САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ ПОЛИТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ

им. Петра Великого

Институт Компьютерных Наук и Кибербезопасности

Направление: 02.03.01 Математика и компьютерные науки

Высшая Школа Технологий Искусственного Интеллекта

Отчет по дисциплине: «Архитектура Суперкомпьютеров»

«Реализация модели массива
RAID 50, XOR, 10 байт, 10 дисков»

Выполнил: _____ **Салимли А.**

Преподаватель:_____ **Чуватов М.В.**

Санкт-Петербург

Весна 2024

Содержание:

Введение 3

1. Определение 4

2. Классификация 5

2.1 RAID 0 5

2.2 RAID 1 5

2.3 RAID 2 5

2.4 RAID 3 6

2.5 RAID 4 6

2.6 RAID 5 6

2.7 RAID 6 7

3. RAID 50 8

4. Особености реализации 9

4.1 Подробное описание методов 10

4.2 Разбиение на блоки и запись 11

4.3 Чтение при целых дисках 11

4.4 Восстановление при сбойных дисках 11

4.5 Чередование строки к строке 11

4.6 Подсчет избыточности 11

4.7 Краткое описание методов 12

5. Заключение 13

6. Список источников 14

7. Приложение А 15

Введение

Цель лабораторной работы - реализация модели избыточного массива RAID 50(5+0), с условиями:

- *Дисков в массиве - 10 дисков.*
- *Адреса в массиве - от 0 до 63 (64 адреса).*
- *Размер входных данных - 10 байт (HEX).*
- *Подсчет избыточности - с помощью XOR операции.*

Принцип работы модели:

- *Записывать данные по адресу.*
- *Прочесть данные по адресу.*
- *Восстановление диска при повреждении (удалении).*

1. Определение

RAID (Redundant Array of Independent Disks) - избыточный массив самостоятельных дисков. Означает, что объединяются два или более дисков в единый элемент с целью повышения отказоустойчивости отдельно взятого элемента массива.

Для дополнительных (избыточных) данных нужен объем. Выделяемый для них объем тоже называется «избыточный».

Само определение избыточный массив - наличие такого объема и чем он больше, тем больше у массива избыточность. Как я сказал ранее, избыточность нужна для отказоустойчивости.

Сам RAID массив - несколько дисков соединенные в единый диск с выделенными местами для хранения информации для исправления ошибок в данных.

Классификация RAID массивов определяются параметрами исполнения RAID-контроллера, по типам поддерживаемых интерфейсов накопителей и по поддерживаемым уровням.

2. Классификация

RAID 0

Этот массив также называется "стримом", потому что при его использовании информация разбивается на фиксированные блоки и записывается поочередно на все имеющиеся диски. Такая технология применяется в библиотеках игр, видеомонтаже или при рендеринге. Минимальное количество дисков для создания RAID 0 - два. Используя данный уровень RAID, скорость чтения и записи информации увеличивается, так как операции выполняются параллельно на всех дисках. Чем больше дисков в массиве, тем выше производительность. Доступен объем всех дисков. Главным недостатком является то, что отказ одного диска в массиве означает полную потерю данных, так как информация хранится разделенной между дисками.

RAID 1

Подходит для хранения важных данных, где безопасность и доступность имеют высший приоритет. Для реализации этого уровня RAID требуется использование двух накопителей. Благодаря дублированию информации обеспечивается высокая надежность системы - при отказе одного диска работа системы будет продолжена с другого. Однако необходимо провести замену поврежденного диска как можно быстрее, при этом данные могут быть восстановлены с зеркального диска. Путем распараллеливания запросов достигается высокая скорость чтения данных. Поскольку диски являются полными копиями друг друга, доступный объем хранилища равен объему одного диска. Логичным дальнейшим шагом является увеличение стоимости хранения на гигабайт данных вдвое. При этом скорость записи информации может оставаться на уровне одиночного диска или даже быть ниже.

