Национальный исследовательский университет информационных технологий, механики и оптики

Факультет Программной Инженерии и Компьютерной Техники

Лабораторная работа №2

По дисциплине:

«Операционные системы»

Работу выполнила:

Студентка группы P33102

Никонова Наталья Игоревна

Преподаватель:

Барсуков Илья Александрович

Санкт-Петербург

2022

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: procfs

Структуры: fpu, task_struct

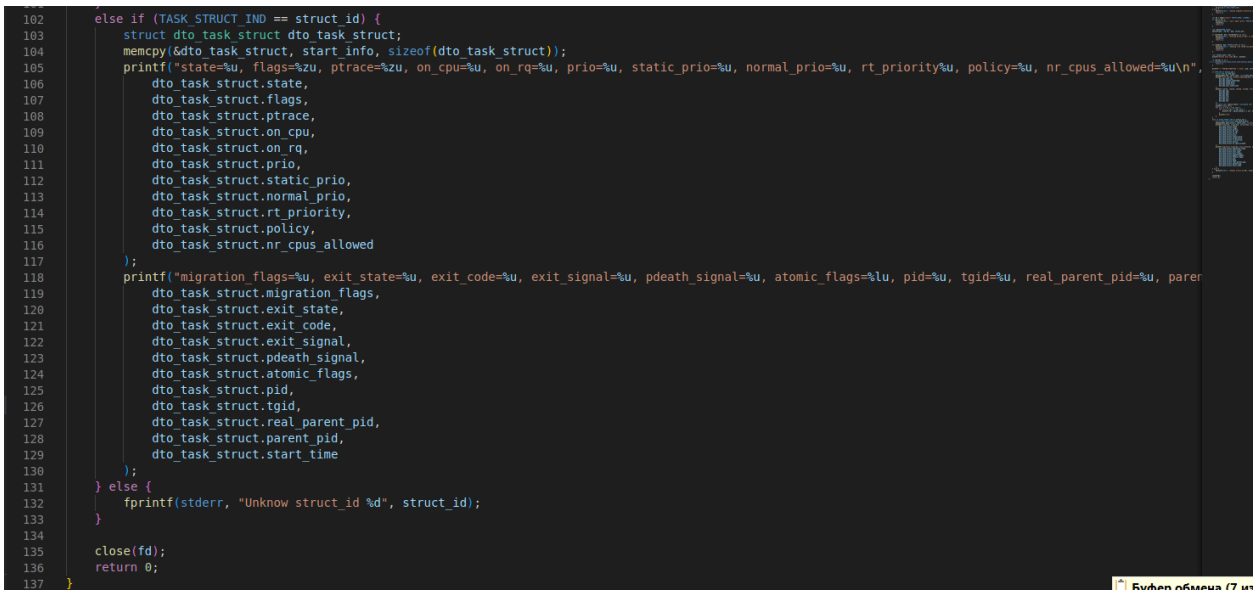
Исходный кодъ: https://github.com/nanikon/operating-systems/tree/main/lab2

