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             В. А. Шайкова

Проверил                          Н. Ю. Гриценко

Минск 2024

СОДЕРЖАНИЕ

[1 Постановка задачи 3](#_Toc157960226)

[2 Теоретические сведения 4](#_Toc157960227)

[3 Результат выполнения 5](#_Toc157960235)

[Выводы 6](#_Toc157960236)

[Список использованных источников 7](#_Toc157960237)

[Приложение А (обязательное) Листинг кода 8](#_Toc157960238)

## 1 ПОСТАНОВКА ЗАДАЧИ

Целью выполнения данной лабораторной работы является изучение среды программирования и основных инструментов: компилятор/сборщик gcc, управление обработкой проекта make и языка makefile, библиотеки. Практическое использование основных библиотек и системных вызовов: ввод-вывод, работа с файлами, обработка текста, распределение памяти, управление выполнением.

## 2 ТЕОРЕТИЧЕСКИЕ СВЕДЕНИЯ

Unix представляет собой семейство многозадачных и многопользовательских операционных систем, изначально разработанных в Bell Labs в начале семидесятых годов. [1]

В Unix часто используется компилятор GCC (GNU Compiler Collection), который поддерживает множество языков программирования, включая C. Сборка проектов обычно осуществляется с помощью утилиты make и языка описания сборки makefile.

При использовании компилятора GCC (GNU Compiler Collection) для компиляции исходного кода происходят следующие этапы:

Препроцессинг (Preprocessing): Препроцессор GCC выполняет предварительную обработку исходного кода. Он обрабатывает директивы препроцессора и удаляет комментарии. Результатом этого этапа является расширенный исходный код без директив препроцессора.

Компиляция (Compilation): Компилятор GCC преобразует измененный исходный код на языке C в язык ассемблера. Он проверяет синтаксис и семантику кода, выполняет оптимизации. Результатом этапа компиляции является файлы объектных кодов (.o), которые содержат машинные инструкции на языке ассемблера.

Ассемблирование (Assembly): На этом этапе ассемблер GCC преобразует файлы объектных кодов, созданные на предыдущем этапе, в машинный код, специфичный для целевой архитектуры процессора. Ассемблер объединяет все объектные файлы и создает исполняемый файл или динамическую библиотеку.

Связывание (Linking): На последнем этапе линковщик GCC (ld) объединяет исполняемые файлы с другими объектными файлами и библиотеками, необходимыми для работы программы. В этом процессе разрешаются ссылки на символы между различными файлами, и создается окончательный исполняемый файл или библиотека. Линковщик также может выполнять дополнительные оптимизации и проверки, а также добавлять информацию о символах и таблицы переходов для поддержки динамической загрузки и связывания. [2]

Make - это утилита командной строки, которая управляет сборкой программного кода, определяя зависимости между файлами и выполняя нужные действия для обновления только измененных файлов.

Чтобы определить, как модули должны быть скомпилированы или перекомпилированы вместе, make использует пользовательские makefile.  Он содержит инструкции о том, как собрать программу, какие исходные файлы и зависимости необходимы, а также какие команды должны быть выполнены для компиляции и связывания кода. Makefile также позволяет определить пользовательские переменные, которые могут быть использованы в процессе сборки для управления параметрами компиляции и другими настройками. [3]

# 3 РЕЗУЛЬТАТ ВЫПОЛНЕНИЯ

В результате выполнения лабораторной работы было создано приложение, выполняющее перевод латинского текста в морзянку и наоборот. В первом случае возможен ввод как с клавиатуры, так и из файла latin.txt (рисунок 1):

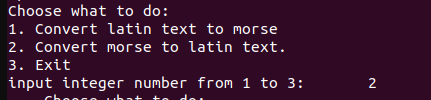


Рисунок 1 – Основное меню

При переводе латиницы в морзянку происходит поиск символа в описанном в файле lab3.c алфавите и запись соответствующей ему строки в файл morse.txt. При обнаружении неизвестного символа происходит запись предусмотренной строки, некоторые символы (например, ‘\n’ и ‘\0’) опускаются. Также возможен обратный перевод в латиницу данных из файла morse.txt в файл latin.txt, если первый файл существует (рисунок 2).



