Laboratório de Sistemas Embarcados - Relatório 7

Grupo:

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

1. Introdução

A proposta desse laboratório é colocar os processos dos códigos dos laboratórios anteriores para rodarem em modo usuário.

Em seguida, será criada a system call getpid(), que retorna o pid do processo que a chamou.

2. Processos em modo usuário

2.1 Mudanças no código

A primeira parte do laboratório consistiu em passar os processos do modo supervisor para o modo usuário. Para isso, tiveram de ser alteradas algumas partes do código (marcadas em negrito):

1. Foi alterada a inicialização dos processos, de forma que o CPSR deles seja iniciado em modo usuário

```
loop_init_process:
      MRS r1, cpsr
                                       :@ inicializa cpsr
      BIC
             r1, r1, #0x8F
                                       ;@ enable interrupts in the cpsr + coloca
processo em modo usuario
finish init process:
      MRS r0, cpsr
                                       ;@ salvando o modo corrente em R0
      MSR cpsr_ctl, #0b11011111
                                              ;@ alterando o modo para system
processo inicial
                                       ;@ a pilha de system eh setada
      LDR
             sp, =stack_top
      MSR cpsr, r0
      BL
             copy_vectors
      BL
             init display
      BL
             Timer Init
      MSR cpsr_ctl, #0b01010000
                                              ;@ alterando o modo do primeiro
```

processo para usuario

2. Em Timer_Handler, inicialmente verificamos se SPSR (CPSR anterior) corresponde ao modo usuário. Se esse for o caso, mudamos para o modo system para resgatar os valores dos registradores LR e SP do modo usuário. Caso o SPSR não corresponda ao modo usuário, mudamos para esse modo (aquele armazenado em SPSR) para resgatar LR e SP. Da mesma forma, a mesma lógica é usada para recuperar os registradores do próximo processo.

```
Timer_Handler:
      MRS r4, spsr
                                       ;@ cpsr anterior
                                       ;@ salvando o modo corrente em R0
      MRS r0, cpsr
            r5, r4, #0b11111
      AND
      CMP r5, #0b10000
      MSREQ cpsr_ctl, #0b11011111
                                              ;@ alterando o modo para system
                                ;@ alterando o modo para o anterior
      MSRNE cpsr_ctl, r4
                                ;@ sp do processo corrente
      MOV r1, sp
      MOV r2, lr
                                ;@ Ir do processo corrente
      MSR cpsr, r0
                                       ;@ volta para o modo anterior
      MRS r0, cpsr
                                       ;@ salvando o modo corrente em R0
            r5, r4, #0b11111
      AND
      CMP r5, #0b10000
                                       ;@ alterando o modo para system
      MSREQ cpsr_ctl, #0b11011111
      MSRNE cpsr_ctl, r4
                                       ;@ alterando o modo para o anterior
                                       ;@ sp = r13 = 13*4 = 52
             sp, [r12, #52]
      LDR
                                       @ Ir = r14 = 14*4 = 56
      LDR
             Ir, [r12, #56]
      MSR cpsr, r0
                                       ;@ volta para o modo anterior
```

2.2 Demonstração

Na figura abaixo, é possível verificar o momento em que os processos são colocados em modo usuário:

```
pypas@pypas-HP-Notebook: ~/EmbeddedAndRTOSSamples/C2.6
 pypas@pypas-HP-Notebook: ~/Embedde
                                                              pypas@pypas-HP-Notebook: ~/Embedde.
                                                                                                                       × pypas@pypas-HP-Notebook: ~/Embedde... ×
    -2147483312
                                                                                                                         0x80000150
                                                                                               Γ3
Γ5
Γ7
                          0x0
                          0x0
                                                                                                                         0x9
                          0x0
                                                                                                                        0 \times 0
                          0x0
                                                                                                                        0x0
                           0x1000
                                                                                                                         0xf8430
                                                                                                                                       1016880
                           0x108430 1082416
                                                                                                                         0xf8430
      0x10040 <loop_init_process+36>
0x10044 <loop_init_process+40>
0x10048 <loop_init_process+44>
0x1004c <loop_init_process+48>
0x10050 <loop_init_process+52>
0x10054 <loop_init_process+56>
                                                                       r1, [r0, #52]
r1, [pc, #480]
r1, [r0, #60]
r1, CPSR
                                                                                                      0x1022c <vectors_end+12>
                                                                        г1, г1, #143
г1, [г0, #64
                                                                                                     0x8f
                                                                                      #64]
      0x10058 <loop_init_process+60>
0x1005c <loop_init_process+64>
remote Thread 1 In: loop_init_process
(gdb) p/t $r1
$6 = 1000000000000000000000000101010000
(gdb)
```

