

SEM. I 2024-2025

Proiect PATR

Sistem de monitorizare a confortului termic într-o clădire

Data: 24 februarie 2025

Componența echipei și contribuția individuală (în %)

Membrii	A	C	D	PR	CB	e-mail
Gioancă Elena-Andra	33%	30%	33%	35%	32.75%	elena_andra.gioanca@stud.acs.upb.ro
Grozea Sibel-Florentina	34%	32%	34%	33%	33.25%	sibel.grozea@stud.acs.upb.ro
Rizea Vlad-Gabriel	33%	38%	33%	32%	34%	vlad_gabriel.rizea@stud.acs.upb.ro
	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	1
2	Analiza problemei	3
3	Aplicația. Structura și soluția de implementare propusă	6
3.1	Definirea structurii aplicației	6
3.2	Definirea soluției în vederea implementării	7
3.2.1	Reglarea automată a temperaturii și presiunii	8
3.3	Detalii de implementare	9
3.3.1	Mecanisme de sincronizare și comunicare	9
3.3.2	Biblioteci utilizate	10
3.3.3	Task-uri implementate	10
3.4	Codul complet al aplicației	12
4	Testarea aplicației și validarea soluției propuse	16
4.1	Testare folosind date generate aleator	16
4.2	Concluzii și observații	21

1 Introducere. Definire problemă

Proiectarea unui sistem de monitorizare a confortului termic

În mediile moderne, confortul termic reprezintă un aspect esențial pentru asigurarea calității vieții și a productivității. Pentru a menține condiții optime într-un spațiu, este necesar un sistem capabil să regleze temperatura și presiunea în mod automat și eficient, cu opțiunea de a opera și manual. Acest proiect se concentrează pe dezvoltarea unui astfel de sistem, care include măsurarea temperaturii cu ajutorul termo-cuplurilor, gestionarea presiunii prin senzori și controlul unui dispozitiv de încălzire ("Heater"), precum și afișarea informațiilor relevante pe un ecran LCD.

Se dorește proiectarea și implementarea unui sistem care să monitorizeze confortul termic într-o clădire, în baza cerințelor specificate:

1. Comutator digital (SW):

- Permite utilizatorului să comute între modurile manual și automat, utilizând un buton conectat la pinul digital 2.
- Implementare în cod: funcționalitatea este realizată în `Task_SW`.

2. Măsurarea temperaturii (T):

- Temperatura este simulată cu ajutorul unui potențiomtru conectat la pinul analogic A0 și convertită pentru afișare.
- Implementare în cod: gestionată de `Task_T`.

3. Unitatea de control (S):

- În modul automat:
 - Sistemul pornește/oprește dispozitivul de încălzire (LED-ul conectat la pinul 13) în funcție de temperatura simulată.
 - Implementare în cod: realizată în `Task_S`.
- În modul manual:
 - Utilizatorul poate seta o temperatură țintă simulată (`temperatura_fake`).
 - Sistemul compară temperatura reală cu cea țintă și controlează Heater-ul.
 - Implementare în cod: realizată în `Task_S`.

4. Controlul presiunii (P):

- Presiunea este simulată cu ajutorul unui potențiomtru conectat la pinul analogic A1 și convertită pentru procesare.
- Dacă presiunea scade sub 300, pompele sunt pornite, dar se opresc în caz de supraîncălzire.
- Implementare în cod: realizată în **Task_P**.

5. Interfața utilizatorului:

- Sistemul utilizează un ecran LCD conectat prin I2C pentru a afișa:
 - Temperatura simulată.
 - Modul de operare (manual/automat).
- Implementare în cod: afișarea este realizată în **Task_S**.

Acest sistem integrează funcționalități de monitorizare și control termic, utilizând componente hardware și software pentru gestionarea temperaturii și presiunii într-un mod eficient și flexibil.

2 Analiza problemei

Acest capitol va cuprinde analiza cerințelor și detaliile esențiale ale implementării sistemului de monitorizare a confortului termic.

Descrierea problemei

Problema principală constă în proiectarea unui sistem capabil să monitorizeze și să regleze automat confortul termic într-o clădire. Sistemul trebuie să fie eficient, sigur și să ofere opțiuni de control manual. Aspectele fundamentale care trebuie tratate includ măsurarea temperaturii, gestionarea presiunii, controlul încălzitorului (*Heater*) și afișarea datelor relevante pe un ecran LCD.

Secvențe de operare

Secvențe corecte de operare

- **Secvența 1:**

- Sistemul detectează o temperatură sub pragul minim setat (22°C).
- Modul automat este activ, iar *Heater-ul* este pornit pentru a încălzi clădirea.
- Temperatura este simulată utilizând un potențiomtru conectat la pinul A0 și monitorizată constant.
- Presiunea din instalație, simulată printr-un potențiomtru conectat la pinul A1, este menținută în limitele admise de controlerul de presiune.
- Când temperatura optimă este atinsă, *Heater-ul* este oprit automat.

- **Secvența 2:**

- Modul manual este activat de utilizator printr-un comutator conectat la pinul 2.
- Utilizatorul setează temperatura dorită folosind un potențiomtru suplimentar conectat la pinul A2 (`temperatura_fake`).
- Sistemul ajustează funcționarea *Heater-ului* în funcție de această temperatură țintă.
- Temperatura este afișată în timp real pe un ecran LCD conectat prin I2C.

