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

«Корзина для программ, использующих системный вызов unlink()»

БГУИР КП 1-40 02 01 116 ПЗ

Студент: гр. 350501 Маслаков Н.А.

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

Минск 2025

Учреждение образования
«Белорусский государственный университет информатики
и радиоэлектроники»

Факультет компьютерных систем и сетей

УТВЕРЖДАЮ
Заведующий кафедрой ЭВМ
(подпись)_____2025 г.

ЗАДАНИЕ
по курсовому проектированию студенту
Римашевскому Захару Дмитриевичу

1. Тема проекта: “Корзина для программ, использующих системный вызов unlink()”.
2. Срок сдачи студентом законченного проекта — 15.05.2025 г.
3. Исходные данные к проекту: Язык программирования - C.
4. Содержание расчетно-пояснительной записки (перечень вопросов, которые подлежат разработке):
Введение. 1. Обзор методов и алгоритмов решения постановленной задачи. 2. Обоснование выбранных методов и алгоритмов. 3. Описание программы для программиста 4. Описание алгоритмов решения задачи. 5. Руководство пользователя. Заключение. Список использованных источников. Приложения.
5. Перечень графического материала (с точным обозначением обязательных чертежей и графиков):
    1. Схема алгоритмов работы функции;
    2. Скриншоты работы программы;
    3. Ведомость документа.
6. Консультант по проекту (с обозначение разделов проекта) Поденок Л.П.
7. Календарный график работы над проектом на весь период проектирования (с обозначением сроков выполнения и трудоемкости отдельных этапов):
    раздел 1 к 01.03. - 15%;
    раздел 2, 3 к 01.04. - 50%;
    раздел 4, 5 к 01.05. - 80%;
    оформление пояснительной записки и графического материала к 15.05.2025 - 100%;
    защита курсового проекта с 29.05 по 09.06.

РУКОВОДИТЕЛЬ (подпись)_____Л. П. Поденок

Задание принял к исполнению Н. А. Маслаков

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

Удаление файлов с помощью системных вызовов `unlink()` и `unlinkat()` является стандартным механизмом в Unix-подобных операционных системах. Однако, в отличие от графических окружений, где удалённые объекты сначала помещаются в «Корзину» и могут быть восстановлены, при прямом вызове `unlink()` файл тут же исчезает из файловой системы без возможности возврата. Это ограничивает удобство и безопасность работы пользователей, особенно в тех сценариях, когда случайное или преждевременное удаление данных может привести к серьёзным потерям.

В рамках данного курсового проекта была поставлена задача создания обобщённого «Корзина»-механизма, не требующего модификации исходного кода ни одной из пользовательских программ. Центральным элементом решения является библиотека-перехватчик, загружаемая через переменную окружения `LD_PRELOAD`. Она переопределяет вызовы `unlink()` и `unlinkat()`, перехватывает запросы на удаление и, вместо непосредственного удаления, перемещает файл в специально организованную структуру «Корзины». Файлы сохраняются под уникальными именами, а для каждого из них создаётся небольшой текстовый «метаданные»-файл, содержащий исходный путь, время удаления и размер.

Параллельно с этим реализуется утилита командной строки `trashctl`, позволяющая просматривать содержимое «Корзины», восстанавливать удалённые файлы и окончательно очищать её. Более того, для повышения удобства работы разработан терминальный «проводник» на базе библиотеки ncurses, предоставляющий двухпанельный интерфейс с навигацией по каталогам, детальной информацией о выделённом объекте и быстродействующим масштабированием при изменении размера окна терминала.

Особенностью данной системы является её максимальная прозрачность для пользовательских программ: любую операцию удаления, будь то стандартная команда `rm`, действие встроенного файлового менеджера или стороннего бинара, можно направить через «Корзину», просто установив переменную `LD_PRELOAD`. При этом сама утилита `trashctl` и встроенный в неё терминальный «проводник» обеспечивают полный контроль над удалёнными объектами, сводя к нулю риск потерять важные данные из-за случайного удаления.

