Ministerul Educației și Cercetării

Universitatea Tehnică a Moldovei

Facultatea Calculatoare, Informatică și Microelectronică

Departamentul Ingineria Software și Automatică

Raport

Curs: Internetul lucrurilor

Tema: Sisteme de Operare - Secventiale. Sisteme de Operare Preemptive - FreeRTOS

A elaborat: st.gr.SI-211 Chirița Stanislav

A verificat: Asist. Univ.Astafi Valentina

Chișinău 2024

Lucrare de laborator nr. 2.1

Partea nr 2.1 : Cod Secvențial

Definire problema

Aplicația va rula minim 3 task-uri printre care

1. Button Led - Schimbare stare LED la detecția unei apăsări pe buton.
2. un al doilea Led Intermitent în faza în care LED-ul de la primul Task e stins
3. Incrementare/decrementare valoare a unei variabile la apăsarea a doua butoane care va reprezenta numărul de recurențe/timp în care ledul de la al doilea task se va afla într-o stare
4. Task-ul de Idle se va utiliza pentru afișarea stărilor din program, cum ar fi, afișare stare LED, și afișare mesaj la detecția apăsării butoanelor, o implementare fiind ca la apăsarea butonului sa se seteze o variabila, iar la afișare mesaj - resetare, implementând mecanismul provider/consumer.

Efectuarea laboratorului este pe 2 părți: codul creat secvențial și altul cu FreeRTOS.

Obiective:

- schimbarea stării unui LED la detecția unei apăsări a unui buton;
- controlarea unui al doilea LED în faza în care primul LED este stins;
- incrementarea sau decrementarea unei variabile la apăsarea a două butoane, care va reprezenta numărul de recurențe/timp în care al doilea LED se va afla într-o anumită stare;
- utilizarea unui task Idle pentru afișarea stărilor din program, cum ar fi afișarea stării LED-ului și afișarea unui mesaj la detectarea apăsării butoanelor.

Introducere

Problema abordată în acest context este legată de dezvoltarea unei aplicații pe platforma Arduino care implică interacțiunea cu LED-uri și butoane pentru a crea diferite funcționalități. Această aplicație se dorește a fi implementată utilizând FreeRTOS pentru a gestiona concurența și planificarea eficientă a task-urilor.

Această problemă este strâns legată de domeniul sistemelor înglobate, deoarece se dezvoltă o aplicație pe o platformă înglobată (Arduino) pentru controlul unor componente hardware (LED-uri și butoane). Aceste sisteme înglobate sunt adesea utilizate într-o gamă largă de dispozitive, de la electrocasnice până la automobile și echipamente medicale.

Dezvoltarea aplicației pentru Arduino implică scrierea de cod pentru o platformă înglobată specifică, ceea ce necesită cunoștințe solide în programare pentru platforme integrate, în special în limbajul C/C++.

Utilizarea FreeRTOS pentru planificarea și gestionarea task-urilor aduce în discuție domeniul sistemelor de operare în timp real. Acest domeniu se concentrează pe dezvoltarea și utilizarea sistemelor de operare specializate pentru aplicații în timp real.

În sistemele înglobate, este crucial să se obțină performanțe optime și să se gestioneze eficient resursele limitate, cum ar fi memoria și puterea de procesare. Problema constă în a crea o aplicație care să fie eficientă în utilizarea acestor resurse.

Implementarea funcționalităților concurente poate aduce provocări legate de sincronizarea corectă a task-urilor și gestionarea evenimentelor în mod corespunzător. Acest lucru este esențial pentru evitarea situațiilor de concurență și pentru asigurarea funcționării corecte a aplicației.

Crearea unei arhitecturi flexibile, care să permită adăugarea și modificarea funcționalităților în viitor, este o altă problemă importantă. Este esențial să se proiecteze aplicația astfel încât să fie ușor de extins și adaptat la cerințele viitoare.

Asigurarea securității este o problemă critică în domeniul sistemelor înglobate. Este necesar să se acorde atenție securității aplicației pentru a evita vulnerabilități și accesul neautorizat la sistem.

Aceste probleme și provocări sunt relevante pentru domeniul dezvoltării de aplicații pentru platforme înglobate, în special în contextul sistemelor de operare în timp real și al interacțiunii cu componente hardware, cum ar fi LED-uri și butoanele, pentru a crea aplicații funcționale și eficiente.

FreeRTOS (Free Real-Time Operating System) este un sistem de operare de timp real open-source, conceput pentru a oferi o platformă ușor de utilizat și eficientă pentru dezvoltarea aplicațiilor încorporate în timp real. A fost creat de Richard Barry și este dezvoltat și susținut de către comunitatea sa.

