PA4 实验报告

221300034 高志轩

必答题：

Question1:

分时多任务的具体过程

分页机制：

分页机制是由 MM（Memory Manager）存储管理器模块进行管理。它需要先准备一些内核页表（框架代码实现），而在 nanos 中，我们只需要在 common.h 中定义宏 HAS_VME，初始化的时候就会调用 init_mm 来对 MM 进行初始化。

而初始化 MM 首先需要通过 new_page 函数来分配空闲的物理页，然后调用 AM 中的 vme_init 函数。vme_init 函数会设置页面分配和回收的回调函数（即 Nanos-lite 中提供的 pg_alloc 和 free_page 函数）。然后，调用了 map 函数来对内核虚拟空间的页目录和页表进行填写。最后，设置一个 satp 寄存器来开启分页机制（satp 寄存器需要我们在 NEMU 的 riscv32_CPU_state 结构体中新添加一个成员 satp）。

与原先没有分页机制不同的是，原来我们都是在真实的物理内存中进行直接访存，而现在引进了虚拟存储的机制，让所有进程都有自己的虚拟存储空间，这样一来，我们就需要在访存的时候进行一些改变。具体地, 我们需要实现以下两点:

- 如何判断 CPU 当前是否处于分页模式?
- 分页地址转换的具体过程应该如何实现?

所以，对应地 NEMU 中添加了 API：isa_mmu_check, isa_mmu_translate. 其中 isa_mmu_check 函数，通过(((cpu.satp & 0x80000000) >> 31) == 1) ? MMU_TRANSLATE : MMU_DIRECT 来检查所查看的地址究竟是不是需要进行地址转换，然后 vaddr.c 中进行虚拟地址访存的时候如果需要进行地址转换，就调用 isa_mmu_translate 的 API 来获得正确的物理地址。最后需要对 vaddr.c 中的几个虚拟地址访问的函数进行一些修改，调用 isa_mmu_check 来进行判断，并进行相应的操作。

硬件中断：

硬件中断实际上添加一个数字信号，CPU 接收到这个信号后进行对中断请求的响应。

具体地，这个地方我们也要考虑两个问题:

第一个问题就是中断信号是怎么传到 CPU 中的。

这里我们依然像上面添加 satp 寄存器一样，在 riscv32_CPU_state 结构体中添加一个新的 bool 成员 INTR，来模拟 CPU 的 INTR 引脚，当设备需要发出中断请求的时候，将中断引脚设置为高电平。

第二个问题是 CPU 如何响应到来的中断请求。

CPU 每次执行完一条指令的时候，就让 CPU 去查看 INTR 引脚，看是否有中断的请求（除非 CPU 处于关中断的状态）。关中断状态可以由软件来控制，在 riscv32 中，如果 mstatus 的 MIE 位为 0，代表 CPU 处于关中断的状态。

落实到代码上，在 cpu-exec.c 中，我们在每条指令执行后，添加对是否要进行中断的询问，即通过调用 isa_query_intr()（nemu/src/isa/riscv32/system）来进行查看，而在 isa_query_intr()API 中，通过对 mstatus 的第四位（从低到高）进行查看，如果是 1 并且 INTR 也为 1，那么就代表需要对当前进程进行打断，返回 IRQ_TIMER 信号。回到 cpu-exec 中，如果需要中断，就会调用 isa_raise_intr()来让处理器进入关中断的状态，即将 mstatus.MIE 保存到 mstatus.MPIE 中, 然后将 mstatus.MIE 位置为 0。此外，还需要对 mret 指令进行修改，将 mstatus.MPIE 还原到 mstatus.MIE 中, 然后 mstatus.MPIE 位置为 1。

对于软件层:

AM: 在 CTE 中我们添加一个时钟中断事件，即在__am_irq_handle 中添加一项 case IRQ_TIMER 来进行判断。（这个地方由于对 EVENT_IRQ_TIMER 和 IRQ_TIMER 有所混淆，将二者混为一谈，导致了 debug 了挺长时间。。。）

