计算机系统设计 PA4

实现分页机制

问题

> 1. i386 不是一个 32 位的处理器吗,为什么表项中的基地址信息只有 20 位,而不是 32 位?
> 2. 手册上提到表项(包括 CR3)中的基地址都是物理地址,物理地址是必须的吗?能否使用虚拟地址?
> 3. 为什么不采用一级页表?或者说采用一级页表会有什么缺点?
> 4. 程序设计课上老师告诉你,当一个指针变量的值等于 NULL 时,代表空,不指向任何东西.仔细想想,真的是这样吗?当程序对空指针解引用的时候,计算机内部具体都做了些什么?你对空指针的本质有什么新的认识?

1. 在32位的x86架构中，虚拟内存管理采用分页机制，将内存划分为固定大小的页面，通常为4KB。每个页面都有一个对应的页表项，记录该页面的物理地址或其他相关信息。页表项结构通常由32位组成，包括页面基地址和其他控制信息。页表的大小为4KB（2^{12}），页与页之间没有重叠区域。由于物理地址总线只有20位，因此最多可以寻址2^{20}个物理内存页面，即1MB。

   在i386架构中，页表项中的基地址字段只使用低20位来存储物理页面的起始地址，高12位用于存储其他控制信息，例如页面权限、缓存策略等。通过基地址加上偏移量，分页机制可以将虚拟内存映射到物理内存。
2. 在手册上提到：

   > "The first paging structure used for any translation is located at the physical address in CR3."

   因此，物理地址是必须的，因为这些表项（包括CR3）的作用是将虚拟地址翻译到物理地址。如果写一个虚拟地址，无法进行翻译。每个页表项（PML4E、PDPTE、PDE、PTE）里的基址都是物理地址。然而，整个页转换表结构存放在内存中，属于虚拟地址，需要进行内存映射。
3. 一级页表存在以下缺点：
   - **内存开销**：一级页表需要为整个地址空间创建一个非常大的表。在i386架构下，地址空间大小为4GB。如果使用一级页表，将需要4GB大小的连续内存来存储页表项，这在当时的硬件条件下非常昂贵且不实际。
   - **内存访问**：一级页表会增加内存访问的复杂性。由于需要直接映射整个地址空间，每次内存访问都需要查找一级页表以获得物理地址。这不仅增加了内存访问的延迟，还使硬件设计更加复杂。

     采用二级页表可以解决这些问题：
   - **内存开销**：二级页表使用两级结构，将整个地址空间分割为更小的单元，每个单元只需要一个较小的页表来映射。这样可以显著减小页表的大小，节省内存开销。

- **内存访问**：二级页表通过两级索引进行内存访问。首先使用页目录（一级页表）查找对应的页表（二级页表），然后在页表中查找页表项并获取物理地址。这样可以减少每次内存访问时需要查找的页表项数量，降低访问延迟。

4. 当一个指针变量的值等于NULL或nullptr时，它被显式地设置为指向地址0x0，该地址的物理存储内容没有访问权限。当程序尝试解引用空指针时，计算机会发生以下情况：
   i. 试图解引用该指针以获得变量的值。
   ii. 访问地址0x0。
   iii. 内存管理单元（MMU）进行地址转换。
   iv. 在页表中找不到相应的映射或没有权限。
   v. 引发异常并进入异常处理程序。
   vi. 进程被终止。

   需要辨别空指针和野指针的区别：
   - **野指针**：指向未知地址的指针，或者指向的内存不可用的指针，通常是声明但未初始化的指针。
   - **空指针**：指向地址0的指针，即NULL。指针的值为0。空指针的主要作用是防止野指针的出现。

准备内核页表

> 注意：
> 在实验开始前，请一定先到 `ics2024/nemu/include/common.h` 中检查 `#define DEBUG` 是否被注释，若被注释，请一定要取消注释，否则你的监视点将不起作用！！！

在 `ics2024/nanos-lite/src/main.c` 中定义宏 `HAS_PTE`

```
/* Uncomment these macros to enable corresponding functionality. */
#define HAS_ASYE
#define HAS_PTE
```

进入 `ics2024/nanos-lite` 执行 `make run`，发现出现以下报错：

```
./build/nemu -l /home/sesame/ics2024/nanos-lite/build/nemu-log.txt /home/sesame/ics2024/nanos-lite/build/nanos-lite-x86-nemu.bin
[src/monitor/monitor.c,65,load_img] The image is /home/sesame/ics2024/nanos-lite/build/nanos-lite-x86-nemu.bin
Welcome to NEMU!
[src/monitor/monitor.c,30,welcome] Build time: 17:42:50, May 30 2024
For help, type "help"
(nemu) c
[src/mm.c,24,init_mm] free physical pages starting from 0x1d6f000
invalid opcode(eip = 0x0010173f): 0f 22 d8 0f 20 c0 89 45 ...

There are two cases which will trigger this unexpected exception:
1. The instruction at eip = 0x0010173f is not implemented.
2. Something is implemented incorrectly.
Find this eip(0x0010173f) in the disassembling result to distinguish which case it is.

If it is the first case, see
 _ ____   ___   __   __ __                  _
(_)___ \ / _ \ / /_  | \/ |                | |
 _  __) | (_) | '_ \ | \  / | __ _ _ __ _   _  __ _| |
| ||__ < > _ <| (_) || |\/| |/ _` | '_ \ | | |/ _` | |
| |___) | (_) | (_) || |  | | (_| | | | | |_| | (_| | |
|_|____/ \___/ \___/ |_|  |_|\__,_|_| |_|\__,_|\__,_|_|

for more details.

If it is the second case, remember:
* The machine is always right!
* Every line of untested code is always wrong!
```

说明有指令没有实现

查询 `ics2024/nanos-lite/build/nanos-lite-x86-nemu.txt` 文件发现是有关于 `cr3` 的 `mov` 指令没有实现

```
  10173f: 0f 22 d8                mov    %eax,%cr3
```

在 `ics2024/nemu/include/cpu/reg.h` 中做出如下修改，添加 CR0 和 CR3 两个控制寄存器：

```
typedef struct {
// 略
    union {vaddr_t eip;
         struct{
             uint32_t CF : 1;
             uint32_t    : 5;
             uint32_t ZF : 1;
             uint32_t SF : 1;
             uint32_t    : 1;
             uint32_t IF : 1;
             uint32_t    : 1;
             uint32_t OF : 1;
             uint32_t    : 20;
         };
         rtlreg_t value;
     } eflags;


    struct {
        uint32_t base;  // 32位base
        uint16_t limit; // 16位limit
    }idtr;
    uint16_t cs;
    uint32_t CR0;
    uint32_t CR3;
    bool INTR;
} CPU_state;
```

