Lab 共享内存

一、背景

在 Android 中有许多不同的 IPC 机制，在这个 lab 中，我们要实现一个最简单有效的 IPC 机制：一个简单的共享内存。我们将实现新的系统调用来创建一个共享内存区域，将现有的内存共享区域映射到进程的地址空间，以及从地址空间分离映射区。

二、任务与要求

Part I：

新建 mm/ssmem.c 文件，并在 ssmem.c 中增加两个系统调用以实现实现共享内存机制。

/* 新增系统调用 ssmem_attach.

参数：

1. 参数 id 表示 ssmem 段的 id 索引。在本次 lab 中采用 char* 类型作为 ssmem 的 id。假设 ssmem segment 最多有 1024 个。
2. 参数 flags 表示当前进程对 ssmem 段的操作权限，默认可以读。（exec 可以根据自身能力实现并测试）

```
#define SSMEM_FLAG_CREATE 0x1
#define SSMEM_FLAG_WRITE  0x2
#define SSMEM_FLAG_EXEC   0x4
```

3. 参数 length 表示 ssmem 段的长度

要求：

1. 如果 ssmem 号为 id 的进程不存在，且设置了 SSMEM_FLAG_CREATE，则创建一个新的共享内存区域。只有在创建共享内存区时，length 参数才会起作用
2. 如果函数调用成功，返回共享内存段的指针；如果无法分配内存空间，返回-ENOMEM；如果参数 id 或 length 无效，返回-EINVAL；如果指定的共享内存区域 id 不存在，且没有设置 SSMEM_FLAG_CREATE，返回-EADDRNOTAVALL*/

```
void *ssmem_attach(int id, int flags, size_t length);
```

/*新增系统调用 ssmem_detach

参数：addr 表示该进程地址空间中某个虚拟地址

要求：

1. 如果当前进程是唯一引用这段共享内存的进程，则释放回收共享内存，并且清空与之关联的数据结构。
2. 如果成功，返回 0，如果地址无效或该地址对应的 vma 并没有 map 一个共享内存段，则返回-EFAULT*/

