NAME

vpm, vpb — Virtual Protocol Machine Protocol and Interface Drivers

DESCRIPTION

This entry describes the *vpm* and *vpb* drivers and gives an introduction to the Virtual Protocol Machine (VPM).

VPM is a software package for implementing link-level protocols on the DEC KMC11 microcomputer in a high-level-language. This is accomplished by a compiler that runs on UNIX and translates a high-level language description of a protocol into an intermediate language that is executed by an interpreter running in the KMC. VPM also provides a framework for implementing higher levels of protocols (levels 3 and above) as UNIX drivers.

The VPM software consists of the following components:

1. A compiler (*vpmc*(1C)) for the protocol description language; it runs on UNIX.
2. An interpreter that controls the overall operation of the KMC and interprets the protocol script.
3. Two UNIX drivers: A Protocol driver and an Interface driver.
4. *vpmstart*(1C): a UNIX command that copies a load module into the KMC and starts it.
5. *vpmset*(1C): a UNIX command that logically connects VPM minor devices and KMC synchronous links. *x25pvc*(1C) and *x25lnk*(1C) may also be used to connect things up.
6. *vpmsnap*(1C): a UNIX command that prints a time-stamped event trace while the protocol is running.
7. *vpmtrace*(1C): a UNIX command that prints an event trace for debugging purposes while the protocol is running.
8. *vpmsave*(1C): a UNIX command that writes unformated trace data to its standard output.
9. *vpmfmt*(1C): a UNIX command that formats the output of *vpmsave*.

The VPM protocol driver provides a simple user interface to a synchronous line controlled by a link-level protocol executed by the VPM interpreter in the KMC. It supports the following UNIX system calls: *open*, *read*, *write*, *close*, and *ioctl*. If higher levels of protocol are required, the VPM protocol driver may be modified or replaced. The VPM interface driver provides a common interface to a synchronous line controlled by a link-level protocol executed by the VPM interpreter. This common interface can be shared by several different protocol modules (see *x25*(4)).

Before a protocol driver minor device can be used, it must be logically connected to a VPM interface driver; the interface driver minor device must in turn be logically connected to a synchronous line of a KMC microprocesser or a KMS11 communication multiplexor. These corrections can be made by means of *ioctl* commands (see below). The command *vpmset*(1C) uses these *ioctl* commands to make these connections.

The VPM Interface Driver.

The VPM interface driver provides a general purpose interface between level 3 protocols executing in the UNIX kernel and level 2 protocols being executed by the VPM interpreter in the KMC. This interface is used by the VPM Protocol driver as well other protocol drivers such as the LEAP, X25, and ST.

The Interface Driver supports *open*, *close*, and *ioctl* systems calls. These calls are used to set-up connections between the interface driver and a synchronous line on a KMC or KMS and to set interpreter options. The system *ioctl* call has the following form:

ioctl (fildes, cmd, arg)

Possible values for the *cmd* argument are:

VPMSDEV	Connect an interface driver minor device to a synchronous line on a KMC or KMS. Bits 6 and 7 of *arg* contain the minor device number of the KMC or KMS. Bits 0-2 of *arg* contain the line number (0-7) if a KMS is being used; for a single-line KMC they must be zero.
VPMGETM	Get the interpreter modes. The currently available modes are the normal mode and the X.25 mode. The normal mode is indicated by an *arg* of all zeros. In this mode, the entire information field of an I frame is copied by the interpreter to the assigned buffer. The X.25 mode is indicated by a 1 in bit 0. In this mode, the VPM interpreter copies the first three bytes of the information field of level 2 I frames into the buffer descriptor of the buffer assigned to receive the frame. These three bytes are the X.25 level 3 header. The remaining bytes, if any, are copied to the buffer pointed to by the buffer descriptor.
VPMSETM	Set the interpreter modes. The modes specifed by a 1 in the mask *arg* are set.
VPMCLRM	Clear interpreter modes. The modes specified by a 1 in the mask *arg* are cleared.

The routines that make up the VPM interface are:

vpmstart	Start the level 2 protocol.
vpmstop	Stop the level 2 protocol.
vpmxmtq	Place a transmit buffer descriptor pointer on the level 2 transmit queue.
vpmempty	Place a empty receive buffer descriptor pointer on the level 2 empty receive queue.
vpmcmd	Send a four byte command to the level 2 protocol.
vpmrpt	Receive a four byte report from the level 2 protocol.
vpmenq	Place a buffer descriptor pointer at the end of the indicator VPM linked list queue.
vpmdeq	Remove the buffer descriptor at the head of the indicated VPM linked list queue.
vpmrmv	Search the indicated VPM linked list queue for the given buffer descriptor pointer and remove it if found.
vpmerr	Get the error counters maintain by the VPM interpreter. After the interpreter has passed the counters to the driver it resets its copy of the counters.
vpmsave	Save an event record using the trace driver minor device zero.
vpmsnap	Save a time-stamped event record using the trace driver minor device 1.

