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 С ПОД UNIX. ИНТСРУМЕНТАРИЙ ПРОГРАММИСТА В UNIX**

Студент В. П. Бычко

Преподаватель Н. Ю. Гриценко

Минск 2024

СОДЕРЖАНИЕ

[1 Постановка задачи 3](file:///E:\Downloads\Telegram%20Desktop\лабораторная%20работа%205%20(2).docx#_Toc146631498)

[2 Краткие теоретические сведения 4](file:///E:\Downloads\Telegram%20Desktop\лабораторная%20работа%205%20(2).docx#_Toc146631499)

[3 Результаты выполнения лабораторной работы 6](file:///E:\Downloads\Telegram%20Desktop\лабораторная%20работа%205%20(2).docx#_Toc146631500)

[Выводы 7](file:///E:\Downloads\Telegram%20Desktop\лабораторная%20работа%205%20(2).docx#_Toc146631501)

[Приложение А (обязательное) Листинг кода 9](file:///E:\Downloads\Telegram%20Desktop\лабораторная%20работа%205%20(2).docx#_Toc146631503)

## 1 ПОСТАНОВКА ЗАДАЧИ

Целью выполнения данной лабораторной работы является 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. Реализовать программу, производящую инверсию порядка байт в потоке (первый → последний, второй → предпоследний, и т.д.). Создать makefile для управления обработкой проекта. Среди целей должны быть сборка и очистка проекта, также тестирование исполняемого файла с заранее заготовленными входными данными.

## 2 КРАТКИЕ ТЕОРЕТИЧЕСКИЕ СВЕДЕНИЯ

Система действительно предоставляет практически все необходимые компоненты: стандартные средства работы с текстами, компилятор, отладчик, библиотеки, утилита make естественным образом присутствуют в ней, а shell играет роль своего рода «IDE». Полнота набора зависит от конкретного дистрибутива и построенной ее конфигурации.

Задача компилятора заключается в преобразовании исходного кода программы в двоичный формат, включая подключение необходимых файлов. В мире Unix существует несколько распространенных компиляторов, таких как cc (C Compiler) и gcc (GNU Compiler Collection), который изначально был разработан как GNU C Compiler. Сегодняшний gcc представляет собой набор компиляторов различных языков, что обеспечивает поддержку множества языков программирования, таких как C/C++, Objective-C/C++, Fortran, Ada, D, Go и BRIG (HSAIL).

Сложность и разнообразие вызовов компиляторов, а также необходимость учета взаимосвязей файлов и внешних параметров, приводят к потребности в автоматизации процесса обработки проекта. Простейшим решением этой задачи может быть использование сценариев командного интерпретатора, которые учитывают все возможные сценарии использования. Это решение функционально, но может быть неудобным для пользователя. В качестве альтернативы существует специализированное средство - утилита make (и ее аналоги).

Утилита make предназначена для автоматизации процесса сборки проектов. Она автоматически определяет, какие части большой программы должны быть перекомпилированы, и осуществляет необходимые действия для этого.

Каждый файл сценария должен содержать хотя бы одно правило. Правило состоит из цели и ее зависимостей, а также одной или нескольких команд, необходимых для достижения этой цели. Существуют две категории целей: цели-файлы, которые считаются достигнутыми, если соответствующий файл существует и актуален, и абстрактные цели (метки).

Переменные в сценарии могут быть только строками. В make поддерживаются предопределенные переменные, такие как "$@", "$^", "$<", которые представляют соответственно имя цели, список зависимостей и имя первой зависимости.

Символы "\*" и "%" в make используются для подстановки (wildcards), но их значение отличается от значений в именах файловой системы. Функция $(patsubst pattern,replacement,text) заменяет в тексте слова, соответствующие шаблону, на указанную подстановку. Если в шаблоне используется символ "%", он соответствует любому количеству символов внутри слова.

Кроме того, существуют утилиты общего назначения, которые используются для анализа и корректировки файлов программ: grep для поиска строк в файлах с использованием регулярных выражений, diff для сравнения файлов и создания описания различий, patch для применения описания различий к исходному файлу, и ldd для вывода списка зависимостей исполняемого файла от внешних библиотек.

## 3 РЕЗУЛЬТАТЫ ВЫПОЛНЕНИЯ ЛАБОРАТОРНОЙ РАБОТЫ

В ходе выполнения лабораторной работы была реализована программа, инвертирующая поток байт. На рисунке 3.1 представлен результат запуска программы, инвертирующей байты в потоке



Рисунок 3.1 – Выполнение программы

На рисунке 3.2 представлен текст файла, в котором содержались входные данные



Рисунок 3.2 – Итоговый файл шифрования

На рисунке 3.3 представлен запуск цели тестов makefile.



Рисунок 3.3 – Запуск тестов makefile

## ВЫВОДЫ

В ходе выполнения данной лабораторной работы были изучены 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. Реализована программа, производящая инверсию порядка байт в потоке (первый → последний, второй → предпоследний, и т.д.). Создан makefile для управления обработкой проекта. Среди целей присутвуют сборка и очистка проекта, также тестирование исполняемого файла с заранее заготовленными входными данными.

## ПРИЛОЖЕНИЕ А

## (обязательное)

## Листинг кода

**Makefile**

TARGET = BufferInversion

CC = gcc

Source\_Folder = ./src/

Object\_Folder = ./obj/

Test\_Folder = ./tests/

SRC = $(wildcard $(Source\_Folder)\*.c)

$(TARGET) : ./obj/Inverter.o ./obj/Source.o

$(CC) ./obj/Inverter.o ./obj/Source.o -o $(TARGET)

$(Object\_Folder)%.o : $(Source\_Folder)%.c

$(CC) -c $< -o $@

clean :

@rm $(wildcard $(Object\_Folder)\*.o)

tests : SimpleTest NewLineTest

SimpleTest :

@echo "12345" > $(Test\_Folder)input.bin

@echo "\n54321" > $(Test\_Folder)answer.bin

@./BufferInversion $(Test\_Folder)input.bin $(Test\_Folder)output.bin

@diff $(Test\_Folder)output.bin $(Test\_Folder)answer.bin

@rm $(Test\_Folder)\*.bin

NewLineTest :

@echo "123\n456" > $(Test\_Folder)input.bin

@echo "\n654\n321" > $(Test\_Folder)answer.bin

@./BufferInversion $(Test\_Folder)input.bin $(Test\_Folder)output.bin

@diff $(Test\_Folder)output.bin $(Test\_Folder)answer.bin

@rm $(Test\_Folder)\*.bin