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

Echipa: Bălan Mihai, Ciobanu Ştefan, Spînu Andrei
e-mail: mihai.balan1409@stud.acs.upb.ro,
stefan.ciobanu@stud.acs.upb.ro,
andrei.spinu1703@stud.acs.upb.ro

Sem. 2, 2021-2022

Temă de casă PATR

1 Introducere. Definire problemă

În acestă secţiune va descrisă problema care va fi implementată.

- Problema bărbierului constă în existența unei frizerii și a unui bărbier, un scaun de frizerie și N scaune de așteptare. Dacă nu există clienți, frizerul se va odihni, însă dacă un client v-a apărea frizerul se va trezi si îl va tunde. Dacă toate scaunele de așteptare sunt ocupate, urmatorii clienți nu vor mai rămane în frizerie.

2 Analiza problemei

Această secţiune va cuprinde analiza cerinţelor.

Pentru realizarea acestei cerințe vom utiliza 3 taskuri:

- task-ul Barber.
- task-ul Clienti.
- task-ul Coada.

Mod de funcționare: Bărbierul ajunge la salon și începe programul de lucru. Acesta așteaptă, stă și doarme, până când vine un client. Atunci când este adăugat clientul în coadă, se blochează MutexClient. După ce se adăugarea în coadă a fost finalizată, se deblochează MutexClient și se verifică disponibilitatea bărbierului, iar dacă acesta este liber, se blochează MutexBarber. Când bărbierul termină de tuns clientul, MutexBarber se deblochează. În cazul în care sala de așteptare este plină, clientul pleacă și se deblochează MutexClient. Taskul Coada calculează poziția de inserare pentru noul client. Dacă programul de lucru al bărbierului s-a terminat, nu mai sunt acceptați clienți în salon, urmând ca acesta să se închidă după ce bărbierul termină de tuns toți clienții deja existenți în sală.

3 Definirea structurii aplicaţiei

În această secţiune se vor identifica taskurile care compun aplicaţia, în aşa fel încât să se folosească toate conceptele prezentare la curs:

- sincronizare
- excludere mutuală
- planificarea taskurilor pe condiţie de timp

Sincronizarea se va realiza folosind taskul Coada:

- TaskCoada - ține evidența numărului de persoane din camera de așteptare.

Excluderea mutuală:

- MutexBarber - folosit pentru a exclude cazurile în care doi, sau mai mulți, clienți se așează simultan pe scaunul bărbierului.
- MutexClient - folosit pentru a exclude cazurile în care doi, sau mai mulți, clienți intra simultan în frizerie.

Planificarea taskurilor pe condiţie de timp:

- Planificarea temporală se poate folosi pentru a realiza un program al frizerului, întrucât acesta nu lucrează 24/7, trebuie să știe și când să plece acasă.
- O altă utilizarea a planificării temporale o reprezintă timpul pe care îl așteaptă clienții în camera de așteptare, până când să se hotărască să părăsească frizeria.

4 Definirea soluţiei în vederea implementării

În această secțiune se va specifica modul în care se va implementa aplicaţia:

- Linux Debian Xenomai, API C/POSIX
- Arduino şi FreeRTOS

Se vor specifica mecanismele de sincronizare, comunicare între taskuri şi de planificare a taskurilor pe condiţie de timp pe care le-aţi ales din fiecare framework de progranare real-time. De exemplu - explicăm detalii de implementare în funcţie de limbaj::

- în Java – nu există semafoare în limbaj, trebuie definite software
- în C / Linux Debian Xenomai – există semafoare generalizate şi mutex-uri (corespunzător semafoarelor binare pentru sincronizarea firelor de execuţie)
- în FreeRTOS avem semafoare binare şi semafoare generalizate, precum si mutex-uri.

Soluţie de implementare

Aleg mecanismele de sincronizare şi comunicare între taskuri, prezentând organigramele taskurilor (e.g. în Fig. 3).