1 ОБЗОР МЕТОДОВ И АЛГОРИТМОВ РЕШЕНИЯ ПОСТАНОВЛЕННОЙ ЗАДАЧИ

1.1 Перехват системных вызовов через LD_PRELOAD

LD_PRELOAD — механизм динамической загрузки библиотеки перед всеми другими при запуске процесса. Путём создания собственной `libtrash.so`, которая экспортирует символы `unlink` и `unlinkat`, мы можем перехватывать эти вызовы, выполнять логику перемещения файла в корзину и лишь при необходимости вызывать оригинальную функцию. Такой подход не требует правки или перекомпиляции удаляемых программ и работает для любых бинарников, где LD_PRELOAD разрешён.

1.2 Структура корзины и метаданных

Для сохранения удалённых данных создаётся каталог `$TRASH_BASE/.trash`, внутри которого имеются две поддиректории: `files/` - сами файлы под уникальными именами `<timestamp>_<rand>[.ext]`, и `info/` - текстовые файлы `<id>.info`, где хранятся `original_path`, `deleted_time` (ISO 8601) и `size`. Такой раздел позволяет быстро сканировать корзину, восстанавливать файл по содержимому `.info` и «чистить» корзину без потерь.

1.3 Консольная утилита управления и терминальный “проводник”

Утилита `trashctl` обеспечивает три базовые команды:

1) `list` - вывод таблицы записей корзины (ID, время, размер, оригинальный путь).

2) `restore <ID>` - возвращает файл из `files/` в исходное место и удаляет `.info`.

3) `purge <ID|--all>` - навсегда удаляет объект(ы).

Дополнительно реализован режим `trashctl browse`, терминальный файловый менеджер на базе ncurses с двумя панелями: файловая навигация и подробная информация по выделенному элементу. Удаление через клавишу Delete вызывает системный `unlink`, перехваченный LD_PRELOAD, что упрощает согласованность логики.

2 ОБОСНОВАНИЕ ВЫБРАННЫХ МЕТОДОВ И АЛГОРИТМОВ РАЗРАБОТКИ

2.1 LD_PRELOAD vs прокси-библиотеки/патчинг ядра

Использование механизма LD_PRELOAD позволяет подключить свою библиотеку раньше всех остальных при запуске любого процесса, фактически «подменив» реализацию системных вызовов `unlink` и `unlinkat` без

переписывания или перекомпиляции целевых программ. Это даёт несколько ключевых преимуществ:

1) Минимальная инвазивность. Любой существующий бинарник (включая закрытый проприетарный) будет работать «из коробки» — достаточно установить переменную окружения, и его удаления начнут попадать в нашу «Корзину».

2) Нет правки исходников. Не нужно вносить изменения в код любых приложений и пересобирать их.

3) Гибкость. Легко отключать/включать перехват, просто убирая или добавляя `LD_PRELOAD`.

Альтернативой является написание прокси-библиотеки, которую программы явно должны линкуют вместо стандартной libc, или использование патчинга ядра (изменение кода ядра или загрузка модуля), чтобы перехватывать вызовы на уровне ядра. Оба этих подхода значительно сложнее:

- При прокси-библиотеке каждый исполняемый файл нужно пересобирать или линкувать заново, что невозможно для закрытых бинарников.

- Патчинг ядра требует прав `root`, может нарушить безопасность системы и усложняет переносимость решения между разными версиями ядра.

Таким образом, LD_PRELOAD — оптимальный компромисс между невидимостью для пользователя, простотой развёртывания и универсальностью.

2.2 Формат метаданных (INI-подобный)

INI-подобный формат прост для ручного чтения и парсинга на C. Каждая запись занимает три строки и не требует внешних библиотек. Это облегчает написание кода CLI и терминального проводника, минимизирует вероятность ошибок при обработке.

2.3 Выбор ncurses для терминального интерфейса

Для создания интерактивного текстового интерфейса был выбран фреймворк ncurses, что обусловлено следующими соображениями:

1) Кросс-платформенность. ncurses стандартно доступен в большинстве дистрибутивов Linux и BSD, требует лишь базовой поддержки терминала.

2) Обработка ввода–вывода. Библиотека предоставляет готовые механизмы для работы со стрелками, функциональными клавишами, мышью и сигналом `SIGWINCH`, без ручного разбора escape-последовательностей.

