

**МИНОБРНАУКИ РОССИИ**

Федеральное государственное бюджетное образовательное учреждение высшего образования

**«МИРЭА – Российский технологический университет»**

|  |
| --- |
| **РТУ МИРЭА** |
|  |
| **Институт кибербезопасности и цифровых технологий (ИКБ)** |
|  |
| КБ-2 «Прикладные информационные технологии» |

**ОТЧЕТ О ВЫПОЛНЕНИИ ПРАКТИЧЕСКОЙ РАБОТЫ №5**

**В РАМКАХ ДИСЦИПЛИНЫ «ОПЕРАЦИОННЫЕ СИСТЕМЫ»**

Выполнил:

Студент 2-ого курса

Учебной группы БИСО-02-22

Зубарев В.С.

Оглавление

[Задание 3](#_Toc163922152)

[Листинг программы 5](#_Toc163922153)

# Задание

Разработать программу моделирующую файловую системы. При моделировании считать, что:

- объем внешней памяти, используемой для хранения файловой системы

– не менее 64К;

- размер кластера или блока – 512 байт.

- модель файловая система должна выполнять операции создания, удаления, копирования, перемещения и переименования файлов. Предусмотреть возможность визуального контроля за распределением пространства памяти.

Листинг программы

class FileSystem:  
 def \_\_init\_\_(self, disk\_size\_kb=64\*1024, cluster\_size\_bytes=512):  
 self.disk\_size = disk\_size\_kb \* 1024 # размер диска в байтах  
 self.cluster\_size = cluster\_size\_bytes  
 self.num\_clusters = self.disk\_size // self.cluster\_size  
 self.cluster\_map = [True] \* self.num\_clusters # False - занят, True - кластер свободен  
 self.files = {}  
  
 def create\_file(self, filename, size\_bytes):  
 if filename in self.files:  
 print("Ошибка: файл с таким именем уже существует.")  
 return  
  
 required\_clusters = (size\_bytes + self.cluster\_size - 1) // self.cluster\_size  
 free\_clusters = self.\_find\_free\_clusters(required\_clusters)  
 if free\_clusters:  
 self.files[filename] = {  
 'size': size\_bytes,  
 'clusters': free\_clusters  
 }  
 self.\_update\_cluster\_map(free\_clusters, False)  
 print(f"Файл '{filename}' успешно создан.")  
 else:  
 print("Ошибка: недостаточно свободного места на диске для создания файла данного размера.")  
  
 def delete\_file(self, filename):  
 if filename not in self.files:  
 print("Ошибка: файл с таким именем не найден.")  
 return  
  
 clusters = self.files[filename]['clusters']  
 self.\_update\_cluster\_map(clusters, True)  
 del self.files[filename]  
 print(f"Файл '{filename}' успешно удален.")  
  
 def copy\_file(self, source\_filename, dest\_filename):  
 if source\_filename not in self.files:  
 print("Ошибка: исходный файл не найден.")  
 return  
  
 source\_file = self.files[source\_filename]  
 self.create\_file(dest\_filename, source\_file['size'])  
 dest\_file = self.files[dest\_filename]  
 dest\_file['clusters'] = source\_file['clusters'][:]  
 print(f"Файл '{source\_filename}' успешно скопирован в '{dest\_filename}'.")  
  
 def move\_file(self, source\_filename, dest\_filename):  
 if source\_filename not in self.files:  
 print("Ошибка: исходный файл не найден.")  
 return  
  
 self.copy\_file(source\_filename, dest\_filename)  
 self.delete\_file(source\_filename)  
 print(f"Файл '{source\_filename}' успешно перемещен в '{dest\_filename}'.")  
  
 def rename\_file(self, old\_filename, new\_filename):  
 if old\_filename not in self.files:  
 print("Ошибка: файл для переименования не найден.")  
 return  
  
 self.files[new\_filename] = self.files.pop(old\_filename)  
 print(f"Файл '{old\_filename}' успешно переименован в '{new\_filename}'.")  
  
