Packet Buffers for Shared Memory Routers

You might recall from Chapter 1 that IOS maintains a set of packet buffers called *system buffers* that are used primarily for process switching packets. IOS on shared memory routers also uses system buffers, but in a distinct way—it uses the system buffers for *all* packet switching, not just process switching.

In addition to the standard public buffer pools, IOS on shared memory platforms also creates some private system buffer pools and special buffer structures for the interface controllers called *rings*.

Private Buffer Pools

Private buffer pools are used for packet switching just like the public pools, but are intended to prevent interface buffer starvation. All interfaces must compete for buffers from the public pools, increasing the probability for buffer contention (which degrades performance) and raising the possibility that a single interface can starve out others needing buffers from a single pool. With the addition of private pools, each interface is given a number of buffers dedicated for its use, so contention is minimized. Although most interfaces receive their own private pool of buffers, some lower speed interfaces, such as asynchronous interfaces, do not.

Unlike the dynamic public pools, the private pools are static and are allocated with a fixed number of buffers at IOS initialization. New buffers cannot be created on demand for these pools. If a buffer is needed and one is not available in the private pool, IOS *falls back*to the public buffer pool for the size that matches the interface's MTU.

The output of the **show buffers** command in Example 3-9 shows both the public and the private pools.

Example 3-9. *show buffers* Command Output

Router#**show buffers** Buffer elements: 500 in free list (500 max allowed) 57288024 hits, 0 misses, 0 created Public buffer pools: Small buffers, 104 bytes (total 50, permanent 50): 50 in free list (20 min, 150 max allowed) 11256002 hits, 0 misses, 0 trims, 0 created 0 failures (0 no memory) Middle buffers, 600 bytes (total 25, permanent 25): 24 in free list (10 min, 150 max allowed) 3660412 hits, 12 misses, 36 trims, 36 created 0 failures (0 no memory) Big buffers, 1524 bytes (total 50, permanent 50): 50 in free list (5 min, 150 max allowed) 585512 hits, 14 misses, 12 trims, 12 created 2 failures (0 no memory) VeryBig buffers, 4520 bytes (total 10, permanent 10): 10 in free list (0 min, 100 max allowed) 0 hits, 0 misses, 0 trims, 0 created 0 failures (0 no memory) Large buffers, 5024 bytes (total 0, permanent 0): 0 in free list (0 min, 10 max allowed) 0 hits, 0 misses, 0 trims, 0 created 0 failures (0 no memory) Huge buffers, 18024 bytes (total 0, permanent 0): 0 in free list (0 min, 4 max allowed) 0 hits, 0 misses, 0 trims, 0 created 0 failures (0 no memory) Interface buffer pools: Ethernet0 buffers, 1524 bytes (total 32, permanent 32): 8 in free list (0 min, 32 max allowed) 3398 hits, 3164 fallbacks 8 max cache size, 7 in cache Serial0 buffers, 1524 bytes (total 32, permanent 32): 7 in free list (0 min, 32 max allowed) 25 hits, 0 fallbacks 8 max cache size, 8 in cache Serial1 buffers, 1524 bytes (total 32, permanent 32): 7 in free list (0 min, 32 max allowed) 25 hits, 0 fallbacks 8 max cache size, 8 in cache

Although most of these fields are explained in Chapter 1, some are unique to the interface buffer pools:

- **fallbacks—**

  The number of times the interface processor had to fall back to the public buffer pools to find a buffer in which to store a packet.

- **max cache size—**

  Some number of buffers in each private buffer pool are cached for faster access; this is the maximum number of buffers that can be cached.

- **in cache—**

  The number of cached private buffers.

Notice the private pools do not have a **creates** or a **trims**field; this is because these pools are static pools.

Receive Rings and Transmit Rings

In addition to public and private buffer pools, IOS also creates special buffer control structures, called rings, in I/O memory. IOS and interface controllers use these rings to control which buffers are used to receive and transmit packets to the media. Rings are actually a common control structure used by many types of media controllers to manage the memory for packets being received or waiting to be transmitted. The rings themselves consist of media controller–specific elements that point to individual packet buffers elsewhere in I/O memory. IOS creates these rings on behalf of the media controllers and then manages them jointly with the controllers.

