|  |  |  |
| --- | --- | --- |
| **KELOMPOK** | 1 | **ACC** |
| **Tanggal Praktikum** | 02 Juni 2018 |  |
|  | Khairunnisa Sekar R. (D400150005) |  |
|  | Dian Arieska (D400150141) |  |
|  | Nindya Kaloka (D400170019) | **Tanggal ACC :** |
|  | Saiful Arifin W.W. (D400160065) | **Revisi Tanggal :** |

**MODUL 3**

**Flexible and Absolute Timing On**

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

Timing atau delay adalah suatu jeda waktu yang digunakan untuk mengeksekusi task selanjutnya. Dalam FreeRTOS, delay terbagi menjadi 2 jenis yaitu flexible dan absolute.

1. **Flexible timing ( vTaskDelay() )**

vtaskDelay() adalah delay yang berdasarkan jumlah ticks, sehingga lama waktu aktual suatu task di blok atau ditahan bergantung pada tick rate. Constant portTICK\_PERIOD\_MS dapat digunakan untuk mengkalkulasi waktu aktual dari tingkat tick - dengan resolusi satu periode tick.

vTaskDelay() menentukan waktu dimana task ingin mulai dijalankan relative terhadap waktu dimana vTaskDelay() dipanggil. Misalnya, menentukan jangka waktu delay 100 tick akan menyebabkan task mulai aktif dijalankan 100 tick setelah vTaskDelay() dipanggil. Oleh karena itu, vTaskDelay() tidak menyediakan metode yang baik untuk mengendalikan frekuensi tugas periodic karena jalur yang diambil melalui kode, serta task dan interupsi lainnya akan mempengaruhi frekuensi dimana vTaskDelay() dipanggil dan waktu dimana task selanjutnya aktif. vTaskDelay() sesuai untuk digunakan pada task yang flexible.

Parameters:

xTicksToDelay : Jumlah waktu dalam tick periode

Contoh penggunaan

void vTaskFunction( void \* pvParameters )

{

/\* Block for 500ms. \*/

const TickType\_t xDelay = 500 / portTICK\_PERIOD\_MS;

for( ;; )

{

/\* Simply toggle the LED every 500ms, blocking between each toggle. \*/

vToggleLED();

vTaskDelay( xDelay );

}

}

1. **Absolute timing ( vTaskDelayUntil() )**

Tunda task hingga waktu yang ditentukan. Fungsi ini dapat digunakan oleh tugas-tugas periodik untuk memastikan frekuensi eksekusi konstan.

Fungsi ini berbeda dari vTaskDelay() dalam satu aspek penting: vTaskDelay() menentukan waktu di mana tugas ingin membuka blokir relatif terhadap waktu di mana vTaskDelay() dipanggil, sedangkan vTaskDelayUntil() menentukan waktu absolut di mana tugas ingin untuk membuka blokir atau mengaktifkan task selanjutnya

vTaskDelay() akan menyebabkan delay aktif dari task selama sejumlah ticks yang ditentukan dari waktu vTaskDelay() dipanggil. Oleh karena itu sulit untuk menggunakan vTaskDelay() dengan sendirinya untuk menghasilkan frekuensi eksekusi tetap sebagai waktu antara task membuka blokir setelah panggilan ke vTaskDelay() dan task berikutnya.

Sedangkan vTaskDelay() menentukan waktu bangun relatif terhadap waktu saat fungsi dipanggil, vTaskDelayUntil() menentukan waktu absolut (tepat) di mana ia ingin membuka blokir atau mengkatifkan task selanjutnya.

Perlu dicatat bahwa vTaskDelayUntil() akan segera kembali (tanpa pemblokiran) jika digunakan untuk menentukan waktu bangun yang sudah di masa lalu. Oleh karena itu, task menggunakan vTaskDelayUntil() untuk melakukan secara berkala harus menghitung ulang waktu bangun yang diperlukan jika eksekusi periodik dihentikan karena alasan apa pun (misalnya, tugas untuk sementara ditempatkan ke status ditangguhkan) yang menyebabkan task untuk melewatkan satu atau eksekusi lebih berkala. Ini dapat dideteksi dengan memeriksa variabel yang dilewatkan oleh referensi sebagai parameter pxPreviousWakeTime terhadap hitungan tick saat ini. Namun ini tidak diperlukan di sebagian besar skenario penggunaan.

PortTICK\_PERIOD\_MS konstan dapat digunakan untuk menghitung waktu aktual dari tingkat tick - dengan resolusi satu periode tick.

Fungsi ini tidak boleh dipanggil ketika scheduler RTOS telah ditangguhkan oleh panggilan ke vTaskSuspendAll ().

