

МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ ДНР
ДОНЕЦКИЙ НАЦИОНАЛЬНЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ
КАФЕДРА ПРОГРАММНОЙ ИНЖЕНЕРИИ

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА
к курсовому проекту по дисциплине
«Операционные системы»

Руководитель:

_____ кафедры

Выполнил:

ст. гр. ПИ–186

Моргунов А.Г.

РЕФЕРАТ

Пояснительная записка к курсовому проекту содержит: 66 страниц, 42 рисунок, 17 таблиц, 3 источника, 4 приложения.

Цель курсового проектирования: выполнить обоснованный проект операционной системы, удовлетворяющей ряду специальных требований, приведенных в техническом задании (приложение А), выполнить эмуляцию работы файловой системы и межпроцессного взаимодействия.

Для реализации цели курсового проекта необходимо: выявить требования к проектируемой системе и сравнить ее с уже существующими, выбрать средства реализации, подходящие под требования будущей системы, спроектировать собственную операционную систему, программно реализовать эмуляторы файловой системы и средства межпроцессного взаимодействия, а именно семафоры и разделяемую память.

Результатом работы является программа эмуляции файловой системы, эмуляция средств межпроцессного взаимодействия на языке C++.

ФАЙЛОВАЯ СИСТЕМА, СУПЕРБЛОК, ФАЙЛ, FAT, ПРОЦЕСС,
ЭМУЛЯТОР, ОПЕРАЦИОННАЯ СИСТЕМА, МЕЖПРОЦЕССНОЕ
ВЗАИМОДЕЙСТВИЕ

СОДЕРЖАНИЕ

Реферат.....	2
Введение	5
1 Структура проектируемой файловой системы.....	6
1.1 Общая организация файловой системы	6
1.1.1 Структура суперблока	7
1.1.2 Структура FAT.....	8
1.1.3 Структура корневого каталога.....	9
1.1.4 Структура информации пользователей	11
1.2 Виртуальная память	12
1.3 Команды для работы с файловой системой	13
1.4 Системные вызовы модуля файловой системы	14
1.5 Способы организации файлов	17
1.6 Алгоритм работы некоторых системных вызовов.....	17
2 Процессы в операционной системе	19
2.1 Команды для работы с процессами	19
2.2 Системные вызовы для управления процессами.....	19
2.3 Диаграмма состояний процесса.....	20
2.4 Приоритеты процессов.....	21
2.5 Межпроцессное взаимодействие	22
2.6 Выбор дисциплины обслуживания планировщика процессов. Алгоритм работы планировщика процессов в соответствии с выбранной дисциплиной обслуживания	26
3 Режимы работы проектируемой ОС	29

3.1 Мультипрограммный режим работы ОС	29
3.2 Многопользовательская защита	29
3.3 Интерактивный режим работы ОС.....	29
3.4 Пакетный режим.....	30
4 Структура операционной системы.....	31
4.1 Общая структура проектируемой ОС.....	31
4.2 Структура ядра проектируемой ОС. Основные функции и назначение файловой подсистемы, подсистемы управления процессами, подсистемы управления устройствами.	32
4.3 Структура управляющих блоков базы данных ОС.....	33
5 Разработка программ эмуляции ОС	39
5.1 Описание программных средств.....	39
5.2 Разработка ФС	39
5.3 Разработка командного интерпретатора	44
5.4 Эмуляция межпроцессного взаимодействия	46
6 Тестирование программы. Анализ результатов	50
Выводы	52
Перечень ссылок.....	53
Приложение А. Техническое задание	54
Приложение Б. Таблица трассировки	55
Приложение В. Экранные формы, отображающие результаты работы программ эмуляции.....	56
Приложение Г. Листинг кода	58

ВВЕДЕНИЕ

Операционная система – это комплекс взаимосвязанных управляющих модулей, которые помогают осуществлять взаимодействие между пользователями, программами и аппаратными составляющими компьютера. Помимо этого, операционная система управляет ресурсами компьютера, а также выделяет их процессам, тем самым осуществляя управление процессами. Также операционная система реализует взаимодействие программ между собой, производит запись/чтение информации с внешних носителей.

При выполнении курсового проекта были проанализированы различные реализации файловых систем и выбрана наиболее эффективная для технического задания реализация; спроектирована и описана структура авторской операционной системы; произведено комбинирование средств межпроцессного взаимодействия, что позволило создать средство взаимодействия процессов, обладающее преимуществами как разделяемой памяти, так и семафоров.

1 СТРУКТУРА ПРОЕКТИРУЕМОЙ ФАЙЛОВОЙ СИСТЕМЫ

Файловая система – это порядок, определяющий способ организации, хранения и именования данных на носителях информации в компьютерах, а также в другом электронном оборудовании: цифровых фотоаппаратах, мобильных телефонах и т. п. Файловая система определяет формат содержимого и способ физического хранения информации, которую принято группировать в виде файлов. Конкретная файловая система определяет размер имен файлов (и каталогов), максимальный возможный размер файла и раздела, набор атрибутов файла.

1.1 Общая организация файловой системы

По техническому заданию проектируемая ФС должна обладать следующими особенностями:

- одноуровневая;
- иметь списки свободных/занятых кластеров;
- поддерживать файлы с последовательным доступом.

Логическая структура диска, формируемая файловой системой, представлена в таблицу 1.1.

Таблица 1.1 – Логическая структура диска

Служебные области		Область данных		
Суперблок	FAT	Информация пользователей	Данные	Корневой каталог

Для того, чтобы определить начало области данных на диске, необходимо посчитать с какого байта начинается область данных. Т.к. размер FAT высчитывается автоматически при форматировании для дисков с разными объемами байтовый отступ будет разным.

Например, в системе, в которой в FAT содержится 10000 элементов, суперблок размером 24 байт, на каждую запись FAT выделено 4 байта, а размер кластера – 4096 байт, размер служебной области будет составлять $24 + (4 * 10000) = 40024$ байт. То есть служебная область будет занимать $40024/4096 = 9,77$ кластеров (10 полных кластеров). Область данных начинается с 11 кластера, т.е. с $4096*11+1 = 45057$ байта. Полный размер такого диска составляет $10000 * 4096 = 40960000$ байт = 39,06 Мбайт

1.1.1 Структура суперблока

В суперблоке помещается информация, определяющая тип ФС, и информация, необходимая для хранения параметров, от которых зависит работа файловой системы.

Структура суперблока представлена в таблице 1.2.

Таблица 1.2 – Структура суперблока

Поле	Размер
Тип файловой системы	8 байт
Количество блоков данных	4 байта
Количество свободных блоков	4 байта
Номер первого кластера области данных	4 байта
Размер одного блока данных	4 байта
Всего	24 байта

Тип файловой системы – поле хранящее числовой идентификатор системы.

Размер одного блока данных – число, задающее размер кластера в байтах.

Количество блоков данных – количество элементов FAT.

Количество свободных блоков – количество блоков, в которые можно записать данные в текущий момент.

Номер первого кластера области данных – число, обозначающее с какого кластера начинается область данных.

1.1.2 Структура FAT

FAT – таблица расположения файлов. В этой таблице содержится информация о расположении файла на диске (в кластерах)

В проектируемой системе FAT – это массив, длина которого определяется в суперблоке (в планах сделать динамическое определение количества кластеров), он содержит в себе ссылку на следующий кластер, или признак того, что элемент свободен или испорчен для всех кластеров системы.

Этот массив имеет по 4 байта для каждого элемента. В 4х байтах можно записать числа до 2147483647. Количество элементов в FAT – это число блоков (заданного в суперблоке размера) на диске. Длина FAT в байтах = количество блоков на диске * 4.

Таким образом максимальный размер диска, который может быть описан подобной FAT с размером блока 4096 байт (4К) составляет:

$$4096 * 2147483647 = 8796093018112 \text{ байт} = 8191 \text{ Гбайт} = 7 \text{ Тбайт}$$

В массиве могут быть такие значения как:

- Положительное число – номер кластера, содержащего следующий пакет данных
- -1(1111 1111 1111 1111) – признак того, что это последний кластер для этого файла (признак конца файла)
- 0(0000 0000 0000 0000) – признак того, что блок не занят (признак свободного блока)
- -3 (1111 1111 1111 1101) – признак того, что блок испорчен
- -4 признак системных областей
- -5 используется системой для работы с FAT

В проектируемой системе FAT описывает служебные области и область данных. При форматировании высчитывается размер служебной области и количество кластеров, которых она будет занимать. В суперблоке записывается номер кластера, который является началом области данных.

1.1.3 Структура корневого каталога

В проектируемой системе корневой каталог содержит только файлы, т.к. система является одноуровневой. Размер корневого каталога неограничен, т.к. он расположен в области данных и может занимать несколько кластеров. Каталогные записи в этом каталоге имеют следующую структуру (таблица 1.3)

Таблица 1.3 – Структура каталоговой записи

Поле	Размер
Размер записи каталога	2 байта
Номер начального блока в FAT	4 байта
Атрибуты	2 байта
ID Владельца	1 байт
ID Группы	1 байт
Дата/время создания файла	8 байт
Дата/время последней модификации файла	8 байт
Точный размер файла в байтах	4 байта
Длина имени	1 байт
Имя	X байт
Всего	31 + X байт (дополняется до 4х байтовой границы)

Номер начального блока FAT – номер блока, с которого начинается файл.

Размер записи каталога – количество байт, которое занимает каталоговая запись вместе с именем. Округляется до 4х байтовой границы. (минимальное значение 32, максимальное – $255 \cdot 2 = 510$)

Атрибуты – 2 байта, которые содержат атрибуты и права доступа к файлу (таблица 1.4).

Таблица 1.4 – Структура поля Атрибуты

Бит	Описание
0	Чтение пользователем - владельцем
1	Запись пользователем – владельцем
2	Исполнение пользователем – владельцем
3	Чтение группой
4	Запись группой
5	Исполнение группой
6	Чтение другими пользователями
7	Запись другими пользователями
8	Исполнение другими пользователями
9	Атрибут «только чтение»
10	Атрибут «скрытый»
11	Атрибут «системный»
12	Зарезервированный
13	Зарезервированный
14	Зарезервированный
15	Зарезервированный

Если бит 0 – 8 равен 1, то данная операция разрешена, 0 – запрещена.

Если бит 9 равен 1, то файл предназначен только для чтения, и его содержимое нельзя изменять. Если 0 – то обычный файл.

Если бит 10 равен 1, то файл скрывается от пользователей. Если 0 – то обычный файл.

Если бит 11 равен 1, то файл скрывается от всех пользователей и взаимодействовать с ним может только система. Если 0 – то обычный файл.

Бит 12-15 зарезервированы.

ID Владельца – содержит идентификатор владельца файла.

ID Группы – содержит идентификатор группы, на которую будут влиять права доступа к файлу для группы.

Дата/время создания файла – содержит дату и время создания файла

Дата/время последней модификации файла – содержит дату и время последнего изменения файла или его каталоговой записи.

Точный размер файла в байтах – количество байтов, которое файл занимает на диске.

Длина имени – количество символов в имени файла

Имя – имя файла

1.1.4 Структура информации пользователей

Информация пользователей хранится в системном файле и содержит всю информацию, необходимую для взаимодействия пользователя с системой.

Для обеспечения безопасности в файле хранятся в виде хэша от паролей. Для получения хэша используется алгоритм SHA-256.

SHA-256 представляет собой однонаправленную функцию для создания цифровых отпечатков фиксированной длины (256 бит, 32 байт) из входных данных размером до 2,31 эксабайт (2^{64} бит) и является частным случаем алгоритма из семейства криптографических алгоритмов SHA-2.

Информация пользователей хранится в формате, описанном в таблице 1.5.

Таблица 1.5 – Структура информации пользователей

Поле	Размер
ID Пользователя	1 байт
ID Группы	1 байт
Логин	61 байт
Хэш пароля	65 байта
Всего	128 байта

1.2 Виртуальная память

Виртуальная память – это дополнение к оперативной памяти, которая расширяет ее объем за счет использования жесткого диска. Для пользователя создается иллюзия что ОЗУ имеет бесконечный объем.

В проектируемой системе используется сегментно-страничная организация виртуальной памяти. Она сочетает в себе преимущества страничной и сегментной организации, что позволяет повысить эффективность использования виртуальной памяти. Минусом сегментно-страничной организации является более долгий процесс трансляции виртуального адреса в физический.

При сегментно-страничной организации виртуальной памяти размер сегмента задается кратным размеру страницы. В таком случае сегмент содержит целое количество страниц, даже если одна из них заполнена не полностью.

Виртуальный адрес имеет следующую структуру (рис. 1.1).

Номер сегмента логической памяти	Номер страницы внутри сегмента	Смещение внутри страницы
-------------------------------------	-----------------------------------	-----------------------------

Рисунок 1.1 – Структура виртуального адреса

При такой организации виртуальной памяти трансляция происходит в 2 этапа (рис 1.2). На 1 этапе определяется начальный адрес сегмента, который будет использоваться. На втором этапе определяется номер страницы в текущем сегменте, а затем, при помощи смещения от начала страницы вычисляется физический адрес, на который указывает разбираемый виртуальный адрес.

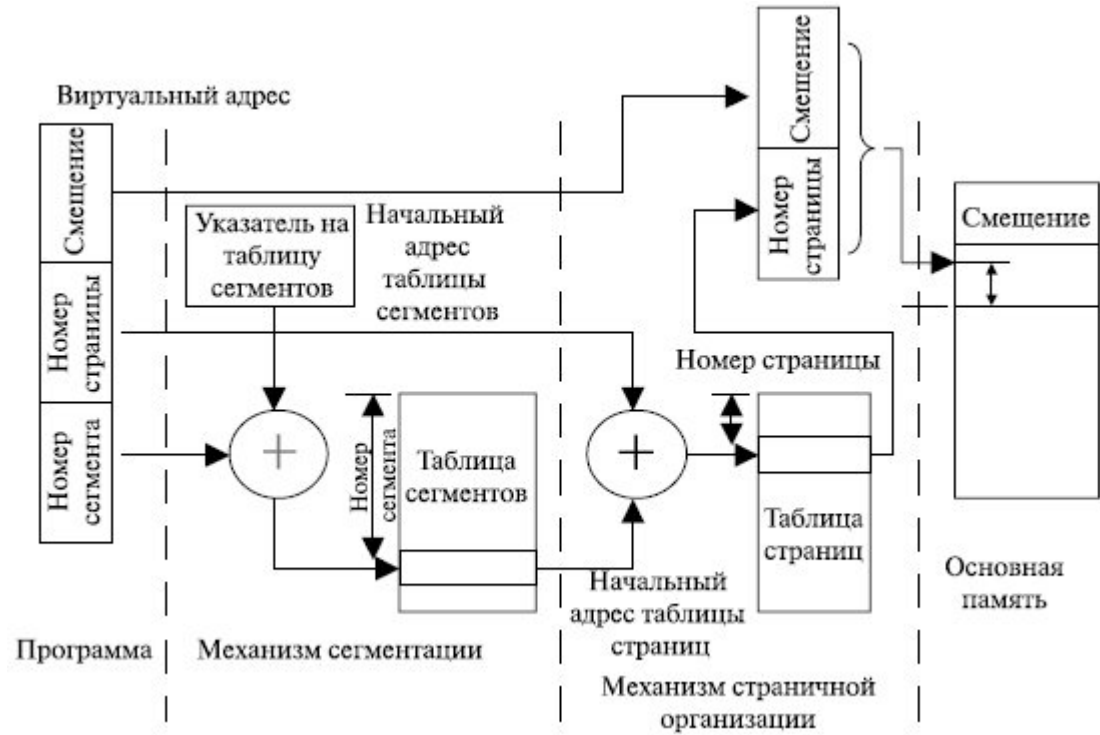


Рисунок 1.2 – Трансляция виртуального адреса в физический

1.3 Команды для работы с файловой системой

По ТЗ система должна производить операции входа и выхода из системы, перемещение (не будет реализовано из-за того, что ФС –

одноуровневая), переименование, создание и удаление. Помимо этих команд также должны быть реализованы команды, позволяющие пользователю комфортно работать в системе.

