**ĐẠI HỌC QUỐC GIA**

**ĐẠI HỌC BÁCH KHOA TP HỒ CHÍ MINH**

🙞∙∙∙☼∙∙∙🙜



**– Embedded System–**

**LAB 4**

**Nhóm 2 – L02**

|  |  |
| --- | --- |
| **Họ và tên** | **MSSV** |
| Trần Nguyễn Minh Duy | 1910095 |
| Đặng Trung Kiên | 1911437 |
| Nguyễn Hải Long | 1911517 |
| Nguyễn Nhật Trường | 1912344 |

*Thành phố Hồ Chí Minh – 2022*

**MỤC LỤC**

[ĐÓNG GÓP CỦA CÁC THÀNH VIÊN 2](#_Toc120304186)

[BÁO CÁO 3](#_Toc120304187)

[**1.** **Prioritized Pre-emptive Scheduling with Time Slicing** 3](#_Toc120304188)

[1.1. Điều chỉnh trong menuconfig 3](#_Toc120304189)

[1.2. Điều chỉnh trong file FreeRTOSConfig.h 3](#_Toc120304190)

[1.3. Hàm in thông tin của Task ứng với ID 3](#_Toc120304191)

[1.4. Hàm dành cho task liên tục (có Priority bằng nhau) 3](#_Toc120304192)

[1.5. Hàm danh cho task có Priority cao hơn 3](#_Toc120304193)

[1.6. Hàm main 4](#_Toc120304194)

[1.7. Kết quả hiện thực và biểu đồ thời gian 5](#_Toc120304195)

[**2.** **Prioritized Pre-emptive Scheduling (without Time Slicing)** 5](#_Toc120304196)

[2.1. Điều chỉnh trong menuconfig 5](#_Toc120304197)

[2.2. Điều chỉnh trong file FreeRTOSConfig.h 5](#_Toc120304198)

[2.3. Phần hiện thực Code 5](#_Toc120304199)

[2.4. Kết quả hiện thực và biểu đồ thời gian 5](#_Toc120304200)

[**3.** **Co-operative Scheduling** 6](#_Toc120304201)

[3.1. Điều chỉnh trong menuconfig 6](#_Toc120304202)

[3.2. Điều chỉnh trong file FreeRTOSConfig.h 6](#_Toc120304203)

[3.3. Hàm in thông tin của Task ứng với ID 6](#_Toc120304204)

[3.4. Hàm task 7](#_Toc120304205)

[3.5. Hàm main 8](#_Toc120304206)

[3.6. Kết quả hiện thực và biểu đồ thời gian 8](#_Toc120304207)

[LINK GITHUB 10](#_Toc120304208)

**ĐÓNG GÓP CỦA CÁC THÀNH VIÊN**

|  |  |  |
| --- | --- | --- |
| **Họ và tên** | **MSSV** | **Đóng góp** |
| Trần Nguyễn Minh Duy | 1910095 | 100% |
| Đặng Trung Kiên | 1911437 | 100% |
| Nguyễn Hải Long | 1911517 | 100% |
| Nguyễn Nhật Trường | 1912344 | 100% |

**BÁO CÁO**

## **1. Ý tưởng hiện thực**

### *1.1. Mô tả queue*

// struct data for queue

typedef struct

{

*uint8\_t* ID;

*uint32\_t* val;

} *Data*;

// this variable hold queue handle

xQueueHandle xQueue;

* Ta tạo 1 struct dữ liệu là Data với:
  + ID: giúp các task lấy dữ liệu từ queue kiểm tra xem dữ liệu này có dành cho task đó hay không (bằng cách so sánh ID của data với ID của task).
  + val: giá trị bổ sung để thực hiện request.

