Tema de casă PATR

Sistem de navigație al unui avion militar

Echipa: 37

Data: 20 decembrie 2023

Componența echipei și contribuția individuală (în %)

Membrii	A	C	D	PR	CB	e-mail
Andrei Marinache	100%	100%	100%	100%	100%	andreialexandru.marinache@gmail.com

A-analiză problemă și concepere soluție, C- implementare cod, D-editare documentație, PR-"proofreading", CB - contribuție individuală totală ([%])

Membrii echipei declară că lucrarea respectă toate regulile privind onestitatea academică. În caz de nerespectare a acestora tema va fi notată cu ***0(zero) puncte***

Cuprins

1 Introducere. Definire problemă

Problema propusă pentru implementare are ca scop calculul vitezei unui avion militar pe baza accelerației și a unghiurilor de ruliu, tangaj și girație. Aceste date ne sunt oferite de către accelerometrul și giroscopul de la bordul avionului, la un interval de 5 milisecunde pentru accelerații, respectiv 40 milisecunde pentru unghiuri. Viteza calculată va fi afișată pe un monitor.

Prin urmare, va fi necesară o sincronizare a achiziției datelor și paralelizarea calculului vitezei pentru cele 3 dimensiuni.

2 Analiza problemei

În primă fază, datele pe care le primim sunt în 3 dimensiuni, atât pentru accelerații, cât și pentru unghiuri. Viteza, de asemenea, va fi afișată în 3 dimensiuni. Prin urmare, accelerometrul ne va oferi accelerațiile $\ddot{x}, \ddot{y}, \ddot{z}$, sub formă de vector, giroscopul unghiurile θ, ϕ, ψ (ruliu, tangaj și girație) sub formă de vector, de asemenea. Vom afișa vitezele, $\dot{x}, \dot{y}, \dot{z}$.

Pentru implementarea în Xenomai, accelerometrul și giroscopul vor fi simulate prin intermediul citirilor valorilor din fișiere .csv. Fiecare fișier conține 15300 de măsurători obținute din seturi de date de pe Kaggle [2, 1].

Deoarece accelerațiile sunt măsurate în direcțiile axelor sistemul de referință al avionului, primul pas îl reprezintă calculul vitezelor $(\dot{x}, \dot{y}, \dot{z})$ ca atare. Pe urmă, folosind unghiurile furnizate de giroscop, se va aplica o matrice de rotație pentru a obține valorile finale ale vitezelor.

Cum accelerațiile reprezintă derivatele vitezelor, se vor putea calcula vitezele folosind algoritmi de integrare. Algoritmul folosit pentru implementare este Regula lui Simpson. [4].

$$\int_a^b f(x)dx \approx \frac{h}{3}(f(x_0) + 4f(x_1) + 2f(x_2) + ... + 4f(x_{n-1}) + f(x_n))$$

unde

$$h = \frac{b-a}{n}$$

și n - numărul de intervale egal depărtate, n fiind un număr par.

Având în vedere că accelerațiile sunt furnizate la fiecare 5 milisecunde, iar unghiurile la fiecare 40 milisecunde, fiecare interval de forma [0, 40], [40, 80], va fi împărțit în 8 subintervale (din 5 în 5 milisecunde). Astfel, diferența $b - a = 40$ mereu, și cum n nu se schimbă (mereu sunt 8 subintervale), rezultă

$$h = \frac{40}{8} = 5$$

Va fi necesară, totuși, o măsurătoare la t = 0. Se vor presupune aceste condiții inițiale nule, adică avionul pornește din repaus, în linie dreapta (toate unghiurile sunt egale cu 0). Acest algoritm va fi aplicat pe $\dot{x}, \dot{y}$, și $\dot{z}$

Matricea de rotație [3] ce va fi aplicată vectorului vitezelor obținut la pasul anterior este:

$$R = \begin{bmatrix} \cos\phi\cos\theta & \sin\psi\sin\phi\cos\theta - \cos\psi\sin\theta & \cos\psi\sin\phi\cos\theta + \sin\psi\sin\theta \\ \cos\phi\sin\theta & \sin\psi\sin\phi\sin\theta + \cos\psi\cos\theta & \cos\psi\sin\phi\sin\theta - \sin\psi\cos\theta \\ -\sin\phi & \sin\psi\cos\phi & \cos\psi\cos\phi \end{bmatrix}$$

Așadar, algoritmul de rezolvare al problemei se poate rezuma, în mare, la:

- Calculul vectorului viteză;
  - Calcul pentru fiecare dimensiune în parte;
- Calcul matrice rotație;
- Aplicare matrice rotație;
- Afișare viteză.

