

Tema de casă

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

Programarea Aplicațiilor în Timp Real

Sem. 2, 2021-2022

Temă de casă PATR

1 Introducere. Definire problemă

În această secțiune va descrie 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 apărut frizerul se va trezi și îl va tunde. Dacă toate scaunele de așteptare sunt ocupate, următorii clienți nu vor mai rămâne în frizerie.

2 Analiza problemei

Această secțiune va cuprinde analiza cerințelor.

Pentru realizarea acestei cerințe vom utiliza 2 taskuri:

- task-ul Barber.
- task-ul Customers.

Mod de funcționare: Când bărbierul se trezește de dimineață și ajunge la serviciu începe să-și execute task-ul de frizer, se va bloca în semaforul `sem_customers` deoarece acesta este inițial 0, iar bărbierul se va pune să se mai odihnească până când v-a sosi primul client și v-a incrementa `sem_customers`. Când un client v-a apărut, acesta v-a relua mutexul `mutex_seats`, v-a verifica dacă sunt locuri în sala de așteptare și apoi, în cazul în care frizerul este ocupat, clientul se așează în sala de așteptare la locul lui, dacă frizerul este liber, clientul îl va trezi, iar dacă sala de așteptare este plină v-a părăsi frizeria și v-a ceda mutexul.

3 Definirea structurii aplicației

În această secțiune se vor identifica taskurile care compun aplicația, în așa fel încât să se folosească toate conceptele prezentate la curs:

- [sincronizare](#)
- [excludere mutuală](#)
- [planificarea taskurilor pe condiție de timp](#)

Sincronizarea se va realiza folosind următoarele semafoare:

- sem_customers - semafor generalizat, ține evidența numărului de persoane din camera de așteptare.
- sem_barber - semafor binar, descrie starea bărbierului, liber sau ocupat.

Excluderea mutuală:

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

Planificarea taskurilor pe condiție 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 utilizare 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 soluției în vederea implementării

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

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

Se vor specifica mecanismele de sincronizare, comunicare între taskuri și de planificare a taskurilor pe condiție de timp pe care le-ați ales din fiecare framework de programare real-time. De exemplu - explicăm detalii de implementare în funcție de limbaj::

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

Exemplu Pas 4. Soluție de implementare

Aleg mecanismele de sincronizare și comunicare între taskuri, prezentând organigramele taskurilor (e.g. în Fig. 2).

- aleg 1 semafor generalizat, cu val. inițială sem_customers = 0
- aleg 1 semafor binar, cu val. inițială sem_barber = 0
- aleg 2 mutex-uri, mutex_seats și mutex_enter, cu val. inițiale lock

Pentru crearea organigramelor, se poate folosi aplicația **Cmap**: <https://cmap.ihmc.us/>