Таблица 1.6 – Команды для работы с ФС

Имя команды	Параметры	Описание
createfile	fileName	Создание пустого файла с именем fileName
deletefile	fileName	Удаление файла с именем fileName
rename	oldFileName, newFileName	Меняет имя файла oldName на newName
exit		Выход из системы
createuser	userName, password, groupID	Создание нового пользователя с именем userName и паролем password, принадлежащим к группе groupID
deleteuser	userName	Удаляет пользователя с именем userName
changeattributes	fileName, newAttributes	Изменяет атрибуты файла filename на новые атрибуты newAttributes
write	fileName, string	Записывает в конец файла с именем fileName строку string
read		Выводит содержимое файла в консоль
rewrite	fileName, string	Записывает в конец файла с именем fileName строку string

1.4 Системные вызовы модуля файловой системы

Таблица 1.7 – Системные вызовы

Вызов	Параметр	Результат	Описание
write	buffer, bytesCount		Запись в текущий файл, bytesCount байт из буфера buffer. Иницирует ошибку в случае неудачи.
read	buffer, bytesCount		Чтение из текущий файл, bytesCount байт и помещение этих данных в область памяти buffer. Иницирует ошибку в случае неудачи.
deleteFile			Удаляет текущий файл. Иницирует ошибку в случае неудачи.
changeAttributes	newAttributes		Изменение атрибутов текущего файла на новые атрибуты newAttributes. Иницирует ошибку в случае неудачи.
getDirectoryNote		DirectoryNote	Возвращает каталоговую запись текущего файла.
seekGet	countOfBytes, position		Смещение указателя чтения файла.
seekPut	countOfBytes, position		Смещение указателя записи файла.
addDirectoryNote	DirectoryNote		Записывает каталоговую запись в корневой каталог.

Продолжение таблицы 1.7

findDirectoryNote	fileName	DirectoryNote Position	Находит каталоговую запись файла с именем filename, возвращает смещение относительно начала корневого каталога (позицию каталоговой записи).
createNewFile	fileName		Создает файл с именем fileName
deleteData			Удаляет данные, записанные в текущем файле
findFirstFreeBlockNumber		blockNumber	Возвращает номер первого свободного блока в FAT
deleteFATBlockChain	blockNumber		Отчищает последовательность блоков в FAT
getLastBlockOfChain	blockNumber	blockNumber	Возвращает номер последнего блока в FAT цепочке. Принимает один из боков FAT цепочки.
GetFirstByteOfBlock	blockNumber	byteNumber	Возвращает номер байта с которого начитается блок с номером blockNumber.
makeBlockLastInFATBlockChain	blockNumber		делает блок blockNumber последним в FAT цепочке и освобождает все блоки, следовавшие за ним в FAT цепочке

1.5 Способы организации файлов

В системе должны быть реализованы файлы с последовательным доступом. Для реализации таких файлов используется FAT, в которой хранится последовательность блоков, в которых записаны данные файла.

В проектируемой системе реализованы файлы с длинными именами. В каталоговой записи помимо атрибутов и системной информации хранится длина записи, длина имени и само имя. Длина записи считается по 4х битовой границе, и при расчете учитывает статическую часть каталоговой записи, так и часть с именем файла, которое может быть разного размера. Длина имени указывает количество символов в имени. Имя содержит последовательность символов.

При считывании файла система воспринимает его как последовательность байтов, которые считываются последовательно. Считывание файла происходит до конца файла (определяющимся по размеру файла).

При записи система проверяет количество свободных блоков. Если свободных блоков достаточно, то система начинает поиск свободных кластеров и записывает в них данные. Если система наткнется на конец кластера, то она ищет следующий свободный кластер, помечая последовательность кластеров в FAT, пока не запишет данные файла полностью.

1.6 Алгоритм работы некоторых системных вызовов

Алгоритм системного вызова «deleteFATBlockChain», который удаляет FAT цепочку:

```

Вход: номер первого блока цепочки
{
Пока есть следующий блок данных {
    Считать следующий блок данных
    Пометит этот блок данных как свободный
    Перейти к следующему блоку
}
}

```

Рисунок 1.3 – Листинг системного вызова «deleteFATBlockChain»

Системный вызов «deleteFATBlockChain» отчищает полностью цепочку кластеров. Алгоритм системного вызова «changeAttribues», который меняет атрибуты файла.

```

Вход: новые атрибуты
{
    Чтение пользователем = новое значение
    Запись пользователем = новое значение
    Запуск пользователем = новое значение
    Чтение группой = новое значение
    Запись группой = новое значение
    Запуск группой = новое значение
    Чтение другими = новое значение
    Запись другими = новое значение
    Запуск другими = новое значение
}

```

Рисунок 1.4 – Листинг системного вызова «changeAttribues»

Системный вызов «changeAttribues» принимает на вход newAttributes в понятном пользователю формате. Затем преобразует комбинацию атрибутов в формат, который описывает поле атрибутов в каталоговой записи. После этого алгоритм присваивает полю атрибутов новое значение, тем самым меняя атрибуты файла.

2 ПРОЦЕССЫ В ОПЕРАЦИОННОЙ СИСТЕМЕ

2.1 Команды для работы с процессами

Команды, позволяющие работать с процессами описаны в таблице 2.1.

Таблица 2.1 – Команды для работы с процессами

Команда	Параметры	Описание
ps		Выводит информацию о текущих процессах
renice	pid newPrior	Изменяет приоритет процессу pid на newPrior
kill	pid	Уничтожает процесс pid

2.2 Системные вызовы для управления процессами

Для управления процессами предоставляются следующие системные вызовы (см. таблицу 2.2).

Таблица 2.2 – Системные вызовы для управления процессами

Вызов	Параметры	Результат	Описание
fork		pid	Создание нового процесса. Возвращает pid или код ошибки.
exit	signal		Завершение процесса с сигналом signal.
sleep	time	time	Приостановка выполнения процесса на время time. Возвращает оставшееся время сна.
wakeup		err	Пробуждение процесса pid.
kill	pid, signal	err	Завершение работы процесса pid с сигналом signal процессу.
renice	pid, newPri	err	Меняет относительный приоритет процесса pid на newPri.

2.3 Диаграмма состояний процесса

В проектируемой ОС процесс может находиться в нескольких состояниях, переходы между которыми изображены на рисунке (см. рис 2.1).

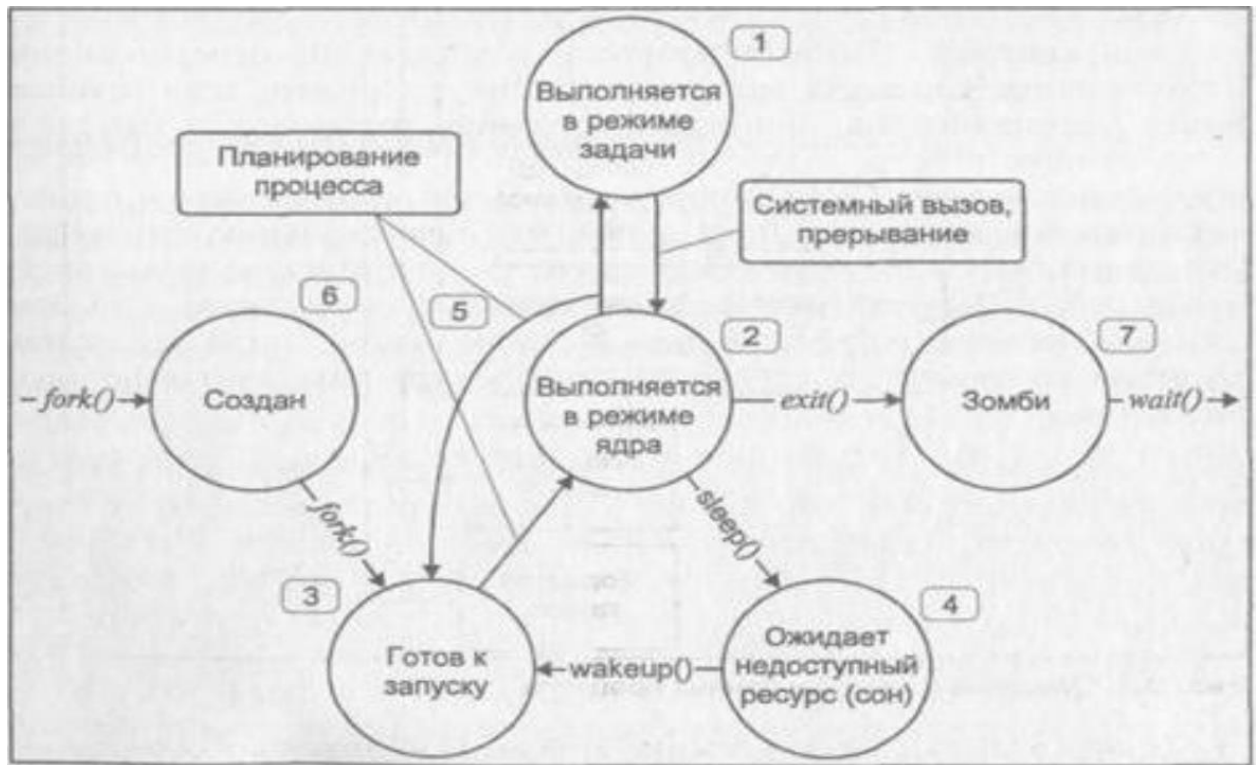


Рисунок 2.1 – Диаграмма состояний процесса

Состояния процесса:

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 (состояние runnable). Процесс находится в очереди на выполнение и обладает всеми необходимыми ему ресурсами, кроме вычислительных.

4. Процесс находится в состоянии сна (sleep), 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 (`zombie`, `defunct`). 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.

2.4 Приоритеты процессов

Проектируемая ОС поддерживает квантование времени, а также все виды приоритетов (Абсолютные, относительные, статические, динамические).

При квантовании времени выделяется определенный интервал времени, называемый квантом, в течении которого процесс может исполняться. По истечении кванта, система выдает новый квант, а кому он достанется зависит от приоритетов. Готовые процессы по очереди получают определенное количество квантов, что позволяет создать иллюзию одновременного исполнения процессов.

Статический приоритет – приоритет который устанавливается пользователем.

Динамический приоритет – приоритет, устанавливаемый и изменяемый системой для повышения эффективности работы.

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

2.5 Межпроцессное взаимодействие

Проектируемая система поддерживает такие средства межпроцессного взаимодействия как семафоры и разделяемая память.

Семафоры – это средства синхронизации множества процессов. Семафоры в UNIX группируются в наборы, каждый из которых содержит один и более семафоров. Семафоры часто используются вместе с разделяемой памятью, реализуя, таким образом, мощный метод межпроцессного взаимодействия.

Функции для работы с семафорами:

1. `semget` – создание нового или получение существующего набора семафоров (рис. 2.2).

Алгоритм `semget`

Входные параметры: ключ семафора, количество семафоров, флаг

```
{  
    Если множество семафоров создано или получен доступ к старому множеству  
    {  
        Вернуть идентификатор множества семафоров  
    }  
    Иначе  
    {  
        Вернуть -1  
    }  
}
```

Рисунок 2.2 – Алгоритм `semget`

Первый аргумент – ключ семафора. Второй аргумент – число семафоров, которое нужно породить. Третий аргумент – флаг, значение которого определяет будет ли команда создавать или искать существующий набор семафоров.

2. `semctl` – управление множеством семафоров (рис. 2.3).

```

Алгоритм semctl
Входные параметры: ключ семафора, номер семафора, команда
{
    Если управление над семафором прошло успешно
    {
        Вернуть натуральное число
    }
    Иначе
    {
        Вернуть -1
    }
}

```

Рисунок 2.3 – Алгоритм semctl

Первый аргумент – ключ семафора. Второй аргумент – номер семафора, над которым производится операция. Третий аргумент – команда.

3. semop – выполнение набора операций над множеством семафоров (рис. 2.4)

```

Алгоритм semop
Входные параметры: ключ семафора, операции над семафором, количество операций
{
    Если все перечисленные операции над семафорами прошли успешно
    {
        Вернуть 0
    }
    Иначе
    {
        Вернуть -1
    }
}

```

Рисунок 2.4 – Алгоритм semop

Первый аргумент – ключ семафора. Второй аргумент – указатель на массив операций. Третий аргумент – количество операций в массиве

Разделяемая память позволяет множеству процессов отображать часть своих виртуальных адресов на общую область памяти. Поток работает с памятью, которая со стороны пользователя является такой-же, как и память в виртуальном пространстве процесса, но при этом несколько процессов работают с одной физической областью памяти.

Функции для работы с разделяемой памятью:

1. `shmget` - создание сегмента разделяемой памяти с привязкой к целочисленному идентификатору (см. рис 2.5).

```

Алгоритм shmget
Входные параметры: ключ разделяемой памяти, размер сегмента, флаг
{
    Если сегмент разделяемой памяти создан или получен доступ к старому сегменту
        Вернуть идентификатор сегмента разделяемой памяти
    }
    Иначе
    {
        Вернуть -1
    }
}

```

Рисунок 2.5 – Алгоритм `shmget`

2. `void* shmat(int desc, void* addr, int flag)`, функция `shmat` подсоединяет область разделяемой памяти, указанную дескриптором `desc`, к виртуальному адресному пространству вызывающего процесса. После выполнения этой функции процесс получает возможность читать и записывать данные в область разделяемой памяти. Параметр `addr` задает начальный виртуальный адрес адресного пространства вызывающего процесса, в которое необходимо отобразить разделяемую память. Если это значение равно нулю, то ядро самостоятельно определит в вызывающем процессе подходящий виртуальный адрес. Функция `shmat` в случае успеха возвращает виртуальный адрес области отображения `int shmdt (void* addr)`

3. Функция `shmdt` отсоединяет область разделяемой памяти от заданного параметром `addr` виртуального адреса вызывающего процесса. Параметр `addr` должен быть получен с помощью функции `shmat`. В случае успешного выполнения функция `shmdt` возвращает 0, а в случае неудачи –1. Тривиальная реализация на уровне псевдокода.

