# FreeBSD mbuf

- mbuf (memory buffer)

- It is a fundamental data structure used in networking stack to handle packet storage

## mbuf Chains

- When data is too large to fit in a single mbuf, mbufs can be linked together to form mbuf chain

- Each mbuf contains pointer to next mbuf.

## Source files

/usr/src/sys/sys/mbuf.h

/usr/src/sys/sys/kern/kern\_mbuf.c

/usr/src/sys/sys/kern/uipc\_mbuf.c

/usr/src/sys/sys/kern/uipc\_mbuf2.c

## Key fields in struct mbuf

- m\_data: pointer to actual data in mbuf

- m\_len: length of the data in current mbuf

- m\_next: pointer to next mbuf in the chain

- m\_flags: flags indicating special properties of the mbuf

- m\_pkthdr: packet header used for managing the metadata

- m\_ext: for mbufs using external storage, m\_ext struct holds a reference to an external buffer

# mbuf APIs

## m\_get()

- Allocates a new mbuf

### Prototype

|  |
| --- |
| struct mbuf \*m\_get(int how, short type); |

### Parameters

**how**

- specifies how memory allocation shoud be performed

- values:

1. M\_WAITOK: call can wait/sleep if necessary for memory to become available.

2. M\_NOWAIT: return immediately if no memory is available, without waiting.

**type**

- specifies type of data the mbuf will hold

- values

1. MT\_DATA: used for generic data packet

2. MT\_HEADER: used for network packet headers

3. MT\_SONAME: used for socket addresses.

### Return Value

- On success: pointer to newly allocated mbuf.

- On failure: NULL

## m\_align()

- adjust the data pointer of an mbuf so that it is aligned with the mbuf's internal buffer

- used to ensure there is enough headroom for adding data

### Prototype

|  |
| --- |
| void m\_align(struct mbuf \*m, int len); |

### Parameters

**m**

- pointer to mbuf that needs alignment

**len**

- length of data to be aligned within the mbuf

## mtod()

- convert mbuf pointer to data pointer of specified type

### Definition

|  |
| --- |
| #define mtod(m, type) ((type) ((m)->m\_data)) |

### Parameters

**m**

- pointer to mbuf that needs alignment

**type**

- desired type that you want mbuf pointer to be converted to

ex: char\*, struct iphdr \*, ...

### Return Value

- pointer to the data contained within mbuf, cast to specified type

## m\_prepend()

- add data to the front of an mbuf chain.

- useful when header needs to be prepended to an existing packet

- If there is enough space at the beginning of the 1st mbuf in the chain, m\_prepend() adjusts the m\_data pointer and updates the m\_len

- If there is not enough space, a new mbuf is allocated and the data is prepended

### Prototype

|  |
| --- |
| struct mbuf \* m\_prepend(struct mbuf \*m, int len, int how); |

### Parameters

**m**

- The mbuf chain to which data needs to be prepended

**len**

- length of the data to be prepended

**how**

- specifies how memory allocation shoud be performed

- values

1. M\_WAITOK: call can wait/sleep if necessary for memory to become available.

2. M\_NOWAIT: return immediately if no memory is available, without waiting.

### Return Value

- On success: returns modified mbuf chain/new mbuf chain

- On failure: NULL

## m\_append()

- add data to the end of an mbuf chain

- useful for appending additional data to a packet, such as adding trailers or extending the payload.

- If there is enough space at the end of the last mbuf in the chain, m\_append() directly copies the data into it

- If there is not enough space, a new mbuf is allocated and linked

### Prototype

|  |
| --- |
| int m\_append(struct mbuf \*m, int len, const char \*cp); |

### Parameters

**m**

- The mbuf chain to which data needs to be appended

**len**

- length of the data to append

**cp**

- pointer to the data that needs to be appended

### Return Value

- On success: returns 1

- On failure: returns 0

## m\_adj()

- adjust length of the data in a mbuf chain

- Trims data from the beginning or the end of the chain

- Useful for removing headers, trailers or unwanted bytes from a packet

### Prototype

|  |
| --- |
| void m\_adj(struct mbuf \*m, int len); |

### Parameters

**m**

- The mbuf chain to adjust

**len**

- Number of bytes to adjust

- If len is positive, it removes bytes from the beginning of the mbuf chain

- if len is negative, it removes bytes from the end

## m\_free()

- free a single mbuf. If there are more mbufs in the chain, it does not free them.

