Федеральное государственное автономное образовательное
учреждение высшего образования
Университет ИТМО

Факультет программной инженерии и компьютерной техники

Отчёт по лабораторной работе №2
по дисциплине «Операционные системы»

Вариант: procfs: cpu_itimer, syscall_info

Выполнил:
Деев Роман Александрович

Группа: **P33102**

Преподаватели:
Барсуков И. А.

Санкт-Петербург 2022 г.

Задание:

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.

Интерфейс передачи между программой пользователя и ядром и целевая структура задается преподавателем. Интерфейс передачи может быть один из следующих:

- syscall - интерфейс системных вызовов.
- ioctl - передача параметров через управляющий вызов к файлу/устройству.
- procfs - файловая система /proc, передача параметров через запись в файл.
- debugfs - отладочная файловая система /sys/kernel/debug, передача параметров через запись в файл.

Целевая структура может быть задана двумя способами:

- Именем структуры в заголовочных файлах Linux
- Файлом в каталоге /proc. В этом случае необходимо определить целевую структуру по пути файла в /proc и выводимым данным.

Вариант: procfs: cpu_itimer, syscall_info

Код модуля:

```
#include <asm/syscall.h>
#include <linux/kernel.h>
#include <linux/module.h>
#include <linux/proc_fs.h>
#include <linux/ptrace.h>
#include <linux/sched/task_stack.h>
#include <linux/uaccess.h>
#include <linux/version.h>
#include <linux/mutex.h>

#include <linux/sched.h>

#if LINUX_VERSION_CODE >= KERNEL_VERSION(5, 6, 0)
#define HAVE_PROC_OPS
#endif

/* Buffer size */
#define PROCFS_MAX_SIZE 1024
/* Name of procfs file */
#define PROCFS_NAME "kmod"

/* Struct MACRO */
#define STRUCT_CPU_ITIMER 0
#define STRUCT_SYSCALL_INFO 1

/* Proc directory */
static struct proc_dir_entry *our_proc_file;

/* Temp buffer */
static char procfs_buffer[PROCFS_MAX_SIZE];

/* Temp length */
static unsigned long procfs_buffer_size = 0;

static int pid = 0;
static int struct_id = 0;

static DEFINE_MUTEX(args_mutex);

// struct cpu_itimer {
//   cputime_t expires;
//   cputime_t incr;
// };

static int write_cpu_itimers(struct task_struct *task, char __user
*buffer,
                             loff_t *offset, size_t buffer_length) {
```

```
  /* Declare structures */
  struct cpu_itimer itimer_virtual, itimer_prof;
  int len = 0;

  /* Init values */
  spin_lock_irq(&task->sighand->siglock);
  itimer_prof = task->signal->it[0];
  itimer_virtual = task->signal->it[1];
  spin_unlock_irq(&task->sighand->siglock);
  /* String to input */
  len += sprintf(procfs_buffer,
                 "virtual: expires: %llu, incr: %lld;\n"
                 "prof: expires: %llu, incr: %lld;\n",
                 itimer_virtual.expires, itimer_virtual.incr,
                 itimer_prof.expires, itimer_prof.incr) +
         1;

  /* Copy string to user space */

  printk("off: %d len: %d", *offset, len);
  if (*offset >= len) {
    pr_info("Can't copy to user space. By offset\n");
    return -EFAULT;
  }
  if (*offset >= len || copy_to_user(buffer, procfs_buffer, len)) {
    pr_info("Can't copy to user space\n");
    return -EFAULT;
  }

  /* Return value */
  *offset += len;
  return len;
}

int write_syscall_info(struct task_struct *task, char __user *buffer,
                       loff_t *offset, size_t buffer_length) {
  /* Declare structures */
  struct syscall_info info;
  int len = 0;
  unsigned long args[6] = {};

  /* Init values */
  struct pt_regs *regs = task_pt_regs(task);
  info.sp = user_stack_pointer(regs);
  info.data.instruction_pointer = instruction_pointer(regs);
  info.data.nr = syscall_get_nr(task, regs);
  if (info.data.nr != -1L) {
    syscall_get_arguments(task, regs, args);
```

```
  }
  info.data.args[0] = args[0];
  info.data.args[1] = args[1];
  info.data.args[2] = args[2];
  info.data.args[3] = args[3];
  info.data.args[4] = args[4];
  info.data.args[5] = args[5];

  /* String to input */
  len += sprintf(procfs_buffer,
                 "Stack pointer is %lld\nInstruction pointer: %lld\n"
                 "Syscall executed is: %d\n"
                 "Args: \n1: %llu\n2: %llu\n3: %llu\n4: "
                 "%llu\n5: %llu\n6: %llu\n",
                 info.sp, info.data.instruction_pointer, info.data.nr,
                 info.data.args[0], info.data.args[1],
info.data.args[2],
                 info.data.args[3], info.data.args[4],
info.data.args[5]);

  /* Copy string to user space */
  if (*offset >= len || copy_to_user(buffer, procfs_buffer, len)) {
    pr_info("Can't copy to user space\n");
    return -EFAULT;
  }

  /* Return value */
  *offset += len;
  return len;
}

static ssize_t procfile_read(struct file *filePointer, char __user
*buffer,
                             size_t buffer_length, loff_t *offset) {
  if (buffer_length < PROCFS_MAX_SIZE) {
    pr_info("Not enough space in buffer\n");
    return -EFAULT;
  }
  mutex_lock(&args_mutex);
  if (pid) {
    struct task_struct *task = get_pid_task(find_get_pid(pid),
PIDTYPE_PID);
    if (task == NULL) {
      pr_info("Can't get task struct for this pid\n");
      mutex_unlock(&args_mutex);
      return -EFAULT;
    }
    if (struct_id == STRUCT_CPU_ITIMER) {
```

```
      mutex_unlock(&args_mutex);
      return write_cpu_itimers(task, buffer, offset, buffer_length);
    }
    if (struct_id == STRUCT_SYSCALL_INFO) {
      mutex_unlock(&args_mutex);
      return write_syscall_info(task, buffer, offset, buffer_length);
    }
  }
  mutex_unlock(&args_mutex);
  return -EFAULT;
}

/* This function calls when user writes to proc file */

static ssize_t procfile_write(struct file *file, const char __user
*buff,
                              size_t len, loff_t *off) {
  int num_of_args, a, b;

  /* We don't need to read more than 1024 bytes */
  procfs_buffer_size = len;
  if (procfs_buffer_size > PROCFS_MAX_SIZE)
    procfs_buffer_size = PROCFS_MAX_SIZE;

  /* Copy data from user space */
  if (copy_from_user(procfs_buffer, buff, procfs_buffer_size)) {
    pr_info("Can't copy from user space\n");
    return -EFAULT;
  }

  /* Read args from input */
  num_of_args = sscanf(procfs_buffer, "%d %d", &a, &b);
  if (num_of_args != 2) {
    pr_info("Invalid number of args\n");
    return -EFAULT;
  }

  /* Copy args to program */
  mutex_lock(&args_mutex);
  struct_id = a;
  pid = b;
  mutex_unlock(&args_mutex);

  pr_info("Struct id is: %d\n", struct_id);
  pr_info("Pid is: %d\n", pid);

  return procfs_buffer_size;
}
```

```
#ifdef HAVE_PROC_OPS

static const struct proc_ops proc_file_fops = {
    .proc_read = procfile_read,
    .proc_write = procfile_write,
};

#else

static const struct file_operations proc_file_fops = {
    .read = procfile_read,
    .write = procfile_write,
};

#endif

static int __init procfs2_init(void) {
  /* Init proc file */
  our_proc_file = proc_create(PROCFS_NAME, 0644, NULL, &proc_file_fops);
  if (NULL == our_proc_file) {
    proc_remove(our_proc_file);
    pr_alert("Error:Could not initialize /proc/%s\n", PROCFS_NAME);
    return -ENOMEM;
  }
  pr_info("/proc/%s created\n", PROCFS_NAME);
  return 0;
}

static void __exit procfs2_exit(void) {
  /* Delete proc file */
  proc_remove(our_proc_file);
  pr_info("/proc/%s removed\n", PROCFS_NAME);
}

module_init(procfs2_init);
module_exit(procfs2_exit);

MODULE_LICENSE("GPL");
```

