计算机系统应用 ISSN 1003-3254, CODEN CSAOBN
Computer Systems & Applications,2018,27(10):279−284 [doi: 10.15888/j.cnki.csa.006578]

E-mail: csa@iscas.ac.cn
http://www.c-s-a.org.cn
Tel: +86-10-62661041

基于 Netlink 机制的 1553B 协议通用接口设计①

顾燕飞, 邹维军, 王子剑, 张庆松

(中国电子科技集团公司第三十二研究所, 上海 201808)
通讯作者: 顾燕飞, E-mail: guyanfei2011@126.com

摘　要: Netlink 是 Linux 系统下的一种全新的协议族, 其实现了进程间通信功能. 文中在参考了 linux 操作系统下 Netlink 机制和 1553B 协议的实现, 研究了将 1553B 协议和基于 Netlink 机制的 socket 套接字相结合之后, 设计了一套基于 Netlink 机制的 1553B 协议通用接口. 该套接口的实现是通过 Netlink 通信机制对 socket 下的协议栈进行了扩展和定义. 文中实现了基于 Netlink 机制下使用 socket API 传输 1553B 数据的功能. 通过实验验证了通用接口的便利性.

关键词: Netlink; 1553B 协议; 通用套接字; BC; RT

引用格式: 顾燕飞,邹维军,王子剑,张庆松.基于 Netlink 机制的 1553B 协议通用接口设计.计算机系统应用,2018,27(10):279–284. http://www.c-s-a.org.cn/1003-3254/6578.html

Design of 1553B Protocol Common Sockets Based on Netlink Mechanism

GU Yan-Fei, ZOU Wei-Jun, WANG Zi-Jian, ZHANG Qing-Song

(The 32nd Research Institute of China Electronic Technology Corporation, Shanghai 201808, China)

Abstract: Netlink that implemented the interprocess communication function is a new protocol family in the Linux system. After referring to the Netlink mechanism of linux OS and implementation of 1553B protocol, and researching on the feasibility of combining sockets based on Netlink mechanism and 1553B protocol, this study designed the1553B protocol common socket based on Netlink mechanism. The socket API extends and defines the socket protocol stacks by Netlink common mechanism. It implements transmission of the1553B data using the socket API in Netlink mechanism. And it proves the convenience of the API through the experiment.

Key words: Netlink; 1553B protocol; common socket; BC; RT

Netlink 作为一种进程间通信机制, 除了提供内核不同模块间通信, 还能实现内核态与用户态进程间通信. 相较于传统的用户进程与内核通信的机制 (如 ioctl 方法和 proc 文件系统) 的单向性, Netlink 可以实现全双工通信[1].

1553B 总线全称为数字时分命令/响应型多路传输数据总线. 最早应用于美军航空电子综合系统之上, 各种航电设备基于 MIL-STD-1553B 总线完成设备间通信[2]. 但随着其不断的发展, 线性局域网络结构使得 1553B 总线成为航空系统或地面车辆系统中分布式设备的理想连接方式. 与点对点连接相比, 它减少了所需电缆、所需空间和系统的重量. 便于维护, 易于增加或删除节点, 提高了设计灵活性, 冗余容错能力等特点, 在航空、航天、航海和民用等领域都得到广泛的使用[3].

Socket 1553B 协议是在 Linux 下 1553B 协议实现的一种方法. Linux 下一般使用 1553B 协议的方法是基于字符设备来实现的, 与之不同的是 Socket 1553B 使用 Netlink 的 socket 接口和 linux 网络协议栈, 这种方

① 收稿时间: 2018-02-06; 修改时间: 2018-03-07; 采用时间: 2018-04-03; csa 在线出版时间: 2018-09-28

法使得 1553B 设备驱动可以通过网络接口来调用. Socket 1553B 的接口被设计的尽量接近 TCP/IP 的协议, 让那些熟悉网络编程的程序员能够比较容易的学习和使用.

1　系统总体架构设计

整个系统环境的最底层是 PCIE-1553B 接口卡, 在操作系统中为 1553B 设备创建网络驱动, 负责配置和收发操作. 在收发网络包过程中对其进行解析报文, 根据协议类型判断是否为 1553B 数据包, 递交给 Netlink 内核层中的接收函数处理[4]. 利用内核提供的 Netlink 接口层, 创建新的网络协议, 屏蔽了底层实现的差异, 为上层 socket 应用提供了统一的应用接口. 系统总体架构如图 1.