3) Удобство рисования окон. С помощью `newwin()`, `box()`, `wrefresh()` и цветовых пар можно быстро собирать двухпанельный интерфейс, разделять области экрана и рисовать строки подсказок.

4) Сообщество и документация. ncurses хорошо задокументирован, имеет множество примеров и устоявшихся приёмов работы, что ускоряет разработку и упрощает отладку.

Хотя существуют альтернативы (прямые вызовы ioctl или библиотеки высокого уровня на Python/Go), ncurses предоставляет оптимальное сочетание производительности, контролируемости и низкого уровня зависимостей, что соответствует целям проекта.

3 ОПИСАНИЕ ПРОГРАММЫ ДЛЯ ПРОГРАММИСТА

3.1 Общая архитектура программы

Проект разделён на две основные части: перехватчик системных вызовов и утилиту управления. Перехватчик скомпонован в виде динамической библиотеки `libtrash.so`, расположенной в папке `src/interceptor`. Его задача — незаметно для конечной программы переопределить функции `unlink()` и `unlinkat()`, перенаправив удаление в «Корзину». В каталоге `src/cli` находится реализация утилиты `trashctl`, включающая в себя три команды (`list`, `restore`, `purge`) и терминальный режим `browse`, предоставляющий дружественный интерфейс на основе ncurses. Общая файловая структура проекта позволяет поддерживать несколько конфигураций сборки (debug/ release) и легко дополнить систему новыми модулями или интеграциями.

3.2 Перехватчик системных вызовов

Перехватчик расположен в файле `trash_unlink.c`. При загрузке через `LD_PRELOAD` он перехватывает вызовы `unlink()` и `unlinkat()`. В функции `unlink()` сначала формируется абсолютный путь к удаляемому файлу, затем проверяется, не находится ли он уже в каталоге «Корзины». Если это не так, генерируется уникальный идентификатор файла, создаются (при необходимости) директории `files/` и `info/` внутри `TRASH_BASE/.trash`, выполняется переименование оригинального файла в подкаталог `files/`, после чего записывается текстовый метаданные-файл в `info/` с указанием исходного пути, времени удаления в формате ISO 8601 и размера файла. При ошибке перемещения или если файл уже находится в корзине, происходит вызов реального `unlink()`. Аналогичная логика реализована в `unlinkat()`, где перед перемещением файл приводится к абсолютному пути через `getcwd()` или чтение `/proc/self/fd`.

3.3 Утилита управления

Утилита командной строки состоит из нескольких модулей. В файле `main.c` производится анализ аргументов и маршрутизация на подкоманды. Модуль `list.c` открывает папку `info/`, последовательно читает все `.info`-файлы и парсит их содержимое, выводя таблицу с колонками: идентификатор

удаления, время, размер и оригинальный путь. Модуль `restore.c` принимает идентификатор, читает соответствующий `.info`, проверяет доступность каталога назначения, осуществляет переименование из `files/` в исходное местоположение и удаляет файл метаданных. Модуль `purge.c` по идентификатору или флагу `--all` удаляет из `files/` и `info/` указанные записи безвозвратно. Все эти операции реализованы на чистом C с минимальным набором зависимостей, что облегчает отладку и поддержку.

3.4 Терминальный «проводник» на ncurses

Файл `browser.c` реализует интерактивный режим `trashctl browse`. При старте инициализируется ncurses, настраиваются цветовые пары и устанавливается обработчик сигнала SIGWINCH для динамического отслеживания изменения размера окна. Создаются три окна: список файлов (левая панель), детальная информация (правая панель) и строка подсказок внизу. Навигация возможна при помощи стрелок, Enter открывает каталоги, Backspace поднимает на уровень вверх, Delete перемещает выделенный файл в корзину через перехватчик, F1 показывает справку, а q завершает работу. При получении KEY_RESIZE или сигнала SIGWINCH вызывается `ioctl(TIOCGWINSZ)` и `resizeterm()`, после чего с помощью `wresize()` и `mvwin()` размеры и позиция всех окон подгоняются под новый размер терминала и интерфейс полностью перерисовывается.

