

# **Информационная безопасность. Отчет по лабораторной работе № 5**

**Дискреционное разграничение прав в Linux. Исследование влияния  
дополнительных атрибутов**

Горбунова Ярослава Михайловна

# Содержание

<b>1</b>	<b>Цель работы</b>	<b>5</b>
<b>2</b>	<b>Теоретическое введение</b>	<b>6</b>
2.1	Подготовка лабораторного стенда . . . . .	6
2.2	Компилирование программ . . . . .	7
<b>3</b>	<b>Выполнение лабораторной работы</b>	<b>11</b>
3.1	Создание программы . . . . .	11
3.2	Исследование Sticky-бита . . . . .	22
<b>4</b>	<b>Выводы</b>	<b>26</b>
<b>5</b>	<b>Список литературы</b>	<b>27</b>

# List of Figures

2.1	В системе установлен компилятор gcc . . . . .	6
2.2	Отключение системы запретов до очередной перезагрузки системы	7
3.1	Создание программы. Программа simpleid.c . . . . .	12
3.2	Создание программы. Пункты 2-5 . . . . .	12
3.3	Создание программы. Программа simpleid2.c . . . . .	14
3.4	Создание программы. Пункты 6-7 . . . . .	14
3.5	Создание программы. Пункты 8-10 . . . . .	15
3.6	Создание программы. Пункт 11 . . . . .	15
3.7	Создание программы. Пункт 12 . . . . .	16
3.8	Создание программы. Пункт 13. Программа readfile.c . . . . .	18
3.9	Создание программы. Пункт 14 . . . . .	18
3.10	Создание программы. Пункт 15 . . . . .	19
3.11	Создание программы. Пункт 16 . . . . .	19
3.12	Создание программы. Пункт 17 . . . . .	20
3.13	Создание программы. Пункт 18 . . . . .	21
3.14	Создание программы. Пункт 19 . . . . .	22
3.15	Исследование Sticky-бита. Пункты 1-3 . . . . .	23
3.16	Исследование Sticky-бита. Пункты 4-8 . . . . .	24
3.17	Исследование Sticky-бита. Пункты 9-14 . . . . .	25
3.18	Исследование Sticky-бита. Пункт 15 . . . . .	25

## List of Tables

# 1 Цель работы

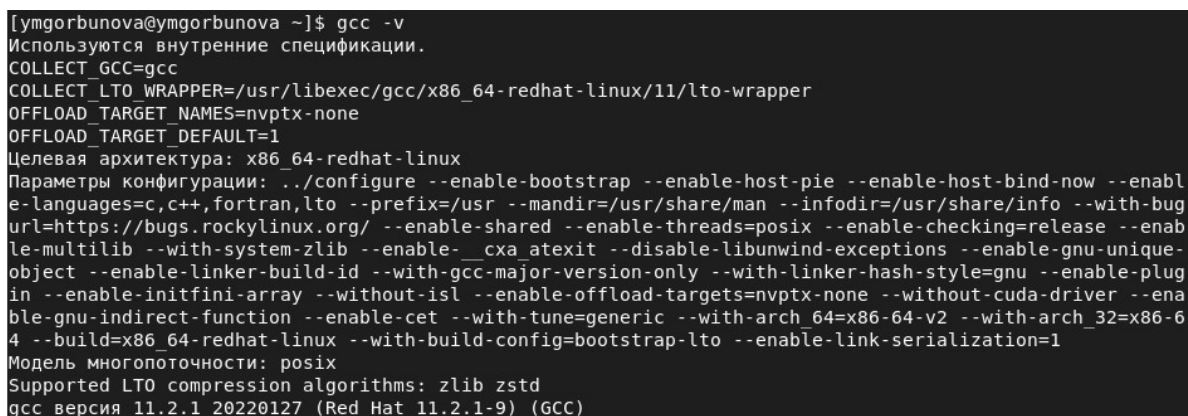
Изучение механизмов изменения идентификаторов, применения SetUID- и Sticky-битов. Получение практических навыков работы в консоли с дополнительными атрибутами. Рассмотрение работы механизма смены идентификатора процессов пользователей, а также влияние бита Sticky на запись и удаление файлов [1].

