FUSE-FTP——基于 FUSE 的 FTP 文件系统

计 54 郑远航 2015011339

计 54 秦一鉴 2015011327

计 54 乔一凡 2015011398

计 55 陈齐斌 2015011403

一、实验背景

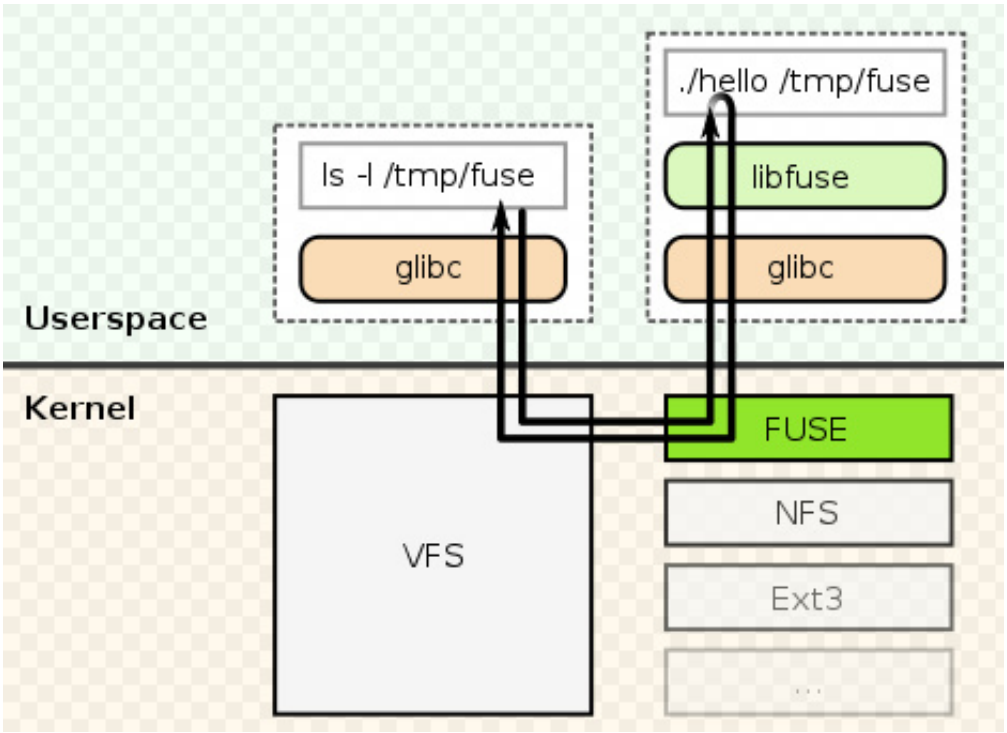
我们大作业的项目是利用 FUSE 实现一个用户态 FTP 文件系统。

FUSE

FUSE (Filesystem in USErspace) 是一套暴露给操作系统内核的用户态文件系统接口。实现一个具体的文件系统需要两部分的支持：

- 一个 FUSE 内核模块（在 Linux 内核中已经实现），会在内核模块被加载后被 VFS 调用。FUSE 被加载后看起来像一个普通的文件系统模块，会获取文件系统的文件操作；当 VFS 发来文件操作请求之后，它将该请求转化为特定格式，并通过设备传递给用户空间进程
- 以及一个用户空间的 FUSE 库（如 Linux 的 libfuse，macOS 的 FUSE for macOS 和 Windows 的 Dokan），用于提供相关函数与 FUSE 内核模块进行通信，完成文件操作并将结果返回给 FUSE 内核模块，内核模块再将其还原为 Linux kernel 需要的格式，并返回给 VFS，完成操作。

FUSE 的具体工作流程如下图所示（图片来源：WikiPedia）



FTP

FTP 是File Transfer Protocol（文件传输协议）的英文简称，用于Internet上的控制文件的双向传输。TCP/IP协议中，FTP标准命令TCP端口号为21，Port方式数据端口为20。需要进行远程文件传输的计算机必须安装和运行ftp客户程序。

与大多数Internet服务一样，FTP也是一个客户机/服务器系统。用户通过一个支持FTP协议的客户机程序，连接到在远程主机上的FTP服务器程序。用户通过客户机程序向服务器程序发出命令，服务器程序执行用户所发出的命令，并将执行的结果返回到客户机。比如说，用户发出一条命令，要求服务器向用户传送某一个文件的一份拷贝，服务器会响应这条命令，将指定文件送至用户的机器上。客户机程序代表用户接收到这个文件，将其存放在用户目录中。FTP的命令被定义在RFC 959。

二、实现概述

我们的实现主要分为两部分：

- 一部分是根据 FTP 协议实现 FTP 基本操作的接口，完成从 FTP 服务器读取，上传，访问目录，查看当前目录等基本 FTP 操作，并将这些接口提供给 FUSE 上层使用；
- 另一部分是利用已经实现好的 FTP 操作接口实现 FUSE 侧具体的文件系统操作相关函数，并通过 FUSE 完成对用户请求的响应。

