友我科技PCSC双界面读写器YW-606 开发指南

1．建立资源管理器的上下文

函数 ScardEstablishContext()用于建立将在其中进行设备数据库操作的资源管理器上下文（范围）。

函数原型：LONG SCardEstablishContext(DWORD dwScope, LPCVOID pvReserved1, LPCVOID pvReserved2, LPSCARDCONTEXT phContext);

各个参数的含义：

（1）dwScope：输入类型；表示资源管理器上下文范围，取值为：SCARD_SCOPE_USER（在用户域中完成设备数据库操作）、SCARD_SCOPE_SYSTEM（在系统域中完成设备数据库操作）。要求应用程序具有相应的操作权限。

（2）pvReserved1：输入类型；保留，必须为 NULL。

（3）pvReserved2：输入类型；保留，必须为 NULL。

（4）phContext：输出类型；建立的资源管理器上下文的句柄。

下面是建立资源管理器上下文的代码：

```
SCARDCONTEXT hSC;
LONG lReturn;
lReturn = SCardEstablishContext(SCARD_SCOPE_USER, NULL, NULL, &hSC);
if ( lReturn!=SCARD_S_SUCCESS )
printf("Failed SCardEstablishContext\n");
```

2． 获得系统中安装的读卡器列表

函数 ScardListReaders()可以列出系统中安装的读卡器的名字。

函数原型：LONG SCardListReaders(SCARDCONTEXT hContext, LPCTSTR mszGroups, LPTSTR mszReaders, LPDWORD pcchReaders);

各个参数的含义：

（1）hContext：输入类型；ScardEstablishContext()建立的资源管理器上下文的句柄，不能为 NULL。

（2）mszGroups：输入类型；读卡器组名，为 NULL 时，表示列出所有读卡器。

（3）mszReaders：输出类型；系统中安装的读卡器的名字，各个名字之间用’\0’分隔，最后一个名字后面为两个连续的’\0’。

（4）pcchReaders：输入输出类型；mszReaders 的长度。

系统中可能安装多个读卡器，因此，需要保存各个读卡器的名字，以便以后与需要的读卡器建立连接。

下面是获得系统中安装的读卡器列表的代码：

```
char mszReaders[1024];
LPTSTR pReader, pReaderName[2];
DWORD dwLen=sizeof(mzsReaders);
int nReaders=0;
lReturn = SCardListReaders(hSC, NULL, (LPTSTR)mszReaders, &dwLen);
if ( lReturn==SCARD_S_SUCCESS )
{
pReader = (LPTSTR)pmszReaders;
while (*pReader !='\0'    )
{
if ( nReaders<2 ) //使用系统中前 2 个读卡器
pReaderName[nReaders++]=pReader;
printf("Reader: %S\n", pReader );
//下一个读卡器名
pReader = pReader + strlen(pReader) + 1;
}
}
```

3． 与读卡器（智能卡）连接

函数 ScardConnect()在应用程序与读卡器上的智能卡之间建立一个连接。

函数原型：LONG SCardConnect(SCARDCONTEXT hContext, LPCTSTR szReader, DWORD dwShareMode, DWORD dwPreferredProtocols, LPSCARDHANDLE phCard, LPDWORD pdwActiveProtocol);

各个参数的含义：

（1）hContext：输入类型；ScardEstablishContext()建立的资源管理器上下文的句柄。

（2）szReader：输入类型；包含智能卡的读卡器名称（读卡器名称由 ScardListReaders()给出）。

（3）dwShareMode：输入类型；应用程序对智能卡的操作方式，SCARD_SHARE_SHARED

（多个应用共享同一个智能卡）、SCARD_SHARE_EXCLUSIVE（应用独占智能卡）、SCARD_SHARE_DIRECT（应用将智能卡作为私有用途，直接操纵智能卡，不允许其它应用访问智能卡）。
（4）dwPreferredProtocols：输入类型；连接使用的协议，SCARD_PROTOCOL_T0（使用 T=0 协议）、SCARD_PROTOCOL_T1（使用 T=1 协议）。
（5）phCard：输出类型；与智能卡连接的句柄。
（6）PdwActiveProtocol：输出类型；实际使用的协议。
下面是与智能卡建立连接的代码：

```
SCARDHANDLE hCardHandle[2];
DWORD dwAP;
lReturn = SCardConnect( hContext, pReaderName[0],       SCARD_SHARE_SHARED,
SCARD_PROTOCOL_T0 | SCARD_PROTOCOL_T1, &hCardHandle[0], &dwAP );
if ( lReturn!=SCARD_S_SUCCESS )
{
printf("Failed SCardConnect\n");
exit(1);
}
```

