운영체제

Assignment4

김태석 교수님

2019202103

이은비

Assignment 4-1

<introduction>

프로세스의 가상 메모리에는 파일에서 불러온 shared library와 같은 여러정보가 적재되어 있습니다.이때 하나의 프로세스에 대하여, 정보가 위치하는 가상 메모리 주소, 프로세스의 데이터주소,코드주소, 힙주소, 정보의 원본파일의 전체 경로를 mem(...~...)code(...~...)data(...~...) heap(...~...) /home/....과 같은 형식으로 출력하게 합니다,

<conclusion>

강의자료 Assignment4 QnA를 참고하여서 강의자료에 vm_area_struct를 이용하여 Kernelinfo를 출력하게 합니다. 또힌 task_struct->mm_struct->vm_area_structure 순으로 t->mm->p_mmap으로 각각의 필요한 정보내용을 출력합니다.

```
File  Edit  View  Search  Terminal  Help
#include <linux/module.h>
#include <linux/kallsyms.h>
#include <linux/sched/mm.h>
#include <linux/syscalls.h>
#include <asm/syscall_wrapper.h>

#define __NR_ftrace 336

void **syscall_table;
char buf[100]={0};

asmlinkage pid_t (*real_ftrace)(struct pt_regs *regs);

__SYSCALL_DEFINEx(1,info,pid_t,pid)
{
        struct task_struct *t;
        struct mm_struct *mm;
        struct vm_area_struct *p_mmap;
        t = pid_task(find_vpid(pid),PIDTYPE_PID);
        mm = get_task_mm(t);
        p_mmap = mm->mmap;
        printk(KERN_INFO"############### LOADED FILES OF A PROCESS'a.out(%d)' in VM ##########\n",pid);

        printk(KERN_INFO "mem(%8x~%8x) code(%8x~%8x) data(%8x~%8x) heap(%8x~%8x) %s",p_mmap->vm_start,p_mma
p->vm_end,t->mm->start_code,t->mm->end_code,t->mm->start_data,t->mm->end_data,t->mm->start_brk,t->mm->brk,d
entry_path_raw((p_mmap->vm_file->f_path.dentry),buf,100));
        mmput(mm);
}
```

추가적으로 wrapping을 위한 함수를 작성하고 그 함수들 안에 접근권한을 변경 할 수 있도록 접근권한을 바꿔주는 함수도 추가적으로 작성합니다.

```
File  Edit  View  Search  Terminal  Help
//for permission
void make_ro(void *addr){
        unsigned int level;
        pte_t *pte = lookup_address((u64)addr, &level);
        pte->pte = pte->pte &~ _PAGE_RW;
}
//for permission
void make_rw(void *addr)
{
        unsigned int level;
        pte_t *pte = lookup_address((u64)addr, &level);
        if(pte->pte &~ _PAGE_RW)
                pte->pte |= _PAGE_RW;
}
//for wrapping
static int __init file_init(void)
{
        syscall_table = (void**) kallsyms_lookup_name("syscall_table");
        make_rw(syscall_table);//permission change->rw
        real_ftrace = syscall_table[__NR_ftrace];
        syscall_table[__NR_ftrace] = __x64_sysinfo;
        return 0;
}
//for wrapping
static void __exit file_exit(void)
{
        syscall_table[__NR_ftrace] = real_ftrace;
        make_ro(syscall_table);
}
module_init(file_init);
module_exit(file_exit);
                                                                    29,1
```

