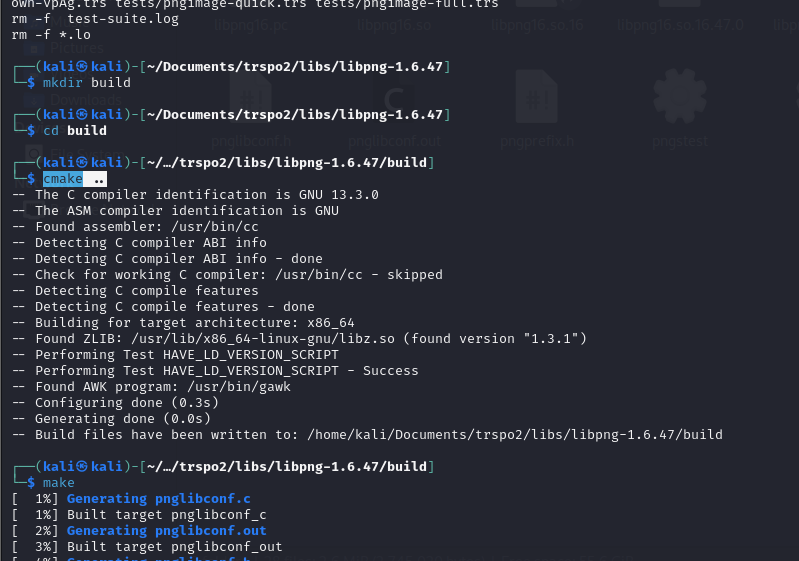
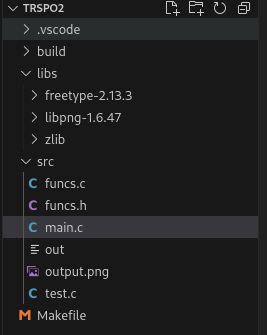
Перед тем, как начинать программировать надо собрать библиотеки командой make

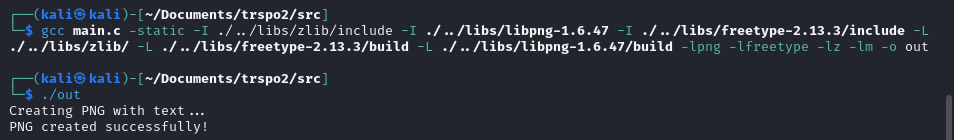


Например, libpng

Далее структурирован репозиторий



Потом была сделана тестовая программа, которая выводит текст в картинку. Программа собрана и работает





# Компилятор и флаги

CC = gcc

CFLAGS = -I ./src -I ./libs/zlib/include -I ./libs/libpng -I ./libs/libpng/include -I ./libs/freetype/include

LDFLAGS = -L ./libs/zlib/lib -L ./libs/libpng/build -L ./libs/freetype/build

LDLIBS = -lpng -lfreetype -lz -lm

# Директории

SRC\_DIR = src

BUILD\_DIR = build

OBJ\_DIR = $(BUILD\_DIR)/obj

BIN\_DIR = $(BUILD\_DIR)/bin

# Цели для разных версий

TARGET=BIN\_DIR

# Исходные файлы и объекты

SRCS = $(wildcard $(SRC\_DIR)/\*.c)

TARGET\_STATIC = $(BIN\_DIR)/app-static

TARGET\_DYNAMIC = $(BIN\_DIR)/app-dynamic

TARGET\_BLOB = $(BIN\_DIR)/app-blob

# Основная цель

all: static dynamic #blob

# Статическая версия (полностью статически линкованная)

static: $(TARGET\_STATIC)

# Сборка исполняемого файла

$(TARGET\_STATIC): $(OBJS) | $(STATIC\_DIR)

$(CC) src/main.c -static $(CFLAGS) $(LDFLAGS) $^ $(LDLIBS) -o $@

# Создание директорий

$(BIN\_DIR):

mkdir -p $@

# Очистка

clean:

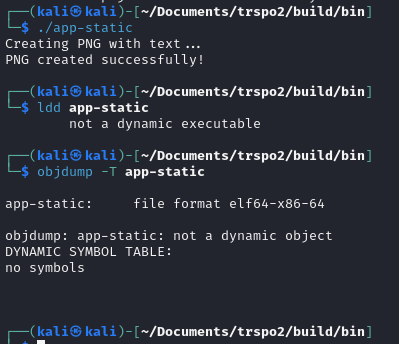
rm -rf $(BUILD\_DIR)

