Министерство образования Республики Беларусь

Учреждение образования

БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ ИНФОРМАТИКИ И РАДИОЭЛЕКТРОНИКИ

Факультет компьютерных систем и сетей

Кафедра информатики

Дисциплина: Операционные среды и системное программирование

**ОТЧЕТ**

к лабораторной работе № 3

на тему

«Основы программирования на C под Unix.  
Инструментарий программиста в Unix»

Выполнил:

студент гр. 153504

Сивый А. А.

Проверил:

Гриценко Н. Ю.

Минск 2024

СОДЕРЖАНИЕ

[1 Цель работы 3](#_Toc146631498)

[2 Краткие теоретические сведения 4](#_Toc146631499)

[3 Полученные результаты 5](#_Toc146631500)

[Выводы 6](#_Toc146631501)

[Список использованных источников 7](#_Toc146631502)

[Приложение А (обязательное) Листинг исходного кода 8](#_Toc146631503)

## 1 ЦЕЛЬ РАБОТЫ

Изучение среды программирования и основных инструментов: компилятор/сборщик («коллекция компиляторов») gcc, управление обработкой проекта make (и язык makefile), библиотеки и т.д.

Практическое использование основных библиотек и системных вызовов: ввод-вывод и работа с файлами, обработка текста, распределение памяти, управление выполнением и т.п.

## 2 КРАТКИЕ ТЕОРЕТИЧЕСКИЕ СВЕДЕНИЯ

Программирование на языке С под операционной системой Unix представляет собой уникальное и мощное средство разработки программного обеспечения. Основы этого процесса включают в себя работу с системными вызовами, управление процессами, вводом-выводом, файлами и директориями, а также использование стандартных библиотек. Эти концепции обеспечивают эффективность, надежность и портируемость кода, что делает программирование на C под Unix важным для создания профессионального и качественного обеспечения для Unix-подобных систем.

Структура программы на языке C под Unix обычно состоит из функций, каждая из которых выполняет определенную задачу, начиная с функции main, которая является точной входа. Для работы с различными функциями и системными вызовами Unix используются стандартные заголовочные файлы, такие как <stdio.h>, <stdlib.h>, а также функции ввода-вывода и управления процессами. Организация кода в функции позволяет повторно использовать его и делает программу более структурированной и понятной.

Системные вызовы в программировании на языке C под Unix представляют собой интерфейс для взаимодействия с операционной системой. Они позволяют программе выполнять различные операции, такие как управление файлами и директориями, управление процессами, ввод-вывод данных, работу с сетью и другие системные задачи. Каждый системный вызов имеет свой уникальный номер, который используется для его идентификации. При вызове системного вызова происходит переход из пользовательского режима работы программы в режим ядра операционной системы, где выполняется соответствующая операция, после чего управление возвращается обратно в программу. Эффективное использование системных вызовов позволяет создавать быстрые, масштабируемые и надежные программы под Unix.

Функции ввода-вывода в программировании на C под Unix предоставляют удобный способ работы с вводом данных с клавиатуры и выводом результатов на экран или в файлы. Стандартная библиотека C предоставляет функции, такие как printf, scanf, fopen, fclose, fread, fwrite, fprintf и т.д.

Makefile – это текстовый файл, содержащий инструкции для сборки и компиляции программы или проекта. В нем определяются зависимости между файлами и команды для их сборки. Makefile упрощает процесс сборки, так как позволяет компилировать только те файлы, которые изменились с момента предыдущей компиляции, что экономит время. Для создания Makefile используется специальный синтаксис с указанием целей (targets), зависимостей и команд для выполнения. Команды в Makefile обычно вызывают компилятор и другие утилиты с определенными параметрами. Makefile часто используется в проектах на C под Unix для автоматизации процесса сборки и управления зависимостями между файлами.

## 3 ПОЛУЧЕННЫЕ РЕЗУЛЬТАТЫ

В ходе выполнения лабораторной работы был разработан программный продукт, представляющий собой криптографический фильтр. Он выполняет шифрование и дешифрование потока, используя криптоалгоритм Цезаря. На рисунке 3.1 представлен исходный текст, который требуется зашифровать.

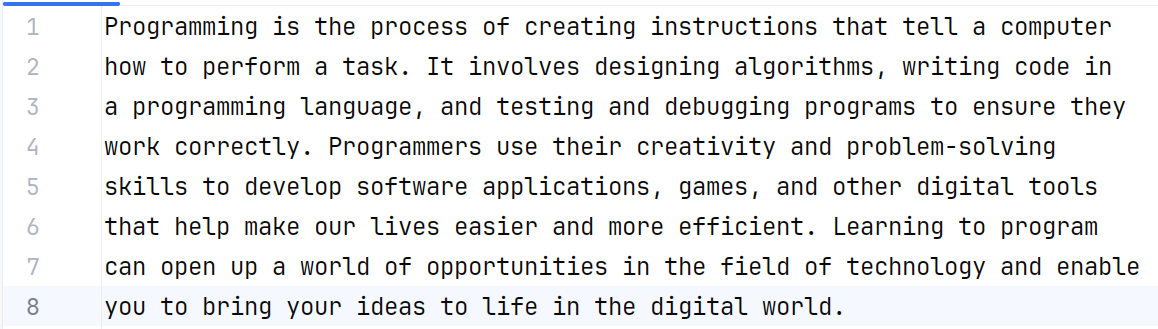


Рисунок 3.1 – Текст для шифрования

Запустим программу используя команды, представленные на рисунке 3.2, после чего получим зашифрованный текст, представленный на рисунке 3.3.

