SOCKET API和TCP STATE的对应关系__三次握手(listen,accept,connect)__四次挥手close及TCP延迟确认(调用一次setsockopt函数，设置TCP_QUICKACK)__长连接API小心“窜包”问题

2011-09-13 21:08:15 | 分类： Linux网络编程

订阅 | 字号 | 举报

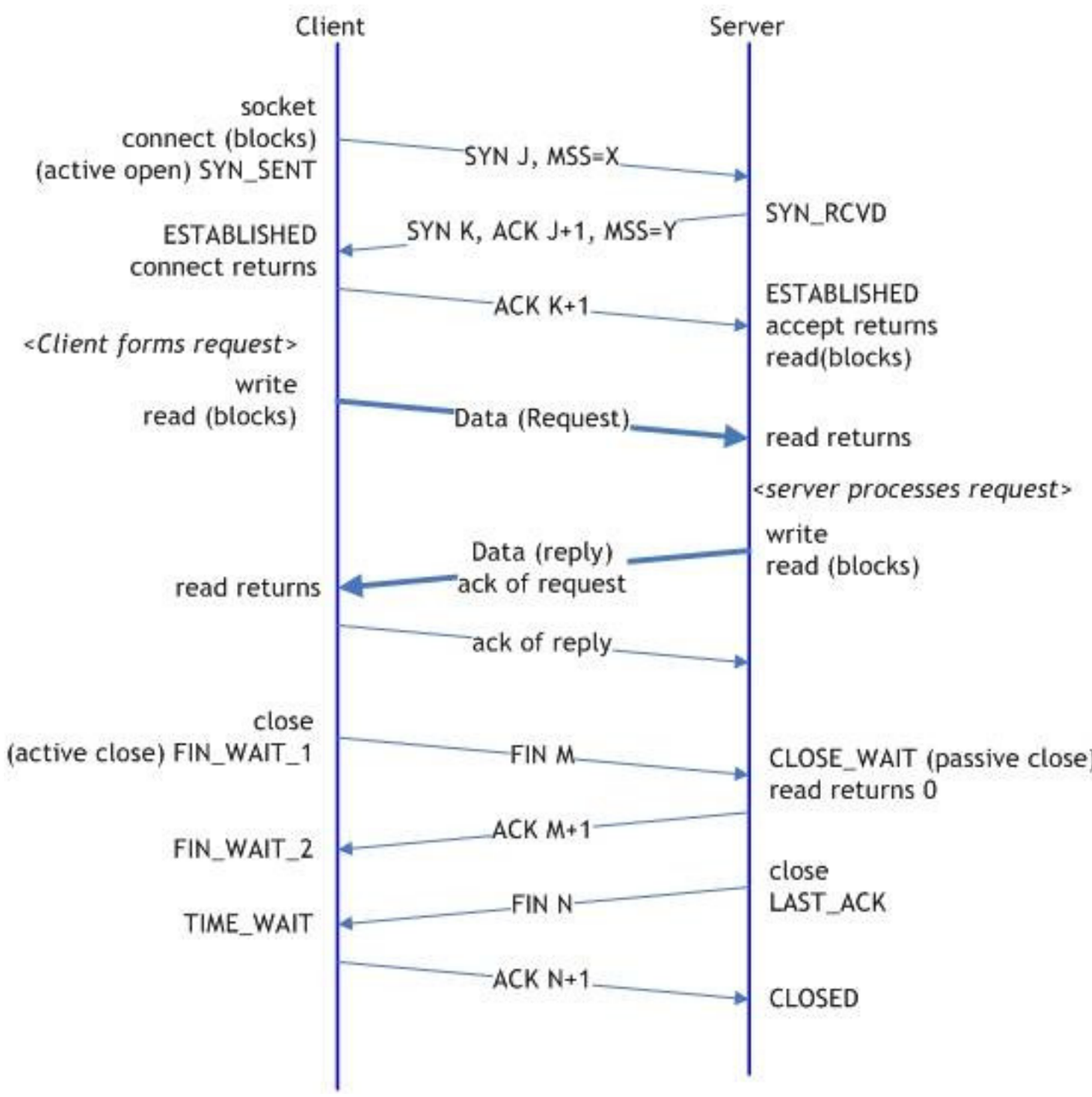
我的照片书 | 下载LOFTER

在我们学习网络基础时，传输层的协议有TCP和UDP；

在Linux网络编程中，我们使用socket API，实现网络通信。

那么:

socket API 和 TCP 协议中各个状态是如何对应的呢？我们可以通过下图来看:



无影

加博友 关注他

LOFTER - 最美图片社区

在socket系统调用中，如何完成三次握手和四次挥手:

SOCK_DGRAM,即UDP中的connect操作知识在内核中注册对方机器的IP和PORT信息，并没有建立链接的过程，即没有发包，close也不发包）。

而SOCK_STREAM对应如下:

connect会完成TCP的三次握手，客户端调用connect后，由内核中的TCP协议完成TCP的三次握手;

close操作会完成四次挥手。

三次握手对应的Berkeley Socket API:

