

PA4 虚实交错的魔法：分时多任务

AC, but not so AC.

1.多道程序

4.1好像要实现的东西还挺多的，首先要实现上下文切换。但这个好像还比较简单。具体实现的一个图如下：

```
[/home/rainy/ics2023/nanos-lite/src/proc.c,16,hello_fun] Hello World from Nanos-  
lite with arg '00000000' for the 994th time!  
[/home/rainy/ics2023/nanos-lite/src/proc.c,16,hello_fun] Hello World from Nanos-  
lite with arg '80002f3c' for the 995th time!  
[/home/rainy/ics2023/nanos-lite/src/proc.c,16,hello_fun] Hello World from Nanos-  
lite with arg '00000000' for the 995th time!  
[/home/rainy/ics2023/nanos-lite/src/proc.c,16,hello_fun] Hello World from Nanos-  
lite with arg '80002f3c' for the 996th time!  
[/home/rainy/ics2023/nanos-lite/src/proc.c,16,hello_fun] Hello World from Nanos-  
lite with arg '00000000' for the 996th time!  
[/home/rainy/ics2023/nanos-lite/src/proc.c,16,hello_fun] Hello World from Nanos-  
lite with arg '80002f3c' for the 997th time!  
[/home/rainy/ics2023/nanos-lite/src/proc.c,16,hello_fun] Hello World from Nanos-  
lite with arg '00000000' for the 997th time!  
[/home/rainy/ics2023/nanos-lite/src/proc.c,16,hello_fun] Hello World from Nanos-  
lite with arg '80002f3c' for the 998th time!
```

然后要修改ucontext：

```
Context *ucontext(AddrSpace *as, Area kstack, void *entry) {  
    char* end = kstack.end - sizeof(Context);  
    Context* c = (Context*)end;  
    //printf("sp: %p\n", sp);  
    //c->gpr[2] = sp;  
    c->mepc = (uintptr_t)entry;  
    //printf("c->mepc:%p\n", c->mepc);  
    c->mstatus = 0x1800;  
    //printf("jijij\n");  
    //printf("as:%p\n", as);  
    c->pdir = as->ptr;  
    //printf("jijij\n");  
    //printf("c->pdir:%p\n", c->pdir);  
    //printf("yes1\n");  
    return c;  
}
```

```
}

