Sistemas Reativos

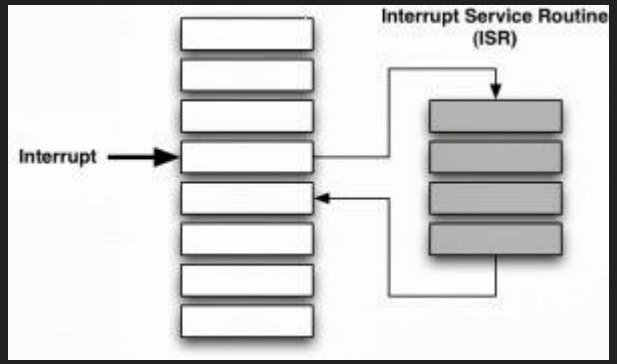
Interrupções

O que são?

- “Callbacks” feitas pelo hardware
  - Só que não exatamente…
- Respostas a estímulos externos ao programa principal
  - Geralmente…

Como?

- Hardware literalmente interrompe o programa que está em execução




```
$000    rjmp    RESET       ; Reset Handler
$001    rjmp    EXT_INT0    ; IRQ0 Handler
$002    rjmp    EXT_INT1    ; IRQ1 Handler
$003    rjmp    TIM2_COMP   ; Timer2 Compare Handler
$004    rjmp    TIM2_OVF    ; Timer2 Overflow Handler
$005    rjmp    TIM1_CAPT   ; Timer1 Capture Handler
$006    rjmp    TIM1_COMPA  ; Timer1 CompareA Handler
$007    rjmp    TIM1_COMPB  ; Timer1 CompareB Handler
$008    rjmp    TIM1_OVF    ; Timer1 Overflow Handler
$009    rjmp    TIM0_OVF    ; Timer0 Overflow Handler
$00a    rjmp    SPI_STC     ; SPI Transfer Complete Handler
$00b    rjmp    USART_RXC   ; USART RX Complete Handler
$00c    rjmp    USART_UDRE  ; UDR Empty Handler
$00d    rjmp    USART_TXC   ; USART TX Complete Handler
$00e    rjmp    ADC         ; ADC Conversion Complete Handler
$00f    rjmp    EE_RDY      ; EEPROM Ready Handler
$010    rjmp    ANA_COMP    ; Analog Comparator Handler
$011    rjmp    TWSI        ; Two-wire Serial Interface
Handler
$012    rjmp    SPM_RDY     ; Store Program Memory Ready
Handler
```

Para que servem?

- Preempção
  - Alternativa ao polling
  - Tratamento de falhas (Faults, Traps, etc)
  - Chamadas de sistema operacional (syscalls)
  - Necessárias para implementação de threads

Utilidade

- Economia de processamento
  - Callback quando ocorre um evento
  - Preempção sem thread
- Resposta rápida a um evento
  - Parada imediata do programa principal para tratamento do evento
- Permite implementar threads
- Podem ser ligadas e desligadas

Tipos

- Transmissão / Recebimento de dados
  - Término do envio / pronto para enviar novamente
  - Erro no envio
  - Dados recebidos
  - Erro no recebimento
  - Timeout
  - Buffer overflow
- Erros
  - Seg fault / Bus error - Acesso de memória inválido
  - Código de programa inválido
- Timer
- Software
  - Syscall

Cuidados

- Bloqueiam toda a execução do programa até retornar
  - Devem ser rápidas
- Preempção !
  - Mas sem mutex, semáforos, etc
    - Tratamento de concorrência diferente de threads
- Região crítica
  - Desligar as interrupções antes de partes do código que não podem ser interrompidas

Usos típicos

- Troca de flag
  - Apenas sinaliza que ocorreu o evento, em vez de trata-lo
- Preenchimento de buffer
  - Lê um dado recebido e o coloca em um buffer
- Consumo de buffer
  - Envia os dados de um buffer para um periférico
- Contagem de tempo
  - Um periférico timer interrompe o programa após um determinado intervalo de tempo
- Tratamento de falha
  - Bloqueia o programa para inspeção com debugger