## 2 Теоретическое введение

### 2.1 Подготовка лабораторного стенда

Помимо прав администратора для выполнения части заданий потребуются средства разработки приложений. В частности, при подготовке стенда следует убедиться, что в системе установлен компилятор gcc (для этого, например, можно ввести команду `gcc -v` (fig. 2.1)). Если же gcc не установлен, то его необходимо установить, например, командой

`yum install gcc`

A screenshot of a terminal window showing the output of the 'gcc -v' command. The output includes the GCC version (11.2.1), the target architecture (x86\_64-redhat-linux), and various configuration options. The terminal text is as follows:

```
[ymgorbunova@ymgorbunova ~]$ gcc -v
Используются внутренние спецификации.
COLLECT_GCC=gcc
COLLECT_LTO_WRAPPER=/usr/libexec/gcc/x86_64-redhat-linux/11/lto-wrapper
OFFLOAD_TARGET_NAMES=nvptx-none
OFFLOAD_TARGET_DEFAULT=1
Целевая архитектура: x86_64-redhat-linux
Параметры конфигурации: ../configure --enable-bootstrap --enable-host-pie --enable-host-bind-now --enable-languages=c,c++,fortran,lto --prefix=/usr --mandir=/usr/share/man --infodir=/usr/share/info --with-bug-url=https://bugs.rockylinux.org/ --enable-shared --enable-threads=posix --enable-checking=release --enable-multilib --with-system-zlib --enable-__cxa_atexit --disable-libunwind-exceptions --enable-gnu-unique-object --enable-linker-build-id --with-gcc-major-version-only --with-linker-hash-style=gnu --enable-plugin --enable-initfini-array --without-isl --enable-offload-targets=nvptx-none --without-cuda-driver --enable-gnu-indirect-function --enable-cet --with-tune=generic --with-arch_64=x86_64-v2 --with-arch_32=x86_64 --build=x86_64-redhat-linux --with-build-config=bootstrap-lto --enable-link-serialization=1
Модель многопоточности: posix
Supported LTO compression algorithms: zlib zstd
gcc версия 11.2.1 20220127 (Red Hat 11.2.1-9) (GCC)
```

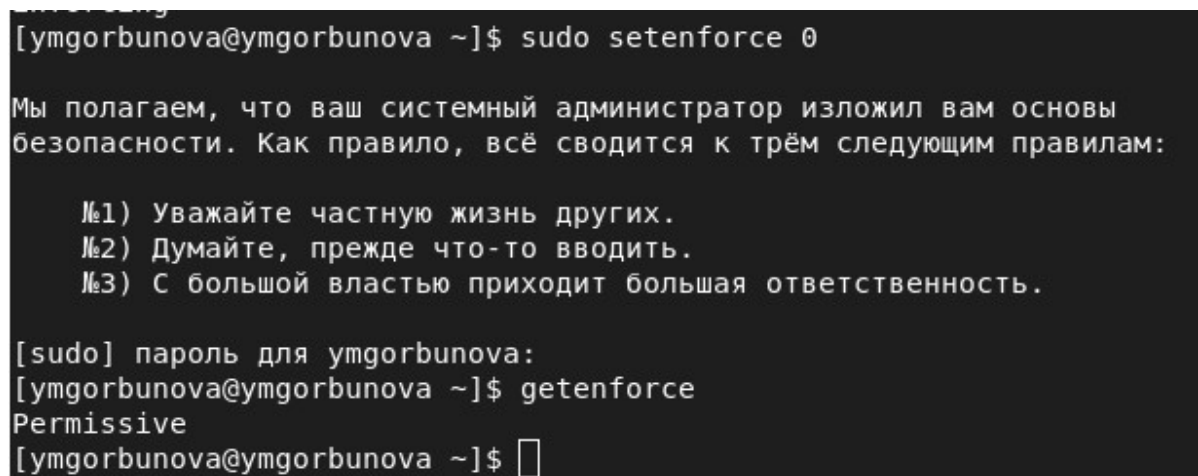
Figure 2.1: В системе установлен компилятор gcc

которая определит зависимости и установит следующие пакеты: gcc, cloogpppl, cpp, glibc-devel, glibc-headers, kernel-headers, libgomp, ppl, cloog-ppl, cpp, gcc, glibc-devel, glibc-headers, kernel-headers, libgomp, libstdc++-devel, mpfr, ppl, glibc, glibc-common, libgcc, libstdc++.

Файловая система, где располагаются домашние директории и файлы пользователей (в частности, пользователя `guest`), не должна быть смонтирована с опцией `nosuid`.

Так как программы с установленным битом `SetUID` могут представлять большую брешь в системе безопасности, в современных системах используются дополнительные механизмы защиты. Проследите, чтобы система защиты `SELinux` не мешала выполнению заданий работы. Если вы не знаете, что это такое, просто отключите систему запретов до очередной перезагрузки системы командой `setenforce 0`

После этого команда `getenforce` должна выводить `Permissive` (fig. 2.2). В этой работе система `SELinux` рассматриваться не будет.



```
[ymgorbunova@ymgorbunova ~]$ sudo setenforce 0

Мы полагаем, что ваш системный администратор изложил вам основы
безопасности. Как правило, всё сводится к трём следующим правилам:

    №1) Уважайте частную жизнь других.
    №2) Думайте, прежде что-то ввести.
    №3) С большой властью приходит большая ответственность.

[sudo] пароль для ymgorbunova:
[ymgorbunova@ymgorbunova ~]$ getenforce
Permissive
[ymgorbunova@ymgorbunova ~]$
```

Figure 2.2: Отключение системы запретов до очередной перезагрузки системы

## 2.2 Компилирование программ

Для выполнения четвёртой части задания вам потребуются навыки программирования, а именно, умение компилировать простые программы, написанные на языке `C` (`C++`), используя интерфейс `CLI`.

