Tarefa 11 - Chaveamento entre 2 Processos

Carlos Eduardo Sena Rebouças Garcia 9345255

Diego Boccoli Gallego 9344744

Gabriel Rocha Amorim 9395112

# Introdução

Nesta experiência vamos desenvolver um mini-kernel que salva os registradores de um processo na memória e em seguida carregá da memória registradores de outro processo de modo a cada vez que a interrupção de tempo for chamada. Será feito basicamente chaveamento entre 2 processos. Para isso, temos previamente este código que salva o estado atual de um processo e recupera o estado do próximo:

handle\_process:

LDR r0, =save\_lr @ Libera lr para outros usos

STR r14, [r0]

LDMFD sp!, {r0-r12} @ Recupera r0-r12

LDR lr, =linhaA - 4 @ Usa lr para armazenar ponteiro de linhaA

STMFA lr!, {r0-r12} @ Guarda em linhaA

LDR r2, [sp], #4 @ Recupera pc

MOV r12, lr @ Guarda ponteiro de linhaA em r12

LDR r5, =save\_cpsr

MRS r1, cpsr @ salvando o modo corrente em r1

STR r1, [r5]

LDR r3, =0b10010011

MSR cpsr\_ctl, r3 @ Alterando o modo para supervisor

MOV r0, sp @ No modo supervisor é possível recuperar sp

MOV r1, lr @ E é possível recuperar lr também

STMFA r12!, {r0-r3} @ Guarda os últimos registradores

LDR r0, save\_cpsr @ Volta para o modo anterior

MSR cpsr, r0

LDR lr, =linhaA @ Agora que está tudo salvo, recuperamos

LDR r0, [lr, #60] @ o estado do próximo processo e guardamos no stack

STMFD sp!, {r0}

LDMFD lr!, {r0-r12}

STMFD sp!, {r0-r12}

LDR lr, save\_lr @ Recupera lr antigo

bx lr @ Retorno de subrotina

save\_lr:

.word 0

save\_cpsr:

.word 0

E temos também o código da experiência 10 que servirá como base e não será mostrado aqui.

Para recuperar todos os registradores, é preciso mudar para o modo supervisor já que ao alcançar o código, estaremos no modo IRQ.

# Desenvolvimento

O código atual salva e recupera o estado de apenas um processo. Precisamos fazer algumas modificações para funcionar para 2 processos. Para isso, adicionamos uma variável que determina o processo atual (nproc) e verificamos qual o processo atual no momento e mudamos para o próximo processo. O código final fica:

handle\_process:

LDR r0, =save\_lr

STR r14, [r0]

LDMFD sp!, {r0-r12}

LDR lr, =last\_process

LDR lr, [lr]

SUB lr, lr, #4

STMFA lr!, {r0-r12}

LDR r2, [sp], #4

MOV r12, lr

LDR r5, =save\_cpsr

MRS r1, cpsr @ salvando o modo corrente em r1

STR r1, [r5]

MRS r3, spsr

MRS r7, spsr

ADD r3,r3,#0x80

ADD r7,r7,#0x80

AND r6, r3, #0x1f

CMP r6, #0x10

ADDEQ r7, r3, #0xf

MSR cpsr\_ctl, r7 @ alterando o modo para supervisor

MOV r0, sp

MOV r1, lr

STMFA r12!, {r0-r3}

LDR r0, save\_cpsr

MSR cpsr, r0

LDR r0, =nproc

LDR r1, [r0]

ADD r1, r1, #1

AND r1, r1, #1

STR r1, [r0]

CMP r1, #0

LDREQ r0, =linhaA

LDRGT r0, =linhaB

STR r1, [r0]

LDR lr, last\_process

LDR r0, [lr, #60]

STMFD sp!, {r0}

LDMFD lr!, {r0-r12}

STMFD sp!, {r0-r12}

LDR lr, save\_lr

BX lr

save\_lr:

.word 0

save\_cpsr:

.word 0

last\_process:

.word 0x3050

Agora, devemos criar funções para inicializar cada um dos processos. Observação: É necessário declarar uma pilha separada para o processo B para guardar as variáveis dele separadamente.

