|  |  |  |
| --- | --- | --- |
| **KELOMPOK** | 1 | **ACC** |
| **Tanggal Praktikum** | 4 Mei 2018 |  |
|  | Khairunnisa Sekar R. (D400150005) |  |
|  | Dian Arieska (D400150141) | **Tanggal ACC :** |
|  | Nindya Kaloka (D400170019) | **Revisi Tanggal :** |

**MODUL 2**

Semaphore dalam FreeRTOS

1. **TUJUAN**
2. Praktikan dapat memahami dan mengetahui fungsi Semaphore dalam FreeRTOS.
3. Praktikan dapat membuat program dengan fungsi Semaphore dalam FreeRTOS.
4. Praktikan dapat membuat program dengan 3 task dengan menggunakan fungsi Semaphore dalam FreeRTOS.
5. **ALAT DAN BAHAN**
6. Laptop / PC yang telah diinstal software Arduino
7. Arduino Uno
8. **DASAR TEORI**

Semaphore adalah suatu isyarat yang digunakan untuk menentukan akses untuk berbagi sumber daya sistem. Dalam software, semaphore adalah sebuah variabel bertipe integer yang selain saat inisialisasi, hanya dapat diakses melalui dua operasi standar, yaitu increment dan decrement. Semaphore digunakan untuk menyelesaikan masalah sinkronisasi secara umum. Berdasarkan jenisnya, semaphore hanya bisa memiliki nilai 1 atau 0, atau lebih dari sama dengan 0. Konsep semaphore pertama kali diajukan idenya oleh Edsger Dijkstra pada tahun 1967.

**Semaphore memiliki dua jenis, yaitu**:

1. *Binary semaphore*. Semaphore ini hanya memiliki nilai 1 atau 0. Sering juga disebut sebagai semaphore primitif
2. *Counting semaphore*. Semaphore ini memiliki nilai 0, 1, serta integer lainnya. Banyak sistem operasi yang tidak secara langsung mengimplementasikan semaphore ini, tetapi dengan memanfaatkan binary semaphore

**Prinsip semaphore** :

1. Dua proses atau lebih dapat bekerja sama dengan menggunakan penanda-penanda sederhana
2. Proses dipaksa berhenti sampai proses memperoleh penanda tertentu
3. Variabel khusus untuk penandaan ini disebut semaphore

**Fungsi Semaphore :**

Seperti telah disebutkan sebelumnya, semaphore berfungsi untuk menangani masalah sinkronisasi secara umum, yaitu:

1. Mutual Exclusion. Sesuai dengan prinsip mutual exclusion, jika suatu thread sedang berada dalam critical section-nya, thread lain harus menunggu thread tersebut keluar dari critical section-nya sebelum dapat memasuki critical section-nya sendiri. Di sinilah semaphore digunakan, thread yang akan memasuki critical section-nya akan memanggil fungsi kunci terlebih dahulu. Jika tidak ada thread lain yang sedang berada dalam critical section, thread ini akan memasuki critical section-nya. Jika terdapat thread lain yang sedang berada dalam critical section-nya, thread ini harus menunggu. Setelah thread keluar dari critical section-nya, thread tersebut akan memanggil fungsi buka sehingga sem\_value akan naik menjadi lebih dari 0, dan satu (dari beberapa) thread yang sedang menunggu akan mendapatkan giliran untuk memasuki critical section-nya.

2. Resource Controller. Bayangkan sebuah restoran yang setiap malamnya ramai dikunjungi pelanggan. Kapasitas restoran terbatas, tetapi pemilik restoran memiliki kebijakan bahwa semua pengunjung yang datang akan mendapatkan kesempatan untuk makan, dengan konsekuensi yaitu pelanggan harus sabar menunggu gilirannya. Oleh karena itu, dikerahkanlah pegawai restoran untuk menahan tamu di luar jika restoran penuh lalu mempersilahkan tamu masuk jika tempat telah tersedia.Dari analogi di atas, pelanggan adalah thread, kapasitas restoran adalah resource, dan pegawai restoran adalah semaphore. Semaphore menyimpan banyaknya resource yang tersedia. Saat thread ingin memakai resource ia akan memanggil fungsi kunci. Jika resource masih tersedia, thread bisa langsung menggunakannya, sebaliknya jika semua resource sedang dipakai, thread tersebut harus menunggu. Setelah resource selesai dipakai thread akan memanggil fungsi buka sehingga resource yang bebas bertambah.

