ĐẠI HỌC QUỐC GIA THÀNH PHỐ HỒ CHÍ MINH
TRƯỜNG ĐẠI HỌC BÁCH KHOA
KHOA KHOA HỌC VÀ KỸ THUẬT MÁY TÍNH

THÍ NGHIỆM

VI XỬ LÍ - VI ĐIỀU KHIỂN (CO 3010)

BÁO CÁO LAB 4

Giảng viên: PHAN VĂN SỸ

Họ và tên	Mã số sinh viên
Nguyễn Văn Sang	2212912

TP.Hồ Chí Minh, 19/11/2025

Mục lục

1 Hàm void SCH_Update(void)

Chức năng của hàm này là cập nhật trạng thái thời gian của các task trong scheduler.

Trong bài tập này SCH_MAX_TASKS được định nghĩa mang giá trị là 8.

1.1 Hàm ban đầu

```
// ----- sche.h ----- //
#ifndef SCH_MAX_TASKS
#define SCH_MAX_TASKS 8
#endif

// ----- sche.c ----- //
void SCH_Update(void) {
    for (int i = 0; i < SCH_MAX_TASKS; i++) {
        if (SCH_tasks_G[i].pTask != NULL) {
            if (SCH_tasks_G[i].Delay == 0) {
                SCH_tasks_G[i].RunMe += 1;
                if (SCH_tasks_G[i].Period != 0) {
                    SCH_tasks_G[i].Delay = SCH_tasks_G[i].Period;
                }
            } else {
                SCH_tasks_G[i].Delay -= 1;
            }
        }
    }
}

// ----- main.c ----- //
void HAL_TIM_PeriodElapsedCallback(TIM_HandleTypeDef *htim) {
    if (htim->Instance == TIM2) {
        SCH_Update();
  }
}
```

- pTask != NULL: Task tồn tại ở slot i khi đó chỉ xử lý các slot đã được thêm task.
- Delay == 0: Thời gian chờ đã hết dẫn đến task đã đến lúc được chạy.
- RunMe += 1: Tăng cờ "cần chạy" lên 1 sau đó báo cho SCH_Dispatch_Tasks() biết cần thực thi task này.
- Period != 0: Task là định kỳ nên sau khi chạy xong, cần lên lịch chạy lại.
- Delay = Period: Nạp lại bộ đếm chờ sau đó task sẽ chạy lại sau đúng Period tick.
- Delay > 0: Chưa đến lúc chạy thì giảm bộ đếm chờ đi 1 đơn vị (1ms).

Độ phức tạp của hàm là O(n) vì:

- Phụ thuộc vào n để tiến hành duyệt với n = SCH_MAX_TASKS.
- Mỗi lần gọi đều duyệt toàn bộ mảng, bất kể có bao nhiêu task thực sự đang hoạt động.

1.2 Hàm sau cải tiến

```
// ----- sche.c ----- //
void SCH_Update(void) {
    static uint32_t currentIndex = 0;
```

```
    if (SCH_tasks_G[currentIndex].pTask != NULL) {
        if (SCH_tasks_G[currentIndex].Delay == 0) {
            SCH_tasks_G[currentIndex].RunMe += 1;
            if (SCH_tasks_G[currentIndex].Period != 0) {
                SCH_tasks_G[currentIndex].Delay = SCH_tasks_G[currentIndex].
                    Period;
            }
        } else {
            SCH_tasks_G[currentIndex].Delay -= 1;
        }
    }
    currentIndex = (currentIndex + 1) % SCH_MAX_TASKS;
}

// ----- main.c ----- //
void HAL_TIM_PeriodElapsedCallback(TIM_HandleTypeDef *htim) {
    if (htim->Instance == TIM2) {
          SCH_Update();
          SCH_Update();
          SCH_Update();
          SCH_Update();
          SCH_Update();
          SCH_Update();
          SCH_Update();
          SCH_Update();
  }
}
```

Sử dụng biến tĩnh currentIndex để chỉ cập nhật một task duy nhất mỗi lần gọi. Cụ thể như sau:

- Nếu task tại currentIndex tồn tại (pTask != NULL): Nếu Delay == 0, tăng RunMe lên 1 và reset Delay = Period nếu Period != 0. Ngược lại nếu Delay > 0, giảm Delay đi 1.
- Sau đó tăng currentIndex modulo SCH_MAX_TASKS để cycle qua mảng ở lần gọi tiếp theo.
- Trong file main.c nếu ta vẫn sử dụng hàm void HAL_TIM_PeriodElapsedCallback(TIM_HandleTypeDef *htim) giống như khi hàm void SCH_Update(void) có độ phức tạp là O(n) thì các task sẽ bị chậm đi 8 lần. Khi đó hàm void HAL_TIM_PeriodElapsedCallback(TIM_HandleTypeDef *htim) phải được thay đổi bằng cách gọi lại 8 lần (bằng SCH_MAX_TASKS) trong callback của timer TIM2 (mỗi tick 1ms) để đảm bảo tất cả task được cập nhật đầy đủ trong một chu kỳ timer.

Độ phức tạp của hàm là O(1) vì:

- Mỗi lần gọi chỉ xử lý một task duy nhất tại currentIndex, không lặp qua toàn bộ mảng.
- Việc cycle qua mảng được phân bổ qua nhiều lần gọi, nên độ phức tạp của từng hàm gọi riêng lẻ là hằng số.

