**НЕГОСУДАРСТВЕННОЕ ОБРАЗОВАТЕЛЬНОЕ ЧАСТНОЕ** **УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ** **«МОСКОВСКИЙ ФИНАНСОВО-ПРОМЫШЛЕННЫЙ УНИВЕРСИТЕТ** **“СИНЕРГИЯ”»**

|  |  |  |
| --- | --- | --- |
| **Факультет/Институт** |  |  |
|  |  | (наименование факультета/ Института) |
| **Направление/специальность** |  |  |
| **подготовки:** |  | (код и наименование направления /специальности подготовки) |
| **Форма обучения:** |  |  |
|  |  | (очная, очно-заочная, заочная) |
|  |  |  |

**Отчет по лабораторной работе №2**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **на тему** |  | Файловые операции средствами системных вызовов | | |
|  |  | (наименование темы) | | |
|  |  |  | | |
| **по дисциплине** | | |  | Системное программирование |
|  | | |  | (наименование дисциплины) |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Обучающийся** |  | Абдуллаев Самвел Олегович |  |  |
|  |  | (ФИО) |  | (подпись) |
| **Группа** |  | ДКИП – 205 прог |  |  |
|  |  |  |  |  |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Преподаватель** |  | Сибирев И.В. |  |  |
|  |  | (ФИО) |  | (подпись) |

**Москва 2025 г.**

**Лабораторная работа №2. «**Файловые операции средствами системных вызовов**»**

**Цель работы:**

получить навыки разработки приложений, реализующих операции с файлами средствами системных вызовов Linux API на языке C в операционных системах семейства Linux.

2. Краткие теоретические сведения: см. главу 3, стр. 25. Пример кода: листинг 3.1 (см. стр. 27) и файл file.c в разделе «Примеры программ», «Файлы» сайта по дисциплине [6].

3. Методические указания.

3.1. Проект может быть реализован в виде консольного приложения в среде ОС Ubuntu, Fedora, CentOS или других средствами компилятора gcc версии не ниже 4.

3.2. Проект должен быть реализован с применением системных вызовов для открытия, чтения, записи и закрытия файла. Ввод исходных данных производится исключительно через аргументы командной строки. Вывод сообщений об ошибках и результатах работы программы может производиться средствами стандартной библиотеки языка С.

3.3. Проект должен предусматривать обработку исключительных ситуаций (отсутствие или неверное количество входных параметров, ошибки открытия входного и/или выходного файла, ошибки чтения и записи).

3.4. Проект может содержать системные вызовы для блокировки файлов (входного – на чтение, выходного – на запись). Наличие функций блокировки файлов соответствует заданию повышенной сложности.

4. Порядок выполнения работы.

4.1. Написать и отладить программу, получающую в аргументах командной строки имя существующего текстового файла, имя выходного файла, который будет переписан при его наличии и в который будет помещен результат работы программы и символ, слово или число, используемый (ое) для обработки файла.

4.2. Результатом работы программы является выходной текстовый файл, содержащий текст, обработанный согласно вариантам, и возвращаемое значение – количество выполненных операций или –1 в случае ошибки.

4.3. При обработке исключительных ситуаций сообщения об ошибках выводятся на терминал.

4.4. Результатом выполнения лабораторной работы считается демонстрация работы программы и обработки исключительных ситуаций преподавателю.

5. Варианты заданий.

Варианты заданий

Номер варианта 1

Задание

Параметры командной строки

Удалить из текста заданный символ

1. Имя входного файла

2. Заданный символ

6. Содержание отчета.

6.1. Цель работы.

