

转载

lybhit

2017-11-29 11:37:24

21220


收藏 112

分类专栏：

linux socketcan

文章标签：

linux socketcan

 linux socketcan 专栏收录该内容

0 订阅

1 篇文章

订阅专栏

转自 <http://velep.com/archives/1181.html>

Linux 系统中CAN 接口配置

在 Linux 系统中，CAN 总线接口设备作为网络设备被系统进行统一管理。在控制台下，CAN 总线的配置和以太网的配置使用相同的命令。

在控制台上输入命令：

```
ifconfig -a
```

可以得到以下结果：

```
can0      Link encap:UNSPEC  HWaddr 00-00-00-00-00-00-00-00-00-00-00-00-00-00-00-00
UP RUNNING NOARP  MTU:16  Metric:1
RX packets:0 errors:0 dropped:0 overruns:0 frame:0
TX packets:0 errors:0 dropped:0 overruns:0 carrier:0
collisions:0 txqueuelen:10
RX bytes:0 (0.0 B)  TX bytes:0 (0.0 B)

can1      Link encap:UNSPEC  HWaddr 00-00-00-00-00-00-00-00-00-00-00-00-00-00-00-00
UP RUNNING NOARP  MTU:16  Metric:1
RX packets:0 errors:0 dropped:0 overruns:0 frame:0
TX packets:0 errors:0 dropped:0 overruns:0 carrier:0
collisions:0 txqueuelen:30
RX bytes:0 (0.0 B)  TX bytes:0 (0.0 B)

eth0      Link encap:Ethernet  HWaddr 00:03:00:00:02:41
inet addr:192.168.1.238  Bcast:192.168.1.255  Mask:255.255.255.0
UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
RX packets:146 errors:0 dropped:0 overruns:0 frame:0
TX packets:85 errors:0 dropped:0 overruns:0 carrier:0
collisions:0 txqueuelen:1000
RX bytes:10061 (9.8 KiB)  TX bytes:5447 (5.3 KiB)
Interrupt:21 Base address:0x4000
```

在上面的结果中，eth0 设备为以太网接口，can0和can1 设备为两个 CAN 总线接口。接下来使用 ip 命令来配置 CAN 总线的位速率：

```
ip link set can0 type cantq 125 prop-seg 6phase-seg1 7 phase-seg2 2 sjw 1
```

也可以使用 ip 命令直接设定位速率：

```
ip link set can0 type can bitrate 125000
```

当设置完成后，可以通过下面的命令查询 can0 设备的参数设置：

```
ip -details link show can0
```

当设置完成后，可以使用下面的命令使能 can0 设备：

```
ifconfig can0 up
```

使用下面的命令取消 can0 设备使能：

```
ifconfig can0 down
```

在设备工作中，可以使用下面的命令来查询工作状态：

```
ip -details -statistics link show can0
```

Linux 系统中CAN 接口应用程序开发

由于系统将 CAN 设备作为网络设备进行管理，因此在 CAN 总线应用开发方面，Linux 提供了SocketCAN 接口，使得 CAN 总线通信近似于和以太网的通信，应用程序开发接口 更加通用，也更加灵活。

此外，通过 <https://gitorious.org/linux-can/can-utils> 网站发布的基于 SocketCAN 的 can-utils 工具套件，也可以实现简易的 CAN 总线通信。

下面具体介绍使用 SocketCAN 实现通信时使用的应用程序开发接口。

(1). 初始化

SocketCAN 中大部分的数据结构和函数在头文件 linux/can.h 中进行了定义。CAN 总线套接字的创建采用标准的网络套接字操作来完成。网络套接字在头文件 sys/socket.h 中定义。套接字的初始化方法如下：

```
1 int s;

2 struct sockaddr_can addr;

3 struct ifreq ifr;

4 s = socket(PF_CAN, SOCK_RAW, CAN_RAW); //创建 SocketCAN 套接字

5 strcpy(ifr.ifr_name, "can0");

6 ioctl(s, SIOCGIFINDEX, &ifr); //指定 can0 设备

7 addr.can_family = AF_CAN;

8 addr.can_ifindex = ifr.ifr_ifindex;

9 bind(s, (struct sockaddr *)&addr, sizeof(addr)); //将套接字与 can0 绑定
```

