

Министерство образования Республики Беларусь

Учреждение образования

БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ

ИНФОРМАТИКИ И РАДИОЭЛЕКТРОНИКИ

Факультет компьютерных систем и сетей

Кафедра электронных вычислительных машин

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА

к курсовой работе

на тему

Восстановление плохих секторов методом многократного чтения

БГУИР КР 1-40 02 01 309 ПЗ

Студент:

Деруго Д. В.

Руководитель:

Глоба А. А.

Минск 2022

Оглавление

ВВЕДЕНИЕ	4
1 ОБЗОР ЛИТЕРАТУРЫ	5
1.1 Обзор используемой литературы	5
1.2 Обзор аналогов	6
1.3 Постановка задачи	6
2 СИСТЕМНОЕ ПРОЕКТИРОВАНИЕ	7
2.1 Постановка задачи	7
2.2 Разбиение на модули	7
2.2.1 Модуль парсера флагов	7
2.2.2 Модуль логирования	7
2.2.3 Модуль карты диска	8
2.2.4 Модуль чтения диска	8
3 ФУНКЦИОНАЛЬНОЕ ПРОЕКТИРОВАНИЕ	9
3.1 Описание структур программы	9
3.1.1 Структура args_t	9
3.1.2 Структура logger_info	9
3.2 Описание модулей программы	10
3.2.1 Модуль логгера my_logger	10
3.2.2 Модуль карты my_mapper	10
3.2.3 Модуль парсера arguments_parser	10
3.2.4 Модуль чтения диска rescue	11
4 РАЗРАБОТКА ПРОГРАММНЫХ МОДУЛЕЙ	12
4.1 Выделение ключевых процедур	12
4.1.1 Процедура инициализации файла карты	12
4.1.2 Процедура заполнения файла карты	12
4.1.3 Процедура обновления карты	12
4.1.4 Процедуры чтения и записи	13
4.1.5 Процедура восстановления	13
5 ПРОГРАММА И МЕТОДИКА ИСПЫТАНИЙ	15
6 РУКОВОДСТВО ПОЛЬЗОВАТЕЛЯ	17
ЗАКЛЮЧЕНИЕ	19
Литература	20
ПРИЛОЖЕНИЕ А	21
ПРИЛОЖЕНИЕ Б	22

ВВЕДЕНИЕ

Несмотря на то, что жесткие диски, HDD, отходят на второй план и активно заменяются быстрыми твердотельными SSD, HDD все еще имеют преимущество в ценовой категории и эксплуатируются массой пользователей, поэтому говорить о бесперспективности их поддержки не приходится.

Любой пользователь может столкнуться с проблемой на своем жестком диске. Если проблема не в сломанной считывающей головке и сектора не теряют данные с катастрофической скоростью ежесекундно, то данные можно перенести с не исправного диска путем использования восстанавливающих программ и утилит.

Устройства хранения данных не имеет предрасположенности к какой-либо файловой системе и, следовательно, их восстановление не касается структуры файлов и ОС в целом. Тем не менее было бы разумно направить вектор разработки на восстановление данных конкретных файловых систем.

Данный курсовой проект есть утилита, способная восстанавливать плохие сектора накопителей данных. Ее структура подобна существующей программе GNU ddrescue, которая выполняет восстановление данных с поврежденных носителей.

Для выполнения проекта следует изучить строение накопителей данных и способы взаимодействия с ними посредством языка Си в среде Linux. Также следует протестировать утилиту на каком-либо устройстве.

1 ОБЗОР ЛИТЕРАТУРЫ

1.1 Обзор используемой литературы

Поскольку в ходе курсового проектирования будет разработана утилита, следует дать пояснение, что это такое.

Утилита – это некая вспомогательная компьютерная программа, входящая в состав программного обеспечения, которая используется для выполнения типовых задач, связанной с работой оборудования или ОС. Даная утилита используется для восстановления данных с жесткого диска.

Для восстановления неисправных дисков используется чтение по секторам, размер которых обычно равен 512 байт. Существует более современный Advanced Format, где размер сектора выбран равным 4 килобайтам. Для разметки диска по 512 байт требуется больше места для служебных меток, а в ходе бурного роста компьютерных технологий столь малый размер сектора уже исчерпывает себя [2, стр. 44-45].

Для оценки состояния жесткого диска существует технология S.M.A.R.T (self-monitoring, analysis and reporting technology), которая отслеживает в том числе и количество ошибок поиска. То есть диск сам может отчитаться о том, сколько секторов сомнительной надежности, а также самостоятельно их переназначить на другие, целые, в соответствии с многочисленными критериями, сектора.

Параметр 05, Reallocated Sectors Count, показывает число переназначенных секторов. Если диск обнаруживает сомнительный сектор, то переносит данные в резервную область, размер которой выбирается при производстве в соответствии с ожидаемым количеством плохих секторов. Перенесение данных на безопасный сектор из резерва называется remapping, переназначенный сектор — гемар.

