Последовательный порт в MINIX3

Цилюрик О.И.

< olej@front.ru >

Редакция 3.11

от 29.11.2009

Оглавление

Относительно использования интерфейса RS-232 в MINIX3 описательная информация (документация, статьи, обсуждения). Поэтому освоение RS-232 в MINIX3 проведём комбинируя 2 метода: а). по аналогии с использованием RS-232 в Linux, и б). экспериментально находя соответствия этим аналогиям в системе MINIX3.

Некоторые описания ниже могут показаться избыточными. Так оно и есть, но это сделано сознательно, и на то есть несколько причин:

– Текст находится в развитии и дополнении, некоторые механизмы (такие как установление в Linux PPP соединений) уже включены в него с расчётом на более позднюю отработку подобной возможности в MINIX, и использование её в построении перекрёсных каналов MINIX3 — Linux.

– Завышенные в некоторых местах по объёму листинги выполнения команд сознательно оставлены в таком именно избыточном виде, чтобы, при воспроизведении этих технологий в отличающихся условиях, они могли служить образцом для сравнений в тех случаях когда что-то пойдёт не так как описывается.

Последовательный порт в Linux

Операции с последовательным портом в Linux нас должны интересовать не только как источник аналогий для MINIX, но и понадобятся при соединении по RS-232 компьютеров с различными ОС. Те принципы, которые мы увидим при соединении по RS-232 с Linux можно использовать практически без изменений и в OpenSolaris, QNX и других ОС. В Linux последовательные интерфейсы представлены как терминальные линии такого вида:

```
# ls -l /dev/ttyS*
crw-rw---- 1 root uucp 4, 64 Ноя 11 14:45 /dev/ttyS0
crw-rw---- 1 root uucp 4, 65 Ноя 11 13:43 /dev/ttyS1
crw-rw---- 1 root uucp 4, 66 Ноя 11 13:43 /dev/ttyS2
crw-rw---- 1 root uucp 4, 67 Ноя 11 13:43 /dev/ttyS3
```

Соединяем 2 последовательных порта, COM1 и COM2 в терминологии MS-DOS, последовательным 3-х проводным кросс-кабелем. Мы проделаем последовательный набор простейших тестовых испытаний, которые

позже по фазам будем воспроизводить в MINIX3. На комбинировании таких операций может быть построен реальный сколь угодно сложный обмен через последовательный канал.

Сырой поток байтовых данных

- запустив в 1-м терминале:

```
# cat /dev/ttyS0
```

- во 2-м можем выполнить

```
# echo 11111111111111111 > /dev/ttyS1
```

- после чего в 1-м увидим:

```
# cat /dev/ttyS0
11111111111111111
```

Передача текстового файла через последовательный канал:

- создадим тестовый файл:

```
# echo 'это новый текстовый файл' > ./1
```

- в 1-м терминале выполняем его передачу:

```
# cat ./1 > /dev/ttyS1
```

- во 2-м — приём:

```
# cp /dev/ttyS0 ./3
^C
```

- проверяем результат:

```
# ls -l
...
-rw-r--r-- 1 root root   46 Ноя 24 10:10 1
-rw-r----- 1 root root   47 Ноя 24 10:13 3
# cat 3
это новый текстовый файл
```

Примечание: результирующий файл (`./3`) на 1 байт длиннее исходного, это — дополнительный завершающий символ перевода строки (`'\n', 0xA`), добавленный операцией cat.

Управление параметрами потока

Прежде чем перейти к управлению параметрами потока, посмотрим их индикацию:

```
# stty -a  < /dev/ttyS0
speed 9600 baud; rows 0; columns 0; line = 0;
intr = ^C; quit = ^\; erase = ^?; kill = ^U; eof = ^D; eol = <undef>; eol2 = <undef>; swtch = <undef>; start = ^Q; stop = ^S; susp = ^Z;
rprnt = ^R; werase = ^W; lnext = ^V; flush = ^O; min = 1; time = 0;
-parenb -parodd cs8 hupcl -cstopb cread clocal -crtscts
-ignbrk -brkint -ignpar -parmrk -inpck -istrip -inlcr -igncr icrnl ixon -ixoff -iuclc -ixany -imaxbel -iutf8
opost -olcuc -ocrnl onlcr -onocr -onlret -ofill -ofdel nl0 cr0 tab0 bs0 vt0 ff0
isig icanon iexten echo echoe echok -echonl -noflsh -xcase -tostop -echoprt echoctl echoke
```

Или, если нас интересует только единичный параметр:

```
# stty speed  < /dev/ttyS0
9600
```

Теперь вернёмся к передаче через канал файла, как это делалось раньше, только передавать станем бинарный файл (файл произвольный):

```
# cp /bin/arch ./2
```

- передача:

```
# cat ./2 > /dev/ttyS1
```

- приём:

```
# cp /dev/ttyS0 ./4
```

- контроль:

```
# ls -l
...
-rwxr-xr-x 1 root root 4988 Ноя 24 10:10 2
-rw-r----- 1 root root  873 Ноя 24 10:30 4
```

Это типичное «не то», потому как линии (обе) находятся в каноническом (терминальном) режиме (некоторые байты потока интерпретируются как управляющие символы потока — приём завершается без нашего `^C`). Переведём линии в режим сырой передачи байт:

```
# stty raw < /dev/ttyS0
# stty raw < /dev/ttyS1
```

Примечание: при **установке** параметров в `stty`, так же как и при их чтении, знак перенаправления (`'<'`) не изменяется, это не ошибка.