(2). 数据发送

在数据收发内容方面，CAN 总线与标准套接字通信稍有不同，每一次通信都采用 can_frame 结构体将数据封装成帧。结构体定义如下：

```
1 struct can_frame {

2     canid_t can_id; //CAN 标识符

3     __u8 can_dlc; //数据场的长度
```

分类专栏

	双系统安装	1篇
	TX2 ROS	2篇
	深度相机 ROS	1篇
	TX2 CAN	1篇
	TX2 USB串口驱动 ttyA...	1篇
	linux socketcan	1篇
	SLAM lidar	1篇
	linux 中文输入法	1篇
	linux	1篇
	IMU	
	差动机器人运动模型	

```
4  _u8 data[8]; //数据
```

```
5  };
```

can_id 为帧的标识符，如果发出的是标准帧，就使用 can_id 的低 11 位；如果为扩展帧，就使用 0 ~ 28 位。can_id 的第 29、30、31 位是帧的标志位，用来定义帧的类型，定义如下：

```
1  #define CAN_EFF_FLAG 0x80000000U //扩展帧的标识
```

```
2  #define CAN_RTR_FLAG 0x40000000U //远程帧的标识
```

```
3  #define CAN_ERR_FLAG 0x20000000U //错误帧的标识, 用于错误检查
```

数据发送使用 write 函数来实现。如果发送的数据帧(标识符为 0x123)包含单个字节(0xAB)的数据，可采用如下方法进行发送：

```
1  struct can_frame frame;
```

```
2  frame.can_id = 0x123; //如果为扩展帧, 那么 frame.can_id = CAN_EFF_FLAG | 0x123;
```

```
3  frame.can_dlc = 1; //数据长度为 1
```

```
4  frame.data[0] = 0xAB; //数据内容为 0xAB
```

```
5  int nbytes = write(s, &frame, sizeof(frame)); //发送数据
```

```
6  if(nbytes != sizeof(frame)) //如果 nbytes 不等于帧长度, 就说明发送失败
```

```
7  printf("Error\n!");
```

如果要发送远程帧(标识符为 0x123)，可采用如下方法进行发送：

```
1  struct can_frame frame;
```

```
2  frame.can_id = CAN_RTR_FLAG | 0x123;
```

```
3  write(s, &frame, sizeof(frame));
```

(3). 数据接收

数据接收使用 read 函数来完成，实现如下：

```
1  struct can_frame frame;
```

```
2  int nbytes = read(s, &frame, sizeof(frame));
```

当然，套接字数据收发时常用的 send、sendto、sendmsg 以及对应的 recv 函数也都可以用于 CAN 总线数据的收发。

(4). 错误处理

当帧接收后，可以通过判断 can_id 中的 CAN_ERR_FLAG 位来判断接收的帧是否为错误帧。如果为错误帧，可以通过 can_id 的其他符号位来判断错误的具体原因。

错误帧的符号位在头文件 linux/can/error.h 中定义。

(5). 过滤规则设置

在数据接收时，系统可以根据预先设置的过滤规则，实现对报文的过滤。过滤规则使用 can_filter 结构体来实现，定义如下：

```
1  struct can_filter {
```

```
2  canid_t can_id;
```

```
3  canid_t can_mask;
```

```
4  };
```

过滤的规则为：

接收到的数据帧的 can_id & mask == can_id & mask

通过这条规则可以在系统中过滤掉所有不符合规则的报文，使得应用程序不需要对无关的报文进行处理。在 can_filter 结构的 can_id 中，符号位 CAN_INV_FILTER 在置位时可以实现 can_id 在执行过滤前的位反转。

用户可以为每个打开的套接字设置多条独立的过滤规则，使用方法如下：

```
1  struct can_filter rfilter[2];
```

```
2  rfilter[0].can_id = 0x123;
```

```
3  rfilter[0].can_mask = CAN_SFF_MASK; //define CAN_SFF_MASK 0x00007FFU
```

```
4  rfilter[1].can_id = 0x200;
```

```
5  rfilter[1].can_mask = 0x700;
```

```
6  setsockopt(s, SOL_CAN_RAW, CAN_RAW_FILTER, &rfilter, sizeof(rfilter)); //设置规则
```