Totuși, câțiva din acești pași pot fi grupați pentru eficientizare. Prin urmare, calculul matricei de rotație și aplicarea acesteia vor fi simultan realizate. De asemenea, la calculul vectorului viteză, suma se va actualiza la fiecare măsurătoare în locul stocării tuturor măsurătorilor în memorie.

3 Aplicația. Structura și soluția de implementare propusă

3.1 Definirea structurii aplicației

Aplicația implementată funcționează pe baza de timere și sigevent-uri. La bază avem un while loop care găzduiește rularea a 3 timere:

- de 5 milisecunde pentru citirea datelor din accelerometru și pentru calculul integrale;
- de 40 milisecunde pentru finalizarea calcului integralelor și pentru calculul vitezelor;
- de o secundă pentru aplicare matricei de rotație și afișarea vitezelor în 3 dimensiuni dar și a modulului vectorului;

Fiecare timer apelează la finalizare o funcție handler ce, în funcție de semnalul generat de timer, execută diverse instrucțiuni.

Semnalul timerului de 5 milisecunde produce generarea a 4 threaduri:

- Primul pentru citirea datelor de la accelerometru;
- Următoarele 3 pentru calculul paralel al integralelor pentru cele 3 accelerații furnizate;

Un semafor mutex blocheaza accesul ultimelor 3 threaduri la variabilele în care se stochează măsurătorile de la senzor pentru a asigura existența datelor înainte de începerea calculului.

Semnalul timerului de 40 milisecunde produce generarea a 5 threaduri:

- Primele două pentru citirea datelor de la accelerometru (a 8-a măsurătoare) și de la giroscop. Același mecanism mutex de mai sus este implementat și aici;
- Următoarele 3 threaduri pentru calculul paralel al vitezelor înainte de aplicarea matricei de rotație;

Ultimul timer, cel de o secundă, produce generarea a unui singur thread: Acesta aplică matricea de rotație pe vectorul obținut precedent, calculează modulul vectorului și afișează atât vectorul, cât și modulul.

3.2 Definirea soluției în vederea implementării

3.2.1 Linux-Xenomai / POSIX

Planul de rulare al aplicației este următorul:

- Pornirea aplicației și inițializarea tuturor variabilelor globale pentru taskuri, timere, sigevent, fișierele pentru obținerea datelor de la senzori, variabile de calcul;
- Odată terminate aceste pregătiri, timerele se armează și intrăm în while loop-ul principal. Momentan, având date limitate, aplicația este setată să ruleze pentru 80 de secunde;
- Timerul de 5 milisecunde se va declanșa timp de 7 ori, functia handler generând threadurile pentru citire și calculul integralelor;
  - Se blochează semaforul mutex. Se citesc datele de la senzor. Se deblochează semaforul;
  - Se blochează semaforul mutex doar in primul thread. Cele 3 threaduri nu actionează asupra unor variabile comune, deci doar unul e necesar să îl blocheze. Se efectuează calculele necesare. Se deblochează semaforul de către ultimul thread.
  - Deoarece la fiecare 40 milisecunde se vor activa atât timerul pentru accelerometru, cât și cel de giroscop, generarea threadurilor este restricționată la doar 7 operații consecutive (citire + calcul);
- Timerul de 40 milisecunde se va declanșa. Cel de 5 milisecunde este restricționat.
  - Se citesc pentru a 8-a oară datele de la accelerometru, de data aceasta doar în cadrul declanșării timerului de 40 milisecunde;
  - Se citesc datele pentru unghiuri. Același mecanism mutex este implementat și aici, pentru ambele citiri;
  - Blocarea semaforului mutex și efectuarea calculului în paralel al celor 3 viteze. Calculul integralei se definitivează și suma este inițiată cu măsurătoarea x_0 pentru următorul interval (adică măsurătoarea numărul 8 din intervalul precedent). Deblocarea semaforului la final;
- Perechea Timer 40 milisecunde + timer 5 milisecunde se va efectua de 25 de ori pe secundă;
- Timerul de o secundă se va declanșa, producând generarea ultimului thread;
  - În cadrul acestui thread, se blochează semaforul, se aplica matricea de rotație, se calculează modulul vitezei, se afișează și se deblochează semaforul.
- La finalizarea celor 80 de secunde de rulare a aplicației, timerele se vor dezactiva iar fișierele se vor închide. Aplicația se va închide. Diagramele vor fi la final, în Anexă.