Параметры C4-6 отвечают за количество попыток переназначения, количество сомнительных и число неисправимых внутренними средствами диска секторов соответственно.

Приведенные выше параметры являются критически важными для работоспособности диска, являются индикаторами возможной скорой поломки.

Следует учесть, что флеш-накопители обычно не поддерживают S.M.A.R.T, поскольку USB-протокол основан на протоколе SCSI, что не содержит аналогичной S.M.A.R.T функциональности.

Для проектирования понадобится использовать массу стандартных библиотек языка C++, а также библиотеку `fcntl.h`, которая позволяет проводить операции над файловыми дескрипторами в ОС Linux. Это необходимо для того, чтобы работать с диском. Следует отметить, что в Linux файлы могут располагаться на несмежных секторах диска, из-за чего работа с ними несет опасность некорректно перенести файловую систему со спасаемого диска на альтернативное хранилище данных.

1.2 Обзор аналогов

Для качественного выполнения работы следует рассмотреть аналоги. Ниже приведены программы, способные работать в ОС Linux.

R-studio — программа, способная восстановить данные, которые были потеряны по различным причинам: форматирование, повреждение, удаление файлов. Программа поддерживает разные файловые системы, будь то FAT, NTFS, EXT. Программа способна формировать образ дисков для восстановления, чтобы в последствии работать с ним, а не непосредственно с носителем. Существенный минус в том, платная.

TestDisk — программа с открытым исходным кодом и лицензией GNU GPL. Она была разработана в первую очередь для восстановления данных, которые повреждены программно, из-за ошибок пользователя или вирусов. TestDisk может восстанавливать файлы разных файловых систем, включая FAT и NTFS, а также просто собрать детальную информацию о незагружающихся дисках, чтобы применить ее при дальнейшем анализе. Очень часто именно ее рекомендуют использовать вместо/с вместо/с GNU ddrescue.

GNU ddrescue — утилита UNIX-like систем, которая позволяет выбрать стратегию чтения и умеет строить карту диска для продолжения работы после сбоев или остановки пользователем процесса восстановления. Сам алгоритм утилиты считается самым сложным из программ с открытым исходным кодом, однако общий принцип чтения понятен: программа читает все сектора, что может прочесть, при этом перенося данные на диск для хранения спасаемых данных, а также запоминает неисправные сектора, «бэды», для последующего их спасения [3, гл. 4]. Утилита написана на языке C++.

1.3 Постановка задачи

Рассмотрев наиболее популярные и близкие к ОС Linux программы, можно сделать вывод о требуемом функционале утилиты:

1. Перенос данных на внешний носитель.
2. Учет плохих секторов.
3. Возможность попытки спасти информацию с плохих секторов.

2 СИСТЕМНОЕ ПРОЕКТИРОВАНИЕ

Сначала требуется поставить конкретные требования к программе, разделить утилиту на блоки и модули. Таким образом можно упростить проектирование, избежать ошибок и обеспечить гибкость модулей.

2.1 Постановка задачи

Необходимый минимум — копирование данных на исправный диск, возврат к неисправным секторам и попытка их прочтения. Для возврата к секторам по записанным адресам и в целом для анализа работы потребуется логирование результатов копирования.

Из вышеперечисленных задач можно выделить модули программы. Также следует отметить, что программа будет принимать флаги, как минимум необходимы директории спасаемого диска и диска, на который данные будут перенесены, а также файл, куда будет помещен отчет о выполнении программы. Следует обеспечить пользователю выбор максимального количества попыток, в ходе которых будут проводиться попытки спасения неисправного сектора, в противном случае будет исполняться заданное изначально количество циклов чтения. Также требуется обеспечить ручной ввод размера сектора на жестком диске.

2.2 Разбиение на модули

Следует разбить программу на модули, чтобы обеспечить простоту проектирования и потенциальной поддержки кода в будущем. Модуль — это полноценная функция программы.

2.2.1 Модуль парсера флагов

Модуль представляет собой обработчик входных данных из терминала, преобразование их во флаги, которые будут использоваться программой в дальнейшем. При отсутствии введенных пользователем флагов будут использованы значения по умолчанию. Этот же модуль отвечает за обработку переданных директорий для спасения и хранения спасенной информации.

2.2.2 Модуль логирования

В данном модуле будет происходить логирование действий программы, а именно запись и вывод на экран информации о прочитанных секторах, их состоянии, информации о размере прочитанных данных и полном времени исполнения программы. Поскольку полное чтение диска может занимать приличное количество времени, такие данные о проделанной работе необходимы. Чтобы не тормозить выполнение основной части программы, логгеру желательно выделить отдельный поток исполнения, который будет

считывать данные, изменяющиеся в ходе выполнения основного блока, производящего спасение.

