Sem. I 2025-2026

Proiect PATR

[Sistem de monitorizare a confortului termic intr-o cladire]

Echipa: ArduinoSRL

Data: 10 ianuarie 2026

Componența echipei și contribuția individuală (în %)

Membrii	A	C	D	PR	CB	e-mail
Ismana Teodor-Ioan	33.3%	33.3%	33.3%	33.3%	33.3%	teodor_ioan.ismana@stud.acs.upb.ro
Ozdemir Ali-Mert	33.3%	33.3%	33.3%	33.3%	33.3%	ali_mert.ozdemir@stud.acs.upb.ro
Traistaru Alexandru-Mihai	33.3%	33.3%	33.3%	33.3%	33.3%	alexandru.traistaru@stud.acs.ub.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

Proiectul presupune implementarea unui sistem de monitorizare a confortului termic intr-o cladire. Scopul temei este asigurarea unui nivel de confort termic stabil (mentinerea temperaturii in jurul unei valori setate) si a functionarii in siguranta a instalatiei (de ex. mentinerea presiunii sub un anumit prag).

Componentele principale pentru organizarea sistemului sunt urmatoarele:

- **SW (selector mod si nivel putere):** Reprezinta un switch controlat digital. Acesta dicteaza regimul de functionare al sistemului:
  - In modul manual, permite setarea unui nivel de putere fix pentru Heater.
  - In modul automat, S este cel care controleaza incalzirea.
- **T (achizitie temperatura de la termocupluri):** Acest bloc preia periodic masuratorile de la un set de termo-cupluri (senzori de temperatura, TC1...TCn) si le transmite modulului central S pentru procesare.
- **P (monitorizare si control presiune in instalatie):** Acest bloc este responsabil de gestiunea presiunii din instalatie. Preia date de la senzorii de presiune si primeste informatii de la modulele S si/sau SW despre puterea Heater-ului, care poate influenta presiunea. Pe baza acestor date, P controleaza pompele/valvele.
- **S (calcul si afisare confort, respectiv generare comanda catre incalzire):** Acesta este nucleul aplicatiei, cu responsabilitati diferite in functie de modul de operare:
  - Daca SW este setat pe modul **automat:** Modulul S calculeaza o comanda (de exemplu, nivelul de putere pentru Heater) pe baza datelor primite de la T. De asemenea, afiseaza pe ecran nivelul de confort termic (de ex: temperatura medie) si presiunea din instalatie (primita de la P).
  - Daca SW este setat pe modul **manual:** Modulul S nu mai calculeaza comenzi pentru Heater. Functia se reduce la a afisa pe ecran starea curenta a sistemului (temperatura de la T, presiunea de la P).

Conditii si constrangeri de implementare:

- **Respectarea modului de operare:** In modul **manual** valoarea puterii heater ului este stabilita de SW si nu trebuie suprascrisa de S. In modul **automat**, S are control asupra puterii heater ului.

- **Achizitie temperatura de la T:** Modulul T trebuie sa furnizeze date actualizate periodic, iar S trebuie sa proceseze intotdeauna cel mai recent set de valori valide disponibile.
- **Marimi valide:** Toate marimile fizice simulate (temperatura, presiune, procent incarcare heater) sunt limitate la domenii valide (ex. 0-100% pentru putere), valorile din afara intervalelor fiind tratate ca erori sau saturate.

2 Analiza problemei

In aceasta sectiune sunt detaliati agentii care compun sistemul, responsabilitatile acestora, precum si modul de executie. Se vor analiza cateva secvente de functionare, atat corecte cat si gresite.

Sistemul functioneaza pe baza schimbului de date intre SW, T, S si P. Modulul T produce periodic masuratori de temperatura (TC1...TCn) care sunt consumate de modulul S. Modulul S calculeaza indicatori (de exemplu temperatura medie) si, in regim automat, produce comanda de incalzire. Modulul P monitorizeaza presiunea si comanda pompa/valva in functie de presiune si de nivelul de incalzire utilizat. Modulul SW selecteaza regimul de operare si, in regim manual, stabileste direct comanda elementului de incalzire.

Din punct de vedere al implementarii concurente, fiecare modul poate fi tratat ca o entitate de executie separata (task/thread) cu rol de producator/consumator de date.

Sistemul are doua moduri distincte:

- **Manual**: puterea elementului de incalzire este stabilita de SW si ramane fixa pana la o noua comanda de la SW. S nu are voie sa o suprascrie. Rolul lui S devine preponderent informativ (afisare temperaturi si presiune).
- **Automat**: S calculeaza puterea pe baza temperaturilor primite de la T (de exemplu, comparand temperatura medie cu o valoare de referinta). P foloseste puterea calculata pentru a anticipa/evalua impactul asupra presiunii si mentine presiunea in limite sigure.

Trecerea intre moduri se realizeaza prin SW si trebuie sa fie vizibila imediat pentru S (si implicit pentru P). La comutare, sistemul trebuie sa evite starile tranzitorii (de exemplu, S sa calculeze automat in timp ce SW tocmai a trecut pe manual).

2.1 Secvente corecte de functionare

Secventa 1: regim automat, control temperatura si presiune

1. SW este setat pe automat.
2. T achizitioneaza temperaturile TC1...TCn si transmite catre S.
3. S calculeaza temperatura medie si confortul termic.
4. S calculeaza puterea elementului de incalzire astfel incat temperatura sa se apropie de valoarea de referinta si transmite comanda catre P.
5. P citeste presiunea si decide actiunea (pompa/valva) tinand cont de puterea data de S, mentinand

presiunea sub o valoare de referinta.

6. S afiseaza temperatura, confortul si presiunea curenta.

Secventa 2: comutare pe manual, mentinere putere element de incalzire fixa

1. Sistemul ruleaza in automat; la un moment dat, SW comuta pe manual.
2. S detecteaza schimbarea modului si opreste calculul automat al comenzii; ramane doar cu rol de afisare.
3. P foloseste puterea data manual pentru controlul presiunii, mentinand presiunea in limite.
4. T continua achizitia periodica a temperaturilor; S continua afisarea temperaturilor si presiunii.

2.2 Secvente gresite de functionare

Secventa 1: incalcarea modului manual

1. SW comuta pe manual.
2. S continua sa calculeze puterea in automat si transmite catre P o valoare diferita.
3. Deci P poate primi comenzi contradictorii si se poate comporta neasteptat.