### Prototype

|  |
| --- |
| struct mbuf \* m\_free(struct mbuf \*m); |

### Parameters

**m**

- pointer to mbuf that should be freed

### Return Value

- returns the next mbuf in the chain(i.e., m->m\_next)

- returns NULL if the mbuf was the last one in the chain

## m\_freem()

- free an entire mbuf chain, including all the mbufs in the chain

- used when the buffer is no longer needed

### Prototype

|  |
| --- |
| void m\_freem(struct mbuf \*m); |

### Parameters

**m**

- pointer to mbuf chain that should be freed

## m\_copydata()

- copy data from an mbuf chain into a contiguous memory buffer

- traverses mbuf chain and copies ‘len’ bytes of data from the given offset ‘off’ into the buffer ‘cp’.

- handles case where data is fragmented across multiple mbufs, ensuring that the requested number of bytes is copied contiguously

- if len of mbuf is shorter than the requested amount, or if the starting offset exceeds total length of the mbuf chain, m\_copydata() may panic.

### Prototype

|  |
| --- |
| void m\_copydata(const struct mbuf \*m, int off, int len, char \*cp); |

### Parameters

**m**

* pointer to mbuf chain from which to copy the data

**off**

* offset (in bytes) from the start of the mbuf chain to begin copying the data

**len**

* number of bytes to copy

**cp**

* pointer to destination buffer where the data will be copied

## m\_copyback()

* copy data into an mbuf chain at specified offset.
* writes ‘len’ bytes of data from the source buffer ‘cp’ into the mbuf chain, starting at the specified offset ‘off’
* if the specified offset or len goes beyond the current size of mbuf chain, m\_copyback() will extend the mbuf chain by allocating new mbufs
* if the specified offset already has data, m\_copydata() will overwrite it.

### Prototype

|  |
| --- |
| void m\_copyback(struct mbuf \*m, int off, int len, const void \*cp); |

### Parameters

**m**

* pointer to mbuf chain where data will be copied

**off**

* offset (in bytes) fron the start of the mbuf chain where data should be written

**len**

* number of bytes to write

**cp**

* pointer to source data that needs to be copied into the mbuf chain

# Tx-Rx

## Using FreeBSD mbuf APIs

Sample program mbuf\_apis/bsd\_tx\_rx.c which has implementation for the below mbuf test case

**Tx path:**

Function: bsd\_txpath()

Parameters: void

Returns: struct mbuf \*

bsd\_txpath() does below operations:

1. Allocate new mbuf: m\_get(M\_NOWAIT, MT\_DATA);

2. Align mbuf data pointer to copy data “Hello World” to mbuf: m\_align(m, m->m\_len); // Total length = 11

3. Prepend 20 bytes of TCP header: m\_prepend(m, tcphdr\_len, M\_NOWAIT); // Total length = 31

4. Prepend 20 bytes of IP header: m\_prepend(m, iphdr\_len, M\_NOWAIT); // Total length = 51

5. Prepend 14 bytes of Eth header: m\_prepend(m, ethhdr\_len, M\_NOWAIT); // Total length = 65

6. Add 4 bytes FCS at the end: m\_append(m, 4, "fcs"); // Total length = 69

7. Dump mbuf data: dump\_mbuf\_data(m);

8. Return pointer to mbuf

**Rx path:**

Function: bsd\_rxpath()