我们在本地建立了一个缓存目录，将 FTP 服务器上的目录结构映射到本地缓存目录下。为了保证性能，我们仅在访问文件时才真正从服务器下载相应文件，避免大量无谓的下载；同样，我们也仅在访问目录时更新目录结构，尽量降低网络通信数据量。

以下我们也将分别就这两方面的实现进行详细说明。

三、具体实现

FTP 端实现

总体思路：

根据 RFC 959 协议，通过 Wireshark 抓取 FTP 操作中客户端和服务端之间数据包，分析客户端和服务端之间数据包的传输过程，并依此实现FTP操作。

函数 `ftp_login` 的实现：

用 Wireshark 抓取登录到 FTP 服务器时，客户端和服务端之间传输的数据包，如下图所示（服务端 IP 地址为 162.211.224.25，客户端 IP 地址为 10.0.2.15，下同）：

18	30.573809977	10.0.2.15	162.211.224.25	TCP	74 51012 → 21 [SYN] Seq=0 Win=29200 Len=0 MSS=1460 S
19	30.740932883	162.211.224.25	10.0.2.15	TCP	60 21 → 51012 [SYN, ACK] Seq=0 Ack=1 Win=65535 Len=0
20	30.740981425	10.0.2.15	162.211.224.25	TCP	54 51012 → 21 [ACK] Seq=1 Ack=1 Win=29200 Len=0
21	30.906789878	162.211.224.25	10.0.2.15	FTP	74 Response: 220 (vsFTPd 2.2.2)
22	30.906839712	10.0.2.15	162.211.224.25	TCP	54 51012 → 21 [ACK] Seq=1 Ack=21 Win=29200 Len=0
23	32.788599435	10.0.2.15	162.211.224.25	FTP	64 Request: USER zyh
24	32.788914664	162.211.224.25	10.0.2.15	TCP	60 21 → 51012 [ACK] Seq=21 Ack=11 Win=65535 Len=0
25	32.948700799	162.211.224.25	10.0.2.15	FTP	88 Response: 331 Please specify the password.
26	32.948827819	10.0.2.15	162.211.224.25	TCP	54 51012 → 21 [ACK] Seq=11 Ack=55 Win=29200 Len=0
27	34.484621361	10.0.2.15	162.211.224.25	FTP	67 Request: PASS zyh123
28	34.484890398	162.211.224.25	10.0.2.15	TCP	60 21 → 51012 [ACK] Seq=55 Ack=24 Win=65535 Len=0
29	34.963133631	162.211.224.25	10.0.2.15	FTP	77 Response: 230 Login successful.
30	34.963252415	10.0.2.15	162.211.224.25	TCP	54 51012 → 21 [ACK] Seq=24 Ack=78 Win=29200 Len=0
31	34.963428776	10.0.2.15	162.211.224.25	FTP	60 Request: SYST
32	34.963827974	162.211.224.25	10.0.2.15	TCP	60 21 → 51012 [ACK] Seq=78 Ack=30 Win=65535 Len=0
33	35.123043716	162.211.224.25	10.0.2.15	FTP	73 Response: 215 UNIX Type: L8
34	35.166619383	10.0.2.15	162.211.224.25	TCP	54 51012 → 21 [ACK] Seq=30 Ack=97 Win=29200 Len=0

从上图中可以看出，登录FTP服务器时的过程如下：

1. 从客户端向服务端 21 端口建立 TCP 连接（即控制连接）；

2. 服务端返回220；
3. 客户端在控制连接中发送`USER 用户名`发送用户名信息；
4. 服务端返回331；
5. 客户端在控制连接中发送`PASS 密码`发送密码信息；
6. 服务端返回230（如果登录成功）；
7. 客户端在控制连接中发送`SYST`请求服务端系统信息；
8. 服务端返回215。

