

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 01 123 ПЗ

Студент:

Ткаченко И.Д.

Руководитель:

Старший преподаватель  
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 Ткаченко Илье Дмитриевичу

1. Тема проекта «Корзина» для программ, использующих системный вызов `unlink()`

2. Срок сдачи студентом законченного проекта 25 мая 2023 г.

3. Исходные данные к проекту Язык программирования – Си

4. Содержание расчетно-пояснительной записки (перечень вопросов, которые подлежат разработке)

Введение. 1. Обзор литературы. 2. Системное проектирование. 3. Функциональное проектирование. 4. Разработка программных модулей. 5. Результаты работы. Заключение. Литература.

5. Перечень графического материала (с точным обозначением обязательных чертежей и графиков) 1. Блок-схема функции `find_first_slash()`. 2. Блок-схема функции `input_file_path()`.

6. Консультант по проекту Л.П. Поденок

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%;

раздел 4 – 20%;

раздел 5 – 35%;

разделы 6,7,8 – 5%;

раздел 9 — 5%;

оформление пояснительной записки и графического материала – 10 %

Защита курсового проекта с 23.05.2023 по 12.06.2023.

РУКОВОДИТЕЛЬ

Л.П. Поденок

\_\_\_\_\_  
(подпись)

Задание принял к исполнению

\_\_\_\_\_  
(дата и подпись студента)

И.Д. Ткаченко

## СОДЕРЖАНИЕ

ВВЕДЕНИЕ.....	2
1 ОБЗОР ЛИТЕРАТУРЫ.....	3
1.1 Обзор методов и алгоритмов решения поставленной задачи.....	3
1.2 Постановка задачи.....	3
2. СИСТЕМНОЕ ПРОЕКТИРОВАНИЕ.....	4
2.1 Модуль перехвата системных вызовов unlink и unlinkat.....	4
2.2 Модуль вывода содержания корзины.....	4
2.3 Модуль безвозвратного удаления файла из корзины.....	5
2.4 Модуль перемещения файла в корзину.....	5
2.5 Модуль восстановления файла из корзины.....	5
2.6 Модуль записи информации в файл журнала.....	5
3. ФУНКЦИОНАЛЬНОЕ ПРОЕКТИРОВАНИЕ.....	5
3.1 Структура tm.....	6
3.2 Структура DIR.....	6
3.3 Структура dirent.....	6
4. РАЗРАБОТКА ПРОГРАММНЫХ МОДУЛЕЙ.....	7
4.1 Алгоритм по шагам функции clear_trashbin().....	7
4.2 Алгоритм по шагам функции check_trash_log().....	7
5. РЕЗУЛЬТАТЫ РАБОТЫ.....	8
5.1 Программа unlink.so.....	8
5.2 Программа trash.....	8
ЗАКЛЮЧЕНИЕ.....	9
СПИСОК ИСПОЛЬЗУЕМЫХ ИСТОЧНИКОВ.....	10
ПРИЛОЖЕНИЕ А.....	11
ПРИЛОЖЕНИЕ Б.....	12
ПРИЛОЖЕНИЕ В.....	13
ПРИЛОЖЕНИЕ Г.....	27

## ВВЕДЕНИЕ

В наше время удаление файлов из компьютера может вызвать серьезные проблемы. Не всегда пользователь хочет окончательно избавиться от файла, но случайно нажатие кнопки "Удалить" может привести к потере важной информации. Для решения этой проблемы была создана функциональность "Корзина", которая позволяет пользователям перемещать файлы в специальную папку, где они будут храниться до момента окончательного удаления.

В данном курсовом проекте мы будем перехватывать системные вызовы `unlink()` и `unlinkat()` на языке программирования Си, для реализации функционала "Корзина". Мы будем создавать программу, которая будет перемещать файлы в корзину вместо их окончательного удаления, что позволит пользователю легко восстановить удаленные файлы в случае необходимости.

Наша программа будет включать в себя создание папки "Корзина" при помощи функции `mkdir()` в домашней директории пользователя, а также файл журнала, в котором будут храниться записи о перемещенных в корзину файлах при помощи функций `fopen()` и `fprintf()`. Также мы будем использовать системный вызов `rename()` для перемещения файлов в корзину.

Цель данного курсового проекта - написание программы "Корзина" на языке программирования Си, которая позволит пользователям безопасно удалять файлы, а также легко восстанавливать их в случае необходимости.

# 1 ОБЗОР ЛИТЕРАТУРЫ

## 1.1 Обзор методов и алгоритмов решения поставленной задачи

Для решения поставленной задачи и создания программы "Корзина" на языке программирования Си будут использованы следующие методы и алгоритмы:

Работа с файловой системой - для создания папки "Корзина", файла журнала, перемещения файлов в корзину, получения переменных среды необходимо использовать методы работы с файловой системой на языке Си. Для этого можно использовать стандартную библиотеку функций Си, такие как `mkdir()`, `rename()`, `getenv()`, `fopen()`.

Обработка ошибок - при работе с файловой системой могут возникать различные ошибки, например, файл не найден или папка уже существует. Для обработки ошибок необходимо использовать стандартную библиотеку Си и проверять результат выполнения функций на наличие ошибок и выполнять соответствующие ошибкам действия.