**Processo A:**

init\_procA:

LDR r0, =linhaA

LDR r1, =0x0

LDR r2, =0x0

loop:

CMP r1, #52

BGT out

STR r2, [r0, r1]

ADD r1, r1, #4

B loop

out:

MOV r4, sp

LDR r5, =0x0

MOV r6, lr

MRS r7, cpsr

ADD r0, r0, r1

STMFA r0!, {r4-r6}

BX lr

**Processo B:**

init\_procB:

LDR r0, =linhaB

LDR r1, =0x0

LDR r2, =0x0

loop2:

CMP r1, #52

BGT out2

STR r2, [r0, r1]

ADD r1, r1, #4

B loop2

out2:

MOV r4, sp

SUB r4, r4, #200

LDR r5, =0x0

MOV r6, lr

MRS r7, cpsr

ADD r0, r0, r1

STMFA r0!, {r4-r6}

BX lr

Também modificamos a função principal para imprimir 1 ou 2 dependendo do processo atual que está rodando.

**Função Main:**

main:

bl c\_entry

LDR r0, =nproc

LDR r1, =0x00

STR r1, [r0]

BL init\_procA

BL init\_procB

BL timer\_init @initialize interrupts and timer 0

stop:

LDR r0, =nproc

LDR r0, [r0]

CMP r0, #0

BLEQ print\_1

BLGT print\_2

B stop

E finalmente as funções que imprimem strings foram desenvolvidas em c e são da seguinte forma:

volatile unsigned int \* const UART0DR = (unsigned int \*)0x101f1000;

void print\_uart0(const char \*s) {

while(\*s != '\0') { /\* Loop until end of string \*/

\*UART0DR = (unsigned int)(\*s); /\* Transmit char \*/

s++; /\* Next char \*/

}

}

void c\_entry() {

print\_uart0("Hello world!\n");

}

void print\_1() {

print\_uart0("1");