- повторяем передачу (сделаем её на этот раз с контролем продолжительности):

```
# time cat ./2 > /dev/ttyS1
real    0m5.260s
user    0m0.002s
sys     0m0.007s
```

- приём:

```
# cp /dev/ttyS0 ./4
^C
```

- контроль:

```
# ls -l
...
-rwxr-xr-x 1 root root 4988 Ноя 24 10:10 2
-rw-r----- 1 root root 4988 Ноя 24 11:07 4
```

Вот теперь ожидаемое совпадает с действительным. Сделаем «обратным осчётом» грубую оценку скорости передачи по данным: 4988 * 8 / 5.26 = 7586 Бод, или с учётом стартового и стопового битов: 4988 * 10 / 5.26 = 9482 Бод, что очень хорошо (я думаю, в пределах точности измерения) совпадает со скоростью линии. Характерно, что процессорное время операции составляет <0.17% (0.09 * 100 / 5.26) общей её протяжённости, общее время определяется, в основном, продвижением информации по низкоскоростному каналу.

Изменим скорость линии:

```
# stty 38400 < /dev/ttyS0
# stty speed < /dev/ttyS0
38400
# stty 38400 < /dev/ttyS1
```

- передача:

```
# time cat ./2 > /dev/ttyS1
real    0m1.409s
user    0m0.000s
sys     0m0.008s
```

Время передачи данных уменьшилось в 4 раза.

Ещё раз изменим скорость:

```
# stty 115200 < /dev/ttyS0
# stty speed < /dev/ttyS0
115200
# stty 115200 < /dev/ttyS1
# time cat ./2 > /dev/ttyS1
real    0m0.493s
user    0m0.002s
sys     0m0.004s
```

SLIP инкапсуляция

SLIP инкапсуляция IP пакетов — это самый старый способ передачи IP в последовательную линию, настолько старый, что он бы и не заслуживал отдельного рассмотрения, если бы в MINIX3 он не оставался основной реализацией. Итак, устанавливаем SLIP канал, как и раньше 2 последовательных порта (COM1 и COM2 в терминологии MS-DOS) соединены последовательным 3-х проводным нуль модемным кабелем:

- в 1-м терминале:

```
# slattach -s 4800 -p slip /dev/ttyS0
...
```

- во 2-м терминале:

```
# slattach -s 4800 -p slip /dev/ttyS1
...
```

Теперь оба эти терминала заблокированы на операциях, а через нуль модемный кабель установлено SLIP соединение. Присваиваем IP адреса обоим концам этого соединения:

```
# ifconfig sl0 192.168.5.1
# ifconfig sl1 192.168.5.2
```

И убеждаемся в наличии такого соединения, и в готовности его для выполнения типовых сетевых операций:

```
# ifconfig sl0
sl0       Link encap:Serial Line IP
          inet addr:192.168.5.1  P-t-P:192.168.5.1  Mask:255.255.255.255
          UP POINTOPOINT RUNNING NOARP MULTICAST  MTU:296  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)
# ifconfig sl1
sl1       Link encap:Serial Line IP
          inet addr:192.168.5.2  P-t-P:192.168.5.2  Mask:255.255.255.255
          UP POINTOPOINT RUNNING NOARP MULTICAST  MTU:296  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)
# ping 192.168.5.1
PING 192.168.5.1 (192.168.5.1) 56(84) bytes of data.
```

```
64 bytes from 192.168.5.1: icmp_seq=1 ttl=64 time=0.197 ms
64 bytes from 192.168.5.1: icmp_seq=2 ttl=64 time=0.168 ms
64 bytes from 192.168.5.1: icmp_seq=3 ttl=64 time=0.167 ms
--- 192.168.5.1 ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 1999ms
rtt min/avg/max/mdev = 0.167/0.177/0.197/0.017 ms
```

Устанавливаем SSH сеанс (пользовательского уровня) сквозь этот SLIP канал:

```
# ssh 192.168.5.1
root@192.168.5.1's password:
Last login: Thu Nov 26 10:05:01 2009 from 192.168.5.1
# whoami
root
# who
...
olej     pts/11       2009-11-26 09:25 (:0)
root     pts/13       2009-11-26 10:05 (192.168.5.1)
# exit
```

Если мы во время этого сеанса SLIP опросим скорость RS-232 порта:

```
# stty speed -F /dev/ttyS0
4800
```

- то видим, как происходит изменение физической скорости, определённое командой `slattach`. Завершаем эту команду, и тем самым разрушаем SLIP канал о стороны соответствующего RS-232 порта, набирая [Ctrl]+[C] в терминале, откуда команда набиралась. Если мы после этого проверим скорость RS-232 порта (`stty`, как показано выше), то убедимся в том, что скорость установлена 4800 — это (изменения скоростей после операций) нужно обязательно учитывать (и восстанавливать) при экспериментах с каналами над RS-232.

