

Event Driven

Erica Riello e Tiago Rabello

Sistemas Reativos

2015.1

Código do Usuário - Exemplo

```
#define LED_PIN 13
#define BUT_PIN 2
#define LED_T1_PIN 1
#define LED_T2_PIN 0
```

```
int led_timer_state = LOW;
int led_timer2_state = LOW;
int timerid1 = -1;
int timerid2 = -1;
```

```
void init_event_driven ()
{
    pinMode(LED_PIN, OUTPUT);
    pinMode(LED_T1_PIN, OUTPUT);
    pinMode(LED_T2_PIN, OUTPUT);
    pinMode(BUT_PIN, INPUT);
    digitalWrite(LED_PIN, LOW);
    digitalWrite(LED_T1_PIN, LOW);
    digitalWrite(LED_T2_PIN, LOW);
    button_listen(BUT_PIN);
    timerid1 = timer_set(2000);
    timerid2 = timer_set(5000);
}
```

```
void button_changed (int pin, int value)
{
    digitalWrite(LED_PIN, value);
}
```

```
void timer_expired (int timerId)
{
    if (timerId == timerid1)
    {
        led_timer_state = !led_timer_state;
        digitalWrite(LED_T1_PIN, led_timer_state);
    }
    else if (timerId == timerid2)
    {
        led_timer2_state = !led_timer2_state;
        digitalWrite(LED_T2_PIN,
led_timer2_state);
    }
}
```

Tarefa 2?!

- Piscar o LED a cada 1 segundo
- Botão 1: Acelerar o pisca pisca a cada pressionamento
- Botão 2: Desacelerar a cada pressionamento
- Botão 1+2 (em menos de 500ms): Parar

Código do Usuário - Tarefa 2

```
#define LED_PIN 13  
#define BUT1_PIN 3  
#define BUT2_PIN 2
```

```
int led_timer_state = HIGH;  
int led_time = 1000;  
int timerid1 = -1;  
int timerid2 = -1;
```

```
int teste_button_is_pressed[2];
```

```
void teste_stop()  
{  
    while (true);  
}
```

```
void teste_speed_up()  
{  
    led_time/=2;  
    change_timer(0, led_time);  
}
```

```
void teste_speed_down()  
{  
    led_time*=2;  
    change_timer(0, led_time);  
}
```

Código do Usuário - Tarefa 2 (Continuação)

```
void init_event_driven ()
{
    pinMode(LED_PIN, OUTPUT);
    pinMode(BUT1_PIN, INPUT);
    pinMode(BUT2_PIN, INPUT);
    digitalWrite(LED_PIN, HIGH);
    button_listen(BUT1_PIN);
    button_listen(BUT2_PIN);
    timerid1 = timer_set(1000);
    timerid2 = timer_set(500);
    deactivate_timer(timerid2);
}

void button_changed (int pin, int value)
{
    int index = -1;
    if (pin==BUT1_PIN) { index = 0; }
    else { index = 1; }
    teste_button_is_pressed[index] = value;
    if (value==1 &&
    teste_button_is_pressed[0]+teste_button_is_p
ressed[1]==1)
    {
        activate_timer(timerid2);
    }
}
```

```
void timer_expired (int timerId)
{
    if (timerId == 0)
    {
        led_timer_state = !led_timer_state;
        digitalWrite(LED_PIN, led_timer_state);
    }
    else /*if (timerId==1)*/
    {
        deactivate_timer(1);
        if
        (teste_button_is_pressed[0]+teste_button_is_p
ressed[1]==2)
        {
            teste_stop();
        }
        else if (teste_button_is_pressed[0]==1)
        {
            teste_speed_up();
        }
        else if (teste_button_is_pressed[1]==1)
        {
            teste_speed_down();
        }
    }
}
```

Implementação de Event Driven

Globais

```
#define maxTimers 2 /* numero maximo de timers */
#define maxButtons 2 /* numero maximo de botoes */

int button_is_pressed[maxButtons]; /* indica se o botao esta pressionado */
int button_pins[maxButtons]; /* pinos aos quais estao conectados os botoes */
int nextButton = 0; /* proximo botao */

int timer_is_active[maxTimers]; /* indica se o timer esta ativo */
int timers[maxTimers]; /* tempo de timeout do timer */
int timersRunning[maxTimers]; /* tempo de execucao do timer */
int nextTimer = 0; /* proximo timer */
int last_time = 0; /* ultimo tempo capturado */

/* inicializacao */
void setup (void)
{
    init_event_driven();
}

/* loop de execucao */
void loop (void)
{
    checkButtonEvents();
    checkTimerEvents();
}
```

Registro de Listeners

```
/* registra listener para um botao se houver espaco no buffer */  
void button_listen(int pin)  
{  
    if (nextButton < maxButtons)  
    {  
        button_pins[nextButton] = pin;  
        nextButton++;  
    }  
}
```

```
/* registra listener para timer se houver espaco no buffer */  
int timer_set(int ms)  
{  
    if (nextTimer < maxTimers)  
    {  
        timer_is_active[nextTimer] = 1;  
        timers[nextTimer] = ms;  
        timersRunning[nextTimer] = 0;  
        nextTimer++;  
        return nextTimer-1;  
    }  
    return -1;  
}
```


Verificação de eventos

```
/* verifica estouro dos timers ativos */
void checkTimerEvents ()
{
    int current_time = millis();
    int diff = current_time - last_time;
    int n;
    for (n=0; n<nextTimer; n++)
    {
        if (timer_is_active[n])
        {
            timersRunning[n] += diff;
            if (timersRunning[n]>=timers[n])
            {
                timersRunning[n] %= timers[n];
                timer_expired(n);
            }
        }
    }
    last_time = current_time;
}
```

```
/* verifica mudanca de tempo dos botoes */
void checkButtonEvents()
{
    int n;
    for (n=0; n<nextButton; n++)
    {
        int button_input_pressed = digitalRead(button_pins[n]);
        if (button_is_pressed[n]!=button_input_pressed)
        {
            button_is_pressed[n] = button_input_pressed;
            button_changed(button_pins[n], button_is_pressed[n]);
        }
    }
}
```

Métodos extra

```
/* ativa timer */  
void activate_timer(int timerId)  
{  
    if (timerId < nextTimer)  
    {  
        timer_is_active[timerId] = 1;  
        timersRunning[timerId] = 0;  
    }  
}
```

```
/* desativa timer */  
void deactivate_timer(int timerId)  
{  
    if (timerId < nextTimer)  
    {  
        timer_is_active[timerId] = 0;  
    }  
}
```

```
/* muda tempo de timeout do timer */  
void change_timer(int timerId, int time)  
{  
    if (timerId < nextTimer)  
    {  
        timers[timerId] = time;  
    }  
}
```

;-)