第一章　一般信息

1·1　手册概述

本手册介绍用于传送 MULTIBUS® Ⅱ报文的 MULTIBUS Ⅱ传输应用界面。应用程序利用传输应用界面(TAI)来与远程应用程序使用 MULTIBUS Ⅱ传输协议进行通讯。

注意：本手册仅包含传输应用界面的信息，而不包含传输核心界面和互连空间的信息。

1·1·1　读者

本手册是为熟悉和了解系统 V/386 操作系统、MULTIBUS Ⅱ传输协议及 MULTIBUS Ⅱ专用术语的软件工程师准备的。附录 E"相关出版物"列出了包含传输应用界面相关信息的手册。为了能全面理解传输应用界面，读者需要熟悉《MULTIBUS Ⅱ传输协议说明和设计者指南》一书，它提供了 MULTIBUS Ⅱ传输操作的背景和理论基础。

1·1·2　手册的组织

本手册包含四章正文、一个附录和一个手册页，组织如下：

第一章　一般信息

包括了读者须知，手册组织和注释约定等信息。

第二章　传输应用界面(TAI)

介绍 MULTIBUS Ⅱ传输应用程序界面，并简要复习 MULTIBUS Ⅱ报文传送。本章还提供了一个综合字典，其中给出了报文传递、传输应用界面和传输核心界面等方面的术语。

第三章　基本操作

讨论基本传输应用界面。

第四章　示例程序

给出一个传输应用界面程序示例。

附录 A　传输应用界面结构

详细介绍传输应用界面程序的结构。

手册页

1·1·3　阅读帮助

手册中某些段落可能被着重说明，如：

注意：注释段用于提供特殊信息，例如，通过注释段来强调一个推荐的步骤序列。

1·1·4 注释约定

下面的注释约定贯穿于整个手册之中。

`input` 用户输入，如命令、选项、参数、变量、目录名和文件名以黑体出现。

output 系统输出，如提示符，命令响应及程序示例，以常规字体出现。

variable 变量名，必须被赋值(如文件名)，以斜体字出现。

command(#) *command*(#tai)

引用一命令、库调用或系统调用。数码的用法遵从AT&T手册的约定。命令名后可跟随

- 数码，表示参见AT&T UNIX系统V/386程序员参考手册中相应的章节。
- 括入括弧中的数码和字符串"tai"，表示参见本手册中的其它参考部分。

注意：本手册中涉及的所有命令均可以在附录B、C、D中找到。

1·2 其它书籍

下面所列出的书籍中，提供了关于系统V/386操作系统和传输的附加信息。

The C Programmer's Handbook, Bell Labs/M·I·Bolsky
The UNIX System User's Handbook, Bell Labs/M·I·Bolsky
The Vi User's Handbook, Bell Labs/M·I·Bolsky
UNIX System Software Readings, AT&T UNIX PACIFIC
UNIX System Readings and Applications, Volume I, Bell Labs
UNIX System Readings and Applications, Volume II, Bell Labs
UNIX System V/386 Manual Set, AT&T (published by Prentice Hall)

Intel® System V/386 MULTIBUS® Installation and Configuration
Intel® System V/386 MULTIBUS® Reference Manual
MULTIBUS® II· Transport Protocol Specification and Designer's Guide
Intel iRMK I·2 Reference Examples
Intel iRMX® III R1·1 Library

第二章　传输应用界面(TAI)

2·1　概述

本章介绍基于 MULTIBUS Ⅱ的 Intel 系统 V/386 的传输应用程序界面。在 Intel 系统 V/386 中运行的应用程序使用传输应用界面(TAI)与其它 MULTIBUS Ⅱ传输端进行通讯。本章还简单地给出报文传递的定义，并列出相应的手册以使读者获得更进一步的信息。

2·1·1　传输应用界面

应用程序可以使用 MULTIBUS Ⅱ传输应用界面与遵从 MULTIBUS Ⅱ传输协议的运行于其它宿主机、代理或 MULTIBUS Ⅱ处理器上的应用程序和设备驱动程序进行通讯。