```

emmm, 除去调试的信息其实也就这些，我们需要修改程序的mepc，mstatus，pdir（新加的一个指针）

然后要实现第一版的uload，这个里面就按照书上的一个图来写就可以了，第二版会加入分页机制然后大改：

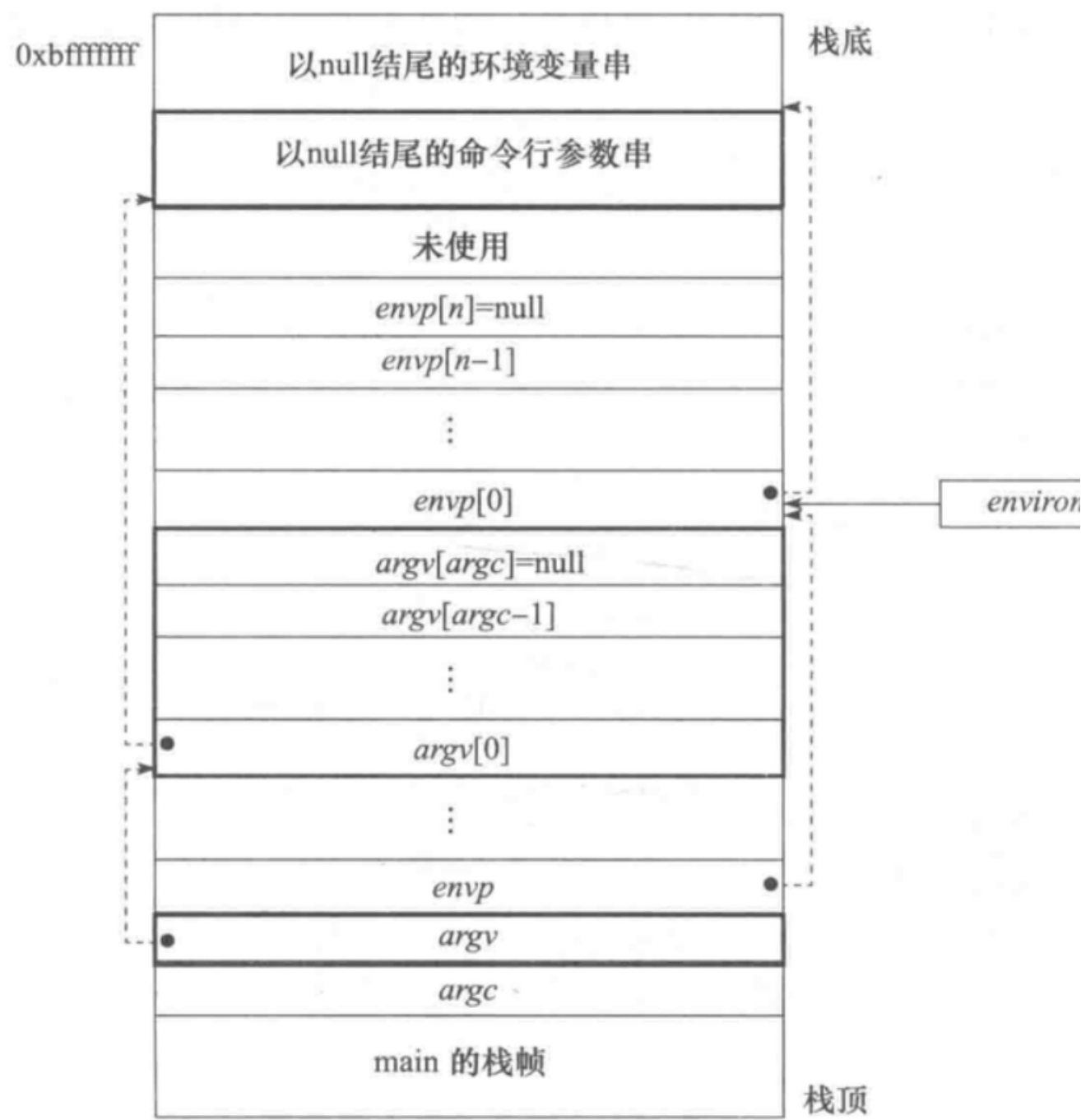


图 7.7 运行一个新程序的 main 函数时用户栈中的典型结构

这里我都实现了，下面有图可看：在第一版loader的情况下，我们可以得到：

```
#ifndef HAS_VME
Elf_Ehdr ehdr;
int filed = fs_open(filename, 0, 0);
fs_read(filed, &ehdr, sizeof(Elf_Ehdr));
assert(*(uint32_t *)ehdr.e_ident == 0x464c457f);
Elf_Phdr phdr[ehdr.e_phnum];
fs_lseek(filed, ehdr.e_phoff, SEEK_SET);
fs_read(filed, phdr, sizeof(Elf_Phdr) * ehdr.e_phnum);
for(int i = 0; i < ehdr.e_phnum; i++){
    if(phdr[i].p_type == PT_LOAD){
        fs_lseek(filed, phdr[i].p_offset, SEEK_SET);
        fs_read(filed, (void*)phdr[i].p_vaddr, phdr[i].p_memsz);
        memset((void*)(phdr[i].p_vaddr + phdr[i].p_filesz), 0,
phdr[i].p_memsz - phdr[i].p_filesz);
    }
}
fs_close(filed);
//printf("1111\n");
//printf("entry: %p\n", ehdr.e_entry);
return ehdr.e_entry;
```