- **Secvența 3:**

- Presiunea scade sub pragul minim admis (300 unități simulate).
- Sistemul activează pompele pentru a restabili presiunea optimă.
- Monitorizarea constantă asigură că presiunea și temperatura rămân în limitele specificate.

Secvențe greșite de operare

- **Secvența 1:**

- Sistemul detectează o temperatură sub pragul minim.
- *Heater-ul* este pornit, dar presiunea din instalație nu este verificată.
- Presiunea crește peste limitele de siguranță, cauzând o posibilă avarie.

- **Secvența 2:**

- Utilizatorul setează o temperatură țintă prea ridicată în modul manual.
- *Heater-ul* funcționează la capacitate maximă fără protecție împotriva supraîncălzirii.
- Senzorii de temperatură transmit date eronate, iar sistemul nu corectează funcționarea.

- **Secvența 3:**

- Presiunea scade sub pragul minim admis.
- Pompele sunt activate, dar unul dintre senzori transmite date incorecte.
- Presiunea este reglată greșit, ceea ce duce la supraîncărcarea instalației.

Constrângeri identificate

- **Limitări hardware:** Utilizarea unui microcontroller Arduino impune restricții privind resursele de memorie și capacitatea de procesare.
- **Timpi de răspuns:** Sistemul trebuie să reacționeze rapid la variațiile de temperatură și presiune pentru a menține confortul termic.
- **Siguranță:** Prevenirea condițiilor periculoase, cum ar fi supraîncălzirea sau creșterea presiunii peste limitele admise.
- **Complexitatea utilizării:** Interfața trebuie să fie intuitivă și ușor de utilizat, atât în modul manual, cât și în cel automat.

Aspecte adiționale de luat în considerare

- **Eficiență energetică:** Sistemul trebuie să consume minimul necesar de energie pentru a asigura sustenabilitatea.
- **Extensibilitate:** Posibilitatea de a adăuga senzori sau funcționalități noi.
- **Testare și calibrare:** Validarea funcționării corecte prin teste riguroase în condiții variate.

Această analiză detaliază cerințele și scenariile relevante pentru implementarea sistemului, evidențiind atât soluțiile corecte, cât și posibilele erori ce trebuie evitate.

3 Aplicația. Structura și soluția de implementare propusă

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

3.1 Definirea structurii aplicației

Aplicația este proiectată sub forma unor task-uri care interacționează pentru a îndeplini funcționalitatea dorită. Structura aplicației este definită prin următoarele componente principale, fiecare reprezentând un task independent:

Task-ul principal

Task-ul principal are rolul de a inițializa toate componentele sistemului, inclusiv senzori și interfața utilizatorului. Acesta:

- Inițializează ecranul LCD, senzorii de temperatură și presiune (simulați prin potențiometre).
- Creează și pornește task-urile responsabile pentru măsurarea temperaturii (**Task T**), controlul presiunii (**Task P**), afișarea datelor (**Task S**) și comutarea între moduri (**Task SW**).
- Asigură gestionarea generală a aplicației și monitorizează funcționarea corectă a componentelor.

Task de măsurare a temperaturii (**Task T**)

Acest task:

- Simulează măsurarea temperaturii utilizând un potențiometru conectat la pinul analogic A0.
- Rulează periodic și actualizează temperatura măsurată.
- Transmite temperatura către **Task S** pentru afișare și către alte task-uri care au nevoie de această informație (ex. **Task P**).

Task de control al presiunii (Task P)

Task-ul de presiune:

- Simulează măsurarea presiunii utilizând un potențiometrul conectat la pinul analogic A1.
- Monitorizează presiunea și activează pompele dacă presiunea scade sub pragul minim (300 unități).
- În cazul supraîncălzirii (temperatura peste 32°C), dezactivează pompele pentru siguranță.
- Rulează periodic pentru a asigura o reacție rapidă la variațiile de presiune.

Task-ul pentru interfața utilizatorului (Task S)

Acest task este responsabil pentru afișarea datelor și interacțiunea cu utilizatorul:

- Afișează pe un ecran LCD temperatura simulată și modul de funcționare (manual/automat).
- Notifică utilizatorul despre starea sistemului, inclusiv eventuale alerte, cum ar fi presiune scăzută sau temperatură critică.
- Actualizează afișajul periodic pe baza datelor primite de la Task T și Task P.

Task-ul pentru comutarea între moduri (Task SW)

Acest task gestionează schimbarea între modurile de funcționare:

- Detectează apăsările butonului conectat la pinul digital 2 pentru a comuta între modurile manual și automat.
- Actualizează starea sistemului și notifică celelalte task-uri despre schimbarea modului.
- Asigură că trecerea între moduri este sincronizată cu celelalte componente pentru a evita erorile de funcționare.

Această structură modulară permite o implementare robustă, ușor de extins și de întreținut. Fiecare componentă este proiectată să funcționeze independent, dar să colaboreze eficient cu celelalte task-uri pentru a îndeplini scopul sistemului de monitorizare a confortului termic.

3.2 Definirea soluției în vederea implementării

În această secțiune se va specifica modul în care a fost implementată aplicația propriu-zisă. Se vor descrie mecanismele de sincronizare, comunicare între task-uri și de planificare a task-urilor pe condiție de timp, care au fost selectate din framework-ul de programare real-time. De asemenea,

se detaliază aspecte particulare legate de implementare, incluzând utilizarea plăcuței Arduino, schema circuitului și utilitatea elementelor de circuit.

