Lab 3 实验报告

潘文峥（学号：520030910232）

思考题1：

（配置页表并开启MMU以后）首先初始化异常向量表，即调用 `set_exception_vector()` ，然后创建第一个线程，调用 `create_root_thread()` ，在其中调用 `create_root_cap_group()` 函数创建cap_group：先通过 `kmalloc` 分配空间，再通过 `alloc_slot_id` 获取slot_id，在初始化vmspace；接着通过调用 `load_binary` 将ELF用户程序加载到地址空间中，完成线程创建。

练习题2：

`cap_group_init()` 依次初始化cap_group结构体的各字段：

```
cap_group->pid = pid;
cap_group->thread_cnt = 0;
slot_table_init(slot_table, size);
init_list_head(&cap_group->thread_list);
```

`sys_create_cap_group()` 依次通过 `object_alloc` 分配cap_group对象、调用 `cap_group_init` 初始化cap_group、再通过 `object_alloc` 分配vmspace对象并初始化。

`create_root_cap_group()` 创建第一个用户进程，首先分配cap_group对象并分配slot_id，分配并初始化vmspace，将root process的PCID设为0。

练习题3：

`load_binary()` 函数将用户程序ELF加载到刚刚创建的地址空间中，首先确定虚拟地址的起始和终止位置，再分配并初始化pmo，最后通过Lab 2实现的 `vmspace_map_range()` 函数进行物理页映射。

练习题4：

填写异常向量表：

```
        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
```

并按异常类型添加跳转到`handle_entry_c`和`unexpected_handler`.

练习题5：

在pagefault.c中将缺页异常转发给处理函数：

```
ret = handle_trans_fault(current_thread->vmspace, fault_addr);
```

练习题6：

完成缺页异常处理函数`handle_trans_fault()`，对`PMO_ANONYM`和`PMO_SHM`的情况，为相应的物理页添加页表映射。

练习题7：

实现系统调用前保存上下文功能，保存各寄存器状态：

```
.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

.macro  exception_exit
    /* LAB 3 TODO BEGIN */
    ldp x22, x23, [sp, #16 * 16]
    ldp x30, x21, [sp, #16 * 15]

    /* LAB 3 TODO END */
    msr sp_el0, x21
    msr elr_el1, x22
    msr spsr_el1, x23
    /* LAB 3 TODO BEGIN */
    ldp x0, x1, [sp, #16 * 0]
    ldp x2, x3, [sp, #16 * 1]
    ldp x4, x5, [sp, #16 * 2]
    ldp x6, x7, [sp, #16 * 3]
    ldp x8, x9, [sp, #16 * 4]
    ldp x10, x11, [sp, #16 * 5]
    ldp x12, x13, [sp, #16 * 6]
    ldp x14, x15, [sp, #16 * 7]
    ldp x16, x17, [sp, #16 * 8]
    ldp x18, x19, [sp, #16 * 9]
    ldp x20, x21, [sp, #16 * 10]
    ldp x22, x23, [sp, #16 * 11]
    ldp x24, x25, [sp, #16 * 12]
    ldp x26, x27, [sp, #16 * 13]
    ldp x28, x29, [sp, #16 * 14]
    add sp, sp, #ARCH_EXEC_CONT_SIZE
    /* LAB 3 TODO END */
    eret
.endm
```

练习题8：

实现 `putc` 、 `getc` 、 `thread_exit` 三个系统调用：

1. raw_syscall.h 添加三个系统调用

```
static inline void __chcore_sys_putc(char ch) {
        __chcore_syscall1(__CHCORE_SYS_putc, ch);
}

static inline u32 __chcore_sys_getc(void) {
        u32 ret = -1;
        ret = (u32) __chcore_syscall0(__CHCORE_SYS_getc);
        return ret;
}
static inline void __chcore_sys_thread_exit(void) {
        __chcore_syscall0(__CHCORE_SYS_thread_exit);
}
```