与智能卡建立连接后，就可以向智能卡发送指令，与其交换数据了。

4． 断开与读卡器（智能卡）的连接
在与智能卡的数据交换完成后，可以使用函数 ScardDisconnect()终止应用与智能卡之间的连接。
函数原型：LONG SCardDisconnect(SCARDHANDLE hCard, DWORD dwDisposition);
各个参数的含义：
（1）hCard：输入类型；与智能卡连接的句柄。
（2）dwDisposition：输入类型；断开连接时，对智能卡的操作，SCARD_LEAVE_CARD（不做任何操作）、SCARD_RESET_CARD（复位智能卡）、SCARD_UNPOWER_CARD（给智能卡掉电）、SCARD_EJECT_CARD（弹出智能卡）。
下面是断开与智能卡连接的代码：

```
lReturn = SCardDisconnect(hCardHandle[0], SCARD_LEAVE_CARD);
if ( lReturn != SCARD_S_SUCCESS )
{
printf("Failed SCardDisconnect\n");
exit(1);
}
```

5． 释放资源管理上下文
在应用程序终止前时，应该调用函数 ScardReleaseContext()释放资源管理器的上下文。
函数原型：LONG SCardReleaseContext(SCARDCONTEXT hContext);

各个参数含义：
（1）hContext：输入类型；ScardEstablishContext()建立的资源管理器上下文的句柄，不能为NULL。
下面是释放资源管理上下文的代码：

```
lReturn = SCardReleaseContext(hSC);
if ( lReturn!=SCARD_S_SUCCESS )
printf("Failed SCardReleaseContext\n");
```

以上介绍的通过 PC/SC 来操作智能卡的流程，可以封装在一个类中。例如，我们可以设计一个类:

```
class CYOWORFIDReader
{
private:
SCARDCONTEXT hSC;
LONG lReturn;
char mszReaders[1024];
LPTSTR pReader, pReaderName[2];
DWORD dwLen;
int nReaders, nCurrentReader;
SCARDHANDLE hCardHandle[2];
DWORD dwAP;
public:
CSmartReader(); //建立上下文、取读卡器列表
~CSmartReader(); //释放上下文
void SetCurrentReader(int currentReader);
int GetReaders(); //获得读卡器数目
int ConnectReader(); //与当前读卡器建立连接
int DisConnectReader(); //与当前读卡器断开连接
int SendCommand(BYTE command[], int commandLength, BYTE result[], int *resultLength); //向读卡器发送命令，并接收返回的数据。返回值为 sw
};
```

这样，我们就可以方便地使用 PC/SC 接口了。

6. 向智能卡发送指令
函数 ScardTransmit()向智能卡发送指令，并接受返回的数据。
函数原型：LONG SCardTransmit(SCARDHANDLE hCard, LPCSCARD_I0_REQUEST pioSendPci, LPCBYTE pbSendBuffer, DWORD cbSendLength, LPSCARD_IO_REQUEST pioRecvPci, LPBYTE pbRecvBuffer, LPDWORD pcbRecvLength);
各个参数的含义：
（1）hCard：输入类型；与智能卡连接的句柄。
（2）pioSendPci：输入类型；指令的协议头结构的指针，由 SCARD_IO_REQUEST 结构定义。后面是使用的协议的协议控制信息。一般使用系统定义的结构，SCARD_PCI_T0（T=0

协议）、 SCARD_PCI_T1（T=1 协议）、SCARD_PCI_RAW（原始协议）。
（3）pbSendBuffer：输入类型；要发送到智能卡的数据的指针。
（4）cbSendLength：输入类型；pbSendBuffer 的字节数目。
（5）pioRecvPci：输入输出类型；指令协议头结构的指针，后面是使用的协议的协议控制信息，如果不返回协议控制信息，可以为 NULL。
（6）pbRecvBuffer：输入输出类型；从智能卡返回的数据的指针。
（7）pcbRecvLength：输入输出类型；pbRecvBuffer 的大小和实际大小。

对于 T=0 协议，收发缓冲的用法如下：
（a）向智能卡发送数据：要向智能卡发送 n>0 字节数据时，pbSendBuffer 前 4 字节分别为 T=0 的 CLA、INS、P1、P2，第 5 字节是 n，随后是 n 字节的数据；cbSendLength 值为 n+5（4 字节头+1 字节 Lc+n 字节数据）。PbRecvBuffer 将接收 SW1、SW2 状态码；pcbRecvLength 值在调用时至少为 2，返回后为 2。

```
BYTE recvBuffer[260];
int sendSize, recvSize;
BTYE sw1, sw2;
BYTE       select_mf[]={0xC0, 0xA4, 0x00, 0x00, 0x02, 0x3F, 0x00};
sendSize=7;
recvSize=sizeof(recvBuffer);
lReturn = SCardTransmit(hCardHandle[0], SCARD_PCI_T0, select_mf, sendSize,
NULL, recvBuffer, &recvSize);
if ( lReturn != SCARD_S_SUCCESS )
{
printf("Failed SCardTransmit\n");
exit(1);
}
//返回的数据，recvSize=2
sw1=recvBuffer[recvSize-2];
sw2=recvBuffer[recvSize-1];
```