### *1.2. Tạo request*

// variables and constants for request

/\*

    0 - get temperature

    1 - Turn on LED

    2 - Turn off LED

    the other - unacceptable request

\*/

#define GET\_TEMP        0

#define TURN\_ON\_LED     1

#define TURN\_OFF\_LED    2

#define N\_REQUEST       3

*uint8\_t* request[N\_REQUEST] = {0, 0, 0};

// check if there is any request

bool flagRequest = false;

void vRequest()

{

*uint8\_t* i = 0;

    while (1)

    {

        // create 3 requests every 5 seconds

        // create request randomly from 0 to 3

        // if request == 3 then error

        bootloader\_random\_enable();

        request[0] = esp\_random() % 4;

        request[1] = esp\_random() % 4;

        request[2] = esp\_random() % 4;

        bootloader\_random\_disable();

        // set flag to notify there is request

        flagRequest = true;

        vTaskDelay(5000 / portTICK\_RATE\_MS);

    }

    vTaskDelete(NULL);

}

* Để mô phỏng việc tạo request, ta giả sử có 3 loại request:
  + Request = 0: Lấy giá trị nhiệt độ.
  + Request = 1: Bật đèn.
  + Request = 2: Tắt đèn.
  + Request = các giá trị khác: request không hợp lệ.
* Tiếp theo, ta tạo mảng **request[3]**, để lưu request.
* Cứ mỗi 5 giây, ta sẽ tạo ra 3 request bằng hàm **vRequest**. Ta tạo bằng cách gán biến request bằng 1 số ngẫu nhiên từ 0 đến 3, với request = 3 là không hợp lệ).
* Sau khi tạo request, ta sẽ set cờ **flagRequest** để báo hiệu rằng đã tạo request. Bằng cờ **flagRequest**, Repception task sẽ biết rằng có request mới hay không.

### *1.3. Functional task id = 0: Task lấy nhiệt độ*

void vGetTemp(void \**parameter*)

{

    // keep the status of receiving data

*BaseType\_t* xStatus;

    // time to block the task until data is available

    const *TickType\_t* xTicksToWait = pdMS\_TO\_TICKS(100);

*Data* data;

*uint8\_t* id = (*uint8\_t*)*parameter*;

*uint32\_t* temperature = 0;

    while (1)

    {

        // Peek data from the queue

        // to check if the next request is for it

        xStatus = xQueuePeek(xQueue, &data, xTicksToWait);

        if (xStatus == pdPASS)

        {

            // check if request

            if (data.ID == id)

            {

                // get random temperature

                bootloader\_random\_enable();

                temperature = 25 + (esp\_random() % 5);

                bootloader\_random\_disable();

                // pop data from the queue

                xStatus = xQueueReceive(xQueue, &data, xTicksToWait);

                if (xStatus == pdPASS)

                {

                    printf("Response: %d\*C\n", temperature);

                }

                else

                {

                    printf("Response: Cound not recieve data\n");

                }

            }

        }

        vTaskDelay(50 / portTICK\_RATE\_MS);

    }

    vTaskDelete(NULL);

}

* Chức năng chính: nếu có request = 0 (lấy nhiệt độ), task sẽ trả về (in ra) giá trị của nhiệt độ.
* Cách hoạt động: task sẽ kiểm tra xem ID (struct Data) của phần tử đầu của Queue có bằng 0 (id của task) hay không.
  + Nếu có, task sẽ tạo ra 1 giá trị nhiệt độ temperature ngẫu nhiên từ 25 tới 29, và in giá trị nhiệt độ. Sau đó, sẽ lấy (pop) phần tử data đã kiểm tra ra khỏi queue.
  + Nếu không, task sẽ bỏ qua.

### *1.4. Functional task id = 1: Task điều khiển LED*

void vSetLed(void \**parameter*)

{

    // keep the status of receiving data

*BaseType\_t* xStatus;

    // time to block the task until data is available

    const *TickType\_t* xTicksToWait = pdMS\_TO\_TICKS(100);

*Data* data;

*uint8\_t* id = (*uint8\_t*)*parameter*;

    while (1)

    {

        // Peek data from the queue

        // to check if the next request is for it

        xStatus = xQueuePeek(xQueue, &data, xTicksToWait);

        if (xStatus == pdPASS)

        {

            // check if request

            if (data.ID == id)

            {

                xStatus = xQueueReceive(xQueue, &data, xTicksToWait);

                // pop data from the queue

                if (xStatus == pdPASS)

                {

                    printf("Response: Set LED to %d\n", data.val);

                }

                else

                {

                    printf("Response: Cound not recieve data\n");

                }

            }

        }

        vTaskDelay(50 / portTICK\_RATE\_MS);

    }

    vTaskDelete(NULL);

}

* Chức năng chính: tắt/bật đèn dựa trên giá trị val của struct Data.
* Cách hoạt động: task sẽ kiểm tra xem ID (struct Data) của phần tử đầu của Queue có bằng id = 1 hay không.
  + Nếu có, task dựa vào giá trị val (struct Data) để bật (nếu val = 1) hoặc tắt (nếu val = 0) đèn.
  + Nếu không, task sẽ bỏ qua.

