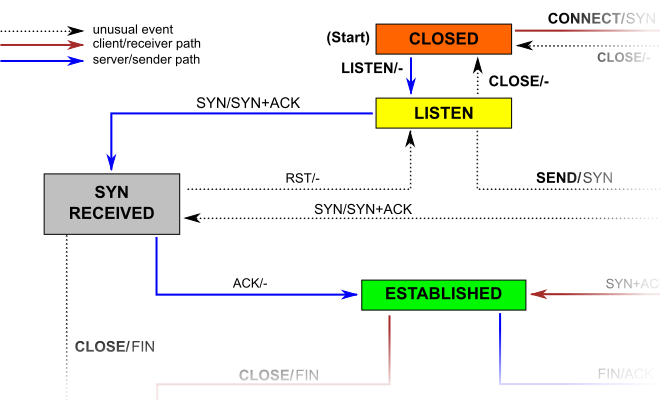
<http://blog.chinaunix.net/uid-28541347-id-5748886.html>

如果你熟悉linux网络编程，那么对listen系统调用一定不会陌生，listen系统调用使一个socket变为一个passive socket。那什么是passive socket呢？就是一个可以用来接收连接，可以在其上调用accept调用的一个socket。

listen系统调用的原型如下：

**int listen(int sockfd, int backlog);**

其中sockfd即要操作的socket，那么第二个参数backlog是什么含义呢？在理解这个参数的含义之前我们先看下TCP建立连接的大概过程，如下图所示：



     这个图是服务端的TCP状态迁移图的一部分。在服务端调用listen调用后，socket进入了LISTEN状态；当收到客户端的syn包（三次握手的第一次）后，服务端回复syn+ack（三次握手的第二次握手），此时socket变为了SYN RECEIVED状态；而当收到来自客户端的第三次握手ack报文后，**socket变为ESTABLISHED状态，这个时候也就是accpet调用返回的时候**。

     所以一个socket在被调用listen后，到accept前是要经历一个中间状态SYN RECEIVED的。如何处理这个中间状态的socket呢？协议栈可以有两个选择：

**（１）协议栈使用一个队列**，这个队列的大小由listen系统调用的backlog参数决定。当一个syn包到达后，服务端协议栈回复syn+ack，然后将这个socket加入这个队列。当客户端第三次握手的ack包到达后，再将这个socket的状态改为ESTABLISHED状态。这也就意味着这个队列可以可以容纳两种不同状态的socket：SYN RECEIVED和 ESTABLISHED，而只有后者可以被accept调用返回。

**（２）协议栈使用两个队列：一个存放未完成连接的队列，一个存放已完成连接的队列**。SYN RECEIVED状态的socket被添加到未完成队列中，当状态变为ESTABLISHED时就将其转移到已完成队列中。这种情况下accept系统调用就仅仅可以实现为在已完成队列中取出socket就可以了，而不用关心其状态。**而此时listen系统调用的backlog参数用来决定已完成队列的大小**。

由于历史原因，BSD系统的TCP协议栈采用了第一种策略。这种选择意味着当队列中的连接数（socket）达到backlog个后，系统收到syn将不再回复syn+ack。这种情况下协议栈通常仅仅是将syn包丢掉，而不是回复rst报文，从而让客户端可以重试。这个在 W. Richard Stevens的UNP中有所介绍。

但是需要注意的是Stevens 在UNP是采用两个队列描述的BSD的实现，但是其行为却如同一个队列，因为两个队列的总大小是用backlog参数决定，当然并不是严格的等于backlog，而是有一定关系，这里不做过多讨论，我们的重点是linux的实现。

Linux中的实现和BSD系统不同，正如其listen函数的man手册中所描述的那样：

**The behavior of the backlog argument on TCP sockets changed with Linux 2.2. Now it specifies the queue length forcompletely established sockets waiting to be accepted, instead of the number of incomplete connection requests. The maximum length of the queue for incomplete sockets can be set using /proc/sys/net/ipv4/tcp\_max\_syn\_backlog.**

**这就意味着linux是采用第二种策略实现的：一个未完成队列（其大小由一个系统范围内的参数/proc/sys/net/ipv4/tcp\_max\_syn\_backlog决定），一个已完成队列（其他小有backlog参数决定）**。

**但是这里又有一个问题：当已完成队列满了，又有第三次握手的ack报文到达会怎么样呢**？我们看下内核协议栈代码是如何处理这种情况的， 相关处理函数是/ipv4/tcp\_minisocks.c中的tcp\_check\_reqnet，具体相关代码如下：

**点击(此处)折叠或打开**

1. child = inet\_csk(sk)->icsk\_af\_ops->syn\_recv\_sock(sk, skb, req, NULL);
2. if (child == NULL)
3. goto listen\_overflow;