Each interface has a pair of rings: a receive ring for receiving packets and a transmit ring for transmitting packets. These rings have fixed sizes determined by several factors. The size of the receive ring depends on the specific interface and is dictated by the media controller specification. The transmit ring size, however, is dependent on the media controller specification and the type of queuing configured on the interface.

Receive rings have a constant number of packet buffers allocated to them that equals the size of the ring. The receive ring's packet buffers initially are allocated from the interface's private buffer pool. During operation, they might be replaced with buffers from either the private pool or a public pool.

The number of buffers allocated to a transmit ring can vary from zero up to the maximum size of the transmit ring. Transmit ring packet buffers come from the receive ring of the originating interface for a switched packet or from a public pool if the packet was originated by IOS. They're de-allocated from the transmit ring and returned to their original pool after the payload data is transmitted.

The **show controller** command in Example 3-10 displays the sizes and the locations of the receive and the transmit rings. Although most interface types produce output similar to what's shown, the exact content of the output varies.

Example 3-10. *show controller* Command Output Displays Receive and Transmit Ring Sizes and Locations

isp-4700b#**show controller ethernet** AM79970 unit 0 NIM slot 1, NIM type code 14, NIM version 1 Media Type is 10BaseT, Half Duplex, Link State is Down, Squelch is Normal idb 0x60AE8324, ds 0x60AE9D10, eim_regs = 0x3C110000 IB at 0x40006E64: mode=0x0010, mcfilter 0000/0000/0100/0000 station address 0060.837c.7089 default station address 0060.837c.7089 buffer size 1524 RX ring with 32 entries at 0x400303D0 Rxhead = 0x400303D0 (0), Rxp = 0x60AE9D28 (0) 00 pak=0x60AF2468 ds=0xA80C745E status=0x80 max_size=1524 pak_size=0 01 pak=0x60AF2254 ds=0xA80C6DA2 status=0x80 max_size=1524 pak_size=0 02 pak=0x60AF2040 ds=0xA80C66E6 status=0x80 max_size=1524 pak_size=0 …. TX ring with 32 entries at 0x40059BD0, tx_count = 0 tx_head = 0x40059BD0 (0), head_txp = 0x60AE9E3C (0) tx_tail = 0x40059BD0 (0), tail_txp = 0x60AE9E3C (0) 00 pak=0x000000 ds=0xA8000000 status=0x03 status2=0x0000 pak_size=0 01 pak=0x000000 ds=0xA8000000 status=0x03 status2=0x0000 pak_size=0 02 pak=0x000000 ds=0xA8000000 status=0x03 status2=0x0000 pak_size=0

The following list describes some of the more interesting fields in the **show controller** output from Example 3-10:

- **RX ring with 32 entries at 0x400303D0—**

  The size of the receive ring is 32 and it begins at memory location 0x400303D0 in I/O memory.

- **00 pak=0x60AF2468 ds=0xA80C745E status=0x80 max_size=1524 pak_size=0—**

This line is repeated for each ring entry in the receive ring. Each ring entry, called a *descriptor*, contains information about a corresponding packet buffer allocated to the receive ring. All the fields in the descriptor except for **pak** are device specific and vary from media controller to media controller. The **pak** field points to the memory address of the *header* for the packet linked to this descriptor. All IOS packet buffers have a corresponding header containing information about the contents of the buffer and a pointer to the actual location of the buffer itself. Although packet buffers on shared memory systems are located in I/O memory, their headers might reside in Local memory, as is the case here.

- **TX ring with 32 entries at 0x40059BD0, tx_count = 0—**

  Shows the transmit ring size and the number of packets waiting to be transmitted. In this case, the transmit ring size is 32 and there are no packets awaiting transmission on this interface.

- **00 pak=0x000000 ds=0xA8000000 status=0x03 status2=0x0000 pak_size=0—**

  This line is repeated for each entry in the transmit ring. Like the receive ring, each entry in the transmit ring is a descriptor containing information about a corresponding packet buffer on the ring. The **pak**field points to the header of a corresponding packet buffer waiting to be transmitted. The other fields are media controller specific. Unlike receive ring descriptors, transmit ring descriptors are linked only to a packet buffer when there is a packet waiting to transmit. After a packet buffer is transmitted, the buffer is unlinked from the transmit descriptor, returned to its original free pool, and its descriptor **pak** pointer reset to 0x000000.

< BACK Make Note | Bookmark CONTINUE >

Index terms contained in this section