3.5 Сборка и конфигурация

Сборка проекта осуществляется через Makefile в корне. Цель `interceptor` переходит в `src/interceptor` и вызывает компиляцию `trash_unlink.c` в `build/debug/libtrash.so`. Цель `cli` переходит в `src/cli`, где компилируются все модули `main.c`, `list.c`, `restore.c`, `purge.c` и `browser.c` с линковкой `-lncursesw` в исполняемый `build/debug/trashctl`. Для работы необходимо установить переменные окружения: `TRASH_BASE` указывает корневую папку проекта, а `LD_PRELOAD` — путь до `libtrash.so`. Такая схема разделения позволяет независимо собирать и тестировать перехватчик и утилиту, а также легко интегрировать систему в различные окружения.

4 ОПИСАНИЕ АЛГОРИТМОВ РЕШЕНИЯ ЗАДАЧИ

В данном разделе рассмотрены алгоритмы работы четырёх функций:
1) `move_to_trash()` - перемещение удаляемого файла в корзину;
2) `build_abs_path()` - построение абсолютного пути для `unlinkat`;
3) `cmd_list()` - загрузка и отображение записей корзины;

4.1 Разработка схем алгоритмов

Схема алгоритма функции, которая перемещает файл в корзину, `move_to_trash()`, приведена в приложении А.

Схема алгоритма функции, которая строит абсолютный путь из `dirfd` и `pathname`, `build_abs_path()`, приведена в приложении Б.

4.2 Разработка алгоритмов

Функция `move_to_trash(const char *path)`:

1) Начало;

2) Входные данные:

- `const char *path` — путь к файлу для удаления.

3) Получение информации о файле

- Вызвать `stat(path, &st)`;

- Если возвращено ≠ 0, перейти к шагу 8

4) Проверка, не в корзине ли мы уже:

- Сформировать `trash_root = TRASH_BASE/.trash`;

-Если `path` начинается с `trash_root`, вызвать оригинальный `unlink(path)` и перейти к шагу 9.

5) Генерация уникального имени:

- Получить базовое имя `basename(path)`;

- Сформировать строку `"<epoch>_<rand>[.ext]"`.

6) Обеспечение структур каталогов:

- `mkdir(TRASH_BASE/.trash, 0755)`;

- `mkdir(.../files, 0755)`;

- `mkdir(.../info, 0755)`.

7) Перемещение и запись метаданных:

- `rename(path, TRASH_BASE/.trash/files/<newname>);`

- Открыть `TRASH_BASE/.trash/info/<newname>.info` и записать путь оригинала, имя удаляемого файла и размер;

- Закрыть файл.

8) Обработка ошибок:

- Если любой из шагов 3–7 завершился не успешно, вызвать оригинальный `unlink(path)` и вернуть -1.

9) Возврат значения:

- При успешном перемещении вернуть 0;

- В противном случае вернуть `-1`.

10) Конец.

Функция `build_abs_path(int dirfd, const char *pathname, char *out, size_t outlen)`:

1) Начало;

2) Входные данные:

- `int dirfd` - файловый дескриптор каталога;

- `const char *pathname` - имя файла (абсолютное или относительное);

- `char *out, size_t outlen` - буфер размера `outlen` для записи результата.

3) Проверка абсолютного пути:

- Если `pathname[0] == '/'`, скопировать `pathname` в `out` и перейти к шагу 8.;

4) Проверка `dirfd == AT_FDCWD`:

- Вызвать `getcwd(cwd, sizeof(cwd));`

- `snprintf(out, outlen, "%s/%s", cwd, pathname);`

- Перейти к шагу 8.

5) Если `dirfd != AT_FDCWD`:

- Сформировать `linkpath = "/proc/self/fd/%d", dirfd;`

- Прочитать `readlink(linkpath, dirpath, sizeof(dirpath));`

- Если успешно, `snprintf(out, outlen, "%s/%s", dirpath, pathname);`

- Иначе нужно вернуть -1.

6) Проверка переполнения буфера:

- Если `outlen` недостаточно, вернуть -1.

7) Возврат значения:

- При успехе вернуть 0;

- При ошибке -1.

8) Конец.

Функция `cmd_list(const char *base)`:

1) Начало;

2) Входные данные:

- `const char *base` - корень проекта.