Рисунок 2 – Содержимое файлов

Буквенные символы в алфавите представлены в верхнем регистре. Поэтому ввод текста может быть осуществлен в любом из регистров, но вывод – только в верхнем. Многострочный ввод поддерживается при вводе текста из файла.

## ВЫВОДЫ

В ходе выполнения данной лабораторной работы было осуществлено изучение среды программирования и основных инструментов, таких как сборщик, компилятор (gcc), управление обработкой проекта с помощью make и язык makefile, а также библиотеки, используемые при разработке программного обеспечения под операционную систему Unix. Практическое применение основных библиотек и системных вызовов, таких как ввод и вывод, работа с файлами, обработка текста, распределение памяти и управление выполнением, также было проведено.

При помощи сборки с использованием makefile было создано приложение, осуществляющее перевод текста в морзянку и обратно, а также предусмотрен пользовательский ввод, обработка многострочного текста при работе с файлами и обработка некорректных символов, не предусмотренных алфавитом.

## СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Введение в Unix [Электронный ресурс]. – Режим доступа: https://unix.com. – Дата доступа: 09.03.2024.
2. Compilation with GCC [Электронный ресурс]. – Режим доступа: https://coffeebeforearch.github.io/2020/09/06/compilation.html. – Дата доступа: 09.03.2024.
3. Unix Makefile Tutorial [Электронный ресурс]. – Режим доступа: https://www.tutorialspoint.com/makefile – Дата доступа: 09.03.2024.

## ПРИЛОЖЕНИЕ А

## (обязательное)

## Листинг кода

Листинг 1 – Файл lab3.c

#include <stdio.h>  
#include <stdbool.h>  
#include <string.h>  
#include <unistd.h>  
#include "consts.h"  
  
Pair alphabet[] = {{'A', ".-"},{'B', "-..."},{'C', "-.-."},{'D', "-.."},{'E', "."},{'F', "..-."},  
 {'G', "--."},{'H', "...."},{'I', ".."},{'J', ".---"},{'K', "-.-"},{'L', ".-.."},{'M', "--"},  
 {'N', "-."},{'O', "---"},{'P', ".--."},{'Q', "--.-"},{'R', ".-."},{'S', "..."},{'T', "-"},  
 {'U', "..-"},{'V', "...-"},{'W', ".--"},{'X', "-..-"},{'Y', "-.--"},{'Z', "--.."},  
  
 {' ', "-...-"},{'.', "......"},{',', ".-.-.-"},{'!', "--..--"},{'?', "..--.."},  
 {':', "---..."},{';', "-.-.-."},{'-', "-....-"},{'(', "-.--.-"},{')', "-.--.-"},  
 {'`', ".----."},{'\_', ".-..-."},{'@', ".--.-."},{'/', "--..-."},  
  
 {'0', "-----"},{'1', ".----"},{'2', "..---"},{'3', "...--"},{'4', "....-"},  
 {'5', "....."},{'6', "-...."},{'7', "--..."},{'8', "---.."},{'9', "----."},  
 {'+', ".-.-"},{'\*', "..-.."},{'\\', "--.--"}  
};  
int alphabet\_size = sizeof(alphabet)/sizeof(alphabet[0]);  
const char\* latin\_file = "latin.txt";  
const char\* morse\_file = "morse.txt";  
  
int main(){  
while(1==1){  
 printf("Choose what to do:\n1. Convert latin text to morse\n2. Convert morse to latin text.\n3. Exit\n");  
 int option = get\_number(1,3);  
 if (option == 1){  
  printf("1. Input from keyboard\n2. Input from file\n");  
  int way = get\_number(1,2);  
  if (way==1){  
   char input[256];  
   printf("Input your text:\n");  
   getchar();  
   fgets(input, sizeof(input), stdin);  
     