Программа пользователя

```c
#include <stdio.h>
#include <string.h>
#include <unistd.h>
#include <fcntl.h>
#include <stdlib.h>
#include <ctype.h>

#include "common.h"

int main(int argc, char *argv[])
{
    if (argc == 2) {
        if (argv[1] == "-h") {
            printf("Two argument required: first is a pid of process, second is name of struct - fpu or task_struct\n");
            return 0;
        } else {
            fprintf(stderr, "Unknow flag %s\n", argv[1]);
            return 1;
        }
    } else if (argc != 3) {
        fprintf(stderr, "Wrong count arguments - %d\n", argc);
        return 1;
    }

    int pid = atoi(argv[1]);
    if (0 == pid &&  !isdigit(argv[1][0])) {
        fprintf(stderr, "First argument should be a number, found %s\n", argv[1]);
        return 1;
    }

    char *struct_name = argv[2];
    int struct_id;
    if (strcmp(struct_name, "fpu") == 0) {
        struct_id = FPU_IND;
    } else if(strcmp(struct_name, "task_struct") == 0) {
        struct_id = TASK_STRUCT_IND;
    } else {
        fprintf(stderr, "Second argument should be \"fpu\" or \"task_struct\", found %s\n", struct_name);
        return 1;
```

```c
    int fd = open("/proc/" PROCFS_NAME, O_RDWR);
    if (fd == -1) {
        fprintf(stderr, "can't open /proc/" PROCFS_NAME "\n");
        close(fd);
        return 1;
    }

    char buf[BUFFER_SIZE];
    sprintf(buf, "%d %d", pid, struct_id);

    if (write(fd, buf, strlen(buf)) == -1) {
        fprintf(stderr, "Writing buffer=\"%s\" to fd=%d failed\n", buf, fd);
        close(fd);
        return 1;
    }

    if (read(fd, buf, BUFFER_SIZE) == -1) {
        fprintf(stderr, "Reading from fd=%d failed\n", fd);
        close(fd);
        return 1;
    }

    char *start_info = buf + 1;
    printf("kernel exit code %d\n", buf[0]);

    if (buf[0] == 1) {
        printf("catch some error from kernel: %s\n", start_info);
        return 1;
    }

    printf("--- PID=%d STRUCT=%s ---\n\n", pid, struct_name);
```

```c
    if (FPU_IND == struct_id) {
        struct dto_fpu dto_fpu;
        memcpy(&dto_fpu, start_info, sizeof(dto_fpu));
        printf("last_cpu=%u, avx512_timestamp=%lu, state_perm=%llu, state_size=%u, user_state_size=%u\n",
            dto_fpu.last_cpu,
            dto_fpu.avx512_timestamp,
            dto_fpu.state_perm,
            dto_fpu.state_size,
            dto_fpu.user_state_size
        );
        printf("cwd=%u, swd=%u, twd=%u, fip=%u, fcs=%u, foo=%u, fos=%u\n",
            dto_fpu.cwd,
            dto_fpu.swd,
            dto_fpu.twd,
            dto_fpu.fip,
            dto_fpu.fcs,
            dto_fpu.foo,
            dto_fpu.fos
        );
        unsigned char *split_stack = (unsigned char *) dto_fpu.stack;
        printf("stack:\n");
        for (int i = 0; i < 8; i++) {
            for (int j = 0; j < 10; j++) {
                printf("%u", split_stack[ i * 10 + j]);
            }
            printf("\n");
        }
    }
```



