Queue

Data structure queue mungkin sudah familiar setelah digunakan pada praktikum pemrograman sebelum sebelumnya. Data structure ini bersifat FIFO dimana data yang masuk pertama kedalam queue akan menjadi data yang pertama keluar dari queue. [3]

Dalam konteks IoT dan FreeRTOS, queue digunakan sebagai medium untuk mengirimkan data antar task atau antar perangkat dan server.

Mengapa queue diperlukan?

Queue diperlukan karena pada sistem multitasking, beberapa task bisa saja mencoba mengakses atau menulis ke memori yang sama secara bersamaan.

Hal ini dapat kita contohkan pada skenario tersebut. Pada skenario ini kita memiliki satu buah global variable (seperti counter) yang akan diakses ataupun diubah oleh task. Task A akan mengubah nilai global variable ini (seperti mengincrement global counter) setelah menjalankan task yang dimiliki. Task C juga akan melakukan hal yang sama setelah menyelesaikan tasknya. Task B akan membaca nilai dari global variable ini pada proses tasknya dan melakukan printing pada serial monitor. Dikarenakan seluruh task berjalan secara bersamaan / paralel, akan terdapat kemungkinan dimana hasil write pada global variable oleh task A akan di overwrite oleh task B.
image

Dengan kata lain, jika pengaksesan dan transfer data pada sistem multithreaded / parallel ini dibiarkan tanpa pengaturan, maka akan terjadi memory overwriting atau race condition, di mana data yang sudah ditulis oleh satu task bisa tertimpa oleh task lain sebelum sempat diproses.

Dengan adanya queue, setiap data yang dikirim akan disimpan secara terpisah dalam antrian, sehingga proses tulis dan baca berlangsung secara atomic (tidak bisa diinterupsi oleh task lain di tengah jalan). Selain itu, sifat FIFO dari queue menjamin bahwa urutan data tetap terjaga, sehingga task penerima dapat memproses data sesuai dengan urutan kedatangannya.

88d32074-22e7-4fc3-943f-9e59b06dede9

Dalam konteks IoT, queue juga berperan sebagai buffer, misalnya ketika perangkat ingin mengirimkan data ke server tetapi koneksi sedang terputus. Data dapat terlebih dahulu disimpan di dalam queue untuk kemudian dikirimkan kembali saat koneksi sudah tersedia, sehingga tidak ada data yang hilang.

Menggunakan Queue pada FreeRTOS

Pada FreeRTOS, disediakan API khusus untuk membuat, menulis, dan membaca queue.

Membuat Queue

Queue dibuat dengan fungsi xQueueCreate(), yang membutuhkan dua parameter utama:

- uxQueueLength → jumlah maksimum item yang dapat disimpan dalam queue.
- uxItemSize → ukuran (dalam byte) tiap item yang akan disimpan.

Contoh: membuat queue yang dapat menyimpan 10 integer (int):

```
QueueHandle_t xQueue;
xQueue = xQueueCreate(10, sizeof(int));
if (xQueue == NULL) {
    Serial.println("Error: Queue creation failed.");
}
```

Dapat dilihat bahwa queue memiliki ukuran tetap, dalam hal ini adalah 10 x ukuran byte dari integer.

if (xQueue == NULL) dapat ditambahkan untuk memastikan queue berhasil dibuat, dan mencegah kegagalan pembuatan queue karena keterbatasan memori.

Mengirim Data ke Queue

Task atau ISR (Interrupt Service Routine) dapat mengirim data ke queue menggunakan: xQueueSend() → untuk mengirim dari task biasa. xQueueSendFromISR() → untuk mengirim dari ISR. Contoh: mengirim data sensor ke queue dari sebuah task:

```
int sensorValue = analogRead(A0);
xQueueSend(xQueue, &sensorValue, portMAX_DELAY);
```