这个是第一版的loader

uload有实现如：

```

rainy@rainy-ASUS-TUF-Gaming-F15-FX507ZM-FX507ZM: ~/ics2023/nanos-lite
es...
[/home/rainy/ics2023/nanos-lite/src/ramdisk.c,29,init_ramdisk] ramdisk info: st
art = 80003be9, end = 821149e9, size = 34672128 bytes
[/home/rainy/ics2023/nanos-lite/src/irq.c,16,init_irq] Initializing interrupt/e
xception handler...
[/home/rainy/ics2023/nanos-lite/src/proc.c,45,init_proc] Initializing processes
...
User stack:
argc: 1
argv[0]: /bin/nterm
[/home/rainy/ics2023/nanos-lite/src/main.c,30,main] Finish initialization
User stack:
argc: 2
argv[0]: /bin/dummy
argv[1]: dummy
cur:82137000
sys-exit
[src/cpu/cpu-exec.c:201 cpu_exec] nemu: HIT GOOD TRAP at pc = 0x80001184
[src/cpu/cpu-exec.c:101 statistic] host time spent = 1,533,157 us
[src/cpu/cpu-exec.c:102 statistic] total guest instructions = 49,173,041
[src/cpu/cpu-exec.c:103 statistic] simulation frequency = 32,073,062 inst/s
make[1]: 离开目录“/home/rainy/ics2023/nemu”
rainy@rainy-ASUS-TUF-Gaming-F15-FX507ZM-FX507ZM:~/ics2023/nanos-lite$

```

2.虚实交错的魔法

讲了一下分段的问题，具体一点来说已经在提示我们的mmu需要实现了，这个时候又要转回到nemu基础了。

3.超越容量的界限

这一章我做到了最后Vme=1加mm那里，不知道为什么运行所有的navy就会爆炸，主要是我现在在虚拟存储上运行hellommy都是没问题的。

先从头看：分页机制在riscv32上面是非常复杂的，具体来说，重写了接近10次。

首先不需要用到虚拟存储的时候，执行的是直接映射，那么在直接映射下面我们实现这个就需要在nemu里面把imm改了，还有vaddr：

```

int isa_mmu_check(word_t vaddr,int len,int type){
    //printf("%d\n",cpu.riscv32_csr.satp);
    if (cpu.riscv32_csr.satp == 0) {
        return MMU_DIRECT;
    }else{
        //word_t valid2 = cpu.riscv32_csr.satp & 0x1;
        //printf("1:%d\n",cpu.riscv32_csr.satp >> 31);
        return MMU_TRANSLATE;
    }
    //return MMU_DIRECT;
}

//static int cnt = 0;

typedef struct {
    uint32_t valid : 1;
    uint32_t read : 1;
    uint32_t write : 1;
    uint32_t exec : 1;
    uint32_t user : 1;
    uint32_t global : 1;
    uint32_t accessed : 1;
    uint32_t dirty : 1;
    uint32_t rsw : 2;
    uint32_t ppn : 22;
} pte_t;

pte_t word_to_pte(word_t word) {
    pte_t pte;
    pte.valid = word & 0x1;
    pte.read = (word >> 1) & 0x1;
    pte.write = (word >> 2) & 0x1;
    pte.exec = (word >> 3) & 0x1;
    pte.user = (word >> 4) & 0x1;
    pte.global = (word >> 5) & 0x1;
    pte.accessed = (word >> 6) & 0x1;
    pte.dirty = (word >> 7) & 0x1;
    pte.rsw = (word >> 8) & 0x3;
    pte.ppn = (word >> 10) & 0x3ffffff;
    return pte;
}

word_t pte_to_word(pte_t pte) {
    word_t word = 0;
    word |= pte.valid;
    word |= pte.read << 1;
    word |= pte.write << 2;
    word |= pte.exec << 3;
    word |= pte.user << 4;
    word |= pte.global << 5;
    word |= pte.accessed << 6;
    word |= pte.dirty << 7;

