Н.А.ТЫМАНОВИЧ[1], Ю.А.СКУДНЯКОВ[1]

ОСОБЕННОСТИ РАБОТЫ КОМПОНОВЩИКА ПРИ РАЗРАБОТКЕ BARE METAL ПРОЕКТОВ

[1]*Учреждение образования «Белорусский государственный университет информатики и радиоэлектроники», г. Минск, Республика Беларусь*

Создание скрипта компоновщика – неотъемлемый процесс при разработке экосистемы для новых микропроцессоров [1,2]. Без него не будет возможности выполнения исполняемой прошивки, так как для каждого микроконтроллера необходимо, в зависимости от архитектуры, предпочтений производителя и т.п., особая подготовка конечного бинарного файла в соответствующую форму.

Процесс компиляции под bare metal (процессоры без установленной OS), в общем, справедлив и для x86/amd64 процессора, за исключением некоторых особенностей: скомпилированная программа выполняется не на том же устройстве, на котором она компилируется, появляются дополнительные опции для указания модели памяти и таблицы векторов прерываний.

По этой причине, несмотря на то, что компоновщик и является частью компилятора, его стоит отдельно рассматривать в контексте, зависящем от таких факторов, как, например, наличие/отсутствие OS, архитектуры, программной модели и т.п.. В большинстве случаев компилятор может сам получить всю необходимую информацию для подготовки конечного исполняемого объекта, но иногда, все же, приходится вмешиваться в процесс его работы. Хоть принято считать стартовой точкой программы фунцию main(), но это, на самом деле, не совсем корректно. До вызова «главной функции» происходит довольно большое количество операций. Когда запускается сборка проекта, каждый модуль собирается в объектный файл. По большей части он состоит из машинного кода под заданную архитектуру, но он не является «автономным», т.е. вы не сможете его запустить. Компилятор помещает туда дополнительную информацию: ссылки на функции и переменные, определенные вне модуля в виде таблицы.

На рисунке 1 представлена схема этапов, начиная от проектирования программного обеспечения и заканчивая вызовом main функции.

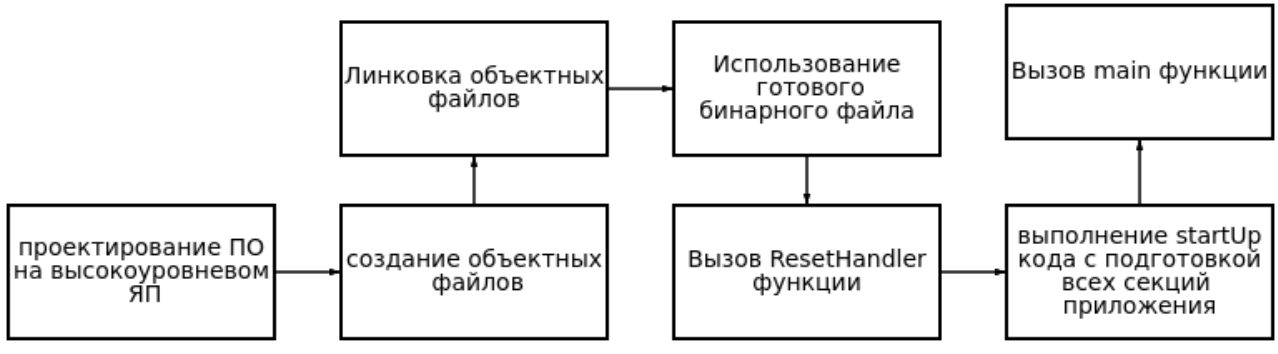

Рисунок 1 – Этапы разработки и подготовки устройства до вызова основной функции программы

Рассмотрим пример с модификатором extern. Допустим, реализуется функция задержки, которая зависит от переменной, хранящей количество миллисекунд, прошедших с момента запуска устройства:

```
// utils.c
volatile uint32_t current_time_ms = 0;

void delay(uint32_t ms)
{
        uint32_t start_time_ms = current_time_ms;
        while( (current_time_ms - start_time_ms) < ms );
}
```

Пока разница текущего времени и времени вызова функции не будет равна или больше переданного в функцию значения ms, цикл while будет сравнивать значения. Переменная current_time_ms, должна увеличиваться каждую миллисекунду на 1.Допустим, это реализуется через прерывание таймера:

```
void SysTick_Handler(void) {
        current_time_ms = current_time_ms + 1;
}
```

Все прерывания, для удобства складываются в один файл, в котором никакой current_time_ms не существует. Если попробовать скомпилировать модуль с обработчиками прерываний, в котором располагается функция обработки таймера, то компилятор выдаст ошибку, в то время как utils.c скомпилируется нормально и даже будет «работать». Просто current_time_ms не будет обновляться, а стало быть, неизменяющееся значение может подвергнуться оптимизации: extern uint32_t current_time_ms.

Данной строчкой сообщается компилятору примерно следующее: у данной переменной тип uint32_t, пока неизвестно место определения переменной, поэтому следует оставить на ее место определенную метку, компоновщик впоследствии подсоединит метки к нужным переменным. После того, как все модули успешно откомпилированы, начинает работу компоновщик, задача которого собрать все файлы в один и решить все внешние зависимости. В случае, если у него этого сделать не получается – возникает ошибка компиляции на этапе линковки.Например, если не будет написан модификатор extern, компоновщик при попытке сшить два модуля обнаружит, что имеется два экземпляра переменной (или это может быть функция) с одинаковым названием – чего быть не должно.

Каждый объектный файл состоит из одной или нескольких секций, в которых хранится либо код, либо данные. Для GCC программный код складывается в секцию .text, проинициализированные глобальные переменные с их значениями складываются в секцию .data, обнуленные - в секцию .bss. Выходной файл компоновщика – это такой же объектный файл, с точно таким же форматом хранения кода и данных. Другими словами, компоновщик группирует код и данные по секциям, пытаясь разрешить внешние связи. Такой файл, однако, не может быть исполнен на целевой платформе. Проблема в том, что в нем нет информации об адресах, где данные и код должны храниться. Следующим в игру вступает локатор. Любая программа, на любом языке, имеет некоторые требования к среде. Для Java это виртуальная машина, для Python интерпретатор, а для C наличие памяти для стека. (грубое упрощение). Место под стек, должно быть выделено до начала выполнения программы и этим занимается startup-файл. Он же выполняет и другие задачи, например производит определенные действия с прерываниями, перемещает нужные данные в оперативную память, и только после этого вызывает функцию main (в прочем функция может называться по-другому). В некоторых компиляторах предусмотрена отдельная утилита, которая занимается локацией адресов, т.е. сопоставлению физических адресов в памяти целевой платформы соответствующим секциям. В GCC она является частью компоновщика. Информация, необходимая для данной процедуры, хранится в специальном файле – в скрипте компоновщика. При работе с интегрированной средой разработки данный файл генерируется автоматически, исходя из указанных параметров микроконтроллера. В некоторых случаях бывает полезно изменить его, чтобы добиться желаемого результата: например поместить программный код в оперативную память для его выполнения.

Таким образом, были рассмотрены основные аспекты проектирования скриптов компоновщика, предоставляющие широкий спектр возможностей по манипуляции с бинарным представлением прошивки.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Создание скриптов компоновщика [Электронный ресурс]. – Режим доступа : https://www.tune-it.ru/web/ myaut/home/-/blogs/25692/. – Дата доступа : 17.09.2022.

2. Руководство новичка по эксплуатации компоновщика [Электронный ресурс].– Режим доступа : https:// habr.com/ru/post/150327/. – Дата доступа : 17.09.2022.