RAID 2

Эти массивы применяют чередование дисков и коды коррекции ошибок (код Хэмминга). Благодаря этому диски в них распределяются на две категории: для самих данных и для упомянутых выше кодов.

Благодаря чередованию обеспечивается высокая скорость работы с данными по сравнению с одним диском. А код Хэмминга позволяет обнаруживать и исправлять ошибки при работе с файлами без снижения скорости работы с данными. Кроме того, при отказе одного накопителя массива данные будут восстановлены по сохранённым кодам коррекции ошибок.

RAID 3

Этот тип массива также использует чередование дисков, но без применения кодов Хэмминга. Вместо них используются контрольные суммы, которые используются для восстановления данных, при этом сами данные разбиваются на байты. Такой массив хорошо подходит для работы с большими файлами и потоковым мультимедиа, однако на практике он встречается редко из-за своей низкой надёжности. Для создания такого массива требуется минимум три диска. Скорость чтения данных достаточно высока, а скорость записи — высока только при работе с большими файлами. RAID 3 предлагает хорошее сочетание доступного объёма и стоимости. Информация теряется в случае выхода из строя более одного диска. К недостаткам относится возможное снижение скорости при работе с небольшими объёмами данных, а также то, то, что не все контроллеры поддерживают работу с таким массивом. Ещё одним минусом является высокая нагрузка нагрузка на диск, который хранит контрольные суммы, что может привести к его преждевременному

RAID 4

RAID 4 использует блочное чередование с одним диском для четности. Это означает, что данные разбиваются на блоки, и каждый блок записывается на разные диски, а блок четности записывается на отдельный диск. Таким образом, при чтении блока данных из RAID 4 массив может получить все необходимые данные с нескольких дисков одновременно, что увеличивает скорость чтения. Однако запись может замедляться, поскольку все диски должны ждать, пока блок четности будет записан на диск четности перед тем, как следующий блок может быть записан. Обеспечивает хорошую производительность при чтении, но имеет низкую производительность при записи из-за необходимости синхронизации записи блока четности. Он также менее эффективен, чем другие уровни RAID, такие как RAID 5, поскольку использует один дополнительный диск только для хранения четности, что уменьшает общий объем доступного пространства для хранения данных.

RAID 5

Тут уже используется почти такое же чередование как в прошлом массиве, это приводит к увеличению скорости записи, поскольку операции могут выполняться параллельно. В массиве должно быть не менее трёх накопителей. Объём для хранения контрольных сумм равен объёму одного накопителя. RAID 5 является наиболее популярным и используется в файловых серверах, общих серверах хранения, серверах резервного копирования, работе с

потоковыми данными и в других средах, где важна хорошая производительность. Объём накопителей зависит от их количества в массиве – чем больше накопителей, накопителей, тем больше объём. Скорость чтения высокая, выше, чем у RAID 4, и скорость записи. За счёт распределения сумм по массиву нагрузка на все диски равномерная. Однако, когда один диск выходит из строя, надёжность массива значительно снижается, и он переходит в критическое состояние. Процесс восстановления занимает много времени, что приводит к снижению производительности массива и увеличению нагрузки на накопители, поскольку с них происходит интенсивное продолжительное

RAID 6

Использует два различных блока четности, которые распределены по всем дискам в массиве. Это позволяет восстановить данные даже в случае двух дисков, которые вышли из строя. Таким образом, RAID 6 обеспечивает высокую степень защиты данных и надежности. Однако, использование двух блоков четности требует больше места для хранения данных, что уменьшает общий объем доступного пространства.

Скорость чтения и записи в RAID 6 обычно выше, чем в RAID 5, благодаря параллельной обработке данных на нескольких дисках. Однако, при записи новых данных массиву требуется больше времени для обновления обоих блоков четности, что может снизить общую производительность. Его обычно используют в крупных корпоративных системах хранения данных, где требуется высокая надежность и защита от сбоев.

3. RAID 50 (5+0)

RAID 50 представляет собой комбинацию уровней RAID 0 и RAID 5. Это означает, что он использует чередование данных и блоков четности, чтобы обеспечить высокую производительность и надежность хранения данных.

