HELM 命令 EOF 报错分析

Email: geekidea@gmail.com

2019/03

0. 问题描述

最近运维开发同学反馈使用 goroutine 方式调用 helm 命令进行业务容器部署时，偶尔失败，终端输出的错误信息为：**"Error: EOF"**（对应 go 的错误码为 **io.EOF**），进一步沟通了解到，为了提升部署效率，go 程序使用了多个 goroutine（**GOMAXPROCS > 1**）并发执行（**helm install *****）业务容器安装部署。

1. 问题分析

基于 UNIX 系统编程基础，对于文件类型，当执行读操作（如：syscall read），返回 0 字节时，表示已经读到了文件末尾（**end of file ， EOF**）；对于 TCP socket 类型，当执行读操作（如：syscall read, recv, readv 等），返回 0 字节时，表示 TCP 对端已经关闭了该连接。

根据本文提及的 io.EOF 错误，可以推断或许由上述两个原因引起，调试该 go 代码，查看该报错产生的相关堆栈信息，快速定位到 helm 的相关代码：

```go
// LoadFile loads from an archive file.
func LoadFile(name string) (*chart.Chart, error) {
    if fi, err := os.Stat(name); err != nil {
        return nil, err
    } else if fi.IsDir() {
        return nil, errors.New("cannot load a directory")
    }

    raw, err := os.Open(name)
    if err != nil {
        return nil, err
    }
    defer raw.Close()

    return LoadArchive(raw)
}
```

```go
func LoadArchive(in io.Reader) (*chart.Chart, error) {
    files, err := loadArchiveFiles(in)
    if err != nil {
        return nil, err
    }
    return LoadFiles(files)
}
```

当 LoadFile（间接）调用 **LoadArchiveFiles** 时返回了 io.EOF 错误：

```go
// loadArchiveFiles loads files out of an archive
func loadArchiveFiles(in io.Reader) ([]*BufferedFile, error) {
    unzipped, err := gzip.NewReader(in) //此处返回io.EOF错误
    if err != nil {
        return nil, err
    }
    defer unzipped.Close()

    files := []*BufferedFile{}
    tr := tar.NewReader(unzipped)
    for {
        b := bytes.NewBuffer(nil)
        hd, err := tr.Next()
        if err == io.EOF {
            break
        }
        if err != nil {
            return nil, err
        }
```

为观察该程序（myhelm）完整的执行过程，使用系统命令 **strace** 执行该程序：

```
strace -f -s 1024 -ttT -o myhelm.st ./myhelm
```

