利用停止等待协议传输数据文件

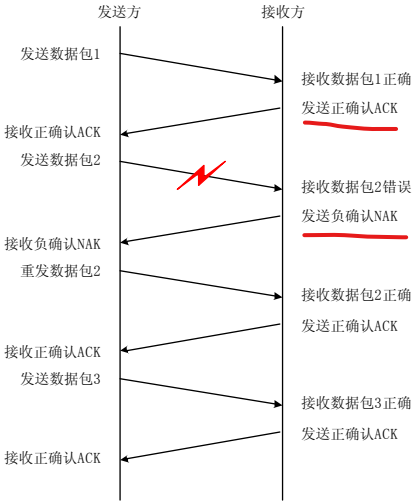
有噪声的停止协议

① receiver buffer=1 ② noisy channel

停止等待协议

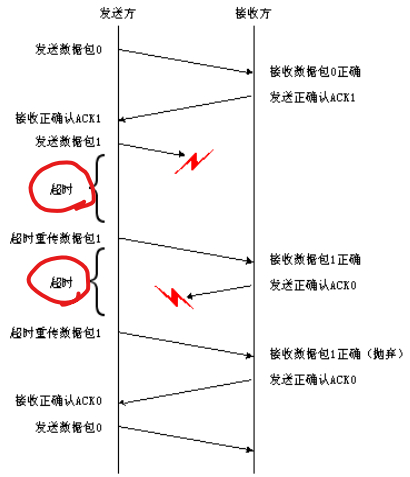
- 停止等待协议：一种最简单的（但效率较低的）差错和流量控制协议
- 基本原理：发送方发完一数据包后，需要等待接收方的应答信息
  - 发送方收到正确认信息ACK:接收方接收正确，发送方可发送下一数据包
  - 发送方收到负确认信息NAK:接收方接收错误，发送方应重发出错的数据包

停止等待协议的基本工作过程



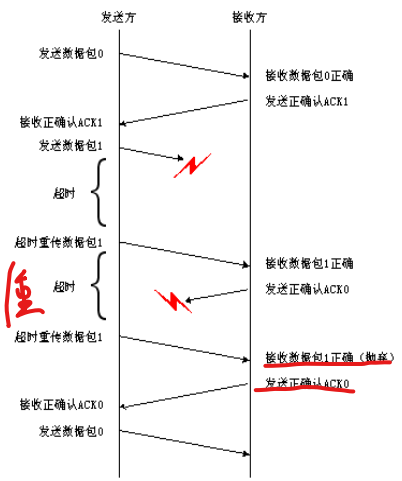
数据包丢失

- 数据包在传输过程中丢失，接收方未接收到任何数据
  - 发送方在发送数据包后启动定时器
  - 规定时间内没有收到确认信息，则认为数据包丢失，需重传该包
  - 重传次数达到一定的值，则数据传输失败

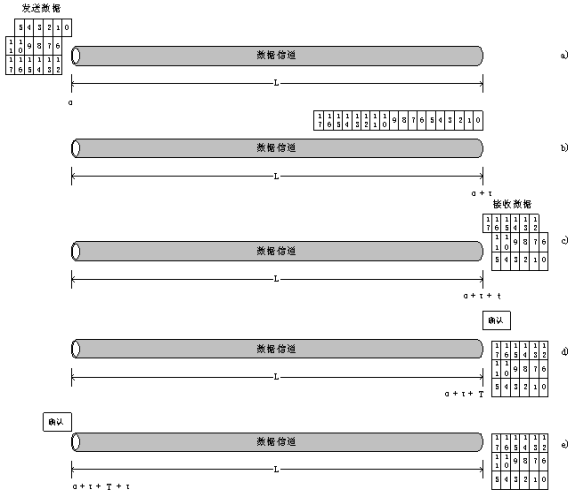


确认信息丢失

- 确认信息在传输过程中丢失
  - 发送方采用定时器，定时器溢出后重发数据包
  - 数据包编号
    - 接收方收到与前一数据包编号相同的数据包后需要将其抛弃
    - 停等机制至少需要两个编号（通常为0和1）
    - 确认信息需要指明收到数据包的编号。ACK1：准备接收编号为1的数据包；ACK0：准备接收编号为0的数据包



停止等待协议的效率



停止等待协议的效率

$$e=\frac{T}{(\alpha+\tau+T+\tau)-\alpha}=\frac{T}{T+2\tau}=\frac{l/v}{\left(l/v+2\times L/V\right)}=\frac{1}{1+\left(2\times Lv/lV\right)}$$

- 传输速率V在某种传输介质中是固定的
- 信道的长度L越长、数据速率v越高、发送的数据位数l越少，传输效率越低

差错检测

- 奇偶校验（parity check）
- 校验和（checksum）
- 循环冗余校验码（CRC，cyclic redundancy code）
- ……

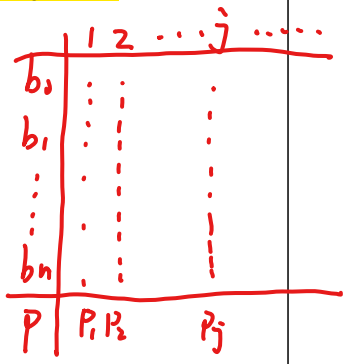
奇偶校验

- 停等协议中通常使用纵向的奇偶检验
  - 发送方在发送的数据块后扩展一个字节
  - 扩展字节中的第j位保证所有发送数据字节中第j位1的个数为奇数或偶数。
- 偶校验码中第j位p_j的计算公式

