Shanghai Jiao Tong University

CS353 Linux Kernel

Project 3: Memory Management

Chao Gao

5142029014

May 5, 2017

1 2B_optional

Before the project 3, I will first talk about the project 2B optional part because I did not submit this part last time.
The requirement is that we should implement a small tool for displaying the ctx of all runing processes dynamicly.
My thought is first writing a module to show all the running processes' name, pid and ctx. The whole framework is similar to the module 3 of project 2A, so I just show core part here.

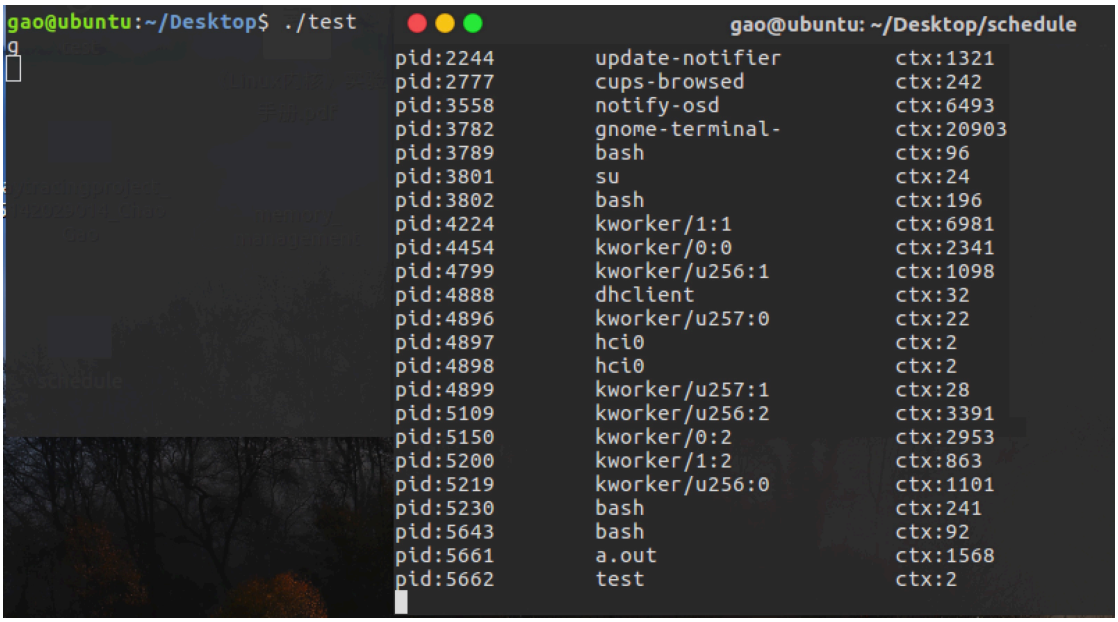
```
1  struct task_struct *task, *p;
2  struct list_head *pos;
3
4  seq_printf(m, "module init\n");
5
6  task = &init_task;
7  list_for_each(pos, &task->tasks)
8  {
9      p = list_entry(pos, struct task_struct, tasks);
10     seq_printf(m, "pid:%-10d\t%-16s\tctx:%-15d\n",p -> pid,
11     p -> comm, p -> ctx);
12 }
```

After make and insert the module. I simply write a program **user.c** to read the content of the proc file which contains the information of running processes and display it on the terminal, and this operation will execute every specific time to update the screen.

```
1  FILE *fp;
2  while(1)
3  {
4      char buf[1024];
5      fp=fopen("/proc/ctx_proc","r");
6      if(fp==NULL)
```

```
7       {
8           perror("fopen");
9           return -1;
10      }
11      while(!feof(fp))
12      {
13          if(fgets(buf,sizeof(buf),fp)!=NULL)
14          {
15              printf("%s",buf);
16              fflush(stdout);
17          }
18      }
19      fflush(stdout);
20      sleep(2);
21      printf("\033c");
22  }
```

Here I just simply show the result.