3.3 Implementarea soluției

3.3.1 Linux- Xenomai

Threadurile asociate timerului de 5 milisecunde:

```
void *read_acc(void *ptr)
{
    pthread_mutex_lock(&semSync);
    fscanf((FILE *)ptr, "%lf,%lf,%lf\n", &acc_x, &acc_y, &acc_z);
    pthread_mutex_unlock(&semSync);
    return NULL;
}
void *calc_acc_x()
{
    pthread_mutex_lock(&semSync);
    if (parity_x == 0)
    {
        sum_x = sum_x + 4 * acc_x;
        parity_x = 1;
    }
    else
    {
        sum_x = sum_x + 2 * acc_x;
        parity_x = 0;
    }

    return NULL;
}

void *calc_acc_y()
{
    if (parity_y == 0)
    {
        sum_y = sum_y + 4 * acc_y;
        parity_y = 1;
    }
    else
    {
        sum_y = sum_y + 2 * acc_y;
        parity_y = 0;
    }

    return NULL;
}

void *calc_acc_z()
{
    if (parity_z == 0)
    {
        sum_z = sum_z + 4 * acc_z;
        parity_z = 1;
    }
    else
    {
```

```
        sum_z = sum_z + 2 * acc_z;
        parity_z = 0;
    }

    pthread_mutex_unlock(&semSync);
    return NULL;
}
```

Threadurile asocitate timerului de 40 milisecunde:

```
void *read_acc(void *ptr)
{
    pthread_mutex_lock(&semSync);
    fscanf((FILE *)ptr, "%lf,%lf,%lf\n", &acc_x, &acc_y, &acc_z);
    pthread_mutex_unlock(&semSync);
    return NULL;
}

// thread pt citire unghiuri + ultima masuratoare de acceleratii
void *read_ang(void *ptr)
{
    pthread_mutex_lock(&semSync);
    fscanf((FILE *)ptr, "%lf,%lf,%lf\n", &theta, &phi, &psi);
    pthread_mutex_unlock(&semSync);
    return NULL;
}
void *calc_vel_x()
{
    pthread_mutex_lock(&semSync);
    sum_x = sum_x + acc_x;
    vel_x = integral_const * sum_x;
    sum_x = acc_x;

    return NULL;
}

void *calc_vel_y()
{
    sum_y = sum_y + acc_y;
    vel_y = integral_const * sum_y;
    sum_y = acc_y;

    return NULL;
}

void *calc_vel_z()
{
    sum_z = sum_z + acc_z;
    vel_z = integral_const * sum_z;
    sum_z = acc_z;

    pthread_mutex_unlock(&semSync);

    return NULL;
}
```

