SKB是linux kernel裡面的一種結構struct sk_buff，這結構是用來管理網路卡所要收進來的(或是要送出去的)封包，在網路上每一個封包都是在kernel裡面佔了一塊記憶體，而這塊記憶體都是經由這個skb結構以及其相關函式來管理，而管理這塊記憶體是藉由skb最重要的四個指標，分別是head、data、tail、end。

skb產生與拷貝

- 產生

  alloc_skb dev_alloc_skb

- 拷貝

  skb_clone skb_copy skb_realloc_headroom

- 初使化

  skb_headerinit skb_init

- 刪除

  kfree_skbmem __kfree_skb skb_orphan kfree_skb kfree_skb_fast

- 條件取代

  skb_cow

- 查詢

  skb_cloned skb_shared skb_datarefp

- 設定

  skb_unshare

skb相關的memory管理

  skb_push __skb_push skb_pull __skb_pull skb_put __skb_put skb_trim __skb_trim skb_reserve

skb_headroom skb_tailroom

skb相關訊息

skb_over_panic skb_under_panic show_net_buffers

---

skb list相關的操作

- 初使化

  skb_queue_head_init

- 插入

  skb_insert __skb_insert skb_append __skb_append skb_queue_head __skb_queue_head skb_queue_tail __skb_queue_tail

- 查詢

  skb_queue_len skb_peek skb_peek_tail skb_queue_empty

- 刪除

  skb_unlink __skb_unlink skb_dequeue __skb_dequeue skb_dequeue_tail __skb_dequeue_tail skb_queue_purge

---

struct sk_buff *skb_clone(struct sk_buff *skb, int gfp_mask)

- 功能：將skb這個多功能的指標再多複製一份，但是真正的packet資料沒有
- 參數：struct sk_buff、int gfp_mask
- 回傳值：回傳所複製出來的sk_buff
- 解釋：

  利用kmem_cache_alloc這函式，在cache上alloc一個新的skb叫n。

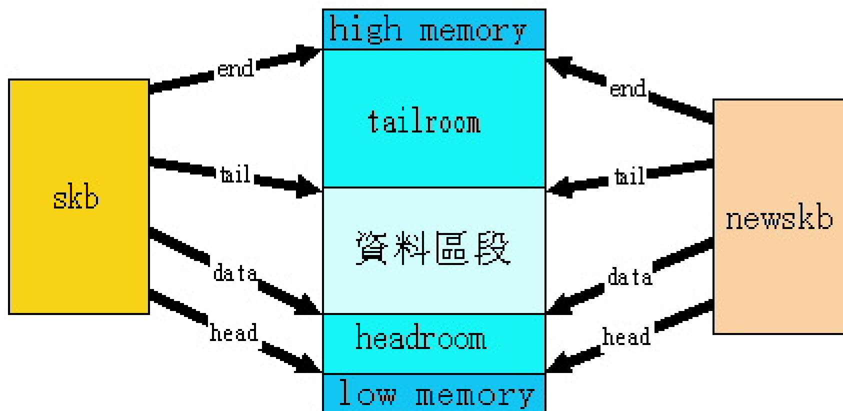
  使用memcpy這函式，將舊的skb複製到新的skb

  一些skb的相關設定，其中較重要的是skb_cloned這是用來判別那一個skb是clone出來的

被clone出來的設為n->is_clone=1

原來的skb會被設定skb->cloned=1(表示clone過了)、atomic_inc(skb_datarefp(skb))這樣表示真正的packet資料又多一個skb來指著reference到它了

相關設定中還包括了一項很重要的東西，就是destination的拷貝，這項操作由dst_clone(skb->dst)來完成



extern __inline__ __u32 skb_queue_len(struct sk_buff_head *list_)

- 功能：查詢skb list有多長（有多少element）
- 參數：一個skb list
- 回傳值：skb list的長度
- 解釋：

回傳struct sk_buff_head成員qlen

extern __inline__ atomic_t *skb_datarefp(struct sk_buff *skb)

