吴章金：通过操作 **Section** 为 **Linux ELF** 程序新增数据

吴章金

《360度剖析Linux ELF系列》相关文章：

吴章金：如何创建一个*可执行*的共享库

吴章金：深度剖析 Linux共享库的“位置无关”实现原理

背景介绍

Section 是 Linux ELF 程序格式的一种核心数据表达方式，用来存放一个一个的代码块、数据块（包括控制信息块），这样一种模块化的设计为程序开发提供了很大的灵活性。

需要增加一个功能，增加一份代码或者增加一份数据都可以通过新增一个 Section 来实现。Section 的操作在 Linux 内核中有着非常广泛的应用，比如内核压缩，比如把 .config 打包后加到内核映像中。

下面介绍三种新增 Section 的方式：汇编、C 和 ELF 工具。

通过内联汇编新增一个 **Section**

如何创建一个可执行的共享库 中有一个很好的例子：

```
asm(".pushsection .interp,\"a\"\n"
    "        .string \"/lib/i386-linux-gnu/ld-linux.so.2\"\n"
    ".popsection");
```

通过上述代码新增了一个 `.interp` Section，用于指定动态链接器。简单介绍一下这段内联汇编：

- `asm` 括号内就是汇编代码，这些代码几乎会被“原封不动”地放到汇编语言中间文件中（hello.s）。
- 这里采用 `.pushsection, .popsection`，而不是 `.section` 是为了避免之后的代码或者数据被错误地加到这里新增的 Section 中来。
- `.pushsection .interp, "a"`，这里的 "a" 表示 Alloc，会占用内存，这种才会被加到程序头表中，因为程序头表会用于创建进程映像。
- `.string` 这行用来指定动态链接器的完整路径。

稍微延伸两点：

- `.string` 可以替换为 `.incbin file`，然后把字符串内容放到名为 file 的文件中。文件末尾记得改为 `\0` 字节，可以用二进制编辑工具修改。
- `.string` 还可以替换为 `.ascii`，不过呢，末尾得主动加个 `\0` 字节，用法如下：

```
".ascii \"/lib/i386-linux-gnu/ld-linux.so.2\\x00\"\n"
```

`.incbin` 方式在 Linux 内核中用处相当广泛，例如：

- `arch/arm/boot/bootp/kernel.S: .incbin "arch/arm/boot/zImage"`
- `kernel/configs.c: .incbin \"kernel/config_data.gz\"`

本节完整演示代码如下：

```
#include <stdio.h>
#include <unistd.h>

#if 1
asm(".pushsection .interp,\"a\"\n"
    "        .ascii \"/lib/i386-linux-gnu/ld-linux.so.2\\x00\"\n"
    ".popsection");

    /* .ascii above equals to .string \"/lib/i386-linux-gnu/ld-linux.so.2\"\n */
#else
asm(".pushsection .interp,\"a\"\n"
    "        .incbin \"interp.section.txt\"\n"
    ".popsection");
#endif

int main(void)
{
        printf("hello\n");

        return 0;
}

void _start(void)
{
        int ret;

        ret = main();
        _exit(ret);
}
```

编译和执行：

```
$ gcc -m32 -shared -fpic -o libhello.so hello.c
$ ./libhello.so
hello
```

通过 **gcc `attribute`** 新增一个 **Section**

上面的需求可以等价地用 gcc `__attribute__` 编译属性来指定：

```
const char interp[] __attribute__((section(".interp"))) = "/lib/i386-linux-gnu/ld-linux.so.2";
```

本节完整演示代码如下：

```
#include <stdio.h>
#include <unistd.h>

const char interp[] __attribute__((section(".interp"))) = "/lib/i386-linux-gnu/ld-linux.so.2";

int main(void)
{
        printf("hello\n");

        return 0;
}

void _start(void)
{
        int ret;

        ret = main();
        _exit(ret);
```

```
}
```