Signal handler

```
void handler_semnale(union sigval semnal)
{
    int semnal_no = semnal.sival_int; // obtinem id-ul semnalului

    switch (semnal_no)
    {
    default: // do nothing
        break;
    case SIGACC: // citire accelerometru + calcul suma pentru integrala
    {
        if (op_num != 1)
        {
            pthread_create(&thread_read_acc, NULL, read_acc, (void *)acc_ptr);

            pthread_create(&thread_acc_x, NULL, calc_acc_x, NULL);

            pthread_create(&thread_acc_y, NULL, calc_acc_y, NULL);

            pthread_create(&thread_acc_z, NULL, calc_acc_z, NULL);

            op_num = op_num - 1;
        }

        break;
    }
    case SIGANG: // citire giroscop + calcul viteze
    {

        pthread_create(&thread_read_acc, NULL, read_acc, (void *)acc_ptr);

        pthread_create(&thread_read_ang, NULL, read_ang, (void *)ang_ptr);

        pthread_create(&thread_vel_x, NULL, calc_vel_x, NULL);

        pthread_create(&thread_vel_y, NULL, calc_vel_y, NULL);

        pthread_create(&thread_vel_z, NULL, calc_vel_z, NULL);

        parity_x = 0;
        parity_y = 0;
        parity_z = 0;
        op_num = 8;

        break;
    }
    case SIGPRN: // afisam vitezele
    {
        pthread_create(&thread_print, NULL, print_vel_all, NULL);
        break;
    }
    }
}
```

Thread timer o secundă:

```
void *print_vel_all()
{
    pthread_mutex_lock(&semSync);
    Rvel_x = cos(phi) * cos(theta) * vel_x + (sin(psi) * sin(phi) * cos(theta) -
        cos(psi) * sin(theta)) * vel_y + (cos(psi) * sin(phi) * cos(theta) + sin(psi)
        * sin(theta)) * vel_z;
    Rvel_y = cos(phi) * sin(theta) * vel_x + (sin(psi) * sin(phi) * sin(theta) +
        cos(psi) * cos(theta)) * vel_y + (cos(psi) * sin(phi) * sin(theta) - sin(psi)
        * cos(theta)) * vel_z;
    Rvel_z = -sin(phi) * vel_x + sin(psi) * cos(phi) * vel_y + cos(psi) * cos(phi) *
        vel_z;
    Rvel_rez = sqrt(Rvel_x * Rvel_x + Rvel_y * Rvel_y + Rvel_z * Rvel_z);
    printf("Vel_x = %.3lf - Vel_y = %.3lf - Vel_z = %.3lf\nVel_modul =
        %.3lf\n---------------\n", Rvel_x, Rvel_y, Rvel_z, Rvel_rez);
    pthread_mutex_unlock(&semSync);
    return NULL;
}
```

Listing - main:

```
int main()
{

    // sigevent logic pt timere
    memset(&sig_read_acc, 0, sizeof(struct sigevent));  // acelerometru
    memset(&sig_read_ang, 0, sizeof(struct sigevent));  // giroscop
    memset(&sig_print_vel, 0, sizeof(struct sigevent)); // afisare

    // sigevent pt accelerometru
    sig_read_acc.sigev_notify = SIGEV_THREAD;
    sig_read_acc.sigev_notify_function = &handler_semnale;
    sig_read_acc.sigev_value.sival_int = SIGACC;

    // sigevent pt giroscop
    sig_read_ang.sigev_notify = SIGEV_THREAD;
    sig_read_ang.sigev_notify_function = &handler_semnale;
    sig_read_ang.sigev_value.sival_int = SIGANG;

    // sigevent pt citire
    sig_print_vel.sigev_notify = SIGEV_THREAD;
    sig_print_vel.sigev_notify_function = &handler_semnale;
    sig_print_vel.sigev_value.sival_int = SIGPRN;

    // timer logic
    memset(&ang_timer, 0, sizeof(struct itimerspec));   // giroscop
    memset(&acc_timer, 0, sizeof(struct itimerspec));   // accelerometru
    memset(&print_timer, 0, sizeof(struct itimerspec)); // afisare

    // cream timerele
    timer_create(CLOCK_REALTIME, &sig_read_acc, &tacc_id);    // accelerometru
    timer_create(CLOCK_REALTIME, &sig_read_ang, &tang_id);    // giroscop
    timer_create(CLOCK_REALTIME, &sig_print_vel, &tprint_id); // afisare

    // acc_timer la fiecare 5ms
    acc_timer.it_interval.tv_sec = 0;
```