下面列举一些能与 TAI 的应用程序进行通讯的其它实体:

- 运行于另一个UNIX处理器上的TAI应用程序。
- 运行于另一个UNIX处理器上的使用传输核心界面的设备驱动程序。
- 使用MULTIBUS Ⅱ传输协议的RMK™应用程序。
- 使用MULTIBUS Ⅱ传输协议的RMX®应用程序。

注意: 应用程序不能使用TAI与运行于同一处理器上的另一个软件进行通讯(亦即，没有回路或短路能力)。

2·1·2　简介什么是报文传递

MULTIBUS Ⅱ报文传递是一种在处理器间高速传递中断和数据的方法。

MULTIBUS Ⅱ报文传递定义了请求与非请求两种报文形式。

《MULTIBUS ®Ⅱ传输协议及设计者指南》手册中给出了报文传递的详细内容。使用传输应用界面之前，应彻底理解该手册的有关内容。如果需要复习报文传递的内容时，应该阅读本手册的二至四章。

2·1·3　传输的概念

一个能够传递报文的 MULTIBUS 板(代理)称作一个宿主机。宿主机用一个称作宿主标识(Host-ID)的 16 比位整数来标识。宿主机上的存取点叫做端口，并用一 16 位的整数来标识，这个数被称做端口标识(Port-ID)。宿主标识与端口标识对(host-ID,port-ID)构成了 MULTIBUSⅡ的全传输地址。这个地址被称为一个插座。在 MULTIBUSⅡ系统中，一个插座唯一地标识宿主机上运行的一个软件端口。

2·1·3·1 委托方和服务方

MULTIBUSⅡ传输应用界面允许程序发出一请求报文并保证收到一应答报文。发出这个请求的程序称作委托方。而回送应答的程序被称作服务方。一对请求-应答报文称作一个事务。

尽管任一单个事务都有一个委托方和一个服务方，但 MULTIBUSⅡ传输在两插座间却是完全双向的。这意味着一个事务可以由任一端口发起。

当一个请求报文发出后，可以期望得到一个应答(另一个报文或是一错误指示)。一个报文若不属于事务中的一部分，则称做非事务报文。假如发出了一个非事务报文将没有任何完成指示或专投递指示返回。

2·1·3·2 分段

由于缓冲区的限制，有时不能一次接收整个请求报文。当这种情况发生时，可发送一报文通知服务方。服务方可以选择分段方式接收报文。同样地，假如服务方没有足够的空间来发送一个完整的应答报文时，它也可以采用分段方式发送应答报文。分段仅允许出现在一事务中，而且总是由特定事务中的服务方所控制。事务中的委托方不需要分段，这是因为由它决定向服务方发出多少数据或从服务方接收多少数据。

第四章给出了一个使用分段技术的示例。关于分段的详尽讨论，请参见《MULTIBUS Ⅱ传输协议指南及设计者指南》手册。

2·1·3·3 流控

流控是用于调节任一端口上排队的报文数量的机制。设置流控有两条理由:

- 防止某一单个端口用尽所有的全局缓冲区资源。
- 防止某一端口连续不断地处理突发性报文，从而挤占其它端口的处理能力。

关于流控的更详尽的介绍列于附录 A 中。

2·1·3·4 超时

超时是一种使委托方或服务方能确定另一方是否已经出错的特性。当一个端口被打开时，服务方就指定了一个超时值(或采用默认值)，使得下层的传输应用界面能确定是否有响应到达。然后将所确定的结果送回给服务方。

传输应用界面的超时特性仅在需要应答报文的例程中提供。如 `mb2s_sendrsvp` 和 `mb2a_getreqfrag`。关于超时的更详尽的介绍列于附录 A 中。

2·1·3·5 同步和异步界面

传输应用界面提供同步和异步两种操作方式。同步方式的最大优点是容易使用。异步方式的主要优点是具有较好的灵活性和性能。依据对下面二个问题的回答，决定选择哪一

种方式。

- 性能是否是最主要的考虑因素?
- 是否需要同时进行多道通讯?
- 应用程序进行通讯的同时是否还要处理其它任务?

