

Xử lý event

Giảng Viên: Lưu An Phú



Agenda

- Tổng quan về event
- Xử lý event bằng cơ chế pooling
- Xử lý event bằng ngắt
- Xử lý exception



Tổng quan về event

2 phương pháp để xử lý event

- Xử lý theo cơ chế pooling.
 - Implement đơn giản.
 - Không hiệu quả, tốn cpu.
- Xử lý bằng interrupt và execption
 - Xử lý phức tạp hơn pooling.
 - Hiệu quả hơn.

- Trong source code có 1 vòng while liên tục kiểm tra sự kiện đã xảy ra hay chưa.
- Tạo một timer theard chạy định kỳ để kiểm tra.
 - struct timer_list
 - init_timer_on_stack
 - add_timer

Example code

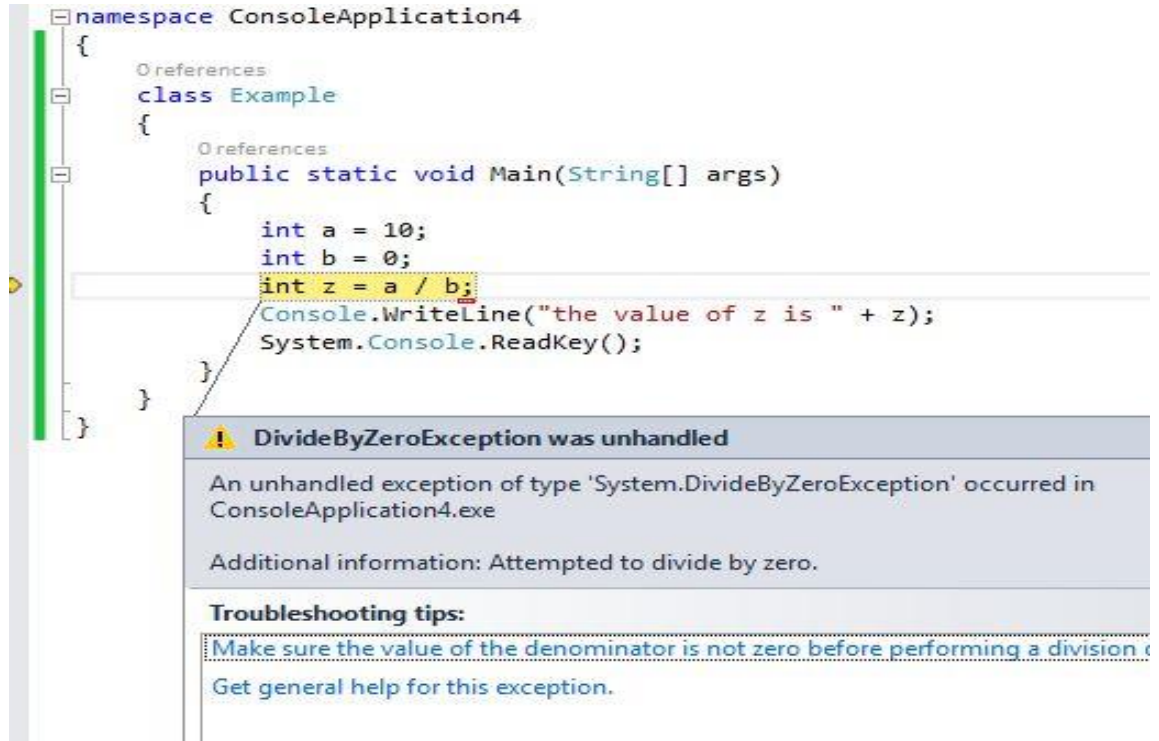
```
9 struct timer_list exp_timer;
10
11 static void do_something(unsigned long data)
12 {
13     printk(KERN_INFO "Your timer expired and app has been called\n");
14 }
15
16 static int __init tst_init(void)
17 {
18     int delay = 300;
19
20     printk(KERN_INFO "Init called\n");
21
22     init_timer_on_stack(&exp_timer);
23
24     exp_timer.expires = jiffies + delay * HZ;
25     exp_timer.data = 0;
26     exp_timer.function = do_something;
27
28     add_timer(&exp_timer);
29
30     return 0;
31 }
32
33 static void __exit tst_exit(void)
34 {
35     del_timer(&exp_timer);
36     printk(KERN_INFO "Exit called\n");
37 }
```

- Kết nối đèn led và button vào chân GPIO của board. Config GPIO cho cả đèn led và button. Viết 1 kernel module khởi tạo 1 timer check trạng thái chân GPIO của button mỗi 1ms. Nếu thấy button được nhấn thì bật đèn led. Nếu thấy button được nhả thì tắt đèn led.

