

МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ ДНР
«ДОНЕЦКИЙ НАЦИОНАЛЬНЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ»
КАФЕДРА ПРОГРАМНОЙ ИНЖЕНЕРИИ

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА
К КУРСОВОМУ ПРОЕКТУ

на тему:

«Проектирование гипотетической операционной системы»

по курсу:

«Операционные системы»

Руководитель:

Чернышова А. В. _____

Московченко А. В. _____

Выполнил:

студент 3 курса, группы ПИ-186

Набатов А. В. _____

РЕФЕРАТ

Пояснительная записка к курсовому проекту содержит: 89 страниц, 20 рисунков, 19 таблиц, 2 приложения, 2 источника.

Объект исследования – принцип работы операционной системы и эмуляция работы планировщика процессов.

Задача исследования – изучить особенности работы операционных систем.

Цель исследования – спроектировать гипотетическую операционную систему и выполнить эмуляцию работы планировщика процессов.

Результат – программа, которая эмулирует работу файловой системы и планировщика процессов.

ОПЕРАЦИОННАЯ СИСТЕМА, ФАЙЛОВАЯ СИСТЕМА, FFS, ФАЙЛ, ИНОД, СУПЕРБЛОК, КЛАСТЕР ПОЛЬЗОВАТЕЛЬ, ГРУППА, ПРАВА ДОСТУПА, ПЛАНИРОВЩИК ПРОЦЕССОВ, ПРИОРИТЕТ, КВАНТОВАНИЕ ВРЕМЕНИ

СОДЕРЖАНИЕ

Введение	5
1 СТРУКТУРА ПРОЕКТИРУЕМОЙ ФАЙЛОВОЙ СИСТЕМЫ	6
1.1 Общая организация файловой системы	6
1.2 Виртуальные страницы	11
1.3 Команды для работы с ФС	13
1.4 Системные вызовы для работы с ФС	17
1.5 Способы организации файлов	19
1.6 Алгоритмы работы некоторых системных вызовов ФС.	20
2 ПРОЦЕССЫ В ОПЕРАЦИОННОЙ СИСТЕМЕ	22
2.1 Команды для работы с процессами.....	22
2.2 Системные вызовы управления процессами	26
2.3 Диаграмма состояний процесса	26
2.4 Приоритеты процессов	28
2.5 Межпроцессное взаимодействие.....	28
2.6 Выбор дисциплины обслуживания планировщика процессов. Алгоритм работы планировщика процессов в соответствии с выбранной дисциплиной обслуживания	35
3 РЕЖИМЫ РАБОТЫ ПРОЕКТИРУЕМОЙ ОС	40
3.1 Мультипрограммный режим работы ОС	40
3.2 Многопользовательская защита	40
3.3 Интерактивный режим работы ОС	41
4 СТРУКТУРА ОПЕРАЦИОННОЙ СИСТЕМЫ	42
4.1 Общая структура проектируемой ОС	42
4.2 Структура ядра проектируемой ОС. Основные функции и назначение файловой подсистемы, подсистемы управления памятью и процессами, подсистемы управления устройствами	42
4.3 Структура управляющих блоков базы данных ОС	43
4.4 Видеотерминал и НМД	44

5 РАЗРАБОТКА ПРОГРАММ ЭМУЛЯЦИИ ОС	45
5.1 Описание программных средств	45
5.2 Разработка ФС	45
5.3 Разработка командного интерпретатора	52
5.4 Эмуляция планирования	52
6 ТЕСТИРОВАНИЕ ПРОГРАММЫ. АНАЛИЗ РЕЗУЛЬТАТОВ	53
ВЫВОДЫ	55
ПЕРЕЧЕНЬ ССЫЛОК	56
ПРИЛОЖЕНИЕ А	57
ТЕХНИЧЕСКОЕ ЗАДАНИЕ	57
ПРИЛОЖЕНИЕ Б	60
ЛИСТИНГ ПРОГРАММ	60

ВВЕДЕНИЕ

Операционная система – это программа, которая после первоначальной загрузки в компьютер управляет всеми другими прикладными программами на компьютере. Прикладные программы взаимодействуют с ней посредством системных вызовов, которые удобно представлены в API операционной системы. Кроме того, пользователи могут напрямую взаимодействовать с ОС с помощью интерфейса командной строки или графического пользовательского интерфейса.

Целью данной курсовой работы является создание эмулятора планировщика и файловой системы. Для этого нужно:

- 1) изучить проектное задание;
- 2) проанализировать требования к системе и выбрать путь их реализации;
- 3) изучить способы организации ФС, а также их внутреннюю структуру;
- 4) изучить процессы в операционной системе, их приоритеты, способы их взаимодействия и планирования;
- 5) разработать основные алгоритмы функционирования гипотетической операционной системы.

1 СТРУКТУРА ПРОЕКТИРУЕМОЙ ФАЙЛОВОЙ СИСТЕМЫ

Файловая система (ФС) - это программное обеспечение, которое отвечает за хранение и извлечение данных на диске. Это компонент логического диска, который управляет внутренними операциями диска и является абстрактным для человека.

1.1 Общая организация файловой системы

Разрабатываемая система включает в себя некоторые принципы файловой системы «FFS». Основные характеристики разрабатываемой ФС:

- Способ организации файлов многоуровневый иерархический.
- Битовая карта свободных/занятых кластеров.
- Файлы с последовательным доступом.
- Файлы с прямым доступом.
- Суперблок.
- Битовые карты свободных блоков и индексных дескрипторов.

Структура логического диска изображена на рис. 1.1.

Суперблок	Список групп	Список пользователей	Битовая карта свободных/занятых инодов	Битовая карта свободных/занятых кластеров	Иноды	Кластеры
59 байт	22*50 байт	35*255 байт	кол-во инодов/8 байт	кол-во кластеров/8 байт	74 * кол-во инодов байта	размер ФС-пред. структуры байт
[1, 59]	[60, 1159]	[1160, 10084]	[10085, ~]	[~, ~]	[~, ~]	[~, размер ФС]

Рисунок 1.1 – структура логического диска

В файловой системе может существовать только один суперблок, который содержит информацию, необходимую для монтирования и управления работой

файловой системы. Суперблок располагается в начале раздела. Структура суперблока представлена в таблице 1.1.

Таблица 1.1 – Структура суперблока

Поле	Тип данных	Размер	Описание
fs_name	char[]	16 байт	Имя файловой системы
fs_type	char[]	16 байт	Тип файловой системы
cluster_size	ushort	2 байт	Размер кластера в байтах
inode_count	uint	4 байт	Количество инодов
inode_free_count	uint	4 байт	Количество свободных инодов
cluster_count	uint	4 байт	Количество кластеров
cluster_free_count	uint	4 байт	Количество свободных кластеров
users_count	byte	1 байт	Актуальное количество пользователей
fs_size	ulong	8 байт	Размер ФС

В данной реализации суперблок будет занимать ровно 59 байт памяти.

Такой выбор типов данных позволяет создать файловую систему с размером до 256 Тб (макс. размер кластера 65535 байт, умноженный на макс. количество кластеров 4294967295, даёт размер 255 Тб).

Количество кластеров получается при делении всего размера ФС на размер кластера.

Количество инодов получается при делении количества кластеров на 2.

Индексный дескриптор (inode, инод) – структура данных, содержащая всю информацию о конкретном файле. Каждый файл связан с одним своим инодом. Структура inode отображена в таблице 1.2.

Таблица 1.2 – Структура индексного дескриптора

Поле	Тип данных	Размер	Описание
uid	byte	1 байт	Идентификатор владельца
gid	byte	1 байт	Идентификатор группы-владельца
permissions	ushort	2 байт	Флаги файла
file_size	uint	4 байт	Размер файла в байтах
create_date	struct Date	7 байт	Дата и время создания
mod_date	struct Date	7 байт	Дата и время последней модификации
addr	uint[]	52 байт	Массив адресов кластеров

Запись индексного дескриптора занимает 74 байта памяти.

Если смотреть слева-направо, то первый бит поля permissions – это режим файла, который если установлен в 0, то инод содержит информацию об обычном файле, если в 1, – каталоге. Биты 2-10 отведены под права доступа. Бит 11 – флаг скрытости.

Права доступа представлены тремя группами по 3 бита. Первая группа – права доступа для владельца, вторая – для группы владельца, третья – для остальных пользователей. Первый бит каждой группы прав доступа – право на чтение, второй – на запись, третий – на исполнение. Значение бита 1 – доступ разрешён, значение бита 0 – доступ запрещён.

Поле addr хранит массив адресов кластеров. В первых 10 элементах массива хранятся адреса кластера с фактическими данными файла. В элементах 11-13 хранятся кластеры, в которых хранятся ссылки на другие кластеры с фактическими данными. Количество этих ссылок равняется размеру кластера, делённому на 4 (т.к. для хранения ссылки на кластер используется тип данных uint, который занимает 4 байта памяти). Таким образом размер файла зависит от размера кластера.

Для хранения дат создания и модификации файла реализована структура Date, которая описана в таблице 1.3.

Таблица 1.3 – Структура типа данных Date

Поле	Тип данных	Размер	Описание
day	byte	1 байт	День
month	byte	1 байт	Месяц
year	ushort	2 байт	Год
hours	byte	1 байт	Часы
minute	byte	1 байт	Минуты
second	byte	1 байт	Секунды

Из структуры иногда можно увидеть, что имена файлов и содержимое файлов в них не хранятся. Содержимое файлов хранится в соответствующих кластерах. Имена файлов хранятся в специальном типе файлов – каталогах. Структура каталога показана в таблице 1.4.

Таблица 1.4 – Структура записи в каталоге

Поле	Тип данных	Размер	Описание
inode_id	uint	4 байт	Номер инода
name	char[]	28 байт	Имя

Из таблицы 1.4 видно, что имена каталогов фиксированного размера.

Максимальное количество пользователей в разрабатываемой файловой системе равняется 255. Структура пользователя показана в таблице 1.5

Таблица 1.5 – Структура пользователя

Поле	Тип данных	Размер	Описание
life	byte	1 байт	Флаг существования пользователя
uid	byte	1 байт	ИД номер пользователя
gid	byte	1 байт	ИД номер группы
login	char[]	16 байт	Логин
password	char[]	16 байт	Пароль

Запись об одном пользователе будет занимать 35 байт памяти.

Флаг life отвечает за существование пользователя в системе. Если он установлен в 1, то пользователь существует, если в 0 – удалён.

Поле gid указывает на ИД номер группы, в которой находится пользователь (см. табл. 1.6).

Таблица 1.6 – Структура группы

Поле	Тип данных	Размер	Описание
life	byte	1 байт	Флаг существования группы
gid	byte	1 байт	ИД номер группы
description_cluster	uint	4 байт	ИД номер кластера с описанием группы
name	char[]	16 байт	Имя группы

Запись об одной группе будет занимать 22 байта памяти.

Битовая карта является массивом бит, каждый из которых отвечает за состояние отдельного инода или кластера. Если бит установлен в 0, то инод/кластер свободен, если в 1, – занят.

1.2 Виртуальные страницы

Виртуальная память – способ управления памятью компьютера, при котором возможно выполнять программы, которым требуется больше оперативной памяти, чем представлено компьютером. Часть данных из основной памяти записывается во вторичное хранилище, а при необходимости считана обратно.

Существуют различные организации виртуальной памяти, но в большинстве операционных систем используется страничная организация, её и рассмотрим подробнее.

При страничной организации виртуальной памяти виртуальные адреса отображаются на физические постранично. Минимальный размер страницы равен 4096 байт для 32-битной архитектуры x86.

Пространство виртуальных адресов разделено на единицы, называемые страницами. Соответствующие единицы в физической памяти называются страничными блоками (page frame). Страницы и их блоки имеют всегда одинаковый размер.

ОС создаёт для каждого процесса таблицу страниц, которая содержит записи обо всех виртуальных страницах процесса (см. рис. 1).

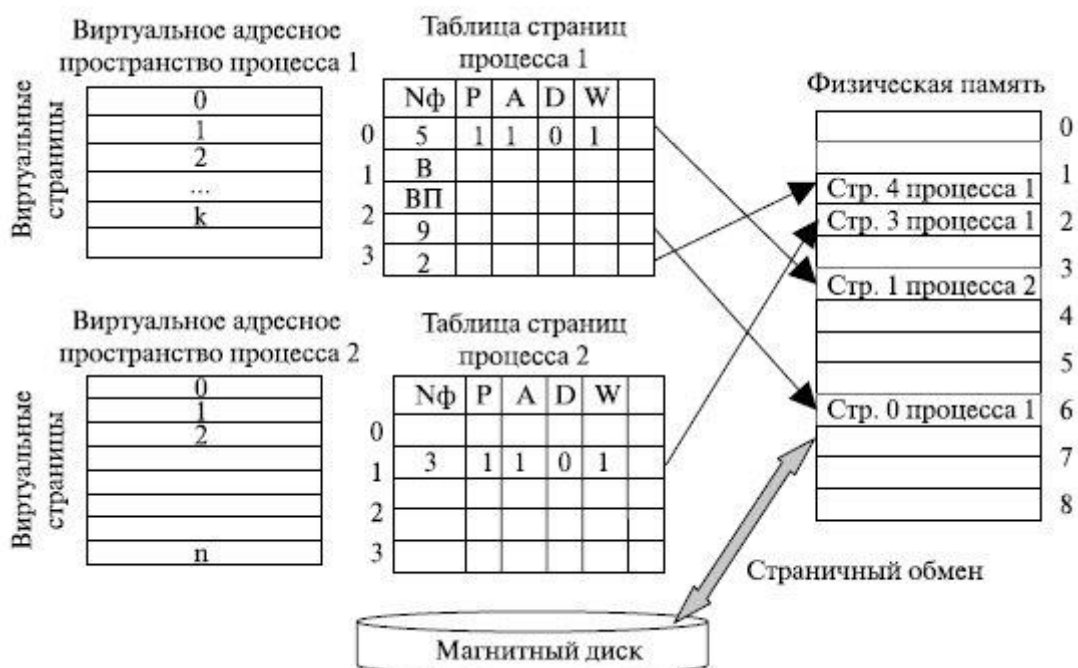


Рисунок 1.2 – Таблицы страниц виртуальной памяти

Запись таблицы (дескриптор страницы) включает следующую информацию:

- номер физической страницы, в которую загружена данная виртуальная страница;
- признак присутствия P, устанавливаемый в единицу, если данная страница находится в оперативной памяти;
- признак модификации страницы D, который устанавливается в единицу всякий раз, когда производится запись по адресу, относящемуся к данной странице;
- признак обращения A к странице, называемый также битом доступа, который устанавливается в единицу при каждом обращении по адресу, относящемуся к данной странице;
- другие управляющие биты, служащие, например, для целей защиты или совместного использования памяти на уровне страниц.

1.3 Команды для работы с ФС

Файловая система реализована с графическим пользовательским интерфейсом. Графический интерфейс отдалён от логики программы, он только может вызывать системные команды и отображать результаты их выполнения (см. табл. 1.7).

Таблица 1.7 – Команды файловой системы

Команда	Параметры	Результат	Описание
Create_File_System	char[] fs_name, char[] fs_type, ushort cluster_size, ulong fs_size, char[] login, char[] password	int	Создание ФС
Load_File_System	char[] login, char[] password	int	Загрузка ФС
Get_Flags	ushort permissions	bool[]	Получить флаги из байтовой записи
Read_Catalog	uint catalog_id	Data_Catalog[]	Получить данные о файлах каталога
Copy	uint inode_id, char[] name	Data_Copy	Скопировать файл
Insert	Data_Copy data_file, uint parent_catalog_id	void	Вставить файл

Продолжение таблицы 1.7

Create_File	char[] name, bool is_catalog, uint parent_catalog_id	uint	Создать новый файл
Get_Data_File	uint inode_id	char[]	Получить содержимое файла
Set_Data_File	uint inode_id, char[] chars	int	Установить содержимое файла
Get_Properties_File	uint inode_id, bool is_catalog	string	Получить свойства файла
Get_Permissions	uint inode_id	ushort	Получить флаги файла в байтовом представлении
Check_Rights_Access	uint inode_id	bool[]	Получить триаду прав доступа
Delete_File	uint inode_id	int	Удалить файл
Rename	char[] old_name, char[] new_name, uint parent_catalog_id	int	Переименование файла
Write_Catalog	byte[] bytes, uint catalog_id	int	Запись в каталог
Is_Owner	uint inode_id	bool	Является ли пользователь владельцем файла

Продолжение таблицы 1.7

Is_Admin		bool	Является ли пользователь администратором
Set_Flags	ushort flags, uint inode_id	void	Установить флаги
Get_All_Groups		Dictionary <string, byte>	Получить словарь групп
Add_User	byte gid, char[] login, char[] password	int	Добавить нового пользователя
Add_Group	char[] name, char[] desc	int	Добавить новую группу
Delete_User	char[] login, char[] password	int	Удалить пользователя
Delete_Group	char[] name	int	Удалить группу
Change_User	char[] login, char[] password	int	Сменить пользователя
Change_Group	char[] name	int	Сменить группу

Некоторые экранные формы графического интерфейса представлены на рисунках 1.2 – 1.5.

START WORK BEST IN THE BEST WORLD BEST FS

Тип файловой системы:

Размер кластера:

Размер файловой системы в Кб:

Логин пользователя:

Пароль пользователя:

Рисунок 1.3 – Создание файловой системы

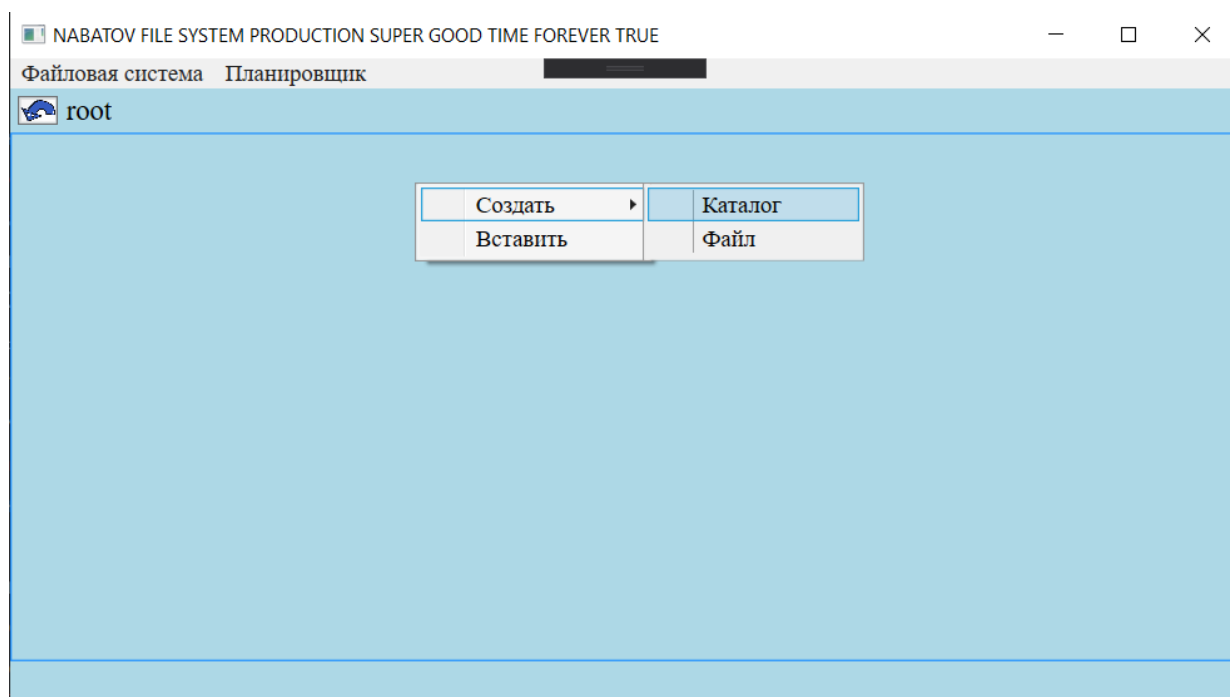


Рисунок 1.4 – Создание каталога

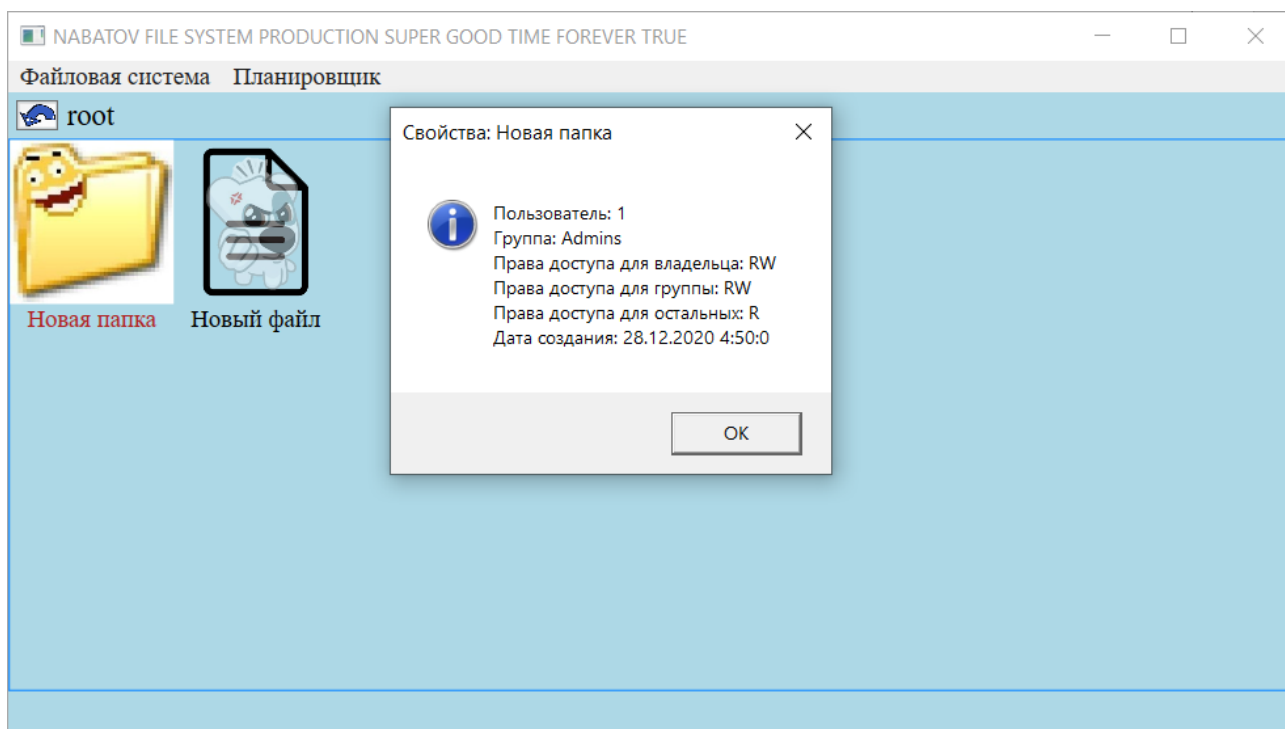


Рисунок 1.5 – Просмотр свойств файла

Рисунок 1.6 – Добавление группы

1.4 Системные вызовы для работы с ФС

Системный вызов – обращение прикладной программы к ядру ОС для выполнения какой-либо операции.

Системные вызовы, реализованные в данном проекте, показаны в таблице 1.8.

Таблица 1.8 – Системные вызовы

Команда	Параметры	Результат	Описание
Cluster_Control. Read	uint id	byte[]	Чтение из кластера
Cluster_Control. Write	byte[] bytes, uint id, int position = 0	int	Запись в кластер
Inode.Read	uint pos	int	Чтение из инода
Inode.Create	byte uid, byte gid, Bitmap bitmap_inode, Superblock superblock, Bitmap bitmap_cluster, bool is_catalog = false, ushort permissions	int	Создание инода
Inode.Write	Superblock superblock, Bitmap bitmap_cluster	int	Запись инода
Inode.Get_Clusters		uint[]	Получить массив занятых кластеров инода
User.Read	char[] input_login, char[] input_password	int	Считать данные пользователя
User.Add	byte gid, char[] login, char[] password, Superblock superblock	int	Добавить пользователя
User.Delete	byte uid, Superblock superblock, Bitmap bitmap	int	Удаление пользователя

1.5 Способы организации файлов

Файлы делятся на два типа: файлы последовательного и прямого доступа.

