I. Основы работы с NetCore. Краткое руководство.

Содержание

О NetCore

NetCore предназначен для интеграции в локальную сеть оборудования, имеющего интерфейсы RS232, RS485, или USB. Устройство позволяет обрабатывать и накапливать потоки, поступающие по последовательным интерфейсам, и передавать их в локальную сеть.

NetCore обладает мощными вычислительными ресурсами на основе ОС Linux, позволяющими выполнять с поступающими потоками прикладные задачи (вести базы данных, производить первичную обработку информации). В частности, в проекте "интеллектуальное здание" устройство используется для сбора информации с электро- и теплосчетчиков, а один из каналов контролирует работу лифтового хозяйства. Интерфейс оператора реализован в виде html страницы, которая хранится непосредственно во флеш-памяти NetCore.

Интерфейсы:	**2xRS232, 2xRS485, USB-A, USB-B, Fast Ethernet 10/100.**
Процессор:	**AMD Alchemy 266 MHz, MIPS32 (TM)**
Память:	**64M SDRAM, 4m Flash.**
Внутренняя ОС:	**Linux 2.4**
Потребляемая мощность при 12В:	**3Вт**

Программирование для NetCore

Основным языком программирования для NetCore является язык С. На данный момент доступны инструменты, которые позволяют компилировать программы только из среды GNU/Linux. Процесс написания программ и компиляции имеет некоторые особенности, связанные с тем, что NetCore содержит процессор AMD Alchemy, который основан на отличной от x86 архитектуре MIPS32. Поэтому для сборки программ нужно использовать специальную версию компилятора gcc, для архитектуры MIPS32.

Весь набор инструментов, необходимый для сборки и отладки программ, находится на компакт диске, распространяемом вместе с NetCore. Для того чтобы начать собирать программы, нужно установить этот пакет инструментов.

Установка инструментария

Можно выбрать один из двух путей установки инструментария - компиляция из исходных кодов или установка уже скомпилированных инструментов (бинарных файлов). Обычно лучше выбрать второй вариант и использовать уже скомпилированные инструменты, но иногда может понадобиться собирать компилятор вручную из исходных кодов.

Установка инструментов из бинарных файлов

Перед установкой нужно скопировать установочный файл в директорию /opt и распаковать. Если они располагаются на компакт диске, это можно сделать, например, так:

```
$ mount /mnt/cdrom
$ cd /mnt/cdrom
$ cp mips32-crosstools-1.0.tgz /opt
$ cd /opt
$ tar -xzf mip32-crosstools-1.0.tgz
```

Теперь осталось настроить переменную окружения PATH. Как это сделать описано ниже в главе "Настройка окружения".

Установка инструментов из исходных кодов

Перед установкой нужно распаковать архив с исходными кодами во временную директорию. Если они располагаются на компакт диске, это можно сделать, например, так:

```
$ mount /mnt/cdrom
$ cd /tmp
$ tar -xzf /mnt/cdrom/mips32-src-1.0.tar.gz
$ ls
...
mips32
...
$ cd mips32
$ make
```

Последняя команда запускает процесс сборки инструментария и может выполняться продолжительное время, в зависимости от конфигурации инструментальной машины. После того, как компиляция закончится, командой

```
$ make install
```

завершаем установку инструментов.

Настройка окружения

Теперь осталось настроить переменную окружения PATH, которая должна ссылаться на только что установленные файлы. Это можно сделать, добавив в файл "/etc/profile" (тогда настройки станут глобальными), либо "~/.bash_profile" (для текущего пользователя) следующую строчку:

```
export PATH=/opt/mips32/bin:$PATH
```

После этого нужно войти в систему заново, чтобы изменения вступили в силу. Так же можно отправить эту строчку на выполнение в окне терминала - текущий сеанс терминала будет настроен на работу с инструментами до следующего входа в систему.

На этом процесс установки закончен. Можно приступать к написанию и компиляции программ. Программисту доступны следующие инструменты:

1.mipsel-linux-gcc - компилятор языка C, соответствует стандарту ISO/IEC 9899.
2.mipsel-linux-as - ассемблер для архитектуры MIPS32
3.mipsel-linux-ld - компоновщик исполняемых файлов и библиотек
4.mipsel-linux-objdump - дизассемблер для архитектуры MIPS32

Пример написания и компиляции программ

Для того чтобы лучше представить себе процесс написания и сборки программ под NetCore рассмотрим небольшой пример. Напишем и скомпилируем простейшую программу. Ничего полезного она делать не будет, всего лишь выводить фразу "Hello World!".

