# C++过渡部分

在厌烦了汇编带来的无穷无尽的麻烦时，我们开始希望C++能带给我们便利的曙光

我们需要先分析下C++能带给我们什么（参看参考文献[1]）：

1. 可以使用C++的基本逻辑（比如 if 语句），gcc将负责将其转换为汇编代码，前途很光明

2. 诸多限制：

2.1 没有 built-in

2.2 没有 RunTime Type Information

2.3 没有 异常处理

2.4 没有 C/C++标准库

3. 暂时不能使用 全局对象/静态对象 （之后我们将着手解决这个问题）

尽管限制诸多，但是能摆脱汇编已经是很诱人了，为此我们将费点力气制定个C++路线图

1. 使用最简单的手法，让C++程序运作起来

2. 解决 全局对象/静态对象的限制

## Hello C++

让C++尽快工作起来很简单：

1. 需要一个C++程序，建立最简单的main.c，如下：

int main0()

{

while (1);

return 0;

}

1. 编译此程序，编译命令：

gcc -c -ffreestanding -nostdlib -fno-builtin -fno-rtti -fno-exceptions -o main.o main.cpp

1. 建立启动程序

为了以后方便，我们建立一个启动程序 sector2.s，内容如下：

[BITS 32]

[extern \_main0]

global greatstart

greatstart:

jmp \_main0

在这里sector2.s的代码暂时没有意义，只是将main0再包装为greatstart这个名字

另外需要提到的是，\_main0这个汇编名字对应于C++中的main0函数，后面还会出现类似情况

1. 编译启动程序
2. 链接脚本

链接时需要注意：之前在bootLoader中我们将代码转向了一个诸如0xc0100000的地址，因此我们需要将greatstart放到这一地址，系统才能正常启动。为达到这一目的，我们使用链接脚本Link.ld，内容如下：

这个脚本有两个作用：

OUTPUT\_FORMAT("binary")

ENTRY(start)

SECTIONS

{

. = 0xc0100000;

.text :

{

start.o

\*(EXCLUDE\_FILE(start.o).text)

}

.data : { \*(.data) }

.bss : { \*(.bss) }

}

1) 将程序的起始对齐了0xc0100000

2) 将start.o的正文段置于所有其他.o文件的正文段之前，保证了start.o的起始位置是0xc0100000

1. 使用链接脚本链接程序

$(LD) -T link.ld -o kernel $(OBJS)

1. 大功告成，关于这段代码如何运行起来的，参看理论部分

## 解决 全局对象/静态对象 的限制

毋庸置疑，全局对象和静态对象的使用大大方便了编程

但是在当前，我们并不能正常使用全局对象和静态对象，原因是全局/静态对象需要在程序启动前构造、在程序结束后析构，而我们现在进入0xc0100000后，直接转入main0运行内核程序，缺少构造/析构的过程

下面我们来添加构造/析构的过程，其实前面我们已经留下了sector2.s这个"后门"：

* sector2.s 如下修改：

[BITS 32]

[extern \_main0]

[extern \_\_main] ;"\_main()"定义在support.c中

[extern \_\_atexit] ; "\_atexit()"定义在support.c中

global greatstart

greatstart:

call \_\_main ;call our own startup code

jmp \_main0

call \_\_atexit ;call our own startup code

我们在这里加入了两个钩子，\_\_main和\_\_atexit（对应于C++函数\_main和\_atexit）

* 实现\_main 构造函数 和\_atexit析构函数

以上代码并不难读懂，大致意思就是依次调用 构造函数链\_\_CTOR\_LIST\_\_ 和 析构函数链 \_\_DTOR\_LIST\_\_来完成构造/析构的过程

需要说明的是\_\_CTOR\_LIST\_\_ 、 \_\_DTOR\_LIST\_\_这两个函数链名字是由gcc在编译时提供

* 自此，我们可以使用全局/静态对象

## 理论部分

### 使用C++的限制

前面提到了，使用C++的限制如下：

1 没有 built-in

2 没有 RunTime Type Information

3 没有 异常处理

4 没有 C/C++标准库

5 暂时不能使用 全局对象/静态对象

除了最后一条，其余各项限制都是跟操作系统有关，我们的操作系统没有实现相应的底层库，所以不能使用诸如 C++异常处理 等的高级特性

至于第五条限制，是因为没有调用 构造/析构函数 进行全局/静态对象的构造/析构，导致限制。在前面我们已经通过插入构造/析构函数(\_main / \_atexit) 解决了这个问题，而使用的手段是链接脚本

### 链接脚本

链接脚本规定了链接的规则，主要规定了 各编译结果的正文段、数据段、BSS段 在链接结果中的位置 和链接方法，在我们代码中，链接脚本用于解决两个问题：

1) 将程序的起始对齐了0xc0100000，可以直接由bootloader转入该地址运行程序

2) 将start.o的正文段置于所有其他.o文件的正文段之前，保证了start.o的起始位置是0xc0100000，让bootloader直接转入start.o程序运行

简略说明一下代码所使用的链接脚本

OUTPUT\_FORMAT("binary") ; 输出格式 : 二进制格式

ENTRY(start) ; 入口函数为 start

SECTIONS

{

. = 0xc0100000 ; 地址对齐到0xc0100000

.text :

{

start.o ; start.o链接到此

\*(EXCLUDE\_FILE(start.o).text) ; 非start.o的.o文件的正文段链接到此

}

.data : { \*(.data) } ;数据段链接到此

.bss : { \*(.bss) } ; BSS段链接到此

}

链接后的程序的运行关系如下图 :