2.2.3 Модуль карты диска

В этом модуле будет производиться запись карты восстанавливаемого диска с отметками «+» или «-» для диагностирования состояния секторов. Программа записывает адреса кластеров и их размер. Размер и адрес следует записывать в 16-й системе счисления, поскольку запись в меньшей СС будет громоздкой и тяжелой для восприятия. Карта диска позволит оценить повреждения накопителя и использовать эту информацию для последующей работы в восстановлении данных.

2.2.4 Модуль чтения диска

Модуль отвечает за чтение и обработку данных диска. Здесь производится основа программы, то есть чтение данных, запись на резервное место, пометка плохих секторов и последующий к ним возврат для повторного чтения. В нем же располагаются переменные, которые принимает модуль логгера для вывода на экран. Расчет плохих секторов будет производиться на основе результата скорости чтения сектора.

3 ФУНКЦИОНАЛЬНОЕ ПРОЕКТИРОВАНИЕ

Данная глава посвящена описанию функциональной составляющей разрабатываемой утилиты. Описывается структура программы, ее модули и функции.

3.1 Описание структур программы

Программа оперирует двумя структурами: `args` и `logger_info`.

Первая структура представляет собой хранилище обработанных модулем парсера директорий и флагов, введенных пользователем при запуске программы, вторая содержит указатели на переменные, отображающие прогресс выполнения программы. Рассмотрим их по-отдельности.

3.1.1 Структура `args_t`

Структура содержит следующие поля:

- `path_in` — переменная типа `char*`, содержащая путь к устройству, которое требуется спасти.
- `path_out` — переменная типа `char*`, содержащая путь к файлу, куда производится запись восстановленных данных.
- `path_map` — переменная типа `char*`, содержащая путь к файлу карты диска.
- `fd_in` и `fd_out` — переменные типа `int`, содержащие файловые дескрипторы файлов устройства и образа соответственно.
- `rescue_start_pos` и `rescue_size` — переменные `ssize_t`, отражающие стартовый адрес, с которого будет читаться устройство, и размер данных, которые нужно восстановить соответственно. Эти поля нужны для того, чтобы можно было ограничить конкретные участки диска для спасения.
- `bad_block_repeat` — `int`-переменная, количество попыток повторного чтения плохого сектора, по умолчанию равно 3.
- `block_size` — размер одного сектора в `int`, в большинстве случаев и оттого стоящий по умолчанию размер в 512 байт.
- `just_cory` — переменная `bool`, если значение 1 — попыток повторного чтения плохих секторов производиться не будет.
- `delete_mapfile` — `bool`-переменная, от которой зависит будет ли удален файл карты диска после работы программы.

Данная структура передается другим модулям и существует для удобства передачи введенных пользователем путей и флагов.

3.1.2 Структура `logger_info`

Эта структура имеет поля:

- `current_read_pos` — указатель `ssize_t`, данные по которому инкрементируется в модуле чтения диска, и отображаются на экране

пользователя во время исполнения программы.

- `good_block_amount` и `bad_block_amount` — указатели типа `long` на данные, обновляемые в том же модуле чтения, непосредственно считающие количество хороших и плохих секторов.

- `status` — указатель на переменную типа `bool`, отвечающая за индикацию прогресса программы, когда она еще копирует или уже приступила к повторным попыткам спасти данные с плохих секторов.

3.2 Описание модулей программы

Описанные структуры заполняются и используются в соответствующих модулях. Рассмотрим их ниже:

3.2.1 Модуль логгера `my_logger`

Этот модуль содержит структуру `logger_info` и единственную функцию `print_log()` с атрибутом `_Noreturn`, поскольку эта функция непрерывно выводит информацию структуры `logger_info` о прогрессе выполнения программы и обрабатывается в отдельном потоке. В информацию входит текущая позиция в читаемом файле, что при чтении с самого начала есть размер прочитанной информации, скорость чтения в данный момент, количество плохих и хороших секторов и время выполнения программы, а также статус спасения: копирование `Copying` или повторное чтение секторов `Rescuing`.

3.2.2 Модуль карты `my_mapper`

Модуль содержит функцию обновления файла карты, принимающий состояния секторов и текущие позиции чтения. Запись производится в текстовый файл со специальным форматированием. Идет учет времени начала и конца работы с помощью переменных типа `time_t`, полностью записывается переданная командная строка. Если новый блок имеют одинаковый статус со старым, то новая строка не записывается, вместо этого перезаписывается размер блока в предыдущую строку. Также есть отдельная функция, создающая новый файл карты и добавляющая необходимую информацию в его начало, если он не был передан изначально.

3.2.3 Модуль парсера `arguments_parser`

Здесь производится обработка командной строки, переданные директории и флаги записываются в структуру `args`. Сначала структура заполняется значениями по умолчанию в отдельной функции. Флаги передаются через «-», при некорректном вводе флагов или недоступности переданных путей будет выведена ошибка «Invalid input». После обработки модуль возвращает указатель на структуру. При вводе только одного флага «-h» без передачи путей директорий будет задействована функция `show_help()`

и выведена колонка помощи по работе с утилитой, после чего программа завершится.

