HỌC VIỆN KỸ THUẬT MẬT MÃ
KHOA CÔNG NGHỆ THÔNG TIN

BÀI TẬP MÔN LẬP TRÌNH DRIVER

LẬP TRÌNH DRIVER I2C CHO MÀN HÌNH LCD 1602 TRÊN LINUX

Khoa: Công nghệ thông tin
Chuyên ngành: Kỹ thuật phần mềm nhúng và di động

Người hướng dẫn
Giảng viên: Triệu Vũ Anh Quân

Nhóm sinh viên thực hiện:

Phạm Văn Dũng	CT040308
Phạm Thị Phương Anh	CT040401
Lại Phương Thảo	CT040445
Cao Văn Giáp	CT030317

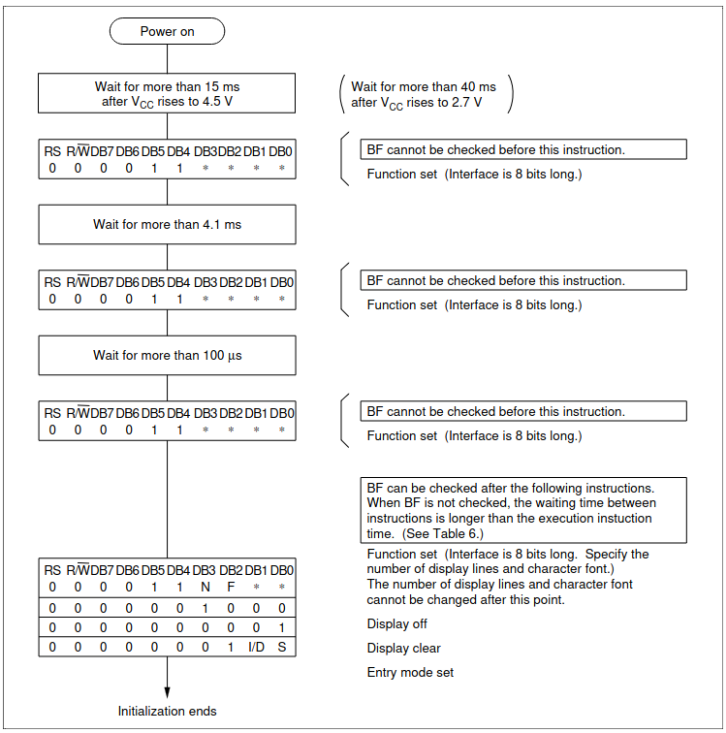
Nhóm 02

Hà Nội - 2023

Chương 1: GIỚI THIỆU CÔNG CỤ

1.1 Màn hình LCD1602 HD44780

Màn hình LCD1602 là màn hình ma trận điểm với 2 hàng, mỗi hàng chứa 16 ký tự kích thước 5x8 điểm. HD44780 là một IC điều khiển màn hình ma trận 5x8 hoặc 5x10 với tối đa 80 ký tự, phù hợp với màn hình LCD1602. Việc điều khiển yêu cầu tối thiểu 6 chân GPIO. Các bước khởi tạo màn hình.



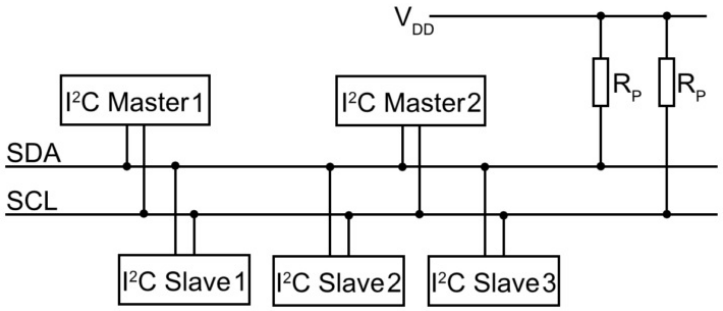
Hình 1.1: Khởi tạo HD44780

1.2 Module mở rộng GPIO PCF8574 cho LCD1602

I2C là một chuẩn được phát triển bởi Philips. Nó là một giao thức 2 dây chậm với tốc độ có thể lên đến 400 kHz. Nó được sử dụng rộng rãi trong hệ thống nhúng. Một số thiết bị sử dụng biến thể không đạt được yêu cầu của I2C nên có thể được gọi bằng tên khác như TWI hoặc IIC.