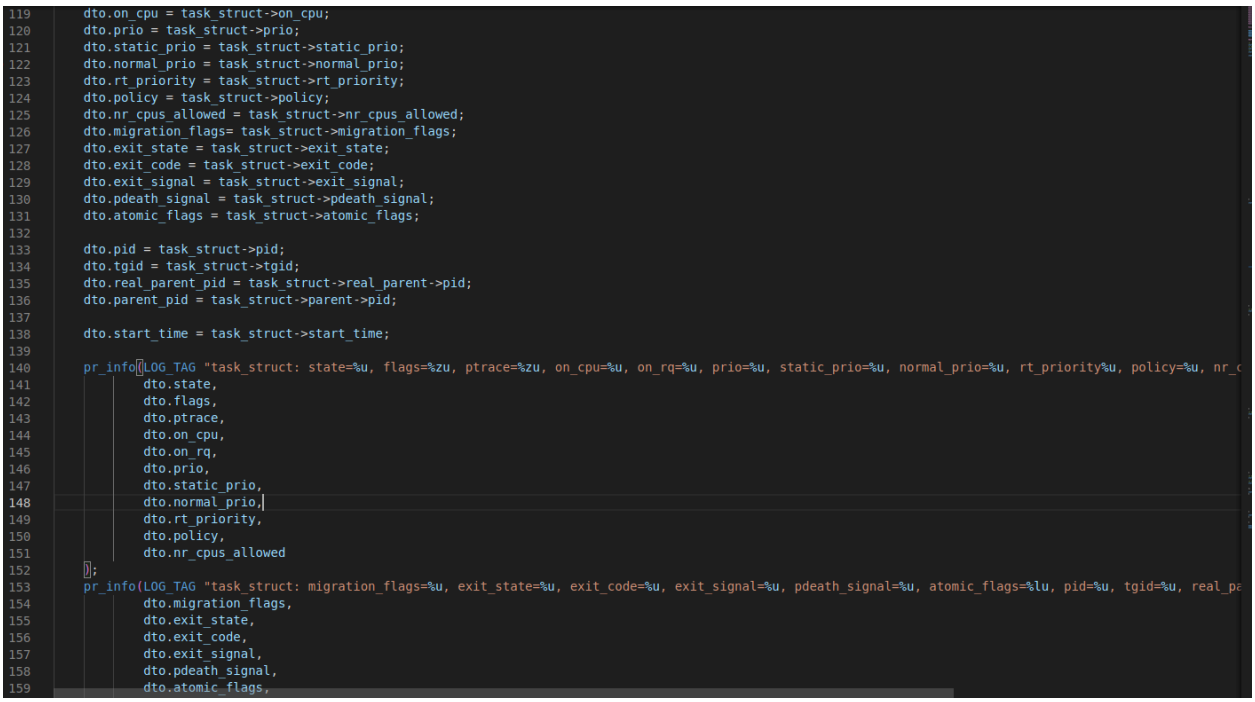
```
    else if (TASK_STRUCT_IND == struct_id) {
        struct dto_task_struct dto_task_struct;
        memcpy(&dto_task_struct, start_info, sizeof(dto_task_struct));
        printf("state=%u, flags=%zu, ptrace=%zu, on_cpu=%u, on_rq=%u, prio=%u, static_prio=%u, normal_prio=%u, rt_priority%u, policy=%u, nr_cpus_allowed=%u\n",
            dto_task_struct.state,
            dto_task_struct.flags,
            dto_task_struct.ptrace,
            dto_task_struct.on_cpu,
            dto_task_struct.on_rq,
            dto_task_struct.prio,
            dto_task_struct.static_prio,
            dto_task_struct.normal_prio,
            dto_task_struct.rt_priority,
            dto_task_struct.policy,
            dto_task_struct.nr_cpus_allowed
        );
        printf("migration_flags=%u, exit_state=%u, exit_code=%u, exit_signal=%u, pdeath_signal=%u, atomic_flags=%lu, pid=%u, tgid=%u, real_parent_pid=%u, parer
            dto_task_struct.migration_flags,
            dto_task_struct.exit_state,
            dto_task_struct.exit_code,
            dto_task_struct.exit_signal,
            dto_task_struct.pdeath_signal,
            dto_task_struct.atomic_flags,
            dto_task_struct.pid,
            dto_task_struct.tgid,
            dto_task_struct.real_parent_pid,
            dto_task_struct.parent_pid,
            dto_task_struct.start_time
        );
    } else {
        fprintf(stderr, "Unknow struct_id %d", struct_id);
    }

    close(fd);
    return 0;
}
```