Na próxima figura, verifica-se que os processos estão rodando em modo usuário:

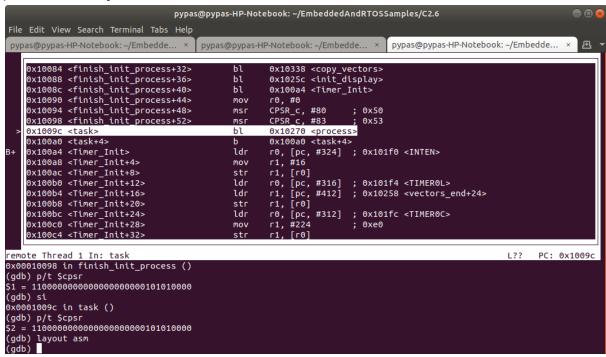
```
pypas@pypas-HP-Notebook: ~/EmbeddedAndRTOSSamples/C2.6
pypas@pypas-HP-Notebook: ~/Embedde...
                                              pypas@pypas-HP-Notebook: ~/Embedde.
                                                                                         × pypas@pypas-HP-Notebook: ~/Embedde... ×
  -Register group: general
                                                                       Γ1
Γ3
Γ5
Γ7
                   0x1
                                                                                          0x0
                                                                                          0x0
г2
                   0x0
                                                                                          0x0
                   0x0
                                                                                          0x0
                                                                       г9
г8
                   0x0
                               0
                                                                                          0x0
                                                                                          0xfffffffc
                   0x0
                                                                                                               -4
г10
                   while(1)
                        show_bmp((x / 3)*100, (x % 3)*100, x);
show_bmp((x / 3)*100, (x % 3)*100, 100);
remote Thread 1 In: process
(gdb) p/t $cpsr
$9 = 10000000000000000000000101010000
(gdb)
```

2.3 Chaveamento modo usuário para modo supervisor

No código abaixo, mudamos o processo para o modo usuário:

```
pypas@pypas-HP-Notebook: ~/EmbeddedAndRTOSSamples/C2.6
                                                              pypas@pypas-HP-Notebook: ~/Embedde... × pypas@pypas-HP-Notebook: ~/Embedde...
                                                                                                                                                                                        × Æ
                                                            [ No Source Available ]
      0x10084 <finish_init_process+32>
0x10088 <finish_init_process+36>
0x1008c <finish_init_process+40>
0x10090 <finish_init_process+44>
0x10094 <finish_init_process+48>
                                                                                       0x10338 <copy_vectors>
0x1025c <init_display>
                                                                           ы
                                                                                       0x100a4 <Timer_Init>
                                                                                       г0, #0
CPSR_c, #80
                                                                           mov
                                                                           msr
                                                                                       CPSR_c, #83 ;
0x10270 cprocess>
0x100a0 <task+4>
       0x10098 <finish_init_process+52>
      0x1009c <task>
0x100a0 <task+4>
                                                                           ы
                                                                           Ь
 remote Thread 1 In: finish_init_process
                                                                                                                                                                               PC: 0x10098
0x000100d0 in Timer_Init ()
0x000100d4 in Timer_Init ()
0x00010090 in finish_init_process ()
0x00010094 in finish_init_process ()
0x00010098 in finish_init_process ()
(gdb) p/t $cpsr
$1 = 1<u>1</u>000000000000000000000101010000
```

