Группа компаний

«ИНКОМ»

Микропроцессорный терминал "ВИП-МК"

Работа с COM-портами ВИП-МК из ОС Linux

Томск 2011

Оглавление

1 Введение

COM-порт (или порт RS232) является последовательным дуплексным интерфейсом для обмена данными. Общие сведения по стандарту на RS232 можно найти в интернете, например тут: http://ru.wikipedia.org/wiki/RS232 или (более подробно) тут: http://www.gaw.ru/html.cgi/txt/interface/rs232/.

В ВИП-МК реализованы COM-порты со следующими особенностями:

- имеется только асинхронный режим обмена данными;
- из линий управления все порты имеют RxD, TxD, DTR и RTS;
- поддержаны режимы передачи 5, 6, 7, 8 и 9-битных символов;

Данные особенности не мешают использовать использованию COM-портов ВИП-МК с различными устройствами, такими как GSM/GPRS-модемы, PSTN-модемы, автоматические метеостанции и другие устройства.

Кроме того, поддержка 5-битных символов позволяет подключать ВИП-МК к телеграфным аппаратам, где используется подобный формат символов.

В документе «РУКОВОДСТВО РАЗРАБОТЧИКА. Структура аппаратных средств микропроцессорного терминала ВИП-МК.» приведено назначение и особенности подключения каждого из четырёх, доступных в ВИП-МК, COM-портов.

2 Представление COM-портов в ОС Linux

COM-порты в Linux представлены как терминальные устройства в каталоге /dev/. Соответствие COM-портов устройствам в каталоге /dev/ показано в табл.1.

Табл. 1: Функции COM-портовВИП-МК

№	Порт	Режимы	Устройство	Примечание
1	COM0	Внутренний	**/dev/ttyS0**	Порт связи со встроенным терминалом ввода-вывода
2	COM1	Внешний	**/dev/ttyS1**	В режиме отладки может использоваться как системная консоль
3	COM2	Переключаемый	**/dev/ttyS2**	Разъём порта внутри корпуса ВИП-МК.
4	COM3	Переключаемый	**/dev/ttyS3**	Разъём порта внутри корпуса ВИП-МК или подключение радиомодема

3 Обращение к COM-портам в ОС Linux

Почти все устройства в ОС Linux представлены как специальные файлы. Поэтому обращение для чтения или записи к этим устройствам абсолютно

аналогично работе с файлами.

Работа с любым существующим файлом сводится к трём основным шагам:

- открыть файл (например функцей **open()** или **fopen()**);
- работать с содержимым файла (запись, чтение, изменение режима работы с файлом функцией fcntl()). Для файлов устройств существует дополнительный системный вызов ioctl(), который изменяет параметры устройства, например, скорость COM-порта или количество бит в символе;
- закрыть файл.

Пример:

```
/*
        Открыть файл COM-порта для чтения и записи.
        Файловый дескриптор поместить в переменную  ttyS0.
*/
int ttyS0=open("/dev/ttyS0",O_RDWR | O_NONBLOCK);
// Помещаем в  text_buf текстовую строку длиной  text_len.
char text_buf[0x100];
int text_len=sprintf(text_buf,"Тест COM-порта /dev/ttyS0.");
// Выводим текстовую строку в /dev/ttyS0
write(ttyS0,text_buf,text_len);
// Закрыть  файл COM-порта
close(ttyS0);
```

4 Программные режимы работы COM-порта в ОС Linux

Существует два основных режима работы COM-порта:

- режим с преобразованием входного и выходного потоков данных;
- режим без преобразования входного и выходного потоков данных (так называемый режим RAW то есть — только чтение и запись);

Режим с преобразованием входного и выходного потоков данных имеет множество настроек, относительно того как именно преобразуются данные и служит главным образом для поддержки выносных терминальных устройств, подключаемых по COM-порту напрямую или через модем. Примером такого устройства может служить эмулятор терминала HyperTerminal (ОС Windows) или эмуляторы терминалов Minicom, Putty (ОС Linux).

Режим без преобразования входного и выходного потоков данных (режим RAW) используется, когда необходимо передать байты между устройствами без всякой обработки. Заметим, что ничто не мешает использованию режима RAW для работы с выносными терминальными устройствами. В этом случае весь протокол обмена и преобразование данных реализуется на уровне программ пользователя.

5 *Вызовы ioctl() и библиотека termios*

Как уже говорилось, для настройки параметров устройств используется вызов ioctl. Но, поскольку режимов работы много, а терминальные устройства часто используются — была разработана и широко используется библиотека termios. Ниже почти все примеры приводятся с использованием библиотеки termios.

