Report of Lab4

王焕宇 522030910212

思考题 1

阅读Lab1中的汇编代码kernel/arch/aarch64/boot/raspi3/init/start.S。说明ChCore是如何选定主CPU，并阻塞其他其他CPU的执行的。

在start.S文件中，通过CPU ID和0xFF进行按位与运算，判断是否为0号CPU，此为主核。如果是0号CPU，程序将跳转到主核的初始化代码primary执行init_c。其余CPU会被阻塞在wait_until_smp_enabled代码处，循环检查secondary_boot_flag，等待主核完成初始化并设置secondary_boot_flag标志后才能够跳出循环，继续执行secondary_init_c，进行其他CPU的初始化操作。

思考题 2

阅读汇编代码kernel/arch/aarch64/boot/raspi3/init/start.S, init_c.c以及kernel/arch/aarch64/main.c，解释用于阻塞其他CPU核心的secondary_boot_flag是物理地址还是虚拟地址？是如何传入函数enable_smp_cores中，又是如何赋值的（考虑虚拟地址/物理地址）？

1. secondary_boot_flag是虚拟地址，因为取用时MMU已经启动
2. secondary_boot_flag通过main函数的boot_flag传入enable_smp_cores
3. 主CPU通过enable_smp_cores(boot_flag)函数，修改secondary_boot_flag的值。其他CPU在阻塞循环wait_until_smp_enabled检测secondary_boot_flag的值，决定是否启动

练习题 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。在完成填写之后，你可以看到输出“Scheduler metadata is successfully initialized!”并通过 Scheduler metadata initialization 测试点。

为每个核心初始化rr_ready_queue_meta

```
int i = 0;
for (i = 0; i < PLAT_CPU_NUM; i++) {
        init_list_head(&(rr_ready_queue_meta[i].queue_head));
        lock_init(&(rr_ready_queue_meta[i].queue_lock));
        rr_ready_queue_meta[i].queue_len = 0;
}
```

练习题 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。

插入ready队列，同时队列长度+1

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

练习题 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。

遍历所有就绪队列，寻找可运行线程。

```
for_each_in_list (
        thread, struct thread, ready_queue_node, thread_list) {
        if (!thread_is_suspend(thread)
                && (thread->thread_ctx->kernel_stack_state == KS_FREE
                || thread == current_thread)) {
                return thread;
        }
}
```

运行后，将线程移出ready队列，队列长度-1

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

练习题 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。

```
sched();
```

为线程设定budget，并重新加入ready队列中

```
rr_sched_refill_budget(old, DEFAULT_BUDGET);
rr_sched_enqueue(old);
```

练习题 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。

读取cntp_freq后，计算tval，并重新写入寄存器

```
asm volatile ("mrs %0, cntfrq_el0":"=r" (cntp_freq));
cntp_tval = (cntp_freq * TICK_MS / 1000);
asm volatile ("msr cntp_tval_el0, %0"::"r" (cntp_tval));
```

设定timer_ctl，写入寄存器

```
timer_ctl = 0 << 1 | 1;
asm volatile ("msr cntp_ctl_el0, %0"::"r" (timer_ctl));
```

练习题 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/timer.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。

设定中断号为INT_SRC_TIMER1时，调用handle_timer_irq

```
case INT_SRC_TIMER1:
        handle_timer_irq();
        return;
```

递减当前运行线程的时间片budget

```
if (current_thread) {
        BUG_ON(!current_thread->thread_ctx->sc);
        BUG_ON(current_thread->thread_ctx->sc->budget == 0);
        current_thread->thread_ctx->sc->budget--;
} else {
        kdebug("Timer: system not runnig!\n");
}
```

练习题 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。之后运行ChCore可以看到 “[TEST] Test IPC finished!” 输出，你可以通过 Test IPC 测试点。

按照注释补全赋值即可

```
config->declared_ipc_routine_entry = ipc_routine;
config->register_cb_thread = register_cb_thread;

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;

arch_set_thread_stack(target, handler_config->ipc_routine_stack);
arch_set_thread_next_ip(target, handler_config->ipc_routine_entry);
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);

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);
arch_set_thread_arg0(register_cb_thread, server_config->declared_ipc_routine_entry);

conn->shm.server_shm_uaddr = server_shm_addr;
```