Министерство образования и науки РФ

Федеральное государственное автономное

образовательное учреждение высшего образования

«Национальный исследовательский университет ИТМО»

**факультет программной инженерии и компьютерной техники**

**ЛАБОРАТОРНАЯ РАБОТА №2**

по дисциплине

“Операционные системы”

*Выполнил:*

Студент группы P33302

Романов А.И.

*Преподаватель:*

Пашнин А.Д.

Санкт-Петербург, 2023

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:

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

Целевая структура может быть задана двумя способами:

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

Опции по варианту

### Утилита, функциональность которой реализовывает модуль ядра:

*df*

### Интерфейс передачи аргументов и результатов:

*debugfs*

Исходный код программы модуля ядра

*Kernel\_module/kernel\_module.c:*

#include <linux/init.h>

#include <linux/module.h>

#include <linux/kernel.h>

#include <linux/fs.h>

#include <linux/debugfs.h>

#include <asm/pgtable.h>

#include <linux/fd.h>

#include <linux/path.h>

#include <linux/mount.h>

#include <linux/file.h>

#include <linux/vfs.h>

#include <linux/mutex.h>

#include <linux/namei.h>

#define BUFFER\_SIZE 128

**static** **DEFINE\_MUTEX**(kernel\_module\_mutex);

MODULE\_LICENSE("Dual BSD/GPL");

MODULE\_DESCRIPTION("artemiyjjj");

MODULE\_VERSION("1.0");

**static** **struct** dentry \*kernel\_module\_root;

**static** **struct** dentry \*kernel\_module\_args\_file;

**static** **struct** dentry \*kernel\_module\_result\_file;

// static struct nameidata\* nd = {0};

**static** **struct** path\* path = {**0**};

**static** **struct** super\_block\* sb;

**static** **struct** kstatfs\* stats;

**static** **char** arg\_filename[BUFFER\_SIZE];

**static** **char** kernel\_buffer[BUFFER\_SIZE];

**static** **int** **get\_path\_by\_name**(**char**\* path\_name);

**static** **int** **get\_sb\_by\_path**(**void**);

**static** **int** **get\_stats\_by\_denty**(**void**);

**static** **int** **print\_data**(**struct** seq\_file \*sf, **void** \*data) {

printk(KERN\_INFO "Writing to file.**\n**");

**if** (NULL == path) {

seq\_printf(sf, "Found no file or directory by given path.**\n**");

**return** **0**;

} **else** **if** (NULL == sb) {

seq\_printf(sf, "Given path does not lead to mount filesystem.**\n**");

**return** **0**;

} **else** **if** (NULL == stats) {

seq\_printf(sf, "Statistics on filesystem are unavaliabe.**\n**");

**return** **0**;

} **else** {

seq\_printf (sf, "======================================");

seq\_printf (sf, "File System statistics:");

seq\_printf (sf, "File system type: %s", sb -> s\_type -> name);

seq\_printf (sf, "Mount on: %s", sb -> s\_id);

seq\_printf (sf, "Block size: %lu", sb -> s\_blocksize);

seq\_printf (sf, "Max file size: %llu bytes", sb -> s\_maxbytes);

seq\_printf (sf, "**\t**Total blocks: %llu", stats -> f\_blocks);

seq\_printf (sf, "**\t**Free blocks: %llu", stats -> f\_bfree);

// seq\_printf (sf, "\tUsed blocks: %llu", stats -> f\_files);

seq\_printf (sf, "**\t**Total inodes: %llu", stats -> f\_files);

seq\_printf (sf, "**\t**Free inodes: %llu", stats -> f\_ffree);

}

printk(KERN\_INFO "Data successfully have been written.**\n**");

**return** **1**;

}

**static** **int** **kmod\_open**(**struct** inode \*inode, **struct** file \*file) {

// if (!mutex\_trylock(&kernel\_module\_mutex)) {

// printk(KERN\_INFO "can't lock file");

// return -EBUSY;

// }

mutex\_lock(&kernel\_module\_mutex);

printk(KERN\_INFO "file is locked by module");

**return** single\_open(file, print\_data, NULL);

}

**static** **ssize\_t** **kmod\_args\_write**( **struct** file\* file, **const** **char** \_\_user\* buffer, **size\_t** length, **loff\_t**\* ptr\_offset ) {

// char user\_data[BUFFER\_SIZE];

// char fs\_path[length];

printk(KERN\_INFO "df module: getting argumments, size = %llu ...**\n**", length);

printk(KERN\_INFO "User buffer contents: %s", buffer);

**if** (copy\_from\_user(kernel\_buffer, buffer, length)) {

printk(KERN\_ERR "df module: failed writing to kernel**\n**");

**return** -EFAULT;

}

printk(KERN\_INFO "df module: user input successfully read.**\n**");