功能：查詢這個skb它的資料區被多少其他的skb所reference到

參數：一個查詢的skb

回傳值：所有有reference到這個skb的數目

skb_queue_tail

- 功能：將所傳進來的skb(newsk)，接到所傳進來的skb queue(list)的尾端
- 參數：struct sk_buff_head *list, struct sk_buff *newsk
- 回傳值：無
- 解釋：

  將newsk插到list這個double link list的後面

extern __inline__ void skb_queue_head(struct sk_buff_head *list, struct sk_buff *newsk)

- 功能：將所傳進來的skb(newsk)，接到所傳進來的skb queue(list)的尾端
- 參數：struct sk_buff_head *list, struct sk_buff *newsk
- 回傳值：無
- 解釋：

  將newsk插到list這個double link list的前面

extern __inline__ void skb_queue_head_init(struct sk_buff_head *list)

- 功能：將所傳進來的skb list，初使化成沒有任何skb有其中
- 參數：struct sk_buff_head* list
- 回傳值：無
- 解釋：

  將list的next和prev的指標指向自己本身

  並且將qlen設為０，表示這list沒有skb element

extern __inline__ int skb_cloned(struct sk_buff *skb)

- 功能：看看這個skb是否被clone過了(這個多功能的指標是否是clone出來的)

- 參數：struct sk_buff* skb
- 回傳值：skb->cloned && atomic_read(skb_datarefp(skb)) != 1
- 解釋：

  依照skb的成員函數中的cloned，只要是經過了skb_clone都會在這寫下記錄

  寫下的記錄是n->cloned=1和atomic_inc(skb_datarefp(skb))

extern __inline__ int skb_shared(struct sk_buff *skb)

- 功能：看看這個skb是否被share的
- 參數：struct sk_buff* skb
- 回傳值：atomic_read(&skb->users) != 1
- 解釋：

  若skb->user為不為1時，表有兩個人以上來共用這個skb

extern __inline__ int skb_queue_empty(struct sk_buff_head *list)

- 功能：看看這個skb list是否是空的list
- 參數：struct sk_buff_head *list
- 回傳值：list->next == (struct sk_buff *) list
- 解釋：

  當list的next和prev這那個指標，指到了自己本身，表示為到了list的兩端點的其中之一

  而next指向自己，表示第一個元素就是自己(表示為空的)

void __init skb_init(void)

- 功能：初使化 skbuff SLAB cache
- 參數：無
- 回傳值：無
- 解釋：

要求一塊為一個skb大小的cache，而他的offset開始是從０，且其在記憶體中的排列會跟硬體的cache line做對齊，而且建構函數為skb_headerinit，但是沒有解構函數。

測試是否creation有無成功。

註：這個函數在sock_init()時，有呼叫到一次

void skb_over_panic(struct sk_buff *skb, int sz, void *here)

- 功能：用來告知skb管理發生了panic的狀況了
- 參數：發生panic的skb、超出範圍的size(sz)、是從那邊開始超出範圍(here)
- 回傳值：無
- 解釋：

當skb的tail指標指的memory address超出了skb的end這個指標所指的memory address時，就是發生了skb_over_panic，因為放資料的地方，放超出了整個skb的範圍了。

特別容易發生在skb_put，在2.2.17中只有在這邊被使用到了。

void skb_under_panic(struct sk_buff *skb, int sz, void *here)

- 功能：用來告知skb管理發生了panic的狀況了
- 參數：發生panic的skb、超出範圍的size(sz)、是從那邊開始超出範圍(here)
- 回傳值：無
- 解釋：

當skb的data指標指的memory address超出了skb的head這個指標所指的memory address時，就是發生了skb_under_panic，因為放資料的地方，放超出了整個skb的範圍了。

