串口编程说明

串口操作采用 UNIX 类似的方式，打开/关闭/发送/接收等基本操作采用类似文件系统的方式进行，而一些属性的设置和控制则使用 termios 来进行。
串口对应的设备文件名为”/dev/ttyS0”。

1. 打开串口

fd = open(“/dev/ttyS0”, O_RDWR);
如果只发送数据，可以使用 O_WRONLY， 如果只接收数据，可以设置成 O_RDONLY。

2. 关闭串口

close(fd);

3. 接收数据

ret = read(fd, buf, 100);
串口默认的打开方式是非阻塞的，因此本函数只是接收缓冲中的数据，而并非直接操作 IO。如果要加入一些 IO 的属性，请参见”使用超时”和”设置串口属性”。
如果缓冲中有接收到的数据，那么本函数将返回实际接收到的数据长度，当然不会超过指定的 100 字节。
如果缓冲中没有数据，那么将返回 0。
如果接收失败，那么将返回-1，错误代码放在 errno 中。

4. 发送数据

ret = write(fd, buf, 100);
返回值表示实际发送的数据长度。

5. 设置串口属性

tcgetattr(int fd, struct termios *termios_p);
tcsetattr(int fd, int optional_actions, struct termios *termios_p);
串口打开后，使用的串口属性实际上是上一次关闭串口前的设置。
这个设置也就是一个结构 struct termios，其中主要有以下的属性：

```
tcflag_t c_iflag        // 输入属性
tcflag_t c_oflag        // 输出属性
tcflag_t c_cflag        // 控制属性
tcflag_t c_lflag        // 本地属性
cc_t c_cc[NCCS]         // 控制字
```

c_iflag

IGNBRK	忽略接收到的 break 信号
BRKINT	如果 IGNBRK 被设置，break 信号将被忽略，否则如果 BRKINT 被设置，接收到 break 信号将导致输入/输出队列被清空，并且当前控制串口的前台进程将收到一个 SIGINT 信号。如果 IGNBRK 和 BRKINT 都没有被设置，收到的 break 信号将被接收为

NULL，即\0。但是如果 PARMARK 被设置，接收到的 break 信号将被接收为\377\0\0。

IGNPAR　忽略帧错误或奇偶校验错。

PARMARK 如果没有设置 IGNPAR，设置本属性表示在接收到的带有错误的帧格式或奇偶校验的字符将被前缀\377\0。如果两者都没有设置，带有错误的帧格式或奇偶校验的字符将被接收为\0。

INPCK　打开输入数据的奇偶校验。

ISTRIP　滤掉第 8 位。

INLCR　将接收到的 NL(换行)转换成 CR(回车)。

IGNCR　忽略接收到的 CR。

ICRNL　将收到的 CR 转换成 NL(除非设置了 IGNCR)。

IUCLC　将接收到的大写字符转换成小写。

IXON　打开输出的 XON/XOFF 控制。

IXOFF　打开输入的 XON/XOFF 控制。

c_oflag

OLCUC　将小写字符转换成大写后输出。

ONLCR　将 NL 转换成 CR-NL 后输出。

OCRNL　将 CR 转换成 NL 后输出。

ONLRET　不发送 CR。

c_cflag

CBAUD　波特率掩码，可以设置为
B2400
B4800
B9600
B19200
B38400
B57600
B115200

CSIZE　字符长度掩码，可以设置为
CS5
CS6
CS7
CS8

CSTOPB　设置为两个停止位。（默认为 1 个）

CREAD　打开接收功能。

PARENB　打开发送的奇偶校验生成功能和接收的奇偶校验检查功能,默认的是偶校验。

PARODD　使用奇校验。

CLOCAL　忽略 modem 信号线。

CRTSCTS　打开 RTS/CTS 硬件流控。

c_lflag

由于使用该类属性不多，因此在此不作介绍。

c_cc

常用的是 c_cc[VMIN]，表示调用 read 函数时等待接收的最少字符个数。例如设置为 1 时，read 函数至少要读到 1 个字符才会返回。

optional_actions

可以设置为

TCSANOW　　立即更新当前的设置。

TCSADRAIN　　在当前发送缓冲的所由数据发送完毕后再更新当前设置。

TCSAFLUSH　　同 TCSADRAIN，只是在更新前所有为被读取的收到的数据将被丢弃。

6. 使用超时

```
int select(int n, fd_set *readfds, fd_set *writefds, fd_set *exceptfds,
        struct timeval *timeout);
```

此函数用于控制接收、发送或异常出现之前的超时。

fd_set 是句柄的集合，其中的句柄都是被监测的对象。

readfds 表示需要监测其中句柄代表的设备是否可以从中读取数据。

writefds 表示需要监测其中句柄代表的设备是否可以向其写入数据。

exceptfds 表示需要监测其中句柄代表的设备是否出现异常。

timeout 是这样一个结构

```
struct timeval{
        long tv_sec;        // 秒
        long tv_usec;       // 微秒
};
```

n 则是所有监测的句柄中的最大值加一。

系统提供一些定义号的操作来操作 fd_set：

FD_CLR(int fd, fd_set *fds);

将 fd 从 fds 集合中去掉。

FD_ISSET(int fd, fd_set *fds);

检查 fd 是否在 fds 集合中。

FD_SET(int fd, fd_set *fds);

将 fd 加入 fds 集合中。

FD_ZERO(fd_set *fds);

将 fds 集合清空。

7. 其它串口操作

```
int tcdrain(int fd);
```

等待所有发送缓冲的数据全部发送出去后返回。

```
int tcflush(int fd, int queue_selector);
```

queue_selector:

TCIFLUSH

丢弃所有未读取的接收到的数据。

TCOFLUSH

丢弃所有为发送的发送缓冲的数据。

TCIOFLUSH

丢弃上面两种数据。

示例

```
#include <termios.h>
```

```
#include <fcntl.h>
#include <sys/stat.h>
#include <sys/types.h>

int main()
{
        int fd;
        struct termios attr;
        fd_set fds;
        struct timeval tv;
        unsigned char buf[1024];

        // 打开串口
        fd = open("/dev/ttyS0", O_RDWR);
        if (fd == -1)
                return -1;

        // 读取串口当前属性
        tcgetattr(fd, &attr);
        // 设置最少接收字符个数为 0
        attr.c_cc[VMIN] = 0;
        // 不处理 iflag、oflag 和 lflag
        attr.c_iflag = 0;
        attr.c_oflag = 0;
        attr.c_lflag = 0;
        // 设置波特率为 9600，字符长度为 8 位，偶校验，允许接收
        attr.c_cflag = B9600 | CS8 | PARENB | CLOCAL | CREAD;
        // 设置串口属性
        tcsetattr(fd, TCSANOW, &attr);

        // 发送字符串
        write(fd, "12345\n", 6);

        // 清除监测集合
        FD_ZERO(&fds);
        // 将串口句柄加入到监测集合中
        FD_SET(fd, &fds);
        // 设置超时为 5 秒
        tv.tv_sec = 5;
        tv.tv_usec = 0;
        // 监测串口是否有数据接收到，超时为 5 秒
        if (select(fd+1, &fds, NULL, NULL, &tv) <= 0)
                return -1;

        // 接收最多 100 个字符
        read(fd, buf, 100);

        // 关闭串口
        close(fd);
        return 0;
}
```

嵌入式资源免费下载

总线协议：

1. 基于 PCIe 驱动程序的数据传输卡 DMA 传输
2. 基于 PCIe 总线协议的设备驱动开发
3. CANopen 协议介绍
4. 基于 PXI 总线 RS422 数据通信卡 WDM 驱动程序设计
5. FPGA 实现 PCIe 总线 DMA 设计
6. PCI Express 协议实现与验证
7. VPX 总线技术及其实现
8. 基于 Xilinx FPGA 的 PCIE 接口实现
9. 基于 PCI 总线的 GPS 授时卡设计
10. 基于 CPCI 标准的 6U 信号处理平台的设计
11. USB30 电路保护
12. USB30 协议分析与框架设计
13. USB 30 中的 CRC 校验原理及实现

VxWorks：

1. 基于 VxWorks 的多任务程序设计
2. 基于 VxWorks 的数据采集存储装置设计
3. Flash 文件系统分析及其在 VxWorks 中的实现
4. VxWorks 多任务编程中的异常研究
5. VxWorks 应用技巧两例
6. 一种基于 VxWorks 的飞行仿真实时管理系统
7. 在 VxWorks 系统中使用 TrueType 字库
8. 基于 FreeType 的 VxWorks 中文显示方案
9. 基于 Tilcon 的 VxWorks 简单动画开发
10. 基于 Tilcon 的某武器显控系统界面设计
11. 基于 Tilcon 的综合导航信息处理装置界面设计
12. VxWorks 的内存配置和管理
13. 基于 VxWorks 系统的 PCI 配置与应用
14. 基于 MPC8270 的 VxWorks BSP 的移植

Linux：

1. Linux 程序设计第三版及源代码
2. NAND FLASH 文件系统的设计与实现
3. 多通道串行通信设备的 Linux 驱动程序实现
4. Zsh 开发指南-数组
5. 常用 GDB 命令中文速览
6. 嵌入式 C 进阶之道

Windows CE：

1. Windows CE.NET 下 YAFFS 文件系统 NAND Flash 驱动程序设计
2. Windows CE 的 CAN 总线驱动程序设计
3. 基于 Windows CE.NET 的 ADC 驱动程序实现与应用的研究
4. 基于 Windows CE.NET 平台的串行通信实现
5. 基于 Windows CE.NET 下的 GPRS 模块的研究与开发
6. win2k 下 NTFS 分区用 ntldr 加载进 dos 源代码
7. Windows 下的 USB 设备驱动程序开发
8. WinCE 的大容量程控数据传输解决方案设计
9. WinCE6.0 安装开发详解
10. DOS 下仿 Windows 的自带计算器程序 C 源码
11. G726 局域网语音通话程序和源代码
12. WinCE 主板加载第三方驱动程序的方法
13. WinCE 下的注册表编辑程序和源代码
14. WinCE 串口通信源代码
15. WINCE 的 SD 卡程序[可实现读写的源码]

PowerPC：

1. Freescale MPC8536 开发板原理图
2. 基于 MPC8548E 的固件设计
3. 基于 MPC8548E 的嵌入式数据处理系统设计
4. 基于 PowerPC 嵌入式网络通信平台的实现
5. PowerPC 在车辆显控系统中的应用

6. 基于 PowerPC 的单板计算机的设计

ARM：

1. 基于 DiskOnChip 2000 的驱动程序设计及应用
2. 基于 ARM 体系的 PC-104 总线设计
3. 基于 ARM 的嵌入式系统中断处理机制研究
4. 设计 ARM 的中断处理
5. 基于 ARM 的数据采集系统并行总线的驱动设计
6. S3C2410 下的 TFT LCD 驱动源码
7. STM32 SD 卡移植 FATFS 文件系统源码
8. STM32 ADC 多通道源码
9. ARM Linux 在 EP7312 上的移植

Hardware：

1. DSP 电源的典型设计
2. 高频脉冲电源设计
3. 电源的综合保护设计
4. 任意波形电源的设计
5. 高速 PCB 信号完整性分析及应用
6. DM642 高速图像采集系统的电磁干扰设计