NAME

X25 − BX.25 network interface

DESCRIPTION

The X25 driver provides multiplexed channels over one or more synchronous communications lines using the Bell System standard BX.25 Level 3 protocol. The current release supports permanent virtual circuits (PVCs) only; the call set-up features needed to support virtual calls have not yet been implemented. There is a separate and independent Level 3 interface for each communications line. Point-to-point connections between hosts are supported as well as connections via an X.25 network.

The X25 driver is implemented as a VPM protocol module (see *vpm*(4)). The X25 driver uses the VPM interface module to access communications lines controlled by KMC11-B microprocessors. Level 2 of BX.25, the link level, is implemented by a VPM protocol script in the KMC.

The special files **/dev/x25/s?** refer to the minor devices of the X25 driver. Each such minor device, also referred to as a *slot*, can be connected by means of a *network control* device (see *nc*(4)) to an arbitrary logical channel (1-4095) on a specified X25 interface. When the other end of the logical channel has been connected in an analogous fashion, each slot so connected is the terminus of a *permanent virtual circuit*, which is a full-duplex connection over a BX.25 logical channel between a set of user processes on the local host and another set of user processes on a remote host. A logical channel is a connection which may be multiplexed with other channels over a physical link to a remote host or an X.25 network. Each X25 interface (also referred to as a *link*) must be connected via the network-control device to a particular KMC microprocessor or to a particular line on a KMS11 communications multiplexor.

A user process accesses a BX.25 minor device (slot) using *open*, *close*, *read*, *write*, and *ioctl* system calls.

There are several internal flags that are maintained by the X25 driver for each slot. The values of these flags can be read and in some cases modified by means of the *ioctl* system call (see below).

An *open* will fail and return the error EIO if the specified slot does not exist, if the slot is not currently connected to a logical channel on some link, or if the link to which the slot is connected is not currently active. The user may request the normal *open* options O_RDONLY, O_WRONLY, and O_RDWR. The user may also request that reads with no data available should not sleep, writes with no transmit queue space return immediately, and that *open* should not wait for *faropen* to be set, using the O_NDELAY *open* flag. The *open*, and all use of the slot, can be made exclusive, using the O_EXCL *open* flag. If an exclusive *open* is requested and cannot be granted, the error EBUSY will be returned. A successful *open* will clear the *isreset* status bit (see the discussion of *ioctl* below). If O_NDELAY is specified, the user is responsible for insuring that the remote end of the slot is ready to receive data before any is sent via *writes*. Note that O_NDELAY can be set via the *fcntl*(2) system call after a successful *open* (via the *ioctl* call **X25FCNTL** for CB-UNIX), which insures that the *open* will not return until the other end is fully connected.

An *open* may or may not block until the far end is also open, depending on the session-establishment protocol requested. There are three choices for the session-establishment protocol. The choice is made by means of the network-control device at the time the permanent virtual circuit is installed. The first mode, referred to as the "no-protocol" session mode, is for the *open* to return immediately. This puts the burden on the user program to determine whether the far end is actually open. The reset session mode, designed mainly for compatibility with certain non-UNIX implementations of BX.25, uses a RESET in-order packet to indicate to the far end that a slot has been opened and a RESET out-of-order packet to indicate to the far end that the slot has been closed. In the current implementation, the RESET in-order and RESET out-of-order packets are recognized when they are received, but are not transmitted