Producator-consumator

Programul gestioneaza activitatea unui sistem de tip producator-consumator, utilizand FreeRTOS pentru a sincroniza doua taskuri concurente: unul care produce date si le adauga intr-un buffer si altul care consuma aceste date.

```
1   #include <Arduino_FreeRTOS.h>
2   #include <semphr.h>
3
4   #define N 5
5   int buffer[N], in, out;
6
7   SemaphoreHandle_t mutex, semPlin, semGol;
```

La inceput, sunt definite dimensiunea bufferului si indicii de scriere si citire. De asemenea, sunt declarate semafoarele pentru sincronizarea accesului la buffer: un mutex pentru protejarea sectiunii critice (unde se efectueaza scrierea si citirea in buffer) si doua semafoare de tip counting pentru a controla fluxul de date intre taskuri.

```
9   void taskProducator(void *pvParameters) {
10    while (1) {
11      xSemaphoreTake(semPlin, portMAX_DELAY);
12      xSemaphoreTake(mutex, portMAX_DELAY);
13
14      buffer[in % N] = in;
15      Serial.print("A fost produs: ");
16      Serial.println(in++);
17
18      xSemaphoreGive(mutex);
19      xSemaphoreGive(semGol);
20      vTaskDelay(pdMS_TO_TICKS(1500));
21    }
22  }
```

“taskProducator” se ocupa cu generarea si adaugarea datelor in buffer. Acesta asteapta sa existe loc in buffer (prin semaforul “semPlin”) inainte de a introduce un nou element. Dupa ce datele sunt adaugate, semaforul “semGol” este incrementat pentru a semnaliza ca exista un element disponibil pentru a fi consumat de catre consumator. De asemenea, taskul utilizeaza un mutex pentru a proteja accesul exclusiv la buffer.

```
24  void taskConsumator(void *pvParameters) {
25    while (1) {
26      xSemaphoreTake(semGol, portMAX_DELAY);
27      xSemaphoreTake(mutex, portMAX_DELAY);
28
29      int w = buffer[out];
30      out = (out + 1) % N;
31      Serial.print("A fost consumat: ");
32      Serial.println(w);
33
34      xSemaphoreGive(mutex);
35      xSemaphoreGive(semPlin);
36      vTaskDelay(pdMS_TO_TICKS(2000));
37    }
38  }
```

“taskConsumator” consuma datele din buffer, asteptand ca acesta sa contina date disponibile (prin semaforul “semGol”). Dupa ce datele sunt consumate, semaforul “semPlin” este incrementat pentru a semnaliza ca exista loc in buffer pentru noi date. La fel ca taskul producator, si acest task utilizeaza un mutex pentru a proteja sectiunea critica de accesul concurent.

```
40  void setup() {
41    Serial.begin(9600);
42
43    mutex = xSemaphoreCreateMutex();
44    semPlin = xSemaphoreCreateCounting(N, N);
45    semGol = xSemaphoreCreateCounting(N, 0);
46
47    xTaskCreate(taskProducator, "Producator", 128, NULL, 1, NULL);
48    xTaskCreate(taskConsumator, "Consumator", 128, NULL, 1, NULL);
49  }
50
51  void loop() {}
```

In partea de setup, se initializeaza semafoarele si mutexul pentru a permite taskurilor sa ruleze in mod corect si sa sincronizeze accesul la buffer. Taskurile sunt apoi create si initializate cu dimensiuni de stack si prioritati corespunzatoare.

```
A fost consumat: 3
A fost produs: 4
A fost produs: 5
A fost consumat: 4
A fost produs: 6
A fost consumat: 5
A fost produs: 7
A fost consumat: 6
```