Windows NT 平台的网络代理程序的开发

许博义 赵钢 (北京系统工程研究所 100101)

摘要:本文讨论了在 Windows NT 开发基于 SNMP 协议的网络代理程序和网络管理程序的方法。

关键词:网络代理 SNMP 协议 MIB 网络管理

一、前言

基于 SNMP 协议的网络管理软件种类繁多,许多网络厂商推出各种平台的网络管理软件,有基于工作组级,如 3COM 公司的 TRANSCENDER;有基于企业级的,如 HP 公司的 NNM。在网络开发中,这些通用的网络管理程序通用性强,使用方便。但对于一些比较特殊设备或单一的应用,这些网络管理软件就显的无能为力。开发自己的网络管理程序和网络代理程序有时显得更加重要。在本文中,讨论如何在 Windows NT 平台上,利用 VC++ 开发基于 SNMP 协议的网络管理程序和代理程序。开发出的网络管理程序和网络代理程序既可以作为一个独立的支持一般和特殊网络管理功能的程序使用,也可作为对通用网络管理平台的扩充。

在 SNMP 模型中,在网络设备上负责收集设备信息的软件部件称为代理。代理负责响应并执行对网络设备的请求,与之相对应的发出请求的是网络管理程序。网络管理程序和代理程序运行在不同的网络设备并通过网络协议 SNMP 进行通信。SNMP 代理所使用的数据按一定格式组织为一定的可收集单元,称为管理信息库 MIB(management information base)。MIB 不是一个明显的数据库,而是在代理上逻辑可以得到的数值。在 Windows NT 平台安装 SNMP 时,系统安装以下文件以支持用户在平台上开发代理程序和或网络管理程序。

DHCPMIB:DHCP 扩展代理 DLL。

INETMIBI.DLL:Internet MIB II 扩展代理 DLL。

LMMIB2.DLL:LAN Manager MIB 2 扩展代理 DLL。

MGMTAPI.DLL:SNMP 管理程序 API 库。

MIB.BIN:包含 MIB 的二进制文件。

SNMP.EXE:SNMP 代理服务。

SNMPTRAP.EXE 接收 SNMP trap 并传递给 MGMTALI.DLL。

WINSMIB.DLL:WINS 扩展代理 DLL。

二、代理程序

Windows NT 的 SNMP 服务(snmp.exe)是一个支持 SNMP 协议的可扩展代理(extendibleagent)程序,它允许用户添加动态连接库(DDL)以支持额外的 MIB。这些 MIB 可以是为特殊的设备或应用而建立的。snmp.exe 负责接收向 NT 工作站或服务器发来的请求并把这些请求传递给相应的 DDL 来处理。响应的数据再返回到代理程序 snmp.exe,由它负责返回到发出请求的网络管理站。缺省的情况,有四个扩展代理(Extension agent)DLL 随 SNMP 服务一起安装在系统上。这些 DLL 服务于 MIB-II、LanMan2、DHCP 和 WINS 的 MIB。从 snmp.exe 的结构可以看出,用户可以利用提供的 API 创建自己的 DLL 来完成网络管理的特殊要求。

可扩展代理通过 Windowsock DLL 与网络接口,如图所示:

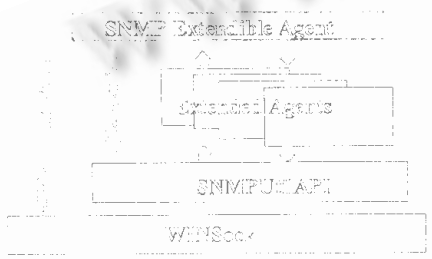

图 1

根据扩展代理功能要求,在扩展代理的 DLL 中至少要提供三个 API(在下面描述)。用户在完成 DLL 后,要在注册文件的相应位置上设置一些值。这些值说明 DLL 位置,版本号等一些信息。可扩展代理根据位于 HKEY _ Local _ Machine/SYSTEM/CurrentControlSet / Services/SNMP /Parameters/ExtensionAgents 下的注册值来确定应该加载哪些 DLL。在服务启动时,一旦访问过注册表,可扩展代理通过 DLL 提供的 SnmpExtensionInit API 获得该 DLL 所支持的 MIB 的前缀。在 SnmpExtensionInit API 函数中有相应的变量指明 DLL 所支持的 MIB 的前缀并传递给可扩展代理。可扩展代理建立一张支持的 MIB 及相应解析例程的表并分类存储。一旦所有的 DLL 加载完毕而且表被初始化好,则可扩展代理程序就可以处理 SNMP 请求。

每一个扩展代理必须提供下列三个 API:

·SnmpExtensionInit:扩展代理返回一个所支持的 OID 名字空间和一个事件地址。

·SnmpExtensionQuery:扩展代理解析 Set/Get/GetNext 请求。

·SnmpExtensionTrap:返回要发送的已格式化好的 trap PDU。

另外,扩展代理可以提供第四个 API,SnmpExtensionInitEx。该 API 允许一个扩展代理在一个 DLL 中支持多个 MIB 或多个 MIB 视(view)。