Программа ядра

```c
#include <linux/cpufeature.h>
#include <linux/kernel.h>
#include <linux/module.h>
#include <linux/mutex.h>
#include <linux/types.h>
#include <linux/proc_fs.h>
#include <linux/uaccess.h>
#include <linux/version.h>
#include <linux/slab.h>
#include <linux/sched.h>
#include <linux/pid.h>

#include "common.h"

MODULE_LICENSE("GPL");
MODULE_AUTHOR("Natalia Nikonova");
MODULE_DESCRIPTION("My kernel module that read some struct from process");
MODULE_VERSION("1.1");

#define LOG_TAG "[os-lab2] "

/* Процесс для которого будет доставаться структура */
static int pid = 0;
/* Идентификатор структуры. 1 - fpu, 2 - task_struct*/
static int struct_id = 0;
/* Мьютекс */
static DEFINE_MUTEX(file_mutex);

static ssize_t copy_to_answer_fpu(char *answer, struct task_struct *task_struct) {
    ssize_t answer_size = 0;

    struct thread_struct thread_struct = task_struct->thread;

    struct fpu fpu = thread_struct.fpu;

    struct dto_fpu dto_fpu;

    dto_fpu.last_cpu = fpu.last_cpu;
    dto_fpu.avx512_timestamp = fpu.avx512_timestamp;

    dto_fpu.state_perm = fpu.perm.__state_perm;
```

```
    dto_fpu.state_size = fpu.perm.__state_size;
    dto_fpu.user_state_size = fpu.perm.__user_state_size;

    if (!static_cpu_has(X86_FEATURE_FPU)) {
        /* Программная эмуляция FPU, совпадает с x87*/
        dto_fpu.cwd = fpu.__fpstate.regs.soft.cwd;
        dto_fpu.swd = fpu.__fpstate.regs.soft.swd;
        dto_fpu.twd = fpu.__fpstate.regs.soft.twd;
        dto_fpu.fip = fpu.__fpstate.regs.soft.fip;
        dto_fpu.fcs = fpu.__fpstate.regs.soft.fcs;
        dto_fpu.foo = fpu.__fpstate.regs.soft.foo;
        dto_fpu.fos = fpu.__fpstate.regs.soft.fos;
        memcpy(dto_fpu.stack, fpu.__fpstate.regs.soft.st_space, 80);
        pr_info(LOG_TAG "choose struct swregs_state");
    } else if (static_cpu_has(X86_FEATURE_XSAVE)) {
        /* самый новый вариант - устаревший fxregs + свой заголовок xstate. XSAVE XRSTOR*/
        dto_fpu.cwd = fpu.__fpstate.regs.xsave.i387.cwd;
        dto_fpu.swd = fpu.__fpstate.regs.xsave.i387.swd;
        dto_fpu.twd = fpu.__fpstate.regs.xsave.i387.twd;
        dto_fpu.fip = fpu.__fpstate.regs.xsave.i387.fip;
        dto_fpu.fcs = fpu.__fpstate.regs.xsave.i387.fcs;
        dto_fpu.foo = fpu.__fpstate.regs.xsave.i387.foo;
        dto_fpu.fos = fpu.__fpstate.regs.xsave.i387.fos;
        memcpy(dto_fpu.stack, fpu.__fpstate.regs.xsave.i387.st_space, 80);
        pr_info(LOG_TAG "choose struct xregs_state");
    } else if (static_cpu_has(X86_FEATURE_FXSR)) {
        /* устаревший формат для SSE/MMX, в отличие от fsave снапример охраняет конец xmm регистров. FXSAVE FXRSTOR*/
        dto_fpu.cwd = fpu.__fpstate.regs.fxsave.cwd;
        dto_fpu.swd = fpu.__fpstate.regs.fxsave.swd;
        dto_fpu.twd = fpu.__fpstate.regs.fxsave.twd;
        dto_fpu.fip = fpu.__fpstate.regs.fxsave.fip;
        dto_fpu.fcs = fpu.__fpstate.regs.fxsave.fcs;
        dto_fpu.foo = fpu.__fpstate.regs.fxsave.foo;
        dto_fpu.fos = fpu.__fpstate.regs.fxsave.fos;
        memcpy(dto_fpu.stack, fpu.__fpstate.regs.fxsave.st_space, 80);
        pr_info(LOG_TAG "choose struct fxregs_state");
```

```
    } else {
        /* устаревший варинат для x87, FSAVE FRSTOR*/
        dto_fpu.cwd = fpu.__fpstate.regs.fsave.cwd;
        dto_fpu.swd = fpu.__fpstate.regs.fsave.swd;
        dto_fpu.twd = fpu.__fpstate.regs.fsave.twd;
        dto_fpu.fip = fpu.__fpstate.regs.fsave.fip;
        dto_fpu.fcs = fpu.__fpstate.regs.fsave.fcs;
        dto_fpu.foo = fpu.__fpstate.regs.fsave.foo;
        dto_fpu.fos = fpu.__fpstate.regs.fsave.fos;
        memcpy(dto_fpu.stack, fpu.__fpstate.regs.fsave.st_space, 80);
        pr_info(LOG_TAG "choose struct fregs_state");
    }

