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

ОТЧЁТ

к лабораторной работе №3

на тему

**ОСНОВЫ ПРОГРАММИРОВАНИЯ НА C ПОД UNIX.**

**ИНСТРУМЕНТАРИЙ ПРОГРАММИСТА В UNIX.**

|  |
| --- |
| Выполнил: студент гр. 253503  Кудош А.С. |
| Проверил: ассистент кафедры информатики Гриценко Н.Ю. |

Минск 2025

**СОДЕРЖАНИЕ**

[1 Постановка задачи и программа для её решения 3](#_Toc189759479)

[1.1 Постановка задачи 3](#_Toc189759480)

[1.2 Демонстрация работы программы 3](#_Toc189759481)

[Заключение 4](#_Toc189759482)

[Список использованных источников 5](#_Toc189759483)

[Приложение А (справочное) Исходный код 6](#_Toc189759484)

**1 ПОСТАНОВКА ЗАДАЧИ И ПРОГРАММА ДЛЯ ЕЁ РЕШЕНИЯ**

**1.1 Постановка задачи**

Целью данной лабораторной работы является изучение среды программирования и освоение основных инструментов разработки, таких как компилятор *GCC*, утилита *make* и язык *makefile*. Предметом исследования выступает процесс создания многомодульных программ, которые используют системные вызовы и стандартные библиотеки для выполнения задач, связанных с обработкой текстовых и бинарных данных. Особое внимание уделяется вопросам управления памятью, организации ввода-вывода и тестирования программного обеспечения.

В рамках работы требовалось разработать программу-фильтр, которая инвертирует порядок байт в потоке данных. Программа должна обрабатывать входной поток, меняя порядок байт таким образом, чтобы первый байт становился последним, второй – предпоследним, и так далее. Дополнительно необходимо реализовать возможность явного указания входного и выходного файлов через параметры командной строки. В случае отсутствия таких указаний программа должна работать с потоками ввода и вывода по умолчанию (*stdin* и *stdout*). Учитывая возможный бесконечный объем входных данных, в программе вводится искусственное ограничение на размер обрабатываемого буфера, что позволяет обеспечить корректную работу в рамках учебной задачи.

**1.2 Постановка задачи**

Программный код предоставлен в приложении А.

Для реализации поставленной задачи была создана многомодульная программа, состоящая из двух основных модулей: головного модуля, отвечающего за обработку аргументов командной строки, управление потоками ввода-вывода и запуск основных операций, и вспомогательного модуля, реализующего функцию инверсии данных. Головной модуль представлен в файле «*main.c*», а вспомогательный модуль содержит функцию «*reverse\_buffer*», определённую в файле «*reverse.c*».

Программа позволяет задавать входной файл с помощью опции «*-i*» и выходной файл с помощью опции «*-o*». Если выходной файл не указан, но стандартный вывод привязан к терминалу, программа завершает работу с сообщением об ошибке, предупреждая о невозможности корректного вывода результата. Это позволяет избежать некорректного отображения бинарных данных в терминале.

Основная логика работы программы заключается в следующем: данные из входного потока считываются в буфер, после чего вызывается функция «*reverse\_buffer*», которая осуществляет инверсию порядка байт. Обработанные данные записываются в выходной поток. Для обработки потенциально больших файлов и обеспечения корректной работы с памятью введено ограничение на размер буфера в 10 МБ, что является разумным компромиссом между производительностью и потреблением ресурсов.

Для автоматизации процесса сборки и тестирования проекта создан *makefile*. В *makefile* предусмотрены цели для компиляции программы, очистки временных файлов и тестирования [1]. Цель «*clean*» удаляет промежуточные объектные файлы и исполняемый файл, а цель «*test*» выполняет автоматическое тестирование программы с использованием заранее подготовленных входных данных и сравнивает результаты работы программы с ожидаемыми. Это позволяет быстро проверять корректность работы программы в различных сценариях [2].

Таким образом, разработанная программа соответствует поставленной задаче и демонстрирует практическое применение инструментов разработки, системных вызовов, управления вводом-выводом и работы с памятью.

При ручном тестировании был предоставлен текстовый файл с текстом «*qwerty*» (рисунок 1.1) и в результате получен текст «*ytrewq*» (рисунок 1.2).

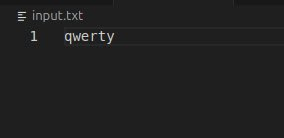


Рисунок 1.1 – Входной файл

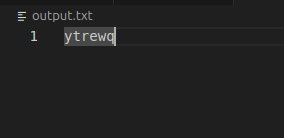


Рисунок 1.2 – Результат работы программы

**ЗАКЛЮЧЕНИЕ**

В ходе выполнения лабораторной работы была разработана программа-фильтр, которая осуществляет инверсию порядка байт в потоке данных. Программа успешно реализует поставленную задачу, предоставляя возможность указания входного и выходного файлов через параметры командной строки, а также корректно работает с потоками ввода-вывода по умолчанию (*stdin* и *stdout*). Для обработки больших объемов данных в программе введено ограничение на размер буфера, что позволяет избежать проблем, связанных с исчерпанием памяти.

Проект был реализован в многомодульной структуре, что обеспечило удобство разработки, сопровождения и повторного использования кода. Основные функции программы разделены на модули: головной модуль отвечает за управление процессом обработки данных, а вспомогательный модуль реализует функцию инверсии. Это позволяет легко добавлять новые функциональные возможности или изменять существующие.

Для автоматизации процесса сборки и тестирования проекта был создан *makefile*. Это позволило упростить управление проектом, обеспечив выполнение таких задач, как сборка программы, удаление промежуточных файлов и автоматическое тестирование с использованием заранее подготовленных тестовых данных. Проведенные тесты подтвердили корректность работы программы в различных сценариях, включая работу с файлами и стандартными потоками.