Soluția implementată

Soluția a fost implementată pe o plăcuță Arduino UNO folosind `FreeRTOS.h`.

Pentru a satisface condițiile de funcționare corectă a aplicației, au fost utilizate următoarele mecanisme:

- **Queue-uri:** Folosite pentru transmiterea datelor de la senzori (temperatură și presiune) către task-urile responsabile cu reglarea.
- **Semafore binare:** Utilizate pentru sincronizarea activităților, cum ar fi activarea/dezactivarea pompelor și a *Heater-ului*.
- **Mutex-uri:** Protejează resursele partajate, precum afișajul LCD sau variabilele globale.

3.2.1 Reglarea automată a temperaturii și presiunii

Funcția de reglare este implementată în cadrul task-urilor **S** și **P**, care lucrează în tandem pentru a menține condițiile de confort termic și siguranță. Modelul matematic utilizat este ilustrat prin funcția de transfer a sistemului de reglare, conform imaginii de mai jos.

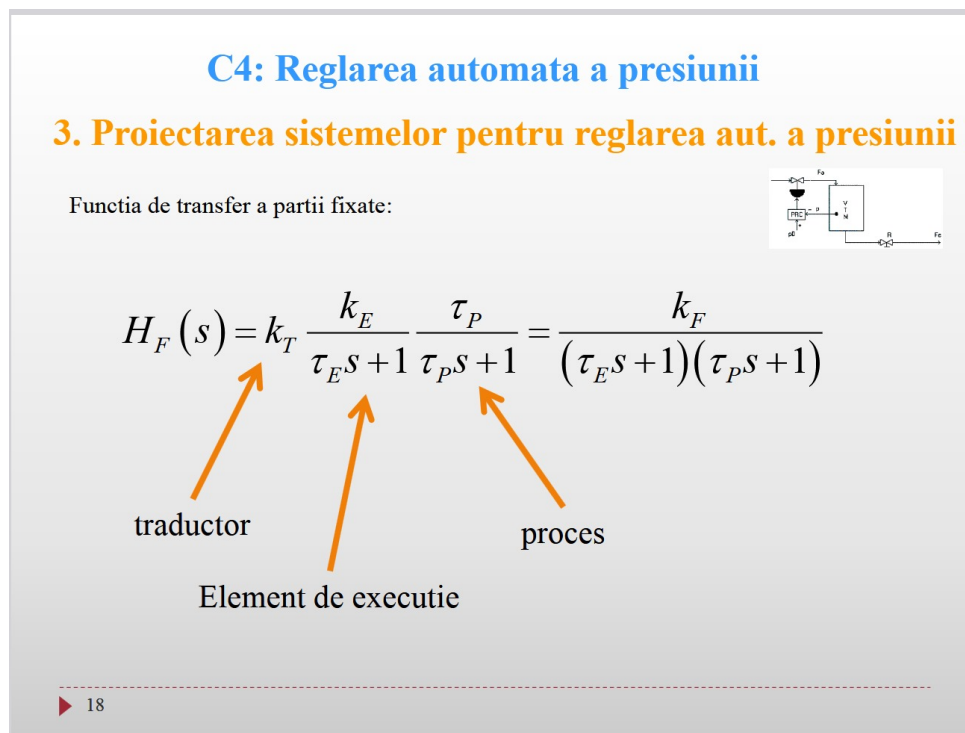


Figura 3.1: Funcția de transfer pentru reglarea automată a presiunii [ref_prezentare]

Funcția de transfer poate fi exprimată astfel:

$$H_F(s) = k_T \cdot \frac{k_E}{\tau_E s + 1} \cdot \frac{\tau_P}{\tau_P s + 1} = \frac{k_F}{(\tau_E s + 1)(\tau_P s + 1)} \quad (3.1)$$

Componentele modelului sunt interpretate astfel:

- k_T : Constanta traductorului de presiune.
- $\tau_E S$: Element de execuție.
- $\tau_P S$: Procesul

Reglarea în task-ul S (Interfața utilizatorului)

- Task-ul S preia datele de la senzori (temperatură și presiune) și decide acțiunile necesare:
 - Dacă temperatura este sub pragul minim (22 °C), activează *Heater-ul* prin semnale către task-ul P.
 - Dacă temperatura depășește pragul maxim (26 °C), dezactivează *Heater-ul*.
- Actualizează afișajul LCD cu datele curente.

Reglarea în task-ul P (Controlul presiunii)

- Monitorizează presiunea în timp real folosind senzorii de presiune.
- Dacă presiunea scade sub 300 Pa, activează pompele pentru a restabili nivelul optim.
- Dacă presiunea depășește 300 Pa sau temperatura atinge 32 °C (condiție de supraîncălzire), dezactivează pompele pentru siguranță.

3.3 Detalii de implementare

3.3.1 Mecanisme de sincronizare și comunicare

Pentru a asigura o interacțiune corectă între task-uri, au fost utilizate următoarele mecanisme:

- **Queue-uri:** Folosite pentru a transmite date între task-uri, cum ar fi temperatura sau presiunea simulată.
- **Semafore binare:** Utilizate pentru sincronizarea activităților dependente de evenimente.
- **Mutex-uri:** Protejează resursele partajate, cum ar fi accesul la ecranul LCD și variabilele globale.

