ChCore Lab1 课程作业

2023 年 10 月 28 日

题目 1. 思考题 1：阅读 ‘_start‘ 函数的开头，尝试说明 ChCore 是如何让其中一个核首先进入初始化流程，并让其他核暂停执行的。

解答.

首先从 mpidr_el1 寄存器获取了当前 cpu 信息，然后取低 8 位，即为当前 core-id。然后检查 core-id 是否为 0,

1. 若为 0 则执行’primary’，进行初始化，调整特权级到 el1, 设置栈并进入 init 函数. 值得一提的是，通过阅读 init 函数可以发现，在初始化完成后，程序会将 clear_bss_flag 置 0， secondary_boot_flag 置为 NOT_BSS (0xBEEFUL)

2. 若非 0, 顺序执行’wait_for_bss_clear’, 等待直至 core-0 将 clear_bss_flag 置 0, 调整特权级为 el1 并设置栈。然后顺序执行’wait_until_smp_enabled’, 等待直到 core-0 将 secondary_boot_flag 置为非 0。等待结束意味着 core-0 的初始化已完成，本 core 执行 secondary_init_c 函数

题目 2. 练习题 2：在 ‘arm64_elX_to_el1‘ 函数的 ‘LAB 1 TODO 1‘ 处填写一行汇编代码，获取 CPU 当前异常级别。

解答.

```
mrs x9, CurrentEL
```

题目 2 的注记.

CurrentEL 寄存器可获得当前异常级别

利用 mrs 指令将 CurrentEL 寄存器的值存入到 x9 即可

题目 3. 练习题 3：在 ‘arm64_elX_to_el1‘ 函数的 ‘LAB 1 TODO 2‘ 处填写大约 4 行汇编代码，设置从 EL3 跳转到 EL1 所需的 ‘elr_el3‘ 和 ‘spsr_el3‘ 寄存器值。具体地，我们需要在跳转到 EL1 时暂时屏蔽所有中断、并使用内核栈（‘sp_el1‘ 寄存器指定的栈指针）。

解答.

```
adr x9, .Ltarget
msr elr_el3, x9
mov x9, SPSR_ELX_DAIF | SPSR_ELX_EL1H
msr spsr_el3, x9
```

题目 3 的注记.

将跳转地址的 label 写入异常链接寄存器。修改程序状态寄存器 spsr_el3，以达到暂时屏蔽所有中断，并使用内核栈 sp_el1 寄存器指定的栈指针。

其中：

DAIF 为异常掩码位，全置为 1 意味着屏蔽所有中断，SPSR_ELX_DAIF 实际上被定义为 (0xb1111 « 6)；

EL 是执行状态控制位，SPSR_ELX_EL1H 对应使用内核栈 ‘sp_el1‘ 寄存器指定的栈指针

题目 4. 思考题 4：说明为什么要在进入 C 函数之前设置启动栈。如果不设置，会发生什么？

解答.

C 语言的程序的运行依赖于栈，函数的调用，函数运行时变量的保存与使用都涉及到栈。

如果没有设置栈，程序将无法正确运行

题目 5. 思考题 5：在实验 1 中，其实不调用 ‘clear_bss‘ 也不影响内核的执行，请思考不清理 ‘.bss‘ 段在之后的何种情况下会导致内核无法工作。

解答. ‘.bss‘ 段存储的是未初始化的全局变量和静态变量，由于这些变量的值是不确定甚至不合法的，如果不清除，可能导致程序访问这些未初始化的变量时出现意外的结果，内核无法正常工作。

但在本例中，程序访问变量时都会先进行初始化，所以不存在上述问题

题目 6. 练习题 6：在 ‘kernel/arch/aarch64/boot/raspi3/peripherals/uart.c‘ 中 ‘LAB 1 TODO 3‘ 处实现通过 UART 输出字符串的逻辑。

解答.

```
early_uart_init();
char *p = str;
while (p && *p != '\0')
{
        early_uart_send((unsigned int)*p);
    p ++;
}
```

题目 6 的注记. 先进行初始化操作，然后逐个读取字符串中的字符，并利用 early_uart_send 实现逐个打印。

题目 7. 练习题 7：在 ‘kernel/arch/aarch64/boot/raspi3/init/tools.S‘ 中 ‘LAB 1 TODO 4‘ 处填写一行汇编代码，以启用 MMU。

解答.

```
orr x8, x8, #SCTLR_EL1_M
```

题目 7 的注记. 设置 ‘SCTLR_EL1‘ 的 ‘M ‘位，使能 ‘MMU‘ 启动

题目 8. 思考题 8：请思考多级页表相比单级页表带来的优势和劣势（如果有的话），并计算在 AArch64 页表中分别以 4KB 粒度和 2MB 粒度映射 0 ~ 4GB 地址范围所需的物理内存大小（或页表页数量）