函数`ftp_login`实现了登录到FTP服务器的过程。

函数 `ftp_get` 的实现：

用 Wireshark 抓取从FTP服务器上下载文件（数据连接使用被动模式，下同）时，客户端和服务端之间传输的数据包，如下图所示：

```
314 2375.3307553… 10.0.2.15       162.211.224.25  FTP      62 Request: TYPE I
315 2375.3310298… 162.211.224.25  10.0.2.15       TCP      60 21 → 51026 [ACK] Seq=97 Ack=38 Win=65535 Len=0
316 2375.5391446… 162.211.224.25  10.0.2.15       FTP      85 Response: 200 Switching to Binary mode.
317 2375.5392715… 10.0.2.15       162.211.224.25  TCP      54 51026 → 21 [ACK] Seq=38 Ack=128 Win=29200 Len=0
318 2375.5394729… 10.0.2.15       162.211.224.25  FTP      60 Request: PASV
319 2375.5399583… 162.211.224.25  10.0.2.15       TCP      60 21 → 51026 [ACK] Seq=128 Ack=44 Win=65535 Len=0
320 2375.7488296… 162.211.224.25  10.0.2.15       FTP     105 Response: 227 Entering Passive Mode (162,211,224,25,79,34)
321 2375.7489960… 10.0.2.15       162.211.224.25  TCP      74 49254 → 20258 [SYN] Seq=0 Win=29200 Len=0 MSS=1460 SACK_PERM=1 TSval=
322 2375.7905646… 10.0.2.15       162.211.224.25  TCP      54 51026 → 21 [ACK] Seq=44 Ack=178 Win=29200 Len=0
323 2375.9547728… 162.211.224.25  10.0.2.15       TCP      60 20258 → 49254 [SYN, ACK] Seq=0 Ack=1 Win=65535 Len=0 MSS=1460
324 2375.9548033… 10.0.2.15       162.211.224.25  TCP      54 49254 → 20258 [ACK] Seq=1 Ack=1 Win=29200 Len=0
325 2375.9549096… 10.0.2.15       162.211.224.25  FTP      64 Request: RETR aaa
326 2375.9551479… 162.211.224.25  10.0.2.15       TCP      60 21 → 51026 [ACK] Seq=178 Ack=54 Win=65535 Len=0
327 2376.1633943… 162.211.224.25  10.0.2.15       FTP-DA…  72 FTP Data: 18 bytes
328 2376.1634239… 10.0.2.15       162.211.224.25  TCP      54 49254 → 20258 [ACK] Seq=1 Ack=19 Win=29200 Len=0
329 2376.1634637… 162.211.224.25  10.0.2.15       TCP      60 20258 → 49254 [FIN, ACK] Seq=19 Ack=1 Win=65535 Len=0
330 2376.1634686… 162.211.224.25  10.0.2.15       FTP     115 Response: 150 Opening BINARY mode data connection for aaa (18 bytes).
331 2376.1634915… 10.0.2.15       162.211.224.25  TCP      54 51026 → 21 [ACK] Seq=54 Ack=239 Win=29200 Len=0
332 2376.1637236… 10.0.2.15       162.211.224.25  TCP      54 49254 → 20258 [FIN, ACK] Seq=1 Ack=20 Win=29200 Len=0
333 2376.1639153… 162.211.224.25  10.0.2.15       TCP      60 20258 → 49254 [ACK] Seq=20 Ack=2 Win=65535 Len=0
334 2376.4043465… 162.211.224.25  10.0.2.15       FTP      78 Response: 226 Transfer complete.
335 2376.4044079… 10.0.2.15       162.211.224.25  TCP      54 51026 → 21 [ACK] Seq=54 Ack=263 Win=29200 Len=0
```

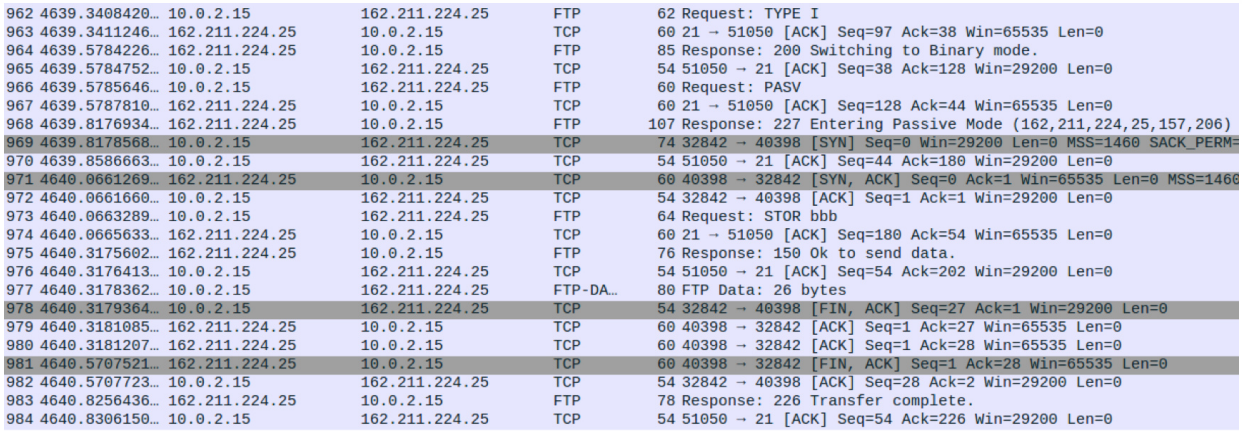
从上图中可以看出，从FTP服务器下载文件的过程如下：

1. 客户端在控制连接中发送`TYPE I`，表明数据连接类型为二进制传输；
2. 服务端返回200；
3. 客户端在控制连接中发送`PASV`，表示建立被动模式的数据连接；
4. 服务端返回227，消息尾部括号中的六个整数中，前4个表示服务端IP地址（IP地址为162.211.224.25），后两个表示数据连接的服务端端口号的高8位和低8位，这里数据连接的服务端端口号为79×28+34=20258；
5. 从客户端向服务端建立TCP连接（即数据连接）；
6. 客户端在控制连接中发送`RETR 文件名`，表示要获取服务端的指定文件；
7. 服务端返回150并向数据连接中发送文件内容；
8. 服务端返回226，表示传输完成。

