全志科技
Allwinner Technology

A64

A64 UART 驱动使用说明文档

/V2.0

全 志 科 技
Allwinner Technology

Confidential

文档履历

版本号	日期	制/修订人	制/修订记录
V1.0	2013-06-28	段敏涛	建立初始版本
V2.0	2015-02-16	黄威	Linux 内核版本从 3.4 变更到 3.10，支持 64 位硬件平台，支持 Device Tree，更新 sys_config 配置项。

目　　录

1. 概述

1.1. 编写目的

介绍 Linux 内核中 UART 驱动的接口及使用方法，为 UART 设备的使用者提供参考。

1.2. 适用范围

适用于 allwinnertech 的 A6x 系列平台。

1.3. 相关人员

UART 驱动、及应用层的开发/维护人员。

2. 模块介绍

2.1. 模块功能介绍

Linux 内核中，UART 驱动的结构图 2.1 所示，可以分为三个层次：

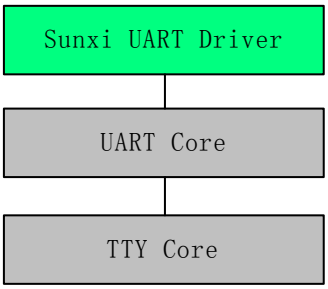

图 2.1 Linux UART 体系结构图

1. Sunxi UART Driver，负责 SUNXI 平台 UART 控制器的初始化、数据通信等，也是我们要实现的部分；
2. UART Core，为 UART 驱动提供了一套 API，完成设备和驱动的注册等；
3. TTY core，实现了内核中所有 TTY 设备的注册和管理。

2.2. 相关术语介绍

术语	解释说明
Sunxi	指 Allwinner 的一系列 SOC 硬件平台。
UART	Universal Asynchronous Receiver/Transmitter，通用异步收发传输器。
console	控制台，Linux 内核中用于输出调试信息的 TTY 设备。
TTY	TeleType/TeleTypewriters 的一个老缩写，原来指的是电传打字机，现在泛指和计算机串行端口连接的终端设备。TTY 设备还包括虚拟控制台，串口以及伪终端设备。

2.3. 模块配置介绍

2.3.1. sys_config.fex 配置说明

在不同的 Sunxi 硬件平台中，UART 控制器的数目也不同，每个 UART 控制器支持的线数也不同。线数不同在配置文件的信息基本类似，以常见的 4 线为例，在 sys_config.fex 中配置参数相似，如下：

```
[uart2]
uart2_used          = 1
uart2_port          = 2
uart2_type          = 4
uart2_tx            = port:PG06<2><1><default><default>
uart2_rx            = port:PG07<2><1><default><default>
uart2_rts           = port:PG08<2><1><default><default>
uart2_cts           = port:PG09<2><1><default><default>
```

其中：

1. uart2_used 置为 1 表示使能，0 表示不使能；
2. uart2_port，表示 UART 端口号；
3. uart2_type，UART 线数的类型，取值有 2、4、8；
4. uart2_tx 等四个配置是相应信号线的 GPIO。