# Пересборка

rebuild: clean all

.PHONY: all clean rebuild

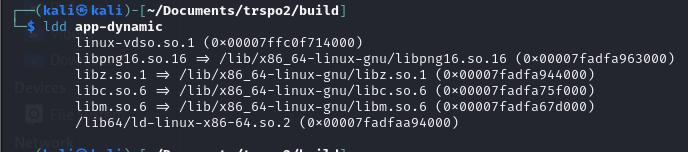
Проверка, что файл статической компоновки

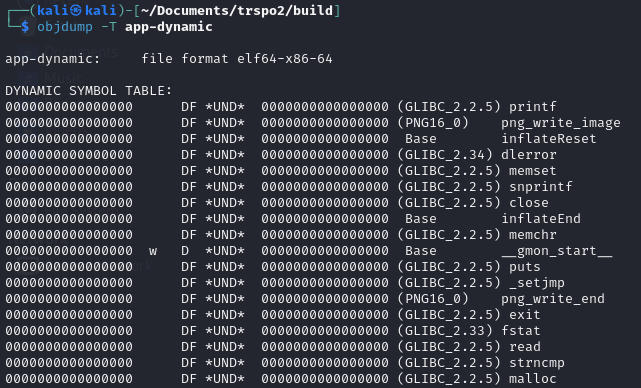


objdump -T app-dynamic

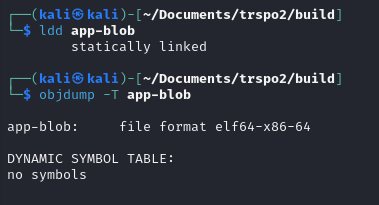
ldd app-dynamic

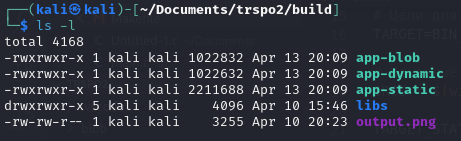
./app-static /usr/share/fonts/truetype/freefont/FreeSans.ttf output.png $'Hello, World!!!))' 36





Blob:





**Стат+динам**

**Main.c**

#ifdef BLOB

#else

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#endif

#ifdef IMPLICIT

#include <dlfcn.h>

#include <png.h>

#include <ft2build.h>

#include FT\_FREETYPE\_H

#define LOAD\_SYMBOL(handle, name) \

\*(void\*\*)(&name) = dlsym(handle, #name); \

if (!name) { \

fprintf(stderr, "Error loading symbol: %s\n", #name); \

exit(1); \

}

// Глобальные переменные для хранения указателей на функции

void\* libpng\_lib = NULL;

void\* freetype\_lib = NULL;

png\_structp (\*png\_create\_write\_structMY)(png\_const\_charp, png\_voidp, png\_error\_ptr, png\_error\_ptr) = NULL;

png\_infop (\*png\_create\_info\_structMY)(png\_const\_structrp) = NULL;

void (\*png\_destroy\_write\_structMY)(png\_structp, png\_infop) = NULL;

void (\*png\_init\_ioMY)(png\_structp, FILE\*) = NULL;

void (\*png\_set\_IHDRMY)(png\_const\_structrp, png\_inforp, png\_uint\_32, png\_uint\_32, int, int, int, int, int) = NULL;

void (\*png\_write\_infoMY)(png\_structrp, png\_const\_inforp) = NULL;

void (\*png\_write\_imageMY)(png\_structrp, png\_bytepp) = NULL;

void (\*png\_write\_endMY)(png\_structrp, png\_inforp) = NULL;

void (\*png\_set\_user\_limitsMY)(png\_structrp, png\_uint\_32, png\_uint\_32) = NULL;

// Указатели на функции freetype

FT\_Error (\*FT\_Init\_FreeTypeMY)(FT\_Library\*) = NULL;

FT\_Error (\*FT\_New\_FaceMY)(FT\_Library, const char\*, FT\_Long, FT\_Face\*) = NULL;

FT\_Error (\*FT\_Set\_Pixel\_SizesMY)(FT\_Face, FT\_UInt, FT\_UInt) = NULL;

void (\*FT\_Done\_FaceMY)(FT\_Face) = NULL;

void (\*FT\_Done\_FreeTypeMY)(FT\_Library) = NULL;

FT\_Error (\*FT\_Load\_CharMY)(FT\_Face, FT\_ULong, FT\_Int32) = NULL;

