Linux Operating System Project 1

小組成員

郭同益、李晨安、呂健豪

Part1 : The kernel address space of all processes maps into the same physical address space.

How does the kernel maintains the above property?

當創建一個新的process時，kernel需要為其建立一新的PGD，並從kernel的page目錄swapper_pg_dir(存有低端物理內存的線性映射關係)複製kernel的page目錄項至新的PGD內，因此對於所有的process來說，在建立時便已將virtual address當中1GB（0xC000 0000－0xFFFF FFFF) 映射至kernel中，並稱為kernel space。

進程地址空間初始化: do_fork()->copy_process()->copy_mm()->dup_mmap()->mm_init()

```
if (!mm_init(mm, tsk, mm->user_ns))
    goto fail_nomem;
```

mm_init()在對部分成員初始化後便調用了mm_alloc_pgd()初始化自己的一級頁表

```
if (mm_alloc_pgd(mm))
    goto fail_nopgd;
```

mm_alloc_pgd只是封裝函數，我們該關注的是pgd_alloc()

_do_fork()->copy_process()->copy_mm()->mm_init()->mm_alloc_pgd()->pgd_alloc()

```
new_pgd = __pgd_alloc();
if (!new_pgd)
    goto no_pgd;

memset(new_pgd, 0, USER_PTRS_PER_PGD * sizeof(pgd_t));

/*
 * Copy over the kernel and IO PGD entries
 */
init_pgd = pgd_offset_k(0);
memcpy(new_pgd + USER_PTRS_PER_PGD, init_pgd + USER_PTRS_PER_PGD,
           (PTRS_PER_PGD - USER_PTRS_PER_PGD) * sizeof(pgd_t));
```

49行之pgd_offset_k(0)直接找到內核頁表存放的物理起始位置(PAGE_OFFSET + TEXT_OFFSET - PG_DIR_SIZE), 50行則根據得到的內核頁表起始位置，將內核頁表完整複製到自己的一級頁表中