 def \_find\_free\_clusters(self, num\_clusters):  
 free\_clusters = []  
 count = 0  
 for i, is\_free in enumerate(self.cluster\_map):  
 if is\_free:  
 free\_clusters.append(i)  
 count += 1  
 if count == num\_clusters:  
 return free\_clusters[:num\_clusters]  
 else:  
 free\_clusters = []  
 count = 0  
 return None  
  
 def \_update\_cluster\_map(self, clusters, is\_allocated):  
 for cluster in clusters:  
 self.cluster\_map[cluster] = is\_allocated  
  
 def display\_disk\_usage(self):  
 used\_clusters = 0  
 for i, is\_free in enumerate(self.cluster\_map):  
 if not(is\_free):  
 used\_clusters+=1  
 free\_clusters = self.num\_clusters - used\_clusters  
 print(f"Использовано: {used\_clusters} кластеров")  
 print(f"Свободно: {free\_clusters} кластеров")  
  
# Пример использования  
fs = FileSystem()  
  
# Создание файлов  
fs.create\_file("file1.txt", 512) # Размер 512 байт  
fs.create\_file("file2.txt", 1024) # Размер 1024 байта  
fs.create\_file("file3.txt", 2048) # Размер 2048 байт  
  
# Вывод информации о занятом и свободном пространстве на диске  
fs.display\_disk\_usage()  
  
# Удаление файла  
fs.delete\_file("file2.txt")  
  
# Копирование файла  
fs.copy\_file("file1.txt", "file1\_copy.txt")  
  
# Перемещение файла  
fs.move\_file("file3.txt", "folder1/file3.txt")  
  
# Переименование файла  
fs.rename\_file("file1.txt", "new\_file1.txt")  
  
# Вывод информации о занятом и свободном пространстве на диске после операций  
fs.display\_disk\_usage()

# Ответы на вопросы

1. Как кодируются в атрибутах файла и каталога права доступа?   
   В атрибутах файла и каталога права доступа обычно кодируются с использованием битовых масок. Например, в POSIX-совместимых системах, таких как Linux и UNIX, права доступа кодируются в виде 9 бит, представляющих разрешения на чтение, запись и выполнение для владельца файла, группы и остальных пользователей.
2. Кто может пользоваться и изменять права доступа к файлам (ответ ОС Windows и Astra Linux)?

На ОС Windows права доступа к файлам могут быть управляемы только администраторами или пользователями, обладающими соответствующими привилегиями. На Astra Linux права доступа к файлам также могут быть изменены администратором системы или владельцем файла.

1. В чем разница в применении команд chmod и umask?

Команда chmod используется для изменения прав доступа к файлам и каталогам в UNIX-подобных системах. Команда umask устанавливает маску по умолчанию, которая определяет права доступа для вновь созданных файлов и каталогов.

1. Какие команды обработки файлов разрешают (или запрещают) права на чтение, модификацию и исполнение (ответ ОС Windows и Astra Linux)?

В ОС Windows команды для управления правами доступа включают icacls, takeown и attrib. В Astra Linux используются команды chmod, chown, chgrp, а также setfacl для более расширенного управления ACL (списками контроля доступа).

1. Назовите команды для защиты файлов (ответ ОС Windows и Astra Linux)?

Для защиты файлов в ОС Windows можно использовать утилиты и функции, такие как шифрование файлов с помощью BitLocker, управление правами доступа через утилиту icacls, а также антивирусные программы. В Astra Linux можно использовать механизмы прав доступа и аудита, шифрование файлов, настройку брандмауэра и другие средства, предоставляемые системой.

1. Обработчик прерываний.

Обработчик прерываний (interrupt handler) — это программа или функция, которая выполняется при возникновении прерывания процессора. Она обрабатывает прерывание, выполняя необходимые действия, и возвращает управление операционной системе.

1. Прерывания.

Прерывания (interrupts) — это сигналы, которые сообщают процессору о необходимости выполнить какую-то задачу вне его текущего потока выполнения. Прерывания могут быть вызваны различными устройствами или программными событиями.

1. Классификация прерываний.

Прерывания могут классифицироваться по различным критериям, таким как источник (внутренние или внешние), приоритет, способ передачи и так далее. Общие классификации включают программные (software) и аппаратные (hardware) прерывания, маскируемые (maskable) и немаскируемые (non-maskable) прерывания, внутренние (вызываемые процессором) и внешние (поступающие от внешних устройств) прерывания, а также асинхронные и синхронные прерывания.