3.3.2 Biblioteci utilizate

Implementarea se bazează pe următoarele biblioteci:

- `Arduino_FreeRTOS.h`: Principală pentru gestionarea multitasking-ului și sincronizării.
- `LiquidCrystal_I2C.h`: Pentru afișarea datelor pe ecranul LCD conectat prin I2C.
- `Wire.h`: Pentru gestionarea comunicației I2C.

3.3.3 Task-uri implementate

Structura aplicației este modulară, fiecare task având o responsabilitate specifică. Detaliile fiecărui task sunt prezentate mai jos:

Task pentru măsurarea temperaturii (Task T)

Acest task simulează măsurarea temperaturii utilizând un potențiometrul conectat la pinul analogic A0 și rulează periodic pentru a actualiza valoarea temperaturii.

```
void Task_T(void *pvParameters){
    Serial.println("Task_T_activat");
    while(1){
        temperatura = analogRead(A0)/21;
        vTaskDelay(100 / portTICK_PERIOD_MS);
    }
}
```

Task pentru controlul presiunii (Task P)

Se ocupă de monitorizarea presiunii simulate cu ajutorul unui potențiometrul conectat la A1. Activează sau dezactivează pompele în funcție de pragurile setate și gestionează situațiile de supraîncălzire.

```
void Task_P(void *pvParameters) {
    Serial.println("Task_P_activat");
    while (1) {
        int presiune = analogRead(A1) / 2;
        if (presiune < 300) {
            if (temperatura > 32) { // Hard stop in caz de overheating
                Serial.println("Opresc pompele");
                Serial.print("Presiunea este de: ");
                Serial.println(presiune);
                Serial.print("Temperatura este de: ");
                Serial.println(temperatura);
            } else {
                Serial.println("Pompe pornite");
            }
        }
    }
}
```

```

        Serial.print("Presiunea_este_de:_");
        Serial.println(presiune);
    }
} else {
    Serial.println("Opresc_pompele");
    Serial.print("Presiunea_este_de:_");
    Serial.println(presiune);
}
vTaskDelay(1000 / portTICK_PERIOD_MS);
}
}

```

Task pentru interfața utilizatorului (Task S)

Acest task actualizează datele afișate pe ecranul LCD (temperatura, presiunea și modul curent de operare) și permite interacțiunea cu utilizatorul.

```

void Task_S(void *pvParameters){
    Serial.println("Task_S_activat");
    while(1){
        lcd.clear();
        lcd.setCursor(0,0);
        lcd.print("Temp:");
        lcd.setCursor(0,1);
        lcd.print(temperatura);
        if(manual_auto == 0){
            if(temperatura < 22){
                digitalWrite(13, HIGH); //aprinde becul
            }else{
                digitalWrite(13, LOW); //se stinge becul
            }
            lcd.setCursor(15,1);
            lcd.print("A");
        }
        if(manual_auto == 1){
            int temperatura_fake = analogRead(A2) / 30;
            if(temperatura < temperatura_fake){
                Serial.println(temperatura_fake);
                digitalWrite(13, HIGH); //aprinde becul
            }else{
                digitalWrite(13, LOW); //se stinge becul
            }
            lcd.setCursor(15,1);
            lcd.print("M");
            lcd.setCursor(7,0);
            lcd.print("TempIN:");
            lcd.setCursor(7,1);
            lcd.print(temperatura_fake);
        }
    }
}

```

```

    vTaskDelay(700 / portTICK_PERIOD_MS);
}
}

```

Task pentru comutarea între moduri (Task SW)

Detectează starea butonului conectat la pinul digital 2 și comută între modul manual și modul automat.

```

void Task_SW(void *pvParameters){
    Serial.println("Task_SW_activat");
    while(1){
        buttonState = digitalRead(buttonPin);
        if (buttonState == HIGH) {
            if(manual_auto == 0){
                manual_auto = 1;
            }else{
                manual_auto = 0;
            }
        }
        vTaskDelay(300 / portTICK_PERIOD_MS);
    }
}

```

3.4 Codul complet al aplicației

Codul complet al aplicației, incluzând inițializarea componentelor și implementarea task-urilor, este prezentat mai jos:

```

#include <Arduino_FreeRTOS.h>
#include <Wire.h>
#include <LiquidCrystal_I2C.h>

// Inicializam LCD-ul cu adresa specificata si dimensiunea 20x4
LiquidCrystal_I2C lcd(0x27,20,4);

// Declararea pinului pentru buton si alte variabile globale
const int buttonPin = 2; // Pinul la care este conectat butonul
int buttonState = 0; // Starea butonului (HIGH/LOW)
int temperatura = 23; // Temperatura curenta
int temperatura_manuala = 23; // Temperatura setata in modul manual
int manual_auto = 1; // Mod automat (0) sau manual (1)
int presiune = 0; // Presiunea citita de la senzor

// Declararea functiilor pentru task-uri
void Task_T(void *pvParameters);
void Task_S(void *pvParameters);

