Tema de casă **Programarea Aplicaţiilor în Timp Real**

Echipa: Necula Leonard-Gabriel — Mihăilescu Dan-Ionuț Sem. 2, 2020-2021
e-mail: necula.leonard.gabriel@gmail.com — dan.mihailescu1007@stud.acs.upb.ro

SISTEM DE ÎMBUTELIERE

1 Introducere - prezentarea problemei

Îmbutelierea sticlelor ce vin pe o bandă transportoare. Linia de automatizare a acestei operaţii conţine două staţii de lucru:

- o staţie de umplere a sticlelor, umplerea unei sticle se face într-un timp $t_{umplere}$
- o staţie de adăugare a dopului, adăugarea dopului se face în timpul t_{dop}

Elemente sistem:

1. motor bandă transportoare
2. senzor detecţie sticlă goală
3. senzor detecţie sticlă fără dop

1.1 Pas 1. Definire problemă

Să se implementeze o aplicaţie de simulare a unui proces de îmbuteliere. Această aplicaţie conţine următoarele taskuri:

- task 1 (umplere sticlă cu lichid)
- task 2 (adăugare dop)
- task 3 (pornire/oprire bandă transportoare)

Condiţiile impuse sunt următoarele:

- banda transportoare se opreşte atunci când apare o sticlă simultan în faţa senzorului de detecţie sticlă goală şi al senzorului de detecţie sticlă fără dop;
- banda transportoare reporneşte atunci când taskurile de umplere cât şi cel de adăugare dop şi-au terminat execuţia;
- procesul începe cu sticle prezente în faţa ambelor staţii de lucru

2 Analiza problemei

2.1 Exemplu: Pas 2. Analiza cerinţelor

Evenimente posibile:

- detectarea unei sticle în dreptul staţiei de umplere
- detectarea unei sticle în dreptul staţiei de adăugare a dopului

Acţiuni posibile:

- pornire/oprire bandă transportoare la deteţia sticle
- umplerea şi adăugarea dopului

Situaţii imposibile:

- sticlele nu vin în acelaşi timp la cele 2 staţii de lucru
- banda nu se opreşte în momentul în care cei doi senzori detectează prezenţa sticlelor la cele 2 staţii de lucru

3 Definirea structurii aplicaţiei

3.1 Pas 3. Definirea taskurilor care compun aplicaţia

În cazul aplicaţiei prezentate taskurile sunt:

- task 1 (umplere sticlă cu lichid)
- task 2 (adăugare dop)
- task 3 (pornire/oprire bandă transportoare)

4 Definirea soluţiei în vederea implementării

Rezolvarea problemei propuse a fost făcută în Eclipse utilizând limbajul Java.

Mecanismele utilizate:

- excludere mutuală
- sincronizare pe condiţie de timp
- împărţirea problemei în clase conform paradigmei OOP

4.1 Pas 4. Soluţie de implementare

Pentru problema prezentată avem nevoie de următoarele mecanisme de comunicare între taskuri semnafoare generalizate, mutex-uri. Pentru vizualizarea taskurilor şi a sincronizarilor vedeţi figura în Fig. 1.

- aleg un semafoar generalizat (e.g. *counting semaphore*) S_2, cu valorea iniţială $S_2 = 0$ şi două semafoare binare S_0, S_1 cu valorile iniţiale $S_0 = 0$, $S_1 = 0$.
- folosesc un mutex pentru excluderea mutuală a accesului la zona de memorie pentru statusul benzii transportoare (*nu vreau ca operaţiile de read şi write să se poată executa concomitent*)

