Министерство образования и науки Российской Федерации
Санкт-Петербургский политехнический университет Петра Великого
—
Институт компьютерных наук и кибербезопасности

ОТЧЕТ ПО ЛАБОРАТОРНОЙ РАБОТЕ №1

«Компилятор, компоновщик и динамические библиотеки»
по дисциплине «Технологии разработки современного программного обеспечения»

Выполнила
студентка гр. 5131001/10302 Куковякина Д. А.

<подпись>

Проверил
преподаватель Кубрин Г. С.

<подпись>

Санкт-Петербург

2024

Формулировка задания

Написать программу, использующую три библиотеки: zlib, libpng, freetype, которая должна осуществлять создание png файла с нарисованным в нем текстовым сообщением с помощью шрифта, загруженного из файла *.ttf.

При написании программы должны быть выполнены несколько условий:

- Программа должна компилироваться с помощью gcc и системы makefile в ОС Linux или WSL.
- Программа должна быть выполнена в трех версиях:

static - статическая компоновка всех трех библиотек.

dynamic – динамическая загрузка всех трех библиотек в начале работы с помощью dlload.

blob – реализация всего функционала в виде блоба и загрузка его с диска с помощью загрузчика elf-loader, который должен быть включен в состав проекта программы.

- Система Makefile должна поддерживать следующие команды:

make static – компиляция исходников в версии со статической компоновкой.

make dynamic – компиляция исходников в версии с динамической компоновкой.

make blob – компиляция исходников в версии с компоновкой в виде блоба;

make clean – удаление всех бинарных файлов.

make all – удаление старых бинарных файлов и компиляция исходников в версиях static, dynamic и blob.

- Программа в независимости от типа сборки по результатам своей работы должна не только генерировать png файл с нарисованным текстовым сообщением в нем, но и печатать на экран общее время своей работы и время, затраченное на загрузку системы (от начала main до конца своей работы).

Ход работы

Реализация

Изначально была разработана программа, которая считывает три аргумента и создает png файла с нарисованным в нем текстовым сообщением с помощью шрифта, загруженного из файла *.ttf.

Первым делом вычисляется будущий размер изображения. Для этого при помощи библиотеки freetype, а именно функций GetGlyph и GetKerning, вычисляются размер глифа каждой буквы и их смещение относительно друг друга.

Потом резервируется память под изображение, которое должно получиться и вычисляется значение цвета и прозрачности каждого пикселя в новом изображении.

Далее вызывается функция для записи полученной информации в png. В ней происходит инициализация библиотеки libpng и подготовка к записи изображения в файл. Затем устанавливаются параметры в IHDR и заголовок файла. Далее в цикле с помощью функции png_write_row изображение построчно записывается в файл.

Makefile

Для сборки blob, dynamic и static используется Makefile. Директория имеет следующую структуру (рисунок 1).

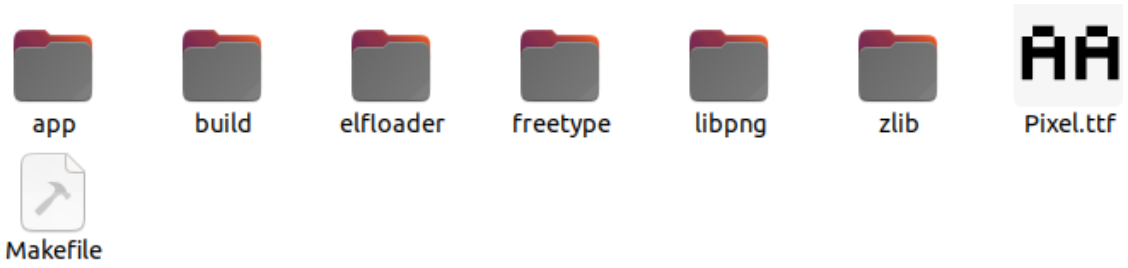

Рисунок 1 – Структура директории

Корневой makefile имеет 4 цели: очистка и три вида сборки. Каждая цель состоит из вызова соответствующих в другой папке, например для

статической сборки вызывается цель static в каждой библиотеке и самом приложении (рисунок 2).

```
all: clean static dynamic blob

clean:
	make -C freetype clean
	make -C zlib clean
	make -C libpng clean
	make -C app clean
	rm -f static
	rm -f dynamic
	rm -f blob
	rm -f *.bin
	rm -f *.so
	rm -f *.png

static:
	@echo "[*] Building static..."
	make -C zlib static
	make -C libpng static
	make -C freetype static
	make -C app static

dynamic:
	@echo "[*] Building dynamic..."
	make -C libpng dynamic
	make -C freetype dynamic
	make -C app dynamic

blob:
	echo "[*] Building blob..."
	make -C zlib static
	make -C libpng static
	make -C freetype static
	make -C app blob