В современном мире программирования навыки разработки приложений, которые эффективно работают с файлами, являются крайне важными. Особенно это касается операционных систем семейства Linux, где системные вызовы предоставляют мощные инструменты для управления файлами и взаимодействия с файловой системой. Для того чтобы стать квалифицированным разработчиком в этой области, необходимо освоить основные концепции работы с системными вызовами в Linux. Это включает в себя изучение таких операций, как создание, открытие, чтение, запись и закрытие файлов. В языке C для этого используются различные функции, такие как open(), read(), write(), close() и многие другие, которые позволяют напрямую взаимодействовать с файловой системой. Кроме того, важно понимать, как работают дескрипторы файлов и как их использовать для управления доступом к ресурсам. Знание о том, как обрабатывать ошибки при выполнении системных вызовов и как правильно управлять памятью, также играет ключевую роль в разработке надежных приложений. Изучение API Linux на языке C предоставляет разработчикам возможность создавать высокопроизводительные и эффективные приложения, которые могут выполнять сложные задачи, связанные с обработкой данных. Это может включать в себя создание утилит для обработки текстовых файлов, работы с бинарными данными или даже разработки серверных приложений, которые требуют интенсивного ввода-вывода. Для достижения этих целей рекомендуется изучать документацию по системным вызовам, а также практиковаться в написании кода, который использует эти вызовы. Создание собственных проектов и участие в open-source инициативах также могут значительно ускорить процесс обучения и углубить понимание работы с файлами в Linux. Таким образом, получение навыков разработки приложений, реализующих операции с файлами средствами системных вызовов Linux API на языке C, является важной частью профессионального роста для любого программиста, стремящегося к успеху в области системного программирования и разработки программного обеспечения.

6.2. Вариант задания.

1 вариант

6.3. Листинг программы.

Код программы:

Файл: remove\_char.c

#include <stdio.h>

#include <stdlib.h>

#include <unistd.h>

#include <fcntl.h>

#include <string.h>

#include <errno.h>

#define BUFFER\_SIZE 1024

// Функция для обработки текста и удаления заданного символа

void process\_file(int input\_fd, int output\_fd, char target) {

char buffer[BUFFER\_SIZE];

ssize\_t bytes\_read;

int operations = 0;

while ((bytes\_read = read(input\_fd, buffer, BUFFER\_SIZE)) > 0) {

char processed\_buffer[BUFFER\_SIZE];

int processed\_size = 0;

for (int i = 0; i < bytes\_read; i++) {

if (buffer[i] != target) {

processed\_buffer[processed\_size++] = buffer[i];

}

}

if (write(output\_fd, processed\_buffer, processed\_size) != processed\_size) {

perror("Ошибка при записи в файл");

close(input\_fd);

close(output\_fd);

exit(EXIT\_FAILURE);

}

operations += processed\_size;

}

if (bytes\_read < 0) {

perror("Ошибка при чтении файла");

close(input\_fd);

close(output\_fd);

exit(EXIT\_FAILURE);

}

printf("Операций выполнено: %d\n", operations);

}

int main(int argc, char \*argv[]) {

if (argc != 4) {

fprintf(stderr, "Использование: %s <входной\_файл> <выходной\_файл> <символ>\n", argv[0]);

exit(EXIT\_FAILURE);

}

char \*input\_file = argv[1];

char \*output\_file = argv[2];

char target = argv[3][0];

// Открытие входного файла с правами только для чтения

int input\_fd = open(input\_file, O\_RDONLY);

if (input\_fd == -1) {

perror("Ошибка открытия входного файла");

exit(EXIT\_FAILURE);

}

// Открытие выходного файла с правами на запись (создание или перезапись)

int output\_fd = open(output\_file, O\_WRONLY | O\_CREAT | O\_TRUNC, 0666);

if (output\_fd == -1) {

perror("Ошибка открытия выходного файла");

close(input\_fd);

exit(EXIT\_FAILURE);

}

// Вызов функции для обработки файла

process\_file(input\_fd, output\_fd, target);

// Закрытие файлов

if (close(input\_fd) == -1) {

perror("Ошибка закрытия входного файла");

exit(EXIT\_FAILURE);

}

if (close(output\_fd) == -1) {

perror("Ошибка закрытия выходного файла");