### *1.4. Reception task*

void vReceptionTask(void)

{

    /\* keep the status of sending data \*/

*BaseType\_t* xStatus;

    /\* time to block the task until the queue has free space \*/

    const *TickType\_t* xTicksToWait = pdMS\_TO\_TICKS(100);

    /\* Data is added to queue\*/

*Data* requestedData;

*uint8\_t* i = 0;

    while (1)

    {

        // check if there is any request

        if (flagRequest)

        {

            // classify 3 tasks,

            // then add data to queue

            for (i = 0; i < N\_REQUEST; i++)

            {

                if (request[i] == GET\_TEMP)

                {

                    requestedData.ID = 0;

                    requestedData.val = 0;

                    printf("Request: get temperature\n");

                }

                else if (request[i] == TURN\_ON\_LED)

                {

                    requestedData.ID = 1;

                    requestedData.val = 1;

                    printf("Request: Turn on LED\n");

                }

                else if (request[i] == TURN\_OFF\_LED)

                {

                    requestedData.ID = 1;

                    requestedData.val = 0;

                    printf("Request: Turn off LED\n");

                }

                else

                {

                    // If no functional task receives the request,

                    // raise an error

                    requestedData.ID = request[i];

                    requestedData.val = 0;

                    printf("Request %d: Error\n", request[i]);

                    // unset flag, and ignore that request

                    flagRequest = false;

                    continue;

                }

                // send data to queue

                xStatus = xQueueSendToFront(xQueue, &requestedData, xTicksToWait);

                // check if sending is ok or not

                if (xStatus == pdPASS)

                {

                    ;

                }

                else

                {

                    printf("Request %d: Could not send data\n", request);

                }

            }

            // unset flag

            flagRequest = false;

        }

        vTaskDelay(50 / portTICK\_RATE\_MS);

    }

    vTaskDelete(NULL);

}

* Chức năng chính: task nhận request và phân loại các request này và gửi chúng đến hàng đợi để các functional task xử lý
* Cách hoạt động:
  + Nếu có request mới được tạo (flagRequest = 1), task sẽ tiến hành phân loại, và thêm data vào Queue với id, và val tương ứng với request đó.
    - Nếu request = 0: ID = 0 (tương ứng với task lấy nhiệt độ), val = x (tùy ý).
    - Nếu request = 1: ID = 1 (tương ứng với task điều khiển LED), val = 1 (bật đèn).
    - Nếu request = 2: ID = 1 (tương ứng với task điều khiển LED), val = 0 (tắt đèn).
    - Nếu request == các giá trị còn lại: báo lỗi, vào bỏ qua.
  + Hạ cờ flagRequest (flagRequest = 0), để chờ request tiếp theo.

### *1.5. Hàm main*

void app\_main()

{

    /\* create the queue which size can contains 5 elements of Data \*/

    xQueue = xQueueCreate(5, sizeof(Data));

    xTaskCreate(vRequest, "vRequest", 2048, NULL, 3, NULL);

    xTaskCreate(vReceptionTask, "vReceptionTask1", 2048, NULL, 2, NULL);

    xTaskCreate(vGetTemp, "vGetTemp1", 2048, (uint8\_t \*)0u, 1, NULL); // task get temp, id = 0

    xTaskCreate(vSetLed, "vSetLed", 2048, (uint8\_t \*)1u, 1, NULL);    // task set led, id = 1

}

* Đầu tiên, ta tạo 1 queue với size = 5.
* Tiếp theo, ta tạo các task như sau:
  + Request task: có priority lớn nhất.
  + Reception task: task có priority lớn thứ nhì.
  + Functional task thứ nhất: task lấy nhiệt độ, priority nhỏ nhất = 1, và bằng priority với các functional task khác.
  + Functional task thứ hai: task điều khiển đèn, priority nhỏ nhất = 1, và bằng priority với các functional task khác.

## **2. Kết quả**