Файлы последовательного доступа реализованы таким образом, что обращаться к элементам данного файла допускается только в той последовательности, в которой они записывались.

Файлы прямого доступа в отличие от последовательного предоставляют возможность доступа к элементам в любой последовательности через адрес элемента.

В реализованной ФС запись пользовательского файла осуществляется в последовательном режиме, а запись служебных файлов, например, каталогов, осуществляется в прямом режиме. Файлы хранятся в массиве кластеров. Первые 10 элементов массива хранят в себе данные файла, а последние 3 хранят в себе ссылки на дополнительные кластеры, которые, в свою очередь, уже хранят фактическую информацию файла (см. рис. 1.6).

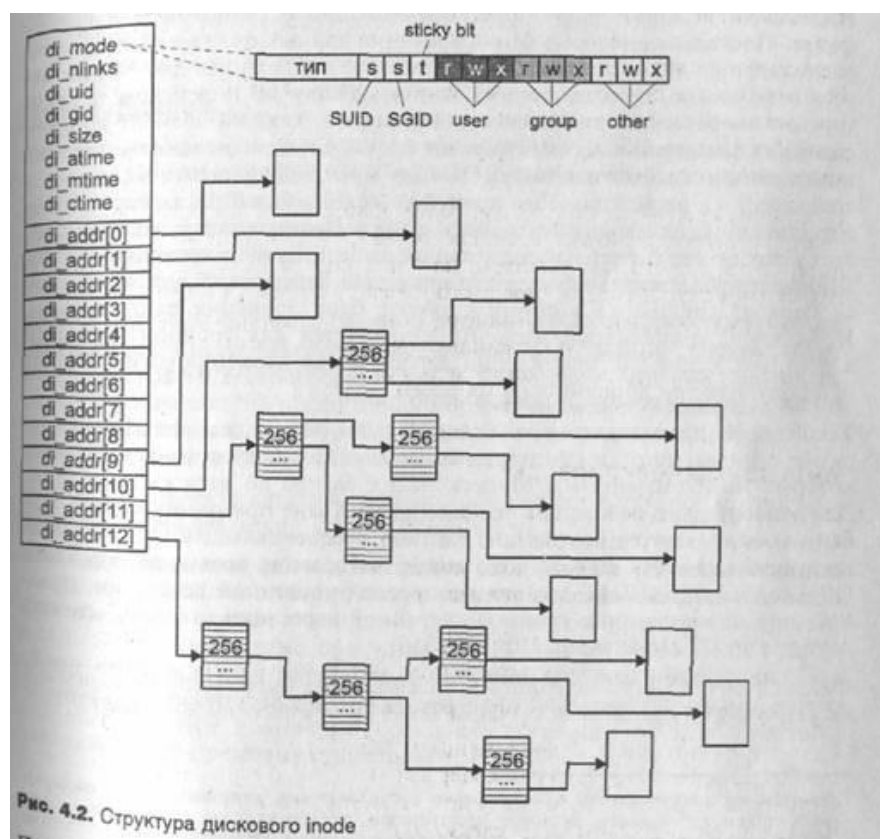


Рис. 4.2. Структура дискового inode

Рисунок 1.7 – Структура инода с архитектурой хранения данных файла

В разработанной ФС используется одинарная косвенная адресация.

1.6 Алгоритмы работы некоторых системных вызовов ФС.

Алгоритм записи файла в кластеры, предоставленные инодом, показан ниже на псевдокоде:

- список кластеров = получить все кластеры инода
- FOR каждый элемент списка кластеров
- IF осталось записать байт > 0 THEN
- записать в кластер байты
- осталось записать байт = осталось записать байт – размер кластера
- ELSE IF старый размер файла > текущего размера файла THEN
- записать в кластеры (старый размер файла – текущий размер файла) нулей
- ELSE
- вернуть код успешного завершения
- ENDIF
- ENDIF
- ENDFOR
- перезаписать инод
- IF осталось записать байт > 0 THEN
- вернуть код ошибочного завершения (закончилось место)
- ELSE
- вернуть успешного завершения
- ENDIF

Алгоритм получения кластеров, используемых инодом, представлен ниже на псевдокоде:

- размер массива кластеров = размер файла / размер кластера
(округлять в большую сторону)
- FOR каждого кластера из массива до 10 по счёту
- IF кластер == 0 THEN
- вернуть массив кластеров
- ENDIF

- добавить кластер в массив кластеров
- ENDFOR
- FOR каждого кластера из массива с 10 по 13 элемент по счёту
- IF кластер == 0 THEN
- вернуть массив кластеров
- ENDIF
- косвенный массив = получить содержимое косвенного кластера
- FOR каждого кластера из косвенный массив
- IF кластер == 0 THEN
- вернуть массив кластеров
- ENDIF
- добавить кластер в массив кластеров
- ENDFOR
- вернуть массив кластеров

2 ПРОЦЕССЫ В ОПЕРАЦИОННОЙ СИСТЕМЕ

Процесс – вычисление, которое может быть выполнено параллельно с другими вычислениями.

Процесс состоит из инструкций, которые выполняет процессор, данных и информации о выполняемой задаче, такой как размещение в памяти, открытие файлов, а также статуса процесса.

Процесс считывает и записывает информацию в раздел данных и в стек, но ему недоступны данные и стеки других процессов. Они могут взаимодействовать между собой с помощью средств межпроцессного взаимодействия.

Поток — определенный способ выполнения процесса. Когда один поток изменяет ресурс процесса, это изменение сразу же становится видно другим потокам этого процесса.

Поток использует то же самое пространства стека, что и процесс, а множество потоков совместно используют данные своих состояний. Как правило, каждый поток может работать (читать и писать) с одной и той же областью памяти, в отличие от процессов, которые не могут просто так получить доступ к памяти другого процесса. У каждого потока есть собственные регистры и собственный стек, но другие потоки могут их использовать.

За постановку процессов в очередь на выполнение отвечает планировщик.

2.1 Команды для работы с процессами

Для выполнения команды добавления потока необходимо выбрать приоритет, время выполнения и состояние потока. После этого поток добавляется в таблицу (см. рис. 2.1).

Scheduler_Window

Приоритет

4

Время выполнения

11

Состояние

R

Добавить процесс

Процесс №	Приоритет	Время вхождения	Полное время выполнения	Оставшееся время выполнения	Состояние	Квант
1	4	0	11	11	R	0

Процессы\время

1

Сделать шаг

Квант

Установить квант

Рисунок 2.1 – Добавление потока

В реализации планировщика процессов предусмотрена возможность изменения кванта времени, который будет предоставлен процессам (см. рис. 2.2).

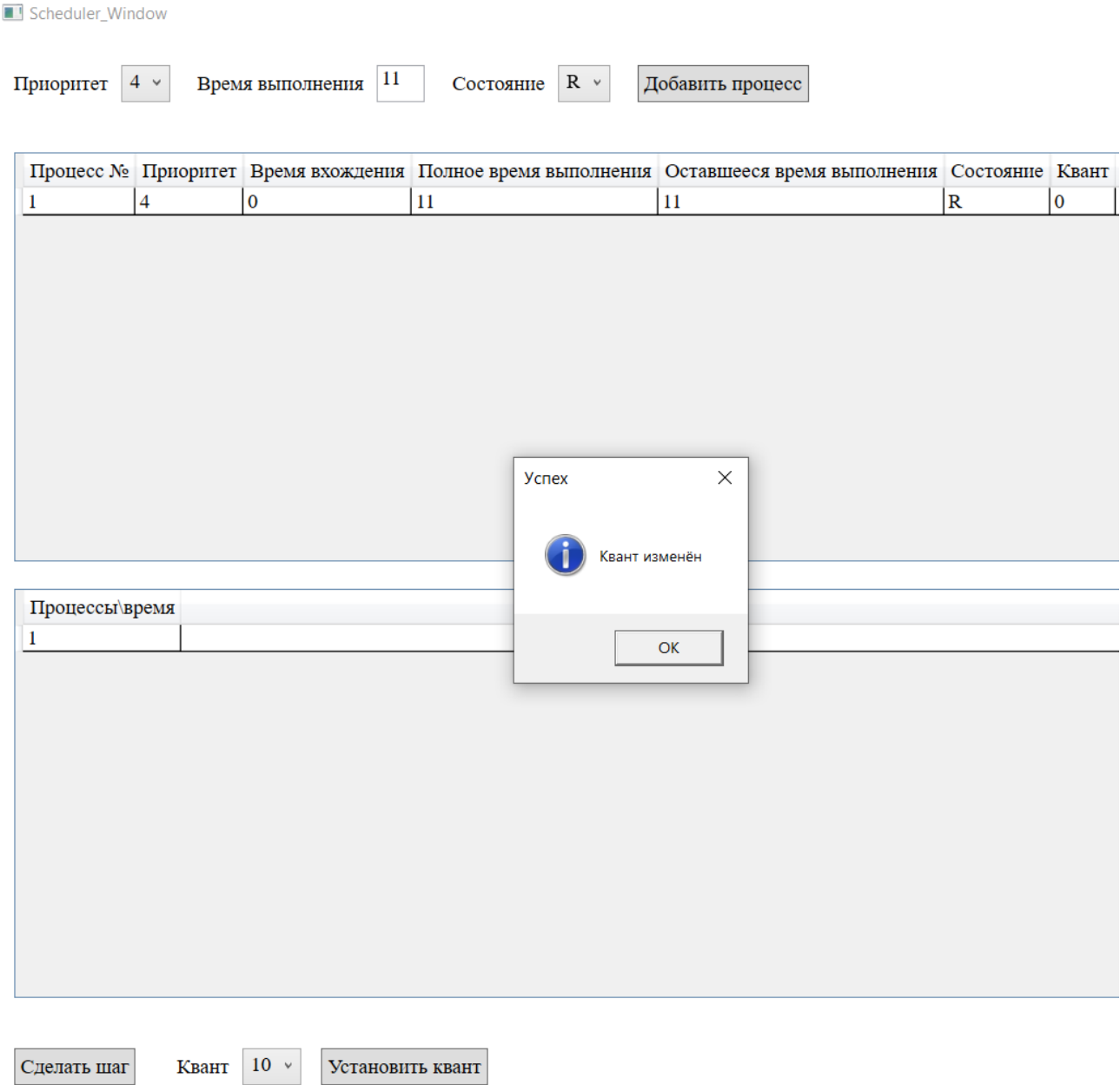


Рисунок 2.2 – Изменение кванта

При выполнении шага выполняется тот процесс, который был выбран планировщиком в результате работы алгоритма (см. рис. 2.3)

Scheduler_Window

Приоритет

10

Время выполнения

2

Состояние

R

Добавить процесс

Процесс №	Приоритет	Время вхождения	Полное время выполнения	Оставшееся время выполнения	Состояние	Квант
1	4	0	11	9	P	8
2	10	0	2	2	R	10

Процессы\время	1	2	3
1	R	P	P
2	R	R	R

Сделать шаг

Квант

10

Установить квант

Рисунок 2.3 – Выполнение шага

Для работы с процессами была реализована структура Process, которая показана в таблице 2.1.

Таблица 2.1 – Структура Process

Поле	Тип данных	Размер	Описание
Num	int	4 байт	Номер процесса
Priority	byte	1 байт	Приоритет

Продолжение таблицы 2.1

Entry_time	int	4 байт	Время вхождения
Work_time	int	4 байт	Время всей работы
Left_time	int	4 байт	Оставшееся время работы
Current_state	byte	1 байт	Текущее состояние
Quant	int	4 байт	Текущее количество квантов

2.2 Системные вызовы управления процессами

Системные вызовы управления процессами описаны в таблице 2.2.

Таблица 2.2 – Системные вызовы управления процессами

Команда	Параметры	Результат	Описание
Add	byte prior, int time, State st		Добавление процесса
Scheduling	Process last	Process	Выполнение шага
Set_Quant	int quant		Установление кванта

2.3 Диаграмма состояний процесса

Жизненный цикл процесса разбит на несколько состояний. Переход процесса из одного состояния в другое происходит в зависимости от наступления тех или иных событий в системе (см. рис 2.4).

1. Процесс выполняется в режиме задачи. При этом процессором выполняются прикладные инструкции данного процесса.

2. Процесс выполняется в режиме ядра. При этом процессором выполняются системные инструкции ядра операционной системы от имени процесса.

3. Процесс не выполняется, но готов к запуску, как только планировщик выберет его. Процесс находится в очереди на выполнение и обладает всеми необходимыми ему ресурсами, кроме вычислительных.

4. Процесс находится в состоянии сна, ожидая недоступного в данный момент ресурса, например, завершения операции ввода/вывода.

5. Процесс возвращается из режима ядра в режим задачи, но ядро прерывает его и производит переключение контекста для запуска более высокоприоритетного процесса.

6. Процесс только что создан вызовом `fork()` и находится в переходном состоянии: он существует, но не готов к запуску и не находится в состоянии сна.

7. Процесс выполнил системный вызов `exit()` и перешел в состояние зомби. Как такового процесса не существует, но остаются записи, содержащие код возврата и временную статистику его выполнения, доступную для родительского процесса. Это состояние является конечным в жизненном цикле процесса.

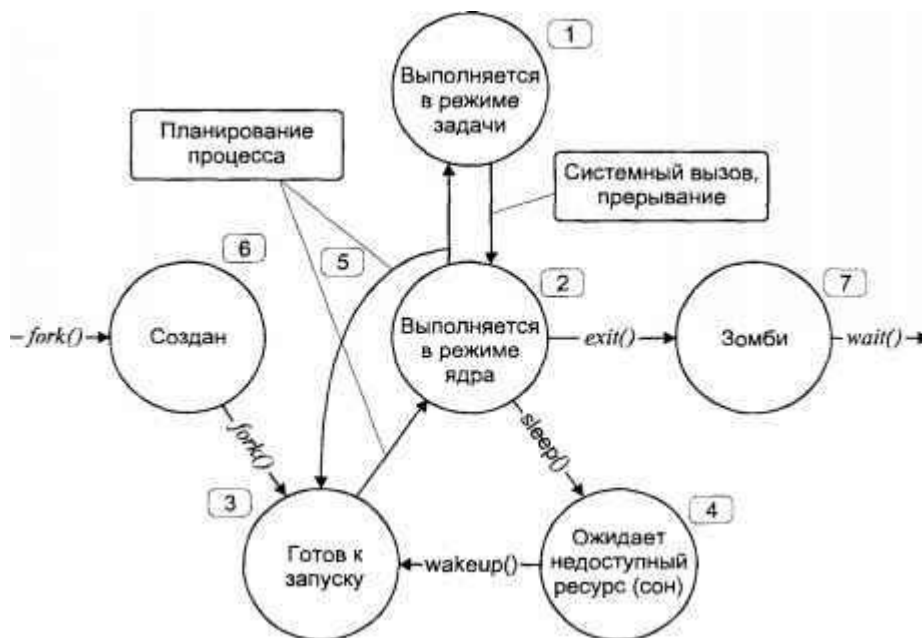


Рисунок 2.4 – Состояния процесса

2.4 Приоритеты процессов

Приоритет – параметр, определяющий преимущества процесса в обладании ресурсами по сравнению с другими процессами.

Приоритет может выражаться целым или дробным, положительным или отрицательным значением. В некоторых ОС принято, что приоритет потока тем выше, чем больше (в арифметическом смысле) число, обозначающее приоритет. В других системах, например, в этом проекте, наоборот, чем меньше число, тем выше приоритет.

Приоритеты бывают статическими и динамическими.

Статический приоритет – приоритет, который устанавливается и изменяется пользователем. Динамический приоритет – приоритет, который устанавливается и изменяется операционной системой для обеспечения эффективной работы компьютера.

Основное их отличие в том, что статический приоритет система только учитывает, а динамический она формирует и модифицирует.

Абсолютные и относительные приоритеты различаются степенью срочности предоставления привилегий их обладателям.

Относительные приоритеты учитываются при выделении свободных ресурсов. Появление процесса с более высоким абсолютным приоритетом при отсутствии свободных ресурсов приводит к захвату ресурсов у менее приоритетных процессов.

2.5 Межпроцессное взаимодействие.

Межпроцессное взаимодействие – обмен данными между потоками одного или разных процессов. Реализуется посредством механизмов, предоставляемых ядром ОС или процессом, использующим механизмы ОС и реализующим новые возможности IPC. Может осуществляться как на одном компьютере, так и между несколькими компьютерами сети.

В рамках курсового проекта подробнее разберем такие средства межпроцессного взаимодействия, как каналы, именованные каналы, разделяемая память.

Канал - однонаправленное средство взаимодействия. Данные, записанные в канал со "стороны записи" читаются со "стороны чтения." Каналы - последовательные устройства; данные всегда читаются в том же порядке, в котором были записаны. Как правило, канал используется для взаимодействия двух потоков в одном процессе или между родительскими и дочерними процессами.

Пример взаимодействия между процессами с помощью каналов представлен ниже:

```
#include <stdlib.h>
#include <stdio.h>
#include <unistd.h>

/* Записать COUNT копий сообщений MESSAGE потоку STREAM, делая секундную
паузу между каждым. */
void writer (const char* message, int count, FILE* stream)
{
    for (; count > 0; --count) {
        /* Написать сообщение потоку, и отослать его немедленно. */
        fprintf (stream, "%s\n", message);
        fflush (stream);
        /* Задержаться */
        sleep (1);
    }
}

/* Читать случайные строки из потока. */
void reader (FILE* stream)
{
    char buffer[1024];

    /* Чтение, пока мы не дойдем до конца потока. fgets считывает данные пока не
новая строка или конец файла. */
    while (!feof (stream) && !ferror (stream))
```

```

        && fgets (buffer, sizeof (buffer), stream) != NULL)
        fputs (buffer, stdout);
    }
    int main ()
    {
        int fds[2];
        pid_t pid;
        /* Создание канала. Дескрипторы файлов канала помещены в fds. */
        pipe (fds);
        /* Создать дочерний процесс. */
        pid = fork ();
        if (pid == (pid_t) 0) {
            FILE* stream;
            /* Это - дочерний процесс. Закрываем копию дескриптора файла записи */
            close (fds[1]);
            /* Преобразование дескриптора файла чтения к объекту FILE, и чтение из
него. */
            stream = fdopen (fds[0], "r");
            reader (stream);
            close (fds[0]);
        }
        else {
            /* Это - родительский процесс. */
            FILE* stream;
            /* Закрывают нашу копию конца чтения дескриптора файла. */
            close (fds[0]);
            /* Преобразование дескриптора файла записи к объекту FILE, и запись в
него. */
            stream = fdopen (fds[1], "w");
            writer ("Hello, world.", 5, stream);
            close (fds[1]);
        }
        return 0;
    }

```

В программе, с помощью команды `fork`, порождается дочерний процесс. Дочерний процесс унаследовал дескрипторы файлов канала. Родитель пишет строку в канал, а потомок считывает ее. Типовая программа преобразовывает эти дескрипторы файлов в потоки `FILE*`, используя команду `fdopen`. Поскольку используются потоки, а не дескрипторы файлов, возможно использование высокоуровневых функций ввода - вывода стандартной библиотеки C, таких как `printf` и `fgets`.

Именованный канал (FIFO) - канал, который имеет имя в файловой системе. Любой процесс может открыть или закрыть именованный канал; процессы на любом конце канала не должны быть связаны друг с другом.

Пример создания именованного канала `/Nabatov/fifo`:

```
mkfifo /Nabatov/fifo
```

Разделяемая память является самым быстрым средством обмена данными между процессами.

В других средствах межпроцессового взаимодействия (IPC) обмен информацией между процессами проходит через ядро, что приводит к переключению контекста между процессом и ядром, т.е. к потерям производительности.

Техника разделяемой памяти позволяет осуществлять обмен информацией через общий для процессов сегмент памяти без использования системных вызовов ядра. Сегмент разделяемой памяти подключается в свободную часть виртуального адресного пространства процесса. Таким образом, два разных процесса могут иметь разные адреса одной и той же ячейки подключенной разделяемой памяти.

После создания разделяемого сегмента памяти любой из пользовательских процессов может подсоединить его к своему собственному виртуальному пространству и работать с ним, как с обычным сегментом памяти. Недостатком такого обмена информацией является отсутствие каких бы то ни было средств

синхронизации, однако для преодоления этого недостатка можно использовать технику семафоров.

Ниже показана реализация технологии «клиент-сервер»:

Содержимое файла server.c:

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/shm.h>
#include <sys/wait.h>
#include <sys/select.h>

#include "magicinput.h"

int main (int argc, const char *argv[])
{
    key_t key = ftok("last_process", 1);

    int *shm, shmid, number;
    struct shmid_ds shm_buf;

    if (argc == 2) // Если один параметр был введен
    {
        sscanf(argv[1], "%d", &number);
    }
    else
    {
        printf("Используйте один параметр number!\n");
        exit(1);
    }

    /* Создание области разделяемой памяти */
    if ((shmid = shmget(key, sizeof(int)*2, IPC_CREAT | 0666)) < 0)
    {
        perror("Ошибка в вызове shmget");
        exit(1);
    }

    /* Получение доступа к разделяемой памяти */
    if ((shm = shmat(shmid, NULL, 0)) == (int *)-1)
    {
        perror("Ошибка в вызове shmat");
        exit(1);
    }

    shm[0] = 0;
```



```

shm[1] = -1;
shm[2] = 1;
shmctl(shmid, IPC_STAT, &shm_buf);
printf("ESC - закончить работу\n");
int c = 0;
while (c != 27)
{
    if (shm[1] == -1)
    {
        shmctl(shmid, IPC_STAT, &shm_buf);
        /* Запись в разделяемую память */
        printf("\nШаг: %d\nСтарое число: %d; последний процесс: %d\n\n",
shm[2], shm[0], shm_buf.shm_lpid);
        shm[0] += number;
        shm[2]++;
        printf("Новое число: %d\n", shm[0]);
        printf("-----\n");
        sleep(5);
        shm[1] = 0;
    }
    if (kbhit())
    {
        c = getch();
    }
    sleep(1);
}

if (shmdt(shm) < 0) // Отключение от разделяемой памяти
{
    perror("Ошибка в вызове shmdt");
    exit(1); // Выход из процесса
}

/* Удаление созданных объектов разделяемой памяти */
if (shmctl(shmid, IPC_RMID, 0) < 0)
{
    perror("Ошибка в вызове shmctl");
    exit(1);
}
}

```

Содержимое файла client.c:

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/shm.h>
#include <sys/wait.h>

#include "magicinput.h"

int main (int argc, const char *argv[])
{
    key_t key = ftok("last_process", 1);

    int *shm, shmid, number;
    struct shmids shm_buf;

    printf("Введите число: ");
    scanf("%d", &number);

    /* Подключение к области разделяемой памяти */
    if ((shmid = shmget(key, sizeof(int)*2, 0666)) < 0)
    {
        perror("Ошибка в вызове shmget");
        exit(1);
    }

    /* Получение доступа к разделяемой памяти */
    if ((shm = shmat(shmid, NULL, 0)) == (int *)-1)
    {
        perror("Ошибка в вызове shmat");
        exit(1);
    }

    shmctl(shmid, IPC_STAT, &shm_buf);
    printf("ESC - закончить работу\n");
    int c = 0;
    while (c != 27)
    {
        if (shm[1] == 0)
        {
            shmctl(shmid, IPC_STAT, &shm_buf);
            /* Чтение из разделяемой памяти */
            printf("\nШаг: %d\nСтарое число: %d; последний процесс: %d\n", shm[2],
shm[0], shm_buf.shm_lpid);

            /* Запись в разделяемую память */
            shm[0] += number;
            shm[2]++;
        }
    }
}
```

```

        printf("\nНовое число: %d\n", shm[0]);
        printf("-----\n");
        sleep(5);
        shm[1] = -1;

    }
    if (kbhit())
    {
        c = getch();
    }
    sleep(1);
}

if (shmdt(shm) < 0) // Отключение от разделяемой памяти
{
    perror("Ошибка в вызове shmdt");
    exit(1); // Выход из процесса
}
}

```

Данная программа реализовывает обмен числом между сервером и клиентом, каждый из которых увеличивает его на свою константу и записывает в разделяемую память.