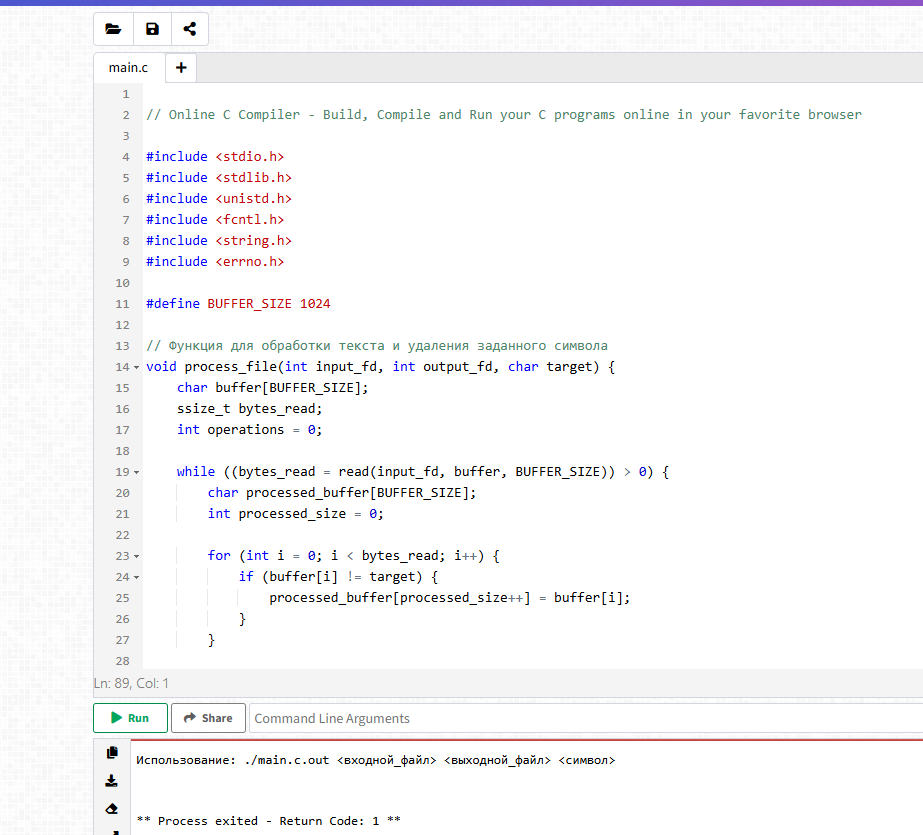
exit(EXIT\_FAILURE);