PPP инкапсуляция

Аналогично предыдущему случаю устанавливаем и простейший PPP канал:

- в 1-м терминале:

```
# pppd ttyS0 local noauth nodetach passive
Using interface ppp0
Connect: ppp0 <--> /dev/ttyS0
...
```

- присвоим IP адрес появившемуся сетевому интерфейсу:

```
# ifconfig ppp0 192.168.4.1
```

Смотрим результат:

```
# ifconfig ppp0
ppp0    Link encap:Point-to-Point Protocol
        inet addr:192.168.4.1  P-t-P:192.168.4.1  Mask:255.255.255.255
        UP POINTOPOINT RUNNING NOARP MULTICAST  MTU:1500  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:3
        RX bytes:0 (0.0 b)  TX bytes:0 (0.0 b)
```

- со 2-го терминала выполняем то же для инициализации другого конца линии:

```
# pppd ttyS1 local noauth nodetach passive
Using interface ppp0
Connect: ppp1 <--> /dev/ttyS1
LCP: timeout sending Config-Requests
...
# ifconfig ppp0 192.168.4.2
```

И наслаждаемся всем спектром сетевых приложений на вновь созданном линке:

```
# ping 192.168.4.1
PING 192.168.4.1 (192.168.4.1) 56(84) bytes of data.
64 bytes from 192.168.3.1: icmp_seq=1 ttl=64 time=0.282 ms
64 bytes from 192.168.3.1: icmp_seq=2 ttl=64 time=0.205 ms
64 bytes from 192.168.3.1: icmp_seq=3 ttl=64 time=0.206 ms
64 bytes from 192.168.3.1: icmp_seq=4 ttl=64 time=0.207 ms
...
# ssh -l olej 192.168.4.1
olej@192.168.4.1's password:
$ who
...
olej     pts/6    2009-11-05 21:56 (:0)
olej     pts/8    2009-11-05 22:38 (:0)
olej     pts/10  2009-11-05 22:46 (192.168.4.1)
...
```

- обратите внимание, что последняя команда (who) выполнена уже из SSH терминала.

Последовательный порт в MINIX3

Информации об использовании RS-232 в MINIX3 нет нигде в описаниях (по крайней мере, в качестве использования аппаратного интерфейса, а не терминальной линии). Поэтому эту информацию можно извлечь только из отрывочных фраз в man-ах, и именно поэтому самые ключевые фразы, даже при их объёмности, я здесь цитирую:

```
# man tty
...
The list below shows all devices that Minix and Minix-vmd have. Not all
of these devices are configured in by default, as indicated by the
numbers (i/j/k, l/m/n) that tell the minimum, default and maximum
possible number of these devices for Minix (i/j/k) and Minix-vmd (l/m/n).
   ...
   /dev/tty0[0-3] Serial lines. (0/2/2, 4/4/4)

# man term
...
The program first sets the baudrate, parity and character length, and
then forks. The parent sits in a loop copying from stdin (usually the
console's keyboard), to the terminal or modem (/dev/tty00). The child
sits in a loop copying from the terminal or modem (/dev/tty00) to
standard output.
...
```

```
Term keeps the modem line open
on file descriptor 9 while running the subshell, so you can type
               <&9 >&9
at the end of your ZMODEM command to connect it to the modem.
...
Important note: to use term, it is essential that /etc/ttytab is
configured so that there is no login session started on the modem line.
If there is, both the login session and term will try to read from the
modem, and nothing will work.
```

Но для полного понимания того, что будет излагаться далее, эти man-ы, и не названные здесь, но перечисленные в конце текст — должны быть вычитаны досконально.

Итак, в MINIX3 последовательные интерфейсы RS-232 отображаются в /dev/tty0[0-3]:

```
# ls -l /dev/tty0*
crw-rw-rw- 1 root  tty  4,  16 Nov  5 10:00 /dev/tty00
crw-rw-rw- 1 root  tty  4,  17 Nov  5 10:00 /dev/tty01
crw-rw-rw- 1 root  tty  4,  18 Nov  5 10:00 /dev/tty02
crw-rw-rw- 1 root  tty  4,  19 Nov  5 10:00 /dev/tty03
```

