**Tema de casă** **Programarea Aplicaţiilor în Timp Real**
Student: Mărăcineanu Tudor-Ioan Sem. 2, 2019-2020
e-mail: tudormaracineanu98@gmail.com

***Monitorizarea activităţii unui magazin***

# 1 Introducere - prezentarea problemei

În cadrul acestei teme am avut ca obiectiv o implementare a unei aplicaţii în timp-real care va monitoriza activitatea unui magazin. În viziunea mea, acest magazin conceptual dispune de trei tipuri de produse, care se afla pe stoc în cantităţi diferite. Aplicaţia doreşte să monitorizeze cantitatea de produse existente pe stoc, urmărind două scenarii: un scenariu de aprovizionare în care cantitatea din produsele dorite va creşte şi un scenariu de cumpărare în care, evident, catitatea de pe stoc din produsele dorite va scădea.

Pentru dezvolatrea acestei aplicaţii am folosit o maşină virtuală cu Linux, unde am creat un program în C folosind un editor de text. Am rulat acest program în linia de comandă folosind comenzile *gcc -o magazin magazin.c -lpthread* şi *./magazin*. Se poate observa din prima comandă de rulare că am folosit biblioteca pthread specifică programării în timp real. Pe lângă această bibliotecă am mai folosit şi biblioteca semaphore pentru a putea folosi semafoare pentru sincronizarea taskurilor necesare de aprovizionare, cumpărare şi monitorizare.

## 1.1 Definire problemă

Să se implementeze o aplicaţie formată din 3 taskuri:

- task aprovizionare - acest task va folosi două semafoare şi va comunica cu utilizatorul în linia de comandă astfel încât să adauge în stoc cantiatea de produse dorită
- task cumpărare - analog taskului de aprovizionare, acest task va folosi la rândul lui două semafoare şi va comunica cu utilizatorul în linia de comandă pentru a extrage din stoc cantitatea de produse dorită cu condiţia ca această cantitate să fie disponibilă pe stoc
- task monitorizare - acest task va face comunicarea dintre cele două taskuri menţionate mai sus

# 2 Analiza problemei

Cele două cerinţe pe care trebuie să le îndeplinească aplicaţia sunt de a permite unui utilizator să facă o aprovizionare, respectiv o achiziţe. Pe lângă aceste două cerinţe de bază, aplicaţia este responsabilă şi de atenţionarea utilizatorului în cazul în care nu poate cumpăra cantitatea de produse dorită şi să îi comunice stocul disponibil. Pentru ca aplicaţia să funcţioneze corect ar fi ideal ca utilizatorul să efectueze mai întâi o aprovizionare având în vedere faptul că se porneşte de la un stoc gol, dar acest lucru nu este imperios necesar deoarece aplicaţia va funcţiona corect indiferent, doar că va face imposibilă realizarea unei cumpărari.

# 3 Definirea structurii aplicaţiei

După cum am menţionat şi mai sus, aplicaţia este alcătuită din trei taskuri: aprovizionare, cumpărare şi monitorizare. Cele trei taskuri sunt sincronizate prin intermediul a patru semafoare (SemA, GataA, SemC, GataC) .De asemenea, am folosit un mutex pentru a realiza excluderea celor două taskuri aprovizionare şi cumpărare, iar pentru a limita taskurile la o durată fixă am folosit comanda sleep(6).

# 4 Definirea soluţiei în vederea implementării

În această sectiune se va specifica modul în care se va implementa aplicaţia (e.g. QNX / C vs. Java). Se vor specifica mecanismele de sincronizare, comunicare între taskuri şi de planificare a taskurilor pe condiţie de timp. Soluţia gândită de mine pentru rezolvarea acestei probleme urmăreşte următorul scenariu: taskul de aprovizionare are prioritate, deci va fi primul care se va efectua blocând mutexul; utilizatorul este întrebat dacă doreşte să efectueze o aprovizionare; în caz afirmativ, este întrebat ce produse doreşte să adauge pe stoc şi în ce cantităţi; aceste valori vor fi adăugate la stocul actual (iniţial 0); după efectuarea actualizării stocului mutexul va fi deblocat; în caz negativ, se va debloca mutexul direct şi se va trece la următorul task după expirarea timpului acordat taskului actual; analog taskului aprovizionare, atunci când se trece la taskul cumpărare mutexul este blocat şi utilizatorul este întrebat dacă doreşte să efectueze o cumpărare; în caz afirmativ, utilizatorul este întrebat ce produse doreşte să cumpere şi în ce cantităţi; cantităţile vor fi stocate în nişte variabile pentru a se putea verifca dacă depăşesc sau nu cantităţile de pe stocul actual; dacă are loc o depăşire nu se va realiza cumpărarea şi utilizatorul va fi atenţionat că cantiatea dorită nu se află pe stoc şi se precizează de asemenea care este cantiatea de pe stoc; dacă nu are loc o depăşire se va scădea din stocul actual conform cantităţii cerute; atunci când utilizatorul nu mai doreşte să efectueze nicio cumpărare mutexul este deblocat şi se trece din nou la taskul de aprovizionare; în cazul în care utilizatorul nu doreşte să efectueze nicio cumpărare se va debloca mutexul direct şi se va trece din nou la taskul de aprovizionare după expirarea timpului alocat taskului actual.