Работа со строками - для работы с путями к файлам, а также для записи данных в файл журнала, необходимо использовать функции работы со строками на языке Си, такие как `sprintf()` и `strcat()`, `strcpy()`, `strncmp()`, `strcmp()`.

Взаимодействие с пользователем - для обеспечения удобства использования программы, необходимо предусмотреть интерфейс, который будет взаимодействовать с пользователем. Для этого можно использовать стандартный ввод/вывод на языке Си.

Работа с датой и временем - для создания уникальных имен файлов в корзине и для записи времени удаления файлов в файл журнала, можно использовать функции работы с датой и временем, такие как `time()` и `localtime()`, `asctime()`.

Обработка команд пользователя - для обеспечения функциональности программы, необходимо обрабатывать команды пользователя, такие как перемещение файлов в корзину, восстановление файлов из корзины и удаление файлов из нее безвозвратно. Для этого можно использовать условные операторы и циклы на языке Си, а также такие функции как `remove()`, `rename()`.

## 1.2 Постановка задачи

1. Создание каталога "Корзина" с именем `trash` в домашнем каталоге пользователя.
2. Создание файла журнала в домашнем каталоге пользователя с именем `trash.log`, в котором будут храниться записи о перемещенных в корзину файлах, дата и время их удаления, а также предыдущее местоположение перемещенного в корзину файла.

3. Реализация функции перемещения файлов в корзину при помощи системного вызова `rename()`.
4. Реализация функции удаления файлов из корзины при помощи системного вызова `unlink()`.
5. Реализация функции восстановления файлов из корзины в их первоначальное место при помощи системного вызова `rename()`.
6. Реализация функции очистки корзины, при которой все файлы будут окончательно удалены из системы.
7. Обработка ошибок, которые могут возникать при работе с файлами и папками, а также сообщение пользователю о всех действиях, производимых программой.
8. Цель данного проекта - создание полноценной программы "Корзина", которая будет обеспечивать безопасное удаление файлов и их легкое восстановление в случае необходимости. Программа должна быть надежной, удобной в использовании и иметь простой и понятный интерфейс для пользователя.

## **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 перехвата системных вызовов `unlink` и `unlinkat`**

Модуль перехвата системных вызовов `unlink` и `unlinkat` предназначен для подмены системных вызовов `unlink` и `unlinkat` на функцию пользователя, что позволяет изменить поведение операционной системы во время проведения данных системных вызовов. Данный модуль реализован методом динамического связывания при помощи динамической библиотеки `unlink.so` и позволяет вместо удаления файла безвозвратно перемещать его в корзину, с возможностью последующего восстановления, а также позволяет записывать точную дату и время перемещения файла в корзину и его прошлый путь для восстановления из корзины.

### **2.2 Модуль вывода содержания корзины**

Модуль вывода содержания корзины предназначен для вывода текущего состояния корзины, а также для просмотра находящихся в ней файлов. Этот модуль дает пользователю возможность выбрать файл для восстановления или безвозвратного удаления из корзины.

## **2.3 Модуль безвозвратного удаления файла из корзины**

Модуль безвозвратного удаления файла из корзины необходим для полного удаления файла из файловой системы, что позволяет освободить свободное место на постоянном запоминающем устройстве для записи новых файлов.

## **2.4 Модуль перемещения файла в корзину**

Модуль перемещения файла в корзину позволяет пользователю указать путь до файла (полный или относительный), чтобы переместить выбранный файл в корзину, а также записать в файл журнала данные о нем. Также данный модуль учитывает возможные коллизии, и при появлении в корзине двух и более файлов с одинаковым именем к последнему перемещенному в корзину файлу добавляется постфикс в виде круглых скобок, внутри которых находится минимальное число, начиная с 1, при котором не возникает коллизий с другими файлами в корзине.

## **2.5 Модуль восстановления файла из корзины**

Модуль восстановления файла из корзины необходим для того, чтобы пользователь по собственному желанию мог вернуть перемещенный в корзину файл на его прежнее место. Данный модуль также учитывает возможные коллизии и при восстановлении файлов в один и тот же каталог, если их имена были одинаковыми до удаления, то к файлу, у которого возникла коллизия добавляется постфикс (аналогично тому, как это сделано в модуле перемещения файла в корзину).

## **2.6 Модуль записи информации в файл журнала**

Модуль записи информации в файл журнала необходим практически для всех операций с корзиной, так как только при помощи него можно получить прежнее местоположения файла в файловой системе, узнать точную дату и время удаления и даже узнать, при помощи какого метода он был удален (системный вызов `unlink` или `unlinkat` или пользователем при помощи модуля перемещения файла в корзину).

## **3. ФУНКЦИОНАЛЬНОЕ ПРОЕКТИРОВАНИЕ**

В данном разделе рассматриваются структуры, которые были использованы при разработке приложения.



