计算机网络课程报告

Protocol Design Project

网络聊天室实验报告

学生姓名 李润泽

学院名称 智能与计算学部

专　　业 计算机科学与技术

班　　级 2019 级计科 1 班

学　　号 3019244266

时　　间 2021 年 4 月 19 日

目录

一、实践内容概述

本次实验要求学生通过 socket 编程技术，设计网络通信协议，并开发一个网络聊天室，实现多客户端的接入与通信功能，然后对其进行系统的评价与测试。本实验旨在通过实验来让同学理解 TCP 和 UDP socket 基本原理，掌握 socket 编程的基本技能；并理解网络应用层协议架构的基本原理，提升网络应用层协议设计和实现的能力。

【关键字】网络聊天室，socket 编程技术，网络通信协议

二、需求分析

2.1 背景

在互联网发展如此发达的今天，通信工具，尤其是即时通信工具已经成为我们日常生活中必不可少的一部分。如今人们对于网络的依赖性越来越强，由此衍生的聊天工具愈发多样，如 QQ、微信、MSN 等。由此我进行了一个关于设计网络聊天室的实验，旨在通过 TCP/UDP socket 上进行开发，把所学的课程相关知识加以融会贯通，全面理解网络编程的含义。

2.2 目标

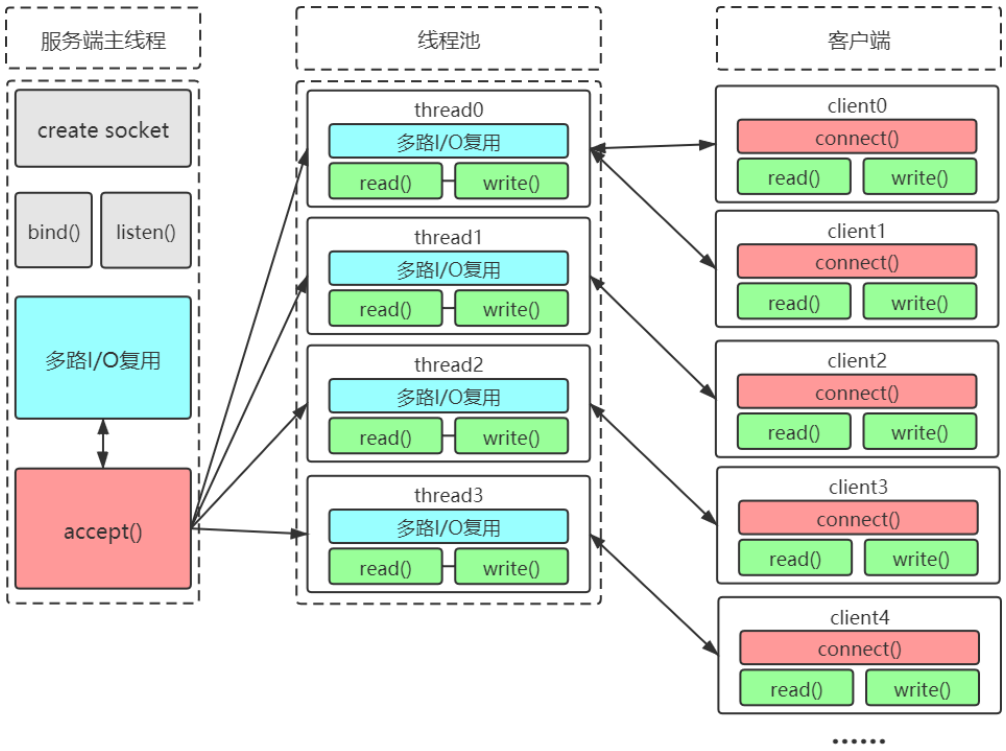
本次实验目标室编写一个基于 socket 编程以及 TCP/UDP 网络协议的多人网络聊天室。通过 socket 编程建立一个服务器端和若干客户端，服务器端负责管理网络聊天室、客户端，而客户端连入客户端进行文本聊天、传输文件以及语音聊天。