1.SnmpExtensionInit

SnmpExtensionInit 是可扩展代理在初始化系统时调用的。一个最简单的示例如下:

```
BOOL WINAPI SnmpExtensionInit(
IN DWORD                dwTimeZeroReference,
OUT HANDLE              * hPollForTrapEvent,
OUT AsnObjectIdentifier * supportedView)
dwTimeZero = dwTimeZeroReference;
if (( * hPollForTrapEvent = CreateEvent ( NULL,
  FALSE, FALSE, NULL)) = = NULL)
{
 * hPollForTrapEvent = NULL;
 * supportedView = NULL;
return FALSE;
}
 * supportedView = MIB-OidPrefix;//NOTE! structure copy
hSimulateTrap = * hPollForTrapEvent;
return TRUE;
}
```

我们对上面 API 的实现作简单的分析。DwTimeZeroReference 是传递到该 API 的变量,SnmpExtensionInit 将此值保存在一个变量中,这样扩展代理有个零参考点。如果因为某种原因扩展代理需要知道它已经运行了多长时间,则可以通过用 GetCurrentTime()的值减去该值得到所需的值。许多 MIB 需要"启动时间"的概念,或者自重启动或初始化以来经过的时间,利用 DwTimeZeroReference 就可得到相应的值。另外,支持 trap 的 MIB 也将需要"启动时间"值,因为这个值将包含在 PDU 报文中。

HpollForTrap 事件是扩展代理准备发送 trap 时将要信号通知的事件。扩展代理中包含的是可扩展代理调用的过程,因此扩展代理在它认为要发送一个 trap 时,不会主动的处理一个请求。一个 trap 是基于一个外部事件,如打印机缺纸这种情况。如果当前没有任何管理工作站

在查询打印机的 MIB,则打印机的 MIB DLL 应该设法通知可扩展代理,它需要被调用来完成某些处理。一种方法是可扩展代理周期的轮询 tarp。如果开发人员把来自打印机的信号与事件关联起来,这种方法就更有效。当信号发生时通知可扩展代理,调用转向 DLL 提供的 SnmpExtensionTrap API。我们在后面将解释 SnmpExtensionTrap API。

最后,supportedView 是一个 OID,标识的是该 DLL 所支持的 MIB 或 MIB 视,它将传递给可扩展代理程序。可扩展代理程序因此可以建立一张 DLL 与所支持的 MIB 的表,因此可以处理 snmp 请求。

2.SnmpExtensionQuery

前面提到,扩展代理程序根据 SnmpExtensionInit 中 supportedView 返回值,可以建立一张 DLL 与所支持的 MIB 的表。当可扩展代理程序判断出某个扩展代理的 DLL 的 SnmpExtensionInit 返回的 supportedView 值与 SNMP 请求的 OID 匹配时,就调用该 DLL 中的 SnmpExtensionQuery。下面给出的是 SnmpExtensionQuery 的简单程序示例。

```
BOOL WINAPI SnmpExtensionQuery (
    IN BYTE                              requestType,
    IN OUT RFC1157VarBindList            * variableBind-
ings,
    OUT AsnInteger                       * errorStatus,
    OUT AsnInteger                       * errorIndex)
    {
    UINT I;
    for ( I=0;I< variableBindings - >len;I + + )
    {
     * errorStatus = ResolveVarBind(
          &variableBindings - >list[I],
               requestType);
    if (  * errorStatus = = SNMP _ ERRORSTATUS _
NOSUCHNAME&&
    requestType = = MIB _ ACTION _ GETNEXT)
    {
        * errorStatus = SNMP _ ERRORSTATUS _ NO-
ERROR;
        SNMP _ oidfree (&variableBindings - >list[I].
name);
        SNMP _ oidcpy( &variableBindings - >list[I].
name,
          &MIB _ OidPrefix);
        variableBindings - >list[I]. name. ids[MIB _
PREFIX _ LEN-1] + + ;
    }
    if ( * errorStatus ! = SNMP-ERRORSTATUS-NO-
ERROR)
    {
      * errorIndex = I + I;
      break;
    }
  }
  if ( * errorStatus = = SNMP-ERRORSTATUS-NOER-
ROR)
    ActionCaches(TOASTER-COMMIT);
  else
    ActionCaches(TOASTER-CLEAR);
}// end SnmpExtensionQuery()
```

requestType 指明被处理的 SNMP 请求的类型。可能的请求类型有:

· ASN _ RFC1157 _ GETREQUEST 指明 SNMP Get 请求

· ASN _ RFC1157 _ GETNEXTREQUEST 指明 SNMP Get Next 请求

· ASN _ RFC1157 _ GETREQUEST 指明 SNMP Get 请求

ResolveVarBind 子程序实际负责执行设置或获取 VarBind 数据。因为 Dll 能够处理一个 VarBindList,在一个 SNMP 请求中就可以修改多个对象。SNMP 要求,如果代理不能成功的完成一个 PDU 中的所有操作,则不能仅完成其中的一个。因此,ResolveVarBind 只修改缓存数据。SnmpExtensionQuery 核实是否 errorStatus 被设置为 SNMP _ ERRORSTATUS _ NOERROR,如果是,则用 TOASTER _ COMMIT 参数调用 Action Caches,处理并提交数据。如果发现错误,使用 TOASTER _ CLEAR 参加调用 Action Caches 调用,抛弃所有的缓存数据。