上述命令将该程序所执行的系统调用保存至 myhelm.st 日志文件中，快速查找下执行 helm install 的相关线程，执行：`grep execve myhelm.st  | grep  install`

```
1379230 15:28:33.824300 execve("/home/platform/go/bin/go", ["go", "run", "helm.go", "install"], [/* 33 vars */]) = 0 <0.000143>
1379404 15:28:34.379670 execve("/tmp/go-build380955838/b001/exe/helm", ["/tmp/go-build380955838/b001/exe/helm", "install"], [/* 33 vars */] <unfinish
ed ...>
1379415 15:28:34.394676 execve("/home/platform/project/src/k8s.io/helm/bin/helm", ["/home/platform/project/src/k8s.io/helm/bin/helm", "install", "cha
rtmuseum/nginx", "--version=0.1.2", "--name=nginx-test-0"], [/* 33 vars */] <unfinished ...>
1379419 15:28:34.397504 execve("/home/platform/project/src/k8s.io/helm/bin/helm", ["/home/platform/project/src/k8s.io/helm/bin/helm", "install", "cha
rtmuseum/nginx", "--version=0.1.2", "--name=nginx-test-1"], [/* 33 vars */] <unfinished ...>
1379420 15:28:34.400548 execve("/home/platform/project/src/k8s.io/helm/bin/helm", ["/home/platform/project/src/k8s.io/helm/bin/helm", "install", "cha
rtmuseum/nginx", "--version=0.1.2", "--name=nginx-test-3"], [/* 33 vars */] <unfinished ...>
1379421 15:28:34.403992 execve("/home/platform/project/src/k8s.io/helm/bin/helm", ["/home/platform/project/src/k8s.io/helm/bin/helm", "install", "cha
rtmuseum/nginx", "--version=0.1.2", "--name=nginx-test-6"], [/* 33 vars */] <unfinished ...>
1379422 15:28:34.408580 execve("/home/platform/project/src/k8s.io/helm/bin/helm", ["/home/platform/project/src/k8s.io/helm/bin/helm", "install", "cha
rtmuseum/nginx", "--version=0.1.2", "--name=nginx-test-7"], [/* 33 vars */] <unfinished ...>
1379423 15:28:34.417000 execve("/home/platform/project/src/k8s.io/helm/bin/helm", ["/home/platform/project/src/k8s.io/helm/bin/helm", "install", "cha
rtmuseum/nginx", "--version=0.1.2", "--name=nginx-test-4"], [/* 33 vars */] <unfinished ...>
1379438 15:28:34.427332 execve("/home/platform/project/src/k8s.io/helm/bin/helm", ["/home/platform/project/src/k8s.io/helm/bin/helm", "install", "cha
rtmuseum/nginx", "--version=0.1.2", "--name=nginx-test-13"], [/* 33 vars */] <unfinished ...>
1379445 15:28:34.435985 execve("/home/platform/project/src/k8s.io/helm/bin/helm", ["/home/platform/project/src/k8s.io/helm/bin/helm", "install", "cha
rtmuseum/nginx", "--version=0.1.2", "--name=nginx-test-16"], [/* 33 vars */] <unfinished ...>
1379455 15:28:34.454287 execve("/home/platform/project/src/k8s.io/helm/bin/helm", ["/home/platform/project/src/k8s.io/helm/bin/helm", "install", "cha
rtmuseum/nginx", "--version=0.1.2", "--name=nginx-test-10"], [/* 33 vars */] <unfinished ...>
1379467 15:28:34.478119 execve("/home/platform/project/src/k8s.io/helm/bin/helm", ["/home/platform/project/src/k8s.io/helm/bin/helm", "install", "cha
rtmuseum/nginx", "--version=0.1.2", "--name=nginx-test-2"], [/* 33 vars */] <unfinished ...>
1379483 15:28:34.508405 execve("/home/platform/project/src/k8s.io/helm/bin/helm", ["/home/platform/project/src/k8s.io/helm/bin/helm", "install", "cha
rtmuseum/nginx", "--version=0.1.2", "--name=nginx-test-17"], [/* 33 vars */] <unfinished ...>
1379505 15:28:34.542560 execve("/home/platform/project/src/k8s.io/helm/bin/helm", ["/home/platform/project/src/k8s.io/helm/bin/helm", "install", "cha
rtmuseum/nginx", "--version=0.1.2", "--name=nginx-test-5"], [/* 33 vars */] <unfinished ...>
1379531 15:28:34.585637 execve("/home/platform/project/src/k8s.io/helm/bin/helm", ["/home/platform/project/src/k8s.io/helm/bin/helm", "install", "cha
rtmuseum/nginx", "--version=0.1.2", "--name=nginx-test-11"], [/* 33 vars */] <unfinished ...>
1379556 15:28:34.634293 execve("/home/platform/project/src/k8s.io/helm/bin/helm", ["/home/platform/project/src/k8s.io/helm/bin/helm", "install", "cha
```

根据截图可知，该程序通过多线程调用 **execve** 方式，并发执行了如下操作：

```
helm install chartmuseum/nginx --version=0.1.2 --name=nginx-test-**
```

