УНИВЕРСИТЕТ ИТМО

Факультет программной инженерии и компьютерной техники

Направление подготовки 09.03.04 Программная инженерия

Дисциплина «Операционные системы»

**Лабораторная работа №2**

*Вариант Syscall:cpu\_itimer, pci\_dev*

Студент

*Митрофанов Е. Ю.*

*P33101*

Преподаватель

*Осипов С. В.*

Санкт-Петербург, 2021 г.

Описание 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 напишем сами системные вызовы, поместим каждый их них в отдельный каталог:

*cpu\_itimer.c*

#include <linux/kernel.h>

#include <linux/syscalls.h>

#include <linux/pid.h>

#include <linux/sched/signal.h>

SYSCALL\_DEFINE1(cpu\_itimer, pid\_t, pid\_input)

{

struct pid\* result\_pid;

struct task\_struct\* result\_task\_struct;

result\_pid = find\_get\_pid(pid\_input);

result\_task\_struct = get\_pid\_task(result\_pid, PIDTYPE\_PID);

struct cpu\_itimer\* it\_ITIMER\_PROF = &result\_task\_struct->signal->it[0];

struct cpu\_itimer\* it\_ITIMER\_VIRTUAL = &result\_task\_struct->signal->it[1];

printk(KERN\_INFO "For ITIMER\_PROF Expires: [%d], Incr: [%d]\n",

it\_ITIMER\_PROF->expires, it\_ITIMER\_PROF->incr);

printk(KERN\_INFO "For ITIMER\_VIRTUAL Expires: [%d], Incr: [%d]\n",

it\_ITIMER\_VIRTUAL->expires, it\_ITIMER\_VIRTUAL->incr);

return 0;

}

*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 found [%d]\n", dev->device);

}

return 0;

}

Для обоих вызовов в каталогах напишем Makefile-ы:

*Makefile*

*obj-y := cpu\_itimer.o*

*Makefile*

*obj-y := pci\_dev.o*

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

asmlinkage long sys\_cpu\_itimer(void);

asmlinkage long sys\_pci\_dev(void);

Добавим новые стистемные вызовы в таблицу системных вызовов *arch/x86/entry/syscalls/syscall\_64.tbl:*

549 64 cpu\_itimer sys\_cpu\_itimer

550 64 pci\_dev sys\_pci\_dev

После этого перекомпилируем ядро и устанавливаем в систему.

Напишем простой скрипт для вызова добавленный syscall-ов.

*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(549);

ans = syscall(550, 1);

return 0;

}

Компилируем код, запускаем бинарный файл и смотрим записи в *dmesg:*

[186.380378] For ITIMER\_PROF Expires: [0], Incr: [0]  
[186.380382] For ITIMER\_VIRTUAL Expires: [0], Incr: [0]  
[ 207.970422] pci found [4663]  
[ 207.970425] pci found [28672]  
[ 207.970428] pci found [28945]  
[ 207.970430] pci found [1029]  
[ 207.970432] pci found [4110]  
[ 207.970434] pci found [51966]  
[ 207.970436] pci found [9237]  
[ 207.970438] pci found [63]  
[ 207.970440] pci found [28947]  
[ 207.970442] pci found [9820]  
[ 207.970444] pci found [10281]

В моей реализации я вывожу два поля структуры *cpu\_itimer* для обоих типов таймера (для примера вывод структуры для процесса с PID = 1), а также поле *device* структуры *pci\_dev* содержащее номера устройств, подключенных к шине PCI.

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.