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
        LDR   sp, =stack_top            ;@ a pilha de system eh setada
        MSR   cpsr, r0

        BL    copy_vectors
        BL    init_display
        BL    Timer_Init
        MSR   cpsr_ctl, #0b01010000             ;@ alterando o modo do primeiro
```

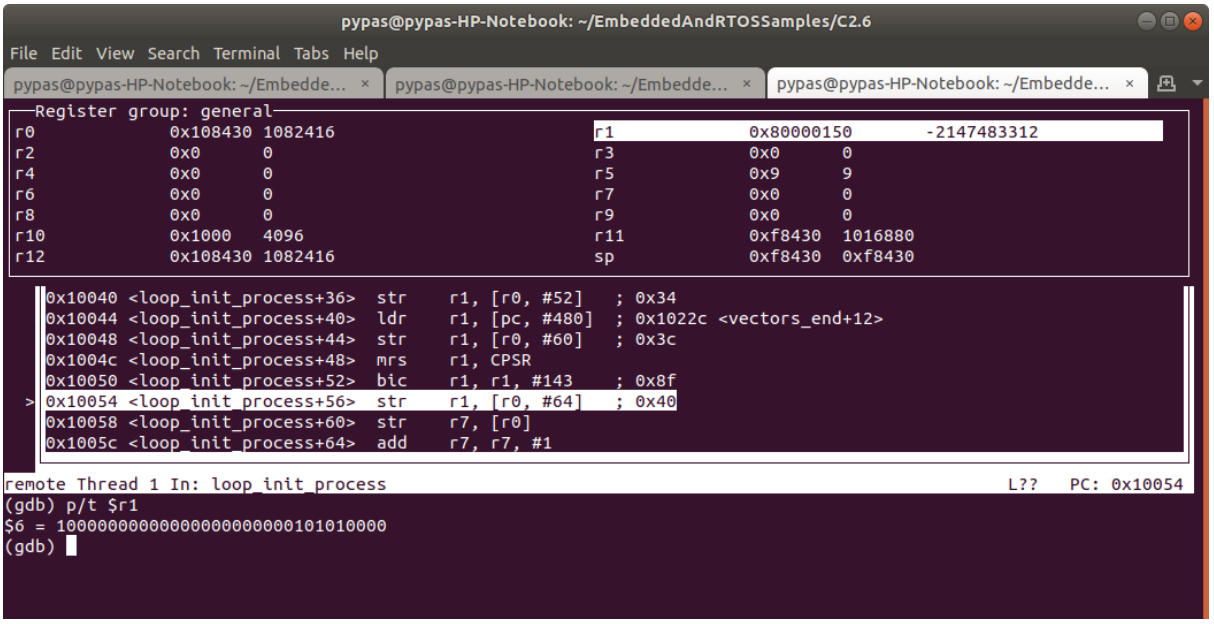
processo para usuario

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.