2.3.2. menuconfig 配置说明

在命令行中进入内核根目录，执行 make ARCH=arm64 menuconfig 进入配置主界面，并按以下步骤操作。

首先，选择 Device Drivers 选项进入下一级配置，如下图所示：

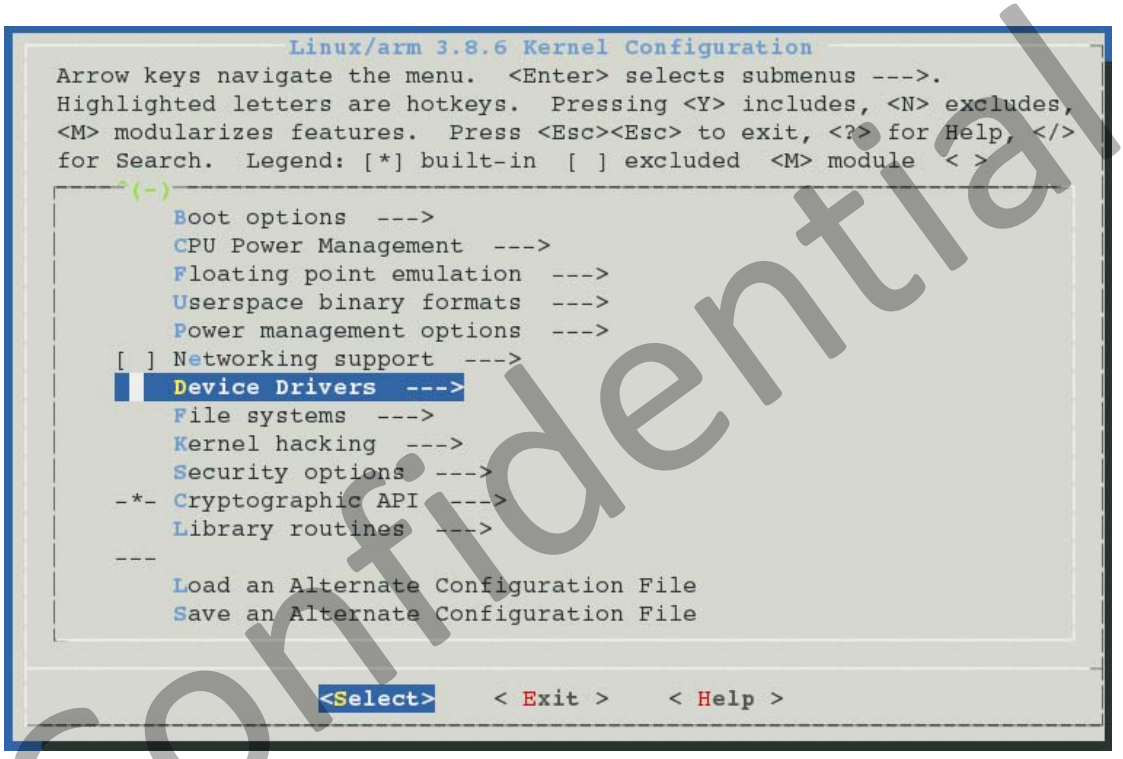

图 2.3 Device Drivers 配置

然后，选择 Character devices 选项，进入下一级配置，如下图所示：

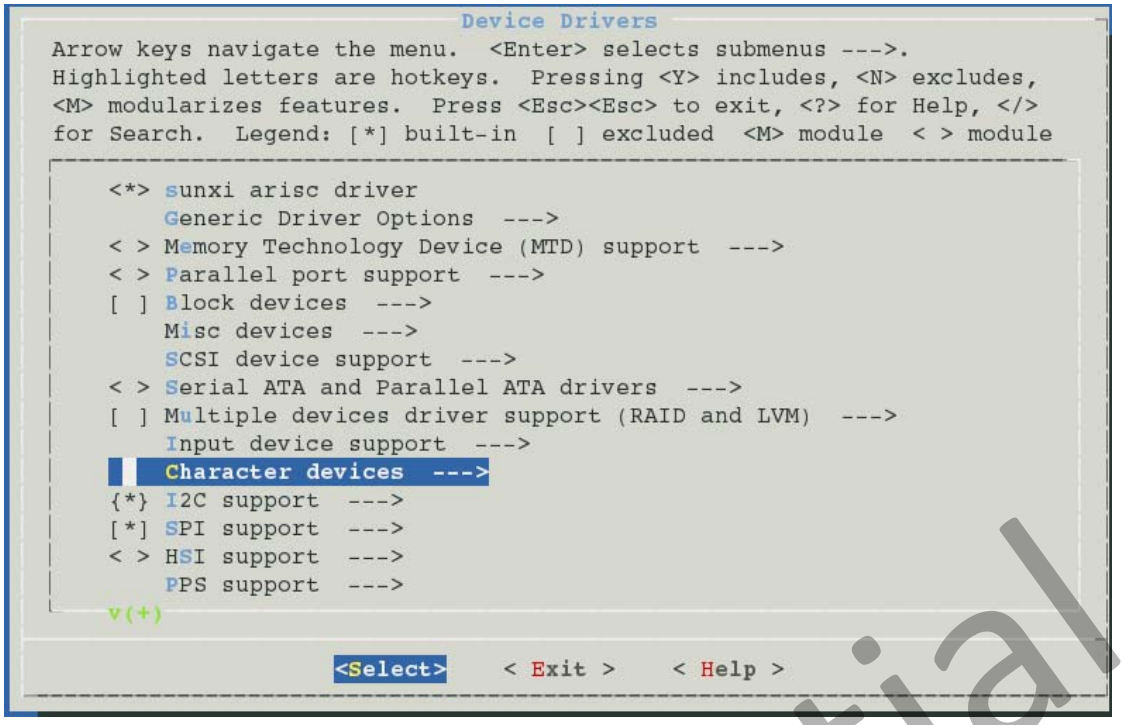

