【信息科学与控制工程】

基于 MFC 的纯方位定位系统串口通讯软件设计

胡科强,王小宁

(海军 91388 部队 ,广东 湛江　524002)

摘要:针对定位系统扩频通信的功能需求和上位机、下位机的硬件特点,在通信程序中使用多线程技术,以层次化结构完成了数据处理芯片和 PC 端的串行口通信程序的设计,解决了通信过程中常见的线程阻塞和死机现象,突出了系统的构造性和组合性;实际应用证明:系统通信实时性好,纠错能力强,可靠性高,满足对水下定位跟踪数据传输过程的模拟要求。

关键词:水下定位;多线程;串口通信;数据传输

中图分类号:TP391　　**文献标识码**:A　　**文章编号**:1006 - 0707(2012)08 - 0102 - 03

串口通讯是计算机和外设进行通讯,获取外设采集到的监测数据的重要手段。基于 GPS 和矢量水听器技术的基阵式水下定位系统来实现对水下目标定位[1]。其独特性在于以较少水听器对目标快速被动定位,且具有良好可操作性。为保证定位基阵传回的加入时间戳的目标方位信息、姿态修正信息和 GPS 定位信息通过串口和主控计算机之间进行高效、可靠、准确的传输,数据通讯软件的设计显得尤为重要[2]。

1　系统总体实现

基阵式水下定位系统工作原理示意图如图 1 所示。主控计算机要同时采集多路(GPS 接收机、矢量水听器、电罗经)经串口输出的原始数据,而主控计算机数据预处理、定位解算、数据关联占用 CPU 的时间较多。需要采用合理的软件结构防止串口缓冲区数据溢出、丢数、死机现象。提高软件的稳定性、可靠性和实时性。线程是操作系统的基本调度单元,可将某个工作模块置于独立的线程中。合理采用多线程技术可以有效地加快程序的反应速度、提高执行效率。如果不采用多线程技术,只能首先接收、处理数据,再进行解算,CPU 将浪费时间用来等待数据,不能保证实时性。因此软件中目标解算和信息显示做为主控制线程(监控和界面线程),同时创建一个辅助线程负责数据接收,这样,能最大限度的保证系统工作的实时性。

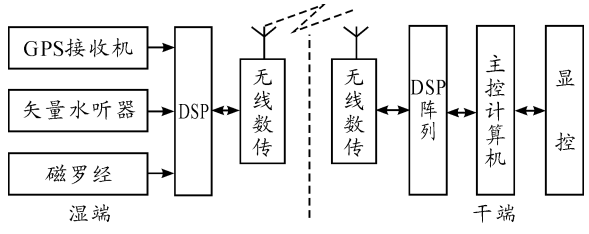

图 1　系统工作原理示意图

2　各模块设计

2.1　串口通信用户层协议

浮标端实时接收的来自矢量水听器、GPS 和罗经信息,需要按照统一的格式从串口发送数据同时将接收的数据中将需要的信息提取出来,在此采用 NMEA - 0183 无线通信协议,该协议包含了帧头、帧尾、帧内数据、校验、换行,帧内数据之间用逗号分隔[3]。将浮标各部分数据通过组帧程序打包成该协议格式后采用十六进制传送出去。

基站端从串口接收数据并将其放置于缓存,在没有进一步处理前在缓存中是一串字节流,在通过解帧程序将各字段的信息提取出来。制定通信协议时须考虑通信系统的数据吞吐量。终端站实时传送的参数包括 GPS 接收机给出的终端站自身的定位信息:经度、纬度、导航状态;状态参数包括通信状态,数据存储器被占用的情况;水声处理模块提供的目标声学数据,这部分数据占用了大部分数据空间这些数据将被打包以帧的形式传送给控制器。考虑到目标个数及脉冲数目的不确定性,用 B 表示单个浮标基元在一个通信周期内需要传送的 bit 数,在每同步周期的数据包中加入以下信息:

$B \geqslant$ 数据帧同步头 + 数据总长度 + 浮标 ID 号 + 浮标位置数据 + 浮标状态数据 + (目标 ID 号 + 目标方位数据 + 测试状态数据) × 目标个数。

其中:数据帧同步头:16 bit;数据总长度:8 bit;浮标 ID 号:2 bit;浮标位置数据:经度 32 bit + 纬度 32 bit = 64 bit;浮标状态数据:1 bit(GPS 定位状态);目标 ID 号:3 bit(目标最大跟踪数:8 个);目标方位数据:16bit;测试状态数据:4 bit(表明浮标基元对目标的检测能力)。

这样,以上数据均考虑了系统进一步扩展的要求,以最大值计算,假设浮标基元数量 M 为 3 个,通信接力 N 为 3 次,

收稿日期:2012 - 05 - 10

作者简介:胡科强(1986—),男,硕士,主要从事水声信号处理研究。

每帧声学数据最多为 $B \times M \times N = 2\ 475$ bps。选用的 GD230 电台数据,其传输码速率在9 600 ~ 19 200 bps,发射功率为 0.1 ~8 W,误码率小于 10^{-5},在通信码速率、通信距离、功耗以及抗干扰性等方面均可以满足水下定位跟踪系统的需要。