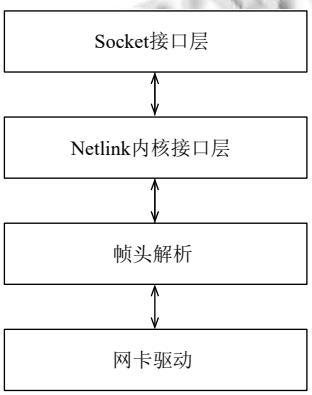

图 1　系统总体架构

2　1553B 的接口设计

2.1　1553B 协议介绍

1553B 总线系统包括串行传输电缆、总线控制器 BC、远程终端 RT、总线监控器 MT[5]. 其组成如图 2 所示.

本文以 BC 与 RT 之间的通信为例, 来说明该总线系统的工作过程. 该过程如下:

(1) 总线控制器 BC 通过初始化完成数据传输的准备工作.

(2) BC 通过向总线发送命令字来决定参与本次数据传输的远程终端和采用的消息格式. 命令字分为发送命令字 (要求 RT 向总线上发送数据), 接收命令字 (要求 RT 接收总线上的数据), 方式代码命令字 (要求 RT 完成特殊操作).

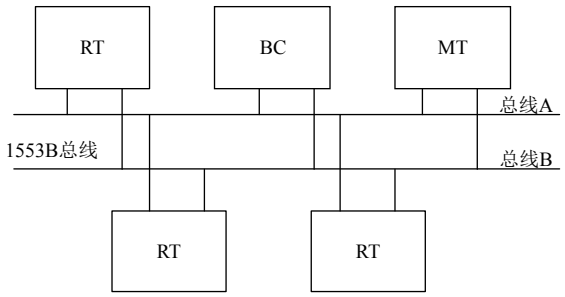

图 2　1553B 总线通信系统组成

(3) RT 检测到 BC 传来的命令字与自己地址匹配时, 将按照命令字的要求参与到数据传输过程中, 完成数据传输后 RT 将在规定时间内返回状态字[6].

(4) BC 通过检测 RT 返回的状态字来判断本次数据传输完成情况. 如果数据正确传输将结束本消息传输.

其中字格式如图 3 所示.

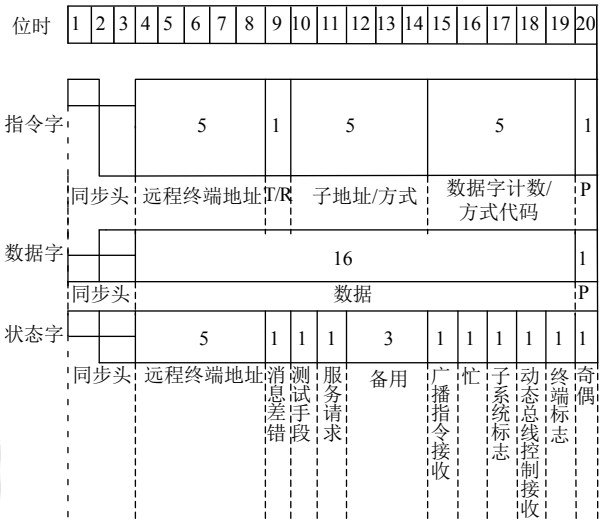

图 3　1553B 协议字格式

1553B 协议驱动实现 BC 与 RT 之间的通信. BC 和 RT 的通信过程可以分为 BC 发送 RT 接收和 RT 发送 BC 接收两种情况.

2.1.1　BC 读 RT

BC 读 RT 表示 RT 发送 BC 接收, 如图 4 所示.

如图 4 所示, 整个过程如下:

(1) RT 将数据写入本地 DMA 发送缓存, 等待 BC 来读取.

(2) BC 发送 BC 读 RT 的命令给目的 RT, 告诉目的 RT, BC 需要从 RT 的哪个子地址 (sa) 读取多长的数据 (len), 同时 BC 开始计时, 在一定的时间内若没有

收到 RT 返回的状态 (帧), 则产生超时中断.

