# 连接建立的过程

抓包：

```
34687→80 [SYN] Seq=0 Win=65535 Len=0 MSS=1460 WS=4 SACK_PERM=1
80→34687 [SYN, ACK] Seq=0 Ack=1 Win=14600 Len=0 MSS=1452 SACK_PERM=1 WS=128
34687→80 [ACK] Seq=1 Ack=1 Win=256960 Len=0
```

# 被动建立连接

对服务端而言，连接建立的过程从接收到客户端的 SYN 开始，到接收到客户端对 SYN/ACK 的确认结束。在这个过程中，服务端需要把中间状态保存起来，直到应用层调用 accept 为止。

在面向连接的套接字 inet_connection_sock 结构体中有一个域 icsk_accept_queue 中，保存着正处于连接过程中和连接已完成但还未被 accept 取走的 request_sock。
icsk_accept_queue 是 request_sock_queue 类型的，request_sock_queue 的结构如下：

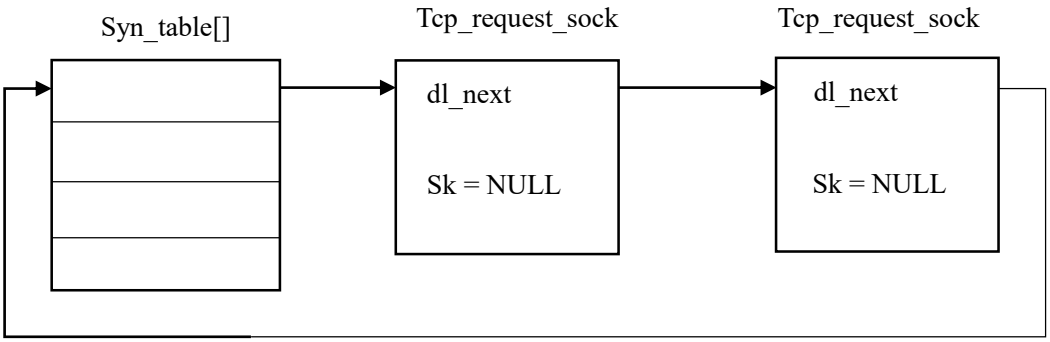
```
struct request_sock_queue {
	struct request_sock	*rskq_accept_head;
	struct request_sock	*rskq_accept_tail;
	rwlock_t		syn_wait_lock;
	u8			rskq_defer_accept;
	/* 3 bytes hole, try to pack */
	struct listen_sock	*listen_opt;
};
```

其中 rskq_accept_head 和 rskq_accept_tail 构成了一个由已完成连接的 request_sock 组成的 FIFO 队列。listen_opt 中的 syn_table[0]指向由还在连接中的 request_sock 构成的队列。
Listen_sock 的结构如下：

```
struct listen_sock {
	u8			max_qlen_log;
	/* 3 bytes hole, try to use */
	int			qlen; //有多少个连接请求
	int			qlen_young; //从未重传过 SYN/ACK 的连接请求
	int			clock_hand;
	u32			hash_rnd;
	u32			nr_table_entries;
	struct request_sock	*syn_table[0];
};
```

在 socket 系统调用创建 tcp_sock 时，rskq_accept_head， rskq_accept_tail 和 listen_opt 全都为 NULL。当调用 listen 系统调用后，TCP 进入 LISTEN 状态，同时为保存 SYN_RECV 状态期间的 request_sock 分配空间，其中 syn_table 哈希表的大小由 listen 系统调用的参数

backlog 控制。然后，tcp_sock 就可以接受连接请求了。如果收到客户端的 SYN 请求后，就会创建 tcp_request_sock，保存了双方的初始序列号，TCP 选项等。然后把 tcp_requst_sock 挂在 syn_table[0]的链表上。




Tcp_request_sock 的结构如下：

```
struct tcp_request_sock {
      struct inet_request_sock      req;
#ifdef CONFIG_TCP_MD5SIG
      /* Only used by TCP MD5 Signature so far. */
      struct tcp_request_sock_ops*af_specific;
#endif
      u32                  rcv_isn;   //接收方的初始序列号，就是发 SYN 包的序列号
      u32                  snt_isn;   //发送初始序列号，就是 SYN/ACK 里的序列号
};
```

当收到客户端对 SYN/ACK 的确认后，服务端才会真正建立一个 tcp_sock，并将 tcp_request_sock 中的 sk 指针指向这个 tcp_sock；然后将这个已完成连接的 tcp_request_sock 移动到 icsk_accept_queue 中，等 accept 系统调用取走。

Accpet 系统调用取走 rskq_accept_head 指向的那个 tcp_request_sock,和 file,socket 关联后(socket 结构体中的 sk 指针指向 tcp_sock)释放 tcp_request_sock。

Tcp 中所有收到的数据包都是由 tcp_v4_do_rcv 来处理的，根据连接的不同状态，调用不同的函数。如果是 TCP_ESTABLISHED 状态，调用 tcp_rcv_established()；其余状态则调用 tcp_rcv_state_process()处理。

# 主动建立连接

客户端调用 connect 系统调用主动发起连接请求，向服务器发 SYN 包，从 CLOSED 状态转为 SYN_SENT 状态。
Connect 系统调用：int connect(int sockfd, const struct sockaddr *addr,  socklen_t addrlen)
如果成功建立连接，返回 0，客户端 TCP 进入 ESTABLISHED 状态，否则返回错误码(负数)。
完成 connect 系统调用需要做这些事情：给 sock 绑定一个端口，发送 SYN 包，并启动重传

定时器，如果没有收到 SYN/ACK，需要重传。实现函数是 tcp_v4_connect，其中调用的函数 inet_hash_connect 来动态绑定端口；调用 tcp_connect 来发 SYN 包。

发完 SYN 包后，TCP 处于 SYN_SENT 状态，这个状态接受包的处理函数是 tcp_rcv_state_processs， 在其中调用函数 tcp_rcv_synsent_state_process。这时候我们希望收到的 SYN/ACK 包，如果是的话，转换为 ESTABLISHED 状态，进行一些初始化，另外还需要向对方回复 ACK 这时候就可以唤醒等待这个 sk 上的进程了，也就是 connect 系统调用可以返回 0 了。如果收到的是 SYN 包的话，说明是两边同时打开，把状态改为 SYN_RECV，并给对方回复 SYN/ACK。当再收到对方的 SYN/ACK 后，变为 ESTABLISHED 状态。