我选用 TCP 协议进行编写：

在服务器端，首先我们需要调用 socket()函数创建一个 socket，再调用 bind()函数获取地址，调用 listen()函数监听 socket，接收客户端需要调用 connect()，在监听到 connect()的请求后，调用 accept()函数取出请求，已达到建立连接的目的。在网络 I/O 操作中，调用 read()与 write()函数来接收和发送消息。关闭聊天室需要调用 close()函数，从而断开连接。

在客户端，首先调用 socket()创建 socket，调用 connect()连接聊天室，调用 read()和 write()函数接收和发送消息。最后调用 close()函数关闭 socket。

2.3 系统架构



（1）服务端在监听之前就开辟固定数量的工作线程；

（2）主线程使用多路 I/O 复用方式接收连接，并分发至各个工作线程；

（3）每个工作线程负责与大量客户端进行通讯。

2.4 功能需求（后续内容会形成表格进行总结）

服务端	客户端
（1）服务端在进行初始化时，根据预设的 IP 地址和端口进行监听，等待客户端的连入；	（1）客户端进行初始化时，输入自己的用户名，以及用户希望连入的聊天室地址，发出连入的申请；
（2）服务端应区分用户，防止重名；	（2）用户进入聊天室，可以让聊天室的其他人知道；
（3）服务端需要负责聊天信息以及文件的接收和转发，让所有聊天室的成员看到；	（3）用户在聊天室里发送消息、文件，聊天室里全员可见；
（4）服务端可以实现语音聊天；	（4）服务端应正确应对粘包和分包的问题；

（5）服务端应正确应对粘包和分包、掉线等异常问题；	（5）客户端接收并展示服务端发来的消息，消息包含用户名与内容，这样可以做到分辨内容的来源；
（6）服务端可以承受上千客户端的连接与聊天的过程；	（6）用户退出聊天室。
（7）有客户端退出的时候，不会影响服务端的正常运行；	
（8）服务端关闭的时候，向所有客户端发送关闭命令。	

2.5 性能需求

（1）服务端显示页面整洁，可以正确处理消息内容，一般显示形式为“用户名：内容”；

（2）服务端可以正确处理异常情况；

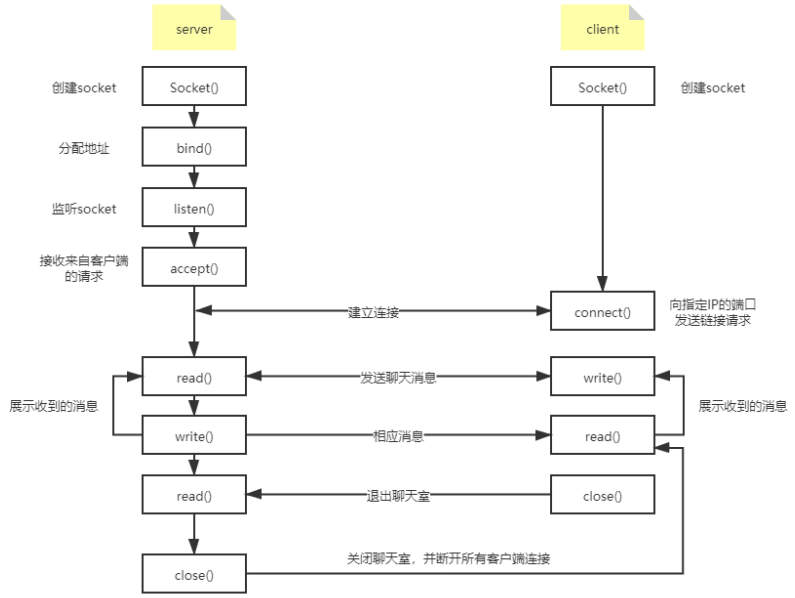
（3）聊天室可以做到高并发优化，可以让上千人在统一聊天室同时聊天；

（4）文件和语音可以正常传输；

（5）客户端发送消息可以让所有人看见；

（6）客户端可以正确、快速连接不掉线。

三、系统设计



四、协议设计

本次实验基于 TCP 协议，在使用 TCP 的基础上，处理粘包分包异常的过程中，我们需要对数据进行包装。

为了解决这两个问题，我们需要自己定义一份协议，标准为：**消息头部+消息长度+消息正文**。

我们需要把从 socket 读取出来的数据放到 dataBuffer 的后面，如果 dataBuffer 的内容长度小于消息长度，则跳出小循环继续接收；大于消息长度，则从缓冲区读取包头并获取包体的长度，再判断整个缓冲区是否大于消息头部+消息长度，如果小于则跳出小循环继续接收，如果大于则读取包体的内容，然后处理数据，最后再把这次的消息头部和消息正文从 dataBuffer 删掉。

