Linux Kernel Project3 Report
Memory Management

蒋逸伟 517161910005

1 Requirement

Write a module `mtest`,when the module being loaded,it will create a proc file `mtest`.It will accept three parameters as follows:

- `listvma`

  It will print the whole virtual memory address of current process. The format is

  `start-addr end-addr permission`

- `findpage addr`

  Convert the virtual address of current process to physical address. If there is no translation print

  `translation not found`

- `writeval addr val`

  write a value to a designated address

2 Implement

2.1 Step1 Implement of the response of write a string to the proc file

```
static ssize_t mtest_proc_write(struct file *file,
const char __user * buffer,
size_t count, loff_t * data)
{
    unsigned long val1, val2;
    char *tmp = kzalloc((count+1),GFP_KERNEL); //Dynamicly alloc memory
    if(!tmp)return -ENOMEM;
    if(copy_from_user(tmp,buffer,count)){
        kfree(tmp);
            return EFAULT;
    }
    if (memcmp(tmp, "listvma", 7) == 0)
        mtest_list_vma();
    else if (memcmp(tmp, "findpage", 8) == 0) {
        if (sscanf(tmp + 8, "%lx", &val1) == 1) {
            mtest_find_page(val1);
        }
    }

    else if (memcmp(tmp, "writeval", 8) == 0) {
        if (sscanf(tmp + 8, "%lx %lx", &val1, &val2) == 2) {
            mtest_write_val(val, val2);
```

```
        }
    }
    return count;
}
```

2.2 Step2 Implement of listvma

The data structure VMA is defined in `include/linux/mm_types.h` The data member `vm_start` and the `vm_end` record the start address and the end address of the VMA in process space. `vm_flags` describes the VMA permission. The all VMAs of current process is organized by a link list,the `vm_next` and the `vm_prev` records the previous and the next VMA.

```
static void mtest_list_vma(void)
{
    struct mm_struct *mm = current->mm;
    struct vm_area_struct *vma;
    down_read(&mm->mmap_sem); //Protect the memory
    for (vma = mm->mmap;vma; vma = vma->vm_next) {
        printk("start-0x%lx end-0x%lx %c%c%c \n",vma->vm_start, vma->vm_end,
         vma->vm_flags & VM_READ ? 'r' : '-',
         vma->vm_flags & VM_WRITE ? 'w' : '-',
         vma->vm_flags & VM_EXEC ? 'x' : '-');
    }// Traverse the VMA link_list.
    up_read(&mm->mmap_sem);
}
```

2.3 Step3 Implement of findpage

Original x86-64 was limited by 4-level paing to 256 TiB of virtual address space and 64 TiB of physical address space. The 5-level paging is a straight-forward extension of the current page table structure adding one more layer of translation. It bumps the limits to 128 PiB of virtual address space and 4 PiB of physical address space. The 5 level paging type add a table named `p4d` to the origin page tables.

```
static void mtest_find_page(unsigned long addr)
{
        p4d_t *p4d;//The p4d is between the pgd and the pud.
        pud_t *pud;
        pmd_t *pmd;
        pgd_t *pgd;
        pte_t *pte;
        spinlock_t *ptl;
    unsigned long kernel_addr;
```

```
    struct mm_struct *mm = current->mm;
    down_read(&mm->mmap_sem);
    struct vm_area_struct *vma;
    vma = find_vma(mm, addr);
        struct page *page = NULL;
        pgd=pgd_offset(mm, addr);
        if(pgd_none(*pgd)||unlikely(pgd_bad(*pgd))){
                goto out;
        }
    p4d = p4d_offset(pgd,addr);
    if(p4d_none(*p4d)||unlikely(p4d_bad(*p4d))){
                goto out;
        }
        pud=pud_offset(p4d,addr);
        if(pud_none(*pud)||unlikely(pud_bad(*pud))){
                goto out;
        }
        pmd = pmd_offset(pud,addr);
        if(pmd_none(*pmd)||unlikely(pmd_bad(*pmd))){
                goto out;
        }
        pte = pte_offset_map_lock(mm,pmd,addr,&ptl);
        if(!pte)
                goto out;
        if(!pte_present(*pte))
                goto unlock;
        page = pfn_to_page(pte_pfn(*pte));
        if(!page)
                goto unlock;
        get_page(page);
unlock:
        pte_unmap_unlock(pte,ptl);
out:
    up_read(&mm->mmap_sem);
    if (!page)
        printk("Translation not Found.\n");
    else
        {kernel_addr=(unsigned long)page_address(page);
            kernel_addr+=(addr&~PAGE_MASK);
            printk("The physical address is 0x%lx\n",kernel_addr);
        }
```

```
}
```

2.4 Step 4 Implement of writeeval addr val

To write a value to a virtual address,should satisfy the following points:

- The address from the user space should between the vma->start and the vma->end.The addr+sizeof(val) should less than the vma->end.
- This vma is writable.
- Should translate the virtual address to the physical address.