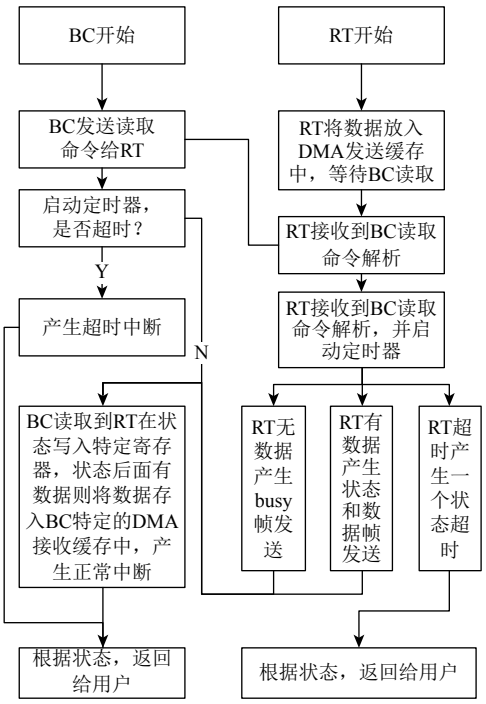

图 4 BC 读 RT 数据原理框图

(3) RT 接收到 BC 的命令后 RT 启动本地定时器 (RT 需要在规定的时间内向 BC 发送状态帧), 根据本地情况返回不同的状态帧给 BC, 具体情况如下:

1) 如果 BC 要求读取的目的子地址没有数据,则返回 busy 帧给 BC, 并产生一次正常数据发送中断.

2) 如果 BC 要求读取的目的子地址有数据,则返回“状态+数据”给 BC, 并产生一次正常数据发送中断.

3) 在 1), 2) 的过程中, 如果定时时间到, 则会产生一个“RT 返回状态超时”的中断[7].

(4) BC 在规定的时间内收到 RT 返回的状态 (busy 帧或数据帧), BC 将 RT 返回的状态写入特定寄存器保存起来, 如果返回的状态后面跟有数据, 则将数据存入 BC 特定的 DMA 接收缓存中, 供驱动程序获取, 并产生一次正常数据接收中断.

(5) 驱动程序根据返回的状态内容, 将 busy 状态或接收到的数据返回给用户.

2.1.2 BC 写 RT

BC 写 RT 表示 BC 发送 RT 接收, 如图 5 所示.

如图 5 所示, 整个过程如下:

(1) BC 将用户数据写入本地特定 DMA 发送缓存.

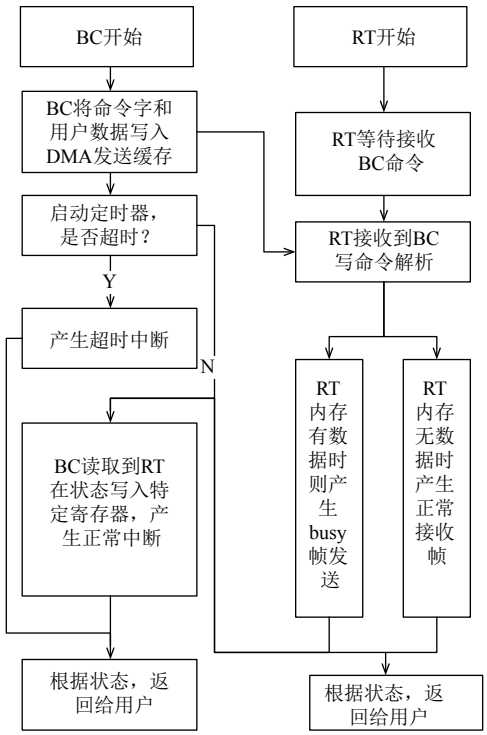

图 5 BC 写 RT 数据原理框图

(2) BC 将目的 RT 号、目的子地址号、数据长度信息写入特定寄存器, 供硬件组“命令字”使用.

(3) 启动命令发送.

(4) BC 以命令字+数据字 (可以有多个) 的形式往外发送数据, 在最后一个数据字发送完后, 启动定时.