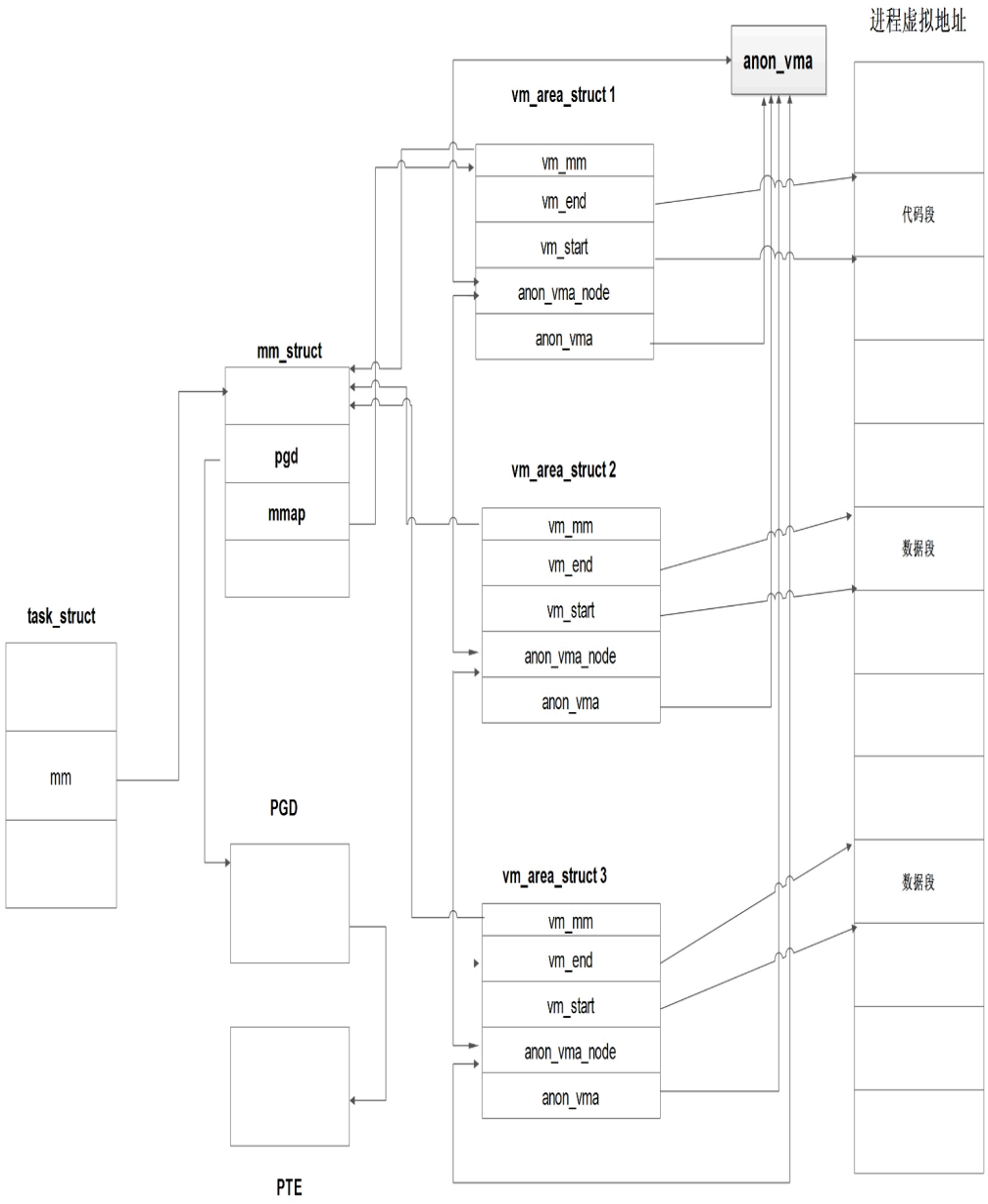
```
int ssmem_detach(void *addr);
```

Tips：

总体而言，本次 lab 需要你对 Android 内存管理的一系列 subsystems 都有比较完善的理解。virtual memory areas 的相关管理在 mmap.c 中，物理页面的管理在 memory.c 中，反向映射机制的实现在 rmap.c

中，swap 机制本次 lab 不作要求，但是代码实现在 vmscan.c 中实现。有能力的同学可以构想支持 swap 机制的做法。以上几个文件代码量比较大，不必全部理解，但是搞清楚每个函数的功能也不容易。具体而谈，有以下几点要求和提示：

1. 不管 length 的值是多少，每个共享内存分配的空间大小总是页大小的整数倍，请编写相关的代码对 length 进行规范化。
2. 本次 lab 不需要支持内存到磁盘的交换，可以把共享内存区域所在的物理页锁定在内存中。
3. 本次 lab 中比较重要的数据结构如 mm_struct，vm_area_struct,page, anon_vma,vm_operations_struct 等重要结构体在 include/linux/mm.h 或者 mm_types.h 中可以找到。请大家结合代码理解 linux 是如何通过 vma 链表/树结构来管理进程的地址空间的，并且需要理解 page, anon_vma 等数据结构构成的反向映射（reverse map）机制。需要大家自行预习操作系统关于页表映射和内存回收等一系列过程，熟悉 page fault 发生时操作系统到底做了哪些事情。可以通过 task_struct->mm 访问一个进程的一系列与内存管理相关的数据结构。



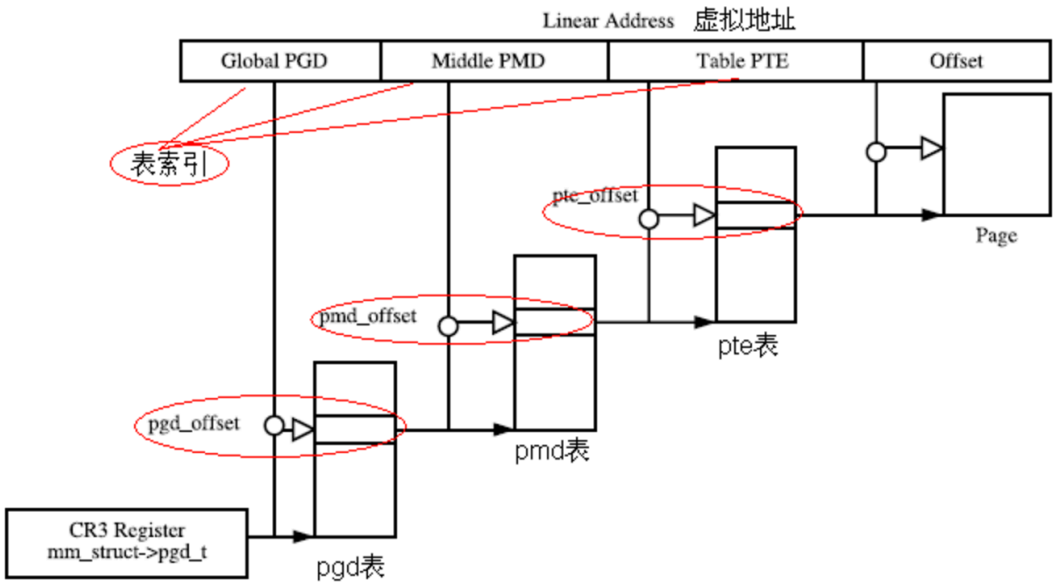
客观地，这张图对于 linux kernel 3.10 并不适合，anon_vma_node 被替换成了 anon_vma_chain，但是大致结构是相近的。纯粹为了帮助理解所用。

文末会推荐相关的博客，希望能够帮助各位更好地理解。

4. 用数组 ssmem data 保存每个 ssmem segment 的基本信息，例如：长度，引用这个区域的进程的数量，引用这个共享内存区域进程的 list(也类似于一个反向映射机制)，当前共享内存区域对应的