从图中，可以看出和连接建立相关的API有：connect, listen, accept　3个，connect用在客户端，另外2个用在服务端。

对于TCP/IP protocol stack来说，TCP层的tcp_in&tcp_out也参与这个过程。我们这里只讨论这3个应用层的API干了什么事情。

（1） connect

发送了一个SYN，收到Server的SYN+ACK后，代表连接完成。发送最后一个ACK是protocol stack,tcp_out完成的。

（2）listen

在server这端，准备了一个未完成的连接队列，保存只收到SYN_C的socket结构;

还准备了已完成的连接队列，即保存了收到了最后一个ACK的socket结构。

（3）accept

应用进程调用accept的时候，就是去检查上面说的已完成的连接队列，如果队列里有连接，就返回这个连接;

如果没有，即空的，blocking方试调用，就睡眠等待;

nonblocking方式调用，就直接返回，一般一"EWOULDBLOCK“ errno告诉调用者，连接队列是空的。

注意:

在上面的socket API和TCP STATE的对应关系中，TCP协议中，客户端收到Server响应时，可能会有会延迟确认。

即客户端收到数据后，会阻塞给Server端确认。

可以在每次收到数据后:

调用setsockopt(fd, IPPROTO_TCP, TCP_QUICKACK, (int[]){1}, sizeof(int))；快速给Server端确认。

我们如何判断有一个建立链接请求或一个关闭链接请求:

建立链接请求:

1、connect将完成三次握手，accept所监听的fd上，产生读事件，表示有新的链接请求;

关闭链接请求:

网易考拉推荐

网易新闻

1、close将完成四次挥手，如果有一方关闭sockfd，对方将感知到有读事件，

如果read读取数据时，返回0，即读取到0个数据，表示有断开链接请求。(在操作系统中已经这么定义)

关闭链接过程中的TCP状态和SOCKET处理，及可能出现的问题:

1. TIME_WAIT

TIME_WAIT 是主动关闭 TCP 连接的那一方出现的状态，系统会在 TIME_WAIT 状态下等待 2MSL（maximum segment lifetime ）后才能释放连接（端口）。通常约合 4 分钟以内。

TIME_WAIT 状态等待 2MSL 的意义:

1、确保连接可靠地关闭； 即防止最后一个ACK丢失。

2、避免产生套接字混淆（同一个端口对应多个套接字）。

为什么说可以用来避免套接字混淆呢?

一方close发送了关闭链接请求，对方的应答迟迟到不了(例如网络原因)，导致TIME_WAIT超时，此时这个端口又可用了，我们在这个端口上又建立了另外一个socket链接。 如果此时对方的应答到了，怎么处理呢？其实这个在TCP层已经处理了，由于有TCP序列号，所以内核TCP层，就会将包丢掉，并给对方发包，让对方将sockfd关闭。所以应用层是没有关系的。即我们用socket API编写程序，就不用处理。

注意::

TIME_WAIT是指**操作系统的定时器会等2MSL，而主动关闭sockfd的一方，并不会阻塞**。（即应用程序在close时，并不会阻塞）。

当主动方关闭sockfd后，对方可能不知道这个事件。那么当对方(被动方)写数据，即send时，将会产生错误，即errno为: ECONNRESET。

服务器产生大量 TIME_WAIT 的原因：(一般我们不这样开发Server，但是web服务器等这种多客户端的Server，是需要在完成一次请求后，主动关闭连接的，否则可能因为句柄不够用，而造成无法提供服务。)

服务器存在大量的主动关闭操作，需关注程序何时会执行主动关闭（如批量清理长期空闲的套接字等操作）。

一般我们自己写的服务器进行主动断开连接的不多，除非做了空闲超时之类的管理。(TCP短链接是指，客户端发送请求给服务器，客户端收到服务器端的响应后，关闭链接)。

2. CLOSE_WAIT

CLOSE_WAIT 是被动关闭 TCP 连接时产生的，

如果**收到另一端关闭连接的请求后**，本地(Server端)不关闭相应套接字就会导致本地套接字进入这一状态。

(如果对方关闭了，没有收到关闭链接请求，就是下面的不正常情况)

按TCP状态机，我方收到FIN，则由TCP实现发送ACK，因此进入CLOSE_WAIT状态。但如果我方不执行close()，就不能由CLOSE_WAIT迁移到LAST_ACK，则系统中会存在很多CLOSE_WAIT状态的连接。

如果存在大量的 CLOSE_WAIT，则说明客户端并发量大，且服务器未能正常感知客

· 白血病人"配捐"被骗 医院只管仍住…
· 女学生挥霍银行2340万被判无罪?…
· 江西5岁留守儿童教室死亡 学校老…
· 民间"梵高"!老农家徒四壁 画出5幅…
· 男子玩手机"斗牛"游戏连赢1万上…
· 87版红楼梦演员为女德班站台授…
· 女子停车被两男子拽开车门搂脖抱…
· 大爷新买"顶配车"烧成铁架 卖车老…
下载网易新闻客户端 >

