PLIP: The Parallel Line Internet Protocol Device

Donald Becker (becker@super.org)
I.D.A. Supercomputing Research Center, Bowie MD 20715

At some point T. Thorn will probably contribute text,
Tommy Thorn (tthorn@daimi.aau.dk)

PLIP Introduction

This document describes the parallel port packet pusher for Net/LGX. This device interface allows a point-to-point connection between two parallel ports to appear as a IP network interface.

What is PLIP?

PLIP is Parallel Line IP, that is, the transportation of IP packages over a parallel port. In the case of a PC, the obvious choice is the printer port. PLIP is a non-standard, but [can use] uses the standard LapLink null-printer cable [can also work in turbo mode, with a PLIP cable]. [The protocol used to pack IP packages, is a simple one initiated by Crynwr.]

Advantages of PLIP

It's cheap, it's available everywhere, and it's easy.

The PLIP cable is all that's needed to connect two Linux boxes, and it can be built for very few bucks.

Connecting two Linux boxes takes only a second's decision and a few minutes' work, no need to search for a [supported] netcard. This might even be especially important in the case of notebooks, where netcards are not easily available.

Not requiring a netcard also means that apart from connecting the cables, everything else is software configuration [which in principle could be made very easy.]

Disadvantages of PLIP

Doesn't work over a modem, like SLIP and PPP. Limited range, 15 m. Can only be used to connect three (?) Linux boxes. Doesn't connect to an existing Ethernet. Isn't standard (not even de facto standard, like SLIP).

Performance

PLIP easily outperforms Ethernet cards....(ups, I was dreaming, but it *is* getting late. EOB)

PLIP driver details

The Linux PLIP driver is an implementation of the original Crynwr protocol, that uses the parallel port subsystem of the kernel in order to properly share parallel ports between PLIP and other services.

IRQs and trigger timeouts
===

When a parallel port used for a PLIP driver has an IRQ configured to it, the PLIP driver is signaled whenever data is sent to it via the cable, such that when no data is available, the driver isn't being used.

However, on some machines it is hard, if not impossible, to configure an IRQ to a certain parallel port, mainly because it is used by some other device. On these machines, the PLIP driver can be used in IRQ-less mode, where the PLIP driver would constantly poll the parallel port for data waiting, and if such data is available, process it. This mode is less efficient than the IRQ mode, because the driver has to check the parallel port many times per second, even when no data at all is sent. Some rough measurements indicate that there isn't a noticeable performance drop when using IRQ-less mode as compared to IRQ mode as far as the data transfer speed is involved. There is a performance drop on the machine hosting the driver.

When the PLIP driver is used in IRQ mode, the timeout used for triggering a data transfer (the maximal time the PLIP driver would allow the other side before announcing a timeout, when trying to handshake a transfer of some data) is, by default, 500usec. As IRQ delivery is more or less immediate, this timeout is quite sufficient.

When in IRQ-less mode, the PLIP driver polls the parallel port HZ times per second (where HZ is typically 100 on most platforms, and 1024 on an Alpha, as of this writing). Between two such polls, there are 10^6/HZ usecs. On an i386, for example, 10^6/100 = 10000usec. It is easy to see that it is quite possible for the trigger timeout to expire between two such polls, as the timeout is only 500usec long. As a result, it is required to change the trigger timeout on the *other* side of a PLIP connection, to about 10^6/HZ usecs. If both sides of a PLIP connection are used in IRQ-less mode, this timeout is required on both sides.

It appears that in practice, the trigger timeout can be shorter than in the above calculation. It isn't an important issue, unless the wire is faulty, in which case a long timeout would stall the machine when, for whatever reason, bits are dropped.

A utility that can perform this change in Linux is plipconfig, which is part of the net-tools package (its location can be found in the Documentation/Changes file). An example command would be 'plipconfig plipX trigger 10000', where plipX is the appropriate PLIP device.

PLIP hardware interconnection

PLIP uses several different data transfer methods. The first (and the

only one implemented in the early version of the code) uses a standard printer "null" cable to transfer data four bits at a time using data bit outputs connected to status bit inputs.

The second data transfer method relies on both machines having bi-directional parallel ports, rather than output-only ``printer'' ports. This allows byte-wide transfers and avoids reconstructing nibbles into bytes, leading to much faster transfers.

Parallel Transfer Mode 0 Cable

The cable for the first transfer mode is a standard printer "null" cable which transfers data four bits at a time using data bit outputs of the first port (machine T) connected to the status bit inputs of the second port (machine R). There are five status inputs, and they are used as four data inputs and a clock (data strobe) input, arranged so that the data input bits appear as contiguous bits with standard status register implementation.

A cable that implements this protocol is available commercially as a "Null Printer" or "Turbo Laplink" cable. It can be constructed with two DB-25 male connectors symmetrically connected as follows:

```
    STROBE output       1*
    D0->ERROR   2 - 15          15 - 2
    D1->SLCT    3 - 13          13 - 3
    D2->PAPOUT  4 - 12          12 - 4
    D3->ACK     5 - 10          10 - 5
    D4->BUSY    6 - 11          11 - 6
    D5,D6,D7 are   7*, 8*, 9*
    AUTOFD output 14*
    INIT   output 16*
    SLCTIN      17 - 17
    extra grounds are 18*,19*,20*,21*,22*,23*,24*
    GROUND      25 - 25
* Do not connect these pins on either end
```

If the cable you are using has a metallic shield it should be connected to the metallic DB-25 shell at one end only.

Parallel Transfer Mode 1

The second data transfer method relies on both machines having bi-directional parallel ports, rather than output-only ``printer'' ports. This allows byte-wide transfers, and avoids reconstructing nibbles into bytes. This cable should not be used on unidirectional ``printer'' (as opposed to ``parallel'') ports or when the machine isn't configured for PLIP, as it will result in output driver conflicts and the (unlikely) possibility of damage.

The cable for this transfer mode should be constructed as follows:

```
    STROBE->BUSY 1 - 11
    D0->D0       2 - 2
```

```
    D1->D1      3 - 3
    D2->D2      4 - 4
    D3->D3      5 - 5
    D4->D4      6 - 6
    D5->D5      7 - 7
    D6->D6      8 - 8
    D7->D7      9 - 9
    INIT -> ACK  16 - 10
    AUTOFD->PAPOUT 14 - 12
    SLCT->SLCTIN 13 - 17
    GND->ERROR  18 - 15
    extra grounds are 19*,20*,21*,22*,23*,24*
    GROUND      25 - 25
* Do not connect these pins on either end
```

Once again, if the cable you are using has a metallic shield it should be connected to the metallic DB-25 shell at one end only.

PLIP Mode 0 transfer protocol
==========

The PLIP driver is compatible with the "Crynwr" parallel port transfer standard in Mode 0. That standard specifies the following protocol:

```
   send header nibble '0x8'
   count-low octet
   count-high octet
   ... data octets
   checksum octet
```

Each octet is sent as

```
        <wait for rx. '0x1?'>   <send 0x10+(octet&0x0F)>
        <wait for rx. '0x0?'>   <send 0x00+((octet>>4)&0x0F)>
```

To start a transfer the transmitting machine outputs a nibble 0x08. That raises the ACK line, triggering an interrupt in the receiving machine. The receiving machine disables interrupts and raises its own ACK line.

Restated:

(OUT is bit 0-4, OUT.j is bit j from OUT. IN likewise)

```
Send_Byte:
   OUT := low nibble, OUT.4 := 1
   WAIT FOR IN.4 = 1
   OUT := high nibble, OUT.4 := 0
   WAIT FOR IN.4 = 0
```