2.6 Выбор дисциплины обслуживания планировщика процессов. Алгоритм работы планировщика процессов в соответствии с выбранной дисциплиной обслуживания

Для обслуживания процессов был разработан новый алгоритм планирования.

Процессы с приоритетами 0-3 образуют свою очередь относительных приоритетов. Внутри этой очереди процессы выполняются по правилам алгоритма относительных приоритетов.

Процессы с приоритетами 4-39 образуют очередь с динамическими приоритетами и квантованием времени. Процессы выполняются согласно алгоритму Round-Robin: каждому процессу выдаётся свой квант времени, когда он будет исчерпан, его вытеснит процесс с более высоким приоритетом, имеющий квант времени (см. рис 2.5).

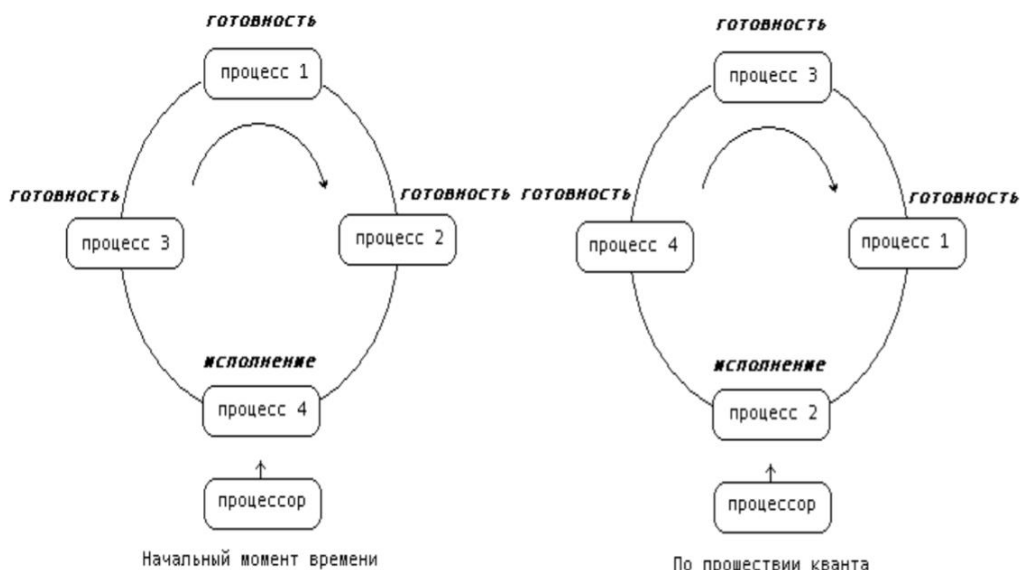


Рисунок 2.5 – Алгоритм Round-Robin

Когда процесс завершает свою работу, начинает выполняться алгоритм Little-Forward: процесс, имеющий наименьшее остаточное время выполнения, получает приоритет 4 и становится впереди других процессов.

Алгоритм работы планировщика показан ниже на псевдокоде:

- FOR каждый процесс
- IF состояние процесса == выполняется AND (приоритет процесса ≤ 3
OR квант процесса > 0) THEN
- квант процесса = квант процесса – 1
- оставшееся время выполнения процесса = оставшееся время
выполнения процесса - 1
- IF оставшееся время выполнения процесса == 0 THEN
- состояние процесса = зомби
- время маленьких = истина
- ELSE IF квант процесса == 0 THEN
- состояние процесса = готов
- ENDIF
- вернуть код успешного завершения
- ENDIF
- ENDFOR
- врем. процесс = первый процесс готовый к выполнению

- IF квант не установлен ни у одного процесса THEN
- FOR каждый процесс с приоритетом больше 3 AND состоянием готов
- квант процесса = квант
-
- FOR каждый процесс
- IF состояние процесса == готов AND (приоритет процесса ≤ 3 OR квант
процесса > 0) THEN
- IF приоритет врем. процесса $>$ приоритет процесса THEN
- врем. процесс = процесс
- ENDIF
- ENDIF
- ENDFOR
- состояние врем. процесса = выполняется
- квант врем. процесса = квант врем. процесса $- 1$
- оставшееся время выполнения врем. процесса = оставшееся время
выполнения врем. процесса $- 1$
- IF время маленьких == истина THEN
- приоритет процесса с наименьшим временем выполнения = 4
- время маленьких = ложь
- ENDIF
- вернуть код успешного завершения

Результаты трассировки показаны на рисунках 2.6-2.7.

Scheduler_Window

Приоритет

24

▼

Время выполнения

1

Состояние

R

▼

Добавить процесс

Процесс №	Приоритет	Время вхождения	Полное время выполнения	Оставшееся время выполнения	Состояние	Квант	
1	3	0	3	0	Z	0	
2	6	0	3	3	R	2	
3	12	0	10	10	R	2	
4	24	0	1	1	R	2	

Маленькие вперёд!

Приоритет процесса №4 был изменён с 24 на 4

OK

Процессы\время	1	2	3	4	
1	P	P	P	Z	
2	R	R	R	R	
3	R	R	R	R	
4	R	R	R	R	

Сделать шаг

Квант

2

▼

Установить квант

Рисунок 2.6 – Планирование процессов

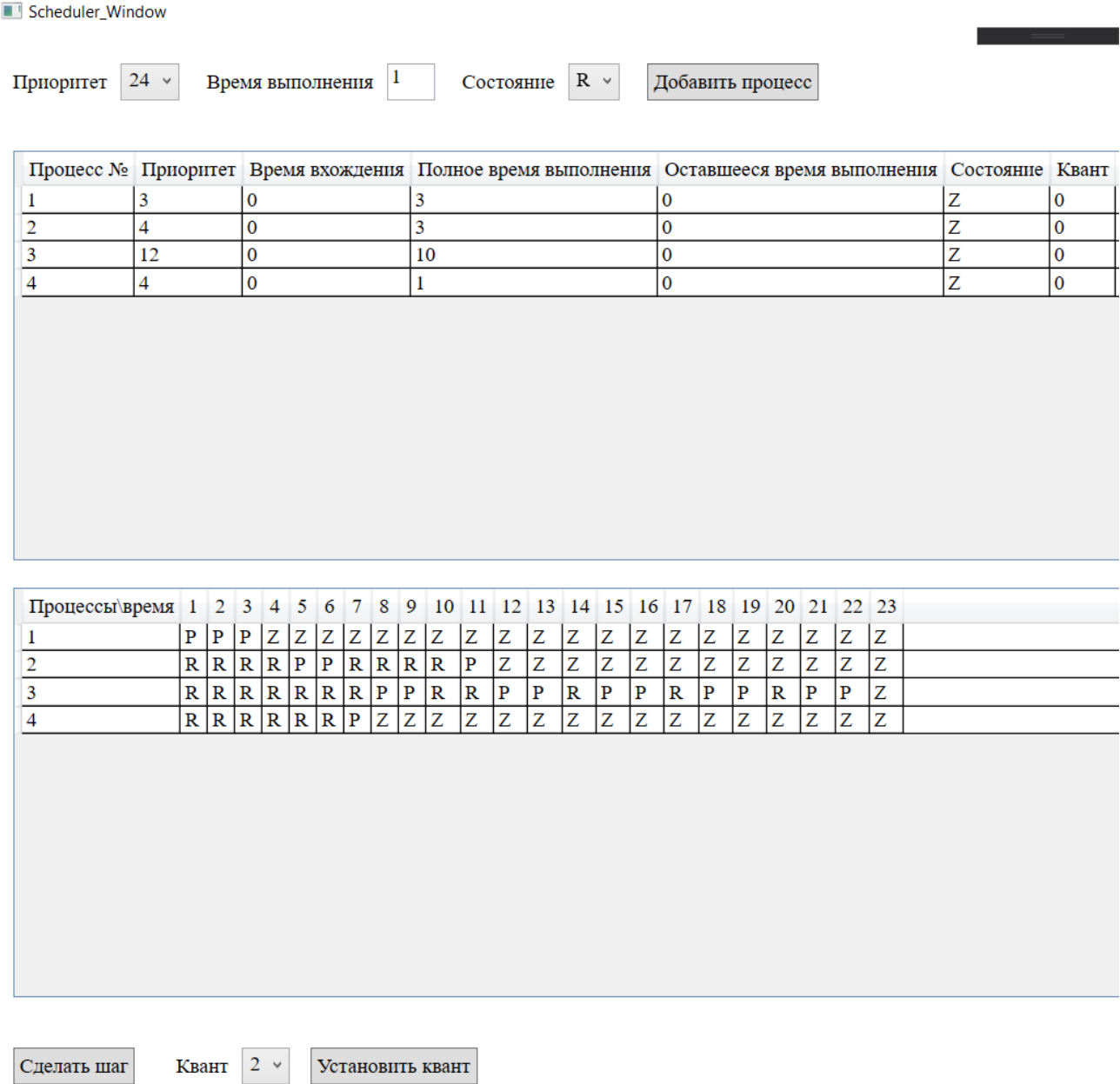


Рисунок 2.7 – Планирование процессов

Свопинг процессов – вытеснение готового к исполнению процесса из оперативной памяти во внешнюю. При этом оперативная память, заполненная процессом, полностью освобождается.

Если процессу недоступны на данный момент определённые ресурсы, то происходит вытеснение данного процесса. Когда появляется возможность, процесс возвращается в состояние готовности, а со временем и выполняется.

3 РЕЖИМЫ РАБОТЫ ПРОЕКТИРУЕМОЙ ОС

3.1 Мультипрограммный режим работы ОС

Мультипрограммным режимом работы называется такой способ организации работы системы, при котором в ее памяти одновременно содержатся программы и данные для выполнения нескольких процессов обработки информации.

Основные особенности мультипрограммного режима:

- в оперативной памяти находятся несколько пользовательских программ в состояниях активности, ожидания или готовности;
- время работы процессора разделяется между программами, находящимися в памяти в состоянии готовности;
- параллельно с работой процессора происходит подготовка и обмен информацией с несколькими устройствами ввода-вывода.

Мультипрограммирование предназначено для повышения пропускной способности вычислительной системы путем более равномерной и полной загрузки всего ее оборудования, в первую очередь процессора. При этом скорость работы самого процессора и номинальная производительность ЭВМ не зависят от мультипрограммирования. [1]

3.2 Многопользовательская защита

Многопользовательская защита – средства ОС, обеспечивающие идентификацию пользователей и различные уровни их привилегий, а также установку и защиту прав собственности на информационные ресурсы, создаваемые в среде ОС. Надежная защита возможна только при наличии специальных аппаратных средств, обеспечивающих защиту оперативной памяти и различные режимы работы процессора. Чтобы начать работать, пользователь должен "войти" в систему, введя свое учетное имя и, возможно, пароль. Пользователь, зарегистрированный в учетных файлах системы и, следовательно,

имеющий учетное имя, называется зарегистрированным пользователем системы.
[2]

3.3 Интерактивный режим работы ОС

Интерактивный режим - режим работы ОС, при котором основными источниками команд являются пользователи, оперативно взаимодействующие с компьютером посредством графического интерфейса. В данном режиме реализовано оптимальное время ответа системы пользователю.

4 СТРУКТУРА ОПЕРАЦИОННОЙ СИСТЕМЫ

4.1 Общая структура проектируемой ОС

Главный компонент ОС – ядро, которое вне зависимости от системы работает в привилегированном режиме. Также помимо ядра в привилегированном режиме работают драйвера – программные модули, которые управляют устройствами.

В состав ОС входят системные библиотеки (DLL) и пользовательские интерфейсы. Интерфейсы делятся на графический интерфейс (GUI) и интерфейс командной строки (CLI).

4.2 Структура ядра проектируемой ОС. Основные функции и назначение файловой подсистемы, подсистемы управления памятью и процессами, подсистемы управления устройствами

Ядро операционной системы состоит из трёх основных подсистем (см. рис. 4.1):

- файловая подсистема;
- подсистема управления процессами и памятью;
- подсистема ввода-вывода.

Файловая подсистема предоставляет доступ к файлам в удобном пользователю виде в соответствии с правами, выполняет запись, чтение, перемещение и удаление файлов.

Подсистема управления процессами и памятью обеспечивает создание и удаление процессов, межпроцессное взаимодействие, планирование процессорного времени, синхронизацию процессов.

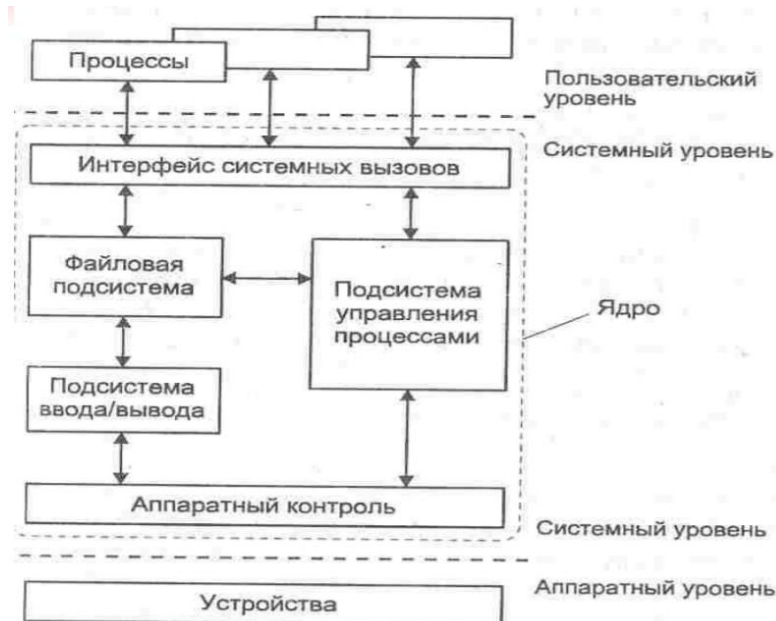


Рисунок 4.1 – Структура ядра ОС

4.3 Структура управляющих блоков базы данных ОС

База данных ОС содержит всю информацию, необходимую ей для функционирования. Состоит из статической и динамической частей.

БД должна содержать следующие управляющие блоки:

- блок управления пользователями;
- блок управления файлами;
- блок управления памятью;
- блок управления процессами;
- блок управления устройствами;
- блок управления сообщениями.

При открытии файла, в БД создается запись для этого файла, при его закрытии, она удаляется.

Блок управления файлами - динамическая часть базы данных, размер этого блока изменяется при изменении системной переменной, определяющей количество открытых файлов.

4.4 Видеотерминал и НМД

Видеотерминал – устройство для дистанционного ввода или вывода информации в вычислительных системах, оснащенное экраном визуального контроля. Видеотерминал состоит из видеомонитора (дисплея) и видеоконтроллера.

Магнитные диски компьютера служат для длительного хранения информации (она не стирается при выключении ЭВМ). При этом в процессе работы данные могут удаляться, а другие записываться.

Накопитель на магнитных дисках сочетает в себе несколько устройств последовательного доступа, причем сокращение времени поиска данных обеспечивается за счет независимости доступа к записи от ее расположения относительно других записей.

5 РАЗРАБОТКА ПРОГРАММ ЭМУЛЯЦИИ ОС

5.1 Описание программных средств

Файловая система была написана на C#, в среде программирования Microsoft Visual Studio 2019.

5.2 Разработка ФС

Для разработки ФС были созданы следующие классы:

- 1) Bitmap – класс, реализовывающий битовую карту;
- 2) Cluster_Control – класс, предоставляющий методы для работы с кластерами;
- 3) Date – класс для работы с датами;
- 4) File_System – класс, связывающий графический интерфейс с логикой программы.
- 5) Group – класс, реализовывающий группу пользователей;
- 6) Inode – класс, реализовывающий инод;
- 7) Superblock – класс, реализовывающий суперблок;
- 8) User – класс, реализовывающий пользователей;

Подробно разобраны методы всех классов в таблицах 5.1 – 5.9.

Таблица 5.1 – класс Bitmap

Команда	Параметры	Результат	Описание
Has_Bit	byte source, int bit_num	bool	Проверяет наличие бита в байте по позиции
Get_Free	uint count_free, uint count_need	uint[]	Получить массив с свободными адресами
Get_Engaged	uint engaged_count	uint[]	Получить массив с занятыми адресами
Change_Free	uint pos, bool is_free, Superblock superblock	int	Изменить флаг в битовой карте

Таблица 5.2 – класс Cluster_Control

Команда	Параметры	Результат	Описание
Read	Inode inode	byte[]	Возвращает содержимое файла по его иноду
Write	byte[] bytes, Inode inode, Superblock superblock, Bitmap bitmap_cluster	int	Записать в файл по его иноду

Таблица 5.3 – класс Date

Команда	Параметры	Результат	Описание
Date_To_Byte	byte[] bytes, int start_index	Date	Перевести дату в байтовое представление
Byte_To_Date	byte[] bytes, int start_index	Date	Перевести байты в дату

Таблица 5.4 – класс File_System

Команда	Параметры	Результат	Описание
Create_File_System	char[] fs_name, char[] fs_type, ushort cluster_size, ulong fs_size, char[] login, char[] password	int	Создание ФС
Load_File_System	char[] login, char[] password	int	Загрузка ФС
Get_Flags	ushort permissions	bool[]	Получить флаги из байтовой записи
Read_Catalog	uint catalog_id	Data_Catalog[]	Получить данные о файлах каталога
Copy	uint inode_id, char[] name	Data_Copy	Скопировать файл
Insert	Data_Copy data_file, uint parrent_catalog_id	void	Вставить файл
Create_File	char[] name, bool is_catalog, uint parrent_catalog_id	uint	Создать новый файл

Продолжение таблицы 5.4

Get_Data_File	uint inode_id	char[]	Получить содержимое файла
Set_Data_File	uint inode_id, char[] chars	int	Установить содержимое файла
Get_Properties_File	uint inode_id, bool is_catalog	string	Получить свойства файла
Get_Permissions	uint inode_id	ushort	Получить флаги файла в байтовом представлении
Check_Rights_Access	uint inode_id	bool[]	Получить триаду прав доступа
Delete_File	uint inode_id	int	Удалить файл
Rename	char[] old_name, char[] new_name, uint parent_catalog_id	int	Переименование файла
Write_Catalog	byte[] bytes, uint catalog_id	int	Запись в каталог
Is_Owner	uint inode_id	bool	Является ли пользователь владельцем файла
Is_Admin		bool	Является ли пользователь администратором
Set_Flags	ushort flags, uint inode_id	void	Установить флаги
Get_All_Groups		Dictionary <string, byte>	Получить словарь групп

Продолжение таблицы 5.4

Add_User	byte gid, char[] login, char[] password	int	Добавить нового пользователя
Add_Group	char[] name, char[] desc	int	Добавить новую группу
Delete_User	char[] login, char[] password	int	Удалить пользователя
Delete_Group	char[] name	int	Удалить группу
Change_User	char[] login, char[] password	int	Сменить пользователя
Change_Group	char[] name	int	Сменить группу

Таблица 5.5 – класс Group

Команда	Параметры	Результат	Описание
Read	char[] input_name	int	Прочитать группу по имени
Add	char[] name, uint description_cluster, char[] description, Cluster_Control cluster	int	Добавить группу
Exist	char[] input_name	int	Проверить существование группы по имени
Delete	byte gid, Superblock superblock, Bitmap bitmap	int	Удалить группу

Таблица 5.6 – класс Inode

Команда	Параметры	Результат	Описание
Read	uint pos	int	Прочитать инод
Create	byte uid, byte gid, Bitmap bitmap_inode, Superblock superblock, Bitmap bitmap_cluster, bool is_catalog = false, ushort permissions = 0b0110_1101_0000_0 000	int	Создать инод
Write	Superblock superblock, Bitmap bitmap_cluster	int	Записать инод
Get_Clusters		uint[]	Получить кластеры инодов
Delete	Superblock superblock, Bitmap bitmap_inode, Bitmap bitmap_cluster	int	Удалить инод
Set_Permissions	ushort perm	void	Установить права

Таблица 5.7 – класс Superblock

Команда	Параметры	Результат	Описание
Create	char[] fs_name, char[] fs_type, ushort cluster_size, ulong fs_size_byte	void	Создание суперблока
Read		int	Считывание данных
Write		int	Запись данных

Таблица 5.8 – класс User

Команда	Параметры	Результат	Описание
Read	char[] input_login, char[] input_password	int	Прочитать пользователя
Add	byte gid, char[] login, char[] password, Superblock superblock	int	Добавить пользователя
Exist	char[] input_login	int	Проверить существование пользователя по логину
Delete	byte uid, Superblock superblock, Bitmap bitmap	int	Удалить пользователя

Все классы состоят в связях между собой, которые отображены на диаграмме классов (см. рис. 5.1)

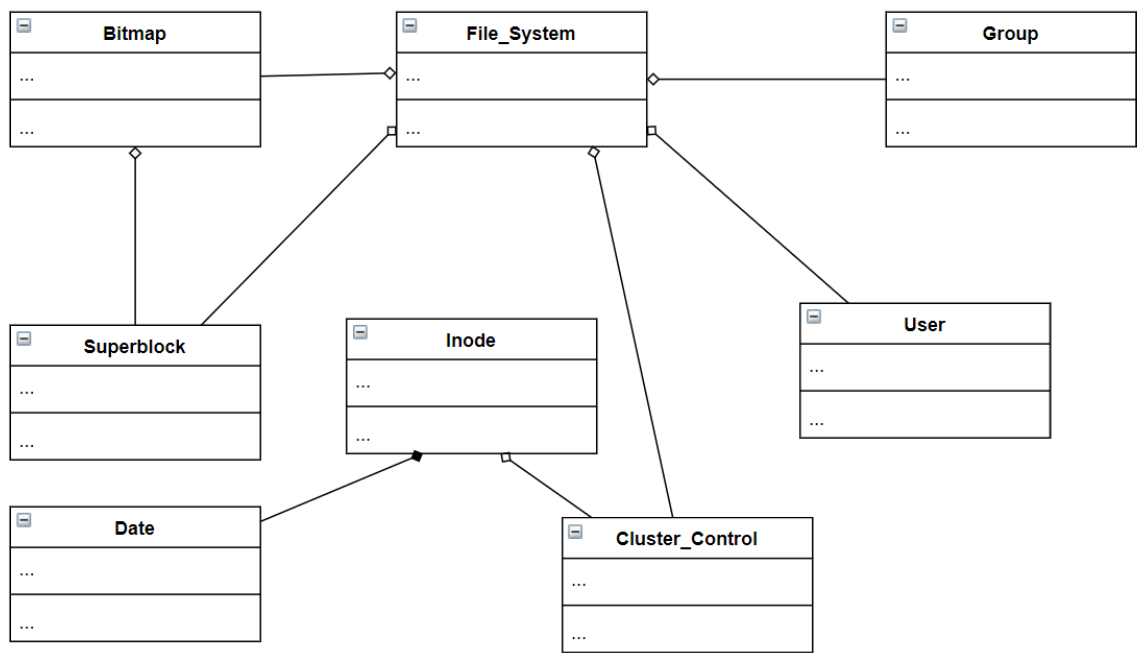


Рисунок 5.1 – Диаграмма классов файловой системы

5.3 Разработка командного интерпретатора

Для представления командного интерпретатора был выбран графический интерфейс пользователя. Взаимодействие с ним производится с помощью клавиатуры и мыши.

