

# 实验4 鸿蒙 LiteOS-a 内核移植——内存移植

《实用操作系统》实验报告    22920212204392 黄勛

## 1 实验环境

Windows10 21H2、Vmware Workstation Pro 16、Ubuntu18.04

配置了相关的软件。


## 2 实验目的

查看教程手册，了解 arm 架构内存映射，阅读修改源码实现内存移植

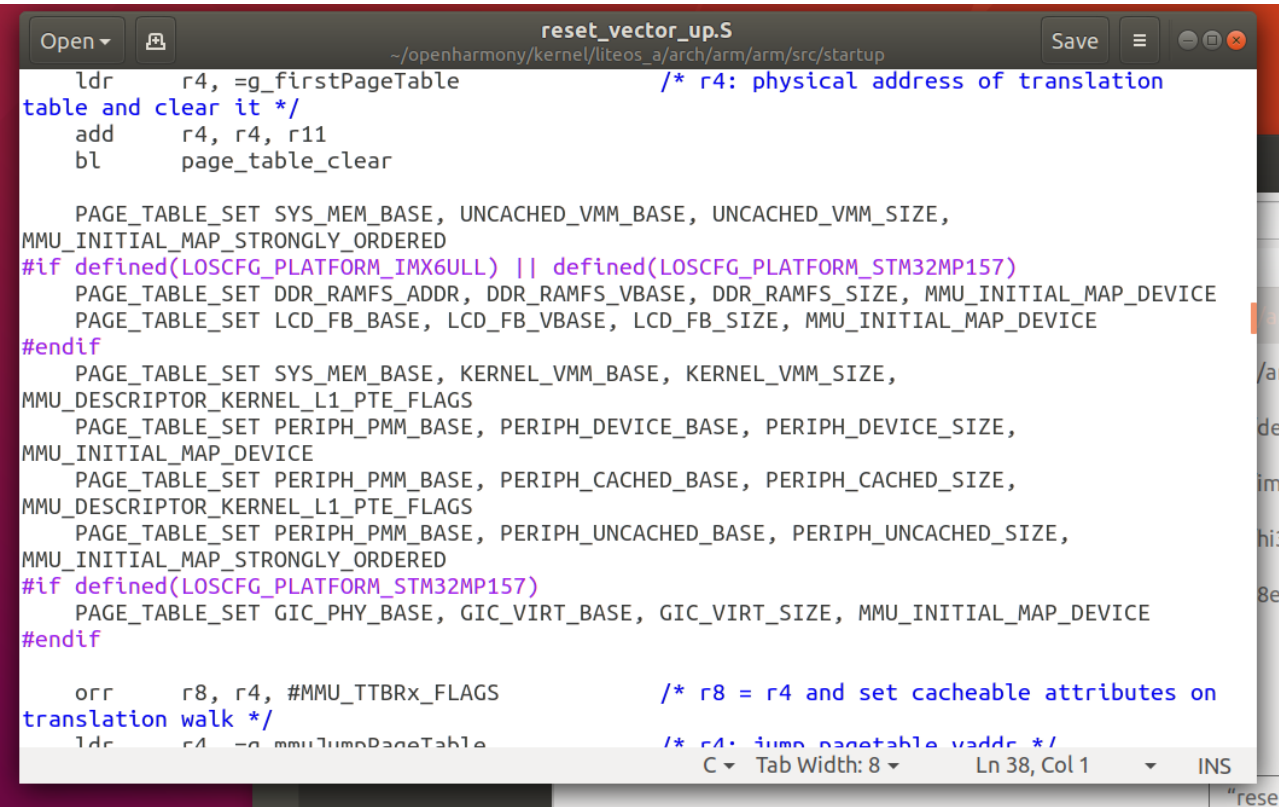
## 3 实验步骤与内容

### 3.1 地址映射分析

分析启动文件 `kernel\liteos_a\arch\arm\arm\src\startup\reset_vector_up.S`

Name	Size	Location
 reset_vector_up.S	16.1 kB	openharmony/kernel/liteos_a/arch/arm/arm/src/startup

