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

|  |  |  |
| --- | --- | --- |
| **Факультет/Институт** |  | МФПУ Синергия |
|  |  | (наименование факультета/ Института) |
| **Направление/специальность** |  | 09.02.07 Информационные технологии и программирование |
| **подготовки:** |  | (код и наименование направления /специальности подготовки) |
| **Форма обучения:** |  | очная |
|  |  | (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** |  | Панцулая Темур Генадьевич |  |  |
|  |  | (ФИО) |  | (подпись) |
| **Группа** |  | ДКИП\_205 ПРОГ |  |  |
|  |  |  |  |  |

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

**Москва 2025.**

Вариант 14.

Задание: Заменить все цифры заданным символом

Параметры командной строки:

1. Имя входного файла

2. Заданный символ

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

#include <stdio.h>

#include <stdlib.h>

#include <fcntl.h>

#include <unistd.h>

#include <errno.h>

void handle\_error(const char \*message) {

perror(message);

exit(EXIT\_FAILURE);

}

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

if (argc != 3) {

fprintf(stderr, "Usage: %s <input\_file> <replacement\_char>\n", argv[0]);

exit(EXIT\_FAILURE);

}

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

char replacement\_char = argv[2][0];

// Проверка, что замещающий символ - это одиночный символ

if (argv[2][1] != '\0') {

fprintf(stderr, "Replacement character must be a single character.\n");

exit(EXIT\_FAILURE);

}

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

if (input\_fd == -1) {

handle\_error("Failed to open input file");

}

int output\_fd = open("output.txt", O\_WRONLY | O\_CREAT | O\_TRUNC, S\_IRUSR | S\_IWUSR);

if (output\_fd == -1) {

close(input\_fd);

handle\_error("Failed to open/create output file");

}

char buffer[1024];

ssize\_t bytes\_read;

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

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

if (buffer[i] >= '0' && buffer[i] <= '9') {

buffer[i] = replacement\_char;

}

}

if (write(output\_fd, buffer, bytes\_read) != bytes\_read) {

close(input\_fd);

close(output\_fd);

handle\_error("Failed to write to output file");

}

}

if (bytes\_read == -1) {

close(input\_fd);

close(output\_fd);

handle\_error("Failed to read from input file");

}

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

handle\_error("Failed to close input file");

}

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

handle\_error("Failed to close output file");

}

printf("Processing completed successfully. Output written to output.txt\n");

return 0;

}

**Пояснение к листингу кода:**

- Программа начинает с проверки количества аргументов. Если их меньше или больше двух, выводится сообщение об ошибке.

- `open()` используется для открытия входного файла и создания выходного файла. Если какой-то из них не удается открыть, выводится сообщение об ошибке и программа завершает работу.

- Чтение осуществляется в цикле до тех пор, пока не будет прочитано всё содержимое файла. Если встречается цифра, она заменяется заданным символом.

- Наконец, результаты записываются в выходной файл и оба файла закрываются.

**Вопросы для самопроверки.**

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

**Ответ:** Стандартные дескрипторы: 0 - stdin (стандартный ввод), 1 - stdout (стандартный вывод), 2 - stderr (стандартный поток ошибок). Они предопределены и открыты по умолчанию, связывая процесс с терминалом или другим процессом. Дескрипторы обычных файлов создаются при открытии файла функцией open() и представляют собой целое число, используемое для обращения к открытому файлу.

**Вопрос 2:** В чем заключается универсальность модели ввода-вывода UNIX?

**Ответ:** Универсальность заключается в том, что все ввод/вывод осуществляется как операции с файлами (байтовыми потоками). Это упрощает работу с различными устройствами (терминалы, файлы, сокеты и т.д.) через единый интерфейс.

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

**Ответ:** При создании нового файла используется флаг O\_CREAT. Если файл существует, то для открытия без перезаписи используется флаг O\_EXCL. Если файл не существует, а O\_CREAT не указан, open() вернет ошибку.