然后在 `nemu/src/monitor/monitor.c` 的 `resart` 函数中初始化 `CR0` 的值

```
static inline void restart() {
  /* Set the initial instruction pointer. */
  cpu.eip = ENTRY_START;
  cpu.eflags.value = 0x2;
  cpu.cs = 0x8;
  cpu.CR0 = 0x60000011;

#ifdef DIFF_TEST
  init_qemu_reg();
#endif
}
```

增加相关控制函数，在 `nemu/include/cpu/rtl.h` 中增加如下代码：

```
static inline void rtl_load_cr(rtlreg_t* dest, int r) {
    switch (r)
    {
        case 0:
            *dest = cpu.CR0;
            return;
            break;
        case 3:
            *dest = cpu.CR3;
            return;
        default:
            assert(0);
    }
    return;
}

static inline void rtl_store_cr(int r, const rtlreg_t* src) {
    switch (r)
    {
        case 0:
            cpu.CR0 = *src;
            return;
        case 3:
            cpu.CR3 = *src;
            return;
        default:
            assert(0);
    }
    return;
}
```

在 `nemu/src/cpu/decode/decode.c` 中增加如下代码：

```
make_DHelper(mov_load_cr) {
    decode_op_rm(eip, id_dest, false, id_src, false);
    rtl_load_cr(&id_src -> val, id_src -> reg);
#ifdef DEBUG
    snprintf(id_src -> str, 5, "%%cr%d", id_dest -> reg);
#endif
}
make_DHelper(mov_store_cr) {
    decode_op_rm(eip, id_src, true, id_dest, false);
#ifdef DEBUG
    snprintf(id_src -> str, 5, "%%cr%d", id_dest -> reg);
#endif
}
```

在 nemu/src/cpu/decode/decode.h 中增加如下代码：

```
make_DHelper(mov_load_cr);
make_DHelper(mov_store_cr);
```

在 nemu/src/cpu/exec/data-mov.c 中增加如下代码：

```
make_EHelper(mov_store_cr) {
    rtl_store_cr(id_dest -> reg, &id_src -> val);
    print_asm_template2(mov);
}
```

> 千万记得在 nemu/src/cpu/exec/all-instr.h 中增加定义

在 nemu/src/cpu/exec/exec.c 的 2 byte op_table 中做出如下修改：

```
  /* 0x20 */    IDEX(mov_load_cr, mov), EMPTY, IDEX(mov_store_cr, mov_store_cr),EMPTY,
```

依据上次出错位置设置监视点，设置完成后查看寄存器如下：

```
(nemu) info r
eax  0x3afdea34
ecx  0x72d91d97
edx  0x1860a5bc
ebx  0x5d8f7dc6
esp  0x2fc07508
ebp  0x7e84d0f6
esi  0x32a17d1d
edi  0x521f06c2
eip  0x100000
ax  0xea34
cx  0x1d97
dx  0xa5bc
bx  0x7dc6
sp  0x7508
bp  0xd0f6
si  0x7d1d
di  0x6c2
al  0x34
cl  0x97
dl  0xbc
bl  0xc6
ah  0xea
ch  0x1d
dh  0xa5
bh  0x7d
eflags:CF=0,ZF=0,SF=0,IF=0,OF=0
CR0=0x60000011, CR3=0x0
```

可以发现 `CR0` 和 `CR3` 已经被初始化，之后运行至断点，查看寄存器如下：

```
(nemu) info r
eax  0xe0000011
ecx  0x8000001
edx  0x1d6e000
ebx  0x20
esp  0x7b90
ebp  0x7ba8
esi  0x20
edi  0x6a9ea344
eip  0x101753
ax  0x11
cx  0x1
dx  0xe000
bx  0x20
sp  0x7b90
bp  0x7ba8
si  0x20
di  0xa344
al  0x11
cl  0x1
dl  0x0
bl  0x20
ah  0x0
ch  0x0
dh  0xe0
bh  0x0
eflags:CF=0,ZF=0,SF=1,IF=0,OF=0
CR0=0xe0000011, CR3=0x1d6e000
```

此时 CR0 与 CR3 均已经设置完毕

虚拟地址转换

在 ics2024/nemu/src/memory/memory.c 中定义下列宏和头文件：

```
#include "memory/mmu.h"

#define PTXSHFT 12 //线性地址偏移量
#define PDXSHFT 22 //线性地址偏移量

#define PTE_ADDR(pte) ((uint32_t)(pte) & ~0xfff)
#define PDX(va) (((uint32_t)(va) >> PDXSHFT) & 0x3ff)
#define PTX(va) (((uint32_t)(va) >> PTXSHFT) & 0x3ff)
#define OFF(va) ((uint32_t)(va) & 0xfff)
```

之后实现 page_translate() 函数，修改 vaddr_read() 和 vaddr_write() 函数：

```
paddr_t page_translate(vaddr_t addr, bool iswrite) {
  CR0 cr0 = (CR0)cpu.CR0;
  if(cr0.paging && cr0.protect_enable) {
    CR3 crs = (CR3)cpu.CR3;

    PDE *pgdirs = (PDE*)PTE_ADDR(crs.val);
    PDE pde = (PDE)paddr_read((uint32_t)(pgdirs + PDX(addr)), 4);

    PTE *ptable = (PTE*)PTE_ADDR(pde.val);
    PTE pte = (PTE)paddr_read((uint32_t)(ptable + PTX(addr)), 4);
    //printf("hhahah%x, jhhh%x\n", pte.present, addr);
    Assert(pte.present, "addr=0x%x", addr);

    pde.accessed=1;
    pte.accessed=1;
    if(iswrite) {
      pte.dirty=1;
    }
    paddr_t paddr = PTE_ADDR(pte.val) | OFF(addr);
    // printf("vaddr=0x%x, paddr=0x%x\n", addr, paddr);
    return paddr;
  }
        return addr;
}

uint32_t vaddr_read(vaddr_t addr, int len) {
  if(PTE_ADDR(addr) != PTE_ADDR(addr + len -1)) {
    // printf("error: the data pass two pages:addr=0x%x, len=%d!\n", addr, len);
    // assert(0);
    int num1 = 0x1000 - OFF(addr);
    int num2 = len - num1;
    paddr_t paddr1 = page_translate(addr, false);
    paddr_t paddr2 = page_translate(addr + num1, false);

    uint32_t low = paddr_read(paddr1, num1);
    uint32_t high = paddr_read(paddr2, num2);

    uint32_t result = high << (num1 * 8) | low;
    return result;
  }
  else {
    paddr_t paddr = page_translate(addr, false);
    return paddr_read(paddr, len);
  }
  // return paddr_read(addr, len);
}
```

```
void vaddr_write(vaddr_t addr, int len, uint32_t data) {
  if(PTE_ADDR(addr) != PTE_ADDR(addr+len-1)) {
    // printf("error: the data pass two pages:addr=0x%x, len=%d!\n", addr, len);
    // assert(0);
    if(PTE_ADDR(addr) != PTE_ADDR(addr + len -1)) {
      int num1 = 0x1000-OFF(addr);
      int num2 = len -num1;
      paddr_t paddr1 = page_translate(addr, true);
      paddr_t paddr2 = page_translate(addr + num1, true);

      uint32_t low = data & (~0u >> ((4 - num1) << 3));
      uint32_t high = data >> ((4 - num2) << 3);

      paddr_write(paddr1, num1, low);
      paddr_write(paddr2, num2, high);
      return;
    }
  }
  else {
    paddr_t paddr = page_translate(addr, true);
    paddr_write(paddr, len, data);
  }
  // paddr_write(addr, len, data);
}
```

