基于多线程的断点续传实现

周翔[1]，阮世颖[2]
（1.南昌大学 软件学院，江西 南昌 330047；2.南昌大学 医学院，江西 南昌 330006）

摘要：网络的不稳定常常造成数据传送的不稳定，为了获得可靠、高效的数据传输，断点续传技术应运而生。本文介绍了所开发的一种基于多线程的文件传输系统，具有断点续传的功能，详细给出了这个系统的设计方案和所用到的关键技术。

关键词：多线程；断点续传；文件传输；http
中图分类号：TP311 **文献标识码：**A **文章编号：**1009-3044(2007)22-41000-03

Implementation of Breakpoint Continuingly Based on Multi-threaded

ZHOU Xiang[1], RUAN Shi-ying[2]
(1.Software College, Nanchang University, Nanchang 330047, China; 2.Medical College, Nanchang University, Nanchang 330006, China)

Abstract: Instable network often create instability of data transmission, in order to obtain reliable and efficient data transmission, breakpoint Continuingly technology came into being. This paper presents a document transmission system based on multi-threaded, with breakpoint Continuingly functions, the system is given in detail of the design of programs and the key technologies.

Key words: multi-threaded; breakpoint Continuingly; document transmission; http

1 引言

在参与开发公安部金盾工程中一个子项目《机动车修理业治安信息管理系统》的过程中，其中一个模块需要实现大文件的断点续传功能，基于效率方面的考虑，使用了多线程技术。本文给出基于多线程技术的断点续传实现方法。

2 基本实现思想

多线程断点续传实现的基本思想就是在发送端（也称客户端）将要传输的文件分割为大小相当的多块，用多个线程，将这些块同时向目标服务器端发送；在服务器端的服务程序监听数据传输请求，每当接到新的请求，则创建一个新的线程，与客户端的发送线程对应，接收数据，记录数据传输进程。

图1是点对点文件断点续传第N块传输过程示意图。在传输发起端（客户端），将大型文件事先分割为大小相当的N块，同时创建N个传输线程，连接目标端服务器。当服务器端接收到每一个连接请求后，告知客户端可以传输文件。当客户端接收到可以传输文件的消息时，首先向服务器发送数据传输信息块（包括第几块、在块中的起始位置）请求，当服务器端接收到该请求后，向客户端发送数据传输信息，客户端然后传输数据传输信息块指定的数据给服务器端，服务器端更新数据传输信息块。

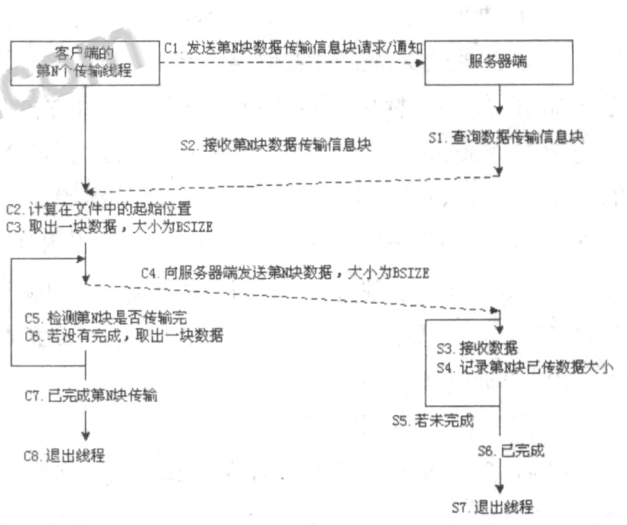

图1 断点续传过程示意图

3 断点续传的实现

在基本下载的代码上实现断点续传功能并不是很复杂，主要是检查本地的下载信息，确定已经下载的字节数。所以应该对打开输出文件的函数作适当修改。我们可以建立一个临时文件保存下载的信息，如已经下载的字节数等。先检查输出文件是否存在，如果存在，再得到其大小，并以此作为已经下载的部分。每次传输停止时，若未完成则将其写入临时文件中。如图2：

收稿日期：2007-09-12

作者简介：周翔（1982-），男，江西南昌市人，助教，硕士，研究方向：软件工程，J2EE，游戏开发；阮世颖（1982-），女，江西新建县人，助教，研究方向：统计学，统计软件。

