Laboratório de Sistemas Embarcados - Relatório 10

Grupo:
Leonardo Borges Mafra Machado - 9345213
Marcos Paulo Pereira Moretti - 9345363
Paula Yumi Pasqualini - 9345280

1. Introdução

O laboratório da aula 10 consistiu em continuar o trabalho das últimas aulas, abaixo segue a lista do que estava faltando ser feito:

- Acrescentar uma pilha para cada processo no modo kernel
- Acrescentar LR e o SP do modo kernel na linha dos processos
- Acrescentar o estado na linha do processo: “bloqueado”, “rodando” ou “pronto”
- Sleep():
    - Guarda na pilha do processo (no modo kernel) os registradores do modo usuário
    - Guarda na linha do processo o SP e LR do modo usuário
    - Guarda na linha do processo o SP e LR do modo kernel
    - Coloca o processo no estado de “bloqueado”
- Timer_Handler():
    - Guarda na linha do processo o SP e LR do modo usuário
    - Guarda na linha do processo o SP e LR do modo kernel
- Wakeup(pid):
    - Coloca o processo no estado de “pronto”.

No laboratório de hoje foi separado as pilhas no modo usuário e no modo kernel, foi acrescentado o LR, SP e SPSR do modo kernel na linha dos processos e foi acrescentado o estado na linha também. Finalmente foi possível fazer a System Call lê linha funcionar como esperado e foi feito as System Calls de sleep e wakeup.

2. Códigos modificados

2.1 t.ld

No arquivo .ld foi acrescentado um espaço para as pilhas dos processos em modo kernel:

```
ENTRY(reset_handler)
SECTIONS
{
   . = 0x10000; /* loading address */
```

```
    .text : {*(.text)}
    .data : {*(.data)}
    .bss : {*(.bss)}
    .data : {*(image.o)}

    . = ALIGN(8);
    . = . + 0x10000; /* 64KB of SVC stack */
    svc_stack_top = .;
    . = . + 0x1000; /* 4kB of stack memory */
    undefined_stack_top = .;
    . = ALIGN(8);
    . = . + 0x1000; /* 4kB of stack memory */
    IRQ_stack_top = .;
    . = ALIGN(8);
    . = . + 0x1000; /* 4kB of stack memory */
    stack_top = .;
    . = ALIGN(8);
    . = . + 0x10000; /* 64kB of stack memory */
    linha = .;
    . = ALIGN(8);
    . = . + 0x10000; /* 64kB of stack memory */
    nproc = .;
}
```

2.2 t.c

Em t.c, foi atualizada a função *process()*, de modo a chamar as system calls GETPID e GETLINHA.

```
void process()
{
    int pid = system_call(GETPID_SYSCALL_NB, 0);

    while(1) {
            char *s, *t;
            int a = system_call(GETLINHA_SYSCALL_NB,0);
            s = (char *) a;
            t = trata_linha(pid,s);
            uprints(&uart[0],t);
    }
}
```

O handler das interrupções também foi atualizado, já incluindo as *syscalls* GETPID, GETLINHA, SLEEP e WAKEUP:

```
int SWI_Handler_C(int nb_syscall, int pid)
{
        if (nb_syscall == GETPID_SYSCALL_NB){
```

```
                return get_pid();
        }
        else if (nb_syscall == GETLINHA_SYSCALL_NB){
                char* s;
                syscall_le_linha(&uart[0], s);
                return (int) s;
        }
        else if (nb_syscall == SLEEP_SYSCALL_NB){
                sleep();
                return 1;
        }
        else if (nb_syscall == WAKEUP_SYSCALL_NB){
                wakeup(pid);
                return 1;
        }
}
```

