

**计算机网络**

**课程实验报告**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 实验名称 | 实验 1：HTTP 代理服务器的设计与实现 | | | | | |
| 姓名 | 陈翔 | | 院系 | 计算机科学与技术学院 | | |
| 班级 | 1603109 | | 学号 | 1161800218 | | |
| 任课教师 | 李全龙 | | 指导教师 | 李全龙 | | |
| 实验地点 | 格物213 | | 实验时间 | 2018.10.27 | | |
| 实验课表现 | 出勤、表现得分(10) |  | 实验报告  得分(40) |  | 实验总分 |  |
| 操作结果得分(50) |  |
| 教师评语 | | | | | | |
|  | | | | | | |

****

|  |
| --- |
| 实验目的： |
| 熟悉并掌握 Socket网络编程的过程与技术；深入理解 HTTP协议，掌握HTTP代理服务器的基本工作原理；掌握HTTP代理服务器设计与编程实现的基本技能。 |
| 实验内容： |
| (1) 设计并实现一个基本 HTTP 代理服务器。要求在指定端口（例如 8080）接收来自客户的 HTTP 请求并且根据其中的 URL 地址访问该地址 所指向的 HTTP 服务器（原服务器），接收 HTTP 服务器的响应报文，并 将响应报文转发给对应的客户进行浏览。  (2) 设计并实现一个支持 Cache 功能的 HTTP 代理服务器。要求能缓 存原服务器响应的对象，并能够通过修改请求报文（添加 if-modified-since 头行），向原服务器确认缓存对象是否是最新版本。  (3) 扩展 HTTP 代理服务器，支持如下功能：  a) 网站过滤：允许/不允许访问某些网站；  b) 用户过滤：支持/不支持某些用户访问外部网站；  c) 网站引导：将用户对某个网站的访问引导至一个模拟网站（钓鱼）。 |
| 实验过程： |
| 1. Socket编程的客户端和服务器主要步骤（以TCP连接为例进行使用的函数说明）  （1）客户端  ①加载套接字库  WSAStartup()（但是实验中使用python，不需要这一步，以下省略）  ②创建套接字  socket.socket(family,type[,protocal])  ③向服务器发出连接请求  socket类中的connect(address)方法，一般address方法的格式为元组（hostname,port host），，如果连接出错，返回 果连接出错，返回socket.error错误  ④和服务器端进行通信  发送数据：  socket类中的send（string[,flag]）方法或者sendall(string[,flag]) 方法将 string中的数据发送到连接套字。返回值是要发送的字节数量，该数量可能小于string的字节大小；sendall将string中的数据发送到连接的套接字，但在返回之前会尝试发送所有数据，成功返回None，失败则抛出异常。  接收数据：  socket 类中的 recv（bufsize[,flag]）方法，数据以字符串形式返回，bufsize 指定要接收的最大数据量。flag 提供有关消息的其他信息，通常可以忽略。  ⑤关闭套接字  socket 类中的close()方法。  ⑥关闭加载的套接字库  WSACleanup()（python 中不需要，以下省略）  （2）服务器端  ①创建套接字  使用的函数及用法与客户端的相同。  ②绑定套接字到相应的 IP 地址和一个端口上  客户端不需要这一步，因为操作系统会完成，使用 socket 类中的 bind(address)方法，以元组（host,port）的形式表示地址  ③将套接字设置为监听模式等待连接请求  使用 socket 类中的 listen(backlog)方法，backlog 指定在拒绝连接之前，操作系统可以挂起的最大连接数量。该值至少为 1，大部分应用程序设为 5 即可。  ④请求到来后，接受连接请求，得到新的对应于此次连接的套接字  使用 socket 类中的 accept()方法，并返回（conn,address）,其中 conn 是新的套接字对象， 可以用来接收和发送数据。address 是连接客户端的地址。  ⑤用返回的套接字和客户端进行通信  使用的函数及用法与客户端的相同。  ⑥关闭套接字  使用的函数及用法与客户端的相同。    2. HTTP 代理服务器的基本原理  HTTP 代理服务器是网络的中间实体,既是Web服务器又是Web客户端。代理位于Web 客户端和 Web 服务器之间，扮演“中间人”的角色。对于客户端来说，代理扮演的是服务器 的角色，接收 request 并返回 response；对于服务器来说，代理扮演的是客户端的角色， 发送 request，接收 response。其通过接受客户端的请求并根据客户端的请求连接远程服务 器，并将请求发送给服务器，然后再将接收到的服务器的响应发送回客户端完成其基本功能。 而代理服务器的扩展功能可以通过对 已经接受 / 要发送的来自 客户端 / 服务器端 的报文进行一定的处理来实现。    3. HTTP 代理服务器的程序流程图 |
| 实验结果： |
| 1.准备工作：设置代理    2.验证http代理服务器基本功能  访问今日哈工大http://today.hit.edu.cn    可以正常访问  访问计算机科学与技术学院http://cs.hit.edu.cn    也能够正常访问  （2）验证缓存功能  访问今日哈工大网站：          Cache文件产生，命中和更新情况  （3）验证网站过滤  取消网站过滤功能的代码注释，将今日哈工大网址设为禁止访问    访问今日哈工大，已无法访问    （4）验证用户过滤功能  取消用户过滤功能的代码注释，将本机地址127.0.0.1设置为禁止用户    再次访问今日哈工大，已无法访问 |
| 问题讨论： |
| 1.输入/输出流代表什么：  DataOutputStream/DataInputStream代表着套接字创建成功后响应的输入流，请求的输出流。  2.缓存相关问题：缓存到哪里？如果缓存到主机，打印结果如何？  在本次实验中我缓存到了本地客户端，打印数据结构后如上图所示。该缓存信息中的状态码304指的是缓存命中并过期（Date+max-age > 当前时间），并且不更新缓存直接返回客户端。  3.cookie和cache区别?  cookie存放在客户端中，而cache一般存放在服务器端中。cookie存的是一些链接等字符串，有4k的大小限制，而cache则存储的是图片等文件。  4.代理服务器既充当客户端的角色又充当服务器的角色。  5. 使用缓存的代理服务器，需要做到保存请求报文相应的响应报文，顺序不能有差错而且信息不能有缺漏，使用日志，在每接受一次请求的时候，将请求的完整url保存到日志中，之后一旦得到相应信息直接保存在url下方，每次通过匹配url得知其下方的响应是否是所需的，这样方便查找和修改。  6.多线程相较于单线程的HTTP代理服务器，可以加快网站响应的进度。 |
| 心得体会： |
| 通过本次实验，我有以下几点收获：  ①更加深入地理解了 HTTP 协议，掌握了 HTTP 代理服务器的基本工作原理；  ②掌握了 HTTP 代理服务器设计与编程实现的基本技能；  ③熟悉并掌握了 Socket 网络编程的过程与技术；  ④学会使用 Python 语言的库进行 socket 编程 |