## 4.1 Mecanisme utilizate

Aleg mecanismele de sincronizare şi comunicare între taskuri, prezentând organigramele taskurilor (e.g. în Fig. 1).

- aleg 4 semafoare binare, cu val. iniţiale SemA= 1, SemC = 0, GataA=0, GataC=0;
- aleg 3 semafoare generalizate cu val. iniţiale 0, cantiateA=0, cantiateB=0, cantiateC=0;
- aleg un mutex.

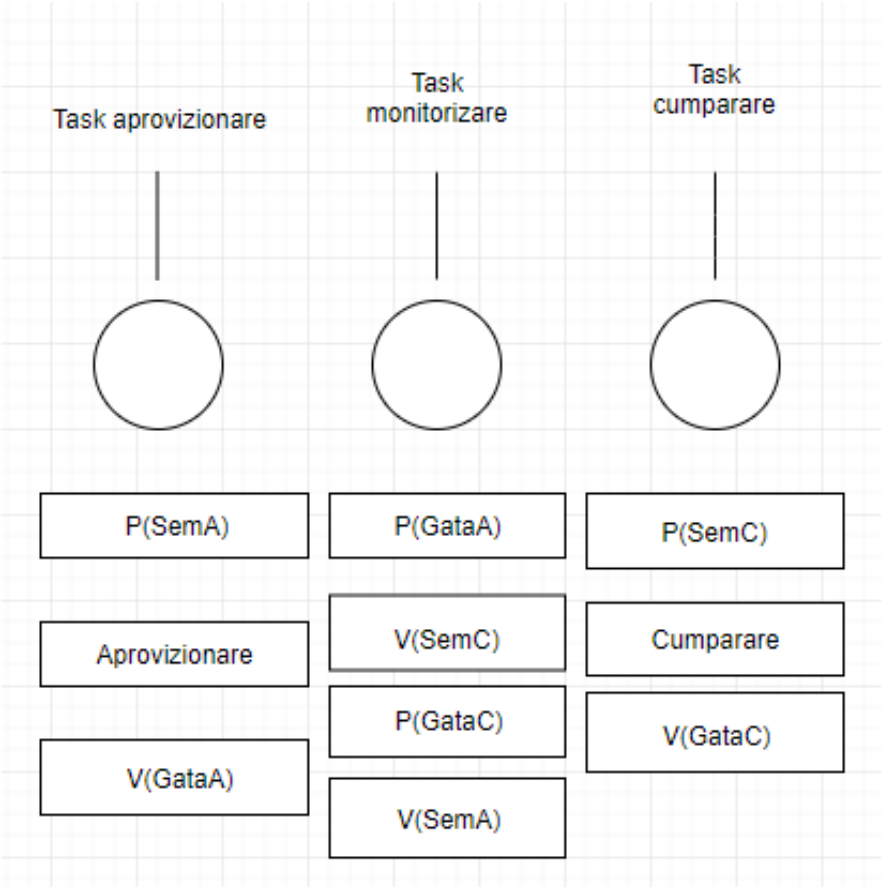


Figure 1: Soluţie implementare - organigrame taskuri

# 5 Implementarea soluţiei

## 5.1 Cod program

***Listing 1:***

```
#include <stdlib.h>
#include <stdio.h>
#include <errno.h>
#include <unistd.h>
#include <pthread.h>
#include <semaphore.h>

void* aprovizionare(void *);
void* cumparare(void *);
void* monitorizare(void *);

pthread_mutex_t mutex = PTHREAD_MUTEX_INITIALIZER;
sem_t SemA, SemC, GataA, GataC;
int cantitateA = 0, cantitateB = 0, cantitateC = 0;

void* (*functie[])(void*) = {aprovizionare, cumparare, monitorizare};
```

```
int main(void){
pthread_t FIR[2];
int i;

sem_init(&SemA, 1, 0);
sem_init(&GataA, 1, 0);
sem_init(&SemC, 1, 0);
sem_init(&GataC, 1, 0);

for(i=0; i<3;i++)
pthread_create(FIR+i,NULL,(void*)(*(functie+i)),NULL);

for(i=0; i<3; i++)
pthread_join(*(FIR+i), NULL);
printf("Main: fireles-au terminat. \n");
pthread_exit(NULL);
}

void* aprovizionare(void *arg){
int s,rasp,prod;
while(1)
{sem_wait(&SemA);
printf("Doriti sa adaugati produse pe stoc (da=1/nu=0)?\n");
fflush(stdin);
scanf("%d", &rasp);
fflush(stdin);
pthread_mutex_lock(&mutex);
while(rasp==1){
printf("Ce produs doriti sa adaugati pe stoc? (1/2/3)\n");
fflush(stdin);
fflush(stdin);
scanf("%d", &prod);
printf("Ce cantitate din produsul %d doriti sa adaugati pe stoc?\n",prod);
fflush(stdin);
fflush(stdin);
scanf("%d", &s);
fflush(stdin);

switch (prod){
case 1: cantitateA+=s;break;
case 2: cantitateB+=s;break;
case 3: cantitateC+=s;break;
default : printf("Va rugam alegeti un produs din cele mentionate?\n");

}
printf("Doriti sa mai adaugati produse pe stoc (da=1/nu=0)?\n");
fflush(stdin);
scanf("%d", &rasp);
}


pthread_mutex_unlock(&mutex);
sem_post(&GataA);
sleep(6);
```

```
}
}

void* cumparare(void *arg){
int s, temp,tempA,tempB,tempC,rasp,prod;

while(1){
sem_wait(&SemC);
printf("Doriti sa cumparati un produs? (da=1/nu=0)\n");
fflush(stdin);
scanf("%d", &rasp);
pthread_mutex_lock(&mutex);
while(rasp==1){
printf("Ce produs doriti sa cumparati(1/2/3)\n");
fflush(stdin);
scanf("%d", &prod);
