Lab3 报告

刘翰文 522030910109

练习题 1：

在 sys_create_cap_group 函数中填写以下代码，依次完成 cap_group 的分配、cap_group 的初始化和 vmspace 的分配：

```
/* cap current cap_group */
/* LAB 3 TODO BEGIN */
/* Allocate a new cap_group object */
new_cap_group = obj_alloc(TYPE_CAP_GROUP, sizeof(*new_cap_group));
/* LAB 3 TODO END */
if (!new_cap_group) {
        r = -ENOMEM;
        goto out_fail;
}
/* LAB 3 TODO BEGIN */
/* initialize cap group from user*/
cap_group_init_user(new_cap_group, BASE_OBJECT_NUM, &args);
new_cap_group->pid = args.pid;
/* LAB 3 TODO END */

cap = cap_alloc(current_cap_group, new_cap_group);
if (cap < 0) {
        r = cap;
        goto out_free_obj_new_grp;
}

/* 1st cap is cap_group */
if (cap_copy(current_thread->cap_group,
             new_cap_group,
             cap,
             CAP_RIGHT_NO_RIGHTS,
             CAP_RIGHT_NO_RIGHTS)
    != CAP_GROUP_OBJ_ID) {
        kwarn("%s: cap_copy fails or cap[0] is not cap_group\n",
              __func__);
        r = -ECAPBILITY;
        goto out_free_cap_grp_current;
}

/* 2st cap is vmspace */
/* LAB 3 TODO BEGIN */
vmspace = obj_alloc(TYPE_VMSPACE, sizeof(*vmspace));
/* LAB 3 TODO END */
```

在 create_root_cap_group 函数中填写以下代码，依次完成 cap_group 的分配、cap_group 的初始化、vmspace 的分配和 slot 的分配：

```
/* LAB 3 TODO BEGIN */
UNUSED(vmspace);
UNUSED(cap_group);
cap_group = obj_alloc(TYPE_CAP_GROUP, sizeof(*cap_group));
/* LAB 3 TODO END */
BUG_ON(!cap_group);

/* LAB 3 TODO BEGIN */
/* initialize cap group with common, use ROOT_CAP_GROUP_BADGE */
cap_group_init_common(cap_group, BASE_OBJECT_NUM, ROOT_CAP_GROUP_BADGE);
/* LAB 3 TODO END */
slot_id = cap_alloc(cap_group, cap_group);

BUG_ON(slot_id != CAP_GROUP_OBJ_ID);

/* LAB 3 TODO BEGIN */
vmspace = obj_alloc(TYPE_VMSPACE, sizeof(*vmspace));
/* LAB 3 TODO END */
BUG_ON(!vmspace);

/* fixed PCID 1 for root process, PCID 0 is not used. */
vmspace_init(vmspace, ROOT_PROCESS_PCID);

/* LAB 3 TODO BEGIN */
slot_id = cap_alloc(cap_group, vmspace);
/* LAB 3 TODO END */
```

练习题 2：

在 create_root_thread 函数中填入以下代码，首先从每个 program header 中读取 offset、vaddr、filesz 和 memsz，然后根据 memsz 分配指定大小的 segment_pmo，接着根据读取的 offset 和 filesz 将 ELF 文件加载到内存中，最后根据 flags 构造 vmr_flags 创建页表映射：

```
/* LAB 3 TODO BEGIN */
/* Get offset, vaddr, filesz, memsz from image*/
UNUSED(flags);
UNUSED(filesz);
UNUSED(offset);
UNUSED(memsz);
memcpy(data,
        (void*)((unsigned long)&binary_procmgr_bin_start
                + ROOT_PHDR_OFF + i * ROOT_PHENT_SIZE
                + PHDR_OFFSET_OFF),
        sizeof(data));
offset = (unsigned long)le64_to_cpu(*(u64 *)data);
memcpy(data,
        (void*)((unsigned long)&binary_procmgr_bin_start
                + ROOT_PHDR_OFF + i * ROOT_PHENT_SIZE
                + PHDR_VADDR_OFF),
        sizeof(data));
vaddr = (unsigned long)le64_to_cpu(*(u64 *)data);
memcpy(data,
        (void*)((unsigned long)&binary_procmgr_bin_start
                + ROOT_PHDR_OFF + i * ROOT_PHENT_SIZE
                + PHDR_FILESZ_OFF),
        sizeof(data));
filesz = (unsigned long)le64_to_cpu(*(u64 *)data);
memcpy(data,
        (void*)((unsigned long)&binary_procmgr_bin_start
                + ROOT_PHDR_OFF + i * ROOT_PHENT_SIZE
                + PHDR_MEMSZ_OFF),
        sizeof(data));
memsz = (unsigned long)le64_to_cpu(*(u64 *)data);
/* LAB 3 TODO END */

struct pmobject *segment_pmo = NULL;
/* LAB 3 TODO BEGIN */
UNUSED(segment_pmo);
size_t pmo_size = ROUND_UP(memsz, PAGE_SIZE);
vaddr_t segment_content_kvaddr = ((unsigned long)&binary_procmgr_bin_start) + offset;
BUG_ON(filesz != memsz);
ret = create_pmo(PAGE_SIZE, PMO_DATA, root_cap_group, 0, &segment_pmo, PMO_ALL_RIGHTS);
/* LAB 3 TODO END */
```

```
/* LAB 3 TODO BEGIN */
/* Copy elf file contents into memory*/
kfree((void *)phys_to_virt(segment_pmo -> start));
segment_pmo -> start = virt_to_phys(segment_content_kvaddr);
segment_pmo -> size = pmo_size;
/* LAB 3 TODO END */

unsigned vmr_flags = 0;
/* LAB 3 TODO BEGIN */
/* Set flags*/
if (flags & PHDR_FLAGS_R) {
        vmr_flags |= VMR_READ;
}
if (flags & PHDR_FLAGS_W) {
        vmr_flags |= VMR_WRITE;
}
if (flags & PHDR_FLAGS_X) {
        vmr_flags |= VMR_EXEC;
}
/* LAB 3 TODO END */
```

