Report

翁牧言 521030910007

思考题1

1. 在start.s中，寄存器mpidr_el1传入寄存器x8中，根据查阅手册可以得知mpidr_el1存储的是CPU的ID。
2. 将x8的值与0xFF做按位与操作，用来判断x8存储的值是否为0.
3. 如果是0，跳转到primary执行
4. 如果不是0，代码会继续运行，并且在wait_until_smp_enabled处陷入循环，实现阻塞的效果，知道主CPU完成初始化后设置secondary-boot_flag，继续执行secondary_init_c进行其他CPU的初始化操作

思考题2

- 是虚拟地址，因为使用secondary_boot_flag时MMU已经启动
- secondary_boot_flag通过main函数的参数boot_flag传入enable_smp_cores
- 在kernel/arch/aarch64/main.c文件中的main函数中可知，boot_flag是smp的boot flag地址，是物理地址，main函数调用了smp.c中的enable_smp_cores函数，在这个函数中调用了对boot_flag调用phys_to_virt将其转化为虚拟地址以获得secondary_boot_flag

练习1

在 kernel/sched/policy_rr.c 中完善 `rr_sched_init` 函数，对 `rr_ready_queue_meta` 进行初始化。

- 通过对rr_ready_queue_meta中每个CPU核心进行初始化，同时进行上锁

练习2

在 kernel/sched/policy_rr.c 中完善 `__rr_sched_enqueue` 函数，将`thread`插入到`cpuid`对应的就绪队列中。

```
        list_append(&thread->ready_queue_node,
&rr_ready_queue_meta[cpuid].queue_head);
        rr_ready_queue_meta[cpuid].queue_len ++;
```

练习3

在 kernel/sched/sched.c 中完善 `find_runnable_thread` 函数，在就绪队列中找到第一个满足运行条件的线程并返回。 在 kernel/sched/policy_rr.c 中完善 `__rr_sched_dequeue` 函数，将被选中的线程从就绪队列中移除。

- 类似于练习2

```
        list_del(&thread->ready_queue_node);
        rr_ready_queue_meta[thread->thread_ctx->cpuid].queue_len--;
```

练习4

在kernel/sched/sched.c中完善系统调用`sys_yield`，使用户态程序可以主动让出CPU核心触发线程调度。此外，请在kernel/sched/policy_rr.c 中完善`rr_sched`函数，将当前运行的线程重新加入调度队列中。

- 为了完成sys_yield，调用sched()
- 通过rr_sched_enqueue(old)将当前正在运行的线程重新加入调度队列中。

练习5

请根据代码中的注释在kernel/arch/aarch64/plat/raspi3/irq/timer.c中完善`plat_timer_init`函数，初始化物理时钟。需要完成的步骤有：

> - 读取 CNTFRQ_EL0 寄存器，为全局变量 cntp_freq 赋值。
> - 根据 TICK_MS（由ChCore决定的时钟中断的时间间隔，以ms为单位，ChCore默认每10ms触发一次时钟中断）和cntfrq_el0 （即物理时钟的频率）计算每两次时钟中断之间 system count 的增长量，将其赋值给 cntp_tval 全局变量，并将 cntp_tval 写入 CNTP_TVAL_EL0 寄存器！
> - 根据上述说明配置控制寄存器CNTP_CTL_EL0。

```
void plat_timer_init(void)
{
    u64 timer_ctl = 0;
    u32 cpuid = smp_get_cpu_id();

    /* Since QEMU only emulate the generic timer, we use the generic timer here
*/
    asm volatile ("mrs %0, cntpct_el0":"=r" (cntp_init));
    kdebug("timer init cntpct_el0 = %lu\n", cntp_init);

    /* LAB 4 TODO BEGIN (exercise 5) */
    /* Note: you should add three lines of code. */
    /* Read system register cntfrq_el0 to cntp_freq*/
    asm volatile ("mrs %0, cntfrq_el0":"=r" (cntp_freq));
    /* Calculate the cntp_tval based on TICK_MS and cntp_freq */
    cntp_tval = cntp_freq * TICK_MS / 1000;
    /* Write cntp_tval to the system register cntp_tval_el0 */
    asm volatile ("msr cntp_tval_el0, %0"::"r" (cntp_tval));
    /* LAB 4 TODO END (exercise 5) */


    tick_per_us = cntp_freq / 1000 / 1000;
    /* Enable CNTPNSIRQ and CNTVIRQ */
    put32(core_timer_irqcntl[cpuid], INT_SRC_TIMER1 | INT_SRC_TIMER3);

    /* LAB 4 TODO BEGIN (exercise 5) */
    /* Note: you should add two lines of code. */
    /* Calculate the value of timer_ctl */
    timer_ctl = 0x1;
    /* Write timer_ctl to the control register (cntp_ctl_el0) */
    asm volatile ("msr cntp_ctl_el0, %0"::"r" (timer_ctl));
    /* LAB 4 TODO END (exercise 5) */

    test_timer_init();
    return;
}
```