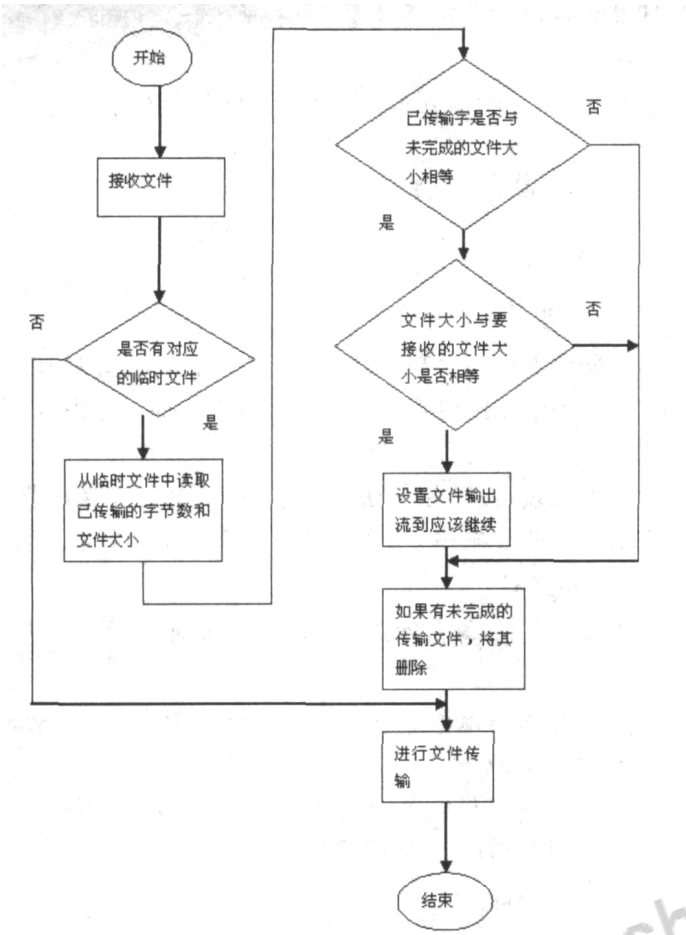

图 2 断点续传流程示意图

4 多线程下载

要实现多线程下载，最主要的问题是下载线程的创建和管理，已经下载完成的文件的各个部分的准确合并，同时，下载线程也要作必要的修改。

4.1 下载线程的修改

为了适应多线程程序，在下载线程加入如下成员变量：

```
int FIndex; // 在线程数组中的索引
DWORD dwStart; // 下载开始的位置
DWORD dwTotal; // 需要下载的字节数
DWORD FGetBytes; // 下载的总字节数
```

并加入如下属性值：

```
__property AnsiString URL = { read=FURL, write=FURL };
__property AnsiString OutFileName = { read=FOutFileName, write=FOutFileName};
__property bool Successed = { read=FSuccess};
__property int Index = { read=FIndex, write=FIndex};
__property DWORD StartPostion = { read=dwStart, write=dwStart};
__property DWORD GetBytes = { read=dwTotal, write=dwTotal};
__property TOnHttpCompelete OnComplete = { read=FOnComplete, write=FOnComplete };
```

4.2 建立多线程下载组件

先建立了以 TComponent 为基类、名为 THttpGetEx 的组件模块，并增加以下成员变量：

```
// 内部变量
THttpGetThread **HttpThreads; // 保存建立的线程
AnsiString *OutTmpFiles; // 保存结果文件各个部分的临时文件
bool *FSuccesss; // 保存各个线程的下载结果
// 以下是属性变量
int FHttpThreadCount; // 使用的线程个数
AnsiString FURL;
AnsiString FOutFileName;
```

因线程的运行具有不可逆性，而组件可能会连续地下载不同的文件，所以下载线程只能动态创建，使用后随即销毁。创建线程的模块如下，其中 GetSystemTemp 函数取得系统的临时文件，OnThreadComplete 是线程下载完成后的事件：

```
// 创建一个下载线程
THttpGetThread * THttpGetEx::CreateHttpThread(void)
{ …… // 初始化事件
HttpThread->OnComplete=OnThreadComplete; // 线程下载完成事件
return HttpThread;
}
// 创建下载线程数组
void THttpGetEx::CreateHttpThreads(void)
{AssignResource();
…// 取得文件大小，以决定各个线程下载的起始位置
…// 把文件分成 FHttpThreadCount 块
…// 修正最后一块的大小
…
HttpThread->OutFileName=OutTmpFiles[FHttpThreadCount-1];
HttpThread->Index=FHttpThreadCount-1;
// 支持断点续传，建立多个线程
for(int i=0;i<FHttpThreadCount-1;i++)
{HttpThread=CreateHttpThread();
 HttpThread->StartPostion=Starts[i];
 HttpThread->GetBytes=Bytes[i];
 HttpThread->OutFileName=OutTmpFiles[i];
```

```
HttpThread->Index=i;
HttpThreads[i]=HttpThread;
    }
  }
```