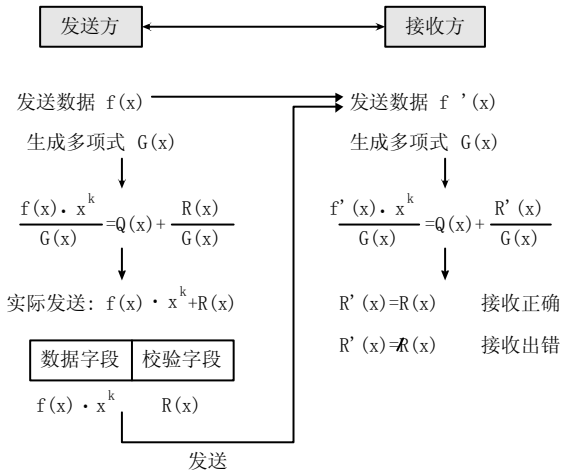
$$p_j = b_{0j} \oplus b_{1j} \oplus b_{2j} \oplus \ldots\ldots \oplus b_{nj}$$

- 奇校验码中第j位p_j的计算公式

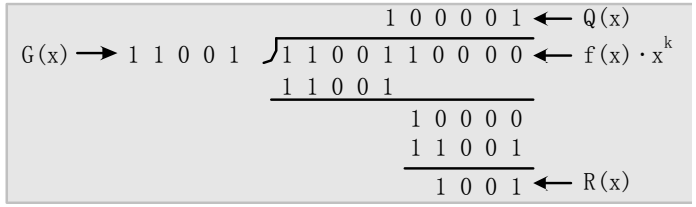
$$p_j = \sim (b_{0j} \oplus b_{1j} \oplus b_{2j} \oplus \ldots\ldots \oplus b_{nj})$$

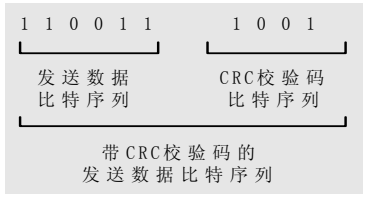


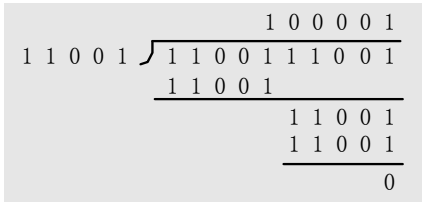
循环冗余校验码



循环冗余校验码举例







停止等待协议BSC

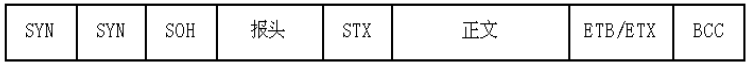
- BSC：一种典型的面向字符型停止等待协议
- BSC：使用ASCII码中的10个控制字符完成通信控制功能
- BSC：规定了数据报文、控制报文的格式
- BSC：规定了协议的操作过程
- 特点：规程简单、容易实现，比较适宜于在中低速网络中使用

控制字符

符号	名称	编码	功能说明
SOH	序始	01H	数据报文中报头的开始
STX	文始	02H	数据报文中正文的开始
ETX	文终	03H	数据报文中正文的结束
EOT	送毕	04H	传输结束
ENQ	询问	05H	询问对方并请求对方予以响应
ACK	正确认	06H	接收方正确接收数据报文后的响应
NAK	负确认	15H	接收方接收数据报文错误的响应
DLE	转义	10H	修改紧随其后字符的语义
SYN	同步	16H	收发双方的字符同步
ETB	组终	17H	成组传输时的组结束标记

报文格式

- 数据报文



- 常用的控制报文
  - 确认: SYN SYN 0/1 ACK
  - 否认: SYN SYN NAK
  - 询问: SYN SYN ENQ
  - 传输结束: SYN SYN EOT

透明数据传输

- 为什么提出透明数据传输?
  - BSC中的数据以字符为单位
  - 数据字符与控制字符有可能相同
- 解决方法：
  - 转义字符DLE
  - 如果数据字符与控制字符相同，则在数据字符前增加DLE
  - DLE也是控制字符，正文中出现DLE时也需要增加DLE