在 ics2024/nanos-lite/src/main.c 中做出如下修改：

```
//   uint32_t entry = loader(NULL, "/bin/pal");
//   ((void (*)(void))entry)();
extern void load_prog(const char *filename);
    load_prog("/bin/dummy");
```

之后运行可以有如下显示：

```
(nemu) c
[src/mm.c,24,init_mm] free physical pages starting from 0x1d91000
[src/main.c,19,main] 'Hello World!' from Nanos-lite
[src/main.c,20,main] Build time: 05:48:28, Jun  1 2024
[src/ramdisk.c,26,init_ramdisk] ramdisk info: start = 0x102a30, end = 0x1d4c9ad, size = 29663101 bytes
[src/main.c,27,main] Initializing interrupt/exception handler...
[src/fs.c,31,init_fs] set FD_FB size = 480000
[src/fs.c,66,fs_open] success open:57:/bin/dummy
[src/loader.c,20,loader] filename=/bin/dummy,fd=57
nemu: HIT GOOD TRAP at eip = 0x00100032
```

跨越边界

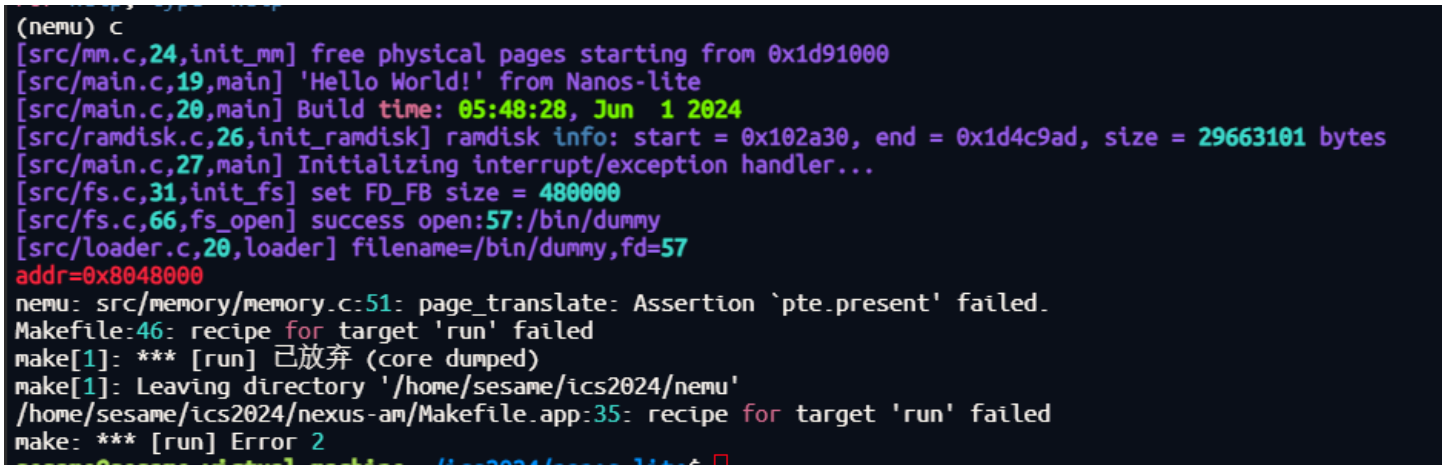
在 ics2024/navy-apps/Makefile.compile 中做出如下修改：

```
ifeq ($(LINK), dynamic)
  CFLAGS   += -fPIE
  CXXFLAGS += -fPIE
  LDFLAGS  += -fpie -shared
else
  LDFLAGS += -Ttext 0x8048000
endif
```

在 ics2024/nanos-lite/src/loader.c 中做出如下修改：

```
#define DEFAULT_ENTRY ((void *)0x8048000)
```

运行后发现出错：



在 ics2024/nexus-am/am/arch/x86-nemu/src/pte.c 中实现如下代码：

```
void _map(_Protect *p, void *va, void *pa) {
  if(OFF(va) || OFF(pa)) {
    // printf("page not aligned\n");
    return;
  }

  PDE *dir = (PDE*) p -> ptr;
        PTE *table = NULL;
        PDE *pde = dir + PDX(va);
        if(!(*pde & PTE_P)) {
                table = (PTE*) (palloc_f());
                *pde = (uintptr_t) table | PTE_P;
        }
        table = (PTE*) PTE_ADDR(*pde);
        PTE *pte = table + PTX(va);
        *pte = (uintptr_t) pa | PTE_P;
}
```

在 ics2024/nanos-lite/src/loader.c 中做出如下修改：

```c
#include "memory.h"

uintptr_t loader(_Protect *as, const char *filename) {
  // TODO();
  // ramdisk_read(DEFAULT_ENTRY, 0, RAMDISK_SIZE);
  int fd = fs_open(filename, 0, 0);
  Log("filename=%s,fd=%d",filename,fd);
  // fs_read(fd, DEFAULT_ENTRY, fs_filesz(fd));
  int size = fs_filesz(fd);
  int ppnum = size / PGSIZE;
  if(size % PGSIZE != 0) {
    ppnum++;
  }
  void *pa = NULL;
  void *va = DEFAULT_ENTRY;
  for(int i = 0; i < ppnum; i++) {
    pa = new_page();
    _map(as, va, pa);
    fs_read(fd, pa, PGSIZE);
    va += PGSIZE;
  }

  fs_close(fd);
  return (uintptr_t)DEFAULT_ENTRY;
}
```

运行 dummy 发现可以正常运行

```
[src/monitor/monitor.c,65,load_img] The image is /home/sesame/ics2024/nanos-lite/build/nanos-lite-x86-nemu.bin
Welcome to NEMU!
[src/monitor/monitor.c,30,welcome] Build time: 14:59:27, May 31 2024
For help, type "help"
(nemu) c
[src/mm.c,24,init_mm] free physical pages starting from 0x1d91000
[src/main.c,19,main] 'Hello World!' from Nanos-lite
[src/main.c,20,main] Build time: 05:48:28, Jun  1 2024
[src/ramdisk.c,26,init_ramdisk] ramdisk info: start = 0x102af0, end = 0x1d4ca6d, size = 29663101 bytes
[src/main.c,27,main] Initializing interrupt/exception handler...
[src/fs.c,31,init_fs] set FD_FB size = 480000
[src/fs.c,66,fs_open] success open:57:/bin/dummy
[src/loader.c,19,loader] filename=/bin/dummy,fd=57
nemu: HIT GOOD TRAP at eip = 0x00100032
```