4. `int shmctl (int desc, int cmd, struct shmid_ds* buf)`, функция `shmctl` позволяет запрашивать управляющие параметры разделяемой области памяти, заданной параметром `desc`, изменять информацию в управляющих параметрах области разделяемой памяти, а также удалять область разделяемой памяти. Является аналогом `msgctl`.

Рассмотрим пример, когда сервер отправляет клиенту сообщение, а клиент его получает (см. рис 2.10 – 2.11).

```

1 #include <stdio.h>
2 #include <sys/shm.h>
3 #include <sys/ipc.h>
4 #include <sys/sem.h>
5 #include <sys/types.h>
6
7 union semun{
8     int val;
9     struct shmid_ds *buf;
10    ushort *array;
11 };
12
13 int main()
14 {
15     int shmid;
16     int semidw;
17     int semidr;
18     char *str;
19     semidw = semget((key_t)77,1,IPC_CREAT|0666);
20     semidr = semget((key_t)78,1,IPC_CREAT|0666);
21     struct sembuf ops[2];
22     ops[0].sem_num = 0;
23     ops[0].sem_op = 0;
24     ops[0].sem_flg = 0;
25     ops[1].sem_num = 0;
26     ops[1].sem_op = 1;
27     ops[1].sem_flg = 0;
28     semop(semidw,&ops[0],2);
29     shmid=shmget((key_t)6,1024,IPC_CREAT|0666);
30     //запись в область
31     str=(char *)shmat(shmid,(char *)0,0);
32     printf("Enter data : ");
33     fgets(str,sizeof(str),stdin);
34     printf("Data successfully written into memory \n");
35     shmdt(str);
36     union semun arg;
37     arg.val=0;
38     shmctl(semidr,0,SETVAL,arg);
39     return 0;
40 }
41

```

Рисунок 2.6 – Разделяемая память (сервер)

```

1 #include <stdio.h>
2 #include <sys/shm.h>
3 #include <sys/ipc.h>
4 #include <sys/sem.h>
5 #include <sys/types.h>
6 union semun{
7     int val;
8     struct semid_ds *buf;
9     ushort *array;
10 };
11 int main()
12 {
13     int shmid;
14     int semidw;
15     int semidr;
16     char *str;
17     semidw = semget((key_t)77,1,IPC_CREAT|0666);
18     semidr = semget((key_t)78,1,IPC_CREAT|0666);
19     struct sembuf ops[2];
20     ops[0].sem_num = 0;
21     ops[0].sem_op = 0;
22     ops[0].sem_flg = 0;
23     ops[1].sem_num = 0;
24     ops[1].sem_op = 1;
25     ops[1].sem_flg = 0;
26     semop(semidr,&ops[0],2);
27     shmid=shmget((key_t)6,1024,IPC_CREAT|0666);
28     //чтение из области
29     str=(char *)shmat(shmid,(char *)0,0);
30     printf("Data read from memory : %s \n",str);
31     shmdt(str);
32     union semun arg;
33     arg.val=0;
34     semctl(semidw,0,SETVAL,arg);
35     return 0;
36 }
37 }

```

Рисунок 2.7 – Разделяемая память (клиент)

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

Для проектируемой ОС был разработан алгоритм планировщика, который реализует квантование времени, статические, динамические, относительные и абсолютные приоритеты.

Реализация предполагает наличие двух очередей, в одной хранятся процессы с абсолютными статическими приоритетами, а в другой с относительными динамическими.

В первой очереди применяется алгоритм RR (Round Robin), а во второй выполняется приоритетное планирование. Размеры квантов времени в обеих очередях одинаковые, но для очереди с абсолютными приоритетами выделяется в 2 раза больше квантов чем для очереди с относительными приоритетами.

У процесса может быть приоритет от максимального(0) до минимального (20). Приоритет от 0 до 4 считается абсолютным, а от 5 до 20 относительным.

Процессы с приоритетом от 5 до 20 добавляются в очередь №2 с относительными динамическими приоритетами. В этой очереди выполняется процесс с наибольшим приоритетом. Если процесс не завершается за выделенное ему количество квантов, то его приоритет увеличивается на 1. Если у процесса приоритет 20 и планировщик пытается увеличить его значение на 1, то приоритет процесса становится равным его приоритету при создании. При добавлении нового процесса с относительным приоритетом вытеснения не происходит.

Процессы с приоритетом от 0 до 4 добавляются в очередь №1 с абсолютными статическими приоритетами. В этой очереди все процессы считаются равноправными, а распределение квантов для выполнения происходит по алгоритму RR. Если при выполнении процесса с относительным приоритетом происходит добавление процесса с абсолютным приоритетом, то происходит вытеснение.

Алгоритм краткосрочного планирования

Входная информация:

Выходная информация:

```
{
  Пока (есть процессы ожидающие выполнения)
  {
    Если (в очереди №1 есть процессы)
    {
      Для каждого процесса в очереди №1 выделить по 10 квантов;
    }
    Если (в очереди №2 есть процессы)
    {
      Если (количество квантов для очереди №1 != 0)
      {
        Выделить (количество квантов для очереди №1 / 2) квантов времени
        для очереди №2
      }
      Иначе
      {
        Выделить 50 квантов для очереди №2
      }
      Пока (количество квантов > 0)
      {
        Выбрать процесс с наибольшим приоритетом
        Если (требуемое количество квантов <= количество квантов)
        {
          количество квантов -= требуемое количество квантов
        }
        Иначе
        {
          требуемое количество квантов -= количество квантов
          Если (Приоритет = 20)
          {
            Приоритет = приоритет при создании процесса
          }
          Иначе
          {
            Приоритет++
          }
        }
      }
    }
  }
}
```

Рисунок 2.8 – Алгоритм планировщика

3 РЕЖИМЫ РАБОТЫ ПРОЕКТИРУЕМОЙ ОС

3.1 Мультипрограммный режим работы ОС

Мультипрограммный режим работы – это способ организации работы системы, позволяющий одновременно выполняться нескольким программам. Помимо этого, при мультипрограммном режиме работы ОС должна обеспечить защиту данных, относящихся к различным задачам, от несанкционированного взаимодействия с другими данными или задачами. Также для полноценной работы мультипрограммной системы необходима реализация переключения между процессами и задачами.

Эффективность мультипрограммных систем в основном зависит от алгоритма выделения ресурсов и объема этих ресурсов. [1]

3.2 Многопользовательская защита

Многопользовательская защита - это средство, которое обеспечивает идентификацию пользователей, уровни привилегий для различных пользователей, защиту пользовательской информации от нежелательного доступа. Для начала работы с пользователем пользователь должен авторизоваться. Информация о зарегистрированных пользователях хранится внутри системы.

Создание новых пользователей осуществляет суперпользователь, учетная запись которого создается в системе при форматировании. Также права файла частично зависят от того, какой пользователь взаимодействует с этим файлом. Все пароли хранятся в зашифрованном виде, что позволяет повысить уровень безопасности системы. Однако если пользователь забудет пароль, то восстановить его пользователь не сможет.

3.3 Интерактивный режим работы ОС.

Интерактивный режим – режим работы ОС, при котором основными источниками команд являются пользователи, оперативно взаимодействующие

с компьютером посредством терминалов. В интерактивном режиме необходимо обеспечить приемлемое для пользователя время исполнения команды (как правило, в пределах нескольких секунд). [1]

В проектируемой системе пользователи имеют доступ к файловой системе через командный интерпретатор. Интерпретатор ждет ввода команды, а затем, в зависимости от команды, производит определенные действия с ФС.

3.4 Пакетный режим

Пакетный режим – это режим в котором в системе присутствуют особые пакетные файлы, которые служат источником команд для системы. Такие файлы подготавливаются пользователем, а затем используются системой. В пакетном режиме время выполнения отдельно взятой команды менее существенно чем в интерактивном режиме. Основным критерием в такой системе является максимальная загрузка оборудования.

4 СТРУКТУРА ОПЕРАЦИОННОЙ СИСТЕМЫ

4.1 Общая структура проектируемой ОС

Структура проектируемой ОС представлена на рисунке 4.1.

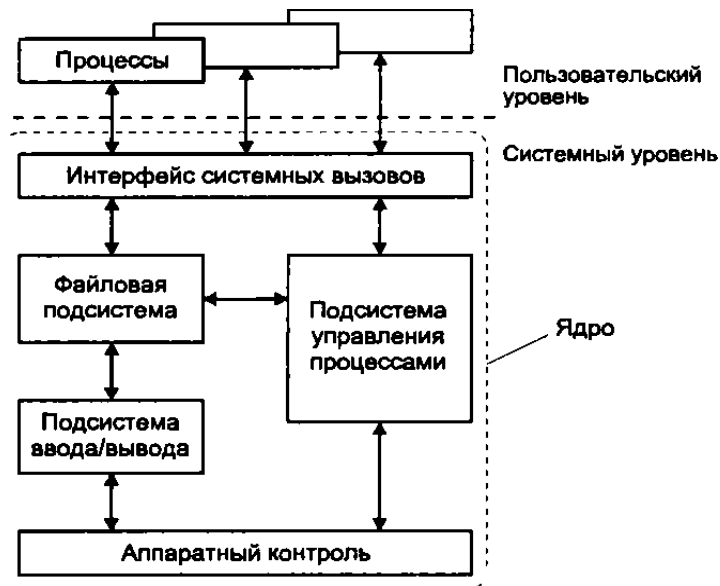


Рисунок 4.1 – Структура проектируемой ОС

На пользовательском уровне все процессы и приложения, независимо от их функций, взаимодействуют с интерфейсом системных вызовов.

Ядро системы включает в себя файловую подсистему, подсистему управления процессами, подсистему ввода/вывода. Ядро связывает пользовательские процессы и аппаратуру, преобразуя процессы в простейшие команды и выполняя их. Благодаря аппаратному контролю ядро, а, следовательно, и система, может посылать сигналы и команды аппаратуре.

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

Ядро ОС состоит из следующих основных подсистем: файловая подсистема, подсистема управления памятью и процессами, подсистема управления устройствами.

Файловая подсистема – это система, которая контролирует базовое взаимодействие с данными, которые расположены на диске. Также она контролирует права доступа к файлам, осуществляет простейшие действия с файлом (удаление, создание, запись, чтение).

Подсистема управления процессами представляет собой совокупность модуля планировщика процессов. Планировщик разрешает конфликты между процессами в конкуренции за системные ресурсы (процессор, память, устройство ввода–вывода). Важнейшей частью операционной системы, непосредственно влияющей на функционирование вычислительной машины, является подсистема управления процессами.

Запущенная на выполнение программа порождает в системе один или более процессов (или задач). Подсистема управления процессами контролирует:

- создание и удаление процессов;
- распределение системных ресурсов (памяти, вычислительных ресурсов) между процессами;
- синхронизацию процессов;
- межпроцессное взаимодействие.

Подсистема ввода/вывода преобразует указания файловой подсистемы в команды для периферийных устройств.

4.3 Структура управляющих блоков базы данных ОС

ОС в основном базируется на таблицах. Природа ОС определяется системными параметрами и константами.

База данных ОС содержит всю информацию, необходимую ей для функционирования. БД состоит из статической части и динамической. Размер статической части не изменяется, а динамическая может изменять свой размер в процессе работы системы. Динамическая часть загружается за пределы системного пула. В БД содержится информация о запущенных процессах, открытых файлах, подключенных пользователях и т.д. [3]

БД должна содержать следующие управляющие блоки:

1. Блок управления пользователями.
2. Блок управления файлами.
3. Блок управления памятью.
4. Блок управления сегментами памяти.
5. Блок управления процессами.
6. Блок управления устройствами.
7. Блок управления сообщениями.

БД должна содержать список очередей процессов:

Каждая очередь базы данных операционной системы представляет собой двусвязный динамический список с указателями на начало и конец очереди.

Каждый процесс имеет свою отдельную очередь сообщений.

Каждый блок управления процессом содержит указатели на начало и конец своей очереди сообщений.

Список всех запущенных в системе процессов, содержит все процессы, которые также содержатся в очереди готовых процессов, очереди заблокированных внешними устройствами процессов или в списке процессов, ожидающих сообщения.

Очередь всех процессов (TQ_IN) Очередь процессов, ожидающих устройство (TQ_DEV_ID) (При загрузке ОС определяет устройства). Каждому устройству присваивается уникальный ID. В зависимости от скорости обслуживания формируется новая очередь процессов, ожидающих доступ к данному устройству. ID устройств сохраняется при выходе из ОС. Если при повторном запуске будет обнаружено новое устройство, будет сделана попытка присвоить ему свой ID и в зависимости от загрузки создать ему очередь).

Очередь к файлу ID (TQ_FDEV_ID). Очередь готовых процессов (TQ_OUT). Очередь к терминалу (TQ_TER). Представляет собой очередь процессов, которые активно взаимодействуют с пользователями.

Операции над очередями:

- взять голову из очереди (ETQ_GET);
- поместить в хвост очереди (ETQ_PUT);
- пересмотреть очередь (ETQ_REF) (данная операция выполняется автоматически – производится сортировка очереди в соответствие с приоритетами после каждого выхода из очереди).

Количество элементов в очереди (ETQ_LNG) переменные и константы:

1. Количество сегментов памяти.
2. Маркер свободен/занят для устройства с номером ID.
3. Маркер свободен/занят для файла с номером ID.
4. Маркер свободен/занят для терминала ID.
5. Количество свободных сегментов памяти.
6. Номер первого свободного сегмента.
7. Максимальное количество запускаемых процессов.
8. Максимальное число открытых файлов.

При входе в систему блок управления пользователями регистрирует пользователя, при выходе пользователя из системы запись о нем удаляется из системы.

При запуске процесса блок управления процессами регистрирует процесс, при завершении процесса, запись о нем удаляется.

При обращении к ОП блок управления памятью регистрирует управление обращение, определяет по ряду признаков возможность доступа и права пользователя, при завершении работы процесса с этим сегментом блок выставляет флаг доступа в 1, разрешая тем самым другому процессу занять этот сегмент.

Блок управления устройствами регистрирует все устройства, их текущее состояние, изменяя его в ходе работы.

Блок управления файлами заносит запись об открытом файле в список, устанавливает признаки, при закрытии файла запись удаляется из списка.

На Рисунке 4.2 представлена общая схема организации БД.