Now create a translate function:

```
static struct page *find_physical_page(struct vm_area_struct *vma, unsigned long addr)
        pud_t *pud;
    p4d_t *p4d;
        pmd_t *pmd;
        pgd_t *pgd;
        pte_t *pte;
        spinlock_t *ptl;
        struct page *page = NULL;
        struct mm_struct *mm = vma->vm_mm;
        pgd=pgd_offset(mm, addr);
        if(pgd_none(*pgd)||unlikely(pgd_bad(*pgd))){
                goto out;
        }
    p4d = p4d_offset(pgd,addr);
    if(p4d_none(*p4d)||unlikely(p4d_bad(*p4d))){
                goto out;
        }
        pud=pud_offset(p4d,addr);
        if(pud_none(*pud)||unlikely(pud_bad(*pud))){
                goto out;
        }
        pmd = pmd_offset(pud,addr);
        if(pmd_none(*pmd)||unlikely(pmd_bad(*pmd))){
                goto out;
        }
        pte = pte_offset_map_lock(mm, pmd, addr,&ptl);
        if(!pte)
                goto out;
        if(!pte_present(*pte))
```

```
                goto unlock;
        page = pfn_to_page(pte_pfn(*pte));
        if(!page)
                goto unlock;
        get_page(page);
unlock:
        pte_unmap_unlock(pte, ptl);
out:
        return page;
}
```

Now for the write function:

```
static void mtest_write_val(unsigned long addr, unsigned long val)
{
    struct vm_area_struct *vma;
    struct mm_struct *mm = current->mm;
    struct page *page;
    unsigned long kernel_addr;
    down_read(&mm->mmap_sem);
    vma = find_vma(mm, addr);
    if (vma && addr >= vma->vm_start && (addr + sizeof(val)) < vma->vm_end) {
        if (!(vma->vm_flags & VM_WRITE)) {
            printk("This vma is not writable for 0x%lx\n", addr);
            goto out;
        }
        page = find_physical_page(vma,addr);
        if (!page) {
            printk("page not found  for 0x%lx\n", addr);
            goto out;
        }

        kernel_addr = (unsigned long)page_address(page);
        kernel_addr += (addr&~PAGE_MASK);
        *(unsigned long *)kernel_addr = val;
        put_page(page);
        printk("write 0x%lx to address 0x%lx\n", val, kernel_addr);}
        else {
        printk("no vma found for %lx\n", addr);
    }
out:
    up_read(&mm->mmap_sem);
```

```
}
```

2.5 Step 5 Create a proc file entry

```
static struct file_operations proc_mtest_operations = {
.write= mtest_proc_write
};
static struct proc_dir_entry *mtest_proc_entry;
static int __init mtest_init(void)
{
    mtest_proc_entry = proc_create("mtest", 0777, NULL,&proc_mtest_operations);
    if (mtest_proc_entry == NULL) {
        printk("Error creating proc entry/n");
        return -1;
    }
    printk("create the filename mtest mtest_init sucess\n");
    return 0;
}
static void __exit mtest_exit(void)
{
    printk("exit the module......mtest_exit\n");
    remove_proc_entry("mtest", NULL);
}
```

3 Result

3.1 listvma

The command is

`echo "listvma">/proc/mtest`

`dmesg`

The result shows the vma of current process.