На инструментальной машине создадим файл программы sample.c:

```
#include <stdio.h>

int main(int argc, char* argv[])
{
        printf("Hello World!\n");
        return 0;
}
```

Для компиляции программы создадим Makefile:

```
CC:= mipsel-linux-gcc

CFLAGS:= -O2 -mips32 -mtune=4kc

sample: sample.o
        $(CC) $(CFLAGS) $(DEFINES) sample.o -o sample

sample.o: sample.c
        $(CC) $(CFLAGS) $(DEFINES) -c sample.c -o sample.o

.PHONY clean:
        rm -f sample.o sample
```

Как видно, в примере используется компилятор "mipsel-linux-gcc", которому передаются параметры "-O2 -mips32 -mtune=4kc". Теперь можно попробовать скомпилировать программу (на компьютере уже должны быть установлены инструменты, как описано выше):

```
$ make
mipsel-linux-gcc -O2 -mips32 -mtune=4kc  -c sample.c -o sample.o
mipsel-linux-gcc -O2 -mips32 -mtune=4kc  sample.o -o sample
```

В случае удачи мы получим файл "sample", готовый к выполнению на NetCore.

Замечание: *так как программа скомпилирована для работы на NetCore, она не может быть выполнена на ПК!*

Для того, чтобы перенести нашу программу на NetCore, осталось сделать две вещи: написать инсталляционный скрипт и упаковать программу в архив.

Когда мы загружаем новую программу на NetCore, запускается установочная программа, которая:
1. Копирует архив в директорию /var/upgrade и распаковывает его:

```
mkdir /var/upgrade
/bin/ncp /var/upgrade/pkg
/bin/tar zxvf pkg
```

2. Если среди распакованных файлов есть файл с именем "script", он выполняется:

```
if [ -f script ] ; then
cd /var/upgrade
./script
...
```

3. После того как "script" завершает работу, установочная программа удаляет архив и распакованные файлы командой

```
rm -rf /var/upgrade
```

Скрипт, запускаемый программой установки, может выполнять любые действия на NetCore. Как правило, первым делом нужно скопировать распакованные файлы на место постоянного хранения - в директорию /usr/local, и после этого выполнять остальные задачи, необходимые для корректной работы программы: изменять конфигурационные файл и прочее.

Директория /usr/local предназначена для хранения всех данных пользователя. Ее содержимое не теряется при перезагрузке NetCore, поэтому именно здесь нужно хранить всю информацию и программы.

Пример файла "script":

```
#!/bin/sh
mkdir /usr/local/bin
cp sample /usr/local/bin/sample
echo "Installation of 'sample' Successful!"
```

Так как скрипт запускается из программы установки обновлений, весь его вывод будет отображаться в окне браузера, что позволяет вывести сообщение об успешной установке программы, либо, наоборот, о неудаче.

Итак, скрипт написан, сделаем его выполняемым:

```
$ chmod 700 script
```

и упакуем вместе с программой в архив:

```
$ tar czf sample.pkg sample script
```

Файл "sample.pkg" готов к загрузке на NetCore.

Загрузка программ на NetCore

На данный момент для того, чтобы загрузить программу на NetCore, следует воспользоваться веб-интерфейсом. Предварительно программа должна быть упакована в архив вместе с инсталляционным скриптом, как описано в предыдущей главе.

Приступим к загрузке. Для этого мы должны знать IP адресс NetCore - по умолчанию это 10.0.0.2, если нет, то настоящий адрес можно узнать у Вашего поставщика.

Открываем веб браузер и вводим IP адрес NetCore. Перед нами страница, с которой можно производить базовую настройку NetCore:

```
Обновление
Сетевые настройки
Настройка даты
Информация об установленных пакетах
Изменение прав доступа
Сбросить настройки по умолчанию
```

Сейчас нас интересует ссылка "Обновление". Выбираем ее. Загружается страница, с которой производится передача файлов в устройство. Кнопка "choose" позволяет выбрать файл (sample.pkg), "Загрузить" - начинает передачу. В случае удачной загрузки появится сообщение нашего скрипта:

```
Installation of 'sample' Successful!
```

Для того, чтобы проверить работу программы, соединимся с NetCore при помощи программы telnet.

```
$telnet 10.0.0.2
Trying 10.0.0.2...
Connected to 10.0.0.2.
Escape character is '^]'.
noname login: root
/ # cd /usr/local/bin
/usr/local/bin # ./sample
Hello World!
/usr/local/bin # exit
Connection closed by foreign host.
```

Программа работает.

