Министерство образования Республики Беларусь
Учреждение образования «Белорусский государственный университет информатики и радиоэлектроники»

Факультет компьютерных систем и сетей
Кафедра электронных вычислительных машин
Дисциплина: Операционные системы и системное программирование

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА
к курсовому проекту
на тему

«УТИЛИТА ВОССТАНОВЛЕНИЯ ПЛОХИХ СЕКТОРОВ (МНОГОКРАТНОЕ ЧТЕНИЕ)»

БГУИР КП 1-40 02 01 112 ПЗ

Студент: гр. 150501 Ковальчук Д.И.

Руководитель: старший преподаватель
каф. ЭВМ Поденок Л. П.

Минск 2023

Учреждение образования
«Белорусский государственный университет информатики
и радиоэлектроники»

Факультет компьютерных систем и сетей

УТВЕРЖДАЮ
Заведующий кафедрой ЭВМ

_____
(подпись)
_____2023 г.

ЗАДАНИЕ
по курсовому проектированию

Студенту *Ковальчук Дарье Ивановна*

1. Тема проекта *“Утилита восстановления плохих секторов (многократное чтение)”*
2. Срок сдачи студентом законченного проекта *17 мая 2023 г.*
3. Исходные данные к проекту *Язык программирования – C*
4. Содержание расчетно-пояснительной записки (перечень вопросов, которые подлежат разработке)
1. Введение.
2. Обзор литературы.
3. Постановка задачи.
4. Системное проектирование.
5. Функциональное проектирование.
6. Разработка программных модулей.
7. Результаты работы.
8. Заключение
9. Литература
5. Перечень графического материала (с точным обозначением обязательных чертежей и графиков)
1. Схема алгоритма функции `analyze_cl()`
2. Схема алгоритма функции `signalhandler()`
6. Консультант по проекту *Поденок Л.П.*
7. Дата выдачи задания *24 февраля 2023 г.*
8. Календарный график работы над проектом на весь период проектирования (с обозначением сроков выполнения и трудоемкости отдельных этапов):
Выбор задания. Разработка содержания пояснительной записки. Перечень графического материала – 15%;
разделы 2, 3 – 10%;
разделы 4 – 20%;

разделы 5 – 35%;

раздел 6,7,8 – 5%;

раздел 9 – 5%;

оформление пояснительной записки и графического материала – 10%

Защита курсового проекта с 29.05 по 09.06.23г.

РУКОВОДИТЕЛЬ _____ Л. П. Поденок

(подпись)

Задание принял к исполнению _____ Д.И. Ковальчук

(дата и подпись студента)

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

Несмотря на то, что HDD сегодня потихоньку отходят на второй план и активно заменяются быстрыми твердотельными SSD, HDD все еще имеют преимущество в ценовой категории и являются довольно популярными среди пользователей, поэтому нет смысла говорить о бесперспективности их поддержки.

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 неисправного диска на исправный путем использования восстанавливающих программ и утилит.

Устройства хранения данных не имеют привязки к какой-либо конкретной файловой системе и, следовательно поэтому, их восстановление не касается структуры файлов и операционной системы 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.

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

1 ОБЗОР ЛИТЕРАТУРЫ

1.1 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, а также просто собрать детальную информацию о незагружающихся дисках, чтобы применить ее при дальнейшем анализе. Очень часто именно ее рекомендуют использовать вместо или с GNU ddrescue.

GNU ddrescue — утилита для UNIX-подобных систем, которая позволяет выбрать стратегию чтения и умеет сроить 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 прочитать, при этом перенося данные на диск для хранения спасаемых данных, а также запоминает неисправные сектора, для последующего их спасения. Утилита написана на языке C++.

1.2 Постановка задачи

Программа должна быть разработана для работы на операционных системах семейства Linux с учётом стандарта C11 языка C и иметь удобный пользовательский интерфейс.

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

- перенос данных на внешний носитель;
- учет плохих секторов;
- возможность попытки спасти информацию с плохих секторов.