RAID 50 состоит из нескольких массивов RAID 5, объединенных в один большой массив с помощью технологии RAID 0. Каждый массив RAID 5 внутри RAID 50 работает независимо, что позволяет сохранять данные даже при отказе одного диска в одном из массивов.

Преимущества RAID 50 включают высокую скорость записи и восстановления данных. Также, благодаря использованию нескольких массивов RAID 5, RAID 50 может поддерживать большую емкость и обеспечивать высокую доступность данных.

Минимальное количество дисков для создания RAID 50 составляет 6!

На рисунке 1 представлена схема массива.

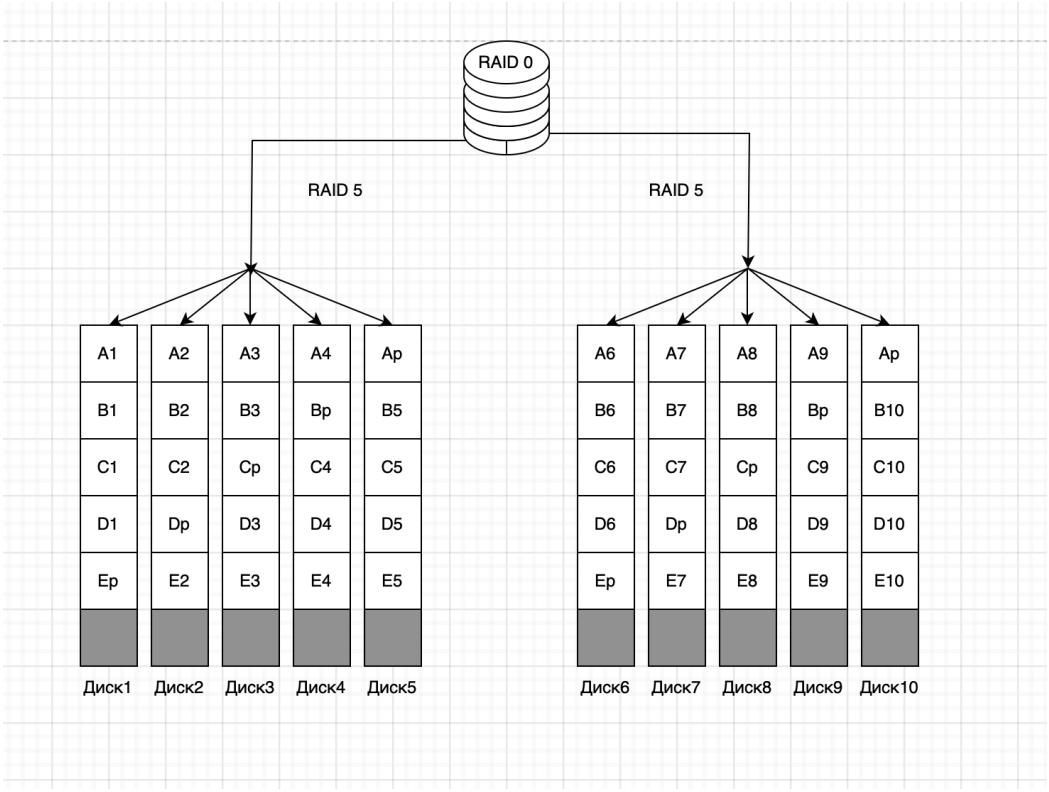

Рис.1 Схема RAID 50 для 10 дисков

Данные разделены на 5 частей: A1, A2, A3, A4, A5. В первую часть массива записывается первая часть данных A1, A2, A3, A4 и рассчитывается избыточность Ap. По такому же принципу записывается вторая часть данных во второй массив: A6-A9 и Ap для этих данных.

4. Особености реализации

4.1 Подробное описание методов.