在极端情况下，如果应用程序不需要接收报文，可以禁用过滤规则。这样的话，原始套接字就会忽略所有接收到的报文。在这种仅仅发送数据的应用中，可以在内核中省略接收队列，以此减少 CPU 资源的消耗。禁用方法如下：

```
1  setsockopt(s, SOL_CAN_RAW, CAN_RAW_FILTER, NULL, 0); //禁用过滤规则
```

通过错误掩码可以实现对错误帧的过滤，例如：

```
1  can_err_mask_t err_mask = ( CAN_ERR_TX_TIMEOUT | CAN_ERR_BUSOFF );
```

```
setsockopt(s, SOL_CAN_RAW, CAN_RAW_ERR_FILTER, err_mask, sizeof(err_mask));
```

(6). 回环功能设置

在默认情况下，本地回环功能是开启的，可以使用下面的方法关闭回环/开启功能：

```
1 int loopback = 0; // 0 表示关闭, 1 表示开启( 默认)
```

```
2 setsockopt(s, SOL_CAN_RAW, CAN_RAW_LOOPBACK, &loopback, sizeof(loopback));
```

在本地回环功能开启的情况下，所有的发送帧都会被回环到与 CAN 总线接口对应的套接字上。默认情况下，发送 CAN 报文的套接字不想接收自己发送的报文，因此发送套接字上的回环功能是关闭的。可以在需要的时候改变这一默认行为：

```
1 int ro = 1; // 0 表示关闭( 默认), 1 表示开启
```

```
2 setsockopt(s, SOL_CAN_RAW, CAN_RAW_RECV_OWN_MSGS, &ro, sizeof(ro));
```

Linux 系统中CAN 接口应用程序示例

该文档提供了一个很简单的程序示例，如下：

1. 报文发送程序

```
01 /* 1. 报文发送程序 */
```

```
02 #include <stdio.h>
```

```
03 #include <stdlib.h>
```

```
04 #include <string.h>
```

```
05 #include <unistd.h>
```

```
06 #include <net/if.h>
```

```
07 #include <sys/ioctl.h>
```

```
08 #include <sys/socket.h>
```

```
09 #include <linux/can.h>
```

```
10 #include <linux/can/raw.h>
```

```
11
```

```
12 int main()
```

```
13 {
```

```
14     int s, nbytes;
```

```
15     struct sockaddr_can addr;
```

```
16     struct ifreq ifr;
```

```
17     struct can_frame frame[2] = {{0}};
```

```
18     s = socket(PF_CAN, SOCK_RAW, CAN_RAW); //创建套接字
```

```
19     strcpy(ifr.ifr_name, "can0" );
```

```
20     ioctl(s, SIOCGIFINDEX, &ifr); //指定 can0 设备
```

```
21     addr.can_family = AF_CAN;
```

```
22     addr.can_ifindex = ifr.ifr_ifindex;
```

```
23     bind(s, (struct sockaddr *)&addr, sizeof(addr)); //将套接字与 can0 绑定
```

```
24     //禁用过滤规则, 本进程不接收报文, 只负责发送
```

```
25     setsockopt(s, SOL_CAN_RAW, CAN_RAW_FILTER, NULL, 0);
```

```
26     //生成两个报文
```

```
27     frame[0].can_id = 0x11;
```

```
28     frame[0].can_dlc = 1;
```

```
29     frame[0].data[0] = 'Y';
```

```
30     frame[0].can_id = 0x22;
```

```
31     frame[0].can_dlc = 1;
```

```
32     frame[0].data[0] = 'N';
```

```
33     //循环发送两个报文
```

```

34     while(1)
35     {
36         nbytes = write(s, &frame[0], sizeof(frame[0])); //发送 frame[0]
37         if(nbytes != sizeof(frame[0]))
38         {
39             printf("Send Error frame[0]\n!");
40             break; //发送错误.退出
41         }
42         sleep(1);
43         nbytes = write(s, &frame[1], sizeof(frame[1])); //发送 frame[1]
44         if(nbytes != sizeof(frame[0]))
45         {
46             printf("Send Error frame[1]\n!");
47             break;
48         }
49         sleep(1);
50     }
51     close(s);
52     return 0;
53 }