图 2.4 Character devices 配置选项

接着选择 Serial drivers 选项，进入下一级配置，如下图所示：

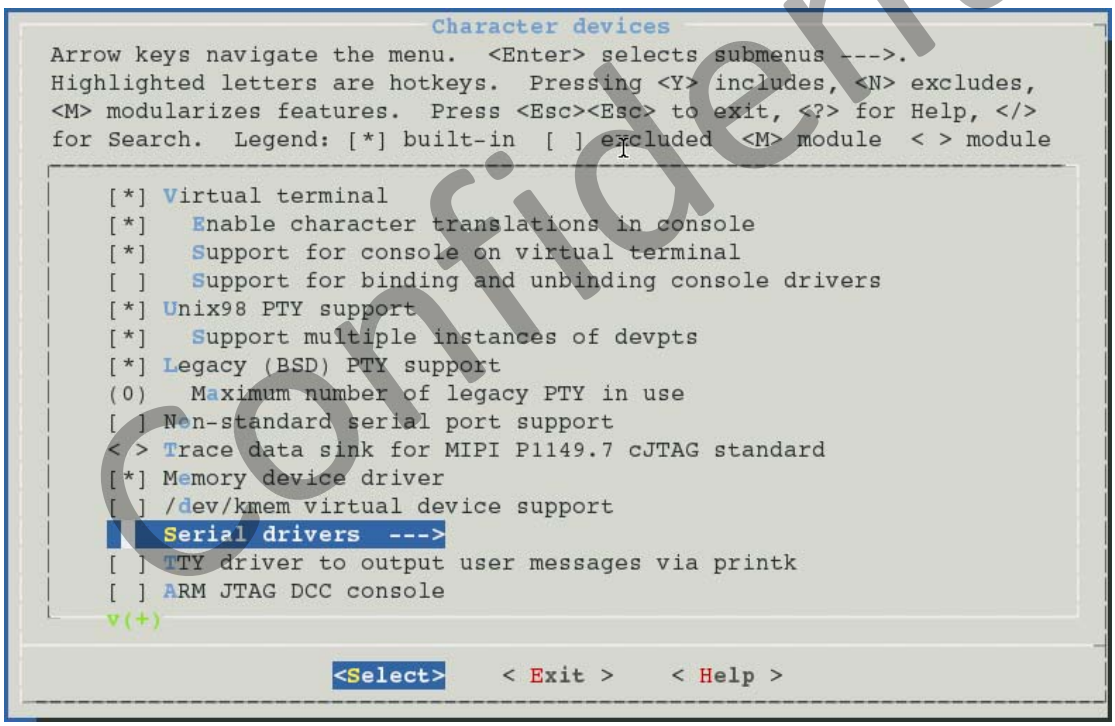

图 2.5 Serial drivers 配置选项

选择 SUNXI UART Controller 选项，因为 UART0 要用作调试串口，所以下面的“Console on SUNXI UART port”默认是选中状态，且不要将 UART 设置成模块的编译方式。如下图：