이후의 강의자료의 있는것처럼 sudo insmod file_varea.ko와 같은 kernel object파일을 생성하게 하고 이후에는 test.c를 컴파일 이후 실행한뒤 dmesg를 통해 강의자료에 나온 것 처럼 나오기를 기대했으나 마지막 부분에는 강의자료와 비슷한 내용을 출력하였지만 그위에 원치않은 많은 정보들이 나왔습니다.

```
os2019202103@ubuntu:~/4-1$ sudo insmod file_varea.ko
insmod: ERROR: could not insert module file_varea.ko: File exists
os2019202103@ubuntu:~/4-1$ gcc test.c
os2019202103@ubuntu:~/4-1$ ./a.out
os2019202103@ubuntu:~/4-1$ dmesg
[    0.000000] Linux version 4.19.672019202103 (os2019202103@ubuntu) (gcc version 5.4.0 20160609 (Ubuntu 5.
4.0-6ubuntu1~16.04.10)) #3 SMP Mon Sep 27 12:27:55 PDT 2021
[    0.000000] Command line: BOOT_IMAGE=/boot/vmlinuz-4.19.672019202103 root=UUID=67150264-f887-4339-9ab4-6
8b8da44a3d9 ro find_preseed=/preseed.cfg auto noprompt priority=critical locale=en_US quiet
[    0.000000] KERNEL supported cpus:
[    0.000000]   Intel GenuineIntel
[    0.000000]   AMD AuthenticAMD
[    0.000000]   Centaur CentaurHauls
[    0.000000] Disabled fast string operations
[    0.000000] x86/fpu: Supporting XSAVE feature 0x001: 'x87 floating point registers'
[    0.000000] x86/fpu: Supporting XSAVE feature 0x002: 'SSE registers'
[    0.000000] x86/fpu: Supporting XSAVE feature 0x004: 'AVX registers'
[    0.000000] x86/fpu: Supporting XSAVE feature 0x020: 'AVX-512 opmask'
[    0.000000] x86/fpu: Supporting XSAVE feature 0x040: 'AVX-512 Hi256'
[    0.000000] x86/fpu: Supporting XSAVE feature 0x080: 'AVX-512 ZMM_Hi256'
```

```
[35351.012105] ############## LOADED FILES OF A PROCESS'a.out(12213)' in VM ##########
[35351.012107] mem(  400000~  401000) code(  400000~  40074c) data(  600e10~  601040) heap(  faa000~  faa00
0) /home/os2019202103/4-1/a.out
[35693.341412] ############## LOADED FILES OF A PROCESS'a.out(13618)' in VM ##########
[35693.341414] mem(  400000~  401000) code(  400000~  40074c) data(  600e10~  601040) heap( 14ac000~ 14ac00
0) /home/os2019202103/4-1/a.out
[35892.201821] ############## LOADED FILES OF A PROCESS'a.out(14093)' in VM ##########
os2019202103@ubuntu:~/4-1$
```

<references>

리눅스 vm_area_structure/https://showx123.tistory.com/92

강의자료/os/assignment4 QnA

Assignment 4-2

<introduction>

Dynamic recompile하는 과정으로 실습과 같은 virtual machine의 특징으로 실행하는 동안에 프로그램의 한 부분을 recompile하는 것입니다. 실습에서는 Shared memory에서 컴파일된 코드에 접근합니다. Code section의 함수를 수정하여 함수를 복사 한뒤 복사된 함수를 최적화 합니다. Operation이 중복으로 나오면 중복을 합쳐서 표현하는 방식의 최적화를 사용합니다. 이후에 수정한 함수를 실행하는 과정을 거칩니다.

<conclusion>

objdump를 뜨는 과정과 dump뜬 화면 입니다. File format은 elf64- x86-64입니다

```
os2019202103@ubuntu:~/4-2$ gcc -c D_recompile_test.c
os2019202103@ubuntu:~/4-2$ objdump -d D_recompile_test.o >TEST
os2019202103@ubuntu:~/4-2$ cat TEST

D_recompile_test.o:     file format elf64-x86-64


Disassembly of section .text:

0000000000000000 <Operation>:
   0:   55                      push   %rbp
   1:   48 89 e5                mov    %rsp,%rbp
   4:   89 7d fc                mov    %edi,-0x4(%rbp)
   7:   8b 55 fc                mov    -0x4(%rbp),%edx
   a:   89 d0                   mov    %edx,%eax
   c:   83 c0 01                add    $0x1,%eax
   f:   83 c0 01                add    $0x1,%eax
  12:   6b c0 02                imul   $0x2,%eax,%eax
  15:   6b c0 04                imul   $0x4,%eax,%eax
  18:   83 c0 01                add    $0x1,%eax
  1b:   83 c0 01                add    $0x1,%eax
  1e:   83 c0 02                add    $0x2,%eax
  21:   83 c0 03                add    $0x3,%eax
  24:   83 c0 01                add    $0x1,%eax
  27:   83 c0 02                add    $0x2,%eax
  2a:   83 c0 01                add    $0x1,%eax
  2d:   83 c0 01                add    $0x1,%eax
  30:   6b c0 02                imul   $0x2,%eax,%eax
  33:   6b c0 02                imul   $0x2,%eax,%eax
  36:   6b c0 02                imul   $0x2,%eax,%eax
  39:   83 c0 01                add    $0x1,%eax
  3c:   83 c0 01                add    $0x1,%eax
  3f:   83 c0 03                add    $0x3,%eax
```

