
218         float procent = (fum / 1023.0) * 100.0;
219         if(xSemaphoreTake(Mutex_Fum, portMAX_DELAY) == pdTRUE){
220           nivel_fum = procent;
221           xSemaphoreGive(Mutex_Fum);
222         }
223         vTaskDelay(200 / portTICK_PERIOD_MS);
224     }
225 }
226
227 // Verificare Nivel
228 void TaskVerificareNivelGaze(void *pvParameters) {
229   for(;;) {
230     bool local_pericol = false;
231     if (xSemaphoreTake(Mutex_Gaze, portMAX_DELAY) == pdTRUE) {
232       if (nivel_gaze > prag_gaz) local_pericol = true;
```

```
233       xSemaphoreGive(Mutex_Gaze);
234     }
235
236     xSemaphoreTake(Mutex_Pericol, portMAX_DELAY);
237     pericol_gaze = local_pericol;
238     xSemaphoreGive(Mutex_Pericol);
239
240     vTaskDelay(200 / portTICK_PERIOD_MS);
241   }
242 }
243
244 void TaskVerificareNivelFum(void *pvParameters) {
245   for(;;) {
246     bool local_pericol = false;
247     if (xSemaphoreTake(Mutex_Fum, portMAX_DELAY) == pdTRUE) {
248       if (nivel_fum > prag_fum) local_pericol = true;
249       xSemaphoreGive(Mutex_Fum);
250     }
251
252     xSemaphoreTake(Mutex_Pericol, portMAX_DELAY);
253     pericol_fum = local_pericol;
254     xSemaphoreGive(Mutex_Pericol);
255
256     vTaskDelay(200 / portTICK_PERIOD_MS);
257   }
258 }
259
260 // Gestionare Trafic (CU DEBOUNCE SOFTWARE)
261 void TaskInSens1(void *pvParameters) {
262   for (;;) {
263     if (xSemaphoreTake(semIn1, portMAX_DELAY) == pdTRUE){
264       vTaskDelay(50 / portTICK_PERIOD_MS); // Debounce
265       if(digitalRead(PIN_In1) == HIGH) { // Verificare fizica
266         if(xSemaphoreTake(Mutex_Trafic, portMAX_DELAY) == pdTRUE){
267           if (nrMSens1 < N) nrMSens1++;
268           xSemaphoreGive(Mutex_Trafic);
269         }
270       }
271     }
272   }
273 }
274
275 void TaskOutSens1(void *pvParameters) {
276   for(;;) {
277     if(xSemaphoreTake(semOut1, portMAX_DELAY) == pdTRUE) {
278       vTaskDelay(50 / portTICK_PERIOD_MS);
279       if(digitalRead(PIN_Out1) == HIGH) {
280         if(xSemaphoreTake(Mutex_Trafic, portMAX_DELAY) == pdTRUE){
281             if(nrMSens1 > 0) nrMSens1--;
282             xSemaphoreGive(Mutex_Trafic);
283         }
284       }
285     }
286   }
287 }
288
289 void TaskInSens2(void *pvParameters) {
290   for (;;) {
291     if (xSemaphoreTake(semIn2, portMAX_DELAY) == pdTRUE){
292       vTaskDelay(50 / portTICK_PERIOD_MS);
293       if(digitalRead(PIN_In2) == HIGH) {
```