Замечание: *программа telnet есть в большинстве распространенных операционных систем, что позволяет подключаться к NetCore и производить любую настройку устройства, с любого компьютера в локальной сети.*

Настройка NetCore через веб-интерфейс

Встроенный в NetCore веб-сервер позволяет производить базовую настройку устройства при помощи обычного веб-браузера. Для того, чтобы подключиться к программе настройки, нужно знать IP адрес NetCore. Часто это 10.0.0.2, точный адрес каждого устройства можно узнать у поставщика.

Открываем любой веб-браузер и вводим в окне адреса - адрес NetCore. Загружается веб-страница настройки устройства, в которой доступны следующие пункты:

```
Обновление
Сетевые настройки
Настройка даты
Информация об установленных пакетах
Изменение прав доступа
Сбросить настройки по умолчанию
```

Рассмотрим каждый пункт отдельно.

Обновление

Пункт обновление служит для загрузки новых программ на NetCore. Подробнее о подготовке программ к загрузке и процедуре обновления можно прочитать в главе "Загрузка программ на NetCore".

Сетевые настройки

Пункт сетевые настройки служит для задания таких параметров, как:

```
Сетевое имя
Домен
IP-адрес
Маска подсети
Основной шлюз
DNS-сервер 1
DNS-сервер 2
Протокол telnet (вкл/выкл)
```

Значение всех опций интуитивно понятно. Переключатель "Протокол telnet" позволяет разрешить/запретить подключение к NetCore с помощью программы telnet. Кнопка "Сохранить" фиксирует внесенные изменения конфигурации.

Настройка даты

В пункте "Настройка даты" можно задать текущую дату/время для встроенных в NetCore часов, так же задать суточную коррекция хода времени и синхронизацию с сервером времени (если такой есть в сети).

Информация об установленных пакетах

Изменение прав доступа

Сбросить настройки по умолчанию

Пункт "Сбросить настройки по умолчанию", возвращает все параметры настройки к первоначальному значению.

Где искать информацию

www.linux-mips.org, www.mips.com - информация об архитектуре MIPS32.
Alchemy Processors on www.amd.com - информация о процессоре AMD Alchemy.
gcc.gnu.org, www.gnu.org, www.uclibc.org - информация о компиляторе gcc, инструментах GNU и библиотеке uclibc.

II. Работа с устройствами на NetCore. Краткое руководство.

Содержание

Работа с устройствами

Работа с устройствами происходит через соответствующие порты ввода-вывода. На данный момент в NetCore доступны порты RS232, RS485 и USB.

В общем случае работа с портами сводится к использованию библиотеки termios.h, основные функции которой описаны в электронном справочном руководстве, которое, как правило, можно вызвать командой **man termios**.

Основные функции для работы с устройствами будут описаны ниже. Рассмотрим для начала работу с портом RS232, на примере чтения данных с подключенного к нему устройства (например, клавиатуры).

RS232, чтение данных с устройства

Ниже приведена программа с комментариями, которая читает байты, поступающие с устройства, подключенного к первому порту RS232 и выводит их на экран. Выполнение программы прекращается при получении одного из заданных сигналов.

Работа с портом в Linux происходит через специальные файлы устройств. Для порта RS232 в NetCore это файлы /dev/ttyS0 и /dev/ttyS2. Первый из них, /dev/ttyS0, используется для работы с системной консолью и не может служить для подключения устройств. /dev/ttyS2 - доступен для работы.