Xử lý event bằng interrupt và exception

- Định nghĩa
- Ngắt cứng
- Ngắt mềm
- Ngắt có thể bỏ qua
- Ngắt không thể bỏ qua

- Định nghĩa



The screenshot shows a C# code editor with a file explorer on the left. The code is in a namespace named `ConsoleApplication4` and contains a class `Example` with a `Main` method. Inside `Main`, variables `a` and `b` are initialized to 10 and 0 respectively. The line `int z = a / b;` is highlighted in yellow, and a red squiggly line under the `b` indicates an error. Below the code, a message box is displayed with the title "DivideByZeroException was unhandled". The message states: "An unhandled exception of type 'System.DivideByZeroException' occurred in ConsoleApplication4.exe. Additional information: Attempted to divide by zero." It also includes troubleshooting tips: "Make sure the value of the denominator is not zero before performing a division" and "Get general help for this exception."

```
namespace ConsoleApplication4
{
    References
    class Example
    {
        References
        public static void Main(String[] args)
        {
            int a = 10;
            int b = 0;
            int z = a / b;
            Console.WriteLine("the value of z is " + z);
            System.Console.ReadKey();
        }
    }
}
```

! DivideByZeroException was unhandled

An unhandled exception of type 'System.DivideByZeroException' occurred in ConsoleApplication4.exe

Additional information: Attempted to divide by zero.

Troubleshooting tips:

Make sure the value of the denominator is not zero before performing a division

Get general help for this exception.

Vector table

- Nơi chứa địa chỉ của tất cả interrupt and exception handler.

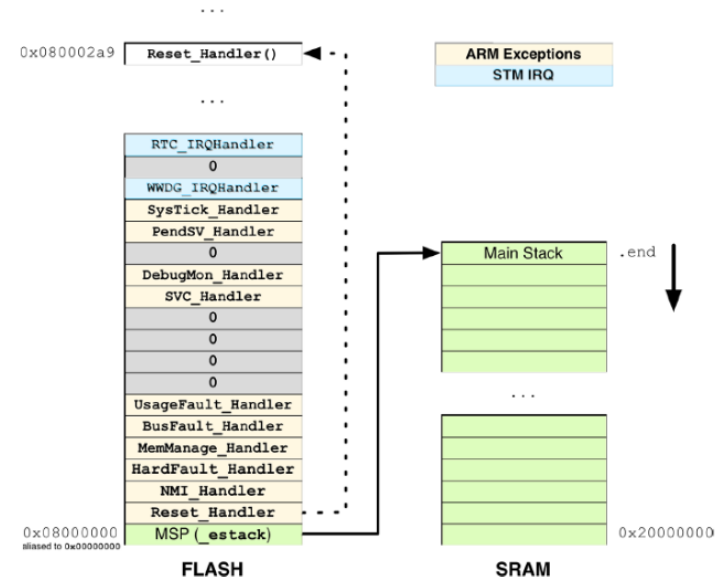


Figure 2: the minimal layout of the vector table in an STM32 MCU based on a Cortex-M3/4/7 core

- Là function được gọi ra khi có interrupt.
- Mỗi khi interrupt xảy ra, cpu stop câu lệnh hiện tại và save trạng thái vào stack. Sau đó thực thi interrupt.
 - Không được sleep trong interrupt handler.
 - Phải thực thi nhanh nhất có thể.
 - Thường chia sẻ data với các irq handler khác