All the implementations can be found in the **2B_optional** folder.

2 Project description

Recently, we have learned something about memory management. First, let me recap the project 3 requirement.
Write a module that is called mtest.
When module loaded, module will create a proc fs entry /proc/mtest.
/proc/mtest will accept 3 kinds of input.

1. "listvma" will print all vma of current processes in the format of start-addr end-addr permission.

2. "findpage addr" will find va->pa translation of address in current process's mm context and print it. If there is not va->pa translation, print "translation not found".

3. "writeval addr val" will change an unsigned long size content in current process's virtual address into val. Note module should write to identity mapping address of addr and verify it from userspace address addr.

3 Environment

Virtual OS: Ubuntu 16.04.
kernel version: 4.9.13.

4 Implementation

Here I will present the whole implementation of this porject.

4.1 listvma

In this part, we should get all virtual memory address of current process.
struct task_struct is the process descriptor, **mm** field records the information of process VMA.

```
1 struct task_struct {
```

```
2       ...
3       struct mm_struct *mm;
4       ...
5   };
```

struct mm_struct is the memory descriptor.

```
1   struct mm_struct {
2       struct vm_area_struct *mmap; /* list of VMAs */
3       struct rb_root mm_rb;
4       ...
5       unsigned long mmap_base;      /* base of mmap area */
6       unsigned long mmap_legacy_base;
7       unsigned long task_size;       /* size of task vm space*/
8       unsigned long highest_vm_end; /* highest vma end address*/
9       pgd_t * pgd;
10      ...
11  };
```

struct vm_area_struct defines a memory VMM memory area.

```
1   struct vm_area_struct {
2       unsigned long vm_start;     /* start address */
3       unsigned long vm_end;       /* end address */
4
5       struct vm_area_struct *vm_next, *vm_prev;
6       struct rb_node vm_rb;
7
8       ...
9       unsigned long vm_flags;
10      ...
11  };
```

vm_flags means the permission.

```
#define VM_READ         0x0001
#define VM_WRITE        0x0002
#define VM_EXEC         0x0004
#define VM_SHARED       0x0008
```

After getting to know this background knowledge, I can simply do the implementation.

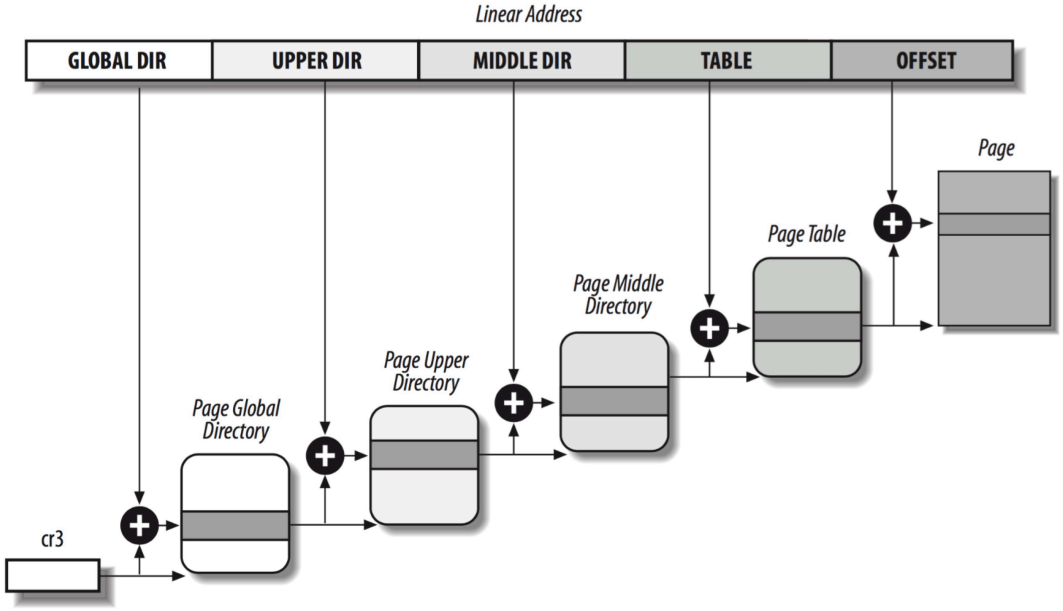
```
struct mm_struct *mm = current->mm;
struct vm_area_struct *vma;
char a[4] = {'-', '-', '-'};
int counter = 0;
down_read(&mm->mmap_sem); //lock
printk("mtest_list_vma:\n");
for (vma = mm->mmap;vma; vma = vma->vm_next)
{
    counter++;

    if (vma->vm_flags & VM_READ)    a[0] = 'r';
    else   a[0] = '-';
    if (vma->vm_flags & VM_WRITE)   a[1] = 'w';
    else   a[1] = '-';
    if (vma->vm_flags & VM_EXEC)    a[2] = 'x';
    else   a[2] = '-';
    printk("%d  0x%lx 0x%lx %c%c%c\n", counter, vma->
    vm_start, vma->vm_end, a[0], a[1], a[2]);
}
up_read(&mm->mmap_sem); //release lock
```