3.2.4 Модуль чтения диска rescue

В данном модуле происходит чтение переданного устройства или файла и запись данных в резервный файл. Все флаги и пути читаются из передаваемой структуры `args_t`, заполненной ранее в модуле парсера флагов. Тут же создается `pthread`-поток для модуля логгера. Определяются структуры типа `timespec` для точного отсчета времени, нужного для определения скорости чтения сектора. В цикле, который повторяет действия до тех пор, пока не будет достигнута заданная позиция или конец устройства, происходит чтение сектора с замером времени до и после, на основании разницы между которым сектор помечается как плохой и его адрес записывается в массив адресов для последующей обработки. После прочитанные данные записываются в выходной файл, в зависимости от результата проверки блока обновляется карта диска, поскольку ее обновление после каждой итерации сильно бы тормозило программу. Запись и чтение происходят в отдельных функциях программы, где используются функции чтения и записи с файловыми дескрипторами.

4 РАЗРАБОТКА ПРОГРАММНЫХ МОДУЛЕЙ

4.1 Выделение ключевых процедур

Для утилиты восстановления плохих секторов ключевой процедурой является процедура чтения данных устройства и записи их в файл-резерв с последующим возвратом к плохим секторам для их повторного прочтения. Для последующей работы с диском в других утилитах понадобится карта диска, поэтому будет рассмотрена процедура заполнения файла карты.

4.1.1 Процедура инициализации файла карты

В ходе этой процедуры проводится проверка существования файла карты путем попытки открытия файла с помощью `foren()` в режиме «r+». Если функция возвращает `NULL`, то вызывается функция `write_map_header`, принимающая как изначально переданные в терминал аргументы, так уже и обработанные парсером `argv`, `argv`, `argv`.

4.1.2 Процедура заполнения файла карты

Данная процедура записывает в файл карты, открытый с использованием функции `foren()`, информацию о выполняющейся программе, о переданной командной строке, времени начала работы и текущее время, которое постоянно обновляется и впоследствии будет представлять время окончания работы. Записывается строка-заголовок для колонок с данными о секторах, а именно позиция, размер и статус сектора. Поскольку измерение времени в этой процедуре не требует высокой точности, для получения текущего времени используется функция `time()` библиотеки `time.h`.

4.1.3 Процедура обновления карты

При обновлении карты файл с именем в `char* file_name` так же открывается с `foren()`, после чего идет цикл, в ходе которого идет чтение файла карты в поисках символа «\n». С помощью этого цикла происходит переход на строку, содержащую информацию «Last time:», которая перезаписывается текущим временем. После перехода в конец файла при помощи `fseek()`, происходит проверка статуса сектора. Если `bool`-переменная нового статуса = 1, то `char` статуса — «+», в ином случае — «-».

Если новый статус блока не идентичен статусу предыдущего (`new_block_status != old_block_status`), то записывается новая строка: происходит запись позиции, на которую будет записана новая строка, в `previous_map_file_pos`, после чего используется `fprintf()`, в аргументы которого входят стартовая позиция чтения, нынешняя позиция, статус. Значения позиций переводятся в шестнадцатеричную систему счисления с соответствующей пометкой «0x» спереди.

В противном случае, если статусы одинаковы, происходит переход на

адрес `previous_map_file_pos`, после чего эта строка перезаписывается новыми данными, то есть избегается написание множества малоинформативных строк путем дозаписи нового конца блока в одну и ту же строку.

Обязательно обновляется `previous_read_pos = current_read_pos` для корректной записи впоследствии.

4.1.4 Процедуры чтения и записи

Поскольку эти процедуры близки друг к другу по коду, они рассмотрены в едином подзаголовке.

Процедура `read_block()` принимает файловый дескриптор, буффер для записи и размер для чтения, то есть все то, что и обычная функция `read()`. Отличие состоит в том, что чтение находится в цикле, который прекратится только после прочтения заданного количества байт или после неудачи чтения, когда `read()` возвратит 0. Используется на стадии копирования. Возвращается `ssize_t succ_read`.

Процедура `read_block_pos()` отличается от предыдущей только тем, что устанавливает указатель чтения/записи файла на переданную позицию `pos`, после чего вызывает уже рассмотренную функцию `read_block()`. Эта процедура используется при повторном чтении, для чего нужно устанавливать конкретные адреса для чтения, а не идти подряд, как на стадии копирования. Возвращается `ssize_t succ_read`.

Функция `write_block_pos()` идентична предыдущим функциям за исключением того, что вместо `read()` используется `write()`. У этой функции нет версии, не принимающей позицию, поскольку на стадии копирования нет гарантии, что сектора будут корректно прочитаны и можно будет производить запись в выходной файл подряд.