特別容易發生在skb_push，在2.2.17中只有在這邊被使用到了。

void show_net_buffers(void)

- 功能：秀出和net相關的buffer訊息
- 參數：無
- 回傳值：無：
- 解釋：

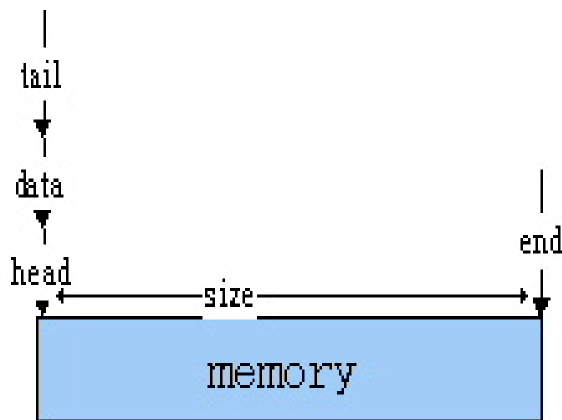
    印出現在正在使用中的skb的個數

    印出有被alloc成功的skb數

    印出alloc失敗的skb數

struct sk_buff *alloc_skb(unsigned int size,int gfp_mask)

- 功能：alloc一塊size大小的記憶體，這塊記憶體由回傳的skb來管理
- 參數：size這是用來說要多大的記憶體，gfp_mask用來設定skb選項
- 回傳值：回傳從快取alloc出來的skb，這skb用來管理所要alloc的記憶體
- 解釋：



    檢查gfp_mask，若是屬於有__GFP_WAIT的話，會有被中斷的可能，要將這個選項給disable

    從memory cache那邊alloc出一個skb

    檢查從memory cache是否alloc成功？

    要將size變成16的倍數，原因是為了在記憶體有一定的對齊，這樣會比較好管理(packet以16byte)

    從memory那邊alloc一塊大小為size + sizeof(atomic_t)的memory，以供將來用來放封包資料用的

檢查從memory是否alloc成功？

增加net_alloc這個數值，其相當是一個系統的記錄，表示成功alloc出來的skb

skb的truesize設定為size

增加net_skbcount這個數值，其相當是一個系統的記錄，表示正在使用的skb

設定skb的四個管理memory指標，成為如右圖的樣子

最後設定這個skb不是被clone過，也不是clone品，len為０

回傳這個skb

static inline void skb_headerinit(void *p, kmem_cache_t *cache, unsigned long flags)

- 功能：其實p是一個skb，這個函數最主要就是對這個skb來進行初使化的動作，所做設定通常是設為NULL表為無人使用或最低
- 參數：void* p（其實是skb)，kmem_cache_t* cache（沒用到），unsigned long flags（沒用到）
- 回傳值：無
- 解釋：

用一個型態為struct sk_buff* skb來接void* p，將它當成一個skb來使用

將skb的next、prev、list、sk設為NULL，表示不屬任何一個skb list及不屬任何一種sock

設成目前沒有check sum、security level最低、沒有目的地、priority最低、ip option設為空的。

extern __inline__ void kfree_skb(struct sk_buff *skb)

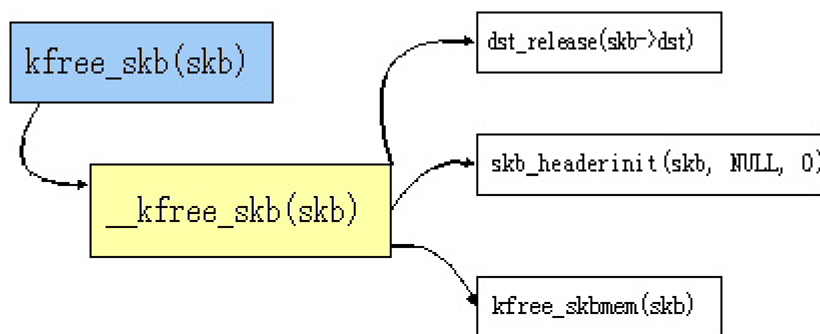
- 功能：檢查是否有user共用，等沒人共用時，再將skb這個header給清除

