西门子SIEMENS模块6ES7416-2XP07-0AB0

产品名称	西门子SIEMENS模块6ES7416-2XP07-0AB0
公司名称	湖南西控自动化设备有限公司
价格	.00/件
规格参数	西门子:授权代理商 PLC:一级代理商 德国:售后保障服务
公司地址	中国（湖南）自由贸易试验区长沙片区开元东路1306号开阳智能制造产业园（一期）4#栋301
联系电话	17838383235 17838383235

产品详情

手把手教你手撸通讯协议(三)-开始手撕TCP

接下去我们还是通过开源的LwIP协议栈来好好了解以太网的真实工作方式，我将会在这一期的终期，给大家实现一个基于STM32的modbusTCP 主站的小demo。

节

初识TCP

TCP中文名叫传输控制协议，它为上层提供一种面向连接的、可靠的字节流服务；那TCP通过什么方法来提供可靠性？（1）先将应用数据分割成TCP认为适合发送的数据块；（2）当TCP发出一个段后，它启动一个定时器，等待目的端确认收到这个报文段，如果不能及时收到一个确认，将重发这个报文段；（3）当TCP收到发自 TCP连接另一端的数据，它将发送一个确认，这个确认不是立即发送，通常将推迟几分之一秒；（4）TCP将保持它首部和数据的检验和，如果收到段的检验和有差错，TCP将丢弃这个报文段并且不发送确认收，以使发送端超时并重发；（5）IP数据报的到达可能会失序，因此TCP 报文段的到达也可能会失序，如果必要，TCP将对收到的数据进行重新排序，将收到的数据以正确的顺序交给应用层；（6）IP数据报会发生重复，TCP的接收端必须丢弃重复的数据；（7）TCP还能提供流量控制。下图是 TCP首部结构，若不计任选字段，其大小为20 字节，与 IP 报首部大小相同。 源端口号和目的端口号，用于标识发送端和接收端的应用进程。这两个值加上IP首部中的源IP地址和目的IP地址就能唯一确定一个TCP连接。一个IP地址和一个端口号也称为一个插口(socket)。在TCP首部中有6个标志比特。它们中的多个可同时被设置为1。在这里简单介绍它们的用法，在以后用到时会详加讲解：URG
紧急指针（urgent pointer）有效标识；ACK确认序号有效标识；PSH接收方应该尽快将这个报文段交给应用层；RST重建连接；SYN同步序号，用来发起一个连接；FIN请求端完成发送任务。根据上图，在LwIP中是这样描述TCP报头：struct tcp_hdr {　PACK_STRUCT_FIELD(u16_t src); // 源端口

PACK_STRUCT_FIELD(u16_t dest); // 目的端口
PACK_STRUCT_FIELD(u32_t seqno); // 序号
PACK_STRUCT_FIELD(u32_t ackno); // 确认序号
PACK_STRUCT_FIELD(u16_t _hdrlen_rsvd_flags); // 首部长度+保留位+标志位
PACK_STRUCT_FIELD(u16_t wnd); // 窗口大小
PACK_STRUCT_FIELD(u16_t chksum); // 校验和
PACK_STRUCT_FIELD(u16_t urgp); // 紧急指针 }PACK_STRUCT_STRUCT;

第二节

TCP的断开和连接

众所周知的TCP有三次握手和四次挥手。 （图来自网上，挺常见的）

2.1

TCP连接建立

TCP要建立连接需要经历三次握手，那如何实现三次握手呢？（1）请求端发送一个SYN标志置1的TCP数据报，数据包中指明自己的端口号及将连接的服务器的端口号，同时通告自己的初始序号ISN。（2）当服务器接收到该数据包并解析后，也发回一个 SYN 报文段作为应答。（3）该回应报文包服务器自身选定的初始序号ISN，同时，将ACK置1，将确认序号设置为请求端的ISN加1以对客户的SYN报文段进行确认。（4）这里的ISN 也表示了服务器希望接收到的下一个字节的序号。由此可见，一个SYN将占用了一个序号。（5）当请求端接收到服务器的 SYN 应答包后，会再次产生一个握手包，这个包中，ACK标志置位，确认序号设置为服务器发送的ISN加1，以此来实现对服务器的SYN报文段的确认。

但这边会存在一个问题，如果两端同时发起连接，即同时发送个 SYN
数据包，这时这两端都处于主动打开状态，TCP中又是如何解决的？

2.2

TCP断开

为什么断开需要四次挥手？四次挥手做了啥？（1）请求端发起中断连接请求，也就是发送FIN报文，意思是说"我请求端没有数据要发给你了，但是如果你还有数据没有发送完成，则不必急着关闭Socket，可以继续发送数据"。（2）服务端接到FIN报文后，先发送ACK，"告诉请求端，你的请求我收到了，但是我还没准备好，请继续你等我的消息"。（3）请求端就进入FIN_WAIT状态，继续等待服务端的FIN报文。（4）当服务端确定数据已发送完成，则向请求端发送FIN报文，"告诉请求端，好了，我这边数据发完了，准备好关闭连接了"。（5）请求端收到FIN报文后，"就知道可以关闭连接了，但是他还是不相信网络，怕服务端不知道要关闭，所以发送ACK后进入TIME_WAIT状态，如果服务端没有收到ACK则可以重传。（6）服务端收到ACK后，"就知道可以断开连接了"。（7）请求端等待了2MSL后依然没有收到回复，则证明服务端已正常关闭，那好，请求端也可以关闭连接了。 这七个步骤可以很清晰的看到为啥要进行四次挥手，确认TCP断开。