接下来开始着手实现仙剑奇侠传的运行

在 ics2024/nanos-lite/src/mm.c 中做出如下修改：

```
int mm_brk(uint32_t new_brk) {
  if(current -> cur_brk == 0) {
    current -> cur_brk = current -> max_brk = new_brk;
  }
  else {
    if(new_brk > current -> max_brk) {
      uint32_t first = PGROUNDUP(current -> max_brk);
      uint32_t end = PGROUNDDOWN(new_brk);
      if((new_brk & 0xfff) == 0) {
        end -= PGSIZE;
      }
      for(uint32_t va = first; va <= end; va += PGSIZE) {
        void *pa = new_page();
        _map(&(current -> as), (void*)va, pa);
      }
      current -> max_brk = new_brk;
    }
    current -> cur_brk = new_brk;
  }
  return 0;
}
```

在 `ics2024/nanos-lite/src/syscall.c` 中做出如下修改：

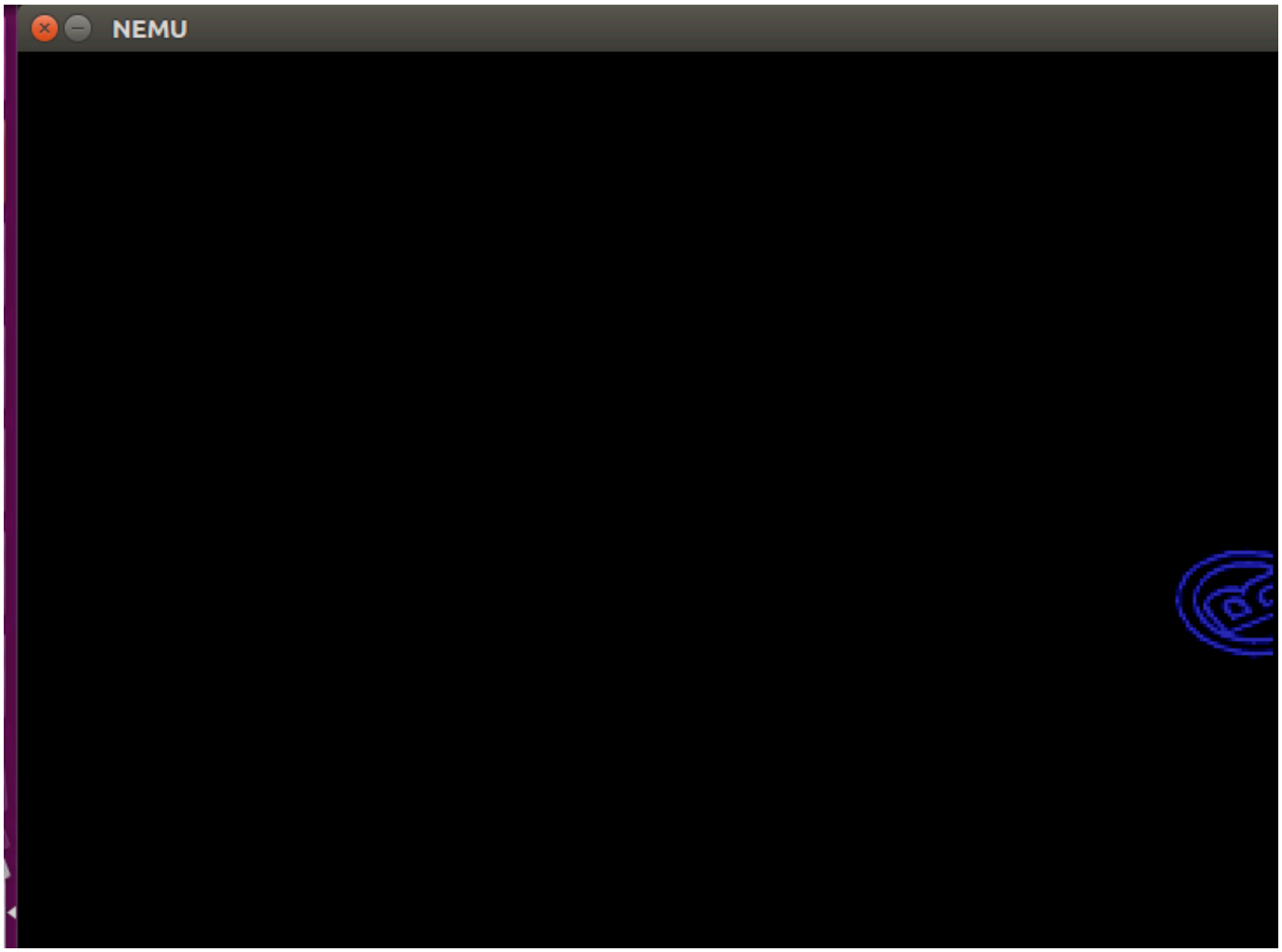
```
int sys_brk(int addr) {
  extern int mm_brk(uint32_t new_brk);
  return mm_brk(addr);
}
// 略
case SYS_brk:SYSCALL_ARG1(r) = sys_brk(a[1]);break;
```

之后再 `ics2024/nexus-am/Makefile.compile` 中做出如下修改：

```
// load_prog("/bin/dummy");
    load_prog("/bin/pal");
```

之后运行可以发现成功运行

> 加载速度太慢了，我等不及了，就只截了个开头—



阶段一完成

上下文切换与调度

内核自陷

在 `ics2024/nanos-lite/src/main.c` 中增加如下代码：

```
_trap();
```

在 `ics2024/nexus-am/am/arch/x86-nemu/src/asye.c` 中做出如下修改：

```
void vecsys();
void vecnull();
void vecself();
void vectime();

_RegSet* irq_handle(_RegSet *tf) {
  _RegSet *next = tf;
  if (H) {
    _Event ev;
    switch (tf->irq) {
      case 0x80:
        ev.event = _EVENT_SYSCALL;
        break;
      case 0x81:
        ev.event = _EVENT_TRAP;
        break;
      default:
        ev.event = _EVENT_ERROR;
        break;
    }

    next = H(ev, tf);
    if (next == NULL) {
      next = tf;
    }
  }

  return next;
}

void _asye_init(_RegSet*(*h)(_Event, _RegSet*)) {
  // initialize IDT
  for (unsigned int i = 0; i < NR_IRQ; i ++) {
    idt[i] = GATE(STS_TG32, KSEL(SEG_KCODE), vecnull, DPL_KERN);
  }

  // -------------------- system call --------------------------
  idt[0x80] = GATE(STS_TG32, KSEL(SEG_KCODE), vecsys, DPL_USER);
  idt[0x81] = GATE(STS_TG32, KSEL(SEG_KCODE), vecself, DPL_USER);
  idt[32] = GATE(STS_TG32, KSEL(SEG_KCODE), vectime, DPL_USER);


  set_idt(idt, sizeof(idt));

  // register event handler
  H = h;
```

```
}


void _trap() {
  asm volatile("int $0x81");
}
```