3. Sinkronisasi Antar-Proses. Ada kalanya suatu thread memerlukan resource yang dihasilkan oleh thread lainnya. Oleh karena itu dibutuhkan suatu mekanisme untuk mengatur urutan eksekusi thread. Mekanisme ini dilakukan dengan memanfaatkan semaphore.

**Keuntungan menggunakan semaphore:**

1. Dari segi programming, penanganan masalah sinkronisasi dengan semaphore umumnya rapi dan teratur, sehingga mudah untuk dibuktikan kebenarannya
2. Semaphore diimplementasikan dalam hard code sehingga penggunaannya bersifat portabel
3. **HASIL PRAKTIKUM**

D.1 Hasil Percobaan Script

#include<FreeRTOS\_AVR.h>

SemaphoreHandle\_t sem;

static void Klakson1(void\* arg){

while (1){

vTaskDelay(100);

if (xSemaphoreTake (sem, portMAX\_DELAY)){

xSemaphoreGive(sem);

Serial.print("klakson Satu ");

bus();

Serial.println("\t\tKlakson Satu berbunyi TELOLET");

vTaskDelay(100);

}

}

}

static void Klakson2(void\* arg){

while (1){

vTaskDelay(200);

if (xSemaphoreTake (sem, portMAX\_DELAY)){

xSemaphoreGive(sem);

Serial.print("Klakson Dua ");

bus();

Serial.println("\t\tKlakson Dua berbunyi TWUIWITWIT");

vTaskDelay(200);

}

}

}

static void Klakson3(void\* arg){

while (1){

vTaskDelay(300);

if (xSemaphoreTake (sem, portMAX\_DELAY)){

xSemaphoreGive(sem);

Serial.print("Klakson Tiga ");

bus();

Serial.println("\t\tKlakson Tiga berbunyi PIUWIUWIUWI");

vTaskDelay(300);

}

}

}

void bus(){

Serial.println("Berdering");

vTaskDelay(500);

}

void setup()

{

portBASE\_TYPE s1,s2,s3;

Serial.begin(9600);

sem = xSemaphoreCreateCounting(1,0);

s1= xTaskCreate(Klakson1, NULL, configMINIMAL\_STACK\_SIZE, NULL, 3,NULL);

s2= xTaskCreate(Klakson2, NULL, configMINIMAL\_STACK\_SIZE, NULL, 2,NULL);

s3= xTaskCreate(Klakson3, NULL, configMINIMAL\_STACK\_SIZE, NULL, 1,NULL);

if (sem==NULL ||s1 !=pdPASS || s2 !=pdPASS || s3 !=pdPASS)

{

Serial.println(F("Creation Problem"));

while(1);

}

xSemaphoreGive(sem);

vTaskStartScheduler();

Serial.println(F("Insufficient RAM"));

while(1);

}

void loop()

{

// put your main code here, to run repeatedly:

}

1. **ANALISA**

E.1. Analisa Percobaan

Pada praktikum 2 ini menggunakan semaphore dan 3 task pada program dan membuat program dapat berjalan secara terus menerus. xSemaphoreGive digunakan untuk memasukkan data kemudian akan diambil oleh xSemaphoreTake dan kemudian ditampilkan pada Serial Monitor.

vTaskDelay digunakan untuk menampilkan data dengan penundaan pemunculan data di serial monitor. Serial.println digunakan untuk menapilkan data keserial monitor sesuai urutan yang ditentukan sesuai prioritas. Sedangkan untuk fungsi dari sem =xSemaphore CreateCounting(1,0); digunakan untuk memberi status bit bernilai 1 (high) atau bernilai 0 (low). s1= xTaskCreate(Klakson1, NULL, configMINIMAL\_STACK\_SIZE, NULL, 3,NULL); digunakan untuk meng*create* Task dan menentukan nomor prioritas untuk pemunculan pada serial monitor.

1. **KESIMPULAN**
2. Semaphore adalah suatu isyarat yang digunakan untuk menentukan akses untuk berbagi sumber daya sistem.
3. xSemaphoreGive dan xSemaphoreTake dalam praktikum ini saling berhubungan dalam jalannya program
4. Dari Serial Monitor terlihat hasil dari xSemaphoreGive dan xSemaphoreTake secara acak karena pengaruh vTaskDelay