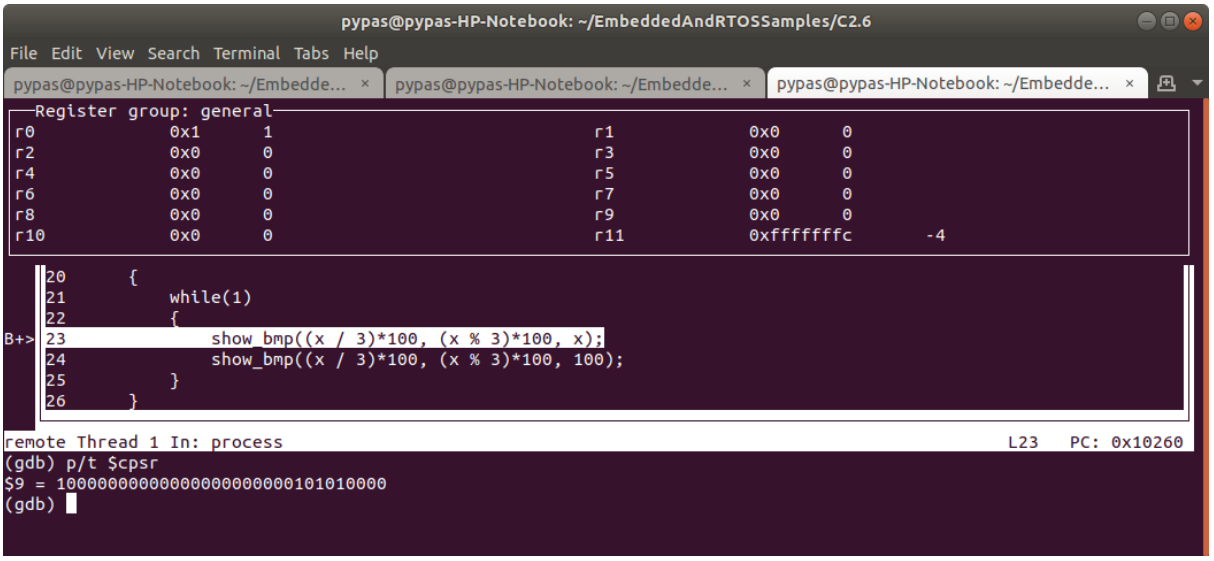
```
Timer_Handler:
        ....
        MRS   r4, spsr                    ;@ cpsr anterior
        MRS   r0, cpsr                    ;@ salvando o modo corrente em R0
        AND   r5, r4, #0b11111
        CMP   r5, #0b10000
        MSREQ   cpsr_ctl, #0b11011111          ;@ alterando o modo para system
        MSRNE   cpsr_ctl, r4          ;@ alterando o modo para o anterior
        MOV   r1, sp                  ;@ sp do processo corrente
        MOV   r2, lr                  ;@ lr do processo corrente
        MSR   cpsr, r0                    ;@ volta para o modo anterior