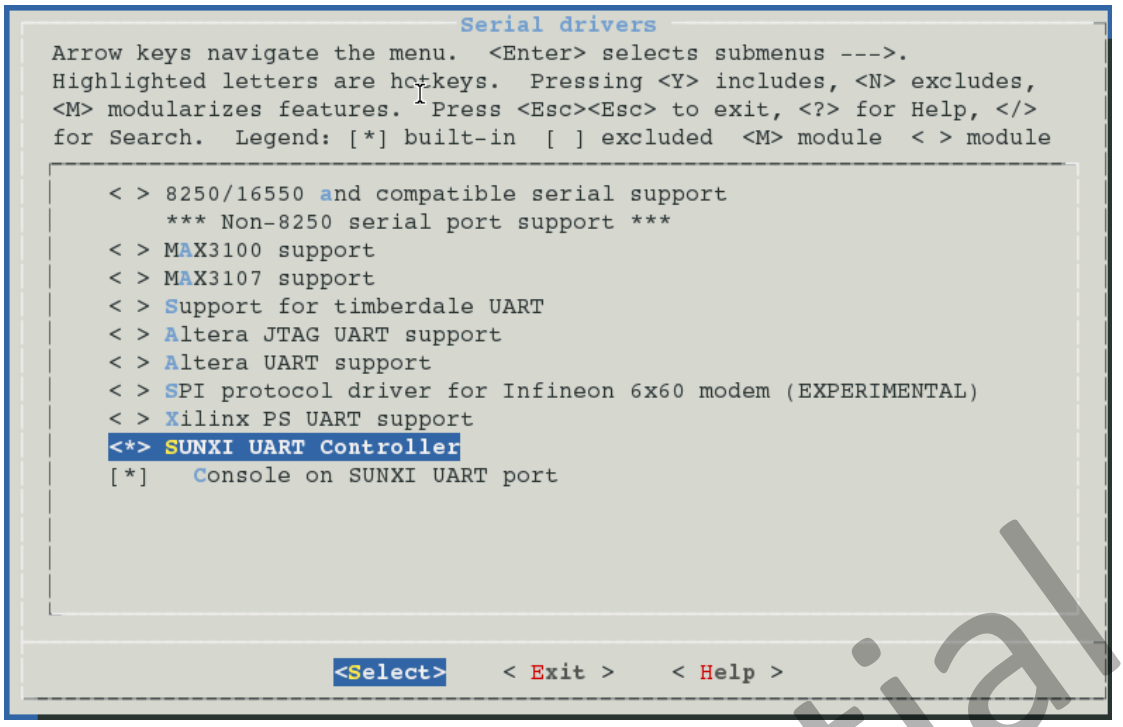

图 2.6 SUNXI UART Controller 配置选项

2.4. 源码结构介绍

UART 驱动的源代码位于内核在 drivers/tty/serial 目录下：

drivers/tty/serial/

├── sunxi-uart.c // Sunxi 平台的 UART 控制器驱动代码

├── sunxi-uart.h // 为 Sunxi 平台的 UART 控制器驱动定义了一些宏、数据结构

3. 接口描述

UART 驱动会注册生成串口设备/dev/ttySx，应用层的使用只需遵循 Linux 系统中的标准串口编程方法即可。

3.1. 打开/关闭串口

使用标准的文件打开函数：

int open(const char *pathname, int flags);

int close(int fd);

需要引用头文件：

#include <sys/types.h>

#include <sys/stat.h>

#include <fcntl.h>

#include <unistd.h>

3.2. 读/写串口

同样使用标准的文件读写函数：

ssize_t read(int fd, void *buf, size_t count);

ssize_t write(int fd, const void *buf, size_t count);

需要引用头文件：

#include <unistd.h>

3.3. 设置串口属性

串口属性包括波特率、数据位、停止位、校验位、流控等，这部分是串口设备特有的接口。串口属性的数据结构 termios 定义如下：（terminos.h）

```
#define NCCS 19
struct termios {
    tcflag_t c_iflag;       /* input mode flags */
    tcflag_t c_oflag;       /* output mode flags */
    tcflag_t c_cflag;       /* control mode flags */
    tcflag_t c_lflag;       /* local mode flags */
    cc_t c_line;            /* line discipline */
    cc_t c_cc[NCCS];        /* control characters */
};
```

其中，**c_iflag** 的标志常量定义如下：

标志	说明
IGNBRK	忽略输入中的 BREAK 状态。
BRKINT	如果设置了 IGNBRK，将忽略 BREAK。如果没有设置，但是设置了 BRKINT，那么 BREAK 将使得输入和输出队列被刷新，如果终端是一个前台进程组的控制终端，这个进程组中所有进程将收到 SIGINT 信号。如果既未设置 IGNBRK 也未设置 BRKINT，BREAK 将视为与 NUL 字符同义，除非设置了 PARMRK，这种情况下它被视为序列 \377 \0 \0。
IGNPAR	忽略桢错误和奇偶校验错。
PARMRK	如果没有设置 IGNPAR，在有奇偶校验错或桢错误的字符前插入 \377 \0。如果既没有设置 IGNPAR 也没有设置 PARMRK，将有奇偶校验错或桢错误的字符视为 \0。
INPCK	启用输入奇偶检测。
ISTRIP	去掉第八位。
INLCR	将输入中的 NL 翻译为 CR。
IGNCR	忽略输入中的回车。
ICRNL	将输入中的回车翻译为新行 (除非设置了 IGNCR)。
IUCLC	(不属于 POSIX) 将输入中的大写字母映射为小写字母。
IXON	启用输出的 XON/XOFF 流控制。
IXANY	(不属于 POSIX.1；XSI) 允许任何字符来重新开始输出。
IXOFF	启用输入的 XON/XOFF 流控制。
IMAXBEL	(不属于 POSIX) 当输入队列满时响零。Linux 没有实现这一位，总是将它视为已设置。

c_oflag 的标志常量定义如下：

标志	说明
OLCUC	(不属于 POSIX) 将输出中的小写字母映射为大写字母。
ONLCR	(XSI) 将输出中的新行符映射为回车-换行。
OCRNL	将输出中的回车映射为新行符
ONOCR	不在第 0 列输出回车。
ONLRET	不输出回车。
OFILL	发送填充字符作为延时，而不是使用定时来延时。
OFDEL	(不属于 POSIX) 填充字符是 ASCII DEL (0177)。如果不设置，填充字符则是 ASCII NUL。
NLDLY	新行延时掩码。取值为 NL0 和 NL1。
CRDLY	回车延时掩码。取值为 CR0, CR1, CR2, 或 CR3。
TABDLY	水平跳格延时掩码。取值为 TAB0, TAB1, TAB2, TAB3 (或 XTABS)。取值为 TAB3，即 XTABS，将扩展跳格为空格 (每个跳格符填充 8 个空格)。
BSDLY	回退延时掩码。取值为 BS0 或 BS1。(从来没有被实现过)
VTDLY	竖直跳格延时掩码。取值为 VT0 或 VT1。