4.1.5 Процедура восстановления

В функции `rescue()` принимает только структуру аргументов `args_t args`. Происходит инициализация структур `timespec`, в которых есть поле `tv_nsec`, что позволяет достаточно точно измерить время чтения сектора. Позиции в входном и выходном файлах устанавливаются в `lseek()` исходя из заданных в структуре `args` параметров. Инициализируется и заполняется указателями на значения текущей позиции, количество плохих и хороших секторов, структура `logger_info` с последующим созданием потока посредством `pthread_create()`, куда передается функция `print_log()`.

Цикл `while` повторяется до тех пор, пока не будет достигнута заданная пользователем позиция или не будет прочитано все устройство до конца. В ходе цикла происходит чтение с помощью функции `read_block()`, перед и после которой фиксируется время `clock_gettime()`. Если время чтения оказалось слишком большим, то статус сектора помечается как плохой, «false», `BAD_BLOCKS_AMOUNT` инкрементируется, а позиция записывается в массив адресов `bad_block_address_array` для последующего чтения. В противном случае инкрементируется `GOOD_BLOCKS_AMOUNT`, а статус

устанавливается как «true».

В том же цикле происходит запись в выходной файл с применением `write_block_pos()`, куда передаются текущая позиция и размер прочитанных данных. Если сектор прочитался не полностью, то запись производится лишь частично, что означает, что в случае записи в непустой файл или директорию пустоты в данных будут заполнены прошлыми данными. Потом происходит обновление позиций, проверка на надобность обновления карты: если статусы старого и нового блока разнятся, то происходит обновление, в противном случае обновление происходит раз в 500 секторов.

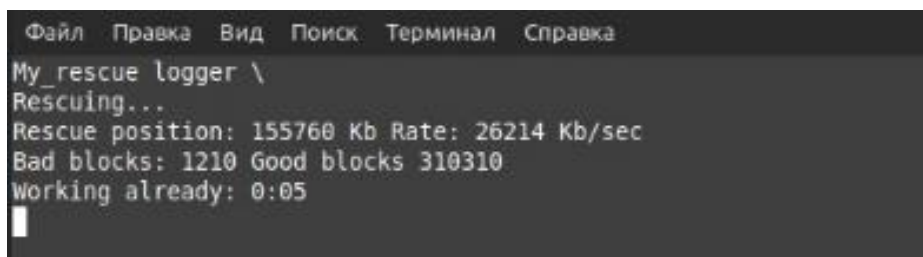
По окончании цикла проводится проверка флага `args->just_copy`. Если значение «0», то происходят попытки прочитать плохие сектора в цикле обхода массива `bad_block_address_array`. Каждый сектор читается `args->bad_block_repeat` раз, по умолчанию 3 раза.

По окончании функции поток логгера закрывается `pthread_cancel()` и `pthread_join()`.

5 ПРОГРАММА И МЕТОДИКА ИСПЫТАНИЙ

Функционал программы несложно проверить с помощью флеш-накопителя небольшой емкости или любого файла небольшого размера, чтобы программа не читала информацию слишком долго.

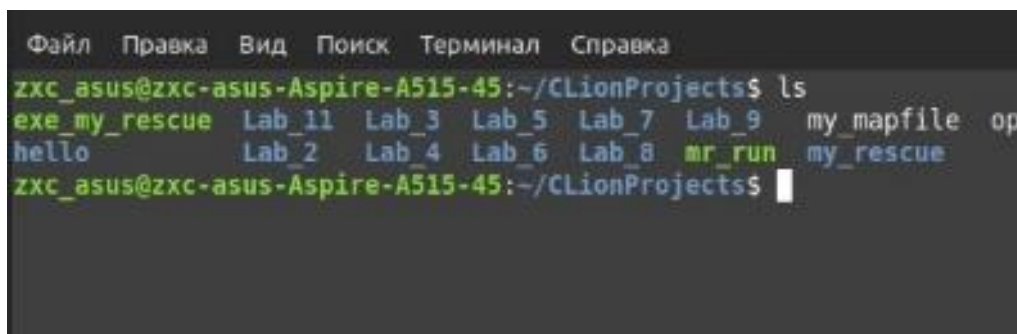
Запуск программы со следующей командной строкой: `sudo ./exe_my_rescue /dev/sda1 op`. Sda1 — это имя флеш-накопителя, с помощью которого осуществлялось тестирование, op — имя выходного файла. Настройки соответствуют чтению от начала до конца.



```
Файл  Правка  Вид  Поиск  Терминал  Справка
My_rescue logger \
Rescuing...
Rescue position: 155760 Kb Rate: 26214 Kb/sec
Bad blocks: 1210 Good blocks 310310
Working already: 0:05
```

