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.**

Студент И. В. Бобко

Преподаватель Н. Ю. Гриценко

Минск 2024

СОДЕРЖАНИЕ

[1 Цель работы 3](#_Toc146752068)

[2 Теоретические сведения 4](#_Toc146752069)

[3 Результат выполнения 6](#_Toc146752070)

[Заключение 6](#_Toc146752071)

[Список использованных источников 7](#_Toc146752072)

[Приложение А (обязательное) Листинг кода 8](#_Toc146752073)

1. **ЦЕЛЬ РАБОТЫ**

Целью данной работы является изучение среды программирования и основных инструментов: компилятор/сборщик («коллекция компиляторов») gcc, управление обработкой проекта make (и язык makefile), библиотеки и т.д. Практически использовать основные библиотеки и системные вызовы: ввод-вывод и работа с файлами, обработка текста.

В ходе работы будет написана программа, представляющая собой инвертирующий фильтр для строк.

1. **ТЕОРЕТИЧЕСКИЕ СВЕДЕНИЯ**

GCC (GNU Compiler Collection) – это коллекция компиляторов, разрабатываемая проектом GNU. Она включает в себя компиляторы для различных языков программирования, таких как C, C++, Objective-C, Fortran, Ada и других [1].

GCC является одним из наиболее популярных и широко используемых компиляторов в сообществе разработчиков. Он доступен для различных операционных систем, таких как Linux, macOS, Windows и других.

GCC предоставляет мощные и гибкие инструменты для компиляции и сборки программного кода. Он преобразует исходный код программы, написанной на одном из поддерживаемых языков, в машинный код, который может быть исполнен компьютером.

Makefile – это текстовый файл, который содержит инструкции для сборки и управления процессом компиляции и сборки программного проекта. Он используется в среде разработки для автоматизации процесса сборки программного кода и управления зависимостями между файлами [2].

Основная цель Makefile состоит в том, чтобы определить, какие файлы в проекте нуждаются в перекомпиляции и в каком порядке это должно происходить. Он содержит набор правил, каждое из которых определяет зависимости между файлами и команды, которые нужно выполнить для получения конечного исполняемого файла или библиотеки.

Makefile использует утилиту make для автоматической обработки правил и определения, какие файлы должны быть перекомпилированы на основе изменений в исходных файлах.

В Makefile обычно присутствуют следующие элементы:

* Объявление переменных. переменные используются для хранения значений, таких как пути к файлам, флаги компилятора и другие параметры.
* Цели (targets). цели определяют конечные результаты, которые нужно получить в результате сборки проекта. Например, цель «all» может указывать на сборку всех исполняемых файлов или библиотек.
* Правила (rules). правила определяют зависимости между файлами и команды, которые нужно выполнить для сборки конечного результата. Они состоят из цели, списка зависимостей и команд, которые нужно выполнить для получения цели.

1. **РЕЗУЛЬТАТ ВЫПОЛНЕНИЯ**

В результате лабораторной работы была написана программа-фильтр для инвертирования строк.

Программа реализует инверсию порядка строк потока. Длинна строк ограниченна 1024 символами. Программа получает на вход файл с исходным текстом (рисунок 1) и файл, в который будет записан результат выполнения программы (рисунок 2).

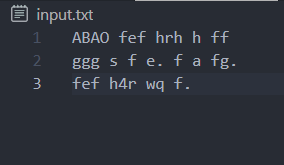


Рисунок 1 – Файл с исходным текстом

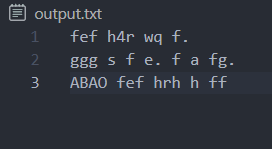


Рисунок 2 – Файл с измененным текстом

# ЗАКЛЮЧЕНИЕ

В ходе данной лабораторной работы были изучены изучение среды программирования и основных инструментов: компилятор/сборщик («коллекция компиляторов») gcc, управление обработкой проекта make (и язык makefile), библиотеки и т.д. Практически использовались основные библиотеки и системные вызовы: ввод-вывод и работа с файлами, обработка текста.

В ходе работы была написана программа, представляющая собой инвертирующий фильтр для строк.

# СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Компилятор GCC [Электронный ресурс]. – Режим доступа: https://parallel.uran.ru/book/export/html/25 – Дата доступа: 19.02.2024.
2. Что такое makefile [Электронный ресурс]. – Режим доступа: https://guides.hexlet.io/ru/makefile-as-task-runner/ – Дата доступа: 19.02.2024.

# ПРИЛОЖЕНИЕ А (обязательное) Листинг кода

Листинг 1 – Файл main.c

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include "text\_utils.h"

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

FILE \*inputFile;

FILE \*outputFile;

if (argc < 2) {

printf("Usage: %s <input\_file> [output\_file]\n", argv[0]);

return 1;

}

inputFile = fopen(argv[1], "r");

if (!inputFile) {

perror("Error opening input file");

return 2;

}

outputFile = (argc >= 3) ? fopen(argv[2], "w") : stdout;

if (!outputFile) {

perror("Error opening output file");

return 3;

}

if (argc == 4 && argv[3][0] == 'N') {

invertLinesN(inputFile, outputFile);

} else {

invertLines(inputFile, outputFile);

}

fclose(inputFile);

if (argc >= 3) {

fclose(outputFile);

}

return 0;

}

Листинг 2 – Файл text\_utils.h

#ifndef TEXT\_UTILS\_H

#define TEXT\_UTILS\_H

#include <stdio.h>

void invertLines(FILE \*inputFile, FILE \*outputFile);

void invertLinesN(FILE \*inputFile, FILE \*outputFile);

#endif

Листинг 3 – Файл text\_utils.c

#include "text\_utils.h"

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#define MAX\_LINE\_LENGTH 1024

void invertLines(FILE \*inputFile, FILE \*outputFile) {

char \*\*lines = malloc(MAX\_LINE\_LENGTH \* sizeof(char \*));

char line[MAX\_LINE\_LENGTH];

int lineCount = 0;

while (fgets(line, MAX\_LINE\_LENGTH, inputFile) != NULL) {

lines[lineCount] = strdup(line);

lineCount++;

}

for (int i = lineCount - 1; i >= 0; i--) {

fputs(lines[i], outputFile);

free(lines[i]);

}

free(lines);

}

Листинг 4 – Файл makefile

CC = gcc

CFLAGS = -Wall -g

OBJ = main.o text\_utils.o

EXEC = invert\_filter

$(EXEC): $(OBJ)

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

main.o: main.c text\_utils.h

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

text\_utils.o: text\_utils.c text\_utils.h

$(CC) $(CFLAGS) -c text\_utils.c

clean:

rm -f $(OBJ) $(EXEC)

test: $(EXEC)

./$(EXEC) input.txt output.txt

testN: $(EXEC)

./$(EXEC) input.txt output.txt N