Само по себе создание программ не относится к теме, по которой выполняется работа, а является вспомогательной частью, позволяющей увидеть, как ре-

ализуются на практике те или иные механизмы дискреционного разграничения доступа. Если при написании (или исправлении существующих) скриптов на `bash`-е у большинства системных администраторов не возникает проблем, то процесс компилирования, как показывает практика, вызывает необоснованные затруднения.

Компиляторы, доступные в Linux-системах, являются частью коллекции GNU-компиляторов, известной как GCC (GNU Compiler Collection, подробнее см. <http://gcc.gnu.org>). В неё входят компиляторы языков C, C++, Java, Objective-C, Fortran и Chill. Будем использовать лишь первые два.

Компилятор языка C называется `gcc`. Компилятор языка C++ называется `g++` и запускается с параметрами почти так же, как `gcc`. Проверить это можно следующими командами:

```
whereis gcc whereis g++
```

Первый шаг заключается в превращении исходных файлов в объектный код:  
`gcc -c file.c`

В случае успешного выполнения команды (отсутствие ошибок в коде) полученный объектный файл будет называться `file.o`.

Объектные файлы невозможно запускать и использовать, поэтому после компиляции для получения готовой программы объектные файлы необходимо скомпоновать. Компоновать можно один или несколько файлов. В случае использования хотя бы одного из файлов, написанных на C++, компоновка производится с помощью компилятора `g++`. Строго говоря, это тоже не вполне верно. Компоновка объектного кода, сгенерированного чем бы то ни было (хоть вручную), производится линкером `ld`, `g++` его просто вызывает изнутри. Если же все файлы написаны на языке C, нужно использовать компилятор `gcc`.

Например, так: `gcc -o program file.o`

В случае успешного выполнения команды будет создана программа `program` (исполняемый файл формата ELF с установленным атрибутом `+x`).

Компилирование — это процесс. Компилятор `gcc` (`g++`) имеет множество па-



раметров, влияющих на процесс компиляции. Он поддерживает различные режимы оптимизации, выбор платформы назначения и пр.

Также возможно использование make-файлов (Makefile) с помощью утилиты make для упрощения процесса компиляции.

Такое решение подойдёт лишь для простых случаев. Если говорить про пример выше, то компилирование одного файла из двух шагов можно сократить вообще до одного, например: `gcc file.c`

В этом случае готовая программа будет иметь название `a.out`.

Механизм компилирования программ в данной работе не мог быть не рассмотрен потому, что использование программ, написанных на `bash`, для изучения SetUID- и SetGID- битов, не представляется возможным. Связано это с тем, что любая `bash`-программа интерпретируется в процессе своего выполнения, т.е. существует сторонняя программа-интерпретатор, которая выполняет считывание файла сценария и выполняет его последовательно. Сам интерпретатор выполняется с правами пользователя, его запустившего, а значит, и выполняемая программа использует эти права.

При этом интерпретатору абсолютно всё равно, установлены SetUID-, SetGID-биты у текстового файла сценария, атрибут разрешения запуска «x» или нет. Важно, чтобы был установлен лишь атрибут, разрешающий чтение «r».

Также не важно, был ли вызван интерпретатор из командной строки (запуск файла, как `bash file1.sh`), либо внутри файла была указана строчка `#!/bin/bash`.

Логично спросить: если установление SetUID- и SetGID- битов на сценарий не приводит к нужному результату как с исполняемыми файлами, то что мешает установить эти биты на сам интерпретатор? Ничего не мешает, только их установление приведёт к тому, что, так как владельцем `/bin/bash` является `root`: `ls -l /bin/bash` все сценарии, выполняемые с использованием `/bin/bash`, будут иметь возможности суперпользователя — совсем не тот результат, который хотелось бы видеть.

Если сомневаетесь в выше сказанном, создайте простой файл `progl.sh` следую-

щего содержания: `#!/bin/bash /usr/bin/id /usr/bin/whoami` и попробуйте поменять его атрибуты в различных конфигурациях.

Подход вида: сделать копию `/bin/bash`, для нее `chown user:users` и потом SUID также плох, потому что это позволит запускать любые команды от пользователя `user`.