假如对上述三个问题的回答均为"否",则应选择同步方式,原因是编写同步方式下的程序要比编写异步方式下的程序容易。然而,假如对上述的问题之一或更多的回答为"是",就应该考虑采用异步方式。

一旦某一端口以同步或异步方式打开,则对该端口的其它操作也必须采用相同的方式。这意味着,打开端口时就必须决定使用何种方式。

注意:所有同步方式调用均以**mb2s_**开头,所有异步方式调用均以**mb2a_**开头。

2·1·4 传输字典

本节定义报文传递中,传输应用界面和传输核心界面所用到的术语。它可以作为快速参考用。

代理	MULTIBUS Ⅱ控制器或处理器板。
缓冲区准许	一种用来准许缓冲区请求并发起请求性数据传输的MULTIBUSⅡ报文。
缓冲区拒绝	一种用来拒绝缓冲区请求和拒绝请求性数据传输请求的MULTIBUSⅡ报文。
缓冲区请求	一种用来为请求性数据传输申请缓冲区的MULTIBUSⅡ报文。
委托方/服务方模式	一种计算或通讯模式。其中一个叫做委托方的进程发出请求给另一个叫做服务方的进程,委托方期望服务方响应它的请求。典型的响应是与请求有关的数据或状态信息。
无连接	一种不需要建立逻辑连接的传输存取点的属性。无连接通讯时常是传输服务中的一种特性。通讯存取点间无须在开始数据传输之前进行逻辑连接的协商。
控制报文	一种短的且定长的报文,其作用与MULTIBUS Ⅱ非请求性报文相同。
控制部分	数据报文中短的且定长的部分,其典型作用是作为协议信息使用。
数据链路协议	主机-主机间报文传送协议。
数据报文	MULTIBUSⅡ请求性报文,由控制部分和数据部分组成。
数据部分	请求性报文中由所要传送的数据部分。
EOT	传输服务接收分段应答报文直到收到标有EOT的分段报文为止。此时,认为该事务已完成。

存在确认	对存在报文的确认报文。确认报文由宿主机标识和发送者的替代号两部分组成。
存在报文	用于向系统其余部分宣布某宿主机存在的广播报文。存在报文由宿主机标识和发送者替代号两部分组成。
分段缓冲区	用于接收特定请求报文分段的缓冲区。参见接收分段报文。
宿主(机)	即控制器或处理器板(代理)，能够传送MULTIBUS II报文。
宿主(机)标识	用来唯一标识宿主机的两字节整数。(详见附录A。)
替代号	用来区别宿主机重置的整数。
MPC	报文传送协处理器。一种用来提供基本总线接口和报文传送服务的专门硬件。
报文标识	一个字节长的MULTIBUS II源MPC或目的MPC的地址。
下一分段报文	由传输服务产生的缓冲区请求报文。它用来传输请求报文的下一个分段报文。
端口	用来引用宿主机MULTIBUS II端点的名字。所有的MULTIBUS II报文均由端口发出或接收。
端口标识	用来唯一标识某一端口的两字节整数。在使用某一端口发送或接收报文时，必须事先为该端口分配一个端口标识。零(0)是端口标识的无效值(详见附录A)。
协议	协议(也称作服务)，是由一些规则和约定组成的集合，用来管理两个通讯存取点间的通讯。
协议标识	用于支持传输服务中，多种协议的一字节长度的标识。
接收	一种用来接收报文的用户操作名。
接收分段	用来接收请求报文中，某一部分的用户操作名。
接收处理	用来异步处理所接收到的非请求性报文及请求性完成事件的程序。
接收应答	用来接收应答报文的用户操作名。
请求-应答事务	一种基于报文的通讯方式。用于对传输用户发起的请求作出响应。它以请求报文开始，以应答报文结束。
rsvp缓冲区	传输用户为请求性应答数据提供的缓冲区。
发送分段	由传输服务产生的一个非请求性报文，它用来指示用户已执行分段接收操作并且将要发送下一个分段请求报文。
发送应答	一种用来产生传输应答报文的用户操作名。
发送rsvp	一种用来产生传输请求报文的用户操作名。
插座	宿主机标识：端口标识(host ID: port ID)对，用以通过数字化方式命名一个端口。
请求性报文	经由MULTIBUS II报文空间的数据传输需要进行缓冲区协商。