(5) RT 收到 BC 写命令进行解析, 当 BC 向 RT 所写的子地址中有数据时, 返回 busy 帧发送 BC; 当 BC 向 RT 所写的子地址中无数据时, RT 接收数据, 并产生正常接收的状态发送 BC[8].

(6) 如果在一定时间内没有收到 RT 返回的状态 (表征 busy 或正常接收), 则产生一次发送超时; 否则产生一次正常发送中断.

(7) RT 将根据状态, 反馈给用户.

2.2 socket 接口层

Netlink 机制用户层的套接字采用标准的套接字接口来定义的, 其中包括 socket、bind、sendto、recvfrom、close 等[9].

int socket(int domain, int type, int protocol);

socket 函数, 指定期望的通信协议类型和传输方式. domain: 地址域, 在 Netlink 机制下应为 AF_NETLINK; type: SOCK_DGRAM 或 SOCK_RAW 两种类型; protocol: 协议使用 NETLINK_1553B_EVENT, 自定义协议号为

NETLINK_1553B_EVENT=17.

int bind (int sockfd, struct sockaddr *maddr, socklen_t addrlen);

bind 函数, 用来绑定一个套接字的地址、协议、端口号[10]. 在 Netlink 机制下绑定地址的结构体如下:

```
struct sockaddr_nl
{
sa_family_t nl_family; /*设置为 AF_NETLINK* /
unsigned short nl_pad; /*填充* /
_u32 nl_pid; /*进程号* /
_u32 nl_groups; /*多播地址掩码* /
} nladdr;
```

int sendto(int sockfd, void *buf, size_t len, unsigned flags, struct sockaddr * addr, int addrlen)

sendto 是向套接字发送消息给内核的 Netlink 层. Netlink 套接字消息包括消息头和数据两部分. Netlink 套接字所传递的消息都有一个固定的消息头, 结构体如下[11]:

```
struct nlmsghdr
{
_u32 nlmsg_len;/*消息长度* /
_u16 nlmsg_type; /*消息类型* /
_u16 nlmsg_flags; /*控制信息* /
_u32 nlmsg_seq; /*序列号* /
_u32 nlmsg_pid; /*进程号* /
};
```

nlmsg_len 表示该消息的总长度; nlmsg_type 表示消息类型, 该值与 Netlink 选用的通信协议相关, 本文设置为 NLMSG_1553B_TYPE=0x12, 表示 1553B 协议消息类型; nlmsg_flags 为控制信息, 如标志设为 NLM_F_REQUEST 或 NLM_F_ACK, NLM_F_ACK 表示要求消息接收方发回一个确认消息给消息发送方; nlmsg_seq 表示序列号; nlmsg_pid 表示发出消息进程的进程号.

int recvfrom(int sockfd, void* buf, size_t size, int flags, struct sockaddr *addr, int addr_len);

recvfrom 表示接收来自内核 Netlink 层的消息, 该函数阻塞等待底层有数据到来.

int close(int sockfd);

close 表示关闭一个 Netlink 网络套接字, 具体操作是把套接字链表中对应的套接字删除, 同时释放套接字的接收缓存和发送缓存, 清除套接字的使用内存,设置套接字的传输状态为关闭.

2.3 Netlink 内核接口层

在 Netlink 内核初始化时, 通过 netlink_kernel_create 函数来创建内核套接字, 并将自定义协议号为 NETLINK_1553B_EVENT=17, 消息类型为 NLMSG_1553B_TYPE=0x12 进行注册.

struct sock* netlink_kernel_create(struct net *net, int unit, void (* input)(struct sk_buff *skb), struct mutex *cb_mutex, struct module *module);

net 表示网络名字空间, 一般使用 init_net 这个全局变量; unit 表示使用的协议号, 这里为 NETLINK_1553B_EVENT=17; cb_mutex 为 NULL; module 为 THIS_MODULE.

input() 函数为具体的消息处理函数[12]. 内核中为某一个通信协议使用 netlink_kernel_create 函数创建了一个 Netlink 套接字, 所有该通信协议的用户方 Netlink 消息都将发送给内核空间的这个 Netlink 套接字, 由该套接字再调用创建时注册的 input() 函数来统一处理所有的消息. 这里在自定义消息处理函数为 msg1553b_receive 中, 会判断消息类型、长度等, 以及 1553B 协议整个收发过程, 包含 BC 读 RT, BC 写 RT, RT 读数据, RT 写数据等.