（b）从智能卡接收数据：为从智能卡接收 n>0 字节数据，pbSendBuffer 前 4 字节分别为 T=0 的 CLA、INS、P1、P2，第 5 字节是 n（即 Le），如果从智能卡接收 256 字节，则第 5 字节为 0；cbSendLength 值为 5（4 字节头+1 字节 Le）。PbRecvBuffer 将接收智能卡返回的 n 字节，随后是 SW1、SW2 状态码；pcbRecvLength 的值在调用时至少为 n+2，返回后为 n+2。

```
BYTE       get_challenge[]={0x00, 0x84, 0x00, 0x00, 0x08};
sendSize=5;
recvSize=sizeof(recvBuffer);
lReturn = SCardTransmit(hCardHandle[0], SCARD_PCI_T0, get_challenge,
sendSize, NULL, recvBuffer, &recvSize);
if ( lReturn != SCARD_S_SUCCESS )
{
printf("Failed SCardTransmit\n");
exit(1);
}
```

```
//返回的数据, recvSize=10
sw1=recvBuffer[recvSize-2];
sw2=recvBuffer[recvSize-1];
//data=recvBuffer[0]----recvBuffer[7]
```

（c）向智能卡发送没有数据交换的命令：应用程序既不向智能卡发送数据，也不从智能卡接收数据，pbSendBuffer 前 4 字节分别为 T=0 的 CLA、INS、P1、P2，不发送 P3；cbSendLength 值必须为 4。PbRecvBuffer 从智能卡接收 SW1、SW2 状态码；pcbRecvLength 值在调用时至少为 2，返回后为 2。

```
BYTE       set_flag[]={0x80, 0xFE, 0x00, 0x00};
sendSize=4;
recvSize=sizeof(recvBuffer);
lReturn = SCardTransmit(hCardHandle[0], SCARD_PCI_T0, set_flag, sendSize,
NULL, recvBuffer, &recvSize);
if ( lReturn != SCARD_S_SUCCESS )
{
printf("Failed SCardTransmit\n");
exit(1);
}
//返回的数据，recvSize=2
sw1=recvBuffer[recvSize-2];
sw2=recvBuffer[recvSize-1];
```

（d）向智能卡发送具有双向数据交换的命令：T=0 协议中，应用程序不能同时向智能卡发送数据，并从智能卡接收数据，即发送到智能卡的指令中，不能同时有 Lc 和 Le。这只能分两步实现：向智能卡发送数据，接收智能卡返回的状态码，其中，SW2 是智能卡将要返回的数据字节数目；从智能卡接收数据（指令为 0x00、0xC0、0x00、0x00、Le）。

```
BYTE get_response={0x00, 0xc0, 0x00, 0x00, 0x00};
BYTE       internal_auth[]={0x00, 0x88, 0x00, 0x00, 0x08, 0x01, 0x02, 0x03, 0x04, 0x05, 0x06,
0x07, 0x08};
sendSize=13;
recvSize=sizeof(recvBuffer);
lReturn = SCardTransmit(hCardHandle[0], SCARD_PCI_T0, internal_auth,
sendSize, NULL, recvBuffer, &recvSize);
if ( lReturn != SCARD_S_SUCCESS )
{
printf("Failed SCardTransmit\n");
exit(1);
}
//返回的数据，recvSize=2
sw1=recvBuffer[recvSize-2];
sw2=recvBuffer[recvSize-1];
if ( sw1!=0x61 )
```

```
{
printf("Failed Command\n");
exit(1);
}
get_response[4]=sw2;
sendSize=5;
recvSize=sizeof(recvBuffer);
lReturn = SCardTransmit(hCardHandle[0], SCARD_PCI_T0, get_response,
sendSize, NULL, recvBuffer, &recvSize);
if ( lReturn != SCARD_S_SUCCESS )
{
printf("Failed SCardTransmit\n");
exit(1);
}
//返回的数据，recvSize=10
sw1=recvBuffer[recvSize-2];
sw2=recvBuffer[recvSize-1];
//data=recvBuffer[0]----recvBuffer[7]
```

RFID产品手册下载