事　务　在MULTIBUSⅡ传输中，委托方与服务方间请求-应答报文对的交换。

事务标识　用来唯一标识某插座上发起的事务的整数。所有与某事务相联系的事务性报文或分段报文都用此事务的事务标识来标识。事务标识0为无效标识，它隐含地表示该报文为非事务性报文。(详见附录A。)

非请求性报文　长度至多为32字节的MULTIBUSⅡ报文，它不可预测地到达宿主机，并用于协商请求性报文传递。

第三章　基本操作

3·1　概述

本章介绍基本传输应用界面的概念。同时，给出一些应用例程，及编写应用例程的提示和建议。尽管本章中使用的所有例程是同步方式下的调用，但其概念也适用于异步方式。

3·1·1　编写第一个应用程序

一些应用程序需要标识程序在上面执行的宿主机。函数 mb2_gethostid( )(如下所示)用来为应用程序提供宿主机的信息。它是最简单的应用程序，不需要发送或接收报文。它的全部功能就是打印出该程序执行所在宿主机的宿主标识。

```
#include <sys/types.h>
#include <sys/mb2talusr.h>
unsigned short local_hostid;
main()
{
    local_hostid=mb2_gethostid();
    printf("local hostid=%d\n",local_hostid);
}
```

注意：mb2_gethostid是一个不区分同步或异步方式的传输应用界面函数。

为了编译这个程序，用户必须链接传输应用界面库。例如使用如下命令：

```
cc -o example example.c - lmb2
```

为了能够接收和发送报文，还必须设计两个以并行方式通信的应用程序以便进行所期望的操作。本手册中大多数的例子均包括本地和远程两个应用程序。

这些应用程序必须在 MULTIBUSⅡ系统中两个不同的代理上运行，并且使用应用接口程序在系统总线上进行通讯。

3·1·2　发送/接收报文

在发送或接收报文之前，用户必须拥有一个与 MULTIBUSⅡ端点一致的文件描述字。在同步发送示例(图 3-1)中，发送报文前必须调用函数 mb2s_openport。

mb2s_openport 需要两个变量，第一个变量是所要打开的端口标识，第二个变量为指向 mb2s_opts 结构的指针。如果端口标识赋值为 0，则传输服务自动为端口分配一个相应的端口标识。

要发送一个报文，还必须提供目的宿主标识(host ID)和端口标识(port ID)。换句话

说，发送方必须知道接收方的"地址"以便向其发送报文。这就需要发送方与接收方事先协商以便接收方在正确的端口上"侦听"。

3·1·2·1 发送报文

mb2s_send 例程用于发送非事务性报文。它既可以用来发送请求性报文，也可以用来发送非请求性报文。当无需得到接收方应答时，就应使用该例程。于是当报文没有到达指定的目的时，便没有任何错误信息返回。假如希望得到完成或错误指示，就必须使用 mb2s_sendrsvp 例程。

传输应用界面不保证 mb2s_send 请求的提交次序。假如一个请求性报文(数据部分非空)后跟随着一个非请求性报文，用 mb2s_send 发向同一目的地址，非请求性报文可能会先于请求性报文到达，即报文的顺序被颠倒了。

3·1·3 建立报文

在报文被发送前，必须事先建立报文。建立报文的第一步是填写与数据一起发送的用户控制部分，然后初始化 mb2s_buf 结构(将要传递给 mb2s_send 例程)。将 mb2s_buf 结构中的 buf 字段置成存放所要发送报文的内存区地址，同时还要设置 len 字段。当建立向外发送的报文时，maxlen 字段可以忽略。