Еще два файла - /dev/ttyS1 и /dev/ttyS3 служат для работы с портом RS485

Для того, чтобы начать работу с устройством, нужно выполнить системный вызов **int open(const char *pathname,int flags);**, передав ему в качестве параметров имя специального файла устройства и флаги, задающие режим доступа:

```
O_RDONLY - открыть файл только для чтения
O_WRONLY - открыть файл только для записи
O_RDWR - открыть файл для чтения/записи
```

Дополнительно, путем объединения логической операцией "или", могут быть заданы другие флаги. Из них для нас наиболее важен флаг O_NONBLOCK, который нужен, чтобы процесс не ожидал ввода при отсутствии данных, тем самым блокируя работу. Имеет смысл всегда ставить этот флаг.

open открывает файл и возвращает файловый дескриптор, который впоследствие используется для работы с устройством. Соответственно после окончания работы, нужно вызвать **int close(int fd);**, который закроет файл.

При ошибке оба вызова возвращают -1.

Теперь для работы можно использовать функции библиотеки termios.h. Параметры настройки устройства хранит структура **struct termios**. Описание ее полей есть в Приложении. Сначала лучше сохранить старые настройки, чтобы была возможность восстановить их после. Для получения параметров устройства служит функция **tcgetattr(int fd,struct termios *_p);**

Далее нужно инициализировать новую структуру termios (функция cfmakeraw - описание в Приложении), выставить в ней нужные для соединения флаги и задать скорость передачи - setispeed (входной поток), setospeed (выходной поток). Возможные варианты констант для задания скорости есть в Приложении.

Теперь функцией **int tcsetattr(int fd, int optional_action, struct termios *_p)** устанавливаем аттрибуты устройства, заданные в структуре termios. Дополнительный параметр может принимать следующие значения:

```
TCSANOW - изменения вступают в силу немедленно
```

```
    TCSADRAIN - изменения вступят в силу после завершения всех операций
                записи на устройство
    TCSAFLUSH - изменения вступят в силу после завершения всех операций
                записи на устройства и после того как данные полученные
                от устройства, но не прочитанные, будут удалены.
```

На этом настройка завершена, можно приступать к чтению данных, используя вызов **ssize_t read(int fd,void *buf, size_t count).**

Текст программы:

```
// Библиотеки, необходимые для работы программы
#include <termios.h>
#include <stdio.h>
#include <unistd.h>
#include <fcntl.h>
#include <signal.h>

int stop = 0;
// Флаг прекращения работы программы - устанавливается в 1
// при получении сигнала

static void sig_handler(int sig);
// Функция обработки сигналов

int main(int argc, char** argv)
{
        unsigned char key; // Читаемый с устройства байт
        int res;
        int fd; // Файловый дескриптор - ассоциируется с нужным устройством

        struct termios new, old_port_attr;

        // Установка диспозиции сигналов на нашу функцию
        signal(SIGTERM, sig_handler);
        signal(SIGQUIT, sig_handler);
        signal(SIGKILL, sig_handler);
        signal(SIGINT, sig_handler);
        signal(SIGSTOP, sig_handler);

        bzero(&old_port_attr, sizeof(old_port_attr));
        /*
        void bzero(void *s,size_t n);
        устанавливает первые n байт области памяти s в ноль
        */
        bzero(&new, sizeof(new));

        // Открываем для чтения специальный файл устройства
        if ( ( fd = open("/dev/ttyS0", O_RDONLY | O_NONBLOCK ) ) < 0){
            printf("Port Open Error!\n");
            return -1;
        }

        res = tcgetattr(fd, &old_port_attr);
        // Сохраняем настройки устройства

        cfmakeraw(&new);
        // Инициализируем новую структуру termios


        new.c_cflag |= CREAD | CBAUD | CSIZE | CS8;
        new.c_cflag |= CSTOPB | CRTSCTS;
        res = cfsetispeed(&new, B4800);
        res = cfsetospeed(&new, B4800);
        // Задаем настройки и устанавливаем скорость переадчи
```

```
        res = tcsetattr(fd, TCSANOW, &new);
        // Устанавливаем новые настройки устройства

        if (res<0) perror("Error!");

        while(!stop){
                res = read(fd, &key, 1);
                // Читаем байты с устройства
                if (res > 0){
                        printf("key 0x%x\n", (unsigned int)key);
                }
        }
        res = tcsetattr(fd, TCSANOW, &old_port_attr);
        // Восстанавливаем старый настойки

        close(fd);
        // Закрываем специальный файл устройства

        return 0;
}

void sig_handler(int sig)
{
        stop = 1;
        printf("stop\n");
}
```

Работа с портом RS485 - режим чтения-записи

Для работы с устройствами, подключенными к порту RS485 нужен специальный модуль ядра, который называется rs485.o и должен находиться в директории /lib/modules. Перед началом работы следует удостовериться, что этот модуль загружен и, при необходимости, загрузить его.

