§4-2.3.1 完成串口的模拟

1. 在 include/config.h 中定义宏 HAS_DEVICE_SERIAL 并 make clean;
2. 实现 in 和 out 指令;

```
make_instr_func(in_b){
        OPERAND ra, rd;
        ra.data_size = 8;
        ra.type = OPR_REG;
        ra.addr = 0x0;
        rd.data_size = 16;
        rd.type = OPR_REG;
        rd.addr = 0x2;
        operand_read(&rd);
        ra.val = pio_read(rd.val, 1);
        operand_write(&ra);
        return 1;
}
make_instr_func(in_v){
        OPERAND ra, rd;
        ra.data_size = data_size;
        ra.type = OPR_REG;
        ra.addr = 0x0;
        rd.data_size = 16;
        rd.type = OPR_REG;
        rd.addr = 0x2;
        operand_read(&rd);
        ra.val = pio_read(rd.val, data_size / 8);
        operand_write(&ra);
        return 1;
}
```

```
make_instr_func(out_b){
        OPERAND ra,rd;
        ra.data_size = 8;
        ra.type = OPR_REG;
        ra.addr = 0x0;
        rd.data_size = 16;
        rd.type = OPR_REG;
        rd.addr = 0x2;
        operand_read(&ra);
        operand_read(&rd);
        pio_write(rd.val, 1, ra.val);
        return 1;
}
make_instr_func(out_v){
        OPERAND ra,rd;
        ra.data_size = data_size;
        ra.type = OPR_REG;
        ra.addr = 0x0;
        rd.data_size = 16;
        rd.type = OPR_REG;
        rd.addr = 0x2;
        operand_read(&ra);
        operand_read(&rd);
        pio_write(rd.val, data_size / 8, ra.val);
        return 1;
}
```

3. 实现 serial_printc()函数;

```
void serial_printc(char ch)
{
        while (!serial_idle())
                ; // wait untile serial is idle
        // print 'ch' via out instruction here

        out_byte(SERIAL_PORT,(uint8_t)ch);
}
~
```

4. 运行 hello-inline 测试用例，对比实现串口前后的输出内容的区别。

```
./nemu/nemu --autorun --testcase hello-inline --kernel
NEMU load and execute img: ./kernel/kernel.img  elf: ./testcase/bin/hello-inline
[src/main.c,82,init_cond] {kernel} Hello, NEMU world!
[src/elf/elf.c,27,loader] {kernel} ELF loading from hard disk.
Hello, world!
nemu: HIT GOOD TRAP at eip = 0x08049023
```

实现串口后没有了红色的 nemu trap output

§4-2.3.2 通过硬盘加载程序

1. 在 include/config.h 中定义宏 HAS_DEVICE_IDE 并 make clean;
2. 修改 Kernel 中的 loader()，使其通过 ide_read()和 ide_write()接口实现从模拟硬盘加载用户程序;