5.4 Эмуляция планирования

Для эмуляции был создан класс Process (см. табл. 5.9)

Таблица 5.9 – класс Scheduler

Команда	Параметры	Результат	Описание
Scheduling	Process last	Process	Выполнение алгоритма планирования
Step		void	Сделать шаг
Add		void	Добавить процесс

6 ТЕСТИРОВАНИЕ ПРОГРАММЫ. АНАЛИЗ РЕЗУЛЬТАТОВ

Тестирование ПО – процесс изучения поведения программы, сравнение реального поведения с ожидаемым (см. рис. 6.1-6.).

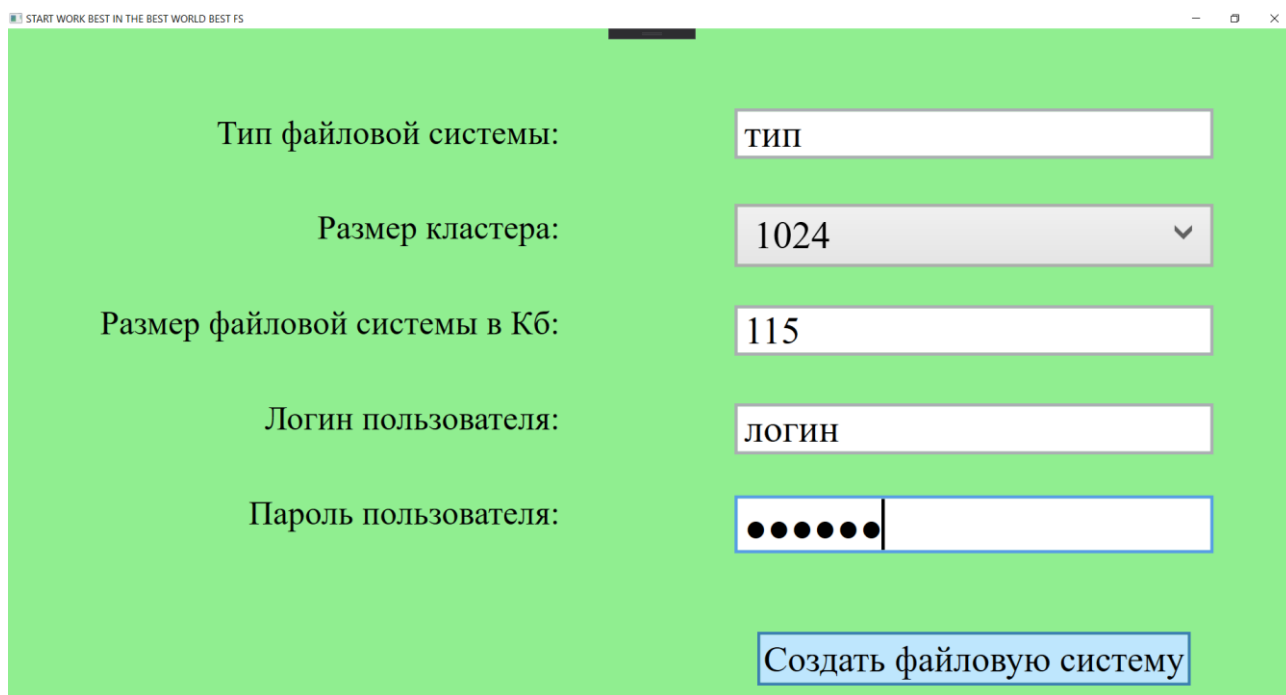


Рисунок 6.1 – Создание файловой системы

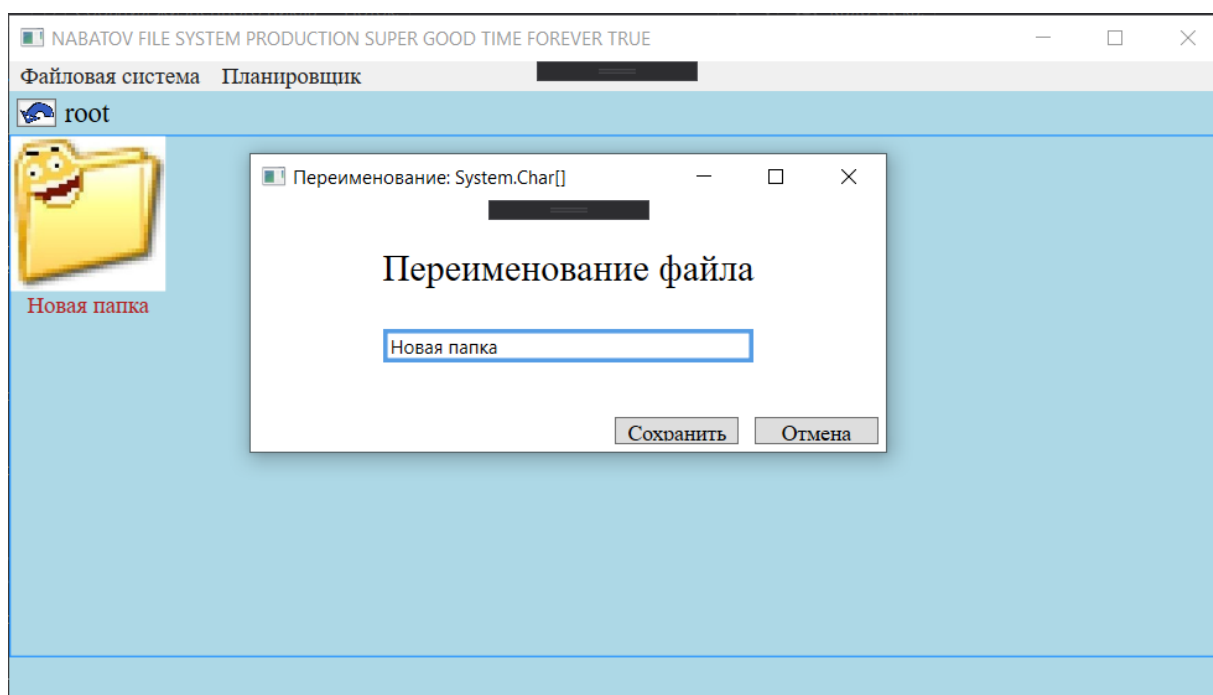


Рисунок 6.2 – Переименование файла

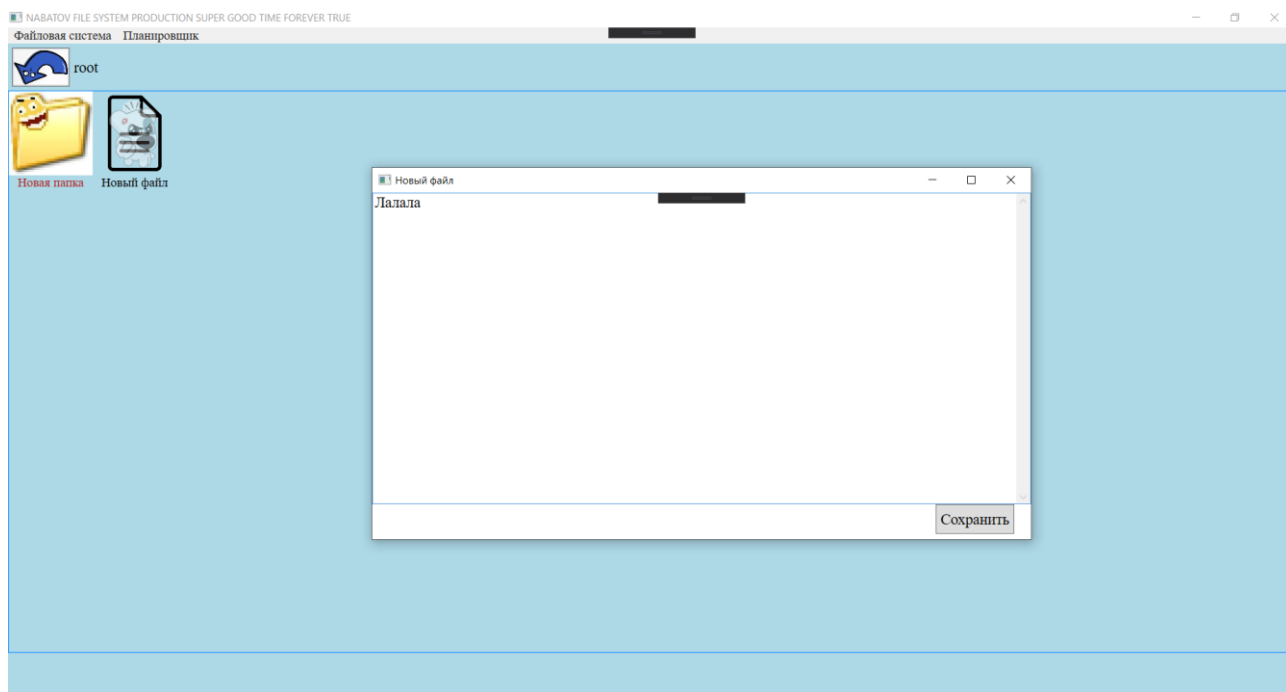


Рисунок 6.3 – Запись в файл

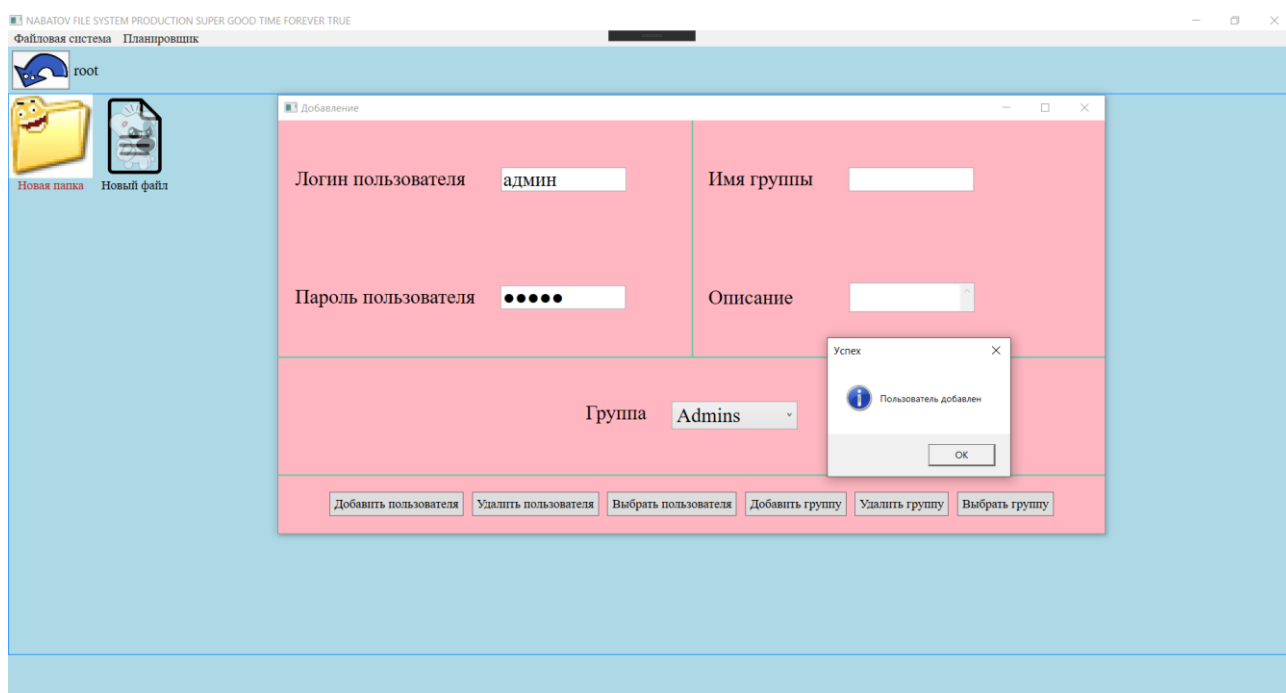


Рисунок 6.4 – Добавление пользователя

ВЫВОДЫ

В ходе работы была разработана файловая система, выполняющая практически все функции современных аналогов, и планировщик процессов с инновационным алгоритмом. Были изучены особенности проектирования ФС, работы межпроцессного взаимодействия и планирования.

ПЕРЕЧЕНЬ ССЫЛОК

1. Мультипрограммный режим работы ЭВМ / Studref.com – [https://studref.com/313870/informatika/multiprogrammnyy_rezhim_raboty#:~:text=Мультипрограммным%20режимом%20работы%20\(многозадачностью\)%20называется,процессов%20обработки%20информации%20\(задач\).](https://studref.com/313870/informatika/multiprogrammnyy_rezhim_raboty#:~:text=Мультипрограммным%20режимом%20работы%20(многозадачностью)%20называется,процессов%20обработки%20информации%20(задач).)
2. Режимы работы проектируемой ОС / Studbooks.net – https://studbooks.net/2320874/informatika/rezhimy_raboty_proektiruемой

ПРИЛОЖЕНИЕ А
ТЕХНИЧЕСКОЕ ЗАДАНИЕ

Утверждено
зав. кафедрой ПИ
_____ Зори А.С.
«_____» _____ 2020 г.

ТЕХНИЧЕСКОЕ ЗАДАНИЕ
по курсовому проекту по дисциплине
«Операционные системы»

студенту группы ПИ-18Б Набатову Арсению Вадимовичу

Тема: “Проектирование гипотетической операционной системы и программная
эмуляция отдельных модулей”

Вариант №44

Задание:

1. Режим работы ОС: мультипрограммный, многопользовательская защита, интерактивный;
2. Управление процессами: квантование времени, относительные приоритеты, динамические приоритеты;
3. Средства взаимодействия процессов: каналы, именованные каналы, разделяемая память;
4. Организация оперативной памяти: страничная;
5. Перечень внешних устройств: видеотерминал, НМД;
6. Организация файловой системы: многоуровневая, битовая карта свободных/занятых кластеров, файлы с последовательным доступом, файлы с прямым доступом;
7. Команды интерпретатора: копирование, вход и выход в систему, переименование, создание и удаление, информация о процессах, изменение приоритета процесса, уничтожение процесса;
8. Эмуляция работы: эмуляция планировщика.

График выполнения курсового проекта:

Неделя	Работа
1-2	Выдача задания и изучение задания
3	Анализ требований к системе и способов их реализации
4-5	Проектирование файловой системы
6-7	Организация процессов

Продолжение таблицы График выполнения курсового проекта

8	Поддержка режимов работы, проектирование общей структуры
9-13	Разработка программ эмуляции
14	Оформление пояснительной записки
15-17	Защита курсового проекта

Дата выдачи задания

01.09.2020

Задание принял

Руководитель проекта

 Чернышова А.В.

ПРИЛОЖЕНИЕ Б

ЛИСТИНГ ПРОГРАММ

```

using System;
using System.Collections.Generic;
using System.IO;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace File_system.File_System
{
    class Bitmap ////////////////////////////////////// Можно
    сделать статическим
    {
        public readonly int OFFSET;
        string fs_file;
        uint count_all;

        public Superblock Superb { private set; get; }
        public Bitmap_Type Type { private set; get; }

        public enum Bitmap_Type
        {
            inode = 1, cluster,
        }

        public Bitmap(Superblock superblock, Bitmap_Type type)
        {
            Superb = superblock;
            Type = type;
            if (type == Bitmap_Type.inode)
            {
                OFFSET = 10084;
                //////////////////////////////////////
                count_all = superblock.Inode_count;
            }
            else if (type == Bitmap_Type.cluster)
            {
                OFFSET = 10084 + (int)((superblock.Inode_count - 1)
/ 8) + 1;
                count_all = superblock.Cluster_count;
            }
            fs_file = superblock.FS_file;
        }

        public static bool Has_Bit(byte source, int bit_num)
        {
            switch (bit_num)
            {
                case 1:
                {
                    return (source & 0b1) != 0;
                }
                case 2:
                {
                    return (source & 0b10) != 0;
                }
                case 3:
                {
                    return (source & 0b100) != 0;
                }
                case 4:
                {
                    return (source & 0b1000) != 0;
                }
                case 5:
                {
                    return (source & 0b1000_0) != 0;
                }
                case 6:
                {
                    return (source & 0b1000_00) != 0;
                }
                case 7:
                {
                    return (source & 0b1000_000) != 0;
                }
                case 8:
                {
                    return (source & 0b1000_0000) != 0;
                }
                default:
                {
                    throw new Exception("Номер бита из
параметра больше общего количества битов в байте");
                }
            }
        }

        /* Получить массив с свободными адресами */
        public uint[] Get_Free(uint count_free, uint count_need)
        {
            /* ext - кол-во байтов, необходимых для
представления всех кластеров. 1 бит - 1 кластер */
            uint ext = (count_all - 1) / 8 + 1; // Округляем
количество считываемых байтов к большему
            uint[] free = new uint[count_need];
            uint count_added = 0;
            short bytes_capacity = (short)(Superb.Cluster_size / 8);
            byte[] bytes = new byte[bytes_capacity];
            //if (count_need > count_free)
            //{
            //    return null;
            //}
            try
            {
                using (BinaryReader reader = new
BinaryReader(File.Open(fs_file, FileMode.Open)))
                {
                    /* offs - сдвиг по байтам внутри битовой карты
                    */
                    uint offs = 0;
                    reader.BaseStream.Seek(OFFSET,
SeekOrigin.Begin);
                    if (ext > bytes_capacity)
                    {
                        for (offs = 0; offs < ext; offs +=
(uint)bytes_capacity) // считывание
                        {
                            bytes = reader.ReadBytes(bytes_capacity);
                            for (uint quad = 0; quad < bytes_capacity;
quad += 4) // обход по 4 байта
                            {

```

```

        int check = BitConverter.ToInt32(bytes,
(int)quad);
        if (check == int.MaxValue) // все биты
установлены в 1, свободных адресов нет
        {
            continue;
        }
        for (uint i = 0; i < 4; i++) // обход по 1 байту
        {
            for (uint j = 0; j < 8; j++) // просмотр
            {
                if (!Has_Bit(bytes[quad + i], 8 - (int)j))
                {
                    free[count_added] = (offs + quad + i)
* 8 + j + 1; // получение порядкового номера свободного
адреса

                    count_added++;
                    if (count_added == count_need)
                    {
                        return free;
                    }
                }
            }
        }
        offs -= (uint)bytes_capacity; // количество
прочитанных байт после выполнения циклов
    }

    byte temp;
    for (uint i = 0; i < ext % bytes_capacity; i += 1) //
Проверка оставшейся части
    {
        temp = reader.ReadByte();
        for (uint k = 0; k < 8; k++)
        {
            if (!Has_Bit(temp, 8 - (int)k))
            {
                free[count_added] = (offs + i) * 8 + (k + 1);
                count_added++;
                if (count_added == count_need)
                {
                    return free;
                }
            }
        }
    }
}
catch (Exception e)
{
    return null;
}
return free;
}

/* Получить массив с занятыми адресами */
public uint[] Get_Engaged(uint engaged_count) //////////
{
    /* ext - кол-во байтов, необходимых для
представления всех кластеров. 1 бит - 1 кластер */
    uint ext = (count_all - 1) / 8 + 1; // Округляем
количество считываемых байтов к большему

```

```

uint[] engaged = new uint[engaged_count];
uint count_added = 0;
short bytes_capacity = (short)(Superb.Cluster_size / 8);
byte[] bytes = new byte[bytes_capacity];
try
{
    using (BinaryReader reader = new
BinaryReader(File.Open(fs_file, FileMode.Open)))
    {
        /* offs - сдвиг по байтам внутри битовой карты
*/
        uint offs = 0;
        reader.BaseStream.Seek(OFFSET,
SeekOrigin.Begin);
        if (ext > bytes_capacity)
        {
            for (offs = 0; offs < ext; offs +=
(uint)bytes_capacity) // считывание
            {
                bytes = reader.ReadBytes(bytes_capacity);
                for (uint quad = 0; quad < bytes_capacity;
quad += 4) // обход по 4 байта
                {
                    int check = BitConverter.ToInt32(bytes,
(int)quad);
                    if (check == 0) // все биты установлены в 0,
занятых кластеров нет
                    {
                        continue;
                    }
                    for (uint i = 0; i < 4; i++) // обход по 1 байту
                    {
                        for (uint j = 0; j < 8; j++) // просмотр
                        {
                            if (!Has_Bit(bytes[quad + i], (int)j + 1))
                            {
                                engaged[count_added] = (offs + quad
+ i) * 8 + j + 1; // получение порядкового номера занятого
адреса

                                count_added++;
                                if (count_added == engaged_count)
                                {
                                    return engaged;
                                }
                            }
                        }
                    }
                }
            }
            offs -= (uint)bytes_capacity; // количество
прочитанных байт после выполнения циклов
        }

        byte temp;
        for (uint i = 0; i < ext % bytes_capacity; i += 1) //
Проверка оставшейся части
        {
            temp = reader.ReadByte();
            for (uint k = 0; k < 8; k++)
            {
                if (Has_Bit(temp, 8 - (int)k))
                {
                    engaged[count_added] = (offs + i) * 8 + (k +
1);

                    count_added++;

```

```

        }
    }
}
}
}
catch (Exception e)
{
    return null;
}
return engaged;
}

public int Change_Free(uint pos, bool is_free, Superblock
superblock)
{
    int offs = (int)((pos - 1) / 8);
    byte byte_temp;
    try
    {
        using (BinaryReader reader = new
BinaryReader(File.Open(fs_file, FileMode.Open)))
        {
            reader.BaseStream.Seek(OFFSET,
SeekOrigin.Begin);
            reader.BaseStream.Seek(offs, SeekOrigin.Current);
            byte_temp = reader.ReadByte();
            reader.BaseStream.Seek(-1, SeekOrigin.Current);
            //byte mask = (byte)(1 << (int)(7 - (pos - 1) % 8)); //
битовая маска для изменения бита на противоположный
            //byte_temp ^= mask; // операция XOR
(исключающее ИЛИ)
            if (Has_Bit(byte_temp, (int)(8 - (pos - 1) % 8)))
            {
                if (is_free)
                {
                    superblock.Free_Increment(Type);
                }
            }
            else
            {
                if (!is_free)
                {
                    superblock.Free_Decrement(Type);
                }
            }
            byte mask = (byte)((is_free ? 0 : 1) << (int)(7 - (pos -
1) % 8));
            byte_temp = (byte)(byte_temp & ~(1 << (int)(7 -
(pos - 1) % 8)) | mask);
        }
        using (BinaryWriter writer = new
BinaryWriter(File.Open(fs_file, FileMode.Open)))
        {
            writer.BaseStream.Seek(OFFSET,
SeekOrigin.Begin);
            writer.BaseStream.Seek(offs, SeekOrigin.Current);
            writer.Write(byte_temp);
        }
    }
    catch (Exception e)
    {
        return -1;
    }
}

```

```

        return 0;
    }
}

using System;
using System.Collections.Generic;
using System.IO;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace File_system.File_System
{
    class Cluster_Control
    {
        public readonly long OFFSET;
        string fs_file;
        public ushort Cluster_size { private set; get; }

        public Cluster_Control(Superblock superblock)
        {
            fs_file = superblock.FS_file;
            OFFSET = superblock.Offset_cluster;
            Cluster_size = superblock.Cluster_size;
        }

        public byte[] Read(uint id)
        {
            byte[] data = new byte[Cluster_size];
            if (id <= 0)
            {
                return null;
            }
            try
            {
                using (BinaryReader reader = new
BinaryReader(File.Open(fs_file, FileMode.Open)))
                {
                    reader.BaseStream.Seek(OFFSET + (id - 1) *
Cluster_size, SeekOrigin.Begin);
                    data = reader.ReadBytes(Cluster_size);
                }
            }
            catch (Exception e)
            {
                return null;
            }
            if (data.Length == 0)
            {
                data = new byte[Cluster_size]; ////////////////
            }
            return data;
        }

        public byte[] Read(Inode inode)
        {
            var clusters = inode.Get_Clusters();
            byte[] data = new byte[inode.File_size];
            int pos_data = 0;
            int left_bytes = (int)inode.File_size;
            try
            {
                for (int i = 0; i < clusters.Length; i++)
                {

