10 min. read

Python程序设计作业#5

2021-07-16 | 2020-11-30 | 1 | 0

Python程序设计#5作业

截止时间：2020年11月23日23:59:59

作业题目

在作业#4的基础上实现remoteProxy对每个用户进行单独流控

SQLite3数据库的每个用户的账号信息中增加带宽信息（用户名、密码、带宽）

带宽的单位为BPS（Bytes / Second，字节每秒），该带宽为某个用户的所有连接的转发数据总和带宽。

此次作业需要在【代码说明】中陈述流控的技术方案和实现机制。

作业内容

程序源代码嵌入下方的code block中。

local-proxy

```
1  import asyncio
2  import struct
3  import socket
4  import logging
5  logging.basicConfig(level=logging.INFO)
6  import nest_asyncio
```

```
nest_asyncio.apply()
import sys
import getopt
VERSION = 5
async def socks5(first, reader, writer):
    addr_from = writer.get_extra_info('peername')
    logging.info(f'connect from{addr_from}')
    header = await reader.read(1)
    header = first + header
    ver, num_method = struct.unpack("!BB", header)
    logging.info(f'ver == VERSION:{ver == VERSION}')
    logging.info('num_method = %d' % num_method)
    methods = []
    for i in range(num_method):
        methods.append(ord(await reader.read(1)))
    if 0 not in methods:#无需认证
        writer.close()
        writer.wait_closed()
        return
    #回应一个数据包，包括协议版本号，指定认证方法
    writer.write(struct.pack("!BB", VERSION, 0))
    await writer.drain()
    request = await reader.read(4)
    ver, cmd, rsv, atype = struct.unpack("!BBBB", request)
    assert ver == VERSION
    #ipv4
    if atype == 1:
        address = socket.inet_ntoa(await reader.read(4))
    #域名
    elif atype == 3:
        domain_length = await reader.read(1)
        address = await reader.read(domain_length[0])
    #ipv6
    elif atype == 4:
        address = socket.inet_ntop(socket.AF_INET6, await reader.read(16))
    else:
        writer.close()
        writer.wait_closed()
        return
    port = struct.unpack('!H', await reader.read(2))
    try:
        if cmd == 1:
            reader_remote,writer_remote = await asyncio.open_connection('127.0.0.
            http_connect = 'CONNECT ' + address + ':' + str(port[0]) + ' HTTP/1.1
            print('http_connect')

            http_connect += ' %' + username + '%' + password + '%'
```

```
            print(http_connect)
            writer_remote.write(http_connect.encode())
            await writer_remote.drain()
            reply = await (reader_remote.read(1024))
        else:
            writer.close()
            writer.wait_closed()
    except Exception as error:
        logging.error(error)
    reply = struct.pack("!BBBBIH", VERSION, 0, 0, 1, 0, 0)
    writer.write(reply)
    await writer.drain()

    #第一个字节为0表示成功代理
    if cmd == 1 and reply[1] == 0:
        tasks = [read_trans(reader, writer_remote), write_trans(reader_remote, wr
        await asyncio.wait(tasks)

async def read_trans(reader, writer_remote):
    while True:
        data = await reader.read(4096)
        if not data:
            logging.info('disconnect')
            break
        writer_remote.write(data)
        await writer_remote.drain()

async def write_trans(reader_remote, writer):
    while True:
        data = await reader_remote.read(4096)
        if not data:
            logging.info('disconnect')
            break
        writer.write(data)
        await writer.drain()

async def httptunnel(first, reader, writer):
    http_connect = (await reader.read(1024))
    http_connect = (first + http_connect).decode()
    http_connect += ' %' + username + '%' + password + '%'
    logging.info(http_connect)

    reader_remote,writer_remote = await asyncio.open_connection('127.0.0.1',10010
    writer_remote.write(http_connect.encode())
    await writer_remote.drain()


```

```
        reply = await (reader_remote.read(1024))
        writer.write(reply)
        await writer.drain()
        #连接建立成功
        tasks = [read_trans(reader, writer_remote), write_trans(reader_remote, writer
        await asyncio.wait(tasks)

async def test(reader, writer):
    first = await reader.read(1)
    if(first == b'\x05'):
        await socks5(first, reader, writer)
    elif(first == b'C'):
        await httptunnel(first, reader, writer)

username = ''
password = ''

async def main():
    global username, password
    if(len(sys.argv) != 3):
        logging.info('usage: local-proxy.py username, password')
    else:
        username = sys.argv[1]
        password = sys.argv[2]
    print(username)

    print(password)
    server = await asyncio.start_server(test, '0.0.0.0', 10086)
    async with server:
        await server.serve_forever()

asyncio.run(main())
```