### 3.1 Структура **tm**

Данная структура представляет дату и время как набор компонентов, определяющих год, месяц, и так далее, для специфического часового пояса. Это представление обычно используется вместе с форматированием значений даты и времени. Такое представление позволяет легко записывать точную дату и время в файл журнала. Структура **tm** включает в себя следующие поля:

- `int tm_sec` - секунды
- `int tm_min` - минуты
- `int tm_hour` - часы
- `int tm_mday` - день
- `int tm_mon` - месяц
- `int tm_year` - год
- `int tm_wday` - день недели
- `int tm_yday` - количество дней в году
- `int tm_isdst` - переход на летнее время
- `long int tm_gmtoff` - смещение от UTC
- `const char *tm_zone` - аббревиатура часового пояса

### 3.2 Структура **DIR**

Данная структура предназначена для хранения информации о каталоге для его последующего открытия, чтения и закрытия при помощи таких функций как `opendir()`, `readdir()` `closedir()`. Структура **DIR** включает в себя следующие поля:

- `int fd` - файловый дескриптор
- `__libc_lock_define(, lock)` - мьютекс для данной структуры
- `size_t allocation` - количество места, выделенного под блок
- `size_t size` - реальный размер блока
- `size_t offset` - текущее смещение в блоке
- `off_t filepos` - позиция следующей записи для чтения
- `char data[0]` `__attribute__((aligned (__alignof__(void*))))` - блок каталога

### 3.3 Структура **dirent**

Данная структура необходима для чтения файлов из каталога и получения информации о файле. Структура **dirent** включает в себя следующие поля:

- `ino_t d_ino` - номер индексного дескриптора
- `off_t d_off` - смещение до следующего `dirent`
- `unsigned short d_reclen` - длина `d_name`
- `unsigned char d_type` - тип файла
- `char d_name[256]` - имя файла

## 4. РАЗРАБОТКА ПРОГРАММНЫХ МОДУЛЕЙ

### 4.1 Алгоритм по шагам функции `clear_trashbin()`

1. Начало.
2. Объявление переменных: `DIR *dir`, `struct dirent *read_dir`.
3. Открытие каталога корзины вызовом функции `opendir(trash_path)` и инициализация `dir` возвращенным `opendir()` значением.
4. Проверка значения `dir` и `errno`, если `(!dir && errno)`, то вывод сообщения о ошибке, переход к шагу 12.
5. Цикл чтения каталога `dir` до тех пор, пока выражение `(read_dir=readdir(dir))` истинно, если нет, то переход к шагу 11.
6. Если имя файла «.» или «..», то переход к шагу 5.
7. Объявление переменной `file_path[MAX_PATH_LEN]`.
8. Копирование строки `trash_path` в `file_path`, конкатенация строки `file_path` сначала со строкой «/», после со строкой `read_dir->d_name`.
9. Если вызов функции `remove(file_path)` вернул истинное значение, то вывести сообщение об ошибке.
10. Переход к шагу 5.
11. Закрытие каталога `dir` при помощи вызова функции `closedir(dir)`.
12. Конец.

### 4.2 Алгоритм по шагам функции `check_trash_log()`

1. Начало.
2. Объявление переменной `int fd`, открытие файла `trash_log_path` и инициализация `fd` возвращенным значением при вызове `open(trash_log_path, O_RDWR)`
3. Если `fd` равно 0, то вывод сообщения о ошибке, переход к шагу 12.
4. Объявление переменной `struct stat s`, запись информации о файле `trash_log_path` в `s` при вызове `fstat(fd, &s)`.
5. Закрытие файла `trash_log_path` при вызове `close(fd)`.
6. Если `s.st_size` равен 0, то корзина пустая, вывод сообщения с этим текстом в `stderr`, возвращаем значение -1, переход к шагу 8.
7. Возвращаем значение 0, переход к шагу 8.
8. Конец.

## 5. РЕЗУЛЬТАТЫ РАБОТЫ

### 5.1 Программа `unlink.so`

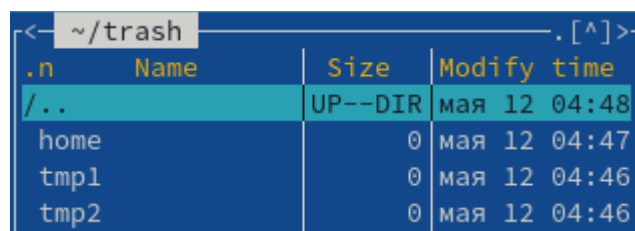
Работа с программой `unlink.so` осуществляется следующим образом в терминале:

```
[safertao@fedora course-work]$ touch tmp1 ../tmp2
[safertao@fedora course-work]$ touch /home/safertao/home
[safertao@fedora course-work]$ LD_PRELOAD=./build/unlink.so unlink ../tmp2
/home/safertao/Downloads/Ткаченко И.Д./tmp2 was renamed to /home/safertao/
trash/tmp2 by unlink syscall
[safertao@fedora course-work]$ LD_PRELOAD=./build/unlink.so rm tmp1 /home/
safertao/home
/home/safertao/Downloads/Ткаченко И.Д./course-work/tmp1 was renamed to
/home/safertao/trash/tmp1 by unlinkat syscall
/home/safertao/home was renamed to /home/safertao/trash/home by unlinkat
syscall
```

В процессе выполнения программы был создан каталог `trash` в домашнем каталоге пользователя, а также файл `trash.log` со следующим содержимым:

```
/home/safertao/Downloads/Ткаченко И.Д./tmp2 was renamed to /home/safertao/
trash/tmp2 by unlink syscall on Fri May 12 04:47:49 2023
/home/safertao/Downloads/Ткаченко И.Д./course-work/tmp1 was renamed to /home/
safertao/trash/tmp1 by unlinkat syscall on Fri May 12 04:48:21 2023
/home/safertao/home was renamed to /home/safertao/trash/home by unlinkat
syscall on Fri May 12 04:48:21 2023
```

Содержимое каталога `trash` представлено далее на рисунке 1.