   对应IPV4，第一行实际调用的函数是net/ipv4/tcp\_ipv4.c中的tcp\_v4\_syn\_recv\_sock函数，相关代码如下：

**点击(此处)折叠或打开**

1. if (sk\_acceptq\_is\_full(sk))
2. goto exit\_overflow;

我们看到这里会检测已完成队列（accept queue），如果队列已经满了，就跳转到 exit\_overflow，而exit\_overflow标签后主要就是执行一些清理工作，更新/proc/net/netstat下的ListenOverflows和ListenDrops统计，然后返回NULL。从第一段代码可以知道返回NULL会使代码跳转到listen\_overflow。

**点击(此处)折叠或打开**

1. listen\_overflow:
2. if (!sysctl\_tcp\_abort\_on\_overflow) {
3. inet\_rsk(req)->acked = 1;
4. return NULL;
5. }

从listen\_overflow后的代码可以看出，**除非系统参数 sysctl\_tcp\_abort\_on\_overflow（即/proc/sys/net/ipv4/tcp\_abort\_on\_overflow）被置为1，这种情况下会回复RST报文；否则系统默认的动作是：什么也不做**。

所以结论就是，当已完成队列已经满了的时候，如果再收到TCP的第三次握手的ack包，linux协议栈的默认处理就是忽略这个包。这种做法看上去有些奇怪，但是如果是想TCP状态转换的同学应该知道SYN RECEIVED状态的连接是和一个定时器关联的，如果定时器时间到了，还处于这个状态就好触发服务端的syn+ack报文重传，重传次数有系统参数/proc/sys/net/ipv4/tcp\_synack\_retries决定。

下面一组抓包结果显示的就是当服务端的“已完成队列”满的时候，客户端尝试连接的情景。

  0.000  127.0.0.1 -> 127.0.0.1  TCP 74 53302 > 9999 [SYN] Seq=0 Len=0

  0.000  127.0.0.1 -> 127.0.0.1  TCP 74 9999 > 53302 [SYN, ACK] Seq=0 Ack=1 Len=0

  0.000  127.0.0.1 -> 127.0.0.1  TCP 66 53302 > 9999 [ACK] Seq=1 Ack=1 Len=0

  0.000  127.0.0.1 -> 127.0.0.1  TCP 71 53302 > 9999 [PSH, ACK] Seq=1 Ack=1 Len=5

  0.207  127.0.0.1 -> 127.0.0.1  TCP 71 [TCP Retransmission] 53302 > 9999 [PSH, ACK] Seq=1 Ack=1 Len=5

  0.623  127.0.0.1 -> 127.0.0.1  TCP 71 [TCP Retransmission] 53302 > 9999 [PSH, ACK] Seq=1 Ack=1 Len=5

  1.199  127.0.0.1 -> 127.0.0.1  TCP 74 9999 > 53302 [SYN, ACK] Seq=0 Ack=1 Len=0

  1.199  127.0.0.1 -> 127.0.0.1  TCP 66 [TCP Dup ACK 6#1] 53302 > 9999 [ACK] Seq=6 Ack=1 Len=0

  1.455  127.0.0.1 -> 127.0.0.1  TCP 71 [TCP Retransmission] 53302 > 9999 [PSH, ACK] Seq=1 Ack=1 Len=5

  3.123  127.0.0.1 -> 127.0.0.1  TCP 71 [TCP Retransmission] 53302 > 9999 [PSH, ACK] Seq=1 Ack=1 Len=5

  3.399  127.0.0.1 -> 127.0.0.1  TCP 74 9999 > 53302 [SYN, ACK] Seq=0 Ack=1 Len=0

  3.399  127.0.0.1 -> 127.0.0.1  TCP 66 [TCP Dup ACK 10#1] 53302 > 9999 [ACK] Seq=6 Ack=1 Len=0

  6.459  127.0.0.1 -> 127.0.0.1  TCP 71 [TCP Retransmission] 53302 > 9999 [PSH, ACK] Seq=1 Ack=1 Len=5

  7.599  127.0.0.1 -> 127.0.0.1  TCP 74 9999 > 53302 [SYN, ACK] Seq=0 Ack=1 Len=0

  7.599  127.0.0.1 -> 127.0.0.1  TCP 66 [TCP Dup ACK 13#1] 53302 > 9999 [ACK] Seq=6 Ack=1 Len=0

 13.131  127.0.0.1 -> 127.0.0.1  TCP 71 [TCP Retransmission] 53302 > 9999 [PSH, ACK] Seq=1 Ack=1 Len=5

 15.599  127.0.0.1 -> 127.0.0.1  TCP 74 9999 > 53302 [SYN, ACK] Seq=0 Ack=1 Len=0

 15.599  127.0.0.1 -> 127.0.0.1  TCP 66 [TCP Dup ACK 16#1] 53302 > 9999 [ACK] Seq=6 Ack=1 Len=0