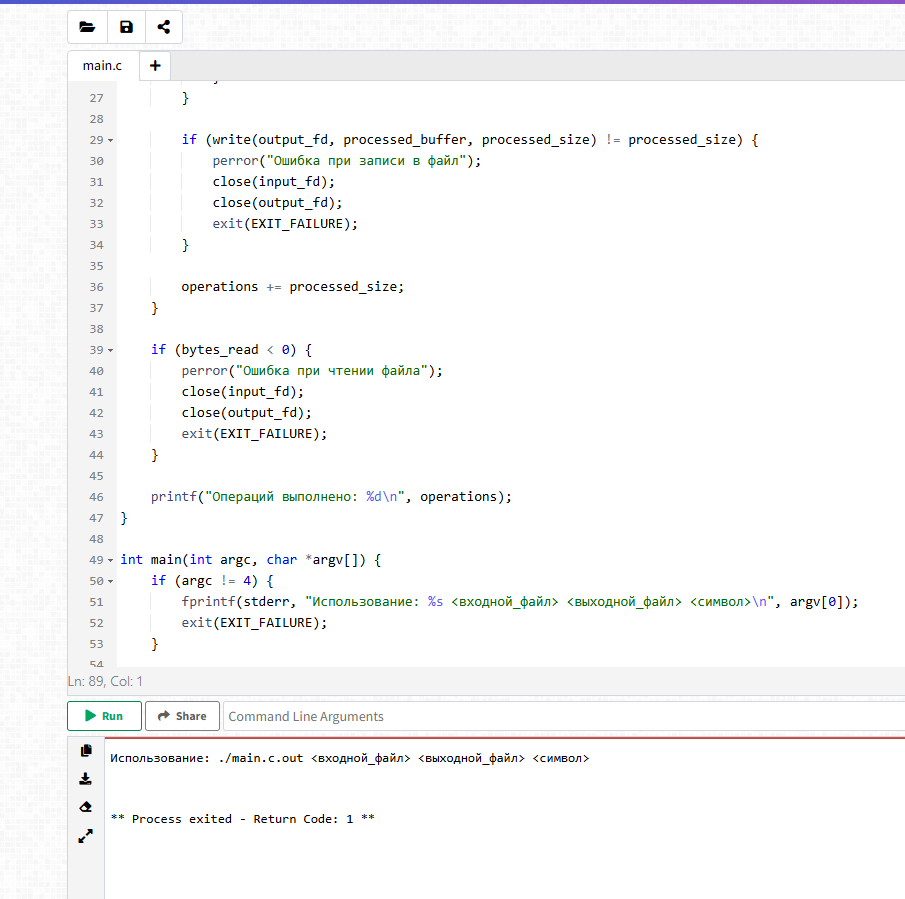
}

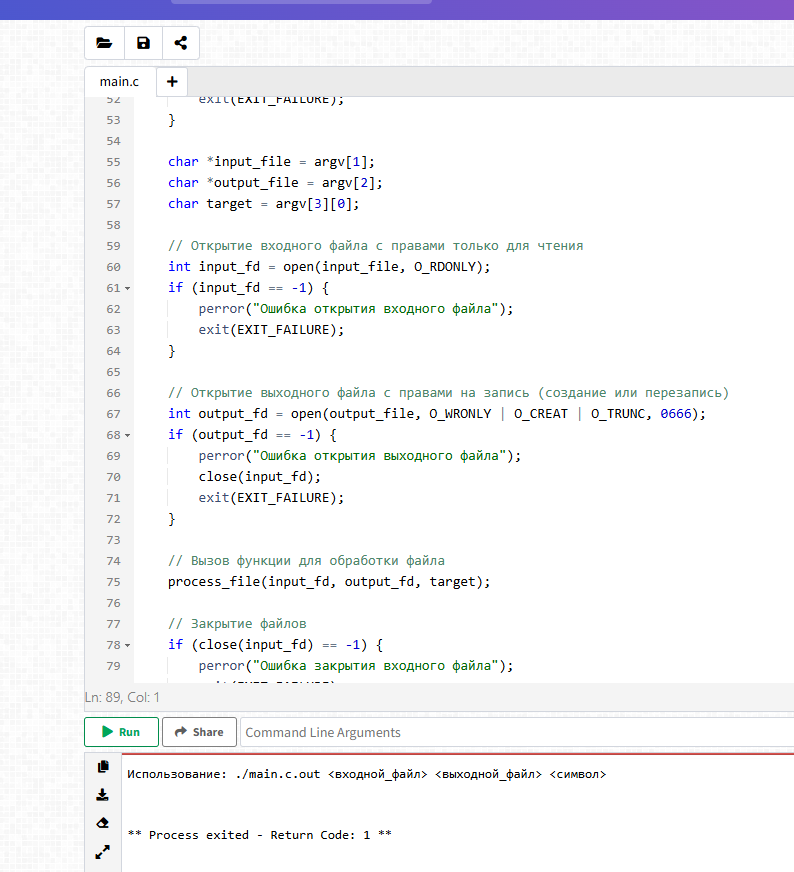
return 0;