pgd_offset_k(addr):获取addr地址对应pgd全局页表中的entry，而这个pgd全局页表正是swapper_pg_dir全局页表；

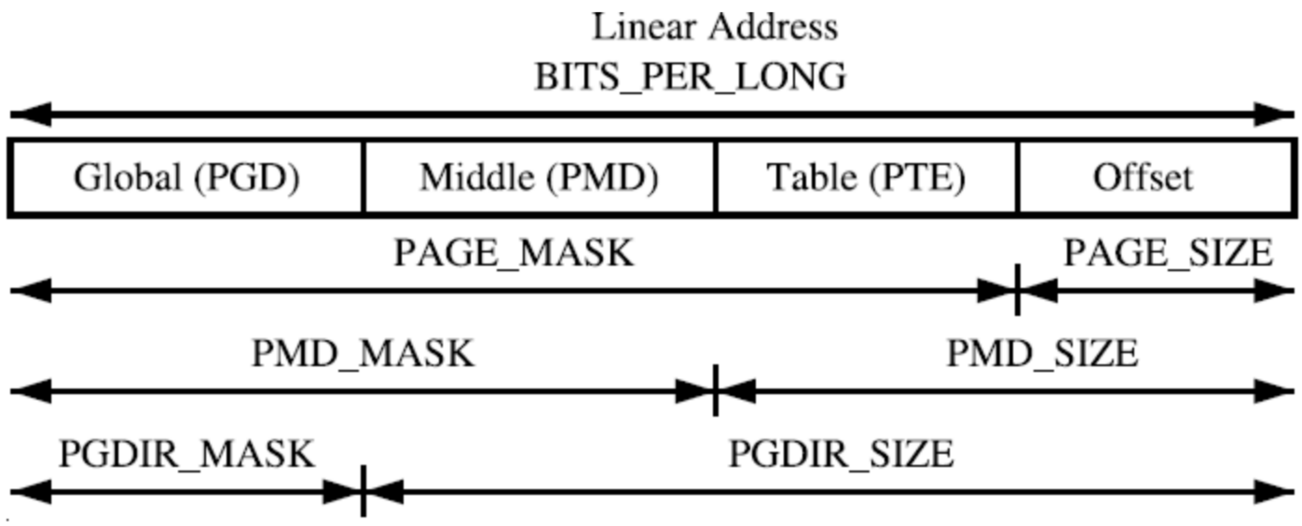
demo code

Linux Version: 14.04 kernel Version: 3.13.11

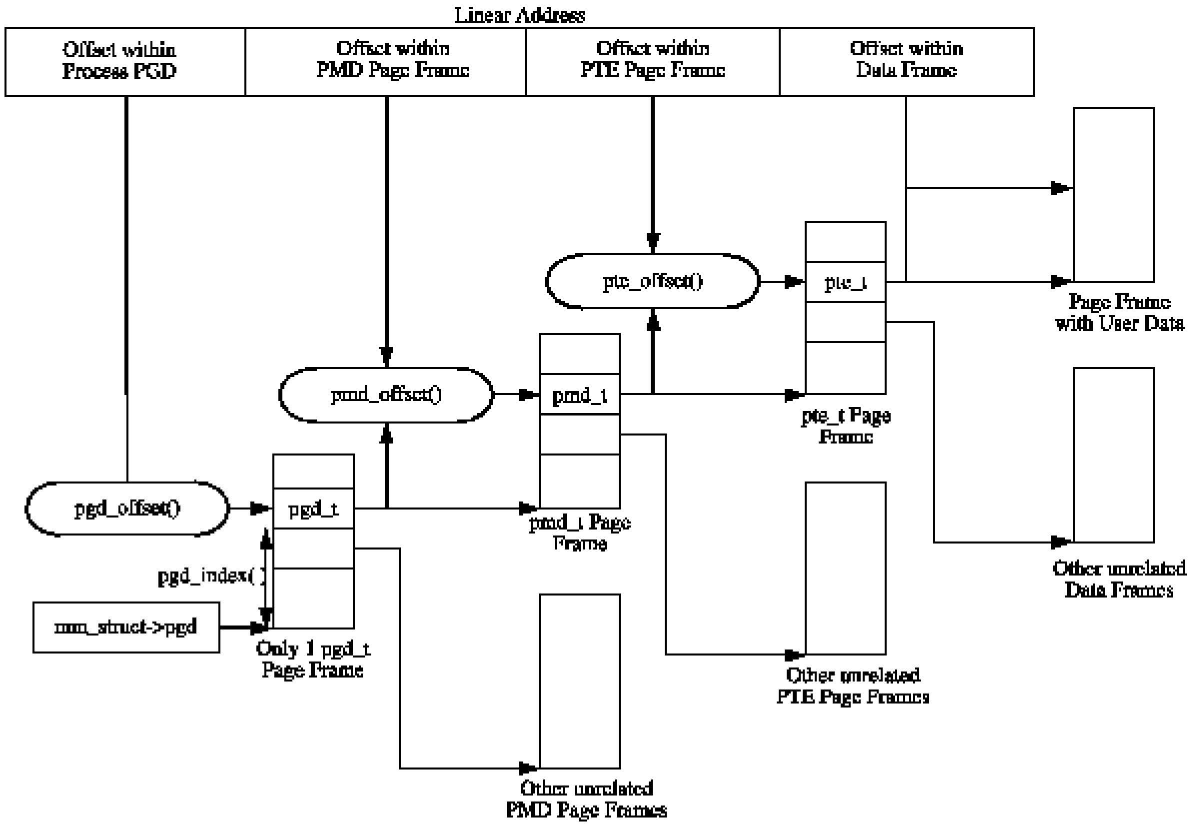
解釋

為了查詢所有process是否都指向同一個區塊的kernel，所以用for_each_process 走遍所有的process，由於kernel process的pgd所指向的位置為NULL，所以要排除掉 kernel process，接著因為我們所使用的為32bit，而kernel版本為3.X，所以查表只需要查四層即可，而中間則是少掉了pud跟p4d兩樣資料結構，而我們所使用的virtual address為 &init_task，因為這個virtual address必定存在，並不會找不到page去map。