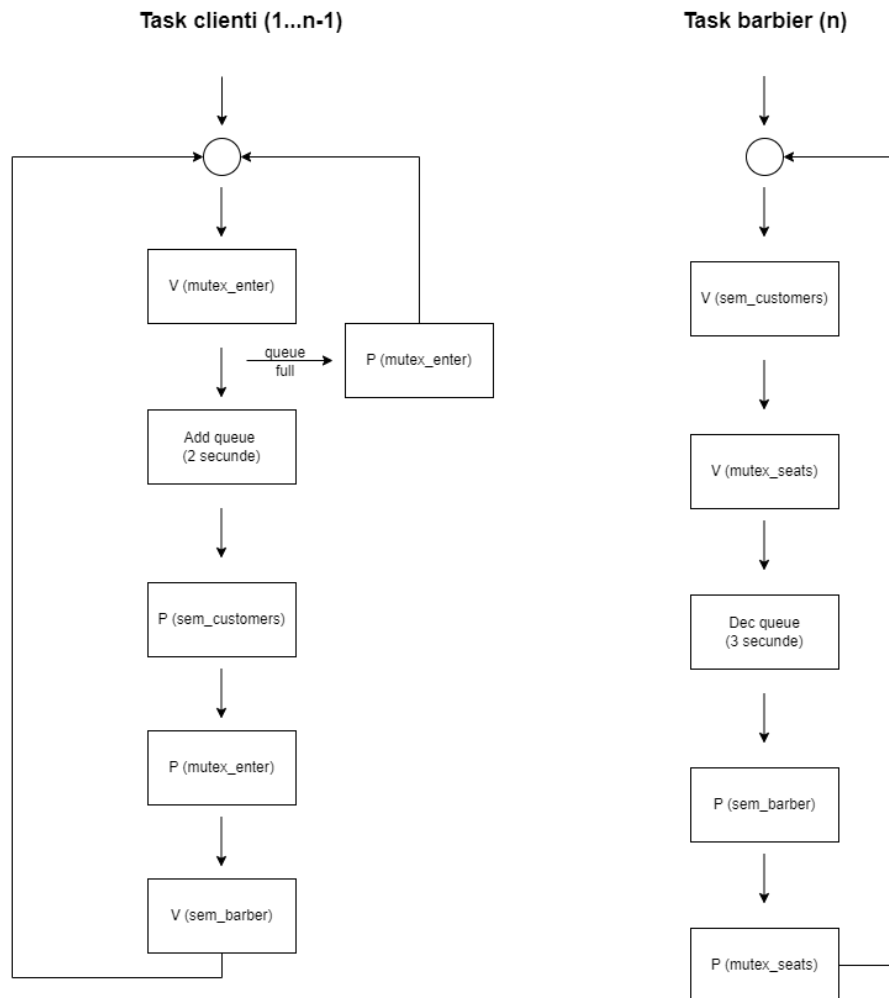


Figure 1: Soluție implementare - organigrame taskuri

5 Implementarea soluției

În această secțiune se va prezenta codul aplicației.

Se va comenta codul pentru a explica modul de lucru.

Codul aplicației în document

Task customers:

```

while (!gata)
{
    gettimeofday(&endCustomer, NULL);
    if(endCustomer.tv_sec - startCustomer.tv_sec >= timeCustomer) {
        pthread_mutex_lock(&mutex_enter);
        if ( !isFull(argum.Queue) && !gata) {
            printf("Ding! Ding! Clientul %s a intrat in sala. - %d\n", argum.nume, argum.ID);
            push(argum.Queue, argum.ID);
            sleep(2);
        }
    }
}

```

```

        sem_post(&sem_customers);
        pthread_mutex_unlock(&mutex_enter);
        sem_wait(&sem_barber);
    } else if(!gata){
        printf("Sala este plina, clientul %s pleaca. - %d\n", argum.nume, argum.ID);
        sleep(2);
        pthread_mutex_unlock(&mutex_enter);
    }
    else {
        pthread_mutex_unlock(&mutex_enter);
    }

    startCustomer = endCustomer;
    timeCustomer = rand() % 4;
}
}

```

Dacă bărbierul încă lucrează, se memorează momentul la care intră clientul în sală, acesta blocând mutex_enter. În cazul în care mai este loc în coadă, clientul se așează la rând incrementând sem_customers și deblocând mutex_enter. În cazul în care bărbierul este liber, sem_barber se decrementează. Dacă sala este plină și programul de lucru încă nu s-a terminat, clientul pleacă și se deblochează mutex_enter. Dacă programul s-a terminat, clientul nu mai intră în sală și se deblochează mutex_enter. La final se resetează timpul după care intră următorul client (între 0 și 3 secunde).

Task barber:

```

while (1)
{
    sem_wait(&sem_customers);
    pthread_mutex_lock(&mutex_seats);
    sleep(3);
    printf("Barbierul a tuns un client - %d.\n", pop(argum.Queue));

    sem_post(&sem_barber);
    pthread_mutex_unlock(&mutex_seats);
    gettimeofday(&end, NULL);
    if(end.tv_sec - start.tv_sec >= 20 && !gata) {
        gata = 1;
        printf("Programul de lucru s-a terminat, nu mai primim clienti!\n");
    }
    if(gata && isEmpty(argum.Queue)) {
        printf("Salonul s-a inchis, o zi buna!\n");
        break;
    }
}
}

```

Se verifică existența clienților în coadă și se decrementează sem_customers. După aceea se blochează mutex_seats (se așează un client pe scaunul bărbierului decrementându-se coada). Se incrementează sem_barber după ce a terminat de tuns clientul și se deblochează mutex_seats. Următoarele linii se referă la programul bărbierului, astfel că după un timp dat, 20 de secunde în cazul nostru, bărbierul nu mai acceptă clienți în coadă, terminând de tuns persoanele care așteaptă deja.