Secventa 2: date incorecte intre S si P

1. S actualizeaza puterea in acelasi timp in care P citeste puterea data de S.
2. In lipsa unei coordonari, P poate folosi o valoare veche.
3. Deci P poate calcula gresit si riscam depasirea pragului de siguranta al presiunii.

3 Aplicația. Structura și soluția de implementare propusă

3.1 Definirea structurii aplicației

Aplicatia este modelata ca un sistem concurent format din mai multe entitati de executie, fiecare corespunzand unui bloc functional: SW, T, S si P. Comunicarea intre un task si altul se realizeaza prin mecanisme de sincronizare (cozi de mesaje, lock-uri), eliminand dependenta de secventialitate.

3.1.1 Task de tip main

Initializeaza parametrii (de ex. T_ref, P_max, perioadele de esantionare), porneste task-urile SW/-T/S/P.

3.1.2 Task de tip SW (interfata)

Acest task gestioneaza interactiunea cu utilizatorul. Genereaza evenimente de schimbare a modului (manual/automat) si, in modul manual, stabileste un nivel de putere fix pentru elementul de incalzire. Evenimentele SW trebuie propagate imediat catre S (si implicit catre P), pentru a evita situatii tranzitorii in care S continua calculul automat dupa comutare.

3.1.3 Task de tip T (achizitie date)

Acest task implementeaza rolul de "producator" al sistemului, fiind ocupat cu achizitia datelor referitoare la temperatura. Functioneaza periodic si produce esantioane de temperatura (TC1...TCn). Fiecare esantion contine valorile curente si informatii referitore la timp (cand a fost masurata temperatura). Esantionul este transmis catre S. T nu decide nimic, ci doar furnizeaza date.

3.1.4 Task de tip S (decizie)

Reprezinta unitatea logica de decizie si functioneaza ca un "consumator" al datelor de la T.

- In modul automat, consuma esantioanele de temperatura de la T, calculeaza temperatura medie si confortul termic, apoi calculeaza comanda de incalzire si o transmite catre P.
- In modul manual, nu mai calculeaza comanda de incalzire, ci foloseste comanda setata de SW si afiseaza starea sistemului (temperaturi, presiune).

S are rol si de prezentare (afisare periodica a starii), astfel incat evolutia sistemului sa fie observabila in testare.

3.1.5 Task de tip P (siguranta/control)

Este un task critic care ruleaza independent de logica termica pentru a asigura siguranta instalatiei. Functioneaza periodic si citeste presiunea. In functie de presiune si de puterea elementului de incalzire (primit de la S sau SW), P decide actiunea asupra instalatiei (pompa/valva) astfel incat presiunea sa ramana sub un prag sigur (P_max). P transmite periodic presiunea curenta catre S pentru afisare.

3.2 Definirea soluției în vederea implementării

In aceasta sectiune definim mecanismele prin care structura concurenta descrisa anterior este realizata in implementare. Implementarea este realizata in Python + threading.

Pentru schimbul de date intre module se utilizeaza cozi de mesaje, astfel:

- Coada pentru esantioane de temperatura (T → S). T produce periodic un mesaj ce contine {timestamp (cand s a masurat), TC1...TCn}. Task-ul S prelucreaza datele pe masura ce devin disponibile.
- Coada pentru comanda de incalzire (S → P), utilizata in modul automat. Mesajul contine {timestamp, `putere` (puterea elementului de incalzire)}.
- Coada pentru presiune (P → S). Permite modulului P sa raporteze presiunea curenta catre S in vederea afisarii. Mesajul contine {timestamp, presiune}.
- Canal pentru semnalizarea schimbarilor de stare generate de utilizator (SW → S/P). Mesajele includ schimbarea modului si, in manual, valoarea setata pentru `putere`.

Avantajul cozilor este ca thread-urile pot astepta eficient aparitia datelor (thread-urile consumator intra in stare de asteptare blocanta), fara a consuma CPU in bucle de asteptare.

Pentru mentinerea consistentei datelor (variabile accesate frecvent din mai multe thread-uri, precum `putere_curenta`), se utilizeaza o zona de memorie partajata protejata printr-un mecanism de excludere mutuala (Mutex).

- Orice modificare a modului de functionare (ex. trecerea din automat in manual si setarea puterii) se executa sub protectia acelui lock, prevenind starile intermediare invalide.
- S si P nu trebuie sa citeasca valori partial actualizate; citirile relevante se fac sub acelasi lock sau prin mesaje (preferabil prin mesaje).

Pentru oprire controlata a sistemului si pentru evitarea blocarii la final, se utilizeaza un mecanism de semnalizare: **Event_stop**: setat de main cand aplicatia trebuie oprita. Toate task-urile verifica periodic starea acestui eveniment.

