Lab Report 4

Wang Haotian

Grizzy@sjtu.edu.cn

519021910685

1.

从`mpidr_el1`中获取CPU的ID，然后进行判断

若ID为0，则执行primary函数，进行 primary CPU的启动

否则等待bss清零，然后等待对应smp的CPU被启动，然后再执行自己的初始化工作

2.

是物理地址。

secondary CPU在给定地址上spin。在primary CPU初始化完成之后已经配置了页表，但是MMU是每个CPU独有的，这时只有primary CPU激活了MMU使用虚拟地址，但是secondary CPU的MMU还没有激活，仍然在使用物理地址，要等待secondary boot完成后才使用虚拟地址

而传参数是根据C函数的规则，将`secondary_boot_flag`数组传入`start_kernel`函数中，然后再由该函数传参数给`main`，从而使`main`调用`enable_smp_cores`时有flag，`secondary_boot_flag`被初始化为`{NOT_BSS, 0, 0, 0}`

4.

具体实现见源文件

从内核态返回用户态时需要解锁，返回用户态的位置是`exception_exit`，因此要在合适的`exception_exit`之前加上`unlock_kernel`

5.

不需要

这些寄存器在`exception_enter`时需要保存是因为后续需要用到一些寄存器状态

而返回时，由于在`exception`发生之前的用户态状态已经被保存过了，就不需要再保存内核态处理过异常时的寄存器状态，直接从栈上恢复就好

8.

空闲线程是运行在内核态，但是并没有持有大内核锁

如果内核态运行的空闲线程释在处理异常结束时释放大内核锁，会导致 `lock->owner++`

此时假设有超过一个的内核态线程放了锁，会导致下一次 `lock` 时，取到的值小于 `lock->owner`

就永远取不到锁，导致内核阻塞