SMBus (System Management Bus) dựa trên giao thức I2C được sử dụng rộng rãi trong hệ thống máy tính. Rất nhiều thiết bị I2C có thể hoạt động trên SMBus như I2C EEPROMs và một số chip theo dõi phần cứng.

Trong một bus I2C, có thể có một hay nhiều master chip được kết nối tới một hay nhiều slave chip.



Hình 1.2: I2C bus

Master chip là thiết bị kết nối tới slave. Trong nhân linux, master chip được gọi là một adapter hoặc bus. Các trình điều khiển của adapter nằm trong thư mục con drivers/i2c/busses.

Một algorithm chứa mã có thể sử dụng để cài đặt nhiều loại I2C adapter. Mỗi adapter driver có thể phụ thuộc vào một algorithm driver trong thư mục con drivers/i2c/algos hoặc chứa mã cho riêng nó.

Một slave chip là nút thực hiện giao tiếp khi có yêu cầu của master. trong Linux, nó được gọi là client. Client driver được lưu trong thư mục nào phụ thuộc vào các chức năng mà nó cung cấp, ví dụ như trong drivers/media/gpio cho thiết bị mở rộng GPIO.

Để đơn giản hóa việc điều khiển một thiết bị sử dụng nhiều GPIO, module mở rộng chân sử dụng kết nối nối tiếp sẽ giúp đơn giản hóa việc kết nối và tiết kiệm số chân GPIO. PCF8574 sử dụng giao tiếp I2C có 8 địa chỉ có thể thay đổi cứng. Địa chỉ mặc định là 0x27 và có 8 chân GPIO song song. Việc đọc 8 chân GPIO có thể thực hiện bằng việc đọc 8 bit từ module PCF8574, tương tự, ghi vào 8 chân GPIO được thực hiện bằng việc ghi 8 bit tương ứng.

1.3 Phát triển I^{2}C device driver

Linux kernel driver là một module được nạp vào nhân linux và chạy trên không gian địa chỉ của nhân linux giúp cung cấp các chức năng mở rộng cho nhân hệ điều hành.

Device driver trong linux là một kernel module cung cấp giao diện giao tiếp với các thiết bị phần cứng như USB, PCI, I2C, SPI, … Device driver thực hiện việc giao tiếp như gửi và nhận dữ liệu, từ thiết bị, đảm bảo cho thiết bị được hoạt động đúng mục đích đặt ra.

Khác với các thiết bị như PCI hay USB, thiết bị sử dụng giao tiếp I2C mặc định không được nhận biết bởi phần cứng mà thay vào đó phần mềm phải nhận biết những thiết bị nào được kết nối ở địa chỉ nào. Có một số cách để thực hiện việc này nhưng một phương pháp hiệu quả là sử dụng Device tree.

Device tree là một cấu trúc dsữ liệu và một hệ thống mô tả phần cứng được sử dụng trong hệ điều hành linux. Trong Device tree, các thiết bị I2C được khai báo tại bus và địa chỉ xác định. Cùng với thông tin bus và địa chỉ, Device tree còn mô tả loại driver tương thích với thiết bị. Khi trình điều khiển thiết bị được nạp, hàm probe sẽ được gọi khi trình điều khiển tương thích với thiết bị được khai báo trong Device tree.

Một Device tree cho thiết bị slave I2C có cấu trúc dạng như sau: thiết bị i2c-device được kết nối tới i2c-bus có địa chỉ 0x50. Driver tương thích là “vendor,i2c-device”:

```
1        i2c {
2                compatible = "i2c-bus";
3                #address-cells = <1>;
4                #size-cells = <0>;
5                i2c-device@50 {
6                        compatible = "vendor,i2c-
                           device";
7                        reg = <0x50>;
8                }
9        }
```

Bên trong driver, sau khi hàm probe được gọi với tham số struct i2c_client tương ứng, thực hiện lưu lại biến này để thực hiện giao tiếp với thiết bị. Để đọc từ i2c device, sử dụng hàm i2c_smbus_read*, và i2c_smbus_write* cho việc ghi.