```
294         if(xSemaphoreTake(Mutex_Trafic, portMAX_DELAY) == pdTRUE){
295             if (nrMSens2 < N) nrMSens2++;
296             xSemaphoreGive(Mutex_Trafic);
297         }
298       }
299     }
300   }
301 }
302
303 void TaskOutSens2(void *pvParameters) {
304   for(;;) {
305     if(xSemaphoreTake(semOut2, portMAX_DELAY) == pdTRUE) {
306       vTaskDelay(50 / portTICK_PERIOD_MS);
307       if(digitalRead(PIN_Out2) == HIGH) {
308         if(xSemaphoreTake(Mutex_Trafic, portMAX_DELAY) == pdTRUE){
309           if(nrMSens2 > 0) nrMSens2--;
310           xSemaphoreGive(Mutex_Trafic);
311         }
312       }
313     }
314   }
315 }
316
317 void TaskPanica(void *pvParameters) {
318   for(;;) {
319     if(xSemaphoreTake(semPanica, portMAX_DELAY) == pdTRUE) {
320       vTaskDelay(50 / portTICK_PERIOD_MS); // Debounce
321       if(digitalRead(PIN_Panica) == HIGH) {
322         if(xSemaphoreTake(Mutex_Panica, portMAX_DELAY) == pdTRUE){
323           panica = !panica;
324           xSemaphoreGive(Mutex_Panica);
325           vTaskDelay(500 / portTICK_PERIOD_MS); // Delay anti-repetitie
326         }
327       }
328     }
329   }
330 }
331
332 // Cele 8 Task-uri pentru bariere
333 void TaskInchideBarieraIn1( void *pvParameters ){
334   int pin = (int)pvParameters;
335   for(;;) {
336     if(xSemaphoreTake(semInchideIn1, portMAX_DELAY) == pdTRUE) {
337       digitalWrite(pin, HIGH);
338     }
339   }
340 }
341 void TaskInchideBarieraOut1( void *pvParameters ){
342   int pin = (int)pvParameters;
343   for(;;) {
344     if(xSemaphoreTake(semInchideOut1, portMAX_DELAY) == pdTRUE) {
345       digitalWrite(pin, HIGH);
346     }
347   }
348 }
349 void TaskInchideBarieraIn2( void *pvParameters ){
350   int pin = (int)pvParameters;
351   for(;;) {
352     if(xSemaphoreTake(semInchideIn2, portMAX_DELAY) == pdTRUE) {
353       digitalWrite(pin, HIGH);
354     }
```

```
355   }
356 }
357 void TaskInchideBarieraOut2( void *pvParameters ){
358   int pin = (int)pvParameters;
359   for(;;) {
360     if(xSemaphoreTake(semInchideOut2, portMAX_DELAY) == pdTRUE) {
361       digitalWrite(pin, HIGH);
362     }
363   }
364 }
365
366 void TaskDeschideBarieraIn1( void *pvParameters ){
367   int pin = (int)pvParameters;
368   for(;;) {
369     if(xSemaphoreTake(semDeschideIn1, portMAX_DELAY) == pdTRUE) {
370       digitalWrite(pin, LOW);
371     }
372   }
373 }
374 void TaskDeschideBarieraOut1( void *pvParameters ){
375   int pin = (int)pvParameters;
376   for(;;) {
377     if(xSemaphoreTake(semDeschideOut1, portMAX_DELAY) == pdTRUE) {
378       digitalWrite(pin, LOW);
379     }
380   }
381 }
382 void TaskDeschideBarieraIn2( void *pvParameters ){
383   int pin = (int)pvParameters;
384   for(;;) {
385     if(xSemaphoreTake(semDeschideIn2, portMAX_DELAY) == pdTRUE) {
386       digitalWrite(pin, LOW);
387     }
388   }
389 }
390 void TaskDeschideBarieraOut2( void *pvParameters ){
391   int pin = (int)pvParameters;
392   for(;;) {
393     if(xSemaphoreTake(semDeschideOut2, portMAX_DELAY) == pdTRUE) {
394       digitalWrite(pin, LOW);
395     }
396   }
397 }
398
399 void TaskManual(void *pvParameters) {
400   for (;;) {
401     if (Serial.available() > 0) {
402       char c = Serial.read();
403       if(strchr("01234", c)) {
404           xSemaphoreTake(Mutex_Manual, portMAX_DELAY);
405           if(c == '0') {
406              modManual = false; manualIn1=0; manualOut1=0; manualIn2=0;
    manualOut2=0;
407           } else {
408              modManual = true;
409              if(c == '1') manualIn1 = !manualIn1;
410              if(c == '2') manualOut1 = !manualOut1;
411              if(c == '3') manualIn2 = !manualIn2;
412              if(c == '4') manualOut2 = !manualOut2;
413           }
414           xSemaphoreGive(Mutex_Manual);
```

```
      }
    }
    vTaskDelay(200 / portTICK_PERIOD_MS);
  }
}

// SERVERUL
void TaskServer(void *pvParameters) {
  // Variabile de MEMORIE pentru starea barierelor
  // false = Deschisa, true = Inchisa
  // Le initializam false (presupunem deschise la start)
  bool stareIn1_Inchisa = false;
  bool stareOut1_Inchisa = false;
  bool stareIn2_Inchisa = false;
  bool stareOut2_Inchisa = false;

  for(;;) {
    bool localPanica, localManual, localPericol;