3) Формирование пути к `info`-каталогу:

- `snprintf(dir, "%s/.trash/info", base).`

4) Открытие каталога:

- `opendir(dir);`

- При ошибке вывести `perror` и вернуть `1`.

5) Вывод заголовка таблицы.

6) Цикл по записям `readdir()`:

- Пропустить имена длиной < 6 или не оканчивающиеся на `.info`;

- Извлечь `id` = имя без суффикса `.info`;

- Открыть `dir/id.info`;

- Последовательно `fgets() + sscanf()` для полей `original_path, deleted_time, size`;

- `printf("%-20s %-20s %-8ld %s\n", id, deleted_time, size, original_path)`.

7) Закрыть каталог и вернуть значение 0.

8) Конец.

5 РУКОВОДСТВО ПОЛЬЗОВАТЕЛЯ

5.1 Установка и сборка

1) Клонировать репозиторий в `$HOME/CourseWork`.
2) Установить зависимости:
$ `sudo dnf install gcc make ncurses-devel`
3) Сборка перехватчика:
$ `cd CourseWork/src/interceptor`
$ `make`
4) Сборка CLI:
$ `cd ../cli`
$ `make`

5.2 Настройка окружения

1) Перед использованием задать:
$ `export TRASH_BASE=$HOME/CourseWork`
$ `export LD_PRELOAD=$TRASH_BASE/build/debug/libtrash.so`
$ `export PATH=$TRASH_BASE/build/debug:$PATH`

5.3 Основные команды CLI

Интерфейс командной строки предоставляет основные команды:
1) `trashctl list` - отобразить содержимое корзины.
2) `trashctl restore <ID>` - восстановить файл.
3) `trashctl purge <ID>` - удалить запись навсегда.
4) `trashctl purge --all` - очистить корзину.

5.4 Терминальный проводник

Терминальный проводник можно запустить используя:
$ `trashctl browse [<start_dir>]`
Основные команды управления проводником:
1) стрелки — выбор;
2) Enter — войти в папку;

3) Backspace — вверх;
4) Del — переместить в корзину;
5) F1 — помощь;
6) q — выход.

РЕЗУЛЬТАТЫ РАБОТЫ

Для просмотра содержимого корзины можно использовать команду `trashctl list`. Результат выполнения показан на рисунке 5.1.

```
hechert@fedora:~/tar_working_dir/Маслаков_Н.А./CourseWork/build/debug$ trashctl list
ID                  Deleted Time         Size     Original Path
1747085035_89383.o  2025-05-13T00:23:55  8232     /home/hechert/tar_working_dir/Маслаков_Н.А./CourseWork/src/cli/main.o
1747085035_30886.o  2025-05-13T00:23:55  10144    /home/hechert/tar_working_dir/Маслаков_Н.А./CourseWork/src/cli/list.o
1747085035_92777.o  2025-05-13T00:23:55  9480     /home/hechert/tar_working_dir/Маслаков_Н.А./CourseWork/src/cli/restore.o
1747085035_36915.o  2025-05-13T00:23:55  7848     /home/hechert/tar_working_dir/Маслаков_Н.А./CourseWork/src/cli/purge.o
1747085035_47793.o  2025-05-13T00:23:55  26720    /home/hechert/tar_working_dir/Маслаков_Н.А./CourseWork/src/cli/browser.o
1747085035_38335    2025-05-13T00:23:55  44112    /home/hechert/tar_working_dir/Маслаков_Н.А./CourseWork/src/cli/../../build/debug/trashctl
```

Рисунок 5.1 — Список содержимого корзины

Также можно очистить корзину командой `trashctl purge --all`. Результат представлен на рисунке 5.2.

```
hechert@fedora:~/tar_working_dir/Маслаков_Н.А./CourseWork/build/debug$ trashctl purge --all
Purged 1747085035_89383.o
Purged 1747085035_30886.o
Purged 1747085035_92777.o
Purged 1747085035_36915.o
Purged 1747085035_47793.o
Purged 1747085035_38335
hechert@fedora:~/tar_working_dir/Маслаков_Н.А./CourseWork/build/debug$ trashctl list
ID                  Deleted Time         Size     Original Path
hechert@fedora:~/tar_working_dir/Маслаков_Н.А./CourseWork/build/debug$
```