.n	Name	Size	Modify time
../		UP--DIR	мая 12 04:48
home		0	мая 12 04:47
tmp1		0	мая 12 04:46
tmp2		0	мая 12 04:46

Рисунок 1 – Содержимое каталога `trash` после выполнения программы

### 5.2 Программа `trash`

Работа с программой `trash` осуществляется следующим образом в терминале:

```
[safertao@fedora course-work]$ ./build/trash
-----
menu:
q - exit
```

```

l - list trash files
p - put file into trash
c - clear trashbin
d - delete file from trash permanently
m - print menu
r - restore file from trash
-----
l
-----
trash files:
tmp2
tmp1
home
-----
r
-----
input name of file you want to restore:
home
/home/safertao/trash/home was restored from trashbin and renamed to
/home/safertao/home
-----
l
-----
trash files:
tmp2
tmp1
-----
d
-----
input name of file you want to delete permanently:
tmp1
/home/safertao/trash/tmp1 was deleted permanently from trashbin
-----
c
-----
trashbin was succesfully cleared
-----
l
-----
trash is empty
-----
p
-----
input name of file you want to put into trashbin:
/home/safertao/home
/home/safertao/home was renamed to /home/safertao/trash/home by user
-----
l
-----
trash files:
home
-----

```

## ЗАКЛЮЧЕНИЕ

В ходе выполнения курсового проекта были выполнены первоначально заданные цели, а именно была разработана и реализована функциональность корзины для программ, использующих системный вызов `unlink()`. Корзина представляет собой механизм сохранения удаленных файлов и возможность их восстановления при необходимости. Также были выполнены следующие задачи: был создан файл журнала `trash.log` для информации о времени удаления каждого файла, были обработаны ошибки, которые могут возникать при работе с файлами, также были предусмотрены коллизии при перемещении файлов в корзину и из нее.

В современном мире удаление файлов с компьютера может вызвать серьезные проблемы. Пользователи не всегда желают окончательно избавляться от файлов, но случайное нажатие кнопки "Удалить" может привести к потере важной информации. Для решения этой проблемы была создана функциональность "Корзина". Она предназначена для перемещения файлов в специальную папку, где они будут храниться до момента окончательного удаления или восстановления в случае ошибочного удаления.

## СПИСОК ИСПОЛЬЗУЕМЫХ ИСТОЧНИКОВ

1. Брайан Керниган, Деннис Ритчи. Язык программирования Си. Издательство: «Вильямс», 2019 г.
2. W. Richard Stevens, Stephen A. Rago. UNIX. Профессиональное программирование. Издательство: «Вильямс», 2017 г.
3. Robert Love. Linux System Programming: Talking Directly to the Kernel and C Library. Издательство: O'Reilly Media, 2013 г.
4. Андрей Робачевский. Программирование на языке Си в UNIX. Издательство: БХВ-Петербург, 2015 г.
5. Андрей Алексеев. Программирование на языке Си в среде Linux. Издательство: БХВ-Петербург, 2015 г.
6. Майкл Керниган, Брайан В. Керниган. UNIX. Руководство системного программиста. Издательство: «Вильямс», 2018 г.
7. Ричард Стивенс, Стивен Раго. Разработка приложений для UNIX. Издательство: «Питер», 2011 г.
8. Алан А. А. Донован, Брайан У. Керниган. Регулярные выражения. Издательство: «Вильямс», 2019 г.
9. The C Programming Language. Издательство: Prentice Hall, 1988 г.
10. Вычислительные машины, системы и сети: дипломное проектирование (методическое пособие) [Электронный ресурс]. – Минск, БГУИР 2019

## **ПРИЛОЖЕНИЕ А**

(обязательное)

Блок-схема функции `find_first_slash()`

## **ПРИЛОЖЕНИЕ Б**

(обязательное)

Блок-схема функции `input_file_path()`



## ПРИЛОЖЕНИЕ В

(обязательное)

### Листинг кода

```
// unlink.c

#define _GNU_SOURCE
#define MAX_PATH_LEN 4096 // максимальная длина пути
#define MAX_FILENAME_LEN 256 // максимальная длина имени файла

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <sys/stat.h>
#include <errno.h>
#include <time.h>
#include <unistd.h>

// функции для поиска первого и последнего '/', начиная со start_pos
int find_last_slash(const char *s, int start_pos);
int find_first_slash(const char *s, int start_pos);
// функция для записи данных в журнал
void logger(const char *new_path, const char *path_name, const char
*function);
// функция, которая работает вместо системных вызовов unlink и unlinkat
int my_unlink(const char *path_name, const char *func_name);
void init(); // функция инициализации данных
// функция для вычисления полного или относительного пути
void compute_full_or_relative_path(char *old_path_name, const char
*path_name,
char *new_path);
// функция для перемещения файла в корзину с проверками
void put_file_to_trash_with_checks(char *old_path_name, char *new_path, const
char *func_name);
void println(); // функция для вывод строки из '-'

char cwd[MAX_PATH_LEN] = {0}; // рабочий каталог
char home_path[MAX_PATH_LEN] = {0}; // домашний каталог
char trash_path[MAX_PATH_LEN] = {0}; // каталог корзины

int unlink(const char *path_name)
{
    my_unlink(path_name, __func__);
    return 0;
}

int unlinkat(int dirfd, const char *path_name, int flags)
{