Ключевым местом цитируемых текстов является предупреждение относительно /etc/ttytab, комментируем строки относительно tty00 и tty01, записанные туда по умолчанию при установке с LiveCD, без этого нашего действия обе в этом терминальные линии будут находиться в ожидании logon бесконечно долго, должно стать:

```
# cat /etc/ttytab
# ttytab - terminals
#
# Device     Type            Program    Init
console      minix           getty
ttyc1        minix           getty
ttyc2        minix           getty
ttyc3        minix           getty
#tty00        unknown
#tty01        unknown
...
```

После правки перегружаем систему, чтобы обновить состояние терминальной подсистемы:

```
# shutdown now
```

Следующим интересующим нас пунктом будет программ term:

- в 1-й консоли:

```
# term /dev/tty00
Connected to /dev/tty00, command key is CTRL-], type ^] for help
```

- во 2-й консоли:

```
# term /dev/tty01
Connected to /dev/tty01, command key is CTRL-], type ^] for help
```

Примечание: выведенная подсказка записана так по неаккуратности разработчиков («CTRL-]» и « ^]» в ней — одно и то же), и выглядеть это должно так:

```
# term /dev/tty00
```

```
Connected to /dev/tty00, command key is CTRL-], type ^]? for help
```

<Ctrl>+<]><s>

```
Term commands:
 ? - this help
 s - subshell (e.g. for file transfer)
 h - hangup (+++ ATH)
 b - send a break
 q - exit term
^] - send a CTRL-]
```

Правильно записывать эти управляющие комбинации было бы, наверное, так: `<Ctrl>+<]s>` - при нажатом CTRL нажать '`]`', а затем (отпустив CTRL) нажать '`s`' ... и так далее.

- теперь на 1-й консоли набираем: "12345", а на 2-й наблюдаем:

 # term /dev/tty01

```
Connected to /dev/tty01, command key is CTRL-], type ^] for help
12345
```

Мы получили контакт через нуль модемный кабель, но пока это мало интересно, поскольку контакт происходит меж двумя экземплярами программы `term`. Но у `term` есть особо интересующий нас режим: запустить дочернюю shell-оболочку - `<Ctrl>+<]><s>` . Повторяем сеанс передачи данных:

- на 1-й текстовой консоли:

 # term /dev/tty00

<Ctrl>+<]><s>

#

консоль перешла в дочерний shell, но в этом shell открыт на чтение-запись дескриптор, связанный с линией `/dev/tty00` (COM1);

- на 2-й текстовой консоли:

 # term /dev/tty01

<Ctrl>+<]><s>

 # tee <&9

```
...
```

то же, но дескриптор 9 этой консоли связан с линией `/dev/tty01` (COM2), здесь же показана следующая команда, ожидающая приёма;

- на 1-й текстовой консоли выполняем передачу:

 # echo qwerty >&9

- и на 2-й, приёмной, текстовой консоли получаем:

```
...
```

 # tee <&9

```
qwerty
```

Мы полностью воспроизвели возможность передачи потока данных, ранее показанную в варианте для Linux; причём, как и в случае Linux, мы обеспечиваем передачу данных командами POSIX, а не специальными программами типа `term`, манипулирующими с последовательными каналами.

Отступление: так какой же кабель вы используете?

В предыдущем разделе мы организовали, пусть и простейший, обмен сквозь RS-232 канал. Здесь самое время остановиться и осмотреться. А что делать, если даже этот элементарный обмен у вас не воспроизводится в том виде как он описан? С большой вероятностью это может быть именно так. Суть в том, что большинство систем (MS DOS, Windows, Linux, QNX) поддерживают обмен через нуль модемный кабель, который известен как 3-х проводное соединение. MINIX3 **не обеспечивает** канал через 3-х проводное соединение. Я не знаю, работает ли MINIX3 с кабелем 5-ти проводного соединения, но все дальнейшее рассмотрение я провожу **только** к случаю

соединения RS-232 портов кабелем с 7-ми проводным соединением. На 3-х проводной линии терминальная система будет нормально **принимать** байтовый поток с RS-232, но при **передаче** будет возвращать управление так, как операция выполнена, фактически ничего не передавая.

Примечание: именно из-за актуальности этой особенности в MINIX3, вопрос рассмотрения вариантов аппаратной реализации кабеля вынесен отдельным разделом в конец текста.

Всё последующее рассмотрение этого раздела мы посвятим разбору совершенно частного вопроса: а почему же не поддерживается 3-х проводное соединение исходя из анализа исходного кода в файле `/usr/src/drivers/tty/rs232.c` в файловой системе MINIX3 (именно в этом, и только в этом, файле сосредоточено **всё**, что относится к работе с RS-232). Всех, кого не интересует такой уровень детализации (а таковых, безусловно, большинство) могут безболезненно пропустить текст до конца раздела без всякого ущерба для дальнейшего понимания.