对数据帧边接收边处理,当串口缓冲区中的字符触发串口通信事件,驱动串口通信处理函数,对接收到的数据帧进行处理如图 2 所示。

采用时分多址协议,实时工作状态下采用基站轮询,集中控制动态分配时隙的方式,利用简化的停等 ARQ 方式实现差错控制:

1) 基站向浮标发完数据帧后,启动定时器,若在设定的时间内未收到浮标传回数据帧,则将该浮标存入误传表;

2) 继续呼叫另一浮标;

3) 从误传表中取出浮标 MAC 地址并呼叫,令其重传上一数据帧,过程同 1、2,将错误的浮标地址存入误传表;

4) 重复 1) -3),最多 3 次。

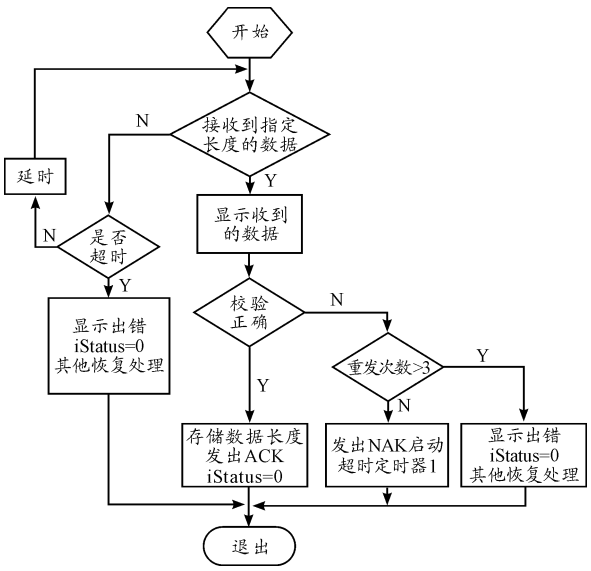

图 2 接收数据处理协议

2.2 基于 API 的多线程串口通信

利用 Win32 操作系统具有的多线程、消息响应和通信事件驱动等机制,采用 Win32 中对串口进行异步操作的 API 函数,编写实时高效的串行通信程序[4]。

主控软件必须具有多任务并行处理能力,典型情况为当系统进入目标跟踪状态时,主控软件要同时处理串口数据,实时解算目标位置,接受指挥人员的控制指令。为了使系统软件能在 Windows 环境下多任务并行处理,采用多线程来实现多任务控制[5-6]。

MFC 应用程序的线程由 CWinThread 对象表示,分为主线程如图 3 所示和工作者线程如图 4 所示,前者能提供界面和用户交互,用于处理用户输入并对各事件和消息进行响应;后者主要用于处理程序的后台任务,即负责实时数据的不间断接收。

使用 Win32API 函数进行串口通信编程,调用 AfxBeginThread( ) 自动创建 CWinThread 对象,开始一个进程,VC ++ 通过事件对象来实现线程同步。使用 ClearCommError 函数查询输入缓冲区是否有字符,如果有,发送消息通知接受处理函数;如果没有,则调用 WaitCommEvent 函数监视 EV_RXCHAR 通信事件,执行 I/O 重叠操作,随即调用 GetOverlappedResult 函数无限等待通信事件,直到 EV_RXCHAR 事件发生,则结束等待。如果只用 ClearCommError 函数,工作者线程将不断耗费 CPU 时间来进行查询,效率较低;如果只用 WaitCommEvent 函数监视,对缓冲区已有字符将不会产生 EV_RXCHAR 事件,易造成数据延误和丢失。两方法联合使用兼顾效率和可靠性。

```
while (pDoc->m_hConnected) {
//当串口已连接,执行下面程序
PurgeComm(pDoc->m_hComm, PURGE_RXCLEAR|PURGE_TXCLEAR|PURGE_RXABORT|PURGE_TXABORT);
for(;;){//无限循环,直到关闭线程
bResult = WaitCommEvent(pDoc->m_hCom, &Event, &port->m_ov);
if(! bResult) {
switch (GetLastError()){
case ERROR_IO_PENDING{
break ;}
default ;{
break ;} } }
else {
bResult = ClearCommError(pDoc->m_hCom, &dwErrorFlag,&ComStat);
if(ComStat.cbInQue==0) //缓冲区无数据
continue ;}
//无限等待,阻碍该线程,直到等待的事件到来。
//等待的事件有:
m_hEventArray[0] = m_ShutdownEvent;
m_hEventArray[1] = m_ov.hEvent;
Event = WaitForSingleObject(pDoc->m_hPostMsgEvent, INFINITE);
ResetEvent(pDoc->m_hPostMsgEvent);//通知视图
switch (Event){
case WAIT_OBJECT_0; {//关闭串口事件
AfxEndThread(100);
break ;}
case WAIT_OBJECT_1; { //读事件
GetCommMask(port->m_hCom,&CommEvent);
if (CommEvent&EV_RXCHAR) {
//接收到数据
port->ReadFromPort(); //读串口
CMyProtocol::run();//通信协议 PostMessage(pDoc->m_hTermWnd,CM_RECEIVE,0,(LPARAM)EV_RXCHAR);
//向主窗体发送处理串口事件的命令
break ;}}} //close forever loop
closeHandle(m_ov.hEvent);
return 0;} }
```