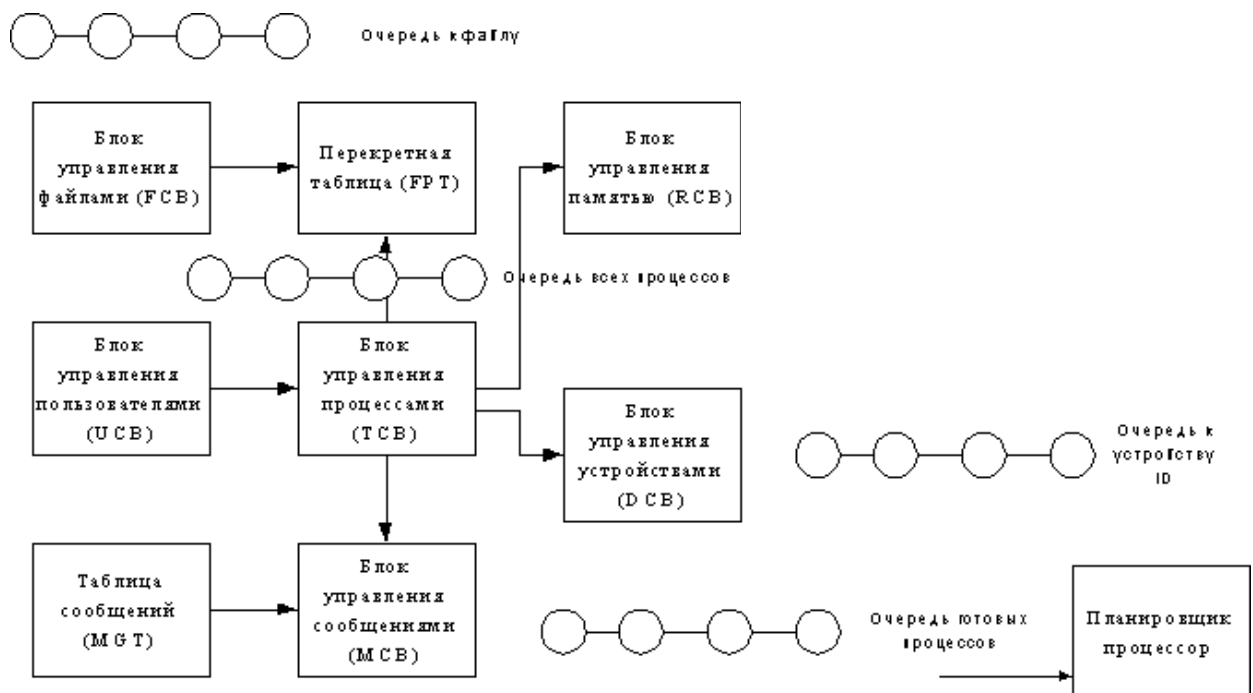


Рисунок 4.2 – Общая схема организации базы данных ОС Блок

Управления пользователями (User Control Block) UCB:

- UCB_ID – ID пользователя;
- UCB_Name – имя пользователя;
- UCB_GROUP – группа пользователя (текущая).

Блок управления файлами File Control Block (FCB): при открытии файла, в БД создается запись для этого файла, при его закрытии, она удаляется. Если с файлом работает несколько процессов одновременно. Блок управления файлами – динамическая часть базы данных, размер этого блока изменяется при изменении системной переменной, определяющей количество открытых файлов.

Структура блока управления файлами:

- FCB_IS – флаг существования файла;
- FCB_CTIME – время создания;
- FCB_MTIME – время изменения;
- FCB_SIZE – размер;
- FCB_FBN – номер первого блока;
- FCB_NSIZE – размер каталоговой записи;
- FCB_ATTR – атрибуты;
- FCB_UID – идентификатор пользователя;
- FCB_GID – идентификатор группы;
- FCB_NLEN – длина имени;
- FCB_NAME – имя файла;

Memory Control Block: информация об оперативной памяти располагается в статической части БД и не является набором структур, в блоке управления памятью хранится информация об общем размере, начальные адреса физических сегментов:

- RCB_HANDLE – дескриптор блока;
- RCB_SIZE – размер блока;
- RCB_CADR – текущий адрес блока;
- RCB_RIGHT – права доступа;

Блок каталога страниц памяти. Page Mapping Table Control Block (PCB).

- PCB_TID - ID таблицы страниц
- PCB_DESCRIPTOR - описание страницы

Блок управления процессами Task Control Block (TCB):

Максимально количество одновременно запущенных процессов определяется системной переменной SYSP_MAXP, которая может быть изменена привилегированным пользователем. Для того, чтобы избежать необходимости перезагрузки системы после изменения значения переменной, дескрипторы процессов располагаются в динамической части БД. В статической части располагается информация о количестве запущенных процессов и адреса таблицы дескрипторов процессов.

- TCB_PID – ID процесса;
- TCB_PPID – идентификатор процесса-родителя;
- TCB_UID – идентификатор пользователя;
- TCB_GUID – идентификатор группы пользователя;
- TCB_PRI – приоритет процесса;
- TCB_NICE – поправка процесса;
- TCB_CPU_BURST – оставшееся время выполнения, в квантах.
- TCB_CONT – контекст процесса (состояние регистров, указатель стека) и объем занимаемой памяти;
- TCB_STATE – состояние процесса;
- TCB_MES – адрес таблицы сообщений процесса;
- TCB_COUNT – количество сообщений процесса.

Блок управления устройствами. Device Control Block (DCB):

В статической части БД хранится только флаг наличия устройства в системе DCB_ST, название устройства DCB_NAME и адрес драйвера в памяти DCB_DRV.

ОС автоматически, через некоторый промежуток времени проверяет статус устройства (DCB_ST). При отсутствии устройства, которое было подключено, драйвер выгружается из памяти. При обнаружении нового устройства, ОС получает данные об устройстве и загружает соответствующий драйвер. При отсутствии необходимого драйвера (например, устройство не

поддерживается системой по умолчанию), привилегированному пользователю предлагается указать местоположение драйвера.

Каждое устройство имеет свой запрос на прерывание и порт ввода вывода. При загрузке драйвера ему сообщается эта информация, и дальнейшая работа с устройством ведется только посредством драйвера. Для незанятых прерываний формируются "заглушки" – пустые устройства, статус которых "отсутствует".

Блок управления устройством содержит следующую информацию:

- DCB_ID – ID устройства;
- DCB_NAME – имя устройства;
- DCB_ST – статус устройства;
- DCB_INT – таблица точек входа;
- DCB_INIT – инициализация;
- DCB_WRITE – флаг записи;
- DCB_READ – флаг чтения;
- DCB_PROP – параметры устройства;
- DCB_DRV – адрес драйвера устройства;
- DCB_NEXT – адрес следующего DCB.

Блок управления сообщениями. Message Control Block (MBC):

Блок управления сообщениями – динамическая часть БД. Производит обработку входящих сообщений ОС. Для каждого процесса создается отдельная таблица входящих сообщений, содержащая ссылку на общую таблицу сообщений (MGT).

Каждый раз, когда возникает системное сообщение, в дескрипторы процессов, подключенных к этому сообщению, добавляется номер сообщения и сопутствующая информация. При получении процессом кванта времени, сначала вызываются обработчики пришедших сообщений. Процесс продолжает выполнение только после этого. Кроме того, процесс может послать сообщение другому процессу.

5 РАЗРАБОТКА ПРОГРАММ ЭМУЛЯЦИИ ОС

5.1 Описание программных средств

Для разработки операционной системы был выбран язык программирования C++, в качестве IDE была выбрана Microsoft Visual Studio 2019.

Для разработки межпроцессного взаимодействия был выбран язык C++, реализация произведена в операционной системе Linux.

5.2 Разработка ФС

Файловая система выполнена при помощи объектно-ориентированного подхода. Основные вызовы для базовой работы с внутренними структурами ФС реализованы в классе FileSystem, а в классе File реализованы команды, которые взаимодействуют с системой на более высоком уровне абстракции.

Для реализации интерпретатора был использован функциональный подход, который позволил упростить код основного блока, а также облегчил сопровождение и изменение кода интерпретатора.

В проекте реализованной ФС имеется 2 основных класса: FileSystem и File. Диаграмма классов приведена на рисунке 5.1.

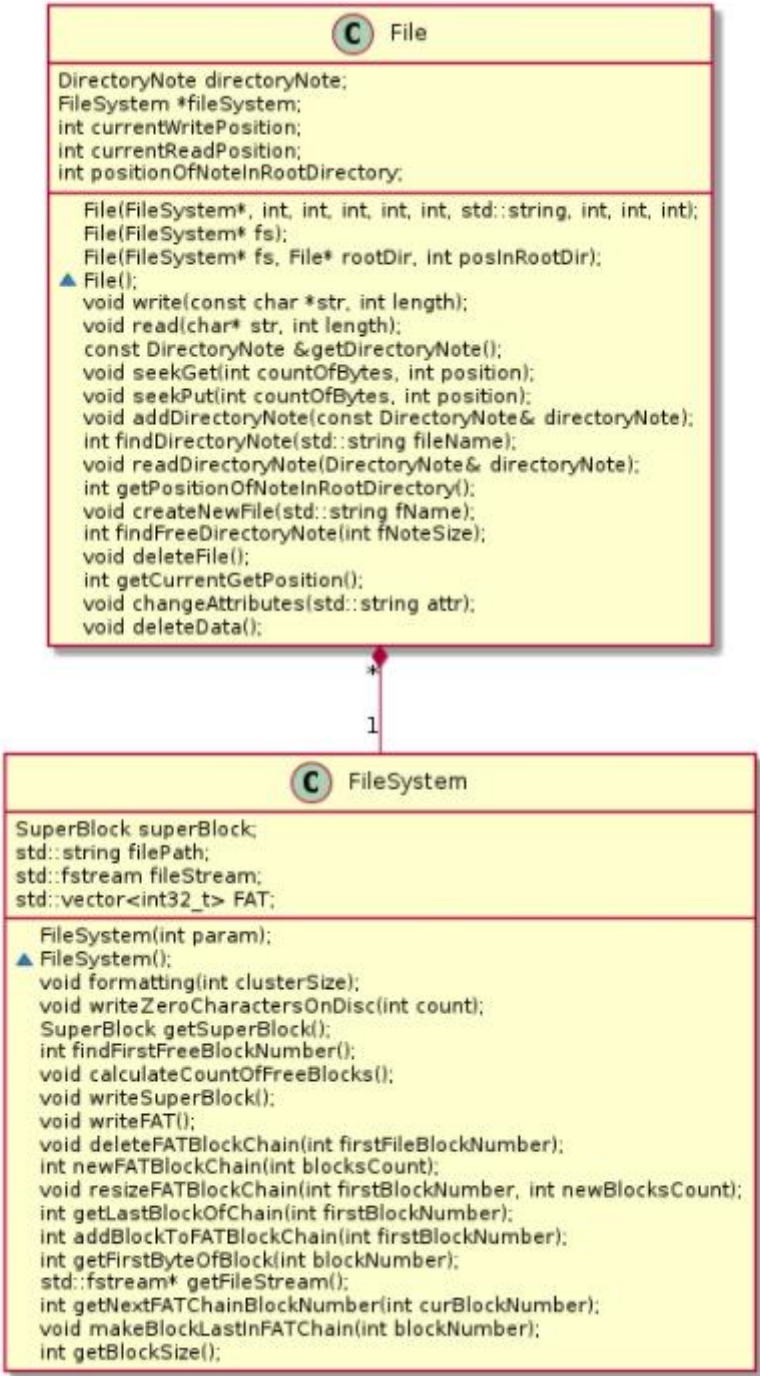


Рисунок 5.1 – Диаграмма классов файловой системы

FileSystem – класс который отвечает за действия связанные с внутреннем строением файловой системы. В функции класса входит работа с FAT и суперблоком, а также управление потоком записи на диск. Методы данного класса описаны в таблице 5.1.

Таблица 5.1 – Методы класса FileSystem

Название метода	Описание
formatting	Функция для форматирования диска
writeZeroCharactersOn Disc	Записывает в файл count нулей (если count больше размера файла, то файл расширяется)
getSuperBlock	Вернуть суперблок (для тестов)
findFirstFreeBlockNumber	Ищет первый свободный блок в FAT
calculateCountOfFreeBlocks	Вычисляет количество свободных блоков на диске по FAT
writeSuperBlock	Запись суперблока на диск
writeFAT	Записывает FAT на диск
deleteFATBlockChain	Отчищает цепочку блоков в FAT
newFATBlockChain	Создает новую цепочку блоков в FAT
resizeFATBlockChain	Изменяет размер цепочки блоков в FAT, которая начинается с blockNumber, на blocksCount блоков
getLastBlockOfChain	Возвращает номер последнего блока в FAT
addBlockToFATBlockChain	Добавляет 1 блок в конец цепочки блоков в FAT и возвращает номер добавленного блока
getFirstByteOfBlock	Возвращает первый байт блока с указанным номером
getFileStream	Возвращает указатель на поток работы с файлом
getNextFATChainBlockNumber	Берет значение указанной ячейки FAT
makeBlockLastInFATChain	Делает указанный блок последним в FAT цепочке (и удаляет оставшиеся за ним записи в FAT цепочке)
getBlockSize	Возвращает размер кластера

Алгоритмы некоторых функций класса FileSystem представлены на рисунках 5.2-5.3.

```
Алгоритм formatting
Входная информация: Размер кластера
Выходная информация:
{
    заполнение структуры суперблока
    выделение 4х байтов для каждого элемента FAT
    Определение номера блока с которого начинается область данных
    Отчистка системной области от данных, которые там хранились
    Создание FAT необходимого размера
    Пометить кластеры, в которых находится системная область
    Расчет количества свободных блоков в системе
    Запись суперблока на диск
    Запись FAT на диск
}
```

Рисунок 5.2 – Алгоритм функции «formatting»

```
Алгоритм deleteFATBlockChain
Входная информация: Первый блок
Выходная информация:
{
    Если (первый блок свободен)
    {
        Вернуться
    }

    Делать
    {
        значение текущего блока = следующему
        текущая FAT запись = 0
        количество свободных блоков в системе++
    } Пока (текущий блок != -1)
}
```

Рисунок 5.3 – Алгоритм функции «deleteFATBlockChain»

Все служебные структуры, используемые системой, описаны в 1 разделе пояснительной записки.

File – класс, который объединяет в себе функции, предназначенные для взаимодействия с файлами: запись, чтение, изменение прав доступа, создание файла, удаление файла. Описание методов класса File представлены в таблице 5.2.

