

《计算机网络第四次作业》

班级：信安 2302 班

学号：202308060227

姓名：石云博

目录

一. 问题描述.....	2
二. 问题分析.....	3
三. 实验过程及代码.....	4
四. 结论.....	11
五. 参考文献.....	11

一.问题描述

What if you were TimBL

1. Suppose you were back to the time when Sir Timothy Berners-Lee OM OBE FREng FRS just about to invent HTTP. Your job is to design a protocol and a console UI software (both server end and client end) to help around 1,000 scientists to download the file they want from your SINGLE server
2. Because you now have modern programming tools, so we need to add a bit difficulty. Your protocol and software must be able to leverage that some users had already downloaded some files. When other users ask for the same file, you may offload the bandwidth burden to those who had downloaded. Detail about your design and prove your design by experiments in your dorm network

二.问题分析

对于基本文件分发功能,我们查到在 HTTP 诞生初期,整个互联网文件传输依赖单一服务器。为满足 1,000 个用户的文件请求,必须设计一种高效且简单的协议。设计思路:服务器存储所有文件原始副本,客户端通过简单的文本命令(如 LIST、GET)请求文件,服务器响应相应的文件列表或文件数据。

我们想要,利用已有下载文件的用户充当“节点”,当其他客户端请求同一文件时,优先从这些节点下载,从而减轻服务器压力。可以在基本协议上增加 REGISTER 命令。客户端下载完成后,将自身信息注册到服务器;当其他客户端请求文件时,服务器先返回已有该文件的节点列表 (PEERS 信息),客户端则选择其中一个节点进行下载。

类似技术有现代的 P2P 文件传输(如 BitTorrent),但本实验中简化为单服务器调度+简单的节点转发机制。

客户端与服务器通信协议我们可以设置为 LIST: 请求文件列表; GET <filename>: 请求指定文件; REGISTER <filename> <peer_port>: 客户端下载后向服务器注册自己 (peer_port 为客户端对外提供文件服务的端口)。

服务器响应可以是对 LIST 命令返回文件名称列表;对 GET 命令,若存在已注册节点,则返回格式为 PEERS ip1:port,ip2:port,... 的信息;否则直接返回文件内容。

客户端设计我们可以提供控制台 UI,允许用户查看文件列表并发起文件下载;下载时判断服务器返回的内容:若为 PEERS 信息,则选择一个节点下载,否则直接从服务器下载;下载完成后,注册自身并启动一个简单的服务线程,等待其他客户端请求文件。

这个问题需要解决两个核心需求：

基础文件分发：提供一个简单的集中式文件分发协议，让客户端可以通过命令交互获取文件列表和下载文件。

带宽分担机制：利用已下载文件的客户端作为“节点”，通过点对点传输减少中心服务器的压力。

三.实验过程及代码

对于系统整体设计与协议首先我们要确定通信协议：采用纯文本命令进行通信，协议命令包括：LIST：请求文件列表。GET <filename>：请求指定文件。服务器若已有其他节点注册，则返回 PEERS ip:port,...，否则直接返回文件内容。REGISTER <filename> <peer_port>：客户端下载后调用，将自己的 IP 与端口注册到服务器。

系统中存在的不同角色：服务器端：负责文件存储、命令解析及节点信息维护；客户端：负责与服务器交互、下载文件、向服务器注册及为其他客户端提供文件服务。

根据上面的分析，我们写下实现服务器端的代码：

服务器端代码（server.py）解释：

```
1 import os
2 import socket
3 import threading
4
5 # 服务器监听地址和端口
6 HOST = '0.0.0.0'
7 PORT = 8000
8
9 # 指定共享文件夹路径（请根据实际情况修改）
10 SHARED_FOLDER = '/home/kali/桌面/gx' # 例如：'/home/user/shared'
```

这里我们选择直接将服务器的监听端口设置为 8000，之后的客户端代码中就直接做相应设置。

os 模块：用于操作文件系统，如获取目录列表、判断文件是否存在等。

socket 模块：用于实现网络通信，创建 TCP 服务器。

threading 模块：用于在每次有客户端连接时启动一个新线程，以实现并发处理。

```
# 指定共享文件夹路径
SHARED_FOLDER = '/home/kali/桌面/gx'
```