Operation of the Standard Protocol Driver.

UNIX user processes transfer data to or from a remote terminal or computer system through VPM using normal *open*, *read*, *write*, and *close* operations. Flow control and error recovery are provided by the protocol executed by the interpreter in the KMC.

The VPM *open* for reading-and-writing is exclusive; *open*s for reading-only or writing-only are not exclusive. The VPM *open* checks that the correct interpreter is running in the KMC and then sends a command to the interpreter which causes it to start interpreting the protocol script. The driver then supplies one or more 512-byte receive buffers to the interpreter.

The VPM *read* returns either the number of bytes requested or the number remaining in the current receive buffer, whichever is less; any bytes remaining in the current receive buffer are used to satisfy subsequent reads. The VPM *write* copies the user data into 512-byte system buffers and passes them to the VPM interpreter in the KMC for transmission.

The VPM *close* arranges for the return of system buffers and for a general cleanup when the last transmit buffer has been returned by the interpreter. It also stops the execution of the protocol script.

The VPM protocol driver supports the following *ioctl* commands:

VPMCMD	Send a command to the protocol script. The first four bytes of the array pointed to by *arg* are passed to the VPM interpreter which saves them and passes them to the protocol script when it executes a *getcmd* primitive. Only the most recent command is kept by the VPM interpreter.
VPMERRS	Get and then reset the interpreter's error counters. The interpreter's four, two-bytes error counters are copied to the array pointed to by *arg*. The interpreter's copy of the counters is then set to zero.
VPMRPT	Get the latest script report. When the protocol script executes a *rtnrpt* primitive, a four-byte report is passed from the protocol script to the VPM protocol driver. Only the most recent script report is kept by the driver. If there is a script report that has not previously been passed to a user via this *ioctl* command, that report is copied to the array pointed to by *arg* and a non-zero value (*one*) is passed as the return value. If no script report is available, a *zero* is passed as the return value.
VPMSDEV	Connect a protocol driver to an interface driver. *Arg* is the minor device number of the interface driver to be connected to this protocol driver. To invoke this *ioctl* command, the file status flag, O_NDELAY must be set.

The VPM Event Trace

The VPM drivers generates a number of event records to allow the activity of the drivers and protocol script to be monitored for debugging purposes. If a program such as *vpmtrace*(1C) or *vpmsave*(1C) has opened minor device 0 of the trace driver and has enabled the appropriate channels on that device, these event records are queued for reading; otherwise, the event records are discarded by the trace driver. Event records associated with interface driver minor device n are put on the read queue for minor device 0 of the trace driver with a channel number of n. Calls to the system functions *vpmopen*, *vpmread*, *vpmwrite*, and *vpmclose* generate event records identified respectively by **o**, **r**, **w**, and **c**. Calls to the *vpmc*(1C) primitive *trace*(*arg1*,*arg2*) cause the VPM interpreter to pass *arg1* and *arg2* along with the current valu of the script location counter to the VPM driver, which generates an event record identified by a **T**. Each event record is structured as follows:

```
struct event {
        short   e_seqn;         /* sequence number */
        char    e_type;         /* record identifier */
        char    e_dev;          /* minor device number */
        short   e_short1;       /* data */
        short   e_short2;       /* data */
}
```

When the script terminates for any reason, the driver is notified and generates an event record identified by an E. This record also contains the minor device number, the script location counter, and a termination code defined as follows:

0	Normal termination; the interpreter received a HALT command from the driver.
1	Undefined virtual-machine operation code.
2	Script program counter out of bounds.
3	Interpreter stack overflow or underflow.
4	Jump address not even.
5	UNIBUS error.
6	Transmit buffer has an odd address; the driver tried to give the interpreter too many transmit buffers; or a *get* or *rtnxbuf* was executed while no transmit buffer was open, i.e., no *getxbuf* was executed prior to the *get* or *rtnxbuf*.
7	Receive buffer has an odd address; the driver tried to give the interpreter too many receive buffers; or a *put* or *rtnrbuf* was executed while no receive buffer was open, i.e., no *getrbuf* was executed prior to the *get* or *rtnxbuf*.
8	The script executed an *exit* primitive.
9	A *crc16* was executed without a preceding *crcloc* execution.
10	The interpreter detected loss of the modem-ready signal at the modem interface.
11	Transmit-buffer sequence-number error.
12	Command error: an invalid command or an improper sequence of commands was received from the driver.
13	Not used.
14	Invalid transmit state (internal error).
15	Invalid receive state (internal error).
16	Not used.
17	*Xmtctl* or *setctl* attempted while transmitter was still busy.
18	Not used.
19	Same as error code 6.
20	Same as error code 7.
21	Script too large.
22	Used for debugging the interpreter.
23	The driver's OK-check has timed out.

SEE ALSO

vpmc(1C), vpmset(1C), vpmstart(1C), x25lnk(1C), x25pvc(1C), trace(4).