打开文件，分析代码：



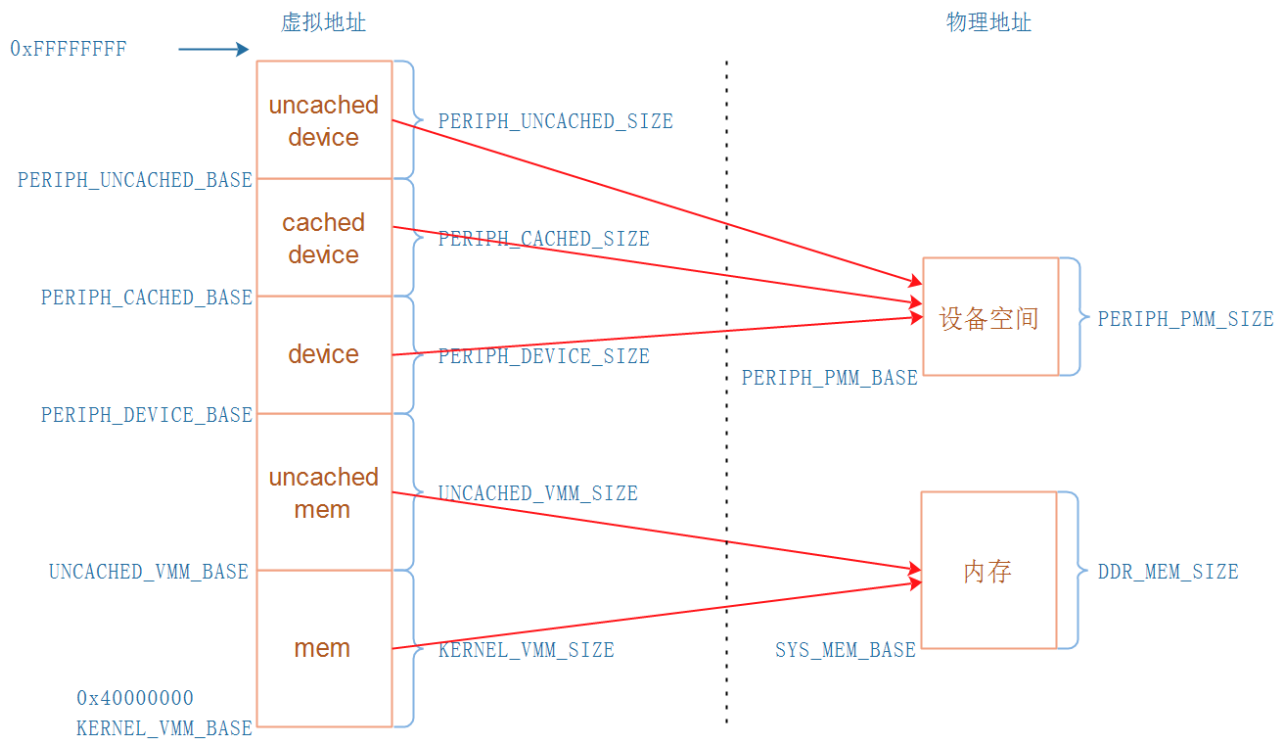
```
Open ▾  reset_vector_up.S  Save  ≡  ◀  ▶  ✕
~/openharmony/kernel/liteos_a/arch/arm/arm/src/startup

    ldr    r4, =g_firstPageTable          /* r4: physical address of translation
table and clear it */
    add    r4, r4, r11
    bl     page_table_clear

    PAGE_TABLE_SET SYS_MEM_BASE, UNCACHED_VMM_BASE, UNCACHED_VMM_SIZE,
MMU_INITIAL_MAP_STRONGLY_ORDERED
    #if defined(LOSCFG_PLATFORM_IMX6ULL) || defined(LOSCFG_PLATFORM_STM32MP157)
    PAGE_TABLE_SET DDR_RAMFS_ADDR, DDR_RAMFS_VBASE, DDR_RAMFS_SIZE, MMU_INITIAL_MAP_DEVICE
    PAGE_TABLE_SET LCD_FB_BASE, LCD_FB_VBASE, LCD_FB_SIZE, MMU_INITIAL_MAP_DEVICE
    #endif
    PAGE_TABLE_SET SYS_MEM_BASE, KERNEL_VMM_BASE, KERNEL_VMM_SIZE,
MMU_DESCRIPTOR_KERNEL_L1_PTE_FLAGS
    PAGE_TABLE_SET PERIPH_PMM_BASE, PERIPH_DEVICE_BASE, PERIPH_DEVICE_SIZE,
MMU_INITIAL_MAP_DEVICE
    PAGE_TABLE_SET PERIPH_PMM_BASE, PERIPH_CACHED_BASE, PERIPH_CACHED_SIZE,
MMU_DESCRIPTOR_KERNEL_L1_PTE_FLAGS
    PAGE_TABLE_SET PERIPH_PMM_BASE, PERIPH_UNCACHED_BASE, PERIPH_UNCACHED_SIZE,
MMU_INITIAL_MAP_STRONGLY_ORDERED
    #if defined(LOSCFG_PLATFORM_STM32MP157)
    PAGE_TABLE_SET GIC_PHY_BASE, GIC_VIRT_BASE, GIC_VIRT_SIZE, MMU_INITIAL_MAP_DEVICE
    #endif

    orr    r8, r4, #MMU_TTBx_FLAGS        /* r8 = r4 and set cacheable attributes on
translation walk */
    ldr    r4, =g_mmuJumpPageTable       /* r4: jump pagetable vaddr */
    C ▾  Tab Width: 8 ▾  Ln 38, Col 1  ▾  INS
```