3.2 findpage

The command is

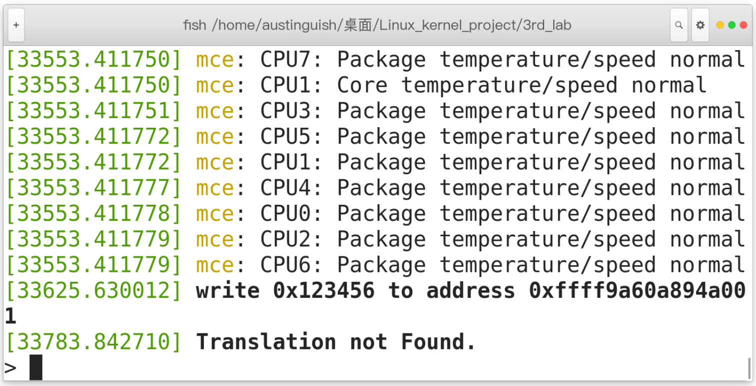
`echo "findpage0x7f9b47503001">/proc/mtest`

`dmesg`

```
fish /home/austinguish/桌面/Linux_kernel_project/3rd_lab
[33127.686015] start-0x7f9b47502000 end-0x7f9b47503000 rw-
[33127.686015] start-0x7f9b47503000 end-0x7f9b47504000 rw-
[33127.686016] start-0x7ffce2dc2000 end-0x7ffce2df1000 rw-
[33127.686016] start-0x7ffce2dfc000 end-0x7ffce2dff000 r--
[33127.686017] start-0x7ffce2dff000 end-0x7ffce2e00000 r-x
[33440.683882] The physical address is 0xffff9a60cdc44001
>
```

If failed (the address is not for current address):

`echo "findpage0x7f9b475">/proc/mtest`



```
fish /home/austinguish/桌面/Linux_kernel_project/3rd_lab
[33553.411750] mce: CPU7: Package temperature/speed normal
[33553.411750] mce: CPU1: Core temperature/speed normal
[33553.411751] mce: CPU3: Package temperature/speed normal
[33553.411772] mce: CPU5: Package temperature/speed normal
[33553.411772] mce: CPU1: Package temperature/speed normal
[33553.411777] mce: CPU4: Package temperature/speed normal
[33553.411778] mce: CPU0: Package temperature/speed normal
[33553.411779] mce: CPU2: Package temperature/speed normal
[33553.411779] mce: CPU6: Package temperature/speed normal
[33625.630012] write 0x123456 to address 0xffff9a60a894a00
1
[33783.842710] Translation not Found.
>
```

It shows the physical address of the va 0x7f9b47503001 is 0xffff9a60cdc44001

3.3 writeval

The command is

`echo "writeval0x7f9b474b3001 123456">/proc/mtest`

`dmesg`

```
fish /home/austinguish/桌面/Linux_kernel_project/3rd_lab
[33553.411749] mce: CPU5: Core temperature/speed normal
[33553.411750] mce: CPU7: Package temperature/speed normal
[33553.411750] mce: CPU1: Core temperature/speed normal
[33553.411751] mce: CPU3: Package temperature/speed normal
[33553.411772] mce: CPU5: Package temperature/speed normal
[33553.411772] mce: CPU1: Package temperature/speed normal
[33553.411777] mce: CPU4: Package temperature/speed normal
[33553.411778] mce: CPU0: Package temperature/speed normal
[33553.411779] mce: CPU2: Package temperature/speed normal
[33553.411779] mce: CPU6: Package temperature/speed normal
[33625.630012] write 0x123456 to address 0xffff9a60a894a00
1
>
```

If the the vma is not writable:

```
fish /home/austinguish/桌面/Linux_kernel_project/3rd_lab
> dmesg | tail
[33553.411751] mce: CPU3: Package temperature/speed normal
[33553.411772] mce: CPU5: Package temperature/speed normal
[33553.411772] mce: CPU1: Package temperature/speed normal
[33553.411777] mce: CPU4: Package temperature/speed normal
[33553.411778] mce: CPU0: Package temperature/speed normal
[33553.411779] mce: CPU2: Package temperature/speed normal
[33553.411779] mce: CPU6: Package temperature/speed normal
[33625.630012] write 0x123456 to address 0xffff9a60a894a00
1
[33783.842710] Translation not Found.
[33940.453039] This vma is not writable for 0x7f9b474d4000
>
```

4 Conclusion

- In this lab,I learned the 5-level paging address method,the p4d table is inserted between the pgd and the pud.To convert the address, I should know their mapping relations.
- I read the vma structure carefully and learned the data member of this structure.It records the key info of a process.The vma is an advanced abstract of the physical address.A process can alloc or free a memory more efficiently.
- Learned the spinlock and the semaphore.The macro pte_offset_map and the pte_unmap are in pairs.It can be used to protect the memory.At alloc_zeroed_user_highpage,other thread may update or set the page.