2 Hàm void SCH_Dispatch_Tasks(void)

```
// ----- sche.c ----- //
void SCH_Dispatch_Tasks(void) {
    for (int i = 0; i < SCH_MAX_TASKS; i++) {
        if (SCH_tasks_G[i].RunMe > 0 && SCH_tasks_G[i].pTask != NULL) {
            (*SCH_tasks_G[i].pTask)();
            SCH_tasks_G[i].RunMe -= 1;
```

```
            if (SCH_tasks_G[i].Period == 0) {
                SCH_Delete_Task(i);
            }
        }
    }
}
```

Chức năng của hàm giúp thực thi các task sẵn sàng trong scheduler. Nó lặp qua toàn bộ mảng SCH_tasks_G, nếu một task có RunMe > 0 và pTask != NULL, hàm sẽ:

- Gọi hàm task ((*SCH_tasks_G[i].pTask)()).
- Giảm RunMe đi 1.
- Nếu Period == 0 (task chỉ chạy một lần), gọi SCH_Delete_Task(i) để xóa task. Hàm này được gọi trong vòng lặp vô hạn của main() để liên tục kiểm tra và chạy task khi chúng sẵn sàng (sau khi SCH_Update() đã cập nhật RunMe). Nó đảm bảo scheduler hoạt động không ưu tiên, chạy task theo thứ tự trong mảng.

Độ phức tạp của hàm là O(n), trong đó n = SCH_MAX_TASKS vì:

- Hàm lặp tuyến tính qua toàn bộ mảng để kiểm tra và thực thi task. Vì n nhỏ, độ phức tạp thực tế gần O(1), nhưng lý thuyết là O(n).
- Việc gọi hàm task bên trong có thể thêm độ phức tạp tùy thuộc vào task, nhưng độ phức tạp của chính SCH_Dispatch_Tasks() chỉ tính vòng lặp.

3 Hàm uint32_t SCH_Add_Task(void (* pFunction)(), uint32_t DELAY, uint32_t PERIOD)

```
// ----- sche.c ----- //
void SCH_Dispatch_Tasks(void) {
    for (int i = 0; i < SCH_MAX_TASKS; i++) {
        if (SCH_tasks_G[i].RunMe > 0 && SCH_tasks_G[i].pTask != NULL) {
            (*SCH_tasks_G[i].pTask)();
            SCH_tasks_G[i].RunMe -= 1;
            if (SCH_tasks_G[i].Period == 0) {
                SCH_Delete_Task(i);
            }
        }
    }
}
```

Chức năng của hàm dùng để thêm một task mới vào scheduler. Nó tìm kiếm một vị trí trống trong mảng SCH_tasks_G (tức là nơi pTask == NULL). Nếu tìm thấy, hàm sẽ gán các giá trị sau cho task tại vị trí đó:

- pTask = pFunction: con trỏ đến hàm task cần thực thi.
- Delay = DELAY: độ trễ ban đầu trước khi task chạy lần đầu, tính bằng đơn vị tick.
- Period = PERIOD: chu kỳ lặp lại task sau lần chạy đầu tiên; nếu PERIOD == 0, task chỉ chạy một lần.

- RunMe = 0: cờ chỉ task chưa sẵn sàng chạy. Hàm trả về chỉ số (index) của task trong mảng nếu thêm thành công, hoặc SCH_INVALID_TASK (0xFFFFFFFFu) nếu mảng đã đầy (không còn vị trí trống).
- Trong file main.c, hàm này được sử dụng để thêm các task như process_timers, process_input, và execute_fsm với độ trễ ban đầu là 0 và chu kỳ là 1 (chạy liên tục).

Độ phức tạp của hàm là O(n), trong đó n = SCH_MAX_TASKS vì:

- Hàm lặp tuyến tính qua mảng để tìm vị trí trống đầu tiên.
- n là hằng số nhỏ, độ phức tạp thực tế gần như O(1) trong hầu hết các trường hợp, nhưng về mặt lý thuyết vẫn là O(n) do vòng lặp.

4 Hàm uint8_t SCH_Delete_Task(uint32_t taskID)

```
// ----- sche.c ----- //
uint8_t SCH_Delete_Task(uint32_t taskID) {
    if (taskID >= SCH_MAX_TASKS) {
        return 0;
    }
    SCH_tasks_G[taskID].pTask = NULL;
    SCH_tasks_G[taskID].Delay = 0;
    SCH_tasks_G[taskID].Period = 0;
    SCH_tasks_G[taskID].RunMe = 0;
    return 1;
}
```

Chức năng của hàm là dùng để xóa một task khỏi scheduler dựa trên chỉ số taskID. Nếu taskID hợp lệ (nhỏ hơn SCH_MAX_TASKS), hàm sẽ đặt lại các giá trị của task tại vị trí đó về mặc định:

- pTask = NULL (đánh dấu vị trí trống).
- Delay = 0.
- Period = 0.
- RunMe = 0.
- Hàm trả về 1 nếu xóa thành công (taskID hợp lệ) hoặc 0 nếu taskID không hợp lệ (vượt quá giới hạn mảng). Hàm này được gọi tự động trong SCH_Dispatch_Tasks() nếu một task có Period == 0 sau khi chạy, để xóa các task chỉ chạy một lần.

Độ phức tạp của hàm là O(1) vì hàm chỉ kiểm tra điều kiện và truy cập trực tiếp vào phần tử mảng tại chỉ số taskID, không có vòng lặp hoặc hoạt động phụ thuộc vào kích thước mảng.

5 Tổng hợp LAB 4

Link video demo: https://drive.google.com/file/d/1iXA3bZZBbTN_pZSBT3JlWQ2iapAxOgyc/view?usp=sharing
Link Github: https://github.com/SangNguyen-232/VXLVDK-LAB4/tree/main