编译和执行方法同上，不做重复介绍。

通过 **objcopy** 把某文件内容新增为一个 **Section**

上面介绍了 C 和汇编层面的方法，再来介绍一个工具层面的方法。

objcopy 这个工具很强大，其中就包括新增 Section。

接下来先准备一个文件 interp.section.text，记得末尾加的 `\0` 字节：

```
$ echo -e -n "/lib/i386-linux-gnu/ld-linux.so.2\x00" > interp.section.txt
```

接着准备一个 hello.c，里头不指定任何 `.interp`：

```
#include <stdio.h>
#include <unistd.h>

int main(void)
{
        printf("hello\n");

        return 0;
}

void _start(void)
{
        int ret;

        ret = main();
        _exit(ret);
}
```

不过需要注意的是，objcopy 不能直接在最终的可执行文件和共享库中加入一个 Section：

```
$ objcopy --add-section .interp=interp.section.txt --set-section-flags .interp=alloc,readonly libhello.so
objcopy:stTyWnxc: can't add section '.interp': File in wrong format
```

怎么办呢，需要先加入到 `.o` 中，再链接，类似这样：

```
$ gcc -m32 -shared -fpic -c -o hello.o hello.c
$ objcopy --add-section .interp=interp.section.txt --set-section-flags .interp=alloc,readonly hello.o
$ gcc -m32 -shared -fpic -o libhello.so hello.o
```

注意，必须加上 `--set-section-flags` 配置为 alloc，否则，程序头会不纳入该 Section，结果将是缺少 INTERP 程序头而无法执行。

需要补充的是，本文介绍的 `.interp` 是一个比较特殊的 Section，链接时能自动处理，如果是新增了一个全新的 Section 类型，那么得修改链接脚本，明确告知链接器需要把 Section 放到程序头的哪个 Segment。

通过 **objcopy** 更新某个 **Section**

以上三种新增 Section 的方式适合不同的需求：汇编语言、C 语言、链接阶段，基本能满足日常的开发需要。

再补充一种方式，举个例子，上面用到的动态链接器来自 `libc6:i386` 这个包：

```
$ dpkg -S /lib/i386-linux-gnu/ld-linux.so.2
libc6:i386: /lib/i386-linux-gnu/ld-linux.so.2
```

如果系统安装的是 `libc6-i386` 呢?

```
$ dpkg -S /lib32/ld-linux.so.2
libc6-i386: /lib32/ld-linux.so.2
```

两个包提供的动态链接器路径完全不一样，那就得替换掉动态编译器，要重新编译 C 或者汇编吗?

其实不需要重新编译，因为可以直接这样换掉:

```
$ objcopy --dump-section .interp=interp.txt libhello.so
$ sed -i -e "s%/lib/i386-linux-gnu/ld-linux.so.2%/lib32/ld-linux.so.2%g" interp.txt
$ objcopy --update-section .interp=interp.txt libhello.so
$ ./libhello.so
hello
```

上面几组指令先把 `.interp` Section 取出来存到 interp.txt 文件中，再替换掉其中的动态链接器路径，最后再把新文件的内容更新进共享库。

小结

以上主要介绍了 Linux ELF 核心数据表达方式 Section 的多种 add 和 update 用法，掌握这些用户可以利于理解 Linux 内核源码中类似的代码，也可以用于实际开发和调试过程去解决类似的需求。

如果想与本文作者进一步探讨 Linux ELF 中各种类型 Section 的作用、更多操作方式（查看、裁剪、修改等）以及链接脚本的用法，欢迎订阅吴老师已经全面上线的 10 小时视频课程，点击:

报名：《360° 剖析 Linux ELF》在线视频课程已经全面上线 。

即刻注册并订阅课程 还可以享受平台的双十一特大优惠（注册有礼+课程专属优惠券+阅码场专属折扣码YM9C），手速快才有哦！

Linux阅码场原创精华文章汇总

更多精彩，尽在**"Linux**阅码场**"**，扫描下方二维码关注

你的随手转发或点个在看是对我们最大的支持！