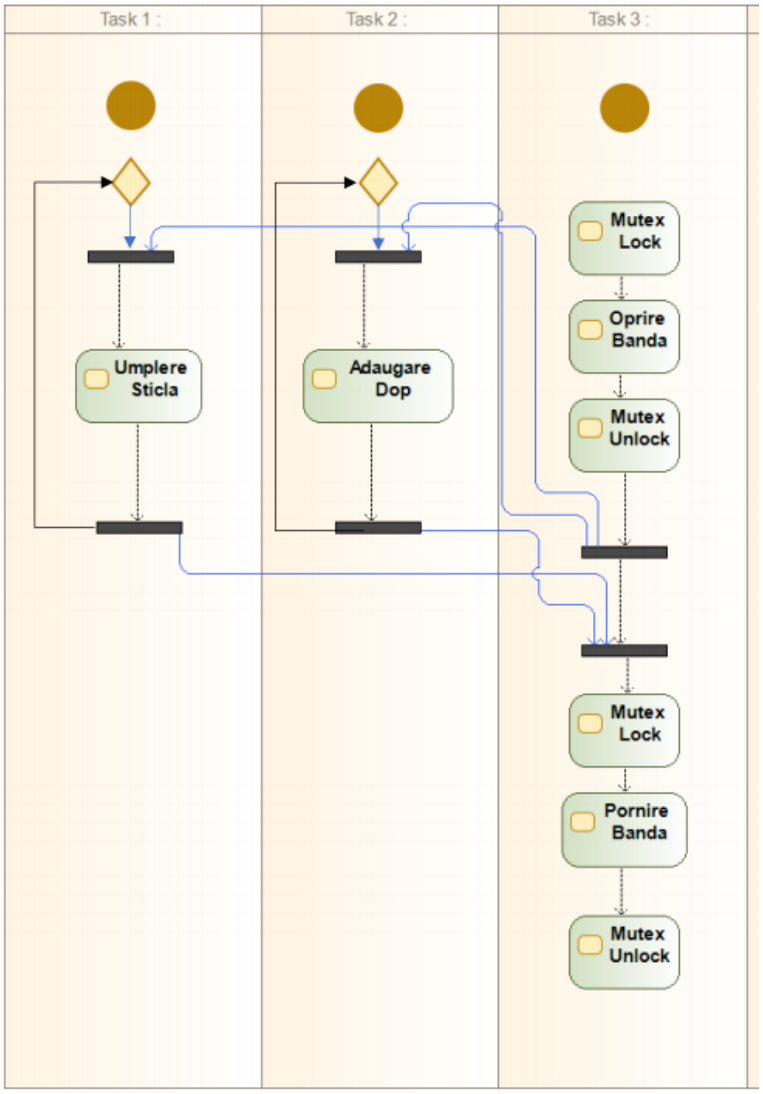

Figure 1: Soluţie implementare - organigrame taskuri

5 Implementarea soluţiei

5.1 Cod program

Detalii de implementare:

- pentru simularea prezenţei sticlelor la începutul simulării utilizăm un fir de execuţie ce rulează la începutul programului, urmând să ruleze ciclic la $t_{detectare}$ secunde
- rularea ciclică a taskului de control al benzii transportoare din $t_{detectare}$ în $t_{detectare}$ secunde am utilizat un timer (*vezi clasa Timer*)
- pentru asigurarea accesului unui singur task la zona de memorie a statusului benzii transportoare folosim un mutex