```

```

        using (BinaryReader reader = new
BinaryReader(File.Open(fs_file, FileMode.Open)))
        {
            reader.BaseStream.Seek(OFFSET + (clusters[i] -
1) * Cluster_size, SeekOrigin.Begin);
            int read_count = (left_bytes > Cluster_size) ?
Cluster_size : left_bytes;
            Array.Copy(reader.ReadBytes(read_count), 0,
data, pos_data, read_count);
            left_bytes -= Cluster_size;
            pos_data += Cluster_size;
        }
    }
}
catch (Exception e)
{
    return null;
}
return data;
}

public int Write(char[] chars, uint id, int position = 0) //
Encoding добавить можно
{
    try
    {
        using (BinaryWriter writer = new
BinaryWriter(File.Open(fs_file, FileMode.Open)))
        {
            writer.BaseStream.Seek(OFFSET + (id - 1) *
Cluster_size + position, SeekOrigin.Begin);
            writer.Write(Encoding.Default.GetBytes(chars));
        }
    }
    catch (Exception e)
    {
        return -1;
    }
    return 0;
}

public int Write(byte[] bytes, uint id, int position = 0)
{
    try
    {
        using (BinaryWriter writer = new
BinaryWriter(File.Open(fs_file, FileMode.Open)))
        {
            writer.BaseStream.Seek(OFFSET + (id - 1) *
Cluster_size + position, SeekOrigin.Begin);
            writer.Write(bytes);
        }
    }
    catch (Exception e)
    {
        return -1;
    }
    return 0;
}

/* В случае работы с разными кодировками дописать
логику */
public int Write(char[] chars, Inode inode, Superblock
superblock, Bitmap bitmap_cluster)

```

```

{
    var clusters = inode.Get_Clusters(superblock,
bitmap_cluster, (uint)((chars.Length - 1) / Cluster_size + 1));
    int pos_bytes = 0;
    int left_bytes = chars.Length;
    try
    {
        for (int i = 0; i < clusters.Length; i++)
        {
            if (left_bytes > 0)
            {
                using (BinaryWriter writer = new
BinaryWriter(File.Open(fs_file, FileMode.Open)))
                {
                    writer.BaseStream.Seek(OFFSET + (clusters[i] -
1) * Cluster_size, SeekOrigin.Begin);

                    writer.Write(Encoding.Default.GetBytes(chars), pos_bytes,
(left_bytes > Cluster_size) ? Cluster_size : left_bytes);
                }
                left_bytes -= Cluster_size;
                pos_bytes += Cluster_size;
            }
            else
            {
                if (inode.File_size > chars.Length) // Если файл
был большего размера
                {
                    byte[] nulls;
                    if (left_bytes < 0)
                    {
                        nulls = new byte[-left_bytes];
                        try
                        {
                            using (BinaryWriter writer = new
BinaryWriter(File.Open(fs_file, FileMode.Open)))
                            {
                                writer.BaseStream.Seek(OFFSET +
(clusters[i] - 1 - 1) * Cluster_size + (-left_bytes),
SeekOrigin.Begin);

                                writer.Write(nulls, 0, -left_bytes);
                                left_bytes = 0;
                            }
                        }
                        catch (Exception e)
                        {
                            return -1;
                        }
                    }
                    try
                    {
                        using (BinaryWriter writer = new
BinaryWriter(File.Open(fs_file, FileMode.Open)))
                        {
                            writer.BaseStream.Seek(OFFSET +
(clusters[i] - 1) * Cluster_size, SeekOrigin.Begin);
                            writer.Write(new byte[Cluster_size], 0,
Cluster_size);
                        }
                    }
                    catch (Exception e)
                    {
                        return -1;
                    }
                }
            }
        }
    }
}

```

```

        inode.File_size = (uint)chars.Length;
        inode.Write(superblock, bitmap_cluster);
        return 0;
    }
}
}
if (left_bytes > 0)
{
    inode.File_size = (uint)(chars.Length - left_bytes);
    inode.Write(superblock, bitmap_cluster);
    return 1; // Места не хватило
}
}
catch (Exception e)
{
    return -1;
}
inode.File_size = (uint)chars.Length;
inode.Write(superblock, bitmap_cluster);
return 0;
}

public int Write(byte[] bytes, Inode inode, Superblock
superblock, Bitmap bitmap_cluster)
{
    var clusters = inode.Get_Clusters(superblock,
bitmap_cluster, (uint)((bytes.Length - 1) / Cluster_size + 1));
    int pos_bytes = 0;
    int left_bytes = bytes.Length;
    try
    {
        for (int i = 0; i < clusters.Length; i++)
        {
            if (left_bytes > 0)
            {
                using (BinaryWriter writer = new
BinaryWriter(File.Open(fs_file, FileMode.Open)))
                {
                    writer.BaseStream.Seek(OFFSET + (clusters[i] -
1) * Cluster_size, SeekOrigin.Begin);
                    writer.Write(bytes, pos_bytes, (left_bytes >
Cluster_size) ? Cluster_size : left_bytes);
                }
                left_bytes -= Cluster_size;
                pos_bytes += Cluster_size;
            }
            else
            {
                if (inode.File_size > bytes.Length) // Если файл
был БОльшого размера
                {
                    byte[] nulls;
                    if (left_bytes < 0)
                    {
                        nulls = new byte[-left_bytes];
                        try
                        {
                            using (BinaryWriter writer = new
BinaryWriter(File.Open(fs_file, FileMode.Open)))
                            {
                                writer.BaseStream.Seek(OFFSET +
(clusters[i] - 1 - 1) * Cluster_size + (-left_bytes),
SeekOrigin.Begin);
                                writer.Write(nulls, 0, -left_bytes);
                                left_bytes = 0;
                            }
                        }
                    }
                }
            }
        }
    }
    catch (Exception e)
    {
        return -1;
    }
    inode.File_size = (uint)bytes.Length;
    inode.Write(superblock, bitmap_cluster);
    return 0;
}

using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace File_system.File_System
{
    class Date
    {
        public byte Day { private set; get; }
        public byte Month { private set; get; }
        public ushort Year { private set; get; }
        public byte Hour { private set; get; }
        public byte Minute { private set; get; }
        public byte Second { private set; get; }

        public enum Date_Mode
        {

```



```

        empty = 0, now_all,
    }

    public Date(Date_Mode mode)
    {
        switch (mode)
        {
            case Date_Mode.empty:
            {
                Day = 0;
                Month = 0;
                Year = 0;
                Hour = 0;
                Minute = 0;
                Second = 0;
                break;
            }

            case Date_Mode.now_all:
            {
                DateTime dt = DateTime.Now;
                Day = (byte)dt.Day;
                Month = (byte)dt.Month;
                Year = (ushort)dt.Year;
                Hour = (byte)dt.Hour;
                Minute = (byte)dt.Minute;
                Second = (byte)dt.Second;
                break;
            }
        }
    }

    public static byte[] Date_To_Byte(Date date)
    {
        byte[] bytes = new byte[7];
        bytes[0] = date.Day;
        bytes[1] = date.Month;
        Array.Copy(BitConverter.GetBytes(date.Year), 0, bytes,
2, 2);
        bytes[4] = date.Hour;
        bytes[5] = date.Minute;
        bytes[6] = date.Second;
        return bytes;
    }

    public static Date Byte_To_Date(byte[] bytes, int
start_index)
    {
        Date date = new Date(Date_Mode.empty);
        date.Day = bytes[start_index];
        date.Month = bytes[start_index + 1];
        date.Year = BitConverter.ToUInt16(bytes, start_index +
2);
        date.Hour = bytes[start_index + 4];
        date.Minute = bytes[start_index + 5];
        date.Second = bytes[start_index + 6];
        return date;
    }

    public override string ToString()
    {
        return string.Format("{0}.{1}.{2} {3}:{4}:{5}", Day,
Month, Year, Hour, Minute, Second);
    }
}

```

```

    }
    using System;
    using System.Collections.Generic;
    using System.IO;
    using System.Linq;
    using System.Text;
    using System.Threading.Tasks;

    namespace File_system.File_System
    {
        public class File_System
        {
            internal Superblock superblock;
            Group group;
            User user;
            Bitmap bitmap_inode;
            Bitmap bitmap_cluster;
            Cluster_Control cluster;

            public struct Data_Catalog
            {
                public uint inode_id;
                public char[] name;
                public ushort permissions;
            }

            public struct Data_Copy
            {
                public char[] name;
                public ushort permissions;
                public Data_Copy[] children;
                public byte[] data;
                public bool is_catalog;
            }

            public File_System(string fs_file)
            {
                superblock = new Superblock(fs_file);
                group = new Group(fs_file);
                user = new User(fs_file);
            }

            public static bool[] Get_Flags(ushort permissions)
            {
                bool[] flags = new bool[16];
                for (int i = 0; i < 16; i++)
                {
                    flags[i] =
Bitmap.Has_Bit(BitConverter.GetBytes(permissions)[(15 - i) /
8], 8 - (i % 8)); ////////////////
                }
                return flags;
            }

            public int Create_File_System(char[] fs_name, char[]
fs_type, ushort cluster_size, ulong fs_size, char[] login, char[]
password)
            {
                superblock.Create(fs_name, fs_type, cluster_size,
fs_size);
                cluster = new Cluster_Control(superblock);

                group.Add("Admins".ToCharArray(), 3, "Группа для
пользователей с правами администратора".ToCharArray(),
cluster);
            }
        }
    }

```

```

        user.Add(1, login, password, superblock);

        bitmap_inode = new Bitmap(superblock,
        Bitmap.Bitmap_Type.inode);
        bitmap_cluster = new Bitmap(superblock,
        Bitmap.Bitmap_Type.cluster);

        Inode inode = new Inode(superblock.FS_file,
        superblock.Offset_inode, cluster);
        inode.Create(1, 1, bitmap_inode, superblock,
        bitmap_cluster, true);

        byte[] write_bytes = new byte[64];
        Array.Copy(BitConverter.GetBytes(inode.Id), 0,
        write_bytes, 0, 4);
        Array.Copy(Encoding.Default.GetBytes("root"), 0,
        write_bytes, 4, 4);
        Array.Copy(BitConverter.GetBytes(0), 0, write_bytes,
        32, 4);
        Array.Copy(Encoding.Default.GetBytes("-"), 0,
        write_bytes, 36, 1);

        cluster.Write(write_bytes, inode, superblock,
        bitmap_cluster);

        return 0;
    }

    public int Load_File_System(char[] login, char[] password)
    {
        switch (user.Read(login, password))
        {
            case -1:
            {
                throw new Exception("Ошибка открытия ФС!");
            }
            case 1:
            {
                throw new Exception("Пользователя с таким
                логином не существует!");
            }
            case 2:
            {
                throw new Exception("Введён неправильный
                пароль!");
            }
            case 0:
            {
                group.Read(user.Gid);
                superblock.Read();
                cluster = new Cluster_Control(superblock);
                bitmap_inode = new Bitmap(superblock,
                Bitmap.Bitmap_Type.inode);
                bitmap_cluster = new Bitmap(superblock,
                Bitmap.Bitmap_Type.cluster);
                break;
            }
        }
        return 0;
    }

    public Data_Catalog[] Read_Catalog(uint catalog_id)
    {

```

```

        Inode catalog = new Inode(superblock.FS_file,
        superblock.Offset_inode, cluster);
        catalog.Read(catalog_id);
        byte[] bytes = cluster.Read(catalog);
        Data_Catalog[] dc = new Data_Catalog[bytes.Length /
        32];
        for (int i = 0; i < dc.Length; i++)
        {
            dc[i].inode_id = BitConverter.ToUInt32(bytes, i * 32);
            dc[i].name = Encoding.Default.GetChars(bytes, i * 32
            + 4, 28);
            dc[i].permissions = Get_Permissions(dc[i].inode_id);
        }
        return dc;
    }

    public Data_Copy Copy(uint inode_id, char[] name)
    {
        Data_Copy data_file = new Data_Copy();
        Inode file = new Inode(superblock.FS_file,
        superblock.Offset_inode, cluster);
        file.Read(inode_id);
        data_file.name = new char[28];
        Array.Copy(name, 0, data_file.name, 0, 28);
        data_file.permissions = file.Permissions;
        if
        (Bitmap.Has_Bit(BitConverter.GetBytes(data_file.permissions)
        [1], 8))//////////////////////// может быть 0 всё-таки
        (ОЧЕНЬ ВРЯД/Ли)
        {
            data_file.is_catalog = true;
            Data_Catalog[] dc = Read_Catalog(inode_id);
            data_file.children = new Data_Copy[dc.Length - 2];
            for (int i = 2; i < dc.Length; i++)
            {
                data_file.children[i-2] = Copy(dc[i].inode_id,
                dc[i].name);
            }
        }
        else
        {
            data_file.is_catalog = false;
            data_file.data = cluster.Read(file);
        }
        return data_file;
    }

    public void Insert(Data_Copy data_file, uint
    parrent_catalog_id)
    {
        byte suffix = 1;
        uint inode_id = Create_File(data_file.name,
        data_file.is_catalog, parrent_catalog_id);
        int len_name = new
        string(data_file.name).Trim('\0').Length;
        while (inode_id == 0) // файл с таким именем
        существует
        {
            data_file.name[len_name < 28 ? len_name : 27] =
            suffix.ToString()[0];
            suffix++;
            inode_id = Create_File(data_file.name,
            data_file.is_catalog, parrent_catalog_id);
        }
    }

```

```

        Set_Flags(data_file.permissions, inode_id);
        if (data_file.is_catalog)
        {
            //Write_Catalog(data_file.data, inode_id);
            for (int i = 0; i < data_file.children.Length; i++)
            {
                Insert(data_file.children[i], inode_id);
            }
        }
        else
        {
            Set_Data_File(inode_id,
Encoding.Default.GetChars(data_file.data));
        }
    }

    public uint Create_File(char[] name, bool is_catalog, uint
parent_catalog_id)
    {
        Inode parent_catalog = new Inode(superblock.FS_file,
superblock.Offset_inode, cluster);
        parent_catalog.Read(parent_catalog_id);
        byte[] bytes;
        bytes = cluster.Read(parent_catalog);
        char[] check_name = new char[28];
        Array.Copy(name, 0, check_name, 0, name.Length < 28
? name.Length : 28);

        for (int i = 64; i < bytes.Length; i += 32)
        {
            if (Encoding.Default.GetChars(bytes, i + 4,
28).SequenceEqual(check_name))
            {
                return 0; // Файл с таким именем уже существует
            }
        }

        Inode inode = new Inode(superblock.FS_file,
superblock.Offset_inode, cluster);
        inode.Create(user.Uid, group.Gid, bitmap_inode,
superblock, bitmap_cluster, is_catalog);

        byte[] append_bytes = new byte[32];
        Array.Copy(BitConverter.GetBytes(inode.Id), 0,
append_bytes, 0, 4);
        Array.Copy(Encoding.Default.GetBytes(check_name), 0,
append_bytes, 4, 28);

        if (is_catalog)
        {
            byte[] new_catalog_bytes = new byte[64];
            append_bytes.CopyTo(new_catalog_bytes, 0);

            Array.Copy(BitConverter.GetBytes(parent_catalog.Id), 0,
new_catalog_bytes, 32, 4);
            Array.Copy(bytes, 4, new_catalog_bytes, 36, 28); //
Копирование имени родительского каталога
            cluster.Write(new_catalog_bytes, inode, superblock,
bitmap_cluster);
        }

        byte[] write_bytes = new byte[bytes.Length + 32];
        bytes.CopyTo(write_bytes, 0);
        append_bytes.CopyTo(write_bytes, bytes.Length);

```

```

        cluster.Write(write_bytes, parent_catalog, superblock,
bitmap_cluster);

        return inode.Id;
    }

    public char[] Get_Data_File(uint inode_id)
    {
        Inode inode = new Inode(superblock.FS_file,
superblock.Offset_inode, cluster);
        inode.Read(inode_id);
        var bytes = cluster.Read(inode);
        return Encoding.Default.GetChars(bytes);
    }

    public int Set_Data_File(uint inode_id, char[] chars)
    {
        Inode inode = new Inode(superblock.FS_file,
superblock.Offset_inode, cluster);
        inode.Read(inode_id);
        return cluster.Write(chars, inode, superblock,
bitmap_cluster);
    }

    public string Get_Properties_File(uint inode_id, bool
is_catalog)
    {
        Inode inode = new Inode(superblock.FS_file,
superblock.Offset_inode, cluster);
        inode.Read(inode_id);
        StringBuilder user_ar = new StringBuilder();
        StringBuilder group_ar = new StringBuilder();
        StringBuilder other_ar = new StringBuilder();
        {
            if
(Bitmap.Has_Bit(BitConverter.GetBytes(inode.Permissions)[1],
7))
            {
                user_ar.Append("R");
            }
            if
(Bitmap.Has_Bit(BitConverter.GetBytes(inode.Permissions)[1],
6))
            {
                user_ar.Append("W");
            }
            if
(Bitmap.Has_Bit(BitConverter.GetBytes(inode.Permissions)[1],
5))
            {
                user_ar.Append("X");
            }
        }
        {
            if
(Bitmap.Has_Bit(BitConverter.GetBytes(inode.Permissions)[1],
4))
            {
                group_ar.Append("R");
            }
            if
(Bitmap.Has_Bit(BitConverter.GetBytes(inode.Permissions)[1],
3))
            {
                group_ar.Append("W");
            }
        }
    }

```

```

    }
    if
(Bitmap.Has_Bit(BitConverter.GetBytes(inode.Permissions)[1],
2))
    {
        group_ar.Append("X");
    }
    {
        if
(Bitmap.Has_Bit(BitConverter.GetBytes(inode.Permissions)[1],
1))
        {
            other_ar.Append("R");
        }
        if
(Bitmap.Has_Bit(BitConverter.GetBytes(inode.Permissions)[0],
8))
        {
            other_ar.Append("W");
        }
        if
(Bitmap.Has_Bit(BitConverter.GetBytes(inode.Permissions)[0],
7))
        {
            other_ar.Append("X");
        }
    }
    return "Пользователь: " + new
string(user.Get_Login()).Trim('\0') + "\nГруппа: " + new
string(group.Name).Trim('\0') + "\nПрава доступа для
владельца: " + user_ar.ToString() +
"\nПрава доступа для группы: " +
group_ar.ToString() + "\nПрава доступа для остальных: " +
other_ar.ToString() + "\nДата создания: " +
inode.Create_date.ToString();
}

public ushort Get_Permissions(uint inode_id)
{
    if (inode_id == 0)
    {
        return 0;
    }
    Inode inode = new Inode(superblock.FS_file,
superblock.Offset_inode, cluster);
    inode.Read(inode_id);
    return inode.Permissions;
}

public bool[] Check_Rights_Access(uint inode_id)
{
    /// <summary>
    ///
    /// </summary>
    /// <returns>[0] - право на чтение. [1] - право на
запись [2] - право на исполнение</returns>
    bool[] rights = new bool[3];
    bool[] flags = Get_Flags(Get_Permissions(inode_id));
    Inode inode = new Inode(superblock.FS_file,
superblock.Offset_inode, cluster);
    inode.Read(inode_id);
    if (user.Uid == inode.Uid)
    {

```

```

        rights[0] = flags[1];
        rights[1] = flags[2];
        rights[2] = flags[3];
    }
    else if (user.Gid == inode.Gid)
    {
        rights[0] = flags[4];
        rights[1] = flags[5];
        rights[2] = flags[6];
    }
    else
    {
        rights[0] = flags[7];
        rights[1] = flags[8];
        rights[2] = flags[9];
    }
    return rights;
}

public int Delete_File(uint inode_id)
{
    Inode inode = new Inode(superblock.FS_file,
superblock.Offset_inode, cluster);
    inode.Read(inode_id);
    return inode.Delete(superblock, bitmap_inode,
bitmap_cluster);
}

public int Rename(char[] old_name, char[] new_name,
uint parent_catalog_id)
{
    if (new_name.Length == 0)
    {
        return -2;
    }
    Inode parent_catalog = new Inode(superblock.FS_file,
superblock.Offset_inode, cluster);
    parent_catalog.Read(parent_catalog_id);
    byte[] bytes;
    bytes = cluster.Read(parent_catalog);
    char[] new_name_check = new char[28];
    Array.Copy(new_name, 0, new_name_check, 0,
new_name.Length < 28 ? new_name.Length : 28);
    int pos = 0;
    //bool is_catalog = false;
    for (int i = 64; i < bytes.Length; i += 32)
    {
        if (Encoding.Default.GetChars(bytes, i + 4,
28).SequenceEqual(new_name_check))
        {
            return -1; // Такое имя есть уже
        }
        if (Encoding.Default.GetChars(bytes, i + 4,
28).SequenceEqual(old_name))
        {
            pos = i;
            if (Inode.Is_Catalog(BitConverter.ToUInt32(bytes,
pos), superblock))
            {
                Inode children_catalog = new
Inode(superblock.FS_file, superblock.Offset_inode, cluster);
                children_catalog.Read(BitConverter.ToUInt32(bytes, pos));
                cluster.Write(new_name_check,
children_catalog.addr[0], 4);

```

```

    }
    }
}

Array.Copy(Encoding.Default.GetBytes(new_name_check), 0,
bytes, pos + 4, 28);
cluster.Write(bytes, parrent_catalog, superblock,
bitmap_cluster);
return 0;
}

public int Write_Catalog(byte[] bytes, uint catalog_id)
{
    Inode inode = new Inode(superblock.FS_file,
superblock.Offset_inode, cluster);
    inode.Read(catalog_id);
    return cluster.Write(bytes, inode, superblock,
bitmap_cluster);
}

public bool Is_Owner(uint inode_id)
{
    Inode inode = new Inode(superblock.FS_file,
superblock.Offset_inode, cluster);
    inode.Read(inode_id);
    return user.Uid == inode.Uid;
}

public bool Is_Admin()
{
    return user.Gid == 1;
}

public void Set_Flags(ushort flags, uint inode_id)
{
    Inode inode = new Inode(superblock.FS_file,
superblock.Offset_inode, cluster);
    inode.Read(inode_id);
    ushort mask = 1 << 15;
    if (Inode.Is_Catalog(inode_id, superblock))
    {
        flags |= mask;
    }
    else
    {
        flags &= (ushort)~mask;
    }
    inode.Set_Permissions(flags);
    inode.Write(superblock, bitmap_cluster);
}

public Dictionary<string, byte> Get_All_Groups()
{
    return Group.Get_Groups(superblock.FS_file);
}

public int Add_User(byte gid, char[] login, char[]
password)
{
    User new_user = new User(superblock.FS_file);
    return new_user.Add(gid, login, password, superblock);
}

public int Add_Group(char[] name, char[] desc)

```

```

{
    Group new_group = new Group(superblock.FS_file);
    uint desc_cluster =
bitmap_cluster.Get_Free(superblock.Cluster_free_count,
1)[0];
    return new_group.Add(name, desc_cluster, desc,
cluster);
}

public int Delete_User(char[] login, char[] password)
{
    User user = new User(superblock.FS_file);
    int res = user.Read(login, password);
    if (res != 0)
    {
        return res;
    }
    return User.Delete(user.Uid, superblock,
bitmap_inode);
}

public int Delete_Group(char[] name)
{
    Group group = new Group(superblock.FS_file);
    int res = group.Read(name);
    if (res != 0)
    {
        return res;
    }
    if (group.Gid == 1)
    {
        return 2;
    }
    return Group.Delete(group.Gid, superblock,
bitmap_inode);
}

public int Change_User(char[] login, char[] password)
{
    User user = new User(superblock.FS_file);
    int res = user.Read(login, password);
    if (res != 0)
    {
        return res;
    }
    this.user = user;
    return 0;
}

public int Change_Group(char[] name)
{
    Group group = new Group(superblock.FS_file);
    int res = group.Read(name);
    if (res != 0)
    {
        return res;
    }
    this.group = group;
    return 0;
}
}

using System;
using System.Collections.Generic;
using System.IO;