代码如下：

```
HEADERSIZE = 12
HEART_BEAT = 0
NORMAL = 1
def packData(data, packetType):
    version = 1
    bodyLen = len(data)
    header = [version, packetType, bodyLen]
    headPack = struct.pack("!3I", *header)
    return headPack + data
```

五、实现方式

5.1 操作系统：Windows 10

5.2 CPU：Intel(R) Core(TM) i7-9750H CPU @ 2.60GHz 2.59 GHz

5.3 内存：16.0GB

5.4 编程语言：Java（实现基本功能）、Python（实现基本功能与扩展功能）

5.5 依赖的类库：

5.5.1 Java：

```
import java.io.*;
import java.net.Socket;
```

```
import java.io.IOException;
import java.net.ServerSocket;
import java.net.Socket;
import java.util.List;
import java.util.Vector;
```

5.5.2 Python：

```
import struct
from socket import *
from common import packData, HEART_BEAT, NORMAL, HEADERSIZE
import threading
import select
import time
from locust import TaskSet, task, between, Locust, events, User
```

5.6 项目文件组织结构

5.6.1 Java

```
chatroom（实现基础功能）
    |—— .idea
    |—— src
    |      |—— client.java          #客户端
    |      |—— server.java          #服务端
    |      |—— ServerThead.java     #多线程处理
    |—— chatroom.iml
```

5.6.2 Python

```
chatroom（实现基础功能与扩展功能）
    |—— client.py                   #客户端
    |—— common.py                   #协议规则
    |—— disconnect_client.py        #客户端（处理掉线问题）
    |—— locustfile.py               #高并发处理
    |—— server.py                   #服务端
    |—— StickyAndSpilt_client.py    #客户端（处理粘包和分包问题）
```

六、性能评价

6.1 系统可行性评价

服务器端功能	完成情况
初始化服务器端	完成
创建聊天室	完成
处理客户端发来的请求	完成
记录不同客户端	完成
记录当前聊天室的用户数量	完成
处理异常情况	完成
对高并发进行优化	完成
文件传输	未完成
语音聊天	未完成
关闭聊天室	完成

客户端功能	完成情况
初始化客户端	完成
连接聊天室	完成
发布消息	完成
接收服务端的消息	完成
退出聊天室	完成

6.2 聊天室功能详细描述

6.2.1 基础功能 1-4：略（已展示）

6.2.2 扩展功能 1：

（1） 粘包和分包

粘包：发送方发送两个字符串“hello”+“world”，而接收方却一次性接收到“helloworld”。出现粘包是因为 TCP 为了提高网络的利用率，使用一个叫做 Nagle 的算法，如果发送的数据很少的话，会延迟发送。如果应用层给 TCP 传送数据很快的话，会把两个应用层的数据报“粘”在一起，最后只发一个 TCP 数据包给接收端。

分包：发送方发送一个字符串“helloworld”，但接收方接收到两个字符串“hello”和“world”。出现分包是因为 TCP 是以段为单位发送数据的，建立 TCP 链接后，如果应用层数据包超过最大消息长度，会把应用层数据包拆分，分成两个段来发送。

为了解决这两个问题，我们需要自己定义一份协议，标准为：消息头部+消息长度+消息正文。我们需要把从 socket 读取出来的数据放到 dataBuffer 的后面，如果 dataBuffer 的内容长度小于消息长度，则跳出小循环继续接收；大于消息长度，则从缓冲区读取包头并获取包体的长度，再判断整个缓冲区是否大于消息头部+消息长度，如果小于则跳出小循环继续接收，如果大于则读取包体的内容，然后处理数据，最后再把这次的消息头部和消息正文从 dataBuffer 删掉。

