ОПИСАНИЕ ИТОГОВОГО ЗАДАНИЯ К МОДУЛЮ 18

По условию задания требовалось написать программу, которая сможет считывать из файла при своей загрузке и записывать в файл состояния объектов некоторых классов, скомпилировать и собрать программу в Linux, и с помощью программного кода установить права доступа к файлам записи данных.

Для реализации задания был выбран код чата, написанный мной к итоговому заданию модуля 16, правда, в несколько сокращенном и упрощенном виде – был исключен класс общих сообщений, а также некоторые вспомогательные функции. В остальном же структура программы осталась той же – классы User, Message, Hash, разделенные на заголовочные файлы и файлы ресурсов, блок main, и дополнительный заголовочный файл Fs.h для определения функции demo_perms из библиотеки <filesystem>. Для сохранения и считывания данных были созданы текстовые файлы hashTable.txt (сохранение логинов и паролей), ListOfUsers.txt (сохранение списка пользователей) и Messages.txt (сохранение сообщений). Запись и чтение данных из этих файлов осуществлялись методами соответствующих классов.

Теперь о том, что касается функции установления прав доступа. В пособии приведен такой код:

```
#include <filesystem>
namespace fs = std::filesystem;

void demo_perms(fs::perms p)
{
   std::cout << ((p & fs::perms::owner_read) != fs::perms::none ? "r" : "-")
      << ((p & fs::perms::owner_write) != fs::perms::none ? "w" : "-")
      << ((p & fs::perms::owner_exec) != fs::perms::none ? "x" : "-")
      << ((p & fs::perms::group_read) != fs::perms::none ? "r" : "-")
      << ((p & fs::perms::group_write) != fs::perms::none ? "w" : "-")
      << ((p & fs::perms::group_exec) != fs::perms::none ? "x" : "-")
      << ((p & fs::perms::others_read) != fs::perms::none ? "r" : "-")
      << ((p & fs::perms::others_write) != fs::perms::none ? "w" : "-")
      << ((p & fs::perms::others_exec) != fs::perms::none ? "x" : "-")
      << '\n';
}
int main()
{
```

```
    std::ofstream("test.txt"); // create file
    std::cout << "Created file with permissions: ";
    demo_perms(fs::status("test.txt").permissions());
    fs::permissions("test.txt", fs::perms::group_write| fs::perms::others_all)
    std::cout << "After removing g-w and o-r:  ";
    demo_perms(fs::status("test.txt").permissions());
    fs::remove("test.txt");
}
```

Экспериментальным путем было установлено, что вся «начинка» тела функции `demo_perms(fs::perms p)` реально на состояние прав не влияет – она только выводит на экран что-то вроде «до» и «после», поэтому ее в принципе можно целиком удалить, превратив функцию в подобие конструктора:

`demo_perms(fs::perms p){}`

Реально же действуют на состояние прав только две выделенные красным строки, в которых задается имя файла и состояние прав доступа для каждой группы в соответствии с таблицей

Константа члена	Значение (восьмеричное)	Эквивалент POSIX	Значение
none	0		Биты разрешений не установлены
owner_read	0400	S_IRUSR	Владелец файла имеет разрешение на чтение
owner_write	0200	S_IWUSR	Владелец файла имеет разрешение на запись
owner_exec	0100	S_IXUSR	Владелец файла имеет разрешение на выполнение / поиск
owner_all	0700	S_IRWXU	Владелец файла имеет права на чтение, запись и выполнение / поиск Эквивалент owner_read | owner_write | owner_exec
group_read	040	S_IRGRP	У группы пользователей файла есть разрешение на чтение
group_write	020	S_IWGRP	У группы пользователей файла есть разрешение на запись
group_exec	010	S_IXGRP	У группы пользователей файла есть разрешение на выполнение / поиск
group_all	070	S_IRWXG	Группа пользователей файла имеет права на чтение, запись и выполнение / поиск Эквивалентно group_read | group_write | group_exec
others_read	04	S_IROTH	У других пользователей есть разрешение на чтение
others_write	02	S_IWOTH	У других пользователей есть разрешение на запись
others_exec	01	S_IXOTH	У других пользователей есть разрешение на выполнение / поиск
others_all	07	S_IRWXO	Другие пользователи имеют разрешения на чтение, запись и выполнение / поиск Эквивалентно others_read | others_write | others_exec
all	0777		Все пользователи имеют права на чтение, запись и выполнение / поиск Эквивалентно owner_all | group_all | others_all