```

Рисунок 2 – Корневой makefile

Makefile для приложения содержит те же самые цели. При очистке удаляются все объектные файлы; при статической сборке создается объектный файл, после чего собирается исполняемый файл со статическими библиотеками; при динамической сборке сразу собирается исполняемый файл с флагами -ldl, -D_DYNAMIC_ и динамически собранными библиотеками; при сборке типа blob сперва собирается исполняемый файл из исходных кодов elfloader, а затем собирается бинарный blob файл, содержащий код приложения и библиотеки (рисунок 3).

```
SOURCES_LOADER=../elfloader/elf_loader.c ../elfloader/main.c ../elfloader/wheelc/list.c

all: clean static dynamic blob

clean:
	rm -f *.o

static:
	gcc -c -I../libpng -I../freetype/include Source.c
	gcc Source.o ../libpng/libpng.a ../zlib/zlib.a ../freetype/freetype.a -lm -o ../static

dynamic:
	gcc -I../freetype/include -I../libpng -D_DYNAMIC_ Source.c -lm -ldl -o ../dynamic

blob:
	gcc $(SOURCES_LOADER) -lm -o ../blob
	gcc Source.c -I../freetype/include -I../libpng ../freetype/freetype.a  ../libpng/libpng.a ../zlib/
zlib.a -nostdlib -pie -fPIE -fPIC -w -e main -D_BLOB_ -o ../blob.bin

```

Рисунок 3 – App makefile

Также были реализованы makefile для каждой из библиотек, рассмотрим их на примере libpng (рисунок 4). В начале перечислены все необходимые объектные файлы libpng, а также объектные файлы zlib, так как libpng зависит от них. При очистке удаляются все созданные объектные файлы, статическая библиотека и динамическая. Статическая сборка заключается в создании объектных файлов (цель .c.o), которые затем объединяются в архив .a. Этот же архив используется для сборки типа blob. Для динамической сборки из объектных файлов libpng и zlib собирается файл типа .so с флагом -shared, который затем перемещается в корневую директорию. Для остальных библиотек makefile имеет аналогичную структуру.

```
OBJECTS=png.o pngerror.o pngget.o pngmem.o pngread.o pngpread.o pngrio.o pngrtran.o pngrutil.o pngset.o
pngtrans.o pngwio.o pngwrite.o pngwtran.o pngwutil.o

OBJECTS_ZLIB=../zlib/adler32.o ../zlib/crc32.o ../zlib/deflate.o ../zlib/infback.o ../zlib/inffast.o ../zlib/
inflate.o ../zlib/inftrees.o ../zlib/trees.o ../zlib/zutil.o ../zlib/compress.o ../zlib/uncompr.o

all: clean static dynamic

clean:
        rm -f *.a
        rm -f *.so
        rm -f *.o

.c.o:
        gcc -pie -fPIE -fno-stack-protector -fno-exceptions -c -fPIC -I../zlib $< -o $@

install: $(OBJECTS)

libpng.a: $(OBJECTS)
        ar rcs libpng.a $(OBJECTS)

libpng.so: $(OBJECTS) $(OBJECTS_ZLIB)
        gcc -shared $(OBJECTS) $(OBJECTS_ZLIB) -o libpng.so
        mv libpng.so ..

static: libpng.a

dynamic: libpng.so

```

Рисунок 4 –Libpng makefile

Для компиляции используется утилита gcc. Флаги компилятора GCC, начинающиеся с одиночного дефиса (-), являются короткими флагами, они обычно представляют собой однобуквенные сокращения для задания определенных параметров компилятора. Флаги, начинающиеся с двух дефисов (--), являются длинными и явно указывают опцию компилятора. В таблице 1 представлены использованные флаги и их описание.