请记住，报文可以是请求性的，也可以是非请求性的。请求性报文控制部分最大长度可为 SOL_LENGTH(16)给出的数据字节个数。而非请求性报文的控制部分最大长度为 USOL LENGTH(20)给出的数据字节个数。

在请求性报文情况下，还必须为报文的数据部分作出说明和初始化另外一个 mb2s_buf 结构。

3·1·4 发送报文

建立报文之后，就可以发送报文了。调用 mb2s_send 时需要四个变量，它们是：

1. MULTIBUSⅡ传输文件描述字(来自 openport调用)。
2. 指向socket ID 结构的指针，该结构中包含目的地址。
3. 指向控制部分的mb2_buf结构的指针。
4. 指向数据部分的另一个mb2 buf结构的指针。

变量#3、变量#4 或者两者均可为空。

3·1·5 发送示例

图 3-1 给出一个同步方式发送报文的简单例子。在这个例子中，将要发送一个非请求性报文。由于报文中没有数据部分，所以 mb2s send 的第四个变元被置为空。另一种说明报文无数据部分的办法是传递一个指向 mb2_buf 结构(其中的 len 字段被置成零)的指针。

在图 3-1 中用户控制部分被置为 0xAA。

注意，在本例中，每一 MULTIBUSⅡ例程调用，都对其返回值进行检查以使编写一个易于调试的应用程序。这一点对于尽早发现程序的错误是十分重要的。

图 3-1　同步发送报文示例

```
#include<stdio.h>
#include<sys/types.h>
include<sys/mb2tansur.h>
#define RECEIVE_PORT 8999
#define RECEIVE_HOST 5
#define PATTERN 0xAA
mb2_buf ctlbuf;
mb2_socket socketid;
char char_buf [UNSOL_LENGTH];
main ()
{
    int fd;
    int i;
    socketid.hostid = RECEIVE_HOST;
    socketid.portid = RECEIVE_PORT;
    for(i = 0; i<UNSOL_LENGTH; ++i) {
            char_buf[i] = PATTERN;
    }
    ctlbuf.buf = &char_buf[0];
    ctlbuf.len = UNSOL_LENGTH;
    fd = mb2s_openport (0, NULL);
    if (fd == -1) {
            perror ("mb2s_openport");
            exit(-1);
    }
    status = mb2s_send (fd, &socketid, &ctlbuf, NULL);
    if (status == -1) {
            perror ("mb2s_send");
            exit(-1);
    }
    printf("Sent Transactionless Message 0);
}
```

3.1.6　接收报文

与发送报文相对应的是另一个应用程序，它接收报文，并显示报文的相关信息及其内容。

3·1·6·1 接收报文

接收应用程序要做的第一件事是初始化一个用于接收报文的缓冲区。maxlen 字段应被初始化为缓冲区的长度，buf 字段应置为存放接收报文的缓冲区地址。

mb2s_receive 例程需要 4 个变元。第一个变元为与相应 MULTIBUS Ⅱ传输端点相关的文件描述字。端口标识应与欲在该端点接收报文的目的端口标识相匹配。第二个变元是指向 mb2_buf 结构的指针。该结构中将填入与报文相关的信息，包括 socket ID 字段及接收报文类型字段。

最后两个变元分别为指向存放报文控制部分和数据部分内容的缓冲区指针。每个缓冲区结构中的 maxlen 和 buf 两个字段应分别初始化成相应接收报文的长度及存放区的地址。

当从 mb2s_receive 返回时，缓冲区结构中的 len 字段被置成所接收报文控制部分和数据部分的实际字节数。因此，对于数据部分长度为零的接收报文，其相应缓冲区结构中的 len 字段被置成零。假如任一指针为空，则报文中相应的部分也被认为是空。如果此时接收到具有非空部分的报文，将返回一个错误。当以 mb2s_receive 成功地返回时，msginfo 结构中的相应字段被相应地填入内容。在 ctlbuf.buf 指针指向的区域中接收报文，而 ctlbuf.len 字段被置成实际接收的字节数。

