# 实验三 内存管理

#### 实验内容

写一个模块mtest, 当模块加载时, 创建一个proc文件 /proc/mtest, 该文件接收三种类型的参数, 具体如下:

- listvma 打印当前进程的所有虚拟内存地址,打印格式为 start-addr end-addr permission
- findpage addr 把当前进程的虚拟地址转化为物理地址并打印,如果不存在这样的翻译,则输出 translation not found
- writeval addr val 向当前地址的指定虚拟地址中写入一个值。

注: 所有输出可以用 printk 来完成,通过 dmesq 命令查看即可。

# 实验提示

- 1. task\_struct 中的 mm\_struct mm 数据成员指向进程拥有的内存描述符,它是在哪里定义的呢? 一个进程的内存空间包括哪些段呢?参考 include/linux/mm\_types.h
- 2. 仔细阅读 vm\_area\_struct 结构体,内存区间是如何组织在一起的?内存区间的读写权限是如何表示的?这是实现listvma的核心。
- 3. linux的虚拟地址到物理地址的转换是怎么实现的?linux的分页机制?参考 arch/arm64/include/asm/pgtable\_types.h 和 arch/arm64/include/asm/pgtable.h。注意,在linux 4.12 (or 某个版本)之后,linux的分页机制已经由四级变为五级,找到这五级页表的映射关系是解决 findpage 的核心。
- 4. 如果需要在某个地址上写数据,首先应该判断该vma 是否有效,其次要判断是否可写。如可写,找到相关页进行改写,计算出 kernel address 再将地址对应的值改变即可。

### 实现思路 (仅供参考)

```
//<mtest.c>
//include相关头文件
/*Print all vma of the current process*/
static void mtest_list_vma(void)
   //TODO: 获取所有vma并输出
/*Find va->pa translation */
static void mtest_find_page(unsigned long addr)
{
       //TODO: 虚拟地址到物理地址的转换,输出转换后的物理地址
}
/*Write val to the specified address */
static void mtest_write_val(unsigned long addr, unsigned long val)
   //TODO: 将val写入指定地址
}
static ssize_t mtest_proc_write(struct file *file,
               const char __user * buffer,
                              size_t count, loff_t * data)
```

```
//<Makefile>
//TODO: 参考模块编程实验
```

```
#测试指令
insmod mtest.ko
echo listvma > /proc/mtest
dmesg

echo findpage xxxxxxxxxxxxx > /proc/mtest
dmesg | tail -1

echo writeval xxxxxxxxxxxx x > /proc/mtest
dmesg | tail -1
```

注:以上模板仅供参考,也可以完全按照自己的思路来实现,实验报告中如果能清晰具体地写出自己的实验思路和具体实现会比较有亮点。

#### 实验效果

listvma - 打印当前进程的所有虚拟内存地址。

findpage - 如果可以转换,则输出转换地址,如果不可以访问,则输出提示信息。

writeval - 如果可写,写入地址,否则输出不可写的提示信息。

# 作业验收与提交

验收方式: 电院3号楼118向助教展示实验效果。

提交渠道: Canvas

提交文件: "学号\_姓名\_project3.zip"

• 源码文件夹"学号\_姓名\_project3\_src" (\*.c Makefile)