## 3 Выполнение лабораторной работы

### 3.1 Создание программы

1. Войдите в систему от имени пользователя guest.
2. Создайте программу simpleid.c (fig. 3.2, fig. 3.1):

```
#include <sys/types.h>

#include <unistd.h>

#include <stdio.h>

int

main ()

{

    uid_t uid = geteuid ();

    gid_t gid = getegid ();

    printf ("uid=%d, gid=%d\n", uid, gid);
```

```
return 0;
```

```
}
```



```
1 #include <sys/types.h>
2 #include <unistd.h>
3 #include <stdio.h>
4 int
5 main ()
6 {
7     uid_t uid = geteuid ();
8     gid_t gid = getegid ();
9     printf ("uid=%d, gid=%d\n", uid, gid);
10    return 0;
11 }
```

Figure 3.1: Создание программы. Программа simpleid.c

3. Скомпилируйте программу и убедитесь, что файл программы создан (fig. 3.2): `gcc simpleid.c -o simpleid`
4. Выполните программу simpleid (fig. 3.2): `./simpleid`
5. Выполните системную программу id (fig. 3.2): `id` и сравните полученный вами результат с данными предыдущего пункта задания.



```
[guest@ymgorbunova dir1]$ touch simpleid.c
[guest@ymgorbunova dir1]$ gcc simpleid.c -o simpleid
[guest@ymgorbunova dir1]$ ls -l
итого 32
-rwxrwxr-x. 1 guest guest 25904 сен 28 14:49 simpleid
-rw-rw-r--. 1 guest guest 175 сен 28 14:49 simpleid.c
[guest@ymgorbunova dir1]$ ./simpleid
uid=1001, gid=1001
[guest@ymgorbunova dir1]$ id
uid=1001(guest) gid=1001(guest) группы=1001(guest) контекст=unconfined_u:unconfined_r:unconfined_t:s0-s0:c0.c1023
```

Figure 3.2: Создание программы. Пункты 2-5

6. Усложните программу, добавив вывод действительных идентификаторов (fig. 3.4, fig. 3.3):

```
#include <sys/types.h>

#include <unistd.h>

#include <stdio.h>

int

main ()

{

    uid_t real_uid = getuid ();

    uid_t e_uid = geteuid ();

    gid_t real_gid = getgid ();

    gid_t e_gid = getegid () ;

    printf ("e_uid=%d, e_gid=%d\n", e_uid, e_gid);

    printf ("real_uid=%d, real_gid=%d\n", real_uid, real_gid);

    return 0;

}
```

Получившуюся программу назовите simpleid2.c.

Figure 3.3: Создание программы. Программа simpleid2.c

7. Скомпилируйте и запустите simpleid2.c (fig. 3.4):

```
gcc simpleid2.c -o simpleid2
./simpleid2
```

Figure 3.4: Создание программы. Пункты 6-7

8. От имени суперпользователя выполните команды (fig. 3.5):

```
chown root:guest /home/guest/simpleid2
```

```
chmod u+s /home/guest/simpleid2
```

9. Используйте `sudo` или повысьте временно свои права с помощью `su`. Поясните, что делают эти команды (fig. 3.5). – Первая команда меняет владельца файла на `root`, вторая - устанавливает UID-бит.

10. Выполните проверку правильности установки новых атрибутов и смены владельца файла `simpleid2` (fig. 3.5): `ls -l simpleid2`

```
[root@ymgorbunova ~]# chown root:guest /home/guest/dirl/simpleid2
[root@ymgorbunova ~]# chmod u+s /home/guest/dirl/simpleid2
[root@ymgorbunova ~]# ls -l simpleid2
ls: невозможно получить доступ к 'simpleid2': Нет такого файла или каталога
[root@ymgorbunova ~]# ls -l /home/guest/dirl/simpleid2
-rwsrwxr-x. 1 root guest 26008 сен 28 14:53 /home/guest/dirl/simpleid2
```

Figure 3.5: Создание программы. Пункты 8-10

11. Запустите `simpleid2` и `id` (fig. 3.6):

```
./simpleid2
```

```
id
```

Сравните результаты. – Результаты полностью совпали.

```
[root@ymgorbunova dirl]# ./simpleid2
e_uid=0, e_gid=0
real_uid=0, real_gid=0
[root@ymgorbunova dirl]# id
uid=0(root) gid=0(root) группы=0(root) контекст=unconfined_u:unconfined_r:unconfined_t:s0-s0:c0.c1023
```