线程下载完成后，会发出 OnThreadComplete 事件，在这个事件中判断是否所有下载线程都已经完成，如果是，则合并文件的各个部分。应该注意，这里有一个线程同步的问题，否则几个线程同时产生这个事件时，会互相冲突，结果也会混乱。同步的方法很多，这里的方法是创建线程互斥对象。

```
const char *MutexToThread="http-get-thread-mutex";
void __fastcall THttpGetEx::OnThreadComplete(TObject *Sender, int Index)
{// 创建互斥对象
 HANDLE hMutex= CreateMutex(NULL,FALSE,MutexToThread);
 DWORD Err=GetLastError();
 if(Err==ERROR_ALREADY_EXISTS) // 已经存在，等待
 {WaitForSingleObject(hMutex,INFINITE);//8000L);
  hMutex = CreateMutex (NULL,FALSE,MutexToThread);
 }
 // 当一个线程结束时，检查是否全部完成
 …
 if(S)// 下载完成，合并文件的各个部分
 {…}
}
```

5 小结

随着计算机技术和网络技术的发展，越来越多的应用需要用到断点续传功能。而出于效率方面的考虑，必然要使用多线程技术。本文论述了基于多线程技术的断点续传的实现方法，该方法已经应用到《机动车修理业治安信息管理系统》中，较好地满足了系统的实际需求。

参考文献：

[1]谢希仁.计算机网络[M].大连:大连理工大学出版社,1996.

[2]蒋东兴.Windows Socket 程序设计指南[M].北京:清华大学出版社,1995.

[3]梅杓春,许世民.Windows 环境下的 Socket 编程技术[J].计算机与通信,1998,(7):28-30.

(上接第 999 页)

HASH 函数的特性确保两个相异报文不可能有相同摘要，从而实现数据完整性验证及用户身份认证。

2.3 QoS 技术

基于公共网络并利用隧道技术和加密技术建立的 VPN 稳定性较差。因为公网流量的不确定性导致其带宽利用率较低，流量高峰易产生网络瓶颈，使实时性要求高的数据不能及时传输，流量低时又会造成带宽浪费。此外，用户及业务的差异对服务质量要求也不同，如移动办公用户提供广泛连接、覆盖是保证 VPN 服务的主要因素；而拥有众多分支机构的交互式内部网则要求提供良好的稳定性等，这些均要求网络根据需要提供不同等级的服务质量 QoS。要充分利用有限的公网资源为重要数据提供可靠的带宽，需在主机网络中采用 QoS 技术。通过流量预测及流量控制策略，按优先级分配带宽资源，实现各类数据的有效传输，预防阻塞等情况的发生。此外，还需从多角度实现 VPN 的有效管理，目的是保证 VPN 系统安全性、提高其可靠性、扩展性。本质上说，VPN 管理就是远程访问服务管理，具体包括安全管理、设备管理、配置管理、访问控制列表管理、QoS 管理等。用户选择 VPN 技术时必须考虑到管理方面的诸多问题，既包括管理使用者权限，还有对存取资源的存取或远程访问原则的建立等。这样，用户才能将其网络管理从局域网无缝地延伸到公用网，甚至是客户和合作伙伴。要实现 VPN 系统的有效管理，网管应实时跟踪、掌握系统使用者、连接情况、异常活动、出错情况等信息。

3 总结

基于公共网的 VPN 通过隧道技术、数据加密技术以及 QoS 机制，使得企业能够降低成本、提高效率、增强安全性。特别是 VPN 能保证传输数据真实性、完整性、通道的机密性，并提供动态密匙交换功能、安全防护措施及访问控制等，更使之成为众多用户实现数据安全传输的首选。可以相信，随着 VPN 技术日趋完善，安全性、可靠性日趋提高，其应用领域将更加广泛。

参考文献：

[1] 孔蕴藉.VPN 技术发展趋势.http://www.21 ctn.com/news/04-0209/c12.htm.

[2]陈劲. 基于第三层交换 VLAN 技术网络应用[J].福建电脑,2005,(4).