FFDLY	进表延时掩码。取值为 FF0 或 FF1。

c_cflag 的标志常量定义如下：

标志	说明
CBAUD	(不属于 POSIX) 波特率掩码 (4+1 位)。
CBAUDEX	(不属于 POSIX) 扩展的波特率掩码 (1 位)，包含在 CBAUD 中。 (POSIX 规定波特率存储在 termios 结构中，并未精确指定它的位置，而是提供了函数 cfgetispeed() 和 cfsetispeed() 来存取它。一些系统使用 c_cflag 中 CBAUD 选择的位，其他系统使用单独的变量，例如 sg_ispeed 和 sg_ospeed 。)
CSIZE	字符长度掩码。取值为 CS5, CS6, CS7, 或 CS8。
CSTOPB	设置两个停止位，而不是一个。
CREAD	打开接受者。
PARENB	允许输出产生奇偶信息以及输入的奇偶校验。
PARODD	输入和输出是奇校验。
HUPCL	在最后一个进程关闭设备后，降低 modem 控制线 (挂断)。
CLOCAL	忽略 modem 控制线。
LOBLK	(不属于 POSIX) 从非当前 shell 层阻塞输出(用于 shl)。
CIBAUD	(不属于 POSIX) 输入速度的掩码。CIBAUD 各位的值与 CBAUD 各位相同，左移了 IBSHIFT 位。
CRTSCTS	(不属于 POSIX) 启用 RTS/CTS (硬件) 流控制。

c_lflag 的标志常量定义如下：

标志	说明
ISIG	当接受到字符 INTR, QUIT, SUSP, 或 DSUSP 时，产生相应的信号。
ICANON	启用标准模式 (canonical mode)。允许使用特殊字符 EOF, EOL, EOL2, ERASE, KILL, LNEXT, REPRINT, STATUS, 和 WERASE，以及按行的缓冲。
XCASE	(不属于 POSIX; Linux 下不被支持) 如果同时设置了 ICANON，终端只有大写。输入被转换为小写，除了以 \ 前缀的字符。输出时，大写字符被前缀 \，小写字符被转换成大写。
ECHO	回显输入字符。
ECHOE	如果同时设置了 ICANON，字符 ERASE 擦除前一个输入字符，WERASE 擦除前一个词。
ECHOK	如果同时设置了 ICANON，字符 KILL 删除当前行。
ECHONL	如果同时设置了 ICANON，回显字符 NL，即使没有设置 ECHO。
ECHOCTL	(不属于 POSIX) 如果同时设置了 ECHO，除了 TAB, NL, START, 和 STOP 之外的 ASCII 控制信号被回显为 ^X, 这里 X 是比控制信号大 0x40 的 ASCII 码。例如，字符 0x08 (BS) 被回显为 ^H。
ECHOPRT	(不属于 POSIX) 如果同时设置了 ICANON 和 IECHO，字符在删除的同时被打印。
ECHOK	(不属于 POSIX) 如果同时设置了 ICANON，回显 KILL 时将删除一行中的每

E	个字符，如同指定了 ECHOE 和 ECHOPRT 一样。
DEFECHO	(不属于 POSIX) 只在一个进程读的时候回显。
FLUSHO	(不属于 POSIX; Linux 下不被支持) 输出被刷新。这个标志可以通过键入字符 DISCARD 来开关。
NOFLSH	禁止在产生 SIGINT, SIGQUIT 和 SIGSUSP 信号时刷新输入和输出队列。
PENDIN	(不属于 POSIX; Linux 下不被支持) 在读入下一个字符时，输入队列中所有字符被重新输出。(bash 用它来处理 typeahead)
TOSTOP	向试图写控制终端的后台进程组发送 SIGTTOU 信号。
IEXTEN	启用实现自定义的输入处理。这个标志必须与 ICANON 同时使用，才能解释特殊字符 EOL2，LNEXT，REPRINT 和 WERASE，IUCLC 标志才有效。