2. syscall.c 添加send和recv函数调用

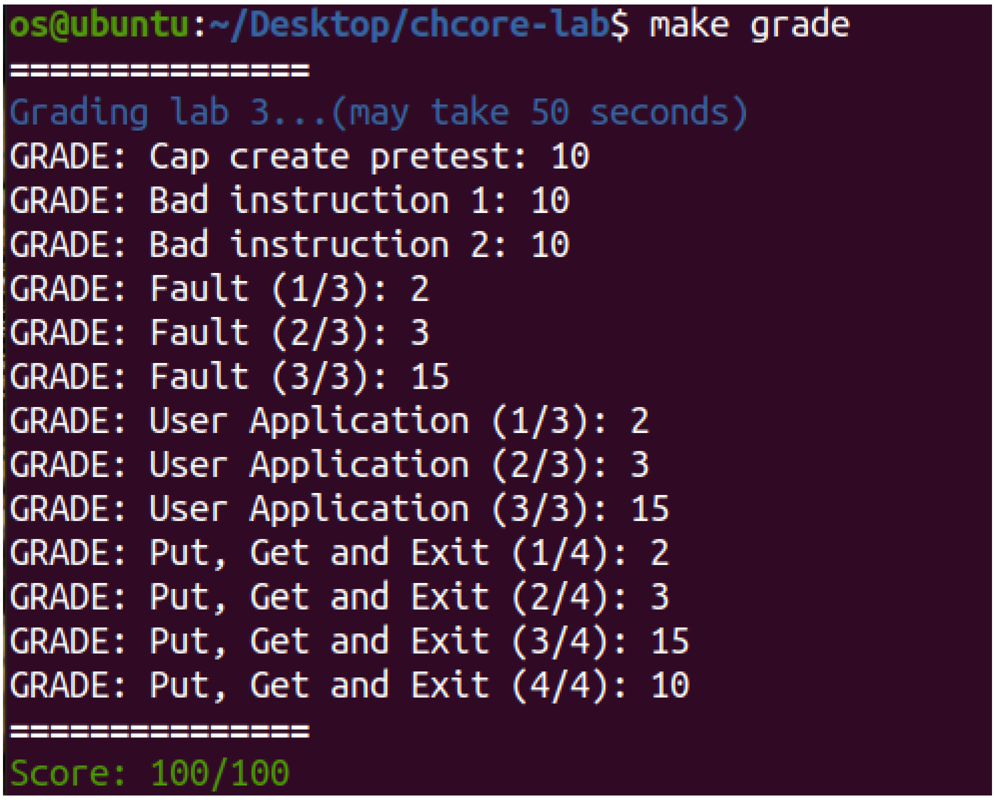
```
void sys_putc(char ch) {
        uart_send(ch);
}

u32 sys_getc(void) {
        return uart_recv();
}
```

3. thread.c 完成线程退出

```
void sys_thread_exit(void)
{
#ifdef CHCORE_LAB3_TEST
        printk("\nBack to kernel.\n");
#endif
        /* LAB 3 TODO BEGIN */

        int cpuid = smp_get_cpu_id();
        struct thread* target = current_threads[cpuid];

        target->thread_ctx->state = TS_EXIT;
        obj_free(target);

        current_threads[cpuid] = NULL;

        /* LAB 3 TODO END */
        printk("Lab 3 hang.\n");
        while (1) {
        }
        /* Reschedule */
        sched();
        eret_to_thread(switch_context());
}
```

实验结果：

按要求完成Lab 3， `make grade` 评分 `100/100` ：



```
os@ubuntu:~/Desktop/chcore-lab$ make grade
===============
Grading lab 3...(may take 50 seconds)
GRADE: Cap create pretest: 10
GRADE: Bad instruction 1: 10
GRADE: Bad instruction 2: 10
GRADE: Fault (1/3): 2
GRADE: Fault (2/3): 3
GRADE: Fault (3/3): 15
GRADE: User Application (1/3): 2
GRADE: User Application (2/3): 3
GRADE: User Application (3/3): 15
GRADE: Put, Get and Exit (1/4): 2
GRADE: Put, Get and Exit (2/4): 3
GRADE: Put, Get and Exit (3/4): 15
GRADE: Put, Get and Exit (4/4): 10
===============
Score: 100/100
```