    // 1. VERIFICARE MANUAL
    xSemaphoreTake(Mutex_Manual, portMAX_DELAY); localManual = modManual;
   xSemaphoreGive(Mutex_Manual);
    if(localManual) {
        xSemaphoreTake(Mutex_Manual, portMAX_DELAY);

        // --- Manual IN 1 ---
        if(manualIn1 == true && !stareIn1_Inchisa) {
            xSemaphoreGive(semInchideIn1); stareIn1_Inchisa = true;
        } else if (manualIn1 == false && stareIn1_Inchisa) {
            xSemaphoreGive(semDeschideIn1); stareIn1_Inchisa = false;
        }

        // --- Manual OUT 1 ---
        if(manualOut1 == true && !stareOut1_Inchisa) {
            xSemaphoreGive(semInchideOut1); stareOut1_Inchisa = true;
        } else if (manualOut1 == false && stareOut1_Inchisa) {
            xSemaphoreGive(semDeschideOut1); stareOut1_Inchisa = false;
        }

        // --- Manual IN 2 ---
        if(manualIn2 == true && !stareIn2_Inchisa) {
            xSemaphoreGive(semInchideIn2); stareIn2_Inchisa = true;
        } else if (manualIn2 == false && stareIn2_Inchisa) {
            xSemaphoreGive(semDeschideIn2); stareIn2_Inchisa = false;
        }

        // --- Manual OUT 2 ---
        if(manualOut2 == true && !stareOut2_Inchisa) {
            xSemaphoreGive(semInchideOut2); stareOut2_Inchisa = true;
        } else if (manualOut2 == false && stareOut2_Inchisa) {
            xSemaphoreGive(semDeschideOut2); stareOut2_Inchisa = false;
        }

        xSemaphoreGive(Mutex_Manual);
        vTaskDelay(100 / portTICK_PERIOD_MS);
        continue;
    }

    // 2. VERIFICARE PANICA
    xSemaphoreTake(Mutex_Panica, portMAX_DELAY); localPanica = panica;
   xSemaphoreGive(Mutex_Panica);
```

```
474     digitalWrite(PIN_LED_Panica, localPanica ? HIGH : LOW);
475     if (localPanica) {
476         // Vrem TOTUL INCHIS. Verificam daca e deja inchis inainte sa dam
    comanda.
477         if(!stareIn1_Inchisa) { xSemaphoreGive(semInchideIn1); stareIn1_Inchisa
    = true; }
478         if(!stareOut1_Inchisa){ xSemaphoreGive(semInchideOut1);
    stareOut1_Inchisa = true; }
479         if(!stareIn2_Inchisa) { xSemaphoreGive(semInchideIn2); stareIn2_Inchisa
    = true; }
480         if(!stareOut2_Inchisa){ xSemaphoreGive(semInchideOut2);
    stareOut2_Inchisa = true; }
481 
482         vTaskDelay(100 / portTICK_PERIOD_MS);
483         continue; // Sarim peste restul logicii
484     }
485 
486     // 3. VERIFICARE PERICOL
487     xSemaphoreTake(Mutex_Pericol, portMAX_DELAY);
488     localPericol = (pericol_gaze || pericol_fum);
489     xSemaphoreGive(Mutex_Pericol);
490     if(localPericol) {
491        // Inchidem intrarile
492        if(!stareIn1_Inchisa) { xSemaphoreGive(semInchideIn1); stareIn1_Inchisa =
     true; }
493        if(!stareIn2_Inchisa) { xSemaphoreGive(semInchideIn2); stareIn2_Inchisa =
     true; }
494 
495        // Deschidem iesirile (Evacuare)
496        if(stareOut1_Inchisa) { xSemaphoreGive(semDeschideOut1);
    stareOut1_Inchisa = false; }
497        if(stareOut2_Inchisa) { xSemaphoreGive(semDeschideOut2);
    stareOut2_Inchisa = false; }
498 
499        vTaskDelay(150 / portTICK_PERIOD_MS);
500        continue;
501     }
502 
503     // 4. AUTOMAT (Trafic Normal)
504     if(xSemaphoreTake(Mutex_Trafic, portMAX_DELAY) == pdTRUE){
505 
506       // Sens 1 Logic
507       if(nrMSens1 < N) {
508           // Vrem deschis
509           if(stareIn1_Inchisa) { xSemaphoreGive(semDeschideIn1);
    stareIn1_Inchisa = false; }
510       } else {
511           // Vrem inchis
512           if(!stareIn1_Inchisa) { xSemaphoreGive(semInchideIn1);
    stareIn1_Inchisa = true; }
513       }
514 
515       // Sens 2 Logic
516       if(nrMSens2 < N) {
517           if(stareIn2_Inchisa) { xSemaphoreGive(semDeschideIn2);
    stareIn2_Inchisa = false; }
518       } else {
519           if(!stareIn2_Inchisa) { xSemaphoreGive(semInchideIn2);
    stareIn2_Inchisa = true; }
520       }
521 
```