3·1·7 合法报文类型

mb2s_receive 能接收两种事务性报文及两种非事务性报文，如下所列：

3·1·7·1 事务性报文

MB2_REQ_MSG　　请求报文
MB2_REQFRAG_MSG　需要分段的请求报文

3·1·7·2 非事务性报文

MB2_NTRAN_MSG　非事务性报文
MB2_BROCST_MSG　广播报文

注意：在异步方式下，mb2a_receive既能接收上述类型的报文，还能接收其它类型的报文，详见附录D。

3·1·8 接收示例

图 3-2 给出一示例应用程序，它接收上个示例应用程序所发送的报文。假定图 3-1 所给出的报文就是与本接收报文相关的发送报文。

由于该应用程序不期望在报文中收到任何数据部分，所以将指向报文数据部分的指针赋值为空。

图 3-2　同步接收报文示例

```
#include <stdio.h>
#include <sys/types.h>
#include <sys/mb2ialloc.h>
#define RECEIVE_PORT 6999
#define RECEIVE_HOST 5
#define PATTERN 0xAA
mb2_buf_ctlbuf;
mb2_msginfo msginfo;
char char_buf [UNSOL_LENGTH];
main ( )
{
    int fd;
    int i;
    ctlbuf.buf = &char_buf[0];
    ctlbuf.maxlen = UNSOL_LENGTH;
    fd = mb2s_openport(RECEIVE_PORT, NULL);
    if (fd == -1){
            perror ("mb2s_openport");
            exit(-1);
    }
    status = mb2s_recvmsg (fd, &msginfo, &ctlbuf, NULL);
    if(status == -1){
            perror ("mb2s_receive" );
            exit(-1);
    }
    switch(msginfo.msgtyp) {
    case MB2_NTRAN_MSG:
            printf("Received Transactionless Message0);
            break;
    default:
            printf ("Unexpected message type received0);
            exit(-1);
    }
    printf("Message Source Hostid = %d0,
                            msginfo.sockmtid.hostid);
    printf("Message Source Portid = %d0,
                            msginfo.sockmtid.portid);
    printf("Control length=%d0, ctlbuf.len);
    printf("Control message contents:" );
    for (i=0;i<ctlbuf.len;++i)
            printf("%x", ctlbuf.buf[i]);
    printf ("0);
    printf("Received Message Displayed0);
}
```

3.1.9　事务性报文

发起事务的代理称作委托方，委托方通过向一称作服务方的远地代理发送一请求报文

而起动一事务。服务方收到请求报文后，发回一个应答报文给委托方。

当委托方收到应答报文后，这一事件便结束了。mb2s_sendrsvp 例程就是用来发起上述事务的。一旦委托方起动该例程发送一个请求报文给远地宿主(服务方)，它就必须等待应答报文的到达。应答报文能被自动接收。用来接收应答报文的缓冲区也必须在调用该例程时提供。这意味着应答报文的最大长度应在事务开始前就已经确定。

应答报文还可能由于下列原因之一而不能到达：

- 服务方程序收到请求报文后垮掉。
- 服务方程序不能发出应答报文。
- 运行服务方程序的宿主机垮掉。

在上述任何一种情况下，传输服务都会以超时结束这一事务，mb2s_sendrsvp将返回−1并且errno被置成ETIME。应用程序可以通过在调用mb2s_sendrsvp时使用选项结构来控制超时值。等待send_rsvp应答报文的默认值为20秒。

注意：关于mb2a_sendrsvp的说明参见附录D。

3·1·9·1 作为事务请求报文发送控制报文

图 3–3 给出了一个委托方程序。它发送一控制报文作为事务请求报文，并期望收到一个包括长达 4K 字节长数据部分的应答报文。为了分配和提供一个 4K 字节的应答数据缓冲区，这里为应答报文定义了 4K 数据长度。应答报文的控制部分将返回到为它提供的缓冲区中。值得注意的是，当 mb2s_sendrsvp 返回−1，应用程序通过检查 errno 是否为 ETIME 的方法来测定是否未能收到应答报文。如果应答报文被成功地接收并存到为它提供的缓冲区中，则 mb2s_sendrsvp 返回 0。其它任何有关所接收应答报文的信息都在 msginfo 结构中返回。