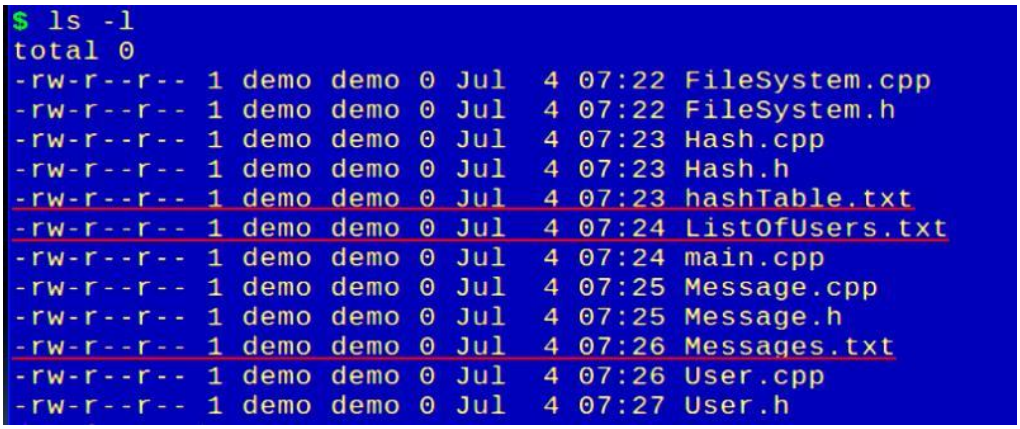
(Сайт https://en.cppreference.com/w/cpp/filesystem/perms)

Поэтому для всех трех текстовых файлов было записано так:

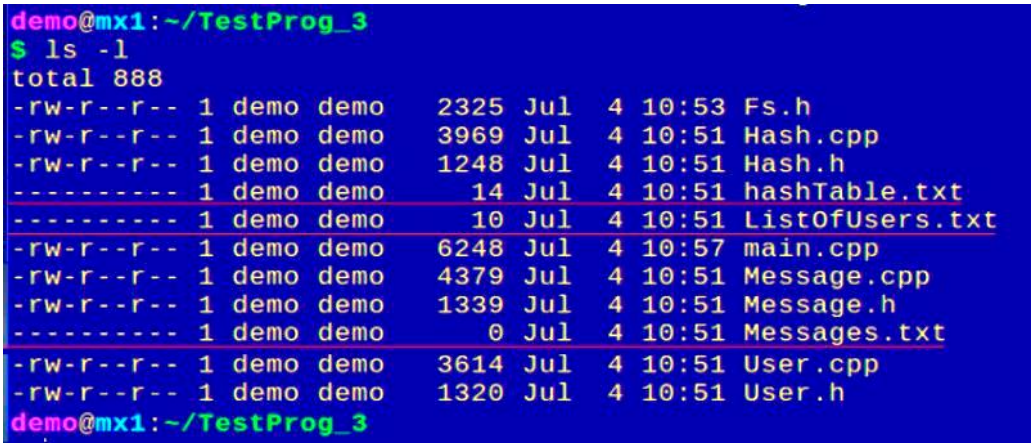
```
fs::permissions("ListOfUsers.txt", fs::perms::owner_read |
fs::perms::owner_write | fs::perms::group_read | fs::perms::group_write |
fs::perms::others_read | fs::perms::others_write);
```

```
demo_perms(fs::status("ListOfUsers.txt").permissions());

fs::permissions("Messages.txt", fs::perms::owner_read |
fs::perms::owner_write | fs::perms::group_read | fs::perms::group_write |
fs::perms::others_read | fs::perms::others_write);
demo_perms(fs::status("Messages.txt").permissions());

fs::permissions("hashTable.txt", fs::perms::owner_read |
fs::perms::owner_write | fs::perms::group_read | fs::perms::group_write |
fs::perms::others_read | fs::perms::others_write);
demo_perms(fs::status("hashTable.txt").permissions());
```