```

```

    word |= pte.rsw << 8;
    word |= pte.ppn << 10;
    return word;
}

/**/
paddr_t isa_mmu_translate(vaddr_t vaddr, int len, int type)
{
#define v1 (vaddr & 0xffc00000)
#define v0 (vaddr & 0x003ff000)
#define vf (vaddr & 0x00000fff)

    //printf("\n");
    //printf("%x\n", vaddr);

    paddr_t add_stap = (cpu.riscv32_csr.satp & 0x003fffff);
    paddr_t first_level_pt = add_stap * PAGE_SIZE + (v1 >> 22) * 4;

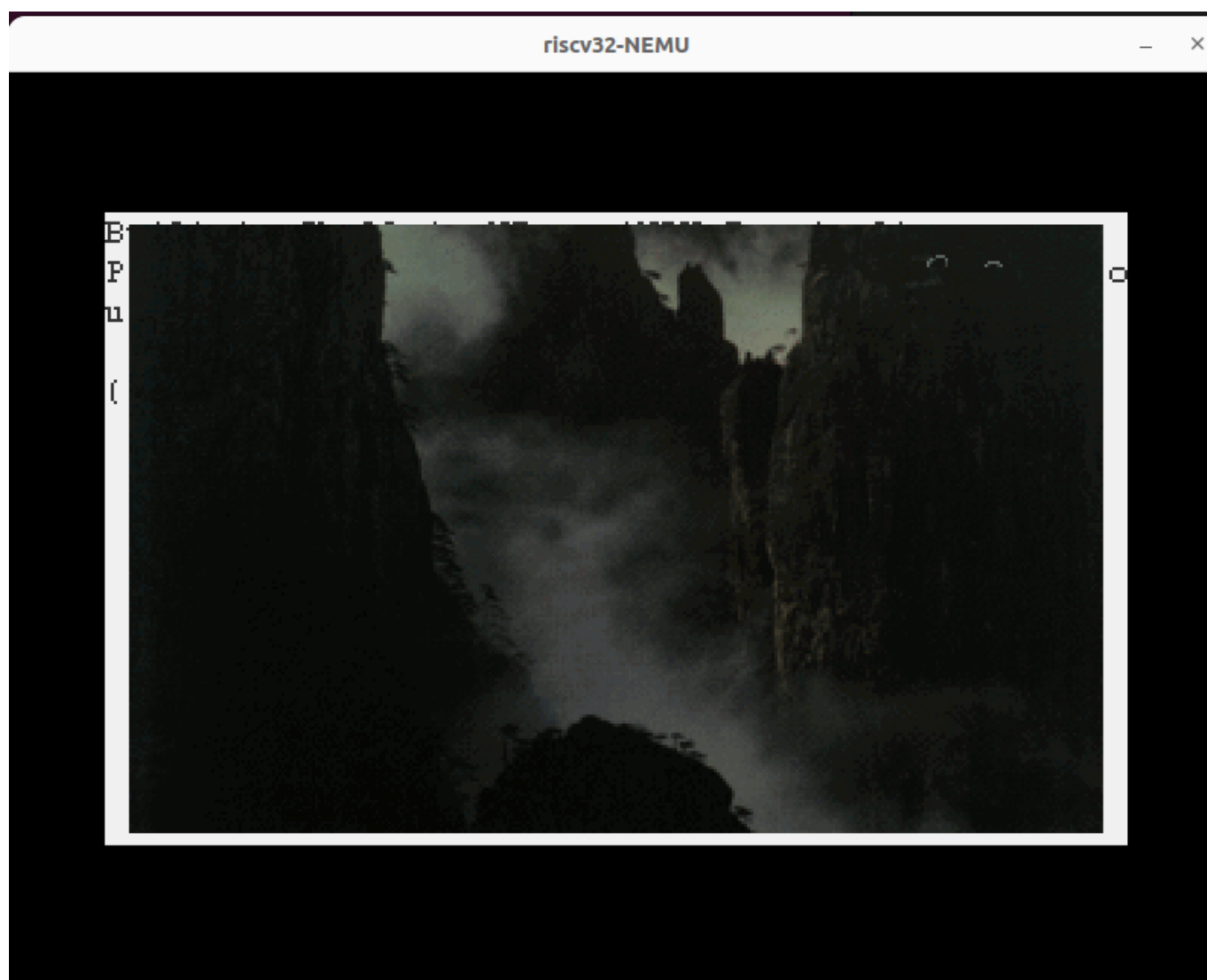
    word_t first_level_pt_word = host_read(guest_to_host(first_level_pt), 4);
    pte_t first_level_pte = word_to_pte(first_level_pt_word);

    //printf("fist: %x\n", first_level_pt_word);

    paddr_t physical_address;
    if(!first_level_pte.valid){
        assert(0);
    }
    if((first_level_pte.read | first_level_pte.write | first_level_pte.exec)
    != 7){
        paddr_t second_level_pt_add = first_level_pte.ppn * PAGE_SIZE + (v0 >>
12) * 4;
        word_t second_level_pt_word =
host_read(guest_to_host(second_level_pt_add), 4);
        //printf("secpt: %x\n", second_level_pt_add);
        pte_t second_level_pte = word_to_pte(second_level_pt_word);
        physical_address = (second_level_pte.ppn << 12) | vf;
    }
    else{
        //printf("jijijiji\n");
        physical_address = (first_level_pte.ppn << 22) | v0 | vf;
    }
    //printf("phy: %x\n", physical_address);
    //assert(physical_address == vaddr);
    return physical_address;
}