        …
        MRS   r0, cpsr                    ;@ salvando o modo corrente em R0
        AND   r5, r4, #0b11111
        CMP   r5, #0b10000
        MSREQ   cpsr_ctl, #0b11011111     ;@ alterando o modo para system
        MSRNE   cpsr_ctl, r4               ;@ alterando o modo para o anterior
        LDR   sp, [r12, #52]              ;@ sp = r13 = 13*4 = 52
        LDR   lr, [r12, #56]              ;@ lr = r14 = 14*4 = 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:

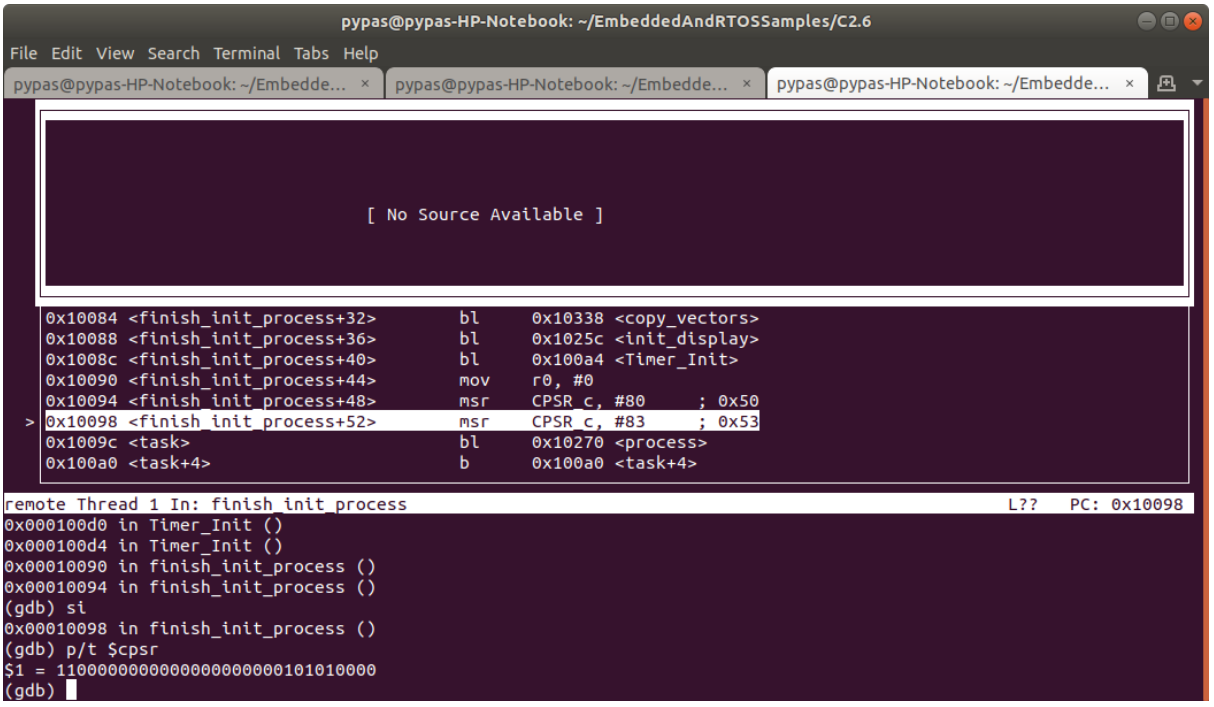