Чтобы посмотреть список загруженных модулей, нужно подключиться к NetCore при помощи программы telnet (как это сделать описано в руководстве 1) и ввести команду "lsmod":

```
$ lsmod
rs485
flash
i2c
```

Если модуль rs485 присутствует в списке загруженных модулей, можно перейти к работе с устройством.

Если модуля нет в списке загруженных модулей, следует использовать команду "insmod":

```
$ insmod rs485
Using /lib/modules/rs485.o
```

Теперь, когда модуль rs485 загружен в память, можно работать с устройствами.

Порту RS485 соответствует в NetCore специальный файл устройства /dev/rs485

Так же как и в случае с портом RS232, для того, чтобы начать работу с устройством, нужно выполнить системный вызов **int open(const char *pathname,int flags);**, передав

ему в качестве параметров имя специального файла устройства и флаги, задающие режим доступа.

После окончания работы следует вызвать **int close(int fd);**

Настройка порта RS485 отличается от настройки RS232. Нужно использовать специальную функцию **int ioctl(int fd,int request, ...);**, которая управляет устройством. В файле rs485.h описаны функции позволяющие управлять битом чётности и скоростью обмена.

ioctl(fd, RS485_SET9BIT) - установка контроля чётности в значение EVEN (чётно)

ioctl(fd, RS485_CLR9BIT) - установка контроля чётности в значение ZERO (всегда 0 - в бите чётности)

ioctl(fd, RS485_SET_PARITY, v) - установка контроля чётности в одно из значений:

v == 0 -> режим "нечётно"
v == 1 -> режим "чётно"
v == 2 -> режим "всегда 1" (MARK parity)
v == 3 -> режим "всегда 0" (ZERO parity)

ioctl(fd, RS485_ENABLE_PARITY, v) - управление контролем чётности:

v == 0 -> выключено
v == 1 -> включено

ioctl(fd, RS485_SET_TIMEOUT, v) - установка количества попыток при ожидании данных (по умолчанию 100).

ioctl(fd, RS485_SET_BAUDRATE, v) - установка скорости приёма/передачи данных, (по умолчанию 9600). Например: **ioctl(fd, RS485_SET_BAUDRATE, 19200);**

После того, как были заданы нужные параметры работы с устройством можно использовать функции для передачи данных **ssize_t read(int fd,void *buf, size_t count); ssize_t write(int fd,void *buf, size_t count);**

Пример работы с устройством:

```
#include <stdio.h>
#include <unistd.h>
#include <fcntl.h>
#include <signal.h>
#include "rs485.h"

// Флаг завершения работы программы
int stop = 0;

// Обработчик сигналов
static void sig_handler(int sig);

int main(int argc, char** argv)
{
        unsigned char *data;
        int res;
        int fd;

        // Установка дипозиции сигналов
```

```
        signal(SIGTERM, sig_handler);
        signal(SIGQUIT, sig_handler);
        signal(SIGKILL, sig_handler);
        signal(SIGINT, sig_handler);
        signal(SIGSTOP, sig_handler);

        // Открываем порт для чтения/записи
        fd = open( "/dev/rs485" , O_RDWR );

        if ( fd < 0 )
        {
                printf("Error opening device.\n");
                return -1;
        }

        // Производим настройку устройства
        ioctl(fd, RS485_ENABLE_PARITY, 1);

        while(!stop){
        ...
        // Чтение данных
        read(fd, &data, 4);
        ...
        // Запись данных
        write(fd, &data, 1);
        ...
        }

        // Восстановление настроек устройства
        ioctl(fd, RS485_ENABLE_PARITY, 0);
        // Закрыть файл устройства
        close(fd);
        return 0;
}

static void sig_handler(int sig)
{
    printf("stop\n");
    stop = 1;
}
```

Работа с USB портом

Для того, чтобы работать на NetCore с устройствами, которые подключаются к шине USB, необходимо иметь драйвер (модуль ядра), обеспечивающий функционирования этого устройства.