Membaca Data dari Queue

Task penerima membaca data dari queue menggunakan xQueueReceive(). Data akan dihapus dari queue setelah berhasil dibaca.

```
int receivedValue;
if (xQueueReceive(xQueue, &receivedValue, portMAX_DELAY) == pdPASS) {
    Serial.print("Received: ");
    Serial.println(receivedValue);
}
```

Parameter-Parameter pada Queue

- Parameter portMAX_DELAY memiliki fungsi berikut:
  - Pada xQueueSend(): task akan menunggu jika queue sedang penuh, sampai ada space kosong untuk memasukkan data.
  - Pada xQueueReceive(): task akan menunggu jika queue kosong, sampai ada data baru masuk.\
- Blocking vs Non-Blocking pada Queue
  - Blocking :Task akan masuk mode blocking ketika kita memberikan timeout > 0 atau portMAX_DELAY pada fungsi queue (xQueueSend() atau xQueueReceive()). Pada saat ini, task akan menunggu hingga block berakhir (dalam hal ini, space tersedia pada queue)
  - Non-Blocking: Task akan masuk mode non-blocking ketika kita memberikan timeout = 0, dimana fungsi queue akan langsung kembali meskipun queue penuh atau kosong. Dengan kata lain, task harus tetap berjalan terus-menerus tanpa tertahan oleh queue.
- Return Value
  - pdTRUE / pdFALSE → digunakan pada fungsi seperti xQueueSend() dan xQueueReceive(), menunjukkan apakah operasi berhasil (pdTRUE) atau gagal (pdFALSE).
  - pdPASS / pdFAIL → digunakan pada operasi yang lebih kompleks atau alokasi memory, menunjukkan keberhasilan (pdPASS) atau kegagalan (pdFAIL).
  - pdTRUE/pdFALSE biasanya dipakai pada fungsi queue (xQueueSend(), xQueueReceive()) untuk menunjukkan status keberhasilan operasi. Sedangkan pdPASS/pdFAIL dipakai pada operasi FreeRTOS yang lebih umum (misal pembuatan queue, semaphore, atau alokasi heap)

Informasi lebih lanjut mengenai parameter-parameter pada API queue dapat dibaca pada dokumentasi FreeRTOS

Contoh Mengirimkan dan Menerima Data Serial Via Queue

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

// Handle untuk Queue
QueueHandle_t xQueue;

// Ukuran buffer pesan
```

```
#define MSG_MAX_LEN 50

// Task untuk membaca input serial sampai newline
void SerialReadTask(void *pvParameters) {
  while (1) {
    if (Serial.available() > 0) {
      // Baca string sampai newline
      String input = Serial.readStringUntil('\n');

      // Pastikan tidak melebihi buffer
      if (input.length() > 0 && input.length() < MSG_MAX_LEN) {
        char buffer[MSG_MAX_LEN];
        input.toCharArray(buffer, MSG_MAX_LEN);

        // Kirim ke queue
        if (xQueueSend(xQueue, buffer, portMAX_DELAY) == pdPASS) {
          Serial.println("[Sent to Queue]");
        }
      }
    }
    vTaskDelay(10 / portTICK_PERIOD_MS);
  }
}

// Task untuk membaca dari queue dan menampilkan pesan
void SerialPrintTask(void *pvParameters) {
  char received[MSG_MAX_LEN];

  while (1) {
    if (xQueueReceive(xQueue, &received, portMAX_DELAY) == pdPASS) {
      Serial.print("Message received: ");
      Serial.println(received);
    }
  }
}

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

  // Buat queue dengan kapasitas 5 pesan
```

```
  xQueue = xQueueCreate(5, MSG_MAX_LEN);

  if (xQueue != NULL) {
    xTaskCreate(SerialReadTask, "SerialRead", 4096, NULL, 1, NULL);
    xTaskCreate(SerialPrintTask, "SerialPrint", 4096, NULL, 1, NULL);
  } else {
    Serial.println("Error: Queue creation failed!");
  }
}

void loop() {
}
```

Revision #1
Created 2025-09-13 19:03:44 UTC by Digilab UI
Updated 2025-09-13 19:05:00 UTC by Digilab UI