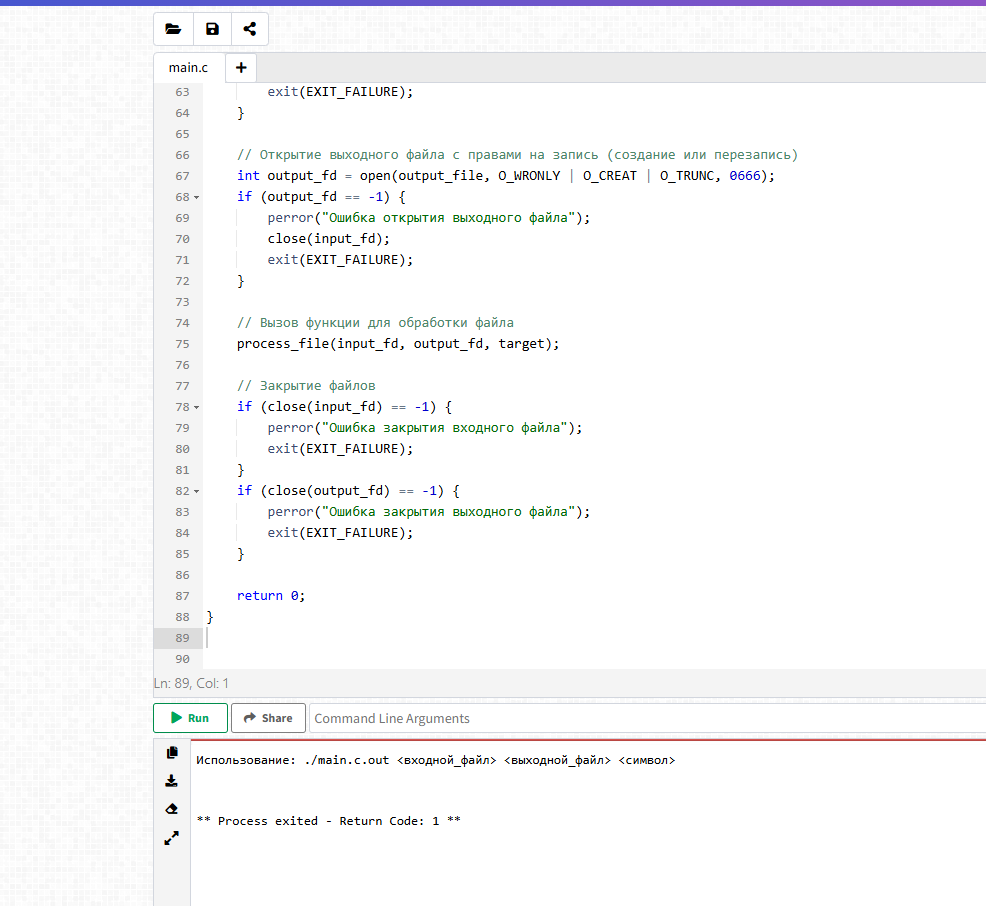
}

На компиляторе









Объяснение кода:

Используются системные вызовы:

- open() — открытие файлов

- read() — чтение данных из файла

- write() — запись данных в файл

- close() — закрытие файлов

Обработка исключений:

- Неверное количество аргументов

- Ошибки открытия/закрытия файлов

- Ошибки чтения/записи данных

Алгоритм:

- Читаем данные порциями в буфер

- Удаляем заданный символ из буфера

- Записываем обработанные данные в выходной файл

- Подсчитываем количество выполненных операций

7. Контрольные вопросы: см. вопросы для самопроверки к главе 3

Вопросы для самопроверки

1. Перечислите стандартные дескрипторы файлов. Чем они отличаются от дескрипторов обычных файлов?

2. В чем заключается универсальность модели ввода-вывода UNIX?

3. В чем отличия вызова функции open() для создания нового файла и открытия существующего?

4. Перечислите форматы и значения третьего аргумента функции open().

5. Перечислите дополнительные (кроме режима доступа) флаги функции open().

6. Каковы основные ошибки, могущие возникнуть при открытии файла?

7. Каковы особенности работы функции read()?

8. Каковы особенности работы функции write()?

9. Почему нужно явно вызывать функцию close()?

10. Для чего служит функция lseek()?

11. Каковы допустимые аргументы функции lseek()?

12. К каким типам файлов нельзя применять функцию lseek()?

13. Какие виды блокировки файла существуют в LINUX?

14. Приведите описание структуры блокировки файла, опишите ее поля.

15. Опишите процесс блокировки файла с помощью функции fcntl().

16. Опишите процесс блокировки файла с помощью функции lockf().

17. Чем отличается применение функций блокировки fcntl() и lockf()?

18. В каких случаях блокировка файла снимается операционной системой?

19. Возможно ли сочетать функции блокировки POSIX с файловыми функциями стандартной библиотеки С ()? Почему?

20. Каковы особенности функций блокировки файлов средствами стандартной библиотеки С ()?