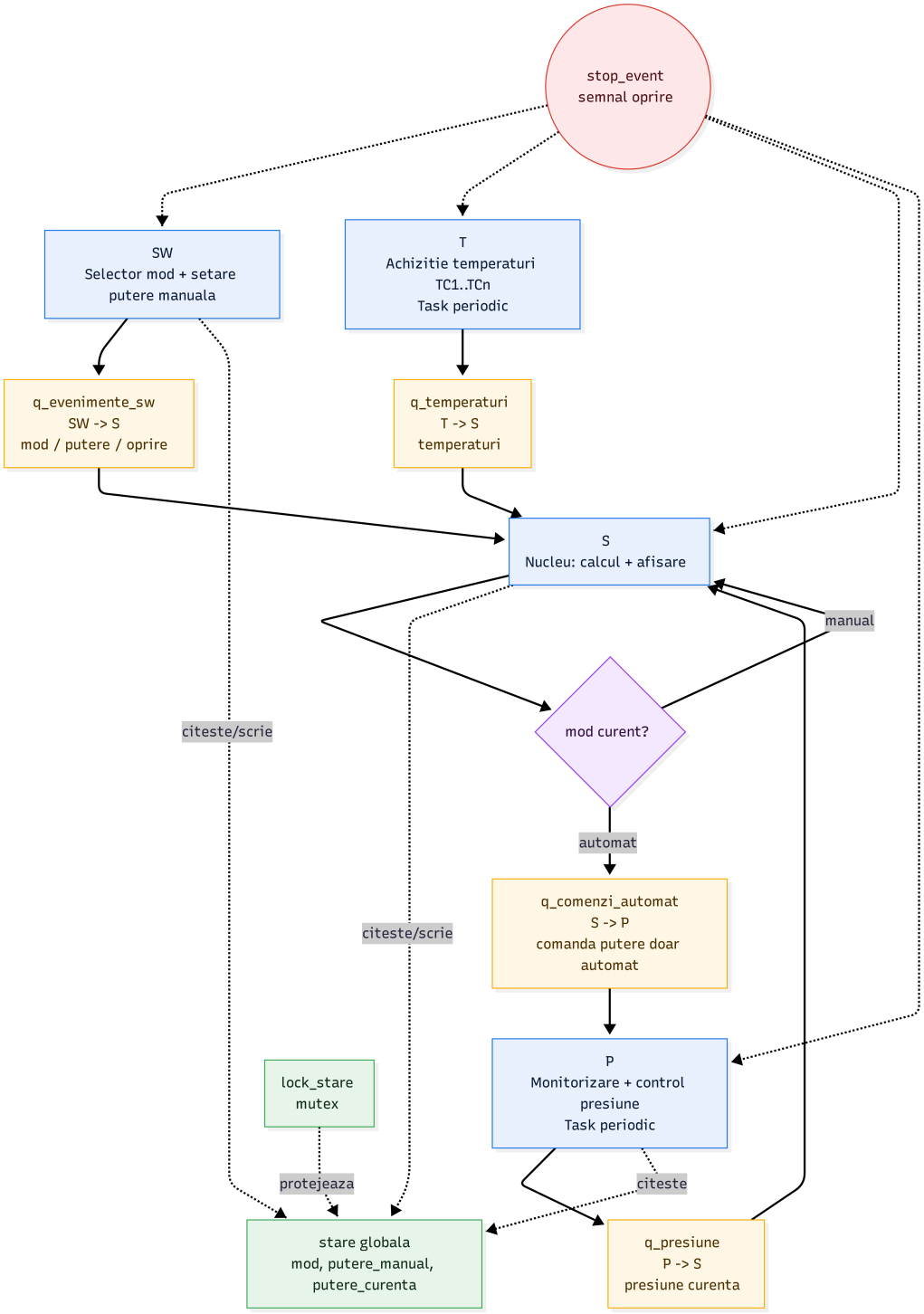
Spre deosebire de o variabila booleana simpla, Event_stop permite intreruperea imediata a asteptarilor

lungi prin notificarea tuturor thread-urilor blocate. Thread-urile asteapta pe cozi (blocant) sau pe evenimente (wait cu timeout). Nu se foloseste delay/busy-wait pentru sincronizare.

Comutarea manual/automat este tratata ca un eveniment cu prioritate logica:

- La comutare, task-ul S actualizeaza variabila `mode` sub lock.
- Daca sistemul trece in manual, S inceteaza trimiterea comenzilor pe coada S-P. Task-ul P va detecta schimbarea de mod si va ignora vechile comenzi ramase in coada, utilizand in schimb valoarea putere_manual setata de SW.
- In modul automat, `putere` este produs exclusiv de S pe baza temperaturilor si P incepe din nou sa preia din coada S-P.

Aceasta regula elimina situatia in care P primeste comenzi contradictorii si asigura separarea clara a sursei comenzii in functie de mod.



In aceasta diagrama se poate observa modul de functionare al aplicatiei.

