NAME

bufmod – STREAMS Buffer Module

SYNOPSIS

ioctl(fd, I_PUSH, "bufmod");

DESCRIPTION

bufmod is a STREAMS module that buffers incoming messages, reducing the number of system calls and the associated overhead required to read and process them. Although **bufmod** was originally designed to be used in conjunction with STREAMS-based networking device drivers, the version described here is general purpose so that it can be used anywhere STREAMS input buffering is required.

Read-side Behavior

bufmod's behavior depends on various parameters and flags that can be set and queried as described below under **IOCTLS**. **bufmod** collects incoming **M_DATA** messages into *chunks*, passing each chunk upstream when the chunk becomes full or the current read timeout expires. It optionally converts **M_PROTO** messages to **M_DATA** and adds them to chunks as well. It also optionally adds to each message a header containing a timestamp, and a cumulative count of messages dropped on the stream read side due to resource exhaustion or flow control. **bufmod**'s default settings allow it to drop messages when flow control sets in or resources are exhausted; disabling headers and explicitly requesting no drops makes **bufmod** pass all messages through. Finally, **bufmod** is capable of truncating upstream messages to a fixed, programmable length.

When a message arrives, **bufmod** processes it in several steps. The following paragraphs discuss each step in turn.

Upon receiving a message from below, if the **SB_NO_HEADER** flag is not set, **bufmod** immediately timestamps it and saves the current time value for later insertion in the header described below.

Next, if **SB_NO_PROTO_CVT** is not set, **bufmod** converts all leading **M_PROTO** blocks in the message to **M_DATA** blocks, altering only the message type field and leaving the contents alone.

It then truncates the message to the current *snapshot length*, which is set with the **SBIOCSSNAP ioctl** described below.

Afterwards, if **SB_NO_HEADER** is not set, **bufmod** prepends a header to the converted message. This header is defined as follows.

```
struct sb_hdr {
        u_int   sbh_origlen;
        u_int   sbh_msglen;
        u_int   sbh_totlen;
        u_int   sbh_drops;
        struct  timeval sbh_timestamp;
};
```

The **sbh_origlen** field gives the message's original length before truncation in bytes. The **sbh_msglen** field gives the length in bytes of the message after the truncation has been done. **sbh_totlen** gives the distance in bytes from the start of the truncated message in the current chunk (described below) to the start of the next message in the chunk; the value reflects any padding necessary to insure correct data alignment for the host machine and includes the length of the header itself. **sbh_drops** reports the cumulative number of input messages that this instance of **bufmod** has dropped due to flow control or resource exhaustion. In the current implementation message dropping due to flow control can occur only if the **SB_NO_DROPS** flag is not set. (Note: this accounts only for events occurring within **bufmod**, and does not count messages dropped by downstream or by upstream modules.) The **sbh_timestamp** field contains the message arrival time expressed as a **struct timeval**.

After preparing a message, **bufmod** attempts to add it to the end of the current chunk, using the chunk size and timeout values to govern the addition. The chunk size and timeout values are set and inspected using the **ioctl( )** calls described below. If adding the new message would make the current chunk grow larger than the chunk size, **bufmod** closes off the current chunk, passing it up to the next module in line, and starts a new chunk. If adding the message would still make the new chunk overflow, the module passes it upward in an over-size chunk of its own. Otherwise, the module concatenates the message to the end of the current chunk.

To ensure that messages do not languish forever in an accumulating chunk, **bufmod** maintains a read timeout. Whenever this timeout expires, the module closes off the current chunk and passes it upward. The module restarts the timeout period when it receives a read side data message and a timeout is not currently active. These two rules insure that **bufmod** minimizes the number of chunks it produces during periods of intense message activity and that it periodically disposes of all messages during slack intervals, but avoids any timeout overhead when there is no activity.

bufmod handles other message types as follows. Upon receiving an **M_FLUSH** message specifying that the read queue be flushed, the module clears the currently accumulating chunk and passes the message on to the module or driver above. (Note: **bufmod** uses zero length **M_CTL** messages for internal synchronization and does not pass them through.) **bufmod** passes all other messages through unaltered to its upper neighbor, maintaining message order for non high priority messages by passing up any accumulated chunk first.