在 ics2024/nanos-lite/src/proc.c 中做出如下修改：

```
//   // TODO: remove the following three lines after you have implemented _umake()
//   _switch(&pcb[i].as);
//   current = &pcb[i];
//   ((void (*)(void))entry)();
```

在 ics2024/nanos-lite/src/irq.c 中做出如下修改：

```
static _RegSet* do_event(_Event e, _RegSet* r) {
  switch (e.event) {
    case _EVENT_SYSCALL:
      return do_syscall(r);
    case _EVENT_TRAP:
      printf("event:self-trapped\n");
      return NULL;
    default:
      panic("Unhandled event ID = %d", e.event);
  }

  return NULL;
}
```

在 ics2024/nexus-am/am/arch/x86-nemu/src/trap.S 中做出如下修改：

```
.globl vecsys;    vecsys:  pushl $0;  pushl $0x80; jmp asm_trap
.globl vecnull;  vecnull:  pushl $0;  pushl   $-1; jmp asm_trap
.globl vecself;  vecself:  pushl $0;  pushl $0x81; jmp asm_trap
.globl vectime; vectime: pushl $0; pushl $32; jmp asm_trap
```

运行后得到如下结果：

```
[src/monitor/monitor.c,30,welcome] Build time: 14:59:27, May 31 2024
For help, type "help"
(nemu) c
[src/mm.c,42,init_mm] free physical pages starting from 0x1d91000
[src/main.c,19,main] 'Hello World!' from Nanos-lite
[src/main.c,20,main] Build time: 08:25:08, Jun  1 2024
[src/ramdisk.c,26,init_ramdisk] ramdisk info: start = 0x102bf0, end = 0x1d4cb6d, size = 29663101 bytes
[src/main.c,27,main] Initializing interrupt/exception handler...
[src/fs.c,31,init_fs] set FD_FB size = 480000
[src/fs.c,66,fs_open] success open:55:/bin/pal
[src/loader.c,19,loader] filename=/bin/pal,fd=55
event:self-trapped
[src/main.c,41,main] system panic: Should not reach here
nemu: HIT BAD TRAP at eip = 0x00100032
```

上下文切换

在 `ics2024/nexus-am/am/arch/x86-nemu/src/pte.c` 中实现如下代码：

```
_RegSet *_umake(_Protect *p, _Area ustack, _Area kstack, void *entry, char *const argv[], char
  extern void* memcpy(void *, const void *, int);
  int arg1 = 0;
  char *arg2 = NULL;
  memcpy((void*)ustack.end - 4, (void*)arg2, 4);
  memcpy((void*)ustack.end - 8, (void*)arg2, 4);
  memcpy((void*)ustack.end - 12, (void*)arg1, 4);
  memcpy((void*)ustack.end - 16, (void*)arg1, 4);

  _RegSet tf;
  tf.eflags = 0x02 | FL_IF;
  tf.cs = 0;
  tf.eip = (uintptr_t) entry;
  void *ptf = (void*) (ustack.end - 16 - sizeof(_RegSet));
  memcpy(ptf, (void*)&tf, sizeof(_RegSet));
  return (_RegSet*) ptf;
}
```

在 `ics2024/nanos-lite/src/irq.c` 中实现如下代码：

```
_RegSet* schedule(_RegSet *prev) {
    if(current != NULL) {
        current -> tf = prev;
    }
    current = pcb[0];
    Log("ptr = 0x%x\n", (uint32_t)current -> as.ptr);
    _switch(&current -> as);
    return current -> tf;
}
```

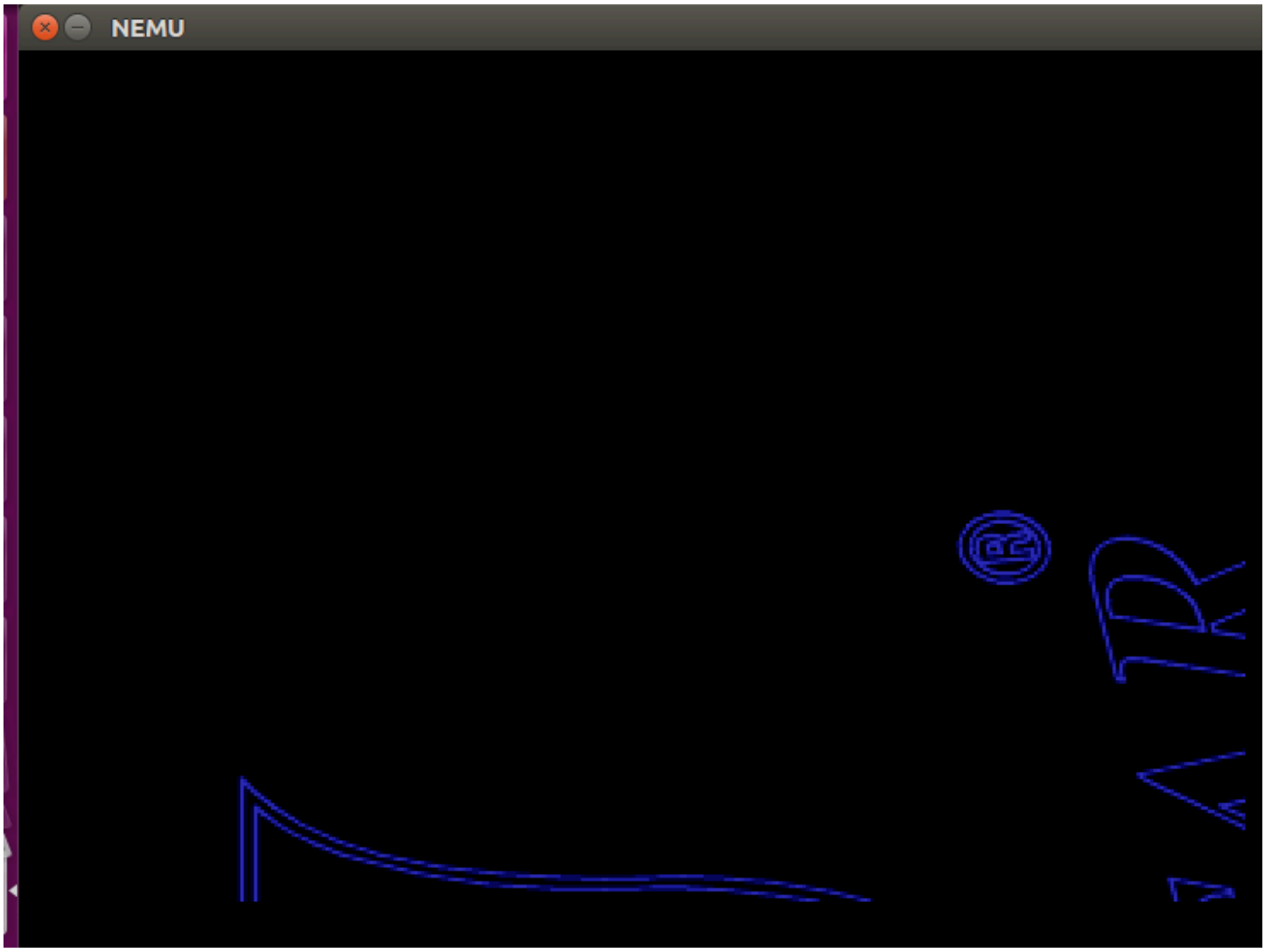
在 `ics2024/nanos-lite/src/irq.c` 中做出如下修改：

```
extern _RegSet* schedule(_RegSet *prev);
static _RegSet* do_event(_Event e, _RegSet* r) {
  switch (e.event) {
    case _EVENT_SYSCALL:
      return do_syscall(r);
    case _EVENT_TRAP:
      printf("event:self-trapped\n");
      return schedule(r);
    default:
      panic("Unhandled event ID = %d", e.event);
  }

  return NULL;
}
```