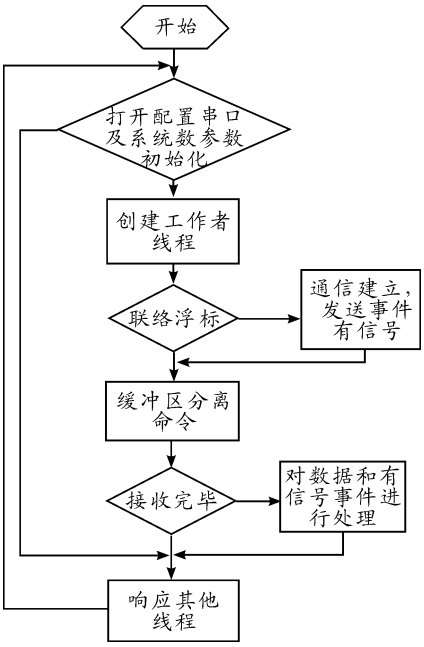

图3 主程序流程

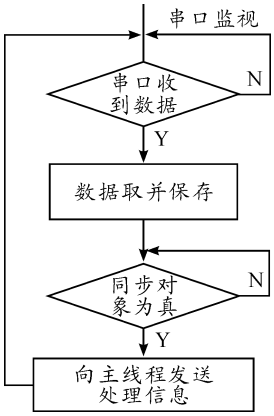

图4 工作者线程流程

2.3 串口通信类设计

串口类封装了系统对串口的所有公共操作。

```
class CComm{
private
CCirQueue m_CirQueue;  //循环队列对象,用来缓存接受的数据。
HANDLE m_hShutdownEvent;  //关闭事件句柄,用于在关闭程序时关闭线程。
CWinThread * m_Thread;  //线程指针,指向监视该串口的线程。
CWnd *pWnd; //该串口对应的窗体指针
public
BOOL InitPort( CWnd * pWnd,int PortID,int BandRate); //串口初始化函数
void ReadFromPort( ); //串口读函数
void WriteToPort( unsigned char *ch,int num); //串口写函数
void ProcessCommand( ); //数据处理函数
static UNIT WatchComm( LPVOID pParam); //串口监视
public
CComm( ); //构造函数
virtual ~CComm( );}; //析构函数
```

环形队列类 CCirQue:

环形缓冲区需要有一个读指针(位置)和写指针(位置)。其中读指针由读数据接口来移动,写指针由写数据接口来移动。在读出和写入数据时,要分别考虑 读指针超前与滞后写指针 2 种情况。该类封装了一个环形队列,定义一个缓冲区(该缓冲区把该数组看作是一个环,支持在一块固定的数组上的无限次读和写,数组的大小不会自动变化)用于暂时存取主控计算机通过串口采集到的数据。又定义了对该缓冲区的一些操作,如 GetCount( )(获得队列中的元素个数),IsFull( )判断队列是否满。

```
Class CCirQue{
public
bool IsEmpty( ); //判断队列是否空
unsigned char GetQueue( ); //获得队首元素
unsigned char DeQueue( ); //提取队首元素
void EnQueue( unsigned char ch); //向队列中插入元素
CCirQue( );
Virtual ~ CCirQue( );
CCritialSection m_buffCriSect; //循环队列临界区
};
```

在对缓冲区的所有操作都用到临界区 CCritialSection 类和 Clock 类来保证串口通信同步的要求。

3 结束语

结合基阵式水下定位系统数据采集的需求,对多线程、串口通信、异步 I/O、通信用户协议技术原理和实现方法进行分析探讨,并为通信系统封装了串口类和环形缓冲类,它们功能强大,具有较好的移植性和控制灵活性。实际应用表明多线程技术应用于串口通信,可以提高程序的执行效率,使程序并行工作。用户在进行费时的 I/O 操作、实时解算的同时也可以进行用户需求响应。

参考文献:

[1] BECHAZ C, THOMAS H. GIB portable tracking systems: the underwater use of GPS[J]. Hydro International, 2000(8):1-9.

[2] 顾晓东. 基于矢量水听器的水下目标被动跟踪研究[D]. 武汉:海军工程大学,2009.

[3] 龚建伟,熊光明. Visual C++/Turbo C 串口通信编程实践[M]. 北京:电子工业出版社,2008.

[4] 候俊杰. 深入浅出 MFC [M]. 2 版. 武汉:华中科技大学出版社,2010.

[5] LIU S, LIU J. The Serial Communication Program using Windows API[J]. Computer Applications, 2002,20(2):43-44.

[6] 晏春海,田蔚风,王俊璞,等. 多线程技术在分时串口通信中的应用[J]. 仪表技术与传感器,2004(5):15-17.

(责任编辑 周江川)