```
522       xSemaphoreGive(Mutex_Trafic);
523     }
524
525     // Iesirile trebuie sa fie deschise in mod normal
526     if(stareOut1_Inchisa) { xSemaphoreGive(semDeschideOut1); stareOut1_Inchisa =
        false; }
527     if(stareOut2_Inchisa) { xSemaphoreGive(semDeschideOut2); stareOut2_Inchisa =
        false; }
528
529     vTaskDelay(150 / portTICK_PERIOD_MS);
530   }
531 }
532
533 void TaskAfisare(void *pvParameters) {
534   // Variabile locale pentru a afisa rapid
535   int localSens1, localSens2;
536
537   for(;;) {
538     Serial.println(" ");
539     Serial.println("====================");
540
541     // --- GAZE ---
542     if(xSemaphoreTake(Mutex_Gaze, portMAX_DELAY) == pdTRUE) {
543       Serial.print("Gaze: ");
544       Serial.print(nivel_gaze);
545       Serial.print("% | Pericol: ");
546       xSemaphoreGive(Mutex_Gaze);
547     }
548
549     if(xSemaphoreTake(Mutex_Pericol, portMAX_DELAY) == pdTRUE) {
550       if(pericol_gaze == true) Serial.println("DA");
551       else Serial.println("NU");
552       xSemaphoreGive(Mutex_Pericol);
553     }
554
555     // --- FUM ---
556     if(xSemaphoreTake(Mutex_Fum, portMAX_DELAY) == pdTRUE) {
557       Serial.print("Fum:  ");
558       Serial.print(nivel_fum);
559       Serial.print("% | Pericol: ");
560       xSemaphoreGive(Mutex_Fum);
561     }
562
563     if(xSemaphoreTake(Mutex_Pericol, portMAX_DELAY) == pdTRUE) {
564       if(pericol_fum == true) Serial.println("DA");
565       else Serial.println("NU");
566       xSemaphoreGive(Mutex_Pericol);
567     }
568
569     // --- TRAFIC ---
570     if(xSemaphoreTake(Mutex_Trafic, portMAX_DELAY) == pdTRUE) {
571       localSens1 = nrMSens1;
572       localSens2 = nrMSens2;
573       xSemaphoreGive(Mutex_Trafic);
574     }
575     Serial.print("Masini sens 1: "); Serial.println(localSens1);
576     Serial.print("Masini sens 2: "); Serial.println(localSens2);
577
578     // --- PANICA ---
579     if(xSemaphoreTake(Mutex_Panica, portMAX_DELAY) == pdTRUE) {
580       Serial.print("Panica: ");
```