在 `ics2024/nexus-am/am/arch/x86-nemu/src/trap.S` 中做出如下修改：

```
asm_trap:
  pushal

  pushl %esp
  call irq_handle

  # addl $4, %esp
  movl %eax, %esp

  popal
  addl $8, %esp

  iret
```

结果如下：



分时多任务

在 `ics2024/nanos-lite/src/main.c` 中做出如下修改:

```
//   uint32_t entry = loader(NULL, "/bin/pal");
//   ((void (*)(void))entry)();
extern void load_prog(const char *filename);
    // load_prog("/bin/dummy");
    load_prog("/bin/pal");
    load_prog("/bin/hello");
```

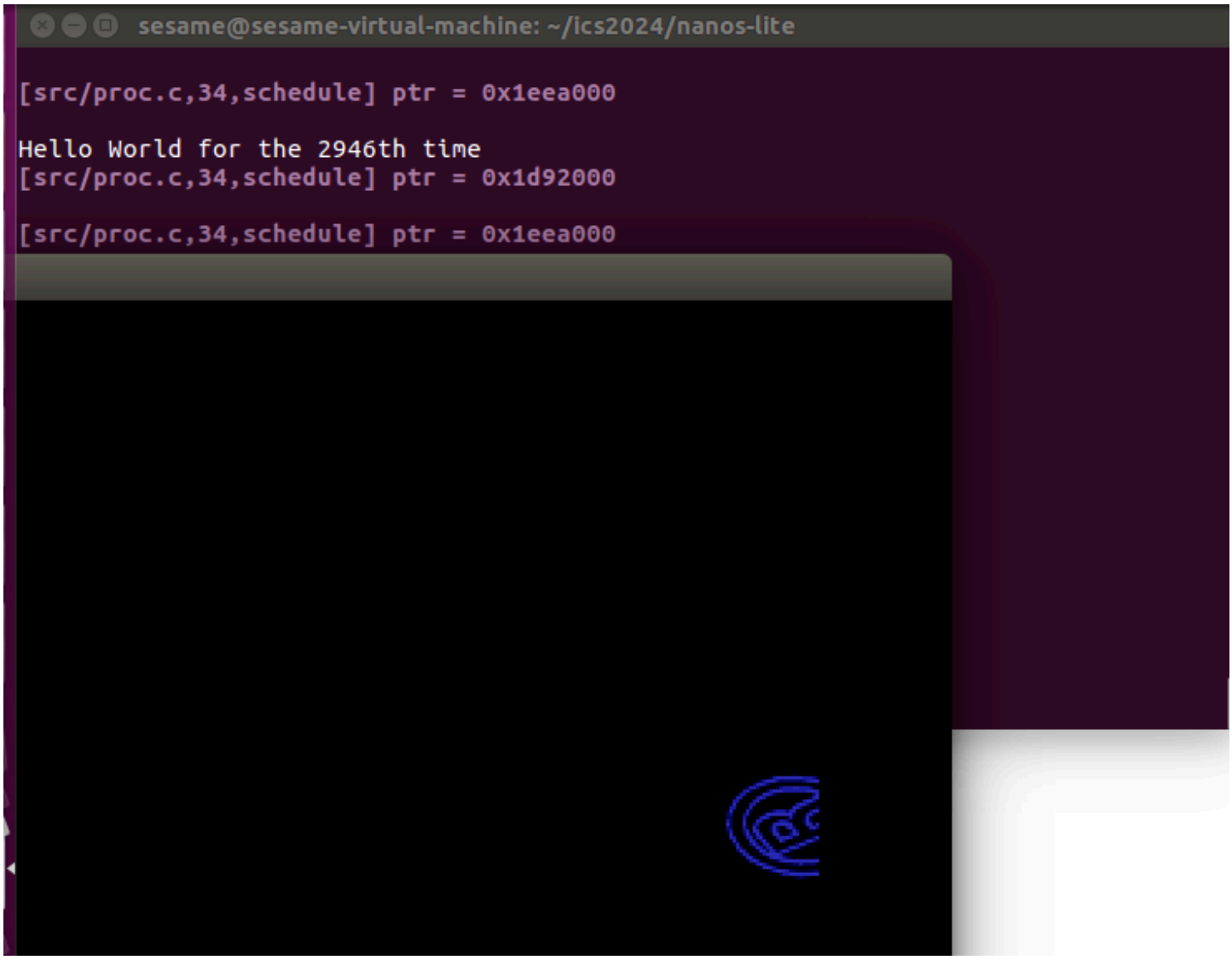
在 `ics2024/nanos-lite/src/proc.c` 中做出如下修改:

```
_RegSet* schedule(_RegSet *prev) {
    if(current != NULL) {
        current -> tf = prev;
    }
    current =  (current == &pcb[0]? &pcb[1] : &pcb[0]);
    Log("ptr = 0x%x\n", (uint32_t)current -> as.ptr);
    _switch(&current -> as);
    return current -> tf;
}
```

在 ics2024/nanos-lite/src/irq.c 中做出如下修改：

```
static _RegSet* do_event(_Event e, _RegSet* r) {
  switch (e.event) {
    case _EVENT_SYSCALL:
      // return do_syscall(r);
        do_syscall(r);
        return schedule(r);
    case _EVENT_TRAP:
      printf("event:self-trapped\n");
      return schedule(r);
    default:
      panic("Unhandled event ID = %d", e.event);
  }

  return NULL;
}
```