Funcții thread-uri:

```

//Creearea threadurilor customers
for (int i = 0; i < NUM_THREAD; i++) {
    if( pthread_create(&threads[i], NULL, customer, (void *)&arguments[i]) != 0 ) {

```

```

        perror("Pthread create customers.");
        return EXIT_FAILURE;
    }
}

//Creearea threadului barber
if( pthread_create(&threads[NUM_THREAD], NULL, barber, (void *)&arguments[NUM_THREAD]) != 0 ) {
    perror("Pthread create barber.");
    return EXIT_FAILURE;
}

//Unirea threadurilor
for (int i = 0; i <= NUM_THREAD; i++) {
    if( pthread_join(threads[i], NULL) != 0 ) {
        perror("Pthread join.");
        return EXIT_FAILURE;
    }
}

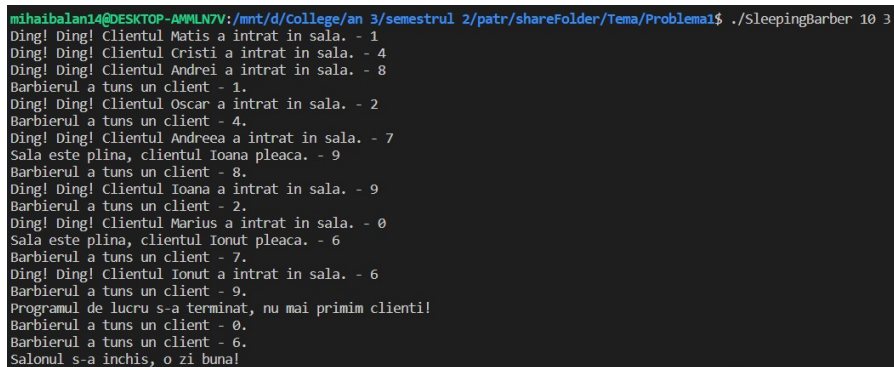
pthread_mutex_destroy(&mutex_seats);
sem_destroy(&sem_customers);
sem_destroy(&sem_barber);

```

6 Testarea aplicației si validarea soluției propuse

Această secțiune include comentarii despre modul în care se execută programul. Se obține ceea ce s-a specificat la pasul 1, în cerințe?

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



```

mihaibalan14@DESKTOP-AMMLN7V:/mnt/d/College/an 3/semestrul 2/patr/shareFolder/Tema/Problema1$ ./SleepingBarber 10 3
Ding! Ding! Clientul Matis a intrat in sala. - 1
Ding! Ding! Clientul Cristi a intrat in sala. - 4
Ding! Ding! Clientul Andrei a intrat in sala. - 8
Barbierul a tuns un client - 1.
Ding! Ding! Clientul Oscar a intrat in sala. - 2
Barbierul a tuns un client - 4.
Ding! Ding! Clientul Andreea a intrat in sala. - 7
Sala este plina, clientul Ioana pleaca. - 9
Barbierul a tuns un client - 8.
Ding! Ding! Clientul Ioana a intrat in sala. - 9
Barbierul a tuns un client - 2.
Ding! Ding! Clientul Marius a intrat in sala. - 0
Sala este plina, clientul Ionut pleaca. - 6
Barbierul a tuns un client - 7.
Ding! Ding! Clientul Ionut a intrat in sala. - 6
Barbierul a tuns un client - 9.
Programul de lucru s-a terminat, nu mai primim clienti!
Barbierul a tuns un client - 0.
Barbierul a tuns un client - 6.
Salonul s-a inchis, o zi buna!

```

Figure 2: Rezultat execuție aplicație - 10 thread-uri și 3 locuri în coadă