No código abaixo, tentamos mudar para o modo supervisor, mas o comando é ignorado, pois essa transição:



3. Syscall getpid

3.1 Arquivos alterados

Em t.s, foram adicionadas as seguintes funções:

- system_call: função que aciona a interrupção de software
- SWI_Handler: função (chamada no vetor de interrupções) de tratamento de interrupções de software
- get_pid: função que recupera o pid do processo em questão (guardada em nproc)

```
system call:
      SWI
            0x0
      MOV pc, Ir
SWI Handler:
                                ;@Empilha os registradores
      STMFD sp!, {r8-r12,lr}
      MOV r0, r7
            SWI_Handler_C
      LDMFD sp!, {r8-r12,pc}^
get pid:
      LDR r1, =nproc
                                      ;@ load addr nproc
      LDR r0, [r1, #4]
                                      ;@ r1 = processo corrente
      MOV pc, Ir
```

Em t.c, foram adicionadas:

- As assinaturas das funções system_call e get_pid (definidas em assembly)
- A função process(), chamada pelos processos em modo usuário. Essa função chama system_call(), passando como parâmetro o código correspondente à system call get_pid
- A função SWI_Handler_C(), que verifica qual a system call a ser chamada, de acordo com o código passado

```
#define GETPID_SYSCALL_NB 1

int system_call(int sc_n);
int SWI_Handler_C(int nb_syscall);
int get_pid();

void process()
{

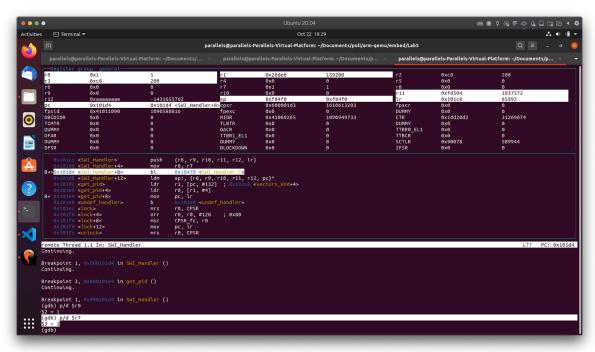
    int x = system_call(GETPID_SYSCALL_NB);
    while(1)
    {

        show_bmp((x / 3)*100, (x % 3)*100, x);
        show_bmp((x / 3)*100, (x % 3)*100, 100);
    }
```

```
int SWI_Handler_C(int nb_syscall)
{
    if (nb_syscall == GETPID_SYSCALL_NB){
        return get_pid();
    }
}
```

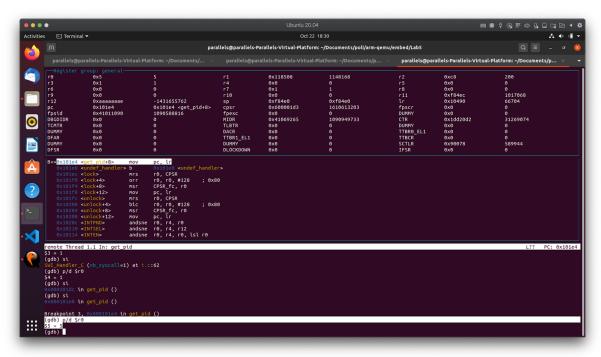
3.2 Demonstração

Na função SWI_Handler é possível observar que o registrador r7 foi setado para 1 que é o número da system call que será chamada.



SWI_Handler_C verifica qual a system call que deve ser chamada, podemos ver que r0 possui o valor 1 que é o da variável nb_syscall.

Abaixo podemos observar o retorno da função get_pid, em r0 temos o número do processo, nesse caso 5.



Por fim retornamos para o processo que chamou a system call, podemos observar que r0 é 5 assim como o valor da variável X.

```
Addition | Oct 27 | 1831 | Oct 27 | Oct 37 |
```

3.3 Resultado Final