虚拟地址MASK和SIZE宏图:



Linear Address Size and Mask Macros



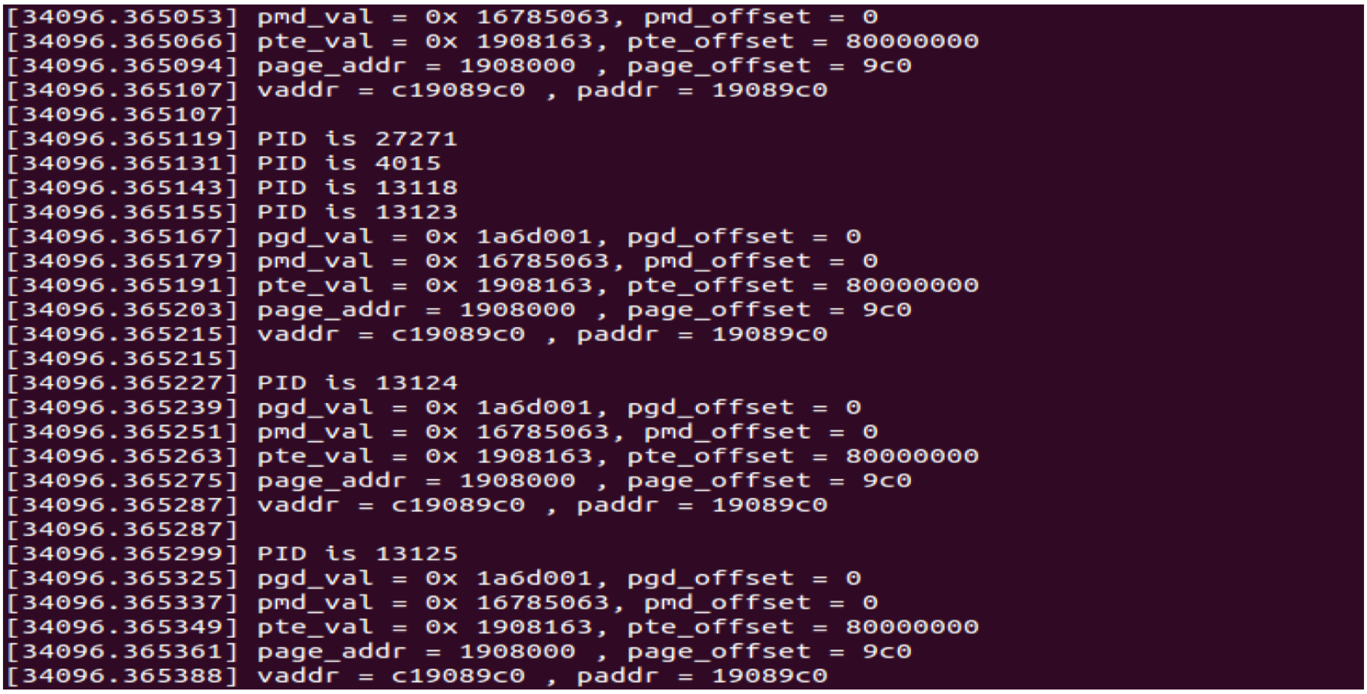

```
#include <linux/init.h>
#include <linux/kernel.h>
#include <linux/module.h>
#include <linux/mm.h>
#include <linux/mm_types.h>
#include <linux/sched.h>
#include <linux/export.h>

static unsigned long vaddr=0;

int vaddr2paddr(unsigned long vaddr){
struct task_struct *task;
    for_each_process(task){
    pgd_t *pgd;
    pmd_t *pmd;
    pte_t *pte;
    unsigned long paddr = 0;
    unsigned long page_addr = 0;
    unsigned long page_offset = 0;
    printk("PID is %d\n", task->pid);
    if(task->mm != NULL){
        pgd = pgd_offset(task->mm, vaddr);
        printk("pgd_val = 0x %lx, pgd_offset = %lx \n", pgd_val(*pgd), pgd);

        pmd = pmd_offset(pgd,vaddr);
        printk("pmd_val = 0x %lx, pmd_offset = %lx \n", pmd_val(*pmd), pmd);

        pte = pte_offset_kernel(pmd, vaddr);
        printk("pte_val = 0x %lx, pte_offset = %lx \n", pte_val(*pte), pte);
```

```
            page_addr = pte_val(*pte) & PAGE_MASK;
            page_offset = vaddr & ~PAGE_MASK;
            paddr = page_addr | page_offset;
            printk("page_addr = %lx , page_offset = %lx\n", page_addr , page_offset);
            printk("vaddr = %lx , paddr = %lx\n\n" , vaddr , paddr);
            }
    }
return 0;
}

static int __init v2p_init(void){
unsigned long vaddr = (unsigned long) &init_task;
int x=0;
printk("Module Running\n");
printk("get page vaddr=0x %lx \n", vaddr);
x=vaddr2paddr(vaddr);
return 0;
}

static void __exit v2p_exit(void){
printk("Module end\n");
}

module_init(v2p_init);
module_exit(v2p_exit);
```

