练习1

在 `kernel/object/cap_group.c` 中完善 `sys_create_cap_group`、`create_root_cap_group` 函数。

1. sys_create_cap_group函数中，需要创建新的cap_group并对其初始化，同时创建vmspace并对其初始化
2. 在create_root_cap_group函数用来创建第一个进程，同样包括创建新的cap_group并对其初始化，同时创建vmspace并对其初始化，最后将两者记录在cap_table中

练习2

在 `kernel/object/thread.c` 中完成 `create_root_thread` 函数，将用户程序 ELF 加载到刚刚创建的进程地址空间中

为了把elf的内容放入，需要分配物理内存，映射到vmspace中去，包括线程上下文等信息

练习3

完成虚拟地址空间和代码环境加载后，还需要初始化通用寄存器的值和特殊寄存器的值，特殊寄存器包括SPSR，eret指令返回的特权级等等。

思考题4

1. 通过create_root_thread,create_root_cap_group等完成进程创建并进行初始化后，通过sched调用初始化后的第一个进程
2. 通过switch_context初始化thread的上下文
3. 调用eret_to_thread,填写异常向量表
4. 根据异常处理函数exception_exit，由系统调用进入EL0，eret返回用户态

练习5

填写异常向量表和函数跳转操作

```
exception_entry sync_el1t
exception_entry irq_el1t
exception_entry fiq_el1t
exception_entry error_el1t

exception_entry sync_el1h
exception_entry irq_el1h
exception_entry fiq_el1h
exception_entry error_el1h

exception_entry sync_el0_64
exception_entry irq_el0_64
exception_entry fiq_el0_64
exception_entry error_el0_64

exception_entry sync_el0_32
exception_entry irq_el0_32
exception_entry fiq_el0_32
```

```
exception_entry error_el0_32
```

练习6

填写 `kernel/arch/aarch64/irq/irq_entry.S` 中的 `exception_enter` 与 `exception_exit`，实现上下文保存的功能，以及 `switch_to_cpu_stack` 内核栈切换函数。

exception_enter主要如下

```
.macro  exception_enter

    /* LAB 3 TODO BEGIN */

    sub sp, sp, #ARCH_EXEC_CONT_SIZE
    stp x0, x1, [sp, #16 * 0]
    stp x2, x3, [sp, #16 * 1]
    stp x4, x5, [sp, #16 * 2]
    stp x6, x7, [sp, #16 * 3]
    stp x8, x9, [sp, #16 * 4]
    stp x10, x11, [sp, #16 * 5]
    stp x12, x13, [sp, #16 * 6]
    stp x14, x15, [sp, #16 * 7]
    stp x16, x17, [sp, #16 * 8]
    stp x18, x19, [sp, #16 * 9]
    stp x20, x21, [sp, #16 * 10]
    stp x22, x23, [sp, #16 * 11]
    stp x24, x25, [sp, #16 * 12]
    stp x26, x27, [sp, #16 * 13]
    stp x28, x29, [sp, #16 * 14]

    /* LAB 3 TODO END */

    mrs x21, sp_el0
    mrs x22, elr_el1
    mrs x23, spsr_el1

    /* LAB 3 TODO BEGIN */

    stp x30, x21, [sp, #16 * 15]
    stp x22, x23, [sp, #16 * 16]

    /* LAB 3 TODO END */

.endm
```

而exception_exit大体结构与其类似，但两者是相反操作所以使用ldp

```
.macro switch_to_cpu_stack
    mrs     x24, TPIDR_EL1
    /* LAB 3 TODO BEGIN */
    add x24, x24, #OFFSET_LOCAL_CPU_STACK
    /* LAB 3 TODO END */
    ldr x24, [x24]
    mov sp, x24
.endm
```

思考7

描述 `printf` 如何调用到 `chcore_stdout_write` 函数

1. printf调用vprintf，再调用vfprintf
2. vfprintf接受stdout，将f->write这个写操作定义为__stdio_write
3. 通过系统调用 `syscall(SYS_writev, f->fd, iov, iovcnt)`，会最后调用到write(),即chcore_stdout_write

练习8

在其中添加一行以完成系统调用，目标调用函数为内核中的 `sys_putstr`。使用 `chcore_syscallx` 函数进行系统调用。

在user/chcore-libc/libchcore/porting/overrides/src/chcore-port/stdio.c中加入 `chcore_syscall2(CHCORE_SYS_putstr,(vaddr_t)buffer,size);` 完成系统调用

练习9

通过chcore的编译器编译二进制文件获得bin文件，可以加载到内核中运行