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

**Ответ:** Третий аргумент (mode) задает права доступа при создании файла (если O\_CREAT указан). Формат - восьмеричное число (например, 0644). Значения определяют права чтения, записи и выполнения для владельца, группы и остальных пользователей.

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

**Ответ:** O\_APPEND, O\_TRUNC, O\_NONBLOCK, O\_SYNC, O\_CREAT, O\_EXCL и др. Они управляют режимом открытия и работы с файлом (добавление, усечение, неблокирующий режим и т.д.).

**Вопрос 6:** Каковы основные ошибки, могущие возникнуть при открытии файла?

**Ответ:** ENOENT (файл не найден), EACCES (нет прав доступа), EMFILE (слишком много открытых файлов), ENFILE (слишком много открытых файлов в системе) и др.

**Вопрос 7:** Каковы особенности работы функции read()?

**Ответ:** Читает заданное количество байт из файла в буфер. Возвращает количество прочитанных байт или -1 в случае ошибки или конца файла.

**Вопрос 8:** Каковы особенности работы функции write()?

**Ответ:** Записывает заданное количество байт из буфера в файл. Возвращает количество записанных байт или -1 в случае ошибки.

**Вопрос 9:** Почему нужно явно вызывать функцию close()?

**Ответ:** close() освобождает ресурсы, связанные с файлом (буферы, дескрипторы). Незакрытые файлы могут привести к утечкам ресурсов и проблемам с данными.

**Вопрос 10:** Для чего служит функция lseek()?

**Ответ:** Изменяет текущую позицию (смещение) в файле, с которой будет производиться следующая операция чтения или записи.

**Вопрос 11:** Каковы допустимые аргументы функции lseek()?

**Ответ:** Смещение (относительно начала, текущей позиции или конца файла) и откуда отсчитывать смещение (SEEK\_SET, SEEK\_CUR, SEEK\_END).

**Вопрос 12:** К каким типам файлов нельзя применять функцию lseek()?

**Ответ:** К терминалам, сокетам, каналам (файлы, не поддерживающие произвольный доступ).

**Вопрос 13:** Какие виды блокировки файла существуют в Linux?

**Ответ:** Разделяемая (shared) блокировка и эксклюзивная (exclusive) блокировка. Разделяемую могут установить несколько процессов одновременно, эксклюзивную - только один.

**Вопрос 14:** Приведите описание структуры блокировки файла, опишите ее поля.

**Ответ:** struct flock содержит поля: l\_type (тип блокировки), l\_whence (откуда отсчитывать), l\_start (начальное смещение), l\_len (длина блокировки), l\_pid (PID процесса, установившего блокировку).

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

**Ответ:** Вызывается fcntl() с командой F\_SETLK (неблокирующая) или F\_SETLKW (блокирующая) и указателем на структуру flock, описывающую блокировку.

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

**Ответ:** Вызывается lockf() с командой F\_LOCK (блокирующая), F\_TLOCK (неблокирующая), F\_ULOCK (разблокировка) и размером блокировки.

**Вопрос 17:** Чем отличается применение функций блокировки fcntl() и lockf()?

**Ответ:** fcntl() более гибкая (позволяет устанавливать разделяемые блокировки и блокировать часть файла). lockf() проще в использовании, но менее функциональна.

**Вопрос 18:** В каких случаях блокировка файла снимается операционной системой?

**Ответ:** При закрытии файла процессом, завершении процесса, явном снятии блокировки.

**Вопрос 19:** Возможно ли сочетать функции блокировки POSIX с файловыми функциями стандартной библиотеки С (<stdio.h>)? Почему?

**Ответ:** Не рекомендуется. Функции из <stdio.h> имеют собственную буферизацию, что может привести к некорректной работе блокировок.

**Вопрос 20:** Каковы особенности функций блокировки файлов средствами стандартной библиотеки С (<stdio.h>)?

**Ответ:** Стандартная библиотека C не предоставляет функций блокировки файлов. Для этого используются POSIX-функции (fcntl(), lockf()). flockfile()/funlockfile() в <stdio.h> предназначены для потоковой безопасности, а не для блокировки доступа к файлу из других процессов.