```

```

    my_unlink(path_name, __func__);
    return 0;
}

int my_unlink(const char *path_name, const char *func_name)
{
    init();
    char old_path_name[MAX_PATH_LEN] = {0};
    char new_path[MAX_PATH_LEN] = {0};
    compute_full_or_relative_path(old_path_name, path_name, new_path);
    put_file_to_trash_with_checks(old_path_name, new_path, func_name);
    return 0;
}

void put_file_to_trash_with_checks(char *old_path_name, char *new_path, const
char *func_name)
{
    char tmp_path[MAX_PATH_LEN + 20] = {0};
    strcpy(tmp_path, new_path);
    FILE* f = fopen(new_path, "r");
    if(!f)
    {
        if(rename(old_path_name, new_path))
        {
            fprintf(stderr, "file with this name doesn't exist\n");
            println();
            return;
        }
    }
    else
    {
        int value = 0;
        // подбираем имена файлу, пока есть коллизии
        do
        {
            value++;
            memset(tmp_path, 0, MAX_FILENAME_LEN);
            sprintf(tmp_path, "%s(%d)", new_path, value);
        } while(fopen(tmp_path, "r"));
        if(rename(old_path_name, tmp_path))
        {
            fprintf(stderr, "file with this name doesn't exist\n");
            println();
            return;
        }
        fclose(f);
    }
    logger(tmp_path, old_path_name, func_name);
}

```

```

void compute_full_or_relative_path(char *old_path_name, const char
*path_name,
char *new_path)
{
    int index = 0;
    if(*path_name == '.' && *(path_name + 1) == '.')
    {
        char tmp_path_name[MAX_PATH_LEN] = {0};
        // берем рабочий каталог, копируем имя предыдущего каталога
        strcpy(tmp_path_name, cwd);
        index = find_last_slash(tmp_path_name, strlen(tmp_path_name));
        strncpy(old_path_name, tmp_path_name, index - 1);
        strcat(old_path_name, path_name + 2); // пропускаем .. получаем имя
        index = find_last_slash(path_name, strlen(path_name));
    }
}
else if(*path_name == '.' && *(path_name + 1) == '/')
{
    strcpy(old_path_name, cwd);
    strcat(old_path_name, "/");
    strcat(old_path_name, path_name + 2);
    index = find_first_slash(path_name, 0);
}
else if(*path_name != '/')
{
    strcpy(old_path_name, cwd);
    strcat(old_path_name, "/");
    strcat(old_path_name, path_name);
    index = find_first_slash(path_name, 0);
}
else
{
    // полный путь от корня
    index = find_last_slash(path_name, strlen(path_name));
    strcpy(old_path_name, path_name);
}
strcpy(new_path, trash_path);
strcat(new_path, "/");
strcat(new_path, path_name + index);
}

void logger(const char *new_path, const char *path_name, const char
*function)
{
    time_t rawtime;
    struct tm * timeinfo;
    time (&rawtime);
    timeinfo = localtime(&rawtime); // переводим в локальное время

```

```

char log_path[MAX_PATH_LEN + 20];
sprintf(log_path, "%s/trash.log", home_path);
FILE *log = fopen(log_path, "a");
if(!log)
{
    perror("fopen");
    exit(errno);
}
fprintf(log, "%s was renamed to %s by %s syscall on %s",
path_name, new_path, function, asctime(timeinfo));
printf("%s was renamed to %s by %s syscall\n",
path_name, new_path, function);
fclose(log);
}

void println()
{
    printf("-----\n");
}

void init()
{
    strcpy(home_path, getenv("HOME"));
    if(!*home_path)
    {
        fprintf(stderr, "ERROR: can't get home_path environment\n");
        exit(1);
    }
    strcpy(trash_path, home_path); // получаем пути
    strcat(trash_path, "/trash");
    getcwd(cwd, sizeof(cwd));
    mkdir(trash_path, 0755);
}

int find_last_slash(const char *s, int start_pos)
{
    int index = 0;
    for(int i = start_pos - 1; i > 0 && s[i]; i--)
    {
        if(s[i] == '/')
        {
            index = i + 1;
            break;
        }
    }
    return index;
}

int find_first_slash(const char *s, int start_pos)

```

```

{
    int index = 0;
    for(int i = start_pos; s[i]; i++)
    {
        if(s[i] == '/')
        {
            index = i + 1;
            break;
        }
    }
    return index;
}