6 *Настройка режима работы COM-порта (режим RAW)*

```
// Подключаем заголовочный файл
#include <termios.h>
// Структура-описатель терминального устройства в библиотеке termios
struct termios settings;
// Открываем порт COM0
int ttyS0=open("/dev/ttyS0",O_RDWR | O_NONBLOCK);
// Получаем текущее состояние порта
tcgetattr(fd,&settings);
// Используем макрос cfmakeraw(), устанавливающий все настройки порта в
// структуре settings для raw-режима
cfmakeraw(&settings);
// Устанавливаем состояние порта
tcsetattr(fd,TCSANOW,&settings);
// Закрываем порт
close(fd);
```

7 *Настройка скорости COM-порта*

```
// Подключаем заголовочный файл
#include <termios.h>
// Структура-описатель терминального устройства в библиотеке termios
struct termios settings;
// Открываем порт COM0
int ttyS0=open("/dev/ttyS0",O_RDWR | O_NONBLOCK);
// Получаем текущее состояние порта
tcgetattr(fd,&settings);
// установить скорость вывода 19200 бит/сек
cfsetospeed(&settings, B19200);
// установить скорость ввода 19200 бит/сек
cfsetispeed(&settings, B19200);
// Устанавливаем состояние порта
tcsetattr(fd,TCSANOW,&settings);
// Закрываем порт
close(fd);
```

Заметим, что существуют две отдельные функции — установки скорости вывода и скорости ввода. Однако, в большинстве систем, в том числе и в ВИП-МК, эти скорости должны быть равны.

Доступные следующие стардартные константы скоростей: B50, B75, B110, B134, B150, B200, B300, B600, B1200, B1800, B2400, B4800, B9600, B19200, B38400, B57600, B115200, соответствующие скоростям от 50бит/сек до 115200бит/сек.

8 Настройка количества бит COM-порта

```
/*
   Установить количество бит в байте.
      fd     - дескриптор файла-терминала.
      bits   - количество бит в байте. (5, 6, 7, 8)

   Возвращаемое значение.
      Возвращает -1, если произошла ошибка. errno - код ошибки.
      Если ошибок нет - возвращает 0.
*/
int tty_SetBitsPerByte(int fd, int bits)
   {
      if(fd<0){return(-1);}
      //
      struct termios settings;
      //
      tcgetattr(fd,&settings);
      //
      settings.c_cflag &= ~(CS5 | CS6 | CS7 | CS8);
      //
      switch (bits)
         {
            case 5 :{settings.c_cflag |= CS5;break;}
            case 6 :{settings.c_cflag |= CS6;break;}
            case 7 :{settings.c_cflag |= CS7;break;}
            case 8 :{settings.c_cflag |= CS8;break;}
            default:
               return(-1);
         }
      //
      tcsetattr(fd,TCSANOW,&settings);
      return(0);
   }
```

В данном примере приведена функция, устанавливающая количество бит в передаваемом или принимаемом символе — от 5 до 8.

9 Использование команд форматного ввода-вывода printf(), scanf() и им подобных

В предыдущих примерах работа с портом осуществлялась системными вызовами ***open()***, ***close()***, возвращающими дескриптор файла (целое неотрицательное число).

Дескриптор файла требуется функциям библиотеки termios для идентификации файла-порта. При наличии дескриптора файла можно выводить информацию в порт функцией ***write()*** и считывать информацию из порта функцией ***read()***. Это не всегда удобно. Например, если порт используется для ввода-вывода форматированного текста, то удобнее использовать функции семейств ***printf()*** и ***scanf()***.

Функции ввода-вывода форматного текста являются частью стандартной библиотеки **glibc**. Библиотека **glibc** использует внутреннее представление файлов

в виде структуры типа **FILE**. В данной структуре помимо файлового дескриптора имеются буфера ввода-вывода и дополнительные флаги состояния файлов. Все функции форматного ввода-вывода, работающие непосредственно с файлами, используют для указания файла тип **FILE***.

Преобразовать дескриптор открытого файла в представление, необходимое библиотеке **glibc** позволяет функция **fdopen()**. Пример:

```
// Подключаем заголовочный файл
#include <termios.h>
#include <stdio.h>
#include <stdlib.h>
// Открываем порт COM0
int ttyS0=open("/dev/ttyS0",O_RDWR | O_NONBLOCK);
/*      Далее с дескриптором файла  ttyS0 производятся любые настроечные действия (установка режимов порта, скорости и так далее)
* /
….............
/*      Преобразуем дескриптор открытого файла в представление, необходимое библиотеке glibc функцией fdopen().      Поскольку порт открыт для чтения и записи, используем режим "r+".
* /
FILE* fttyS0=fdopen(ttyS0,"r+");
// Вывод текста в порт
fprintf(fttyS0,"Тестовый текст.\n Вывод целого [%i]\n",1234);
// Ввод текста из порта в буфер ввода.
char buf[1024];
fgets(buf, sizeof(buf), fttyS0);
// Печать введённого текста
fprintf(fttyS0,"Пользователь ввёл текст: [%s]\n", buf);
/*      Закрываем порт. Заметим, что поскольку мы используем для закрывания порта функцию fclose(), то функция close() не нужна.
*/
fclose(fttyS0);
```

Использование функций семейства ***printf()*** осуществляется одним из следующих способов:

- подготовка текста для вывода функцией **sprintf()** и вывод текста функцией в файл **write()**;
- вывод текста в файл оператором форматного вывода **fprintf()**;
- переопределение стандартного потока вывода программы и вывод текста в оператором форматного вывода **printf()**;

Использование функций семейства ***scanf()*** осуществляется одним из следующих способов:

- ввод текста из файла в буфер функцией ***read()*** и последующий разбор введённых данных функций ***sscanf()***;
- ввод и разбор введённых данных функций ***fscanf()***;
- переопределение стандартного потока ввода программы и вывод текста в оператором форматного вывода ***scanf()***;