3.3 Implementarea soluției

```
1
2  # a - mod automat
3  # m - mod manual
4  # p <0..100> - seteaza puterea in mod manual
5  # q - oprim . stop_event se seteaza si task ul se termina
6
7  # T (achizitie temperatura) produce periodic TC1...TCn si trimite catre S
8  # S (decizie) calculeaza confortul si:
9  #       in modul automat, calculeaza puterea si o transmite catre P
10 #       in modul manual, nu calculeaza puterea; foloseste puterea data de SW
       (si doar afiseaza)
11 # P mentine presiunea in limite, folosind puterea curenta
12 # SW (interfata) selecteaza modul si (in manual) stabileste puterea
13 # queue - cozi pentru comunicare intre thread-uri (mesaje)
14 # random - punem un zgomot mic in temperatura/presiune ca sa fie mai realist
15 # threading -thread-uri + Lock (mutex) + Event (semnal de oprire)
16 # cand un ask ia lock ul doar el poate citi/scrie in zona partajata.
       celelalte thread-uri asteapta, se previne race condition
17 # stop_event - semnal de oprire pentru toate thread-urile. E ca un steag,
       cat timp nu e setat, thread-urile ruleaza, dupa ce e setat se opresc
18 # time - time.monotonic(), este un ceas care nu se da inapoi (nu e afectat
       de schimbari de ora sistem), pentru perioade stabile de timp (ex: fac
       ceva la fiecare 0.5 secunde)
19 import queue
20 import random
21 import threading
22 import time
23
24 def limiteaza(valoare, minim, maxim):
25     # intoarce valoarea in intervalul [minim, maxim]
26     # ex: calculez puterea si da 110%, o limitez la 100%
27     # ex: utilizatorul incearca sa seteze -20%, o limitez la 0%
28     # ex: puterea trebuie sa fie intre 0 si 100.
29     if valoare < minim:
30         return minim
31     if valoare > maxim:
32         return maxim
33     return valoare
34
35 def ultimul_mesaj(coada, mesaj):
36     # S trebuie sa lucreze pe date recente, nu pe o lista de date vechi care
           s-au acumulat.
37     # coada are maxsize=1
38     # daca e plina, aruncam mesajul vechi
39     # punem mesajul nou
40     try:
41         if coada.full():
42             coada.get_nowait() # daca coada e plina, scoatem vechiul mesaj
43     except queue.Empty: # daca coada s-a golit intre timp, un alt thread a
           scos mesajul
44         pass
45
46     try:
47         coada.put_nowait(mesaj) # incercam sa punem mesajul nou
```

```
48      except queue.Full:
49          # in acel moment coada s-a umplut din nou, renuntam (alt thread a
                pus ceva intre timp)
50          pass
51
52  def goleste_coada(coada):
53      # Scoate toate elementele ramase in coada, fara sa blocheze thread-ul
54      # Folositor la comutare de mod: vrem sa aruncam comenzi vechi
55      while True:
56          try:
57              coada.get_nowait()
58          except queue.Empty:
59              break
60
61
62  def asteapta_pana_la_urmatoarea_activare(next_release, stop_event):
63      # temporizarea corecta pentru un task periodic
64      # thread-ul intra in asteptare si nu consuma CPU inutil
65      # se trezeste fie la timeout (cand a venit timpul urmator), fie daca
            stop_event se seteaza
66      acum = time.monotonic()
67      timp_ramas = next_release - acum
68      if timp_ramas > 0:
69          stop_event.wait(timeout=timp_ramas)
70
71
72  def calcul_confort(t_medie, t_ref, banda):
73      # Decide daca este "rece", "confortabil" sau "cald" fata de temperatura
            de referinta
74      if t_medie < (t_ref - banda):
75          return "rece"
76      if t_medie > (t_ref + banda):
77          return "cald"
78      return "confortabil"
79
80
81  def calcul_putere_mod_automat(t_medie, t_ref):
82      # Daca temperatura medie este < decat temperatura de referinta - vrem
            putere mai mare
83      # Daca temperatura medie este > decat temperatura de referinta - vrem
            putere mai mica
84      # k = cat de rapid reactioneaza controlul
85      k = 12.0
86      putere_baza = 30.0  # puterea de baza (cu ea pornim, ajustam dupa)
87      eroare = t_ref - t_medie  # daca eroare > 0 - e prea rece
88      putere = k * eroare + putere_baza
89      return putere
90
91  def task_sw(q_evenimente_sw, stop_event, lock_consola):
92      # task SW citeste de la tastatura si trimite "evenimente" catre S.
93      # interfata cu utilizatorul
94      # input() este blocant, dar nu e busy-wait (nu consuma CPU in bucla)
95      # blocant adica programul se opreste aici pana se intampla ceva (
            utilizatorul apasa Enter in cazul nostru)
96      # nu tinem lock-ul pe durata input() fiindca altfel S nu mai poate afisa
            .
```

```
97      # lock_consola il folosim doar ca sa nu se amestece print-urile intre
            ele (S si SW pot afisa in acelasi timp si se amesteca liniile)
98
99      with lock_consola:
100         print("\n[SW] Introdu comenzi: a / m / p <0..100> / q\n")
101
102     while not stop_event.is_set(): #atata timp cat nu e setat semnalul de
            oprire
103         try:
104             linie = input("[SW] > ").strip()
105         except (EOFError, KeyboardInterrupt):
106             linie = "q"
107
108         if not linie:
109             continue
110
111         # Comanda "a" - automat
112         if linie.lower() == "a":
113             ultimul_mesaj(q_evenimente_sw, {"tip": "set_mod", "mod": "
                    automat"})
114             continue
115
116         # Comanda "m" - manual
117         if linie.lower() == "m":
118             ultimul_mesaj(q_evenimente_sw, {"tip": "set_mod", "mod": "manual
                    "})
119             continue
120
121         # Comanda "p ..." - setare putere manuala
122         if linie.lower().startswith("p"):
123             parti = linie.split()
124             if len(parti) != 2:
125                 with lock_consola:
126                     print("[SW] Format corect: p <0..100>")
127                 continue
128
129             try:
130                 putere = float(parti[1])
131             except ValueError:
132                 with lock_consola:
133                     print("[SW] Valoare invalida. Exemplu: p 80")
134                 continue
135
136             putere = limiteaza(putere, 0.0, 100.0)
137             ultimul_mesaj(q_evenimente_sw, {"tip": "set_putere_manual", "
                    putere": putere})
138             continue
139
140         # Comanda "q" - oprire
141         if linie.lower() == "q":
142             ultimul_mesaj(q_evenimente_sw, {"tip": "oprire"})
143             stop_event.set()
144             break
145
146         with lock_consola:
```

```
147             print("[SW] Comanda necunoscuta. Foloseste: a / m / p <0..100> / q")
148
149 def task_t(configurare, stare, lock_stare, q_temperaturi, stop_event):
150     # task T este un task periodic, la fiecare perioada_T secunde genereaza TC1...TCn si trimite rezultatul catre S prin q_temperaturi
151     # parametrii sunt stabiliti in "configurare" in main(), pentru usurinta si testare
152     # ambient - temperatura din cladire daca nu ar exista incalzirea
153     # delta_max - cat se poate urca peste ambient la putere de 100%
154     # alpha - cat de repede se apropie temperatura de baza de tinta
155     # temperatura_tinta = ambient + delta_max*(putere/100)
156     # temperatura_baza se apropie gradual de temperatura_tinta
157
158     ambient = configurare["temperatura_ambient"]         # avem 18 greade C fara incalzire
159     delta_max = configurare["delta_max_incalzire"]       # +10C la 100%