附录：实验源码

import hashlib  
import os  
import socket  
import time  
import threading  
from urllib.parse import urlparse  
  
  
config = {  
 'HOST': '127.0.0.1',  
 'PORT': 8888,  
 'MAX\_LENGTH': 4096,  
 'TIMEOUT': 100,  
 'CACHE\_SIZE': 1000  
}  
  
CACHE\_DIR = os.path.join(os.path.dirname(\_\_file\_\_), 'cache')  
if not os.path.exists(CACHE\_DIR):  
 os.mkdir(CACHE\_DIR)  
  
BLOCKED\_HOST = [  
 # 'today.hit.edu.cn'  
]  
  
BLOCKED\_USER = [  
 '127.0.0.1'  
]  
  
FISHING\_RULE = {  
 #'www.7k7k.com': 'cs.hit.edu.cn'  
}  
  
  
def isHostBlocked(host):  
 if host in BLOCKED\_HOST:  
 return True  
 return False  
  
  
def isUserBlocked(user):  
 if user in BLOCKED\_USER:  
 return True  
 return False  
  
  
class ProxyServer:  
 def \_\_init\_\_(self, host=config['HOST'], port=config['PORT']):  
 self.serverSocket = socket.socket(socket.AF\_INET, socket.SOCK\_STREAM)  
 self.serverSocket.bind((host, port))  
 self.serverSocket.listen(50)  
 self.host = host  
 self.port = port  
  
 def start(self):  
 print('Proxy server is listening on {host}:{port}...'.format(  
 host=self.host, port=self.port  
 ))  
 while True:  
 connect, address = self.serverSocket.accept()  
 proxyThread = threading.Thread(target=self.\_proxyThread, args=(connect, address))  
 proxyThread.start()  
  