Таблица 5.2 – Описание методов класса File

Метод	Описание
write	Записать в файл
read	Прочитать из файла
getDirectoryNote	Возвращает каталоговую запись файла
seekGet	Смещение указателя чтения
seekPut	Смещение указателя записи
addDirectoryNote	Запись каталоговой записи на диск
findDirectoryNote	Поиск каталоговой записи по имени
readDirectoryNote	Считывание 1 каталоговой записи
getPositionOfNoteInRootDirectory	Возвращает позицию каталоговой записи в корневом каталоге
createNewFile	Создает новый файл
findFreeDirectoryNote	Поиск свободной каталоговой записи, подходящей по размеру
deleteFile	Удаление файла
getCurrentGetPosition	Вернуть текущую позицию указателя чтения
changeAttributes	Изменить атрибуты файла
deleteData	Удалить данные файла
deleteUser	Удалить пользователя

Рассмотрим алгоритм записи в файл (см. рис. 5.4).

```

Алгоритм write
Входная информация: Данные, Размер данных
Выходная информация:
{
    определить расположение указателя записи
    прибавить к текущему расположению указателя размер данных
    изменить размер файла
    сместится к блоку, в который будет производится запись
    делать {
        сместиться к расположению указателя записи
        написать данные полностью или заполнить до конца кластера
        размер данных -= количество записанных символов
        если (данные записаны не полностью) {
            перейти к следующему блоку
            если (текущий блок последний) {
                выделить новый кластер
            }
        }
    } пока (Размер данных != 0)
}

```

Рисунок 5.4 – Алгоритм метода «write»

5.3 Разработка командного интерпретатора

Для взаимодействия с файловой системой разработан командный интерпретатор. Пользователь вводит команды, а интерпретатор сообщает системе, что ей необходимо сделать.

В таблице 1.6 приведены все возможности командного интерпретатора.

На рисунке 5.5 продемонстрирована диаграмма вариантов использования для пользователя и суперпользователя.

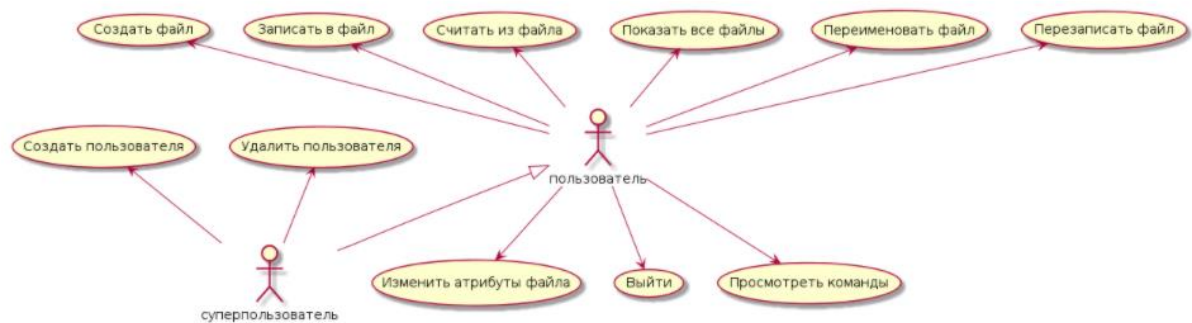


Рисунок 5.5 – Диаграмма вариантов использования для пользователя и суперпользователя

На рисунке 5.6 продемонстрирована работа с командным интерпретатором.

```
C:\WINDOWS\system32\cmd.exe

Flags      File name File size      Creation time  Group ID  Owner ID
rwxrwxrwx  file       7      Thu Dec  3 20:50:24 2020    1        1
rwxrwxrwx  file2      0      Thu Dec  3 20:51:15 2020    2        2
End of file
Your command: help
=====HELP=====
show - show files
createfile <fileName> - create file
deletefile <fileName> - delete file
rename <fileName> <newFileName> - show files
write <fileName> - write in file (to the end)
read <fileName> - read from file
rewrite <fileName> - write in file (delete old data and write new data)
createuser - create new user
changeattributes <fileName> <newAttributes> - change attributes of file. <newAttributes> must have 9 numbers (1 or 0) wh
ich correspond rwxrwxrwx
exit - leave from system
=====HELP=====
Your command: createfile file5
Your command: write file5
Enter your data: aaaaaaaaaa
Your command: read file5
aaaaaaaaa
Your command: show
Flags      File name File size      Creation time  Group ID  Owner ID
rwxrwxrwx  file       7      Thu Dec  3 20:50:24 2020    1        1
rwxrwxrwx  file2      0      Thu Dec  3 20:51:15 2020    2        2
rwxrwxrwx  file5      8      Sat Dec  5 23:15:19 2020    1        1
End of file
Your command:
```

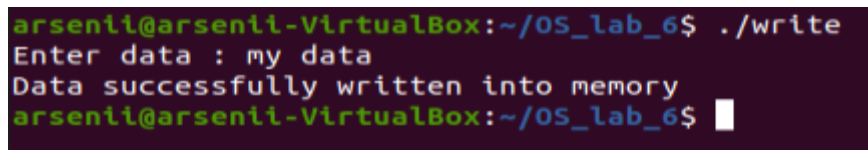
Рисунок 5.6 – Работа с командным интерпретатором

5.4 Эмуляция межпроцессного взаимодействия

Для того, чтобы реализовать межпроцессное взаимодействие в проектируемой системе были скомбинированы семафоры и разделяемая память. Алгоритм предоставляет возможность разным процессам передавать информацию друг другу при помощи разделяемой памяти. Также алгоритм не позволяет произвести запись в разделяемую память, пока предыдущее сообщение не будет считано, а также не позволяет читать из разделяемой памяти, пока не выполнена запись.

Пример работы алгоритма описан далее.

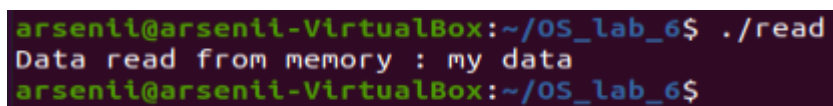
Процесс write записывает в область разделяемой памяти сообщение (рис. 2.1).



```
arsenii@arsenii-VirtualBox:~/OS_lab_6$ ./write
Enter data : my data
Data successfully written into memory
arsenii@arsenii-VirtualBox:~/OS_lab_6$
```

Рисунок 5.7 – Запись в разделяемую память

Процесс read считывает информацию из области разделяемой памяти (рис 2.2).



```
arsenii@arsenii-VirtualBox:~/OS_lab_6$ ./read
Data read from memory : my data
arsenii@arsenii-VirtualBox:~/OS_lab_6$
```

Рисунок 5.8 – Чтение из разделяемой памяти

При попытке записи в область разделяемой памяти, пока предыдущая запись не была считана, семафоры запрещают производить запись (рис. 2.3).

```

arsenii@arsenii-VirtualBox:~/OS_lab_6$ ./write
Enter data : data
Data successfully written into memory
arsenii@arsenii-VirtualBox:~/OS_lab_6$ ./write

```

Рисунок 5.9 – Попытка записи в разделяемую память без чтения

После считывания данных из разделяемой памяти семафоры перестают блокировать процесс записи (рис. 2.4).

```

arsenii@arsenii-VirtualBox:~/OS_lab_6$ ./write
Enter data : data
Data successfully written into memory
arsenii@arsenii-VirtualBox:~/OS_lab_6$ ./write
Enter data : 

```

```

arsenii@arsenii-VirtualBox:~/OS_lab_6$ ./read
Data read from memory : data
arsenii@arsenii-VirtualBox:~/OS_lab_6$ 

```

Рисунок 5.10 – Запись после чтения

При попытке чтения из разделяемой памяти пока не произведена запись данных семафоры приостанавливают выполнение процесса чтения (рис. 2.5).

```
arsenii@arsenii-VirtualBox:~/OS_lab_6$ ./write
Enter data : data
Data successfully written into memory
arsenii@arsenii-VirtualBox:~/OS_lab_6$ ./write
Enter data : 

arsenii@arsenii-VirtualBox:~/OS_lab_6$ ./read
Data read from memory : data

arsenii@arsenii-VirtualBox:~/OS_lab_6$ ./read
```

Рисунок 5.11 – Попытка чтения без записи

После того, как запись была произведена, процесс чтения возобновляется (рис. 2.6).

```
arsenii@arsenii-VirtualBox:~/OS_lab_6$ ./write
Enter data : data
Data successfully written into memory
arsenii@arsenii-VirtualBox:~/OS_lab_6$ ./write
Enter data : data2
Data successfully written into memory
arsenii@arsenii-VirtualBox:~/OS_lab_6$ 

arsenii@arsenii-VirtualBox:~/OS_lab_6$ ./read
Data read from memory : data

arsenii@arsenii-VirtualBox:~/OS_lab_6$ ./read
Data read from memory : data2

arsenii@arsenii-VirtualBox:~/OS_lab_6$
```

Рисунок 5.12 – Чтение после записи


```
arsenii@arsenii-VirtualBox:~/OS_lab_6$ ipcs -m

----- Сегменты совм. исп. памяти -----
ключ  shmid      владелец права  байты natch      состояние      назначение
0x000000000 5          arsenii         600          16384          1             назначение
0x000000000 8          arsenii         600          9338880        2             назначение
0x000000000 12         arsenii         600          524288         2             назначение
0x000000000 32782      arsenii         600          2883584        2             назначение
0x000000000 32785      arsenii         600          4194304        2             назначение
0x000000000 23         arsenii         600          524288         2             назначение
0x000000000 25         arsenii         600          2457600        2             назначение
0x000000006 32794      arsenii         666          1024           0             назначение
0x000000000 27         arsenii         600          4194304        2             назначение
0x000000000 28         arsenii         600          2457600        2             назначение
0x000000000 31         arsenii         600          524288         2             назначение
0x000000000 32815      arsenii         600          2621440        2             назначение
0x000000000 32818      arsenii         600          16384          2             назначение
0x000000000 32819      arsenii         600          16384          2             назначение
```

Рисунок 5.13 – Список сегментов разделяемой памяти

```
arsenii@arsenii-VirtualBox:~/OS_lab_6$ ipcs -s

----- Массивы семафоров -----
ключ  semid      владелец права  nsems
0x00000004d 3          arsenii         666          1
0x00000004e 4          arsenii         666          1
```

Рисунок 5.14 – Список семафоров

6 ТЕСТИРОВАНИЕ ПРОГРАММЫ. АНАЛИЗ РЕЗУЛЬТАТОВ

Были протестированы исключительные случаи при работе с интерпретатором и файловой системой в целом.

```

Your command: createfile
this function takes 1 parameters

```

Рисунок 6.1 – Ввод команды с неверным числом аргументов

```

Your command: show
Flags          File name File size      Creation time  Group ID  Owner ID
rwxrwxrwx      file         0      Sun Dec  6 01:14:55 2020      1        1
End of file
Your command: createfile file
File with this name is already exist

```

Рисунок 6.2 – Попытка создания файла с повторяющимся именем

```

Flags          File name File size      Creation time  Group ID  Owner ID
rwxrwxrwx      file         0      Sun Dec  6 01:14:55 2020      1        1
End of file
Your command: write fole
File with this name isn't exist

```

Рисунок 6.2 – Попытка обращения к несуществующему файлу

```

this function takes 0 parameters
Your command: createuser
Enter username: Root
User with this name is already exist
Your command:

```

Рисунок 6.3 – Попытка создания пользователя с повторяющимся именем

```

      Flags          File name File size      Creation time  Group ID  Owner ID
-----
rwxrwxrwx          file         0      Sun Dec  6 01:14:55 2020        1        1
rwxrwxrwx          file2        0      Sun Dec  6 01:16:41 2020        2        2
End of file
Your command: write file
You haven't write rights

```

Рисунок 6.4 – Попытка записи в файл при отсутствии прав

```

Your command: dance ^
Command is not exist

```

Рисунок 6.5 – Попытка ввода несуществующей команды

Все исключительные ситуации, которые могут возникнуть при работе с межпроцессным взаимодействием были описаны в разделе 5.4.

ВЫВОДЫ

При выполнении курсового проекта было выполнено планирование собственной операционной системы, а также была произведена эмуляция таких модулей операционной системы как файловая система и модуль межпроцессного взаимодействия.

В результате работы была спроектирована одноуровневая файловая система с использованием списка свободных/занятых кластеров (FAT). Выполнена реализация файлов с длинными именами, что позволяет оптимизировать работу с каталоговыми записями, при работе с именами файлов разной длины. Был создан интерпретатор, который позволяет пользователю взаимодействовать с файловой системой.

Также были реализованы такие средства межпроцессного взаимодействия как семафоры и разделяемая память, а точнее их комбинация. Такое взаимодействие различных средств является исключительно эффективным, поскольку разделяемая память не имеет встроенных средств синхронизации.

При продолжении работы над проектом в будущем можно будет получить полноценную операционную систему. Возможными методами улучшения эффективности работы системы является добавление поддержки экстенгов, программная реализация планировщика процессов, объединение всех реализованных программно модулей в один многофункциональный комплекс.

ПЕРЕЧЕНЬ ССЫЛОК

1. Таненбаум Э., Бос Х. T18 Современные операционные системы. 4-е изд. — СПб.: Питер, 2015. — 1120 с.: ил. — (Серия «Классика computer science»).
2. Робачевский А. М. Операционная система UNIX. — СПб.: БХВ-Петербург, 2002. — 528 с.
3. Столлингс В. Операционные системы — М.: «Вильямс», 2002 — 786 с.

ПРИЛОЖЕНИЕ А. ТЕХНИЧЕСКОЕ ЗАДАНИЕ

ПРИЛОЖЕНИЕ Б. ТАБЛИЦА ТРАССИРОВКИ

Рассмотрим работу команды `write`, которая позволяет записывать данные в файл.

1. Вводим команду `write <filename>`.
2. Если файла с таким именем нет, то система выведет соответствующую ошибку.
3. Если пользователь не имеет права записывать в файл и не является суперпользователем, то система выдает соответствующую ошибку.
4. Файл открывается для записи, и у пользователя запрашивается строка, которую он хочет записать в файл.
5. После ввода строки и нажатия `Enter` строка записывается в файл.