160     alpha = configurare["viteza_raspuns_temperatura"]    # 0.08 (mai mare = mai rapid)
161
162     temperatura_baza = ambient
163
164     next_release = time.monotonic()
165
166     while not stop_event.is_set():
167         next_release += configurare["perioada_T"]
168
169         # Citim puterea curenta sub mutex, ca sa fie consistenta
170         with lock_stare:
171             putere_curenta = stare["putere_curenta"]
172
173         temperatura_tinta = ambient + delta_max * (putere_curenta / 100.0)
174         temperatura_baza = temperatura_baza + alpha * (temperatura_tinta - temperatura_baza)
175
176         temperaturi = []
177         for _ in range(configurare["numar_TC"]):
178             zgomot = random.uniform(-0.15, 0.15)
179             # adaugam zgomot la fiecare termocuplu, senzorii reali nu sunt perfect identici
180             # temperatura finala = temperatura_baza + zgomot
181             # alegem un numar aleator intre -0.15 si +0.15, uniform adica toate valorile din interval au sanse egale
182             temperaturi.append(temperatura_baza + zgomot)
183
184         # mesajul catre S: dict (dictionar) cu timp + lista temperaturi
185         mesaj = {"timestamp": time.monotonic(), "temperaturi": temperaturi}
186
187         # pastram doar ultimul mesaj (latest only)
188         ultimul_mesaj(q_temperaturi, mesaj)
189
190         # asteptare periodica fara busy-wait
191         asteapta_pana_la_urmatoarea_activare(next_release, stop_event)
192
193 def task_p(configurare, stare, lock_stare, q_comenzi_automat, q_presiune, stop_event):
```

```
194     # ruleaza periodic (perioada_P)
195     # citeste presiunea si decide actiunea asupra valvei
196     # trebuie sa foloseasca puterea corecta in functie de mod:
197     #     in modul automat, puterea vine de la S prin q_comenzi_automat
198     #     in modul manual, puterea vine direct din stare (setata de SW)
199 
200     # previne o secventa gresita de funcionare:
201     # daca SW trece pe manual, in coada pot ramane comenzi automate vechi, P
            trebuie sa ignore acele comenzi vechi:
202     # cand detectam mod="manual", golim coada q_comenzi_automat
203 
204     presiune = configurare["presiune_referinta"]
205     actiune_valva = 0.0
206 
207     mod_anterioar = None  # ca sa detectam schimbare de mod
208     next_release = time.monotonic()
209 
210     while not stop_event.is_set():
211         next_release += configurare["perioada_P"]
212 
213         # Citim modul si puterea curenta din zona partajata, folosita de mai
                multe task uri (sub mutex)
214         with lock_stare:
215             mod_curent = stare["mod"]
216             putere_curenta = stare["putere_curenta"]
217 
218         # Daca tocmai am intrat in manual, aruncam comenzile automate ramase
219         if mod_curent == "manual" and mod_anterioar != "manual":
220             goleste_coada(q_comenzi_automat)
221 
222         mod_anterioar = mod_curent
223 
224         # Daca suntem in modul automat, incercam sa luam ultima comanda
                automata de la S
225         # get_nowait() = neblocant: daca nu exista mesaj, arunca exceptie
                imediat, nu asteapta deloc
226         # get(timeout=...) = blocant: asteapta pana la timeout sa apara un
                mesaj
227         # Pentru P, vrem sa nu ne blocam mult; P trebuie sa ruleze periodic.
228         if mod_curent == "automat":
229             ultima_comanda = None
230             try:
231                 # luam ce e disponibil acum, fara asteptare
232                 ultima_comanda = q_comenzi_automat.get_nowait()
233                 # folosim get_nowait() ca sa nu blocam P daca nu e nimic in
                        coada
234                 # daca sunt mai multe, o pastram pe ultima (cea mai noua)
235                 while True:
236                     try:
237                         ultima_comanda = q_comenzi_automat.get_nowait()
238                     except queue.Empty:
239                         break
240             except queue.Empty:
241                 ultima_comanda = None
242 
243             if ultima_comanda is not None:
```

```
                # comanda automata e un dict: {"timestamp":..., "putere":...}
                putere_curenta = ultima_comanda["putere"]

        # Crestere presiune daca puterea e mare + diminuare spre referinta
        crestere = 0.08* (putere_curenta / 100.0)
        diminuare = 0.01 * (configurare["presiune_referinta"] - presiune)
        presiune = presiune + crestere + diminuare

        # Decidem actiunea valvei in functie de presiune
        if presiune > configurare["presiune_maxima_siguranta"]:
            actiune_valva = 1.0
        elif presiune > configurare["presiune_referinta"] + 0.3:
            actiune_valva = 0.6
        else:
            actiune_valva = 0.0

        presiune -= 0.08 * actiune_valva
        presiune += random.uniform(-0.01, 0.01)
        # adaugam un zgomot la fiecare presiune

        # Trimitem presiunea catre S pentru afisare
        ultimul_mesaj(
    q_presiune,
    {"timestamp": time.monotonic(), "presiune": presiune, "valva": actiune_valva}
)

        # asteptare periodica fara busy-wait
        asteapta_pana_la_urmatoarea_activare(next_release, stop_event)

def task_s(configurare, stare, lock_stare, q_evenimente_sw, q_temperaturi, q_comenzi_automat, q_presiune, stop_event, lock_consola):
    # proceseaza evenimentele SW (manual/automat, setare putere manuala)
    # citeste temperatura cea mai recenta de la T
    # citeste presiunea cea mai recenta de la P
    # in modul automat: calculeaza puterea si o transmite catre P
    # in modul manual: nu calculeaza puterea (nu suprascrie), doar afiseaza starea

    # previne o secventa gresita de functionare:
    # daca SW e pe manual, S nu are voie sa calculeze/trimita comenzi de putere. In modul manual, S nu pune nimic in q_comenzi_automat

    # previne race condition: orice update la stare["mod"], stare["putere_*"] se face sub lock_stare

    ultima_temperatura = None
    ultima_presiune = None

    next_afisare = time.monotonic()

    while not stop_event.is_set():
        # procam toate evenimentele SW disponibile acum (neblocant)
        while True:
            try:
```

```
294                 ev = q_evenimente_sw.get_nowait()
295             except queue.Empty:
296                 break
297 
298             tip = ev.get("tip")
299 
300             if tip == "oprire":
301                 stop_event.set()
302                 break
303 
304             if tip == "set_mod":
305                 mod_nou = ev.get("mod")
306                 if mod_nou in ("manual", "automat"):
307                     with lock_stare:
308                         stare["mod"] = mod_nou
309 
310                         # Daca am trecut in manual, punerea puterii curente devine exclusiva SW
311                         if mod_nou == "manual":
312                             stare["putere_curenta"] = stare["putere_manual"]
313 
314             if tip == "set_putere_manual":
315                 putere = ev.get("putere")
316                 if putere is not None:
317                     putere = limiteaza(float(putere), 0.0, 100.0)
318                     with lock_stare:
319                         stare["putere_manual"] = putere
320                         # daca suntem in manual, aplicam imediat puterea
321                         if stare["mod"] == "manual":
