Tipe Memory Allocation

Dalam sebuah program, baik secara umum maupun pada Sistem Embedded, terdapat beberapa jenis alokasi memory yang dapat dilakukan, diantaranya sebagai berikut: [1]

60c3b8b7-f4af-4a07-b139-0acae5a846fb

Static Variable

Static memory digunakan untuk menyimpan variabel global maupun variabel yang dideklarasikan sebagai static di dalam kode. Berbeda dengan variabel lokal biasa, variabel static tidak hilang setelah fungsi selesai dijalankan, melainkan tetap ada (persist) sepanjang program berjalan. Nilainya akan diingat pada pemanggilan fungsi berikutnya.

Variable-Variable seperti global counter, variable pin / port, merupakan contoh-contoh static variable pada sistem Embedded.

Dalam praktikum ini, karena pemrograman menggunakan bahasa C++, setiap variabel bersifat type-based, artinya ukuran memori sudah ditentukan berdasarkan tipe data yang digunakan. Oleh karena itu, pada static variable, pemilihan tipe dan ukuran variabel harus tepat agar penggunaan memori lebih efisien dan sesuai kebutuhan. Oleh sebab itu, tidak jarang ditemukan fixed-width integer seperti int8_t, int16_t, uint16_t, uint32_t dan sebagainya.

Informasi mengenai fixed width integer dapat dipelajari secara lebih pada link berikut.

Stack

Stack digunakan ketika terjadi alokasi otomatis oleh variabel lokal. Memori stack diatur dengan prinsip Last-In-First-Out (LIFO), di mana variabel dari suatu fungsi akan didorong (push) ke stack saat fungsi dipanggil. Setelah fungsi selesai dan kembali, variabel-variabel tersebut akan dikeluarkan (pop) dari stack, sehingga fungsi dapat melanjutkan eksekusi seperti sebelumnya.

Pada FreeRTOS, ketika sebuah task dibuat menggunakan xTaskCreate(), sistem operasi akan mengalokasikan sebagian memori heap untuk task tersebut. Alokasi ini terdiri dari dua bagian:

- Task Control Block (TCB) Berisi informasi penting tentang task, seperti prioritas, state, dan pointer ke local stack.
- Local Stack Task Digunakan khusus untuk menyimpan variabel lokal dan data saat fungsi dalam task tersebut dipanggil, mirip seperti stack global pada program utama tetapi terbatas hanya untuk task itu.

Setiap variabel lokal yang dibuat dalam sebuah fungsi task akan ditempatkan di local stack milik task tersebut. Oleh karena itu, sangat penting untuk memperkirakan kebutuhan stack sebelum membuat task, lalu menentukan ukurannya pada parameter stack size di xTaskCreate(). Jika ukuran stack terlalu kecil, task bisa mengalami stack overflow yang menyebabkan error atau crash pada sistem.

Heap

Heap adalah area memori yang digunakan untuk alokasi dinamis. Tidak seperti alokasi stack yang dilakukan secara otomatis, heap harus dialokasikan secara eksplisit oleh programmer. Pada bahasa pemrograman C dan C++, proses alokasi memory pada heap dilakukan melalui fungsi berikut.

- malloc() → untuk mengalokasikan memori
- free() → untuk melepaskan kembali memori yang sudah tidak dipakai

Proses ini disebut dynamic allocation. Jika memori heap tidak dibebaskan setelah selesai digunakan, maka akan terjadi memory leak, yang bisa menyebabkan sistem kehabisan memori, crash, atau bahkan korupsi data pada area memori lain.

Dalam FreeRTOS, terdapat potensi dimana dua buah task berbeda mencoba mengakses lokasi memory yang sama, sehingga fungsi malloc() dan free() tidak thread-safe dan tidak aman digunakan antar task. Karena itu, FreeRTOS menyediakan fungsi khusus:

- pvPortMalloc() → untuk mengalokasikan memori dari heap global FreeRTOS
- vPortFree() → untuk mengembalikan memori yang sudah tidak digunakan

Dengan cara ini, alokasi heap lebih aman di lingkungan multitasking, karena RTOS mengatur sinkronisasi dan pengelolaan memori agar tidak saling bertabrakan antar task.

Hubungan Stack dan Heap

Pada kebanyakan sistem, stack dan heap tumbuh saling mendekati dalam ruang memori yang sama. Jika penggunaan keduanya tidak dikontrol, keduanya bisa bertabrakan (stack-heap collision), yang menyebabkan data saling menimpa dan mengakibatkan error serius.

b9d52446-ebca-4ae5-9e8d-9a4a035d1e4d

Demonstrasi Memory Allocation

Berikut merupakan contoh task yang dapat menyebabkan memory leak akibat kurangnya memory deallocation menggunakan vPortFree. Jika dijalankan, sewaktu-waktu jumlah memory yang ada akan habis sehingga sistem akan berhenti membaca sensor.

```
#include <Arduino.h>
#include <FreeRTOS.h>

void SensorTask(void *pvParameters) {
    while (1) {
        // Alokasikan buffer untuk data sensor
        int *sensorData = (int *) pvPortMalloc(50 * sizeof(int)); // buffer 50 data
        if (sensorData == NULL) {
            Serial.println("Error - Heap exhausted.");
        }

        // Pengambilan data sensor
        for (int i = 0; i < 50; i++) {
            sensorData[i] = analogRead(A0);
        }

        // Cetak sebagian hasil
        Serial.print("Sensor[0] = ");
        Serial.print(sensorData[0]);
        Serial.print(", Sensor[49] = ");
        Serial.println(sensorData[49]);

        // Tidak ada vPortFree(sensorData);


        vTaskDelay(1000 / portTICK_PERIOD_MS);
    }
}

void setup() {
    Serial.begin(115200);
    delay(1000);

    xTaskCreate(
        SensorTask,
        "SensorTask",
        2048,
        NULL,
        1,
        NULL
```

```
    );
}

void loop() {

}
```

Revision #1
Created 2025-09-13 19:00:52 UTC by Digilab UI
Updated 2025-09-13 19:02:07 UTC by Digilab UI