Программа была написана на языке python в среде разработки Visual Studio Code. Программа реализует модель RAID 50 массива для 10 дисков, где данные записываются и хранятся в 16-ом формате (HEX). RAID 50 объединяет массивы RAID 5 и RAID 0: данные делятся на группы (каждая из которых является RAID 5 массивом), а затем эти группы соединяются в RAID 0 для увеличения скорости и емкости.

Сама программа состоит из двух основных классов: RAID 50 и RAID 5.

В сумме 18 методов два из которых функция вне класса и меню.

Функция hex_xor(hex_str1, hex_str2):

Выполняет побитовое XOR между двумя строками в 16-ом формате. Параметры: hex_str1: Первая строка в 16-ом формате. hex_str2, а вторая строка в 16-ом формате.

Возвращает результат XOR в виде строки в 16-ом формате.

Используется для расчета избыточности в RAID 5.

Класс RAID50:

- Конструктор **__init__(self, base_path, num_disks=10, group_size=5, max_layers=64)**:

Инициализирует RAID 50 массив. Параметры: base_path: Базовый путь для хранения файлов дисков. num_disks: Общее количество дисков. group_size: Размер группы дисков = 5. max_layers: Максимальное количество слоев = 64 (От нуля до 63).

- Метод **write(self, data, index)**: Записывает данные в RAID 50 массив. Параметры: data: Строка данных длиной 10 символов (HEX). index: Индекс слоя для записи. Ничего не возвращает.
- Метод **read(self, index)**: Описание: Читает данные из RAID 50 массива. Параметры: index: Индекс слоя для чтения.

Возвращает прочитанные данные (или сообщение об ошибке).

- Метод **recover(self)**: Восстанавливает данные на дисках. Без параметров и ничего не возвращает
- Метод **reset(self)**: Удаляет все данные на дисках. Без параметров и ничего не возвращает.
- Метод **get_written_indices(self)**: Возвращает список индексов слоев, в которых записаны данные. Так же без параметров.

Класс RAID5:

- Конструктор **__init__(self, base_path, group_index, num_disks, max_layers)**:

Инициализирует RAID 5 группу. Параметры: base_path: Базовый путь для хранения файлов дисков. group_index: Индекс группы. num_disks: Количество дисков в группе. max_layers: Максимальное количество слоев. Создает файлы для дисков.

- Метод **write(self, data, layer)**: Записывает данные в RAID 5 группу. Параметры: data: Строка данных длиной 5 символов. layer: Индекс слоя для записи. Ничего не возвращает.
- Метод **read(self, layer)**: Читает данные из RAID 5 группы. Параметры: layer: Индекс слоя для чтения. Возвращает прочитанные данные.
- Метод **recover(self)**: Восстанавливает данные на дисках. Без параметров и ничего не возвращает.
- Метод **is_disk_failed(self, disk_index)**: Описание: Проверяет, потерян ли диск. Параметры: disk_index: Индекс проверяемого диска. Возвращает True, если диск потерян, иначе False.
- Метод **rebuild_disk(self, disk_index)**: Перестраивает данные на потерянном диске. Параметры: disk_index: Индекс перестраиваемого диска. Ничего не возвращает.
- Метод **reset(self)**: Описание: Удаляет все данные на дисках. Без параметров и тоже ничего не возвращает.

Метод get_written_indices(self): Возвращает список индексов слоев, в которых записаны данные. Без параметров. Возвращает список индексов.

- Функция **menu()**:

Предоставляет интерфейс с терминала для взаимодействия с модели RAID 50.
Позволяет пользователю записывать данные, читать данные, восстанавливать данные, удалять данные и выходить из программы.

4.2 Разбиение на блоки и запись:

Запись данных в RAID 50: Данные принимаются строкой длиной 10 символов. Данные делятся на два блока по 5 символов. Первый блок записывается в первую группу RAID 5. Второй блок записывается во вторую группу RAID 5. Запись данных в RAID 5: Данные длиной 5 символов разбиваются на 4 части по одному символу и одну часть длиной 2 символа. Вычисляется избыточность (parity) путем поочередного применения побитового XOR для каждого символа данных. Чередование избыточности осуществляется через индекс слоя: вычисляется индекс диска для хранения избыточности на основе текущего слоя.