运行结果如下：



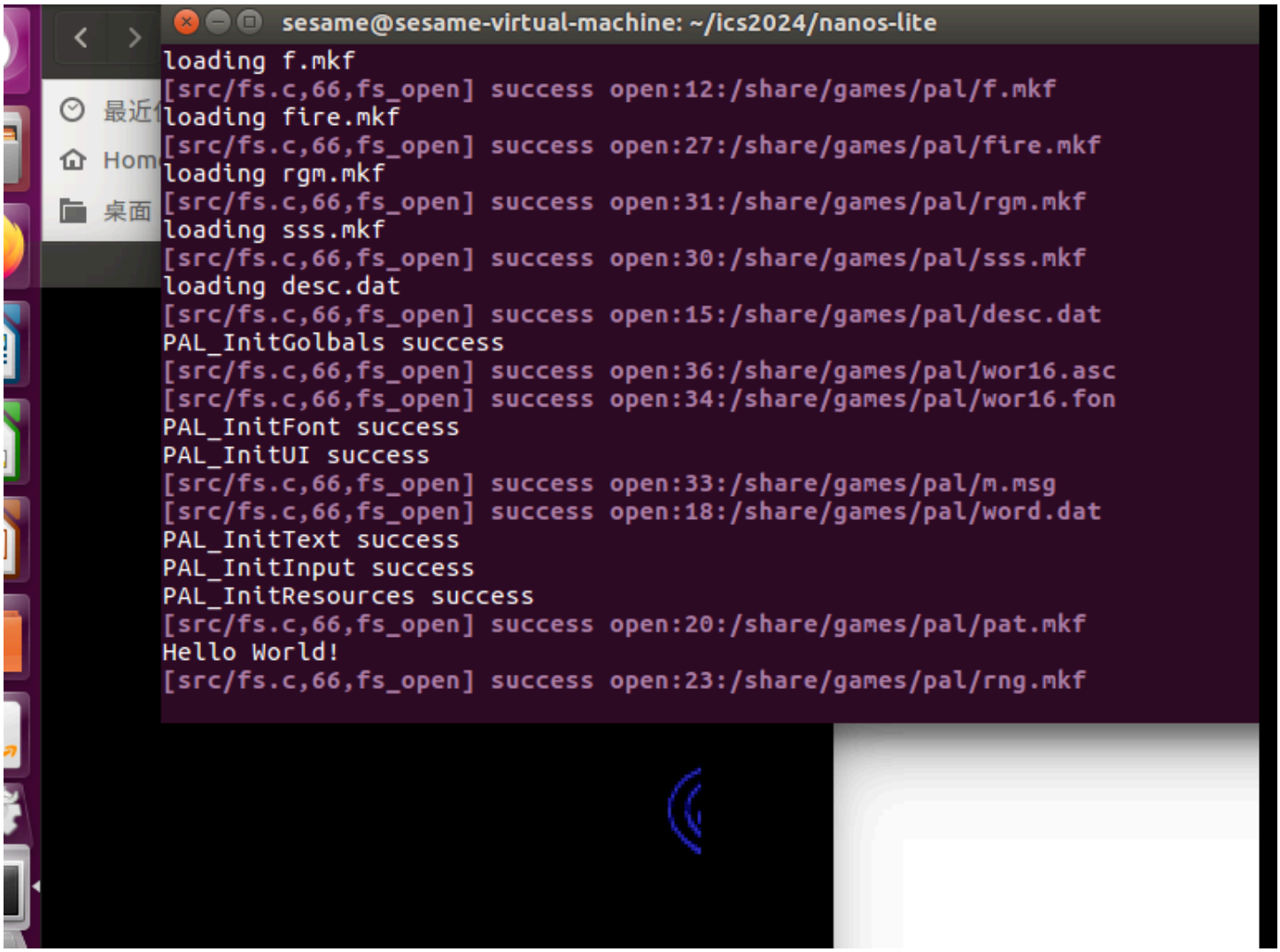
> 速度实在是太慢了，原因分析一下可知是分给仙剑奇侠传的时间少。从图中可知将近3000次输出后才勉强动了一点

优先级调度

为了使仙剑奇侠传运行更快，在 `ics2024/nanos-lite/src/proc.c` 中进行如下修改：

```
int count = 0;
_RegSet* schedule(_RegSet *prev) {
  current->tf = prev;
  current = &pcb[0];
  count++;
  if (count == 100) {
    current = (current == &pcb[0] ? &pcb[1] : &pcb[0]);
    count = 0;
  }
  _switch(&current->as);
  return current->tf;
}
```

运行后速度大大提升



时钟中断

在 `ics2024/nemu/src/cpu/intr.c` 中做出如下修改：

```
void dev_raise_intr() {
    cpu.INTR = true;
}
```

依据实验指导手册修改 `ics2024/nemu/src/cpu/exec/exec.c`

```
#define TIME_IRQ 32
  if(cpu.INTR & cpu.eflags.IF) {
    cpu.INTR = false;
    extern void raise_intr(uint8_t NO, vaddr_t ret_addr);
    raise_intr(TIME_IRQ, cpu.eip);
    update_eip();
  }
```

在 ics2024/nemu/src/cpu/intr.c 中做出如下修改：

```
void raise_intr(uint8_t NO, vaddr_t ret_addr) {
  /* TODO: Trigger an interrupt/exception with ``NO''.
   * That is, use ``NO'' to index the IDT.
   */
  //获取门描述符
  vaddr_t gate_addr = cpu.idtr.base + 8 * NO;

  //P位校验
  if (cpu.idtr.limit < 0) assert(0);

  //将eflags、cs、返回地址压栈
  t0 = cpu.cs; //cpu.cs 只有16位，需要转换成32位
  rtl_push(&cpu.eflags.value);
  rtl_push(&t0);
  cpu.eflags.IF = 0;
  rtl_push(&ret_addr);

  //组合中断处理程序入口点
  uint32_t high, low;
  low = vaddr_read(gate_addr, 4) & 0xffff;
  high = vaddr_read(gate_addr + 4, 4) & 0xffff0000;

  //设置eip跳转
  decoding.jmp_eip = high | low;
  decoding.is_jmp = true;
}
```

在 ics2024/nexus-am/am/arch/x86-nemu/src/asye.c 中的irq_handle中增加如下代码：

```
      case 32:
        ev.event = _EVENT_IRQ_TIME;
        break;
```