图 3–3 委托方程序作为事务请求报文发送一控制报文

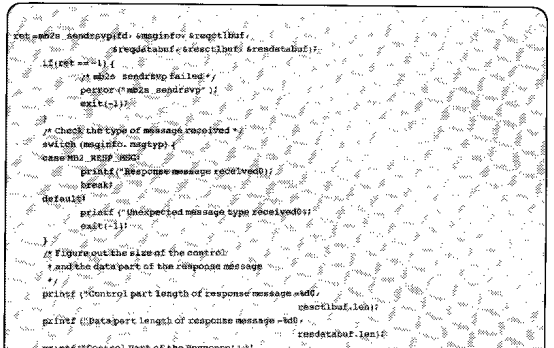
```
/*
 * Example program showing an application that
 * sends a 4k data message in the request of a transaction
 * and expects to receive up to 4k of data in the response
 */
#include <stdio.h>
#include <sys/types.h>
#include <sys/mb2s_user.h>
/* Maximum size of the data part of a message */
#define MAX_DATA_SIZE 0x1000
/* Port ID of port where to send the message */
```

(转下页)

```
#define EXAMPLE_PORT 0x400
/* Host ID of the destination */
#define DEST_HOST 5
/* Buffers for the request message */
char reqctlpart [SOL_LENGTH];
char reqdatapart [MAX_DATA_SIZE];
mb2_buf reqctlbuf = {SOL_LENGTH, 0, &reqctlpart[0]};
mb2_buf reqdatabuf = {MAX_DATA_SIZE, 0, &reqdatapart[0]};
/* Buffers for receiving the response message */
char rspctlpart [SOL_LENGTH];
char rspdatapart [MAX_DATA_SIZE];
mb2_buf rspctlbuf = {SOL_LENGTH, 0, &rspctlpart[0]};
mb2_buf rspdatabuf = {MAX_DATA_SIZE, 0, &rspdatapart[0]};
/* msginfo structure to store destination socket id
 * and to receive the message type etc when send_rcvp returns
 */
mb2_msginfo msginfo;
main( )
{
    int fd; /* file descriptor for the endpoint */
    int i;

    /* As before start each application by opening
     * a transport Endpoint. Use a dynamically allocated
     * port ID and use default endpoint values.
     */
    fd = mb2s_openport(0, NULL);
    if(fd == -1) {
            perror("mb2s_openport");
            exit (-1);
    }
    /* Set up the buffers for sending the message */
    /* Fill the control and data portions */
    for (i = 0; i< SOL_LENGTH; ++i)
            reqctlpart[i] = 'x';
    for (i=0; i<MAX_DATA_SIZE; ++i)
            reqdatapart[i]='y';
    reqctlbuf.len = SOL_LENGTH;
    reqdatabuf.len = MAX_DATA_SIZE;
    /* Now initialize the address of server
     * of the transaction. That's where the
     * request message is to be sent
     */
    msginfo.socketid.hostid=DEST_HOST;
    msginfo.socketid.portid =EXAMPLE_PORT;
    /* Note that the buffers to receive the control and
     * data part of the response message have already
     * been initialized. When receiving messages only
 * the maxlen and buf fields need to be initialized.
 * On return, the len field will be filled by the
 * actual number of bytes received.
 */
```