```

```

using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace File_system.File_System
{
    class Group
    {
        public const byte OFFSET = 59;
        string fs_file;
        public byte Life { private set; get; }
        public byte Gid { private set; get; }
        public uint Description_cluster { private set; get; }
        public char[] Name { private set; get; }

        public Group(string fs_file)
        {
            this.fs_file = fs_file;
            Life = 1;
            Name = new char[16];
        }

        public int Read(char[] input_name)
        {
            byte[] bytes = new byte[22];
            char[] check_name = new char[16];
            Array.Copy(input_name, 0, check_name, 0,
input_name.Length < 16 ? input_name.Length : 16);
            int gid = Exist(check_name);
            if (gid > 0)
            {
                {
                    if (Read(gid) == 0)
                    {
                        return 0;
                    }
                }
                else
                {
                    return -1;
                }
            }
            else
            {
                return 1; // Такой группы не существует
            }
        }

        public int Read(int pos)
        {
            byte[] bytes;
            try
            {
                using (BinaryReader reader = new
BinaryReader(File.Open(fs_file, FileMode.Open)))
                {
                    reader.BaseStream.Seek(OFFSET + 22 * (pos - 1),
SeekOrigin.Begin);
                    bytes = new byte[22];
                    bytes = reader.ReadBytes(22);
                }
            }
            catch (Exception e)
            {
                return -1;
            }
            Life = bytes[0];

```

```

        Gid = bytes[1];
        Description_cluster = BitConverter.ToUInt32(bytes, 2);
        Array.Copy(Encoding.Default.GetChars(bytes, 6, 16), 0,
Name, 0, 16);
        return 0;
    }

    public int Add(char[] name, uint description_cluster,
char[] description, Cluster_Control cluster)
    {
        if (Exist(name) > 0)
        {
            return 2; // Группа с таким именем уже существует
        }
        Life = 1;
        Array.Copy(name, 0, Name, 0, (name.Length < 16 ?
name.Length : 16));
        Description_cluster = description_cluster;

        byte[] bytes = new byte[22];
        bytes[0] = Life;
        Array.Copy(BitConverter.GetBytes(Description_cluster),
0, bytes, 2, 4);
        Array.Copy(Encoding.Default.GetBytes(Name), 0, bytes,
6, 16);

        try
        {
            using (BinaryReader reader = new
BinaryReader(File.Open(fs_file, FileMode.Open)))
            {
                Gid = 0;
                reader.BaseStream.Seek(OFFSET,
SeekOrigin.Begin);
                for (byte i = 0; i < 50; i++)
                {
                    if (reader.ReadByte() == 0)
                    {
                        Gid = (byte)(i + 1);
                        break;
                    }
                    reader.BaseStream.Seek(21,
SeekOrigin.Current);
                }
            }
        }
        catch (EndOfStreamException e)
        {
            Gid = 1; // Список групп был пуст
        }
        catch (Exception e)
        {
            return -1;
        }

        if (Gid == 0)
        {
            return 1; // Ошибка: список групп заполнен
        }
        bytes[1] = Gid;
        try
        {
            using (BinaryWriter writer = new
BinaryWriter(File.Open(fs_file, FileMode.Open)))
            {

```

```

        writer.Seek(OFFSET + 22 * (Gid - 1),
SeekOrigin.Begin);
        writer.Write(bytes);
    }
    cluster.Write(description, Description_cluster);
}
catch (Exception e)
{
    return -1;
}
return 0;
}

public int Exist(char[] input_name)
{
    byte[] bytes = new byte[22];
    char[] check_name = new char[16];

    Array.Copy(input_name, 0, check_name, 0,
input_name.Length < 16 ? input_name.Length : 16);
    try
    {
        using (BinaryReader reader = new
BinaryReader(File.Open(fs_file, FileMode.Open)))
        {
            reader.BaseStream.Seek(OFFSET,
SeekOrigin.Begin);
            for (int i = 0; i < 50; i++)
            {
                bytes = reader.ReadBytes(22);
                if (bytes[0] == 1)
                {
                    if (Encoding.Default.GetChars(bytes, 6,
16).SequenceEqual(check_name))
                    {
                        return bytes[1]; // Группа с таким именем
есть в системе
                    }
                }
            }
        }
    }
    catch (Exception e)
    {
        return -2; // заглушка
    }
    return -1; // Такой группы не существует
}

public static Dictionary<string, byte> Get_Groups(string
fs_file)
{
    var dict = new Dictionary<string, byte>();
    byte[] bytes = new byte[22];
    try
    {
        using (BinaryReader reader = new
BinaryReader(File.Open(fs_file, FileMode.Open)))
        {
            reader.BaseStream.Seek(OFFSET,
SeekOrigin.Begin);
            for (int i = 0; i < 50; i++)
            {
                bytes = reader.ReadBytes(22);
                if (bytes[0] == 1)

```

```

        {
            dict.Add(Encoding.Default.GetString(bytes, 6,
16), bytes[1]);
        }
    }
    catch (Exception e)
    {
        return null; // заглушка
    }
    return dict;
}

public static int Delete(byte gid, Superblock superblock,
Bitmap bitmap)
{
    byte[] nulls = new byte[22];
    try
    {
        using (BinaryWriter writer = new
BinaryWriter(File.Open(superblock.FS_file, FileMode.Open)))
        {
            writer.BaseStream.Seek(OFFSET + (gid - 1) * 22,
SeekOrigin.Begin);
            writer.Write(nulls);
        }
    }
    catch (Exception e)
    {
        return -1; // заглушка
    }
    return Inode.Update_Inode_After_Delete_Group(gid,
superblock, bitmap);
}

}
using System;
using System.Collections.Generic;
using System.IO;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace File_system.File_System
{
    class Inode
    {
        public readonly int OFFSET;
        string fs_file;
        public byte Uid { private set; get; }
        public byte Gid { private set; get; }
        public ushort Permissions { private set; get; }
        public uint File_size { internal set; get; }
        public Date Create_date { private set; get; }
        public Date Mod_date { private set; get; }
        internal uint[] addr;

        public uint Id { private set; get; }
        public uint File_size_cluster_max { private set; get; }
        public Cluster_Control Cluster { private set; get; }

        public Inode(string fs_file, int offset, Cluster_Control
cluster)
        {

```

```

        this.fs_file = fs_file;
        OFFSET = offset;
        addr = new uint[13];
        Cluster = cluster;
        File_size_cluster_max = 10 + 3 *
(uint)(Cluster.Cluster_size / 4);
    }

    public int Read(uint pos)
    {
        byte[] bytes;
        try
        {
            using (BinaryReader reader = new
BinaryReader(File.Open(fs_file, FileMode.Open)))
            {
                reader.BaseStream.Seek(OFFSET + 74 * (pos - 1),
SeekOrigin.Begin);
                bytes = new byte[74];
                bytes = reader.ReadBytes(74);
            }
        }
        catch (Exception e)
        {
            return -1;
        }
        Id = pos;
        Uid = bytes[0];
        Gid = bytes[1];
        Permissions = BitConverter.ToUInt16(bytes, 2);
        File_size = BitConverter.ToUInt32(bytes, 4);
        Create_date = Date.Byte_To_Date(bytes, 8);
        Mod_date = Date.Byte_To_Date(bytes, 15);
        for (int i = 0; i < 13; i++)
        {
            addr[i] = BitConverter.ToUInt32(bytes, 22 + i * 4);
        }
        return 0;
    }

    public int Create(byte uid, byte gid, Bitmap bitmap_inode,
Superblock superblock, Bitmap bitmap_cluster, bool
is_catalog = false, ushort permissions =
0b0110_1101_0000_0000)
    {
        if (is_catalog)
        {
            permissions |= 1 << 15; // 16-ый бит
устанавливается в 1
        }
        Uid = uid;
        Gid = gid;
        Permissions = permissions;
        File_size = 0;
        Create_date = new Date(Date_Mode.now_all);
        Mod_date = Create_date;

        Id =
bitmap_inode.Get_Free(superblock.Inode_free_count, 1)[0];
        Write(superblock, bitmap_cluster);
        bitmap_inode.Change_Free(Id, false, superblock);

        return 0;
    }

```

```

    public int Write(Superblock superblock, Bitmap
bitmap_cluster)
    {
        Update_Addr(superblock, bitmap_cluster);
        byte[] bytes = new byte[74];
        bytes[0] = Uid;
        bytes[1] = Gid;
        Array.Copy(BitConverter.GetBytes(Permissions), 0,
bytes, 2, 2);
        Array.Copy(BitConverter.GetBytes(File_size), 0, bytes,
4, 4);
        Array.Copy(Date.Date_To_Byte(Create_date), 0, bytes,
8, 7);
        Array.Copy(Date.Date_To_Byte(Mod_date), 0, bytes,
15, 7);
        for (int i = 0; i < 13; i++)
        {
            Array.Copy(BitConverter.GetBytes(addr[i]), 0, bytes,
22 + i * 4, 4);
        }
        try
        {
            using (BinaryWriter writer = new
BinaryWriter(File.Open(fs_file, FileMode.Open)))
            {
                writer.BaseStream.Seek(OFFSET + 74 * (Id - 1),
SeekOrigin.Begin);
                writer.Write(bytes);
            }
        }
        catch (Exception e)
        {
            return -1;
        }
        return 0;
    }

    public uint[] Get_Clusters()
    {
        List<uint> id_clusters = new List<uint>(((int)((File_size -
1) / Cluster.Cluster_size + 1)); // -1, +1 для округления в
большую сторону
        for (int i = 0; i < 10; i++)
        {
            if (addr[i] == 0)
            {
                return id_clusters.ToArray();
            }
            id_clusters.Add(addr[i]);
        }
        for (int i = 10; i < 13; i++)
        {
            if (addr[i] == 0)
            {
                return id_clusters.ToArray();
            }
            byte[] temp = Cluster.Read(addr[i]);
            for (int j = 0; j < Cluster.Cluster_size; j += 4)
            {
                uint rd = BitConverter.ToUInt32(temp, j);
                if (rd == 0)
                {
                    return id_clusters.ToArray();
                }
                id_clusters.Add(rd);
            }
        }
    }

```



```

    }
    }
    return id_clusters.ToArray();
}

public uint[] Get_Clusters(Superblock superblock, Bitmap
bitmap_cluster, uint need_count)
{
    var id_clusters_now = Get_Clusters();
    if (id_clusters_now.Length >= need_count)
    {
        return id_clusters_now;
    }
    var id_clusters = new List<uint>(id_clusters_now);
    need_count -= (uint)id_clusters.Count;

    var need_clusters = new
List<uint>(bitmap_cluster.Get_Free(superblock.Cluster_free_c
ount, need_count));
    for (int i = 0; i < 10; i++)
    {
        if (addr[i] == 0)
        {
            addr[i] = need_clusters[0];
            id_clusters.Add(addr[i]);
            bitmap_cluster.Change_Free(addr[i], false,
superblock);
            need_clusters.RemoveAt(0);
            if (need_clusters.Count == 0)
            {
                Write(superblock, bitmap_cluster);
                return id_clusters.ToArray();
            }
        }
    }
    for (int i = 10; i < 13; i++)
    {
        if (addr[i] == 0)
        {
            addr[i] =
bitmap_cluster.Get_Free(superblock.Cluster_free_count,
1)[0];
            bitmap_cluster.Change_Free(addr[i], false,
superblock);
        }
        byte[] temp = Cluster.Read(addr[i]);
        for (int j = 0; j < Cluster.Cluster_size; j += 4)
        {
            uint rd = BitConverter.ToUInt32(temp, j);
            if (rd == 0)
            {
                try
                {
                    using (BinaryWriter writer = new
BinaryWriter(File.Open(fs_file, FileMode.Open)))
                    {
                        id_clusters.Add(need_clusters[0]);
                        bitmap_cluster.Change_Free(addr[i], false,
superblock);
                    }
                }
                catch { }
            }
            Cluster.Write(BitConverter.GetBytes(need_clusters[0]),
addr[i], j); // запись номера нового кластера в кластер с
кластерами =)
        }
    }
}

```

```

bitmap_cluster.Change_Free(need_clusters[0], false,
superblock);
        need_clusters.RemoveAt(0);
        if (need_clusters.Count == 0)
        {
            Write(superblock, bitmap_cluster);
            return id_clusters.ToArray();
        }
    }
    catch (Exception e)
    {
        return null;
    }
}
return null; // Место в файле закончилось
}

public int Delete(Superblock superblock, Bitmap
bitmap_inode, Bitmap bitmap_cluster)
{
    bitmap_inode.Change_Free(Id, true, superblock);
    Cluster.Write(new byte[1], this, superblock,
bitmap_cluster);
    try
    {
        using (BinaryWriter writer = new
BinaryWriter(File.Open(fs_file, FileMode.Open)))
        {
            writer.BaseStream.Seek(OFFSET + 74 * (Id - 1),
SeekOrigin.Begin);
            writer.Write(new byte[74]);
        }
    }
    catch (Exception e)
    {
        return -1;
    }
    return 0;
}

public int Update_Addr(Superblock superblock, Bitmap
bitmap_cluster)
{
    byte[] bytes = new byte[Cluster.Cluster_size];
    byte[] nulls = new byte[Cluster.Cluster_size];
    for (int i = 0; i < 10; i++)
    {
        try
        {
            using (BinaryReader reader = new
BinaryReader(File.Open(fs_file, FileMode.Open)))
            {
                bytes = Cluster.Read(addr[i]);
            }
        }
        catch (Exception e)
        {
            return -1;
        }
        if (bytes == null) // addr уже 0
        {

```

```

        break;
    }
    else if (bytes.SequenceEqual(nulls)) // addr есть, но в
нём пусто
    {
        for (int j = i; j < 10; j++)
        {
            bitmap_cluster.Change_Free(addr[j], true,
superblock);
            addr[j] = 0;
        }
    }
    byte[] cluster_temp = new byte[Cluster.Cluster_size];
    for (int i = 10; i < 13; i++)
    {
        cluster_temp = Cluster.Read(addr[i]);
        if (cluster_temp == null)
        {
            continue;
        }
        else if (cluster_temp.SequenceEqual(nulls))
        {
            bitmap_cluster.Change_Free(addr[i], true,
superblock);
            addr[i] = 0;
            continue;
        }
        for (int j = 0; j < cluster_temp.Length; j += 4)
        {
            uint rd = BitConverter.ToUInt32(cluster_temp, j);
            if (rd == 0)
            {
                break;
            }
            else
            {
                bitmap_cluster.Change_Free(rd, true,
superblock);
            }
            try
            {
                using (BinaryWriter writer = new
BinaryWriter(File.Open(fs_file, FileMode.Open)))
                {
                    writer.BaseStream.Seek(Cluster.OFFSET +
(rd - 1) * Cluster.Cluster_size, SeekOrigin.Begin);
                    writer.Write(new
byte[Cluster.Cluster_size]);
                }
            }
            catch (Exception e)
            {
                return -1;
            }
        }
    }
    try
    {
        using (BinaryWriter writer = new
BinaryWriter(File.Open(fs_file, FileMode.Open)))
        {
            writer.BaseStream.Seek(Cluster.OFFSET +
(addr[i] - 1) * Cluster.Cluster_size, SeekOrigin.Begin);
            writer.Write(new byte[Cluster.Cluster_size]);

```

```

        bitmap_cluster.Change_Free(addr[i], true,
superblock);
        addr[i] = 0;
    }
    catch (Exception e)
    {
        return -1;
    }
    return 0;
}

public static bool Is_Catalog(uint id, Superblock
superblock)
{
    ushort flags;
    try
    {
        using (BinaryReader reader = new
BinaryReader(File.Open(superblock.FS_file, FileMode.Open)))
        {
            reader.BaseStream.Seek(superblock.Offset_inode
+ 74 * (id - 1) + 2, SeekOrigin.Begin);
            flags = reader.ReadUInt16();
        }
    }
    catch (Exception e)
    {
        throw e;
    }
    if (Bitmap.Has_Bit(BitConverter.GetBytes(flags)[1], 8))
    // Проверяем 2 байт, потому что записаны они в обратном
порядке (почему-то)
    {
        return true;
    }
    else
    {
        return false;
    }
}

public void Set_Permissions(ushort perm)
{
    Permissions = perm;
}

internal static int Update_Inode_After_Delete_User(byte
uid, Superblock superblock, Bitmap bitmap)
{
    List<uint> delete_pos_list = new List<uint>();

    try
    {
        using (BinaryReader reader = new
BinaryReader(File.Open(superblock.FS_file, FileMode.Open)))
        {
            foreach (var item in
bitmap.Get_Engaged(superblock.Inode_count))
            {
                reader.BaseStream.Seek(bitmap.OFFSET + 74 *
(item - 1), SeekOrigin.Begin);
                if (reader.ReadByte() == uid)
                {

```

```

        delete_pos_list.Add(item);
    }
}
}
catch (Exception e)
{
    return -1;
}

try
{
    using (BinaryWriter writer = new
BinaryWriter(File.Open(superblock.FS_file, FileMode.Open)))
    {
        foreach (var item in delete_pos_list)
        {
            writer.BaseStream.Seek(superblock.Offset_inode
+ item * 74, SeekOrigin.Begin);
            writer.Write((byte)1);
        }
    }
}
catch (Exception e)
{
    return -1; // заглушка
}

return 0;
}

internal static int
Update_Inode_After_Delete_Group(byte gid, Superblock
superblock, Bitmap bitmap)
{
    List<uint> delete_pos_list = new List<uint>();

    try
    {
        uint[] engaged =
bitmap.Get_Engaged(superblock.Inode_count -
superblock.Inode_free_count);
        using (BinaryReader reader = new
BinaryReader(File.Open(superblock.FS_file, FileMode.Open)))
        {
            foreach (var item in engaged)
            {
                reader.BaseStream.Seek(bitmap.OFFSET + 74 *
(item - 1) + 1, SeekOrigin.Begin);
                if (reader.ReadByte() == gid)
                {
                    delete_pos_list.Add(item);
                }
            }
        }
    }
}
catch (Exception e)
{
    return -1;
}

try
{
    using (BinaryWriter writer = new
BinaryWriter(File.Open(superblock.FS_file, FileMode.Open)))

```

```

    {
        foreach (var item in delete_pos_list)
        {
            writer.BaseStream.Seek(superblock.Offset_inode
+ item * 74 + 1, SeekOrigin.Begin);
            writer.Write((byte)1);
        }
    }
}
catch (Exception e)
{
    return -1; // заглушка
}

return 0;
}
}

using System;
using System.Collections.Generic;
using System.IO;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace File_system.File_System
{
    class Superblock
    {
        public const byte OFFSET = 0;
        public string FS_file { private set; get; }

        public char[] FS_name { private set; get; }
        public char[] FS_type { private set; get; }
        public ushort Cluster_size { private set; get; }
        public uint Inode_count { private set; get; }
        public uint Inode_free_count { private set; get; }
        public uint Cluster_count { private set; get; }
        public uint Cluster_free_count { private set; get; }
        public byte User_count { internal set; get; } /// МОЖЕТ
ДОБАВИТЬ ИНКРЕМЕНТ И ДЕКРЕМЕНТ
        public ulong FS_size { private set; get; } /// Текущий
размер ФС в байтах

        public ulong FS_size_max { private set; get; }
        public int Offset_inode { private set; get; }
        public long Offset_cluster { private set; get; }

        public Superblock(string fs_file)
        {
            FS_file = fs_file;
            FS_name = new char[16];
            FS_type = new char[16];
        }

        public void Create(char[] fs_name, char[] fs_type, ushort
cluster_size, ulong fs_size_byte)
        {
            Array.Copy(fs_name, 0, FS_name, 0, (fs_name.Length <
16 ? fs_name.Length : 16));
            Array.Copy(fs_type, 0, FS_type, 0, (fs_type.Length < 16
? fs_type.Length : 16));
            Cluster_size = cluster_size;
            Cluster_count = (uint)(fs_size_byte / cluster_size);

```

```

        Cluster_free_count = Cluster_count; // При
        монтировании системы 1 кластер нужен под папку root
        Inode_count = Cluster_count / 2;
        Inode_free_count = Inode_count; // При
        монтировании системы 1 инод нужен под папку root
        User_count = 1;
        FS_size = 10084 + (Inode_count - 1) / 8 + 1 +
        (Cluster_count - 1) / 8 + 1;

        FS_size_max = fs_size_byte;
        Offset_inode = (int)(10084 + (Inode_count - 1) / 8 + 1 +
        (Cluster_count - 1) / 8 + 1);
        Offset_cluster = Offset_inode + 74 * Inode_count;
        Write();
    }

    public int Read()
    {
        byte[] bytes;
        try
        {
            using (BinaryReader reader = new
            BinaryReader(File.Open(FS_file, FileMode.Open)))
            {
                bytes = new byte[59];
                bytes = reader.ReadBytes(OFFSET + 59);
            }
        }
        catch (Exception e)
        {
            return -1;
        }
        Array.Copy(Encoding.Default.GetChars(bytes, 0, 16), 0,
        FS_name, 0, 16);
        Array.Copy(Encoding.Default.GetChars(bytes, 16, 16),
        0, FS_type, 0, 16);
        Cluster_size = BitConverter.ToUInt16(bytes, 32);
        Inode_count = BitConverter.ToUInt32(bytes, 34);
        Inode_free_count = BitConverter.ToUInt32(bytes, 38);
        Cluster_count = BitConverter.ToUInt32(bytes, 42);
        Cluster_free_count = BitConverter.ToUInt32(bytes, 46);
        User_count = bytes[50];
        FS_size = BitConverter.ToUInt64(bytes, 51);

        FS_size_max = Cluster_count * Cluster_size;
        Offset_inode = (int)(10084 + (Inode_count - 1) / 8 + 1 +
        (Cluster_count - 1) / 8 + 1);
        Offset_cluster = Offset_inode + 74 * Inode_count;
        return 0;
    }

    public int Write()
    {
        byte[] bytes = new byte[59];
        Array.Copy(Encoding.Default.GetBytes(FS_name), 0,
        bytes, 0, 16);
        Array.Copy(Encoding.Default.GetBytes(FS_type), 0,
        bytes, 16, 16);
        Array.Copy(BitConverter.GetBytes(Cluster_size), 0,
        bytes, 32, 2);
        Array.Copy(BitConverter.GetBytes(Inode_count), 0,
        bytes, 34, 4);
        Array.Copy(BitConverter.GetBytes(Inode_free_count),
        0, bytes, 38, 4);

```

```

        Array.Copy(BitConverter.GetBytes(Cluster_count), 0,
        bytes, 42, 4);
        Array.Copy(BitConverter.GetBytes(Cluster_free_count),
        0, bytes, 46, 4);
        Array.Copy(BitConverter.GetBytes(User_count), 0,
        bytes, 50, 1);
        Array.Copy(BitConverter.GetBytes(FS_size), 0, bytes,
        51, 8);
        try
        {
            using (BinaryWriter writer = new
            BinaryWriter(File.Open(FS_file, FileMode.OpenOrCreate)))
            {
                writer.Write(bytes, OFFSET, 59);
            }
        }
        catch (Exception e)
        {
            return -1;
        }
        return 0;
    }

    public void Free_Decrement(Bitmap.Bitmap_Type type)
    {
        if (type == Bitmap.Bitmap_Type.inode)
        {
            Inode_free_count--;
        }
        else if (type == Bitmap.Bitmap_Type.cluster)
        {
            Cluster_free_count--;
        }
        Write();
    }

    public void Free_Increment(Bitmap.Bitmap_Type type)
    {
        if (type == Bitmap.Bitmap_Type.inode)
        {
            Inode_free_count++;
        }
        else if (type == Bitmap.Bitmap_Type.cluster)
        {
            Cluster_free_count++;
        }
        Write();
    }

    public void Update()
    {
        // Подсчёт свободных инодов/кластеров в битмапах,
        кол-во пользователей (ЗАПИСЬ!!!!!!)
    }
}

using System;
using System.Collections.Generic;
using System.IO;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace File_system.File_System
{