4.2 findpage addr

A simple firgure showing 4-level paging in linux.

Paging in Linux



In order to translate virtual memory address to physical address, first get the page with the VMA.

```
1  pgd_t *pgd;   //top level page table
2  pud_t *pud;   //second level page table
3  pmd_t *pmd;   //third level page table
4  pte_t *pte;   //last level page table
5  spinlock_t *ptl;
6
7  struct page *page = NULL;
8  struct mm_struct *mm = vma->vm_mm;
9
10 pgd = pgd_offset(mm, addr);
11 if(pgd_none(*pgd) || unlikely(pgd_bad(*pgd)))return NULL;
12
13 pud = pud_offset(pgd, addr);
```

```
14  if(pud_none(*pud) || unlikely(pud_bad(*pud)))return NULL;
15
16  pmd = pmd_offset(pud, addr);
17  if(pmd_none(*pmd) || unlikely(pmd_bad(*pmd)))return NULL;
18
19  pte = pte_offset_map_lock(mm, pmd, addr, &ptl);
20  if(!pte)  return NULL;
21
22  if(!pte_present(*pte))
23  {
24      pte_unmap_unlock(pte, ptl);
25      return NULL;
26  }
27
28  page = pfn_to_page(pte_pfn(*pte));
29  if(!page)
30  {
31      pte_unmap_unlock(pte, ptl);
32      return NULL;
33  }
34
35  get_page(page);
36  pte_unmap_unlock(pte, ptl);
37  return page;
```

After getting the page address, then we just add the offset to the page address. Now we get the physical address.

```
1  kernel_addr = (unsigned long)page_address(page);
2  kernel_addr += (addr & ~PAGE_MASK);
```

4.3 writeval addr val

In this part, we are required to change an unsigned long size content in current process's virtual address into val.
Actually this part is mainly based on part 2, so this part can be easily done after konwing how to translate virtual address to physical address in part 2.

```
vma = find_vma(mm, addr);
if (vma && addr >= vma->vm_start && (addr + sizeof(val))
                < vma->vm_end)
{
    if (!(vma->vm_flags & VM_WRITE))
    {
        printk("vma is not writable for 0x%lx\n", addr);
        up_read(&mm->mmap_sem);
        return;
    }
    page = mtest_seek_page(vma, addr);
    if (!page)
    {
        printk("Page 0x%lx not found\n", addr);
        up_read(&mm->mmap_sem);
        return;
    }

    kernel_addr = (unsigned long)page_address(page);
    kernel_addr += (addr&~PAGE_MASK);
    *(unsigned long *)kernel_addr = val;
    printk("Written 0x%lx to address 0x%lx\n", val, kernel_addr);

    put_page(page);
}
```