Насколько я понимаю, эта особенность использования RS-232 линии отражена вот в этих фрагментах (приводится в неизменном виде, фрагменты только переставлены местами):

```
/* Line status bits. */
#define LS_OVERRUN_ERR          2
#define LS_PARITY_ERR           4
#define LS_FRAMING_ERR          8
#define LS_BREAK_INTERRUPT   0x10
#define LS_TRANSMITTER_READY 0x20
...
PRIVATE void rs_ostart(rs)
rs232_t *rs;                    /* which rs line */
{
/* Tell RS232 there is something waiting in the output buffer. */
  rs->ostate |= OQUEUED;
  if (txready(rs)) out_int(rs);
}
...
/* Macro to tell if transmitter is ready. */
#define txready(rs) (my_inb(rs->line_status_port) & LS_TRANSMITTER_READY)
```

Примечание: код `rs_ostart()` выглядит действительно несколько странно — если условие готовности выполняется, то производится операция вывода, а если не выполняется ... то происходит выход как и при нормальной операции.

Регистр RS-232, с которым оперирует макрос `txready()` — это [6]:

Внутренний регистр состояния линии (БА+5): (только чтение)

А флаг `LS_TRANSMITTER_READY` это [7]:

бит 5 – готовность передатчика для записи;

Переписав этот фрагмент, как я понимаю, вы вполне можете научить MINIX3 понимать 3-х проводную линию.

ASCII поток через RS-232 канал

Возвращаемся к передаче данных. Следующий шаг — передача ASCII файлов через канал RS-232. Заготавливаем эталонный файл `test` для передачи:

```
# echo 'this is new file' > test
```

Выполням (в тех же предварительно созданных после команды `term` дочерних shell, о которых шла речь ранее):

- на 2-й (приёмной) консоли:

```
# tee <&9 >test1
```

- и выполняем на 1-й (передающей) консоли:

```
# cat test >&9
```

- и по:

```
# ls -l
```

убеждаемся в идентичности `test` и `test1`.

Бинарная информация и управление параметрами

Используем в качестве тестового бинарного потока файл (выбран произвольно):

```
# cp /bin/loadkeys ./test
# ls -l test
-rw-rw-rw- 1 root  operator  16064  Nov 24 10:00 test
```

- на 2-й текстовой консоли:

```
# term /dev/tty01
<Ctrl>+<]><s>
# tee <&9 >test1
```

- на 1-й текстовой консоли выполняем передачу:

```
# term /dev/tty00
<Ctrl>+<]><s>
# time cat test >&9
15.38
```

- сверяем идентичность ./test и ./test1 по длине

```
# ls -l test*
```

- и по содержимому

```
# cmp test test1
```

Всё идентично!

Теперь изменяем скорость канала. Но для этого (на каждой консоли) нужно последовательно выйти и из дочернего shell, и из `term`:

```
# exit
[back to term]
<Ctrl>+<]><q>
```

После чего как ранее открываем сессии но на другой скорости:

```
# term 8 38400 /dev/tty01
<Ctrl>+<]><s>
```

И проводим такое же тестирование по передаче файла `test`. На скорости 38400 (время передачи 4.00) идентичность (у меня) не сохранилась, как и на скорости 115200 (время передачи 3.28). На скорости 19200 (время передачи 7.98) передача завершилась нормально. Возможно, это определяется и конструктивными особенностями нуль модемного кабеля.

Опять относительно скорости потока (с учётом старт-стоповых бит): 16064 * 10 / 15.38 = 10444 Бод — при установленной физической скорости канала 9600, и 16064 * 10 / 7.98 = 20130 Бод — при установленной физической скорости канала 19200 Бод (расхождения не такие значительные и определяются, вероятно, неточностью измерения времени). По крайней мере, хорошо видно, что скорость потока определяется

установками RS-232 и никак не скоростью компьютера.

Последовательный порт и QEMU

Было бы в высшей степени удобно выполнять MINIX3 под системой виртуализации QEMU так, чтобы сохранить доступ к последовательным портам. Но эта возможность требует проверки. Загрузим виртуальную машину (гостевую систему) MINIX3 из реальной установки в раздел диска так:

```
# qemu -hda /dev/hda -serial /dev/ttyS0 -serial /dev/ttyS1 -boot c
```

Загрузка гостевой системы происходит через загрузчик GRUB, в меню выберем «Minix 3»... После загрузки и регистрации выполним:

- на 2-й текстовой консоли, приёмной (переход в неё <правый Alt>+<F2>):

```
# term /dev/tty01
<Ctrl>+<]><s>
# tee <&9
```

- на 1-й текстовой консоли, передающей (переход в неё <правый Alt>+<F2>):

```
# term /dev/tty01
<Ctrl>+<]><s>
# echo qwerty >&9
```

- после чего на 1-й консоли видим:

```
...
# tee <&9
qwerty
```

Связь между компьютерами

Интерес поставленного вопроса здесь состоит не в том, чтобы связать физически разнесённые компьютеры, а в том, чтобы эти компьютеры работали под управлением различных операционных систем, и имели возможность обмениваться информацией между разными системами.