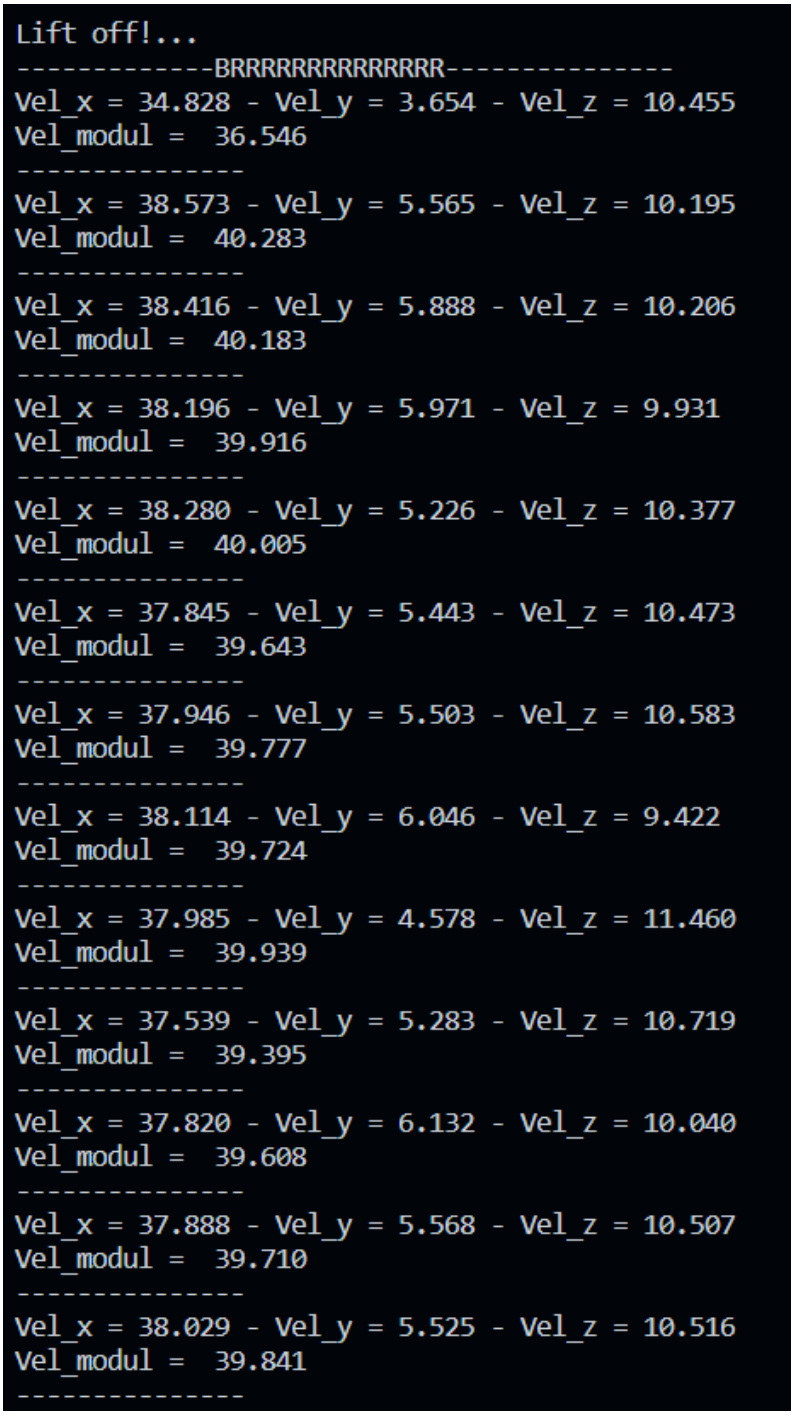
```
    acc_timer.it_interval.tv_nsec = 5000000;
    acc_timer.it_value.tv_sec = 0;
    acc_timer.it_value.tv_nsec = 5000000;

    // ang_timer la fiecare 40ms
    ang_timer.it_interval.tv_sec = 0;
    ang_timer.it_interval.tv_nsec = 40000000;
    ang_timer.it_value.tv_sec = 0;
    ang_timer.it_value.tv_nsec = 40000000;

    // print timer la fiecare 1s
    print_timer.it_interval.tv_nsec = 0;
    print_timer.it_interval.tv_sec = 1;
    print_timer.it_value.tv_nsec = 0;
    print_timer.it_value.tv_sec = 1;

    // in cazul in care "senzorii" nostri esential lipsesc, nu putem porni programul
    if ((acc_ptr = fopen("acc_data", "r")) == NULL || (ang_ptr = fopen("ang_data", "r")) == NULL)
    {
        perror("Nu avem accelerometru sau giroscop!\n");
        exit(1);
    }
    else
    {

        // da-i nicule da-i tare
        printf("Lift off!...\n");
        // tureaza motorul tatiiii
        printf("--------------BRRRRRRRRRRRRRR---------------\n");

        // initializam timerele
        timer_settime(tacc_id, 0, &acc_timer, NULL);
        timer_settime(tang_id, 0, &ang_timer, NULL);
        timer_settime(tprint_id, 0, &print_timer, NULL);

        while (1) // main loop
        {
            sleep(80); // rulam doar pt 80 secunde momentan

            // stergem timerele si inchidem "senzorii" la final
            timer_delete(tacc_id);
            timer_delete(tang_id);
            timer_delete(tprint_id);
            fclose(acc_ptr);
            fclose(ang_ptr);
            printf("Avion cu motor!\n");
            exit(0);
        }
    }

    return 0;
}
```