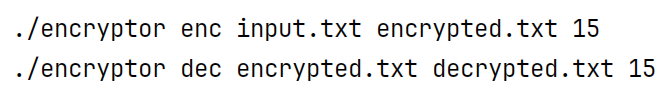


Рисунок 3.2 – Пример использования программы

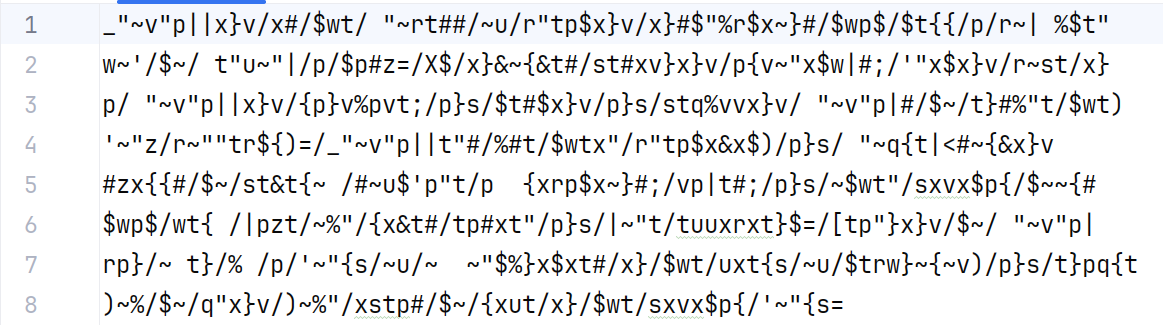


Рисунок 3.3 – Зашифрованный текст

После дешифрования текста, в результате получим текст, представленный на рисунке 3.1. Программа выполняет обработку различных исключительных ситуаций, а также сохраняет формат текста.

## ВЫВОДЫ

В ходе выполнения лабораторной работы по основам программирования на языке C под Unix были изучены основные концепции работы с системными вызовами, управлением процессами, вводом-выводом, файлами и директориями, а также применение стандартных библиотек. Эти знания являются ключевыми для разработки эффективных и надежных приложений под Unix-подобными системами. Кроме того, были рассмотрены основные инструменты программиста в Unix, такие как текстовые редакторы, компиляторы, отладчики и утилиты управления версиями кода, которые значительно облегчают процесс разработки и отладки программ. В результате выполнения лабораторной работы были закреплены навыки программирования на языке C под Unix и освоены основные инструменты, необходимые для создания качественного программного обеспечения.

## СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

[1] Брайан Керниган, Роб Пайк. UNIX — универсальная среда программирования. – М.: Изд. Дом Вильямс, 1992. – 230 с.

[2] Руководство по Си [Электронный ресурс] – Режим доступа: <https://metanit.com/c/tutorial/> – Дата доступа: 01.03.2024

[3] Makefile Tutorial [Электронный ресурс] – Режим доступа: <https://makefiletutorial.com/> – Дата доступа: 01.03.2024

**ПРИЛОЖЕНИЕ А**

## (обязательное)

## Листинг исходного кода

Листинг 1 – Программный код разработанного приложения

#include <stdlib.h>

#include <ctype.h>

#include "file\_utils.c"

#include "caesar.c"

int is\_number(const char\* str) {

int len = strlen(str);

for (int i = 0; i < len; i++) {

if (!isdigit(\*str))

return 0;

}

return 1;

}

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

if (argc != 5) {

fprintf(stderr, "Using: %s <enc/dec> <input\_file> <output\_file> <shift>\n", argv[0]);

return 1;

}

const char\* action = argv[1];

const char\* input = argv[2];

const char\* output = argv[3];

if (!is\_number(argv[4])) {

fprintf(stderr, "Shift must be a number\n");

return 1;

}

int shift = atoi(argv[4]);

char\* text = read\_file(input);

if (!strcmp(action, "enc")) {

encrypt(text, shift);

}

else if (!strcmp(action, "dec")) {

decrypt(text, shift);

}

else {

fprintf(stderr, "Incorrect operation parameter. Use 'enc' or 'dec'\n");

free(text);

return 1;

}

write\_file(output, text);

free(text);

return 0;

}

#include <stdio.h>

#include <stdlib.h>

static char\* read\_file(const char\* filename) {

FILE\* file = fopen(filename, "r");

if (!file) {

fprintf(stderr, "Error reading file\n");

exit(1);

}

fseek(file, 0, SEEK\_END);

long size = ftell(file);

fseek(file, 0, SEEK\_SET);

char\* buffer = (char\*)malloc(size + 1);

fread(buffer, 1, size, file);

buffer[size] = '\0';

fclose(file);

return buffer;

}

static void write\_file(const char\* filename, char\* text) {

FILE\* file = fopen(filename, "w");

if (!file) {

fprintf(stderr, "Error writing file\n");

exit(1);

}

fprintf(file, "%s", text);

fclose(file);

}

#include <string.h>

#include <stdlib.h>

static void encrypt(char\* text, int shift) {

int len = strlen(text);

for (int i = 0; i < len; i++) {

if (text[i] >= 32 && text[i] <= 126) {

text[i] = ((text[i] - 32 + shift) % 95) + 32;

}

}

}

static void decrypt(char \*text, int shift) {

int len = strlen(text);

for (int i = 0; i < len; i++) {

if (text[i] >= 32 && text[i] <= 126) {

text[i] = ((text[i] - 32 - shift + 95) % 95) + 32;

}

}

}