解决粘包问题效果如下：

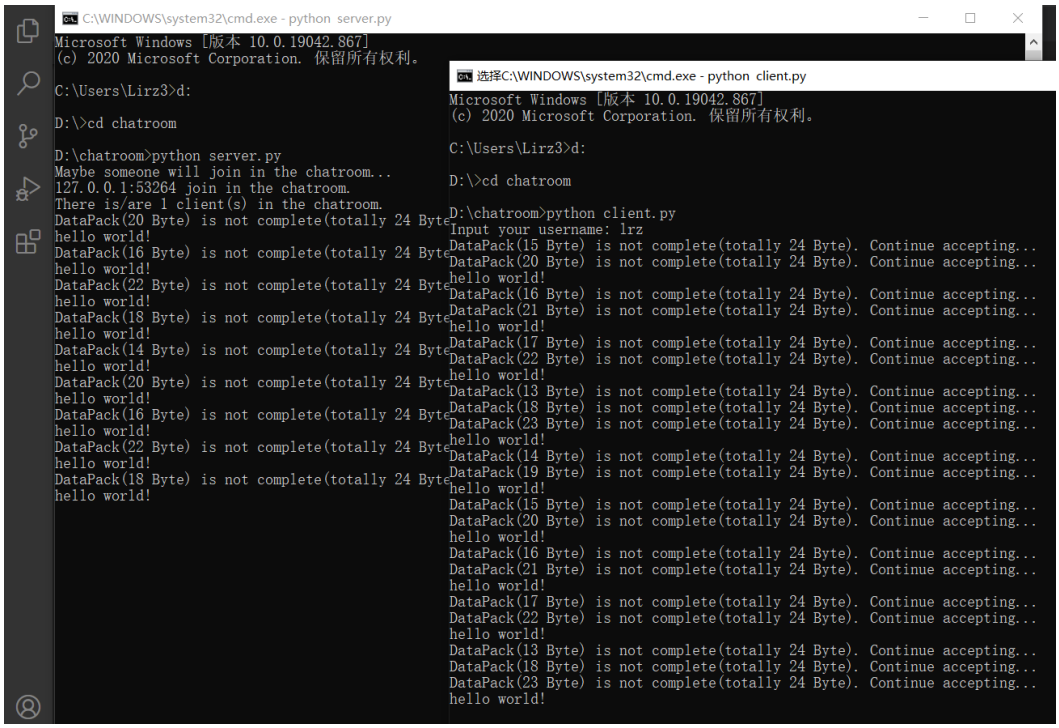
```
25  def sendText():
26      global nickName
27      #while True:
28          #sentence = input()
29      sentence = 'hello world!'
30      for i in range(1,10):
31          if sentence == 'exit':
32              # Client wanna exit the chatroom.
33              client.send(packData(b'q', NORMAL))
34          else:
35              #sentence = nickName + ': ' + sentence
36              client.send(packData(sentence.encode('utf-8'), NORMAL))
37              #time.sleep(1)
```

```
C:\WINDOWS\system32\cmd.exe - python  server.py
Microsoft Windows [版本 10.0.19042.867]
(c) 2020 Microsoft Corporation.  保留所有权利。

C:\Users\Lirz3>d:

D:\>cd chatroom

D:\chatroom>python server.py
Maybe someone will join in the chatroom...
127.0.0.1:53081 join in the chatroom.
There is/are 1 client(s) in the chatroom.
hello world!
hello world!
hello world!
hello world!
hello world!
hello world!
hello world!
hello world!
hello world!
```

```
C:\WINDOWS\system32\cmd.exe - python  client.py
Microsoft Windows [版本 10.0.19042.867]
(c) 2020 Microsoft Corporation.  保留所有权利。

C:\Users\Lirz3>d:

D:\>cd chatroom

D:\chatroom>python client.py
Input your username: lrz
hello world!
hello world!
hello world!
hello world!
hello world!
hello world!
hello world!
hello world!
hello world!
```

解决分包问题效果如下：

```
while True:
    #data = client.recv(1024)
    data = client.recv(5)
```