进一步分析该 strace 日志文件，共有 20 个相应的 helm install 操作，可推断该程序一共开启了 20 个相应的 goroutine。

分析 strace 日志文件，过滤错误信息" Error: EOF "确定出错的线程 ID：`grep 'Error: EOF' myhelm.st`

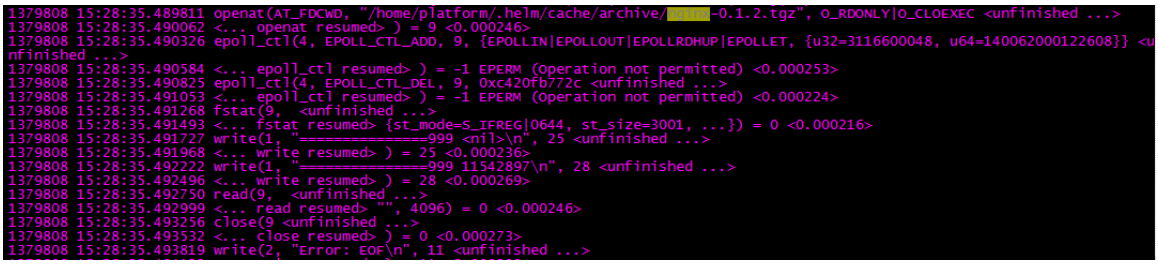
```
1379808 15:28:35.493819 write(2, "Error: EOF\n", 11 <unfinished ...>
1379417 15:28:35.496528 <... read resumed> "Error: EOF\n", 512) = 11 <0.000514>
1379773 15:28:35.504194 write(1, "17 -- exit status 1 -- ==============999 <nil>\n==============999 11542897\n -- Error: EOF\n\n", 92 <u
nfinished ...>
```

如图，有三个不同的线程 ID（1379808，1379417，1379773）与该错误相关，分别观察各自线程 ID 的执行过程，如：`grep '^1379808 ' myhelm.st > t1379808.st`

打开该线程 1379808 的日志文件（t1379808.st）查看该错误关联的上下文：

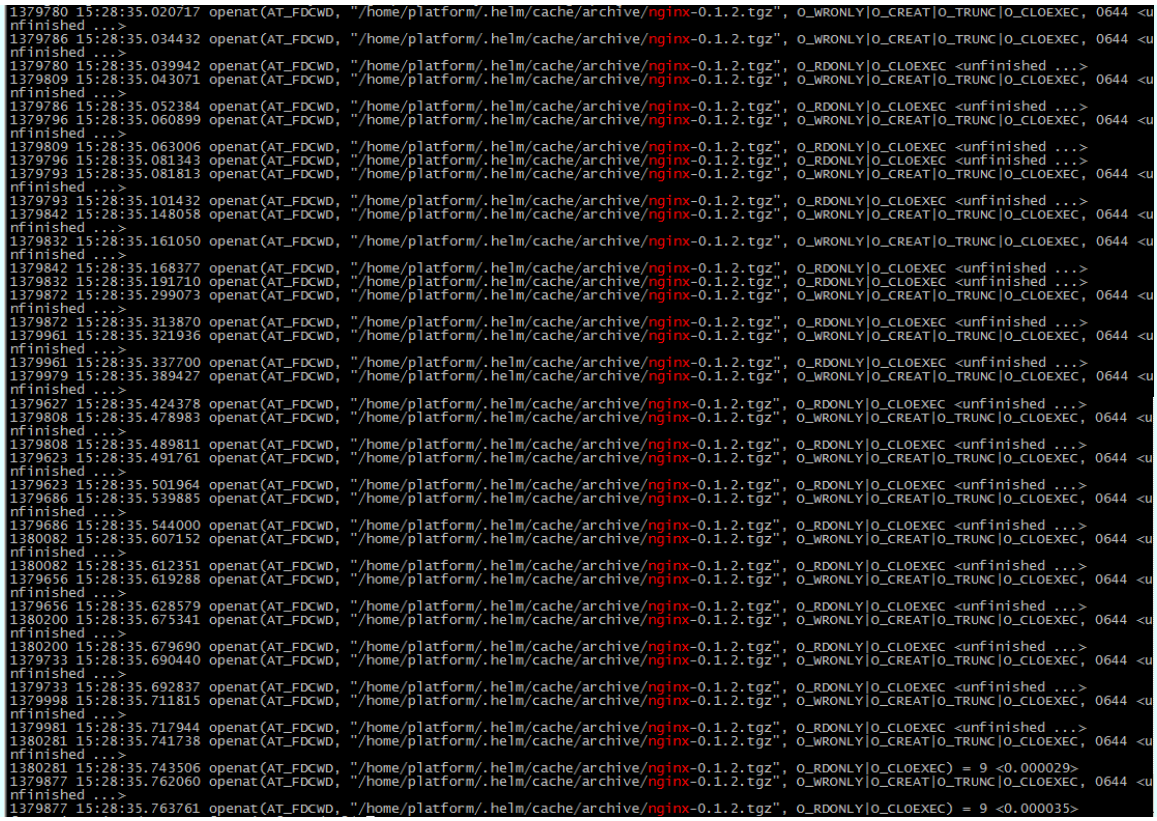
如上截图可知，该线调用 openat syscall 打开文件 nginx-0.1.2.tgz，在写入了 3001 字节后，关闭该文件。值得注意的是 openat 使用了 **O_TRUNC** 方式，含义为若文件存在则将文件截短为 0 字节。