基于 Netlink 机制的用户态与内核态之间的通信如图 6 所示.

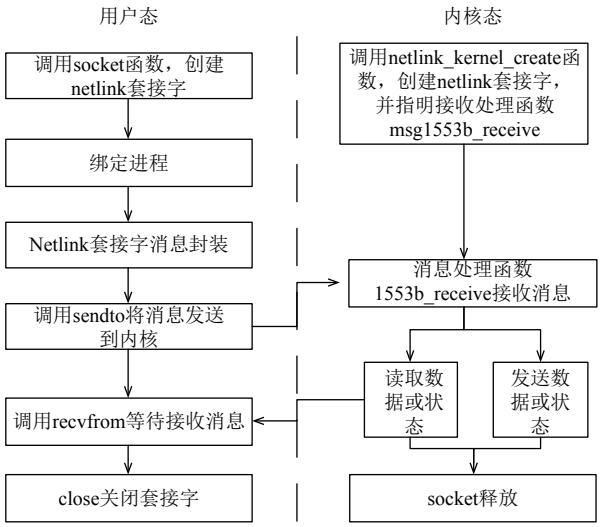

图 6 1553B 协议在内核态与用户态的通信流程

用户态中使用 Netlink 套接字 socket 创建时, 需要设置协议号为 NETLINK_1553B_EVENT=17. 在 Netlink 套接字封装消息时, 需要将消息类型设置为 NLMSG_1553B_TYPE=0x12. 并且在消息数据中规定了以下几种模式:

DEV_INIT: 表示设备的初始化, 规定该设备的角色, 是 BC 还是 RT.

BC_RD_RT: 表示 BC 读 RT, 即 BC 需要从 RT 的某个子地址中读取数据.

BC_WR_RT: 表示 BC 写 RT, 即 BC 需要向 RT 的某个子地址中写入数据.

RT_RD_DATA: 表示 RT 读数据, 即 RT 从某个子地址中读取数据.

RT_WR_DATA: 表示 RT 写数据, 即 RT 向某个子地址中写入数据.

内核态中使用 Netlink 套接字 socket 创建时, 需要注册协议号 NETLINK_1553B_EVENT=17, 并指明接收处理函数 msg1553b_receive. 在 msg1553b_receive 函数中会判断消息类型是否为 NLMSG_1553B_TYPE, 如果是, 继续处理消息; 如果不是则返回错误. 对应的消息处理函数中针对以上这几种模式会做出不同的响应:

(1) 当收到 DEV_INIT 模式时, 将判断是 BC 还是 RT, 分别进行对应的初始化.

(2) 当收到 BC_RD_RT 模式时, 将 1553B 的状态字发送给 RT, 并读取 RT 返回的数据或状态, 再返回给用户态. 若超时或无数据, 则返回对应的状态给用户态.

(3) 当收到 BC_WR_RT 模式时, 将 1553B 的“状态字+数据”发送给 RT, 并读取 RT 返回状态, 再返回给用户态. 若超时或无法写数据, 则返回对应的状态给用户态.

(4) 当收到 RT_RD_DATA 模式时, 将从底层 RT 对应子地址中的读取数据, 并返回给用户态. 若超时或无数据, 则返回对应的状态给用户态.

(5) 当收到 RT_WR_DATA 模式时, 将向底层 RT 对应子地址中的写入数据, 并返回给用户态. 若超时或无法写入数据, 则返回对应的状态给用户态.

3 实验测试

搭建测试环境, 测试设备连线示意图如图 7 所示.

按图 7 所示连接开发机、测试计算机、1553B 监控设备 (监控 1553B 总线上的数据). 测试计算机上, 通过测试软件打开 1553B 监控设备, 对整个通信过程进行监控; Linux 软件开发机上加载 1553B 驱动模块; 分别打开测试机上 RT 设备 Netlink 套接字测试程序和 Linux 开发机上 BC 设备 Netlink 套接字测试程序, 进行 BC 和 RT 设备间 1553B 数据通信测试. 本测试以 BC 读 RT 数据为例, 具体测试步骤如下.