(转下页)



```
ret=mb2s_sendrcvp(fd, &msginfo, &reqctlbuf,
                  &reqdatabuf, &resctlbuf, &resdatabuf);
if(ret == -1) {
        /* mb2s_sendrcvp failed */
        perror("mb2s_sendrcvp");
        exit(-1);
}
/* Check the type of message received */
switch (msginfo.msgtyp) {
case MB2_RESP_MSG:
        printf("Response message received\n");
        break;
default:
        printf ("Unexpected message type received\n");
        exit(-1);
}
/* Figure out the size of the control
 * and the data part of the response message
 */
printf ("Control part length of response message =%d\n",
                                        resctlbuf.len);
printf ("Data part length of response message =%d\n",
                                        resdatabuf.len);
printf("Control Part of the Response:\n");
for(i=0; i<resctlbuf.len;++i)
        printf("%x ", resctlbuf.buf[i]);
printf("\n");
```

3·1·9·2 接收请求报文并发送应答报文

当接收一个来自委托方的事务请求报文时，报文的类型字段 msgtype 被置成 MB2_REQ_MSG 或 MB2_REQFRAG_MSG。图 3—4 给出一个服务方应用程序，它接收请求报文并发送应答报文给委托方。

接收之前，必须提供一个足够大的缓冲区以便存放所接收的数据。

一旦从 mb2s_receive 返回，报文类型和事务标识将在 msginfo 结构中给出。假如报文类型为 MB2_REQ_MSG，则整个请求报文都已收到，程序可以继续发送应答报文，假如报文类型为 MB2-REQFRAG-MSG，它隐含着事务已经被置于分段接收状态。在这种情况下、接收方应使用 mb2s_getreqfrag 例程接收请求报文的相应分段的报文数据。

对 mb2_getfrag 进行了一或多次调用后，事务仍处于请求报文分段接收状态，应答报文一直不能发出直到所有数据都已收到或请求过程因请求 0 长度分段报文而结束。数据的长度字段决定所接收的请求报文分段的长度。

完成事务请求过程后，就可以使用 mb2s_sendreply 例程来发送应答报文。对这一

例程的调用与对 mb2s_send 例程的调用非常类似。它需要一个目的地址以及控制部分和数据部分缓冲区。除此之外，它还需要一个事务标识来标识相应的事务，一个标志用来标识本段是否为分段应答报文的最后一个。

当这个标志为 MB2_EOT 时，它意味着本应答报文分段是该事务的最后一个报文分段。当这一标志为 MB2_NOTEOT 时，它意味着本应答报文分段之后还有该事务的其它应答报文分段。

图 3-4 所示的程序为发送两个长度 2K 的报文分段。注意，对 mb2s_sendreply 的第一次调用使用的是 MB2_NOTEOT 标志。MB2-NOTEOT 标志只在报文分段的数据长度不为 0 时有效。对于只含有控制部分的应答报文，则必须用 MB2_EOT 标志发送。

图 3-4　服务方

```
/*
 * Example program showing an application that
 * receives a 4K data message in the request of a transaction
 * and sends two 2K fragments data portions in the response
 * message
 */
#include <stdio.h>
#include <sys/types.h>
#include <sys/mb2tciver.h>
/* Maximum Size of the data part of the message */
#define MAX_DATA_SIZE 0x1000
/* Size of each fragment of the response message */
#define RESPONSE_LEN 0x400
/* Port ID of port where to listen for request messages */
#define EXAMPLE_PORT 0x400
/* Notes that a server does not need to know the host ID
 * Port ID of the sender.
 * It just needs to "listen" on the "well-known port" to receive
 * the request messages.
 * Information about the sender will be returned when a message
 * is received
 */
/* Buffers to receive the request message */
char rcvctlpart[SOL_LENGTH];
char rcvdatapart[MAX_DATA_SIZE];
mb2_buf rcvctlbuf={SOL_LENGTH, 0, &rcvctlpart[0]};
mb2_buf rcvdatabuf = {MAX_DATA_SIZE, 0, &rcvdatapart[0]};
/* Buffers to build and send the response message */
char resctlpart[SOL_LENGTH];
char resdatapart[MAX_DATA_SIZE];
mb2_buf resctlbuf={SOL_LENGTH, 0, &resctlpart[0]};
mb2_buf resdatabuf={MAX_DATA_SIZE,0, &resdatapart[0]}
```

（转下页）