我们在服务器上创建一个共享文件夹以供其他客户端下载文件。设置好 gx 文件夹的路径。

主干代码：

```
def handle_client(conn, addr):
    with conn:
        print(f"【服务器】连接来自 {addr}")
        data = conn.recv(1024).decode().strip()
        if not data:
            return
        parts = data.split()
        cmd = parts[0].upper()

        if cmd == "LIST":
            # 列出共享文件夹下所有文件（只列出普通文件）
            try:
                files = os.listdir(SHARED_FOLDER)
                file_list = [f for f in files if os.path.isfile(os.path.join(SHARED_FOLDER, f))]
                response = "\n".join(file_list) if file_list else "共享文件夹为空"
            except Exception as e:
                response = "ERROR: " + str(e)
            conn.sendall(response.encode())

        elif cmd == "GET" and len(parts) == 2:
            filename = parts[1]
            filepath = os.path.join(SHARED_FOLDER, filename)
            if os.path.exists(filepath) and os.path.isfile(filepath):
                try:
                    # 以二进制模式打开文件并读取内容
                    with open(filepath, "rb") as f:
                        file_data = f.read()
                        conn.sendall(file_data)
                except Exception as e:
                    conn.sendall(("ERROR: " + str(e)).encode())
            else:
                conn.sendall("ERROR: 文件不存在".encode())
        else:
            conn.sendall("ERROR: 未知命令".encode())
```

这里使用 with conn 保证连接结束后自动关闭。从连接中接收最多 1024 字节数据，使用 decode() 解码为字符串，并用 strip() 去除前后空白字符。

将接收到的字符串用空格分割，获取命令和参数。cmd 为命令部分，并转换为大写以方便后续比较。

创建命令“LIST”来打印出当前共享文件夹中的文件名：

当命令为 LIST 时，使用 os.listdir() 获取共享文件夹中的所有条目。

通过列表推导式和 os.path.isfile 筛选出文件（排除子目录）。

如果文件列表不为空，则将文件名称用换行符连接形成字符串；否则返回“共享文件夹为空”。

使用 conn.sendall 将结果编码后发送给客户端。如果发生异常，则发送错误信息。

“GET”来获取想要的文件：

当命令为 GET 且参数个数为 2（命令和文件名），将第二部分作为文件名。

利用 os.path.join 拼接得到完整的文件路径。

判断文件是否存在且确实是一个文件（而非目录）。

如果存在，则以二进制模式打开文件（确保能传输任何类型的文件数据），读取全部内容，然后用 conn.sendall 发送数据。

若文件不存在或者出现异常，则发送错误提示。

```
def run_server():
    with socket.socket(socket.AF_INET, socket.SOCK_STREAM) as s:
        s.bind((HOST, PORT))
        s.listen()
        print(f"【服务器】服务运行在 {PORT} 端口")
        while True:
            conn, addr = s.accept()
            threading.Thread(target=handle_client, args=(conn, addr), daemon=True).start()
```

设置监听和回显。创建一个 TCP 套接字，并使用 bind 绑定到设置的 HOST 和 PORT 上。

客户端代码：

```

1 import socket
2 import sys
3
4 SERVER_HOST = '192.168.219.130'
5 SERVER_PORT = 8000

```

导入 socket 和 sys 模块用于网络通信和处理系统退出等功能。

设置 SERVER_HOST 为服务器的实际 IP 地址，SERVER_PORT 与服务器端口保持一致。

```

def list_files():
    with socket.socket(socket.AF_INET, socket.SOCK_STREAM) as s:
        s.connect((SERVER_HOST, SERVER_PORT))
        s.sendall("LIST".encode())
        data = s.recv(4096).decode()
        print("【客户端】共享文件夹中的文件列表：\n" + data)

```

创建一个 TCP 套接字，并连接到服务器。

发送字符串命令 "LIST" 给服务器，要求获取共享文件列表。

使用 recv(4096) 接收服务器返回的数据，并用 decode() 解码成字符串。

打印返回的文件列表。

```

def download_file(filename):
    with socket.socket(socket.AF_INET, socket.SOCK_STREAM) as s:
        s.connect((SERVER_HOST, SERVER_PORT))
        s.sendall(f"GET {filename}".encode())
        file_data = s.recv(4096)

