嵌入式系统应用程序方案之二

——利用有限状态机实现 FTP 文件传输

英创信息技术有限公司

2006 年 5 月

结合“嵌入式系统应用程序方案之一”，本文着重介绍以英创公司嵌入式 PC 模块为平台，利用事件驱动机制来实现 FTP 文件传输的具体实现方案。常规的 FTP 传送采用的是阻塞方式进行的，这样在 FTP 传送过程中，CPU 不能做其他的操作，这种情况在使用 GPRS 进行数据传输时尤为突出。利用事件驱动机制，可实现非阻塞的 FTP 文件传输，既程序无需等待对端响应，而是返回系统，这样系统就可利用 FTP 等待的时间进行其他的必要操作，然后再继续 FTP 的文件传输，从而提高整个系统实时响应的能力，降低总的系统开销。FTP 内部采用有限状态机进行管理，一旦程序需要等待对端的响应时，程序记录下当前的状态，并返回系统控制，等待下次再进入该任务模块时，程序根据当前的状态继续相应的处理。

在该方案中提供的 FTP 文件传输功能还实现了 FTP 文件断点续传。这一功能在进行较大文件传输或网络状况不稳定时非常有用，当文件传输中途遇到失败时，就可以启动断点续传功能，使得下一次传输直接从断点处继续传输，而不需要再重头开始，从而大大提高了文件传输的效率。

函数说明

对于 FTP 文件传输，采用 C++的类来实现，定义了一个 FTPClient 的 C++类，在该类对象定义了 6 个公共函数：Init( )、SendFile( )、ReceiveFile( )、Do( )、Resume( )和 Stop( )函数。下面对各个函数作详细介绍。

（1） int Init(char *host, char *FTPusername, char *FTPpassword, unsigned long timeout, int FTPMode);

功能描述：

初始化设置 FTP 文件传输参数。

输入参数：

char *host 远端主机的 IP 地址，如“192.168.201.34”。

char *FTPusername 登录时使用的用户名，如“guest”

char *FTPpassword 登录时使用的密码，如“888”

unsigned long timeout 定义的 timeout 时间，单位为毫秒。

int FTPMode 登录 FTP 的模式，= 0 为标准模式

= 1 为 passive 模式

返回值:

!=0 该函数调用失败

=0 该函数调用成功

备注：

该函数只需要在系统初始化时调用一次。

（2） int SendFile(char *file, int mode);

功能描述：

以 FTP 客户端方式，启动向远端 FTP 服务器发送文件。

输入参数：

char *file 被操作的文件名，如“myfile.txt”。

int mode =0; 从远端服务器发送 ASCII 文件

=1；从远端服务器发送 2 进制文件

返回值:

!=0 该函数调用失败

=0 该函数调用成功

（3） int ReceiveFile(char *file, int mode);

功能描述：

以 FTP 客户端方式启动从远端 FTP 服务器获取文件。

输入参数：

char *file 被操作的文件名，如“myfile.txt”。

int mode =0; 从远端服务器获取 ASCII 文件

=1；从远端服务器获取 2 进制文件

返回值:

!=0 该函数调用失败

=0 该函数调用成功

（4） int Do( )

功能描述：

以 FTP 客户端方式，执行远端 FTP 服务器发送文件或从远端 FTP 服务器获取文件，只有当该函数的返回值为 0 时才表明 FTP 文件传输成功。

返回值:

= 0 FTP 文件传输成功。

< 0 FTP 文件传输失败，并返回相应的错误代码。

> 0 FTP 文件传输过程中的各个状态。

（5） int Resume( );

功能描述：

在调用 Do( )函数返回 FTP 传输失败时，可通过调用该函数启动文件断点续传功能，再配合调用 Do( )函数完成文件剩余部分的传输。

（6） int Stop( );

功能描述：

终止当前的 FTP 文件传输。

函数调用

在具体的应用中，首先调用 Init( )函数初始化设置 FTP 传输的相关参数，并通过 SendFile( )、RecieveFile( )函数来启动 FTP 文件传输，然后应用程序不断调用 Do( )函数进行 FTP 文件传输，应用程序可直接通过检查该函数的返回值来判断 FTP 文件传输是否成功。在我们提供的例程 APP2.PRJ，利用 FTPCLient 类提供的这些函数来实现 FTP 文件传输任务。通过 CMD_TICK 来启动 FTP 文件传输任务 CMD_FTP，CMD_TICK 是系统自动产生。