2 СИСТЕМНОЕ ПРОЕКТИРОВАНИЕ

После определения требований к функционалу разрабатываемого приложения его следует разбить на функциональные блоки. Такой подход упростит понимание проекта, позволит устранить проблемы в архитектуре, обеспечит гибкость программного продукта в будущем путем добавления новых блоков.

2.1 Модуль анализа командной строки

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 каталогов для хранения спасенной информации.

2.2 Модуль обработки сигнала досрочного завершения программы

Модуль представляет собой обработчик сигнала нажатия CTRL-C. Наличие такого обработчика обусловлено тем, что при возникшей необходимости досрочно завершить выполнение, файлы, с которыми работала программа корректно закрылись, а также очистились используемые буферы.

2.3 Модуль чтения диска

Модуль отвечает за чтение и обработку данных диска. Здесь производится основная часть программы: чтение диска и, при обнаружении поврежденного сектора, многократное чтение. В нем же располагаются переменные, которые принимает модуль логгера для вывода на экран.

2.4 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.

3 ФУНКЦИОНАЛЬНОЕ ПРОЕКТИРОВАНИЕ

В данном разделе описываются структуры, которые были использованы при разработке утилиты.

3.1 Структура `args_t`

Первая структура представляет собой хранилище обработанных модулем анализа командной строки директ флагов, введенных пользователем при запуске программы.

Структура `args_t` включает в себя следующие поля:

`FILE *src` – исходный файл;

`FILE *dst` – конечный файл;

`long int retries` – количество попыток многократного чтения;

long int sector_size – размер сектора;

`long int seek` – позиция, с которой производить чтение.

`long int limit` – позиция, до которой производить чтение.

`long int src_size` - размер исходного файла.

`long int max_bad` – максимальное количество плохих секторов, при достижении которого программа перестанет пытаться производить спасение далее.

3.2 Структура `logger_data`

Данная структура содержит указатели на переменные, отображающие прогресс выполнения программы.

Структура `logger_data` включает в себя следующие поля:

`long int *current_pointer` – текущий указатель в исходном файле;

`long int amount_of_bad` – общий размер плохих секторов плохих секторов;

`long int acquired` – размер успешно спасенных данных;

`float *progress` – процент считанных в исходном файле данных;

`bool *status` – индикатор этапа выполнения.

4 РАЗРАБОТКА ПРОГРАММНЫХ МОДУЛЕЙ

В данном разделе рассмотрены алгоритмы работы четырёх функций: функции анализа введенной пользователем командной строки `analyze_cl()`, функции обработчика сигнала `signalhandler()`, функции, в которой происходит многократное чтение `case_of_bad_block()` и функции поиска числа при обработке опций, введенных пользователем `while_not_space()`.

4.1 Разработка схем алгоритмов

Схема алгоритма функции, которая анализирует введенную пользователем командную строку и заполняет структуру флагов, `analyze_cl()` приведена в приложении А.

Схема алгоритма функции обработчика сигнала, которая вызывается при досрочном завершении пользователем программы `signalhandler()` приведена в приложении Б.

4.2 Разработка алгоритмов

Функция case_of_bad_block():

1) начало;
2) входные данные:

`args_t *args` – указатель на структуру флагов;

`long *cur_pointer` – указатель на текущую позицию в исходном файле;

выходные данные: `bool success` – переменная, показывающая успешно ли прошло многократное чтение;

3) объявление переменных char tmp[2*BUFFSZ], long retry, size_t slen;
4) начало цикла с параметром `retry` и декрементированием параметра при каждой итерации, пока `retry` больше 0;
5) установка указателя: fseek(args->src,*cur_pointer,SEEK_SET);
6) обнуление массива `tmp`;
7) Попытка повторного чтения данного сектора:
 slen = fread(tmp,1,args->sector_size, args->src);
8) если `slen` равно `args->sector_size`, переход на шаг 8, иначе переход на шаг 10;
9) запись данных в конечный файл: `fwrite(tmp,1,slen,args->dst);` переход на шаг 12;
10) уменьшение текущего указателя на размер считанных данных: *cur_pointer-=slen;
11) конец цикла параметром `retry;`
12) конец.