В результате выполнения работы были изучены и на практике применены основные инструменты разработки: компилятор *GCC*, утилита *make* и язык *makefile*. Также были закреплены навыки работы с системными вызовами, стандартными библиотеками *C*, потоками ввода-вывода и управления памятью. Полученные знания и опыт могут быть использованы для разработки более сложных проектов и решения практических задач в программировании.

**СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ**

[1] Makefile для самых маленьких [Электронный ресурс]. – Режим доступа: https://habr.com/ru/articles/155201/.

[2] Автоматизируем работу с контейнерами через Makefile: сборка, тестирование и развёртывание за один вызов make [Электронный ресурс]. – Режим доступа: https://habr.com/ru/companies/macloud/articles/566046/.

ПРИЛОЖЕНИЕ А  
(справочное)  
Исходный код

Листинг А.1 – Главная программа

#include <stdio.h>

#include <stdlib.h>

#include <unistd.h>

#include <string.h>

#include "reverse.h"

#define MAX\_BUFFER\_SIZE (10 \* 1024 \* 1024) // 10 МБ

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

char \*input\_file = NULL;

char \*output\_file = NULL;

int opt;

while ((opt = getopt(argc, argv, "i:o:")) != -1) {

switch (opt) {

case 'i':

input\_file = optarg;

break;

case 'o':

output\_file = optarg;

break;

default:

fprintf(stderr, "Usage: %s [-i input] [-o output]\n", argv[0]);

exit(EXIT\_FAILURE);

}

}

if (optind < argc) {

if (input\_file == NULL) {

input\_file = argv[optind];

optind++;

}

if (optind < argc && output\_file == NULL) {

output\_file = argv[optind];

optind++;

}

if (optind < argc) {

fprintf(stderr, "Too many arguments\n");

exit(EXIT\_FAILURE);

}

}

FILE \*in\_stream = stdin;

FILE \*out\_stream = stdout;

if (input\_file != NULL) {

in\_stream = fopen(input\_file, "rb");

if (!in\_stream) {

perror("Failed to open input file");

exit(EXIT\_FAILURE);

}

}

if (output\_file != NULL) {

out\_stream = fopen(output\_file, "wb");

if (!out\_stream) {

perror("Failed to open output file");

if (in\_stream != stdin) fclose(in\_stream);

exit(EXIT\_FAILURE);

}

} else if (isatty(fileno(stdout))) {

fprintf(stderr, "Error: Output file not specified and stdout is a terminal\n");

if (in\_stream != stdin) fclose(in\_stream);

exit(EXIT\_FAILURE);

}

unsigned char \*buffer = malloc(MAX\_BUFFER\_SIZE);

if (buffer == NULL) {

perror("Failed to allocate memory");

if (in\_stream != stdin) fclose(in\_stream);

if (out\_stream != stdout) fclose(out\_stream);

exit(EXIT\_FAILURE);

}

size\_t bytes\_read = fread(buffer, 1, MAX\_BUFFER\_SIZE, in\_stream);

if (bytes\_read == 0) {

fprintf(stderr, "No data read from input\n");

free(buffer);

if (in\_stream != stdin) fclose(in\_stream);

if (out\_stream != stdout) fclose(out\_stream);

exit(EXIT\_FAILURE);

}

reverse\_buffer(buffer, bytes\_read);

size\_t bytes\_written = fwrite(buffer, 1, bytes\_read, out\_stream);

if (bytes\_written != bytes\_read) {

perror("Error writing output");

free(buffer);

if (in\_stream != stdin) fclose(in\_stream);

if (out\_stream != stdout) fclose(out\_stream);

exit(EXIT\_FAILURE);

}

free(buffer);

if (in\_stream != stdin) fclose(in\_stream);

if (out\_stream != stdout) fclose(out\_stream);

return EXIT\_SUCCESS;

}

Листинг А.2 – Заголовочный файл функции

#ifndef REVERSE\_H

#define REVERSE\_H

#include <stddef.h>

void reverse\_buffer(unsigned char \*buffer, size\_t size);

#endif

Листинг А.3 – Реализация функции

#include "reverse.h"

void reverse\_buffer(unsigned char \*buffer, size\_t size) {

if (buffer == NULL || size == 0) {

return;

}

for (size\_t i = 0; i < size / 2; i++) {

unsigned char temp = buffer[i];

buffer[i] = buffer[size - 1 - i];

buffer[size - 1 - i] = temp;

}

}

Листинг А.4 – Make файл

CC = gcc

CFLAGS = -Wall -Wextra -std=gnu99

OBJECTS = main.o reverse.o

TARGET = invert

all: $(TARGET)

$(TARGET): $(OBJECTS)

$(CC) $(CFLAGS) -o $@ $^

main.o: main.c reverse.h

$(CC) $(CFLAGS) -c main.c

reverse.o: reverse.c reverse.h

$(CC) $(CFLAGS) -c reverse.c

clean:

rm -f $(OBJECTS) $(TARGET)

test: $(TARGET)

@echo "Running tests..."

@echo -n "hello" > test\_input.txt

@./$(TARGET) -i test\_input.txt -o test\_output.txt

@echo -n "olleh" > expected\_output.txt

@cmp test\_output.txt expected\_output.txt && echo "Test 1 passed" || echo "Test 1 failed"

@rm -f test\_input.txt test\_output.txt expected\_output.txt

@echo -n "abc" > test\_input.txt

@./$(TARGET) < test\_input.txt > test\_output.txt

@echo -n "cba" > expected\_output.txt

@cmp test\_output.txt expected\_output.txt && echo "Test 2 passed" || echo "Test 2 failed"

@rm -f test\_input.txt test\_output.txt expected\_output.txt