练习6

请在kernel/arch/aarch64/plat/raspi3/irq/irq.c中完善`plat_handle_irq`函数，当中断号irq为INT_SRC_TIMER1（代表中断源为物理时钟）时调用`handle_timer_irq`并返回。 请在kernel/irq/irq.c中完善`handle_timer_irq`函数，递减当前运行线程的时间片budget，并调用sched函数触发调度。 请在kernel/sched/policy_rr.c中完善`rr_sched`函数，在将当前运行线程重新加入就绪队列之前，恢复其调度时间片budget为DEFAULT_BUDGET。

```
    /* LAB 4 TODO BEGIN (exercise 6) */
    /* Call handle_timer_irq and return if irq equals INT_SRC_TIMER1 (physical timer) */
    case INT_SRC_TIMER1:
        handle_timer_irq();
        return;
    /* LAB 4 TODO END (exercise 6) */
```

```
                        /* LAB 4 TODO BEGIN (exercise 6) */
                        /* Refill budget for current running thread (old) */
                        old->thread_ctx->sc->budget = DEFAULT_BUDGET;
                        /* LAB 4 TODO END (exercise 6) */
```

练习7

在user/chcore-libc/musl-libc/src/chcore-port/ipc.c与kernel/ipc/connection.c中实现了大多数IPC相关的代码，请根据注释补全kernel/ipc/connection.c中的代码。

```
        /* LAB 4 TODO BEGIN (exercise 7) */
        /* Complete the config structure, replace xxx with actual values */
        /* Record the ipc_routine_entry  */
        config->declared_ipc_routine_entry = ipc_routine;

        /* Record the registration cb thread */
        config->register_cb_thread = register_cb_thread;
        /* LAB 4 TODO END (exercise 7) */
```

```
/* LAB 4 TODO BEGIN (exercise 7) */
        /* Complete the following fields of shm, replace xxx with actual values */
        // conn->shm.client_shm_uaddr = xxx;
        // conn->shm.shm_size = xxx;
        // conn->shm.shm_cap_in_client = xxx;
        // conn->shm.shm_cap_in_server = xxx;
        conn->shm.client_shm_uaddr = shm_addr_client;
        conn->shm.shm_size = shm_size;
        conn->shm.shm_cap_in_client = shm_cap_client;
        conn->shm.shm_cap_in_server = shm_cap_server;
        /* LAB 4 TODO END (exercise 7) */
```

```
/* LAB 4 TODO BEGIN (exercise 7) */
        /*
         * Complete the arguments in the following function calls,
         * replace xxx with actual arguments.
         */

        /* Note: see how stack address and ip are get in sys_ipc_register_cb_return */
        /*
                handler_config->ipc_routine_entry =
                arch_get_thread_next_ip(ipc_server_handler_thread);
                handler_config->ipc_routine_stack =
                arch_get_thread_stack(ipc_server_handler_thread);
        */
        arch_set_thread_stack(target, handler_config->ipc_routine_stack);
        arch_set_thread_next_ip(target, handler_config->ipc_routine_entry);

        /* see server_handler type in uapi/ipc.h */
        arch_set_thread_arg0(target, shm_addr);
        arch_set_thread_arg1(target, shm_size);
        arch_set_thread_arg2(target, cap_num);
        arch_set_thread_arg3(target, conn->client_badge);
        /* LAB 4 TODO END (exercise 7) */
```

```
/* LAB 4 TODO BEGIN (exercise 7) */
        /* Set target thread SP/IP/arg, replace xxx with actual arguments */
        /* Note: see how stack address and ip are get in sys_register_server */
        arch_set_thread_stack(register_cb_thread, register_cb_config->register_cb_stack);
        arch_set_thread_next_ip(register_cb_thread, register_cb_config->register_cb_entry);

        /*
         * Note: see the parameter of register_cb function defined
         * in user/chcore-libc/musl-libc/src/chcore-port/ipc.c
         */
        /* set the first parameter of call-back thread, which is also the parameter of register_cb. in connection.h, it says
        " Record the argument from the server thread"  */
        arch_set_thread_arg0(register_cb_thread, server_config->declared_ipc_routine_entry);
        /* LAB 4 TODO END (exercise 7) */
```

```
        /* LAB 4 TODO BEGIN (exercise 7) */
        /* Complete the server_shm_uaddr field of shm, replace xxx with the actual value */
        conn->shm.server_shm_uaddr = server_shm_addr;
        /* LAB 4 TODO END (exercise 7) */
```