

МИНОБРНАУКИ РОССИИ
Федеральное государственное автономное образовательное
учреждение высшего образования
«ЮЖНЫЙ ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ»

Институт компьютерных технологий и информационной безопасности
Кафедра математического обеспечения и применения ЭВМ

Отчет
По лабораторной работе №7
на тему: «Создание системного вызова в ОС MINIX»

по курсу
**«Операционные системы и системное программное
обеспечение»**
Вариант №7

Выполнили:
студенты гр. КТ602-8
Жалнин Д.И.
Нестеренко П.А.

Проверил:
Ассистент каф. МОП ЭВМ
Альминене Т. А.

1 Цель работы

Освоение структуры микроядерной ОС MINIX, а также выработка навыков создания и использования системных вызовов.

2 Задание

Необходимо разработать следующий системный вызов и протестировать его: По заданному **pid** получить, в зависимости от заданного ключа, его реальный **uid**, эффективный **uid**, реальный **gid** или эффективный **gid**.

3 Ход работы

По ходу лабораторной работы был написан исходный код нового системного вызова **GETFIELDS** (getfields.c):

```
#include "pm.h"
#include "mproc.h"
#include <stdlib.h>

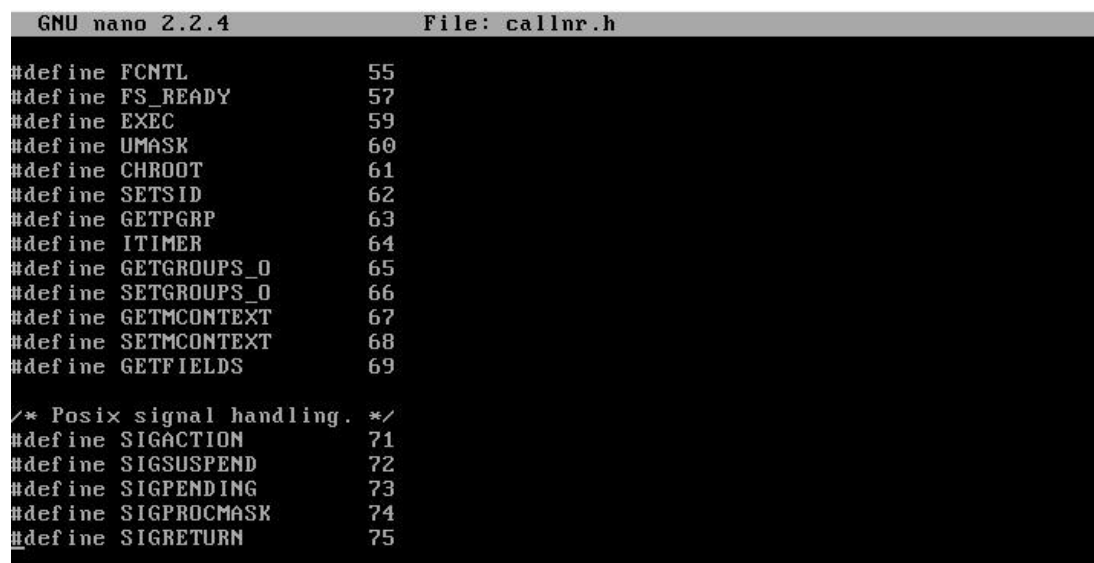
int do_getfields()
{
    int i;

    for (i=0; i < NR_PROCS; i++)
    {
        if (mproc[i].mp_pid == m_in.m1_i1)
        {
            break;
        }
    }
    char *c = (char*) calloc(17, sizeof(char));
    if (i < NR_PROCS) {
        int i1 = (int) &mproc[i].mp_realuid;
        int i2 = (int) &mproc[i].mp_effuid;
        int i3 = (int) &mproc[i].mp_realgid;
        int i4 = (int) &mproc[i].mp_effgid;

        c[0] = (i1 >> 24) & 0xFF;
        c[1] = (i1 >> 16) & 0xFF;
        c[2] = (i1 >> 8) & 0xFF;
        c[3] = i1 & 0xFF;
        c[4] = (i2 >> 24) & 0xFF;
        c[5] = (i2 >> 16) & 0xFF;
        c[6] = (i2 >> 8) & 0xFF;
        c[7] = i2 & 0xFF;
        c[8] = (i3 >> 24) & 0xFF;
        c[9] = (i3 >> 16) & 0xFF;
        c[10] = (i3 >> 8) & 0xFF;
        c[11] = i3 & 0xFF;
        c[12] = (i4 >> 24) & 0xFF;
        c[13] = (i4 >> 16) & 0xFF;
        c[14] = (i4 >> 8) & 0xFF;
        c[15] = i4 & 0xFF;
        c[16] = 0;
    }
    return sys_vircopy(SELF, (vir_bytes)c, m_in.m_source,
(vir_bytes)m_in.m1_p1, (phys_bytes) 17);
}
```