Na próxima figura, verifica-se que os processos estão rodando em modo usuário:

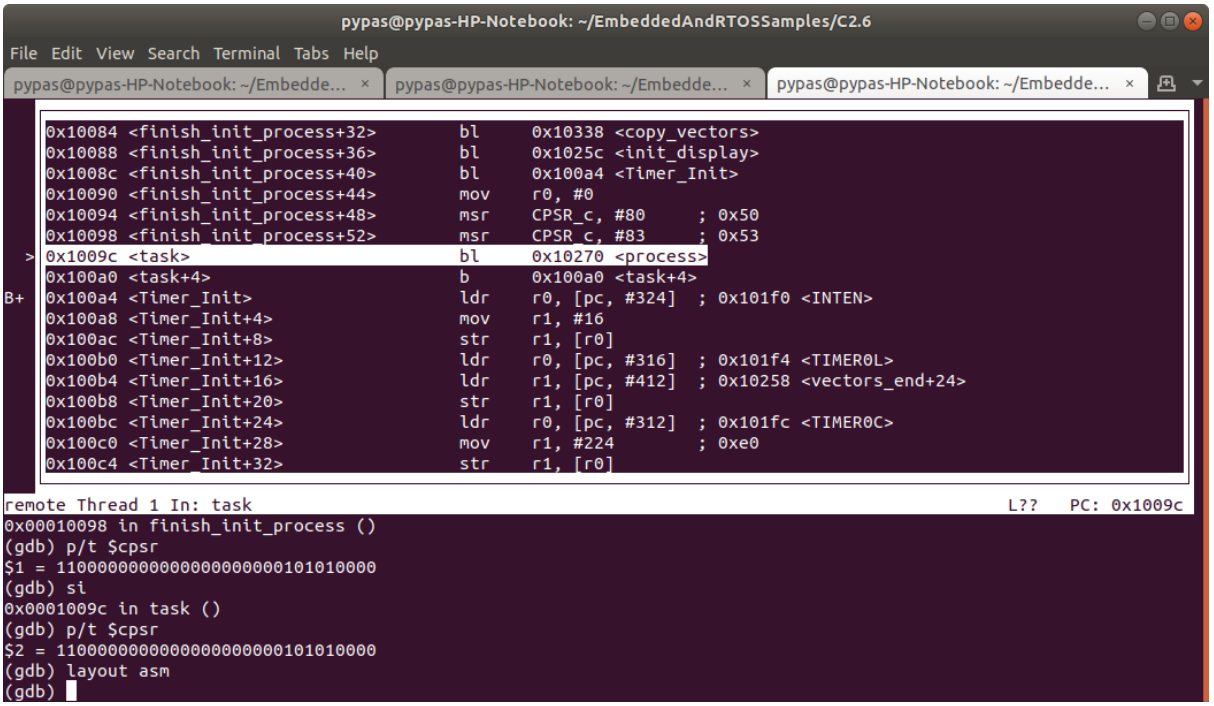


2.3 Chaveamento modo usuário para modo supervisor

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



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, lr

SWI_Handler:
        STMFD   sp!, {r8-r12,lr}            ;@Empilha os registradores
        MOV     r0, r7
        BL      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, lr
```

Em t.c, foram adicionadas:

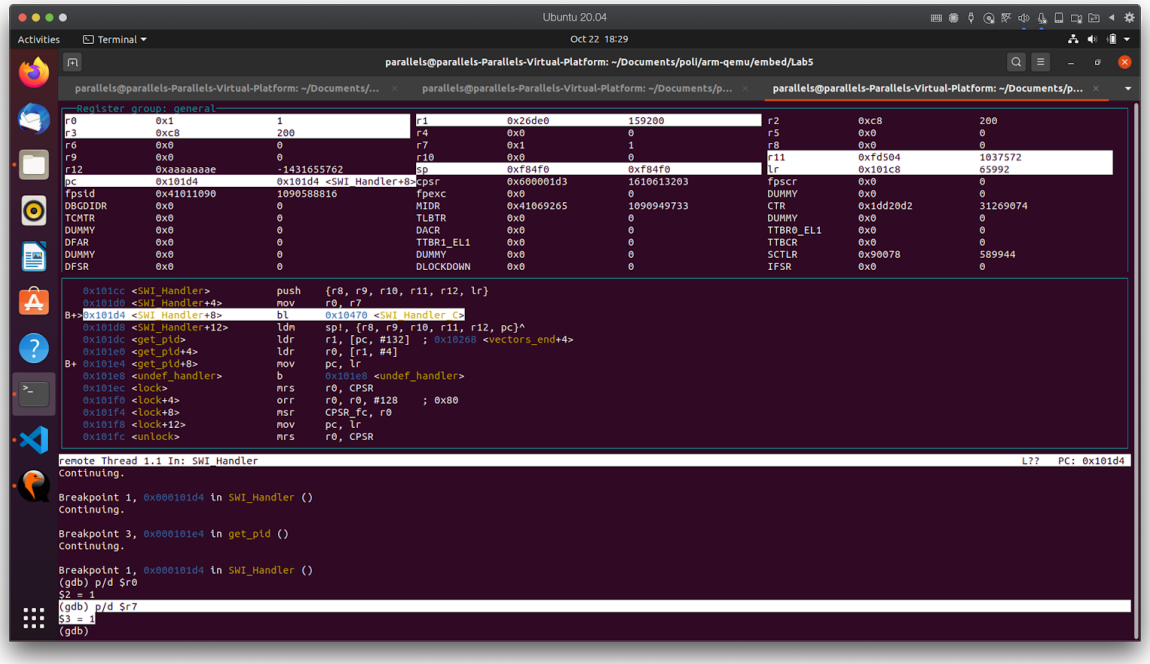
- As assinaturas das funções *system_call* e *get_pid* (definidas em assembly)