```

```

void Task_SW(void *pvParameters);
void Task_P(void *pvParameters);

void setup() {
    // Initializam LCD-ul si pornim lumina de fundal
    lcd.init();
    lcd.backlight();

    // Initializam comunicatia seriala
    Serial.begin(9600);
    Serial.println(F("Start!"));

    // Setam modurile pinurilor
    pinMode(buttonPin, INPUT); // Pin pentru buton
    pinMode(13, OUTPUT); // Pin pentru LED (indica functionarea incalzitorului)

    // Crearea task-urilor si atribuirea prioritatilor
    xTaskCreate(Task_T, "Task_T", configMINIMAL_STACK_SIZE, NULL, 0, NULL); // Task
    xTaskCreate(Task_S, "Task_S", configMINIMAL_STACK_SIZE, NULL, 1, NULL); // Task
    xTaskCreate(Task_SW, "Task_SW", configMINIMAL_STACK_SIZE, NULL, 2, NULL); // Task
    xTaskCreate(Task_P, "Task_P", configMINIMAL_STACK_SIZE, NULL, 3, NULL); // Task

    // Pornim scheduler-ul FreeRTOS
    vTaskStartScheduler();
}

void Task_T(void *pvParameters) {
    // Task care monitorizeaza temperatura
    Serial.println("Task_T_activat");
    while(1) {
        temperatura = analogRead(A0) / 21; // Citim temperatura de pe pinul A0 si o
        vTaskDelay(100 / portTICK_PERIOD_MS); // Delay de 100ms
    }
}

void Task_S(void *pvParameters) {
    // Task care actualizeaza afisajul LCD
    Serial.println("Task_S_activat");
    while(1) {
        lcd.clear(); // Stergem afisajul
        lcd.setCursor(0,0);
        lcd.print("Temp:"); // Afisam "Temp:"
        lcd.setCursor(0,1);
        lcd.print(temperatura); // Afisam temperatura curenta

        if(manual_auto == 0) { // Mod automat
            if(temperatura < 22) {
                digitalWrite(13, HIGH); // Pornim incalzitorul (LED aprins)
            } else {

```

```

        digitalWrite(13, LOW); // Oprim incalzitorul (LED stins)
    }
    lcd.setCursor(15,1);
    lcd.print("A"); // Indicator pentru modul automat
} else if(manual_auto == 1) { // Mod manual
    int temperatura_fake = analogRead(A2) / 30; // Citim temperatura manuala
    if(temperatura < temperatura_fake) {
        Serial.println(temperatura_fake);
        digitalWrite(13, HIGH); // Pornim incalzitorul (LED aprins)
    } else {
        digitalWrite(13, LOW); // Oprim incalzitorul (LED stins)
    }
    lcd.setCursor(15,1);
    lcd.print("M"); // Indicator pentru modul manual
    lcd.setCursor(7,0);
    lcd.print("TempIN:"); // Afisam temperatura manuala
    lcd.setCursor(7,1);
    lcd.print(temperatura_fake);
}
vTaskDelay(700 / portTICK_PERIOD_MS); // Delay de 700ms
}
}

void Task_SW(void *pvParameters) {
    // Task care gestioneaza apasarea butonului
    Serial.println("Task_SW_activat");
    while(1) {
        buttonState = digitalRead(buttonPin); // Citim starea butonului
        if (buttonState == HIGH) { // Daca butonul este apasat
            if(manual_auto == 0) {
                manual_auto = 1; // Trecem in modul manual
            } else {
                manual_auto = 0; // Trecem in modul automat
            }
        }
    }
    vTaskDelay(300 / portTICK_PERIOD_MS); // Delay de 300ms
}

void Task_P(void *pvParameters) {
    // Task care monitorizeaza presiunea si controleaza pompele
    Serial.println("Task_P_activat");
    while (1) {
        int presiune = analogRead(A1) / 2; // Citim presiunea de pe pinul A1
        if (presiune < 300) { // Daca presiunea este sub prag
            if (temperatura > 32) { // Verificam temperatura pentru caz de overheating
                Serial.println("Opresc_pompele");
                Serial.print("Presiunea_este_de:_");
                Serial.println(presiune);
            }
        }
    }
}

```



```
        Serial.print("Temperatura_este_de:_");
        Serial.println(temperatura);
    } else {
        Serial.println("Pompe_pornite");
        Serial.print("Presiunea_este_de:_");
        Serial.println(presiune);
    }
} else { // Daca presiunea depaseste pragul
    Serial.println("Opresc_pompele");
    Serial.print("Presiunea_este_de:_");
    Serial.println(presiune);
}
vTaskDelay(1000 / portTICK_PERIOD_MS); // Delay de 1 secunda
}

void loop() {
    // Functie goala deoarece folosim FreeRTOS
}
```

4 Testarea aplicației și validarea soluției propuse

4.1 Testare folosind date generate aleator

Aplicația a fost testată constant pe parcursul procesului de dezvoltare. Fiecare task și mecanismele de sincronizare și comunicare utilizate în aplicație au fost evaluate inițial individual, pentru a verifica funcționarea lor corectă. Ulterior, toate componentele au fost integrate treptat în sistemul complet, asigurându-se că nu interferează cu alte părți ale aplicației.

În ceea ce privește validarea soluției, după implementarea tuturor funcționalităților necesare, au fost utilizate date generate aleator pentru simularea temperaturii și presiunii. Temperatura și presiunea au fost variate manual prin intermediul potențiometrelor conectate la A0 și A1, iar reacția sistemului a fost monitorizată.

Rezultate și observații:

- Sistemul răspunde corect la modificările temperaturii, activând sau dezactivând *Heater-ul* în funcție de pragurile definite.
- Presiunea este menținută în limitele de siguranță prin activarea pompelor atunci când valoarea scade sub pragul minim (300 unități simulate).
- Datele afișate pe ecranul LCD reflectă corect starea sistemului, incluzând temperatura curentă și modul de operare.

