Федеральное государственное автономное образовательное учреждение высшего образования «Национальный исследовательский университет ИТМО»

*Факультет программной инженерии и компьютерной техники*

**Отчёт по лабораторной работе №2**

**по предмету «Операционные системы»**

**Вариант: procfs: thread\_struct, dentry**

Выполнил:

Мельников Н. С.

Группа: P33222.

Проверил:

Осипов С. В.

Санкт-Петербург

2023

**Цель лабораторной работы:**

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.

**Содержание программы:**

**my\_module.h**

#include <linux/init.h>  
#include <linux/module.h>  
#include <linux/kernel.h>  
#include <linux/types.h>  
#include <linux/slab.h>  
#include <linux/proc\_fs.h>  
#include <linux/pid.h>  
#include <linux/sched.h>  
#include <linux/moduleparam.h>  
#include <linux/sched/task\_stack.h>  
#include <linux/path.h>  
#include <linux/namei.h>  
#include <linux/fs.h>  
#include <linux/dcache.h>  
#include <linux/file.h>  
  
MODULE\_LICENSE("GPL");  
  
#define MODULE\_NAME "request"  
#define BUFFER\_MAX\_SIZE 1024  
#define THREAD\_STRUCT 1  
#define DENTRY 2  
  
struct proc\_dir\_entry\* request = NULL;  
u64 pid = NULL;  
u64 struct\_num = NULL;  
char user\_path[64];  
  
char buffer [BUFFER\_MAX\_SIZE];  
size\_t buffer\_size = 0;  
  
struct task\_struct\* task;

**my\_module.c**

#include "my\_module.h"  
  
DEFINE\_MUTEX(lock);  
  
ssize\_t write\_response(char \_\_user \*user\_buff, size\_t user\_buffer\_length, loff\_t \*offset){  
ssize\_t ret = buffer\_size;  
  
if (\*offset >= buffer\_size || copy\_to\_user(user\_buff, buffer, buffer\_size)) {  
ret = 0;  
} else {  
\*offset += buffer\_size;  
}  
mutex\_unlock(&lock);  
return ret;  
}  
  
ssize\_t get\_thread\_struct(char \_\_user \*user\_buff, size\_t user\_buffer\_length, loff\_t \*offset){  
  
task = get\_pid\_task(find\_get\_pid(pid), PIDTYPE\_PID);  
if (task == NULL){  
pr\_err("Fail to find pid task");  
mutex\_unlock(&lock);  
return -EFAULT;  
}  
  
pr\_info("%lu\n", task->thread.fsbase);  
pr\_info("%lu\n", task->thread.sp);  
pr\_info("%lu\n", task->thread.gsbase);  
pr\_info("%hu\n", task->thread.gsindex);  
  
buffer\_size = sprintf(buffer, "The thread\_struct structure was obtained successfully!\n{\n\  
 thread.es: %hu \n \  
 thread.ds: %hu \n \  
 thread.fsindex: %hu \n \  
 thread.gsindex: %hu \n \  
 thread.fsbase: %lu \n \  
 thread.gsbase: %lu \n \  
 thread.sp: %lu \n}\n",  
 task->thread.es, task->thread.ds, task->thread.fsindex, task->thread.gsindex, task->thread.fsbase, task->thread.gsbase, task->thread.sp);  
  
return write\_response(user\_buff, user\_buffer\_length, offset);  
}  
  
ssize\_t get\_dentry(char \_\_user \*user\_buff, size\_t user\_buffer\_length, loff\_t \*offset){  
  
struct path\* my\_path = kmalloc(sizeof(struct page), GFP\_ATOMIC);  
if(my\_path == NULL){  
pr\_err("Fail to get path");  
mutex\_unlock(&lock);  
return(-EFAULT);  
}  
int status = kern\_path(user\_path, LOOKUP\_FOLLOW, my\_path);  
if (status){  
pr\_err("My\_module: Cant find file from user path");  
kfree(my\_path);  
mutex\_unlock(&lock);  
return(-EFAULT);  
}  
  
buffer\_size = sprintf(buffer, "The dentry structure was obtained successfully!\n{\n\  
 dentry.d\_name.name: %s \n \  
 dentry.d\_parent.name: %s \n \  
 dentry.d\_inode.i\_uid: %d \n \  
 dentry.d\_inode.i\_gid: %d \n \  
 dentry.d\_inode.flags: %d \n}\n",  
 my\_path->dentry->d\_name.name, my\_path->dentry->d\_parent->d\_name.name, my\_path->dentry->d\_inode->i\_uid, my\_path->dentry->d\_inode->i\_gid, my\_path->dentry->d\_inode->i\_flags);  
  