Таблица 1 – Описание использованных флагов

Флаг	**Описание**

-c	Указывает компилятору создать объектные файлы без последующей линковки.
-I	Используется для добавления директорий поиска заголовочных файлов.
-l	Линковка программы с библиотекой. В качестве аргумента указывается название библиотеки без префикса lib и расширения файла.
-l (m)	Опция линковки, указывающая на необходимость подключения библиотеки математических функций при компиляции программы.
-l(dl)	Опция линковки, указывающая на необходимость линковки с библиотекой динамической загрузки (Dynamic Loading Library).
-o	Позволяет задать имя выходного файла.
-D	Используется для определения макросов во время компиляции. Макросы – это именованные константы или куски кода, которые могут быть заменены на другой код во время компиляции.
-D (FT2_BUILD_LIBRARY)	Используется при компиляции библиотеки freetype.
-D (_DYNAMIC)	Используется для указания динамической сборки программы.
-D (_BLOB_)	Используется для указания сборки программы в blob.
-shared	Указывается на создание разделяемой библиотеки.

-nostdlib	Указывает компилятору не использовать стандартные библиотеки. Необходимо для создания исполняемого файла без зависимости от стандартных библиотек.
-e main	Указывает, что точкой входа в программу является main. Аргумент интерпретируется компоновщиком; может принимать либо имя символа, либо адрес.
-f	Передача дополнительных параметров компилятору.
-f (PIC)	Указывает на необходимость генерировать позиционно-независимый код при компиляции. В таком случае внутренняя адресация строится от Global offset table, что позволяет использовать данную библиотеку нескольким процессам одновременно.
-pie	Используется для компиляции в позиционно-независимый исполняемый файл (Position Independent Executable, PIE). PIE — это исполняемый формат, который может выполняться по любому адресу, загруженному в память, не полагаясь на фиксированный базовый адрес.
-f (PIE)	Аналогичен -fPIC, но сгенерированный позиционно-независимый код может быть связан только с исполняемыми файлами. Обычно эти параметры используются для компиляции кода, который будет связан с помощью опции -pie GCC.

Для цели clean в Makefile используется утилита rm. Удаляются все объектные файлы, .so файлы, .a файлы. Чтобы игнорировать ошибки, при отсутствии файла для удаления, используется флаг -f. С помощью данной утилиты можно удалять как конкретные файлы, так и файлы, заданные регулярным выражением, например «*.so».

Результат работы

На рисунках 1–3 представлен результат работы для статической и динамической сборки.

```
blinmda@blinmda-VirtualBox:~/Desktop/lab1$ ./app-static Pixel.ttf static.png "hello"
loading...
load: 0.1 ms
creating...
work: 3.8 ms
```

Рисунок 5 – Выполнение при статической сборке

```
blinmda@blinmda-VirtualBox:~/Desktop/lab1$ ./app-dynamic Pixel.ttf dynamic.png "hello"
loading...
load: 0.4 ms
creating...
work: 4.3 ms
```

Рисунок 6 – Выполнение при динамической сборке

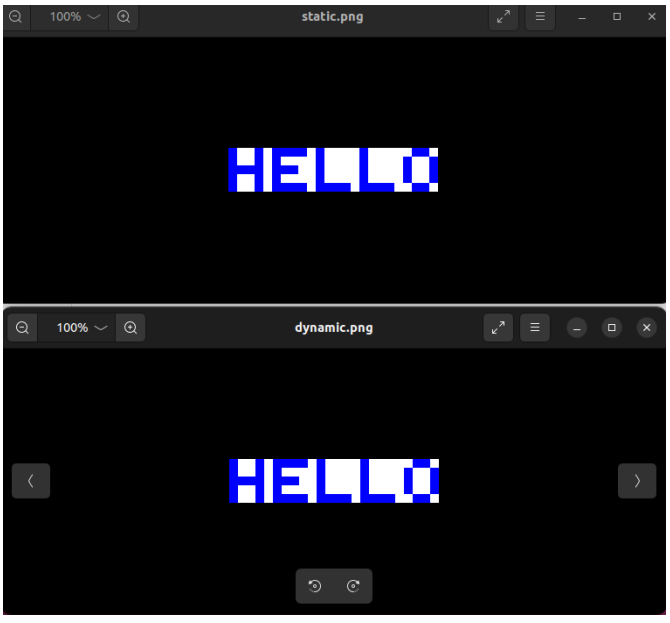

Рисунок 7 – Результат работы

Сравнение статической и динамической компоновки

Статическая компоновка происходит во время компиляции программы. На этапе компиляции все необходимые библиотеки и модули включаются

непосредственно в исполняемый файл программы. Это означает, что вся программа и все ее зависимости становятся частью одного исполняемого файла, что обеспечивает независимость программы от внешних библиотек, но может привести к увеличению размера исполняемого файла.