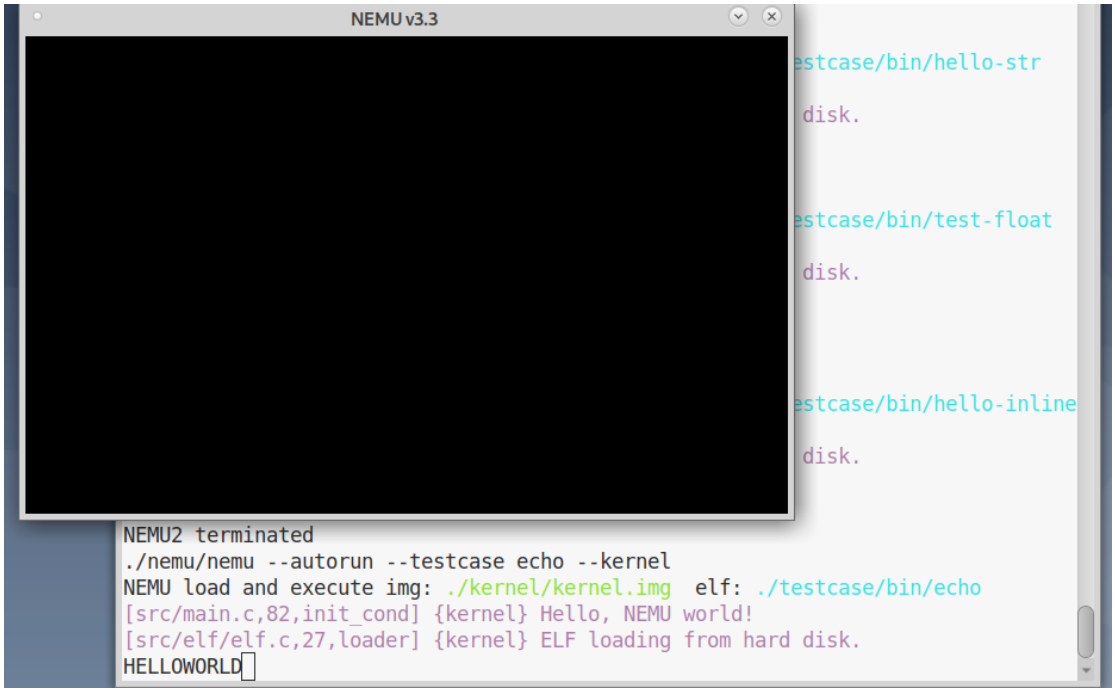
```
#ifndef HAS_DEVICE_IDE
 /* TODO: copy the segment from the ELF file to its proper memory area */
        memcpy((void*)addr, (void*)ph->p_offset, ph->p_filesz);
#else
        ide_read((void*)addr, ph->p_offset ,ph->p_filesz);
#endif
```

3. 通过 make test_pa-4-2 执行测试用例，验证加载过程是否正确。

```
make-[1]: Leaving directory '/home/pa221220074/pa_nju'
./nemu/nemu --autorun --testcase hello-inline --kernel
NEMU load and execute img: ./kernel/kernel.img  elf: ./testcase/bin/hello-inline
[src/main.c,82,init_cond] {kernel} Hello, NEMU world!
[src/elf/elf.c,27,loader] {kernel} ELF loading from hard disk.
Hello, world!
nemu: HIT GOOD TRAP at eip = 0x08049023
NEMU2 terminated
./nemu/nemu --autorun --testcase echo --kernel
NEMU load and execute img: ./kernel/kernel.img  elf: ./testcase/bin/echo
[src/main.c,82,init_cond] {kernel} Hello, NEMU world!
[src/elf/elf.c,27,loader] {kernel} ELF loading from hard disk.
```

§4-2.3.3 完成键盘的模拟

1. 在 include/config.h 中定义宏 HAS_DEVICE_KEYBOARD 并 make clean;
2. 通过 make test_pa-4-2 运行 echo 测试用例；（可以通过关闭窗口或在控制台 Ctrl-c 的方式退出 echo）



§4-2.3.4 实现 VGA 的 MMIO

1. 在 include/config.h 中定义宏 HAS_DEVICE_VGA;
2. 在 nemu/src/memory/memory.c 中添加 mm_io 判断和对应的读写操作;

```
uint32_t paddr_read(paddr_t paddr, size_t len)
{
    uint32_t ret = 0;

#ifdef HAS_DEVICE_VGA
    int num = is_mmio(paddr);
    if(num != -1){
        return mmio_read(paddr, len, num);
    }
#endif
```

```
void paddr_write(paddr_t paddr, size_t len, uint32_t data)
{

#ifdef HAS_DEVICE_VGA
    int num = is_mmio(paddr);
    if(num != -1){
        mmio_write(paddr, len, data, num);
        return;
    }
#endif
```

3. 在 kernel/src/memory/vmem.c 中完成显存的恒等映射;