Nanos-lite: 在 irq.c 中需要对 do_event 函数进行扩充，添加一项 EVENT_IRQ_TIMER，从而让操作系统对当前的进程进行调度，即调用 schedule 函数给当前进程。换句话说就是让当前的进程到时间了就让出位置给其他进程。

此外，我们修改了 kcontext()和 ucontext()函数，来设置正确的中断状态，使得将来恢复上下文之后 CPU 处于开中断的状态。（在这一点上我一开始并没有注意，只是实现了上面几点，发现 mstatus 一直是 0，并不会进行时钟中断，后来发现是这个地方没有修改，也是 debug 了挺长时间。。。）

Question2

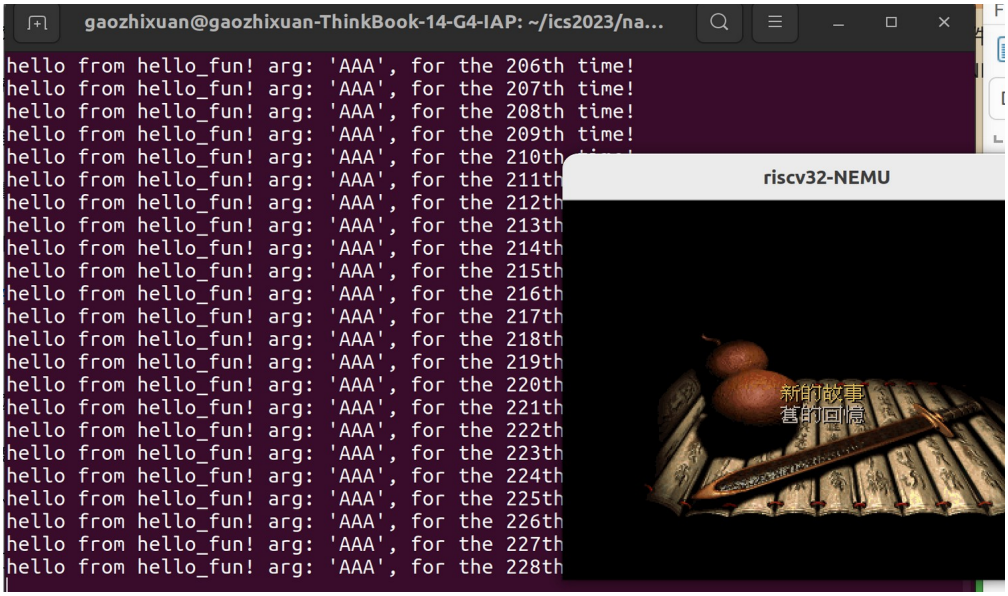
理解计算机系统:

该程序中定义了一个只读字符串，生成 ELF 文件的时候该数据将会在.rodata 中，连接后将会映射到虚拟存储空间的只读段（包括.init, .text, .rodata），不可修改。

于是当程序执行时，CPU 执行到一条想要修改只读段中的数据时，会尝试执行写操作，当 CPU 通过 MMU 试图进行地址转换时会检测到想要访问的数据是只读属性却想要对其修改时，会触发一个硬件中断(page fault)，操作系统捕捉到页故障中断，于是生成一个段错误信号(SIGSEGV)，是一个通知进程发生了非法内存访问的信号。进程接收到了段错误的信号通常就会被操作系统终止，防止对系统产生进一步损害。

