实验报告

黄文橡 211240032

2023 年 1 月 15 日

实验进度

我完成了所有实验内容

分时多任务的具体过程

硬件中断

此处，仅解释硬件中断如何从hello切换至pal，pal至hello基本同理。

NEMU 每条指令结束后，通过isa_query_intr通过检查cpu的INTR来确定是否有设备发出了中断请求

NEMU 当收到中断请求，进入isa_raise_intr，将mstatus的MIE置0以防止嵌套中断，进入AM的中断处理

AM 进入trap.S，hello是用户程序，故将sp切换至内核栈，在内核栈上保存hello的Context，通过__am_irq_handle识别出中断来源于硬件(时钟)中断，进入NANOS的事件处理

NANOS 在do_event中识别出硬件（时钟）中断，调用schedule将hello内核栈上的上下文保存到current→cp，并切换至pal内核栈上的上下文（，输出NANOS接收到时钟中断的信息）返回至AM

AM 返回trap.S，恢复为pal内核栈中保存的上下文，并通过pal的Context中的np将sp切换至pal的用户栈，调用mret

NEMU 执行mret指令，将pc移至pal将要执行的下一条指令，并将mstatus中的MIE置1（用以继续接收中断）

分页机制

NEMU vaddr_read, vaddr_ifetch, vaddr_write提供虚拟内存的访问，isa_mmu_translate提供虚拟内存地址转换为物理内存地址的服务

AM 在vme.c中提供了protect拷贝原数据, map构建虚拟地址到物理地址的映射

NANOS 在loader和context_uload，context_kload中通过以上函数以及new_page申请物理页完成了程序的加载

AM 在__am_irq_handle操作中，首先将当前的SATP保存至hello的Context中，待切换至pal的Context，再将pal的Context中的页表目录设置为SATP，完成分页切换。

理解计算机系统

其汇编代码如下

```
0000000000000000 <main>:
   0:   f3 0f 1e fa             endbr64
   4:   55                      push   %rbp
   5:   48 89 e5                mov    %rsp,%rbp
   8:   48 8d 05 00 00 00 00    lea    0x0(%rip),%
      rax        # f <main+0xf>
   f:   48 89 45 f8             mov    %rax,-0x8(%
      rbp)
  13:   48 8b 45 f8             mov    -0x8(%rbp),%
     rax
  17:   c6 00 41                movb   $0x41,(%rax)
  1a:   b8 00 00 00 00          mov    $0x0,%eax
  1f:   5d                      pop    %rbp
  20:   c3                      ret
```

回答 第8行（¡main+0x17¿）指令为出错操作（p[0] = 'A'）。这条指令进行了内存访问，MMu会检查当前进程对这块内存的访问权限。“abc”在编译时会被放入只读代码段，因而无权访问。由MMU记录异常信息，转到操作系统的异常处理程序，操作系统向该进程发送SIGSEGV信号，表示发生了段错误。用户进程收到SIGSEGV的默认行为是终止。