Рисунок 5.1 — логгер во время исполнения программы

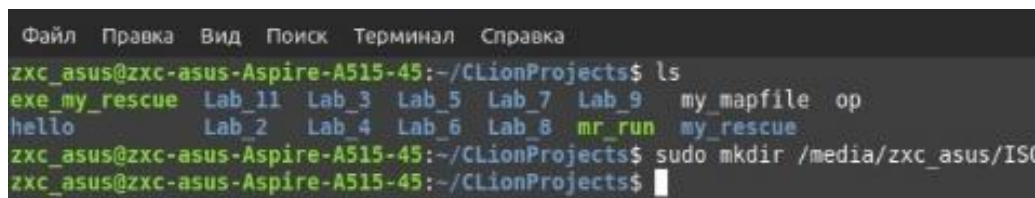
После работы программы по указанным директориям появляются файлы карты диска и его образа, в нашем случае это `my_mapfile` и `op`.



```
Файл  Правка  Вид  Поиск  Терминал  Справка
zxc_asus@zxc-asus-Aspire-A515-45:~/CLionProjects$ ls
exe_my_rescue  Lab_11  Lab_3  Lab_5  Lab_7  Lab_9  my_mapfile  op
hello         Lab_2   Lab_4  Lab_6  Lab_8  mr_run  my_rescue
```

Рисунок 5.2 — выходные файлы после выполнения

Нужно смонтировать диск. Для этого требуется создать отдельный раздел.



```
Файл  Правка  Вид  Поиск  Терминал  Справка
zxc_asus@zxc-asus-Aspire-A515-45:~/CLionProjects$ ls
exe_my_rescue  Lab_11  Lab_3  Lab_5  Lab_7  Lab_9  my_mapfile  op
hello         Lab_2   Lab_4  Lab_6  Lab_8  mr_run  my_rescue
zxc_asus@zxc-asus-Aspire-A515-45:~/CLionProjects$ sudo mkdir /media/zxc_asus/ISO
zxc_asus@zxc-asus-Aspire-A515-45:~/CLionProjects$
```

Рисунок 5.3 — выделение раздела для монтирования

Теперь требуется смонтировать выходной файл. Для этого нужно знать файловую систему устройства, которое читала утилита. Узнать ее можно с помощью команды `fdisk`, подробнее в руководстве пользователя. Для тестового устройства файловой системой является NTFS.

```
Файл  Правка  Вид  Поиск  Терминал  Справка
zxc_asus@zxc-asus-Aspire-A515-45:~/CLionProjects$ ls
exe_my_rescue  Lab_11  Lab_3  Lab_5  Lab_7  Lab_9  my_mapfile  op
hello          Lab_2   Lab_4   Lab_6   Lab_8  mr_run  my_rescue
zxc_asus@zxc-asus-Aspire-A515-45:~/CLionProjects$ sudo mkdir /media/zxc_asus/ISO
zxc_asus@zxc-asus-Aspire-A515-45:~/CLionProjects$ sudo mount.ntfs op /media/zxc_
asus/ISO/
zxc_asus@zxc-asus-Aspire-A515-45:~/CLionProjects$
```

Рисунок 5.4 — монтирование диска

Диск смонтирован и можно рассмотреть результат.