322                             stare["putere_curenta"] = stare["putere_manual"]
323 
324         if stop_event.is_set():
325             break
326 
327         # citim temperatura cea mai recenta de la T
328         # q_temperaturi.get(timeout=0.1) inseamna: "astept maxim 0.1 sec sa apara un mesaj; daca nu apare, merg mai departe"
329         # asta reduce consumul CPU
330         try:
331             msg = q_temperaturi.get(timeout=0.1)
332             ultima_temperatura = msg
333 
334             # Daca au venit mai multe, pastram ultimul (cel mai recent)
335             while True:
336                 try:
337                     ultima_temperatura = q_temperaturi.get_nowait()
338                 except queue.Empty:
339                     break
340         except queue.Empty:
341             pass
342 
343         # citim presiunea cea mai recenta de la P
344         try:
345             ultima_presiune = q_presiune.get_nowait()
346             while True:
347                 try:
```

```
                ultima_presiune = q_presiune.get_nowait()
            except queue.Empty:
                break
    except queue.Empty:
        pass

    # calulam temperatura medie si confortul
    if ultima_temperatura is not None:
        temperaturi = ultima_temperatura["temperaturi"]
        t_medie = sum(temperaturi) / len(temperaturi)
        confort = calcul_confort(t_medie, configurare["
            temperatura_referinta"], configurare["banda_confort"])
    else:
        t_medie = float("nan")
        confort = "necunoscut"

    # stabilim puterea curenta in functie de modul de functionare
    with lock_stare:
        mod_curent = stare["mod"]
        putere_manual = stare["putere_manual"]

    if mod_curent == "automat" and ultima_temperatura is not None:
        # mod automat: S calculeaza puterea
        putere_calc = calcul_putere_mod_automat(t_medie, configurare["
            temperatura_referinta"])
        putere_calc = limiteaza(putere_calc, 0.0, 100.0)

        # Update stare sub mutex
        with lock_stare:
            stare["putere_curenta"] = putere_calc

        # Trimitem comanda automata catre P
        ultimul_mesaj(q_comenzi_automat, {"timestamp": time.monotonic(),
            "putere": putere_calc})

    else:
        # mod manual: S nu calculeaza puterea si nu trimite comenzi
            catre P.
        # puterea curenta este stabilita de SW (prin stare["
            putere_manual"]).
        with lock_stare:
            stare["putere_curenta"] = putere_manual

        # nu trimitem nimic pe q_comenzi_automat in manual

    # Afisam starea o data la perioada_afisare_S secunde
    acum = time.monotonic()
    if acum >= next_afisare:
        next_afisare = acum + configurare["perioada_afisare_S"]

        with lock_stare:
            mod_afis = stare["mod"]
            putere_afis = stare["putere_curenta"]

        pres = ultima_presiune["presiune"] if ultima_presiune is not
            None else float("nan")
```

```
398
399             valva = ultima_presiune.get("valva", float("nan")) if
                    ultima_presiune is not None else float("nan")
400
401             with lock_consola:
402                 print(
403                     f"[S] mod={mod_afis:7s} ; T_medie={t_medie:5.2f} C ;
                            confort={confort:12s} ; "
404                     f"presiune={pres:4.2f} ; putere={putere_afis:5.1f}% ;
                            valva={valva:3.1f}"
405                 )
406
407         # eliberam CPU-ul fara busy-wait
408         stop_event.wait(timeout=0.02)
409
410 def main():
411     # "configurare" contine parametrii sistemului
412     configurare = {
413         # parametri confort
414         "temperatura_referinta": 22.0, # temperatura tinta
415         "banda_confort": 1.0, # +/- 1 grad C fata de tinta
416
417         # parametri model temperatura
418         "temperatura_ambient": 18.0, # fara incalzire
419         "delta_max_incalzire": 10.0, # cu cate grade urca peste ammbient la
                putere 100%
420         "viteza_raspuns_temperatura": 0.08, # cat de repede urca/scade
                temperatura
421
422         # parametri presiune
423         "presiune_referinta": 3.0, # nivel normal de presiune
424         "presiune_maxima_siguranta": 4.0, #prag de siguranta, peste, se
                deschide valva complet
425
426         # perioade (secunde)
427         "perioada_T": 0.5, # perioada de citire temperatura
428         "perioada_P": 0.2, # perioada de reglare presiune
429         "perioada_afisare_S": 1.0, # perioada de afisare stare in S
430
431         # numar termocupluri
432         "numar_TC": 4, # cate termocupluri avem
433     }
434
435     # "stare" este zona partajata intre thread-uri. Orice citire/scriere din
            stare trebuie facuta sub lock_stare (mutex).
436     # Asta previne secventa gresita "date incorecte intre S si P".
437     stare = {
438         "mod": "automat",
439         "putere_manual": 30.0, # setata de SW in mod manual
440         "putere_curenta": 0.0, # actualizata de S si folosita de T si P
441     }
442
443     # Mutex (Lock) = excludere mutuala pentru "stare"
444     lock_stare = threading.Lock() # pentru a proteja accesul la "stare"
445
446     # Lock separat pentru print-uri ca sa nu se amestece liniile
```

```
447     lock_consola = threading.Lock()
448
449     # main seteaza stop_event, celelalte thread-uri verifica stop_event si
            ies
450     stop_event = threading.Event()
451
452     # Cozi de mesaje:
453     # maxsize=1 pentru temperaturi/presiune/comenzi automate - pastram doar
            ultimul mesaj
454     q_evenimente_sw = queue.Queue(maxsize=10)
455     q_temperaturi = queue.Queue(maxsize=1)
456     q_comenzi_automat = queue.Queue(maxsize=1)
457     q_presiune = queue.Queue(maxsize=1)
458
459     # Thread-urile sunt create cu daemon=True.
460     # daca thread-ul principal (main) se termina, thread-urile daemon nu mai
            tin procesul in viata.
461     # util sa nu ramana procesul blocat.
462     # facem si join(timeout) ca sa fim siguri ca se termina
463     th_sw = threading.Thread(target=task_sw, name="SW", args=(
            q_evenimente_sw, stop_event, lock_consola), daemon=True)
464     th_t = threading.Thread(target=task_t, name="T", args=(configurare,
            stare, lock_stare, q_temperaturi, stop_event), daemon=True)
465     th_p = threading.Thread(target=task_p, name="P", args=(configurare,
            stare, lock_stare, q_comenzi_automat, q_presiune, stop_event), daemon
            =True)
466     th_s = threading.Thread(target=task_s, name="S", args=(configurare,
            stare, lock_stare, q_evenimente_sw, q_temperaturi, q_comenzi_automat,
             q_presiune, stop_event, lock_consola), daemon=True)
467
468     # Pornim thread-urile
469     th_t.start()
470     th_p.start()
471     th_s.start()
472     th_sw.start()
473
474     # Main asteapta oprirea fara busy-wait
475     try:
476         while not stop_event.is_set():
477             stop_event.wait(timeout=0.5)
478     except KeyboardInterrupt:
479         stop_event.set()
480
481     # Asteptam terminarea thread-urilor cu timeout
482     for th in (th_sw, th_t, th_s, th_p):
483         try:
484             th.join(timeout=1.0)
485         except RuntimeError:
486             pass
487
488     with lock_consola:
489         print("\n Oprire program")
490
491 if __name__ == "__main__":
492     main()
```