ПРИЛОЖЕНИЕ В. ЭКРАННЫЕ ФОРМЫ, ОТОБРАЖАЮЩИЕ РЕЗУЛЬТАТЫ РАБОТЫ ПРОГРАММ ЭМУЛЯЦИИ

```
1. Recreate file system
2. Open file system
3. Exit
Your choise:
```

Рисунок В.1 – Стартовое меню

```
1. Recreate file system
2. Open file system
3. Exit
Your choise: 2
Enter username: Root
Enter password: Root
You entered in system
Your command:
```

Рисунок В.2 – Ввод логина и пароля пользователя

```
Your command: .
```

Рисунок В.2 – Строка взаимодействия с интерпретатором

```
Your command: show
Flags      File name File size      Creation time  Group ID  Owner ID
End of file .
```

Рисунок В.3 – Просмотр всех файлов


```

Your command: createfile file
Your command: write file
Enter your data: my data
Your command: read file
my data

```

Рисунок В.4 – Создание, запись, чтение файла

```

Your command: createuser
Enter username: AS
Enter password: 123
Enter group ID: 4
Your command:

```

Рисунок В.1 – Создание пользователя

```

Your command: help
=====HELP=====
show - show files
createfile <fileName> - create file
deletefile <fileName> - delete file
rename <fileName> <newFileName> - show files
write <fileName> - write in file (to the end)
read <fileName> - read from file
rewrite <fileName> - write in file (delete old data and write new data)
createuser - create new user
changeattributes <fileName> <newAttributes> - change attributes of file. <newAttributes> must have 9 numbers (1 or 0) wh
ich correspond rwxrwxrwx
exit - leave from system
=====HELP=====

```

Рисунок В.1 – Помощь по интерпретатору

```

Your command: changeattributes file 100111111
Your command: rename file file2

```

Рисунок В.1 – Изменение атрибутов и переименование файла

ПРИЛОЖЕНИЕ Г. ЛИСТИНГ КОДА

```

main.cpp
#include<iostream>
#include<string>
#include<fstream>
#include<vector>
#include<filesystem>
#include"SuperBlock.h"
#include"FileSystem.h"
#include"File.h"
#include"User.h"
#include"Functions.h"
#include"SHA256.hpp"
#include "DirectoryNote.h"
void checkFlags(char* flags,DirectoryNote& dirNote) {
    if (!dirNote.ownerRead) flags[0] = '-';
    if (!dirNote.ownerWrite) flags[1] = '-';
    if (!dirNote.ownerUse) flags[2] = '-';
    if (!dirNote.groupRead) flags[3] = '-';
    if (!dirNote.groupWrite) flags[4] = '-';
    if (!dirNote.groupUse) flags[5] = '-';
    if (!dirNote.otherRead) flags[6] = '-';
    if (!dirNote.otherWrite) flags[7] = '-';
    if (!dirNote.otherUse) flags[8] = '-';
}
int main() { using namespace std; while (true) {
    cout << "1. Recreate file system" << endl;
    cout << "2. Open file system" << endl;
    cout << "3. Exit" << endl;
    //cout << "4. Test" << endl;
    cout << "Your choose: ";
    int switchPar;
    cin >> switchPar;
    switch (switchPar) {
        case 1: {
            FileSystem fileSystem(2);
            fileSystem.writeZeroCharactersOnDisc(16000);
            fileSystem.formatting(4096);
            File rootCatalog(&fileSystem,
            fileSystem.getSuperBlock().firstDataBlockNumber, 1, 1, 0,0,
            "RootCatalog",1,0,0);
            rootCatalog.addDirectoryNote(rootCatalog.getDirectoryNote(
            );
            rootCatalog.seekPut(0, BEGIN_POS);
            rootCatalog.addDirectoryNote(rootCatalog.getDirectoryNote(
            );
            File users(&fileSystem,
            fileSystem.findFirstFreeBlockNumber(), 1, 0, 0);
            rootCatalog.findFreeDirectoryNote(40),0, "users", 1, 0, 0);
            // Запись каталоговой записи в корневой каталог
            // Запись, содержащая количество пользователей
            User reserved; reserved.groupID = 1;
            reserved.userID = 1; reserved.userName[0]
            = 'R'; reserved.userName[1] = '\0';
            reserved.password[0] = 'R'; reserved.password[1]
            = '\0'; users.write((char*)&reserved, sizeof(reserved));
            User root; root.groupID = 1;
            root.userID = 1; root.userName[0] = 'R';
            root.userName[1] = 'o'; root.userName[2]
            = 'o'; root.userName[3] = 't';root.userName[4] = '\0';
            strcpy_s(root.password, sha256("Root").c_str());
            users.write((char*)&root, sizeof(root));
            rootCatalog.seekPut(users.getPositionOfNoteInRootDirectory(
            ), BEGIN_POS);
            rootCatalog.addDirectoryNote(users.getDirectoryNote());
            rootCatalog.seekPut(0, BEGIN_POS);
            rootCatalog.addDirectoryNote(rootCatalog.getDirectoryNote(
            ); //case 1 break; break;case 3:return 0; break; case 2: {
            FileSystem fileSystem(1); File
            rootCatalog(&fileSystem); User curUser;
            vector<string> vec; { int fPos = -1; string fName
            = "users"; fPos = rootCatalog.findDirectoryNote(fName);
            File file(&fileSystem, &rootCatalog, fPos);
            file.seekGet(0, BEGIN_POS); User usersCount;
            file.read((char*)&usersCount, sizeof(usersCount));
            string userName, password; cout << "Enter username: ";
            cin.ignore(32767, '\n');getline(cin, userName);cout << "Enter
            password: "; getline(cin, password);for (int i = 0; i <
            usersCount.userID; i++) { User user;file.read((char*)&user,
            sizeof(user)); if (!strcmp(user.userName, userName.c_str()) &
            !strcmp(sha256(password.c_str()).c_str(), user.password)) {curUser
            = user;throw runtime_error("You entered in system");
            } break; } catch (const std::exception& ex){
            cout << ex.what() << endl;} }while (true) { cout <<
            "Your command: "; string str;getline(cin, str); vec = split(str, ' ');
            string command = vec[0]; if (!strcmp(command.c_str(),
            "createfile")) {if (vec.size() != 2) { cout << "this function takes 1
            parameters" << endl; continue;}createFile(rootCatalog, vec[1],
            fileSystem,curUser); }else {if (!strcmp(command.c_str(),
            "deletefile")) {if (vec.size() != 2) { cout << "this function takes 1
            parameters" << endl; continue;}
            deleteFile(rootCatalog, vec[1], fileSystem, curUser);
            } else {
            if (!strcmp(command.c_str(), "write"))
            {
            if (vec.size() != 2) {

```

```

            cout << "this function takes 1
            parameters" << endl;
            continue;
            }
            write(rootCatalog, vec[1],
            fileSystem,curUser);
            else {
            if (!strcmp(command.c_str(),
            "read"))
            {
            if (vec.size() != 2) {
            cout << "this function takes
            1 parameter" << endl;
            continue;
            }
            read(rootCatalog, vec[1],
            fileSystem,curUser);
            else {
            if (!strcmp(command.c_str(),
            "show"))
            {
            if (vec.size() != 1) {
            cout << "this function
            takes 0 parameters" << endl;
            continue;
            }
            show(rootCatalog,
            fileSystem);
            else {
            if (!strcmp(command.c_str(), "createuser")) {
            if (vec.size() != 1) {
            cout << "this function takes 0 parameter" << endl;
            continue;
            }
            try {
            createuser(rootCatalog, fileSystem, curUser);
            } catch (const
            std::exception& ex) {
            cout <<
            ex.what() << endl; }
            } else {
            if (!strcmp(command.c_str(), "rename"))
            {
            if (vec.size() != 3)
            cout << "this
            function takes 2 parameters" << endl;
            continue;
            }
            rename(rootCatalog, vec[1], fileSystem, vec[2], curUser);
            } else {
            if (!strcmp(command.c_str(), "rewrite"))
            {
            if (vec.size()
            != 2) {
            cout <<
            "this function takes 1 parameter" << endl;
            continue;
            }
            rewrite(rootCatalog, vec[1], fileSystem, curUser);
            } else {
            if (!strcmp(command.c_str(), "changeattributes")) {
            if (vec.size() != 3) {
            cout
            << "this function takes 2 parameters" << endl;
            continue;
            }
            changeattributes(rootCatalog, vec[1], fileSystem, vec[2],
            curUser);
            } else {
            if (!strcmp(command.c_str(), "exit")) {
            if (vec.size() != 1) {
            cout << "this function takes 0 parameter" << endl;

```

```

        continue;
    }

    break;
    } else {
        if (!strcmp(command.c_str(), "help")) {
            if (vec.size() != 1) {
                cout << "this function takes 0 parameter" << endl;
                continue;
            }
            cout << "=====HELP=====<< endl;

            cout << "show - show files" << endl;
            cout << "createfile <fileName> - create file" << endl;
            cout << "deletefile <fileName> - delete file" << endl;
            cout << "rename <fileName> <newFileName> - show files" << endl;
            cout << "write <fileName> - write in file (to the end)" << endl;
            cout << "read <fileName> - read from file" << endl;
            cout << "rewrite <fileName> - write in file (delete old data and write new data)" << endl;
            cout << "createuser - create new user" << endl;
            cout << "deleteuser <username> - delete user" << endl;
            cout << "changeattributes <fileName> <newAttributes> - change attributes of file." << endl;
            cout << "<newAttributes> must have 9 numbers (1 or 0) which correspond rwxrwxrwx" << endl;
            cout << "exit - leave from system" << endl;
            cout << "=====HELP=====<< endl;
        }
    } else {
        if (!strcmp(command.c_str(), "deleteuser")) {
            if (vec.size() != 2) {
                cout << "this function takes 1 parameter" << endl;
                continue;
            }
            deleteUser(rootCatalog, vec[1], fileSystem, curUser);
        } else {
            cout << "Command is not exist" << endl;
        }
    }
}

default:
    break;
} //switch

} //while
return 0;
}

std::vector<std::string> split(const std::string& text, char sep) {
    std::vector<std::string> tokens;
    std::size_t start = 0, end = 0;
    while ((end = text.find(sep, start)) != std::string::npos) {
        if (end != start) {
            tokens.push_back(text.substr(start, end - start));
        }
        start = end + 1;
    }
    if (end != start) {
        tokens.push_back(text.substr(start));
    }
    return tokens;
}

void createFile(File& rootCatalog, std::string fName, FileSystem& fileSystem, User& user) {
    using namespace std;

```

```

    try {
        rootCatalog.findDirectoryNote(fName);
        cout << "File with this name is already exist" << endl;
    }
    catch (const std::exception& ex) {
        DirectoryNote oldDirectoryNote;
        oldDirectoryNote.noteSize = 0;

        rootCatalog.seekGet(rootCatalog.findFreeDirectoryNote(sizeof(DirectoryNote) - sizeof(fName) + fName.length() + 4 - fName.length() % 4), BEGIN_POS);
        if (rootCatalog.getDirectoryNote().fileSize != rootCatalog.getCurrentGetPosition()) {
            rootCatalog.readDirectoryNote(oldDirectoryNote);
        }
        // Расчет размера каталоговой записи: размер структуры DirectoryNote - размер поля fileName типа string + для имени + заполнение до 4х байтовой границы
        File newFile(&fileSystem, user.userID, fileSystem.findFirstFreeBlockNumber(), user.userID, rootCatalog.findFreeDirectoryNote(sizeof(DirectoryNote) - sizeof(fName) + fName.length() + 4 - fName.length() % 4), oldDirectoryNote.noteSize, fName, 0, 0, 0);

        // Запись каталоговой записи в корневой каталог
        rootCatalog.seekPut(newFile.getPositionOfNoteInRootDirectory(), BEGIN_POS);

        rootCatalog.addDirectoryNote(newFile.getDirectoryNote());

        // Обновление каталоговой записи корневого каталога
        rootCatalog.seekPut(0, BEGIN_POS);

        rootCatalog.addDirectoryNote(rootCatalog.getDirectoryNote());
    }
}

void deleteFile(File& rootCatalog, std::string fName, FileSystem& fileSystem, User& user) {
    using namespace std;
    int fPos = -1;
    try {
        fPos = rootCatalog.findDirectoryNote(fName);
        File file(&fileSystem, &rootCatalog, fPos);
        if (user.userID != file.getDirectoryNote().ownerID) {
            if (user.userID != 1)
                throw runtime_error("You are not file owner");
            file.deleteFile();
        }

        rootCatalog.seekPut(file.getPositionOfNoteInRootDirectory(), BEGIN_POS);
        rootCatalog.addDirectoryNote(file.getDirectoryNote());
    }
    catch (const std::exception& ex) {
        cout << ex.what() << endl;
    }
}

void show(File& rootCatalog, FileSystem& fileSystem) {
    using namespace std;
    rootCatalog.seekGet(0, BEGIN_POS);
    cout << setw(10) << "Flags" << setw(20) << "File name" << setw(10) << "File size" << setw(30) << "Creation time" << setw(10) << "Group ID" << setw(10) << "Owner ID" << endl;
    while (true) {
        try {
            DirectoryNote dirNote;
            rootCatalog.readDirectoryNote(dirNote);
            if ((dirNote.is == 1) & (dirNote.system == 0) & (dirNote.hidden == 0)) {
                char flags[] { 'r', 'w', 'x', 'r', 'w', 'x', 'r', 'w', 'x', '\0' };
                checkFlags(flags, dirNote);
                char time[26];
                ctime_s(time, sizeof(time), &dirNote.creationTime);
                time[24] = time[25];
                cout << setw(10) << flags << setw(20) << dirNote.fileName << setw(10) << dirNote.fileSize <<

```

```

        setw(30)<< time << setw(10) <<
(int)dirNote.groupID << setw(10) << (int)dirNote.ownerID << endl;
    }
    catch (const std::exception& ex) {
        cout << ex.what() << endl;
        break;
    }
}

void write(File& rootCatalog, std::string fName, FileSystem&
fileSystem, User& user) {
    using namespace std;
    int fPos = -1;
    try {
        fPos = rootCatalog.findDirectoryNote(fName);
        File file(&fileSystem, &rootCatalog, fPos);
        if (user.userID != 1)
            checkWriteRights(user, file.getDirectoryNote());
        file.seekPut(0, END_POS);
        cout << "Enter your data: ";
        string fData;

        getline(cin, fData);
        file.write(fData.data(), fData.size());

        rootCatalog.seekPut(file.getPositionOfNoteInRootDirectory(),
BEGIN_POS);
        rootCatalog.addDirectoryNote(file.getDirectoryNote());

    }
    catch (const std::exception& ex)
    {
        cout << ex.what() << endl;
    }
}

void read(File& rootCatalog, std::string fName, FileSystem&
fileSystem, User& user) {
    using namespace std;
    int fPos = -1;
    try {
        fPos = rootCatalog.findDirectoryNote(fName);
        File file(&fileSystem, &rootCatalog, fPos);
        file.seekGet(0, BEGIN_POS);
        if (user.userID != 1)
            checkReadRights(user, file.getDirectoryNote());
        string fData;
        fData.resize(file.getDirectoryNote().fileSize);
        file.read(fData.data(), file.getDirectoryNote().fileSize);
        cout << fData << endl;
    }
    catch (const std::exception& ex) {
        cout << ex.what() << endl;
    }
}

void rename(File& rootCatalog, std::string fName, FileSystem&
fileSystem, std::string fNewName, User& user) {
    using namespace std;
    int fPos = -1;
    try {
        fPos = rootCatalog.findDirectoryNote(fName);
        File file(&fileSystem, &rootCatalog, fPos);
        if (user.userID != file.getDirectoryNote().ownerID) {
            if (user.userID != 1)
                throw runtime_error("You are not file owner");
        }
        try {
            rootCatalog.findDirectoryNote(fNewName);
            cout << "File with this name is already exist" << endl;
        }
        catch (const std::exception& ex) {
            // Удаление каталоговой записи
            DirectoryNote curDirNote = file.getDirectoryNote();
            curDirNote.is = 0;
            curDirNote.fileSize = 0;
            curDirNote.firstFATBlockNumber = 0;

            rootCatalog.seekPut(file.getPositionOfNoteInRootDirectory(),
BEGIN_POS);
            rootCatalog.addDirectoryNote(curDirNote);

            // Создание каталоговой записи с новым именем
            DirectoryNote newDirNote = file.getDirectoryNote();
            newDirNote.fileNameLength = fNewName.length();
            newDirNote.fileName = fNewName;

            rootCatalog.seekGet(rootCatalog.findFreeDirectoryNote(sizeof
f(DirectoryNote) - sizeof(string) +