```

```

class User
{
    public const short OFFSET = 1159;
    string fs_file;
    public byte Life { private set; get; }
    public byte Uid { private set; get; }
    public byte Gid { private set; get; }
    char[] login;
    char[] password; ///
    //////////////////////////////////////////
    СДЕЛАТЬ ОБНУЛЕНИЕ ПОЛЕЙ, В СЛУЧАЕ ОШИБОЧНОГО
    ДОБАВЛЕНИЯ, ЧТОБЫ В ПОЛЯХ НЕ БЫЛО МУСОРА

    public User(string fs_file)
    {
        this.fs_file = fs_file;
        Life = 0;
        login = new char[16];
        password = new char[16];
    }

    public char[] Get_Login()
    {
        return login;
    }

    public int Read(char[] input_login, char[] input_password)
    {
        byte[] bytes = new byte[35];
        char[] check_password = new char[16];
        Array.Copy(input_password, 0, check_password, 0,
input_password.Length < 16 ? input_password.Length : 16);
        int uid = Exist(input_login);
        if (uid > 0)
        {
            if (Read(uid) == 0)
            {
                if (password.SequenceEqual(check_password))
                {
                    return 0;
                }
                else
                {
                    return 2; // Неправильный пароль
                }
            }
            else
            {
                return -1;
            }
        }
        else
        {
            return 1; // Такого пользователя не существует
        }
    }

    public int Read(int pos)
    {
        byte[] bytes;
        try
        {
            using (BinaryReader reader = new
BinaryReader(File.Open(fs_file, FileMode.Open)))
            {

```

```

                reader.BaseStream.Seek(OFFSET + 35 * (pos - 1),
SeekOrigin.Begin);
                bytes = new byte[35];
                bytes = reader.ReadBytes(35);
            }
        }
        catch (Exception e)
        {
            return -1;
        }
        Life = bytes[0];
        Uid = bytes[1];
        Gid = bytes[2];
        Array.Copy(Encoding.Default.GetChars(bytes, 3, 16), 0,
login, 0, 16);
        Array.Copy(Encoding.Default.GetChars(bytes, 19, 16),
0, password, 0, 16);
        return 0;
    }

    public int Add(byte gid, char[] login, char[] password,
Superblock superblock)
    {
        if (Exist(login) > 0)
        {
            return 2; // Пользователь с таким логином уже
существует
        }
        Life = 1;
        Gid = gid;
        Array.Copy(login, 0, this.login, 0, (login.Length < 16 ?
login.Length : 16));
        Array.Copy(password, 0, this.password, 0,
(password.Length < 16 ? password.Length : 16));

        byte[] bytes = new byte[35];
        bytes[0] = Life;
        bytes[2] = Gid;
        Array.Copy(Encoding.Default.GetBytes(this.login), 0,
bytes, 3, 16);
        Array.Copy(Encoding.Default.GetBytes(this.password),
0, bytes, 19, 16);

        if (superblock.User_count < 255)
        {
            try
            {
                using (BinaryReader reader = new
BinaryReader(File.Open(fs_file, FileMode.Open)))
                {
                    Uid = 0;
                    reader.BaseStream.Seek(OFFSET,
SeekOrigin.Begin);
                    for (int i = 0; i < 255; i++)
                    {
                        if (reader.ReadByte() == 0)
                        {
                            Uid = (byte)(i + 1);
                            break;
                        }
                    }
                    reader.BaseStream.Seek(34,
SeekOrigin.Current);
                }
            }
        }
    }

```

```

    }
    catch (EndOfStreamException e)
    {
        Uid = 1; // Список пользователей был пуст
    }
    catch (Exception e)
    {
        return -1;
    }

    if (Uid == 0)
    {
        return 1; // Ошибка: список пользователей
        заполнен
    }
    bytes[1] = Uid;
    try
    {
        using (BinaryWriter writer = new
        BinaryWriter(File.Open(fs_file, FileMode.Open)))
        {
            writer.Seek(OFFSET + 35 * (Uid - 1),
            SeekOrigin.Begin);
            writer.Write(bytes);
            superblock.User_count++;
        }
    }
    catch (Exception e)
    {
        return -1;
    }
    else
    {
        return 1; // Ошибка: список пользователей
        заполнен
    }
    return 0;
}

public static int Delete(byte uid, Superblock superblock,
Bitmap bitmap)
{
    byte[] nulls = new byte[35];
    try
    {
        using (BinaryWriter writer = new
        BinaryWriter(File.Open(superblock.FS_file, FileMode.Open)))
        {
            writer.BaseStream.Seek(OFFSET + (uid - 1) * 35,
            SeekOrigin.Begin);
            writer.Write(nulls);
        }
    }
    catch (Exception e)
    {
        return -1; // заглушка
    }
    superblock.User_count--;
    return Inode.Update_Inode_After_Delete_User(uid,
    superblock, bitmap);
}

public int Exist(char[] input_login)
{

```

```

    byte[] bytes = new byte[35];
    char[] check_login = new char[16];

    Array.Copy(input_login, 0, check_login, 0,
    input_login.Length < 16 ? input_login.Length : 16);
    try
    {
        using (BinaryReader reader = new
        BinaryReader(File.Open(fs_file, FileMode.Open)))
        {
            reader.BaseStream.Seek(OFFSET,
            SeekOrigin.Begin);
            for (int i = 0; i < 255; i++)
            {
                bytes = reader.ReadBytes(35);
                if (bytes[0] == 1)
                {
                    if (Encoding.Default.GetChars(bytes, 3,
                    16).SequenceEqual(check_login))
                    {
                        return bytes[1]; // Пользователь с таким
                        логином есть в системе
                    }
                }
            }
        }
    }
    catch (Exception e)
    {
        return -2; // заглушка
    }
    return -1; // Такого пользователя не существует
}

using File_system.File_System;
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows;
using System.Windows.Controls;
using System.Windows.Data;
using System.Windows.Documents;
using System.Windows.Input;
using System.Windows.Media;
using System.Windows.Media.Imaging;
using System.Windows.Shapes;

namespace File_system_coursework
{
    public partial class Add_Window : Window
    {
        File_System fs;
        Dictionary<string, byte> groups;

        public Add_Window(File_System fs)
        {
            InitializeComponent();

            this.fs = fs;
            groups = fs.Get_All_Groups();

            CB_Group.ItemsSource = groups.Keys;

```

```

    }

    private void Btn_Add_User_Click(object sender,
RoutedEventArgs e)
    {
        if (String.IsNullOrEmpty(TB_User_Login.Text) ||
String.IsNullOrEmpty(PB_User_Password.Password))
        {
            MessageBox.Show("Заполните все поля
пользователя!", "Ошибка", MessageBoxButton.OK,
MessageBoxImage.Error);
            return;
        }
        byte gid;
        if (CB_Group.SelectedItem == null)
        {
            gid = 0;
        }
        else
        {
            gid = groups[(string)CB_Group.SelectedItem];
            if (gid == 1 && !fs.Is_Admin())
            {
                MessageBox.Show("Добавлять пользователей в
группу Admins может только администратор!", "Ошибка",
MessageBoxButton.OK, MessageBoxImage.Error);
                return;
            }
        }
        switch (fs.Add_User(gid,
TB_User_Login.Text.ToCharArray(),
PB_User_Password.Password.ToCharArray()))
        {
            case -1:
            {
                MessageBox.Show("Ошибка создания!",
"Ошибка", MessageBoxButton.OK, MessageBoxImage.Error);
                return;
            }

            case 0:
            {
                MessageBox.Show("Пользователь добавлен",
"Успех", MessageBoxButton.OK,
MessageBoxImage.Information);
                return;
            }

            case 1:
            {
                MessageBox.Show("Добавление нового
пользователя невозможно! Список пользователей
заполнен.", "Ошибка", MessageBoxButton.OK,
MessageBoxImage.Error);
                return;
            }

            case 2:
            {
                MessageBox.Show("Такой логин уже занят!",
"Ошибка", MessageBoxButton.OK, MessageBoxImage.Error);
                return;
            }
        }
    }
}

```

```

    private void Btn_Delete_User_Click(object sender,
RoutedEventArgs e)
    {
        if (String.IsNullOrEmpty(TB_User_Login.Text) ||
String.IsNullOrEmpty(PB_User_Password.Password))
        {
            MessageBox.Show("Заполните все поля
пользователя!", "Ошибка", MessageBoxButton.OK,
MessageBoxImage.Error);
            return;
        }

        switch
(fs.Delete_User(TB_User_Login.Text.ToCharArray(),
PB_User_Password.Password.ToCharArray()))
        {
            case -1:
            {
                MessageBox.Show("Ошибка удаления!",
"Ошибка", MessageBoxButton.OK, MessageBoxImage.Error);
                return;
            }

            case 1:
            {
                MessageBox.Show("Такого пользователя не
существует!", "Ошибка", MessageBoxButton.OK,
MessageBoxImage.Error);
                return;
            }

            case 2:
            {
                MessageBox.Show("Введён неправильный
пароль!", "Ошибка", MessageBoxButton.OK,
MessageBoxImage.Error);
                return;
            }

            case 0:
            {
                MessageBox.Show("Пользователь удалён",
"Успех", MessageBoxButton.OK,
MessageBoxImage.Information);
                groups = fs.Get_All_Groups();
                CB_Group.ItemsSource = null;
                CB_Group.ItemsSource = groups.Keys;
                break;
            }
        }
    }

    private void Btn_Add_Group_Click(object sender,
RoutedEventArgs e)
    {
        if (String.IsNullOrEmpty(TB_Group_Name.Text) ||
String.IsNullOrEmpty(TB_Group_Description.Text))
        {
            MessageBox.Show("Заполните все поля группы!",
"Ошибка", MessageBoxButton.OK, MessageBoxImage.Error);
            return;
        }

        switch
(fs.Add_Group(TB_Group_Name.Text.ToCharArray(),
TB_Group_Description.Text.ToCharArray()))
        {
            case -1:

```

```

        {
            MessageBox.Show("Ошибка создания!",
"Ошибка", MessageBoxButton.OK, MessageBoxImage.Error);
            return;
        }

        case 0:
        {
            MessageBox.Show("Группа добавлена",
"Успех", MessageBoxButton.OK,
MessageBoxImage.Information);
            groups = fs.Get_All_Groups();
            CB_Group.ItemsSource = null;
            CB_Group.ItemsSource = groups.Keys;
            return;
        }

        case 1:
        {
            MessageBox.Show("Добавление новой группы
невозможно! Список групп заполнен.", "Ошибка",
MessageBoxButton.OK, MessageBoxImage.Error);
            return;
        }

        case 2:
        {
            MessageBox.Show("Такая группа уже
существует!", "Ошибка", MessageBoxButton.OK,
MessageBoxImage.Error);
            return;
        }
    }

    private void Btn_Delete_Group_Click(object sender,
RoutedEventArgs e)
    {
        if (CB_Group.SelectedItem == null)
        {
            MessageBox.Show("Выберите группу!", "Ошибка",
MessageBoxButton.OK, MessageBoxImage.Error);
            return;
        }
        switch
(fs.Delete_Group(((string)CB_Group.SelectedItem).ToCharArra
y()))
        {
            case -1:
            {
                MessageBox.Show("Ошибка удаления!",
"Ошибка", MessageBoxButton.OK, MessageBoxImage.Error);
                return;
            }
            case 0:
            {
                MessageBox.Show("Группа удалена", "Успех",
MessageBoxButton.OK, MessageBoxImage.Information);
                groups = fs.Get_All_Groups();
                CB_Group.ItemsSource = null;
                CB_Group.ItemsSource = groups.Keys;
                break;
            }
            case 1:
            {

```

```

            MessageBox.Show("Такой группы не
существует!", "Ошибка", MessageBoxButton.OK,
MessageBoxImage.Error);
            return;
        }
        case 2:
        {
            MessageBox.Show("Группу Admins удалить
нельзя!", "Ошибка", MessageBoxButton.OK,
MessageBoxImage.Error);
            return;
        }
    }

    private void Btn_Change_User_Click(object sender,
RoutedEventArgs e)
    {
        if (String.IsNullOrEmpty(TB_User_Login.Text) ||
String.IsNullOrEmpty(PB_User_Password.Password))
        {
            MessageBox.Show("Заполните все поля
пользователя!", "Ошибка", MessageBoxButton.OK,
MessageBoxImage.Error);
            return;
        }

        switch
(fs.Change_User(TB_User_Login.Text.ToCharArray(),
PB_User_Password.Password.ToCharArray()))
        {
            case -1:
            {
                MessageBox.Show("Ошибка!", "Ошибка",
MessageBoxButton.OK, MessageBoxImage.Error);
                return;
            }
            case 1:
            {
                MessageBox.Show("Такого пользователя не
существует!", "Ошибка", MessageBoxButton.OK,
MessageBoxImage.Error);
                return;
            }
            case 2:
            {
                MessageBox.Show("Введён неправильный
пароль!", "Ошибка", MessageBoxButton.OK,
MessageBoxImage.Error);
                return;
            }
            case 0:
            {
                MessageBox.Show("Пользователь изменён",
"Успех", MessageBoxButton.OK,
MessageBoxImage.Information);
                break;
            }
        }
    }

    private void Btn_Change_Group_Click(object sender,
RoutedEventArgs e)
    {
        if (String.IsNullOrEmpty(TB_Group_Name.Text))

```



```

        {
            MessageBox.Show("Введите имя группы!",
"Ошибка", MessageBoxButton.OK, MessageBoxImage.Error);
            return;
        }
        switch
(fs.Change_Group(((string)CB_Group.SelectedItem).ToCharArray()))
        {
            case -1:
            {
                MessageBox.Show("Ошибка!", "Ошибка",
MessageBoxButton.OK, MessageBoxImage.Error);
                return;
            }
            case 1:
            {
                MessageBox.Show("Такой группы не
существует!", "Ошибка", MessageBoxButton.OK,
MessageBoxImage.Error);
                return;
            }
            case 0:
            {
                MessageBox.Show("Группа изменена", "Успех",
MessageBoxButton.OK, MessageBoxImage.Information);
                break;
            }
        }
    }
}

using File_system.File_System;
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows;
using System.Windows.Controls;
using System.Windows.Data;
using System.Windows.Documents;
using System.Windows.Input;
using System.Windows.Media;
using System.Windows.Media.Imaging;
using System.Windows.Shapes;

namespace File_system_coursework
{
    public partial class File_Window : Window
    {
        File_System fs;
        uint inode_id;
        public File_Window(uint inode_id, char[] name,
File_System fs, bool can_write)
        {
            InitializeComponent();
            this.fs = fs;
            this.inode_id = inode_id;
            Title = new string(name);
            TextBox_File.Text = new
string(fs.Get_Data_File(inode_id));
            TextBox_File.IsReadOnly = !can_write;
        }
    }
}

```

```

        private void Button_Save_Click(object sender,
RoutedEventArgs e)
        {
            int res = fs.Set_Data_File(inode_id,
Encoding.Default.GetChars(Encoding.Default.GetBytes(TextBox_File.Text)));
            switch (res)
            {
                case 0:
                {
                    MessageBox.Show("Данные сохранены!",
"Успех", MessageBoxButton.OK,
MessageBoxImage.Information);
                    return;
                }
                case 1:
                {
                    MessageBox.Show("Места в файле
недостаточно!", "Не успех", MessageBoxButton.OK,
MessageBoxImage.Information);
                    return;
                }
                case -1:
                {
                    MessageBox.Show("Во время записи
произошла ошибка!", "Ошибка", MessageBoxButton.OK,
MessageBoxImage.Error);
                    return;
                }
            }
        }
    }
}

using File_system.File_System;
using System.Windows;
using System.Windows.Controls;
using System.Collections.Generic;
using System;
using System.Text;
using static File_system.File_System.File_System;
using File_system;
using System.Linq;

namespace File_system_coursework
{
    public partial class MainWindow : Window
    {
        File_System fs;
        string path_file_image = @"Resources\file_image.png";
        string path_catalog_image =
@"Resources\catalog_image.jpg";

        Stack<uint> path_stack;
        Data_Copy buffer;

        public MainWindow(File_System fs)
        {
            InitializeComponent();
            this.fs = fs;
            path_stack = new Stack<uint>();

            path_stack.Push(1);

            Show_Data_Catalog(1); // читаем root
        }
    }
}

```

```

Update_Path_TextBlock();

MessageBox.Show($"Приветствую тебя, брат!",
"Здарова", MessageBoxButton.OK,
MessageBoxImage.Information);
}

private void Update_Path_TextBlock()
{
    StringBuilder path = new StringBuilder();
    uint[] path_array = path_stack.ToArray();

    path.Append(fs.Read_Catalog(path_array[path_array.Length -
1])[0].name);
    for (int i = 1; i < path_array.Length; i++)
    {
        path.Append(" > ");
    }

    path.Append(fs.Read_Catalog(path_array[path_array.Length -
1 - i])[0].name);
    }
    TextBlock_Current_Path.Text = path.ToString();
}

private void Move_To_Catalog(uint catalog_id)
{
    path_stack.Push(catalog_id);
    Show_Data_Catalog(catalog_id);
    Update_Path_TextBlock();
}

private void Show_Data_Catalog(uint catalog_id)
{
    WrapPanel_Work_Field.Children.Clear();
    WrapPanel_Work_Field.Children.Add(new StackPanel()
{ MinHeight = 100 });
    Data_Catalog[] data = fs.Read_Catalog(catalog_id);
    for (int i = 2; i < data.Length; i++)
    {
        bool[] flags = Get_Flags(data[i].permissions);
        if (!MenuItem_Show_Hidden.IsChecked && flags[10])
        {
            continue;
        }
        var sp = new StackPanel();
        var image = new Image();
        var tb = new TextBlock();
        if (Inode.Is_Catalog(data[i].inode_id, fs.superblock))
        {
            image.Source = new
System.Windows.Media.Imaging.BitmapImage(new
Uri(path_catalog_image, UriKind.Relative));
            image.Height = 100;
            tb.TextAlignment = TextAlignment.Center;
            tb.Style =
(Style)this.Resources["Text_Style_File_Name"];
            tb.Foreground =
System.Windows.Media.Brushes.Firebrick;
            tb.Text = new string(data[i].name);
        }
        else
        {

```

```

            image.Source = new
System.Windows.Media.Imaging.BitmapImage(new
System.Uri(path_file_image, System.UriKind.Relative));
            image.Height = 100;
            tb.TextAlignment = TextAlignment.Center;
            tb.Style =
(Style)this.Resources["Text_Style_File_Name"];
            tb.Text = new string(data[i].name);
        }
        if (flags[10]) // Файл скрытый, режим просмотра
скрытых файлов включен
        {
            sp.ContextMenu =
(ContextMenu)this.Resources["Context_Menu_Hidden_Item"];
        }
        else
        {
            sp.ContextMenu =
(ContextMenu)this.Resources["Context_Menu_Visible_Item"];
        }

        sp.Children.Add(image);
        sp.Children.Add(tb);
        sp.MouseLeftButtonDown +=
Work_Field_Double_Click;

        WrapPanel_Work_Field.Children.Add(sp);
    }
}

private void Create_File(char[] name = null, bool
is_catalog = false)
{
    if ((path_stack.Peek() != 1) &&
!fs.Check_Rights_Access(path_stack.Peek())[1] &&
!fs.Is_Admin())
    {
        MessageBox.Show("У вас нет права на создание
файлов в этом каталоге!", "Ограниченные права",
MessageBoxButton.OK, MessageBoxImage.Warning);
        return;
    }
    if (name == null)
    {
        if (is_catalog)
        {
            name =
Encoding.Default.GetChars(Encoding.Default.GetBytes("Новая
папка"));
        }
        else
        {
            name =
Encoding.Default.GetChars(Encoding.Default.GetBytes("Новы
й файл"));
        }
    }
    char[] check_name = new char[28];
    Array.Copy(name, 0, check_name, 0, name.Length < 28
? name.Length : 28);
    byte suffix = 1;
    while (fs.Create_File(check_name, is_catalog,
path_stack.Peek()) == 0)

```

```

    {
        check_name[name.Length < 28 ? name.Length : 28] =
suffix.ToString()[0];
        suffix++;
    }

    var sp = new StackPanel();
    var image = new Image();
    var tb = new TextBlock();
    if (is_catalog)
    {
        image.Source = new
System.Windows.Media.Imaging.BitmapImage(new
System.Uri(path_catalog_image, System.UriKind.Relative));
        image.Height = 100;
        tb.TextAlignment = TextAlignment.Center;
        tb.Style =
(Style)this.Resources["Text_Style_File_Name"];
        tb.Foreground =
System.Windows.Media.Brushes.Firebrick;
        tb.Text = new string(check_name);
        sp.ContextMenu =
(ContextMenu)this.Resources["Context_Menu_Visible_Item"];
    }
    else
    {
        image.Source = new
System.Windows.Media.Imaging.BitmapImage(new
System.Uri(path_file_image, System.UriKind.Relative));
        image.Height = 100;
        tb.TextAlignment = TextAlignment.Center;
        tb.Style =
(Style)this.Resources["Text_Style_File_Name"];
        tb.Text = new string(check_name);
        sp.ContextMenu =
(ContextMenu)this.Resources["Context_Menu_Visible_Item"];
    }

    sp.Children.Add(image);
    sp.Children.Add(tb);
    sp.MouseLeftButtonDown +=
Work_Field_Double_Click;

    WrapPanel_Work_Field.Children.Add(sp);
}

private void Open_File(char[] name)
{
    uint inode_id = 0;
    Data_Catalog[] data =
fs.Read_Catalog(path_stack.Peek());
    for (int i = 2; i < data.Length; i++)
    {
        if (data[i].name.SequenceEqual(name))
        {
            inode_id = data[i].inode_id;
            break;
        }
    }
    if (inode_id != 0)
    {
        if (!fs.Check_Rights_Access(inode_id)[0] &&
!fs.Is_Admin())
        {

```

```

            MessageBox.Show("У вас нет права на чтение
файла!", "Ограниченные права", MessageBoxButton.OK,
MessageBoxImage.Warning);
            return;
        }
        if (!fs.Is_Catalog(inode_id, fs.superblock))
        {
            Move_To_Catalog(inode_id);
        }
        else
        {
            bool can_write = false;
            if (fs.Check_Rights_Access(inode_id)[1] &&
fs.Is_Admin())
            {
                can_write = true;
            }
            File_Window fw = new File_Window(inode_id,
name, fs, can_write);
            fw.Show();
            if (!can_write)
            {
                MessageBox.Show("Только для чтения!",
"Ограниченные права", MessageBoxButton.OK,
MessageBoxImage.Warning);
            }
        }
    }
}

```

```

private void Delete_File(char[] name, uint catalog_id)
{
    uint inode_id = 0;
    Data_Catalog[] data = fs.Read_Catalog(catalog_id);
    byte[] bytes = new byte[(data.Length - 1) * 32];
    Array.Copy(BitConverter.GetBytes(data[0].inode_id), 0,
bytes, 0, 4);
    Array.Copy(Encoding.Default.GetBytes(data[0].name),
0, bytes, 4, 28);
    Array.Copy(BitConverter.GetBytes(data[1].inode_id), 0,
bytes, 32, 4);
    Array.Copy(Encoding.Default.GetBytes(data[1].name),
0, bytes, 36, 28);
    int writes_count = 2; // информация о текущем
каталоге и о каталоге-родителе уже записана
    for (int i = 2; i < data.Length; i++)
    {
        if (data[i].name.SequenceEqual(name))
        {
            inode_id = data[i].inode_id;
        }
        else
        {
            Array.Copy(BitConverter.GetBytes(data[i].inode_id), 0, bytes,
writes_count * 32, 4);//////////
            Array.Copy(Encoding.Default.GetBytes(data[i].name), 0, bytes,
writes_count * 32 + 4, 28);//// была проверка на длину
            writes_count++;
        }
    }
    if (inode_id != 0)
    {