Данные и избыточность записываются на диски, данные записываются на оставшиеся диски кроме диска с избыточностью.

4.3 Чтение данных при целых дисках:

Для чтения данных по указанному индексу сначала читаются данные из первой группы RAID 5. Затем читаются данные из второй группы RAID 5. Если данные существуют, обе части объединяются и возвращаются пользователю. Чтение данных в RAID 5: Определяется индекс диска с избыточностью для текущего слоя. Данные считываются с остальных дисков, кроме диска с избыточностью. Считанные данные объединяются и возвращаются пользователю.

4.4 Восстановление данных при одном, двух и трех поврежденных дисках

Восстановление данных в RAID 50: В каждой группе RAID 5 вызывается метод recover(), чтобы восстановить поврежденные диски. Восстановление данных в RAID 5: Проверяется, существует ли файл каждого диска. Если файл не существует, он считается поврежденным. Для поврежденного диска вызывается метод rebuild_disk() для восстановления данных на нем. Восстановление данных на поврежденном диске: Для каждого слоя определяется индекс диска с избыточностью. Для поврежденного диска с данными: Данные для каждого слоя восстанавливаются с помощью избыточности и данных с остальных дисков. Для поврежденного диска с избыточностью: Избыточность для каждого слоя вычисляется заново, используя данные с остальных дисков.

4.5 Чередование данных от слоя к слою

Чередование избыточности (parity) осуществляется по принципу кругового смещения: каждый слой использует следующий диск в группе для хранения избыточности.

4.6 Подсчет избыточности

Данные, которые нужно записать в RAID 5, передаются в метод write в виде строки длиной 5 символов. Эти данные разбиваются на части:
1) 2
2) 2-3
3) 3-4
4) 4-5
Таким образом, у нас получается 4 части данных: первые 2 символа, третий символ,четвертый символ и пятый символ. Далее инициализация «00» и замена на R. Для каждой части данных

выполняется операция побитового XOR с текущим значением parity. Данные записываются на все диски, кроме диска с избыточностью.
Избыточность записывается на диск, который определяется параметром parity_index.

4.7 Короткое описание методов

hex_xor(hex_str1, hex_str2):

Выполняет побитовый XOR двух шестнадцатеричных строк и возвращает результат в виде шестнадцатеричной строки.

Raid50:

- __init__(self, base_path, num_disks=10, group_size=5, max_layers=64): Инициализация класса RAID 50, создание групп RAID 5.
- write(self, data, index): Запись данных в RAID 50.
- read(self, index): Чтение данных из RAID 50.
- recover(self): Восстановление данных в RAID 50.
- reset(self): Сброс данных в RAID 50. get_written_indices(self): Получение индексов записанных данных в RAID 50.

Raid5:

- __init__(self, base_path, group_index, num_disks, max_layers): Инициализация группы RAID 5.
- write(self, data, layer): Запись данных в RAID 5.
- read(self, layer): Чтение данных из RAID 5.
- recover(self): Восстановление данных в RAID 5.
- is_disk_failed(self, disk_index): Проверка на повреждение диска.
- rebuild_disk(self, disk_index): Восстановление данных на поврежденном диске.
- reset(self): Сброс данных в RAID 5.
- get_written_indices(self): Получение индексов записанных данных в RAID 5.

5. Заключение

В ходе выполнения лабораторной работы, была реализована модель, которая имитирует работу избыточного массива независимых дисков RAID 50, по условиям, для 10 дисков и для ограничения в 10 символов (HEX), с подсчетом избыточности при помощи XOR операции.

- Были получены знания о принципах работы RAID массивов.
- Была реализована модель работы RAID массива на языке программирования Python 3.9 в среде разработки VS Code.
- Было произведено тестирование нашей модели, посредством выполнения вставки 10-символьного значения HEX, удаления диска, восстановления диска и чтения информации по адресам записи.

