NAME

lp - printer port Internet Protocol driver

SYNOPSIS

ifconfig *plip0 myaddress hisaddress* [**-link0**]

device ppbus
device plip
device ppc

DESCRIPTION

The **lp** driver allows a PC parallel printer port to be used as a point-to-point network interface between two similarly configured systems. Data is transferred 4 bits at a time, using the printer status lines for input: hence there is no requirement for special bidirectional hardware and any standard AT-compatible printer port with working interrupts may be used.

During the boot process, for each **plip** device which is probed and has an interrupt assigned, a corresponding **network** device is created.

Configuring an **lp** device with ifconfig(8) causes the corresponding **parallel port bus** to be reserved for PLIP until the network interface is configured 'down'.

The communication protocol is selected by the **link0** flag:

-link0 (default) Use FreeBSD mode (LPIP). This is the simpler of the two modes and therefore slightly more efficient.

link0 Use Crynwr/Linux compatible mode (CLPIP). This mode has a simulated Ethernet packet header, and is easier to interface to other types of equipment.

The interface MTU defaults to 1500, but may be set to any value. Both ends of the link must be configured with the same MTU.

Cable Connections

The cable connecting the two parallel ports should be wired as follows:

Pin	Pin	Description
2	15	Data0 -> ERROR*
3	13	Data1 -> SLCT
4	12	Data2 -> PE

5	10	Data3 -> ACK*
6	11	Data4 -> BUSY
15	2	ERROR* -> Data0
13	3	SLCT -> Data1
12	4	PE -> Data2
10	5	ACK* -> Data3
11	6	BUSY -> Data4
18-25	18-25	Ground

Cables with this wiring are widely available as 'Laplink' cables, and are often coloured yellow.

The connections are symmetric, and provide 5 lines in each direction (four data plus one handshake). The two modes use the same wiring, but make a different choice of which line to use as handshake.

FreeBSD LPIP mode

The signal lines are used as follows:

Data0 (Pin 2) Data out, bit 0.

Data1 (Pin 3) Data out, bit 1.

Data2 (Pin 4) Data out, bit 2.

Data3 (Pin 5) Handshake out.

Data4 (Pin 6) Data out, bit 3.

ERROR (pin 15)*
Data in, bit 0.

SLCT (pin 13) Data in, bit 1.

PE (pin 12) Data in, bit 2.

BUSY (pin 11) Data in, bit 3.

ACK (pin 10)* Handshake in.

When idle, all data lines are at zero. Each byte is signalled in four steps: sender writes the 4 most significant bits and raises the handshake line; receiver reads the 4 bits and raises its handshake to

acknowledge; sender places the 4 least significant bits on the data lines and lowers the handshake; receiver reads the data and lowers its handshake.

The packet format has a two-byte header, comprising the fixed values 0x08, 0x00, immediately followed by the IP header and data.

The start of a packet is indicated by simply signalling the first byte of the header. The end of the packet is indicated by inverting the data lines (i.e., writing the ones-complement of the previous nibble to be transmitted) without changing the state of the handshake.

Note that the end-of-packet marker assumes that the handshake signal and the data-out bits can be written in a single instruction - otherwise certain byte values in the packet data would falsely be interpreted as end-of-packet. This is not a problem for the PC printer port, but requires care when implementing this protocol on other equipment.

Crynwr/Linux CLPIP mode

The signal lines are used as follows:

Data0 (Pin 2) Data out, bit 0.

Data1 (Pin 3) Data out, bit 1.

Data2 (Pin 4) Data out, bit 2.

Data3 (Pin 5) Data out, bit 3.

Data4 (Pin 6) Handshake out.

ERROR (pin 15)*
Data in, bit 0.

SLCT (pin 13) Data in, bit 1.

PE (pin 12) Data in, bit 2.

ACK (pin 10)* Data in, bit 3.

BUSY (pin 11) Handshake in.

When idle, all data lines are at zero. Each byte is signalled in four steps: sender writes the 4 least

significant bits and raises the handshake line; receiver reads the 4 bits and raises its handshake to acknowledge; sender places the 4 most significant bits on the data lines and lowers the handshake; receiver reads the data and lowers its handshake. [Note that this is the opposite nibble order to LPIP mode].

Packet format is:

Length (least significant byte)
Length (most significant byte)
12 bytes of supposed MAC addresses (ignored by FreeBSD).
Fixed byte 0x08
Fixed byte 0x00
<IP datagram>
Checksum byte.

The length includes the 14 header bytes, but not the length bytes themselves nor the checksum byte.

The checksum is a simple arithmetic sum of all the bytes (again, including the header but not checksum or length bytes). FreeBSD calculates outgoing checksums, but does not validate incoming ones.

The start of packet has to be signalled specially, since the line chosen for handshake-in cannot be used to generate an interrupt. The sender writes the value 0x08 to the data lines, and waits for the receiver to respond by writing 0x01 to its data lines. The sender then starts signalling the first byte of the packet (the length byte).

End of packet is deduced from the packet length and is not signalled specially (although the data lines are restored to the zero, idle state to avoid spuriously indicating the start of the next packet).

SEE ALSO

ppbus(4), ppc(4), ifconfig(8)

BUGS

Busy-waiting loops are used while handshaking bytes, (and worse still when waiting for the receiving system to respond to an interrupt for the start of a packet). Hence a fast system talking to a slow one will consume excessive amounts of CPU. This is unavoidable in the case of CLPIP mode due to the choice of handshake lines; it could theoretically be improved in the case of LPIP mode.

Polling timeouts are controlled by counting loop iterations rather than timers, and so are dependent on CPU speed. This is somewhat stabilised by the need to perform (slow) ISA bus cycles to actually read the port.