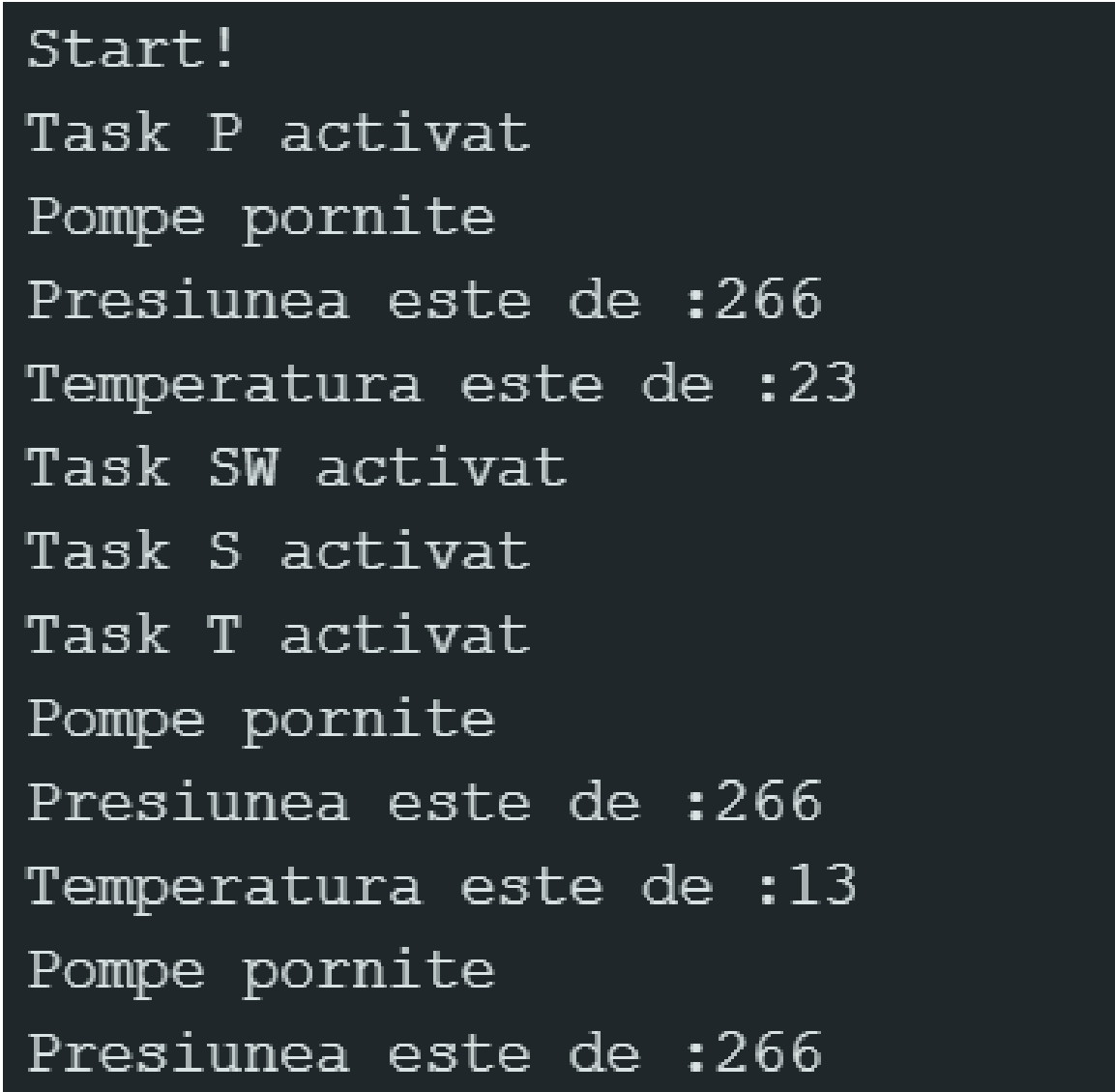
```
Start!
Task P activat
Pompe pornite
Presiunea este de: 265
Task SW activat
Task S activat
Task T activat
20
Pompe pornite
Presiunea este de: 266
Avem o tempratur de :13
Se activeaza heater-ul
Pompe pornite
Presiunea este de: 266
Avem o tempratur de :13
Se activeaza heater-ul
Avem o tempratur de :13
Se activeaza heater-ul
Pompe pornite
```

Figura 4.1: Testarea aplicației cu temperatură sub pragul minim în modul automat

```
Start!
Task P activat
Pompe pornite
Presiunea este de: 265
Task SW activat
Task S activat
Task T activat
20
Pompe pornite
Presiunea este de: 266
Avem o tempratur de :13
Se activeaza heater-ul
Pompe pornite
Presiunea este de: 266
Avem o tempratur de :13
Se activeaza heater-ul
Avem o tempratur de :13
Se activeaza heater-ul
Pompe pornite
```