}

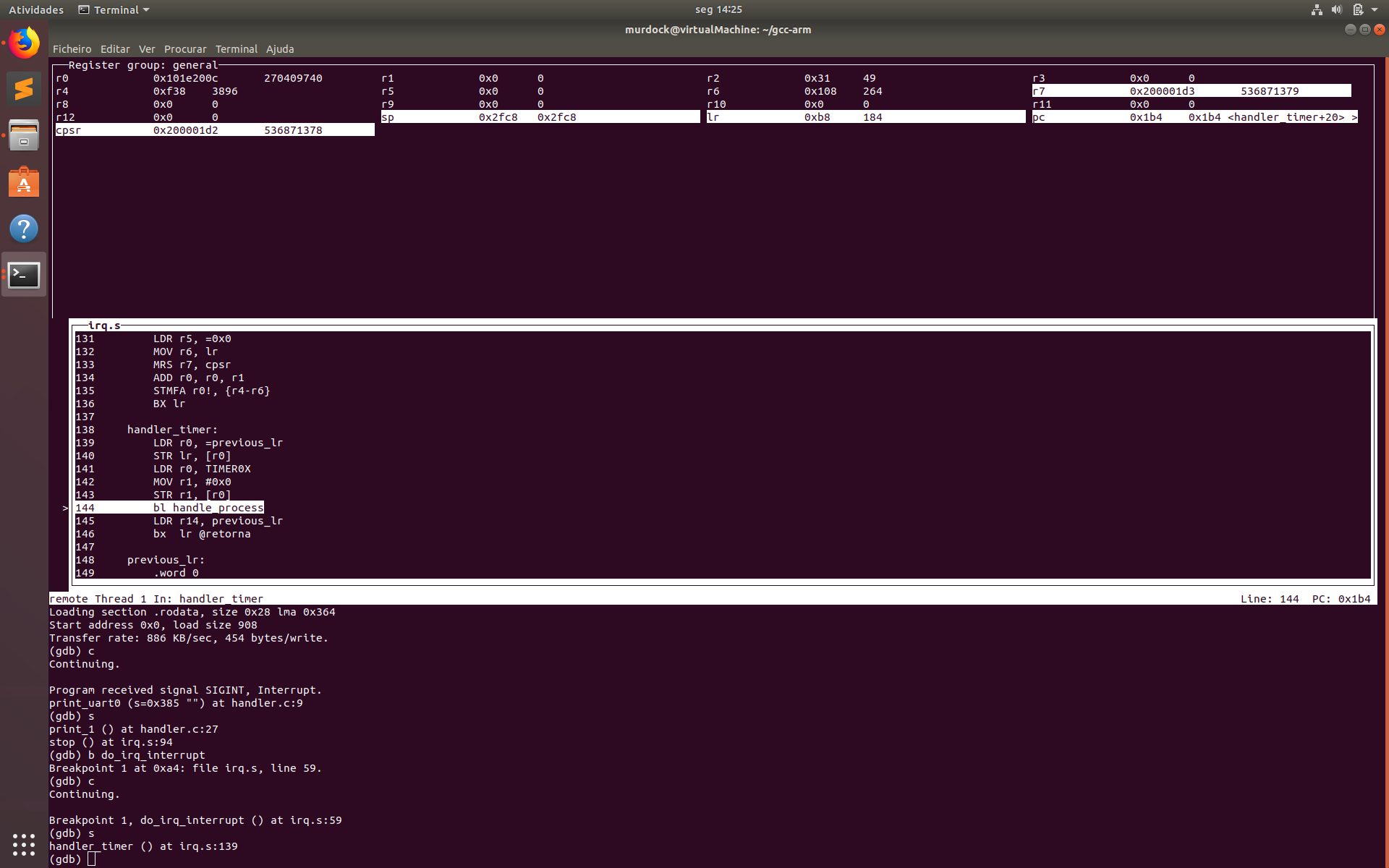
void print\_2() {

print\_uart0("2");

}

# Conclusão

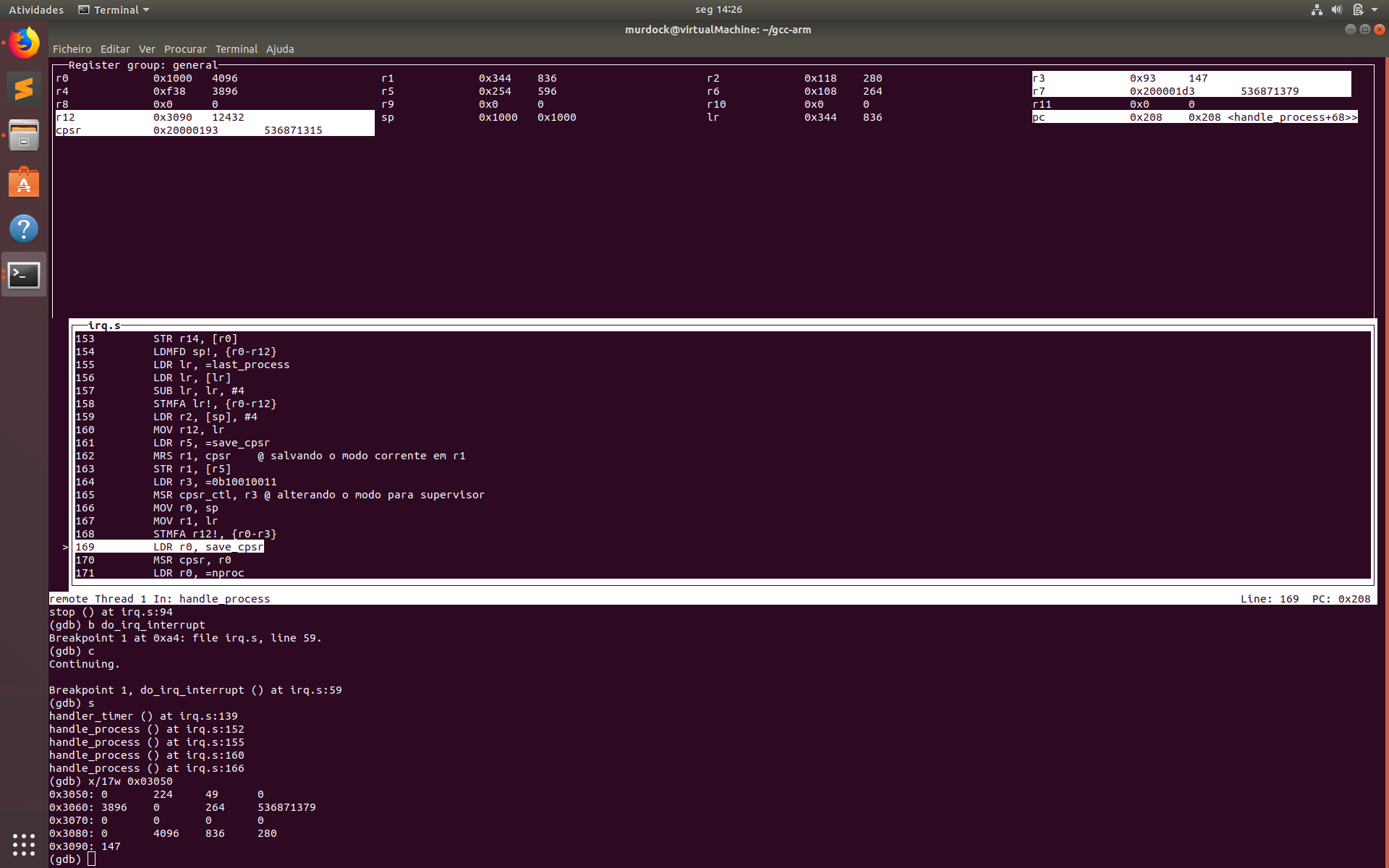
Fazendo as alterações necessárias, podemos ver a função handle\_process ser chamada para trocar o processo ativo:



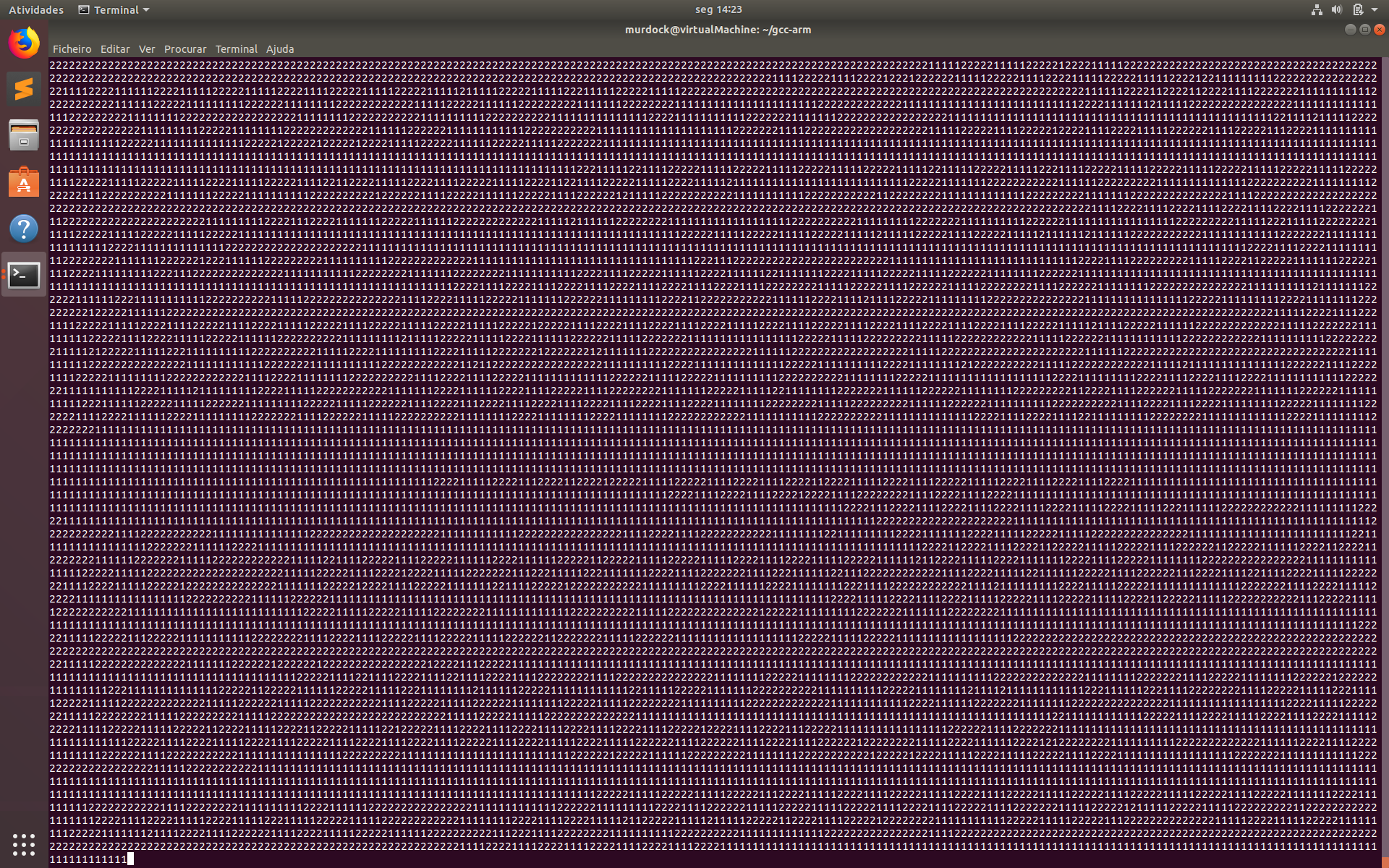
Usando o comando:

x/17w 0x03050

para mostrar o valor da estrutura linhaA, podemos ver os registradores salvos de maneira correta.



E este é o resultado final: impressão de 1’s e de 2’s de maneira alternada que depende da interrupção do timer.



E com isso, concluímos esta experiência permitindo assim um maior entendimento em relação ao funcionamento do kernel, de processos e do vetor de interrupções.