// Загрузка библиотек и функций

void load\_libraries() {

// Загружаем libpng

libpng\_lib = dlopen("./libs/libpng/libpng.so", RTLD\_LAZY);

if (!libpng\_lib) {

fprintf(stderr, "Error loading libpng: %s\n", dlerror());

exit(1);

}

// Загружаем функции libpng

png\_create\_write\_structMY = dlsym(libpng\_lib, "png\_create\_write\_struct");

png\_create\_info\_structMY = dlsym(libpng\_lib, "png\_create\_info\_struct");

png\_destroy\_write\_structMY = dlsym(libpng\_lib, "png\_destroy\_write\_struct");

png\_init\_ioMY = dlsym(libpng\_lib, "png\_init\_io");

png\_set\_IHDRMY = dlsym(libpng\_lib, "png\_set\_IHDR");

png\_write\_infoMY = dlsym(libpng\_lib, "png\_write\_info");

png\_write\_imageMY = dlsym(libpng\_lib, "png\_write\_image");

png\_write\_endMY = dlsym(libpng\_lib, "png\_write\_end");

// Загружаем freetype

freetype\_lib = dlopen("./libs/freetype/dynamic/libfreetype.so", RTLD\_LAZY);

if (!freetype\_lib) {

fprintf(stderr, "Error loading freetype: %s\n", dlerror());

dlclose(libpng\_lib);

exit(1);

}

// Загружаем функции freetype

FT\_Init\_FreeTypeMY = dlsym(freetype\_lib, "FT\_Init\_FreeType");

FT\_New\_FaceMY = dlsym(freetype\_lib, "FT\_New\_Face");

FT\_Set\_Pixel\_SizesMY = dlsym(freetype\_lib, "FT\_Set\_Pixel\_Sizes");

FT\_Done\_FaceMY = dlsym(freetype\_lib, "FT\_Done\_Face");

FT\_Done\_FreeTypeMY = dlsym(freetype\_lib, "FT\_Done\_FreeType");

FT\_Load\_CharMY = dlsym(freetype\_lib, "FT\_Load\_Char");

}

// Освобождение библиотек

void unload\_libraries() {

if (libpng\_lib) dlclose(libpng\_lib);

if (freetype\_lib) dlclose(freetype\_lib);

}

#else

#include <string.h>

#include <png.h>

#include <ft2build.h>

#include FT\_FREETYPE\_H

#define FT\_Load\_CharMY FT\_Load\_Char

#define FT\_Init\_FreeTypeMY FT\_Init\_FreeType

#define FT\_New\_FaceMY FT\_New\_Face

#define FT\_Set\_Pixel\_SizesMY FT\_Set\_Pixel\_Sizes

#define FT\_Done\_FaceMY FT\_Done\_Face

#define FT\_Done\_FreeTypeMY FT\_Done\_FreeType

#define png\_create\_write\_structMY png\_create\_write\_struct

#define png\_create\_info\_structMY png\_create\_info\_struct

#define png\_destroy\_write\_structMY png\_destroy\_write\_struct

#define png\_init\_ioMY png\_init\_io

#define png\_set\_IHDRMY png\_set\_IHDR

#define png\_write\_infoMY png\_write\_info

#define png\_write\_imageMY png\_write\_image

#define png\_write\_endMY png\_write\_end

#define png\_set\_user\_limitsMY png\_set\_user\_limits

#endif

typedef struct {

unsigned char red;

unsigned char green;

unsigned char blue;

unsigned char alpha;

} Pixel;

char create\_png\_with\_text(const char\* filename, const char\* font\_path, const char\* text, int font\_size) {

// Проверка входных параметров

if (!filename || !font\_path || !text || font\_size <= 0) {

fprintf(stderr, "Invalid input parameters\n");

return 0;

}

// Инициализация FreeType

FT\_Library ft\_lib;

if (FT\_Init\_FreeType(&ft\_lib)) {

fprintf(stderr, "Could not initialize FreeType library\n");

return 0;

}

// Загрузка шрифта

FT\_Face face;

if (FT\_New\_Face(ft\_lib, font\_path, 0, &face)) {

fprintf(stderr, "Could not open font %s\n", font\_path);

FT\_Done\_FreeType(ft\_lib);

return 0;

}

// Установка размера шрифта

FT\_Set\_Pixel\_Sizes(face, 0, font\_size);

// Расчет размеров изображения

int padding = font\_size;

int width = 0;

int height = 0;