2.3 ts.S

No *loop_init_process* foi acrescentado a inicialização das pilhas dos processos no modo kernel.

```
loop_init_process:
        CMP   r7, r5                    ;@ count < qtd processos?
        BEQ   finish_init_process

        LDR   r12, =linha               ;@ encontra linhaX
        LDR   r10, =0x1000
        MUL   r1, r7, r10
        ADD   r0, r1, r12

        LDR   r11, =stack_top           ;@ inicializa stack pointer
        MUL   r1, r7, r10
        ADD   r1, r1, r11
        STR   r1, [r0, #52]             ;@ sp = r13 = 13*4 = 52

        LDR   r11, =svc_stack_top               ;@ inicializa stack pointer do modo
kernel
        MUL   r1, r7, r10
        ADD   r1, r1, r11
        STR   r1, [r0, #68]             ;@ sp_kernel = 17*4 = 68

        LDR   r1, =task                 ;@ inicializa pc
        STR   r1, [r0, #60]             ;@ pc = r15 = 15*4 = 60

        MRS   r1, cpsr                  ;@ inicializa cpsr
```

```
        BIC     r1, r1, #0x8F                   ;@ enable interrupts in the cpsr + coloca
processo em modo usuario
        STR     r1, [r0, #64]                   ;@ cpsr = 16*4 = 64

        STR     r7, [r0]                ;@ inicializa r0 com numero de seu proprio processo

        ADD     r7, r7, #1
        B       loop_init_process
```

Um ponto importante do laboratório foi a ativação de interruções em SWI_Handler, possibilitando que a função *uart_handler* fosse chamada:

```
SWI_Handler:
        STMFD   sp!, {lr}               ;@Empilha os registradores
        MRS     r0, cpsr
        BIC     r0, r0, #0x80
        MSR     cpsr_c, r0                      ;@enabling interrupts in the cpsr
        MOV     r0, r7
        MOV     r1, r8
        BL      SWI_Handler_C
        LDMFD   sp!, {pc}^
```

Abaixo é possível observar as system calls de sleep e wakeup.

```
sleep:
        STMFD   sp!, {r0-r12,lr}
        LDR     r1, =nproc                      ;@ load addr nproc
        LDR     r0, [r1, #4]                    ;@ r0 = processo corrente
        LDR     r12, =linha                     ;@ encontra linhaX
        LDR     r10, =0x1000
        MUL     r3, r0, r10
        ADD     r12, r3, r12                    ;@ r12 = linhaX
        MOV     r5, #1
        STR     r5, [r12, #80]          ;@ coloca estado = bloqueado
        B       Timer_Handler

wakeup:
        LDR     r12, =linha                     ;@ encontra linhaX
        LDR     r10, =0x1000
        MUL     r3, r0, r10
        ADD     r12, r3, r12                    ;@ r12 = linhaX
        MOV     r5, #0
        STR     r5, [r12, #80]          ;@ coloca estado = pronto
        MOV     pc, lr
```