- aleg 2 mutex-uri, MutexBarber si MutexClient, cu val. iniţiale unlock

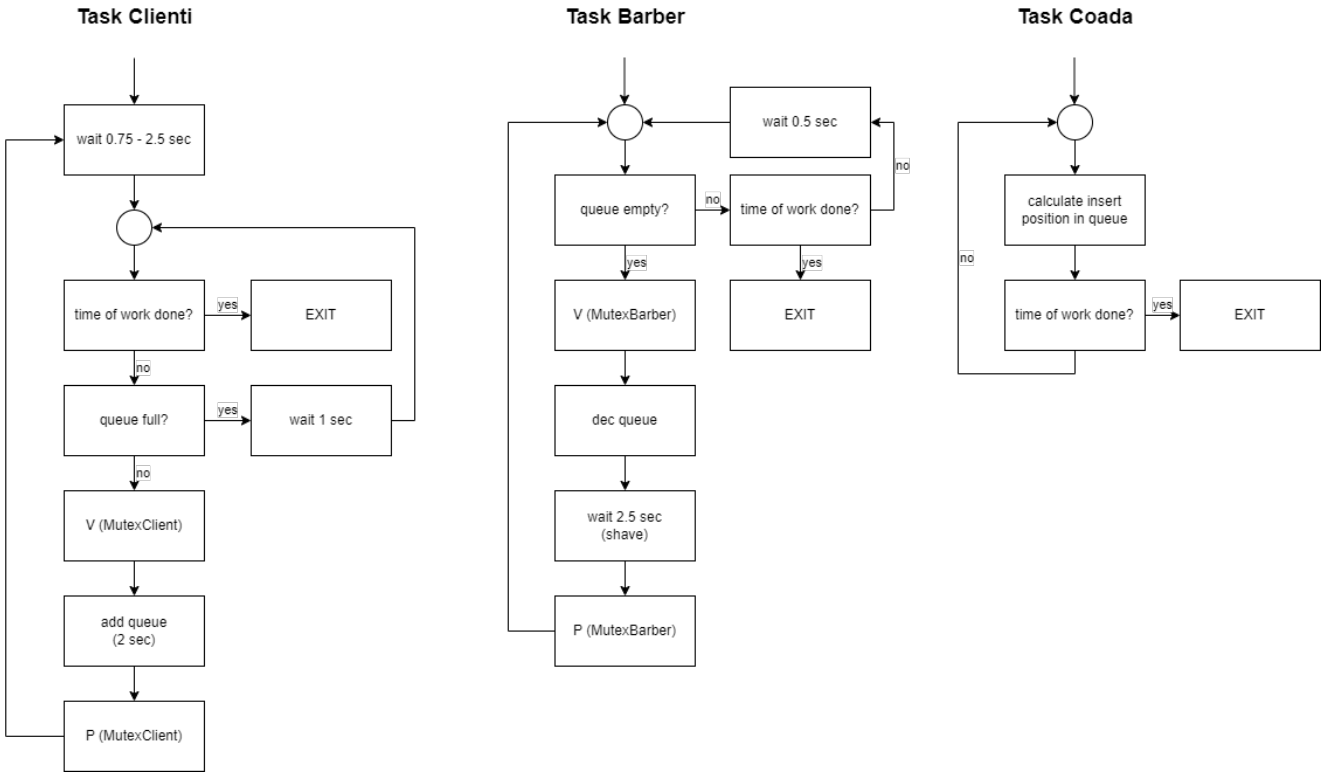


Figure 1: Soluţie implementare - organigrame taskuri

5 Implementarea soluţiei

În această secţiune se va prezenta codul aplicaţiei.

Se va comenta codul pentru a explica modul de lucru.

Codul aplicaţiei în document

***Task Barber**:*

```
while(1) {
if(scaune[0] and xSemaphoreTake(MutexBarber, portMAX_DELAY) == pdTRUE) {
  //Extragere din coada
  for(int i = 0; i < 7; i++) {
    scaune[i] = scaune[i + 1];
  }
  scaune[7] = 0;
  vTaskDelay(200 / portTICK_PERIOD_MS);
  digitalWrite(2, HIGH);
  Serial.println("Barbierul tunde o persoana.");
  //Barbierul tunde 2.5 secunde
  vTaskDelay(2500 / portTICK_PERIOD_MS);
  digitalWrite(2, LOW);
  Serial.println("Barbierul a tuns o persoana.");
  xSemaphoreGive(MutexBarber);
}
```

```
  else if(!scaune[0]){
    //Oprire task
    if(!tunde) {
      Serial.println("Salonul s-a inchis, o zi buna!");
      vTaskDelete(NULL);
    }
    Serial.println("Nu e nimeni in coada!");
    vTaskDelay(500 / portTICK_PERIOD_MS);
  }
}
```

Se verifică dacă este vreun client în coadă și dacă bărbierul nu tunde pe altcineva, caz în care se blochează MutexBarber. Se realizează extragerea unui client din coadă și se așteaptă 200 de milisecunde până când acesta se așează pe scaunul bărbierului. După 2.5 secunde, bărbierul termină de tuns persoana și se deblochează MutexBarber. În cazul în care coada este goală, bărbierul așteaptă clienți, iar dacă programul s-a terminat, taskul este oprit.