函数`ftp_get`实现了从FTP服务器下载文件的过程，并将数据连接中收到的数据写入到本地文件中，以完成文件的下载。

函数 `ftp_put` 的实现：

用 Wireshark 抓取向FTP服务器上传文件时，客户端和服务端之间传输的数据包，如下图所示：



962	4639.3408420…	10.0.2.15	162.211.224.25	FTP	62	Request: TYPE I
963	4639.3411246…	162.211.224.25	10.0.2.15	TCP	60	21 → 51050 [ACK] Seq=97 Ack=38 Win=65535 Len=0
964	4639.5784226…	162.211.224.25	10.0.2.15	FTP	85	Response: 200 Switching to Binary mode.
965	4639.5784752…	10.0.2.15	162.211.224.25	TCP	54	51050 → 21 [ACK] Seq=38 Ack=128 Win=29200 Len=0
966	4639.5785646…	10.0.2.15	162.211.224.25	FTP	60	Request: PASV
967	4639.5787810…	162.211.224.25	10.0.2.15	TCP	60	21 → 51050 [ACK] Seq=128 Ack=44 Win=65535 Len=0
968	4639.8176934…	162.211.224.25	10.0.2.15	FTP	107	Response: 227 Entering Passive Mode (162,211,224,25,157,206)
969	4639.8178568…	10.0.2.15	162.211.224.25	TCP	74	32842 → 40398 [SYN] Seq=0 Win=29200 Len=0 MSS=1460 SACK_PERM=
970	4639.8586663…	10.0.2.15	162.211.224.25	TCP	54	51050 → 21 [ACK] Seq=44 Ack=180 Win=29200 Len=0
971	4640.0661269…	162.211.224.25	10.0.2.15	TCP	60	40398 → 32842 [SYN, ACK] Seq=0 Ack=1 Win=65535 Len=0 MSS=1460
972	4640.0661660…	10.0.2.15	162.211.224.25	TCP	54	32842 → 40398 [ACK] Seq=1 Ack=1 Win=29200 Len=0
973	4640.0663289…	10.0.2.15	162.211.224.25	FTP	64	Request: STOR bbb
974	4640.0665633…	162.211.224.25	10.0.2.15	TCP	60	21 → 51050 [ACK] Seq=180 Ack=54 Win=65535 Len=0
975	4640.3175602…	162.211.224.25	10.0.2.15	FTP	76	Response: 150 Ok to send data.
976	4640.3176413…	10.0.2.15	162.211.224.25	TCP	54	51050 → 21 [ACK] Seq=54 Ack=202 Win=29200 Len=0
977	4640.3178362…	10.0.2.15	162.211.224.25	FTP-DA…	80	FTP Data: 26 bytes
978	4640.3179364…	10.0.2.15	162.211.224.25	TCP	54	32842 → 40398 [FIN, ACK] Seq=27 Ack=1 Win=29200 Len=0
979	4640.3181085…	162.211.224.25	10.0.2.15	TCP	60	40398 → 32842 [ACK] Seq=1 Ack=27 Win=65535 Len=0
980	4640.3181207…	162.211.224.25	10.0.2.15	TCP	60	40398 → 32842 [ACK] Seq=1 Ack=28 Win=65535 Len=0
981	4640.5707521…	162.211.224.25	10.0.2.15	TCP	60	40398 → 32842 [FIN, ACK] Seq=1 Ack=28 Win=65535 Len=0
982	4640.5707723…	10.0.2.15	162.211.224.25	TCP	54	32842 → 40398 [ACK] Seq=28 Ack=2 Win=29200 Len=0
983	4640.8256436…	162.211.224.25	10.0.2.15	FTP	78	Response: 226 Transfer complete.
984	4640.8306150…	10.0.2.15	162.211.224.25	TCP	54	51050 → 21 [ACK] Seq=54 Ack=226 Win=29200 Len=0

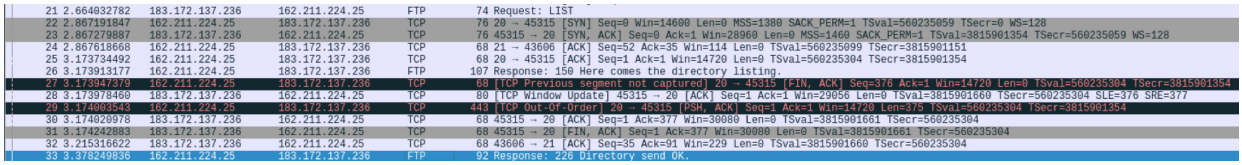
从上图中可以看出，向FTP服务器上传文件的过程如下：

1. 建立被动模式的数据连接，同下载文件过程中的第1-5步；
2. 客户端在控制连接中发送`STOR 文件名`，表示要将文件传输到服务端，指定上传文件在服务端上的文件名；
3. 服务端返回150，表示客户端可以开始上传文件；
4. 客户端在数据连接中发送文件内容；
5. 服务器返回226，表示传输完成。