解答. 优势：可以节省大量空间；在物理地址存在大量空洞的情况下，非连续的多级页表可以减少空间不必要的开支。

劣势：多级页表增加了访问时间，带来更多的时间开销。” 以时间换空间”

4KB 粒度：4KB / 64 = 4 * 1024 * 8 / 64 = 512. 最底层的物理页数为 4GB / 4KB = 2^{20}.所需$L3$页表数$2^{20}/512 = 2^{11}$ 所需 L2 页表数 2048/512 = 4, 所需 L1, L0 页表各一个，总共需要 $1 + 1 + 4 + 2^{11} = 2054$个页表页。

所需物理内存为 2054 ×4KB = 8216KB

2MB 粒度:以 2MB 大页为单位映射,共有 2^{11}个大页,至多需要$L2$级页表.

所需 L2 页表数 $2^{11}/2^9 = 4$,一个$L1$页表和一个$L0$页表,总共需要$1+1+4 = 6$个页表页。所需物理内存为$6 \times 4KB = 24KB$

题目 9. 练习题 9：请在 ‘init_kernel_pt‘ 函数的 ‘LAB 1 TODO 5‘ 处配置内核高地址页表（‘boot_ttbr1_l0‘、‘boot_ttbr1_l1‘ 和 ‘boot_ttbr1_l2‘），以 2MB 粒度映射。

解答.

```
/* LAB 1 TODO 5 BEGIN */
/* Step 1: set L0 and L1 page table entry */
/* BLANK BEGIN */
vaddr = KERNEL_VADDR + PHYSMEM_START;
boot_ttbr1_l0[GET_L0_INDEX(vaddr)] = ((u64)boot_ttbr1_l1)
        | IS_TABLE | IS_VALID | NG;
boot_ttbr1_l1[GET_L1_INDEX(vaddr)] = ((u64)boot_ttbr1_l2)
        | IS_TABLE | IS_VALID | NG;
/* BLANK END */

/* Step 2: map PHYSMEM_START ~ with 2MB granularity*/
/* BLANK BEGIN */
for (; vaddr < KERNEL_VADDR + PERIPHERAL_BASE;
    vaddr += SIZE_2M) {
        boot_ttbr1_l2[GET_L2_INDEX(vaddr)] =
                (vaddr - KERNEL_VADDR)
                /* high mem, va - KERNEL_VADDR  = pa*/
                | UXN /* Unprivileged execute never */
                | ACCESSED /* Set access flag */
                | NG /* Mark as not global */
                | INNER_SHARABLE /* Sharebility */
                | NORMAL_MEMORY /* Normal memory */
                | IS_VALID;
}
/* BLANK END */
```

```
/* Step 2: map PERIPHERAL_BASE 2MB granularity */
/* BLANK BEGIN */
for (vaddr = PERIPHERAL_BASE + KERNEL_VADDR;
     vaddr < KERNEL_VADDR + PHYSMEM_END; += SIZE_2M) {
          boot_ttbr1_l2[GET_L2_INDEX(vaddr)] =
                    (vaddr - KERNEL_VADDR)
                    /* high mem, va - KERNEL_VADDR = pa */
                    | UXN /* Unprivileged execute never */
                    | ACCESSED /* Set access flag */
                    | NG /* Mark as not global */
                    | DEVICE_MEMORY /* Device memory */
                    | IS_VALID;
}
/* BLANK END */
/* LAB 1 TODO 5 END */
```

题目 9 的注记.

第一步： 设置 L1、L0 页表对应的入口, 并设置好对应表示的 IS_TABLE、IS_VAILD、NG 字段

第二步:设置 L2 页表对应的每个页表项的值利用 for 循环一次将对应的虚拟地址对应的页表项设置为对应虚拟地址 - kernel virtual address ，并设置好对应的属性字段这里需要分成两次循环，分别对物理内存（SDRAM)、共享外设内存进行映射，因为对应的属性字段不同.

题目 10. 思考题 10：请思考在 ‘init_kernel_pt‘函数中为什么还要为低地址配置页表，并尝试验证自己的解释。

解答. 在启动 mmu 的 ‘el1_mmu_activate ‘函数中设置 ‘sctlr_el1‘ 后，

chcore 将使用虚拟地址，然而下一条指令仍位于低地址空间，使得 chcore 无法继续初始化。

验证：删除掉低地址配置代码后, chcore 停止在 ‘[BOOT] Install kernel page table‘, gdb 显示进入 ‘invalidate_cache_all‘.