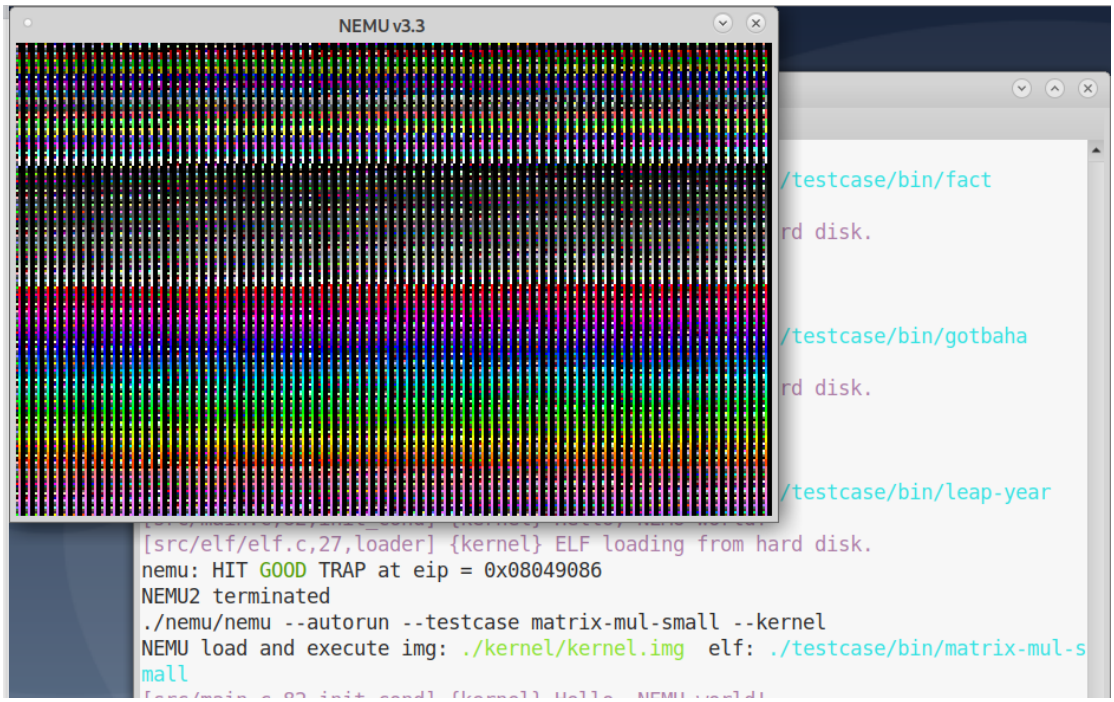
```
void create_video_mapping()
{

        /* TODO: create an identical mapping from virtual memory area
         * [0xa0000, 0xa0000 + SCR_SIZE) to physical memeory area
         * [0xa0000, 0xa0000 + SCR_SIZE) for user program. You may define
         * some page tables to create this mapping.
         */

        PDE *pdir = (PDE *)va_to_pa(get_updir());
        PTE *ptable = (PTE *)va_to_pa(table);
        uint32_t pdir_idx, ptable_idx, pframe_idx;
        pdir_idx = 0x0;
        pdir[pdir_idx].val = make_pde(ptable);
        ptable += 0x0;
        pframe_idx = 0x0;
        for (ptable_idx = 0x0; ptable_idx < 1024; ptable_idx++){
            ptable->val = make_pte(pframe_idx << 12);
                pframe_idx++;
                ptable++;
        }

}
```

4. 通过 make test_pa-4-2 执行测试用例，观察输出测试颜色信息，并通过 video_mapping_read_test()。



*** 实验报告要求**

针对 echo 测试用例，在实验报告中，结合代码详细描述：

1. 注册监听键盘事件是怎么完成的？

   在 echo 启动后，它会向 Kernel 注册监听键盘事件，并在每一个键盘事件到来后，判断是否进行输出。

   在 echo.c 中：

   先 add_irq_handler()，执行 int 0x80，系统陷入内核态，调用 do_syscall()，接着调用 add_irq_handle()函数，将 IRQ_t 类型存入 handle 数组中，完成注册。

2. 从键盘按下一个键到控制台输出对应的字符，系统的执行过程是什么？如果涉及与之前报告重复的内容，简单引用之前的内容即可。

   键盘事件首先在 nemu/src/device/sdl.c 中由 NEMU_SDL_Thread()线程捕获。当检测到相应事件后，将对应键的扫描码作为参数传送给 keyboard.c 中的模拟键盘函数。模拟键盘缓存扫描码，并通过中断请求的方式通知 CPU 有按键或抬起的事件，键盘的中断请求号为 1。CPU 收到中断请求后调用 Kernel 的中断响应程序。在响应程序中，Kernel 会查找是否有应用程序注册了对键盘事件的响应，若有，则通过调用注册的响应函数的方式来通知应用程序。此时在应用程序的键盘响应函数中，可以通过 in 指令从键盘的数据端口读取扫描码完成数据交换。