Соединим тем же нуль модемным кабелем (ещё раз обращаю внимание, и читаем внимательно изложенную выше разницу между 3-х проводным и 7-ми проводным соединениями для MINIX3) последовательные порты 2-х разных компьютеров, для организации передачи потока между ними. Проблемой здесь может стать то обстоятельство, что, если компьютеры имеют по 2 COM порта (COM1 и COM2), то не всегда просто идентифицировать какой из портов подключен к какому из внешних разъёмов (определяется шлейфами внутри корпуса системного блока). Итого, у нас может быть 4 варианта соединения (1<->1, 1<->2, 2<->1, 2<->2). Если у вас случилась такая ситуация, и отсутствует желание вскрывать системный блок, то есть очень простой способ разрешить этот конфликт прежде чем двигаться дальше:

1. Контролируем соединение в Linux (не в MINIX3), где работа с последовательным портом проще, и гораздо лучше контролируема.

2. Соединяем один из последовательных портов (который мы **предполагаем** является COM1) одного компьютера, с последовательным портом другого (который мы также **предполагаем** как COM1).

3. В 2-х различных терминалах одного (приёмного) компьютера вводим команды:

```
# cat < /dev/ttyS0
...
```

и

```
# cat < /dev/ttyS1
...
```

Оба терминала заблокированы в ожидании ввода.

4. С терминала (можно с одного терминала, последовательно) другого компьютера выполняем:

```
# echo 'this send from COM1' > /dev/ttyS0
# echo 'this send from COM2' > /dev/ttyS1
```

Поскольку у нас используется только один нуль модемный кабель (я надеюсь), и одно соединение, то только на одном из терминалов приёмного компьютера может быть увиден отклик. Я в своём эксперименте наблюдаю:

```
# cat < /dev/ttyS1
this send from COM1
```

- что сразу указывает на то, что моё соединение соответствует 1<->2. Некоторую подобную проверку я рекомендую провести **в любом случае**, прежде чем переходить к обмену с MINIX3 — здесь мы уже независимо проверили обмен данными Linux<->Linux, и корректность всех элементов тракта передачи (а этих элементов, к сожалению, слишком много: кабель, контакты, драйверы, программы обмена ... и в каждом звене на каждом конце может возникнуть несоответствие).

Напоминание: если вы станете использовать нуль модемный кабель с 3-х проводной схемой, то в Linux все описанные выше эксперименты пройдут успешно. Но как только вы перейдёте к MINIX3, то по такому кабелю будет обеспечиваться **приём**, но не будет осуществляться **передача**.

Прямая связь по потоку данных

В описываемых мной далее экспериментах, таким точно образом было установлено соответствие подключения: COM1 в компьютере с MINIX к COM2 в компьютере с Linux. Предварительно вы можете выполнить проверки на посимвольную передачу (`term`, в точности так, как это было показано выше в описании относительно связи 2-х портов в MINIX3). Я же сразу покажу только проверку возможности передачи файла (передаём файл `/usr/src/drivers/tty/rs232.c`) с контролем идентичности полученного результата.

Со стороны MINIX3 хоста:

```
# cd  /usr/src/drivers/tty
# ls -l rs232.c
-rw-r--r-- 1 bin  operator  26068 Apr  6  2009 rs232.c
# term /dev/tty00
<Ctrl>+<]><s>
# time cat rs232.c >&9
26.36 real ...
```

Со стороны Linux хоста:

```
# cat </dev/ttyS1 >rs_232.c
<Ctrl>+<C>
# ls -l rs_232.c
-rw-rw-rw- 1 root root 26068 Ноя 28 23:03 rs_232.c
```

Инкапсуляция RS-232 потока в SLIP

Куда интереснее прямой RS-232 передачи инкапсуляция потока RS-232 в поток IP пакетов — это расширяет зону достижимости хоста до всего пространства Интернет. Сделаем это через SLIP протокол:

1. В Linux SLIP запускается примерно так (там может быть ещё множество параметров, которые уточняем в `man`):

```
# slattach -p slip /dev/ttyS1
```

....

2. Смотрим что изменилось у нас с сетевыми интерфейсами:

```
# ifconfig sl0
sl0       Link encap:Serial Line IP
          POINTOPOINT NOARP MULTICAST  MTU:296  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)
```

- появился новый интерфейс `sl0.`

3. Новый сетевой интерфейс (`sl0`) создан, но у него пока ещё нет IP адреса, присвоим ему IP адрес:

```
# ifconfig sl0 192.168.6.6
# ifconfig sl0
sl0       Link encap:Serial Line IP
          inet addr:192.168.6.6  P-t-P:192.168.6.6  Mask:255.255.255.255
          UP POINTOPOINT RUNNING NOARP MULTICAST  MTU:296  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)
```