```

## // trash.c

```

#define _GNU_SOURCE
#define MAX_PATH_LEN 4096 // максимальная длина пути
#define MAX_FILENAME_LEN 256 // максимальная длина имени файла

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <sys/stat.h>
#include <errno.h>
#include <time.h>
#include <stdbool.h>
#include <dirent.h>
#include <unistd.h>
#include <fcntl.h>
#include <sys/mman.h>
#include <locale.h>

void println(); // функция вывод линии из '-'
void print_menu(); // функция вывода меню на экран
void input_option(char *option); // функция ввода опции с проверками
void list_trash_files(); // функция вывод всех файлов, находящихся в корзине
// функция восстановления файла из корзины
void restore_file(const char *filename);
// функция для записи данных в журнал
void logger(const char *path_name, const char *new_path);
void input_file_path(char *new_path); // функция для ввода имени файла
void compute_paths(); // функция для вычисления путей
// функции для нахождения первого и последнего '/', начиная с start_pos
int find_last_slash(const char *s, int start_pos);
int find_first_slash(const char *s, int start_pos);
int check_trash_log(); // функция для проверки заполненности корзины
// функция для удаления строки из файла журнала
void delete_line_from_trashlog_file(const char *dest);
// функция для перманентного удаленного файла
void delete_file_permanently(const char *filename);
void init(); // функция для инициализации данных
void clear_trash_log(); // функция для очистки файла журнала
void clear_trashbin(); // функция для очистки корзины
// функция для ввода полного или относительного пути
void input_full_or_relative_file_path(char *old_path_name, char *path_name,

```

```

char *new_path);
// функция для перемещения файла в корзину с проверками
void put_file_to_trash_with_checks(char *old_path_name, char *new_path);

char cwd[MAX_PATH_LEN] = {0}; // рабочий каталог
char trash_log_path[MAX_PATH_LEN] = {0}; // путь до файла-журнала
char trash_path[MAX_PATH_LEN + 10] = {0}; // путь до корзины
char home_path[MAX_PATH_LEN] = {0}; // домашний каталог

int main()
{
    init();
    char option;
    while(true)
    {
        input_option(&option);
        switch (option)
        {
            case 'l':
            {
                list_trash_files();
                break;
            }
            case 'd':
            {
                if(check_trash_log() == -1) break;
                char path_name[MAX_PATH_LEN];
                printf("input name of file you want to delete permanently:\n");
                input_file_path(path_name);
                delete_file_permanently(path_name);
                break;
            }
            case 'c':
            {
                if(check_trash_log() == -1) break;
                clear_trash_log();
                clear_trashbin();
                printf("trashbin was succesfully cleared\n");
                printf("\n");
                break;
            }
            case 'p':
            {
                char old_path_name[MAX_PATH_LEN] = {0};
                char path_name[MAX_FILENAME_LEN] = {0};
                char new_path[MAX_PATH_LEN] = {0};
                input_full_or_relative_file_path(old_path_name, path_name,
                new_path);
                put_file_to_trash_with_checks(old_path_name, new_path);
                printf("\n");
                break;
            }

            case 'r':
            {
                if(check_trash_log() == -1) break;
                char new_path[MAX_PATH_LEN] = {0};
                printf("input name of file you want to restore:\n");

```

```

        input_file_path(new_path);
        restore_file(new_path);
        println();
        break;
    }
    case 'm':
    {
        print_menu();
        break;
    }
    case 'q':
    {
        exit(0);
        break;
    }
}
}
return 0;
}

```

```

void put_file_to_trash_with_checks(char *old_path_name, char *new_path)
{
    char tmp_path[MAX_PATH_LEN + 20];
    strcpy(tmp_path, new_path);
    FILE* f = fopen(new_path, "r");
    if(!f)
    {
        if(rename(old_path_name, new_path))
        {
            fprintf(stderr, "file with this name doesn't exist\n");
            return;
        }
    }
    else
    {
        int value = 0;
        // подбираем имена файлу, пока есть коллизии
        do
        {
            value++;
            memset(tmp_path, 0, MAX_FILENAME_LEN);
            sprintf(tmp_path, "%s(%d)", new_path, value);
        } while(fopen(tmp_path, "r"));
        if(rename(old_path_name, tmp_path))
        {
            fprintf(stderr, "file with this name doesn't exist\n");
            return;
        }
        fclose(f);
    }
    logger(old_path_name, tmp_path);
}

```

```

void input_full_or_relative_file_path(char *old_path_name, char *path_name,
char *new_path)
{
    printf("input name of file you want to put into trashbin:\n");
    fflush(stdin);
    fgets(path_name, MAX_FILENAME_LEN, stdin);
}