Рисунок 5.2 — Результат очистки корзины

С помощью команды `trashctl purge <ID>` можно удалить определенный файл навсегда из корзины. Также с помощью команды `trashctl restore <ID>` возможно восстановление файла по исходному пути. Результат представлен на рисунке 5.3.

```
hechert@fedora:~/tar_working_dir/Маслаков_Н.А./CourseWork/build/debug$ rm -f /home/hechert/tar_working_dir/Маслаков_Н.А./CourseWork/src/interceptor/test.txt
hechert@fedora:~/tar_working_dir/Маслаков_Н.А./CourseWork/build/debug$ trashctl list
ID                  Deleted Time         Size     Original Path
1747086742_89383.txt 2025-05-13T00:52:22  0        /home/hechert/tar_working_dir/Маслаков_Н.А./CourseWork/src/interceptor/test.txt
hechert@fedora:~/tar_working_dir/Маслаков_Н.А./CourseWork/build/debug$ trashctl restore 1747086742_89383.txt
Restored 1747086742_89383.txt → /home/hechert/tar_working_dir/Маслаков_Н.А./CourseWork/src/interceptor/test.txt
hechert@fedora:~/tar_working_dir/Маслаков_Н.А./CourseWork/build/debug$ trashctl list
ID                  Deleted Time         Size     Original Path
hechert@fedora:~/tar_working_dir/Маслаков_Н.А./CourseWork/build/debug$ if [ -e /home/hechert/tar_working_dir/Маслаков_Н.А./CourseWork/src/interceptor/test.txt ]; then
    echo "Файл существует."
else
    echo "Файл не найден."
fi
Файл существует.
hechert@fedora:~/tar_working_dir/Маслаков_Н.А./CourseWork/build/debug$ rm -f /home/hechert/tar_working_dir/Маслаков_Н.А./CourseWork/src/interceptor/test.txt
hechert@fedora:~/tar_working_dir/Маслаков_Н.А./CourseWork/build/debug$ trashctl list
ID                  Deleted Time         Size     Original Path
1747086886_89383.txt 2025-05-13T00:54:46  0        /home/hechert/tar_working_dir/Маслаков_Н.А./CourseWork/src/interceptor/test.txt
hechert@fedora:~/tar_working_dir/Маслаков_Н.А./CourseWork/build/debug$ trashctl purge 1747086886_89383.txt
Purged 1747086886_89383.txt
hechert@fedora:~/tar_working_dir/Маслаков_Н.А./CourseWork/build/debug$ if [ -e /home/hechert/tar_working_dir/Маслаков_Н.А./CourseWork/src/interceptor/test.txt ]; then
    echo "Файл существует."
else
    echo "Файл не найден."
fi
Файл не найден.
```

Рисунок 5.3 — Результат восстановления и удаления файла

Для открытия терминального проводника нужно использовать команду `trashctl browse`. Результат представлен на рисунке 5.4.