户端的退出，也并未及时 close 这些套接字。(如果不及时处理，将会出现**没有可用的socket描述符的问题**，原因是sockfd耗尽)。

正常情况下::

一方关闭sockfd，另外一方将会有读事件产生， 当recv数据时，如果返回值为0，表示对端已经关闭。此时我们应该调用close，将对应的sockfd也关闭掉。

不正常情况下::

一方关闭sockfd，另外一方并不知道，(比如在close时，自己断网了，对方就收不到发送的数据包)。此时，如果另外一方在对应的sockfd上写send或读recv数据。

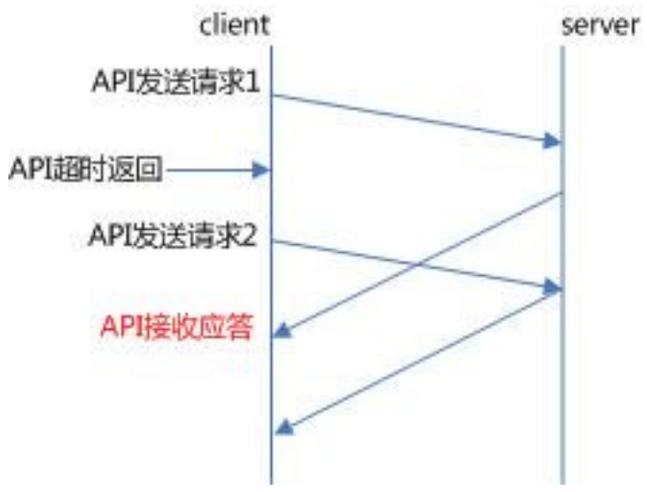
recv时，将会返回0，表示链接已经断开。

send时， 将会产生错误，errno为ECONNRESET。

长连接API小心“串包”问题:

有时候，我们以API的方式为客户提供服务，如果此时你提供的API采用TCP长连接，而且还使用了TCP接收超时机制（API一般都会提供设置超时的接口，例如通过setsockopt设置SO_RCVTIMEO或这select），那你可能需要小心下面这种情况（这里姑且称之为“窜包”,应用程序没有将应答包与请求包正确对应起来）：

如果某一笔以TCP接收的请求超时（例如设置为3秒）返回客户，此时客户继续使用该链接发送第二个请求，此时后者就有可能收到前一笔请求的应答（前一笔的应答在3秒后才到达），倘若错误的将此应答当做后者的应答处理，那就可能会导致严重的问题。如果网络不稳定，或者后台处理较慢，超时严重，其中一笔请求应答窜包了，很可能导致后续多个请求应答窜包。例如网上常见的抽奖活动，第一个用户中了一个iPad，而第二个用户在后台中仅为一个虚拟物品，若此时出现窜包，那第二个用户也会被提示中了iPad。



这个问题，初看起来最简单的解决办法就是：一旦发现有请求超时，就断开并重新建立连接，但这种方案理论上是不严谨的，考虑下面这种情况：

1、应答超时的原因是因为应答包在网络中游荡（例如某个路由器崩溃等原因，这类在网络中游荡的包，俗称迷途的分组）；

2、API在检测到超时后，断开并重新建立的连接的IP和Port与原有连接相同（新连接为被断开连接的化身）；

3、在新连接建立后，立即发送了一个新的请求，但随后那个迷途的应答包又找到了回家的路，重新到达，此时新连接很有可能将这个不属于自己的包，当做第二个请求的应答（该包的TCP Sequence恰好是新连接期望的TCP Sequence，这种情况是可能的，但是基本不可能发生）。

注：正常情况下，TCP通过维持TIME_WAIT状态2MSL时间，以避免因化身可能带来的问题。但是在实际应用中，我们可以通过调整系统参数，或者利用SO_LINGER选项使得close一个连接时，直接到CLOSE状态，跳过TIME_WAIT状态，又或者利用了端口重用，这样就可能会出现化身。**在实际应用中，上面这种情况基本不会发生，但是从理**

论上来说，是可能的。

再仔细分析，就会发现这个问题表面上看是因为“窜包”导致，但本质原因是程序在应用层没有对协议包效验。例如另外一种情况：A、B两个客户端与Server端同时建立了两个连接，如果此时Server端有BUG，错将A的应答，发到B连接上，此时如果没有效验，那同样会出现A请求收到B应答的情况。**所以这个问题解决之道就是：在应用层使用类似序列号这类验证机制，确保请求与应答的一一对应。**

参考文章：

http://pananq.com/index.php/page/9/

阅读(14330) | 评论(1)

转载 推荐

C/C++调正则表达式__1、regcomp编译正则… socket链接的关闭close和shutdown的区别_…

评论

登录后你可以发表评论，请先登录。登录>>

小寒 2014-09-20 16:41

谢谢楼主

我的照片书 - 博客风格 - 手机博客 - 下载LOFTER APP - 订阅此博客

网易公司版权所有 ©1997-2018