// Вычисляем общую ширину и максимальную высоту текста

for (int i = 0; text[i] != '\0'; i++) {

if (FT\_Load\_Char(face, text[i], FT\_LOAD\_RENDER)) {

continue;

}

width += face->glyph->bitmap.width;

if (face->glyph->bitmap.rows > height) {

height = face->glyph->bitmap.rows;

}

printf("\n w=%d h=%d\n", width, height);

}

// Проверка на пустой текст

if (width == 0 || height == 0) {

width = 100; // Минимальные размеры если текст пустой

height = 30;

}

width += 2 \* padding;

width += (int)width/2;

height += 2 \* padding;

printf("\n w=%d h=%d\n", width, height);

// Создание изображения

Pixel\* image = (Pixel\*)malloc(width \* height \* sizeof(Pixel));

if (!image) {

fprintf(stderr, "Could not allocate memory for image\n");

FT\_Done\_Face(face);

FT\_Done\_FreeType(ft\_lib);

return 0;

}

// Заполнение фона (белый с прозрачностью)

for (int i = 0; i < width \* height; i++) {

image[i].red = 255;

image[i].green = 255;

image[i].blue = 255;

image[i].alpha = 0;

}

// Рендеринг текста (если текст не пустой)

if (strlen(text) > 0) {

int x\_offset = padding;

for (int i = 0; text[i] != '\0'; i++) {

if (FT\_Load\_Char(face, text[i], FT\_LOAD\_RENDER)) {

continue;

}

FT\_Bitmap\* bitmap = &face->glyph->bitmap;

for (int row = 0; row < bitmap->rows; row++) {

for (int col = 0; col < bitmap->width; col++) {

int img\_x = x\_offset + col + face->glyph->bitmap\_left;

int img\_y = padding + row + (height - 2 \* padding - bitmap->rows) / 2 +

(face->size->metrics.ascender >> 6);

if (img\_x >= 0 && img\_x < width && img\_y >= 0 && img\_y < height) {

unsigned char val = bitmap->buffer[row \* bitmap->pitch + col];

int idx = img\_y \* width + img\_x;

// Черный текст с альфа-каналом

image[idx].red = 0;

image[idx].green = 0;

image[idx].blue = 0;

image[idx].alpha = val;

}

}

}

x\_offset += face->glyph->advance.x >> 6;

}

}

// Сохранение PNG

FILE\* fp = fopen(filename, "wb");

if (!fp) {

fprintf(stderr, "Could not open file %s for writing\n", filename);

free(image);

FT\_Done\_Face(face);

FT\_Done\_FreeType(ft\_lib);

return 0;

}

png\_structp png\_ptr = png\_create\_write\_struct(PNG\_LIBPNG\_VER\_STRING, NULL, NULL, NULL);

if (!png\_ptr) {

fprintf(stderr, "Could not allocate png write struct\n");

fclose(fp);

free(image);

FT\_Done\_Face(face);

FT\_Done\_FreeType(ft\_lib);

return 0;

}

png\_infop info\_ptr = png\_create\_info\_struct(png\_ptr);

if (!info\_ptr) {

fprintf(stderr, "Could not allocate png info struct\n");

png\_destroy\_write\_struct(&png\_ptr, NULL);

fclose(fp);

free(image);

FT\_Done\_Face(face);

FT\_Done\_FreeType(ft\_lib);

return 0;

}

if (setjmp(png\_jmpbuf(png\_ptr))) {

fprintf(stderr, "Error during png creation\n");

png\_destroy\_write\_struct(&png\_ptr, &info\_ptr);

fclose(fp);

free(image);

FT\_Done\_Face(face);

FT\_Done\_FreeType(ft\_lib);

return 0;

}

png\_init\_io(png\_ptr, fp);

// Установка параметров PNG

png\_set\_IHDR(png\_ptr, info\_ptr, width, height,

8, PNG\_COLOR\_TYPE\_RGBA, PNG\_INTERLACE\_NONE,

PNG\_COMPRESSION\_TYPE\_BASE, PNG\_FILTER\_TYPE\_BASE);

png\_write\_info(png\_ptr, info\_ptr);

// Подготовка строк изображения для libpng

png\_bytep\* row\_pointers = (png\_bytep\*)malloc(sizeof(png\_bytep) \* height);

for (int y = 0; y < height; y++) {

row\_pointers[y] = (png\_bytep)(image + y \* width);

