实验报告

黄文橡 211240032

2022 年 12 月 18 日

上下文结构体的前世今生

赋值位置

1. gpr 于trap.S将寄存器的值依次复制到gpr中
2. mcause 于trap.S复制cpu.sr中的值，cpu.sr于isa_raise_intr赋值
3. mstatus 于trap.S复制cpu.sr中的值，cpu.sr未进行赋值
4. mepc 于trap.S复制cpu.sr中的值，cpu.sr于isa_raise_intr赋值
5. pdir 于vme.c通过__am_get_cur_as或__am_switch进行赋值

联系

trap.S中通过新指令访问cpu中的sr固定位置的值，保存与$ISA-nemu.h中固定内存偏移的位置。因而需要组织Context中的成员，保证赋值给正确的成员。

理解穿越时空的旅程

· 所有异常出现前进行cte_init

· cte_init将__am_asm_trap的地址通过csrw赋给cpu.sr中MTVEC

· yield()将-1置于a7寄存器中

· yield()调用ecall指令

· ecall指令调用isa_raise_intr函数，传入EVENT_YIELD与当前pc

- isa_raise_intr将传入的编号与异常地址分别赋给cpu.sr中MCAUSE与MEPC位置
- isa_raise_intr返回cpu.sr中MTVEC位置的值并在ecall指令中赋给dnpc作为下一条指令的pc
- 跳转至__am_asm_trap
- 找到Context所在位置存于栈指针寄存器
- 通过MAP宏将除去零寄存器与栈指针寄存器均复制至Context.gpr中
- 将cpu.sr中MCAUSE、MSTATUS、MEPC读取保存到Context中
- 令cpu.sr[MSTATUS]—=0x20000，用于进行DiffTest
- 传入Context参数，并跳转至__am_irq_handle
- 如果用户对事件具有处理方法，那么根据mcause进行对Event进行处理
- 若mcause编号为EVENT_YIELD，则根据a7的值判断自陷或系统调用（系统调用没有对a7进行赋值）
- 调用NANOS-LITE的do_event，根据Event的事件类型进行处理
- 此处进入EVENT_YIELD类型，打印信息便沿着调用路径直接返回至__am_asm_trap
- 将mstatus与mepc重新保存到cpu.sr的MSTATUS与MEPC位置
- 通过MAP宏将Context中的gpr除去零寄存器与栈指针寄存器均恢复至cpu.gpr中
- 另栈指针寄存器恢复到调用前的值
- 通过mret将下一条指令位置设置为异常地址。

hello程序是什么

该回答是基于PA3所有内容完成后编写

一开始在哪里

由make ARCH=$ISA-nemu update命令完成编译链接置于navy-apps/fsimg/bin/hello再整合为ramdisk.img置于nanos-lite/build/, 并在files.h中设置其偏移与大小

如何出现在内存

由resources.S读取置于nanos-lite的image的.data节，由nemu读取。

为什么会出现在目前的内存位置

由nemu读取至nemu的内存位置，加上ramdisk_start偏移以及hello前面各项文件的大小得到目前的内存位置

第一条指令位置

第一条指令位置存在hello对应的内存地址中第25 28字节，具体值为0x8304d24

如何执行到其第一条指令

nemu运行nanos-lite，通过nanos-lite的loader读取内存中的hello获取其Elf文件的header获取程序入口（即其第一条指令位置），将其强制转换为void(*)()类型函数指针，并调用，通过nemu的指令跳转至第一条指令位置。

字符如何输出

第一行字符： 其首字符指针经由 _write, _syscall_, do_event 通过Context传到 do_syscall 中，再传入 fs_write 通过 serial_write 逐个字符将整行通过 putch(AM), outb, 指令解析, paddr_write, mmio_write, map_write, host_write 写入串口所在的物理内存, 并通过 invoke_callback, serail_io_handler, serial_putc, putch(PM) 打印至终端。

其余字符串： 首字符指针经由 printf 传入 _vfprintf_r 进行匹配替换，将内容保存直至遇到换行符或字符串内容大于指定值，再调用 write 通过上一段的方式进行输出。

仙剑奇侠传如何运行

- 在PAL_SplashScreen中传入mgo.mkf的fp两次调用PAL_MKFReadChunk分别读取title和crane
- title和crane是仙鹤的两个部分，是两部分数据在仙鹤文件中的offset

- 在PAL_MKFReadChunk函数中，通过libc中fseek→libos中的_lseek→Nanos中的fs_lseek确定好数据头的位置
- 通过PAL_fread→libc中的fread读取出数据块长度和位置
- 通过fseek设定读取起始位置
- 由libc中fread→libos中_read→Nanos中的fs_read→Nanos中的ramdisk_read→AM中的memcpy→NEMU指令完成数据读取
- libos中的_read→Nanos中的函数还有一段进行系统调用的路径
- libos中_read→libos中_syscall_调用ecall指令
- NEMU中ecall保存当前pc并跳转→__am_asm_trap保存寄存器并跳转
- __am_irq_handle将event类型设置为EVENT_SYSCALL通过函数指针跳转
- Nanos中do_event调用do_syscall
- Nanos中do_syscall根据libos中_syscall_保存在a0寄存器的系统调用类型识别出SYS_read并跳转至fs_read
- Nanos中的fs_read得到数据并返回长度
- Nanos中do_syscall将长度保存在a7寄存器，一直返回
- __am_asm_trap恢复寄存器并调用mret
- NEMU中mret返回到最初的pc的下一条指令开始处。