4 Testarea aplicației și validarea soluției propuse

Aplicația a fost testată pe parcursul dezvoltării ei. La început, doar afișarea brută a valorilor la fiecare citire, atât pentru accelerometru cât și pentru giroscop. Pe urmă au fost implementate timerele, pentru a organiza citirea și afișarea. În continuare, setarea semnalelor timerelor și asigurarea că funcționează. Implementarea relațiilor matematice, testarea corectitudinii acestor relații și, la final, implementarea threadurilor, acestea fiind cele mai dificile (multe seg fault-uri :( )

Aplicația rulează conform așteptărilor. Vitezele sunt furnizate în fiecare secundă, atât sub formă de vector în 3 dimensiuni, cât și ca modul al rezultantei totale a vitezei.

4.1 Linux- Xenomai



Test 1

Figura 4.1: Teste - Xenomai

5 Bibliografie

Referințe

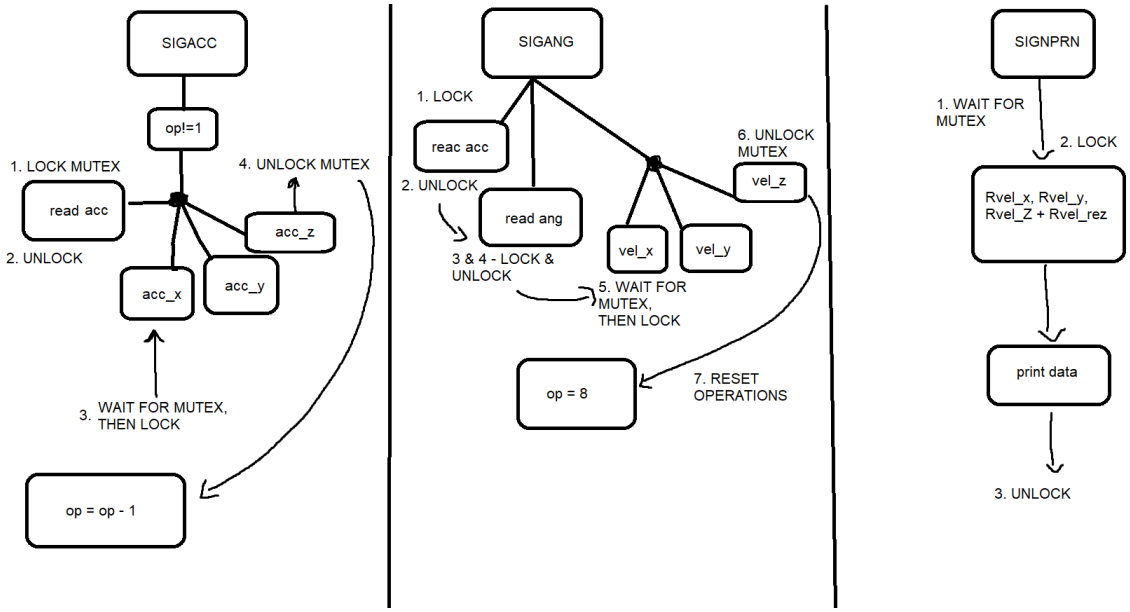
[1] CHUKA J. UZO, *1-million Instances - Vehicle Orientation Dataset*, [Online; accessed 20-December-2023], URL: https://www.kaggle.com/datasets/chukajuzo/high-resolution-imu-car-trajectory-dataset?rvi=1.