c_cc 数组定义了特殊的控制字符。符号下标 (初始值) 和意义为：

标志	说明
VINTR	(003, ETX, Ctrl-C, or also 0177, DEL, rubout) 中断字符。发出 SIGINT 信号。当设置 ISIG 时可被识别，不再作为输入传递。
VQUIT	(034, FS, Ctrl-\) 退出字符。发出 SIGQUIT 信号。当设置 ISIG 时可被识别，不再作为输入传递。
VERASE	(0177, DEL, rubout, or 010, BS, Ctrl-H, or also #) 删除字符。删除上一个还没有删掉的字符，但不删除上一个 EOF 或行首。当设置 ICANON 时可被识别，不再作为输入传递。
VKILL	(025, NAK, Ctrl-U, or Ctrl-X, or also @) 终止字符。删除自上一个 EOF 或行首以来的输入。当设置 ICANON 时可被识别，不再作为输入传递。
VEOF	(004, EOT, Ctrl-D) 文件尾字符。更精确地说，这个字符使得 tty 缓冲中的内容被送到等待输入的用户程序中，而不必等到 EOL。如果它是一行的第一个字符，那么用户程序的 read() 将返回 0，指示读到了 EOF。当设置 ICANON 时可被识别，不再作为输入传递。
VMIN	非 canonical 模式读的最小字符数。
VEOL	(0, NUL) 附加的行尾字符。当设置 ICANON 时可被识别。
VTIME	非 canonical 模式读时的延时，以十分之一秒为单位。
VEOL2	(not in POSIX; 0, NUL) 另一个行尾字符。当设置 ICANON 时可被识别。
VSTART	(021, DC1, Ctrl-Q) 开始字符。重新开始被 Stop 字符中止的输出。当设置 IXON 时可被识别，不再作为输入传递。
VSTOP	(023, DC3, Ctrl-S) 停止字符。停止输出，直到键入 Start 字符。当设置 IXON 时可被识别，不再作为输入传递。
VSUSP	(032, SUB, Ctrl-Z) 挂起字符。发送 SIGTSTP 信号。当设置 ISIG 时可被识别，不再作为输入传递。
VLNEXT	(not in POSIX; 026, SYN, Ctrl-V) 字面上的下一个。引用下一个输入字符，取消它的任何特殊含义。当设置 IEXTEN 时可被识别，不再作为输入传递。
VWERASE	(not in POSIX; 027, ETB, Ctrl-W) 删除词。当设置 ICANON 和 IEXTEN 时可被识别，不再作为输入传递。
VREPRINT	(not in POSIX; 022, DC2, Ctrl-R) 重新输出未读的字符。当设置 ICANON 和 IEXTEN 时可被识别，不再作为输入传递。

3.3.1. tcgetattr()

【函数原型】：int tcgetattr(int fd, struct termios *termios_p)

【功能描述】：获取串口设备的属性。

【参数说明】：fd，串口设备的文件描述符

termios_p，用于保存串口属性

【返 回 值】：0，成功；-1，失败，errnor 给出具体错误码

3.3.2. tcsetattr()

【函数原型】：int tcsetattr(int fd, int optional_actions, const struct termios *termios_p)

【功能描述】：设置串口设备的属性。

【参数说明】：fd，串口设备的文件描述符

optional_actions，本次设置什么时候生效

termios_p，指向要设置的属性结构

【返 回 值】：0，成功（有一项参数成功也认为成功）；-1，失败，errnor 给出具体错误码

其中，optional_actions 的取值有：

TCSANOW：会立即生效

TCSADRAIN：当前的输出数据完成传输后生效，适用于修改了输出相关的参数

TCSAFLUSH：当前的输出数据完成传输，如果输入有数据可读但没有读就会被丢弃。

3.3.3. cfgetispeed()

【函数原型】：speed_t cfgetispeed(const struct termios *termios_p);

【功能描述】：返回串口属性中的输入波特率。

【参数说明】：termios_p，指向保存有串口属性的结构

【返 回 值】：波特率，取值是一组宏，定义在 terminos.h:

```
#define   B0   0000000        /* hang up，用来中断连接 */
#define   B50 0000001
#define   B75 0000002
#define   B110      0000003
#define   B134      0000004
#define   B150      0000005
#define   B200      0000006
#define   B300      0000007
#define   B600      0000010
#define   B1200     0000011
#define   B1800     0000012
#define   B2400     0000013
#define   B4800     0000014
#define   B9600     0000015
#define   B19200    0000016
```

```
#define   B38400   0000017
#define   B57600 0010001
#define    B115200 0010002
#define    B230400 0010003
#define    B460800 0010004
#define    B500000 0010005
#define    B576000 0010006
#define    B921600 0010007
#define   B1000000 0010010
#define   B1152000 0010011
#define   B1500000 0010012
#define   B2000000 0010013
#define   B2500000 0010014
#define   B3000000 0010015
#define   B3500000 0010016
#define   B4000000 0010017
```