we use kernel module to read pgd

demo result

kernel space中會有一塊位置是所有process都會指向



```
[34096.365053] pmd_val = 0x 16785063, pmd_offset = 0
[34096.365066] pte_val = 0x 1908163, pte_offset = 80000000
[34096.365094] page_addr = 1908000 , page_offset = 9c0
[34096.365107] vaddr = c19089c0 , paddr = 19089c0
[34096.365107]
[34096.365119] PID is 27271
[34096.365131] PID is 4015
[34096.365143] PID is 13118
[34096.365155] PID is 13123
[34096.365167] pgd_val = 0x 1a6d001, pgd_offset = 0
[34096.365179] pmd_val = 0x 16785063, pmd_offset = 0
[34096.365191] pte_val = 0x 1908163, pte_offset = 80000000
[34096.365203] page_addr = 1908000 , page_offset = 9c0
[34096.365215] vaddr = c19089c0 , paddr = 19089c0
[34096.365215]
[34096.365227] PID is 13124
[34096.365239] pgd_val = 0x 1a6d001, pgd_offset = 0
[34096.365251] pmd_val = 0x 16785063, pmd_offset = 0
[34096.365263] pte_val = 0x 1908163, pte_offset = 80000000
[34096.365275] page_addr = 1908000 , page_offset = 9c0
[34096.365287] vaddr = c19089c0 , paddr = 19089c0
[34096.365287]
[34096.365299] PID is 13125
[34096.365325] pgd_val = 0x 1a6d001, pgd_offset = 0
[34096.365337] pmd_val = 0x 16785063, pmd_offset = 0
[34096.365349] pte_val = 0x 1908163, pte_offset = 80000000
[34096.365361] page_addr = 1908000 , page_offset = 9c0
[34096.365388] vaddr = c19089c0 , paddr = 19089c0
```

Part2 : After a person uses Fix-Mapped Linear Addresses to map a 4K kernel address space to a 4k page frame, will existing processes get this

new mapping?

fixmap之理解

其中有一段虛擬地址用於固定對映，也就是fixed map。固定對映的線性地址(fix-mapped linear address)是一個固定的線性地址，它所對應的實體地址是人為強制指定的。每個固定的線性地址都對映到一塊實體記憶體頁。固定對映線性地址能夠對映到任何一頁實體記憶體，在x86架構下為從0xfffff000開始向低地址進行分配，在kernel初始化階段指定其對映的實體地址。