Код тестовой программы для запуска отслеживаемого таймера:

```
#include <fcntl.h>
#include <signal.h>
#include <stdio.h>
#include <string.h>
#include <sys/time.h>
#include <unistd.h>

struct itimerval value_virt;
struct itimerval it_set;

void alarm_wakeup(int i) { printf("alarm:%d\n", i); }

int main(int argc, char *argv[]) {
  it_set.it_interval.tv_sec = 5;
  it_set.it_interval.tv_usec = 10;
  it_set.it_value.tv_sec = 5;
  it_set.it_value.tv_usec = 10;
  setitimer(ITIMER_VIRTUAL, &it_set, &value_virt);

  while (1) {
    int tick = 0;
    sleep(1);
    tick++;
    printf("%d", tick);
  }
  return 0;
}
```

Makefile:

```
obj-m += kmod.o

all:
	echo "Targets: clean, build, install"

build:
	make -C /lib/modules/$(shell uname -r)/build M=$(PWD) modules

clean:
	make -C /lib/modules/$(shell uname -r)/build M=$(PWD) clean

install: build
	sudo rmmod kmod.ko
	sudo insmod kmod.ko
	sudo chmod 777 /proc/kmod
```

Пример запуска тестовой программы:

```
./main &  echo 1 $! > /proc/kmod && cat /proc/kmod 2> /dev/null

./main &  echo 0 $! > /proc/kmod && cat /proc/kmod 2> /dev/null
```

Выводы:

В ходе выполнения данной лабораторной работы я познакомился со способами передачи информации пользователю из ядра Linux, и реализовал интерфейс передачи с помощью procfs и записи в файл, который передают пользователю содержимое структур cpu_itimer, syscall_info.