5 Result

Creating a writable proc file is similar to the Project 2A. So here I directly show the result.

type **echo "listvma" > /proc/mtest**.

type **dmesg** to get the following result.

```
[38357.513657] Succeed creating proc entry
[38666.448583] mtest_list_vma:
[38666.448586] 1  0x400000 0x4f4000 r-x
[38666.448587] 2  0x6f3000 0x6f4000 r--
[38666.448587] 3  0x6f4000 0x6fd000 rw-
[38666.448587] 4  0x6fd000 0x703000 rw-
[38666.448588] 5  0x108e000 0x10f7000 rw-
[38666.448588] 6  0x7fdf78812000 0x7fdf7881d000 r-x
[38666.448589] 7  0x7fdf7881d000 0x7fdf78a1c000 ---
[38666.448589] 8  0x7fdf78a1c000 0x7fdf78a1d000 r--
[38666.448590] 9  0x7fdf78a1d000 0x7fdf78a1e000 rw-
[38666.448591] 10  0x7fdf78a1e000 0x7fdf78a24000 rw-
[38666.448591] 11  0x7fdf78a24000 0x7fdf78a2f000 r-x
[38666.448592] 12  0x7fdf78a2f000 0x7fdf78c2e000 ---
[38666.448597] 13  0x7fdf78c2e000 0x7fdf78c2f000 r--
[38666.448597] 14  0x7fdf78c2f000 0x7fdf78c30000 rw-
[38666.448598] 15  0x7fdf78c30000 0x7fdf78c46000 r-x
[38666.448598] 16  0x7fdf78c46000 0x7fdf78e45000 ---
[38666.448599] 17  0x7fdf78e45000 0x7fdf78e46000 r--
[38666.448599] 18  0x7fdf78e46000 0x7fdf78e47000 rw-
[38666.448600] 19  0x7fdf78e47000 0x7fdf78e49000 rw-
[38666.448600] 20  0x7fdf78e49000 0x7fdf78e51000 r-x
[38666.448601] 21  0x7fdf78e51000 0x7fdf79050000 ---
[38666.448601] 22  0x7fdf79050000 0x7fdf79051000 r--
[38666.448602] 23  0x7fdf79051000 0x7fdf79052000 rw-
```

type **echo "findpage 0x108e000" > /proc/mtest**.

type **dmesg** to get the following result.

```
[38666.448611] 40  0x7fdf79d3c000 0x7fdf79d43000 r--
[38666.448612] 41  0x7fdf79d43000 0x7fdf79d45000 rw-
[38666.448612] 42  0x7fdf79d45000 0x7fdf79d46000 r--
[38666.448613] 43  0x7fdf79d46000 0x7fdf79d47000 rw-
[38666.448614] 44  0x7fdf79d47000 0x7fdf79d48000 rw-
[38666.448614] 45  0x7fffceb81000 0x7fffceba3000 rw-
[38666.448615] 46  0x7fffcebdf000 0x7fffcebe1000 r--
[38666.448616] 47  0x7fffcebe1000 0x7fffcebe3000 r-x
[39191.805897] mtest_find_page:
[39191.805900] Translate 0x108e000 to kernel address 0xffff8ce10781d000
```

type **echo "writeval 0x108e000 12345" > /proc/mtest**
type **dmesg** to get the following result.

```
[39191.805897] mtest_find_page:
[39191.805900] Translate 0x108e000 to kernel address 0xffff8ce10781d000
[39379.191008] mtest_write_val:
[39379.191010] Written 0x12345 to address 0xffff8ce10781d000
```

6 Summary

Although understanding linux memory management is a hard job, I did not give up and really learned a lot. And the usage of some functions required me to read the source code in linux kernel, which helped me get to know its architecture and how it is implemented.

Due to time limitation, I have not finished the optional part yet. And I will submit my code and report file about the optional part in next assignment submission.

Thanks a lot to Prof.Chen for impressive teaching about memory management and TAs for answering questions..