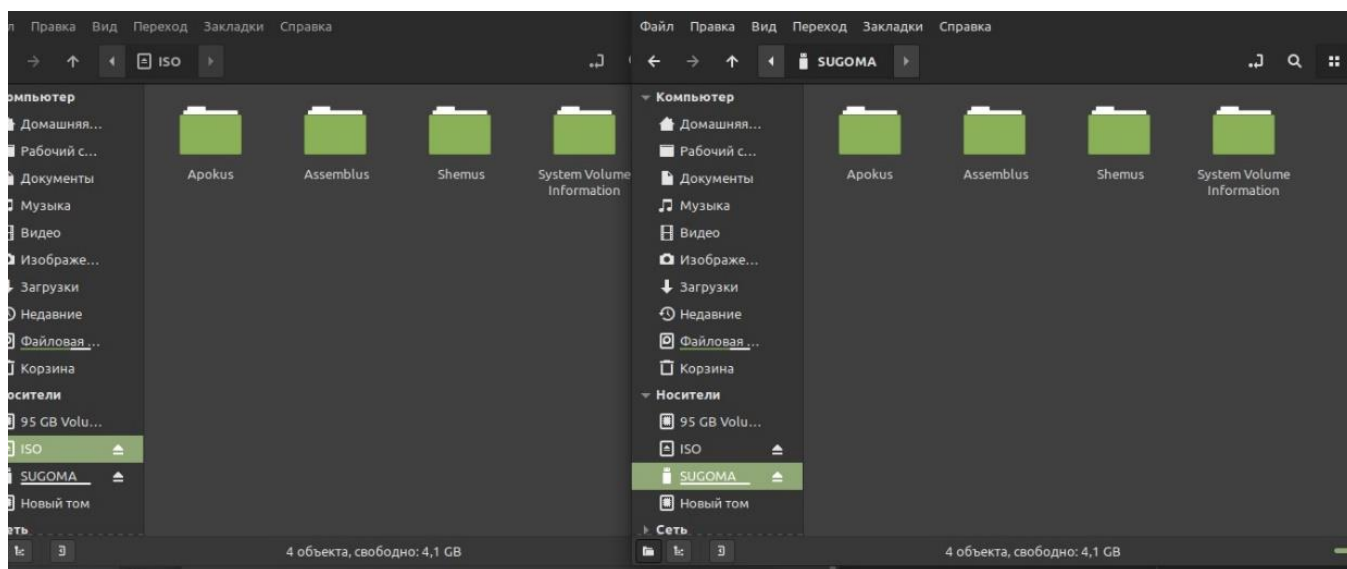
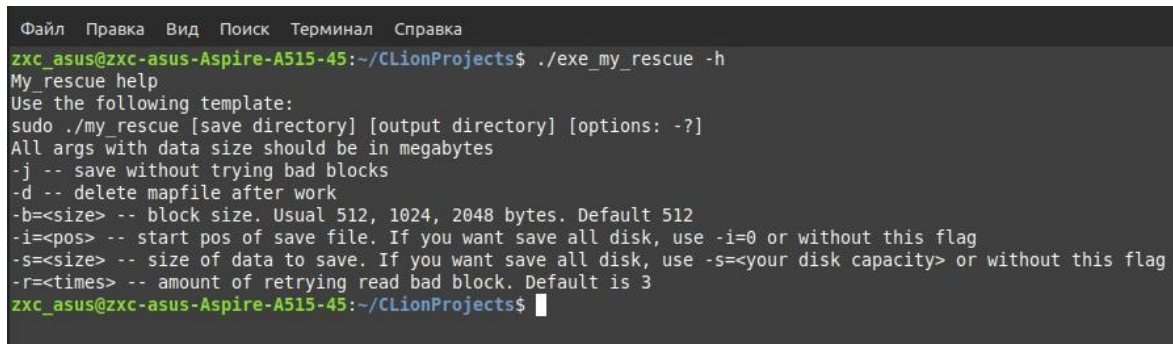


Рисунок 5.5 — проверка результата

Содержание директорий идентично, перенесенные в директорию «ISO» файлы открываются без ошибок и содержат изначальную информацию. Перенос совершен корректно.

6 РУКОВОДСТВО ПОЛЬЗОВАТЕЛЯ

Несмотря на то, что эта утилита не содержит того обилия флагов GNU ddrescue, у пользователя есть возможность использовать флаг «-h», активирующий функцию в парсере и выводящий окно помощи с информацией о работе утилиты.

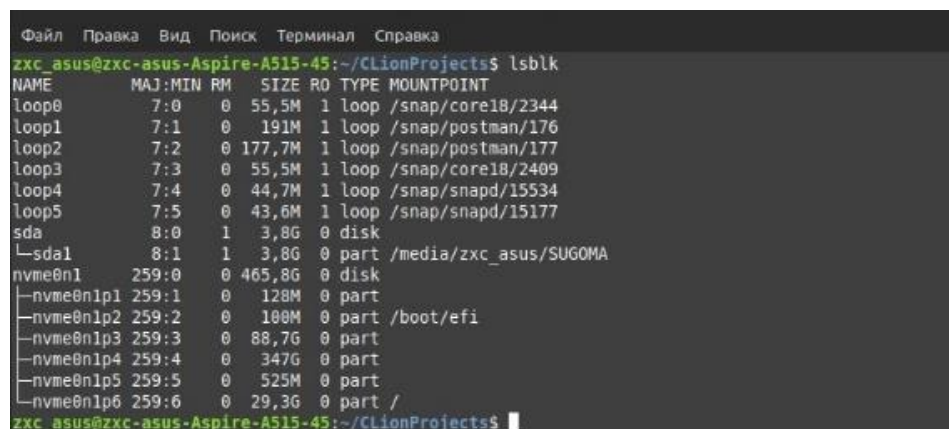


```
Файл  Правка  Вид  Поиск  Терминал  Справка
zxc_asus@zxc-asus-Aspire-A515-45:~/CLionProjects$ ./exe_my_rescue -h
My_rescue help
Use the following template:
sudo ./my_rescue [save directory] [output directory] [options: -?]
All args with data size should be in megabytes
-j -- save without trying bad blocks
-d -- delete mapfile after work
-b=<size> -- block size. Usual 512, 1024, 2048 bytes. Default 512
-i=<pos> -- start pos of save file. If you want save all disk, use -i=0 or without this flag
-s=<size> -- size of data to save. If you want save all disk, use -s=<your disk capacity> or without this flag
-r=<times> -- amount of retrying read bad block. Default is 3
zxc_asus@zxc-asus-Aspire-A515-45:~/CLionProjects$
```

Рисунок 6.1 — окно помощи

Эта страница показывает допустимые флаги и требуемый шаблон командной строки для корректной работы.