Как правило все драйверы GNU/Linux, не встроенные в ядро, находятся в директории "/lib/modules" и яляются обычными объектными файлами. Они могут быть загружены в адресное пространство ядра без перезагрузки системы. Такие драйверы называются Loadable Kernel Modules - загружаемые модули ядра. Для функционирования каждого устройства, подключаемого к шине USB, требуется специальный драйвер.

Для загрузки модуля служит программа **insmod**. Например, чтобы загрузить модуль "foo.o", находящийся в "/lib/modules", следует выполнить:

```
$ insmod foo
```

Модуль "/lib/modules/foo.o" будет загружен в ядро.

Для просмотра списка загруженных модулей служит команда **lsmod**, выводящая список всех загруженных на данный момент модулей.

Удалить (выгрузить) модуль из ядра можно командой **rmmod**. Например:

```
$ rmmod foo
```

Модуль "foo.o" будет выгружен из ядра.

Сейчас доступны драйверы для работы со следующими USB устройствами:
1. Переходник USB<=>RS232.
2. Соединение с компьютером через USB-b.
3. D-Link-DWL120e (WiFi адаптер).
4. Накопители flash.

Подробнее о каждом устройстве.

Переходник USB<=>RS232.

Для работы следует подключить переходник и последовательно загрузить модули ядра "usbserial.o" и "ftdi_sio.o":

```
$ insmod /lib/modules/usbserial.o
$ insmod /lib/modules/ftdi_sio.o
```

Теперь для работы с устройством и конфигурирования становится доступным специальный файл устройства "/dev/usb/tts/0". Работа с ним аналогична работе с любым устройством, подключаемым к шине RS232.

Соединение с компьютером через USB-b

При помощи кабеля USB-b можно подключить NetCore к компьютеру. Необходимые модули "au1x00_bi", "usbdcore.o" и "network_fd.o" загружаются командами:

```
$ insmod /lib/modules/usbdcore.o
$ insmod /lib/modules/network_fd.o
$ insmod /lib/modules/au1x00_bi.o
```

После загрузки данных драйверов и подключения устройства становится доступным для конфигурирования и использования сетевой интерфейс usbl0.

D-Link-DWL120e (WiFi адаптер)

Для работы с платами D-Link-DWL120e необходима следующая последовательность загрузки модулей:

```
$ insmod /lib/modules/usbdfu.o
$ insmod /lib/modules/at76c503.o
$ insmod /lib/modules/at76c503-rfmd.o
```

Теперь, после подключения устройства становится доступным для конфигурирования и использования сетевой интерфейс wlan0.

Накопители flash

Следует загрузить модули в таком порядке:

```
$ insmod /lib/modules/scsi_mod.o
$ insmod /lib/modules/sd_mod.o
$ insmod /lib/modules/usb-storage.o
   /* если карта содержит файловую систему FAT16, FAT32, тогда еще: */
$ insmod /lib/modules/vfat.o
```

После загрузки данных драйверов и подключения устройства становится доступным для конфигурирования и использования набор специальных файлов устройств /dev/discs/disc1/disc, /dev/discs/disc1/part1 и т.д.

Приложение

```
    struct termio {
        unsigned short c_iflag;              /* флаги ввода
        unsigned short c_oflag;              /* флаги вывода
        unsigned short c_cflag;              /* фдаги управления
        unsigned short c_lflag;              /* локальные флаги
        unsigned char c_line;          /* line discipline
        unsigned char c_cc[NCC];             /* управляющий символы
    };
```

cfmakeraw - устанавливает аттрибуты терминала в следующий значения:

```
        termios_p->c_iflag &=
~(IGNBRK|BRKINT|PARMRK|ISTRIP|INLCR|IGNCR|ICRNL|IXON);
        termios_p->c_oflag &= ~(OPOST);
        termios_p->c_lflag &= ~(ECHO|ECHONL|ICANON|ISIG|IEXTEM);
        termios_p->c_cflag &= ~(CSIZE|PARENB);
        termios_p->c_cflag |= CS8
```

Константы задания скорости передачи:

```
        B0      B2400
        B50     B4800
        B75     B9600
        B110    B19200
        B134    B38400
         B150   B57600
        B200    B115200
        B300    B230400
        B600
        B1200
        B1800
```

Установка скорости передачи входного потока в B0, устанавливает скорость передачи, аналогичную скорости выходного потока.

Установка скорости передачи выходного потока в B0, разрывает связь.