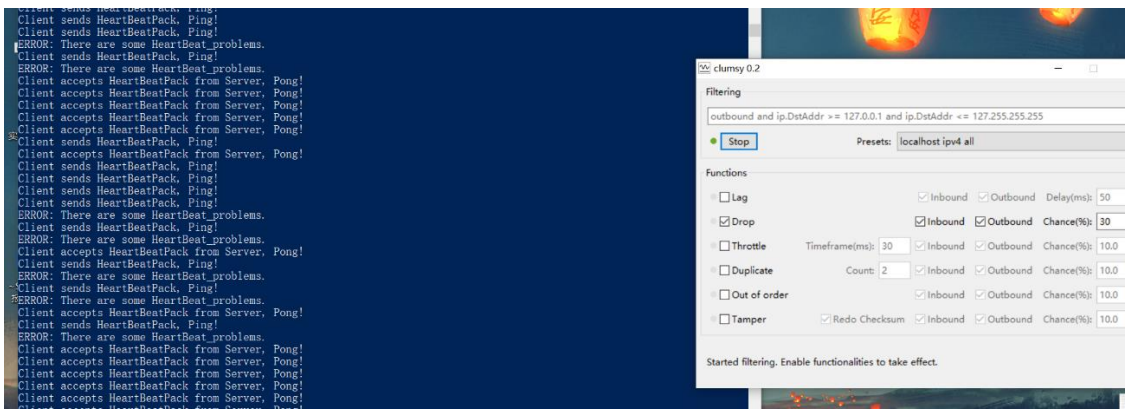
（2）掉线异常

TCP 连接不是物理的连接，而是由通讯双方维护的逻辑上的连接，需要消耗资源（如内存和端口），操作系统为提升网络效率，设置了 TCP KeepAlive 机制，指的是系统会自动关闭并清除长时间没有数据传输的 TCP Socket 连接。