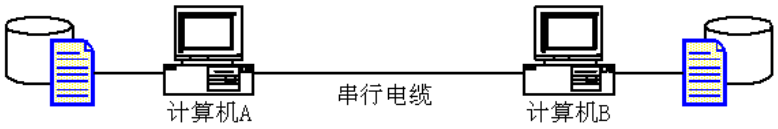
BSC协议的执行过程

收发双方的交互过程	含义	阶段
S：ENQ	“建立链路好吗？”	
R：NAK	“抱歉，我还没准备好”	链路建立
S：ENQ	“建立链路好吗？”	
R：ACK 0	“好的，请发送编号为 0 的数据报。”	
S：数据报 0	“发送编号为 0 的数据报。”	
R：ACK 1	“编号为 0 的数据报接收正确，请发送编号为 1 的数据报。”	
S：数据报 1	“发送编号为 1 的数据报。”	
R：NAK	“抱歉，数据接收错误。请重新发送刚才发送的数据报。”	数据传输
S：数据报 1	“发送编号为 1 的数据报”	
R：ACK 0	“编号为 1 的数据报接收正确，请发送编号为 0 的数据报。”	
S：……	……	
R：……	……	
S：EOT	“传输完毕，可以结束通信了。”	链路拆除
R：EOT	“知道了。”	

注：S：发送方，R：接收方

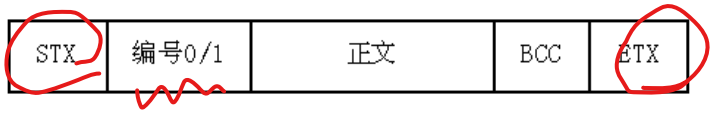
停止等待协议编程实验

- 在异步串行口上实现停等协议，实现文件的可靠传输

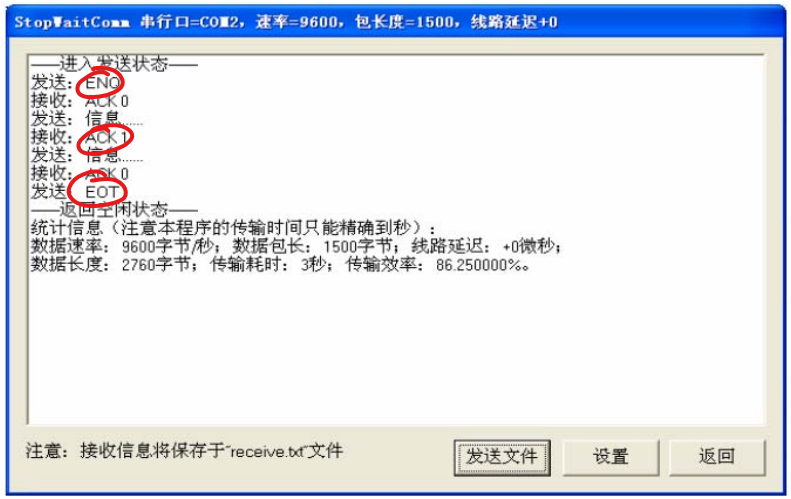


简化的停止等待协议

- 目的：在不影响停等协议基本思想的基础上简化编程和实验过程
- 简化后使用的控制字符：STX、ETX、EOT、ENQ、ACK和DLE
- 简化后的数据报文格式



界面示意图



选择发送和接收文件

```
explicit CFileDialog(
       BOOL bOpenFileDialog,
       LPCTSTR lpszDefExt = NULL,
       LPCTSTR lpszFileName = NULL,
       DWORD dwFlags = OFN_HIDEREADONLY | OFN_OVERWRITEPROMPT,
       LPCTSTR lpszFilter = NULL,
       CWnd* pParentWnd = NULL,
       DWORD dwSize = 0);
```

选择发送和接收文件示例

```
CString FileName;
CFileDialog GetFileName(TRUE,
              NULL,
              NULL,
              OFN_HIDEREADONLY,
              "文本文件(*.txt)|*.txt||",
              NULL,
              0);
if (GetFileName.DoModal()==IDOK)
              FileName=GetFileName.GetPathName();
```

磁盘文件操作

```
virtual BOOL Open(
              LPCTSTR lpszFileName,
              UINT nOpenFlags,
              CFileException* pError = NULL );
virtual UINT Read(
              void* lpBuf,
              UINT nCount );

virtual void Write(
              const void* lpBuf,
              UINT nCount );
virtual void Close( );
```

磁盘文件操作示例

```
CFile   rwFile;                 //需要读取的文件
char    Buf[BUFFER_SIZE];       //读取和写入缓冲区
int     len,Len;
if (!rwFile.Open(FileName,
        CFile::modeRead| CFile::modeWrite| CFile::typeBinary,
        NULL))                  //打开文件
{
        ……                      //错误处理
}
……
rwFile.Write(Buf, len);         //将缓冲区Buf中len个字节写入文件
……
len=rwFile.Read(Buf,Len);       //从文件中读取Len个字节的数据放入Buf中
                                //返回的len为实际读到的字节数
RecvFile.Close ();              //关闭文件，文件操作结束
```

练习和思考

- 通过改变传输速率、数据包长度、模拟长线路等操作定性地观察停止等待协议的效率
- 简化的停止等待协议使用奇偶校验码对传输的正文信息进行校验。奇偶校验实现简单但检错率不高。请查阅相关资料，将本实验的奇偶校验改为循环冗余校验，以提高检错率