```

```

        newDirNote.fileName.length() + 4 -
newDirNote.fileName.length() % 4, BEGIN_POS);
        DirectoryNote oldDirectoryNote;

        if (rootCatalog.getDirectoryNote().fileSize !=
rootCatalog.getCurrentGetPosition())
        {
            rootCatalog.readDirectoryNote(oldDirectoryNote);
            newDirNote.noteSize =
oldDirectoryNote.noteSize;
        }

        rootCatalog.seekPut(rootCatalog.findFreeDirectoryNote(sizeof
f(DirectoryNote) - sizeof(string) +
        newDirNote.fileName.length() + 4 -
newDirNote.fileName.length() % 4, BEGIN_POS);
        rootCatalog.addDirectoryNote(newDirNote);
    }
}
}
catch (const std::exception& ex)
{
    cout << ex.what() << endl;
}
}

void changeattributes(File& rootCatalog, std::string fName,
FileSystem& fileSystem, std::string newAttr, User& user)
{
    using namespace std;
    int fPos = -1;
    try {
        fPos = rootCatalog.findDirectoryNote(fName);
        File file(&fileSystem, &rootCatalog, fPos);
        //checkWriteRights(user, file.getDirectoryNote());
        if (user.userID != file.getDirectoryNote().ownerID) {
            if (user.userID != 1)
                throw runtime_error("You are not file owner");
        }

        file.changeAttributes(newAttr);

        rootCatalog.seekPut(file.getPositionOfNoteInRootDirectory(),
BEGIN_POS);
        rootCatalog.addDirectoryNote(file.getDirectoryNote());

    }
    catch (const std::exception& ex)
    {
        cout << ex.what() << endl;
    }
}

void deleteUser(File& rootCatalog, std::string uName,
FileSystem& fileSystem, User& user) {
    using namespace std;
    string fName = "users";
    int uld = 1;
    int fPos = -1;
    if (user.userID == 1)
        try {
            fPos = rootCatalog.findDirectoryNote(fName);
            File file(&fileSystem, &rootCatalog, fPos);
            file.seekGet(0, BEGIN_POS);
            User usersCount;
            file.read((char*)&usersCount, sizeof(usersCount));
            string userName = "Deleted";

            for (int i = 0; i < usersCount.userID; i++)
            {
                User user;
                file.read((char*)&user, sizeof(user));
                if (!strcmp(user.userName, uName.c_str())) {
                    file.seekPut(sizeof(user) * user.userID,
BEGIN_POS);
                    uld = user.userID;
                    strcpy_s(user.userName, uName.c_str());
                    file.write((char*)&user, sizeof(user));
                }
            }
        }
    }
}

```

```

        //throw runtime_error("You entered in system");
    }
}
catch (const std::exception& ex)
{
    cout << ex.what() << endl;
}
while (true)
{
    try
    {
        DirectoryNote dirNote;
        rootCatalog.readDirectoryNote(dirNote);
        if ((dirNote.is == 1) & (dirNote.system == 0) &
            (dirNote.hidden == 0) & (dirNote.ownerID == uId)) {
            dirNote.ownerID = 1;
            fPos
            rootCatalog.findDirectoryNote(dirNote.fileName);
            rootCatalog.seekPut(fPos, BEGIN_POS);
            rootCatalog.addDirectoryNote(dirNote);
        }
    }
    catch (const std::exception& ex) {
        cout << ex.what() << endl;
        break;
    }
}

void createuser(File& rootCatalog, FileSystem& fileSystem, User&
user) {
    using namespace std;
    string fName = "users";
    int fPos = -1;
    if (user.userID == 1)
    try
    {
        fPos = rootCatalog.findDirectoryNote(fName);
        File file(&fileSystem, &rootCatalog, fPos);
        file.seekGet(0, BEGIN_POS);
        User usersCount;
        file.read((char*)&usersCount, sizeof(usersCount));

        file.seekPut(0, END_POS);
        User newUser;
        string newName, newPassWord;

        cout << "Enter username: ";
        //cin >> newUser.userName;
        getline(cin, newName);
        strcpy_s(newUser.userName, newName.c_str());
        /*(newUser.userName) = *(sha256("a").c_str()); //232
        for (int i = 0; i < usersCount.userID; i++)
        {
            User user;
            file.read((char*)&user, sizeof(user));
            if (!strcmp(user.userName, newUser.userName)) {
                //cin.ignore(32767, '\n');
                throw runtime_error("User with this name is
already exist");
            }
        }
        cout << "Enter password: ";
        //cin >> newUser.password;
        getline(cin, newPassWord);

        strcpy_s(newUser.password, sha256(newPassWord).c_str());
        cout << "Enter group ID: ";
        int gID;
        cin >> gID;
        cin.ignore(32767, '\n');
        newUser.groupID = gID;
        newUser.userID = ++usersCount.userID;

        file.write((char*)&newUser, sizeof(newUser));
        file.seekPut(0, BEGIN_POS);
        file.write((char*)&usersCount, sizeof(usersCount));

        rootCatalog.seekPut(file.getPositionOfNoteInRootDirectory(),
BEGIN_POS);
        rootCatalog.addDirectoryNote(file.getDirectoryNote());
    }
    catch (const std::exception& ex)
    {
        cout << ex.what() << endl;
    }
}

```

```

    }
}

void rewrite(File& rootCatalog, std::string fName, FileSystem&
fileSystem, User& user) {
    using namespace std;
    int fPos = -1;
    try
    {
        fPos = rootCatalog.findDirectoryNote(fName);
        File file(&fileSystem, &rootCatalog, fPos);

        if (user.userID != 1)
            checkWriteRights(user, file.getDirectoryNote());
        file.deleteData();
        file.seekPut(0, END_POS);
        cout << "Enter your data: ";
        string fData;

        getline(cin, fData);
        file.write(fData.data(), fData.size());

        rootCatalog.seekPut(file.getPositionOfNoteInRootDirectory(),
BEGIN_POS);
        rootCatalog.addDirectoryNote(file.getDirectoryNote());
    }
    catch (const std::exception& ex)
    {
        cout << ex.what() << endl;
    }
}

void checkWriteRights(User& user, const DirectoryNote& dn) {
    if ((dn.ownerID == user.userID) & (dn.ownerWrite == 1)) {
        return ;
    }
    else {
        if ((dn.groupID == user.groupID) & (dn.groupWrite == 1))
        {
            return ;
        }
        else {
            if (dn.otherWrite == 1) {
                return ;
            }
            else {
                throw std::runtime_error("You haven't write
rights");
                return ;
            }
        }
    }
}

void checkReadRights(User& user, const DirectoryNote& dn) {
    if ((dn.ownerID == user.userID) & (dn.ownerRead == 1)) {
        return ;
    }
    else {
        if ((dn.groupID == user.groupID) & (dn.groupRead == 1))
        {
            return ;
        }
        else {
            if (dn.otherRead == 1) {
                return ;
            }
            else {
                throw std::runtime_error("You haven't read
rights");
                return ;
            }
        }
    }
}

SuperBlock.h
#include<stdint.h>
#pragma once

struct SuperBlock{
    int64_t type;
    int32_t blockCount;
    int32_t freeBlockCount;
    int32_t firstDataBlockNumber;
    int32_t blockSize;
};

```

```

User.h
#pragma once
#include <cinttypes>

struct User
{
    char userName[61];
    char password[65];
    int8_t userID;
    int8_t groupID;
};

DirectoryNote.h
#pragma once
#include <cinttypes>
#include <string>
#include <ctime>

#pragma pack(push, 1)
struct DirectoryNote {

    int8_t is;

    time_t creationTime;

    time_t modificationTime;

    int32_t fileSize;

    int32_t firstFATBlockNumber;

    int16_t noteSize;

    //uint16_t attributes;

    uint16_t ownerRead : 1;
    uint16_t ownerWrite : 1;
    uint16_t ownerUse : 1;
    uint16_t groupRead : 1;
    uint16_t groupWrite : 1;
    uint16_t groupUse : 1;
    uint16_t otherRead : 1;
    uint16_t otherWrite : 1;
    uint16_t otherUse : 1;

    uint16_t readOnly : 1;
    uint16_t hidden : 1;
    uint16_t system : 1;

    uint16_t reserved1 : 1;
    uint16_t reserved2 : 1;
    uint16_t reserved3 : 1;
    uint16_t reserved4 : 1;

    int8_t ownerId;

    int8_t groupId;

    int8_t fileNameLength;

    std::string fileName;

};
#pragma pack(pop)

File.h
#pragma once
#include "DirectoryNote.h"
#include "FileSystem.h"

#define BEGIN_POS 1
#define CURRENT_POS 2
#define END_POS 3

class File {
    DirectoryNote directoryNote;

    FileSystem *fileSystem;
    int currentWritePosition;
    int currentReadPosition;
    int positionOfNoteInRootDirectory;

public:
    // Конструктор для создания
    File(FileSystem* fs, int firstFATBlockNumber,
    int ownerId, int groupId, int
    positionOfNoteInRootDirectory, int noteSize,
    std::string fileName, int isSystem, int isHidden,
    int isReadOnly);

    // Конструктор для чтения корневого каталога
    File(FileSystem* fs);

    // Конструктор для чтения
    File(FileSystem* fs, File* rootDir, int
    posInRootDir);

    // Деструктор
    ~File();

    // Запись в файл
    void write(const char *str, int length);

    // Чтение из файла
    void read(char* str, int length);

    // Возвращает каталоговую запись файла
    const DirectoryNote &getDirectoryNote();

    // Смещение указателя чтения
    void seekGet(int countOfBytes, int position);

    // Смещение указателя записи
    void seekPut(int countOfBytes, int position);

    // Запись каталоговой записи на диск
    void addDirectoryNote(const DirectoryNote&
    directoryNote);

    // Поиск каталоговой записи по имени
    int findDirectoryNote(std::string fileName);

    // Считывание 1 каталоговой записи
    void readDirectoryNote(DirectoryNote&
    directoryNote);

    // Возвращает позицию каталоговой записи в
    // корневом каталоге
    int getPositionOfNoteInRootDirectory();

    // Создает новый файл
    void createNewFile(std::string fName);

    // Поиск свободной каталоговой записи,
    // подходящей по размеру
    int findFreeDirectoryNote(int fNoteSize);

    // Удаление файла
    void deleteFile();

    int getCurrentGetPosition();

    void changeAttributes(std::string attr);

    void deleteData();
};

File.cpp
#include "File.h"
#include<iostream>

File::File(FileSystem* fs, int
firstFATBlockNumber, int ownerId, int groupId,
int positionInRoot, int noteSize, std::string

```

```

fileName, int isSystem, int isHidden, int
isReadOnly) {
    fileSystem = fs;
    currentWritePosition = 0;
    currentReadPosition = 0;
    positionOfNoteInRootDirectory =
positionInRoot;
    // Заполнение структуры DirectoryNote
    directoryNote.is = 1;
    directoryNote.creationTime = time(0);
    directoryNote.modificationTime = time(0);
    directoryNote.firstFATBlockNumber =
firstFATBlockNumber;
    fileSystem-
>makeBlockLastInFATChain(firstFATBlockNumber);
    directoryNote.ownerID = ownerID;
    directoryNote.groupID = groupID;
    directoryNote.fileSize = 0;
    directoryNote.fileName = fileName;
    directoryNote.fileNameLength =
(int)directoryNote.fileName.size();
    directoryNote.noteSize =
std::max(noteSize, (int) (sizeof(directoryNote) -
sizeof(directoryNote.fileName) +
directoryNote.fileNameLength + 4 -
directoryNote.fileNameLength % 4));
    // Заполнение атрибутов файла
    directoryNote.groupRead = 1;
    directoryNote.groupUse = 1;
    directoryNote.groupWrite = 1;
    directoryNote.ownerRead = 1;
    directoryNote.ownerUse = 1;
    directoryNote.ownerWrite = 1;
    directoryNote.otherRead = 1;
    directoryNote.otherUse = 1;
    directoryNote.otherWrite = 1;

    directoryNote.system = isSystem;
    directoryNote.hidden = isHidden;
    directoryNote.readOnly = isReadOnly;
}

File::File(FileSystem* fs) {
    fileSystem = fs;
    currentWritePosition = 0;
    currentReadPosition = 0;
    positionOfNoteInRootDirectory = 0;
    directoryNote.firstFATBlockNumber = fs-
>getSuperBlock().firstDataBlockNumber;
    readDirectoryNote(directoryNote);
}

File::File(FileSystem* fs, File* rootDir, int
posInRootDir) {
    fileSystem = fs;
    currentWritePosition = 0;
    currentReadPosition = 0;
    positionOfNoteInRootDirectory = posInRootDir;
    rootDir->seekGet(posInRootDir, BEGIN_POS);
    rootDir->readDirectoryNote(directoryNote);
}

File::~File() {
}

void File::read(char* str, int length) {
    std::fstream* fs = fileSystem-
>getFileStream();
    int curBlock =
directoryNote.firstFATBlockNumber;
    int pos = currentReadPosition;
    int length1 = length;
    for (int i = 0; i < pos / 4096; i += 4096) {
        pos -= 4096;
        curBlock = fileSystem-
>getNextFATChainBlockNumber(curBlock);
    }
    do {
        fs->seekg(fileSystem-
>getFirstByteOfBlock(curBlock) + pos,
std::ios::beg);
        fs->read(str, sizeof(char) *
std::min(fileSystem->getBlockSize() - pos,
length));
        str += std::min(fileSystem->getBlockSize()
- pos, length);
        length -= std::min(fileSystem-
>getBlockSize() - pos, length);
        if (length != 0) {
            curBlock = fileSystem-
>getNextFATChainBlockNumber(curBlock);
        }
        pos = 0;
    } while (length != 0);
    seekGet(length1, CURRENT_POS);
}

void File::write(const char* str, int length) {
    std::fstream* fs = fileSystem-
>getFileStream();
    int curBlock =
directoryNote.firstFATBlockNumber;
    int pos = currentWritePosition;
    seekPut(length, CURRENT_POS);
    directoryNote.fileSize =
std::max(currentWritePosition,
directoryNote.fileSize);
    for (int i = 0; i < pos / 4096; i += 4096) {
        pos -= 4096;
        curBlock = fileSystem-
>getNextFATChainBlockNumber(curBlock);
        if (curBlock == -1) {
            curBlock = fileSystem-
>addBlockToFATBlockChain(directoryNote.firstFATBl
ockNumber);
        }
    }
    do {
        fs->seekp(fileSystem-
>getFirstByteOfBlock(curBlock) + pos,
std::ios::beg);
        int writeCount = std::min(fileSystem-
>getBlockSize() - pos, length);
        fs->write(str, writeCount);
        str += writeCount;
        length -= writeCount;
        if (length != 0) {
            curBlock = fileSystem-
>getNextFATChainBlockNumber(curBlock);
            if (curBlock == -1) {
                curBlock = fileSystem-
>addBlockToFATBlockChain(directoryNote.firstFATBl
ockNumber);
            }
        }
        pos = 0;
    } while (length != 0);
}

void File::seekGet(int countOfBytes, int
position) {
    switch (position)
    {