1.4 Giao tiếp với device driver

Một số lựa chọn để giao tiếp với device driver của thiết bị nhúng bao gồm:

- Sử dụng Character device driver
- Sử dụng sysfs
- Kết hợp các phương pháp

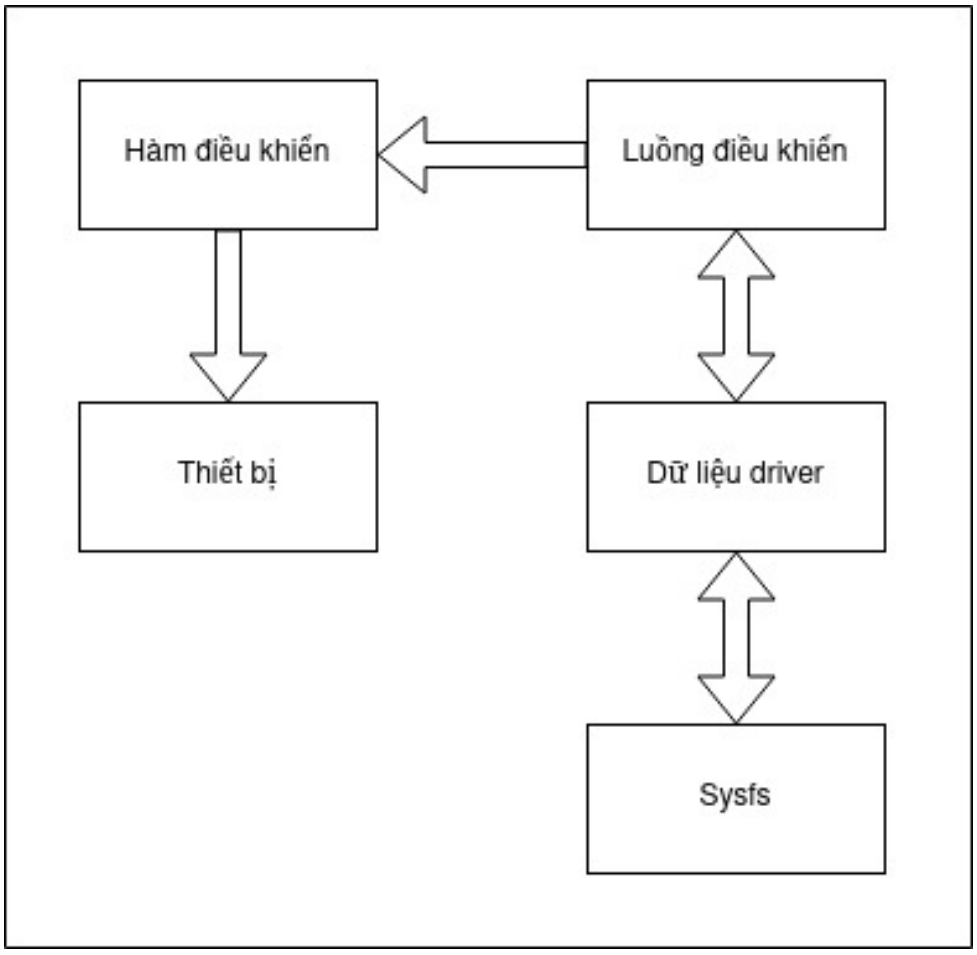
Với character device driver thiết bị được xuất hiện trên không gian người dùng với một file duy nhất bên trong /dev. Các thao tác có thể được thực hiện thông qua các lời gọi từ struct file_operations như open, close, read, write, ioctl. Thông qua device file, thiết bị dạng vào/ra sẽ đơn giản hơn thông qua các lời gọi hệ thống. Một số ứng dụng sử dụng device file bao gồm: Giao tiếp với cổng serial, đọc từ ổ đĩa, …

Khác với character device driver, ứng dụng có thể sử dụng sysfs cho việc thiết lập và quản lý cấu hình. Các file trong không gian người dùng xuất hiện dưới dạng thư mục trong /sys/kernel/. Bên trong thư mục này là các file cấu hình được tạo ra với device driver giúp chương trình trong không gian người dùng có thể tương tác với driver thông qua việc đọc ghi các file. Giao tiếp với thiết bị phần cứng sẽ dễ dàng hơn thông qua sysfs do phần cứng có nhiều tham số cần đọc và ghi và chúng được tách riêng chúng ra từng file.

