ChCore Lab4 报告

2023 年 12 月 24 日

思考题 1. 阅读汇编代码 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。

解答.

1. 从 mpidr_el1 中获取 CPU 信息，并利用 and x8,x8,#0xFF 或得 cpu-id，如果是 0, 则是主 cpu, 跳转到 primary. 如果不为零，继续顺序执行代码。

2. (1) 非主 cpu 进入 wait_for_bss_clear 函数后,

等待 bss_clear, 主 cpu 在进入 primary 函数以后清除 bss，如果非主 cpu 检测到 bss 已被 clear，则跳出阻塞，继续执行代码。

(2) 非主 cpu 进入 wait_until_smp_enabled 函数后,

secondary_boot_flag[PLAT_CPU_NUMBER] 是一个数组，长度为 cpu 数目. 主 cpu 初始化时会执行 start_kernel(secondary_boot_flag)，将上述数组中元素置为非 0.

非主 cpu 不断查询数组中相应位置的值，并检查是否为 0。不为 0, 解除阻塞；为 0, 再次查询直至不为 0.

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）?

解答.

1.void enable_smp_cores(paddr_t boot_flag) 表明 secondary_boot_flag 是物理地址.

2. primary CPU 执行完 init_c 后，执行 start_kernel(secondary_boot_flag); 调用 kernel 的 main 函数，并将 secondary_boot_flag 作为参数传递给 main 函数, 在 main 函数中，调用 enable_smp_cores(boot_flag); 将 secondary_boot_flag 传给 enable_smp_cores 函数

3. 通过 smp_get_cpu_id 获取对应的 CPU id, 将当前 CPU 状态 cpu_status 置为 cpu_run. 将 boot_flag 由物理地址转化为虚拟地址（此时 MUU 已经激活），然后遍历 secondary_boot_flag 数组，将各个位置全部置为 1, 从而实现对各个 cpu flag 的修改. 在修改每一位的同时，利用 flush_dcache_area 刷新一次，以确保修改写入内存. 如果某个 cpu 处于 cpu_hang 待机状态，则需要等待直至 cpu 跳出 hang 状态.

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。

```
1 int rr_sched_init(void)
2 {
3         /* LAB 4 TODO BEGIN (exercise 1) */
4         /* Initial the ready queues (rr_ready_queue_meta)
             for each CPU core */
```

```
5         //struct queue_meta rr_ready_queue_meta[PLAT_CPU_NUM
              ];
6         for (int i = 0; i < PLAT_CPU_NUM; i ++)
7         {
8                 init_list_head(&rr_ready_queue_meta[i].
                      queue_head);
9                 lock_init(&rr_ready_queue_meta[i].queue_lock
                      );
10                rr_ready_queue_meta[i].queue_len = 0;
11        }
12        /* LAB 4 TODO END (exercise 1) */
13
14        test_scheduler_meta();
15        return 0;
16 }
```

查看 queue_meta 的定义，可知其有 3 个字段需要初始化，对每个 CPU 的 queue_meta 三个字段进行初始化即可

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。在完成填写之后，你可以看到输出 “Successfully enqueue root thread” 并通过 Schedule Enqueue 测试点。

```
1  int __rr_sched_enqueue(struct thread *thread, int cpuid)
2  {
3         ...
4         /* LAB 4 TODO BEGIN (exercise 2) */
```

```
5         /* Insert thread into the ready queue of cpuid and
              update queue length */
6         /* Note: you should add two lines of code. */
7         list_append(&(thread->ready_queue_node), &(
              rr_ready_queue_meta[cpuid].queue_head));
8         rr_ready_queue_meta[cpuid].queue_len += 1;
9         /* LAB 4 TODO END (exercise 2) */
10
11        return 0;
12 }
```

利用 list_append 函数将当前 thread 插入到指定 cpu-iu 的就绪队列当中，并将队列长度加一。

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。在完成填写之后，运行 ChCore 将可以成功进入用户态，你可以看到输出 “Enter Procmgr Root thread (userspace)” 并通过 Schedule Enqueue 测试点。

```
1 struct thread *find_runnable_thread(struct list_head *
      thread_list)
2 {
3         struct thread *thread = NULL;
4
5         /* LAB 4 TODO BEGIN (exercise 3) */
```

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

        }

        /* LAB 4 TODO END (exercise 3) */
        return thread;
}
```

根据提示，利用 for_each_in_list 遍历就绪队列，找到符合要求的 thread，并返回即可。

```
int __rr_sched_dequeue(struct thread *thread)
{
    ...
    /* LAB 4 TODO BEGIN (exercise 3) */
    /* Delete thread from the ready queue and upate the
        queue length */
    /* Note: you should add two lines of code. */
    list_del(&(thread->ready_queue_node));
    rr_ready_queue_meta->queue_len --;
    /* LAB 4 TODO END (exercise 3) */
```

```
10      thread->thread_ctx->state = TS_INTER;
11      obj_put(thread);
12      return 0;
13  }
```

与练习题 2 过程类似。利用 list_del 将当前 thread 从就绪队列中删除，然后将就绪队列的长度减一。

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。