Principalele caracteristici și obiective ale FreeRTOS includ:

- Timp real (real-time): FreeRTOS este proiectat pentru a răspunde și a executa operații în timp real, ceea ce înseamnă că poate gestiona sarcini și evenimente într-un mod predictibil și la momentul potrivit.
- Multi-platformă: Este proiectat să funcționeze pe o varietate de microcontrolere și microprocesoare, făcându-l versatil și ușor de portat pe diferite arhitecturi hardware.
- Eficiență: FreeRTOS are un nucleu compact și are un consum redus de resurse, inclusiv consum redus de memorie RAM și ROM, ceea ce îl face potrivit pentru dispozitivele încorporate cu resurse limitate.
- Planificare bazată pe priorități (preemptive priority-based scheduling): Sarcinile sunt planificate în funcție de prioritățile lor, iar sistemul de operare asigură că sarcinile cu prioritate mai mare sunt executate înaintea celor cu prioritate mai mică.

- Gestionarea sarcinilor (task management): Permite crearea, ștergerea și coordonarea sarcinilor și oferă mecanisme de sincronizare și comunicare între ele.
- Mecanisme de sincronizare: Oferă semafoare, cozi, mutex-uri și alte mecanisme de sincronizare pentru a permite comunicarea și coordonarea între diferite sarcini.
- Licență open-source: FreeRTOS este licențiat sub licența MIT, permițând utilizatorilor să îl utilizeze, să îl modifice și să îl distribuie în conformitate cu această licență.

Datorită caracteristicilor sale și naturii open-source, FreeRTOS este utilizat extensiv în domeniul dispozitivelor încorporate, precum roboți, aparate electrocasnice, automobile, echipamente medicale și multe altele, pentru a gestiona sarcini în timp real și a asigura un comportament fiabil al sistemului.

Elemente:

Pentru efectuarea sarcinii propuse vor fi necesare următoarele elemente pentru construirea circuitului:

1. Placa Arduino;
2. Doua led-uri, unul roșu și unul galben;
3. Cinci rezistoare;
4. Trei butoane;
5. Fire de conexiune.

Pașii de executare a creării sistemului:

1. Led-urile se conectează în modul următor: cu catodul ce e destinat pentru semnul minus la secțiunea GND – ground, iar celălalt capăt mai lung e anodul ce reprezintă + și e conectat la un număr de pe placa Arduino prin fire;
2. Butoanele se conectează în modul următor: numărul ce raspunde de puton este plasat în perte dreaptă de sus a butonului, iar din partea de jos în stânga legătura la 5V, iar la dreapta la GND.
3. De pe placa Arduino de la GND se duce la secțiune minus(-) un fir de conexiune, apoi de pe aceast de la 5V se duce la secțiune (+);
4. Specificare că la conectare placa breadBoard este împărțită în 2 jumătăți, adică daca 5V și GND e conectat la doar o jumătate, la cealaltă nu va ajunge acestea.
5. Rezistoarele și firele de conexiune și, de asemenea, întreg sistemul se vede în figura 1.

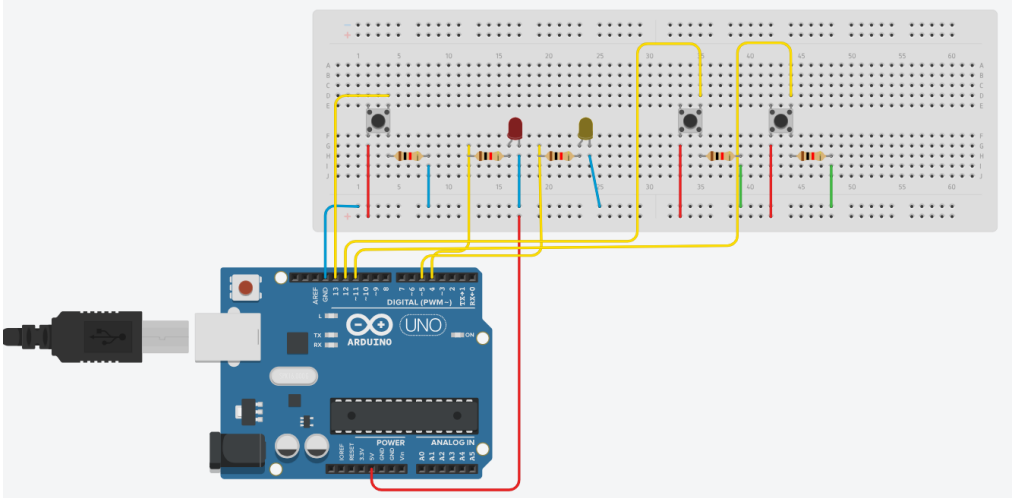

Figura 1 – Schema sarcinii

Codul partea 1. Secvențial

Codul scris este secvențial, adică este scris într-o manieră liniară și nu utilizează un sistem de operare în timp real (RTOS). Într-un sistem secvențial, instrucțiunile sunt executate în ordinea în care sunt scrise în funcții și în loop(). Nu există concurență între mai multe task-uri, iar execuția este deterministă.