可以得到下图所示的地址映射关系：



## • 内存地址

- `KERNEL_VMM_BASE`开始的这块虚拟地址，使用Cache，速度快
- `UNCACHED_VMM_BASE`开始的这块虚拟地址，不使用Cache，适合DAM传输、LCD Framebuffer等


## • 设备空间：就是各种外设，比如UART、LCD控制器、I2C控制器、中断控制器

- `PERIPH_DEVICE_BASE`开始的这块虚拟地址，不使用Cache不使用Buffer
- `PERIPH_CACHED_BASE`开始的这块虚拟地址，使用Cache使用Buffer
- `PERIPH_UNCACHED_BASE`开始的这块虚拟地址，不使用Cache但是使用Buffer

## 3.2 Liteos-a的地址空间分配

`KERNEL_VMM_BASE` 等于0x40000000，并且在

`kernel\liteos_a\kernel\base\include\los_vm_zone.h` 看到如下语句：

Name	Size	Location
 <code>los_vm_zone.h</code>	4.3 kB	openharmony/kernel/liteos_a/kernel/base/include

```

Open  Save  ~openharmony/kernel/liteos_a/kernel/base/include

#define IO_DEVICE_ADDR(paddr)    (paddr - PERIPH_PMM_BASE + PERIPH_DEVICE_BASE)
#define IO_CACHED_ADDR(paddr)   (paddr - PERIPH_PMM_BASE + PERIPH_CACHED_BASE)
#define IO_UNCACHED_ADDR(paddr) (paddr - PERIPH_PMM_BASE + PERIPH_UNCACHED_BASE)

#define MEM_CACHED_ADDR(paddr)   (paddr - DDR_MEM_ADDR + KERNEL_VMM_BASE)
#define MEM_UNCACHED_ADDR(paddr) (paddr - DDR_MEM_ADDR + UNCACHED_VMM_BASE)

#define VMM_TO_UNCACHED_ADDR(vaddr) (vaddr - KERNEL_VMM_BASE + UNCACHED_VMM_BASE)
#define UNCACHED_TO_VMM_ADDR(vaddr) (vaddr - UNCACHED_VMM_BASE + KERNEL_VMM_BASE)

#define VMM_TO_DMA_ADDR(vaddr)   (vaddr - KERNEL_VMM_BASE + SYS_MEM_BASE)
#define DMA_TO_VMM_ADDR(vaddr)  (vaddr - SYS_MEM_BASE + KERNEL_VMM_BASE)

#if (PERIPH_UNCACHED_BASE >= (0xFFFFFFFF - PERIPH_UNCACHED_SIZE))
#error "Kernel virtual memory space has overflowed!"
#endif

#ifdef __cplusplus
if __cplusplus
}
#endif /* __cplusplus */
#endif /* __cplusplus */

#endif
C/ObjC Header  Tab Width: 8  Ln 1, Col 1  INS