Figure 3.6: Создание программы. Пункт 11

12. Проделайте тоже самое относительно SetGID-бита (fig. 3.7).

```

[root@ymgorbunova dir1]# chmod g+s /home/guest/simpleid2
chmod: невозможно получить доступ к '/home/guest/simpleid2': Нет такого файла или каталога
[root@ymgorbunova dir1]# chmod g+s simpleid2
[root@ymgorbunova dir1]# ls -l simpleid2
-rwsrwsr-x. 1 root guest 26008 сен 28 14:53 simpleid2
[root@ymgorbunova dir1]# ls -l
итого 64
-rwxrwxr-x. 1 guest guest 25904 сен 28 14:49 simpleid
-rwsrwsr-x. 1 root guest 26008 сен 28 14:53 simpleid2
-rw-rw-r--. 1 guest guest 303 сен 28 14:52 simpleid2.c
-rw-rw-r--. 1 guest guest 175 сен 28 14:49 simpleid.c
[root@ymgorbunova dir1]# ./simpleid2
e_uid=0, e_gid=1001
real_uid=0, real_gid=0
[root@ymgorbunova dir1]# id
uid=0(root) gid=0(root) группы=0(root) контекст=unconfined_u:unconfined_r:unconfined_t:s0-s0:c0.c1023

```

Figure 3.7: Создание программы. Пункт 12

13. Создайте программу readfile.c (fig. 3.8):

```

#include <fcntl.h>

#include <stdio.h>

#include <sys/stat.h>

#include <sys/types.h>

#include <unistd.h>

int

main (int argc, char* argv[])

{

    unsigned char buffer[16];

    size_t bytes_read;

```



```
int i;

int fd = open (argv[1], O_RDONLY);

do

{

bytes_read = read (fd, buffer, sizeof (buffer));

for (i =0; i < bytes_read; ++i) printf("%c", buffer[i]);

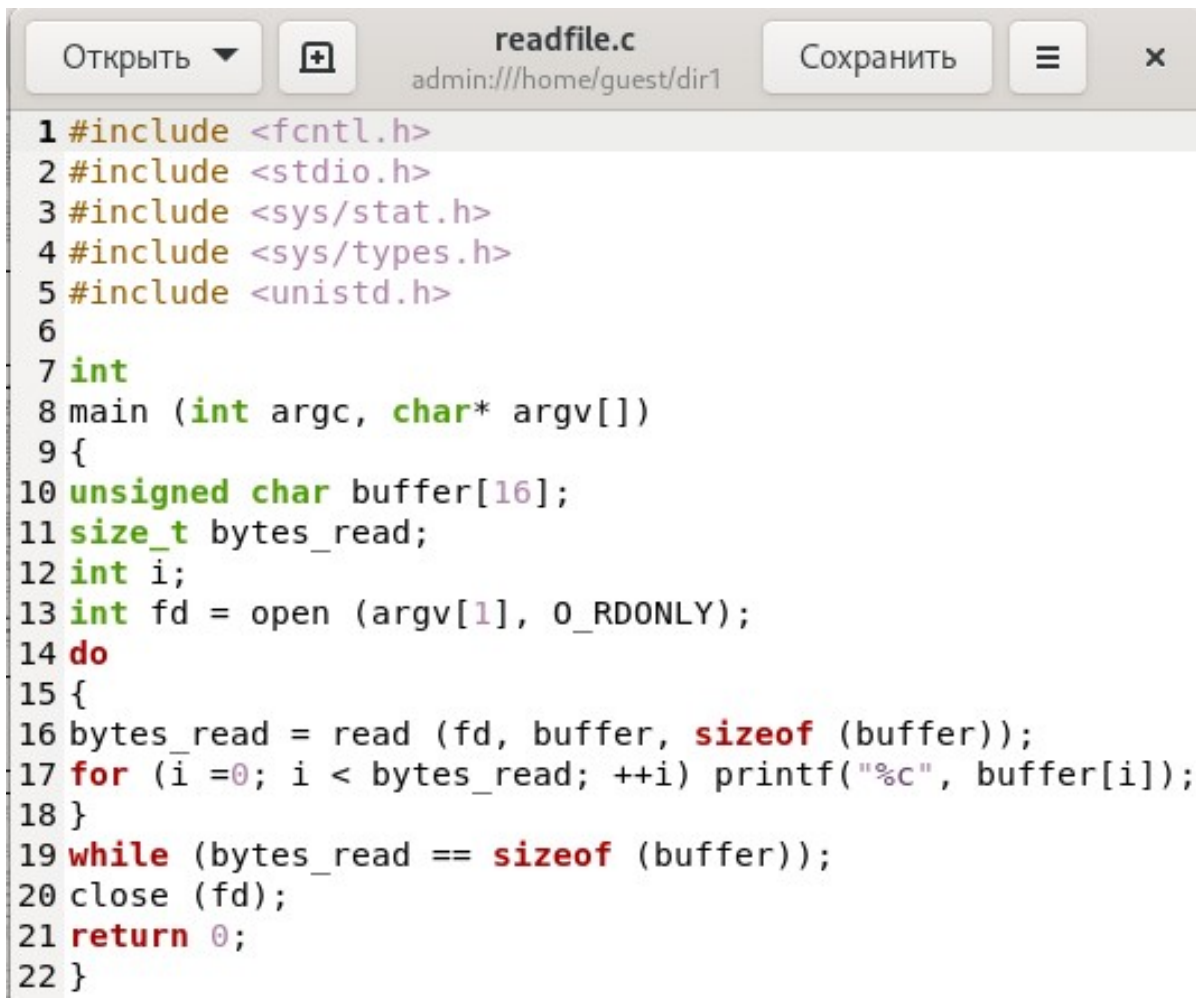
}

while (bytes_read == sizeof (buffer));

close (fd);

return 0;

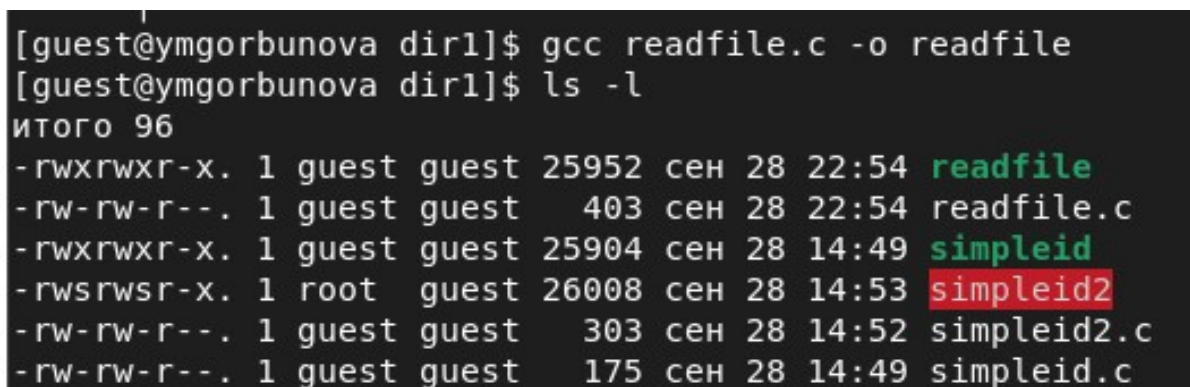
}
```



```
1 #include <fcntl.h>
2 #include <stdio.h>
3 #include <sys/stat.h>
4 #include <sys/types.h>
5 #include <unistd.h>
6
7 int
8 main (int argc, char* argv[])
9 {
10 unsigned char buffer[16];
11 size_t bytes_read;
12 int i;
13 int fd = open (argv[1], O_RDONLY);
14 do
15 {
16 bytes_read = read (fd, buffer, sizeof (buffer));
17 for (i = 0; i < bytes_read; ++i) printf("%c", buffer[i]);
18 }
19 while (bytes_read == sizeof (buffer));
20 close (fd);
21 return 0;
22 }
```