```
581       if(panica == true) Serial.println("ACTIVA");
582       else Serial.println("inactiva");
583       xSemaphoreGive(Mutex_Panica);
584     }
585
586     // --- MOD DE OPERARE (MANUAL / AUTOMAT) ---
587     if(xSemaphoreTake(Mutex_Manual, portMAX_DELAY) == pdTRUE) {
588       Serial.print("Mod operare: ");
589       if(modManual == true) Serial.println("MANUAL");
590       else Serial.println("AUTOMAT");
591
592       // Detalii doar daca e Manual
593       if(modManual == true) {
594         Serial.print("  In1: ");
595         if(manualIn1) Serial.println("inchisa"); else Serial.println("deschisa")
    ;
596
597         Serial.print("  Out1: ");
598         if(manualOut1) Serial.println("inchisa"); else Serial.println("deschisa"
    );
599
600         Serial.print("  In2: ");
601         if(manualIn2) Serial.println("inchisa"); else Serial.println("deschisa")
    ;
602
603         Serial.print("  Out2: ");
604         if(manualOut2) Serial.println("inchisa"); else Serial.println("deschisa"
    );
605       }
606       xSemaphoreGive(Mutex_Manual);
607     }
608
609     Serial.println("=========================");
610
611     // Refresh la 0.5 secunde
612     vTaskDelay(500 / portTICK_PERIOD_MS);
613   }
614 }
```

4 Testarea aplicației și validarea soluției propuse

Aplicația a fost analizată și verificată constant pe parcursul definirii logicii de control. Toate mecanismele de gestionare a traficului, precum și funcționalitățile de detecție a pericolelor de mediu, au fost analizate, pe cât posibil, mai întâi individual, pentru a se asigura faptul că acestea funcționează conform cerințelor impuse. Ulterior, acestea au fost corelate în cadrul sistemului complet, astfel încât interacțiunea dintre butoane, potențiometre și indicatorii vizuali să nu conducă la comportamente nedorite.

În ceea ce privește validarea logicii propuse, au fost luate în considerare secvențe de evenimente similare celor descrise în secțiunea 2 a documentației, astfel încât să fie acoperită o plajă cât mai largă de situații posibile. Validarea constă în simularea intrărilor și ieșirilor din tunel prin intermediul celor două butoane, precum și în modificarea valorilor potențiometrelor asociate simulării senzorilor de gaz și fum, în diferite momente ale funcționării sistemului.

În urma observațiilor realizate pe baza acestor secvențe, logica de control a fost ajustată pentru a obține un comportament cât mai apropiat de cel dorit. În forma actuală, sistemul reacționează corect la variațiile numărului de vehicule și la condițiile de mediu simulate, semnalizând corespunzător starea barierelor prin intermediul LED-urilor roșii și respectând prioritățile de siguranță stabilite.

4.1 Scenarii de Testare și Rezultate Experimentale

Pentru a demonstra corectitudinea implementării și robustețea soluției propuse, am izolat și testat patru scenarii critice de funcționare. Figurile de mai jos prezintă capturi din monitorul serial (care expun starea internă a variabilelor și a mutex-urilor) în paralel cu starea fizică a sistemului (LED-urile de pe breadboard).

Caz 1: Detecție Pericol Mediu (Gaz)

În acest scenariu, am simulat o creștere a concentrației de gaz peste pragul critic stabilit la 60%.

- **Comportament observat:** Task-ul de verificare a semnalat depășirea pragului de 60%. Serverul a detectat flag-ul de pericol activat și a comandat închiderea intrării (pentru a bloca accesul) și deschiderea ieșirii (pentru evacuare).
- **Validare:** În consola serială, indicatorul de "Pericol" a trecut pe "DA", iar LED-ul roșu de la intrare s-a aprins.

```
====================
Gaze: 83.19% | Pericol: DA
Fum:  21.51% | Pericol: NU
Masini sens 1: 0
Masini sens 2: 0
Panica: inactiva
Mod operare: AUTOMAT
========================
```

Figura 4.1: Monitor Serial: Nivel gaz > 60%, Pericol activ

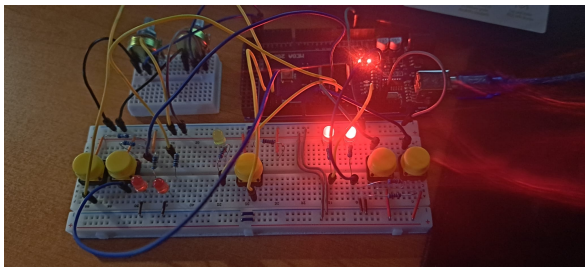

Figura 4.2: Circuit Fizic: Intrare blocată (LED Roșu aprins)

Caz 2: Modul Panică (Urgență)

Acest test validează prioritatea mecanismului de panică. La apăsarea butonului, sistemul trebuie să intre în izolare totală.

- **Comportament observat:** La detectarea apăsării, variabila globală `panica` a fost activată. Serverul a comandat închiderea imediată a tuturor barierelor (intrări și ieșiri), ignorând senzorii de trafic.
- **Validare:** Consola raportează starea "Panica: ACTIVA", iar fizic toate barierele sunt coborâte.