- A função *process()*, chamada pelos processos em modo usuário. Essa função chama *system_call*(), passando como parâmetro o código correspondente à system call *get_pid*
- A função *SWI_Handler_C()*, que verifica qual a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

        }
```

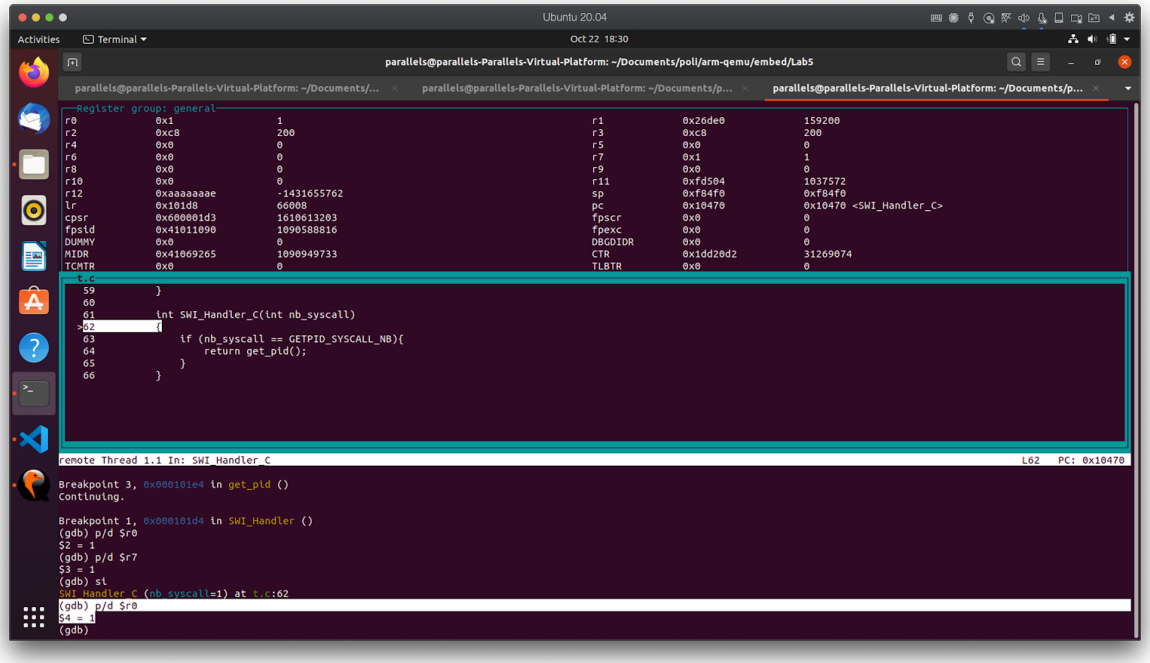
```
}

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

3.2 Demonstração

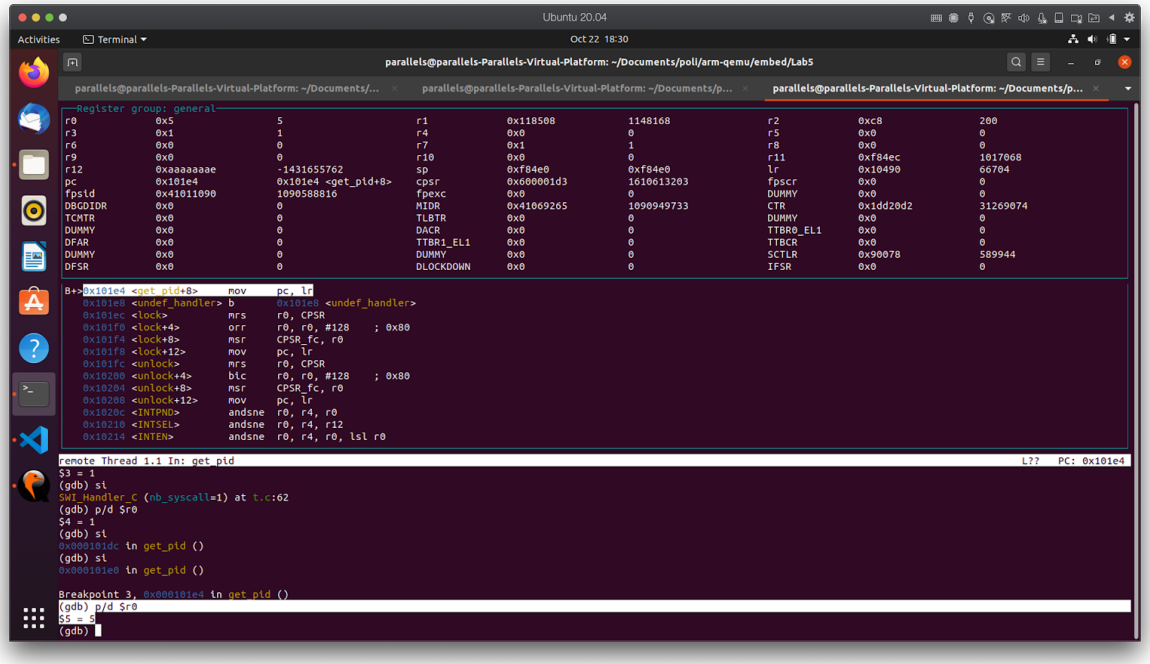
Na função SWI_Handler é possível observar que o registrador r7 foi setado para 1 que é o número da system call que será chamada.