为解决这个问题，我们通过心跳检测手段完成。

每间隔固定时间向对方发送一个心跳包，并验证对方可以正确回应心跳包。这样可以确认目前链路畅通，无故障发生。

心跳检测效果如下：

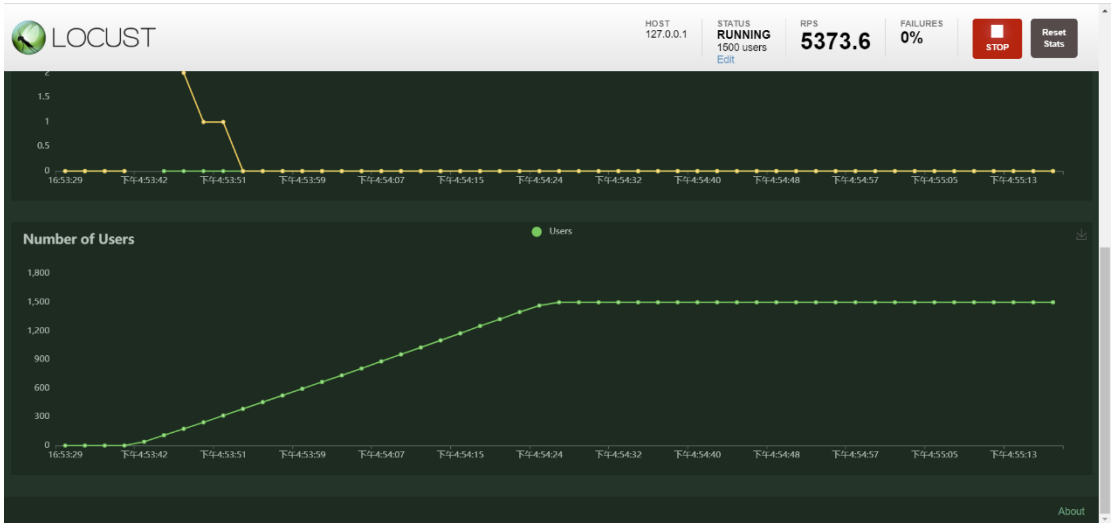


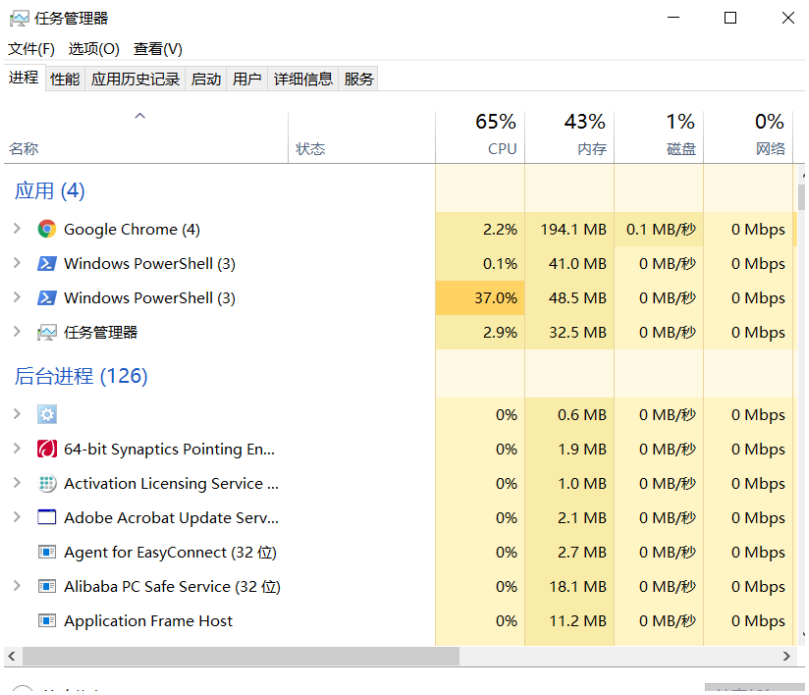
6.2.3 扩展功能 2：

目前我们虽然通过多线程实现了多人聊天室的基本功能，但缺乏高并发的优化。为实现此优化，我们使用多路复用 I/O 来进行改进。

我们使用 Python 中的 select 模块。具体来说，我们首先在 server 监听前开辟固定数量的工作线程，之后，主线程使用多路复用 I/O 接收连接，并将连接分发至工作线程，每个工作线程负责使用 I/O 复用与客户端进行通讯。