Достоинства:

- Можно восстанавливать диски без потери данных
- Возможность чтения по адресу

Недостатки:

- Не самое лучшее количество дисков.

6. Список источников

1. RAID, https://ru.wikipedia.org/wiki/RAID,[электронный ресурс], (дата обращения - 21.05.2024г.)
2. RAID-МАССИВЫ, ИХ УРОВНИ, https://mixtelecom.ru/blog/raid-massiv,[электронный ресурс],(дата обращения - 21.05.2024г.)
3. Когда используют RAID 50, https://www.datarc.ru/articles/when-to-use-raid-50.html, [электронный ресурс], (дата обращения - 21.05.2024г.)

7. Приложение А

```
import os


def hex_xor(hex_str1, hex_str2):
    hex_str1 = str(hex_str1)
    hex_str2 = str(hex_str2)
    hex_str1 = hex_str1.replace('R', '0') #если вдруг нули
    hex_str2 = hex_str2.replace('R', '0')
    num1 = int(hex_str1, 16)
    num2 = int(hex_str2, 16)
    xor_result = num1 ^ num2
    xor_hex_str = format(xor_result, 'x')
    return xor_hex_str

class Raid50:
    def __init__(self, base_path, num_disks=10, group_size=5, max_layers=64): #диски #группы (чанки) #абобус 64
        #5 = 2 + 2 + 1 избыт
        self.num_disks = num_disks
        self.group_size = group_size
        self.max_layers = max_layers
        self.base_path = base_path
        self.groups = [Raid5(base_path, i, group_size, max_layers) for i in range(num_disks // group_size)]

    def write(self, data, index):
        if len(data) != 10:
            print("Строка должна быть в 10 символов!.")
            return
        if index >= self.max_layers:
            print(f"Ошибка! адресс должен быть в пределах нуля до {self.max_layers-1}")
            return
        chunk1 = data[:5]
        chunk2 = data[5:]
        self.groups[0].write(chunk1, index)
        self.groups[1].write(chunk2, index)

    def read(self, index):
        if index >= self.max_layers:
            print(f"Ошибка! адресс должен быть в пределах нуля до {self.max_layers-1}")
            return None
        data1 = self.groups[0].read(index)
        data2 = self.groups[1].read(index)
        if not data1 and not data2:
            return "Диски пусты."
        cdata = data1 + data2
        return cdata

    def recover(self):
        for group in self.groups:
            group.recover()

    def reset(self):
        for group in self.groups:
            group.reset()

    def get_written_indices(self):
        written_indices = set()
        for group in self.groups:
            written_indices.update(group.get_written_indices())
        return sorted(list(written_indices))

class Raid5:
    def __init__(self, base_path, group_index, num_disks, max_layers):
        self.num_disks = num_disks
        self.data_disks = num_disks - 1
        self.base_path = base_path
        self.group_index = group_index
        self.max_layers = max_layers
```

```
        self.files = [os.path.join(base_path, f"disk{i + (group_index*5)}.txt") for i in range(num_disks)]
        for file in self.files:
            if not os.path.exists(file):
                with open(file, 'w') as f:
                    f.write('\n' * max_layers)

    def write(self, data, layer):
        if len(data) != 5:
            print("Ошибка! Должно быть в 5 байтов.")
            return

        parts = [data[:2], data[2:3], data[3:4], data[4:5]] # Для того что б 1 был избыточный
        parity = "00"
        for part in parts:
            parity = hex_xor(parity,part)

        parity_index = (self.num_disks - 1 - (layer % self.num_disks)) % self.num_disks

        for i in range(self.num_disks):
            with open(self.files[i], 'r+') as f:
                lines = f.readlines()
                if i == parity_index:
                    lines[layer] = parity + '\n'
                else:
                    lines[layer] = parts.pop(0).rjust(2, 'R') + '\n'
                f.seek(0)
                f.writelines(lines)

    def read(self, layer):
        data = []
        parity_index = (self.num_disks - 1 - (layer % self.num_disks)) % self.num_disks
        for i in range(self.num_disks):
            if i == parity_index:
                continue
            try:
                with open(self.files[i], 'r') as f:
                    lines = f.readlines()
                    if len(lines) > layer:
                        data.append(lines[layer].replace('R', '').strip())
            except FileNotFoundError:
                continue
        cdata = ''.join(data) if data else None
        return str(cdata) if data else None