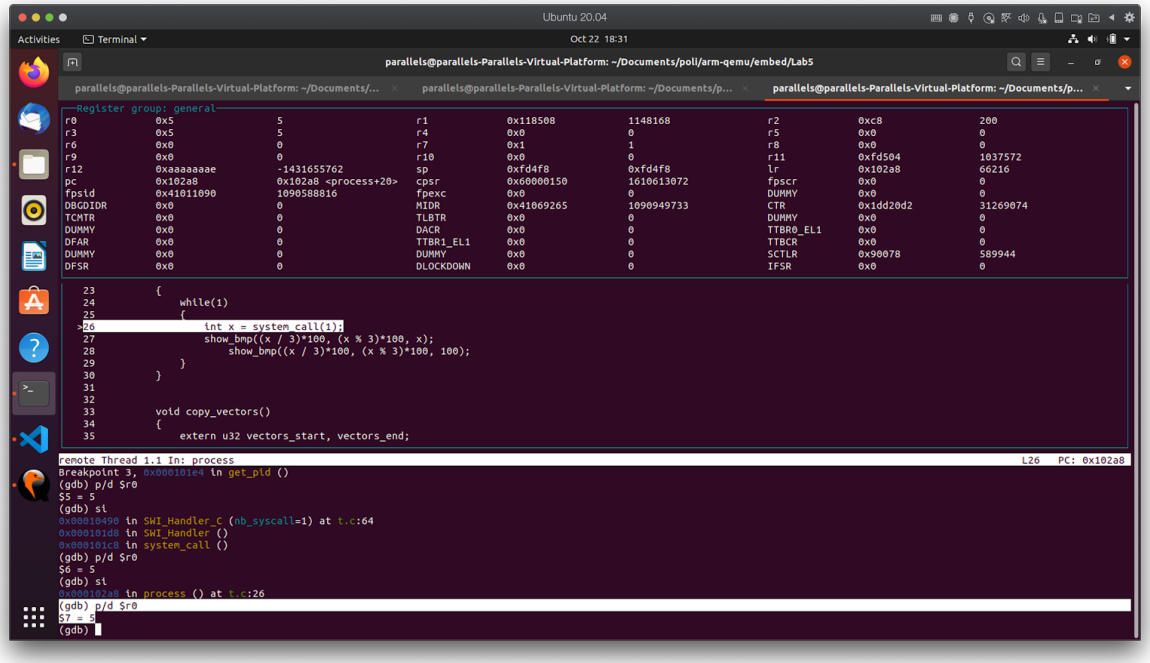
SWI_Handler_C verifica qual a system call que deve ser chamada, podemos ver que r0 possui o valor 1 que é o da variável nb_syscall.



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.



3.3 Resultado Final