4 Testarea aplicației și validarea soluției propuse

In acest capitol am testat cateva scenarii de functionare:

4.0.1 Scenariul 1: Regim automat - stabilizare temperatura si putere

In acest scenariu sistemul ruleaza in mod **automat**, iar task-ul S calculeaza puterea de incalzire in functie de eroarea fata de temperatura de referinta. Se observa ca, pornind de la o temperatura medie mai mica (confort=rece), puterea este initial ridicata pentru a incalzi rapid, apoi scade treptat pe masura ce temperatura se apropie de zona de confort (confort=confortabil). Presiunea ramane in jurul valorii de referinta, iar valva este actionata ocazional (0.6) atunci cand presiunea depaseste pragul intermediar (`presiune_referinta`+0.3), confirmand ca P intervine preventiv.

```
[SW] > [S] mod=automat ; T_medie=19.15 C ; confort=rece         ; presiune=3.09 ; putere= 64.2% ; valva=0.0
[S] mod=automat ; T_medie=19.91 C ; confort=rece         ; presiune=3.16 ; putere= 55.1% ; valva=0.0
[S] mod=automat ; T_medie=20.41 C ; confort=rece         ; presiune=3.26 ; putere= 49.1% ; valva=0.0
[S] mod=automat ; T_medie=20.78 C ; confort=rece         ; presiune=3.25 ; putere= 44.7% ; valva=0.6
[S] mod=automat ; T_medie=21.06 C ; confort=confortabil  ; presiune=3.25 ; putere= 41.3% ; valva=0.6
[S] mod=automat ; T_medie=21.28 C ; confort=confortabil  ; presiune=3.26 ; putere= 38.7% ; valva=0.6
[S] mod=automat ; T_medie=21.33 C ; confort=confortabil  ; presiune=3.30 ; putere= 38.0% ; valva=0.0
[S] mod=automat ; T_medie=21.39 C ; confort=confortabil  ; presiune=3.27 ; putere= 37.3% ; valva=0.6
[S] mod=automat ; T_medie=21.40 C ; confort=confortabil  ; presiune=3.28 ; putere= 37.2% ; valva=0.0
[S] mod=automat ; T_medie=21.48 C ; confort=confortabil  ; presiune=3.28 ; putere= 36.3% ; valva=0.6
[S] mod=automat ; T_medie=21.47 C ; confort=confortabil  ; presiune=3.27 ; putere= 36.4% ; valva=0.0
[S] mod=automat ; T_medie=21.46 C ; confort=confortabil  ; presiune=3.30 ; putere= 36.5% ; valva=0.0
[S] mod=automat ; T_medie=21.48 C ; confort=confortabil  ; presiune=3.30 ; putere= 36.2% ; valva=0.0
[S] mod=automat ; T_medie=21.54 C ; confort=confortabil  ; presiune=3.28 ; putere= 35.5% ; valva=0.0
[S] mod=automat ; T_medie=21.45 C ; confort=confortabil  ; presiune=3.29 ; putere= 36.7% ; valva=0.0
[S] mod=automat ; T_medie=21.54 C ; confort=confortabil  ; presiune=3.26 ; putere= 35.6% ; valva=0.6
[S] mod=automat ; T_medie=21.53 C ; confort=confortabil  ; presiune=3.27 ; putere= 35.7% ; valva=0.0
[S] mod=automat ; T_medie=21.48 C ; confort=confortabil  ; presiune=3.28 ; putere= 36.2% ; valva=0.0
[S] mod=automat ; T_medie=21.54 C ; confort=confortabil  ; presiune=3.28 ; putere= 35.5% ; valva=0.0
[S] mod=automat ; T_medie=21.54 C ; confort=confortabil  ; presiune=3.26 ; putere= 35.6% ; valva=0.0
[S] mod=automat ; T_medie=21.52 C ; confort=confortabil  ; presiune=3.26 ; putere= 35.8% ; valva=0.0
[S] mod=automat ; T_medie=21.57 C ; confort=confortabil  ; presiune=3.27 ; putere= 35.2% ; valva=0.6
[S] mod=automat ; T_medie=21.55 C ; confort=confortabil  ; presiune=3.26 ; putere= 35.4% ; valva=0.6
[S] mod=automat ; T_medie=21.58 C ; confort=confortabil  ; presiune=3.28 ; putere= 35.1% ; valva=0.0
```

Scenariul 1: mod automat - putere calculata de S, temperatura converge spre referinta, valva intervine preventiv.

4.0.2 Scenariul 2: Comutare pe modul manual si setare putere fixa la 40%

Aici se demonstreaza comutarea corecta din automat in **manual**. Dupa comanda `m`, modul devine manual, iar puterea afisata nu mai este calculata de S, ci este fixata de utilizator prin `p 40`. Se vede ca

puterea ramane constanta la 40.0% pe mai multe iteratii, ceea ce confirma prevenirea secventei gresite "S continua sa controleze in manual". Temperaturile continua sa fie achizitionate de T si afisate, iar P continua sa controleze presiunea (valva ramane 0.0 / 0.6 in functie de presiune).

