说白了，这些连接都是由系统来帮你管理的。你只需做的一件事就是：开启一个线程来accept进来的连接，剩下的工作交由系统来处理。而你，则需要提供给系统一个回调函数，发生新的网络事件的时候系统将执行这个函数：

void CALLBACK CompletionROUTINE(DWORD, DWORD, LPWSAOVERLAPPED, DWORD);

//相应的完成例程函数

CALLBACK CompletionROUTINE(DWORD dwError,  
        DWORD cbTransferred,  
        LPWSAOVERLAPPED lpOverlapped,  
        DWORD dwFlags)；

然后告诉系统用WorkerRoutine函数处理接收到的数据：

int WSARecv(

SOCKET s, // 当然是投递这个操作的[套接字](http://baike.haosou.com/doc/383276-405861.html" \t "_blank)，与Recv函数不同

LPWSABUF lpBuffers, // 这里需要一个由WSABUF结构构成的[数组](http://baike.haosou.com/doc/5545345-5760453.html" \t "_blank)

DWORD dwBufferCount, // [数组](http://baike.haosou.com/doc/5545345-5760453.html" \t "_blank)中WSABUF结构的数量

LPDWORD lpNumberOfBytesRecvd, // 如果接收操作立即完成，这里会返回[函数调用](http://baike.haosou.com/doc/2151920-2276849.html" \t "_blank)所接收到//的字节数

LPDWORD lpFlags,

**LPWSAOVERLAPPED lpOverlapped,**

**LPWSAOVERLAPPED\_COMPLETION\_ROUTINE lpCompletionRoutine** // 完成例程中将//用到的参数

);

然后......没有什么然后了，系统什么都给你做了！

typedef struct \_**OVERLAPPED** {

DWORD Internal; //预留给操作系统使用。它指定一个独立于系统的状态,当//GetOverlappedResult函数返回时没有设置扩展错误信息ERROR\_IO\_PENDING时有效。

DWORD InternalHigh; //预留给操作系统使用。它指定长度的数据转移,当//GetOverlappedResult函数返回TRUE时有效。

**DWORD Offset;**  //该文件的位置是从文件起始处的字节偏移量。调用进程设置这个成员之前//调用ReadFile或WriteFile函数。当读取或写入命名管道和通信设备时这个成员被忽略设为零。

**DWORD OffsetHigh;**  //指定文件传送的[字节](http://baike.haosou.com/doc/1114609-1179328.html" \t "_blank)[偏移量](http://baike.haosou.com/doc/6117681-6330824.html" \t "_blank)的高位字。当读取或写入命名管道和通信//设备时这个成员被忽略设为零。

**HANDLE hEvent;**  //在转移完成时处理一个事件设置为有信号状态。调用进程集这个成员在 //调用ReadFile、 WriteFile、TransactNamedPipe、 ConnectNamedPipe函数之前。

} OVERLAPPED