```

这是我根据riscv32表写出来的一个抽象的映射函数，但是目前看了，直接映射是没有问题的（如果想测试把loader改一改就行了，我定义了宏的）
具体实现效果如下：



截止目前，在Vme=1之前的我全部都实现了，仙剑也是正常运行

目前看起来一切良好，然后最抽象的事情来了，我就在loader上面加了一个分页的map映射（本质上map也是写好了的之前，而且真的mm_brk确实没啥理由可以错，莫名其妙的报错就出现了）

```
rainy@rainy-ASUS-TUF-Gaming-F15-FX507ZM-FX507ZM: ~/ics2023/nanos-lite
[/home/rainy/ics2023/nanos-lite/src/proc.c,45,init_proc] Initializing processes.
..
80000000
ustack , usp_va: 82173fff 7fffffff
pcb:7fff8000 8216c000
pcb:7fff9000 8216d000
pcb:7fffa000 8216e000
pcb:7fffb000 8216f000
pcb:7fffc000 82170000
pcb:7fffd000 82171000
pcb:7fffe000 82172000
pcb:7ffff000 82173000
arglen envlen:1 0
4000ded8
sp: 7fffffe4
pcb->cp: 82126f70
[/home/rainy/ics2023/nanos-lite/src/main.c,30,main] Finish initialization
cin : 82147f50
cin : 82136f60
cin : 7ffffef0
===== Do SYS_brk now! =====
cin : 7ffffef0
===== Do SYS_brk now! =====
address (0x00000054) is out of bound at pc = 0x40002310
118 //printf("%c\n", filename);
```

尝试着去修改吧，然后试了各种方法，gdb，elf，printf一条一条的看，汇编一条一条的跟踪，还是00000054越位，把printf删了就换了一个地方越位，但是这种越位限于navy的程序，意思是程序进入用户了，巨抽象的事情就是我拿dummy和hello都来看了，还是越位，我寻思是不是对齐的问题吧，把对齐的机制在loader加上了，还是炸。我又觉得有可能是缓冲区的问题，改了长度，还是不对。


```
rainy@rainy-ASUS-TUF-Gaming-F15-FX507ZM-FX507ZM:~/ics2023/nanos-lite/build$ readelf ramdisk.img -l
```

Elf 文件类型为 EXEC (可执行文件)

Entry point 0x4000018c

There are 5 program headers, starting at offset 52

程序头:

Type	Offset	VirtAddr	PhysAddr	FileSiz	MemSiz	Flg	Align
RISCV_ATTRIBUTES	0x022fd9	0x00000000	0x00000000	0x0002e	0x00000	R	0x1
LOAD	0x000000	0x40000000	0x40000000	0x22311	0x22311	R E	0x1000
LOAD	0x022318	0x40023318	0x40023318	0x00cc1	0x0113c	RW	0x1000
NOTE	0x0000d4	0x400000d4	0x400000d4	0x00024	0x00024	R	0x4
GNU_STACK	0x000000	0x00000000	0x00000000	0x00000	0x00000	RW	0x10

Section to Segment mapping:

段节...

00	.riscv.attributes
01	.note.gnu.build-id .text .rodata
02	.data .sdata .sbss .bss
03	.note.gnu.build-id
04	

```
rainy@rainy-ASUS-TUF-Gaming-F15-FX507ZM-FX507ZM:~/ics2023/nanos-lite/build$ make ARCH=riscv32-nemu run
```

为了防止是之前1, 2, 3的一些历史遗留吧, 我把讲义从1-4重新看了一遍, 发现在3的里面有一个地方提到:

“如果你不解决这个问题, 有可能在pa4遇到难以解释的bug”