No Timer_Handler, pulamos os processos bloqueados e carregamos os valores do lr e do sp dos modos supervisor e usuário do próximo processo pronto:

```
Timer_Handler:
        LDR    r0, TIMER0X
        MOV    r1, #0x0
        STR    r1, [r0]                ;@Escreve no registrador TIMER0X para limpar o
pedido de interrupção

        LDR    r0, =nproc                      ;@ load addr nproc
        LDR    r2, [r0]                ;@ qtd de processos total
        LDR    r1, [r0, #4]                    ;@ r1 = processo corrente

        LDR    r12, =linha                     ;@ encontra linhaX
        LDR    r10, =0x1000
        MUL    r3, r1, r10
        ADD    lr, r3, r12                     ;@ lr = linhaX

Loop_Estado:
        ADD    r1, r1, #1                      ;@ r1++
        SUB    r2, r1, r2                      ;@ compara r1 com qtd de processos
        CMP    r2, #0
        MOVEQ   r1, #0                         ;@ caso r1 > processos, r1 = 0
        LDR    r12, =linha                     ;@ encontra linhaX
        LDR    r10, =0x1000
        MUL    r3, r1, r10
        ADD    r4, r3, r12                     ;@ r4 = linhaX
        LDR    r5, [r4, #80]            ;@ estado = 20*4 = 80
        CMP             r5, #1                          ;@ verifica se estado == bloqueado
        BEQ             Loop_Estado                             ;@ se estiver bloqueado, vai
para o proximo processo

        STR    r1, [r0, #4]                    ;@ atualiza nproc = r1

        LDMFD   sp!, {r0-r12}
        STMIA   lr!, {r0-r12}

        LDMFD   sp!, {r3}                      ;@ pega o lr da pilha
        MRS    r4, spsr                        ;@ cpsr anterior

        MRS    r0, cpsr                        ;@ salvando o modo corrente em R0

        MSR    cpsr_ctl, #0b11011111                   ;@ alterando o modo para system
        MOV    r1, sp                          ;@ sp do modo usuário
        MOV    r2, lr                  ;@ lr do modo usuário
        MSR    cpsr_ctl, #0b11010011                   ;@ alterando o modo para o
supervisor
        MOV    r5, sp                          ;@ sp do modo supervisor
        MOV    r6, lr                  ;@ lr do modo supervisor
        MRS             r7, spsr                        ;@ spsr do modo supervisor
```

```
        MSR   cpsr, r0                  ;@ volta para o modo anterior

        STMIA   lr!, {r1,r2,r3,r4,r5,r6,r7}     ;@ salva sp, lr, pc, cpsr, sp_supervisor,
lr_supervisor, spsr_supervisor

        LDR   r0, =nproc                ;@ load addr nproc
        LDR   r1, [r0, #4]              ;@ r1 = processo corrente

        LDR   r12, =linha               ;@ encontra linhaX
        LDR   r10, =0x1000
        MUL   r3, r1, r10
        ADD   r12, r3, r12              ;@ r12 = linhaX

        MRS   r0, cpsr                  ;@ salvando o modo corrente em R0

        MSR   cpsr_ctl, #0b11011111             ;@ alterando o modo para system
        LDR   sp, [r12, #52]            ;@ sp = r13 = 13*4 = 52
        LDR   lr, [r12, #56]            ;@ lr = r14 = 14*4 = 56
        MSR   cpsr_ctl, #0b11010011             ;@ alterando o modo para o
supervisor
        LDR   sp, [r12, #68]            ;@ sp = r17 = 17*4 = 68
        LDR   lr, [r12, #72]            ;@ lr = r18 = 18*4 = 72
        LDR   r1, [r12, #76]            ;@ cpsr = 19*4 = 76
        MSR   spsr, r1                   ;@ atualiza spsr do modo supervisor

        MSR   cpsr, r0                  ;@ volta para o modo anterior

        MOV   lr, r12
        LDR   r0, [lr, #60]             ;@ pc = r15 = 15*4 = 60
        STMFD   sp!, {r0}
        LDMIA   lr, {r0-r12}
        STMFD   sp!, {r0-r12}           ;@ Empilha os registradores

        LDR   r0, [lr, #64]             ;@ cpsr = 16*4 = 64
        MSR   spsr, r0
        LDMFD   sp!, {r0-r12,pc}^
```

Salvamos para cada processo os registradores sp, lr do modo supervisor, e também o spsr nas posições a seguir, após as posições da linha originais:

- 17: svc_sp
- 18: svc_lr
- 19: svc_spsr
- 20: state (0, se o processo está pronto, ou 1, se o processo está bloqueado)

Abaixo está uma demonstração do resultado final obtido com somente um processo rodando:

```
pypas@pypas-HP-Notebook:~/EmbeddedAndRTOSSamples/C2.6$ telnet localhost 1111
Trying 127.0.0.1...
Connected to localhost.
Escape character is '^]'.
teste
TESTE
abcd
ABCD
oqwertyuiop
OQWERTYUIOP
```

3. Conclusão e próximos passos

Alguns pontos interessantes de mencionar foi a necessidade de ativar as interrupções no modo supervisor toda vez que é feita uma SWI e que o empilhamento dos registradores no SWI_Handler estava causando um comportamento estranho no retorno (será investigado nos próximos laboratórios o motivo).

Para a próxima aula será necessário as seguintes alterações:

- Fazer a system call de escrita na tela.
- Fazer a fila de processos que estão aguardando para fazer as system calls de leitura e escrita.
- Fazer a ligação entre as system calls de leitura a escrita e as de sleep e wakeup.