Riscv学习中的一些坑

2024年12月21日 | 1089字 | 5分钟

目录

1. 汇编时不包含标准库的情况下，全局指针`gp`不会被初始化，导致全局变量寻址错误。

tl;dr: 在编译指令中添加`-Wl,--no-relax`选项禁用*链接器松弛*?即可。

~~这就是我们连接器的松弛感啊，你们有没有这样的松弛感呢？~~

刚开始跟着`CS61C`学到riscv时，我发现gcc编译出来的二进制文件由于包含了标准库导致文件很大，便选择了在汇编时不使用标准库：

```
riscv32-unknown-elf-gcc -g -nostdlib -static "$(VIM_FILEPATH)" -o "$(VIM_FILEDI
```

没想到就这个举动导致我花了整整一个下午来排查。>:(

当时大致代码如下，编译后发现能输出`foobar`，但不能输出`bazboo`，非常诡异：

```asm
word1:
    .asciz "foobar"

word2:
    .asciz "bazboo"
```

```
.text

.global _start
_start:
    la  a0, word1
    li a1, 7
    jal ra, print

    la  a0, word2
    li  a1, 7
    jal ra, print

    li    a0, 0
    li    a7, 93 # Exit syscall number
    ecall # Exit

print:
    mv a2, a1
    mv a1, a0
    li a0, 1
    li a7, 64
    ecall
    ret
```

`strace`以后发现后者`write`系统调用失败，再看发现`la a0, word2`获取到的地址是一个很大的值：

```
402674 write(1,0x110c6,7)foobar = 7
402674 write(1,0xfffff807,7) = -1 errno=14 (Bad address)
402674 exit(0)
```

这个`0xfffff807`是什么鬼？原来是负的2042。再查，反汇编后发现：

```
❯ riscv32-unknown-elf-objdump -d lab03/tmp

lab03/tmp:     file format elf32-littleriscv

Disassembly of section .text:
```

```
00010094 <_start>:
   10094:   00001517           auipc   a0,0x1
   10098:   03250513           addi    a0,a0,50 # 110c6 <__DATA_BEGIN__>
   1009c:   459d                   li  a1,7
   1009e:   018000ef           jal 100b6 <print>
   100a2:   80718513           addi    a0,gp,-2041 # 110cd <word2>
   100a6:   459d                   li  a1,7
   100a8:   00e000ef           jal 100b6 <print>
   100ac:   4501                   li  a0,0
   100ae:   05d00893           li  a7,93
   100b2:   00000073           ecall

000100b6 <print>:
   100b6:   862e                   mv  a2,a1
   100b8:   85aa                   mv  a1,a0
   100ba:   4505                   li  a0,1
   100bc:   04000893           li  a7,64
   100c0:   00000073           ecall
   100c4:   8082                   ret
```

这两个`la`汇编出来的指令居然不一样！一个是根据`.data`段的偏移寻址，一个是用`gp`寄存器寻址。虽然注释中的地址写得很美，但实际上肯定是这个奇怪的`gp`导致的。调试发现，`gp`的值是0，明显没有被正常初始化。

> RISC-V gp全局指针寄存器说明

gp寄存器在启动代码中加载为__global_pointer$的地址，并且之后不能被改变。linker时使用__global_pointer$来比较全局变量的地址，如果在范围内，就替换掉lui或puipc指令的 absolute/pc-relative寻址，变为gp-relative寻址，使得代码效率更高。该过程被称为linker relaxation(链接器松弛)，也可以使用-Wl,—no-relax来关闭此功能。

普通调用方式为：

```
lui a5,0x20000    /* 将0x20000100高20位0x20000 左移12位赋给a5寄存器 */

lw  a5,256(a5)    /* 加载a5+256（0x100，0x20000100低12位）的值至a5寄存器 */
```

gp指针优化调用方式：

```
lw a5,-1792(gp) /* 加载gp-1792地址处的值至a5，即0x20000100处的值*/
```

通过gp指针，访问其值±2KB，即4KB范围内的全局变量，可以节约一条指令。

的确，-2041刚好和-2KB相近。那么启动代码又在哪里？为什么别人就不用手动设置`gp`？

找到两位难兄难弟：

> >RISC-V assembly: global pointer set to a weird value
>
> SOLUTION: the issue was due to the global pointer gp being not initialized

然后又发现其实c库会帮我们做这个事情：

> >RISC-V 中的 global pointer 寄存器
>
> 虽然看上去是要我们写程序（program）时自己去初始化这个 gp 寄存器，但实际上这个动作都是由 c 库帮助我们去完成的，ISA 规范中也提了最好是在 _start 中做，就是比较晦涩没有明说是 c 库

那么我们要么禁用链接器松弛，要么带上c库一起走。经过群友提示，还可以用`lla`替代`la`来禁止这个行为：

> *"la 地址加载 (Load Address). 伪指令(Pseudoinstruction), RV32I and RV64I. 将 symbol 的地址加载到 x[rd]中。当编译位置无关的代码时，它会被扩展为对全局偏移量表(Global Offset Table)的加载。对于 RV32I，等同于执行 auipc rd, offsetHi，然后是 lw rd， offsetLo(rd);对于 RV64I，则等同于 auipc rd，offsetHi 和 ld rd, offsetLo(rd)。**如果 offset 过大，开始的算加载地址的指令会变成两条，先是 auipc rd, offsetHi 然后是 addi rd, rd, offsetLo。***
> *(http://staff.ustc.edu.cn/~llxx/cod/reference_books/RISC-V-Reader-Chinese-v2p12017.pdf)"*

结案。~~~~ 那么为什么`word1`就没有这个优化？是上面的`offset`过大`0x1`刚好在`0x800`-2KB的边界？暂时作为一个未解之谜。