Exemplo

```
int main(void) {
    // Configura timer para interromper a cada 500ms
    Timer.Config(500);

    // Processamento lento de um vetor de dados
    for (i=0; i<Quantidade; i++)
        processa(dados[i]);
}

void timer_interrupt(void) {
    // Função executada a cada 500ms
    LED.toggle();
}
```

Exemplo no Arduino

- Objetivo
  - Trocar a forma que um LED pisca utilizando comandos via porta serial
  - Mostrar que a resposta aos comandos não é imediata se o programa estiver processando outra tarefa antes de ler a mensagem recebida
  - Mostrar que apenas não utilizar funções como delay não é solução
- Exemplo
  - Adaptação do exemplo do XTEA
  - Utilizar a interrupção de USART (porta serial) para trocar o comportamento do LED
    - Comandos:
      - 1 - Apaga o LED
      - 2 - Acende o LED
      - Enter - Troca entre LED piscando e estático
- Código
  - github.com/naves-thiago/reativos/blob/master/code/arduino/xtea_irq.ino

Exemplo no Arduino

Main loop

```
uint32_t key[] = { 1, 2, 3, 4 };
uint32_t v[]   = { 10, 20 };
int flag = 1; // Inicialmente vamos piscar o LED
int led = 0;  // Acende na primeira iteração

void loop() {
  led = led ^ flag; // Pisca o LED se flag = 1
  digitalWrite(LED, led);

  unsigned long t1 = millis();

  // Ocupa a CPU com umas contas
  serial_printf("Antes:   %ul %ul\n\r", v[0], v[1]);
  encipher(ROUNDS, v, key);
  serial_printf("Durante: %ul %ul\n\r", v[0], v[1]);
  decipher(ROUNDS, v, key);
  serial_printf("Depois:  %ul %ul\n\r", v[0], v[1]);

  unsigned long t2 = millis();

  // Imprime quanto tempo ficamos fazendo conta (e escrevendo na serial)
  serial_printf("\n\rTempo %ul\n\r\n\r" , t2-t1);

  // Espera mais um pouco...
  delay(500);
}
```

Função de interrupção da serial

```
// Função mágica de interrupção da porta serial
ISR(USART_RX_vect) {
 char a = UDR0; // Lê o caracter recebido

 switch (a) {
   case '\r': flag = flag ^ 1; break;                    // Muda o flag
   case '1':  led = 0; digitalWrite(LED, LOW);  break; // Desliga o LED
   case '2':  led = 1; digitalWrite(LED, HIGH); break; // Liga o LED
   default: break;
 }
}
```

Nota: ISR = Interrupt Service Routine

Exemplo no Arduino

- Inicialização manual da porta serial
- Lib de serial do Arduino usa a interrupção internamente (e nos impede de usa-la)
- Fabricantes de processador gostam de nomes crípticos...

```
#define USART_BAUDRATE 9600
#define BAUD_PRESCALE (((F_CPU / (USART_BAUDRATE * 16UL))) - 1)
void serial_init(void) {
  // Desliga as interrupções
  cli();

  /* 9600 baud */
  UBRR0L = (uint8_t)(BAUD_PRESCALE & 0xff);
  UBRR0H = (uint8_t)(BAUD_PRESCALE >> 8);
  UCSR0B =
        /* interrupt on receive */
        (1 << RXCIE0) |
        (1 << RXEN0)  |
        (1 << TXEN0);
  UCSR0C =
        /* no parity bit */
        (0 << UPM01) |
        (0 << UPM00) |
        /* asyncrounous USART */
        (0 << UMSEL01) |
        (0 << UMSEL00) |
        /* one stop bit */
        (0 << USBS0) |
        /* 8-bits of data */
        (1 << UCSZ01) |
        (1 << UCSZ00);

  // Liga as interrupções
  sei();
}
```