函数`ftp_put`从本地文件读取文件内容，并实现了将本地文件上传到FTP服务器的过程。

函数 `ftp_dir` 的实现：

用Wireshark抓取在FTP服务器上调dir命令时，客户端和服务端之间传输的数据包，如下图所示：



21	2.664032782	183.172.137.236	162.211.224.25	FTP	74	Request: LIST
22	2.867191847	162.211.224.25	183.172.137.236	TCP	76	20 → 45315 [SYN] Seq=0 Win=14600 Len=0 MSS=1380 SACK_PERM=1 TSval=560235059 TSecr=0 WS=128
23	2.867279887	183.172.137.236	162.211.224.25	TCP	76	45315 → 20 [SYN, ACK] Seq=0 Ack=1 Win=28960 Len=0 MSS=1460 SACK_PERM=1 TSval=3815901354 TSecr=560235059 WS=128
24	2.867618668	162.211.224.25	183.172.137.236	TCP	68	21 → 43606 [ACK] Seq=52 Ack=35 Win=114 Len=0 TSval=560235099 TSecr=3815901151
25	3.173734492	162.211.224.25	183.172.137.236	TCP	68	20 → 45315 [ACK] Seq=1 Ack=1 Win=14720 Len=0 TSval=560235304 TSecr=3815901354
26	3.173913177	162.211.224.25	183.172.137.236	FTP	107	Response: 150 Here comes the directory listing.
27	3.173947379	162.211.224.25	183.172.137.236	TCP	68	[TCP Previous segment not captured] 20 → 45315 [FIN, ACK] Seq=376 Ack=1 Win=14720 Len=0 TSval=560235304 TSecr=3815901354
28	3.173978460	183.172.137.236	162.211.224.25	TCP	80	[TCP Window Update] 45315 → 20 [ACK] Seq=1 Ack=1 Win=29056 Len=0 TSval=3815901660 TSecr=560235304 SLE=376 SRE=377
29	3.174003543	162.211.224.25	183.172.137.236	TCP	443	[TCP Out-Of-Order] 20 → 45315 [PSH, ACK] Seq=1 Ack=1 Win=14720 Len=375 TSval=560235304 TSecr=3815901354
30	3.174029978	183.172.137.236	162.211.224.25	TCP	68	45315 → 20 [ACK] Seq=1 Ack=377 Win=30080 Len=0 TSval=3815901661 TSecr=560235304
31	3.174242883	183.172.137.236	162.211.224.25	TCP	68	45315 → 20 [FIN, ACK] Seq=1 Ack=377 Win=30080 Len=0 TSval=3815901661 TSecr=560235304
32	3.215316622	183.172.137.236	162.211.224.25	TCP	68	43606 → 21 [ACK] Seq=35 Ack=91 Win=229 Len=0 TSval=3815901660 TSecr=560235304
33	3.378249836	162.211.224.25	183.172.137.236	FTP	92	Response: 226 Directory send OK.

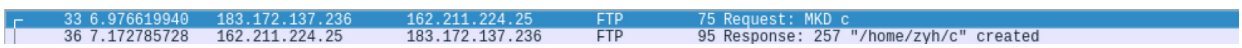
从上图中可以看出，与FTP服务器交互的过程如下：

1. 建立被动模式的数据连接，同下载文件过程中的第1-5步，但区别在于要使用ACSII连接；
2. 客户端在控制连接中发送`LIST`，表示列出当前目录下的文件列表；
3. 服务端返回150，表示服务器准备传输文件列表；
4. 客户端在数据连接中发送文件列表内容；
5. 服务器返回226，表示传输完成。

函数`ftp_dir`实现了从服务器上获取当前目录下文件列表的功能。

函数 `ftp_mkdir` 的实现：

用Wireshark抓取在FTP服务器上调mkdir命令时，客户端和服务端之间传输的数据包，如下图所示：



33	6.976619940	183.172.137.236	162.211.224.25	FTP	75	Request: MKD c
36	7.172785728	162.211.224.25	183.172.137.236	FTP	95	Response: 257 "/home/zyh/c" created

从上图中可以看出，与FTP服务器交互的过程如下：

1. 客户端在控制连接中发送`MKD 目录名`，表示新建名为该名字的 目录；
2. 服务器返回257，表示新建 目录完成。

函数`ftp_mkdir`实现了在服务器上新建 目录的功能。

函数 `ftp_rmdir` 的实现：

用Wireshark抓取在FTP服务器上调rm 目录命令时，客户端和服务端之间传输的数据包，如下图所示：

```
135 19.854132055  183.172.137.236      162.211.224.25       FTP        75 Request: RMD c
136 20.076279885  162.211.224.25       183.172.137.236      FTP       112 Response: 250 Remove directory operation successful.
```

从上图中可以看出，与FTP服务器交互的过程如下：

1. 客户端在控制连接中发送`RMD 目录名`，表示删除该名字的目录；
2. 服务器返回250，表示删除 目录完成。

函数 `ftp_rmdir` 实现了在服务器上删除 目录的功能。

函数 `ftp_rm` 的实现：

用Wireshark抓取在FTP服务器上调delete文件的命令时，客户端和服务端之间传输的数据包，如下图所示：

```
3425 134.125896228 183.172.137.236      162.211.224.25       FTP        80 Request: DELE a.tar
3426 134.362743073 162.211.224.25       183.172.137.236      FTP       102 Response: 250 Delete operation successful.
```