[2] PUSHKAR AMBASTHA, *Accelerometer. Prediction of Motor failure Time*, [Online; accessed 20-December-2023], URL: https://www.kaggle.com/datasets/pushkar007/accelerometer/data.

[3] Gregory G Slabaugh, "Computing Euler angles from a rotation matrix", în *Retrieved on August* 6.2000 (1999), pp. 39–40.

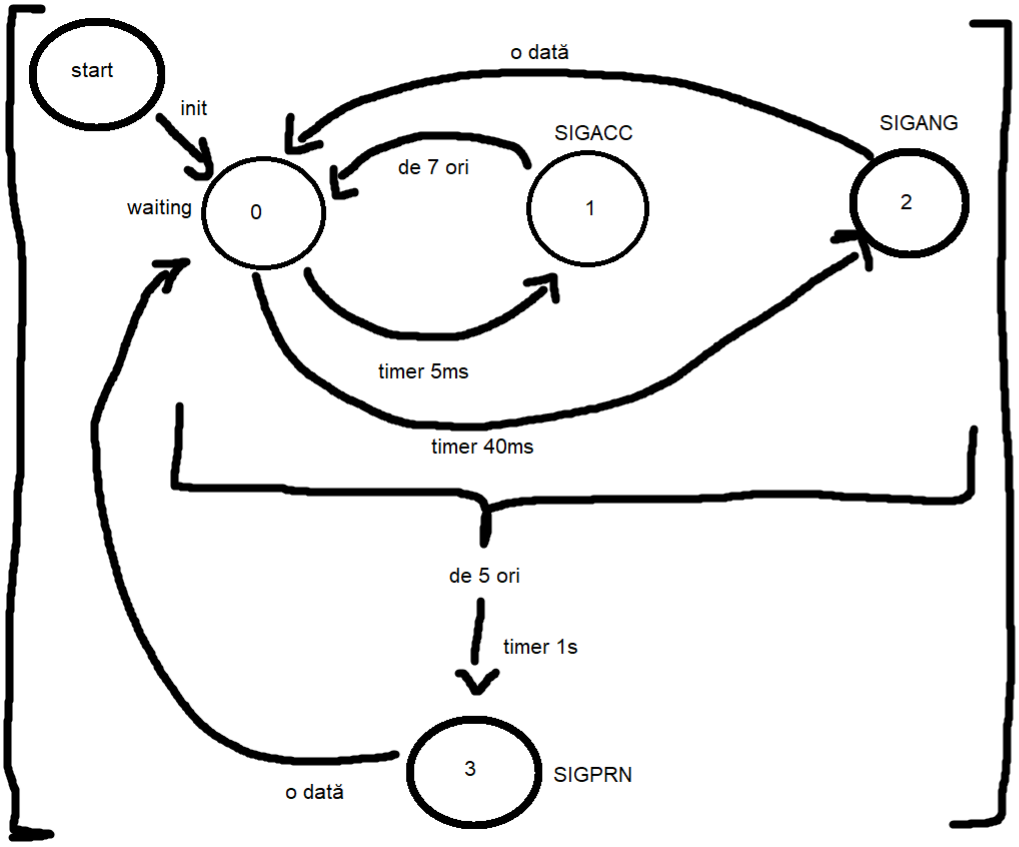
[4] StudySmarter, *Integration in C*, [Online; accessed 20-December-2023], URL: https://www.studysmarter.co.uk/explanations/computer-science/computer-programming/integration-in-c/.

A Diagrama 1



Schema organizare taskuri/timere

B Diagrama 2



Schema organizare aplicație