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;
}
```

2. В директории pci_dev создадим файл Makefile:
   obj-y := pci_dev.o
3. 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;
}
```

4. 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.