NAME

pfmod – STREAMS Packet Filter Module

SYNOPSIS

#include <sys/pfmod.h>

ioctl(*fd*, **I_PUSH**, **"pfmod");**

DESCRIPTION

pfmod is a STREAMS module that subjects messages arriving on its read queue to a packet filter and passes only those messages that the filter accepts on to its upstream neighbor. Such filtering can be very useful for user-level protocol implementations and for networking monitoring programs that wish to view only specific types of events.

Read-side Behavior

pfmod applies the current packet filter to all **M_DATA** and **M_PROTO** messages arriving on its read queue. The module prepares these messages for examination by first skipping over all leading **M_PROTO** message blocks to arrive at the beginning of the message's data portion. If there is no data portion, **pfmod** accepts the message and passes it along to its upstream neighbor. Otherwise, the module ensures that the part of the message's data that the packet filter might examine lies in contiguous memory, calling the **pullupmsg**(9F) utility routine if necessary to force contiguity. (Note: this action destroys any sharing relationships that the subject message might have had with other messages.) Finally, it applies the packet filter to the message's data, passing the entire message upstream to the next module if the filter accepts, and discarding the message otherwise. See PACKET FILTERS below for details on how the filter works.

If there is no packet filter yet in effect, the module acts as if the filter exists but does nothing, implying that all incoming messages are accepted. IOCTLS below describes how to associate a packet filter with an instance of **pfmod**.

pfmod passes all other messages through unaltered to its upper neighbor.

Write-side Behavior

pfmod intercepts **M_IOCTL** messages for the *ioctl* described below. The module passes all other messages through unaltered to its lower neighbor.

IOCTLS

pfmod responds to the following *ioctl*.

PFIOCSETF This *ioctl* directs the module to replace its current packet filter, if any, with the filter specified by the **struct packetfilt** pointer named by its final argument. This structure is defined in **<sys/pfmod.h>** as:

```
struct packetfilt {
        u_char   Pf_Priority;                  /* priority of filter */
        u_char   Pf_FilterLen;                 /* length of filter cmd list */
        u_short  Pf_Filter[ENMAXFILTERS];      /* filter command list */
};
```

The **Pf_Priority** field is included only for compatibility with other packet filter implementations and is otherwise ignored. The packet filter itself is specified in the **Pf_Filter** array as a sequence of two-byte commands, with the **Pf_FilterLen** field giving the number of commands in the sequence. This implementation restricts the maximum number of commands in a filter (**ENMAXFILTERS**) to 255. The next section describes the available commands and their semantics.

PACKET FILTERS

A packet filter consists of the filter command list length (in units of **u_short**s), and the filter command list itself. (The priority field mentioned above is ignored in this implementation.) Each filter command list specifies a sequence of actions that operate on an internal stack of **u_short**s ("shortwords"). Each shortword of the command list specifies one of the actions ENF_PUSHLIT, ENF_PUSHZERO, ENF_PUSHONE, ENF_PUSHFFFF, ENF_PUSHFF00, ENF_PUSH00FF, or ENF_PUSHWORD+*n*, which respectively push the next shortword of the command list, zero, one, 0xFFFF, 0xFF00, 0x00FF, or shortword *n* of the subject message on the stack, and a binary operator from the set {ENF_EQ, ENF_NEQ, ENF_LT, ENF_LE, ENF_GT, ENF_GE, ENF_AND, ENF_OR, ENF_XOR} which then operates on the top two elements of the stack and replaces them with its result. When both an action and operator are specified in the same shortword, the action is performed followed by the operation.

The binary operator can also be from the set {ENF_COR, ENF_CAND, ENF_CNOR, ENF_CNAND}. These are "short-circuit" operators, in that they terminate the execution of the filter immediately if the condition they are checking for is found, and continue otherwise. All pop two elements from the stack and compare them for equality; ENF_CAND returns false if the result is false; ENF_COR returns true if the result is true; ENF_CNAND returns true if the result is false; ENF_CNOR returns false if the result is true. Unlike the other binary operators, these four do not leave a result on the stack, even if they continue.

The short-circuit operators should be used when possible, to reduce the amount of time spent evaluating filters. When they are used, you should also arrange the order of the tests so that the filter will succeed or fail as soon as possible; for example, checking the IP destination field of a UDP packet is more likely to indicate failure than the packet type field.

The special action ENF_NOPUSH and the special operator ENF_NOP can be used to only perform the binary operation or to only push a value on the stack. Since both are (conveniently) defined to be zero, indicating only an action actually specifies the action followed by ENF_NOP, and indicating only an operation actually specifies ENF_NOPUSH followed by the operation.

After executing the filter command list, a non-zero value (true) left on top of the stack (or an empty stack) causes the incoming packet to be accepted and a zero value (false) causes the packet to be rejected. (If the filter exits as the result of a short-circuit operator, the top-of-stack value is ignored.) Specifying an undefined operation or action in the command list or performing an illegal operation or action (such as pushing a shortword offset past the end of the packet or executing a binary operator with fewer than two shortwords on the stack) causes a filter to reject the packet.

EXAMPLES

The packet filter module is not dependent on any particular device driver or module but is commonly used with datalink drivers such as the Ethernet driver. If the underlying datalink driver supports the Data Link Provider Interface (DLPI) message set, the appropriate STREAMS DLPI messages must be issued to attach the stream to a particular hardware device and bind a datalink address to the stream before the underlying driver will route received packets upstream. Refer to the DLPI Version 2 specification for details on this interface.

The reverse ARP daemon program may use code similar to the following fragment to construct a filter that rejects all but RARP packets. That is, is accepts only packets whose Ethernet type field has the value ETHERTYPE_REVARP.

```
struct ether_header eh;          /* used only for offset values */
struct packetfilt pf;
register u_short *fwp = pf.Pf_Filter;
u_short offset;
int      fd;

/*
 * Push packet filter streams module.
 */
if (ioctl(fd, I_PUSH, "pfmod") < 0)
        syserr("pfmod");

/*
 * Set up filter.  Offset is the displacement of the Ethernet
 * type field from the beginning of the packet in units of
 * u_shorts.
 */
offset = ((u_int) &eh.ether_type - (u_int) &eh.ether_dhost) /
        sizeof (u_short);
*fwp++ = ENF_PUSHWORD + offset;
*fwp++ = ENF_PUSHLIT;
*fwp++ = htons(ETHERTYPE_REVARP);
*fwp++ = ENF_EQ;
pf.Pf_FilterLen = fwp - &pf.Pf_Filter[0];
```

This filter can be abbreviated by taking advantage of the ability to combine actions and operations:

```
*fwp++ = ENF_PUSHWORD + offset;
*fwp++ = ENF_PUSHLIT | ENF_EQ;
*fwp++ = htons(ETHERTYPE_REVARP);
```

SEE ALSO

bufmod(7M), **dlpi**(7P), **ie**(7D), **le**(7D), **pullupmsg**(9F)