```

```

#if (PERIPH_UNCACHED_BASE >= (0xFFFFFFFFFU - PERIPH_UNCACHED_SIZE))
#error "Kernel virtual memory space has overflowed!"
#endif

```

所以可以粗略地认为：

- 内核空间：0x40000000 ~ 0xFFFFFFFF
- 用户空间：0 ~ 0x3FFFFFFF

### 3.3 修改内存地址范围

**Table 2-1. System memory map**

Start address	End address	Size	Description
8000_0000	FFFF_FFFF	2048 MB	MMDC—x16 DDR Controller.
7000_0000	7FFF_FFFF	256 MB	Reserved

100ASK\_IMX6ULL开发板上DDR容量是512M，所以做如下针对修改：

```

7  /* physical memory base and size */
8  -#define DDR_MEM_ADDR      0xC0000000
9  -#define DDR_MEM_SIZE      0x1C000000 /* 448M for os, reserved (512-448) for ramfs and framebuffer */
10 +#define DDR_MEM_ADDR      0x80000000
11 +#define DDR_MEM_SIZE      0x20000000

// vendor\democop\demochip\board\include\board.h
#define DDR_MEM_ADDR      0x80000000
#define DDR_MEM_SIZE      0x20000000

```

### 3.4 修改设备地址范围

IMX6ULL芯片上设备地址分部太零散，从0到0x6FFFFFFF都有涉及，中间有很多保留的地址不用，如下图：

**Table 2-1. System memory map**

Start address	End address	Size	Description
8000_0000	FFFF_FFFF	2048 MB	MMDC—x16 DDR Controller.
7000_0000	7FFF_FFFF	256 MB	Reserved
6000_0000	6FFF_FFFF	256 MB	QSPI1 Memory
5800_0000	5FFF_FFFF	128 MB	EIM Aliased
5000_0000	57FF_FFFF	128 MB	EIM (NOR/SRAM)
1000_0000	4FFF_FFFF	1024 MB	Reserved
0E00_0000	0FFF_FFFF	32 MB	Reserved
0C00_0000	0DFF_FFFF	32 MB	QSPI1 Rx Buffer
0900_0000	0BFF_FFFF	48 MB	Reserved
0800_0000	08FF_FFFF	16 MB	Reserved
02C0_0000	07FF_FFFF	84 MB	Reserved
0230_0000	02BF_FFFF	9 MB	Reserved
0220_0000	022F_FFFF	1 MB	Table 2-4 AIPS-3. See IP listing on the separate map.
0210_0000	021F_FFFF	1 MB	Table 2-3 AIPS-2. See the IP listing on the separate map.
0200_0000	020F_FFFF	1 MB	Table 2-2 AIPS-1. See the IP listing on the separate map.
0181_0000	01FF_FFFF	8128 KB	Reserved
0180_C000	0180_FFFF	16 KB	Reserved