```
====================
Gaze: 3.23% | Pericol: NU
Fum:  0.98% | Pericol: NU
Masini sens 1: 0
Masini sens 2: 0
Panica: ACTIVA
Mod operare: AUTOMAT
========================
```

Figura 4.3: Monitor Serial: Starea Panică este ACTIVA

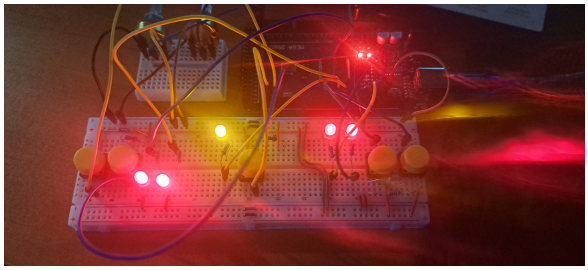

Figura 4.4: Circuit Fizic: Blocare totală (Toate LED-urile aprinse)

Caz 3: Limitare Trafic (Capacitate Maximă)

S-a testat gestionarea fluxului până la atingerea capacității maxime $N = 10$ vehicule pe sens.

- **Comportament observat:** Contorul de mașini a ajuns la 10. Logica serverului (`nrMSens1 >= N`) a declanșat închiderea barierei de intrare pentru a preveni supraaglomerarea.
- **Validare:** Monitorul serial indică 10 mașini pe sens, iar sistemul blochează accesul noilor vehicule.

```
====================
Gaze: 0.29% | Pericol: NU
Fum:  0.68% | Pericol: NU
Masini sens 1: 10
Masini sens 2: 3
Panica: inactiva
Mod operare: AUTOMAT
========================
```

Figura 4.5: Monitor Serial: 10 Mașini (Capacitate atinsă)

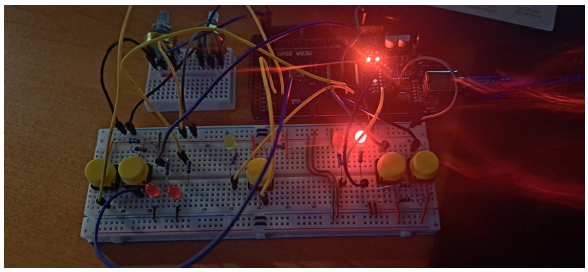

Figura 4.6: Circuit Fizic: Bariera de intrare închisă din cauza aglomerării

Caz 4: Control Manual

Acest scenariu demonstrează flexibilitatea operatorului de a controla individual barierele, suprascriind automatismele.

- **Comportament observat:** Operatorul a transmis succesiv comenzile '1', '2' și '4' prin interfața serială. Sistemul a intrat în modul MANUAL și a forțat închiderea barierelor specifice (Intrare 1, Ieșire 1 și Ieșire 2), lăsând Intrare 2 pe starea anterioară sau deschisă.
- **Validare:** Monitorul serial confirmă explicit starea barierelor: In1, Out1 și Out2 apar marcat ca fiind "inchisa", iar sistemul ignoră logica senzorilor pentru aceste puncte de acces.

```
====================
Gaze: 0.88% | Pericol: NU
Fum:  0.98% | Pericol: NU
Masini sens 1: 6
Masini sens 2: 3
Panica: inactiva
Mod operare: MANUAL
  In1: inchisa
  Out1: inchisa
  In2: deschisa
  Out2: inchisa
========================
```

Figura 4.7: Monitor Serial: Mod MANUAL - In1, Out1, Out2 ÎNCHISE

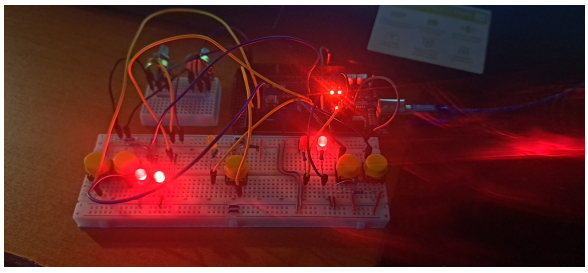

Figura 4.8: Circuit Fizic: 3 LED-uri aprinse (In1, Out1, Out2)

Bibliografie

[1] FreeRTOS.org, *Semaphore and Mutexes — API reference*, https://www.freertos.org/Documentation/02-Kernel/04-API-references/10-Semaphore-and-Mutexes/00-Semaphores, accesat la 5 decembrie 2025.

[2] FreeRTOS.org, *xSemaphoreGiveFromISR — turn a semaphore from an ISR*, https://www.freertos.org/Documentation/02-Kernel/04-API-references/10-Semaphore-and-Mutexes/17-xSemaphoreGiveFromISR, accesat la 5 decembrie 2025.

[3] Arduino.cc, *Arduino MEGA 2560 Hardware Overview*, https://docs.arduino.cc/hardware/mega-2560/, accesat la 5 decembrie 2025.

[4] EmbeddedRelated.com, *Debouncing a Switch (Software Debounce Approaches)*, https://www.embeddedrelated.com/showarticle/1650.php, accesat la 6 ianuarie 2026.