 @staticmethod  
 def \_proxyThread(connect, address):  
 request = connect.recv(config['MAX\_LENGTH'])  
 if len(request) == 0:  
 return  
 http = request.decode('gbk', 'ignore').split('\n')[0]  
 if http.startswith('CONNECT'):  
 return  
 print(http)  
  
 url = urlparse(http.split()[1])  
  
 if url.hostname is None:  
 connect.send(str.encode('HTTP/1.1 404 Not Found\r\n'))  
 connect.close()  
 return  
  
 if isHostBlocked(url.hostname):  
 connect.send(str.encode('HTTP/1.1 403 Forbidden\r\n'))  
 connect.close()  
 return  
  
 if isUserBlocked(address[0]):  
 connect.send(str.encode('HTTP/1.1 403 Forbidden\r\n'))  
 connect.close()  
 return  
  
 if url.hostname == "today.hit.edu.cn":  
 temp = request.decode().replace(url.hostname, "cs.hit.edu.cn")  
 request = str.encode(temp)  
  
 port = 80 if url.port is None else url.port  
  
 m = hashlib.md5()  
 m.update(str.encode(url.netloc + url.path))  
 filename = os.path.join(CACHE\_DIR, m.hexdigest() + '.cached')  
 if os.path.exists(filename):  
 forwardSocket = socket.socket(socket.AF\_INET, socket.SOCK\_STREAM)  
 forwardSocket.settimeout(config['TIMEOUT'])  
 forwardSocket.connect((url.hostname, port))  
  
 temp = http + '\n'  
 t = (time.strptime(time.ctime(os.path.getmtime(filename)),  
 "%a %b %d %H:%M:%S %Y"))  
 temp += 'If-Modified-Since: ' + time.strftime(  
 '%a, %d %b %Y %H:%M:%S GMT', t) + '\n'  
 for line in request.decode().split('\n')[1:]:  
 temp += line + '\n'  
  
 forwardSocket.sendall(str.encode(temp))  
  
 first = True  
 while True:  
 data = forwardSocket.recv(config['MAX\_LENGTH'])  
 if first:  
 if data.decode('iso-8859-1').split()[1] == '304':  
 print('Cache hit: {path}'.format(path=url.hostname + url.path))  
 connect.send(open(filename, 'rb').read())  
 break  
 else:  
 o = open(filename, 'wb')  
 print('Cache updated: {path}'.format(path=url.hostname + url.path))  
 if len(data) > 0:  
 connect.send(data)  
 o.write(data)  
 else:  
 break  
 first = False  
 else:  
 o = open(filename, 'ab')  
 if len(data) > 0:  
 connect.send(data)  
 o.write(data)  
 else:  
 break  
 else:  
 print('Cache miss: {path}'.format(path=url.hostname + url.path))  
 forwardSocket = socket.socket(socket.AF\_INET, socket.SOCK\_STREAM)  
 forwardSocket.settimeout(config['TIMEOUT'])  
 forwardSocket.connect((url.hostname, port))  
  
 forwardSocket.sendall(request)  
  
 o = open(filename, 'ab')  
 while True:  
 data = forwardSocket.recv(config['MAX\_LENGTH'])  
 if len(data) > 0:  
 connect.send(data)  
 o.write(data)  
 else:  
 break  
 o.close()  
  
 connect.close()  
 forwardSocket.close()  
  
 cacheCounter = 0  
 cacheFiles = []  
 for file in os.listdir(os.path.join(os.path.dirname(\_\_file\_\_), 'cache')):  
 if file.endswith('.cached'):  
 cacheCounter += 1  
 cacheFiles.append(file)  
 if cacheCounter > config['CACHE\_SIZE']:  
 for i in range(len(cacheFiles)-1):  
 for j in range (i+1, len(cacheFiles)):  
 if os.path.getmtime(cacheFiles[i]) < os.path.getmtime(cacheFiles[j]):  
 temp = cacheFiles[i]  
 cacheFiles[i] = cacheFiles[j]  
 cacheFiles[j] = temp  
 for file in cacheFiles[config['CACHE\_SIZE']:]:  
 os.remove(file)  
  
 def stop(self):  
 mainThread = threading.current\_thread()  
 for thread in threading.enumerate():  
 if thread is mainThread:  
 continue  
 thread.join()  
 self.serverSocket.close()  
 exit(0)  
  
  
if \_\_name\_\_ == '\_\_main\_\_':  
 server = ProxyServer()  
server.start()