If the **SB_DEFER_CHUNK** flag is set, buffering does not begin until the second message is received within the timeout window.

If the **SB_SEND_ON_WRITE** flag is set, **bufmod** passes up the read side any buffered data when a message is received on the write side. **SB_SEND_ON_WRITE** and **SB_DEFER_CHUNK** are often used together.

Write-side Behavior

bufmod intercepts **M_IOCTL** messages for the **ioctl**s described below. The module passes all other messages through unaltered to its lower neighbor. If **SB_SEND_ON_WRITE** is set, message arrival on the writer side suffices to close and transmit the current read side chunk.

IOCTLS

bufmod responds to the following **ioctl**s.

SBIOCSTIME Set the read timeout value to the value referred to by the **struct timeval** pointer given as argument. Setting the timeout value to zero has the side-effect of forcing the chunk size to zero as well, so that the module will pass all incoming messages upward immediately upon arrival. Negative values are rejected with an EINVAL error.

SBIOCGTIME Return the read timeout in the **struct timeval** pointed to by the argument. If the timeout has been cleared with the **SBIOCCTIME ioctl**, return with an ERANGE error.

SBIOCCTIME Clear the read timeout, effectively setting its value to infinity. This results in no timeouts being active and the chunk being delivered when it is full.

SBIOCSCHUNK Set the chunk size to the value referred to by the *u_int* pointer given as argument. See **NOTES** for description of effect on stream head high water mark.

SBIOCGCHUNK Return the chunk size in the *u_int* pointed to by the argument.

SBIOCSSNAP Set the current snapshot length to the value given in the **u_long** pointed to by the **ioctl**'s final argument. **bufmod** interprets a snapshot length value of zero as meaning infinity, so it will not alter the message. See **NOTES** for description of effect on stream head high water mark.

SBIOCGSNAP Returns the current snapshot length in the **u_long** pointed to by the **ioctl**'s final argument.

SBIOCSFLAGS Set the current flags to the value given in the **u_long** pointed to by the **ioctl**'s final argument. Possible values are a combination of the following.

- **SB_SEND_ON_WRITE** Transmit the read side chunk on arrival of a message on the write side.
- **SB_NO_HEADER** Do not add headers to read side messages.
- **SB_NO_DROPS** Do not drop messages due to flow control upstream.
- **SB_NO_PROTO_CVT** Do not convert **M_PROTO** messages into **M_DATA**.
- **SB_DEFER_CHUNK** Begin buffering on arrival of the second read side message in a timeout interval.

SBIOCGFLAGS Returns the current flags in the **u_long** pointed to by the **ioctl**'s final argument.

SEE ALSO

dlpi(7P), **ie**(7D), **le**(7D), **pfmod**(7M)

NOTES

Older versions of **bufmod** did not support the behavioral flexibility controlled by the **SBIOCSFLAGS ioctl**. Applications that wish to take advantage of this flexibility can guard themselves against old versions of the module by invoking the **SBIOCGFLAGS ioctl** and checking for an **EINVAL** error return.

When buffering is enabled by issuing an **SBIOCSCHUNK** ioctl to set the chunk size to a non zero value, **bufmod** sends a **SETOPTS** message to adjust the stream head high and low water marks to accommodate the chunked messages.

When buffering is disabled by setting the chunk size to zero, message truncation can have a significant influence on data traffic at the stream head and therefore the stream head high and low water marks are adjusted to new values appropriate for the smaller truncated message sizes.

BUGS

bufmod does not defend itself against allocation failures, so that it is possible, although very unlikely, for the stream head to use inappropriate high and low water marks after the chunk size or snapshot length have changed.