```

2. 报文过滤接收程序

```

01 /* 2. 报文过滤接收程序 */
02 #include <stdio.h>
03 #include <stdlib.h>
04 #include <string.h>
05 #include <unistd.h>
06 #include <net/if.h>
07 #include <sys/ioctl.h>
08 #include <sys/socket.h>
09 #include <linux/can.h>
10 #include <linux/can/raw.h>
11
12 int main()
13 {
14     int s, nbytes;
15     struct sockaddr_can addr;
16     struct ifreq ifr;
17     struct can_frame frame;
18     struct can_filter rfilter[1];
19     s = socket(PF_CAN, SOCK_RAW, CAN_RAW); //创建套接字
20     strcpy(ifr.ifr_name, "can0" );
21     ioctl(s, SIOCGIFINDEX, &ifr); //指定 can0 设备
22     addr.can_family = AF_CAN;

```

```
23 //addr.can_ifindex = ifr.ifr_ifindex;

24 bind(s, (struct sockaddr *)&addr, sizeof(addr)); //将套接字与 can0 绑定

25 //定义接收规则, 只接收表示符等于 0x11 的报文

26 rfilter[0].can_id = 0x11;

27 rfilter[0].can_mask = CAN_SFF_MASK;

28 //设置过滤规则

29 setsockopt(s, SOL_CAN_RAW, CAN_RAW_FILTER, &rfilter, sizeof(rfilter));

30 while(1)

31 {

32     nbytes = read(s, &frame, sizeof(frame)); //接收报文

33     //显示报文

34     if(nbytes > 0)

35     {

36         printf("ID=0x%X DLC=%d data[0]=0x%X\n", frame.can_id,

37             frame.can_dlc, frame.data[0]);

38     }

39 }

40 close(s);

41 return 0;

42 }
```

人人都会编程的时代已经到来了？

站能力认证官方博客

1053

首先我们应该清楚：“能编程”，“会编程”和“学编程”是不同的概念。从根本上说，编程就是和计算机沟通，学习编程可以让自己变得理性、严谨起来，从电...

linux下socket can 编程详解

09-08

1、can总线介绍 2、CAN工作原理 3、CAN总线工作特点 4 can总线协议 5、CAN总线报文结构 6、总线配置



优质评论可以帮助作者获得更高权重



评论



李徕徕: 你好，请问使用write发送数据，频率升高就会返回-1是怎么回事？？ 1年前 回复 ...



明天发论文 回复: 亲你解决了吗这个问题 4月前 回复 ...



Linux CAN编程详解_JJWilliams

8-10

Linux CAN编程详解 最近写了个自认为不错的基于linux socket can程序,主要功能: 程序具备全部CAN功能,包括CAN标准帧/扩展帧接收与发送、CAN总线错...

linux_CAN编程详解_socketcan-C文档类资源

9-8

java开发can接口,再linux环境下编译的jar,直接添加到lib使用即可。 linux socket can编程说明文档.txt 30浏览 2021-06-05 linux socket can编程说明文...

Linux CAN编程详解配套代码

09-04

主要描述了以下内容： 1. can总线介绍及其帧类型； 2. Linux 系统中CAN 总线配置； 3. Linux 系统中CAN 总线应用开发接口； 4. Linux 系统中CAN编程...

canutils-linux下socketcan驱动

01-02

linux下socketcan驱动,包括cansend,candump,config,很不错的学习代码

linux can命令详解,Linux CAN编程详解_缥缈孤鸿影子的博客

8-4

Linux 系统中CAN接口应用程序开发 由于系统将 CAN 设备作为网络设备进行管理,因此在 CAN 总线应用开发方面, Linux 提供了SocketCAN 接口,使得 CA...

Linux socket CAN编程示例

jiryzhang的博客

8512

如下所示，代码展示了Linux下CAN的发送和接收：#include <stdio.h>; #include <stdlib.h>; #include <string.h>; #include <unistd.h>; #inc...