对于超出该 DLL 所支持的 MIB 视的 GETNEXT 请求的处理,程序核查从 ResolveVarBind 返回的错误状态是否是 SNMP _ ERRORSTATUS _ NOSUCHNAME,如果是,他将改变传递进来的 VarBind,使它指向当前 DLL 支持的视的边界,并指示没有错误发生,可扩展代理将检查这个条件并相应处理这个请求。

ResolveVarBind 子程序在处理查询时,如果发现异常现象或某些变量满足特定条件,则调用 SetEvent (hSimulateTrap),由此开始发送 trap 的过程。

3.SnmpExtensionTrap

当可扩展代理通过信号通知事件,收到某个代理要发送 trap 的通知时,调用 SnmpExtensionTrap API。如果

它能够成功完成初始化等工作，则返回 TRUE 并希望被再次调用。原因有两个：第一个原因是把 trap 送到多个管理工作站，第二个原因是给 SnmpExtensionTrap 一个机会使它能够释放为 VarBinds 动态分配的内存。可以通过使用记载状态信息的静态变量（如下面例子中的 whichTime），能够决定 API 是否被调用来发送 trap 或在发送 trap 后释放内存。从下面的例子可以看出，当所有的数据处理完毕，程序返回 TRUE，这是通知可扩展代理，在它成功的发送完 trap 后，它应该被再次调用。因为状态变量已被设置为 TRAP _ CLEANUP，等到它被再次调用时，将使用 SnmpUtilVarBindFree API 释放 VarBind 相关的内存，重置状态变量为 TRAP _ GENERATION 且返回 FALSE，通知可扩展代理不需要采取其他的动作。

```
BOOL WINAPI SnmpExtensionTrap(
OUT AsnObjectIdentifier  * enterprise,
OUT AsnInteger           * genericTrap,
OUT AsnInteger           * specificTrap,
OUT AsnTimeticks         * timeStamp,
OUT RFC1157VarBindList   variableBindings)
{
static UINT OidList[ ] = {1,3,6,1,4,1,12,2,6,0};
static UINT OidListLen = 10;
static RFC1157VarBind ToastUpVarBind;
static whichTime = TRAP _ GENERATION;
if (whichTime = = TRAP _ GENERATION)
{
   whichTime = TRAP _ CLEANUP; //Supports the
   simulation.
   // Communicate the trap data to the Extendible A-
   gent.
   enterprise - >idLength = OidListLen;
   enterprise - >ids = (UINT * ) SNMP _ alloc(size-
   of(UINT) *
   7);
   memcpy(enterprise - >ids, OidList, sizeof(UINT)
   *
   7);
   * genericTrap = SNMP _ GENERICTRAPEN-
   TERSPECIFIC;
   * specificTrap = 0;
   * timeStamp = (Get)CurrentTime( )/10) - dw-
   TimeZero;
   ToastUpVarBind - > name. ids = SNMP _ alloc
   (OidListLen);
   ToastUpVarBind - >name. len = OidListLen;
   memcpy( ToastUpVarBind - > name. ids, OidList,
   OidListLen);
   ToastUpVarBind - > value. type = ASN _ INTE-
   GER;
   ToastUpVarBind - >value. value = ReasonUp;
   variableBindings - >list = ToastUpVarBind;
   variableBindings - >len = 1;
   // Indicate that valid trap data exists in the
   // and parameters.
   return TRUE;
}
else
{
   whichTime = TRAP _ GENERATION;
   SnmpUtilVarBindFree(varableBindings - >list);
   return FALSE;
}
} //end SnmpExtensionTrap( )
```

四、网络管理程序

上一节提到，用户可以开发支持特殊设备或服务的代理程序。代理程序可以同任何的支持 SNMP 协议的网络管理程序通信。但在一些条件下，需要开发自己的网络管理程序来管理这些代理程序。在 Windows NT 平台，用户可以开发网络管理程序来完成对网络管理。

开发网络管理程序相对开发代理程序要简单一些，在 Windows NT 平台上，提供如下函数用于开发网络管理程序。SnmpMgrClose，SnmpMgrGetTrap，SnmpMgrOidToStr，SnmpMgrOpen，SnmpMgrRequest，SnmpMgrStrToOid，SnmpMgrTrapListen。另外还有一些开发实用函数。

建立网络管理程序的一般步骤是：首先使用 SnmpMgrOpen 建立一个与远程代理的对话，双方约定 Community String 和通信的超时时间及通信重试次数。然后使用 SnmpMgrRequest 发出请求，对收到的报文进行分析和处理。对于事件 trap，要使用 SnmpMgrTrapListen 对事件侦听，然后使用 WaitForSingleObjict 等待事件，然后使用 SnmpMgrGetTrap 处理每一个事件。

（来稿时间：1998 年 9 月）