4. Но создание сетевого ещё мало — нам нужно обеспечить отправку любых пакетов для посети этого интерфейса 192.168.6.* (роутинг) именно через интерфейс `sl0`, для правим таблицу роутинга (добавляем запись подсети):

```
# route -v  add -net 192.168.6.0 netmask 255.255.255.0 gw 192.168.6.6
# route
Kernel IP routing table
Destination     Gateway         Genmask         Flags Metric Ref    Use Iface
192.168.1.0     *               255.255.255.248 U     0      0        0 eth0
192.168.6.0     192.168.6.6     255.255.255.0   UG    0      0        0 sl0
169.254.0.0     *               255.255.0.0     U     0      0        0 eth0
default         192.168.1.1     0.0.0.0         UG    0      0        0 eth0
```

К этому моменту мы подготовил канал SLIP со стороны Linux, но нам ещё необходим встречный канал со стороны MINIX3:

1. Запускаем SLIP интерфейс и привязываем его к последовательному каналу (COM1) `/dev/tty00`:

```
# term /dev/tty00
<Ctrl>+<]><s>
# slip /dev/psip1 <&9 >&9
```

Примечание: для справки - сетевые интерфейсы при старте MINIX3 хоста были определены в файле `/etc/inet.conf` как:

```
# cat /etc/inet.conf
```

```
eth0 dp8390 0 ;
psip1 { default; } ;
```

Нас здесь интересует интерфейс `/dev/psip1`, он же `/dev/ip1` :

```
# ls /dev/*ip*
/dev/ip   /dev/ip0   /dev/ip1   /dev/ipstat   /dev/psip   /dev/psip1
```

2. Связываем сетевой интерфейс (`-I`) `/dev/ip1` (`/dev/psip1`) с IP адресом (адресом интерфейса `-h`) 192.168.6.4:

```
# ifconfig -I /dev/psip1 -h 192.168.6.4
# ifconfig -av
/dev/ip1: address 192.168.6.4 netmask 255.255.255.0 mtu 576
```

3. Добавляем роутинг для этого интерфейса на хост Linux:

```
# add_route -g 192.168.6.4
```

Позже, убедившись что это работает, мы можем вписать эту команду, вместе с предыдущим `ifconfig`, в стартовый конфигурационный файл `/etc/rc.net`.

```
# pr_routes
ent #   if               dest         gateway dist  pref  mtu flags
    0  ip1         0.0.0.0/0      192.168.6.4    1     0    0 static
```

Вот только к этому времени у нас IP канал сквозь нуль модемный RS-232 кабель уже установлен. На Linux хосте выполняем:

```
# ping 192.168.6.4
PING 192.168.6.4 (192.168.6.4) 56(84) bytes of data.
64 bytes from 192.168.6.4: icmp_seq=1 ttl=96 time=184 ms
64 bytes from 192.168.6.4: icmp_seq=2 ttl=96 time=183 ms
64 bytes from 192.168.6.4: icmp_seq=3 ttl=96 time=183 ms
--- 192.168.6.4 ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 2000ms
rtt min/avg/max/mdev = 183.945/184.073/184.326/0.526 ms
```

И далее:

```
# ssh 192.168.6.4
root@192.168.6.4's password:
Last login: Sun Nov 29 11:37:08 2009
...
# uname -a
Minix 192.168.6.4 3 1.5 i686
# ping 192.168.6.6
192.168.6.6 is alive
```

Для убедительности (передачи достаточно объёмного файла), в этой же сессии SSH можем выполнить что-то типа следующего:

```
# cd /usr/src/drivers/tty
# ls -l rs232.c
-rw-r--r-- 1 bin  operator  26068 Apr  6  2009 rs232.c
# cat rs232.c
```

```
...
```

- все эти операции выполнялись с хоста MINIX3 (хотя сами команды и вводились в SSH сеансе с хоста Linux).

Примечание: полученный IP канал несколько «со странностями», например, со стороны Linux:

```
# traceroute 192.168.6.4
traceroute to 192.168.6.4 (192.168.6.4), 30 hops max, 40 byte packets
send: Недостаточно буферного пространства
```

Однако его вполне достаточно для подключения сетевых клиентов и выполнения пользовательских операций, таких как передача файлов, например, что и было продемонстрировано в листингах выше.

В принципе, `man` рекомендует несколько другой способ запуска `slip`, воспользуемся ним:

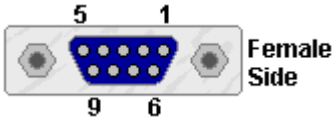
```
# ( stty raw ; slip /dev/psip1 ) </dev/tty00 >/dev/tty00
...
```

Вот, достаточно показательный, как мне кажется, выполненный после такого подключения сеанс SSH, «туда и обратно»:

```
# ssh 192.168.6.4
root@192.168.6.4's password:
Last login: Sun Nov 29 16:16:38 2009 from 192.168.6.6
...
# uname
Minix
# ssh 192.168.6.6
root@192.168.6.6's password:
Last login: Sun Nov 29 18:12:03 2009 from 192.168.6.4
# uname
Linux
...
# exit
Connection to 192.168.6.6 closed.
#
```

Нуль модемные кабели

DB9 розетка (мама), нумерация контактов со стороны подключаемой части (со стороны распайки кабеля — обратная, слева-направо для 1...5):