Với phần cứng hỗ trợ cả cấu hình các tham số và vào ra lượng thông tin lớn, kết hợp cả character device driver và sysfs sẽ tận dụng được lợi thế của cả hai phương pháp. Các thao tác vào ra sẽ được thực hiện qua character device driver, các thao tác cấu hình sẽ được thực hiện thông qua hệ thống sysfs và thao tác ioctl của character device driver. Khi đó người dùng có thể làm quen với thiết bị thông qua sysfs và thực hiện các thao tác qua chúng. Khi lập trình, việc sử dụng character device driver sẽ đem lại hiệu quả cao hơn.

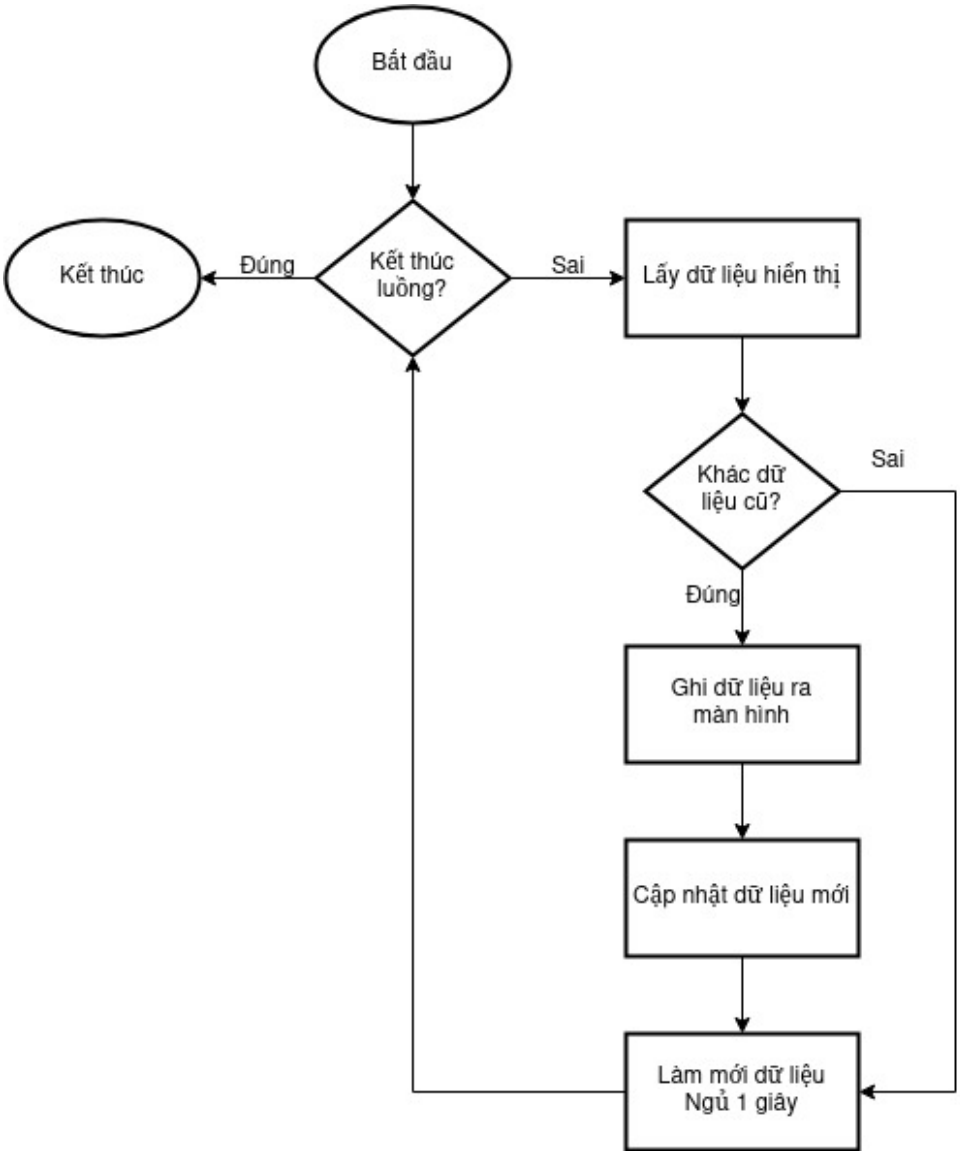
Chương 2: THIẾT KẾ DRIVER

Driver được triển khai trên nhiều file. File lcd_data.h chứa dữ liệu cho hoạt động điều khiển của driver (nội dung hiển thị, các dữ liệu điều khiển hiển thị, cuộn trang, ...). File lcd_sysfs.h giúp xuất dữ liệu driver và thay đổi dữ liệu driver theo yêu cầu của người dùng sử dụng hệ thống sysfs. File lcd_i2c_client.h chứa các hàm tương tác với thiết bị như điều khiển đèn nền, điều khiển cuộn trang hay thay đổi dữ liệu. File pcf_lcd.c là file chính của driver, thực hiện khai báo driver, khởi tạo luồng điều khiển, các hàm probe, remove.



Hình 2.1: Sơ đồ khối driver

Trong luồng điều khiển hoạt động của driver, thực hiện liên tục lấy dữ liệu hiển thị cho màn hình. Dữ liệu này có thể thay đổi sau mỗi lần lặp do dữ liệu mới được cập nhật hoặc sau khi làm mới dữ liệu, ở chế độ cuộn văn bản, dữ liệu hiển thị bị dịch chuyển sang ký tự khác. Sơ đồ luồng điều khiển được thể hiện trong hình 2.2.



Hình 2.2: Luồng điều khiển