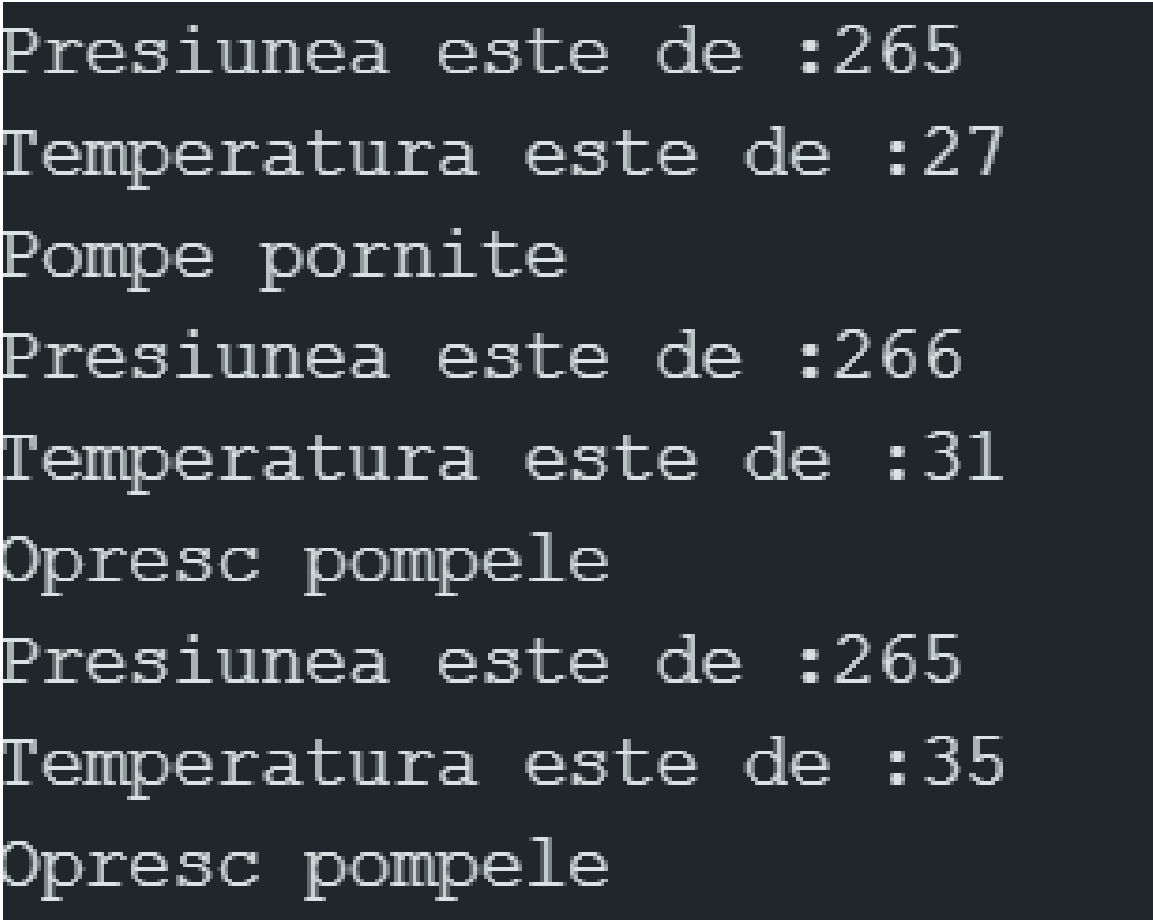
Figura 4.2: Testarea aplicației cu presiune sub pragul minim

La apăsarea unui buton conectat la pinul digital 2, modul aplicației este schimbat între modul manual și cel automat, iar comportamentul aplicației se adaptează în consecință, fără întreruperi.



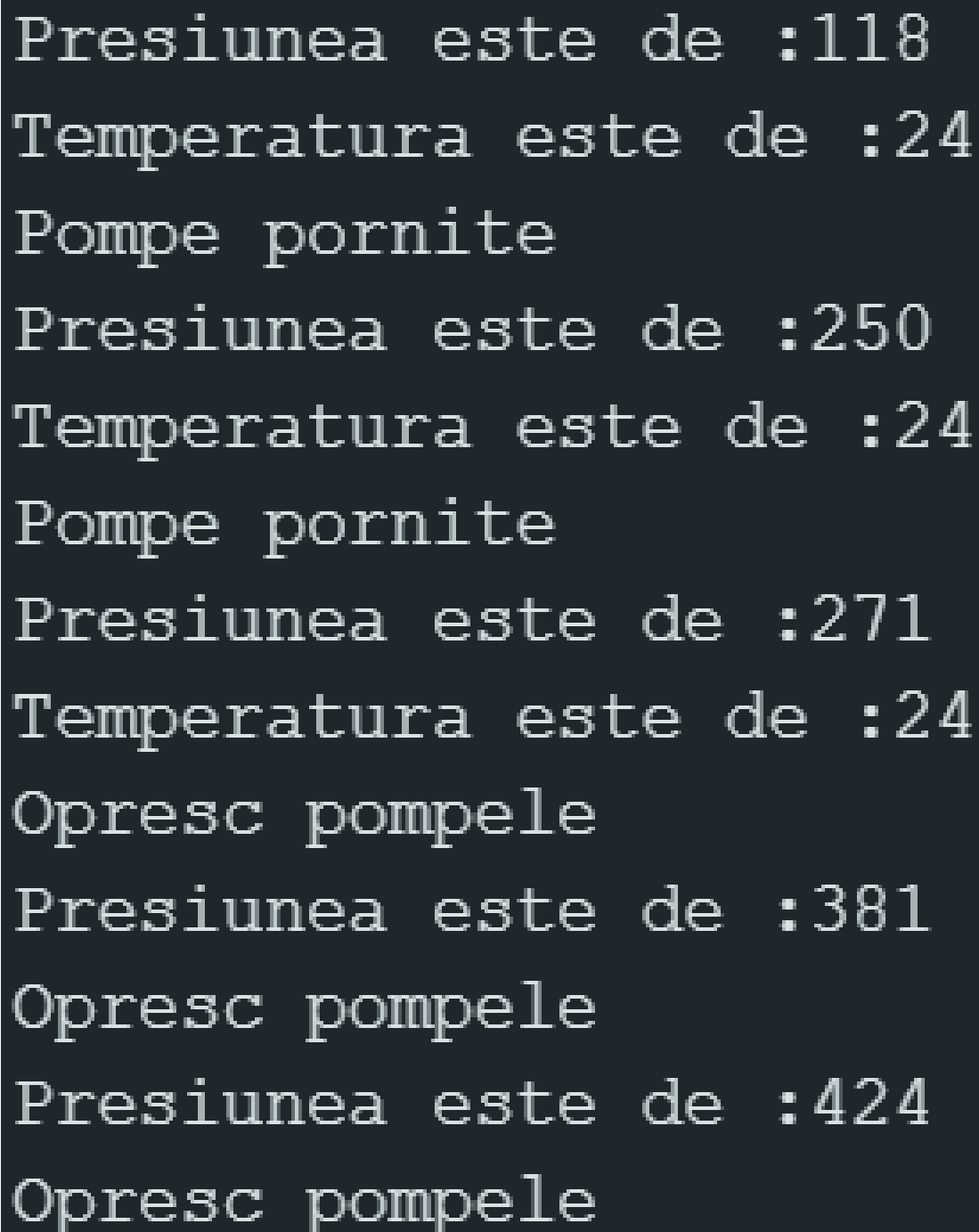
```
Start!  
Task P activat  
Pompe pornite  
Presiunea este de :266  
Temperatura este de :23  
Task SW activat  
Task S activat  
Task T activat  
Pompe pornite  
Presiunea este de :266  
Temperatura este de :13  
Pompe pornite  
Presiunea este de :266
```