Figure 3.8: Создание программы. Пункт 13. Программа readfile.c

14. Откомпилируйте её (fig. 3.9). gcc readfile.c -o readfile



```
[guest@ymgorbunova dir1]$ gcc readfile.c -o readfile
[guest@ymgorbunova dir1]$ ls -l
итого 96
-rwxrwxr-x. 1 guest guest 25952 сен 28 22:54 readfile
-rw-rw-r--. 1 guest guest 403 сен 28 22:54 readfile.c
-rwxrwxr-x. 1 guest guest 25904 сен 28 14:49 simpleid
-rwsrwsr-x. 1 root guest 26008 сен 28 14:53 simpleid2
-rw-rw-r--. 1 guest guest 303 сен 28 14:52 simpleid2.c
-rw-rw-r--. 1 guest guest 175 сен 28 14:49 simpleid.c
```

Figure 3.9: Создание программы. Пункт 14

15. Смените владельца у файла readfile.c (или любого другого текстового файла в системе) и измените права так, чтобы только суперпользователь (root) мог прочитать его, а guest не мог (fig. 3.10).

```
[root@ymgorbunova dir1]# chown root:guest readfile.c
[root@ymgorbunova dir1]# ls -l
итого 96
-rwxrwxr-x. 1 guest guest 25952 сен 28 22:54 readfile
-rw-rw-r--. 1 root guest 403 сен 28 22:54 readfile.c
-rwxrwxr-x. 1 guest guest 25904 сен 28 14:49 simpleid
-rwsrwsr-x. 1 root guest 26008 сен 28 14:53 simpleid2
-rw-rw-r--. 1 guest guest 303 сен 28 14:52 simpleid2.c
-rw-rw-r--. 1 guest guest 175 сен 28 14:49 simpleid.c
[root@ymgorbunova dir1]# chmod 600 readfile.c
[root@ymgorbunova dir1]# ls -l
итого 96
-rwxrwxr-x. 1 guest guest 25952 сен 28 22:54 readfile
-rw-----. 1 root guest 403 сен 28 22:54 readfile.c
-rwxrwxr-x. 1 guest guest 25904 сен 28 14:49 simpleid
-rwsrwsr-x. 1 root guest 26008 сен 28 14:53 simpleid2
-rw-rw-r--. 1 guest guest 303 сен 28 14:52 simpleid2.c
-rw-rw-r--. 1 guest guest 175 сен 28 14:49 simpleid.c
```