Контакт	Обозн.	Направление	Описание
1	CD	<--	Carrier Detect
2	RXD	<--	Receive Data
3	TXD	-->	Transmit Data
4	DTR	-->	Data Terminal Ready
5	GND	---	System Ground
6	DSR	<--	Data Set Ready

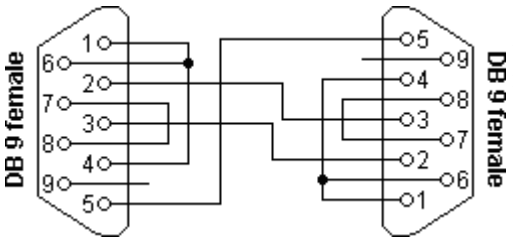
```
7           RTS         -->         Request to Send
8           CTS         <--         Clear to Send
9           RI          <--         Ring Indicator
```

В отношении нуль модемных кабелей всегда существовали такие мнения:

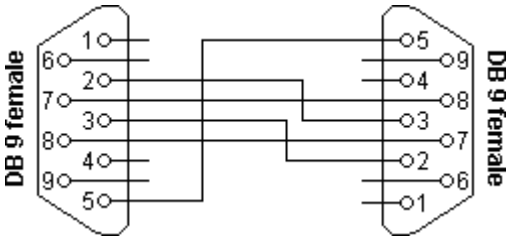
– Можно спаять кабель нескольких разных конфигураций (3-х проводное соединение, 5-ти проводное соединение, 7-ми проводное соединение, различные упрощения того же 3-х проводного соединения).

– Там, где работает 3-х проводное соединение — всегда будет работать 5-ти проводное соединение; а там, где работает 5-ти проводное соединение — всегда будет работать 7-ми проводное соединение (но не наоборот).

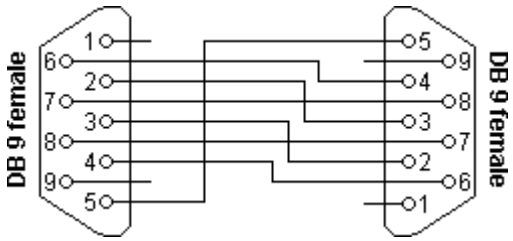
В литературе утверждается, что в 90-95% систем и устройств вполне удовлетворительно работает 3-х проводное соединение (MINIX3, как выясняется, не попадает в это число).



3-х проводное соединение. В литературе встречаются отдельные утверждения, что 3-х проводное соединение не должно использоваться на скоростях не выше 19200.



5-ти проводное соединение



полное 7-ми проводное соединение

Примечание: в практике используются и дальнейшие вариации даже этого набора схем: упрощение 3-х проводной схемы [7], и усложнение 7-ми проводной [9].

Стандарт RS-232 не нормирует максимальную длину соединения (нуль модемного кабеля. В различных источниках чаще всего называется максимальная длина 15-20 метров, некоторые источники [8] утверждают, что нормальная работоспособность сохраняется до длин в 50 и даже 100 метров, а на пониженных скоростях — и свыше 900 метров.

Более детально о RS-232, и о соединениях см. [2] и другие перечисленные источники, рисунки и схемы соединений заимствованы из [2].

Дополнительные источники информации

1. Руководство FreeBSD. ч.17 Последовательные соединения

http://www.nbuv.gov.ua/books/2004/freebsd/serial.html

2. Обзор стандарта RS-232

http://www.gaw.ru/html.cgi/txt/interface/rs232/start.htm

- здесь описаны разъёмы DB9, DB25, схемы нуль-модемных кабелей: 3-х, 5-ти и 7-ми проводных.

3. `man serial-ip`

http://minix1.woodhull.com/current/2.0.4/wwwman/man8/serial-ip.8.html

- такой команды или файла нет, это общие рассуждения о том, как поднять IP над последовательной линией.

4. `man tty :`

http://minix1.woodhull.com/current/2.0.4/wwwman/man4/tty.4.html

5. `man` по команде `term`

http://minix1.woodhull.com/current/2.0.4/wwwman/man1/term.1.html

6. Использование RS-232 для передачи данных

http://www.nf-team.org/drmad/stuff/lvrs232.htm

- аппаратные регистры интерфейса RS-232

7. Программирование на аппаратном уровне :: RS-232

http://bugtraq.ru/library/programming/rs232.html

8. Последовательный асинхронный адаптер (COM порт)

http://cxem.net/comp/comp38.php

Примечания: по всей видимости, за период времени между версиями MINIX 2.0.4 и 3.1.5, состояние программной поддержки последовательных линий нисколько не изменилось (по крайней мере я не обнаружил таких отличий). Вероятно, это определяется тем, что терминальная подсистема — одна из старейших и отработанных частей, начиная с ранних версий системы. Поэтому я даю URL на `man` страницы для чтения на WWW ресурс версии MINIX 2.04.