```

```

        if (!fs.Check_Rights_Access(inode_id)[1] &&
!fs.Is_Admin())
        {
            MessageBox.Show("У вас нет права на
изменение файла!", "Ограниченные права",
MessageBoxButton.OK, MessageBoxImage.Warning);
            return;
        }
        if (!fs.Is_Catalog(inode_id, fs.superblock))
        {
            Data_Catalog[] data_rec =
fs.Read_Catalog(inode_id);
            for (int i = 2; i < data_rec.Length; i++)
            {
                Delete_File(data_rec[i].name, inode_id);
            }
            fs.Delete_File(inode_id);
            fs.Write_Catalog(bytes, catalog_id);
            Show_Data_Catalog(catalog_id);
        }
    }

    private void Properties_File(char[] name)
    {
        uint inode_id = 0;
        Data_Catalog[] data =
fs.Read_Catalog(path_stack.Peek());
        for (int i = 2; i < data.Length; i++)
        {
            if (data[i].name.SequenceEqual(name))
            {
                inode_id = data[i].inode_id;
                break;
            }
        }
        if (inode_id != 0)
        {
            MessageBox.Show(fs.Get_Properties_File(inode_id,
fs.Is_Catalog(inode_id, fs.superblock)), "Свойства: " + new
string(name).Trim("\0"), MessageBoxButton.OK,
MessageBoxImage.Information);
        }
    }

    private void Work_Field_Double_Click(object sender,
System.Windows.Input.MouseButtonEventArgs e)
    {
        if (e.ClickCount >= 2)
        {
            StackPanel sp = (StackPanel)sender;

            Open_File(((TextBlock)sp.Children[1]).Text.ToCharArray());
        }
    }

    private void Menu_Create_Catalog_Click(object sender,
RoutedEventArgs e)
    {
        Create_File(is_catalog: true);
    }

    private void Menu_Create_File_Click(object sender,
RoutedEventArgs e)
    {

```

```

        Create_File();
    }

    private void Menu_Copy_Click(object sender,
RoutedEventArgs e)
    {
        StackPanel parent =
(StackPanel)((ContextMenu)((MenuItem)sender).Parent).Place
mentTarget;
        char[] name =
((TextBlock)parent.Children[1]).Text.ToCharArray();
        uint inode_id = 0;
        Data_Catalog[] data =
fs.Read_Catalog(path_stack.Peek());
        for (int i = 2; i < data.Length; i++)
        {
            if (data[i].name.SequenceEqual(name))
            {
                inode_id = data[i].inode_id;
                break;
            }
        }
        if (!fs.Check_Rights_Access(inode_id)[0] &&
!fs.Is_Admin())
        {
            MessageBox.Show("У вас нет права на копирование
этого файла!", "Ограниченные права",
MessageBoxButton.OK, MessageBoxImage.Warning);
            return;
        }
        buffer = fs.Copy(inode_id, name);
        MessageBox.Show("Скопировано в буфер", "Успех",
MessageBoxButton.OK, MessageBoxImage.Information);
    }

    private void Menu_Insert_Click(object sender,
RoutedEventArgs e)
    {
        if (buffer.Equals(default(Data_Copy)))
        {
            MessageBox.Show("Буфер копирования пуст!",
"Ошибка", MessageBoxButton.OK, MessageBoxImage.Error);
            return;
        }
        fs.Insert(buffer, path_stack.Peek());
        Show_Data_Catalog(path_stack.Peek());
    }

    private void Menu_Open_Click(object sender,
RoutedEventArgs e)
    {
        StackPanel parent =
(StackPanel)((ContextMenu)((MenuItem)sender).Parent).Place
mentTarget;

        Open_File(((TextBlock)parent.Children[1]).Text.ToCharArray());
    }

    private void Menu_Delete_Click(object sender,
RoutedEventArgs e)
    {
        StackPanel parent =
(StackPanel)((ContextMenu)((MenuItem)sender).Parent).Place
mentTarget;

```

```

Delete_File(((TextBlock)parent.Children[1]).Text.ToCharArray()
, path_stack.Peek());
}

private void Menu_Properties_Click(object sender,
RoutedEventArgs e)
{
    StackPanel parent =
(StackPanel)((ContextMenu)((MenuItem)sender).Parent).Place
mentTarget;

Properties_File(((TextBlock)parent.Children[1]).Text.ToCharArr
ay());
}

private void Menu_Change_Rights_Click(object sender,
RoutedEventArgs e)
{
    StackPanel parent =
(StackPanel)((ContextMenu)((MenuItem)sender).Parent).Place
mentTarget;
    char[] name =
((TextBlock)parent.Children[1]).Text.ToCharArray();
    uint inode_id = 0;
    Data_Catalog[] data =
fs.Read_Catalog(path_stack.Peek());
    for (int i = 2; i < data.Length; i++)
    {
        if (data[i].name.SequenceEqual(name))
        {
            inode_id = data[i].inode_id;
            break;
        }
    }
    if (inode_id != 0)
    {
        if (!fs.Is_Owner(inode_id) && !fs.Is_Admin())
        {
            MessageBox.Show("У вас нет права на
изменение файла!", "Ограниченные права",
MessageBoxButton.OK, MessageBoxImage.Warning);
            return;
        }
        Rights_Window rw = new Rights_Window(fs,
inode_id, name);
        rw.ShowDialog();
    }
}

private void Menu_Rename_Click(object sender,
RoutedEventArgs e)
{
    StackPanel parent =
(StackPanel)((ContextMenu)((MenuItem)sender).Parent).Place
mentTarget;
    char[] name =
((TextBlock)parent.Children[1]).Text.ToCharArray();
    uint inode_id = 0;
    Data_Catalog[] data =
fs.Read_Catalog(path_stack.Peek());
    for (int i = 2; i < data.Length; i++)
    {
        if (data[i].name.SequenceEqual(name))
        {

```

```

            inode_id = data[i].inode_id;
            break;
        }
    }
    if (inode_id != 0)
    {
        if (!fs.Check_Rights_Access(inode_id)[1] &&
!fs.Is_Admin())
        {
            MessageBox.Show("У вас нет права на
изменение файла!", "Ограниченные права",
MessageBoxButton.OK, MessageBoxImage.Warning);
            return;
        }
        Rename_Window rw = new Rename_Window(name,
path_stack.Peek(), fs);
        rw.ShowDialog();
        Show_Data_Catalog(path_stack.Peek());
    }
}

private void Menu_Hidden_Click(object sender,
RoutedEventArgs e)
{
    StackPanel parent =
(StackPanel)((ContextMenu)((MenuItem)sender).Parent).Place
mentTarget;
    char[] name =
((TextBlock)parent.Children[1]).Text.ToCharArray();
    uint inode_id = 0;
    ushort permissions = 0;

    Data_Catalog[] data =
fs.Read_Catalog(path_stack.Peek());
    for (int i = 2; i < data.Length; i++)
    {
        if (data[i].name.SequenceEqual(name))
        {
            inode_id = data[i].inode_id;
            permissions = data[i].permissions;
            break;
        }
    }
    if (inode_id != 0)
    {
        if (!fs.Check_Rights_Access(inode_id)[1] &&
!fs.Is_Admin())
        {
            MessageBox.Show("У вас нет права на
изменение файла!", "Ограниченные права",
MessageBoxButton.OK, MessageBoxImage.Warning);
            return;
        }
        ushort mask = 1;
        mask = (ushort)(mask << 5);
        permissions |= mask; //

        fs.Set_Flags(permissions, inode_id);
        Show_Data_Catalog(path_stack.Peek());
    }
}

private void Menu_Visible_Click(object sender,
RoutedEventArgs e)
{

```

```

StackPanel parent =
(StackPanel)((ContextMenu)((MenuItem)sender).Parent).Place
mentTarget;
char[] name =
((TextBlock)parent.Children[1]).Text.ToCharArray();
uint inode_id = 0;
ushort permissions = 0;

Data_Catalog[] data =
fs.Read_Catalog(path_stack.Peek());
for (int i = 2; i < data.Length; i++)
{
    if (data[i].name.SequenceEqual(name))
    {
        inode_id = data[i].inode_id;
        permissions = data[i].permissions;
        break;
    }
}
if (inode_id != 0)
{
    if (!fs.Check_Rights_Access(inode_id)[1] &&
!fs.Is_Admin())
    {
        MessageBox.Show("У вас нет права на
изменение файла!", "Ограниченные права",
MessageBoxButton.OK, MessageBoxImage.Warning);
        return;
    }
    ushort mask = 1;
    mask = (ushort)(mask << 5);
    permissions &= (ushort)~mask; //

    fs.Set_Flags(permissions, inode_id);
    Show_Data_Catalog(path_stack.Peek());
}

private void Button_Back_Click(object sender,
RoutedEventArgs e)
{
    if(path_stack.Peek() != 1)
    {
        path_stack.Pop();
        Show_Data_Catalog(path_stack.Peek());
        Update_Path_TextBlock();
    }
}

private void MenuItem_Show_Hidden_Click(object
sender, RoutedEventArgs e)
{
    Show_Data_Catalog(path_stack.Peek());
}

private void MenuItem_Users_Groups_Click(object
sender, RoutedEventArgs e)
{
    var aw = new Add_Window(fs);
    aw.ShowDialog();
}

private void MenuItem_Exit_Click(object sender,
RoutedEventArgs e)
{

```

```

    MessageBox.Show("Не болей, бувай!", "До
свидания", MessageBoxButton.OK,
MessageBoxImage.Information);
    this.Close();
}

private void MenuItem_Scheduler_Click(object sender,
RoutedEventArgs e)
{
    Scheduler_Window sw = new Scheduler_Window();
    sw.Show();
}
}

using File_system.File_System;
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows;
using System.Windows.Controls;
using System.Windows.Data;
using System.Windows.Documents;
using System.Windows.Input;
using System.Windows.Media;
using System.Windows.Media.Imaging;
using System.Windows.Shapes;

namespace File_system_coursework
{
    public partial class Rename_Window : Window
    {
        File_System fs;
        uint inode_id;
        char[] old_name;

        public Rename_Window(char[] old_name, uint inode_id,
File_System fs)
        {
            InitializeComponent();
            Title = "Переименование: " + old_name;
            TB_Rename.Text = new string(old_name);
            this.inode_id = inode_id;
            this.fs = fs;
            this.old_name = old_name;
        }

        private void Btn_Save_Click(object sender,
RoutedEventArgs e)
        {
            switch (fs.Rename(old_name,
TB_Rename.Text.ToCharArray(), inode_id))
            {
                case 0:
                {
                    MessageBox.Show("Файл переименован",
"Успех", MessageBoxButton.OK,
MessageBoxImage.Information);
                    break;
                }

                case -1:
                {

```

```

        MessageBox.Show("Такое имя уже занято!",
"Ошибка", MessageBoxButton.OK, MessageBoxImage.Error);
        break;
    }

    case -2:
    {
        MessageBox.Show("Имя не может быть
пустым!", "Ошибка", MessageBoxButton.OK,
MessageBoxImage.Error);
        break;
    }
}

private void Btn_Cancel_Click(object sender,
RoutedEventArgs e)
{
    this.Close();
}
}

using File_system.File_System;
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows;
using System.Windows.Controls;
using System.Windows.Data;
using System.Windows.Documents;
using System.Windows.Input;
using System.Windows.Media;
using System.Windows.Media.Imaging;
using System.Windows.Shapes;

namespace File_system_coursework
{
    public partial class Rights_Window : Window
    {
        File_System fs;
        uint inode_id;

        public Rights_Window(File_System fs, uint inode_id,
char[] name)
        {
            InitializeComponent();
            this.fs = fs;
            this.inode_id = inode_id;
            Title = "Изменение прав доступа: " + new
string(name);
        }

        private void Btn_Save_Click(object sender,
RoutedEventArgs e)
        {
            ushort rights = 0;
            if (CB_User_R.IsChecked == true)
            {
                rights |= 1 << 14; // Оставляем первый бит под
флаг для проверки на каталог
            }
            if (CB_User_W.IsChecked == true)
            {

```

```

                rights |= 1 << 13;
            }
            if (CB_User_X.IsChecked == true)
            {
                rights |= 1 << 12;
            }
            if (CB_Group_R.IsChecked == true)
            {
                rights |= 1 << 11;
            }
            if (CB_Group_W.IsChecked == true)
            {
                rights |= 1 << 10;
            }
            if (CB_Group_X.IsChecked == true)
            {
                rights |= 1 << 9;
            }
            if (CB_Other_R.IsChecked == true)
            {
                rights |= 1 << 8;
            }
            if (CB_Other_W.IsChecked == true)
            {
                rights |= 1 << 7;
            }
            if (CB_Other_X.IsChecked == true)
            {
                rights |= 1 << 6;
            }
            fs.Set_Flags(rights, inode_id);
            MessageBox.Show("Права установлены", "Успех",
MessageBoxButton.OK, MessageBoxImage.Information);
        }

        private void Btn_Cancel_Click(object sender,
RoutedEventArgs e)
        {
            this.Close();
        }
    }
}

using System;
using System.Collections.Generic;
using System.Collections.ObjectModel;
using System.Data;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows;
using System.Windows.Controls;
using System.Windows.Data;
using System.Windows.Documents;
using System.Windows.Input;
using System.Windows.Media;
using System.Windows.Media.Imaging;
using System.Windows.Shapes;

namespace File_system_coursework
{
    public partial class Scheduler_Window : Window
    {
        DataTable DT_work;
        int seq_proc;
        int seq_time;
    }
}

```

```

int quant;
Process last_proc;

bool is_time_little;

ObservableCollection<Process> collection_proc = null;

public Scheduler_Window()
{
    InitializeComponent();

    int[] prior = Enumerable.Range(0, 40).ToArray();
    CB_Priority.ItemsSource = prior;

    int[] quant = Enumerable.Range(1, 20).ToArray();
    CB_Quant.ItemsSource = quant;

    seq_proc = 0;
    seq_time = 0;
    this.quant = 1;
    is_time_little = false;
    last_proc = null;

    DT_work = new DataTable();
    DT_work.Columns.Add("Процессы\время");
    DG_Work.ItemsSource = DT_work.DefaultView;
}

private void Btn_Add_Click(object sender,
RoutedEventArgs e)
{
    if ((CB_Priority.SelectedItem == null) ||
String.IsNullOrEmpty(TB_Time.Text) ||
(CB_State.SelectedItem == null))
    {
        MessageBox.Show("Введите все данные о
процессе!", "Ошибка", MessageBoxButton.OK,
MessageBoxImage.Warning);
        return;
    }

    if (collection_proc == null)
    {
        collection_proc = new
ObservableCollection<Process>();
        DG_Process.ItemsSource = collection_proc;
    }
    seq_proc++;
    collection_proc.Add(new Process(seq_proc,
Convert.ToByte(CB_Priority.Text), seq_time,
Convert.ToInt32(TB_Time.Text),
(State)CB_State.SelectedIndex));

    DataRow row = DT_work.NewRow();
    row[0] = collection_proc[collection_proc.Count -
1].Num;
    DT_work.Rows.Add(row);
}

private void Btn_Step_Click(object sender,
RoutedEventArgs e)
{
    seq_time++;
    DT_work.Columns.Add(seq_time.ToString());

```

```

last_proc = Scheduling(last_proc);

for (int i = 0; i < DT_work.Rows.Count; i++)
{
    DT_work.Rows[i][seq_time] =
collection_proc[i].Current_state;
}

DG_Process.Items.Refresh();
DG_Work.ItemsSource = null;
DG_Work.ItemsSource = DT_work.DefaultView;
}

public Process Scheduling(Process last)
{
    bool is_quants = false;
    Process prc = null;
    if (last != null)
    {
        if (last.Left_time > 0)
        {
            if (last.Priority <= 3)
            {
                last.Left_time -= 1;
                return last;
            }
            if (last.Quant > 0)
            {
                last.Left_time -= 1;
                last.Quant -= 1;
                return last;
            }
            last.Current_state = State.R;
        }
        else
        {
            last.Quant = 0;
            last.Current_state = State.Z;
            is_time_little = !is_time_little;
        }
    }
    int s;
    for (s = 0; s < collection_proc.Count; s++)
    {
        if (collection_proc[s].Quant > 0)
        {
            is_quants = true;
        }
        if ((collection_proc[s].Current_state == State.R) &&
((collection_proc[s].Quant > 0) || (collection_proc[s].Priority
<= 3)))
        {
            prc = collection_proc[s];
            break;
        }
    }
    for (int i = s+1; i < collection_proc.Count; i++)
    {
        if (collection_proc[i].Quant > 0)
        {
            is_quants = true;
        }
    }

```



```

        if ((collection_proc[i].Current_state != State.R) ||
            ((collection_proc[i].Priority > 3) && (collection_proc[i].Quant
            <= 0)))
        {
            continue;
        }
        if (prc.Priority > collection_proc[i].Priority)
        {
            prc = collection_proc[i];
        }
        else if (prc.Priority == collection_proc[i].Priority)
        {
            if (prc.Left_time > collection_proc[i].Left_time)
            {
                prc = collection_proc[i];
            }
        }
    }
    if (prc != null)
    {
        prc.Current_state = State.P;
        if (prc.Priority <= 3) // Алгоритм для относительных
приоритетов
        {
            prc.Left_time -= 1;
            return prc;
        }
    }
    if (!is_quants)
    {
        for (int i = 0; i < collection_proc.Count; i++)
        {
            if ((collection_proc[i].Current_state == State.R) ||
                (collection_proc[i].Current_state == State.P))
            {
                collection_proc[i].Quant = quant;
            }
        }
    }
    if (prc == null)
    {
        return null;
    }

    prc.Quant -= 1;
    prc.Left_time -= 1;

    if (is_time_little)
    {
        Process ltl = null;
        for (int i = 0; i < collection_proc.Count; i++)
        {
            if (collection_proc[i].Current_state != State.R)
            {
                continue;
            }
            if (prc.Work_time > collection_proc[i].Work_time)
            {
                ltl = collection_proc[i];
            }
        }
        if (ltl != null)
        {
            MessageBox.Show($"Приоритет процесса
№{ltl.Num} был изменён с {ltl.Priority} на 4", "Маленькие

```

```

вперед!", MessageBoxButtons.OK,
MessageBoxImage.Information);
        ltl.Priority = 4;
        is_time_little = !is_time_little;
    }
}
return prc;
}

private void Btn_Set_Quant_Click(object sender,
RoutedEventArgs e)
{
    quant = CB_Quant.SelectedIndex + 1;
    MessageBox.Show("Квант изменён", "Успех",
    MessageBoxButtons.OK, MessageBoxImage.Information);
}

public enum State
{
    /// <summary>
    /// Z - zombie, R - runnable, P - performed, S - sleep
    /// </summary>
    Z, R, P, S,
}

public class Process
{
    public int Num { set; get; }
    public byte Priority { set; get; }
    public int Entry_time { set; get; }
    public int Work_time { set; get; }
    public int Left_time { set; get; }
    public State Current_state { set; get; }
    public int Quant { set; get; }

    public Process(int num, byte prior, int entry_t, int work_t,
    State state)
    {
        Num = num;
        Priority = prior;
        Entry_time = entry_t;
        Work_time = work_t;
        Left_time = Work_time;
        Current_state = state;
        Quant = 0;
    }
}

using System;
using System.Collections.Generic;
using System.IO;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows;
using System.Windows.Controls;
using System.Windows.Data;
using System.Windows.Documents;
using System.Windows.Input;
using System.Windows.Media;
using System.Windows.Media.Imaging;
using System.Windows.Shapes;

```

```

using File_system.File_System;
using Microsoft.Win32;

namespace File_system_coursework
{
    public partial class Start_Window : Window
    {
        string fs_name;

        public Start_Window()
        {
            InitializeComponent();
            Grid_Start.Visibility = Visibility.Visible;
            Grid_Create_Input_FS.Visibility = Visibility.Hidden;

            ///////////
            Scheduler_Window sw = new Scheduler_Window();
            sw.Show();
        }

        private void Button_Create_FS_Click(object sender,
            RoutedEventArgs e)
        {
            Grid_Start.Visibility = Visibility.Collapsed;
            Grid_Create_Input_FS.Visibility = Visibility.Visible;
        }

        private void Button_Choose_FS_Click(object sender,
            RoutedEventArgs e)
        {
            OpenFileDialog open_FD = new OpenFileDialog();
            open_FD.Filter = "File System (*.fsn)|*.fsn";
            open_FD.InitialDirectory =
            AppDomain.CurrentDomain.BaseDirectory;
            if (open_FD.ShowDialog() == true)
            {
                Grid_Start.Visibility = Visibility.Collapsed;
                Grid_Choose_Input_FS.Visibility = Visibility.Visible;

                string[] split = open_FD.FileName.Split("\");
                fs_name = split[split.Length - 1];
            }
            else
            {
                MessageBox.Show("Невозможно открыть ФС!",
                    "Ошибка", MessageBoxButton.OK, MessageBoxImage.Error);
            }
        }

        private void Button_Create2_FS_Click(object sender,
            RoutedEventArgs e)
        {
            if (String.IsNullOrEmpty(TextBox_Type_FS.Text) ||
                String.IsNullOrEmpty(ComboBox_Cluster_Size.Text) ||
                String.IsNullOrEmpty(TextBox_Size_FS.Text) ||
                String.IsNullOrEmpty(TextBox_Create_Login_User.Text) ||
                String.IsNullOrEmpty>PasswordBox_Create_Password_User.Password))
            {
                MessageBox.Show("Заполните все поля!",
                    "Ошибка", MessageBoxButton.OK, MessageBoxImage.Error);
            }
            else if (Convert.ToInt32(TextBox_Size_FS.Text) < 0)
            {

```

```

                MessageBox.Show("Размер файловой системы не
                    может быть меньше 0 КБ!", "Ошибка",
                    MessageBoxButton.OK, MessageBoxImage.Error);
            }
            else
            {
                SaveFileDialog save_FD = new SaveFileDialog();
                save_FD.Filter = "File System (*.fsn)|*.fsn";
                save_FD.InitialDirectory =
                AppDomain.CurrentDomain.BaseDirectory;
                if (save_FD.ShowDialog() == true)
                {
                    if (File.Exists(save_FD.FileName))
                    {
                        File.Delete(save_FD.FileName);
                    }
                    File_System fs = new
                    File_System(save_FD.FileName);
                    string[] split = save_FD.FileName.Split("\");
                    fs_name = split[split.Length - 1];
                    fs.Create_File_System(fs_name.ToCharArray(),
                    TextBox_Type_FS.Text.ToCharArray(),
                    Convert.ToUInt16(ComboBox_Cluster_Size.Text),
                    Convert.ToUInt64(TextBox_Size_FS.Text)*1024,
                    TextBox_Create_Login_User.Text.ToCharArray(),
                    PasswordBox_Create_Password_User.Password.ToCharArray()
                    );

                    MessageBox.Show("Файловая система создана!",
                        "Успех", MessageBoxButton.OK,
                        MessageBoxImage.Information);

                    MainWindow mw = new MainWindow(fs);
                    this.Close();
                    mw.Show();
                }
                else
                {
                    MessageBox.Show("Невозможно создать ФС!",
                        "Ошибка", MessageBoxButton.OK, MessageBoxImage.Error);
                }
            }
        }

        private void Button_Choose2_FS_Click(object sender,
            RoutedEventArgs e)
        {
            File_System fs = new File_System(fs_name);
            try
            {
                fs.Load_File_System(TextBox_Choose_Login_User.Text.ToCha
                    rArray(),
                    PasswordBox_Choose_Password_User.Password.ToCharArray()
                    );

                MainWindow mw = new MainWindow(fs);
                this.Close();
                mw.Show();
            }
            catch (Exception exc)
            {
                MessageBox.Show(exc.Message, "Ошибка",
                    MessageBoxButton.OK, MessageBoxImage.Error);
            }
        }
    }
}

```