Figure 3.10: Создание программы. Пункт 15

16. Проверьте, что пользователь guest не может прочитать файл readfile.c (fig. 3.11).

```
[guest@ymgorbunova dir1]$ cat readfile.c
cat: readfile.c: Отказано в доступе
```

Figure 3.11: Создание программы. Пункт 16

17. Смените у программы readfile владельца и установите SetU'D-бит (fig. 3.12).

```

[root@ymgorbunova dir1]# chown root:guest readfile
[root@ymgorbunova dir1]# chmod u+s readfile
[root@ymgorbunova dir1]# ls -l
итого 96
-rwsrwxr-x. 1 root  guest 25952 сен 28 22:54 readfile
-rw-----. 1 root  guest  403 сен 28 22:54 readfile.c
-rwxrwxr-x. 1 guest  guest 25904 сен 28 14:49 simpleid
-rwsrwsr-x. 1 root  guest 26008 сен 28 14:53 simpleid2
-rw-rw-r--. 1 guest  guest  303 сен 28 14:52 simpleid2.c
-rw-rw-r--. 1 guest  guest  175 сен 28 14:49 simpleid.c

```

Figure 3.12: Создание программы. Пункт 17

18. Проверьте, может ли программа readfile прочитать файл readfile.c (fig. 3.13)? – Программа может прочитать файл.

```
[root@ymgorbunova dir1]# ./readfile readfile.c
#include <fcntl.h>
#include <stdio.h>
#include <sys/stat.h>
#include <sys/types.h>
#include <unistd.h>

int
main (int argc, char* argv[])
{
    unsigned char buffer[16];
    size_t bytes_read;
    int i;
    int fd = open (argv[1], O_RDONLY);
    do
    {
        bytes_read = read (fd, buffer, sizeof (buffer));
        for (i = 0; i < bytes_read; ++i) printf("%c", buffer[i]);
    }
    while (bytes_read == sizeof (buffer));
    close (fd);
    return 0;
}
```

Figure 3.13: Создание программы. Пункт 18

19. Проверьте, может ли программа readfile прочитать файл /etc/shadow (fig. 3.14)? – Программа может прочитать файл. Отрадите полученный результат и ваши объяснения в отчёте.

```
[root@ymgorbunova dir1]# ./readfile /etc/shadow
root:$6$67zd8mWm4E6RCrcf$A8rm6YQPZ5XDKNFqtDM7EKLZXLeF.M6t15rt3QkhsJzVNNQA2mFMpy9/zzYKR.hAtnwzc449IRRBv7z
tx0oQ4.::0:99999:7:::
bin:!:19123:0:99999:7:::
daemon:!:19123:0:99999:7:::
adm:!:19123:0:99999:7:::
lp:!:19123:0:99999:7:::
sync:!:19123:0:99999:7:::
shutdown:!:19123:0:99999:7:::
halt:!:19123:0:99999:7:::
mail:!:19123:0:99999:7:::
operator:!:19123:0:99999:7:::
games:!:19123:0:99999:7:::
ftp:!:19123:0:99999:7:::
nobody:!:19123:0:99999:7:::
systemd-coredump:!:19242::::
dbus:!:19242::::

```

Figure 3.14: Создание программы. Пункт 19

## 3.2 Исследование Sticky-бита

1. Выясните, установлен ли атрибут Sticky на директории /tmp, для чего выполните команду (fig. 3.15) `ls -l / | grep tmp`
2. От имени пользователя guest создайте файл file01.txt в директории /tmp со словом test (fig. 3.15): `echo "test" > /tmp/file01.txt`
3. Просмотрите атрибуты у только что созданного файла и разрешите чтение и запись для категории пользователей «все остальные» (fig. 3.15):

```
ls -l /tmp/file01.txt
chmod o+rw /tmp/file01.txt
ls -l /tmp/file01.txt
```

```
[guest@ymgorbunova ~]$ ls -l / | grep tmp
drwxrwxrwt. 17 root root 4096 сен 28 23:36 tmp
[guest@ymgorbunova ~]$ echo "test" > /tmp/file01.txt
[guest@ymgorbunova ~]$ ls -l /tmp/file01.txt
-rw-rw-r--. 1 guest guest 5 сен 28 23:40 /tmp/file01.txt
[guest@ymgorbunova ~]$ chmod o+rw /tmp/file01.txt
[guest@ymgorbunova ~]$ ls -l /tmp/file01.txt
-rw-rw-rw-. 1 guest guest 5 сен 28 23:40 /tmp/file01.txt
[guest@ymgorbunova ~]$
```

