1. 概述

本文研究服务端架构

1. 应用场景

目前服务端采用UDP进行通信，同时也部分支持TCP方式。

1.对于UDP目前采用select单线程简单模型，优点是简单方便，避免线程切换，不用维护连接，对于简单请求响应良好。缺点是当处理复杂请求时，线程忙于计算，无法及时收取socket消息。好在我们的服务端逻辑处理比较简单。

客户端

客户端

客户端

服务端

线程

接收请求

应答请求

处理请求

2.为了解决以上问题，可采用select单线程收/发，多工作线程处理消息队列的模型

当客户端请求到来时，select线程收取消息，并放入消息队列，工作线程从消息队列取出消息，处理完发送反馈，或另用消息队列进行反馈。需要做好线程同步，对消息队列加锁。考虑使用无锁队列。

客户端

客户端

客户端

服务端

select+recv线程

接收请求

应答请求

加入请求消息队列

加入应答

消息队列

工作线程

取出

应答消息

取出

请求消息

send线程

3.以上采用UDP传输，无法忽视可能的丢包问题。可部分采用TCP方式。每建立一个连接则发起一个线程维护此连接。优点是简单明了，缺点是连接过多时线程数过多，导致线程间切换开销大。一种优化方法是采用select + 线程池

客户端1

客户端2

客户端3

服务端

工作线程1

接收请求

应答请求

创建线程

工作线程2

工作线程3

accept主线程

接受连接

线程与客户端一一对应

4.为避免以上过多线程切换，一个线程用于accept，一个线程用于select，并分配任务。一个线程池用于获取任务并执行。优点是线程少，且便于移植。缺点是可监控的socket数目有限（1024），且轮询较费时。

客户端

客户端

客户端

服务端

accept线程

接受连接

应答请求

加入SOCKET

监视队列

加入任务队列

select线程

线程池

从队列取出任务

取出可读

SOCKET

线程

线程

接收请求

长连接时再放回

SOCKET监视队列

5.满足高并发要求的服务器，可以采用iocp（win）或epoll（linux）。以下为WINDOWS下的IOCP完成端口模型。不受socket数目限制，异步非轮询效率更高。

客户端

客户端

客户端

服务端

accept线程

接受连接

处理请求

投递

异步读请求

（消息头）。

并提供参数

（BUFF）

IO完成后唤醒线程池并传递参数（BUFF）

IOCP完成端口

取出投递的请求并进行IO

线程

线程

接收请求

应答请求

投递

异步读请求

（消息头

后跟数据）。

..

读完成

写完成

投递异步读请求

（消息头）。..

..

读完消息头

读完消息头

后跟数据

投递异步写请求

（消息头+后跟数据）。..

..

6.以上IOCP未实现IO与逻辑分离，可以再加入工作线程。IOCP线程只进行IO，并将结果入队列。由工作线程取队列，并将处理结果投递到IOCP

accept线程

2.投递读0字节

线程

线程

4．获取任务

TR1,TR2,WT

1．TR1任务加入列表

3．IO完成会

唤醒线程

TR1

校验线程

5．投递读任务

TR1,TR2

7．获取

TR1结果

..

6．TR1读取结果入队列

8．TR2任务加入列表

10．TR2读取结果入队列

11．获取

TR2结果

TR2

处理线程

9．投递读0字节

13．投递写0字节

12．TW任务加入列表

15．WT任务完成

16．TR1任务加入列表

17．投递读0字节

14．投递写任务

WT

状态顺序：1-2

3-4-5-3-6-7-8-9

3-4-5-3-10-11-12-13

3-4-14-3-15-16-17

3-4-5-3--6-7-8-9