

Echipa: APOSTU Ș Raluca, PLESCA Natalia, CONTIU Alexandru  
e-mail: raluca.marian54@yahoo.com, plescanatalia2000@gmail.com, contiu.marian@gmail.com

## INTERSECȚIE SEMAFORIZATĂ

### 1 Introducere - prezentarea problemei

Problema pe care echipa noastră a abordat-o este aceea a unei intersecții semaforizate. Procesele vor fi reprezentate de stările unui semafor și direcțiile de mers ale șoferilor.

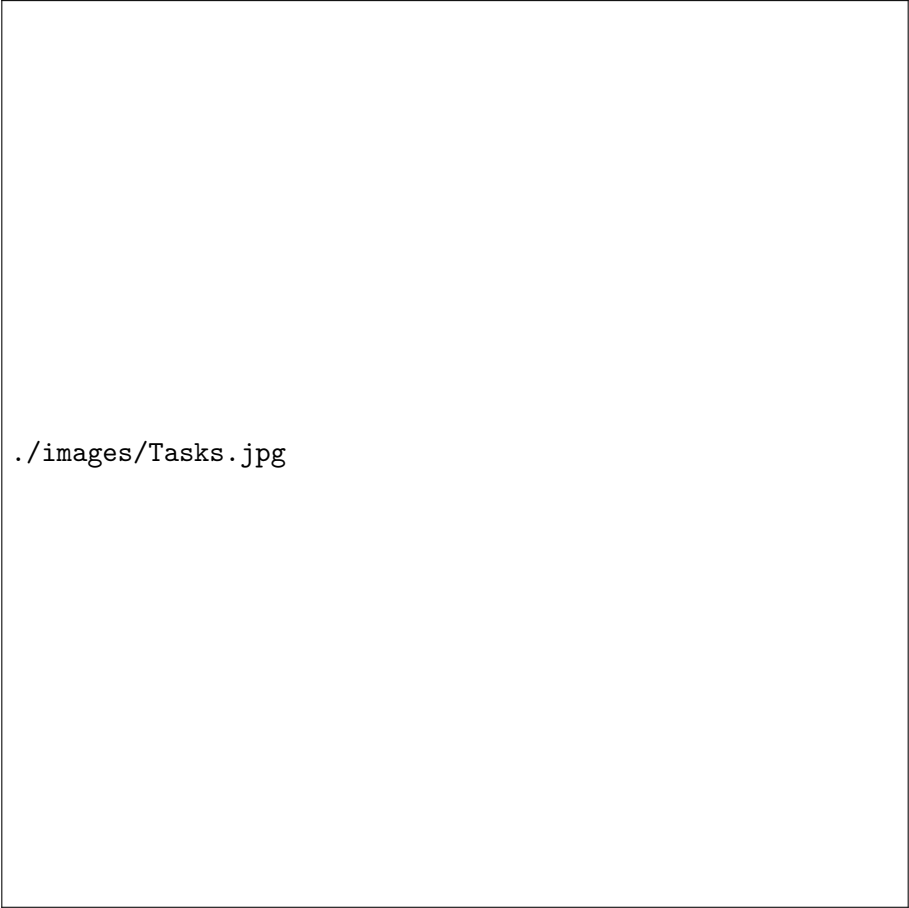
#### 1.1 Exemplu: Pas 1. Definire problemă

Se va implementa o aplicație formată din 4 taskuri:

- Task 1 (funcția PerecheA) - semafoarele pentru vehicule (V2, V3) devin roșii, în timp ce semafoarele pentru vehicule (V0, V1) și cele pentru pietoni (P2, P3) își schimbă starea din roșu în verde;
- Task 2 (funcția TranzițieAB) - pentru a avertiza șoferii de pe șoseaua verticală cu privire la schimbarea culorii semafoarelor, semafoarele pentru vehicule (V0, V1) își schimbă culoarea în galben, semafoarele pentru pietoni (P2, P3) se fac roșii, iar celelalte semafoare își păstrează starea;
- Task 3 (funcția PerecheB) - semafoarele pentru vehicule (V0, V1) devin roșii, în timp ce semafoarele pentru vehicule (V2, V3) și cele pentru pietoni (P0, P1) își schimbă starea din roșu în verde;
- Task 4 (funcția TranzițieBA) - pentru a avertiza șoferii de pe șoseaua orizontală cu privire la schimbarea culorii semafoarelor, semafoarele pentru vehicule (V2, V3) își schimbă culoarea în galben, semafoarele pentru pietoni (P0, P1) se fac roșii, iar celelalte semafoare își păstrează starea.

