Relatório P3 SO II

Gustavo Kundlatsch, Paola de Oliveira, Pedro Souza

28 de setembro de 2022

O objetivo do grupo era corrigir os problemas que apareceram no P2 e implementar todos os requisitos do P3. As modificações que fizemos de inicio pareceram corrigir o P2, e os testes estavam passando. Porém, conforme fomos desenvolvendo a integração dos requisitos do P3, chegamos em algum ponto onde o sistema não inicializa mais, apesar de gerar a imagem corretamente. Como não percebemos em que momento isso começou a ocorrer, não conseguimos descobrir o motivo, e escrevemos o relatório para explicar a entrega, destacando os principais pontos da implementação sucintamente.

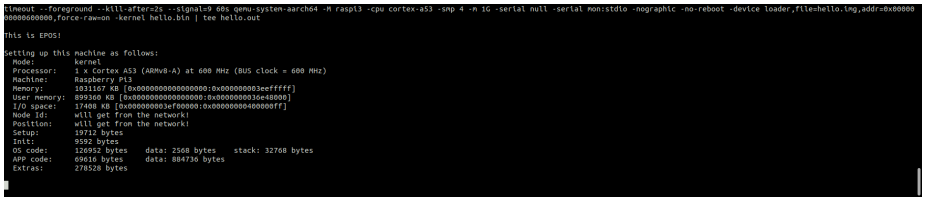

Figura 1: Erro não resolvido.

1 Stubs/Agents

A arquitetura adotada foi a seguinte: os arquivos relacionados a stubs foram dispostos no diretório `/include/syscall`, e cada um dos componentes possui um header próprio com o nome `stub_<componente>`: address_space, alarm, chronometer, clock, condition, delay, fork, mutex, segment, semaphore, task e thread. Todos os métodos dos stubs são descritos por um enum presente na classe Message. Por fim, a classe Agent possui um handler para cada tipo de mensagem recebida, e a trata de acordo com o comportamento esperado:

```
void exec() {
        switch(entity()) {
            case Message::ENTITY::FORK:
                handle_fork();
                break;
            case Message::ENTITY::DISPLAY:
                handle_display();
                break;
            case Message::ENTITY::THREAD:
                handle_thread();
                break;
            case Message::ENTITY::TASK:
                handle_task();
                break;
            case Message::ENTITY::ADDRESS_SPACE:
                handle_address_space();
                break;
            case Message::ENTITY::SEGMENT:
                handle_segment();
                break;
            case Message::ENTITY::MUTEX:
                handle_mutex();
                break;
            case Message::ENTITY::SEMAPHORE:
                handle_semaphore();
                break;
            case Message::ENTITY::CONDITION:
                handle_condition();
                break;
            case Message::ENTITY::CLOCK:
                handle_clock();
                break;
            case Message::ENTITY::ALARM:
                handle_alarm();
                break;
            case Message::ENTITY::DELAY:
```

```
            handle_delay();
            break;
        case Message::ENTITY::CHRONOMETER:
            handle_chronometer();
            break;
        default:
            break;
    }
}
```

2 Syscall

A syscall está separada em duas etapas: syscall e syscalled, ambas implementadas no diretório `/src/architecture/armv8`. O arquivo `armv8_cpu_syscall.cc` possui a interface para realizar uma SVC passando o código da syscall (mensagem) pelo registrador x0. Já a o arquivo `armv8_cpu_syscalled.cc` executa a função `_sysexec`, que é a função `_exec()` da classe Agent, responsável por chavear o código da chamada com uma função de componente do SO conforme descrito na seção anterior.

Syscall:

```
#include <architecture/armv8/armv8_cpu.h>

__BEGIN_SYS

void CPU::syscall(void * msg)
{

    ASM(
        "str x0, [sp, #-8]!   \n"
        "mov x0, %0     \n"
        "SVC 0x0    \n"
        "ldr x0, [sp, #8]!   \n"
        "" :: "r"(msg)
    );
}

__END_SYS
```

Syscalled:

```
#include <architecture/armv8/armv8_cpu.h>

extern "C" { void _sysexec(); }

__BEGIN_SYS

void CPU::syscalled() {
      ASM("str lr, [sp, #-8]!   \n"
          "str x0, [sp, #-8]!   \n"
          "bl _sysexec          \n"
          "ldr x0, [sp, #8]!    \n"
          "ldr lr, [sp, #8]!    \n"
      );
}

__END_SYS
```

3 User Stack

Para utilizar uma stack do usuário foi inserida a função `init_user_stack` no arquivo `armv8_cpu.h`, que libera espaço para um contexto, assim como a função `init_stack` já fazia mas liberado espaço para um contexto adicional com a função `_go_user_mode`, que é implementada no arquivo `cortex_ic.cc` que basicamente realiza um pop para receber os valores nos registradores desse contexto. Para isso funcionar, no pop do contexto precisou ser alterado.

Função init_user_stack:

```
static Context * init_user_stack(Log_Addr usp,
Log_Addr ksp, void (* exit)(), int (* entry)(Tn ...), Tn ... an) {
        ksp -= sizeof(Context);
        Context * ctx = new(ksp) Context(entry, exit, usp, false);
        init_stack_helper(&ctx->_x0, an ...);
        ksp -= sizeof(Context);
        ctx = new(ksp) Context(&_go_user_mode, 0, 0, true);
        return ctx;
    }
```

Função _go_user_mode:

```
void _go_user_mode() {
    ASM("       ldr   x30, [sp], #8                         \t\n\
                ldp    x0,  x1, [sp], #16                   \t\n\
                ldp    x2,  x3, [sp], #16                   \t\n\
                ldp    x4,  x5, [sp], #16                   \t\n\
                ldp    x6,  x7, [sp], #16                   \t\n\
                ldp    x8,  x9, [sp], #16                   \t\n\
                ldp   x10, x11, [sp], #16                   \t\n\
                ldp   x12, x13, [sp], #16                   \t\n\
                ldp   x14, x15, [sp], #16                   \t\n\
                ldp   x16, x17, [sp], #16                   \t\n\
                ldp   x18, x19, [sp], #16                   \t\n\
                ldp   x20, x21, [sp], #16                   \t\n\
                ldp   x22, x23, [sp], #16                   \t\n\
                ldp   x24, x25, [sp], #16                   \t\n\
                ldp   x26, x27, [sp], #16                   \t\n\
                ldp   x28, x29, [sp], #16                   \t\n\
                msr   spsr_el1, x30                         \t\n\
                ldr   x30, [sp], #8                         \t\n\
                msr   ELR_EL1, x30                          \t\n\
                eret                                  \t" : : : "cc");
}
```

4 Interrupts

Foi adicionado uma lógica no `IC::entry` para verificar se a interrupção deve simplesmente chamar dispatch ou se é uma syscall acontecendo e deve realizar o procedimento correto:

```
switch (CPU::esr_el1() >> 26) {
    case 0:
        dispatch(i);
        break;
    case 0x15:
        CPU::esr_el1(0);
        CPU::int_enable();
        CPU::syscalled();
        break;
    default:
        dispatch(i);
        break;
}
```