在执行 CMD_FTPDO 中调用函数 Do( )，通过其返回值来判断 FTP 文件传输是否完成，如果没有完成就继续发送该命令，这样程序就不会阻塞在 FTP 文件传输过程中，在这其间还可以执行别的操作，比如说中断产生的命令。如果返回值<0 表明 FTP 文件传输失败，此时可调用 Resume( )启动文件断点续传，在发送 CMD_FTPDO 命令来继续文件剩余部分的传输。

下面主程序的代码作相关的说明。

主程序代码分析

```
int     SysInit( );                                  // 系统初始化函数定义
int     SysExit( );                                  // 系统退出处理

int main( )
{
    int             i1, len, FirstFlag, ExitFlag;    // 局部变量
    CMD             CmdCode;                         // 系统命令枚举变量
    char            FTPFileName[40];
    long            xlen;
    PPPGPRSState    PPPState;
    unsigned char   OwnIPStr[20];

    union CMD_PARAMETER ThisPar;                     // 系统命令所带参数

    i1 = SysInit( argc, argv );                      // 首先进行初始化
    for( FirstFlag=0, ExitFlag=0; ; )                // 系统主循环
    {
        //ReloadWDT( );                              // 加载 watchdog
        CmdCode = CmdQueue.GetCmd( (char*)&ThisPar );  // 从系统任务队列读取命令
        switch( CmdCode )
        {
            case CMD_NOP:
                PPPState = PPP_Running( );           // GPRS 自动拨号上网
                if( PPPState!=PPPLINKUP )
                    break;
                if( FirstFlag==0 )
                {
                GetOWNIP( OwnIPStr );
                printf( "\nIP=%d.%d.%d.%d\n", OwnIPStr[0], OwnIPStr[1], OwnIPStr[2],
                        OwnIPStr[3] );
                FirstFlag = 1;
```

```
        }
    break;

case CMD_TICK:                  // on every 55ms
    if( !FirstFlag )    break;
    if( FTPDone )       break;
    if( argc > 1 )  strcpy( FTPFileName, argv[1] );
    else            strcpy( FTPFileName, "ftpclnt2.exe" );
    if( argc > 2 )  i1 = atoi( argv[2] );
    else            i1 = 3;

    if( i1&0x02 )
    {
        printf( "send file %s\n", FTPFileName );
        pFTPClnt->SendFile( FTPFileName, i1 );
    }
    else
    {
        printf( "receive file %s\n", FTPFileName );
        pFTPClnt->ReceiveFile( FTPFileName, i1 );
    }
    CmdQueue.PushCmd( CMD_FTPDO );
    FTPDone = 1;
    break;

case CMD_FTPDO:
    i1 = pFTPClnt->Do( );
    if( i1 > 0 )
    {
        xlen = pFTPClnt->XLength( );
        if( xlen >= 0 )  printf( "%2d %7ld \r", i1, xlen );
        CmdQueue.PushCmd( CMD_FTPDO ); // keep going
    }
    else if( i1 < 0 )
    {
        printf( "\nFTP fail %d\n", i1 );
        printf( "FTP resume!\n" );
        pFTPClnt->Resume( );
        CmdQueue.PushCmd( CMD_FTPDO ); // keep going
    }
    else
    {
        printf( "\nFTP do ok!\n" );
        FTPDone = 0;
```

```
                        }
                        break;

                default: ExitFlag = 1;                          // 非法命令，退出
            }
            if( ExitFlag ) break;
        }

        SysExit( );
        return 0;
}

// return = 0: ok!
//          < 0: fail
int SysInit( int argc, char** argv )
{
        int i1;

        pFTPClnt = new class FTPClientManager( );
        pFTPClnt->Init( "222.210.195.109", "guest", "888", 10000, 0 );

        //EnableWDT( 10.0 );        // 10 sec Watchdog
        CmdQueue.StartQueue( );
        return 0;
        // Let's go to main loop!!
}

int SysExit( )
{
        CmdQueue.StopQueue( );
        delete pFTPClnt;
        return 0;
        // Let's go back to DOS
}
```