    pr_info(LOG_TAG "dto_fpu: last_cpu=%u, avx512_timestamp=%lu, state_perm=%llu, state_size=%u, user_state_size=%u\n",
            dto_fpu.last_cpu,
            dto_fpu.avx512_timestamp,
            dto_fpu.state_perm,
            dto_fpu.state_size,
            dto_fpu.user_state_size
    );
    pr_info(LOG_TAG "dto_fpu: cwd=%u, swd=%u, twd=%u, fip=%u, fcs=%u, foo=%u, fos=%u\n",
            dto_fpu.cwd,
            dto_fpu.swd,
            dto_fpu.twd,
            dto_fpu.fip,
            dto_fpu.fcs,
            dto_fpu.foo,
            dto_fpu.fos
    );
    memcpy(answer, &dto_fpu, sizeof(dto_fpu));
    pr_info(LOG_TAG "dto_fpu copy to answer, size %zu\n", sizeof(struct dto_fpu));
    return sizeof(dto_fpu);
}

static ssize_t copy_to_answer_task_struct(char *answer, struct task_struct *task_struct) {
    struct dto_task_struct dto;

    dto.state = task_struct->__state;
    dto.flags = task_struct->flags;
    dto.ptrace = task_struct->ptrace;
    dto.on_rq = task_struct->on_rq;
```



```
    dto.on_cpu = task_struct->on_cpu;
    dto.prio = task_struct->prio;
    dto.static_prio = task_struct->static_prio;
    dto.normal_prio = task_struct->normal_prio;
    dto.rt_priority = task_struct->rt_priority;
    dto.policy = task_struct->policy;
    dto.nr_cpus_allowed = task_struct->nr_cpus_allowed;
    dto.migration_flags= task_struct->migration_flags;
    dto.exit_state = task_struct->exit_state;
    dto.exit_code = task_struct->exit_code;
    dto.exit_signal = task_struct->exit_signal;
    dto.pdeath_signal = task_struct->pdeath_signal;
    dto.atomic_flags = task_struct->atomic_flags;

    dto.pid = task_struct->pid;
    dto.tgid = task_struct->tgid;
    dto.real_parent_pid = task_struct->real_parent->pid;
    dto.parent_pid = task_struct->parent->pid;

    dto.start_time = task_struct->start_time;

    pr_info(LOG_TAG "task_struct: state=%u, flags=%zu, ptrace=%zu, on_cpu=%u, on_rq=%u, prio=%u, static_prio=%u, normal_prio=%u, rt_priority%u, policy=%u, nr_c
            dto.state,
            dto.flags,
            dto.ptrace,
            dto.on_cpu,
            dto.on_rq,
            dto.prio,
            dto.static_prio,
            dto.normal_prio,
            dto.rt_priority,
            dto.policy,
            dto.nr_cpus_allowed
    );
    pr_info(LOG_TAG "task_struct: migration_flags=%u, exit_state=%u, exit_code=%u, exit_signal=%u, pdeath_signal=%u, atomic_flags=%lu, pid=%u, tgid=%u, real_pa
            dto.migration_flags,
            dto.exit_state,
            dto.exit_code,
            dto.exit_signal,
            dto.pdeath_signal,
            dto.atomic_flags,
```

```
            dto.pid,
            dto.tgid,
            dto.real_parent_pid,
            dto.parent_pid,
            dto.start_time
    };

    memcpy(answer, &dto, sizeof(dto));
    pr_info(LOG_TAG "dto_task_struct copy to answer, dto_task_struct size %zu\n", sizeof(struct dto_task_struct));
    return sizeof(dto);
}

/* Эта функция вызывается при считывании файла /proc. */
static ssize_t procfile_read(struct file *filePointer, char __user *buffer,
                             size_t buffer_length, loff_t *offset)
{
    char *answer = kmalloc(BUFFER_SIZE, GFP_KERNEL);
    char answer_code = 0;
    ssize_t answer_size = 0;

    struct pid *pid_struct = find_get_pid(pid);
    if (NULL == pid_struct) {
        pr_info(LOG_TAG "Can't found process with pid = %d\n", pid);
        answer_code = 1;
        copy_to_user(buffer, &answer_code, 1);
        answer_size = sprintf(answer, "Can't found process with pid = %d\n", pid);
        copy_to_user(buffer + 1, answer, answer_size);
        kfree(answer);
        *offset += (answer_size + 1);
        mutex_unlock(&file_mutex);
        pr_info(LOG_TAG "file freed");
        return answer_size + 1;
