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

Echipa: Necula Leonard-Gabriel
e-mail: necula.leonard.gabriel@gmail.com

Sem. 2, 2020-2021

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)
- check status (verificare status 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 lucr

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)
- check status (verificare status bandă transportoare)

4 Definirea soluţiei în vederea implementării

Rezolvarea problemei propuse a fost făcută în Linux Xenomai utilizând limbajul C utilizând standardul de programare în timp real POSIX.

Mecanismele utilizate:

- excludere mutuală
- sincronizare pe condiţie de timp
- sincronizare prin semnale

4.1 Pas 4. Soluţie de implementare

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

- aleg 3 semafoare generalizate (e.g. *counting semaphore*) S_0, S_1, S_2, cu valorile iniţiale $S_0 = 0$, $S_1 = 0$ şi $S_2 = 0$
- 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*)
- folosesc semnalul SIGUSR1 (*cod 10* în lista semnalelor *POSIX*)

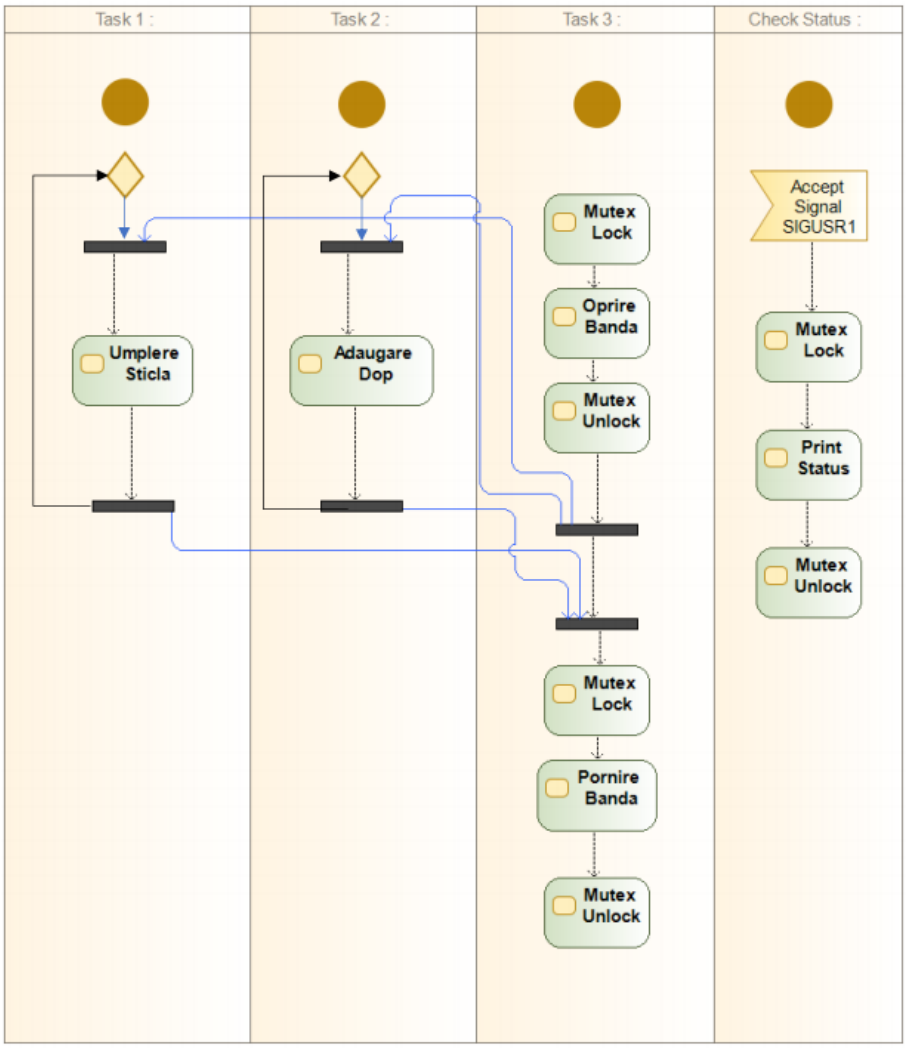

Figure 1: Soluţie implementare - organigrame taskuri

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
- pentru asigurarea accesului unui singur task la zona de memorie a statusului benzii transportoare folosim un mutex

