ChCoreLab4

1 思考题1

在start.s文件中，首先将系统寄存器mpidr_el1中的CPU ID存储到寄存器x8中。然后通过将该值与0xFF进行按位与运算，判断是否为0号CPU（即主核）。如果是0号CPU，程序将跳转到主核的初始化代码primary执行。而对于其他CPU，它们将继续执行，并被阻塞在wait_until_smp_enabled代码处，等待主核完成初始化并设置secondary-boot_flag标志后才继续执行secondary_init_c，进行其他CPU的初始化操作。

2 思考题2

- 是虚拟地址，因为 MMU 已经启动；
- 通过 `main` 函数的参数 `boot_flag` 传入 `enable_smp_cores`
- 由kernel/arch/aarch64/main.c文件中的main.c函数中可知， `boot_flag` 是smp的boot flag地址，是物理地址，在 `enable_smp_cores` 中通过调用 `phys_to_virt` 将其转换为虚拟地址，即为 `secondary_boot_flag` 。

```
void enable_smp_cores(paddr_t boot_flag)
{
  int i = 0;
  long *secondary_boot_flag;

  /* Set current cpu status */
  cpu_status[smp_get_cpu_id()] = cpu_run;
  secondary_boot_flag = (long *)phys_to_virt(boot_flag);
  for (i = 0; i < PLAT_CPU_NUM; i++) {
    secondary_boot_flag[i] = 1;
    flush_dcache_area((u64) secondary_boot_flag,
          (u64) sizeof(u64) * PLAT_CPU_NUM);
    asm volatile ("dsb sy");
    while (cpu_status[i] == cpu_hang)
    ;
    kinfo("CPU %d is active\n", i);
  }
  /* wait all cpu to boot */
  kinfo("All %d CPUs are active\n", PLAT_CPU_NUM);
  init_ipi_data();
}
```

3 练习1

已知 PLAT_CPU_NUM 中存储着CPU的数目，在此基础上进行遍历，对每个 CPU 核心的就绪队列都进行初始化。注意要进行锁的初始化。

4 练习2

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

根据提示完成即可，注意要使用 `list_append` 而不是 `list_add`，前者才是把元素添加到尾部！

5 练习3

1. 利用好注释中的提示，使用 `for_each_in_list` 获取属性，用提示中的条件找到相应的 `thread` 即可。
2. 使用 `list_del` 将被选中的线程从就绪队列中移除，然后减少相应 `cpuid` 的就绪队列长度。

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

6 练习4

1. 在 `sys_yield` 中调用 sched() 函数来完成调度器的调度工作。
2. `rr_sched_enqueue(old)`，将当前正在运行的线程重新加入调度队列中。

7 练习5

模仿代码中已有的对寄存器赋值/读值的格式，按照README中的要求进行即可。

```
/* 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) */
```

8 练习6

```
/* LAB 4 TODO BEGIN (exercise 6) */
 /* Decrease the budget of current thread by 1 if current thread is not NULL */
 if (current_thread != NULL) {
         current_thread->thread_ctx->sc->budget -= 1;
         sched();
 }

 /* LAB 4 TODO END (exercise 6) */
```

根据README中的要求，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) */
/* Refill budget for current running thread (old) */
old->thread_ctx->sc->budget = DEFAULT_BUDGET;
/* LAB 4 TODO END (exercise 6) */
```

9 练习7

1. 在 `connection.h` 中， `declared_ipc_routine_entry` 和 `register_cb_thread` 的意义如下：

   ```
   struct ipc_server_config {
     /* Callback_thread for handling client registration */
     struct thread *register_cb_thread;

     /* Record the argument from the server thread */
     unsigned long declared_ipc_routine_entry;
   };
   ```

   于是我们结合代码上下文进行填写：

   ```
   /* 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) */
   ```

2. 可以结合命名得到提示进行填写。shm字段会记录共享内存相关的信息（包括大小、分别在客户端进程和服务器进程当中的虚拟地址和capability）。

```
    /* 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) */
```

3. 在uapi/ipc.h已知:

   ```
   * @param shm_ptr: pointer to start address of IPC shared memory. Use
   * SHM_PTR_TO_CUSTOM_DATA_PTR macro to convert it to concrete custom
   * data pointer.
   * @param max_data_len: length of IPC shared memory.
   * @param send_cap_num: number of capabilites sent by client in this request.
   * @param client_badge: badge of client.
   */
   ```

   从而我们根据上面的提示进行填写即可

   ```
       /* 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) */
   ```

4. `arch_set_thread_arg0` 的填写方法要理解传入这个函数的参数同样是 `register_cb` 的参数，然后根据 `register_cb` 参数的意义，并对照 `declared_ipc_routine_entry` "Record the argument from the server thread" 的作用完成填写。

   ```
         /* 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) */
   ```

5. 根据上下文填写即可

   ```
       /* 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) */
   ```