第三节

TCP的状态转换

从上述三次握手建立连接到四次挥手断开连接过程中，其实可以总结到两张图：请求端状态切换图和服务端状态切换图。 这两个图结合起来就是TCP的状态转换图了，（图来自详解）。

第四节

TCP控制块解读

上面主要让大家对TCP这个协议有基本的认识，接下去我们要进行一些源码解读。struct tcp_pcb {
IP_PCB; //这是一个宏，描述了连接的 IP相关信息，包括双方IP地址，TTL等信息
struct tcp_pcb *next; //用于连接各个TCP控制块的链表指针
enum tcp_state state; //TCP 连接的状态，即为状态图中描述的那些状态
u8_t prio; //该控制块的优先级　void* callback_arg;//
u16_t local_port; //本地端口　u16_t remote_port; //远程端口
u8_t flags;// 附加状态信息，如连接是快速恢复、一个被延迟的 ACK 是否被发送等
#define TF_ACK_DELAY (u8_t)0x01U /延迟发送 ACK（推迟确认）
#define TF_ACK_NOW (u8_t)0x02U /立即发送 ACK
#define TF_INFR ((u8_t)0x04U) //连接处于快重传状态
#define TF_TIMESTAMP ((u8_t)0x08U) //连接的时间戳选项已使能
#define TF_FIN ((u8_t)0x20U) //应用程序已关闭该连接
#define TF_NODELAY ((u8_t)0x40U) //禁止 Nagle 算法
#define TF_NAGLEMEMERR ((u8_t)0x80U) //本地缓冲区溢出　// 接收相关字段
u32_t rcv_nxt; //期望接收的下一个字节，即它向发送端 ACK 的序号
u16_t rcv_wnd; //接收窗口　u16_t rcv_ann_wnd; //通告窗口大小
u32_t tmr; // 该字段记录该 PCB 被创建的时刻
u8_t polltmr, pollinterval; // 三个定时器，后续讲解
u16_t rtime; //重传定时，该值随时间增加，当大于 rto 的值时则重传发生
u16_t mss; //大数据段大小　//RTT 估计相关的参数
u32_t rttest; //估计得到的 500ms 滴答数　u32_t rtseq; //用于测试 RTT 的包的序号
s16_t sa, sv; //RTT 估计出的平均值及其时间差
u16_t rto; // 重发超时时间，利用前面的几个值计算出来
u8_t nrtx; // 重发的次数，该字段在数据包多次超时时被使用到，与设置 rto 的值相关
// 快速重传/恢复相关的参数　u32_t lastack; // 大的确认序号，该字段不解
u8_t dupacks; // 上面这个序号被重传的次数　// 阻塞控制相关参数
u16_t cwnd; //连接的当前阻塞窗口　u16_t ssthresh; // 慢速启动阈值
// 发送相关字段　u32_t snd_nxt, // 下一个将要发送的字节序号
snd_max, // 高的发送字节序号　snd_wnd, // 发送窗口
snd_wl1, snd_wl2, // 上次窗口更新时的数据序号和确认序号
snd_lbb; // 发送队列中后一个字节的序号　u16_t acked; //
u16_t snd_buf; // 可用的发送缓冲字节数　u8_t snd_queuelen; // 可用的发送包数
struct tcp_seg *unsent; // 发送的数据段队列
struct tcp_seg *unacked; // 发送了未收到确认的数据队列
struct tcp_seg *ooseq; // 接收到序列以外的数据包队列
#if LWIP_CALLBACK_API // 回调函数
err_t (* sent)(void *arg, struct tcp_pcb *pcb, u16_t space)? //当数据被成功发送后被调用
err_t (* recv)(void *arg, struct tcp_pcb *pcb, struct pbuf *p, err_t err)? //接收到数据后被调用
err_t (* connected)(void *arg, struct tcp_pcb *pcb, err_t err)? //连接建立后被调用
err_t (* poll)(void *arg, struct tcp_pcb *pcb)? //该函数被内核周期性调用
void (* errf)(void *arg, err_t err)? //连接发生错误时调用
#endif /* LWIP_CALLBACK_API */　　u32_t keep_idle;
#if LWIP_TCP_KEEPALIVE

u32_t keep_intvl; // 保活定时器，用于检测空闲连接的另一端是否崩溃
u32_t keep_cnt; //坚持定时器计数值　#endif /* LWIP_TCP_KEEPALIVE */
u32_t persist_cnt; // 这两个字段可以使窗口大小信息保持不断流动
u8_t persist_backoff;//坚持定时器探查报文发送的数目
u8_t keep_cnt_sent; //保活报文发送的次数