**if** (sscanf(kernel\_buffer, "path: %s", arg\_filename) == **1**) {

printk(KERN\_INFO "df module: path from argument: %s**\n**", arg\_filename);

} **else** {

printk(KERN\_ERR "df module: argument is not formatted correctly. Check README file.**\n**");

**return** -EINVAL;

}

get\_path\_by\_name(arg\_filename);

**if** (!get\_sb\_by\_path()) {

printk(KERN\_ERR "Given path is not mount root.**\n**");

**return** -EFAULT;

}

get\_stats\_by\_denty();

single\_open(file, print\_data, NULL);

printk(KERN\_INFO "Structures are filled successfully.**\n**");

**return** strlen(kernel\_buffer);

}

**static** **int** **kmod\_release**(**struct** inode \*inode, **struct** file \*file) {

mutex\_unlock(&kernel\_module\_mutex);

printk(KERN\_INFO "file is unlocked by module");

**return** **0**;

}

**static** **struct** file\_operations kernel\_module\_args\_ops = {

.owner = THIS\_MODULE,

.read = seq\_read,

.write = kmod\_args\_write,

.open = kmod\_open,

.release = kmod\_release

};

**static** **int** **get\_path\_by\_name**(**char**\* name) {

**int** res = kern\_path(name, LOOKUP\_FOLLOW, path);

**if** (res) {

printk(KERN\_ERR "df module: Failed to lookup %s.**\n**", name);

**return** **1**;

}

**return** **0**;

}

**static** **int** **get\_sb\_by\_path**(**void**) {

**if** (path == NULL || !path\_is\_mountpoint(path)) {

**return** **1**;

}

sb = path -> dentry -> d\_sb;

**return** **0**;

}

**static** **int** **get\_stats\_by\_denty**(**void**) {

**return** vfs\_statfs(path, stats);

}

**static** **int** \_\_init **kernel\_module\_init**(**void**) {

printk(KERN\_INFO "Kernel df reader: module loaded.**\n**");

kernel\_module\_root = debugfs\_create\_dir("df\_stat", NULL);

kernel\_module\_args\_file = debugfs\_create\_file("kernel\_module\_args", **0777**, kernel\_module\_root, NULL, &kernel\_module\_args\_ops);

kernel\_module\_result\_file = debugfs\_create\_file("kernel\_module\_result", **0777**, kernel\_module\_root, NULL, &kernel\_module\_args\_ops);

**return** **0**;

}

**static** **void** \_\_exit **kernel\_module\_exit**(**void**) {

mutex\_destroy(&kernel\_module\_mutex);

debugfs\_remove\_recursive(kernel\_module\_root);

printk(KERN\_INFO "Kernel df reader: module unloaded.**\n**");

}

module\_init(kernel\_module\_init);

module\_exit(kernel\_module\_exit);

*kernel\_module/Makefile:*

obj-m += kernel\_module.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 insmod kernel\_module.ko

*user/user.c:*

#include <stdio.h>

**int** **main**(**int** argc, **char** \*argv[]) {

**char**\* path;

**if** (argc != **2**) {

fprintf( stderr, "Failed: Program should start with 1 arguments.**\n**" );

**return** -**1**;

}

path = argv[**1**];

printf("Given input: path = %s**\n**", path);

**FILE** \*kernel\_module\_args = fopen("/sys/kernel/debug/df\_stat/kernel\_module\_args", "w+");

**if** (kernel\_module\_args == NULL) {

fprintf(stderr, "File with args not found.**\n**");

**return** -**1**;

}

fprintf(kernel\_module\_args, "path: %s", path);

fprintf(stdout, "Args have been written to file.**\n**");

fclose(kernel\_module\_args);

**FILE** \*kernel\_module\_result = fopen("/sys/kernel/debug/df\_stat/kernel\_module\_result", "r");

**if** (kernel\_module\_result == NULL) {

fprintf(stderr, "File with result not found.**\n**");

**return** -**1**;

}

**char**\* result;

**while** (fscanf(kernel\_module\_result, "%s", result) != EOF) {

printf("Result:**\n**%s", result);

}

fclose(kernel\_module\_result);

**return** **0**;

}

*user/Makefile:*

**all**:

@echo "Targets: clean, build, run"

**build**:

@gcc user\_module.c -o main

**clean**:

rm -f main

**run**: **build**

@echo

@sudo ./main ./main.c

@echo

Использование

Загрузка модуля в ядро:

*make install*

Запуск пользовательской программы:

*sudo ./main /dev/sha2*

Заключение

Создал собственный загружаемый модуль для ядра Linux версии 6.5, осуществляющий создание виртуальных директорий и файлов в виртуальной файловой системе debugfs, чтение и запись в эти файлы, а получение структур ядра, отвечающих за хранение статистики об блоках и inode требуемой смонтированной файловой системы.