PA4 中的完成进度：

1.实现了基本的多道程序系统，支持带参数的仙剑奇侠传与 hello 内核线程的分时运行，可以通过带有 --skip 参数跳过片头动画。

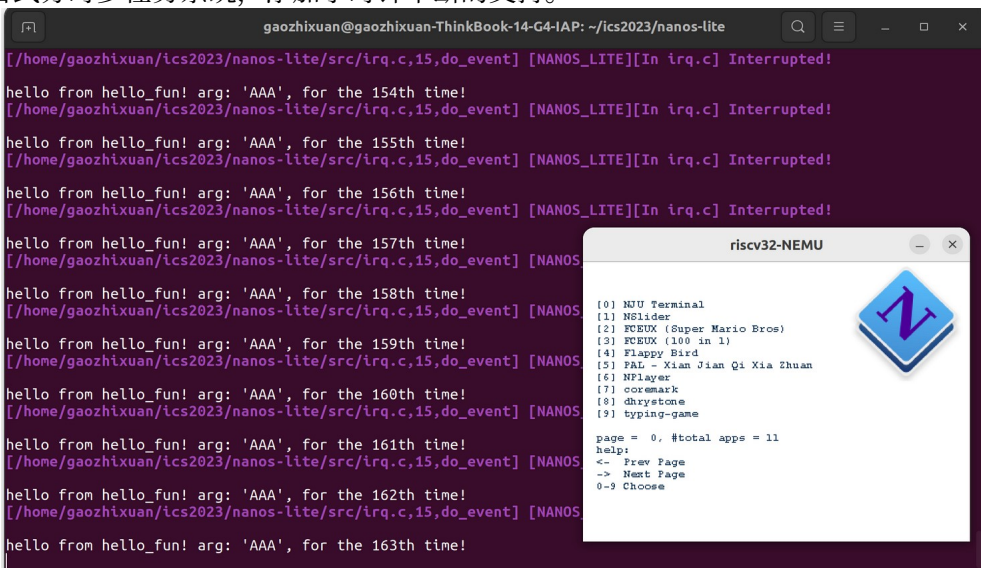


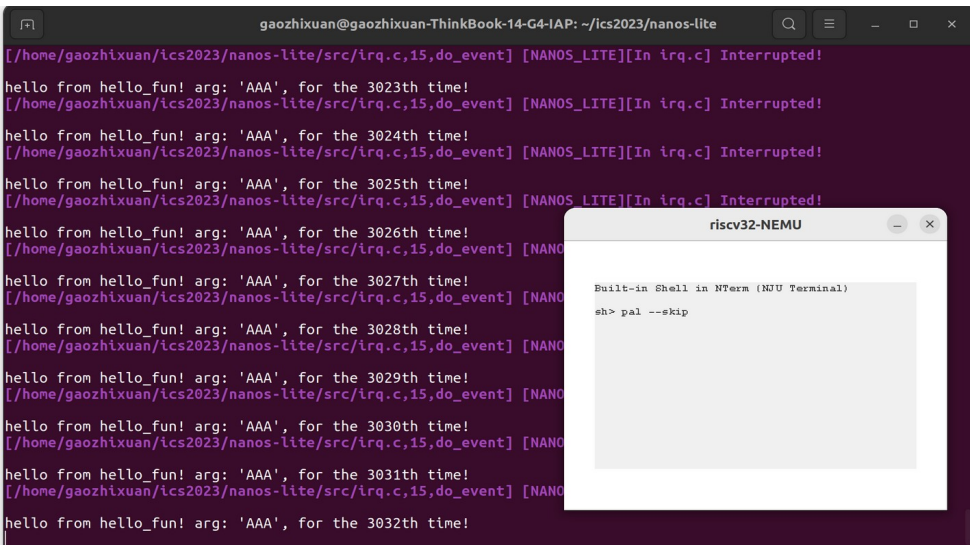
这里不清楚是由于没有搞清 setenv 和 execvp 的机制，还是 Busybox 的编译方式有问题，没有能成功使用 Busybox，而且最后并没有使用 execvp，而是使用了 execve 来进行进程的切换，而且环境是我自己人工加上的 :(