Рисунок 1 – Исходный код нового системного вызова

Для вызова был выбран код 69, так как он не использовался в системе. Определение нового системного вызова было занесено в файл «/usr/src/include/minix/callnr.h» с помощью добавления следующей строки (рис.2): « #define GETFIELDS 69 »



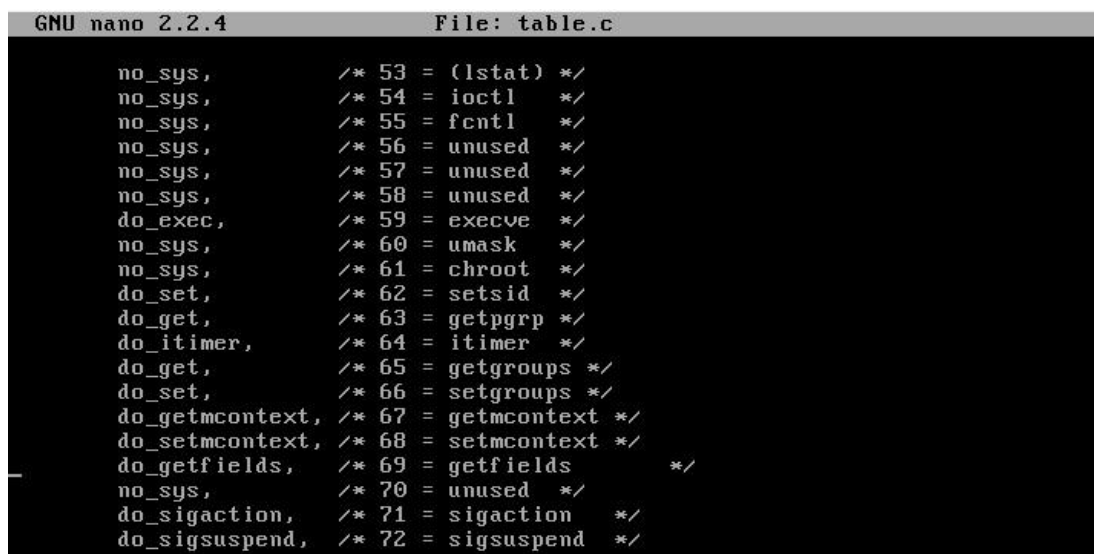
```
GNU nano 2.2.4 File: callnr.h
#define FCNTL 55
#define FS_READY 57
#define EXEC 59
#define UMASK 60
#define CHROOT 61
#define SETSID 62
#define GETPGRP 63
#define ITIMER 64
#define GETGROUPS_0 65
#define SETGROUPS_0 66
#define GETMCONTEXT 67
#define SETMCONTEXT 68
#define GETFIELDS 69

/* Posix signal handling. */
#define SIGACTION 71
#define SIGSUSPEND 72
#define SIGPENDING 73
#define SIGPROCMASK 74
#define SIGRETURN 75
```

Рисунок 2 – Объявление вызова в «callnr.h»

А также в файл «/usr/src/servers/pm/table.c» (рис.3):