Task Clienti:

```
while(1) {
 vTaskDelay(200 / portTICK_PERIOD_MS);
 //Calculare timp ramas pentru a mai primii clienti
 timpClienti = millis();
 if(timpClienti >= 60000) {
   tunde = false;
   Serial.println("Salonul se inchide, nu mai primim clienti!");
   vTaskDelete(NULL);
 }

 //Alegere aleatoare intre 2 clienti in intervalul 0.75 si 2.5 secunde
 if(clientOk) {
   clientOk = false;
   timpRandom = random(750, 2500);
 }

 unsigned long timp = millis();
 if(timp - timpClient >= timpRandom) {
   //Mesaj pentru ca nu se poate adauga in coada un client
   if(pozitieInserare == -1) {
     Serial.println("Coada este plina!");
     digitalWrite(11, HIGH);
     vTaskDelay(1000 / portTICK_PERIOD_MS);
     digitalWrite(11, LOW);
     timpClient = timp;
     clientOk = true;
   }
   else if(pozitieInserare != -1 and xSemaphoreTake(MutexClient, portMAX_DELAY) == pdTRUE){
     //Ocupare scaun un salon de catre client, 1 secunda
     digitalWrite(12, HIGH);
     vTaskDelay(1000 / portTICK_PERIOD_MS);
     digitalWrite(12, LOW);
     scaune[pozitieInserare] = 1;
     Serial.println("A intrat o noua persoana!");
     timpClient = timp;
     clientOk = true;
     xSemaphoreGive(MutexClient);
```

```
    }
  }
}
```

Se verifică dacă s-a terminat programul de lucru al bărbierului, caz în care salonul nu mai primeste clienți, iar bărbierul v-a tunde doar persoanele care sunt în coadă. Se alege aleator, între 750 și 2500 de milisecunde, timpul de intrare al următorului client în salon. Acesta este adăugat în coadă dacă mai este loc liber, altfel o să plece. Timpul în care se face verificarea disponibilității durează o secundă. Dacă mai este loc liber pentru client în coadă, se blochează MutexClient, iar după ce acesta este adăugat, MutexClient este deblocat.

Task Coada:

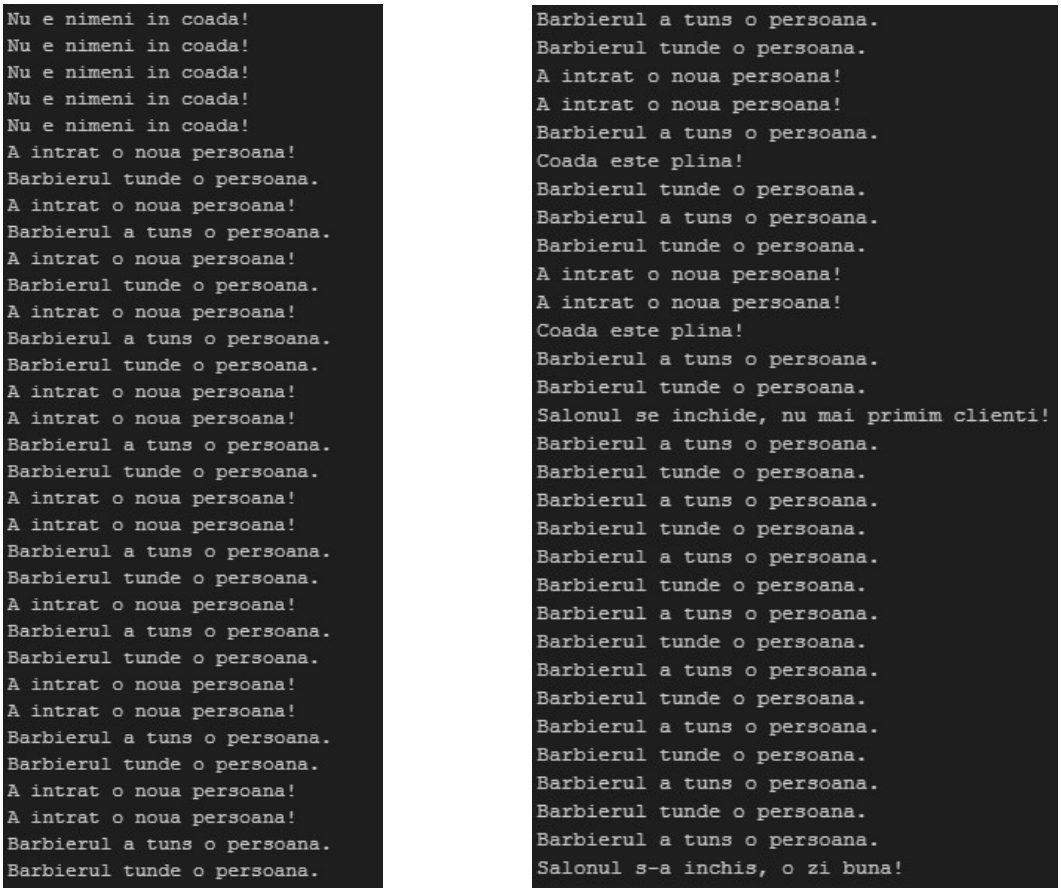
```
while(1) {
 //Calculare pozitie inserare
 int i;
 for(i = 0; i < 8; i++) {
   if(!scaune[i]) {
      break;
   }
 }
 if(i != 8) {
   pozitieInserare = i;
 }
 else {
   pozitieInserare = -1;
 }

 //Setare leduri
 for(int j = 0; j < 8; j++) {
   if(scaune[j]) {
     digitalWrite(j + 3, HIGH);
   }
   else {
     digitalWrite(j + 3, LOW);
   }
 }

 //Oprite task
 if(pozitieInserare == 0 and !tunde and xSemaphoreTake(MutexBarber, portMAX_DELAY) == pdTRUE) {
   vTaskDelete(NULL);
 }
}
```