Khi khai báo driver, thực hiện tạo đối tượng struct i2c_driver trong đó bao gồm các trường probe_new cho việc khởi tại thiết bị, .remove cho việc gỡ bỏ thiết bị, .id_table là đối tượng struct i2c_device_id cho thiết bị, .of_match_table là struct of_device_id chứa thông tin tên thiết bị tương thích với driver được lưu trong device tree.

```
66 static int lcd_probe(struct i2c_client *client) {
67   pr_info("Probed %x", client->addr);
68   lcd_client = client;
69   lcd_init(lcd_client);
70   lcd_sysfs_init();
71   task_struct = kthread_run(work_loop, NULL, "pcf lcd
       work loop");
72   set_content("LCD\n Hello!");
73   return 0;
74   return 0;
75 }
76 static int lcd_remove(struct i2c_client *client) {
77   set_content("LCD\n Goodbye!");
78   refresh();
79   msleep(refresh_timeout * 2);
80   kthread_stop(task_struct);
81   lcd_sysfs_deinit();
82
83   pr_info("Removed\n");
84   return 0;
85 }
86
87 static const struct of_device_id my_driver_ids[] = {
88     {.compatible = "pcf,lcd"},
89     {},
90 };
91 MODULE_DEVICE_TABLE(of, my_driver_ids);
92
93 static const struct i2c_device_id lcd_device_id[] = {
94     {DEVICE_NAME, 0},
95     {},
96 };
97
98 MODULE_DEVICE_TABLE(i2c, lcd_device_id);
```

```
99
100 static struct i2c_driver lcd_driver = {
101     .probe_new = lcd_probe,
102     .remove = lcd_remove,
103     .id_table = lcd_device_id,
104     .driver = {.name = DEVICE_NAME,
105                .owner = THIS_MODULE,
106                .of_match_table = my_driver_ids},
107 };
108
109 module_i2c_driver(lcd_driver);
```

Việc làm mới dữ liệu được thực hiện liên tục để phục vụ mục đích hiển thị dữ liệu mới từ người dùng hoặc phục vụ mục đích cuộn nội dung. Hàm làm mới dữ liệu được khai báo bên trong file chứa dữ liệu.