이후의 최적화를 위해서 add,imul,div,sub의 0x83,0x6b와 같은 instrunction의 해당하는 opcode를 찾았으며 disassembly화면에서 볼 수 없는 div,sub는 인터넷에서 해당하는 bit수에 따라서 각각의 값을 추가적으로 구하였습니다. 또한 assingment4 강의 자료에 나와있는 예시에 따라서 1값을 더하는 add instruction일 때, 2를 곱하는 imul instruction...에 해당하게 작성하였습니다.

```
        if(func[i] == 0x83)//add

        afunc[0]=func[i];
        afunc[1]=func[i+1];
        afunc[2]=0x01;
        a=1;
        while(1)
        {
                i=i+3;//83 c0 01
                if(func[i] != afunc[0] || func[i+1] != afunc[1] || func[i+2] != afunc[2])
                break;
                else
                a++;
        }
        compiled_code[j]        =afunc[0];
        compiled_code[j+1]      =afunc[1];
        compiled_code[j+2]      =a;
        j=j+3;
        continue;

         else if(func[i]== 0x6b)//imul

                mfunc[0]=func[i];
                mfunc[1]=func[i+1];
                mfunc[2]=0x02;
                m=1;

        while(1)
        {
                i=i+3;
                if(func[i] != mfunc[0] || func[i+1] != mfunc[1] || func[i+2] != mfunc[2])
                break;
                else
                m++;
                                                                107,1-8       60%
```

최종적으로 최적화 하기 전 과 최적화 한 후의 결과를 확인하였는데 처음에는 sec값만 적용하려 하니 둘다 0으로 차이가 없어서 추가적으로 sec,nsec즉 nanosecond까지 변수를 추가하여서 작성하였습니다.

```
os2019202103@ubuntu:~/4-2$ sync
os2019202103@ubuntu:~/4-2$ echo 3 | sudo tee /proc/sys/vm/drop_caches
[sudo] password for os2019202103:
3
os2019202103@ubuntu:~/4-2$ ./test2
Data was filled to shared memory.
os2019202103@ubuntu:~/4-2$ make default
gcc -o test2 D_recompile_test.c
gcc -o drecompile D_recompile.c -lrt
os2019202103@ubuntu:~/4-2$ ./drecompile
result : 158113
total execution time : 0.154125
os2019202103@ubuntu:~/4-2$ sync
os2019202103@ubuntu:~/4-2$ echo 3 | sudo tee /proc/sys/vm/drop_caches
3
os2019202103@ubuntu:~/4-2$ make dynamic
gcc -o test2 D_recompile_test.c
gcc -Drecmp -o drecompile.c D_recompile.c -lrt
os2019202103@ubuntu:~/4-2$ sync
os2019202103@ubuntu:~/4-2$ echo 3 | sudo tee /proc/sys/vm/drop_caches
3
os2019202103@ubuntu:~/4-2$ ./test2
Data was filled to shared memory.
os2019202103@ubuntu:~/4-2$ make dynamic
gcc -o test2 D_recompile_test.c
gcc -Drecmp -o drecompile.c D_recompile.c -lrt
os2019202103@ubuntu:~/4-2$ ./drecompile
result : 158113
total execution time : 0.111260
os2019202103@ubuntu:~/4-2$
```

<references>

Div opcode/https://www.felixcloutier.com/x86/div

Sub opcode/https://pdos.csail.mit.edu/6.828/2005/readings/i386/SUB.htm

강의자료/os/assignment4