Condițiile de schimbare a culorii semaforului sunt următoarele:

- Condiția 1 - semafoarele de pe șoseaua verticală și cele de pe șoseaua orizontală nu pot indica culoarea verde sau galbenă în același timp;
- Condiția 2 - semafoarele pentru pietoni de pe o șosea nu pot fi verzi atâta timp cât semafoarele pentru vehicule de pe acea șosea sunt verzi sau galbene;
- Condiția 3 - tranzițiile (culoarea galbenă) sunt condiționate de un interval de 5 secunde, iar stările (culoarea roșie sau verde) sunt condiționate de un interval de 10 secunde.



```
./images/Tasks.jpg
```

Figure 1: Task-uri

## 2 Analiza problemei

### 2.1 Exemplu: Pas 2. Analiza cerințelor

Secvențele posibile sunt ilustrate în **Figure 1** (această reprezentare este metoda de afișare din terminal).

Secvențe imposibile:

- Semafoarele de pe șoseaua verticală și cele de pe șoseaua orizontală vor indica culoarea verde sau galbenă în același timp;
- Semafoarele pentru pietoni de pe o șosea sunt verzi în timp ce semafoarele pentru vehicule de pe acea șosea sunt verzi sau galbene.

## 3 Definirea structurii aplicației

### 3.1 Exemplu: Pas 3. Definirea taskurilor care compun aplicația

În cazul problemei noastre, acestea sunt taskurile:

- Task 1 (funcția `PerecheA`) : **Task\_A**

- Task 2 (funcția TranzițieAB) : **Task\_B**
- Task 3 (funcția PerecheB) : **Task\_C**
- Task 4 (funcția TranzițieBA) : **Task\_D**

## 4 Definirea soluției în vederea implementării

Am implementat soluția pentru problema aleasă în *Linux Xenomai / C*. Am creat câte un fir de execuție pentru fiecare task în parte, iar pentru a realiza comunicarea între task-uri, am folosit semafoare binare. Deoarece intersecția trebuie să își păstreze starea setată de fiecare task pentru un anumit interval de timp, am folosit funcția ***sleep()*** în cadrul firului de execuție corespunzător task-ului respectiv înainte de a debloca task-ul următor.

### 4.1 Exemplu: Pas 4. Soluție de implementare

Aleg mecanismele de sincronizare și comunicare între taskuri, prezentând organigramele taskurilor (vezi **Figure 2**):

- Am folosit 4 semafoare binare: SemPerecheA, SemPerecheB, SemTranzitieAB, SemTranzitieBA;
- Am creat câte un fir de execuție pentru fiecare task în funcția main, după care am folosit funcția ***sem\_post*** pentru a debloca task-ul 1 (PerecheA);
- În fiecare task, după setarea culorii corespunzătoare a semafoarelor, am folosit funcția ***Intersectie()*** pentru a reprezenta grafic culorile semafoarelor, după care am folosit funcția ***sleep()*** pentru a păstra starea curentă a intersecției pentru un interval de timp stabilit în prealabil;
- La sfârșitul acestui interval de timp este apelată funcția ***sem\_post*** pentru a debloca semaforul task-ului următor.