    }

    struct task_struct *task_struct = pid_task(pid_struct, PIDTYPE_PID);
    if (NULL == task_struct) {
        pr_info(LOG_TAG "Failed to get task_struct from process with pid = %d\n", pid);
        answer_code = 1;
        copy_to_user(buffer, &answer_code, 1);
        answer_size = sprintf(answer, "Failed to get task_struct from process with pid = %d\n", pid);
        copy_to_user(buffer + 1, answer, answer_size);
        kfree(answer);
        *offset += (answer_size + 1);
        mutex_unlock(&file_mutex);
        pr_info(LOG_TAG "file freed");
        return answer_size + 1;
    }

    if (FPU_IND == struct_id) {
        answer_size = copy_to_answer_fpu(answer, task_struct);
    } else if (TASK_STRUCT_IND == struct_id) {
        answer_size = copy_to_answer_task_struct(answer, task_struct);
    } else {
        pr_info(LOG_TAG "Struct is not defined = %d\n", struct_id);
        answer_code = 1;
        answer_size = sprintf(answer, "Struct is not defined = %d\n", struct_id);
    }

    pr_info(LOG_TAG "start answer copy to user, answer size %d, answer_code=%d\n", answer_size, answer_code);
    copy_to_user(buffer, &answer_code, 1);
    copy_to_user(buffer + 1, answer, answer_size);
    pr_info(LOG_TAG "answer copy to user\n");

    kfree(answer);
    *offset += (answer_size + 1);
    mutex_unlock(&file_mutex);
    pr_info(LOG_TAG "file freed");
    return answer_size + 1;
}

/* Эта функция вызывается при записи файла /proc. */
static ssize_t procfile_write(struct file *file, const char __user *buff,
                              size_t len, loff_t *off)
{
    pr_info(LOG_TAG "procfile_write: open");
    mutex_lock(&file_mutex);
    pr_info(LOG_TAG "mutex locked");

    unsigned long user_input_size = len;
    if (user_input_size > BUFFER_SIZE) {
        user_input_size = BUFFER_SIZE;
    }
```

```
243     char *user_input = kmalloc(user_input_size, GFP_KERNEL);
244     unsigned long lost_bytes = copy_from_user(user_input, buff, user_input_size);
245     if (lost_bytes) {
246         pr_info(LOG_TAG "copy_from_user can't copy %lu bytes\n", lost_bytes);
247         kfree(user_input);
248         mutex_unlock(&file_mutex);
249         return 0;
250     }
251     pr_info(LOG_TAG "proc_write write %zu bytes\n", len);
252 
253     int arg1, arg2, args_num;
254     args_num = sscanf(user_input, "%d %d", &arg1, &arg2);
255     if (2 == args_num) {
256         pr_info(LOG_TAG "read two arguments: arg1=%d, arg3=%d\n", arg1, arg2);
257         pid = arg1;
258         struct_id = arg2;
259     } else {
260         pr_info(LOG_TAG "sscanf found %d arguments - not two", args_num);
261         mutex_unlock(&file_mutex);
262     }
263 
264     kfree(user_input);
265     return user_input_size;
266 }
267 
268 /* Эта структура содержит информацию о файле /proc. */
269 static struct proc_dir_entry *proc_file_info;
270 
271 /* Для версии ядра > 5.6.0, иначе надо использовать  file_operations*/
272 static const struct proc_ops proc_file_fops = {
273     .proc_read = procfile_read,
274     .proc_write = procfile_write,
275 };
276 
277 static int __init procfs_init(void)
278 {
279     proc_file_info = proc_create(PROCFS_NAME, 0644, NULL, &proc_file_fops);
280     if (NULL == proc_file_info) {
281         proc_remove(proc_file_info);
282         pr_alert(LOG_TAG "Error:Could not initialize /proc/%s\n", PROCFS_NAME);
```

```
283         return -ENOMEM;
284     }
285 
286     pr_info(LOG_TAG "/proc/%s created\n", PROCFS_NAME);
287     return 0;
288 }
289 
290 static void __exit procfs_exit(void)
291 {
292     proc_remove(proc_file_info);
293     pr_info(LOG_TAG "/proc/%s removed\n", PROCFS_NAME);
294 }
295 
296 module_init(procfs_init);
297 module_exit(procfs_exit);
```