从上图中可以看出，与FTP服务器交互的过程如下：

1. 客户端在控制连接中发送`DELE 文件名`，表示删除该名字的文件；
2. 服务器返回250，表示删除文件完成。

函数 `ftp_rm` 实现了在服务器上删除文件的功能。

函数 `ftp_cd` 的实现：

用Wireshark抓取在FTP服务器上调cd命令时，客户端和服务端之间传输的数据包，如下图所示：

```
304 59.353334412  183.172.137.236      162.211.224.25       FTP        79 Request: CWD hello
305 59.600643872  162.211.224.25       183.172.137.236      FTP       105 Response: 250 Directory successfully changed.
```

从上图中可以看出，与FTP服务器交互的过程如下：

1. 客户端在控制连接中发送`CWD 目录名`，表示切换至该文件目录；
2. 服务器返回105，表示切换路径完成。

函数 `ftp_cd` 实现了在服务器上切换路径的功能

函数 `ftp_mv` 的实现：

用Wireshark抓取在FTP服务器上调rename命令时，客户端和服务端之间传输的数据包，如下图所示：

```
378 93.997175875  183.172.137.236      162.211.224.25       FTP        80 Request: RNFR a.zip
379 94.212525525  162.211.224.25       183.172.137.236      FTP        89 Response: 350 Ready for RNTO.
381 94.212736525  183.172.137.236      162.211.224.25       FTP        80 Request: RNTO a.tar
382 94.426037381  162.211.224.25       183.172.137.236      FTP        92 Response: 250 Rename successful.
```

从上图中可以看出，与FTP服务器交互的过程如下：

1. 客户端在控制连接中发送`RNFR 文件名`，表示被剪切的文件；
2. 服务器返回350，表示准备好剪切文件，等待下一步输入。
3. 客户端在控制连接中发送`RNTO 文件名`，表示粘贴文件的地址；
4. 服务器返回250，表示粘贴文件成功。

函数 `ftp_mv` 实现了在服务器上剪切文件的功能。

其他辅助函数的实现：

1. `ftp_response`：

   获取服务端的 response 信息头部的三位数字。实现过程为：

   ① 读取服务端返回的信息；

   ② 获取信息的前3个字符；

   ③ 将信息的前3个字符转换为整数并作为函数返回值。

2. `ftp_data_socket`：

   建立被动模式的数据连接。实现过程同“从FTP服务器下载文件”的过程的第1-5步，返回数据连接的 socket 描述符。函数 `ftp_data_socket` 的参数 `type` 表示数据连接的类型，值为 `"I"` 表示二进制传输，值为 `"A"` 表示ASCII传输。

FUSE 端实现

总体思路

在 FUSE 端，我们需要完成 FUSE 相关的文件操作，并在 `main` 函数中 调用 `fuse_main`，使 libfuse 在接收到 FUSE 内核模块发送来的请求后可以调用我们实现的文件操作函数完成 FTP 操作并返回。

而我们实现的文件操作函数，基本上是把对给定 path 的 “本地文件” 的访问解析为最接近的 FTP 的指令，并封装为对文件的读写。同时让本地来缓存服务器上目录结构（而非文件内容），以正确返回本地文件系统的信息。

实现的函数有 `getattr`, `access`, `mkdir`, `rename`, `unlink`, `truncate`, `create`, `open`, `read`, `write`, `release` 等。

文件操作具体实现

以传入 path、打开一个文件返回文件描述符 fd 的 `xmp_open`, `xmp_release` 函数为例：

```
static int xmp_open(const char *path, struct fuse_file_info *fi)
{
    int res, fd;
    char cache_path[PATH_MAX], ftp_path[PATH_MAX];
    // 生成该文件在本地对应的临时文件的路径
    map_to_cache_path(path, cache_path);
    createMultiLevelDir(cache_path);
    // 找到文件对应在 FTP 服务器上的路径
    map_to_ftp_path(path, ftp_path);

    // 以 write only 的方式，打开本地缓存文件，其中包含了错误处理
    // 从 FTP 服务器上 get 文件内容，确保打开的是最新的文件
    fd = open(cache_path, O_WRONLY);
    if (fd == -1)
```

```
    {
        return -errno;
    }
    res = ftp_get(fd, ftp_path);
    if (res == -1)
    {
        return -errno;
    }
    close(fd);

    // 以用户原本所希望的文件的打开方式打开该文件
    res = open(cache_path, fi->flags);
    if (res == -1) {
        return -errno;
    }

    fi->fh = res;
    return 0;
}

static int xmp_release(const char *path, struct fuse_file_info *fi)
{
    // 找到该文件在本地对应的临时文件的路径
    char ftp_path[PATH_MAX];
    map_to_ftp_path(path, ftp_path);

    // 将用户已经读写完毕后的临时文件内容，发送到 ftp 服务器上进行文件的更新
    int res = ftp_put(fi->fh, ftp_path);
    if (res == -1)
    {
        res = -errno;
    }
    close(fi->fh);
    return res;
}
```

