

Laboratório de Sistemas Embarcados - Relatório 5

Grupo:

Leonardo Borges Mafra Machado - 9345213

Marcos Paulo Pereira Moretti - 9345363

Paula Yumi Pasqualini - 9345280

1. Introdução

O objetivo deste experimento é utilizar a tarefa do laboratório da semana passada para orquestrar vários processos, cada um responsável por controlar uma porção da tela do QEMU.

Em seguida, o grupo respondeu às perguntas propostas no site do professor, sobre o funcionamento do UART, e debugamos um programa para entender a relação entre o UART e o VIC.

2. Experimente fazer com que cada processo pisque um quadradinho no LCD

Para que cada processo pisque um quadradinho no LCD, alguns códigos tiveram de ser alterados:

Em vid.c, foi atualizada a função show_bmp:

```
void show_bmp(u32 start_row, u32 start_col, int x)
{
    u32 h, w, pixel, rsize, i, j;
    u8 r, g, b;
    for(i = start_row; i < start_row + 100; i++)
    {
        for(j = start_col; j < start_col + 100; j++)
        {
            if(x == 100) {
                b = 0; g = 0; r = 0;
            }
            else {
                b = (x % 3) * 20;
                g = x * 20;
                r = 255 - (x % 3) * 10;
            }
            pixel = (b << 16) | (g << 8) | r; // 24-bit color BMP
            fb[i * gDisplayContext.screen_width + j] = pixel; // in the frame buffer, a pixel occupies
32-bit.
        }
    }
}
```

```
}  
}
```

Em t.c foi atualizada a função process (chamada por cada um dos 9 processos):

```
void process(int x)  
{  
    while(1)  
    {  
        show_bmp((x / 3)*100, (x % 3)*100, x);  
        show_bmp((x / 3)*100, (x % 3)*100, 100);  
    }  
}
```

Em vid.h, foi atualizada a assinatura de show_bmp():

```
void show_bmp(u32 start_row, u32 start_col, int x);
```

Em ts.s, foi atualizada a quantidade de processos:

```
reset_handler:  
    LDR    sp, =stack_top  
  
    MOV    r5, #9                ;@ qtd de processos
```

Observa-se na imagem abaixo o resultado:



3. Perguntas

3.1. Interrupção na transmissão de caracteres

Por que a interrupção é usada na transmissão de caracteres?

A interrupção UART é usada para indicar que um byte foi completamente transferido do registrador TXREG para o buffer TSR, de modo que o registrador está pronto para receber o próximo caractere.

Dessa forma, a interrupção é feita quando transmitimos uma string de caracteres, de modo a evitar que precisemos checar a finalização de cada caractere transmitido, o que poderia degradar a performance.

3.2 Vetor de interrupção

Em que arquivo está declarado o vetor de interrupção?

O vetor de interrupções está declarado no arquivo **ts.s**, nas linhas indicadas abaixo:

```
vectors_start:
    ldr pc, reset_handler_addr
    ldr pc, undef_handler_addr
    ldr pc, swi_handler_addr
    ldr pc, prefetch_abort_handler_addr
    ldr pc, data_abort_handler_addr
    b .
    ldr pc, irq_handler_addr
    ldr pc, fiq_handler_addr

reset_handler_addr:      .word reset_handler
undef_handler_addr:     .word Undefined_Handler
swi_handler_addr:       .word swi_handler
prefetch_abort_handler_addr: .word prefetch_abort_handler
data_abort_handler_addr: .word data_abort_handler
irq_handler_addr:       .word IRQ_Handler
fiq_handler_addr:       .word fiq_handler
vectors_end:
```

3.3 Interrupção de hardware

Qual a função chamada quando ocorre uma interrupção de hardware?

A função chamada é a `irq_handler`.

Coloque um breakpoint nessa interrupção

Para rodar o código em modo debug, em um terminal rodamos:

```
qemu-system-arm -s -M versatilepb -cpu arm926 -kernel build/t.bin -serial  
telnet:localhost:1122,server -S
```

E em outro terminal:

```
telnet localhost 1122
```

E em um terceiro terminal:

```
gdb-multiarch
```

Em seguida:

```
(gdb) target remote :1234  
(gdb) file build/t.elf  
(gdb) load
```

Colocamos os seguintes breakpoints:

```
(gdb) b main  
(gdb) b irq_handler
```

E damos 'continue' até que o programa tenha printado "[Input something through UART and then press Enter to show some chars:]" e esteja esperando a entrada do usuário.