Example code

```
1  irqreturn_t irq_handler(int irq, void *dev_id, struct pt_regs *regs)
2  {
3      → static int initialised = 0;
4      → static unsigned char scancode;
5      → static struct work_struct task;
6      → unsigned char status;
7
8      → return IRQ_HANDLED;
9  }
10
11 int init_module()
12 {
13     → free_irq(1, NULL);
14
15     → return request_irq(1, → /* The number of the keyboard IRQ on PCs */
16     → → → → → irq_handler, → /* our handler */
17     → → → → → SA_SHIRQ, "test_keyboard_irq_handler",
18     → → → → → (void *) (irq_handler));
19 }
```

View interrupt list in user-space

- `cat /proc/interrupt`

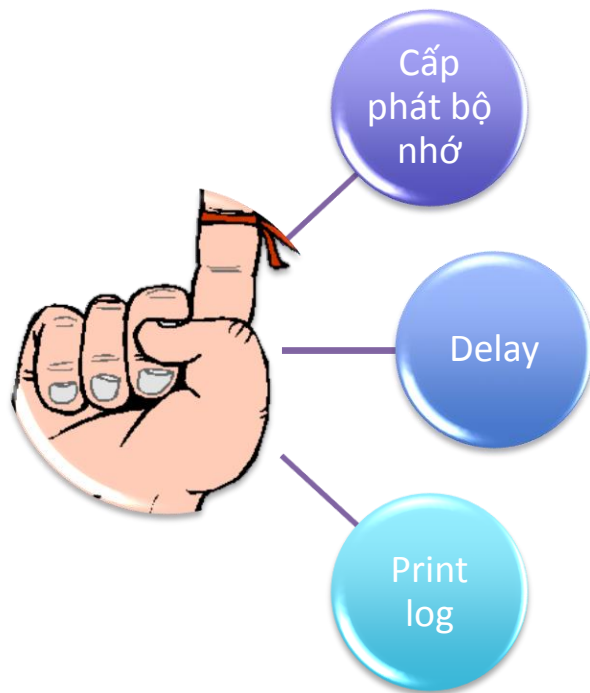
```
phula@alb-machine-test:~$ cat /proc/interrupts
```

	CPU0	CPU1	CPU2	CPU3			
0:	13	0	0	0	IO-APIC	2-edge	timer
1:	2	0	0	0	IO-APIC	1-edge	i8042
7:	0	0	0	0	IO-APIC	7-edge	parport0
8:	1	0	0	0	IO-APIC	8-edge	rtc0
9:	4	0	0	0	IO-APIC	9-fasteoi	acpi
17:	72	1701401	3908202	0	IO-APIC	17-fasteoi	enp3s0
18:	1044	7912246	2169330	0	IO-APIC	18-fasteoi	ehci_hcd:usb1
19:	6537	2834	715905	20997	IO-APIC	19-fasteoi	ata_piix, ata_piix
23:	29	0	0	0	IO-APIC	23-fasteoi	ehci_hcd:usb2
24:	106	95	2664599	2059	PCI-MSI	32768-edge	i915
25:	13	0	0	0	PCI-MSI	360448-edge	mei_me
26:	711	241	0	0	PCI-MSI	442368-edge	snd_hda_intel:card0
27:	1	640	8640	354	PCI-MSI	2097152-edge	enp4s0

- Kết nối đèn led và button vào chân GPIO của board. Config GPIO cho cả đèn led và button. Viết 1 kernel module khởi tạo 1 interrupt handler cho button. Mỗi khi button được nhấn thì gọi interrupt handler ra để bật led. Nếu button nhả ra thì gọi handler ra để tắt led.

- Top half
 - Xử lý những công việc critical, không thể delay được.
- Bottom half
 - Xử lý các việc cần nhiều thời gian, có thể delay được.
 - Mẹo: Trong interrupt handler gọi hàm `add_timer` để tạo 1 timer xử lý sau.

- Kmalloc with GFP_ATOMIC
- Delay() thay thế cho msleep()
- Printk()



Disable interrupt

- `local_irq_disable`
- `local_irq_save`
- `local_bh_disable`
- `spinlock_irq_get`

- Dùng cho các hàm có thể được gọi từ interrupt context và non-interrupt context.
- `Irqs_disabled()`
- `In_interrupt()`
- `In_irq()`

- Không thể bị bỏ qua
- Không bị ngắt quãng bởi exception khác
- Không cần protect data
- Sau khi kết thúc, có thể không chạy tiếp instruction trước đó.

Thank you