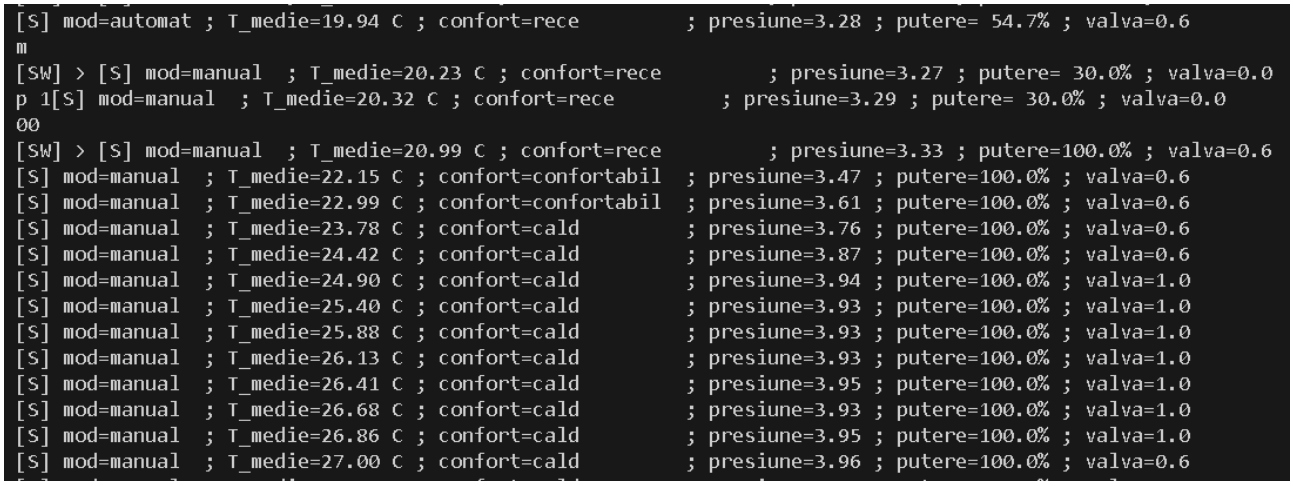
```
m[S] mod=automat ; T_medie=21.49 C ; confort=confortabil  ; presiune=3.27 ; putere= 36.1% ; valva=0.6

[SW] > [S] mod=manual  ; T_medie=21.56 C ; confort=confortabil  ; presiune=3.26 ; putere= 30.0% ; valva=0.6
p [S] mod=manual  ; T_medie=21.46 C ; confort=confortabil  ; presiune=3.26 ; putere= 30.0% ; valva=0.0
40
[SW] > [S] mod=manual  ; T_medie=21.37 C ; confort=confortabil  ; presiune=3.29 ; putere= 40.0% ; valva=0.0
[S] mod=manual  ; T_medie=21.46 C ; confort=confortabil  ; presiune=3.29 ; putere= 40.0% ; valva=0.0
[S] mod=manual  ; T_medie=21.57 C ; confort=confortabil  ; presiune=3.26 ; putere= 40.0% ; valva=0.0
[S] mod=manual  ; T_medie=21.66 C ; confort=confortabil  ; presiune=3.27 ; putere= 40.0% ; valva=0.6
[S] mod=manual  ; T_medie=21.73 C ; confort=confortabil  ; presiune=3.26 ; putere= 40.0% ; valva=0.0
[S] mod=manual  ; T_medie=21.85 C ; confort=confortabil  ; presiune=3.25 ; putere= 40.0% ; valva=0.6
[S] mod=manual  ; T_medie=21.87 C ; confort=confortabil  ; presiune=3.26 ; putere= 40.0% ; valva=0.6
[S] mod=manual  ; T_medie=21.94 C ; confort=confortabil  ; presiune=3.30 ; putere= 40.0% ; valva=0.0
[S] mod=manual  ; T_medie=21.87 C ; confort=confortabil  ; presiune=3.29 ; putere= 40.0% ; valva=0.0
[S] mod=manual  ; T_medie=21.83 C ; confort=confortabil  ; presiune=3.29 ; putere= 40.0% ; valva=0.0
[S] mod=manual  ; T_medie=21.94 C ; confort=confortabil  ; presiune=3.26 ; putere= 40.0% ; valva=0.6
[S] mod=manual  ; T_medie=22.00 C ; confort=confortabil  ; presiune=3.27 ; putere= 40.0% ; valva=0.0
[S] mod=manual  ; T_medie=21.93 C ; confort=confortabil  ; presiune=3.28 ; putere= 40.0% ; valva=0.0
[S] mod=manual  ; T_medie=21.94 C ; confort=confortabil  ; presiune=3.28 ; putere= 40.0% ; valva=0.0
[S] mod=manual  ; T_medie=21.98 C ; confort=confortabil  ; presiune=3.28 ; putere= 40.0% ; valva=0.0
[S] mod=manual  ; T_medie=21.94 C ; confort=confortabil  ; presiune=3.28 ; putere= 40.0% ; valva=0.0
[S] mod=manual  ; T_medie=21.95 C ; confort=confortabil  ; presiune=3.27 ; putere= 40.0% ; valva=0.0
[S] mod=manual  ; T_medie=21.96 C ; confort=confortabil  ; presiune=3.25 ; putere= 40.0% ; valva=0.0
[S] mod=manual  ; T_medie=21.94 C ; confort=confortabil  ; presiune=3.25 ; putere= 40.0% ; valva=0.6
[S] mod=manual  ; T_medie=21.98 C ; confort=confortabil  ; presiune=3.29 ; putere= 40.0% ; valva=0.0
[S] mod=manual  ; T_medie=22.04 C ; confort=confortabil  ; presiune=3.28 ; putere= 40.0% ; valva=0.0
[S] mod=manual  ; T_medie=21.99 C ; confort=confortabil  ; presiune=3.28 ; putere= 40.0% ; valva=0.0
[S] mod=manual  ; T_medie=21.93 C ; confort=confortabil  ; presiune=3.28 ; putere= 40.0% ; valva=0.0
```

Scenariul 2: comutare in manual si setare p 40 - puterea ramane fixa, S nu suprascrie comanda manuala.

4.0.3 Scenariul 3: Siguranta presiune - manual cu putere 100% (valva 1.0 la prag maxim)

In acest scenariu se testeaza situatia de solicitare maxima: comutare in **manual** si setare `p 100`. Temperatura medie creste semnificativ (confort ajunge cald), iar presiunea urca spre pragul de siguranta. Se observa clar ca, atunci cand presiunea ajunge in zona critica, P comuta valva pe **1.0** (deschidere maxima), adica mecanismul de siguranta intervine pentru a preveni depasirea necontrolata a presiunii. Astfel, scenariul demonstreaza comportamentul corect de protectie al modulului P.



Scenariul 3: manual cu p 100 - presiunea creste si declanseaza valva la 1.0 (regim de siguranta).

4.0.4 Scenariul 4: Revenire la modul automat dupa supraincalzire (puterea scade)

Aici se demonstreaza revenirea la control automat dupa ce sistemul a facut prea multa caldura. La comanda `a`, modul devine **automat**, iar S reincepe sa calculeze puterea. Se observa ca, pentru temperatura medie foarte mare (confort=cald), puterea calculata scade pana la **0.0%**, apoi creste din nou gradual pe masura ce temperatura revine spre zona de confort.

Scenariul 4: revenire in automat, puterea scade la 0 cand temperatura e prea mare, apoi se ajusteaza spre stabilizare.

Bibliografie

[1] Python Software Foundation, *Python Documentation: threading — Thread-based parallelism*, https://docs.python.org/3/library/threading.html, Documentatie oficiala Python (threading), 2025.

[2] Python Software Foundation, *Python Documentation: queue — A synchronized queue class*, https://docs.python.org/3/library/queue.html, Documentatie oficiala Python (queue), 2025.

[3] *POSIX and RTOS (Chapter 13) - POSIX Compliance and Conformance*, Fisier PDF furnizat in cadrul laboratorului/proiectului PATR, Material suport: POSIX, thread-uri, extensii real-time si conformanta, 2021.

[4] *Real-Time OSs - Note de curs/laborator*, Fisier PDF furnizat in cadrul laboratorului/proiectului PATR, Material suport: concepte RTOS, prioritati, scheduling, sincronizare, 2021.

[5] *Multitask Resource Sharing - Resource Deadlock*, Fisier PDF furnizat in cadrul laboratorului/proiectului PATR, Material suport: conditii de deadlock, partajare resurse, sectiuni critice, 2025.