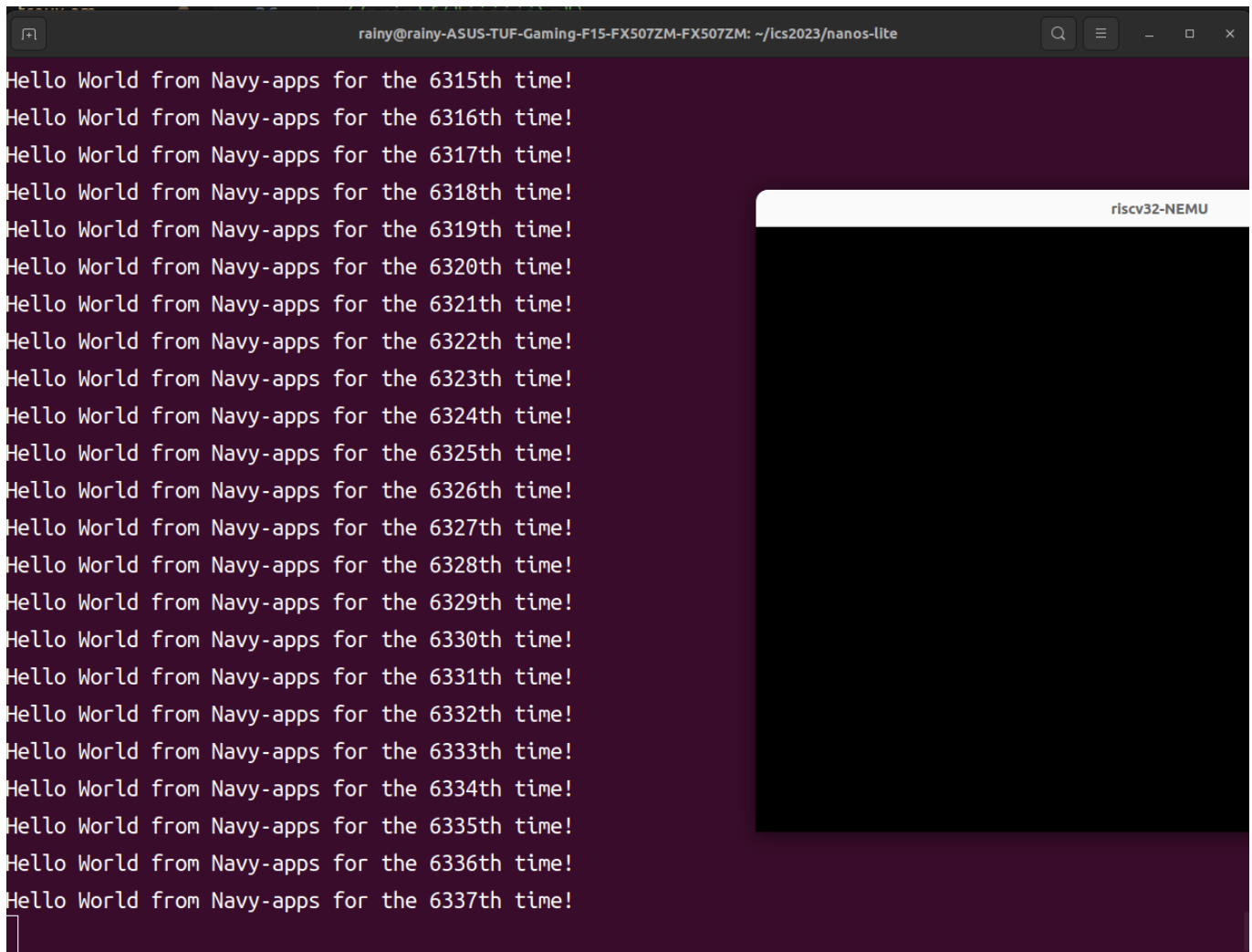
看到这句话本来我觉得没希望了, 又去根据context的问题去追溯, 好家伙, 没有任何的收获。