fixmap 初始化

early_fixmap_init 用於建立pgd/pmd表

```
static pte_t bm_pte[PTRS_PER_PTE] __page_aligned_bss;              /* 1 */
static pmd_t bm_pmd[PTRS_PER_PMD] __page_aligned_bss __maybe_unused;/* 1 */
static pud_t bm_pud[PTRS_PER_PUD] __page_aligned_bss __maybe_unused;/* 1 */

void __init early_fixmap_init(void)
{
    pgd_t *pgd;
    pud_t *pud;
    pmd_t *pmd;
    unsigned long addr = FIXADDR_START;                            /* 2 */

    pgd = pgd_offset_k(addr);
    if (pgd_none(*pgd))
        __pgd_populate(pgd, __pa_symbol(bm_pud), PUD_TYPE_TABLE);   /* 2 */
    pud = fixmap_pud(addr);
    if (pud_none(*pud))
        __pud_populate(pud, __pa_symbol(bm_pmd), PMD_TYPE_TABLE);   /* 2 */
    pmd = fixmap_pmd(addr);
    __pmd_populate(pmd, __pa_symbol(bm_pte), PMD_TYPE_TABLE);       /* 2 */

    BUILD_BUG_ON((__fix_to_virt(FIX_BTMAP_BEGIN) >> PMD_SHIFT)
             != (__fix_to_virt(FIX_BTMAP_END) >> PMD_SHIFT));       /* 3 */

    if ((pmd != fixmap_pmd(fix_to_virt(FIX_BTMAP_BEGIN)))
             || pmd != fixmap_pmd(fix_to_virt(FIX_BTMAP_END))) {    /* 3 */
        WARN_ON(1);
    }
}
```

early_ioremap_page_table_range_init()，這個函式用來建立固定記憶體對映區域的。

```
void __init early_ioremap_page_table_range_init(void)
{
    pgd_t *pgd_base = swapper_pg_dir;
    unsigned long vaddr, end;

```

```
    /*
     * Fixed mappings, only the page table structure has to be
     * created - mappings will be set by set_fixmap():
     */
    vaddr = __fix_to_virt(__end_of_fixed_addresses - 1) & PMD_MASK;
    end = (FIXADDR_TOP + PMD_SIZE - 1) & PMD_MASK;  //unsigned long __FIXADDR_TOP = 0xfffff000;
    page_table_range_init(vaddr, end, pgd_base);
    early_ioremap_reset();
}
```

demo code

```
check fixmap mapping

// 利用在fixmap建立空間
set_fixmap(idx, phys);

check fixmap mapping again for mapping we just make
```

/arch/x86/mm/pgtable.c

```
void __set_fixmap(enum fixed_addresses idx, phys_addr_t phys, pgprot_t flags)
{
    unsigned long address = __fix_to_virt(idx);

    if (idx >= __end_of_fixed_addresses)
        BUG();

    map_page(address, phys, pgprot_val(flags));
}
```

```
int map_page(unsigned long va, phys_addr_t pa, int flags)
{
    pmd_t *pd;
    pte_t *pg;
    int err = -ENOMEM;
    /* Use upper 10 bits of VA to index the first level map */
    pd = pmd_offset(pgd_offset_k(va), va);
    /* Use middle 10 bits of VA to index the second-level map */
    pg = pte_alloc_kernel(pd, va); /* from powerpc - pgtable.c */
    /* pg = pte_alloc_kernel(&init_mm, pd, va); */

    if (pg != NULL) {
        err = 0;
        set_pte_at(&init_mm, va, pg, pfn_pte(pa >> PAGE_SHIFT,
                __pgprot(flags)));
```

```
        if (unlikely(mem_init_done))
            _tlbie(va);
    }
    return err;
}
```

how does the kernel complete this mapping?

基於上面兩個部份的function，我們認為在fixed_map裡面使用set_fixmap()只會更新該process的fixed map並將其映射到physical address，而其他process中的page table並不會有這段映射存在，因此若使用fixed map不會讓其他process看到新創建的mapping。

reference

- Linux：页表中PGD、PUD、PMD等概念介绍
- Linux用戶進程創建過程淺析
- 記憶體知識梳理2. Linux 頁表的建立過程-x86
- Linux內存管理實踐-虛擬地址轉換物理地址
- 手動模擬Linux 內核mmu 內存尋址
- Linux kernel 内存 - 页表映射（SHIFT，SIZE，MASK）和转换(32位，64位)
- Page Table Management
- FIXMAP Allocator
- Fix-Mapped Addresses
- /arch/microblaze/mm/pgtable.c
-
- /arch/powerpc/mm/pgtable_32.c