pr\_info("name %s\n", my\_path->dentry->d\_name.name);  
kfree(my\_path);  
return write\_response(user\_buff, user\_buffer\_length, offset);  
}  
  
ssize\_t file\_read(struct file \*filePointer, char \_\_user \*user\_buff,  
size\_t user\_buffer\_length, loff\_t \*offset){  
if (struct\_num == NULL){  
pr\_err("My\_module: No input data, please enter them");  
mutex\_unlock(&lock);  
return(-EFAULT);  
}  
  
if (struct\_num == THREAD\_STRUCT ){  
return get\_thread\_struct(user\_buff, user\_buffer\_length, offset);  
}  
else if (struct\_num == DENTRY){  
return get\_dentry(user\_buff, user\_buffer\_length, offset);  
}  
mutex\_unlock(&lock);  
return -EFAULT;  
}  
  
ssize\_t file\_write(struct file \*file, const char \_\_user \*user\_buff,  
size\_t len, loff\_t \*off){  
if(mutex\_trylock(&lock)){  
buffer\_size = len;  
if (buffer\_size > BUFFER\_MAX\_SIZE)  
buffer\_size = BUFFER\_MAX\_SIZE;  
  
if (copy\_from\_user(buffer, user\_buff, buffer\_size))  
return -EFAULT;  
  
int count\_of\_args = sscanf(buffer, "%d", &struct\_num);  
  
if (struct\_num == THREAD\_STRUCT){  
count\_of\_args = sscanf(buffer, "%d %d", &struct\_num, &pid);  
}  
  
if (struct\_num == DENTRY){  
count\_of\_args = sscanf(buffer, "%d %s", &struct\_num, &user\_path);  
}  
  
// pr\_info("pid %d\n", pid);  
// pr\_info("struct num %d\n", struct\_num);  
// pr\_info("user path %s\n", user\_path);  
  
return buffer\_size;  
}  
else{  
return -EFAULT;  
}  
}  
  
  
  
struct proc\_ops add\_ops = {  
 .proc\_read = file\_read,  
 .proc\_write = file\_write,  
};  
  
int \_\_init my\_module\_init(void) {  
 pr\_info("My\_module start init\n");  
 request = proc\_create(MODULE\_NAME, 0777, NULL, &add\_ops);  
 if (!request) {  
 pr\_err("procfs: failed to create file \n");  
 return -1;  
 }  
 return 0;  
}  
  
void \_\_exit my\_module\_exit(void) {  
 proc\_remove(request);  
 pr\_info("my\_module unloaded\n");  
}  
  
module\_init(my\_module\_init);  
module\_exit(my\_module\_exit);

**user.c**

int main(){  
  
 char buffer[128];  
 char path[64];  
 int request;  
 int iter;  
 int pid;  
  
 printf("%s\n", "Hello, select the structure type\n1: thread\_struct\n2: dentry");  
 scanf("%d", &request);  
  
 if(request == 1){  
 printf("%s\n", "Enter the pid");  
 scanf("%d", &pid);  
 snprintf(buffer, 64, "echo %d %d >> /proc/request", request, pid);  
 }  
 if (request == 2){  
 printf("%s\n", "Enter path to file");  
 scanf("%s", &path);  
 snprintf(buffer, 128, "echo %d %s >> /proc/request", request, path);  
 }  
  
 iter = system(buffer);

int r = system(“cat /proc/request”);

return r;  
}

**Makefile**

obj-m += my\_module.o  
 KDIR := /lib/modules/$(shell uname -r)/build  
  
 all:  
gcc user.c -o user  
make -C $(KDIR) M=$(PWD) modules  
 chmod 777 /proc  
 insmod my\_module.ko  
  
 clean:  
make -C $(KDIR) M=$(PWD) clean

Файл user.c является собой простым скриптом для использования созданного модуля. Пользователь вводит 1 или 2, в зависимости от того, информации о какой структуре он хочет получить.

Если выбрана thread\_struct, надо ввести PID процесса.

Если выбрана dentry, надо ввести путь до директории/файла.

Пример работы программы:

thread\_struct:

Изображение выглядит как текст

Автоматически созданное описание

dentry:

Изображение выглядит как текст

Автоматически созданное описание

**Вывод:**

Во время выполнения данной лабораторной работы, я разработал комплекс программ на пользовательском уровне и уровне ярда, который собирает информацию на стороне ядра и передает информацию на уровень пользователя.