Cod program:

```
1
2  package Implementare;
3
4  import Semafoare.SemBin;
5  import Semafoare.Semafor;
6
7  // Definire lista de variabile globale prin intermediul
8  // unei interfete
9
10 public interface GVL {
11   public static Semafor S2 = new Semafor();
12   public static SemBin  S0 = new SemBin();
13   public static SemBin  S1 = new SemBin();
14 }
15
16 package Implementare;
17
18 public class TaskDop extends Thread {
19
20   private String task_name;
21   private int task_time;
22
23   public TaskDop(String task_name, int task_time)
24   {
25     this.task_name = task_name;
26     this.task_time = task_time;
27   }
28
29   public TaskDop()
30   {
31     this.task_name = "Default_Name";
32     this.task_time = 1000;
33   }
34
35   @Override
36   public void run()
37   {
38     while(true)
39     {
40
41       GVL.S1.sem_wait();
```

```
42
43        System.out.println("A inceput task-ul de " + task_name);
44
45        try {
46          TaskDop.sleep(task_time);
47        } catch (InterruptedException e) {
48          e.printStackTrace();
49        }
50
51        System.out.println("S-a terminat task-ul de " + task_name);
52        GVL.S2.sem_post();
53
54      }
55    }
56  }
57
58  package Implementare;
59
60  public class TaskUmplere extends Thread {
61
62    private String task_name;
63    private int task_time;
64
65    public TaskUmplere(String task_name, int task_time)
66    {
67      this.task_name = task_name;
68      this.task_time = task_time;
69    }
70
71    public TaskUmplere()
72    {
73      this.task_name = "Default_Name";
74      this.task_time = 1000;
75    }
76
77    @Override
78    public void run()
79    {
80      while(true)
81      {
82
83        GVL.S0.sem_wait();
84        System.out.println("A inceput task-ul de " + task_name);
85
86        try {
87          TaskUmplere.sleep(task_time);
88        } catch (InterruptedException e) {
89          e.printStackTrace();
90        }
91
92        System.out.println("S-a terminat task-ul de " + task_name);
93        GVL.S2.sem_post();
94
95      }
96    }
97  }
98
99  package Implementare;
100
101 import java.util.TimerTask;
```

```
102
103 public class TaskBanda extends TimerTask {
104
105   int state = 0;
106   static int initial = 0;
107
108   public TaskBanda(int state)
109   {
110     this.state = state;
111   }
112
113   public TaskBanda()
114   {
115     this.state = 0;
116   }
117
118   @Override
119   public void run() {
120
121     if(initial != 0) {
122       synchronized(this) {
123         System.out.println("Oprire banda");
124         this.state = 0; // Oprire banda
125       }
126
127     }
128     else
129       initial = 1;
130
131     GVL.S0.sem_post();
132     GVL.S1.sem_post();
133
134     GVL.S2.sem_wait();
135     GVL.S2.sem_wait();
136
137     synchronized(this) {
138       System.out.println("Repornire banda");
139       this.state = 1; // Repornire banda
140     }
141
142
143   }
144
145 }
146
147
148
149 package Implementare;
150
151 import java.util.Timer;
152
153 import Semafoare.SemBin;
154 import Semafoare.Semafor;
155
156
157
158 public class Runner {
159
160   public static void main(String[] args) {
161     // TODO Auto-generated method stub
```

```
162     int t_umplere = 1000 * 2;
163     int t_dop = 1000 * 5;
164     int t_detectare = 1000 * 15;
165 
166     TaskUmplere T1 = new TaskUmplere("umplere a sticlei", t_umplere);
167     TaskDop T2 = new TaskDop("adaugare a dopului", t_dop);
168     TaskBanda T3 = new TaskBanda();
169 
170 
171     Timer timer_banda = new Timer();
172     timer_banda.schedule(T3, 0, t_detectare);
173     T1.start();
174     T2.start();
175 
176   }
177 
178 }
```

6 Testarea aplicaţiei si validarea soluţiei propuse

Pentru rularea aplicaţiei deschideţi aplicaţia Eclipse, importaţi proiectul şi rulaţi clasa *Runner.java*.

Mai jos aveţi un exemplu de rulare a codului în Eclipse:

```
A inceput task-ul de adaugare a dopului
A inceput task-ul de umplere a sticlei
S-a terminat task-ul de umplere a sticlei
S-a terminat task-ul de adaugare a dopului
Repornire banda
Oprire banda
A inceput task-ul de umplere a sticlei
A inceput task-ul de adaugare a dopului
S-a terminat task-ul de umplere a sticlei
S-a terminat task-ul de adaugare a dopului
Repornire banda
Oprire banda
A inceput task-ul de umplere a sticlei
A inceput task-ul de adaugare a dopului
S-a terminat task-ul de umplere a sticlei
S-a terminat task-ul de adaugare a dopului
Repornire banda
Oprire banda
A inceput task-ul de umplere a sticlei
A inceput task-ul de adaugare a dopului
```

Figure 2: Exemplu output

După cum puteţi observa rezultatele sunt cele aşteptate, de asemenea sunt identice cu cele din cadrul aplicaţiei implementate în C-POSIX, singura diferenţă fiind dată de lipsa semnalelor pentru afişare statusului benzii, această afişare fiind făcută acum de task-ul ce porneşte şi opreşte banda transportoare.