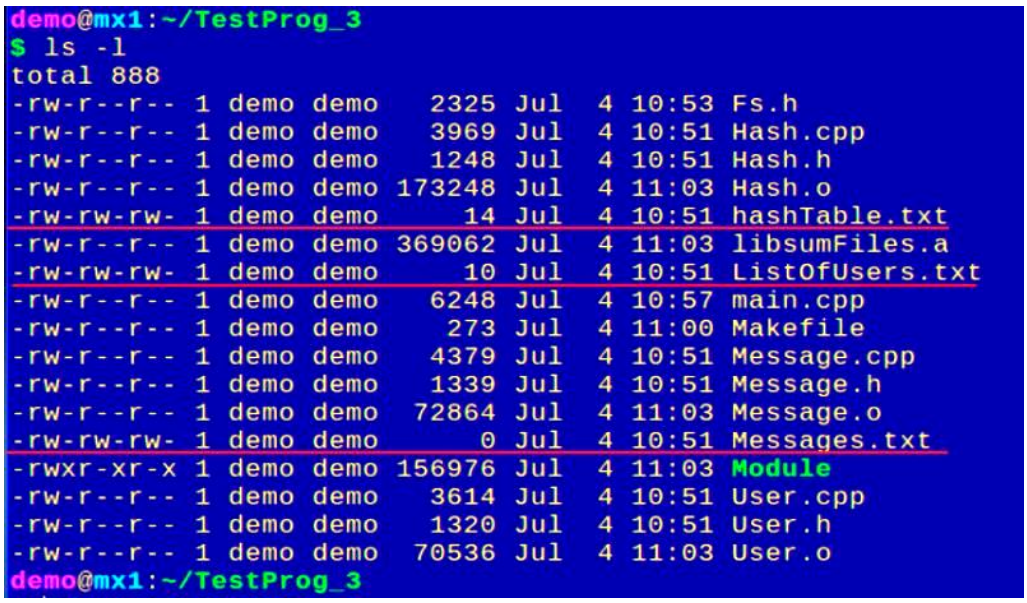
После создания всех нужных файлов командой touch права доступа по умолчанию выглядели так:



Затем командой `sudo chmod a=-` все права текстовых файлов были удалены:



После чего после запуска программы восстановились в соответствии с заданным условием:



Что касается строки `std::ofstream("test.txt").` Во-первых, создание текстовых файлов и запись данных в них предусмотрены методами соответствующих классов. Во-вторых, в таком виде при каждом запуске программы текстовые файлы будут пересоздаваться заново, и все данные в них будут потеряны. Чтобы избежать этого, в аргумент нужно добавить флаг `std::ios::app.` Либо вообще убрать эту строку, а все текстовые файлы создать заранее командой `touch.`

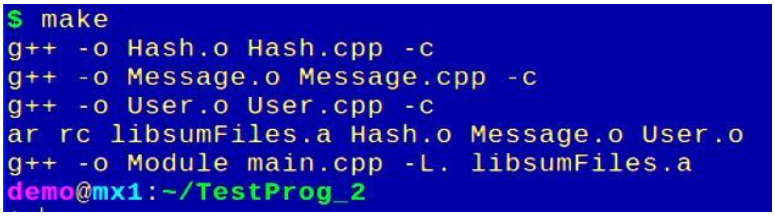
Все строки установки прав были упакованы в функцию `void Settings(),` куда была добавлена опция вызова функции по вводу дополнительного пароля администратора. Конечно, примитивно, но я уже не стал заморачиваться с дополнительной хеш-таблицей. Крайне желательно было бы также иметь возможность при вызове функции задавать любые параметры прав для каждой группы в соответствии с таблицей, не залезая в программный код. Но как это сделать, я к сожалению, не нашел способа.

Сборка и линковка программы были осуществлены с помощью файла `Makefile:`

```
Module: main.cpp lib
	g++ -o Module main.cpp -L. libsumFiles.a

lib:	Fs.h Hash.cpp main.cpp Message.h User.cpp Hash.h Message.cpp User.h
	g++ -o Hash.o Hash.cpp -c
	g++ -o Message.o Message.cpp -c
```

```
	g++ -o User.o User.cpp -c
	ar rc libsumFiles.a Hash.o Message.o User.o
clean:
	rm *.o *.a
install:
	install ./Module /usr/local/bin
```

После выполнения команды `make` (еще без добавления целей) программа скомпилировалась



И затем, после добавления целей и выполнения команд `make clean` и `make install` получился список файлов

Как видно, после сборки остался только исполняемый файл Module и файлы-исходники. Объектные файлы и файл библиотеки были удалены.

Файлы в `Linux` я заполнял, открывая их в редакторе `vim` и копируя в них код из файлов `Windows`. Но тут следует иметь в виду, что `Linux` не принимает директиву `#pragma once` (считает ошибкой, приходилось ее удалять). Также компилятор `Linux` ругается на библиотеку `<Windows.h>`, которая необходима для работы функций `SetConsoleCP(1251)` и `SetConsoleOutputCP(1251)` в блоке `main.`