   FILE\* mfw = fopen(morse\_file, "w");  
   put\_string\_to\_morse(input, mfw, alphabet, alphabet\_size);  
   fclose(mfw);  
  } else if (way==2){  
   FILE\* lfr = fopen(latin\_file, "r");  
   if(lfr == NULL){  
    printf("file does not exist");  
    continue;  
   }  
   char buffer[256];  
   FILE\* mfw = fopen(morse\_file, "w");  
   while((fgets(buffer, 256, lfr))!=NULL){  
    put\_string\_to\_morse(buffer, mfw, alphabet, alphabet\_size);  
    fprintf(mfw, "\n");  
   }  
   fclose(mfw);  
   fclose(lfr);  
  }  
 }if (option == 2) {  
      if (access(morse\_file, F\_OK) != -1) {  
          FILE\* mfr = fopen(morse\_file, "r");  
          FILE\* lfw = fopen(latin\_file, "w");  
          if (mfr == NULL || lfw == NULL) {  
               printf("Error while opening file(s)");  
          }  
          char buffer[256 \* 7];  
          while (fgets(buffer, 256 \* 7, mfr) != NULL) {  
               char\* token = strtok(buffer, " ");  
               printf("%s", token);  
               while (token != NULL) {  
                   char symbol = morse\_to\_latin(alphabet, alphabet\_size, token);  
                   fprintf(lfw, "%c", symbol);  
                   token = strtok(NULL, " "); // Исправленное обновление значения token  
               }  
               fprintf(lfw, "\n");  
          }  
          fclose(mfr);  
          fclose(lfw);  
      }  
 } else{break;}}  
}

Листинг 2 – Файл funcs.c

#include <stdio.h>  
#include <string.h>  
#include <unistd.h>  
#include "consts.h"  
  
int get\_number(int min, int max){  
 int answer;  
 do{  
  printf("input integer number from %d to %d:\t", min, max);  
  if(scanf("%d", &answer) != 1){  
   while (getchar() != '\n');  
  } else if (answer < min || answer > max){  
   printf("Number does not belong to interval needed");  
  }  
 }while(answer < min || answer > max);  
 return answer;}  
  
const char\* latin\_to\_morse(const Pair\* alphabet, int size, char character){  
 char upper = character;  
 if (character == 0 || character == 10) return "";  
 if(character >= 'a' && character <= 'z'){  
  upper = character - 'a' + 'A';}  
 for(int i = 0; i < size; i++){  
  if(alphabet[i].latin==upper){  
   return alphabet[i].morse;  
  }  
 }  
 return "--------";  
}  
  
char morse\_to\_latin(const Pair\* alphabet, int size, const char\* token){  
 if(strcmp(token, "--------")==0){return '$';}  
        if(strcmp(token, "")==0){return ' ';}  
 for (int i = 0; i < size; i++){  
  if (strcmp(token, alphabet[i].morse) == 0) {  
                     return alphabet[i].latin;  
                }  
 }  
 return ' ';  
}  
  
void put\_string\_to\_morse(char\* str, FILE\* mfw, const Pair\* alphabet, int alphabet\_size){  
 char\* token = strtok(str, " ");  
 while (token!=NULL){  
  int length = strlen(token);  
  for (int i = 0; i < length; i++){  
   char character = token[i];  
   const char\* code = latin\_to\_morse(alphabet, alphabet\_size, character);  
   fprintf(mfw, "%s ", code);  
  }  
  fprintf(mfw, "%s ", latin\_to\_morse(alphabet, alphabet\_size, ' '));  
  
  token = strtok(NULL, " ");}  
}

Листинг 3 – Файл consts.h

#ifndef DATA\_H  
#define DATA\_H  
  
#include <stdio.h>  
typedef struct{  
 char latin;  
 char morse[10];  
}Pair;  
int get\_number(int min, int max);  
const char\* latin\_to\_morse(const Pair\* alphabet, int size, char character);  
char morse\_to\_latin(const Pair\* alphabet, int size, const char\* token);  
void put\_string\_to\_morse(char\* str, FILE\* mfw, const Pair\* alphabet, int alphabet\_size);  
  
#endif  
  
Листинг 4 – Файл makefile

CC = gcc  
CFLAGS = -Wall -Wextra  
TARGET = morse  
SOURCES = lab3.c funcs.c  
OBJECTS = $(SOURCES:.c=.o)  
.PHONY: all clean run  
all: $(TARGET)  
$(TARGET): $(OBJECTS)  
 $(CC) $(CFLAGS) $(OBJECTS) -o $(TARGET)  
run: $(TARGET)  
 ./$(TARGET)  
%.o: %.c  
 $(CC) $(CFLAGS) -c $< -o $@  
clean:  
 rm -f $(OBJECTS) $(TARGET)