```

```

    case 1:// Begin
        currentReadPosition = countOfBytes;
        break;
    case 2:// Current
        currentReadPosition += countOfBytes;
        break;
    case 3:
        currentReadPosition =
directoryNote.fileSize - countOfBytes;
        break;
    default:
        break;
    }
    if (currentReadPosition >
directoryNote.fileSize) {
        throw std::runtime_error("End of file");
    }
}

void File::seekPut(int countOfBytes, int
position) {
    switch (position)
    {
    case 1:// Begin
        currentWritePosition = countOfBytes;
        break;
    case 2:// Current
        currentWritePosition += countOfBytes;
        break;
    case 3:
        currentWritePosition =
directoryNote.fileSize - countOfBytes;
        break;
    default:
        break;
    }
    directoryNote.fileSize =
std::max(currentWritePosition,
directoryNote.fileSize);
}

const DirectoryNote& File::getDirectoryNote() {
    return directoryNote;
}

void File::addDirectoryNote(const DirectoryNote&
directoryNote1) {

    write((char*)&directoryNote1,
sizeof(directoryNote1) -
sizeof(directoryNote1.fileName));
    write((char*)&directoryNote1.fileName.data(),
directoryNote1.fileNameLength);
    seekPut(4 - directoryNote1.fileNameLength %
4, CURRENT_POS);
}

int File::findDirectoryNote(std::string fileName)
{
    DirectoryNote dirNote;
    seekGet(0, BEGIN_POS);
    do {
        readDirectoryNote(dirNote);
        if ((dirNote.fileName == fileName) &
(dirNote.is == 1)) {
            return currentReadPosition -
dirNote.noteSize;
        }
    } while (currentReadPosition <
directoryNote.fileSize);
    throw std::runtime_error("File with this name
isn't exist");
}

}

void File::readDirectoryNote(DirectoryNote&
directoryNote1) {
    read((char*)&directoryNote1,
sizeof(directoryNote1) -
sizeof(directoryNote1.fileName));
    directoryNote1.fileName.clear();
    directoryNote1.fileName.resize(directoryNote1
.fileNameLength);
    read(directoryNote1.fileName.data(),
directoryNote1.fileNameLength);
    seekGet(directoryNote1.noteSize -
(sizeof(directoryNote1) -
sizeof(directoryNote1.fileName) +
directoryNote1.fileNameLength), CURRENT_POS);
}

int File::getPositionOfNoteInRootDirectory() {
    return positionOfNoteInRootDirectory;
}

void File::createNewFile(std::string fName) {
}

int File::findFreeDirectoryNote(int fNoteSize) {
    DirectoryNote dirNote;
    seekGet(0, BEGIN_POS);
    do {
        readDirectoryNote(dirNote);
        if ((dirNote.noteSize>=fNoteSize) &
(dirNote.is == 0)) {
            return currentReadPosition -
dirNote.noteSize;
        }
    } while (currentReadPosition <
directoryNote.fileSize);
    return directoryNote.fileSize;
}

void File::deleteFile() {
    directoryNote.is = 0;
    try
    {
        fileSystem->
deleteFATBlockChain(directoryNote.firstFATBlockN
umber);
    }
    catch (const std::exception& ex)
    {
        std::cout << ex.what() << std::endl;
    }
    directoryNote.firstFATBlockNumber = 0;
    directoryNote.fileSize = 0;
}

int File::getCurrentGetPosition() {
    return currentReadPosition;
}

void File::changeAttributes(std::string attr) {
    directoryNote.ownerRead = attr.at(0);
    directoryNote.ownerWrite = attr.at(1);
    directoryNote.ownerUse = attr.at(2);
    directoryNote.groupRead = attr.at(3);
    directoryNote.groupWrite = attr.at(4);
    directoryNote.groupUse = attr.at(5);

    directoryNote.otherRead = attr.at(6);
    directoryNote.otherWrite = attr.at(7);
    directoryNote.otherUse = attr.at(8);
}

void File::deleteData() {
}

```



```

        fileSystem->makeBlockLastInFATChain(directoryNote.firstFATBlockNumber);
        directoryNote.fileSize = 0;}
void File::changeOwnerToRoot() {
    directoryNote.ownerID = 1;}
FileSystem.h
#pragma once
#include "SuperBlock.h"
#include <string>
#include <fstream>
#include <vector>
class FileSystem{
    SuperBlock superBlock;
    std::string filePath;
    std::fstream fileStream;
    std::vector<int32_t> FAT;
public:
    FileSystem(int param);
    ~FileSystem();
    void formatting(int clusterSize);
    void writeZeroCharactersOnDisc(int count);
    SuperBlock getSuperBlock();
    int findFirstFreeBlockNumber();
    void calculateCountOfFreeBlocks();
    void writeSuperBlock();
    void writeFAT();
    void deleteFATBlockChain(int firstFileBlockNumber);
    int newFATBlockChain(int blocksCount);
    void resizeFATBlockChain(int firstBlockNumber, int newBlocksCount);
    int getLastBlockOfChain(int firstBlockNumber);
    int addBlockToFATBlockChain(int firstBlockNumber);
    int getFirstByteOfBlock(int blockNumber);
    std::fstream* getFileStream();
    int getNextFATChainBlockNumber(int curBlockNumber);
    void makeBlockLastInFATChain(int blockNumber);
    int getBlockSize();};

FileSystem.cpp
#include "FileSystem.h"
#include <iostream>
#include <filesystem>
#define CLUSTER_SIZE 4096
#define IDENTIFICATION_NUMBER - 8613303245920329199
FileSystem::FileSystem(int param){
    using namespace std;
    filePath = "FileForTest";
    fileStream.open(filePath, ios::out|ios::in|ios::binary);
    if (!fileStream) {
        cout << "Error: file isn't opened!" << endl;
        exit(0);
    }
    if (param == 1) {
        // Работает с отформатированным файлом
        fileStream.seekg(0, std::ios::beg);
        fileStream.read((char*)&superBlock, sizeof(superBlock));
        FAT.resize(superBlock.blockCount);
        for (int i = 0; i < FAT.size(); i++) {
            fileStream.read((char*)&FAT.at(i), sizeof(int32_t));
        }
    }
    FileSystem::~~FileSystem() {
        writeSuperBlock();
        writeFAT();
        fileStream.close();
    }
}

```

```

void FileSystem::formatting(int clusterSize) {
    fileStream.seekg(0, std::ios::end);
    int32_t fileSize = (int32_t)fileStream.tellg();
    // Число - идентификатор системы
    superBlock.type = IDENTIFICATION_NUMBER;
    superBlock.blockSize = clusterSize;
    int32_t blockCount = fileSize / superBlock.blockSize;
    superBlock.blockCount = blockCount;
    // Для каждого элемента FAT выделяется 4 байта
    int FATSize = 4 * superBlock.blockCount;
    // Вычисляется номер следующего за системной областью блока (т.к. нумерация идет с 0, то мы не прибавляем еще 1)
    superBlock.firstDataBlockNumber = (sizeof(superBlock) + FATSize) / superBlock.blockSize;
    // Если системная область занимала не целое количество блоков, то первый блок данных смещается еще на 1 блок
    if ((sizeof(superBlock) + FATSize) % superBlock.blockSize != 0) {
        superBlock.firstDataBlockNumber++;
    }
    // Затираем системную область данных диска
    writeZeroCharactersOnDisc(sizeof(superBlock) + FATSize);
    FAT.clear();
    FAT.resize(superBlock.blockCount);
    for (int i = 0; i < superBlock.firstDataBlockNumber; i++) {
        FAT.at(i) = -4;
    }
    FAT.at(superBlock.firstDataBlockNumber) = -1;
    // Количество свободных блоков = количество блоков - размер системной области
    calculateCountOfFreeBlocks();
    writeSuperBlock();
    writeFAT();}

void FileSystem::writeSuperBlock() {
    fileStream.seekp(0, std::ios::beg);
    fileStream.write((char*)&superBlock, sizeof(superBlock));
    fileStream.seekp(sizeof(superBlock), std::ios::beg);}

void FileSystem::writeFAT() {
    fileStream.seekp(sizeof(superBlock), std::ios::beg);
    for (int i = 0; i < FAT.size(); i++) {
        fileStream.write((char*)&FAT.at(i), sizeof(int32_t));
    }
}

SuperBlock FileSystem::getSuperBlock() {
    return superBlock;}

void FileSystem::writeZeroCharactersOnDisc(int count) {
    fileStream.seekp(0, std::ios::beg);
    char zero = 0x00;
    for (int i = 0; i < count; i++){
        fileStream.write((char*)&zero, sizeof(char));
    }
}

int FileSystem::findFirstFreeBlockNumber() {
    for (int i = superBlock.firstDataBlockNumber; i < superBlock.blockCount; i++) {
        if (FAT.at(i) == 0) {
            superBlock.freeBlockCount--;
            FAT.at(i) = -1;
            return i;
        }
    }
    return -1;}

void FileSystem::calculateCountOfFreeBlocks() {
    superBlock.freeBlockCount = 0;
}

```

```

        for (int i = superBlock.firstDataBlockNumber;
i < superBlock.blockCount; i++) {
            if (FAT.at(i) == 0) {
                superBlock.freeBlockCount++;
            } }
void FileSystem::deleteFATBlockChain(int
blockNumber) {
    int currentBlockValue;
    if (FAT.at(blockNumber) == 0) {
        //throw std::runtime_error("This file
hadn't first block number");
        return;
    } do {
        currentBlockValue = FAT.at(blockNumber);
        FAT.at(blockNumber) = 0;
        blockNumber = currentBlockValue;
        superBlock.freeBlockCount++;
    } while (currentBlockValue != -1);
}
int FileSystem::newFATBlockChain(int blocksCount)
{
    int firstChainBlockNumber =
findFirstFreeBlockNumber();
    if (blocksCount == 1) {
        FAT.at(firstChainBlockNumber) = -1;
    } else {
        int chainBlockNumber =
firstChainBlockNumber;
        for (int i = 0; i < blocksCount - 1; i++){
            FAT.at(chainBlockNumber) = -5;
            FAT.at(chainBlockNumber) =
findFirstFreeBlockNumber();
            chainBlockNumber =
FAT.at(chainBlockNumber); }
        FAT.at(chainBlockNumber) = -1; }
    superBlock.freeBlockCount -= blocksCount;
    return firstChainBlockNumber;
}
void FileSystem::resizeFATBlockChain(int
blockNumber, int blocksCount) {
    blocksCount--;
    do { blocksCount--;
        blockNumber = FAT.at(blockNumber);
    } while ((FAT.at(blockNumber) != -1) &
(blocksCount != 1));
    // Если новый и старый размер файла
совпадают, то ничего не менять
    if ((FAT.at(blockNumber) == -1) &
(blocksCount == 1)) {
    }
    else {
        // Если конец файла, то расширяем его
        if (FAT.at(blockNumber) == -1) {
            FAT.at(blockNumber) =
newFATBlockChain(blocksCount);
        }
        // Если новый размер файла меньше старого,
то удаляем оставшуюся цепочку блоков в FAT
        if (blocksCount == 1) {

            deleteFATBlockChain(FAT.at(blockNumber));
            FAT.at(blockNumber) = -1;
        } }
}
int FileSystem::getLastBlockOfChain(int
blockNumber) {

    int currentBlockValue;
    do
    {
        currentBlockValue = FAT.at(blockNumber);
        if (currentBlockValue == -1) {
            return blockNumber;
        }
        blockNumber = currentBlockValue;
    } while (true);
}

```

```

int FileSystem::addBlockToFATBlockChain(int
blockNumber) {
    if (superBlock.freeBlockCount == 0){
        throw std::runtime_error("Count of free
blocks = 0"); }
    int lastBlock = blockNumber;
    if (FAT.at(blockNumber) != 0) {
        lastBlock =
getLastBlockOfChain(blockNumber); }
    FAT.at(lastBlock) =
findFirstFreeBlockNumber();
    FAT.at(FAT.at(lastBlock)) = -1;
    superBlock.freeBlockCount--;
    return FAT.at(lastBlock);
}
int FileSystem::getFirstByteOfBlock(int
blockNumber) {
    if (blockNumber < 0) {
        throw std::runtime_error("Trying to get
blocks that not exist"); }
    return (blockNumber*superBlock.blockSize);
}
std::fstream* FileSystem::getFileStream() {
    return &fileStream;
}
int FileSystem::getNextFATChainBlockNumber(int
blockNumber) {
    return FAT.at(blockNumber); }
void FileSystem::makeBlockLastInFATChain(int
blockNumber) {
    if (FAT.at(blockNumber) != -1) {
        deleteFATBlockChain(blockNumber);
        FAT.at(blockNumber) = -1;
        superBlock.freeBlockCount--;
    }
}
int FileSystem::getBlockSize() {
    return superBlock.blockSize;
}
Functions.h
#pragma once
#include<vector>
#include<string>
#include<iostream>
#include"File.h"
#include"FileSystem.h"
// Разделение строки на слова
std::vector<std::string> split(const std::string&
text, char sep);
void createFile(File& rootCatalog, std::string
fName, FileSystem& fileSystem, User& user);
void deleteFile(File& rootCatalog, std::string
fName, FileSystem& fileSystem, User& user);
void write(File& rootCatalog, std::string fName,
FileSystem& fileSystem, User& user);
void read(File& rootCatalog, std::string fName,
FileSystem& fileSystem, User& user);
void rewrite(File& rootCatalog, std::string
fName, FileSystem& fileSystem, User& user);
void createuser(File& rootCatalog, FileSystem&
fileSystem, User& user);
void show(File& rootCatalog, FileSystem&
fileSystem);
void rename(File& rootCatalog, std::string fName,
FileSystem& fileSystem, std::string fNewName,
User& user);
void changeattributes(File& rootCatalog,
std::string fName, FileSystem& fileSystem,
std::string newAttr, User& user);
void checkWriteRights(User& user, const
DirectoryNode& dn);
void checkReadRights(User& user, const
DirectoryNode& dn);
void deleteUser(File& rootCatalog, std::string
uName, FileSystem& fileSystem, User& user);

```