Программа для нагрузки fpu

fpu_example.asm

```asm
global _start

section .text
_start:
  fldpi
  fld1
  fldln2
  start_die:
  test rax, rax
  jmp start_die
  ret

```

Примеры работы

```
nanikon@nanikon-VirtualBox:~/osItmo/lab2/work$ ./fpu_example &
[1] 66655
```

```
nanikon@nanikon-VirtualBox:~/osItmo/lab2/work$ ./user-mode 66655 fpu
kernel exit code 0
--- PID=66655 STRUCT=fpu ---

last_cpu=0, avx512_timestamp=0, state_perm=0, state_size=512, user_state_size=512
cwd=895, swd=10240, twd=224, fip=0, fcs=0, foo=0, fos=0
stack:
172121207209247231141772546 3
0000000000
000128255630000
0053194104331622181520 1
06400000000
0000000000
0000000000
0000000000
nanikon@nanikon-VirtualBox:~/osItmo/lab2/work$ ./user-mode 66655 task_struct
kernel exit code 0
--- PID=66655 STRUCT=task_struct ---

state=0, flags=4194304, ptrace=0, on_cpu=0, on_rq=1, prio=120, static_prio=120, normal_prio=120, rt_priority0, p
olicy=0, nr_cpus_allowed=1
migration_flags=0, exit_state=0, exit_code=0, exit_signal=17, pdeath_signal=0, atomic_flags=0, pid=66655, tgid=6
6655, real_parent_pid=3441, parent_pid=3441, start_time=36874810574666
nanikon@nanikon-VirtualBox:~/osItmo/lab2/work$ kill -9 66655
nanikon@nanikon-VirtualBox:~/osItmo/lab2/work$ ./user-mode 66655 task_struct
kernel exit code 1
catch some error from kernel: Can't found process with pid = 66655
```

```
nanikon@nanikon-VirtualBox:~/osItmo/lab2/work$ ./user-mode 9922 task_struct
kernel exit code 0
--- PID=9922 STRUCT=task_struct ---

state=1, flags=4194560, ptrace=0, on_cpu=0, on_rq=0, prio=120, static_prio=120, normal_prio=120, rt_priority0, p
olicy=0, nr_cpus_allowed=1
migration_flags=0, exit_state=0, exit_code=0, exit_signal=17, pdeath_signal=0, atomic_flags=0, pid=9922, tgid=99
22, real_parent_pid=1, parent_pid=1, start_time=15119161729992
nanikon@nanikon-VirtualBox:~/osItmo/lab2/work$ ./user-mode 9922 fpu
kernel exit code 0
--- PID=9922 STRUCT=fpu ---

last_cpu=0, avx512_timestamp=0, state_perm=0, state_size=512, user_state_size=512
cwd=895, swd=0, twd=0, fip=0, fcs=0, foo=0, fos=0
stack:
0000000000
0000000000
0000000000
0000000000
0000000000
0000000000
0000000000
0000000000
nanikon@nanikon-VirtualBox:~/osItmo/lab2/work$ ps -e | grep 9922
   9922 ?        00:00:00 cupsd
```

Логи

```
дек 30 12:21:13 nanikon-VirtualBox kernel: [os-lab2] proc_write write 6 bytes
дек 30 12:21:13 nanikon-VirtualBox kernel: [os-lab2] read two arguments: arg1=9922, arg3=1
дек 30 12:21:13 nanikon-VirtualBox kernel: [os-lab2] choose struct fxregs_state
дек 30 12:21:13 nanikon-VirtualBox kernel: [os-lab2] dto_fpu: last_cpu=0, avx512_timestamp=0, state_perm=0, state_size=5
12, user_state_size=512
дек 30 12:21:13 nanikon-VirtualBox kernel: [os-lab2] dto_fpu: cwd=895, swd=0, twd=0, fip=0, fcs=0, foo=0, fos=0
дек 30 12:21:13 nanikon-VirtualBox kernel: [os-lab2] dto_fpu copy to answer, size 144
дек 30 12:21:13 nanikon-VirtualBox kernel: [os-lab2] start answer copy to user, answer size 144, answer_code=0
дек 30 12:21:13 nanikon-VirtualBox kernel: [os-lab2] answer copy to user
```