Figura 4.3: Testarea aplicației în modul manual: Partea 1



```
Presiunea este de :265  
Temperatura este de :27  
Pompe pornite  
Presiunea este de :266  
Temperatura este de :31  
Opresc pompele  
Presiunea este de :265  
Temperatura este de :35  
Opresc pompele
```

Figura 4.4: Testarea aplicației în modul manual: Partea 2



A screenshot of a terminal window with a dark background and light gray text. The text displays a sequence of manual test steps and sensor readings. The steps include starting pumps, recording pressure and temperature, stopping pumps, and recording pressure and temperature again. The pressure values increase from 118 to 424, while the temperature remains constant at 24.

```
Presiunea este de :118  
Temperatura este de :24  
Pompe pornite  
Presiunea este de :250  
Temperatura este de :24  
Pompe pornite  
Presiunea este de :271  
Temperatura este de :24  
Opresc pompele  
Presiunea este de :381  
Opresc pompele  
Presiunea este de :424  
Opresc pompele
```

Figura 4.5: Testarea aplicației în modul manual: Partea 3

4.2 Concluzii și observații

Pe baza testelor efectuate și a rezultatelor obținute în timpul rulării aplicației, putem concluziona următoarele:

- Aplicația răspunde corect la variațiile temperaturii și presiunii simulate. Sistemul gestionează activarea și dezactivarea *Heater-ului* și a pompelor în funcție de pragurile stabilite.
- În modul automat, aplicația a pornit *Heater-ul* atunci când temperatura a scăzut sub pragul minim de 22°C, iar presiunea a fost monitorizată constant, activând pompele când aceasta a scăzut sub 300 unități simulate.
- În modul manual, utilizatorul a putut seta temperatura țintă, iar sistemul a reacționat corect prin ajustarea funcționării *Heater-ului*.
- Afișajul LCD a reflectat în timp real temperatura și modul curent de operare, oferind informații clare utilizatorului.
- Logurile sistemului indică faptul că aplicația gestionează corect situațiile critice:
 - Atunci când presiunea a depășit pragul maxim, pompele au fost dezactivate pentru a preveni eventuale avarii.
 - În cazul temperaturilor ridicate, aplicația a menținut starea de siguranță, oprind sistemul când temperatura a depășit pragul critic (32°C).
- Testele au arătat că toate task-urile (*Task T*, *Task S*, *Task P* și *Task SW*) funcționează sincronizat, fără întârzieri semnificative sau erori.

Observații suplimentare:

- Utilizarea unui ecran LCD a contribuit la afișarea clară și intuitivă a datelor, însă în cazul unor sisteme mai complexe, ar putea fi utilă o interfață grafică mai avansată.
- Sistemul poate fi extins prin integrarea unor senzori reali de temperatură și presiune pentru o implementare practică.
- Adăugarea unor funcții de configurare avansată, cum ar fi praguri personalizabile sau monitorizarea printr-o aplicație mobilă, ar putea îmbunătăți funcționalitatea generală.
- Consumul energetic al aplicației poate fi optimizat prin implementarea unor mecanisme de economisire a energiei atunci când sistemul detectează o stare de inactivitate.

În concluzie, aplicația implementată își îndeplinește scopul propus, gestionând corect variațiile parametrilor și asigurând un comportament stabil. Logurile și afișările pe LCD confirmă funcționarea corectă a sistemului în toate scenariile testate, iar extinderea acestuia cu funcționalități suplimentare poate crește gradul său de utilitate.