printf("Ce cantitate din produsul %d doriti sa cumparati?\n",prod);
fflush(stdin);
scanf("%d", &s);

switch (prod){
case 1: {tempA=cantitateA;
           tempA-=s;
           if (tempA < 0)
           printf("Nu puteti cumpara, stoc insuficient(%d).\n", cantitateA);
           else
           cantitateA = tempA;
           break;
           }
case 2: {tempB=cantitateB;
           tempB-=s;
           if (tempB < 0)
           printf("Nu puteti cumpara, stoc insuficient(%d).\n", cantitateB);
           else
           cantitateB = tempB;
           break;
           }
case 3: {tempC=cantitateC;
           tempC-=s;
           if (tempC < 0)
           printf("Nu puteti cumpara, stoc insuficient(%d).\n", cantitateC);
           else
           cantitateC = tempC;
           break;
           }
default : printf("Va rugam alegeti un produs din cele mentionate?\n");
}

printf("Doriti sa mai cumparati un produs? (da=1/nu=0)\n");
fflush(stdin);
scanf("%d", &rasp);
}


pthread_mutex_unlock(&mutex);
```

```
sem_post(&GataC);
sleep(6);
}
}

void* monitorizare(void* arg){
sem_post(&SemA);
while(1){
sem_wait(&GataA);
sem_post(&SemC);
sem_wait(&GataC);
sem_post(&SemA);
}
}
```

## 6 Testarea aplicaţiei si validarea soluţiei propuse

În urma rulării programului din linia de comanda, interacţiunea cu utilizatorul arată ca în Fig.fig:rezultat

```
Doriti sa adaugati produse pe stoc (da=1/nu=0)?
1
Ce produs doriti sa adaugati pe stoc? (1/2/3)
2
Ce cantitate din produsul 2 doriti sa adaugati pe stoc?
10
Doriti sa mai adaugati produse pe stoc (da=1/nu=0)?
1
Ce produs doriti sa adaugati pe stoc? (1/2/3)
3
Ce cantitate din produsul 3 doriti sa adaugati pe stoc?
20
Doriti sa mai adaugati produse pe stoc (da=1/nu=0)?
0
Doriti sa cumparati un produs? (da=1/nu=0)
1
Ce produs doriti sa cumparati(1/2/3)
1
Ce cantitate din produsul 1 doriti sa cumparati?
67
Nu puteti cumpara, stoc insuficient(0).
Doriti sa mai cumparati un produs? (da=1/nu=0)
1
Ce produs doriti sa cumparati(1/2/3)
2
Ce cantitate din produsul 2 doriti sa cumparati?
5
Doriti sa mai cumparati un produs? (da=1/nu=0)
0
Doriti sa adaugati produse pe stoc (da=1/nu=0)?
0
Doriti sa cumparati un produs? (da=1/nu=0)
1
Ce produs doriti sa cumparati(1/2/3)
2
Ce cantitate din produsul 2 doriti sa cumparati?
3
Doriti sa mai cumparati un produs? (da=1/nu=0)
0
Doriti sa adaugati produse pe stoc (da=1/nu=0)?
```

Figure 2: Soluţie implementare - rezultat