心态炸了。目前以上的这个越位bug我从11.27看到接近12.10, 真的没有任何办法可以实现解决。

以下附上一个Vme=1下面正常运行的dummy和hello截图:

```
rainy@rainy-ASUS-TUF-Gaming-F15-FX507ZM-FX507ZM: ~/ics2023/nanos-lite
#####$` .:$|`';!!!;`'%$%' ;#####$
#####$` .:$&|`.'.'%&%' ;@#####$
#####$` .:$&$&&%' ;#####$
#####$` .:$&%' ;@#####$
#####$` ;#####$

**Project-N**
Nanjing University Computer System Project Series
Build a computer system from scratch!
[/home/rainy/ics2023/nanos-lite/src/main.c,13,main] 'Hello World!' from Nanos-lite
[/home/rainy/ics2023/nanos-lite/src/main.c,14,main] Build time: 22:40:52, Dec 6 2023
[/home/rainy/ics2023/nanos-lite/src/mm.c,56,init_mm] free physical pages starting from 82147000
[/home/rainy/ics2023/nanos-lite/src/device.c,86,init_device] Initializing devices...
[/home/rainy/ics2023/nanos-lite/src/ramdisk.c,29,init_ramdisk] ramdisk info: start = 80003cd9, end = 8
21114d9, size = 34658304 bytes
[/home/rainy/ics2023/nanos-lite/src/irq.c,17,init_irq] Initializing interrupt/exception handler...
[/home/rainy/ics2023/nanos-lite/src/proc.c,45,init_proc] Initializing processes...
[/home/rainy/ics2023/nanos-lite/src/main.c,30,main] Finish initialization
[src/cpu/cpu-exec.c:201 cpu_exec] nemu: HIT GOOD TRAP at pc = 0x80001148
[src/cpu/cpu-exec.c:101 statistic] host time spent = 66,280 us
[src/cpu/cpu-exec.c:102 statistic] total guest instructions = 2,066,137
[src/cpu/cpu-exec.c:103 statistic] simulation frequency = 31,172,857 inst/s
make[1]: 离开目录“/home/rainy/ics2023/nemu”
rainy@rainy-ASUS-TUF-Gaming-F15-FX507ZM-FX507ZM:~/ics2023/nanos-lite$
```



```
rainy@rainy-ASUS-TUF-Gaming-F15-FX507ZM-FX507ZM: ~/ics2023/nanos-lite
Hello World from Navy-apps for the 6315th time!
Hello World from Navy-apps for the 6316th time!
Hello World from Navy-apps for the 6317th time!
Hello World from Navy-apps for the 6318th time!
Hello World from Navy-apps for the 6319th time!
Hello World from Navy-apps for the 6320th time!
Hello World from Navy-apps for the 6321th time!
Hello World from Navy-apps for the 6322th time!
Hello World from Navy-apps for the 6323th time!
Hello World from Navy-apps for the 6324th time!
Hello World from Navy-apps for the 6325th time!
Hello World from Navy-apps for the 6326th time!
Hello World from Navy-apps for the 6327th time!
Hello World from Navy-apps for the 6328th time!
Hello World from Navy-apps for the 6329th time!
Hello World from Navy-apps for the 6330th time!
Hello World from Navy-apps for the 6331th time!
Hello World from Navy-apps for the 6332th time!
Hello World from Navy-apps for the 6333th time!
Hello World from Navy-apps for the 6334th time!
Hello World from Navy-apps for the 6335th time!
Hello World from Navy-apps for the 6336th time!
Hello World from Navy-apps for the 6337th time!
```

后来，想到了一个不是办法的办法，找到了做完的同学一起研究研究，用他的代码来diff test（我承认我用了这个代码，但是仅限于思路），后来发现根本不是这里的问题，就我自己又用回我之前写的了。

目前没有找到任何解决的办法，我通过询问助教和同学已经试了太多次，实在是没招了。

如果寒假有机会我会继续完成这个pa4.2这个残留的bug，一定给他找出来，12月份实在是太忙了，学业压力太大。

问题回答：

分时多任务的具体过程 请结合代码，解释分页机制和硬件中断是如何支撑仙剑奇侠传和hello程序在我们的计算机系统(Nanos-lite, AM, NEMU)中分时运行的。

理解计算机系统 尝试在Linux中编写并运行以下程序:

```
int main() { char *p = "abc"; p[0] = 'A'; return 0; }
```

你会看到程序因为往只读字符串进行写入而触发了段错误. 请你根据学习的知识和工具, 从程序, 编译器, 链接器, 运行时环境, 操作系统和硬件等视角分析"字符串的写保护机制是如何实现的". 换句话说, 上述程序在执行p[0] =

'A'的时候, 计算机系统究竟发生了什么而引发段错误? 计算机系统又是如何保证段错误会发生? 如何使用合适的工具来证明你的想法?

回答：分页机制的运作。