如果把0到0x6FFFFFFF全部映射完，地址空间不够；

**正确的做法**应该是忽略那些保留的地址空间，为各个模块**单独映射地址**。

但是Liteos-a尚未实现这样的代码。

我们至少要映射2个设备的地址：UART1(100ASK\_IMX6ULL开发板使用UART1)、GIC，如下图：

**Table 2-1. System memory map (continued)**

Start address	End address	Size	Description
0180_8000	0180_BFFF	16 KB	BCH
0180_6000	0180_7FFF	8 KB	GPMI
0180_4000	0180_5FFF	32 KB	APBH DMA
0180_0000	0180_3FFF	16 KB	Reserved
0120_0000	017F_FFFF	6 MB	Reserved
0110_0000	011F_FFFF	1 MB	Reserved
0100_0000	010F_FFFF	1 MB	Reserved
00F0_0000	00FF_FFFF	1 MB	Reserved
00E0_0000	00EF_FFFF	1 MB	(per_m) configuration port
00D0_0000	00DF_FFFF	1 MB	(cpu) configuration port
00C0_0000	00CF_FFFF	1 MB	GPV_1 PL301
00B0_0000	00BF_FFFF	1 MB	GPV_0 PL301 configuration port
00A0_8000	00AF_FFFF	992 KB	Reserved
00A0_0000	00A0_7FFF	32 KB	ARM Peripherals: GIC400 Only visible to ARM core(s)
009C_0000	009F_FFFF	256 KB	Reserved

0203_4000	0203_7FFF
0203_0000	0203_3FFF
0202_C000	0202_FFFF
0202_8000	0202_BFFF
0202_4000	0202_7FFF
0202_0000	0202_3FFF
0201_C000	0201_FFFF

AIPS-1 (via SPBA)  
Glob,Module ENABLE

ASRC	16 KB
SAI3	16 KB
SAI2	16 KB
SAI1	16 KB
ESAI	16 KB
UART1	16 KB
Reserved for SDMA internal registers	16 KB

所以做如下修改：

```

16  #define LCD_FB_SIZE    0x400000 /* 4M */
17
18  /* Peripheral register address base and size */
19  #define PERIPH_PMM_BASE    0x40000000
20  #define PERIPH_PMM_SIZE    0x20000000
21  #define PERIPH_PMM_BASE    0x00a00000 // GIC的基地址
22  #define PERIPH_PMM_SIZE    0x02300000 // 尽可能大一点,以后使用其他外设时就不用映射了

// source\vendor\democom\demochip\board\include\board.h
#define PERIPH_PMM_BASE    0x00a00000 // GIC的基地址
#define PERIPH_PMM_SIZE    0x02300000 // 尽可能大一点,以后使用其他外设时就不用映射了

```

### 3.5 编译

make clean

```

book@100ask: ~/openharmy/kernel/liteos_a
File Edit View Search Terminal Help
make[1]: Leaving directory '/home/book/openharmony/base/hiviewdfx/frameworks/hilog_lite/featured'
make[1]: Entering directory '/home/book/openharmony/kernel/liteos_a/shell'
make[1]: Leaving directory '/home/book/openharmony/kernel/liteos_a/shell'
make[1]: Entering directory '/home/book/openharmony/kernel/liteos_a/net/telnet'
make[1]: Leaving directory '/home/book/openharmony/kernel/liteos_a/net/telnet'
make[1]: Entering directory '/home/book/openharmony/kernel/liteos_a/syscall'
make[1]: Leaving directory '/home/book/openharmony/kernel/liteos_a/syscall'
make[1]: Entering directory '/home/book/openharmony/kernel/liteos_a/kernel/user'
make[1]: Leaving directory '/home/book/openharmony/kernel/liteos_a/kernel/user'
make[1]: Entering directory '/home/book/openharmony/kernel/liteos_a/security'
make[1]: Leaving directory '/home/book/openharmony/kernel/liteos_a/security'
make[1]: Entering directory '/home/book/openharmony/kernel/liteos_a/apps'
make[2]: Entering directory '/home/book/openharmony/kernel/liteos_a/apps/shell'
make[2]: Leaving directory '/home/book/openharmony/kernel/liteos_a/apps/shell'
make[2]: Entering directory '/home/book/openharmony/kernel/liteos_a/apps/init'
make[2]: Leaving directory '/home/book/openharmony/kernel/liteos_a/apps/init'
make[1]: Leaving directory '/home/book/openharmony/kernel/liteos_a/apps'
make[1]: Entering directory '/home/book/openharmony/kernel/liteos_a'
make[1]: Leaving directory '/home/book/openharmony/kernel/liteos_a'
clean demochip finish

