**Network实验4**

1.实验3.3.3 TCP客户机服务器编程(PPT中代码)

2.将上述实验服务器和客户机之间对话改为持久性对话

3.将上述实验服务器和客户机之间对话改为多线程对话

与线程有关的函数

1. 线程的创建与终止

线程创建API

1. HANDLE CreateThread(
2. LPSECURITY\_ATTRIBUTES lpThreadAttributes,
3. SIZE\_T dwStackSize,
4. LPTHREAD\_START\_ROUTINE lpStartAddress,
5. LPVOID lpParameter,
6. DWORD dwCreationFlags,
7. LPDWORD lpThreadId);

• lpThreadAttributes，描述线程安全的结构体，默认传NULL.

• dwStackSize，堆栈大小，默认1MB.

• lpStartAddress，线程函数入口地址。

• lpParameter，线程函数参数。

• dwCreationFlags，线程创建时的状态，0表示线程创建之后立即运行。CREATE\_SUSPENDED表示线程创建完挂起,直到调用ResumeThread才运行。

• lpThreadId，指向1个变量接受线程ID，可为NULL。

线程终止API

void **ExitThread**(DWORD dwExitCode);

函数将强制终止线程的运行，并导致损伤系统清除该线程所使用的所有操作系统资源。但是C++对象可能由于析构函数没有正常调用导致资源不能得到正确释放。附加的退出码，可以用**GetExitCodeThread()**函数可以获取。不建议使用此线程终止函数，因为可能导致资源没有正确的释放,一般都让线程正常退出。另外，即便要强制终止线程，也要使用\_endThreadEx(不使用\_endThread),因为它兼顾了多线程资源安全。

BOOL **TerminateThread**(HANDLE hThread, DWORD dwExitCode);

该函数也是强制退出线程的，只不过此函数是异步的，即它告诉系统去终止指定线程，但是不能保证函数返回时线程已经被终止了。因此调用者必须使用**WaitForSingleObject**函数来确定线程是否终止。因此此函数调用后终止的线程堆栈资源不会得到释放。一般不建议使用此函数。

2. 线程安全

对线程安全没有一个比较具体的说明，简单来说线程函数的操作是安全的。这里的操作对象主要为：变量、函数、类对象。

线程安全变量

这里的变量指非自定义类型的全局变量/静态变量，或者通过线程参数传入的变量。

•所有线程只读取该变量，那么该变量肯定线程安全的。

•有1个线程写操作该变量，其他线程读取该变量。这时就需要考虑volatile。当一段线程代码多次读取变量的值时，编译器默认会优化代码只第1次会从内存上读取值，其他时候直接是从寄存器上读取的。这样如果其他线程更新了变量的值，读取的线程可能依然是从寄存器上读取的。这个时候就需要告诉编译器该变量不要优化，永远是从内存上读取。效率可能低一点，但是保证线程中变量的安全更重要。

•有多个线程同时写操作该变量，那么就必须考虑临界区读写锁等方法。

线程安全函数

多线程出现之前就已经有C/C++运行时库，所以C/C++运行时库不一定是线程安全的。例如**GetLastError()**获取的就是一个全局的变量值，针对多线程可能就会出错。针对这个问题，MS提供了C/C++多线程运行时库，并且需要配合相应的多线程创建函数。

•**\_beginthreadex**

不建议使用\_beginthread，因为它是早期不成熟的函数，因为它创建完成线程之后立即结束了句柄，导致不能有效控制线程。C/C++运行时库函数**\_beginthreadex**是对操作系统函数CreateThread的封装，并且这里使用了线程局存储(TLS)来保证每个线程都有自已的单独的一些共用变量，例如像GetLastError()使用的变量。这样每个线程就能够保证所有的API函数都是线程安全的。

•**AfxBeginThread**

如果当前代码环境是基于MFC库的，那么多线程创建函数必须使用MFC库函数**AfxBeginThread**。这是因为MFC库是对C/C++运行库的再封装，同样会面临MFC库本身存在的一些线程不安全变量的操作。**AfxBeginThread**其实是对\_beginthreadex函数的再封装，在调用\_beginthreadex之前完成一些安全载入MFC DLL库的的操作。这样基于MFC的库函数的调用才是安全的。

线程安全类

除了C/C++运行时库、MFC库因为已经有处理线程安全外，其他第三方库，甚至包括STL都不是线程安全的。这些自定义的类库，都需要自已去考虑线程安全。 这里可以利用锁、同步及异步等内核对象来解决，当然也可以使用TLS来解决。

3. 线程的暂停与恢复

在线程内核对象的内部有一个值，用于指明线程的暂停计数。当调用CreateThread函数时，就创建了线程的内核对象，并且内核对象里的暂停计数被初始化为 1，这样操作系统就不会再分配时间片给线程。当创建的线程指定CREATE\_SUSPENED标志时，那么线程就处于暂停状，这个时候可以给线程进行一些优先级设置等其他初始化。当初始化完成之后，可以调用ResumeThread来恢复。单个线程可以暂时多次，如果暂停了3次，则需要ResumeThread恢复3次才能重新让线程获得时间片。

除了创建线程指定CREATE\_SUSPENED来暂停线程外，还可以调用SuspendThread来暂时线程。调用SuspendThread时，因为不知道当前线程正在做什么，如果是正在进行内存分配或者正在一个锁操作当中，可能导致其他线程锁死之类的。所以使用SuspendThread时一定要加强措施来避免可能出现的问题。

4.实现一个简单的web服务器(选做)