证明

由readelf检查程序的out文件即可确认”abc”进入了只读代码段。

通过strace检查程序运行的具体细节，如下所示，下面有个人的理解

```
1  execve("./a.out", ["./a.out"], 0x7fffe8af6700
       /* 49 vars */) = 0
2  brk(NULL)                                = 0
       x55f4091ec000
3  arch_prctl(0x3001 /* ARCH_??? */, 0
       x7ffd15aa8400) = -1 EINVAL (Invalid argument
       )
4  mmap(NULL, 8192, PROT_READ|PROT_WRITE,
       MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) = 0
       x7f3af0639000
5  access("/etc/ld.so.preload", R_OK)       = -1
       ENOENT (No such file or directory)
6  openat(AT_FDCWD, "/etc/ld.so.cache", O_RDONLY|
       O_CLOEXEC) = 3
7  newfstatat(3, "", {st_mode=S_IFREG|0644,
       st_size=87407, ...}, AT_EMPTY_PATH) = 0
8  mmap(NULL, 87407, PROT_READ, MAP_PRIVATE, 3, 0)
        = 0x7f3af0623000
9  close(3)                                 = 0
10 openat(AT_FDCWD, "/lib/x86_64-linux-gnu/
       libc.so.6", O_RDONLY|O_CLOEXEC) = 3
11 read(3, "\177ELF
       \2\1\1\3\0\0\0\0\0\0\0\0\3\0>\0\1\0\0\0P
```

```
    \237\2\0\0\0\0\0"..., 832) = 832
12  pread64(3, "\6\0\0\0\4\0\0\0@\0\0\0\0\0\0\0@
    \0\0\0\0\0\0\0@\0\0\0\0\0\0\0"..., 784, 64)
    = 784
13  pread64(3, "\4\0\0\0␣\0\0\0\5\0\0\0GNU
    \0\2\0\0\300\4\0\0\0\3\0\0\0\0\0\0\0"...,
    48, 848) = 48
14  pread64(3, "\4\0\0\0\24\0\0\0\3\0\0\0GNU\0i8
    \235HZ\227\223\333\350s\360\352,\223\340."
    ..., 68, 896) = 68
15  newfstatat(3, "", {st_mode=S_IFREG|0644,
    st_size=2216304, ...}, AT_EMPTY_PATH) = 0
16  pread64(3, "\6\0\0\0\4\0\0\0@\0\0\0\0\0\0\0@
    \0\0\0\0\0\0\0@\0\0\0\0\0\0\0"..., 784, 64)
    = 784
17  mmap(NULL, 2260560, PROT_READ, MAP_PRIVATE|
    MAP_DENYWRITE, 3, 0) = 0x7f3af03fb000
18  mmap(0x7f3af0423000, 1658880, PROT_READ|
    PROT_EXEC, MAP_PRIVATE|MAP_FIXED|
    MAP_DENYWRITE, 3, 0x28000) = 0x7f3af0423000
19  mmap(0x7f3af05b8000, 360448, PROT_READ,
    MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0
    x1bd000) = 0x7f3af05b8000
20  mmap(0x7f3af0610000, 24576, PROT_READ|
    PROT_WRITE, MAP_PRIVATE|MAP_FIXED|
    MAP_DENYWRITE, 3, 0x214000) = 0x7f3af0610000
21  mmap(0x7f3af0616000, 52816, PROT_READ|
    PROT_WRITE, MAP_PRIVATE|MAP_FIXED|
    MAP_ANONYMOUS, -1, 0) = 0x7f3af0616000
22  close(3)                                = 0
23  mmap(NULL, 12288, PROT_READ|PROT_WRITE,
    MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) = 0
    x7f3af03f8000
```

```
24      arch_prctl(ARCH_SET_FS, 0x7f3af03f8740) = 0
25      set_tid_address(0x7f3af03f8a10)         = 2416
26      set_robust_list(0x7f3af03f8a20, 24)     = 0
27      rseq(0x7f3af03f90e0, 0x20, 0, 0x53053053) = 0
28      mprotect(0x7f3af0610000, 16384, PROT_READ) = 0
29      mprotect(0x55f407325000, 4096, PROT_READ) = 0
30      mprotect(0x7f3af0673000, 8192, PROT_READ) = 0
31      prlimit64(0, RLIMIT_STACK, NULL, {rlim_cur
           =8192*1024, rlim_max=RLIM64_INFINITY}) = 0
32      munmap(0x7f3af0623000, 87407)           = 0
33      --- SIGSEGV {si_signo=SIGSEGV, si_code=
           SEGV_ACCERR, si_addr=0x55f407324004} ---
34      +++ killed by SIGSEGV +++
35      Segmentation fault
```

由execve a.out进入程序，中间如20行，尝试对固定的不可写的部分进行了读写操作，并在最后确实抛出了SIGSEGV信号，且程序由SIGSEGV信号终止。