```

创建套接字并连接服务器，发送 "GET <文件名>" 命令，请求服务器返回对应的文件。直接用 s.recv(4096) 接收数据。这里的参数被我设置为 4096，表示这里会直接一次性接收 4096 字节，如果文件较大或网络较慢，我们可能需要使用到循环分块接收。

```

# 下载后注册为节点，并启动服务线程
register_with_tracker(filename)
threading.Thread(target=serve_file, args=(filename, file_data), daemon=True).start()

```

下载后将自己注册为节点，使得下一个请求该文件的客户端能直接从该客户端上获取文件。

```

if __name__ == "__main__":

    while True:
        print("\n命令: list, get <文件名>, exit")
        cmd = input("请输入命令: ").strip()
        if cmd.lower() == "exit":
            sys.exit(0)
        elif cmd.lower() == "list":
            list_files()
        elif cmd.lower().startswith("get"):
            parts = cmd.split()
            if len(parts) == 2:
                download_file(parts[1])
            else:
                print("用法: get <文件名>")
        else:
            print("未知命令")

```

一个简单的交互界面的设置，告知用户可以使用的指令。

我们使用一台虚拟机作为 server，另一台虚拟机为 client。在 kali 虚拟机中使用 python3 打开 server.py 文件：

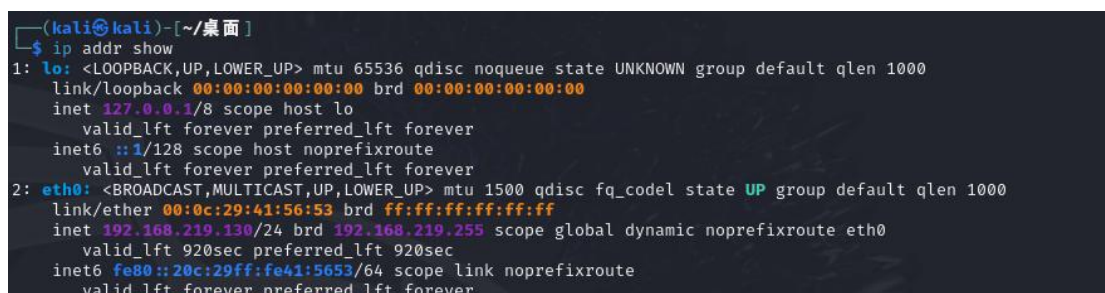


```

(kali㉿kali)-[~/桌面]
$ python3 server.py
【服务器】运行在 8000 端口

```

看到此时 kali 虚拟机已经作为服务器运行在了 8000 号端口中。然后我们在 oslab 虚拟机中打开 client 的文件，这里需要把代码中的 server host 改成 kali 虚拟机的 ip 地址，所以我们在 kali 虚拟机中使用指令“ip addr show”来获取当前的 ip 地址：



```

(kali㉿kali)-[~/桌面]
$ ip addr show
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN group default qlen 1000
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    inet 127.0.0.1/8 scope host lo
        valid_lft forever preferred_lft forever
    inet6 ::1/128 scope host noprefixroute
        valid_lft forever preferred_lft forever
2: eth0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc fq_codel state UP group default qlen 1000
    link/ether 00:0c:29:41:56:53 brd ff:ff:ff:ff:ff:ff
    inet 192.168.219.130/24 brd 192.168.219.255 scope global dynamic noprefixroute eth0
        valid_lft 920sec preferred_lft 920sec
    inet6 fe80::20c:29ff:fe41:5653/64 scope link noprefixroute
        valid_lft forever preferred_lft forever

```

这里有两个地址，注意我们使用第二个 eth0 的口，“lo”接口表示本地回环地址（127.0.0.1），仅用于同一台机器内部的通信。其他设备或虚拟机无法通过“lo”地址访问你的服务，所以需要使用网卡（如 eth0）分配的实际 IP 地址（例如

192.168.219.130), 这样才能保证网络中的其他设备能够连接到你的服务器。

完成后我们使用 client 接入 server:

```
命令: list, get <文件名>, exit  
请输入命令: list  
[客户端] 共享文件夹中的文件列表:  
t2  
test  
d
```