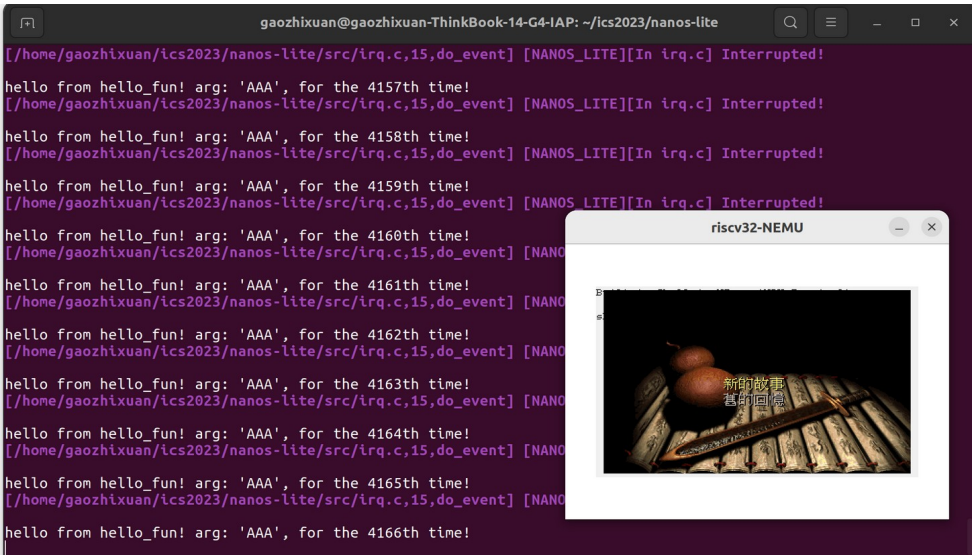
Nterm 的内置 Shell 代码如下:

```c
static void sh_handle_cmd(const char *cmd) {
  char command[128];
  strcpy(command, cmd);
  command[strlen(command) - 1] = '\0';

  const char split[2] = " ";
  char *token;
  char *argv[16];
  int argc = 0;
  char fname[64]="/bin/";
  token = strtok(command, split);
  strcat(fname,token);
  while( token != NULL ) {
    argv[argc++] = token;
    token = strtok(NULL, split);
  }
  argv[argc] = NULL;

  execve(fname, argv,NULL);
  //execvp(fname,argv);
}
```

2.实现了支持虚存管理的多道程序系统，可以在分页机制上运行 Nanos-lite，实际效果与上面一样。
3.实现了抢占式分时多任务系统，添加了时钟中断的支持。






gaozhixuan@gaozhixuan-ThinkBook-14-G4-IAP: ~/ics2023/nanos-lite
[/home/gaozhixuan/ics2023/nanos-lite/src/irq.c,15,do_event] [NANOS_LITE][In irq.c] Interrupted!
hello from hello_fun! arg: 'AAA', for the 4157th time!
[/home/gaozhixuan/ics2023/nanos-lite/src/irq.c,15,do_event] [NANOS_LITE][In irq.c] Interrupted!
hello from hello_fun! arg: 'AAA', for the 4158th time!
[/home/gaozhixuan/ics2023/nanos-lite/src/irq.c,15,do_event] [NANOS_LITE][In irq.c] Interrupted!
hello from hello_fun! arg: 'AAA', for the 4159th time!
[/home/gaozhixuan/ics2023/nanos-lite/src/irq.c,15,do_event] [NANOS_LITE][In irq.c] Interrupted!
hello from hello_fun! arg: 'AAA', for the 4160th time!
hello from hello_fun! arg: 'AAA', for the 4161th time!
hello from hello_fun! arg: 'AAA', for the 4162th time!
hello from hello_fun! arg: 'AAA', for the 4163th time!
hello from hello_fun! arg: 'AAA', for the 4164th time!
hello from hello_fun! arg: 'AAA', for the 4165th time!
hello from hello_fun! arg: 'AAA', for the 4166th time!
riscv32-NEMU
新的故事
舊的回憶