Cod program:

```
1
2  #include <stdio.h>
3  #include <pthread.h>
4  #include <stdlib.h>
5  #include <semaphore.h>
6  #include <string.h>
7  #include <unistd.h>
8  #include <signal.h>
9  #include <time.h>
10
11 sem_t Sem[3];
12
13 void task1(); // umplere sticla
14 void task2(); // adaugare dop
15 void task3(); // oprire banda transportoare
16
17 void (*tasks[])() = {task1, task2, task3};
18 void status_check(); // verificare status banda
19
20 // Constante de timp in [secunde]
21 int t_umplere = 2;
22 int t_dop = 5;
23 int t_detectare = 15;
24
25 // stare banda, 1 - banda pornita
26 // 0 - banda oprita
27 int banda = 1;
28
29 // Mecanism de excludere mutuala pentru accesarea
30 // zonei de memorie cu statusul benzii transportoare
31 pthread_mutex_t mutex;
32
33 int main(int argc, char **argv)
34 {
35
36     pthread_t FIR[3];    // definire fire de executie
37     pthread_attr_t attr; // definire atribute fire
38
39     timer_t timerid;           // definire timer
40     struct sigevent sev;       // definire ss
41     struct itimerspec trigger; // definire structura timp
```

```
42                                  // pentru timer
43      struct sigaction act;       // actiuni la aparitia semnalelor
44      sigset_t set;               // set de semnale tratate
45
46      sigemptyset(&set);         // golire set semnale
47      sigaddset(&set, SIGUSR1); // adaugare semnal
48
49      // Configurare actiune la aparitia unui semnal
50      act.sa_flags = 0;
51      act.sa_mask = set;
52      act.sa_handler = &status_check;
53      // Configurare actiune -> semnal
54      sigaction(SIGUSR1, &act, NULL);
55
56      memset(&sev, 0, sizeof(struct sigevent));
57      memset(&trigger, 0, sizeof(struct itimerspec));
58
59      // Setare task ciclic generat de timer
60      sev.sigev_notify = SIGEV_THREAD;
61      sev.sigev_notify_function = &task3;
62
63      // Setare timer
64      timer_create(CLOCK_REALTIME, &sev, &timerid);
65      trigger.it_value.tv_sec = t_detectare;
66      trigger.it_interval.tv_sec = t_detectare;
67      timer_settime(timerid, 0, &trigger, NULL);
68
69      if ((pthread_mutex_init(&mutex, NULL)))
70      {
71          perror("pthread_mutex_init() failed");
72          return EXIT_FAILURE;
73      }
74
75      int i;
76
77      // Initializare semafoare generalizate
78      for (i = 0; i < 3; i++)
79          if (!sem_init(Sem + i, 1, 0))
80          {
81              ; // Nu facem nimic in cazul in care initializarea a fost
82                // facuta cu succes
83          }
84          else
85              printf("Eroare la initializarea semaforului numarul %d \n", i + 1);
86
87      // Creare fire de executie
88      for (i = 0; i < 3; i++)
89          if (pthread_create(FIR + i, &attr, (void *)(*(tasks + i)), NULL) != 0)
90          {
91              perror("pthread_create");
92              return EXIT_FAILURE;
93          }
94
95      while (1)
96          ;
97
98      return 0;
99  }
100
101 void task1()
```

```
{
    while (1)
    {
        sem_wait(Sem + 0); // P(S0)
        fflush(stdout);
        puts("Umplere sticla - START");
        sleep(t_umplere);
        puts("Umplere sticla - FINISH");
        sem_post(Sem + 2); // V(S2)
    }
}

void task2()
{
    while (1)
    {
        sem_wait(Sem + 1); // P(S1)
        fflush(stdout);
        puts("Adaugare dop - START");
        sleep(t_dop);
        puts("Adaugare dop - FINISH");
        sem_post(Sem + 2); // V(S2)
    }
}

void task3()
{
    pthread_mutex_lock(&mutex);
    banda = 0;
    pthread_mutex_unlock(&mutex);
    sem_post(Sem + 0); // V(S0)
    sem_post(Sem + 1); // V(S1)
    sem_wait(Sem + 2); // P(S2)
    sem_wait(Sem + 2); // P(S2)
    pthread_mutex_lock(&mutex);
    banda = 1;
    pthread_mutex_unlock(&mutex);
}

void status_check()
{
    fflush(stdout);
    pthread_mutex_lock(&mutex);
    printf("\n###### Banda = %d ######\n", banda);
    pthread_mutex_unlock(&mutex);
}
```

6 Testarea aplicaţiei si validarea soluţiei propuse

Pentru rularea aplicaţiei puteţi utiliza comenzile din fisierul makefile atasat. Comenzile pe care le gasiţi în acest fisier sunt:

- *make build file* = *numefisiersursa* - rutină pentru generarea fisierului executabil
- *make run* - rutină pentru rularea fisierului executabil
- *make clean* - rutină pentru stergerea fisierului executabil

Observaţii:

- Atunci când scrieţi numele fisierului pe care doriţi să-l compilaţi **NU** scrieţi şi extensia, aceasta a fost deja setată!
- Pentru a transmite semnalul $SIGUSR1$ către proces, deschideţi un terminal şi scrieţi comanda $pidof\ executabil\_tema$ urmată de comanda $kill -10\ pid\ executabil\_tema$, pid identificat cu ajutorul primei comenzi rulate

Mai jos aveţi un exemplu de rulare a codului sub Xenomai:

```
Adaugare dop - START
Umplere sticla - START
Umplere sticla - FINISH
Adaugare dop - FINISH
Umplere sticla - START
Adaugare dop - START
Umplere sticla - FINISH
Adaugare dop - FINISH

####### Banda = 1 #######
Adaugare dop - START
Umplere sticla - START

####### Banda = 0 #######
Umplere sticla - FINISH

####### Banda = 0 #######
Adaugare dop - FINISH

####### Banda = 1 #######
```

Figure 2: Exemplu output