« do_getfields, /* 69 = getfields */ ».



```
GNU nano 2.2.4 File: table.c
no_sys, /* 53 = (lstat) */
no_sys, /* 54 = ioctl */
no_sys, /* 55 = fcntl */
no_sys, /* 56 = unused */
no_sys, /* 57 = unused */
no_sys, /* 58 = unused */
do_exec, /* 59 = execve */
no_sys, /* 60 = umask */
no_sys, /* 61 = chroot */
do_set, /* 62 = setsid */
do_get, /* 63 = getpgrp */
do_itimer, /* 64 = itimer */
do_get, /* 65 = getgroups */
do_set, /* 66 = setgroups */
do_getmcontext, /* 67 = getmcontext */
do_setmcontext, /* 68 = setmcontext */
do_getfields, /* 69 = getfields */
no_sys, /* 70 = unused */
do_sigaction, /* 71 = sigaction */
do_sigsuspend, /* 72 = sigsuspend */
```

Рисунок 3 – Объявление вызова в «table.c»

После объявления системного вызова в каталоге «/usr/src» была вызвана команда **make includes**.

Сам исходный код вызова расположен в директории «/usr/src/servers/pm». После внесения информации о вызове и его создания, была произведена компиляция с помощью **make** в текущем каталоге. После успешной компиляции файлов в каталоге, была произведена перекомпиляция системы с помощью команды **make install** в каталоге «/usr/src».

Для тестирования системного вызова была написана дополнительная программа, исходный код, который расположен на рисунке 4. Результат работы программы можно увидеть на рисунке 5.

```
#include <sys/cdefs.h>
#include <lib.h>
#include <stdlib.h>
#include <string.h>
#include <minix/callnr.h>
#include <stdio.h>

int main(){
    int pid;
    scanf("%d",&pid);
    message m;
    m.ml_i1 = pid;
    char data[17];
    m.ml_p1 = (char*) data;
    int ret = _syscall(PM_PROC_NR, GETFIELDS, &m);

    int i1 = 0;
    i1 |= (unsigned char) (data[0]) << 24 | (unsigned char)
(data[1]) << 16 | (unsigned char) (data[2]) << 8 | (unsigned char)
(data[3]);

    int i2 = 0;
    i2 |= (unsigned char) (data[4]) << 24 | (unsigned char)
(data[5]) << 16 | (unsigned char) (data[6]) << 8 | (unsigned char)
(data[7]);

    int i3 = 0;
    i3 |= (unsigned char) (data[8]) << 24 | (unsigned char)
(data[9]) << 16 | (unsigned char) (data[10]) << 8 | (unsigned char)
(data[11]);

    int i4 = 0;
    i4 |= (unsigned char) (data[12]) << 24 | (unsigned char)
(data[13]) << 16 | (unsigned char) (data[14]) << 8 | (unsigned char)
(data[15]);

    printf("%d %d %d %d", i1, i2, i3, i4);

    return 0;
}
```

Рисунок 4 – Исходный код программы для тестирования системного вызова

```

12 ? 0:00 um
6 ? 0:00 sched
1 ? 0:00 init
17 ? 0:00 /usr/sbin/pci
27 ? 0:00 /sbin/procfs
42 ? 0:00 /sbin/is
62 ? 0:00 /sbin/devman (null) /sys -o rw,rslabel=devman
77 ? 0:00 /usr/sbin/log
104 ? 0:00 /usr/sbin/lance instance=0
114 ? 0:00 /usr/sbin/ipc
118 ? 0:00 /usr/sbin/vbox
83 ? 0:00 devmand -d /etc/devmand -d /usr/pkg/etc/devmand
121 ? 0:00 update
123 ? 0:00 cron
128 ? 0:00 syslogd
131 ? 0:00 dhcpd
133 ? 0:00 nonamed -L
138 co 0:00 -sh
139 c1 0:00 getty
140 c2 0:00 getty
142 c3 0:00 getty
191 co 0:00 ps -x
# ./a.out
118
@ 5120257 1179648 37055#

```

Рисунок 5 – Результат тестирования системного вызова

5 Выводы

В ходе работы были получены навыки составления, объявления и использования системных вызовов в ОС «MINIX».