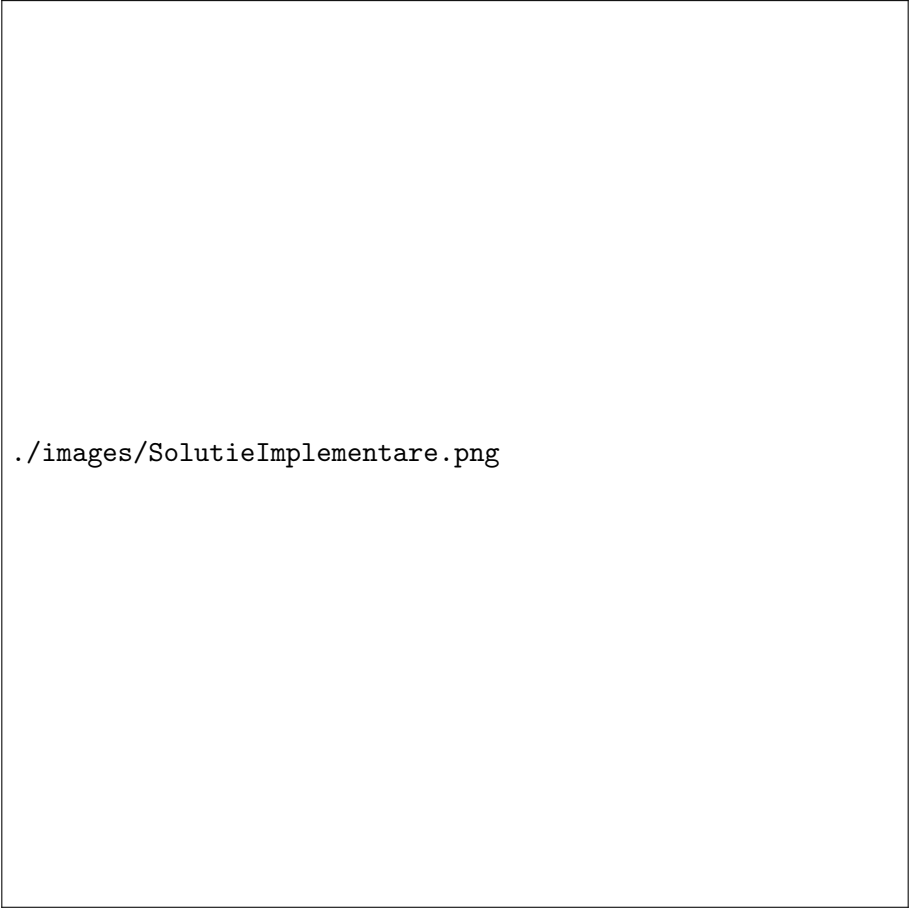
## 5 Implementarea soluției

În această secțiune se va prezenta codul pentru implementare.

***Linux Xenomai /C:***

```
#include <stdlib.h>
#include <stdio.h>
#include <pthread.h>
#include <semaphore.h>
#include <stdbool.h>

#define RED "\e[1;31m"
#define GRN "\e[1;32m"
#define YEL "\e[1;33m"
#define ENDER "\033[0m"
```



./images/SolutieImplementare.png

Figure 2: Soluție implementare - organigrame taskuri

```
struct semafor_vehicule{
bool verde;
bool galben;
bool rosu;
};
typedef struct semafor_vehicule SEM_V;

struct semafor_pietoni{
bool verde;
bool rosu;
};
typedef struct semafor_pietoni SEM_P;

SEM_P P[4];
SEM_V V[4];
sem_t SemPerecheA, SemTranzitieAB, SemPerecheB, SemTranzitieBA;

void* PerecheA(void *);
void* TranzitieAB(void *);
void* PerecheB(void *);
void* TranzitieBA(void *);

void Intersectie(); // Ilustrare grafica a intersectiei (in culori)
```

```

void* (*functie[])(void*) = {PerecheA, TranzitieAB, PerecheB, TranzitieBA};

int main(void)
{
pthread_t FIR[4];
int i;

sem_init(&SemPerecheA, 1, 0);
sem_init(&SemTranzitieAB, 1, 0);
sem_init(&SemPerecheB, 1, 0);
sem_init(&SemTranzitieBA, 1, 0);

for(i=0; i<4;i++)
pthread_create(FIR+i,NULL,(void*)(*(functie+i)),NULL);

sem_post(&SemPerecheA);

for(i=0; i<4; i++)
pthread_join(*(FIR+i), NULL);

printf("Main: Toate firele de executie s-au terminat! \n");
pthread_exit(NULL);
}

void* PerecheA(void *arg)
{
int k;
while(1)
{
sem_wait(&SemPerecheA);
for(k = 0; k < 2; k++)
{
V[k].verde = true;
V[k].rosu = false;
V[k].galben = false;

P[k].verde = false;
P[k].rosu = true;
}
for(k = 2; k < 4; k++)
{
V[k].verde = false;
V[k].rosu = true;
V[k].galben = false;

P[k].verde = true;
P[k].rosu = false;
}
Intersectie();
sleep(10);
sem_post(&SemTranzitieAB);
}
}