此类函数主要的实现过程抽象包含：

1. 找到该文件在 FTP 服务器上对应的路径，以及本地临时文件的路径；
2. 以特定的模式打开临时文件，从 FTP 服务器上获取关于该文件的信息，经过处理后写入本地临时文件；
3. 以请求的模式访问临时文件，作为该文件的信息返回。

目录操作具体实现

与读写文件不同，用户操作目录时需要知道目录下所有文件的信息（如 ls 命令、GUI 中打开 目录等），同时 FTP 的 `LIST` 命令返回内容与 fuse 中 `readdir` 要求的返回结果有较大差异，因此本函数的实现也是所有 fuse 函数中最复杂的。

首先需要提供正确的文件权限、修改时间等信息，我们从 FTP LIST 函数返回的内容中选择对应的部分，用 pipe 的方法调用 `chmod`, `touch -d` 等程序，将本地临时文件的这些信息修改为与 ftp 服务器上一样的信息。之后使用 readdir 的默认部分对临时目录进行相同的操作。

其中一个必要的优化是，在获取目录内容的同时无需下载文件内容。对于文件，我们只使用 touch 来创建具有相同 attr 的文件；对于目录，我们只使用 mkdir，而不递归获取子目录。这是由于 readdir 会被频繁调用（如 ls，cd，以及在 bash 中按 tab 自动补全目录下文件名称时）。而真正的下载文件内容则在 `xmp_open` 时进行。

其他操作的实现

- `getattr` 函数：由于我们在 readdir 的时候已经将目录下的所有文件都创建了本地临时缓存文件，并将权限信息和修改时间信息等设置正确（虽然内容可能还没有下载，为空），因此在 `getattr` 时只需要将本地临时缓存文件的信息返回即可。
- 其他很多操作也类似，由于只涉及文件的元信息而不需要文件的具体内容，因此可以根据情况减少很多 ftp 操作，将操作转移到本地临时文件上即可。

问题以及解决方法

在使用 ftp 接口的过程中，发现由于 fuse 相关的函数并行地被调用，导致 ftp 多次读写缓冲区时出现冲突。找到问题之后使用了 `pthread` 库中的 `pthread_mutex_lock`, `pthread_mutex_unlock`，将上述提到的 ftp 函数变为原子操作，化解了问题。

不足以及优化想法

- `xmp_release` 函数在 close 文件时被调用，由于无法确定用户是否对其进行了更改，目前的方案是总是将该文件调用 ftp put，传到服务器上。但由于多数情况下用户对文件的访问以读为主，并且写具有一定的局部性，可以参考 rsync 的方法：首先服务器上计算校验和，在客户端上进行搜索、比对后，将修改部分使用增量同步的方式上传至 ftp 服务器，可以减少不必要的通信开销。但由于需要在 ftp 服务器端提供额外的函数（计算校验和等函数），并且整个算法实现过于复杂，我们仅找到此可以尝试的方案，没有具体实现。