```

```

fgets(path_name, MAX_FILENAME_LEN, stdin);
path_name[strlen(path_name) - 1] = '\\0';

int index = 0;
if(*path_name == '.' && *(path_name + 1) == '.')
{
    char tmp_path_name[MAX_PATH_LEN];
    // берем рабочий каталог, копируем имя предыдущего каталога
    strcpy(tmp_path_name, cwd);
    index = find_last_slash(tmp_path_name, strlen(tmp_path_name));
    strncpy(old_path_name, tmp_path_name, index - 1);
    strcat(old_path_name, path_name + 2); // пропускаем .. и получаем имя
    index = find_last_slash(path_name, strlen(path_name));
}
else if(*path_name == '.' && *(path_name + 1) == '/')
{
    strcpy(old_path_name, cwd);
    strcat(old_path_name, "/");
    strcat(old_path_name, path_name + 2); // пропускаем ./
    index = find_first_slash(path_name, 0);
}
else if(*path_name != '/')
{
    strcpy(old_path_name, cwd);
    strcat(old_path_name, "/");
    strcat(old_path_name, path_name);
    index = find_first_slash(path_name, 0);
}
else
{
    // полный путь от корня
    index = find_last_slash(path_name, strlen(path_name));
    strcpy(old_path_name, path_name);
}
strcpy(new_path, trash_path);
strcat(new_path, "/");
strcat(new_path, path_name + index);
}

void input_file_path(char *new_path)
{
    char path_name[MAX_FILENAME_LEN];
    fflush(stdin);
    fgets(path_name, sizeof(path_name)/sizeof(*path_name), stdin);
    fgets(path_name, sizeof(path_name)/sizeof(*path_name), stdin);
    path_name[strlen(path_name) - 1] = '\\0';

    strcpy(new_path, trash_path);
    strcat(new_path, "/");
    strcat(new_path, path_name);
}

int check_trash_log()
{
    int fd = open(trash_log_path, O_RDWR);
    if(!fd)
    {
        perror("open");
    }
}

```



```

        exit(errno);
    }
    struct stat s;
    fstat(fd, &s);
    close(fd);
    if(!s.st_size)
    {
        fprintf(stderr, "trash is empty, put files to it first\n");
        println();
        return -1;
    }
    return 0;
}

void println()
{
    printf("-----\n");
}

void delete_file_permanently(const char *filename)
{
    FILE *f = fopen(trash_log_path, "r+");
    if(!f)
    {
        perror("fopen");
        exit(errno);
    }
    fseek(f, 0, SEEK_SET);
    char file_path[MAX_PATH_LEN];
    char dest[MAX_PATH_LEN];
    while(!feof(f))
    {
        memset(file_path, 0, MAX_PATH_LEN);
        memset(dest, 0, MAX_PATH_LEN);
        if(!fgets(file_path, MAX_PATH_LEN, f)) break;
        // становимся в нужную позицию, рассчитываем индексы для имен файлов
        char *path_end = strstr(file_path, " was renamed to ");
        if(path_end)
        {
            strncpy(dest, file_path, ((path_end - file_path)/sizeof(char)));
        }
        int dest_after_index = find_last_slash(dest, strlen(dest));
        int filename_index = find_last_slash(filename, strlen(filename));
        if(strcmp(dest + dest_after_index, filename + filename_index))
            continue;
        // удаляем найденный файл
        if(remove(filename))
        {
            perror("remove");
            exit(errno);
        }
        printf("%s was deleted permanently from trashbin\n",filename);
        println();
        delete_line_from_trashlog_file(dest);
        fclose(f);
        return;
    }
    fclose(f);
    printf("there is no such file in trashbin\n");
}

```

```

        println();
    }

void restore_file(const char *filename)
{
    FILE *f = fopen(trash_log_path, "r+");
    if(!f)
    {
        perror("fopen");
        exit(errno);
    }
    fseek(f, 0, SEEK_SET);
    char file_path[MAX_PATH_LEN];
    char dest[MAX_PATH_LEN];
    while(!feof(f))
    {
        memset(file_path, 0, MAX_PATH_LEN);
        memset(dest, 0, MAX_PATH_LEN);
        if(!fgets(file_path, MAX_PATH_LEN, f)) break;
        char *path_end = strstr(file_path, " was renamed to ");
        // становимся в правильную позицию, рассчитываем индексы для
        //сравнения
        // имен файлов
        if(path_end)
        {
            strncpy(dest, file_path, ((path_end - file_path)/sizeof(char)));
        }
        int dest_after_index = find_last_slash(dest, strlen(dest));
        int filename_index = find_last_slash(filename, strlen(filename));
        // сравниваем прошлое имя и текущее имя с введенным
        if(strcmp(dest + dest_after_index, filename + filename_index))
        {
            int last_idx = find_last_slash(file_path, strlen(file_path));
            int tmp = last_idx;
            int count = 0;
            while(*(file_path + tmp) != ' ')
            {
                tmp++;
                count++;
            }
            if(strncmp(filename + filename_index, file_path + last_idx,
                count))
                continue;
        }
        fclose(f);
        FILE *end = fopen(dest, "r");
        char tmp_path[MAX_PATH_LEN + 20];
        if(!end)
        {
            if(rename(filename, dest))
            {
                perror("rename");
                println();
                break;
            }
            printf("%s was restored from trashbin and renamed to %s\n",
                filename, dest);
        }
        else