```
1  /* SYSCALL functions */
2  void sys_yield(void)
3  {
4      current_thread->thread_ctx->sc->budget = 0;
5      /* LAB 4 TODO BEGIN (exercise 4) */
6      /* Trigger sched */
7      /* Note: you should just add a function call (one line
           of code) */
8      sched();
9      /* LAB 4 TODO END (exercise 4) */
10     eret_to_thread(switch_context());
11 }
```

调用 sched() 函数。

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

```
void plat_timer_init(void)
{
    ...
        /* 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);
```

```
18
19          /* LAB 4 TODO BEGIN (exercise 5) */
20          /* Note: you should add two lines of code. */
21          /* Calculate the value of timer_ctl */
22          timer_ctl = 0 << 1 | 1;
23          /* Write timer_ctl to the control register (
                cntp_ctl_el0) */
24          asm volatile("msr cntp_ctl_el0, %0"::"r" (timer_ctl
                ));
25          /* LAB 4 TODO END (exercise 5) */
26
27          test_timer_init();
28          return;
29  }
```

1. 先从 cntfrq_el0 寄存器中读取时钟频率,然后根据要求计算 cntp_tval,并将其写入 cntp_tval_el0 寄存器。

2.

* CNTP_CTL_EL0: 物理时钟的控制寄存器，第 0 位 ENABLE 控制时钟是否开启，1 代表 enble，0 代表 disable；第 1 位 IMASK 代表是否屏蔽时钟中断，0 代表不屏蔽，1 代表屏蔽。

根据要求配置掩码，并将其写入 timer_ct1 寄存器。

练习题 6.

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。

```
1  void plat_handle_irq(void)
2  {
3          ...
4          /* LAB 4 TODO BEGIN (exercise 6) */
5          /* Call handle_timer_irq and return if irq equals
              INT_SRC_TIMER1 (physical timer) */
6          case INT_SRC_TIMER1 :
7                  handle_timer_irq();
8                  break;
9          /* LAB 4 TODO END (exercise 6) */
10         ...
11 }
```

当中断号 irq 为 INT_SRC_TIMER1 时调用 ‘handle_timer_irq ‘并返回

```
1  void handle_timer_irq(void)
2  {
3      /* LAB 4 TODO BEGIN (exercise 6) */
4      /* Decrease the budget of current thread by 1 if
          current thread is not NULL */
5      if (current_thread)
```

```
6             current_thread->thread_ctx->sc->budget --;
7     sched();
8     /* Then call sched to trigger scheduling */
9
10    /* LAB 4 TODO END (exercise 6) */
11 }
```

将当前运行的线程时间片减一。

```
1  int rr_sched(void)
2  {
3  ...
4     /* LAB 4 TODO BEGIN (exercise 6) */
5     /* Refill budget for current running thread (old) */
6     old->thread_ctx->sc->budget = DEFAULT_BUDGET;
7     /* LAB 4 TODO END (exercise 6) */
8
9     old->thread_ctx->state = TS_INTER;
10
11    /* LAB 4 TODO BEGIN (exercise 4) */
12    /* Enqueue current running thread */
13    /* Note: you should just add a function call (one line
          of code) */
14    rr_sched_enqueue(old);
15 ...
16 }
```

恢复 old_thread 的调度时间片 budget 为 DEFAULT_BUDGET。

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。

```
/* 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) */
```

根据提示，记录 ipc_routine_entry 和 register_cb_thread.

```
/* LAB 4 TODO BEGIN (exercise 7) */
/* Complete the following fields of shm, replace xxx with
   actual values */
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) */
```

根据提示配置虚拟内存空间的 client_shm_uadd, sshm_size, shm_cap_in_client, shm_cap_in_server 字段.

```
1  * Set the target thread SP/IP/arguments */
2  /* LAB 4 TODO BEGIN (exercise 7) */
3  /*
4   * Complete the arguments in the following function calls,
5   * replace xxx with actual arguments.
6   */
7
8  /* Note: see how stack address and ip are get in
       sys_ipc_register_cb_return */
9  arch_set_thread_stack(target, handler_config->
       ipc_routine_stack);
10 arch_set_thread_next_ip(target, handler_config->
       ipc_routine_entry);
11
12 /* see server_handler type in uapi/ipc.h */
13 arch_set_thread_arg0(target, shm_addr);
14 arch_set_thread_arg1(target, shm_size);
15 arch_set_thread_arg2(target, cap_num);
16 arch_set_thread_arg3(target, (conn->client_badge));
17 /* LAB 4 TODO END (exercise 7) */
```

通过查阅 ernel/user-include/uapi/ipc.h

```
1  typedef void (*server_handler)(void *shm_ptr, unsigned int
       max_data_len, unsigned int send_cap_num, badge_t
       client_badge);
```

可知要为 server_handler 配置四个参数

```
/* 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
 */
arch_set_thread_arg0(register_cb_thread, server_config->
   declared_ipc_routine_entry);
/* LAB 4 TODO END (exercise 7) */
```

根据提示，为 thread 设置 stack 和 next_ip. 通过查阅 user/chcore-libc/libchcore/porting/overrides/include/chcore/ipc.h

```
void *register_cb(void *ipc_handler);
```

按图示代码设置参数.

```
/* 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;
```

```
4  /* LAB 4 TODO END (exercise 7) */
```

根据提示设置 server 端的共享内存.