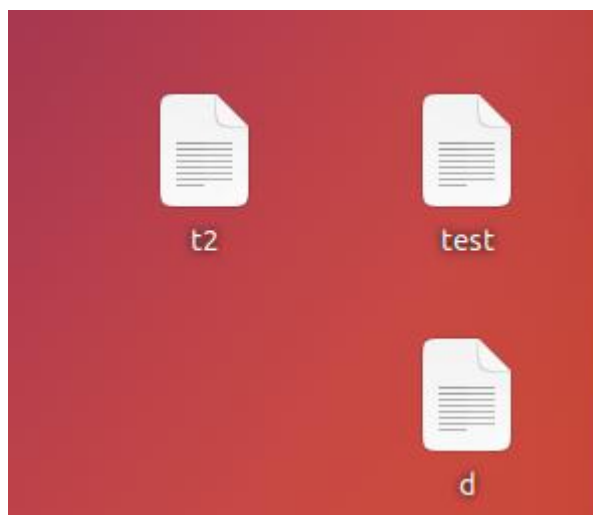
使用“list”指令来获取当前共享文件夹下的文件列表, 看到测试文件“

T2”。

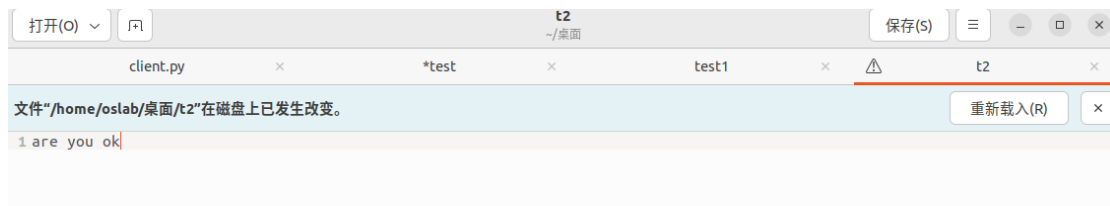
使用 get 指令来获取文件内容:

```
命令: list, get <文件名>, exit  
请输入命令: get t2  
[客户端] 文件 t2 下载完成。  
[客户端] 注册响应: 已注册 192.168.219.132:9001 对文件 t2
```

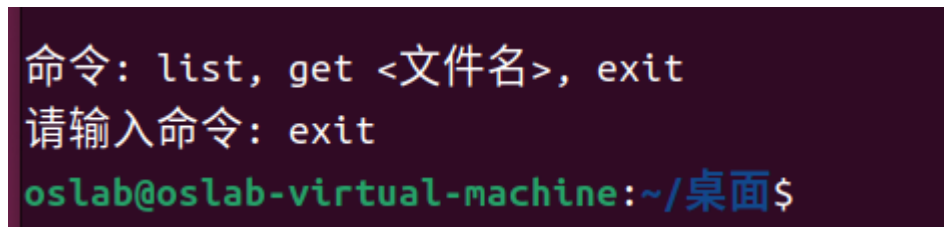
此时我们看到桌面上出现了 t2 文件



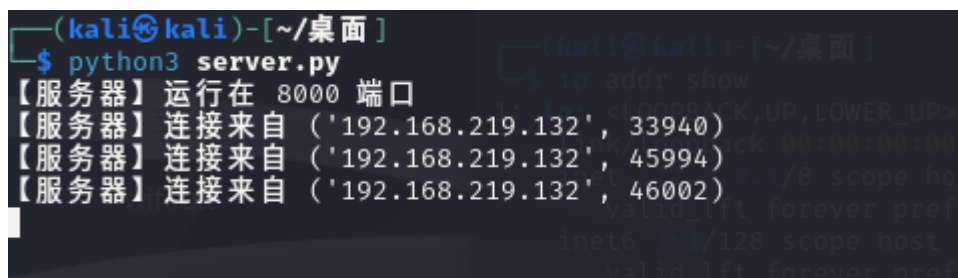
打开文件:



使用 exit 指令退出连接：



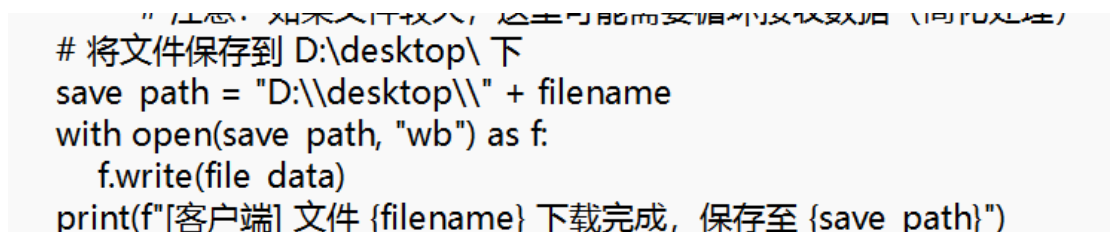
此时在服务器端我们也能看到有客户端接入：



可以看到此时服务器中有了接入客户端的信息，利用这个信息，我们可以实现让别的客户端请求文件时可以从该客户端中请求。

我们在 windows 下运行 client 代码，请求 t2。

刚开始在输入 get 请求之后程序会直接闪退，网上查找资料之后发现是 windows 对写入文件有权限的限制，所以我们要稍微修改一下 client 的代码，将下载路径直接设置到“D:\desktop”(我的桌面保存在 D 盘的)。



执行程序：

```
命令: list, get <文件名>, exit
请输入命令: get t2
[客户端] 从节点 192.168.219.132:9001 下载文件 t2
[客户端] 文件 t2 下载完成, 保存至 D:\desktop\t2
[客户端] 注册响应: 已注册 192.168.219.1:9001 对文件 t2
```

可以发现这里的 t2 是从节点 192.168.219.132 获取的, 并不是从 192.168.219.130.

也就是说当前客户端从上一个下载了文件 t2 的客户端那里获取并下载了文件 t2.

我们可以最后查看服务器的情况:

```
—$ python3 server.py
服务器] 文件共享服务运行在端口 8000
服务器] 连接来自 ('192.168.219.132', 38076)
服务器] 连接来自 ('192.168.219.132', 58512)
服务器] 连接来自 ('192.168.219.132', 58528)
服务器] 连接来自 ('192.168.219.1', 54601)
服务器] 连接来自 ('192.168.219.1', 54602)
服务器] 连接来自 ('192.168.219.1', 54604)
```

四.结论

首先我们分析了该文件主要的问题和实现思路, 然后我们设计了这套方案试图结合了早期 HTTP 协议的简单文本指令与现代 P2P 技术的思想: 用中心服务器来负责文件存储和节点调度; 用客户端除了从服务器获取文件外, 还主动充当“服务器”角色, 为其他用户提供文件数据。

这种架构在网络环境中 (例如宿舍网络或者我的不同虚拟机之间) 能够有效地分摊服务器负载, 降低中心服务器的带宽压力, 同时利用分布式下载加速文件传输。

虽然示例较为简单, 但已具备了基础功能。

五.参考文献

1 https://blog.csdn.net/sinat_26809255/article/details/121968641, cs 架构接口协议 (常用 socket 协议) 与 bs 架构接口协议 (常用 http 协议). 计算机辅助工程.

2021.12.16.

[2] <https://blog.csdn.net/XieWinter/article/details/102498947>. http 协议及基于 http 协议的文件下载. 专业游手好闲. 2019.10.12.

[3] <https://www.cnblogs.com/xiumusheng/p/18546500>. 使用 GET 方法请求 HTTP 下载文件.朽木生.2021.01.24.

[4] <https://hexingxing.cn/deeply-understand-the-internet-download-protocol/>. 深入理解互联网下载协议.何星星.2025.3.12.

[5] <https://www.cnblogs.com/jianyong-long/p/9693061.html>. 通过 httpClient 请求文件流（普通文件和压缩文件）示例. 一亩水稻田.2022.3.24

[6] https://blog.csdn.net/qq_33850438/article/details/79700133. P2P 原理以及如何实现（整理）. andy cong.2022.03.22.

[7] <https://blog.csdn.net/gengduc/article/details/132885610>. 集中式存储和分布式存储. gengduc. 2023.9.14.