```

```

    {
        int value = 0;
        // цикл подбора имен для избегания коллизий
        do
        {
            value++;
            memset(tmp_path, 0, MAX_FILENAME_LEN);
            sprintf(tmp_path, "%s(%d)", dest, value);
        } while(fopen(tmp_path, "r"));
        if(rename(filename, tmp_path))
        {
            perror("rename");
            println();
            break;
        }
        printf("%s was restored from trashbin and renamed to %s because
            of collisions\n", filename, tmp_path);
        fclose(end);
    }
    delete_line_from_trashlog_file(dest);
    return;
}
fclose(f);
printf("there is no such file in trashbin\n");
}

void delete_line_from_trashlog_file(const char *dest)
{
    int fd = open(trash_log_path, O_RDWR);
    if(!fd)
    {
        perror("open");
        exit(errno);
    }
    struct stat s;
    fstat(fd, &s); // для размера файла
    // отображаем файл в file_text, находим вхождение искомой строки
    char *file_text = (char*) mmap(NULL, s.st_size, PROT_READ | PROT_WRITE,
        MAP_SHARED, fd, 0);
    char *line_begin = strstr(file_text, dest);
    char *line_end = line_begin;
    while(*line_end != '\n' && line_end) line_end++;
    if(*line_end) line_end++;
    int new_file_size = s.st_size - (line_end - line_begin);
    // нашли '\n' или конец файла, новую длину файла
    // сдвигаем всю информацию влево
    while (line_end < file_text + s.st_size)
    {
        *line_begin = *line_end;
        line_begin++;
        line_end++;
    }
    ftruncate(fd, new_file_size);
    munmap(file_text, s.st_size);
    close(fd);
}

void clear_trashbin()
{

```

```

    struct dirent *read_dir;
    DIR *dir = opendir(trash_path);
    if (!dir && errno)
    {
        perror("opendir");
        errno = 0;
        return;
    }
    // читаем все файлы из trash_path, кроме . и .. и удаляем их
    while((read_dir = readdir(dir)))
    {
        if (!(strcmp(read_dir->d_name, ".") && strcmp(read_dir->d_name,
            ".."))) continue;
        char file_path[MAX_PATH_LEN];
        strcpy(file_path, trash_path);
        strcat(file_path, "/");
        strcat(file_path, read_dir->d_name);
        if(remove(file_path))
        {
            perror("remove");
        }
    }
    closedir(dir);
}

void clear_trash_log()
{
    FILE *f = fopen(trash_log_path, "w");
    if(!f)
    {
        perror("fopen");
        exit(errno);
    }
    fclose(f);
}

void compute_paths()
{
    strcpy(home_path, getenv("HOME"));
    if(!(*home_path))
    {
        fprintf(stderr, "ERROR: can't get home_path environment\n");
        exit(1);
    }
    sprintf(trash_path, "%s/trash", home_path); // вычисляем все пути
    strcpy(trash_log_path, home_path);
    strcat(trash_log_path, "/trash.log");
    getcwd(cwd, sizeof(cwd));
}

void list_trash_files()
{
    DIR *dir = opendir(trash_path);
    struct dirent *read_dir;
    if (!dir && errno)
    {
        perror("opendir");
        errno = 0;
        return;
    }

```

```

    }
    int files_count = 0;
    // читаем все файлы из trash_path, кроме . и .. и выводим их в stdout
    while((read_dir = readdir(dir)))
    {
        if (!(strcmp(read_dir->d_name, ".") && strcmp(read_dir->d_name,
            "..")))
            continue;
        if(!files_count)
        {
            printf("trash files:\n");
        }
        char tmp[MAX_PATH_LEN];
        printf("%s\n", read_dir->d_name);
        files_count++;
    }
    if(!files_count)
    {
        fprintf(stderr, "trash is empty\n");
    }
    println();
    closedir(dir);
}

void input_option(char *option)
{
    char tmp;
    while(true)
    {
        scanf("%c", &tmp);
        if(tmp == 'l' || tmp == 'r' || tmp == 'd' || tmp == 'm'
            || tmp == 'c' || tmp == 'p' || tmp == 'q') break;
    }
    *option = tmp;
    println();
}

void print_menu()
{
    printf("menu:\n");
    printf("q - exit\nl - list trash files\np - put file into trash\n");
    printf("r - restore file from trash\nd - delete file from trash ");
    printf("permanently\nc - clear trashbin\nm - print menu\n");
    println();
}

int find_last_slash(const char *s, int start_pos)
{
    int index = 0;
    for(int i = start_pos - 1; i > 0 && s[i]; i--)
    {
        if(s[i] == '/')
        {
            index = i + 1;
            break;
        }
    }
    return index;
}

```

```

int find_first_slash(const char *s, int start_pos)
{
    int index = 0;
    for(int i = start_pos; s[i]; i++)
    {
        if(s[i] == '/')
        {
            index = i + 1;
            break;
        }
    }
    return index;
}

void logger(const char *path_name, const char *new_path)
{
    time_t rawtime;
    struct tm * timeinfo;
    time (&rawtime);
    timeinfo = localtime(&rawtime); // переводим в локальное время
    FILE *log = fopen(trash_log_path, "a");
    if(!log)
    {
        perror("fopen");
        exit(errno);
    }
    fseek(log, 0, SEEK_END);
    fprintf(log, "%s was renamed to %s by user on %s", path_name, new_path,
    asctime(timeinfo));
    printf("%s was renamed to %s by user\n", path_name, new_path);
    fclose(log);
}

void init()
{
    println();
    print_menu();
    compute_paths();
    mkdir(trash_path, 0755);
}

```

**ПРИЛОЖЕНИЕ Г**  
(обязательное)

Ведомость документов