Функция while_not_space():
1) начало;
2) входные данные:
`char* command_line` – командная строка;
`int pos` – начальная позиция в командной строке;
выходные данные: char *tmp – строка с результатом;
3) объявление переменных `char* tmp_str` – результирующая строка, `int i` – переменная для цикла;
4) начало цикла до встречи пробела в командной строке;
5) если `str[pos] == ' \0'`, переход на шаг 11, иначе переход на шаг 6;
6) если `str[pos] < '0' || str[pos] > '9'`, вывод сообщения об ошибке и выход из программы, иначе переход на шаг 7;
7) копирования символа: `tmp_str[i] = str[pos];`
8) инкрементирование переменных `pos` и `i`;
9) конец цикла;
10) добавление символа конца строки в полученный массив символов: tmp_str[i] = '\0';
11) конец.

5 РЕЗУЛЬТАТЫ РАБОТЫ

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

Если пользователь количество параметров командной строки менее трех, он увидит на экране окно помощи, в котором дан шаблон для запуска программы и описание возможных опций.

Для дальнейшей работы нужно смонтировать диск. Для этого требуется создать отдельный раздел. Пример: `sudo mkdir /media/darya/ISO`.

Для последующего монтирования диска нужно узнать название спасаемого устройство и его файловую систему. Это можно осуществить с помощью команды `fdisk –l`. Для тестового устройства файловой системой является `FAT32`.

Для проверки программы производится запуск программы со следующей командной строкой: `sudo ./myrescue /dev/sda1 myres`. `Dev/sda1` — это имя флеш-накопителя, с помощью которого осуществлялось тестирование, `myres` — имя выходного файла. Настройки соответствуют чтению от начала до конца.

Терминал во время выполнения программы:

```
РАБОТАЕТ ЛОГГЕР /
ВЫПОЛНЕНИЕ
ТЕКУЩИЙ УКАЗАТЕЛЬ: 46032576
ПЛОХИХ СЕКТОРОВ: 0
ХОРОШИХ СЕКТОРОВ: 46032576
ПРОГРЕСС: 11.432397
ВРЕМЯ РАБОТЫ: 0:38
```

После завершения работы программы требуется смонтировать выходной файл. Для этого нужно знать файловую систему устройства, которое читала утилита. Монтирование диска производится командой `mount` с указанием опции `-t` в соответствии с файловой системой: `sudo mount -t vfat myres /media/darya/ISO`.

После сравнения исходной и полученной директорий видно, что они абсолютно идентичны. Результат представлен на рисунке 4.1.