Для последующего монтирования диска нужно узнать название спасаемого устройства и его файловую систему. Это можно осуществить с помощью команд lsblk или fdisk -l.



```
Файл  Правка  Вид  Поиск  Терминал  Справка
zxc_asus@zxc-asus-Aspire-A515-45:~/CLionProjects$ lsblk
NAME        MAJ:MIN RM  SIZE RO TYPE MOUNTPOINT
loop0        7:0      0   55,5M  1 loop /snap/core18/2344
loop1        7:1      0   191M   1 loop /snap/postman/176
loop2        7:2      0  177,7M  1 loop /snap/postman/177
loop3        7:3      0   55,5M  1 loop /snap/core18/2409
loop4        7:4      0   44,7M  1 loop /snap/snapd/15534
loop5        7:5      0   43,6M  1 loop /snap/snapd/15177
sda          8:0      1    3,8G  0 disk
└─sda1       8:1      1    3,8G  0 part /media/zxc_asus/SUGOMA
nvme0n1     259:0    0  465,8G  0 disk
├─nvme0n1p1 259:1    0   128M  0 part
├─nvme0n1p2 259:2    0   100M  0 part /boot/efi
├─nvme0n1p3 259:3    0    88,7G  0 part
├─nvme0n1p4 259:4    0   347G  0 part
├─nvme0n1p5 259:5    0   525M  0 part
└─nvme0n1p6 259:6    0    29,3G  0 part /
zxc_asus@zxc-asus-Aspire-A515-45:~/CLionProjects$
```

Рисунок 6.2 — результат работы lsblk

Самая левая колонка, колонка имени, содержит название устройства. Например, тут есть название тестового устройства sda1.

При применении команды fdisk с флагом «-l» дает результат, представленный на рисунке 6.3. Внешние устройства располагаются снизу отчета, конкретно на рисунке 6.3 можно видеть устройство sda1 с файловой системой NTFS.


```
Файл  Правка  Вид  Поиск  Терминал  Справка
/dev/nvme0n1p4 247887872 975697919 727810048 347G Microsoft basic data
/dev/nvme0n1p5 975697920 976773119 1075200 525M Windows recovery environment
/dev/nvme0n1p6 186447872 247887871 61440000 29,3G Linux filesystem

Partition table entries are not in disk order.

Disk /dev/sda: 3,83 GiB, 4089446400 bytes, 7987200 sectors
Disk model: Flash Disk
Units: sectors of 1 * 512 = 512 bytes
Sector size (logical/physical): 512 bytes / 512 bytes
I/O size (minimum/optimal): 512 bytes / 512 bytes
Disklabel type: dos
Disk identifier: 0xb7552050

Device      Boot Start      End Sectors  Size Id Type
/dev/sda1                128 7983231 7983104   3,8G  7 HPFS/NTFS/exFAT
zxc_asus@zxc-asus-Aspire-A515-45:~/CLionProjects$ sudo fdisk -l
```

Рисунок 6.3 — результат работы fdisk

ЗАКЛЮЧЕНИЕ

В ходе выполнения курсовой работы были получены и закреплены знания о устройстве POSIX-совместимых файловых систем, работе с файлом при помощи файлового дескриптора, структурах, отвечающих за характеристики накопителей данных и другие знания в области системного программирования. При помощи этих знаний удалось создать утилиту восстановления плохих секторов путем многократного чтения. С помощью знаний о многопоточности удалось выделить отдельный поток чтения информации о прогрессе исполнения.

Программа расположена к внесению изменений в алгоритм восстановления плохих секторов, код располагается на [github](#) и любой может предложить изменения или запросить `fork` для улучшения утилиты.

Данный курсовой проект дал мне ценный опыт в области написания самодостаточных программ в целом и в POSIX-системах в частности. Следует тщательно продумать весь путь создания проекта сразу, предварительно изучив теоретический материал по техническому заданию.

Литература

1. Документация команды GNU ddrescue – [Электронный ресурс]. – Адрес ресурса:
https://www.gnu.org/software/ddrescue/manual/ddrescue_manual.html -
Дата доступа 12.05.2022
2. Лав Р. Системное программирование на Linux/ 2-е издание 2014.
3. Луцик Ю.А. Объектно-ориентированное программирование на языке C++ / Ю.А.Луцик, А.М.Ковальчук, И.В.Лукьянова. – Мн.: БГУИР, 2003

ПРИЛОЖЕНИЕ А

(Обязательное)

Структурная схема приложения

ПРИЛОЖЕНИЕ Б

(Обязательное)

Блок-схема алгоритма обновления карты