物理页 list 等。这个数组是“task independent”的，也就是我们希望它是全局性质的。所以当编号为 id 的 ssmem segment 被某进程创建或者需要被删除的时候，对应的数组中的相应信息也要被初始化或者清除。

5. 当把共享内存区域映射到一个进程空间时，创建一个新的 vma，代表特定的共享内存空间映射。所以，attach 时需要在进程的 vma 链表中创建一个新的 vma，哪怕该进程已经 map 过这个 ssmem segment 了。mm/mmap.c 里有一些函数，会对你学习如何处理 vma 有所帮助。请重点看一下 do_brk()，或者去网上搜索 do_mmap() 的代码，体会函数的处理过程。
6. 调用过 attach 后，注意创建的 vma 的权限问题。同时这段 vma 需要修改 vm_operations_struct,为 page fault 提供异常处理函数。当进程试图访问新创建的 vma 对应的内存共享段时，会触发 page fault，因为这个时候 pte 还没有与对应物理页建立联系。你的错误处理函数必须将正确的共享物理页映射到进程的 pte 中。如果这是第一次有进程试图访问这个物理页，它需要首先被分配。分配后的 page 地址需要被添加到对应数组成员的物理页面链表中。所以在 fault 中，为了方便知道该 vma 对应了哪个数组成员，可以把对应的指针存放到 vm_private_data 中，这样在 handler 中就可以直接访问到对应的数组成员，从而更新它的 page list。注意 page fault 发生后操作系统的一系列行为。



Page Table Layout

7. 当一个共享内存空间被释放时，在进程中删掉相关的 vma。假如这是最后一个映射到该共享内存的 vma，那在 detach 后，我们需要进行页面回收的操作，并且从我们维护的全局数组中删掉该共享内存空间对应的信息。请通过 vm_operations_struct 中的 close 函数实现，当进行 detach 操作的时候，过程可以参考 mmap. c/do_munmap 函数，注意清空 page table 和 tlb 等表项。
8. 因为我们直接在 ssmem_data 数组中维护了每个 ssmem segment 物理页的 list，所以在本题中我们不需要支持 swap。请思考一下原因。假如需要支持 swap，有能力的同学可以想一种合理的解决方式。
9. 在共享数据结构中需要采用同步机制，如 mmap，page tables, ssmem data structs。mmap 被 task_struct->mm->mmap_sem 保护，page table 被 page_table_lock 保护。因为 ssmem segment 对应的数组是全局性的，所以对其中的临界数据进行改动的时候需要

写者独占。请合理地使用信号量和锁。

更贴心的提示：

我们维护一个全局数组的作用就是方便进行共享内存信息的管理和检查，让其对程序编写者可见。数组成员的类型应该是各位自己定义的一个 struct，它能够描述一段共享内存的大小、是否是第一次创建、有多少个进程能够映射到它，映射到它的 vma 链表（想想应该利用什么数据结构实现？）、它对应的共享物理页面链表(初始为NULL)等等。同时，对于一个 map 到共享内存空间的 vma，我们也需要知道它到底对应了数组中哪一个元素。这些添加和修改需要大家自己实现。

Part II

写一个测试程序 ssmpipe(char* name, char* ssmem_id, char* type)。name 表示当前读者/写者的名字，ssmem_id 表示共享内存的 id，type 表示 reader/writer。每一个进程创建或者 attach 一个 ssmem segment，大小为 8k，2 pages。

要求：

写入共享内存的格式为：writername says: blabla

读共享内存时输出的格式为：readername: writername says: blabla

Tips：

1. 为了简单起见，本次 lab 可以只考虑一个 writer 和最多 8 个 reader。

2. 当 writer 在修改内存区域时 reader 不能读。若 writer 想修改共享内存区域时有 reader 在读该区域，则需要等待所有 reader 读完才修改。(否则它睡眠 1 秒后，周期性检查是否所有 reader 都已经读完)。请使用 atomic 操作增强安全性。
3. 输入输出示例：

```
$ ssmpipe 1 1 reader
$ ssmpipe 2 1 reader
$ ssmpipe 1 1 writer
hello, world!
reader1: writer1 says hello, world!
reader2: writer1 says hello, world!
```

三、 参考资料

几种数据结构的概述：
http://blog.sina.com.cn/s/blog_aed82f6f0101fg2u.html
do_brk()函数的分析：
http://2ndmoon.blog.51cto.com/6542214/1283728
do_munmap（）的分析：
http://blog.csdn.net/yunsongice/article/details/5637554
匿名页反向映射机制的实现：
http://www.cnblogs.com/tolimit/p/5398552.html
http://www.cnblogs.com/visayafan/archive/2011/12/24/2300758.html
Linux 中共享内存的实现：
http://blog.csdn.net/ljianhui/article/details/10253345