**3Http Server 3.0 : Select模型**

老大说： “我们仔细合计合计， 对我来说，一个Socket连接就是一个所谓的文件描述符（File Descriptor ,简称 fd , 是个整数） , 这个fd 背后是一个简单的数据结构， 但是我们用了一个非常重量级的东西-- 进程 --来表示对它的读写操作， 有点浪费啊。”

我说： “要不咱们还切换回单进程模型？ 但是又会回到老路上去， 一个receive 的阻塞就什么事都干不了了”

“单进程也不是不可以， 但是我们要改变一下工作方式。”

“改成什么？” 我想不透老大在卖什么关子。

“你想想你阻塞的本质原因， 还不是因为人家浏览器还没有把数据发过来， 我自然也没法给你， 而你又迫不及待的想去读， 我只好把你阻塞。 在单进程情况下， 一阻塞，别的事儿都干不了。“

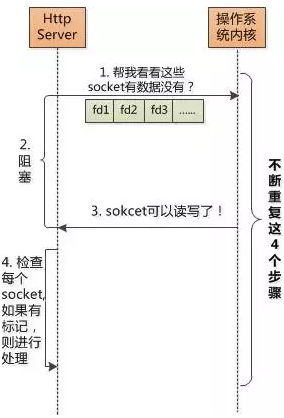
“对，就是这样”

“所以你接受了客户端连接以后， 不能那么着急的去读， 咱们这么办， 你的每个socket fd 都有编号， 你把这些编号告诉我， 就可以阻塞休息了 ”

我问道：“这不和以前一样吗？ 原来是调用receive 时阻塞， 现在还是阻塞”

“听我说完， 我会在后台检查这些编号的socket, 如果发现这些socket 可以读写， 我会把对应的socket 做个标记， 把你唤醒去处理这些socket 的数据， 你处理完了，再把你的那些socket fd 告诉我， 再次进入阻塞，如此循环往复。”

我有点明白了： “ 这是我们俩的一种通信方式， 我告诉你我要等待什么东西， 然后阻塞， 如果事件发生了， 你就把我唤醒， 让我做事情。”



“对， 关键点是你等我的通知， 我把你从阻塞状态唤醒后， 你一定要去遍历一遍所有的socket fd，看看谁有标记， 有标记的做相应处理。 我把这种方式叫做 select ”

我用select的方式改写了Http server， 抛弃了一个socket请求对于一个进程的模式， 现在我用一个进程就可以处理所有的socket了。

**4Http Server4.0 : epoll**

这种称为select的方式运行了一段时间， 效果还不错， 我只管把socket fd 告诉老大， 然后等着他通知我就行了。

有一次我无意中问老大：“我每次最多可以告诉你多少个socket fd？”

“1024个”

“那就是说我一个进程最多只能监控1024个socket了？ ”

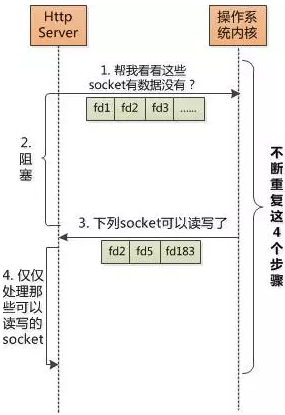
"是的， 你可以考虑多用几个进程啊"

这倒是一个办法， 不过"select"的方式用的多了， 我就发现了弊端， 最大的问题就是我从阻塞中恢复以后，需要遍历这1000多个socket fd， 看看有没有标志位需要处理。

实际的情况是， 很多socket 并不活跃， 在一段时间内浏览器并没有数据发过来， 这1000多个socket 可能只有那么几十个需要真正的处理， 但是我不得不查看所有的socket fd， 这挺烦人的。

难道老大不能把那些发生了变化的socket 告诉我吗？

我把这个想法给老大说了下， 他说：“嗯， 现在访问量越来越大， select 方式已经不满足要求， 我们需要与时俱进了， 我想了一个新的方式，叫做epoll”



“看到没有， 使用epoll和select 其实类似“ 老大接着说 ： ” 不同的地方是第3步和第4步， 我只会告诉你那些可以读写的socket , 你呢只需要处理这些'ready' 的socket 就可以了“

“看来老大想的很周全， 这种方式对我来说就简单的多了。 ”

我用epoll 把Http Server 再次升级， 由于不需要遍历全部集合， 只需要处理哪些有变化的， 活跃的socket 文件描述符， 系统的处理能力有了飞跃的提升。

我的Http Server 受到了广泛的欢迎， 全世界有无数人在使用， 最后死党数据库小明也知道了， 他问我：“ 大家都说你能轻松的支持好几万的并发连接， 真是这样吗？ ”

我谦虚的说： “过奖， 其实还得做系统的优化啦。”