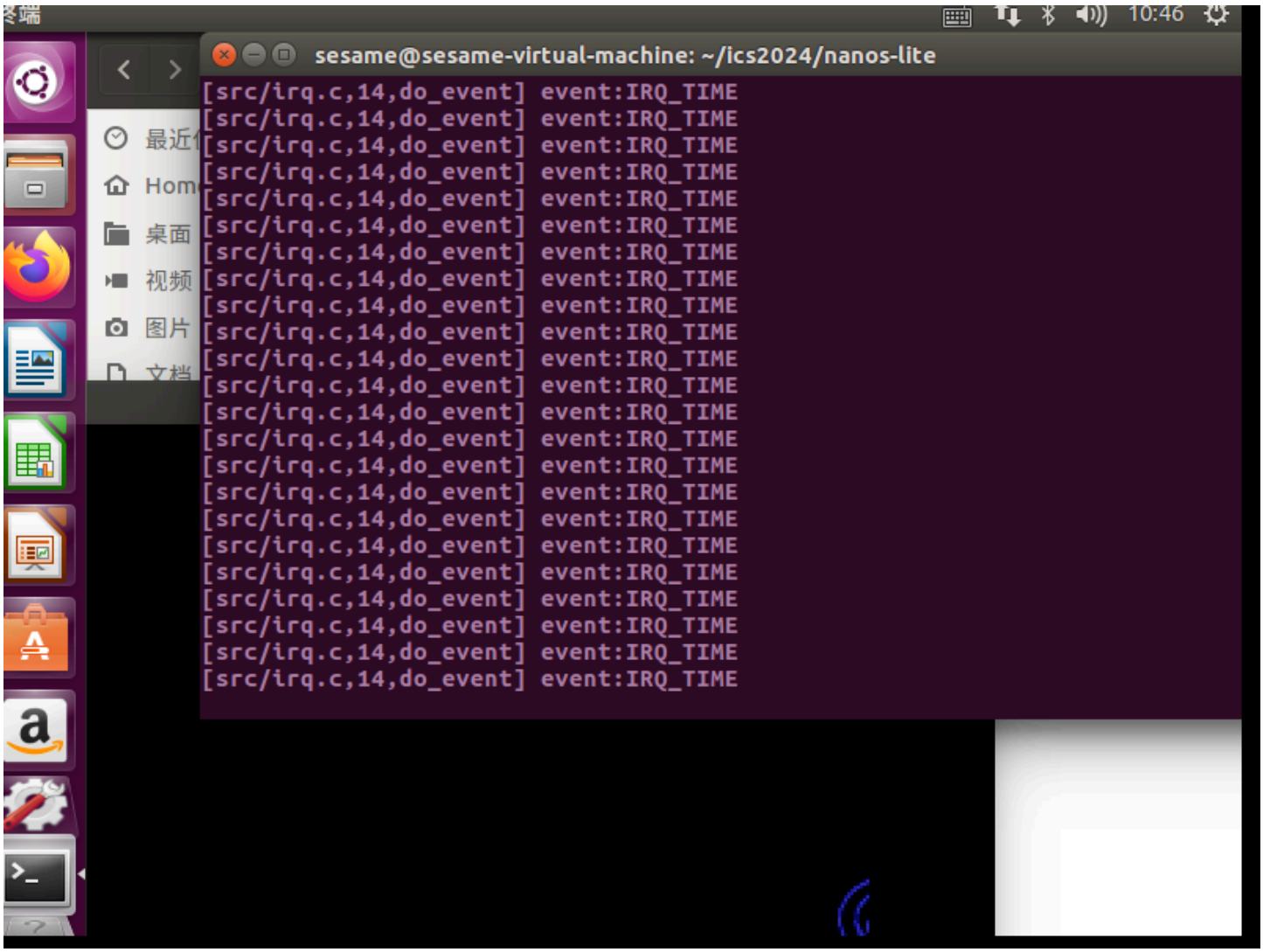
在 ics2024/nanos-lite/src/irq.c 中的do_event中增加如下代码：

```
case _EVENT_IRQ_TIME:
      Log("event:IRQ_TIME");
      return schedule(r);
```

在 ics2024/nexus-am/am/arch/x86-nemu/src/pte.c 中修改如下代码：

```
_RegSet *_umake(_Protect *p, _Area ustack, _Area kstack, void *entry, char *const argv[], char
  extern void* memcpy(void *, const void *, int);
  int arg1 = 0;
  char *arg2 = NULL;
  memcpy((void*)ustack.end - 4, (void*)arg2, 4);
  memcpy((void*)ustack.end - 8, (void*)arg2, 4);
  memcpy((void*)ustack.end - 12, (void*)arg1, 4);
  memcpy((void*)ustack.end - 16, (void*)arg1, 4);

  _RegSet tf;
  tf.eflags = 0x02 | FL_IF;
  tf.cs = 0;
  tf.eip = (uintptr_t) entry;
  void *ptf = (void*) (ustack.end - 16 - sizeof(_RegSet));
  memcpy(ptf, (void*)&tf, sizeof(_RegSet));
  return (_RegSet*) ptf;
}
```

实验结果如下



必答题

> 请结合代码，解释分页机制和硬件中断是如何支持《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。

首先，Nanos-lite 的入口函数是 `main.c` 。系统的第一步是进行内存管理（MM）初始化，具体操作是将 TRM 提供的堆区起始地址作为空闲物理页的首地址，随后通过 `new_page()` 函数来分配空闲的物理页。接着是磁盘和设备的初始化、中断/异常初始化以及文件系统初始化，如下所示：

```
init_ramdisk();
init_device();
```

然后系统加载需要运行的程序，执行如下：

```
init_fs();
extern void load_prog(const char *filename);
load_prog("/bin/pal");
load_prog("/bin/hello");
load_prog("/bin/videotest");
_trap();
```

在加载每个进程时，系统将每个进程的上下文保存在程序控制块（PCB）中。具体操作是通过 AM 层的 `_protect` 函数初始化一个虚拟地址空间并进行映射，其信息存放在 PCB 的 `as` 字段中。然后，通过 Nanos-lite 层的 `loader` 函数以页为单位加载用户程序，接着通过 AM 层的 `_umake` 函数创建用户进程的上下文。这种映射关系使得所有进程存储在不同的物理空间中，从而解决了虚拟地址重叠的问题。

接下来，操作系统会陷入一个自陷，即进入 0x81 号中断入口，其定义在 `nexus-am/am/arch/x86-nemu/src/asye.c` 文件中的 `_asye_init` 函数中，初始化中断描述符表（IDT）的代码如下：

```
void _asye_init(_RegSet*(*h)(_Event, _RegSet*)) {
    // initialize IDT
    for (unsigned int i = 0; i < NR_IRQ; i++) {
        idt[i] = GATE(STS_TG32, KSEL(SEG_KCODE), vecnull, DPL_KERN);
    }
    ...
}
```

通过这些步骤，分页机制和硬件中断得以实现，支持《仙剑奇侠传》和 `hello` 程序在计算机系统中分时运行.

一些bug

一些bug，权当总结。

1. **每次运行和手册上的展示不一样**
   make clean
2. **执行虚拟内存和实际内存打印时候的出现的 bug**
   正如前文所述，不要忘了在all-intr中定义