继续查看该日志文件（t1379808.st，略过加载本地 kube config 等相关操作），可以看到如下执行过程：



如上截图可知，该线程以只读方式（**O_RDONLY**）打开文件 nginx-0.1.2.tgz，然后调用 fstat syscall 方式查看该文件大小为 3001 字节，但是当调用 read syscall 读该文件时，返回了 0 字节，进而报错 **Error: EOF**。

由于文件在第一次写入时已经成功写入了 3001 字节，但是考虑到有其他线程（goroutine）在同时执行 helm install，因此推断该文件被其他 helm install 的线程通过 openat **O_TRUNC** 方式截短了，查看打开该文件 nginx-0.1.2.tgz 的所有线程（包含上面分析过的线程 1379808）： `grep openat myhelm.st | grep nginx`



如上截图，共 40 个 nginx-0.1.2.tgz 文件的打开操作，分别对应上文提到的 20 个 goroutine 中的写操作与后续的读操作。当线程 1379808 读 nginx-0.1.2.tgz 文件时，被紧随其后的线程 **1379623** 以 openat **O_TRUNC** 方式清空了，进而导致了线程 1379808 读文件时 EOF 报错。

2. 解决方案

综上该错误主要是由于并行执行 helm 部署时，多线程并发读写同一文件冲突所致，因此可以采取如下几种解决方案：

1. 使用单一的 go 逻辑 CPU：环境变量 **GOMAXPROCS=1**

2. 为各自的 goroutine，使用不同的 helm repo home 目录，并在 helm install 时指定之。

3. 修改 helm 源码，通过某环境变量指定不同的文件路径，并在 helm install 时指定之。

4. 修改 helm 源码，对相同文件打开读写时进行加锁，以保证原子性。

方案 1 通过指定环境变量 **GOMAXPROCS=1**，尽量保证了多个 goroutine 对同一文件基本的读写操作按照同步方式顺序执行，但是，当文件并不能通过一次 syscall write/read 全部读写完时，该 goroutine（g-1）会被挂起，然后其他的文件读写 goroutine（g-2）会被调度，当 g-1 再次被调度执行时可能会读到 g-2 所覆盖的内容，因此也会导致读写冲突。

方案 2，需要初始化指定不同的 helm repo，在实际使用过程中，仅保留少数几个 helm repo，该方案与实际场景不符。

方案 3，新增一个环境变量标识文件读写的路径，当文件写入以及读取时，首先判断加载该环境变量，然后拼接出该文件的完整路径，供后续读写，多个 helm install goroutine 可以指定不同的环境变量，由于该方案可能会影响 helm 其他的相关操作（如：helm fetch/get/inspect 等），需要实际权衡该方案所带来的影响。

方案 4，为确保多线程（多进程）正确执行 helm install 需要在文件以写入方式打开前，对该文件加锁，例如：使用 syscall.Flock 对该文件关联的锁文件加锁，然后维持原方式依次执行：文件打开，写入操作，关闭文件，再次以只读方式打开，读取数据，关闭文件，最后释放该文件锁。

由于方案 4 的代码改动复杂度较高，最终使用方案 3，具体改动如下：

```
// Archive returns the path to download chart archives.
func (h Home) Archive() string { //此处优先使用os.Getenv("SOME_VAR")环境变量值
    return h.Path("cache", "archive")
}
```

3. 后续思考

上述方案 3 在执行 helm install 时，通过读取预置环境变量的方式，将本地 helm 的缓存目录指向不同的路径，这样会导致下载至本地的文件在不同的目录下出现多份拷贝情况，需要引起注意。

查看官方问题列表（#3253），如果直接使用 helm client API 方式，应该可以支持并发访问，需要对该方式其进行测试确认。