Parameters:

pxPreviusWakeTime : Penunjuk ke variabel yang menyimpan waktu saat task terakhir diblokir. Variabel harus diinisialisasi dengan waktu saat ini sebelum penggunaan pertama (lihat contoh di bawah). Setelah ini, variabel secara otomatis diperbarui dalam vTaskDelayUntil ().

xTimeIncrement : Periode waktu siklus. Task akan dibebaskan pada waktunya (\* pxPreviousWakeTime + xTimeIncrement). Memanggil vTaskDelayUntil dengan nilai parameter xTimeIncrement yang sama akan menyebabkan task dijalankan dengan periode interval tetap.

Contoh penggunaan

// Perform an action every 10 ticks.

void vTaskFunction( void \* pvParameters )

{

TickType\_t xLastWakeTime;

const TickType\_t xFrequency = 10;

// Initialise the xLastWakeTime variable with the current time.

xLastWakeTime = xTaskGetTickCount();

for( ;; )

{

// Wait for the next cycle.

vTaskDelayUntil( &xLastWakeTime, xFrequency );

// Perform action here.

}

}

1. **HASIL PRAKTIKUM**

D.1 Hasil Percobaan Script

#include "FreeRTOS\_AVR.h"

#include "basic\_io\_AVR.h"

void vContinousProcessingTask (void \*pvParameters);

void vPeriodicTask (void \*pvParameters);

const char \*pcTextForTask1 = "Continous task 1 is running\r\n";

const char \*pcTextForTask2 = "Continous task 2 is running\r\n";

const char \*pcTextForPeriodicTask3 = "Periodic task3 is running\r\n";

const char \*pcTextForPeriodicTask4 = "Periodic task4 is running\r\n";

void setup(void)

{

Serial.begin(9600);

xTaskCreate(vContinousProcessingTask, "Task 1", 200, (void\*)pcTextForTask1, 2, NULL);

xTaskCreate(vContinousProcessingTask, "Task 2", 200, (void\*)pcTextForTask2, 2, NULL);

xTaskCreate(vPeriodicTask3, "Task 3", 200, (void\*)pcTextForPeriodicTask3, 3, NULL);

xTaskCreate(vPeriodicTask4, "Task 4", 200, (void\*)pcTextForPeriodicTask4, 3, NULL);

vTaskStartScheduler();

for(;;);

}

void vContinousProcessingTask (void \*pvParameters)

{

char \*pcTaskname;

pcTaskname = (char\*)pvParameters;

for(;;)

{

vPrintString(pcTaskname);

}

vTaskDelay(500);

}

void vPeriodicTask3 (void \*pvParameters)

{

TickType\_t xLastWakeTime3;

xLastWakeTime3 = xTaskGetTickCount ();

for(;;)

{

vPrintString("Periodic task3 is running------------------\r\n");

vTaskDelayUntil(&xLastWakeTime3, (1000/portTICK\_PERIOD\_MS));

}

}

void vPeriodicTask4 (void \*pvParameters)

{

TickType\_t xLastWakeTime4;

xLastWakeTime4 = xTaskGetTickCount ();

for(;;)

{

vTaskDelayUntil(&xLastWakeTime4, (500/portTICK\_PERIOD\_MS));

vPrintString("Periodic task4 is running------------------\r\n");

vTaskDelayUntil(&xLastWakeTime4, (500/portTICK\_PERIOD\_MS));

}

}

void loop() {

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

}

1. **ANALISA**

E.1. Analisa Percobaan

Pada praktikum 3 ini menggunakan vTaskdelay(), vTaskDelayUntil() dan 4 task pada program dan membuat program dapat berjalan secara terus menerus. vTaskDelay() digunakan pada void vContinousProcessingTask sedangkan vTaskDelayUntil() digunakan pada void vPeriodicTask. Task 1, 2 terdapat pada void vContinousProcessingTask sedangkan Task 3 terdapat pada void vPeriodicTask3 dan Task 4 terdapat pada void vPeriodicTask4. Untuk hasil dari program ini ditampilkan pada Serial Monitor.

vTaskDelay() menentukan waktu di mana tugas ingin membuka blokir relatif terhadap waktu di mana vTaskDelay() dipanggil, sedangkan vTaskDelayUntil() menentukan waktu absolut di mana tugas ingin untuk membuka blokir atau mengaktifkan task selanjutnya. Serial.println digunakan untuk menampilkan data ke serial monitor sesuai urutan yang ditentukan sesuai prioritas.

1. **KESIMPULAN**
2. vTaskDelay() menentukan waktu di mana tugas ingin membuka blokir relatif terhadap waktu di mana vTaskDelay() dipanggil.
3. vTaskDelayUntil() menentukan waktu absolut di mana tugas ingin untuk membuka blokir atau mengaktifkan task selanjutnya.