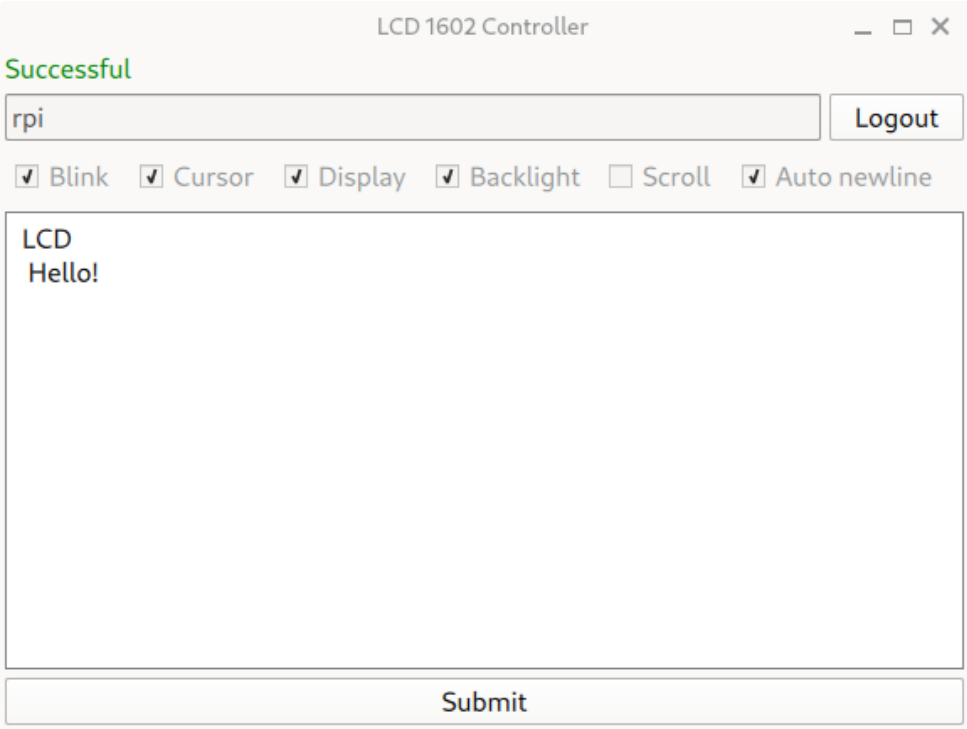
```
8  char __content[CONTENT_MAX_LEN + 2] = {0};
9  char __display_lines[100][101];
10 uint __nline;
11 uint __l1_idx = -1;
12 uint __l2_idx = -1;
13 uint __l1_off = -1;
14 uint __l2_off = -1;
15 uint __backlight = 1;
16 uint __cursor = 1;
17 uint __blink = 1;
18 uint __display = 1;
19 uint __auto_newline = 1;
20 uint __scroll = 0;
21 uint __scroll_start = 0;
22 uint __l1_scroll_done = 0;
23 uint __l2_scroll_done = 0;
```

```
143
144 void inc_offset(void) {
```

```
145   if (strlen(__display_lines[__l1_idx]) - __l1_off >
        16) {
146     __l1_off += 1;
147   } else {
148     __l1_off = 0;
149     __l1_scroll_done = 1;
150   }
151   if (strlen(__display_lines[__l2_idx]) - __l2_off >
        16) {
152     __l2_off += 1;
153   } else {
154     __l2_off = 0;
155     __l2_scroll_done = 1;
156   }
157 }
158
159 void refresh(void) {
160   if (get_scroll()) {
161     if (__scroll_start) {
162       if (__l1_scroll_done && __l2_scroll_done) {
163         __l1_idx = (__l1_idx + 2) % __nline;
164         __l2_idx = (__l2_idx + 2) % __nline;
165         __l1_off = 0;
166         __l2_off = 0;
167         __l1_scroll_done = 0;
168         __l2_scroll_done = 0;
169       } else {
170         inc_offset();
171       }
172     } else {
173       __scroll_start = 1;
174     }
175   }
176 }
```

Chương 3: THIẾT KẾ GIAO DIỆN

Để giúp việc điều khiển thiết bị được đơn giản hơn, nhóm thực hiện thiết kế một giao diện đơn giản dựa trên QT. Các thao tác trên giao diện là được thực hiện tương ứng với giao diện trong sysfs của driver.



Hình 3.1: Giao diện đồ họa cho thiết bị