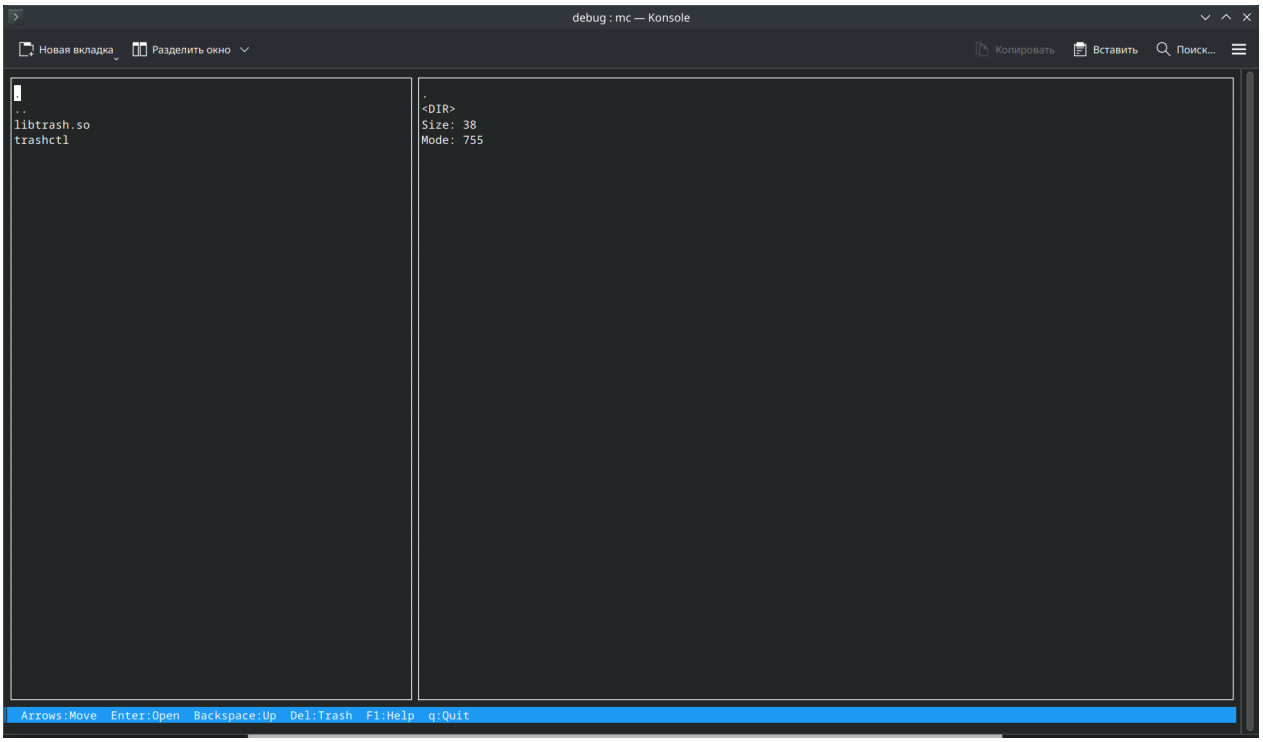

Рисунок 5.4 — Терминальный проводник

ЗАКЛЮЧЕНИЕ

В ходе выполнения курсового проекта была спроектирована и реализована кросс-приложенческая система «Корзина» для Unix-утилит, использующих системные вызовы `unlink()` и `unlinkat()`. Основной идеей решения стало применение механизма LD_PRELOAD для динамического перехвата вызовов удаления файлов без необходимости модификации исходных программ. Такой подход обеспечивает максимальную прозрачность: любая команда или сторонняя утилита, при запуске с нашей библиотекой, автоматически перенаправляет удаление в специально созданный каталог-корзину, где для каждого объекта сохраняются полные метаданные (оригинальный путь, время удаления, размер).

Для управления содержимым корзины разработана утилита `trashctl`, включающая традиционные команды `list`, `restore`, `purge` и интерактивный режим `browse`, реализованный на базе ncurses. Интерфейс двухпанельного файлового проводника, поддерживающего динамическое изменение размеров окна, обеспечивает удобство навигации и мгновенный отклик на действия пользователя. Все операции, от сканирования каталога с `.info`-файлами до переименования и восстановления, выполнены с учётом возможных исключительных ситуаций и ошибок ввода-вывода, что повышает надёжность системы.

Таким образом, поставленная цель по созданию удобного и универсального механизма «Корзины» для командного и программного удаления файлов достигнута, а архитектура и код остаются открытыми для дальнейшего развития в учебных и практических проектах.

СПИСОК ИСПОЛЬЗУЕМЫХ ИСТОЧНИКОВ

[1] Брайан Керниган, Деннис Ритчи. Язык программирования Си. – Москва, «Вильямс», 2019 г.

[2] Андрей Робачевский. Программирование на языке Си в UNIX. – СПб, «БХВ-Петербург», 2015 г.

[3] Андрей Алексеев. Программирование на языке Си в среде Linux. – СПб, «БХВ-Петербург», 2015 г.

[4] Майкл Керниган, Брайан В. Керниган. UNIX. Руководство системного программиста. – Москва, «Вильямс», 2018 г.

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

ПРИЛОЖЕНИЕ А

Схема функции `move_to_trash()`

ПРИЛОЖЕНИЕ Б

Схема функции `build_abs_path()`

ПРИЛОЖЕНИЕ В

Листинг кода

ПРИЛОЖЕНИЕ Г

Ведомость документов