С другой стороны, динамическая компоновка происходит во время выполнения программы. В этом случае исполняемый файл программы содержит лишь базовую информацию о том, какие библиотеки нужно загрузить. Фактические библиотеки загружаются в память по мере необходимости при запуске программы. Это позволяет экономить память, поскольку одна и та же библиотека может использоваться несколькими программами одновременно, а также облегчает обновление библиотек.

Статические библиотеки следует использовать для создания программ, независимых от внешних библиотек. Это упрощает их распространение, так как все необходимые компоненты уже включены в исполняемый файл. Динамические библиотеки следует использовать в тех случаях, когда нужно сэкономить память, а также иметь возможно обновления библиотек.

В таблице 2 представлено сравнение статической и динамической компоновок.

Таблица 2 – Сравнение статической и динамической компоновки

Критерий	**Статическая**	**Динамическая**
Размер исполняемого файла	Увеличивается при добавлении библиотек	Не меняется
Обновление	Пересобирается весь проект	Пересобирается измененная библиотека
Зависимость	Исполняемый файл независим	Исполняемый файл зависит от внешних библиотек

BLOB

Blob (Binary Large Object) — массив двоичных данных, бинарный файл. В данной работе он включает в себя исходный код программы и все необходимые библиотеки в статической сборке. Файл был собран с опциями -pie и -fPIE, что позволяет создавать исполняемые файлы и код, который могут быть загружены в память по произвольному адресу.

ELF-loader – программа, используемая для запуска исполняемых файлов. Занимается загрузкой файла в память, разрешением всех символов и зависимостей, запуском программы.

Полученная программа работает по следующему алгоритму:

a. Запомнить текущее время;

b. Обработать аргументы командной строки;

c. Сформировать структуру, для передачи ее в блоб;

d. Загрузить блоб с помощью elf-loader;

e. Получить указатель на точку входа в блобе;

f. Вызвать точку входа;

g. Распечатать общее время работы приложения как текущее время – сохраненное время.

На рисунке 4 представлен результат работы программы.

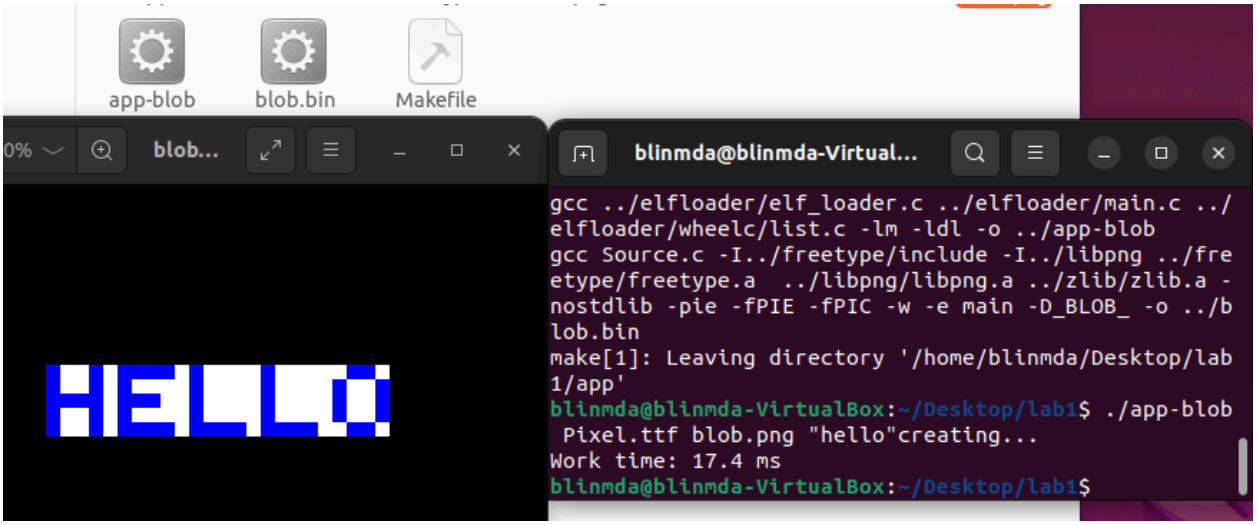

Рисунок 8 – Результат работы

Вывод

В ходе выполнения данной лабораторной работы были получены навыки работы с Makefile, изучены статические и динамические библиотеки, их преимущества и недостатки. Также была реализована полезная нагрузка в виде преобразования текста в png с определенным шрифтом. Помимо этого, был изучен принцип работы blob и его преимущества перед другими библиотеками.