Демонстрация мьютекса (в код на уровне пользователя была введена задержка в 10 секунд между записью в файл и чтением из него)

```
nanikon@nanikon-VirtualBox:~/osItmo/lab2/work$ ./user-mode 9922 task_struct &
[2] 74303
nanikon@nanikon-VirtualBox:~/osItmo/lab2/work$ ./user-mode 9922 fpu &
[3] 74304
nanikon@nanikon-VirtualBox:~/osItmo/lab2/work$ kernel exit code 0
--- PID=9922 STRUCT=task_struct ---

state=1, flags=4194560, ptrace=0, on_cpu=0, on_rq=0, prio=120, static_prio=120, normal_prio=120, rt_priority0, policy=0, nr_cpus_allowed=1
migration_flags=0, exit_state=0, exit_code=0, exit_signal=17, pdeath_signal=0, atomic_flags=0, pid=9922, tgid=9922, real_parent_pid=1, parent_pid=1, start_t
ime=15119161729992
kernel exit code 0
--- PID=9922 STRUCT=fpu ---

last_cpu=0, avx512_timestamp=0, state_perm=0, state_size=512, user_state_size=512
cwd=895, swd=0, twd=0, fip=0, fcs=0, foo=0, fos=0
stack:
0000000000
0000000000
0000000000
0000000000
0000000000
0000000000
0000000000
0000000000

[2]-  Завершён        ./user-mode 9922 task_struct
[3]+  Завершён        ./user-mode 9922 fpu
nanikon@nanikon-VirtualBox:~/osItmo/lab2/work$
```

```
дек 30 21:31:45 nanikon-VirtualBox kernel: [os-lab2] /proc/lab2_os_module created
дек 30 21:31:58 nanikon-VirtualBox kernel: [os-lab2] procfile_write: open
дек 30 21:31:58 nanikon-VirtualBox kernel: [os-lab2] mutex locked
дек 30 21:31:58 nanikon-VirtualBox kernel: [os-lab2] proc_write write 6 bytes
дек 30 21:31:58 nanikon-VirtualBox kernel: [os-lab2] read two arguments: arg1=9922, arg3=2
дек 30 21:32:08 nanikon-VirtualBox kernel: [os-lab2] procfile_write: open
дек 30 21:32:08 nanikon-VirtualBox kernel: [os-lab2] task_struct: state=1, flags=4194560, ptrace=0, on_cpu=0, on_rq=0, prio=120, static_prio=120, normal_pri
o=120, rt_priority0, policy=0, nr_cpus_allowed=1
дек 30 21:32:08 nanikon-VirtualBox kernel: [os-lab2] task_struct: migration_flags=0, exit_state=0, exit_code=0, exit_signal=17, pdeath_signal=0, atomic_flag
s=0, pid=9922, tgid=9922, real_parent_pid=1, parent_pid=1, start_time=15119161729992
дек 30 21:32:08 nanikon-VirtualBox kernel: [os-lab2] dto_task_struct copy to answer, dto_task_struct size 112
дек 30 21:32:08 nanikon-VirtualBox kernel: [os-lab2] start answer copy to user, answer size 112, answer_code=0
дек 30 21:32:08 nanikon-VirtualBox kernel: [os-lab2] answer copy to user
дек 30 21:32:08 nanikon-VirtualBox kernel: [os-lab2] file freed
дек 30 21:32:08 nanikon-VirtualBox kernel: [os-lab2] mutex locked
дек 30 21:32:08 nanikon-VirtualBox kernel: [os-lab2] proc_write write 6 bytes
дек 30 21:32:08 nanikon-VirtualBox kernel: [os-lab2] read two arguments: arg1=9922, arg3=1
дек 30 21:32:18 nanikon-VirtualBox kernel: [os-lab2] choose struct fxregs_state
дек 30 21:32:18 nanikon-VirtualBox kernel: [os-lab2] dto_fpu: last_cpu=0, avx512_timestamp=0, state_perm=0, state_size=512, user_state_size=512
дек 30 21:32:18 nanikon-VirtualBox kernel: [os-lab2] dto_fpu: cwd=895, swd=0, twd=0, fip=0, fcs=0, foo=0, fos=0
дек 30 21:32:18 nanikon-VirtualBox kernel: [os-lab2] dto_fpu copy to answer, size 144
дек 30 21:32:18 nanikon-VirtualBox kernel: [os-lab2] start answer copy to user, answer size 144, answer_code=0
дек 30 21:32:18 nanikon-VirtualBox kernel: [os-lab2] answer copy to user
```

Выводы

В ходе выполнения лабораторной работы я вспомнила языки си и ассемблер и сборку программ на них, познакомилась со спецификой написания модулей ядра (kmalloc вместо malloc, перенос информации между пользователем и ядром) и как их загружать, поработала с интерфейсом, через который можно программировать виртуальную файловую систему procfs, а также узнала о библиотеке для вычислений с плавающей точкой – FPU – и что в моем ноутбуке регистры FPU сохраняются в структуре fxregs, хотя она и считается устаревшей.