3.3.4. cfgetospeed()

【函数原型】：speed_t cfgetospeed(const struct termios *termios_p);

【功能描述】：返回串口属性中的输出波特率。

【参数说明】：termios_p，指向保存有串口属性的结构

【返 回 值】：波特率，取值同 3.3.3

3.3.5. cfsetispeed()

【函数原型】：int cfsetispeed(struct termios *termios_p, speed_t speed);

【功能描述】：设置输入波特率到属性结构中

【参数说明】：termios_p，指向保存有串口属性的结构

speed，波特率，取值同 3.3.3

【返 回 值】：0，成功；-1，失败，errnor 给出具体错误码

3.3.6. cfsetospeed()

【函数原型】：int cfsetospeed(struct termios *termios_p, speed_t speed);

【功能描述】：设置输出波特率到属性结构中

【参数说明】：termios_p，指向保存有串口属性的结构

speed，波特率，取值同 3.3.3

【返 回 值】：0，成功；-1，失败，errnor 给出具体错误码

3.3.7. cfsetspeed()

【函数原型】：int cfsetspeed(struct termios *termios_p, speed_t speed);

【功能描述】：同时设置输入和输出波特率到属性结构中

【参数说明】：termios_p，指向保存有串口属性的结构

speed，波特率，取值同 3.3.3

【返 回 值】：0，成功；-1，失败，errnor 给出具体错误码

3.3.8. tcflush()

【函数原型】：int tcflush(int fd, int queue_selector);