```

```

void* TranzitieAB(void *arg)
{
    int k;
    while(1)
    {
        sem_wait(&SemTranzitieAB);
        for(k = 0; k < 2; k++)
        {
            V[k].verde = false;
            V[k].rosu = false;
            V[k].galben = true;

            P[k].verde = false;
            P[k].rosu = true;
        }
        for(k = 2; k < 4; k++)
        {
            V[k].verde = false;
            V[k].rosu = true;
            V[k].galben = false;

            P[k].verde = false;
            P[k].rosu = true;
        }
        Intersectie();
        sleep(5);
        sem_post(&SemPerecheB);
    }
}

void* PerecheB(void *arg)
{
    int k;
    while(1)
    {
        sem_wait(&SemPerecheB);
        for(k = 0; k < 2; k++)
        {
            V[k].verde = false;
            V[k].rosu = true;
            V[k].galben = false;

            P[k].verde = true;
            P[k].rosu = false;
        }
        for(k = 2; k < 4; k++)
        {
            V[k].verde = true;
            V[k].rosu = false;
            V[k].galben = false;

            P[k].verde = false;
            P[k].rosu = true;
        }
        Intersectie();
    }
}

```

```

sleep(10);
sem_post(&SemTranzitieBA);
}
}

void* TranzitieBA(void *arg)
{
int k;
while(1)
{
sem_wait(&SemTranzitieBA);
for(k = 0; k < 2; k++)
{
V[k].verde = false;
V[k].rosu = true;
V[k].galben = false;

P[k].verde = false;
P[k].rosu = true;
}
for(k = 2; k < 4; k++)
{
V[k].verde = false;
V[k].rosu = false;
V[k].galben = true;

P[k].verde = false;
P[k].rosu = true;
}
Intersectie();
sleep(5);
sem_post(&SemPerecheA);
}
}

void Intersectie()
{
system("clear");
printf("                |    \\\\/    |    /\\" |\\n");
printf("                |    \\\\/    |    /\\" |\\n");
printf("                |    \\\\/    ");

if(P[0].rosu) printf(RED);
else if(P[0].verde) printf(GRN);
printf("[P0]");
printf(ENDER);

printf("    /\\" |\\n");
printf("                |    \\\\/    |    /\\" |\\n");
printf("----- ");

if(V[0].rosu) printf(RED);
else if(V[0].verde) printf(GRN);
else if(V[0].galben) printf(YEL);
printf("[V0]");

```

```

printf(ENDER);

printf(" ----- \n");
printf("                                | \n");
printf("          <<<<<<<          | \n");

if(V[2].rosu) printf(RED);
else if(V[2].verde) printf(GRN);
else if(V[2].galben) printf(YEL);
printf("[V2]");
printf(ENDER);

printf("          <<<<<<<\n");
printf("                                | \n");
printf("-----");

if(P[3].rosu) printf(RED);
else if(P[3].verde) printf(GRN);
printf("[P3]");
printf(ENDER);

printf("-----");

if(P[2].rosu) printf(RED);
else if(P[2].verde) printf(GRN);
printf("[P2]");
printf(ENDER);

printf("----- \n");
printf("                                | \n");
printf("          >>>>>>>          | \n");

if(V[3].rosu) printf(RED);
else if(V[3].verde) printf(GRN);
else if(V[3].galben) printf(YEL);
printf("[V3]");
printf(ENDER);

printf("                                | >>>>>>>\n");
printf("                                | \n");
printf("----- ");

if(V[1].rosu) printf(RED);
else if(V[1].verde) printf(GRN);
else if(V[1].galben) printf(YEL);
printf("[V1]");
printf(ENDER);

printf(" ----- \n");
printf("                                | \ \ / | / \ | \n");
printf("                                | \ \ / | \n");

if(P[1].rosu) printf(RED);
else if(P[1].verde) printf(GRN);
printf("[P1]");

```



```

printf(ENDER);

printf("  /\\"      |\n");
printf("          |  \\/      |  /\\"      |\n");
printf("          |  \\/      |  /\\"      |\n");
}

```

## 6 Testarea aplicației si validarea soluției propuse

În final, am realizat ceea ce ne-am propus, rezultatele obținute fiind cele specificate în cerință. În urma execuției aplicației dezvoltate în temă, va rezulta reprezentarea grafică din **Figure 1**.