Linux CAN编程详解

lizhu_cadn的专栏

47万+

原文博客地址 http://velep.com/archives/1181.html 通过读这篇博客是我搜索can通讯以来讲解的最详细的一篇，还有其自己写的一刻关于can控制的程序...

java-socketcan

12-03

java开发can接口，再linux环境下编译的jar，直接添加到lib使用即可。

Linux下CAN总线速率设置，socketCAN。

jittl的专栏

8199

背景：飞思卡尔Freescale的ARM9处理器iMX25系列。

socketcan提供的linux下的CAN网络工具包

01-05

socketcan提供的linux下的CAN网络工具包 canutils-3.0.2.tar.bz2

2019RaspberryPi项目：Phoenix-Linux-SocketCAN-示例-源码

02-22

Phoenix-Linux-SocketCAN-示例 在Linux / RaspPi平台上使用CTRE Phoenix类库的一般示例。 演示了两个用例... 对于非FRC用例（ 将无线游戏手柄插入R...

linux socket can程序cantool

reille的笔记

9526

最近写了个自认为不错的基于linux socket can程序，主要功能： 1. 程序具备全部CAN功能，包括CAN标准帧/扩展帧接收与发送、CAN总线错误判断、环...

linux的socket CAN驱动介绍

linyangspring的专栏

5313

在linux中，CAN总线的驱动有两种实现

Linux Socket CAN——数据发送接收流程

xiaohouye的专栏

4753

Linux Socket CAN在用户空间提供socket接口，在内核空间实现CAN Frame协议，并协同CAN控制器驱动控制CAN控制器的驱动，实现CAN通信。 一、...

Linux内核Socket CAN中文文档

yuaniuluo的博客

27万+

自己在年假中空闲之余翻译的内核中Socket CAN的文档，原文地址在： http://lxr.linux.no/linux+v2.6.34/Documentation/networking/can.txt 但是这篇文档...

Linux应用编程之Iseek详解

07-13

Linux应用编程之Iseek详解 1、Iseek函数介绍（1）、文件指针：当我们要对一个文件进行读写时，一定要先打开这个文件，所以我们读写的所有文件都...

React+antd后台管理系统模板

12-01

基于React+antd写了一个后台管理模板。主要是熟悉antd组件和React，页面主要还是展示页面,比较简单不涉及后台交互的系统模板Demo

linux 多进程编程详解

春秋繁露

1万+



lybhit

码龄5年

暂无认证

3

31万+

94万+

13万+

等级

原创

周排名

总排名

访问

590

31

42

12

218

积分

粉丝

获赞

评论

收藏



私信

关注

搜博文文章

热门文章

ubuntu 16.04中文输入法安装

90009

linux socketcan编程详解

21187

ROS 深度相机 奥比中光

8464

TX2 CAN通信配置

5083

TX2 USB串口驱动ttyACM模块 通过编译内核设置

4501

最新评论

linux socketcan编程详解

明天发论文: 亲你解决了吗这个问题

ubuntu 16.04中文输入法安装

JustSim: 感谢楼主，google输入法yyds! so

ugou的配置配了不怎么好用

ROS 深度相机 奥比中光

亏到死董事长: 有深度摄像头测距的包吗

linux socketcan编程详解

李徕徕: 你好，请问使用write发送数据，频率升高就会返回-1是怎么回事？？

ubuntu 16.04中文输入法安装

xuxiao_0606: 有用

您愿意向朋友推荐“博客详情页”吗？











强烈推荐

推荐

一般般

不推荐

强烈不推荐



红包奖励



最新文章	
win10 ubuntu18.04 双系统安装	
linux 时间自动更新	
ros 程序开机自启动	
2021年 1篇	2017年 11篇

创建进程fork() 1.1头文件 #include<unistd.h>; #include<sys/types.h>; 1.2函数原型 pid_t fork(void); pid_t 是一个宏定义，其实是int 被定义...

09-30

linux套接字编程详解

介绍linux套接字编程原理，以及函数的说明和使用。

©2020 CSDN 皮肤主题: 编程工作室 设计师:CSDN官方博客 返回首页

关于我们 招贤纳士 广告服务 开发助手 400-660-0108 kefu@csdn.net 在线客服 工作时间 8:30-22:00

lybhit

关注

13

2

112

专栏目录