În continuare urmează analiza codului secvențial. Principala logică a codului din Anexa 1 este:

- Metoda setup(): Inițializează pini ai dispozitivului pentru a fi utilizați ca ieșiri sau intrări. Inițializează portul serial pentru comunicarea cu computerul sau alte dispozitive.
- Metoda taskLedWithButton(): Citeste starea butonului și o compară cu starea anterioară a butonului pentru a detecta apăsările de buton. Schimbă starea LED-ului în funcție de apăsarea butonului.
- Metoda taskLedBlink(): Controlează blink-ul unui al doilea LED în funcție de starea LED-ului principal și un delay specificat.
- Metoda taskButton23(): Modifică delay-ul în funcție de apăsarea butoanelor specifice (BUTTON_LED_INCR și BUTTON_LED_DECR) și anunță utilizatorul prin portul serial.
- Bucla principală loop(): Rulează în mod continuu și afișează mesajul "BUTTON PRESSED". Apoi, apelează funcțiile taskLedWithButton(), taskLedBlink(), și taskButton23() pentru a executa acțiunile corespunzătoare. Adaugă o mică întârziere de 200 de milisecunde pentru a evita execuția excesivă a funcțiilor.

Acest design permite execuția concurentă a funcțiilor taskLedWithButton(), taskLedBlink(), și t taskButton23() în mod asincron. Fiecare funcție îndeplinește o anumită funcționalitate și reacționează la stările

butoanelor și a LED-urilor. Totul este coordonat prin bucla principală loop() care rulează în mod continuu și apelează aceste funcții în mod repetat.

Concluzie:

La efectuarea acestui laborator s-a abordat subiectul dezvoltarii și implementarii sistemelor de operare secvențiale și utilizarea FreeRTOS ,pentru gestionarea sarcinilor a oferit oportunitatea de a înțelege în profunzime conceptele și tehnologiile fundamentale legate de sistemele de operare încorporate. Această experiență a relevat importanța utilizării unui sistem de operare în dezvoltarea aplicațiilor pentru dispozitive încorporate, precum și beneficiile aduse de un planificator de tip preemptive precum FreeRTOS.

Unul dintre principalele obiective ale acestui laborator a fost să se realizeze o aplicație care să utilizeze un LED și două butoane pentru a demonstra funcționalitatea și utilitatea unui sistem de operare în gestionarea sarcinilor. Aplicația a fost împărțită în patru task-uri diferite, fiecare cu propriul său rol în funcționarea sistemului:

1. Task-ul "Button Led" a fost responsabil pentru schimbarea stării LED-ului la detecția unei apăsări pe buton. Aceasta a demonstrat modul în care o acțiune fizică poate declanșa o sarcină specifică în cadrul sistemului de operare.
2. Task-ul "Led Intermitent" a controlat al doilea LED în funcție de starea LED-ului din primul task. Acest lucru a ilustrat cum task-urile pot comunica între ele și pot reacționa la stările altor sarcini.
3. Task-ul de incrementare/decrementare a variabilei la apăsarea butoanelor a adus o dimensiune de interactivitate la aplicație. Utilizatorul a putut controla comportamentul LED-ului în funcție de numărul de recurențe sau timp specificat.
4. Utilizarea task-ului Idle pentru afișarea stărilor din program a fost o abordare eficientă pentru monitorizarea și raportarea stării sistemului. Implementarea mecanismului de producător-consumator pentru afișarea mesajelor a oferit o soluție elegantă pentru gestionarea evenimentelor din sistem.

În concluzie, acest laborator a demonstrat beneficiile utilizării unui sistem de operare precum FreeRTOS în dezvoltarea aplicațiilor pentru dispozitive încorporate. Acesta oferă o abordare organizată și eficientă pentru gestionarea sarcinilor și comunicarea între ele. Prin implementarea cu succes a obiectivelor, am dobândit o înțelegere mai profundă a funcționării sistemelor de operare și a programării într-un mediu încorporat. Această experiență va servi ca bază valoroasă în dezvoltarea ulterioară a aplicațiilor pentru dispozitive încorporate și în înțelegerea conceptelor de sistem de operare.