Quando apertamos uma tecla, o caractere correspondente não é automaticamente exibido na tela. Inicialmente, ocorre uma primeira interrupção:

```

File Edit View Search Terminal Tabs Help
pypas@pypas-HP-Noteb... x pypas@pypas-HP-Noteb... x pypas@pypas-HP-Notebo... x
B+> 0x1001c <irq_handler>      sub    lr, lr, #4
0x10020 <irq_handler+4>      push   {r0, r1, r2, r3, r4, r5, r6, r7, r8,
0x10024 <irq_handler+8>      bl     0x101a0 <IRQ_handler>
0x10028 <irq_handler+12>     ldm     sp!, {r0, r1, r2, r3, r4, r5, r6, r7
0x1002c <undef_handler>      b       0x1002c <undef_handler>
0x10030 <lock>               mrs     r0, CPSR
0x10034 <lock+4>             orr     r0, r0, #128      ; 0x80
0x10038 <lock+8>             msr     CPSR_fc, r0
0x1003c <lock+12>            mov     pc, lr
0x10040 <unlock>             mrs     r0, CPSR
0x10044 <unlock+4>           bic     r0, r0, #128      ; 0x80
0x10048 <unlock+8>           msr     CPSR_fc, r0
0x1004c <unlock+12>          mov     pc, lr
0x10050 <vectors_start>      ldr     pc, [pc, #24]    ; 0x10070 <reset_han
0x10054 <vectors_start+4>    ldr     pc, [pc, #24]    ; 0x10074 <undef_han

remote Thread 1 In: irq_handler      L??  PC: 0x1001c
(gdb)

```

A primeira interrupção chama a função `do_rx` (referente à recepção de caracteres), dentro de `uart_handler()`.

```

File Edit View Search Terminal Tabs Help
pypas@pypas-HP-Notebook: ~/EmbeddedAndRTOS... x pypas@pypas-HP-Notebook: ~/EmbeddedAndRTOS... x pypas@pypas-HP-Notebook: ~/EmbeddedAndRTOS... x
Register group: general
r0      0xf2de0  994784      r1      0xf420c  999948
r2      0xd      13          r3      0x10     16
r4      0x0      0           r5      0x0     0
r6      0x0      0           r7      0x0     0
r8      0x0      0           r9      0x0     0
r10     0x0      0           r11     0xf5254  1004116
r12     0x0      0           sp      0xf5240  0xf5240
lr      0x101d0  66000      pc      0x104c4  0x104c4 <uart_handler+48>
cpsr    0x20000192  536871314  fpscr   0x0     0
fpsidr  0x4101090  1090588816  fpexc   0x0     0

uart.c
99
{
100     u8 mis = *(up->base + MIS); //read MIS register
101     if(mis & RX_BIT)
102     {
> 103         do_rx(up);
104     }
105     else if(mis & TX_BIT)
106     {
107         do_tx(up);
108     }
}

remote Thread 1 In: uart_handler      L103  PC: 0x104c4
Continuing.

Breakpoint 1, IRQ_handler () at t.c:66
Continuing.

Breakpoint 1, IRQ_handler () at t.c:66
(gdb) n
(gdb) si
(gdb) si
uart_handler (up=0xf2de0 <uart>) at uart.c:99
(gdb) n
(gdb)

```

Uma segunda interrupção chama a função `do_tx` (referente à transmissão de caracteres), dentro de `uart_handler()`.

```
File Edit View Search Terminal Tabs Help
pypas@pypas-HP-Notebook: ~/EmbeddedAndRTOS... x pypas@pypas-HP-Notebook: ~/EmbeddedAndRTOS... x pypas@pypas-HP-Notebook: ~/EmbeddedAndRTOS... x
Register group: general
r0 0xf2de0 994784 r1 0x61 97
r2 0x61 97 r3 0x20 32
r4 0x0 0 r5 0x0 0
r6 0x0 0 r7 0x0 0
r8 0x0 0 r9 0x0 0
r10 0x0 0 r11 0xf5254 1004116
r12 0x0 0 sp 0xf5240 0xf5240
lr 0x101d0 66000 pc 0x104e0 0x104e0 <uart_handler+76>
cpsr 0x20000192 536871314 fpscr 0x0 0
fpsid 0x41011090 1090588816 fpexc 0x0 0

uart.c
101 if(mis & RX_BIT)
102 {
103     do_rx(up);
104 }
105 else if(mis & TX_BIT)
106 {
> 107     do_tx(up);
108 }
109 else
110 {

remote Thread 1 In: uart_handler L107 PC: 0x104e0
(gdb) si
uart_handler (up=0xf2de0 <uart>) at uart.c:99
(gdb) n
(gdb) c
continuing.

Breakpoint 1, IRQ_handler () at t.c:66
(gdb) n
(gdb) si
uart_handler (up=0xf2de0 <uart>) at uart.c:99
(gdb) n
(gdb) 
```