【功能描述】：清空输出缓冲区、或输入缓冲区的数据，具体取决于参数 queue_selector。

【参数说明】：fd，串口设备的文件描述符

queue_selector，清空数据的操作

【返 回 值】：0，成功；-1，失败，errnor 给出具体错误码

参数 queue_selector 的取值有三个：

TCIFLUSH：清空输入缓冲区的数据

TCOFLUSH：清空输出缓冲区的数据

TCIOFLUSH：同时清空输入/输出缓冲区的数据

4. demo

此 demo 程序是打开一个串口设备，然后侦听这个设备，如果有数据可读就读出来并打印。设备名称、侦听的循环次数都可以由参数指定。

```
#include	<stdio.h>	/*标准输入输出定义*/
#include	<stdlib.h>	/*标准函数库定义*/
#include	<unistd.h>	/*Unix 标准函数定义*/
#include	<sys/types.h>
#include	<sys/stat.h>
#include	<fcntl.h>	/*文件控制定义*/
#include	<termios.h>	/*PPSIX 终端控制定义*/
#include	<errno.h>	/*错误号定义*/
#include <string.h>

enum parameter_type {
	PT_PROGRAM_NAME = 0,
	PT_DEV_NAME,
	PT_CYCLE,

	PT_NUM
};

#define DBG(string, args...) \
	do { \
		printf("%s, %s()%u---", __FILE__, __FUNCTION__, __LINE__); \
		printf(string, ##args); \
		printf("\n"); \
	} while (0)

void usage(void)
{
	printf("You should input as: \n");
	printf("\t select_test [/dev/name] [Cycle Cnt]\n");
}

int OpenDev(char *name)
{
	int	fd = open(name, O_RDWR );		//| O_NOCTTY | O_NDELAY
	if (-1 == fd)
		DBG("Can't Open(%s)!", name);

	return fd;
}
```

```
/**
*@brief   设置串口通信速率
*@param   fd        类型  int   打开串口的文件句柄
*@param   speed   类型  int   串口速度
*@return   void
*/
void set_speed(int fd, int speed){
      int     i;
      int     status;
      struct termios    Opt = {0};
      int speed_arr[] = { B38400, B19200, B9600, B4800, B2400, B1200, B300,
                  B38400, B19200, B9600, B4800, B2400, B1200, B300, };
      int name_arr[] = {38400,   19200,   9600,   4800,      2400,   1200,   300, 38400,
                  19200,   9600, 4800, 2400, 1200,   300, };

      tcgetattr(fd, &Opt);

      for ( i= 0;   i < sizeof(speed_arr) / sizeof(int);   i++) {
            if   (speed == name_arr[i])
                  break;
      }

      tcflush(fd, TCIOFLUSH);
      cfsetispeed(&Opt, speed_arr[i]);
      cfsetospeed(&Opt, speed_arr[i]);

      Opt.c_lflag   &= ~(ICANON | ECHO | ECHOE | ISIG);   /*Input*/
      Opt.c_oflag   &= ~OPOST;     /*Output*/

      status = tcsetattr(fd, TCSANOW, &Opt);
      if   (status != 0) {
            DBG("tcsetattr fd");
            return;
      }
      tcflush(fd, TCIOFLUSH);
}

/**
*@brief     设置串口数据位，停止位和效验位
*@param   fd        类型   int   打开的串口文件句柄
*@param   databits  类型   int  数据位     取值 为 7 或者 8
*@param   stopbits  类型   int  停止位     取值为 1 或者 2
*@param   parity   类型   int   效验类型  取值为 N,E,O,,S
*/
int set_Parity(int fd,int databits,int stopbits,int parity)
```

```
{
    struct termios options;

    if   ( tcgetattr( fd,&options)   !=   0) {
        perror("SetupSerial 1");
        return -1;
    }
    options.c_cflag &= ~CSIZE;

    switch (databits) /*设置数据位数*/
    {
    case 7:
        options.c_cflag |= CS7;
        break;
    case 8:
        options.c_cflag |= CS8;
        break;
    default:
        fprintf(stderr,"Unsupported data size\n");
        return -1;
    }

    switch (parity)
    {
    case 'n':
    case 'N':
        options.c_cflag &= ~PARENB;     /* Clear parity enable */
        options.c_iflag &= ~INPCK;      /* Enable parity checking */
        break;
    case 'o':
    case 'O':
        options.c_cflag |= (PARODD | PARENB); /*  设置为奇效验*/
        options.c_iflag |= INPCK;                   /* Disnable parity checking */
        break;
    case 'e':
    case 'E':
        options.c_cflag |= PARENB;        /* Enable parity */
        options.c_cflag &= ~PARODD;     /*  转换为偶效验*/
        options.c_iflag |= INPCK;           /* Disnable parity checking */
        break;
    case 'S':
    case 's':   /*as no parity*/
        options.c_cflag &= ~PARENB;
        options.c_cflag &= ~CSTOPB;break;
    default:
```

```
            fprintf(stderr,"Unsupported parity\n");
            return -1;
        }

        /*  设置停止位*/
        switch (stopbits)
        {
            case 1:
                options.c_cflag &= ~CSTOPB;
                break;
            case 2:
                options.c_cflag |= CSTOPB;
               break;
            default:
                 fprintf(stderr,"Unsupported stop bits\n");
                 return -1;
        }

        /* Set input parity option */
        if (parity != 'n')
            options.c_iflag |= INPCK;
        tcflush(fd,TCIFLUSH);
        options.c_cc[VTIME] = 150; /*  设置超时 15 seconds*/
        options.c_cc[VMIN] = 0; /* Update the options and do it NOW */
        if (tcsetattr(fd,TCSANOW,&options) != 0)
        {
            perror("SetupSerial 3");
            return -1;
        }
        return 0;
}

void str_print(char *buf, int len)
{
        int i;

        for (i=0; i<len; i++) {
            if (i%10 == 0)
                printf("\n");

            printf("0x%02x ", buf[i]);
        }
        printf("\n");
}
```

```
int main(int argc, char **argv)
{
	int i = 0;
	int fd = 0;
	int cnt = 0;
	char buf[256];

	int ret;
	fd_set rd_fdset;
	struct timeval dly_tm;		// delay time in select()

	if (argc != PT_NUM) {
		usage();
		return -1;
	}

	sscanf(argv[PT_CYCLE], "%d", &cnt);
	if (cnt == 0)
		cnt = 0xFFFF;

	fd = OpenDev(argv[PT_DEV_NAME]);
	if (fd < 0)
		return -1;

	set_speed(fd,19200);
	if (set_Parity(fd,8,1,'N') == -1)	{
		printf("Set Parity Error\n");
		exit (0);
	}

	printf("Select(%s), Cnt %d. \n", argv[PT_DEV_NAME], cnt);
	while (i<cnt) {
		FD_ZERO(&rd_fdset);
		FD_SET(fd, &rd_fdset);

		dly_tm.tv_sec	= 5;
		dly_tm.tv_usec = 0;
		memset(buf, 0, 256);

		ret = select(fd+1, &rd_fdset, NULL, NULL, &dly_tm);
//		DBG("select() return %d, fd = %d", ret, fd);
		if (ret == 0)
			continue;

		if (ret < 0) {
```

```
                printf("select(%s) return %d. [%d]: %s \n", argv[PT_DEV_NAME], ret, errno, strerror(errno));
                continue;
            }

            i++;
            ret = read(fd, buf, 256);
            printf("Cnt%d: read(%s) return %d.\n", i, argv[PT_DEV_NAME], ret);
            str_print(buf, ret);
        }

        close(fd);
        return 0;
}
```

5. 总结