Санкт-Петербургский Национальный Исследовательский Университет

Информационных Технологий, Механики и Оптики

Факультет программной инженерии и компьютерной техники

Домашнее задание №2

по дисциплине

«Операционные системы»

Вариант :   
syscall: pci\_dev, memblock\_type

Выполнил:

Студент группы P33101,

Патутин Владимир

Преподаватель:

Осипов Святослав Владимирович

Санкт-Петербург

2021г.

**Текст задания:**   
Разработать комплекс программ на пользовательском уровне и уровне ярда, который собирает информацию на стороне ядра и передает информацию на уровень пользователя, и выводит ее в удобном для чтения человеком виде. Программа на уровне пользователя получает на вход аргумент(ы) командной строки (не адрес!), позволяющие идентифицировать из системных таблиц необходимый путь до целевой структуры, осуществляет передачу на уровень ядра, получает информацию из данной структуры и распечатывает структуру в стандартный вывод. Загружаемый модуль ядра принимает запрос через указанный в задании интерфейс, определяет путь до целевой структуры по переданному запросу и возвращает результат на уровень пользователя.

**Выполнение:**

**Сначала создадим директории memblock\_type и pci\_dev:**

1. В директории pci\_dev создадим файл pci\_dev.c :

#include <linux/kernel.h>

#include <linux/syscalls.h>

#include <linux/pci.h>

struct pci\_dev\* dev;

SYSCALL\_DEFINE0(pci\_dev)

{

while ((dev = pci\_get\_device(PCI\_ANY\_ID, PCI\_ANY\_ID, dev))) {

printk(KERN\_INFO "pci devide found: pci\_id = %d\n", dev->device);

}

return 0;

}

1. В директории pci\_dev создадим файл Makefile:

obj-y := pci\_dev.o

1. В директории memblock\_type создадим файл memblock\_type.c :

#include <linux/kernel.h>

#include <linux/syscalls.h>

#include <linux/fs.h>

#include <linux/path.h>

#include <linux/memblock.h>

#include <linux/types.h>

SYSCALL\_DEFINE0 (memblock\_type)

{

struct memblock\_type\* memory =&memblock.memory

printk(KERN\_INFO "memory name \"%s\"\n", memory->name);

printk(KERN\_INFO "count \"%llu\"\n", (u64 )memory->total\_size);

printk(KERN\_INFO "count \"%lu\"\n", memory->cnt);

return 0;

}

1. В директории memblock\_type создадим файл Makefile :

obj-y := memblock\_type.o

**Теперь обратимся к главному Makefile:**

*core-y += kernel/ certs/ mm/ fs/ ipc/ security/ crypto/ block/ memblock\_type/ pci\_dev/*

**Добавим системные вызовы в arch/x86/entry/syscalls/syscall\_64.tbl**

438 common pci\_dev \_\_x64\_sys\_pci\_dev

439 common memblock\_type \_\_x64\_sys\_ memblock\_type

**Добавим системные вызовы в arch/x86/entry/syscalls/syscall\_32.tbl**

438 i386 pci\_dev sys\_pci\_dev \_\_ia32\_sys\_pci\_dev

439 i386 memblock\_type sys\_ memblock\_type \_\_ia32\_sys\_ memblock\_type

**Добавим интерфейсы в *include/linux/syscalls.h:***

asmlinkage long sys\_pci\_dev(void);

asmlinkage long sys\_memblock\_type(void);

**Напишем программу для тестирования *test.c* :**

#include <linux/module.h>

#include <linux/kernel.h>

#include <linux/unistd.h>

#include <sys/syscall.h>

int main()

{

int ans = 0;

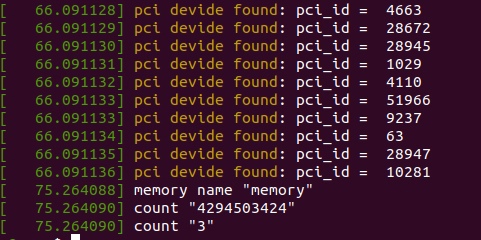
ans = syscall(438);

ans = syscall(439);

return 0;

}

**Компилируем файл и смотрим записи в *dmesg:***



**Выводы**

Во время выполнения лабораторной работы я углубился в работу ядра linux. Написал собственные системные вызовы, добавил их в ядро и перекомпилировал, а также проверил их на работоспособность.