 26.491  127.0.0.1 -> 127.0.0.1  TCP 71 [TCP Retransmission] 53302 > 9999 [PSH, ACK] Seq=1 Ack=1 Len=5

 31.599  127.0.0.1 -> 127.0.0.1  TCP 74 9999 > 53302 [SYN, ACK] Seq=0 Ack=1 Len=0

 31.599  127.0.0.1 -> 127.0.0.1  TCP 66 [TCP Dup ACK 19#1] 53302 > 9999 [ACK] Seq=6 Ack=1 Len=0

 53.179  127.0.0.1 -> 127.0.0.1  TCP 71 [TCP Retransmission] 53302 > 9999 [PSH, ACK] Seq=1 Ack=1 Len=5

106.491  127.0.0.1 -> 127.0.0.1  TCP 71 [TCP Retransmission] 53302 > 9999 [PSH, ACK] Seq=1 Ack=1 Len=5

106.491  127.0.0.1 -> 127.0.0.1  TCP 54 9999 > 53302 [RST] Seq=1 Len=0

当TCP客户端收到多个syn+ack报文后，就会假设发送的第三次握手的ack报文丢失，所以就会重传ACK（抓包结果中的Dup ACK）。此时如果服务端通过accept调用从已完成队列中取出一个连接后，且服务端重传syn+ack尚未达到系统设置的最大次数时，服务端就会最终在收到重传的ack报文后，将socket状态由SYN RECEIVED改为ESTABLISHED，并转移至已完成队列；否则客户端最终将收到一个RST报文。

    从上面的抓包分析我们还能看到另外一个现象：从客户端的角度看，当收到来自服务端的syn+ack报文后，客户端的连接状态将变为ESTABLISHED。这个时候如果客户端要发送数据，那么也将会出现重传现象。当然由于TCP的慢开始算法，这个阶段发送的字节数会受到一定限制。

    另一方面，如果服务端由于某种原因一直不能减少“已完成队列”中的连接数，那么从客户端看这个连接是ESTABLISHED，而对于服务端相当于CLOSED，这也是“半打开连接”的一种情况。

最后还有一个问题：**如果“已完成队列”满了，系统再收到syn包（第一次握手）会怎么处理呢？**还是会直接放入“未完成队列”中吗？我们还是看下相关代码，位于 net/ipv4/tcp\_ipv4.c中的tcp\_v4\_conn\_request函数 (这个函数会在收到syn包时调用)，相关代码如下：

**点击(此处)折叠或打开**

1. /\* Accept backlog is full. If we have already queued enough
2. \* of warm entries in syn queue, drop request. It is better than
3. \* clogging syn queue with openreqs with exponentially increasing
4. \* timeout.
5. \*/
6. if (sk\_acceptq\_is\_full(sk) && inet\_csk\_reqsk\_queue\_young(sk) > 1) {
7. NET\_INC\_STATS\_BH(sock\_net(sk), LINUX\_MIB\_LISTENOVERFLOWS);
8. goto drop;
9. }

    这就意味着，如果“已完成队列”已经满了，内核会对syn包的接收强加一个速率的限制。如果syn包太多，就会有一部分被丢掉。

**总结**

1. socket listen backlog 指定的其实就是 accept 队列，也就是 ESTABLISHED 状态的连接，这个数值不能超过 /proc/sys/net/core/somaxconn；syn 队列里面放的是未建立的连接，数值由内核参数 /proc/sys/net/ipv4/tcp\_max\_syn\_backlog 定义，应用代码没法修改。

2. 如果 accept 队列满，client 发来 ack，连接从 syn 队列移到 accept 队列的时候会发生什么呢？

1). 如果 /proc/sys/net/ipv4/tcp\_abort\_on\_overflow 为1，会发送 RST；如果为0，则「什么都不做」，也就是「忽略」。

2). 但是，即使被忽略，对于 SYN RECEIVED 状态， 会有重试，重试次数定义在 /proc/sys/net/ipv4/tcp\_synack\_retries(重试时间有个算法)。

3). client 在收到 server 发来的重试 synack 之后，它认为之前发给 server 的 ack 丢失，会重发，此时如果 server 的 accept 队列有「空位」，会把连接移到 accpet 队列，并把 SYN RECEIVED 改成 ESTABLISHED。

4). 从另一个角度看， 即使 client 发的 ack 被忽略，因为 client 已经收到了 synack，client 认为连接已经建立，它可能会直接发送数据(ack 和 数据一起发送)，这部分数据也会被忽略，会重传，幸好有「慢」启动机制保证重传的数据不会太多。

5). 如果 client 先等待 server 发来的数据，在 client 端连接是 ESTABLISHED，server 认为连接是 CLOSED，这会造成「半连接」。

6). 事实上，如果 accept 队列满了，内核会限制 syn 包的进入速度，如果太快，有些包会被丢弃。