    def recover(self):
        for i in range(self.num_disks):
            if self.is_disk_failed(i):
                self.rebuild_disk(i)

    def is_disk_failed(self, disk_index):
        return not os.path.exists(self.files[disk_index])

    def rebuild_disk(self, disk_index):
        linescor = self.get_written_indices()
        disk_size = len(linescor)
        rebuilt_data = [""] * self.max_layers
        for pos in linescor:
            parity_index = (self.num_disks - 1 - (pos % self.num_disks)) % self.num_disks
            if disk_index == parity_index:
                parity = "00"
                for i in range(self.num_disks):
                    try:
                        with open(self.files[i], 'r') as f:
                            lines = f.readlines()
                            if len(lines) > pos:
                                data = lines[pos].strip()
                                if data == "\n" :
                                    parity = "\n"
                                    break
```

```
                                parity = hex_xor(parity, data)
                    except FileNotFoundError:
                        continue
                rebuilt_data[pos] = parity
            else:
                parity = "00"
                for i in range(self.num_disks):
                    try:
                        with open(self.files[i], 'r') as f:
                            lines = f.readlines()
                            if len(lines) > pos:
                                data = lines[pos].strip()
                                if data == "\n" :
                                    parity = "\n"
                                    break
                                parity = hex_xor(parity, data)
                    except FileNotFoundError:
                        continue
                rebuilt_data[pos] = parity

        with open(self.files[disk_index], 'w') as f:
            f.write('\n'.join(rebuilt_data) + '\n')

    def reset(self):
        for file in self.files:
            with open(file, 'w') as f:
                f.write('\n' * self.max_layers)

    def get_written_indices(self):
        written_indices = set()
        try :
            with open(self.files[0], 'r') as f:
                lines = f.readlines()
                for i, line in enumerate(lines):
                     if line.strip():
                        written_indices.add(i)
        except  :
             with open(self.files[1], 'r') as f:
                lines = f.readlines()
                for i, line in enumerate(lines):
                     if line.strip():
                        written_indices.add(i)
        return written_indices

def menu():
    base_path = './raid_disks'
    os.makedirs(base_path, exist_ok=True)
    raid50 = Raid50(base_path)

    while True:
        print("Меню:")
        print("1) Записать в диски")
        print("2) Прочесть инфу")
        print("3) Восстонавить диски")
        print("4) Удалить все")
        print("5) Выйти")
        choice = input("Выберите пункт меню: ")

        if choice == "1":
            data = input("Введите строку в 10 символов: ")
            if len(data) != 10:
                print("Ошибка! Строка должна быть в 10 байт.")
                continue
            index = int(input(f"Введите индекс от нуля до {raid50.max_layers - 1}: "))
            if index < 0:
                print("Ошибка! Индекс должен быть больше нуля!")
                continue
            else:
                raid50.write(data, index)
            continue
        elif choice == "2":
            written_indices = raid50.get_written_indices()
```

```
            if not written_indices:
                print("Диски пусты.")
            else:
                print(f"Тут есть что-то: {written_indices}")
                index = int(input(f"Введите одно из представленных индексов: "))
                if index not in written_indices:
                    print("Не верно введен индекс.")
                    continue
                result = raid50.read(index)
                print(result)
        elif choice == "3":
            raid50.recover()
            print("Восстонавление завершено!")
        elif choice == "4":
            raid50.reset()
            print("Удалено!")
        elif choice == "5":
            break
        else:
            print("Введите цифру которая есть в меню.")

if __name__ == "__main__":
    menu()
```