高并发优化效果如下：



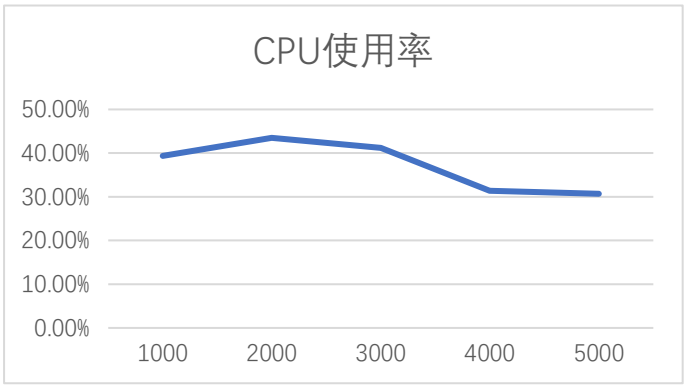


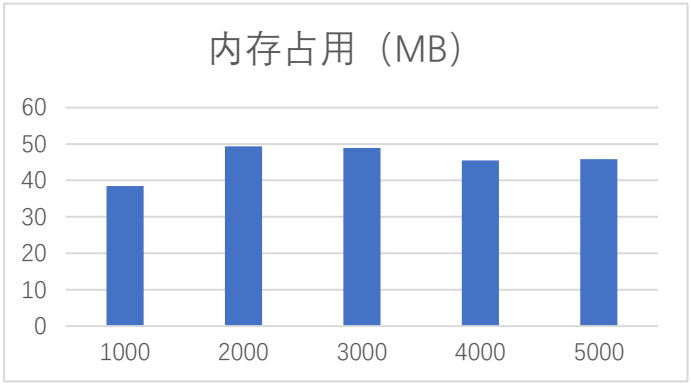
6.3 性能评价

6.3.1 图表统计

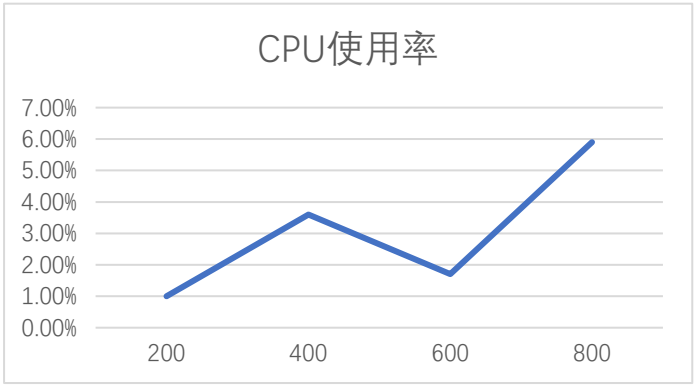
	A	B	C	D	E	F
1	多线程多路复用I/O	1000	2000	3000	4000	5000
2	CPU使用率	39.40%	43.50%	41.20%	31.40%	30.70%
3	内存占用（MB）	38.5	49.4	48.9	45.5	45.8
4	进程数	2	3	3	3	3
5	线程数	4	4	4	4	4
6						
7						
8	多线程	200	400	600	800	
9	CPU使用率	1.00%	3.60%	1.70%	5.90%	
10	内存占用（MB）	1073	1874.9	4041.6	4255.7	
11	进程数	4	4	4	4	
12	线程数	165	270	582	613	

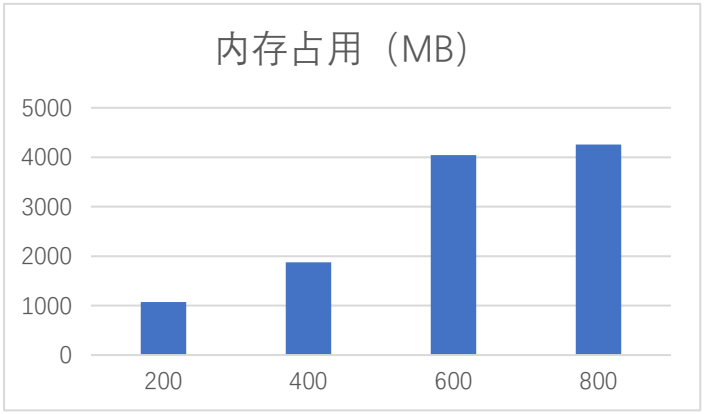
6.3.2 多线程+多路复用 I/O





6.3.2 多进程





通过图表可以看出，基于多进程的网络聊天室内存占用十分巨大，仅 200 人就达到了 1GB 左右，而且均未达到预期的聊天室人数（如 200 人的聊天室仅有不到 170 人）。

相比于多进程的聊天室，多线程+多路复用 I/O 的聊天室内存占用很小，而且可以达到预期的聊天室人数，且聊天室人数可以达到五千人，甚至更多。

综上，多线程+多路复用 I/O 的聊天室效率更高，可以满足上千人同时在线的需求。

七、总结

基于赵老师和助教讲解的 socket 网络聊天室设计，我通过 Java 与 Python 两种编程语言编写了基于 socket 的网络聊天室的设计。通过这次设计我对 TCP socket 基本原理有了更加深入的理解也提升了自己对于 socket 编程的基本技能。此外，我也对网络应用层协议架构的基本原理也有了更多的认识，提升了对于应用层协议设计和实现的实践能力。

在设计的过程中，我也发现了自己的许多不足，我对理论知识的掌握还是相对有限，使得程序不够完善，也没有能够完成布置的所有任务。此外，就聊天室而言，程序还有许多问题亟待完善。比如服务器的图形界面以及文件传输等问题，都是需要尽自己所能尽快解决的。

回顾此次网络聊天室的编写，我从理论到实践，学到了很多，不仅巩固了以前学到的理论知识，而且学以致用，将其运用在实践中。从理论中进行实践并获得实质成果，才能真正掌握这门技术，提高自己独立思考的能力。未来我会不断提高自己的理论和实践能力，并对实验进行改进与完善，提升自己在计算机网络方面的理论知识和实践能力。