Anexa 1

```
#include <stdio.h>  // Includerea bibliotecii standard pentru funcții I/O
#define BUTTON_PIN 13         // Definirea pinului pentru buton
#define BUTTON_LED_INCR 12    // Definirea pinului pentru butonul de incrementare
#define BUTTON_LED_DECR 11    // Definirea pinului pentru butonul de decrementare
#define LED_PIN 4             // Definirea pinului pentru LED-ul principal
#define LED_BLINK_PIN 5       // Definirea pinului pentru LED-ul secundar (intermitent)

int led2Rec = 0;              // Variabilă pentru a ține cont de recurența LED-ului secundar
int ledState = 0;             // Starea curentă a LED-ului principal
int led2State = 0;            // Starea curentă a LED-ului secundar
int buttonState = 0;          // Starea curentă a butonului
int lastButtonState = 1;      // Starea anterioară a butonului (inițial setată pe 1,
presupunând că butonul nu este apăsat)
int timeDelay = 500;          // O variabilă pentru controlul intervalului de timp dintre
blicurile LED-ului secundar

void setup()
{
     pinMode(LED_PIN, OUTPUT);           // Setarea pinului LED-ului principal ca ieșire
     pinMode(LED_BLINK_PIN, OUTPUT);     // Setarea pinului LED-ului secundar ca ieșire
     pinMode(BUTTON_PIN, INPUT);         // Setarea pinului butonului ca intrare
     pinMode(BUTTON_LED_DECR, INPUT);    // Setarea pinului butonului de decrementare ca
intrare
     pinMode(BUTTON_LED_INCR, INPUT);    // Setarea pinului butonului de incrementare ca
intrare
     Serial.begin(9600);                 // Inițializarea comunicării seriale la o rată de
9600 de biți pe secundă
}

void taskLedWithButton()
{
     buttonState = digitalRead(BUTTON_PIN);  // Citirea stării butonului

     // Verificarea dacă butonul a fost apăsat și eliberat (flanc pozitiv)
     if (buttonState == 1 && lastButtonState == 0)
     {
          ledState = !ledState;               // Inversarea stării LED-ului principal
          digitalWrite(LED_PIN, ledState);   // Actualizarea stării LED-ului principal
     }

     // Dacă LED-ul principal este aprins, se oprește LED-ul secundar
     if (ledState == 1)
     {
          digitalWrite(LED_BLINK_PIN, LOW);
     }

     lastButtonState = buttonState;        // Actualizarea stării anterioare a butonului
}

void taskLedBlink()
{
     static unsigned long previousBlinkTime = 0;  // Variabilă statică pentru a ține evidența
timpului anterior

     // Verificarea dacă LED-ul principal este stins
     if (ledState == 0)
     {
          unsigned long currentTime = millis();  // Obținerea timpului curent în milisecunde
```

```
          // Verificarea dacă a trecut timpul necesar pentru a face LED-ul secundar să
clipească
          if (currentTime - previousBlinkTime >= timeDelay)
          {
               previousBlinkTime = currentTime;  // Actualizarea timpului anterior

               // Inversarea stării LED-ului secundar pentru a-l face să clipească
               if (led2State == 0)
               {
                    led2State = 1;
                    digitalWrite(LED_BLINK_PIN, led2State);
               }
               else
               {
                    led2State = 0;
                    digitalWrite(LED_BLINK_PIN, led2State);
               }
          }
     }
}

void taskButton23 ()
{
     // Verificarea dacă intervalul de timp este mai mare sau egal cu 400 ms
     if (timeDelay >= 400)
     {
          // Verificarea dacă butonul de incrementare este apăsat
          if (digitalRead(BUTTON_LED_INCR) == 1)
          {
               timeDelay = timeDelay + 100;         // Incrementarea intervalului de timp
               Serial.print("Button for incremention pressed! ");
               Serial.println(timeDelay);
          }

          // Verificarea dacă butonul de decrementare este apăsat și intervalul este mai mare
de 400 ms
          if (digitalRead(BUTTON_LED_DECR) == 1 && timeDelay > 400)
          {
               timeDelay = timeDelay - 100;         // Decrementarea intervalului de timp
               Serial.print("Button for decremention pressed! ");
               Serial.println(timeDelay);
          }
     }
}

void loop()
{
     printf("%s", "\nBUTTON PRESSED");  // Afisare mesaj la fiecare trecere prin buclă
     taskLedWithButton();               // Apelarea funcției pentru gestionarea LED-ului
principal și a butonului
     taskLedBlink();                    // Apelarea funcției pentru gestionarea LED-ului
secundar (intermitent)
     taskButton23 ();           // Apelarea funcției pentru gestionarea butoanelor de
incrementare și decrementare
     delay(200);                         // O pauză mică pentru a evita prea multe citiri ale
butonului într-un interval scurt
}
```