```

make -j 8



```
book@100ask: ~/openharmy/kernel/liteos_a
File Edit View Search Terminal Help
l/liteos_a/out/demochip/liteos.map -o /home/book/openharmony/kernel/liteos_a/out
/demochip/liteos --start-group -lclang_rt.builtins -lunwind --no-dependent-libr
aries -lcortex-a7 -lbsp -lrootfs -lbase -lboard -lmt_common -lspinor_flash -lst
m32mp157-uart -lcpup -ldynload -lvdso -ltickless -lliteipc -lpipes -lc -lsec -ls
crew -lc++ -lc++abi -lcppsupport -lz -lposix -lbsd -llinuxkpi -lvfs -lmulti_part
ition -lbch -lfat -lvirpart -ldisk -lbcache -lramfs -lnfs -lproc -ljffs2 -llwip
--whole-archive -lhdf -lhdf_config -lhello --no-whole-archive -lhievent -lmem -l
mt_common -lhilog -lshell -ltelnet -lsyscall -lsecurity --end-group
/home/book/llvm/bin/../../bin/llvm-objcopy -R .bss -O binary /home/book/openharmon
y/kernel/liteos_a/out/demochip/liteos /home/book/openharmony/kernel/liteos_a/out
/demochip/liteos.bin
/home/book/llvm/bin/../../bin/llvm-objdump -t /home/book/openharmony/kernel/liteos
_a/out/demochip/liteos |sort >/home/book/openharmony/kernel/liteos_a/out/demochi
p/liteos.sym.sorted
/home/book/llvm/bin/../../bin/llvm-objdump -d /home/book/openharmony/kernel/liteos
_a/out/demochip/liteos >/home/book/openharmony/kernel/liteos_a/out/demochip/lite
os.asm
make[1]: Entering directory '/home/book/openharmony/kernel/liteos_a/apps'
make[2]: Entering directory '/home/book/openharmony/kernel/liteos_a/apps/shell'
make[2]: Leaving directory '/home/book/openharmony/kernel/liteos_a/apps/shell'
make[2]: Entering directory '/home/book/openharmony/kernel/liteos_a/apps/init'
make[2]: Leaving directory '/home/book/openharmony/kernel/liteos_a/apps/init'
make[1]: Leaving directory '/home/book/openharmony/kernel/liteos_a/apps'
book@100ask:~/openharmy/kernel/liteos_a$
```

至此实验完成。

## 4 问题和解决方法

本次实验未出现问题。

## 5 实验体会

在本次实验中，我进行了鸿蒙LiteOS-a内核移植的一部分，以下是一些关键的概念和经验总结：

- 虚拟地址向物理地址转换**：虚拟地址是程序代码中使用的地址，而物理地址是实际RAM中的地址。操作系统负责虚拟地址到物理地址的映射，以便为不同的进程提供隔离和内存管理。本次实验中，我分析了LiteOS-a的虚拟地址空间，了解了内核空间 and 用户空间之间的划分。
- Cache和Buffer**：Cache是一种高速缓存，用于存储最近访问的数据，以提高访问速度。Buffer通常用于存储未提交的数据，以提高I/O性能。在LiteOS-a中，可以看到不同的内存区域是否使用Cache和Buffer，这会影响存储和访问数据的速度和一致性。
- 页表映射**：页表是虚拟内存管理的核心概念。它们用于将虚拟地址映射到物理地址，允许操作系统在不同进程之间共享物理内存。

在未来的实验中，可以继续深入研究操作系统内核和其他系统组件的移植和定制，以扩展技能和知识。