NFS

黄悦源
21307110014

一、设计思路

NFS服务器

服务器通过TCP socket监听客户端请求，提供如下接口：

- `open`: 打开并读取服务器目录下的文件
- `read`: 从文件中读取指定偏移和长度数据
- `write`: 写入数据到文件（缓存阶段）
- `close`: 保存本地缓存文件至服务器

使用`json`协议进行请求/响应通信，简化序列化与解析。

NFS客户端

客户端维护本地缓存目录并解析处理命令行传入的命令。

- `open`：从服务器下载文件为临时文件
- `read` / `write`：操作本地临时文件
- `close`：将本地缓存上传覆盖服务器文件

分布式锁机制

使用 Zookeeper 实现 `/nfslk/filename` 分布式锁。这里使用简单的加锁机制，即`open` 前获取锁，`close` 后释放锁，保证对文件的独有访问。

如果改用读写锁会有更好的并发性能，但读写时需要涉及对服务器的频繁访问，无法像当前实现在close时再把文件同步到服务器。

二、实现细节

服务端（nfs_server.py）

- 基于 socket + threading 处理并发客户端
- 接收 JSON 请求并解析方法名和参数
- 每个客户端请求在独立线程中处理
- 文件持久化在本地 server_files/ 目录

关键结构：

```
def handle_client(conn):  # 请求处理主逻辑
def open_file(path):  # 打开并返回内容
def read_file(path, offset, length):  # 读取内容
def write_file(path, offset, data):  # 写入
def close_file(path, data):  # 保存更新
```

客户端（nfs_client.py）

- 基于 KazooClient 使用 Zookeeper 加锁，指定非阻塞
- 使用 uuid 保证临时文件名唯一
- 客户端缓存目录：client_cache/
- 所有 read/write 操作基于本地缓存
- close 时上传更新文件至服务器并释放锁

主要结构：

```
def open(self, path):  # 加锁 + 下载文件
def read(self, local_path, offset, size):  # 本地读
def write(self, local_path, offset, data):  # 本地写
def close(self, path):  # 上传 + 解锁
```

三、运行结果

运行环境

- 系统：MacOS
- 启动ZooKeeper：brew services start zookeeper
- 启动服务器：python nfs_server.py
- 启动两个客户端：在另外两个终端中分别执行python nfs_client.py

- 同时访问同一文件 a.txt，观察锁机制与缓存生效

测试日志摘要

先启动服务器和两个客户端，在客户端1中打开a.txt。

服务器：

```
% python nfs_server.py
NFS server listening on 0.0.0.0:9000
[+] Connection from ('127.0.0.1', 51743)
[+] Connection from ('127.0.0.1', 51930)
```

可以看到接受了两个连接。

客户端1：

```
% python nfs_client.py
NFS Client is ready. Input command: open/read/write/close/exit
> open a.txt
Acquiring lock for a.txt...
Lock acquired.
Downloaded and cached as ./client_cache/8ab65daf-d747-41b0-b246-4f0892b6c778_a.txt
>
```

1中成功打开了a.txt。

客户端2：

```
% python nfs_client.py
NFS Client is ready. Input command: open/read/write/close/exit
> open a.txt
Acquiring lock for a.txt...
Open error: cannot acuire lock for a.txt
```

2中由于a.txt已被1上锁，打开失败。

接下来在客户端1内访问a.txt，然后关闭：

```
> write a.txt 0123456789
Wrote to local cache at offset 0.
> read a.txt 0 10
0123456789
> read a.txt 5 2
56
> write a.txt 111
Wrote to local cache at offset 0.
> read a.txt 0 10
1113456789
> close a.txt
Closed a.txt, lock released, cache removed.
> exit
```

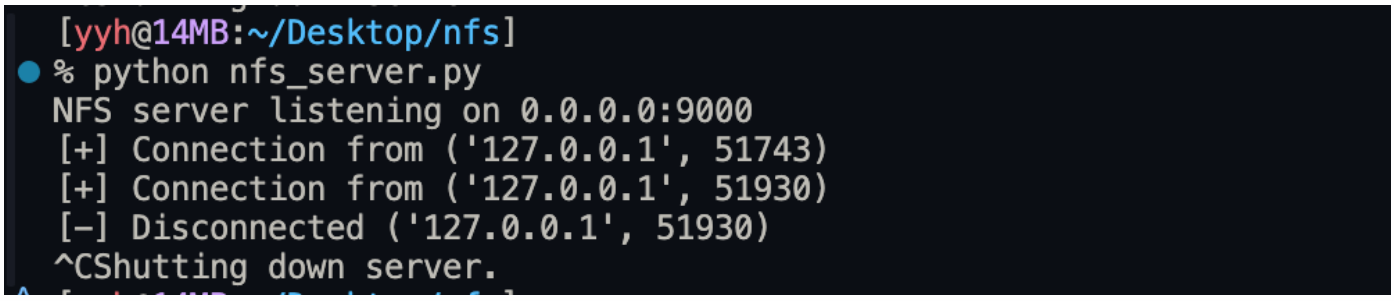
再在客户端2内访问a.txt：

```
> open a.txt
Acquiring lock for a.txt...
Lock acquired.
Downloaded and cached as ./client_cache/b6c4a2e9-e8d8-48a5-9594-0fc93c33acbc_a.txt
> read a.txt 0 10
1111111189
> write a.txt 222222
Wrote to local cache at offset 0.
> read a.txt 0 10
2222221189
> close a.txt
Closed a.txt, lock released, cache removed.
> exit
```

表明1中的修改对2可见。

完整运行结果：

服务器：



客户端1：

```
[yyh@14MB:~/Desktop/nfs]
% python nfs_client.py
NFS Client is ready. Input command: open/read/write/close/exit
> open a.txt
Acquiring lock for a.txt...
Lock acquired.
Downloaded and cached as ./client_cache/a2bd7b40-91f4-4c06-ae6b-c4b3a295f74e_a.txt
> write a.txt 0123456789
Wrote to local cache at offset 0.
> read a.txt 0 10
0123456789
> read a.txt 5 2
56
> write a.txt 111
Wrote to local cache at offset 0.
> read a.txt 0 10
1113456789
> close a.txt
Closed a.txt, lock released, cache removed.
> exit
```

客户端2：

```
[yyh@14MB:~/Desktop/nfs]
% python nfs_client.py
NFS Client is ready. Input command: open/read/write/close/exit
> open a.txt
Acquiring lock for a.txt...
Open error: cannot acuire lock for a.txt
> open a.txt
Acquiring lock for a.txt...
Lock acquired.
Downloaded and cached as ./client_cache/c37e2e8d-6aa2-4976-be55-08fc0de118c0_a.txt
> read a.txt 0 10
1113456789
> write a.txt 22
Wrote to local cache at offset 0.
> read a.txt 0 10
2213456789
> close a.txt
Closed a.txt, lock released, cache removed.
> exit
```

结果分析

- 多客户端通过Zookeeper成功互斥访问
- `read/write`均基于本地缓存文件进行，效率高
- `close`时自动上传覆盖原文件，确保一致性

四、总结

通过本实验，掌握了：

- 客户端-服务器文件操作接口设计
- 本地缓存机制实现
- 分布式锁（Zookeeper）的使用方式

对分布式文件系统有了更深入的理解。