四、效果展示

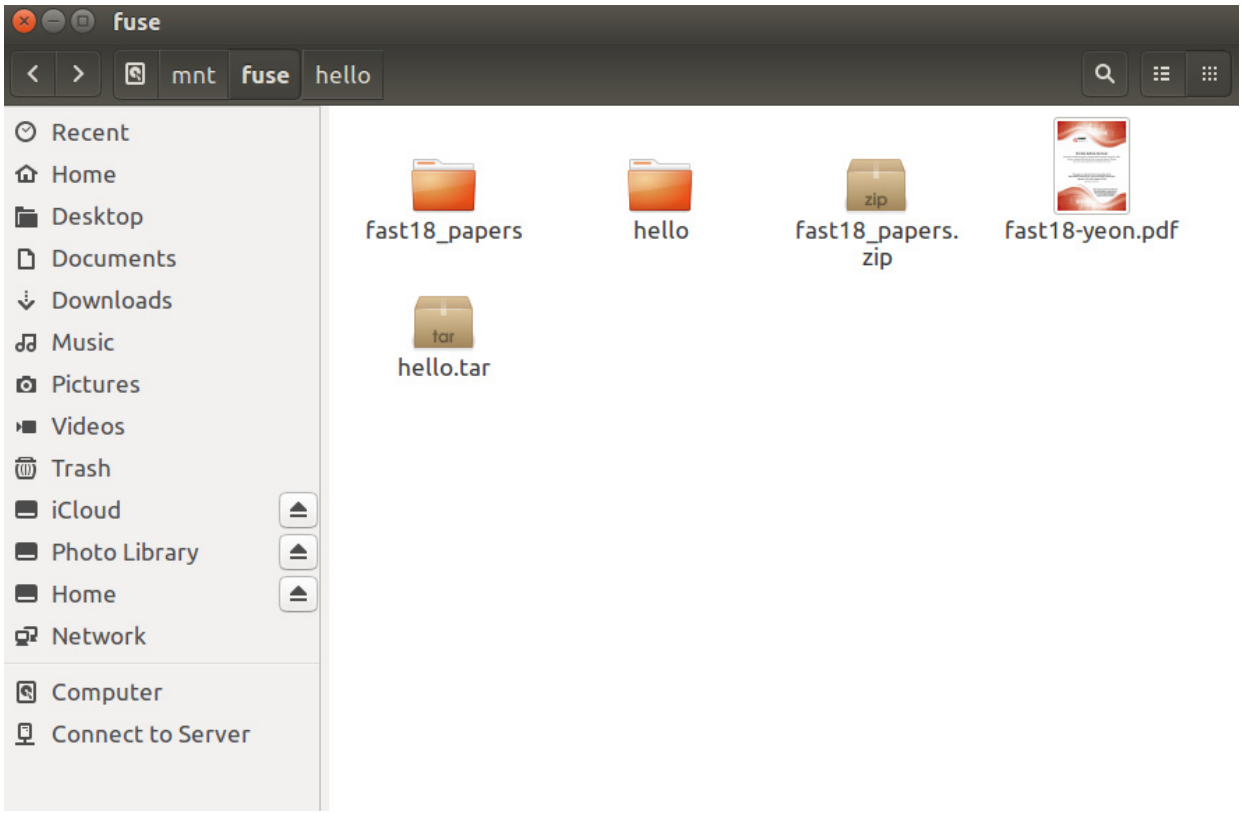
实验环境

我们使用了 Linux 的 libfuse 开源实现，并基于 libfuse 完成了 `FUSE-FTP` 文件系统。运行环境为 Ubuntu 16.04 LTS 。具体编译运行的方法以及相关库依赖可以参考我们工程根目录下的 README 文档。

运行效果

我们启动 `fuse-ftp`，并将 FTP 服务器目录映射到本地 `/mnt/fuse` 目录下。我们将分别使用图形化的文件管理器和终端进行效果展示：

- 图形化界面



我们的 `FUSE-FTP` 文件系统可以被文件管理器正常使用，包括查看，修改，创建删除等等。同时，可以看到经过一段时间的加载后文件的缩略图等信息可以正常显示，说明我们的实现可以很好地支持文件操作。

- 终端界面

```
# ivanium @ ubuntu in /mnt/fuse [17:10:31]
$ ls -la
ls: .Trash-1000: No such file or directory
ls: fast18-yeon.pdf: No such file or directory
ls: hello.tar: No such file or directory
ls: fast18_papers.zip: No such file or directory
ls: fast18_papers: No such file or directory
ls: hello: No such file or directory
total 3656
drwxrwxr-x 5 ivanium ivanium    4096 Jun 11 17:02 .
drwxr-xr-x 3 root    root       4096 Jun 11 16:57 ..
drwxr--r-- 4 ivanium ivanium    4096 Jun 11 09:02 fast18_papers
-rw-r--r-- 1 ivanium ivanium 2960355 Jun 11 09:03 fast18_papers.zip
-rw-r--r-- 1 ivanium ivanium  755411 Jun 11 09:05 fast18-yeon.pdf
drwxr--r-- 2 ivanium ivanium    4096 Jun  9 10:17 hello
-rw-r--r-- 1 ivanium ivanium     180 Jun 11 08:58 hello.tar
drwx------ 4 ivanium ivanium    4096 Jun 11 16:59 .Trash-1000
```

注意到服务器时间是 UTC+0 的，从图中我们可以看出文件的权限和修改时间均与 FTP 服务器同步，保证了文件权限和修改日期不会混淆。

五、心得体会

总体来说这次大作业我们收获也很多。从开发的过程上看，在整个过程中我们遇到了很多问题，也不断地通过合作与讨论解决了这些问题，最终完成了整个工程：

- 为了在本地调试 ftp 操作函数，我们学习了如何使用虚拟机搭建局域网；
- 为了了解 ftp 操作中发包的情况和包的内容，我们使用了 WireShark 进行了抓包并查看包经过解析后的内容；
- 为了在真实的机器上进行测试，我们配置了一台真实的 FTP 服务器；
- 为了解决开发平台的差异，我们组的同学们使用了物理机，虚拟机，docker等方法测试运行 `fuse-ftp` 程序；
- 除此以外，我们在开发中也遇到了很多问题（如并发网络发送接收包间的互相干扰等等），最终也通过思考讨论解决了代码中很多的 bug。

从开发的结果上看，我开发的结果上看，FUSE-FTP 也解决了我们平时的一个需求痛点。平时在想要随时与服务器同步文件时，一般只能使用 sshfs 等。现在我们可以使用 FTP 协议来进行文件同步，多了一个选项。目前我们已经在使用我们的 FUSE-FTP 文件系统，提高了工作效率。

六、参考资料

[1] libfuse. https://github.com/libfuse/libfuse

[2] RFC 959 FTP 协议规范 https://tools.ietf.org/html/rfc959