Parameters: struct mbuf \*

Returns: void

bsd\_rxpath() does below operations

1. Dump mbuf data: dump\_mbuf\_data(m); // Total length = 69

2. Remove 4 bytes FCS: m\_adj(m, -fcs\_len); // Total length = 65

3. Remove 14 bytes Eth header: m\_adj(m, ethhdr\_len); // Total length = 51

4. Remove 20 bytes IP header: m\_adj(m, iphdr\_len); // Total length = 31

5. Remove 20 bytes TCP header: m\_adj(m, tcphdr\_len); // Total length = 11

6. Dump mbuf data: dump\_mbuf\_data(m);

7. Free the mbuf: m\_freem(m);

Refer mbuf\_apis/bsd\_tx\_rx\_output.txt for output

## Using linuxKPI skb APIs

Sample program skb\_linuxkpi/bsd\_skb\_tx\_rx.cwhich has implementation for the below skb test case

**Tx path:**

Function: ln\_tx\_path()

Parameters: void

Returns: struct sk\_buff \*

ln\_tx\_path() does below operations

1. Allocate new skb: alloc\_skb(len, GFP\_KERNEL);

2. Reserve headroom for the allocated skb: skb\_reserve(skb, head\_len);

3. copy data “Hello World” to skb: skb\_put(skb, data\_len); // Total length = 11

4. Prepend 20 bytes of TCP header: skb\_push(skb, sizeof(struct tcphdr)); // Total length = 31

5. Prepend 20 bytes of IP header: skb\_push(skb, sizeof(struct iphdr)); // Total length = 51

6. Prepend 14 bytes of Eth header: skb\_push(skb, sizeof(struct ethhdr)); // Total length = 65

7. Add 4 bytes FCS at the end: skb\_put(skb, 4); // Total length = 69

8. Dump skb data: print\_skb(skb);

**Rx path:**

Function: ln\_rx\_path

Parameters: struct sk\_buff \*

Returns: void

1. Dump skb\_data: print\_skb(skb); // Total length = 69

2. Remove 4 bytes FCS: skb\_trim(skb, 65); // Total length = 65

3. Remove 14 bytes Eth header: skb\_pull(skb, sizeof(struct ethhdr)); // Total length = 51

4. Remove 20 bytes IP header: skb\_pull(skb, sizeof(struct iphdr)); // Total length = 31

5. Remove 20 bytes TCP header: skb\_pull(skb, sizeof(struct tcphdr)); // Total length = 11

6. Dump skb data: print\_skb(skb);

7. Free skb: kfree\_skb(skb);

Refer skb\_linuxkpi/ bsd\_skb\_tx\_rx\_output.txt for output

# skb to mbuf mapping in Linux KPI

**Tx path:**

wlan.ko: works on mbuf

linux\_kpi.ko: mbuf data is copied to skb using m\_copydata()

iwlwifi.ko: works on skb

|  |
| --- |
| // wlan.ko  ieee80211\_vap\_transmit(…, mbuf, …)  ic->ic\_transmit(…, mbuf, …) //lkpi\_ic\_transmit |
| // linux\_kpi.ko  lkpi\_ic\_transmit(…, mbuf, …)  lkpi\_ic\_raw\_xmit(…, mbuf, …)  task\_enqueue(txq\_task);  lkpi\_80211\_txq\_task()  lkpi\_80211\_txq\_tx\_one(…, struct mbuf \*m)  {  skb=dev\_alloc\_skb(…);  skb\_reserve(…);  skb\_put(…);  m\_copydata(); // copy mbuf data to skb  }  lhw->ops->tx(…, skb, …); // iwl\_mvm\_mac\_tx |

**Rx path:**

iwlwifi.ko: works on skb

linux\_kpi.ko: skb data is copied to mbuf using m\_copyback()

wlan.ko: works on mbuf

|  |
| --- |
| // iwlwifi.ko  ieee80211\_rx\_napi(…, skb, …)  linuxkpi\_ieee80211\_rx(…, skb, …) |
| // linux\_kpi.ko  linuxkpi\_ieee80211\_rx(…, skb, …)  {  struct mbuf \*m = m\_get2(…);  m\_copyback(…);  } |

# References

<https://man.freebsd.org/cgi/man.cgi?query=mbuf&apropos=0&sektion=0&manpath=FreeBSD+14.1-RELEASE+and+Ports&arch=default&format=html>