NAME

mc_tx, **mri_tx** **-** transmit a message block chain

SYNOPSIS

#include <sys/mac_provider.h>

*mblk_t **
prefix_m_tx(*void *driver*, *mblk_t *mp_chain*);

*mblk_t **
prefix_ring_tx(*void *rh*, *mblk_t *mp_chain*);

INTERFACE LEVEL

illumos DDI specific

The **mri_tx**() entry point is **Evolving**. API and ABI stability is not guaranteed.

PARAMETERS

driver	A pointer to the driver's private data that was passed in via the *m_pdata* member of the mac_register(9S) structure to the mac_register(9F) function.
rh	A pointer to the ring's private data that was passed in via the *mri_driver* member of the mac_ring_info(9S) structure as part of the mr_rget(9E) entry point.
mp_chain	A series of mblk(9S) structures that may have multiple independent packets linked together on their *b_next* member.

DESCRIPTION

The **mc_tx**() entry point is called when the system requires a device driver to transmit data. The device driver will receive a chain of message blocks. The *mp_chain* argument represents the first frame. The frame may be spread out across one or more mblk(9S) structures that are linked together by the *b_cont* member. There may be multiple frames, linked together by the *b_next* pointer of the mblk(9S).

For each frame, the driver should allocate the required resources and prepare it for being transmitted on the wire. The driver may opt to copy those resources to a DMA buffer or it may bind them. For more information on these options, see the *MBLKS AND DMA* section of mac(9E).

As it processes each frame in the chain, if the device driver has advertised either of the MAC_CAPAB_HCKSUM or MAC_CAPAB_LSO flags, it must check whether either apply for the given frame using the mac_hcksum_get(9F) and mac_lso_get(9F) functions respectively. If either is enabled for the given frame, the hardware must arrange for that to be taken care of.

For each frame that the device driver processes it is responsible for doing one of three things with it:

1. Transmit the frame.

2. Drop the frame by calling freemsg(9F) on the individual mblk_t.

3. Return the frames to indicate that resources are not available.

The device driver is in charge of the memory associated with *mp_chain*. If the device driver does not return the message blocks to the MAC framework, then it must call freemsg(9F) on the frames. If it does not, the memory associated with them will be leaked. When a frame is being transmitted, if the device driver performed DMA binding, it should not free the message block until after it is guaranteed that the frame has been transmitted. If the message block was copied to a DMA buffer, then it is allowed to call freemsg(9F) at any point.

In general, the device driver should not drop frames without transmitting them unless it has no other choice. Times when this happens may include the device driver being in a state where it can't transmit, an error was found in the frame while trying to establish the checksum or LSO state, or some other kind of error that represents an issue with the passed frame.

The device driver should not free the chain when it does not have enough resources. For example, if entries in a device's descriptor ring fill up, then it should not drop those frames and instead should return all of the frames that were not transmitted. This indicates to the stack that the device is full and that flow control should be asserted. Back pressure will be applied to the rest of the stack, allowing most systems to behave better.

Once a device driver has returned unprocessed frames from its **mc_tx**() entry point, then the device driver will not receive any additional calls to its **mc_tx**() entry point until it calls the mac_tx_update(9F) function to indicate that resources are available again. Note that because it is the device driver that is calling this function to indicate resources are available, it is very important that it only return frames in cases where the device driver itself will be notified that resources are available again. For example, when it receives an interrupt indicating that the data that it transmitted has been completed so it can use entries in its descriptor ring or other data structures again.

The device driver can obtain access to its soft state through the *driver* member. It should cast it to the appropriate structure. The device driver should employ any necessary locking to access the transmit related data structures. Note that the device driver should expect that it may have its transmit endpoints called into from other threads while it's servicing device interrupts related to them.

The **mri_tx**() entry point is similar to the **mc_tx**() entry point, except that it is used by device drivers that have negotiated the MAC_CAPAB_RINGS capability for transmitting. The driver should follow all of the same rules described earlier, except that it will access a ring-specific data structure through *rh* and when it needs to update that there is additional space available, it must use mac_tx_update_ring(9F) and not mac_tx_update(9F).

When the **mri_tx**() entry point is called, the ring that should be used has been specified. The driver must not attempt to use any other ring than the one specified by *rh* for any reason, including a lack of resources or an attempt to perform its own hashing.

CONTEXT

The **mc_tx**() entry point may be called from **kernel** or **interrupt** context.

RETURN VALUES

Upon successful completion, the device driver should return NULL. Otherwise, it should return all unprocessed message blocks and ensure that it calls either mac_tx_update(9F) or mac_tx_ring_update(9F) some time in the future.

SEE ALSO

mac(9E), mac_capab_rings(9E), mr_rget(9E), freemsg(9F), mac_hcksum_get(9F), mac_lso_get(9F), mac_register(9F), mac_tx_ring_update(9F), mac_tx_update(9F), mac_register(9S), mac_ring_info(9S), mblk(9S)