练习题 3：

在 init_thread_ctx 中填写以下代码，利用传入的参数修改线程上下文中 SP_EL0、ELR_EL1 和 SPSR_EL1 寄存器中的值：

```
/* LAB 3 TODO BEGIN */
/* SP_EL0, ELR_EL1, SPSR_EL1*/
thread->thread_ctx->ec.reg[SP_EL0] = stack;
thread->thread_ctx->ec.reg[ELR_EL1] = func;
thread->thread_ctx->ec.reg[SPSR_EL1] = SPSR_EL1_EL0t;
/* LAB 3 TODO END */
```

思考题 4：

内核在完成初始化后首先调用 create_root_thread 函数创建原始进程线程，然后调用 sched 函数调度选中首个用户线程，接着调用 switch_context 函数切换线程上下文，将返回的 thread_ctx 地址传入 eret_to_thread 函数，通过调用 __eret_to_thread 函数，将 thread_ctx 地址写入 sp 寄存器并调用 eret 指令返回用户态，完成了向用户态的切换。

练习题 5：

在 irq_entry.S 中根据注释填写如下异常向量表：

```
/* LAB 3 TODO BEGIN */
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
/* LAB 3 TODO END */
```

对于 irq_el1t、fiq_el1t、fiq_el1h、error_el1t、error_el1h、sync_el1t，利用 unexpected_handler 函数处理异常：

```
irq_el1t:
fiq_el1t:
fiq_el1h:
error_el1t:
error_el1h:
sync_el1t:
    /* LAB 3 TODO BEGIN */
    bl unexpected_handler
    /* LAB 3 TODO END */
```

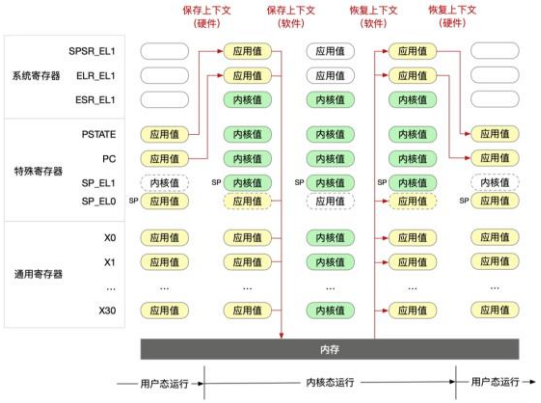
对于 sync_el1h，利用 handle_entry_c 函数处理异常，并将 x0 中的返回值存入 ESR_EL1 寄存器：

```
sync_el1h:
    exception_enter
    mov x0, #SYNC_EL1h
    mrs x1, esr_el1
    mrs x2, elr_el1

    /* LAB 3 TODO BEGIN */
    /* jump to handle_entry_c, store the return value as the ELR_EL1 */
    bl handle_entry_c
    str x0, [sp, #16 * 16]
    /* LAB 3 TODO END */
    exception_exit
```

练习题 6：

在 exception_enter 和 exception_exit 函数中，分别完成 x0-x30、SP_EL0、ELR_EL1 和 SPSR_EL1 寄存器在栈中的存入和取出操作，如下图：



填入代码分别为：exception_enter:

```
/* See more details about the bias in registers.h */
.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
```

exception_exit:

```
.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
```

对于 switch_to_stack 函数, 将 TPIDR_EL1 寄存器中的地址加上偏移量作为 cpu_stack 的地址, 填入代码如下:

```
.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
```

思考题 7：

Printf 函数首先调用 vprintf 函数，vprintf 函数调用__stdout_write 函数，__stdout_write 函数再调用__stdio_write 函数将 stdout 文件描述符传入 SYS_writev 系统调用，从而间接调用 chcore_writev 函数，接着 chcore_writev 函数调用 chcore_write 函数，最后 chcore_write 函数完成对相应文件描述符的 chcore_stdout_write 函数的调用。

练习题 8：

再 stdio.c 中填入以下代码，利用 chcore_syscall2 函数，将 buffer 和 size 传入 CHCORE_SYS_putstr 系统调用：

```
static void put(char buffer[], unsigned size)
{
        /* LAB 3 TODO BEGIN */
        chcore_syscall2(CHCORE_SYS_putstr, (vaddr_t)buffer, size);
        /* LAB 3 TODO END */
}
```

练习题 9：

首先，编写 hello_world.c:

```
# include "build/chcore-libc/include/stdio.h"

int main()
{
    printf("Hello ChCore!");
    return 0;
}
```

随后通过 libc 执行系统调用，利用 Chcore 的 libc 进行编译，将得到的 hello_world.bin 放入 ramdisk 目录下：

./build/chcore-libc/bin/musl-gcc hello_world.c -o ./ramdisk/hello_world.bin