Abaixo, podemos verificar o caractere printado na tela:

```
File Edit View Search Terminal Tabs Help
pypas@pypas-HP-Noteb... x pypas@pypas-HP-Noteb... x pypas@pypas-HP-Notebo... x
[Input something through UART and then press Enter to show some chars:]
asd
[And you have entered below line from UART:]
asd
[Input something through UART and then press Enter to show some chars:]
Connection closed by foreign host.
pypas@pypas-HP-Notebook:~/EmbeddedAndRTOSSamples/C2.6$ telnet localhost 1111
Trying 127.0.0.1...
Connected to localhost.
Escape character is '^'.
[Input something through UART and then press Enter to show some chars:]
Connection closed by foreign host.
pypas@pypas-HP-Notebook:~/EmbeddedAndRTOSSamples/C2.6$ telnet localhost 1111
Trying 127.0.0.1...
Connected to localhost.
Escape character is '^'.
[Input something through UART and then press Enter to show some chars:]
Connection closed by foreign host.
pypas@pypas-HP-Notebook:~/EmbeddedAndRTOSSamples/C2.6$ telnet localhost 1111
Trying 127.0.0.1...
Connected to localhost.
Escape character is '^'.
[Input something through UART and then press Enter to show some chars:]
a
```

3.4 VIC

O que é o VIC (controlador da interrupção) e para que serve?

O VIC é um hardware que funciona como um gerenciador de interrupções. Ele contém um vetor de endereços das rotinas de interrupção associadas aos pedidos de interrupções feitos via software.

Esse hardware permite determinar a localização da rotina de interrupção que trata um pedido de interrupção feito por um processo, assim como identificar o hardware responsável por atender a interrupção.

O VIC provê o endereço inicial (armazenado no registrador VICVECTADDR) ou o vetor de endereço da rotina associada à interrupção.

Além disso, o VIC organiza os pedidos de interrupção segundo uma ordem de prioridade, de modo que no registrador VICVECTADDR sempre consta o endereço da rotina associado à interrupção que foi escolhida para ser atendida imediatamente.

O que faz o código abaixo em t.c:

```
VIC_INTENABLE |= UART0_IRQ_VIC_BIT;  
VIC_INTENABLE |= UART1_IRQ_VIC_BIT;  
u32 vicstatus = VIC_STATUS;  
  
//UART 0  
if(vicstatus & UART0_IRQ_VIC_BIT)
```

O registrador IRQ Status Register (vicstatus no código) contém o status das interrupções [31:0] após após IRQ masking. Um bit setado indica que a interrupção está ativa, e gera uma interrupção para o processador.

O registrador Interrupt Enable Register (vicintenable no código) habilita as linhas de requisição de interrupção, mascarando as fontes de interrupção para a interrupção IRQ. 1 = Interrupções habilitadas (permite pedidos de interrupção ao processador). 0 = interrupções desabilitadas.

O código abaixo faz o OU lógico entre VIC_INTENABLE e UART0_IRQ_VIC_BIT, habilitando as interrupções provenientes da UART0. Em seguida, o mesmo é feito para a UART1

```
VIC_INTENABLE |= UART0_IRQ_VIC_BIT;  
VIC_INTENABLE |= UART1_IRQ_VIC_BIT;
```

Já no código abaixo, chamado quando ocorre uma interrupção IRQ, o vicstatus é comparado ao UART0_IRQ_VIC_BIT, e se o bit correspondente à UART0 for 1, é chamada a função `uart_handler`, passando como parâmetro o registrador MIS (Masked Interrupt Status) da UART0. O mesmo é feito em relação à UART1:

```
void IRQ_handler()
{
    u32 vicstatus = VIC_STATUS;

    //UART 0
    if(vicstatus & UART0_IRQ_VIC_BIT)
    {
        uart_handler(&uart[0]);
    }

    //UART 1
    if(vicstatus & UART1_IRQ_VIC_BIT)
    {
        uart_handler(&uart[1]);
    }
}
```

A função `uart_handler` lê o registrador MIS para determinar o tipo de interrupção. Se o bit 4 do registrador MIS for 1, a interrupção é do tipo RX (recepção). Se o bit 5 do registrador MIS for 1, a interrupção é do tipo TX (transmissão).