Se calculează care este următorul scaun liber pe care poate să se așeze clientul nou venit. În cazul în care coada este liberă, programul salonului este terminat și bărbierul nu mai tunde pe nimeni, se poate bloca MutexBarber, taskul se oprește.

6 Testarea aplicaţiei si validarea soluţiei propuse

Această secţiune include comentarii despre modul în care se execută programul. Se obţine ceea ce s-a specificat la pasul 1, în cerinţe?

Programul rulează conform descrierii din cadrul pasului 1, astfel obținându-se gestionarea și sincronizarea task-urilor pentru îndeplinirea cerinței.



(a) (b)

Figure 2: Rezultat execuție aplicație - 8 locuri în coadă

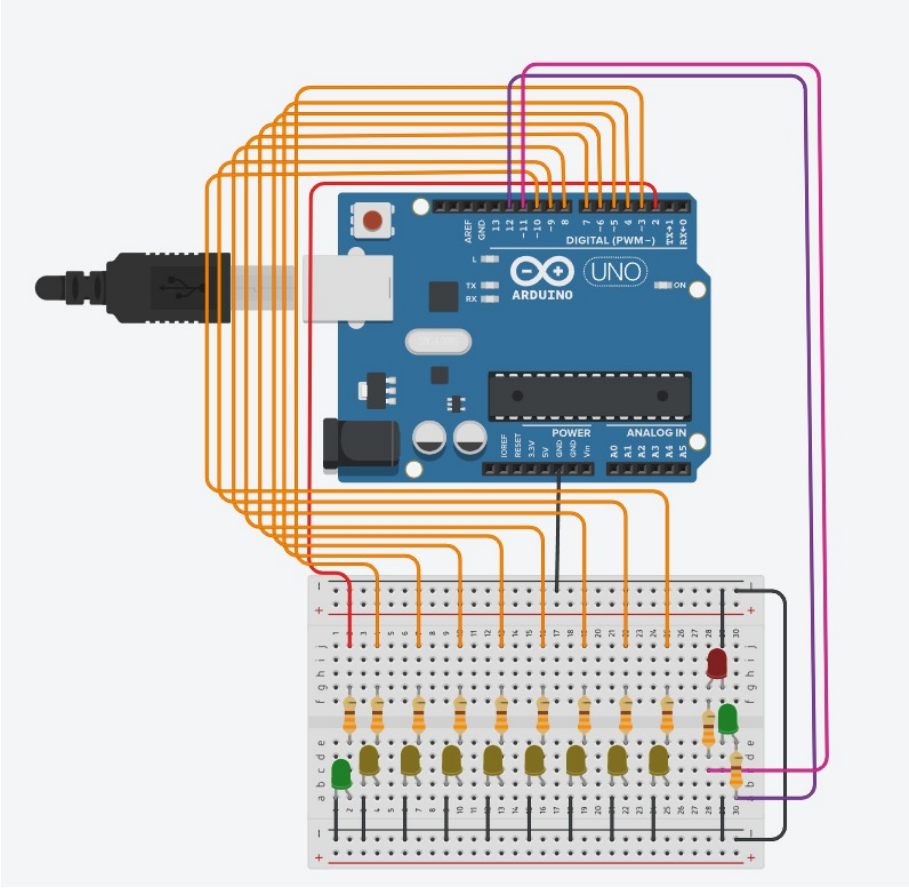


Figure 3: Implementare Simulink