remote-proxy

```
import asyncio
import struct
import socket
import logging
import time
logging.basicConfig(level=logging.INFO)
import nest_asyncio
nest_asyncio.apply()
import aiosqlite
```

```
async def handle(reader_local, writer_local):
    print('start working')
    global username_to_token_bucket
    db = await aiosqlite.connect('account.db')
    http_connect = (await reader_local.read(1024))
    http_connect = http_connect.decode()
    logging.info(http_connect)

    i = 0
    while(http_connect[i] != ':'):
        i += 1
    domain_name = http_connect[8 : i]
    j = i
    while(http_connect[j] != ' '):
        j += 1
    port = http_connect[i + 1 : j]
    i = 0
    while(http_connect[i] != '%'):
        i += 1
    j = i + 1
    while(http_connect[j] != '%'):
        j += 1
    k = j + 1
    while(http_connect[k] != '%'):
        k += 1
    username = http_connect[i + 1: j]
    password = http_connect[j + 1: k]
    print(username)
    print(password)
    sql = 'SELECT * FROM accout where username = \'' + username + '\' and passwor
    print(sql)
    cursor = await db.execute(sql)
    row = await cursor.fetchall()
    await cursor.close()
    if(len(row) != 1):
        logging.error('wrong account')
        return
    else:
        logging.info('right account')
    if(username not in username_to_token_bucket.keys()):#用户名不在dict中，要创建令
        username_to_token_bucket[username] = 0
        print('init bucket')
    reader_remote,writer_remote = await asyncio.open_connection(domain_name,port)
    sql = 'select speed FROM accout where username = \'' + username + '\''
    cursor = await db.execute(sql)
    row = await cursor.fetchall()
    await cursor.close()
```

```
    speed = row[0][0]
    reply = 'HTTP/1.1 200 OK\r\n\r\n'
    writer_local.write(reply.encode())
    await writer_local.drain()

    tasks = [read_trans(reader_local, writer_remote), write_trans(reader_remote,
    await asyncio.wait(tasks,return_when=asyncio.FIRST_COMPLETED)
    await db.close()


async def read_trans(reader, writer_remote):
    while True:
        data = await reader.read(4096)
        if not data:
            logging.info('disconnect from clinet')
            return
        writer_remote.write(data)
        await writer_remote.drain()

async def write_trans(reader_remote, writer, username, speed):
    global username_to_token_bucket
    data = ''
    while True:
        if(data == ''):
            data = await reader_remote.read(int(0.01 * speed))# 10.01s    speed
        if not data:
            logging.info('disconnect from server')
            return
        if(username_to_token_bucket[username] < 10):#如果桶里的令牌不够那么就等待，让
            # 目的是让cpu去执行其他部分的代码，防止在此处阻塞
            await asyncio.sleep(0)
            continue
        else:#令牌够了，把data发出去，同时把data清空
            username_to_token_bucket[username] -= 10
            writer.write(data)
            await writer.drain()
            data = ''
async def token_bucket_plus_one():
    global username_to_token_bucket
    while True:
        for k in username_to_token_bucket.keys():# 每1秒可以攒够1000个令牌，所以平均
            if username_to_token_bucket[k] < 10000:
                username_to_token_bucket[k] += 10
                print(username_to_token_bucket[k])
        await asyncio.sleep(0.01)
async def _task():
    server = await asyncio.start_server(handle, '127.0.0.1', 10010)
```

```
104         async with server:
105             await server.serve_forever()
106     async def main():
107         tasks = [_task(), token_bucket_plus_one()]
108         await asyncio.wait(tasks)
109     username_to_token_bucket = {}
110     asyncio.run(main())
```

代码说明

源代码中不要出现大段的说明注释，所有文字描述在本节中以行号引用说明。

用一个全局变量username_to_token_bucket{username, bucket}表示某个用户对应令牌桶的令牌数量

这样就解决了同一用户的所有连接的总速率不会超过限制的速率

每个用户的令牌桶令牌数每隔一段时间自动增加，一旦加到桶满了，就不加了

当有数据要发送时，首先检查令牌够不够，如果够，直接发送，然后把桶里的令牌数减少

如果桶里令牌数不够，那么就等待，注意，为了不使程序阻塞，这里使用了

```
1   await asyncio.sleep(0)
2   continue
```

来让出cpu，使cpu去执行程序的其他部分

‹ Python程序设计作业#3 Python程序设计作业#6 ›

提交

来发评论吧~