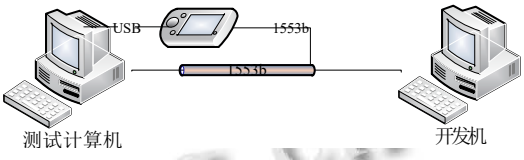

图 7 设备连线关系示意图

(1) BC 向 0 号 RT 设备发送请求模式字命令, 以获知当前 0 号 RT 设备的哪些子地址拥有数据.

(2) RT 在收到 BC 的矢量请求命令后回复一个数据字, 告知 BC 此时自身拥有数据的子地址号, 如果没有返回 0.

(3) BC 根据 RT 返回的结果, 向当前 RT 的拥有数据的某个子地址发送“RT 发送数据”命令字, 来获取该子地址的数据.

(4) RT 在收到“RT 发送数据”命令字后, 将相应子地址的数据发送给 BC, 记录此时发送的数据.

测试结果如下表所示.

通过表 1 可见, BC 发送给 RT 命令为 0x0410, 而 RT 可以接收到 0x0410; RT 回复状态和数据表示 1 子地址有数据; BC 发送给 RT 命令 0x0412 表示从 RT 的 1 子地址上取两个数据; RT 回复状态和数据表示 1 子地址上的 1, 2 数据传给了 BC. 以上的每个步骤在记录表中的数据与 1553B 监控设备监控的数据一致, 由此可见, 利用 Netlink 套接字编程可以实现 BC 与 RT 之间正常通信.

表 1 1553B 接口测试记录表

	发送数据/命令/状态	接收数据/命令/状态	监控数据
1	BC: 0x0410	RT: 0x0410	0x0410
2	RT: 0x0100 0x0001	BC: 0x0100 0x0001	0x0100 0x0001
3	BC: 0x0412	RT: 0x0412	0x0412
4	RT: 0x0 0x001 0x002	BC: 0x0 0x001 0x002	0x0 0x001 0x002

4 结束语

针对 1553B 协议提供了基于 Netlink 机制的通用 socket 接口, 给熟悉网络编程的程序员带来了极大的方便, 并且移植性好. 本文采用的是基于 linux 的 1553B

协议的实现, 但是就协议本身而言, 与操作系统相关性不是很大. 下一步工作是对 1553B 协议的优化以及多平台, 多系统上的移植, 进一步完成在工程上的实现.

参考文献

1 陈莉君. Linux 操作系统内核分析. 北京: 人民邮电出版社, 2000.

2 王鹏. 基于 1553B 总线监视与仿真软件的设计和研究[硕士学位论文]. 北京: 中国地质大学 (北京), 2017.

3 朱正国. 基于 1553B 通信协议的总线模块设计与实现[硕士学位论文]. 西安: 西安电子科技大学, 2013.

4 蒋伟. 基于 PCI 总线的高速 1553B 总线通信卡的设计与实现. 中国新通信, 2017, 19(14): 32.

5 徐丽清. 1553B 总线接口技术研究及 FPGA 实现[硕士学位论文]. 西安: 西北工业大学, 2006.

6 张浩. 嵌入式 1553B 总线通信卡的设计与研究[硕士学位论文]. 南京: 南京理工大学, 2008.

7 樊江锋, 黄意, 姚莉娟. 1553B 总线接口模块测试设备的设计与实现. 现代电子技术, 2015, 38(15): 22–24, 28.

8 张晓敏. 基于 ICD 的 1553B 通信技术研究[硕士学位论文]. 北京: 北京理工大学, 2016.

9 周莉, 柯健, 顾小晶. Netlink 套接字在 Linux 系统通信中的应用研究. 计算机与现代化, 2007, (3): 109–111.

10 熊伟, 丁涵, 罗云锋. 支持多线程并发与消息异步处理的 Linux Netlink 通信机制研究. 软件导刊, 2017, 16(10): 99–103.

11 黄超, 赵建平, 韦勇钢. Netlink 消息通信机制的 IPSec VPN 实现研究. 网络空间安全, 2017, 8(Z5): 41–44.

12 汪敏. 基于 Netlink 的 linux 服务器集群统一外设事件监听机制. 电子设计工程, 2016, 24(23): 76–78.