Figure 3.15: Исследование Sticky-бита. Пункты 1-3

4. От пользователя guest2 (не являющегося владельцем) попробуйте прочит-  
тать файл /tmp/file01.txt (fig. 3.16): cat /tmp/file01.txt
5. От пользователя guest2 попробуйте дозаписать в файл /tmp/file01.txt слово  
test2 командой (fig. 3.16)

echo "test2" » /tmp/file01.txt

Удалось ли вам выполнить операцию? – Удалось.

6. Проверьте содержимое файла командой (fig. 3.16) cat /tmp/file01.txt
7. От пользователя guest2 попробуйте записать в файл /tmp/file01.txt слово  
test3, стерев при этом всю имеющуюся в файле информацию командой  
(fig. 3.16)

echo "test3" > /tmp/file01.txt

Удалось ли вам выполнить операцию? – Удалось.

8. Проверьте содержимое файла командой (fig. 3.16) cat /tmp/file01.txt

```
[guest2@ymgorbunova ~]$ cat /tmp/file01.txt
test
[guest2@ymgorbunova ~]$ echo "test2" >> /tmp/file01.txt
[guest2@ymgorbunova ~]$ cat /tmp/file01.txt
test
test2
[guest2@ymgorbunova ~]$ echo "test3" > /tmp/file01.txt
[guest2@ymgorbunova ~]$ cat /tmp/file01.txt
test3
```

Figure 3.16: Исследование Sticky-бита. Пункты 4-8

9. От пользователя guest2 попробуйте удалить файл /tmp/file01.txt командой (fig. 3.17) `rm /tmp/file01.txt`

Удалось ли вам удалить файл? – Не удалось.

10. Повысьте свои права до суперпользователя следующей командой (fig. 3.17) `su -` и выполните после этого команду, снимающую атрибут t (Sticky-бит) с директории /tmp: `chmod -t /tmp`
11. Покиньте режим суперпользователя командой (fig. 3.17) `exit`
12. От пользователя guest2 проверьте, что атрибута t у директории /tmp нет (fig. 3.17): `ls -l / | grep tmp`
13. Повторите предыдущие шаги (fig. 3.17). Какие наблюдаются изменения? – Удаётся выполнить удаление файла.
14. Удаётся ли вам удалить файл от имени пользователя, не являющегося его владельцем? Ваши наблюдения занесите в отчёт (fig. 3.17). – Удаётся выполнить удаление файла.



```

[guest2@ymgorbunova ~]$ rm /tmp/file01.txt
rm: невозможно удалить '/tmp/file01.txt': Операция не позволена
[guest2@ymgorbunova ~]$ su
Пароль:
[root@ymgorbunova guest2]# chmod -t /tmp
[root@ymgorbunova guest2]# exit
exit
[guest2@ymgorbunova ~]$ ls -l / | grep tmp
drwxrwxrwx. 17 root root 4096 сен 28 23:45 tmp
[guest2@ymgorbunova ~]$ rm /tmp/file01.txt
[guest2@ymgorbunova ~]$ ls -l file01.txt
ls: невозможно получить доступ к 'file01.txt': Нет такого файла или каталога
[guest2@ymgorbunova ~]$ ls -l /tmp/file01.txt
ls: невозможно получить доступ к '/tmp/file01.txt': Нет такого файла или каталога

```

Figure 3.17: Исследование Sticky-бита. Пункты 9-14

15. Повысьте свои права до суперпользователя и верните атрибут `t` на директорию `/tmp` (fig. 3.18):

`su -`

`chmod +t /tmp`

`exit`

```

[guest2@ymgorbunova ~]$ su
Пароль:
[root@ymgorbunova guest2]# chmod +t /tmp
[root@ymgorbunova guest2]# ls -l / | grep tmp
drwxrwxrwt. 17 root root 4096 сен 28 23:47 tmp
[root@ymgorbunova guest2]# exit
exit
[guest2@ymgorbunova ~]$ █

```

Figure 3.18: Исследование Sticky-бита. Пункт 15

## 4 Выводы

Изучены механизмы изменения идентификаторов, применения SetUID- и Sticky-битов. Получены практические навыки работы в консоли с дополнительными атрибутами. Рассмотрена работа механизма смены идентификатора процессов пользователей, а также влияние бита Sticky на запись и удаление файлов.

## **5 Список литературы**

1. Методические материалы курса