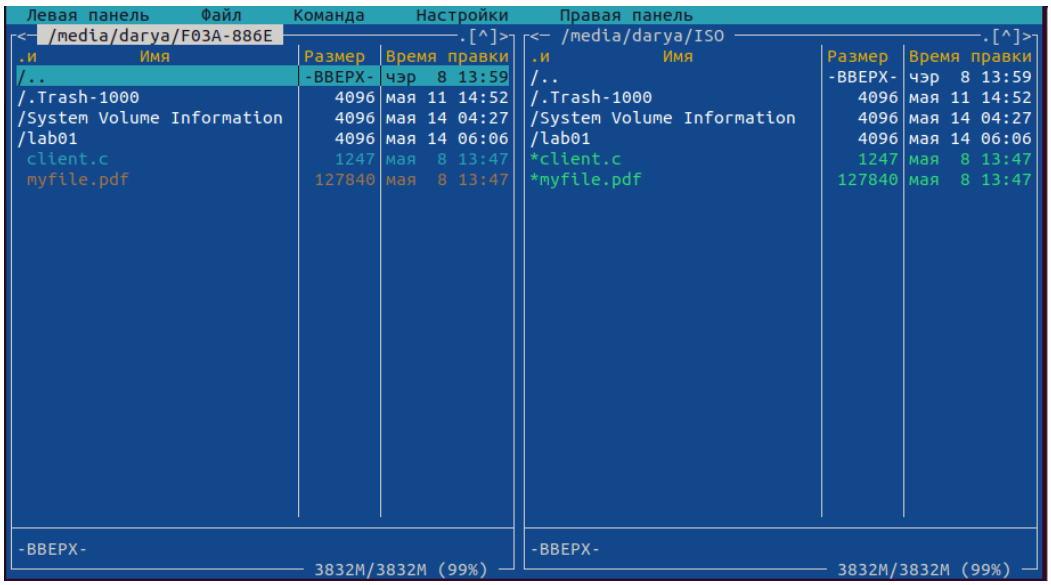

Рисунок 4.1 – Сравнение исходного и полученного дисков.

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

ЗАКЛЮЧЕНИЕ

В ходе выполнения курсовой работы были углублены знания о языке программирования Си, получены и закреплены знания об устройстве POSIX-совместимых файловых систем, работе с файлом при помощи файлового дескриптора, сигналах и другие знания в области системного программирования.

В ходе выполнения курсовой работы была реализована утилита восстановления плохих секторов путем многократного чтения.

Тем не менее, программа определённо имеет потенциал к улучшению, так как данный курсовой проект рассчитан на студентов второго курса.

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.

СПИСОК ИСПОЛЬЗУЕМЫХ ИСТОЧНИКОВ

[1] Керниган Б., Ритчи Д. Язык программирования Си. Издательство: «Вильямс», 2019.

[2] Лав Р. Linux. Системное программирование. 2-е изд. — СПб.: Питер, 2014.

[3] Робачевский А. Программирование на языке Си в UNIX. Издательство: БХВ-Петербург, 2015.

[4] Стивенс Р., Раго С. UNIX. Профессиональное программирование, 2-е издание. — СПб.: Символ-Плюс, 2007.

[5] Вычислительные машины, системы и сети: дипломное проектирование (методическое пособие) [Электронный ресурс]. – Минск, БГУИР 2019.

ПРИЛОЖЕНИЕ А

(обязательное)

Схема алгоритма функции `analyze_cl( )`

ПРИЛОЖЕНИЕ Б

(обязательное)

Схема алгоритма функции `signalhandler( )`

ПРИЛОЖЕНИЕ В

(обязательное)

Ведомость документов

ГУИР.400201.112 ПД1

					ГУИР.400201.112 ПД1	
Изм	Лист	№ документа	Подпись	Дата	Схема функции analyze_cl()	Лит. У / Масса / Масштаб
Разраб.		Ковальчук				Лист / Листов 1
Пров.		Поденок				Кафедра ЭВМ гр. 150501

ГУИР.400201.112 ПД2

					ГУИР.400201.112 ПД2
Изм	Лист	№ документа	Подпись	Дата	Схема функции signalhandler()
Разраб.		Ковальчук			Лит. У / Масса / Масштаб
Пров.		Поденок			Лист / Листов 1
					Кафедра ЭВМ гр. 150501

Обозначение	Наименование	Примечание
	Текстовые документы	
ГУИР КП 1-40 02 01 112 ПЗ	Пояснительная записка	20 с.
	Графические документы	
ГУИР.400201.112 ПД1	Схема функции analyze_cl()	А4
ГУИР.400201.112 ПД2	Схема функции signalhandler()	А4
ГУИР.400201.112-01 12	Текст программы	

Изм	Лист	№ документа	Подпись	Дата	БГУИР КП 1-40 02 01 112 Д1		
Разраб.		Ковальчук			Утилита восстановления плохих секторов (многократное чтение) Ведомость документов	Лит.	Лист	Листов
Пров.		Поденок				Т		1
						Кафедра ЭВМ гр. 150501		