}

png\_write\_image(png\_ptr, row\_pointers);

png\_write\_end(png\_ptr, NULL);

// Очистка

free(row\_pointers);

png\_destroy\_write\_struct(&png\_ptr, &info\_ptr);

fclose(fp);

free(image);

FT\_Done\_Face(face);

FT\_Done\_FreeType(ft\_lib);

return 1;

}

void print\_usage(const char\* program\_name) {

printf("Usage: %s <font\_path> <output.png> <text> [font\_size]\n", program\_name);

printf("Example: %s /usr/share/fonts/FreeSans.ttf output.png \"Hello World\" 24\n");

}

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

#ifdef IMPLICIT

load\_libraries();

#endif

// Проверка аргументов командной строки

if (argc < 4) {

print\_usage(argv[0]);

return 1;

}

const char\* font\_path = argv[1];

const char\* output\_filename = argv[2];

const char\* text = argv[3];

int font\_size = 12; // Значение по умолчанию

if (argc >= 5) {

font\_size = atoi(argv[4]);

if (font\_size <= 0) {

fprintf(stderr, "Invalid font size, using default 12\n");

font\_size = 12;

}

}

printf("Creating PNG with text...\n");

printf("Font: %s\n", font\_path);

printf("Output file: %s\n", output\_filename);

printf("Text: %s\n", text);

printf("Font size: %d\n", font\_size);

if (create\_png\_with\_text(output\_filename, font\_path, text, font\_size)) {

printf("PNG created successfully!\n");

return 0;

} else {

printf("Failed to create PNG.\n");

return 1;

}

#ifdef IMPLICIT

unload\_libraries();

#endif

}

**Makefile**

# Компилятор и флаги

CC = gcc

FLAGS = -I ./$(PATH\_LIBPNG) -L ./$(BUILD\_LIBS)/libpng/ -I ./$(PATH\_FREETYPE)/include -L ./$(BUILD\_LIBS)/freetype -lfreetype

LDLIBS = -lfreetype -lpng -lz -lm

# Директории

SRC\_DIR = src

BUILD\_DIR = build

BUILD\_LIBS = $(BUILD\_DIR)/libs

PATH\_ZLIB = libs/zlib

PATH\_LIBPNG = libs/libpng

PATH\_FREETYPE = libs/freetype

OBJ\_DIR = $(BUILD\_DIR)/obj

# Цели для разных версий

TARGET=BIN\_DIR

# Исходные файлы и объекты

SRCS = $(wildcard $(SRC\_DIR)/\*.c)

TARGET\_STATIC = $(BUILD\_DIR)/app-static

TARGET\_DYNAMIC = $(BUILD\_DIR)/app-dynamic

TARGET\_BLOB = $(BUILD\_DIR)/app-blob

# Основная цель

all: static dynamic #blob

# Статическая версия (полностью статически линкованная)

static: zlib.a libpng.a freetype.a

# Сборка исполняемого файла

$(CC) -o $(TARGET\_STATIC) src/main.c -static $(FLAGS) $(LDLIBS)

dynamic\_neuavn: zlib.so libpng.so freetype.so

$(CC) -o $(TARGET\_DYNAMIC) src/main.c $(FLAGS) $(LDLIBS)

dynamic: zlib.so libpng.so freetype.so

$(CC) -o $(TARGET\_DYNAMIC) src/main.c $(FLAGS) $(LDLIBS) -DIMPLICIT

# Создание директорий

$(BIN\_DIR):

mkdir -p $@

zlib.so: zlib.a

libpng.so: libpng.a

zlib.a:

cmake $(PATH\_ZLIB) -B $(BUILD\_LIBS)/zlib

make -C $(BUILD\_LIBS)/zlib

libpng.a:

cmake $(PATH\_LIBPNG) -B $(BUILD\_LIBS)/libpng

make -C $(BUILD\_LIBS)/libpng

freetype.a:

cmake $(PATH\_FREETYPE) -B $(BUILD\_LIBS)/freetype

make -C $(BUILD\_LIBS)/freetype

freetype.so:

cmake $(PATH\_FREETYPE) -B $(BUILD\_LIBS)/freetype/dynamic -D BUILD\_SHARED\_LIBS=true -D CMAKE\_BUILD\_TYPE=Release

make -C $(BUILD\_LIBS)/freetype/dynamic

# Очистка

clean:

rm -rf $(BUILD\_DIR)

# Пересборка

rebuild: clean all

.PHONY: all clean rebuild