狀態，並且將其所管的memory給free掉

- 參數：一個skb
- 回傳值：無
- 解釋：

    其實這個函式是一個介面，它會call其他函式來完成任務，真正完成任務是其他的函式，不是它

    而它會檢查這個skb到底有多少user共用它，會一直將共用user減少並且再測試，一直到沒有人共有才執行__kfree_skb



void __kfree_skb(struct sk_buff *skb)

- 功能：將所傳進來的skb這個header給清除狀態
- 參數：一個skb
- 回傳值：無
- 解釋：

    首先它會釋放和這個skb有關的destination

    若這skb有destructor的話，就呼叫destructor

    利用skb_headerinit來清除這個skb的狀態

    呼叫kfree_skbmem(skb)來將所管理的memory給free掉

extern __inline__ void kfree_skb_fast(struct sk_buff *skb)

- 功能：檢查到只有一個user在用它時，直接呼叫kfree_skbmem將skb給

free掉
- 參數：一個skb
- 回傳值：無
- 解釋：

  它不像kfree_skb還經過呼叫了__kfree_skb，而是像kfree_skb經過了user數目的檢查後就直接callkfree_skbmem將skb給free掉，不像kfree_skb這麼麻煩，還多呼叫了幾個函式。

void kfree_skbmem(struct sk_buff *skb)

- 功能：將所傳進來的skb從快取中給free掉，並且將其所管理的memory給free掉
- 參數：一個skb
- 回傳值：無
- 解釋：

  檢查是否其他的skb也管理跟它一樣的memory，就是看看有沒有被clone過或是skb的data有沒有被reference到。

  若沒有其他的skb管理同一塊，所以可以將其所管理的memory給free掉。

  從memory cache中將skb給清楚

  減少net_skbcount表示少一個正在使用的skb

extern __inline__ void skb_insert(struct sk_buff *old, struct sk_buff *newsk)

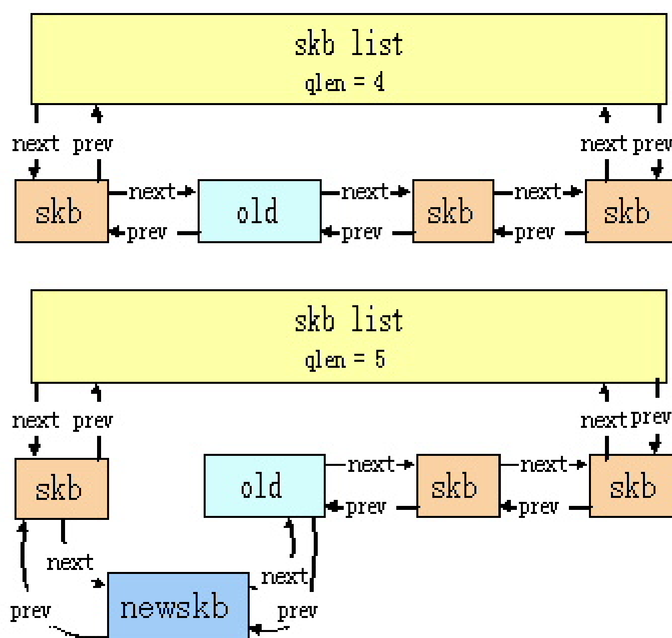
- 功能：將newsk這個skb插到原本old所屬的skb list的位置，會在old前面
- 參數：兩個skb，一個是old指出要插入的位置和所要插入的list，一個是要插入的skb
- 回傳值：無
- 解釋：

這個函式不是做真正插入skb list的動作，而真正作用的是__skb_insert

而其所做只是在呼叫__skb_insert(newsk, old->prev, old, old->list)之前，

要將這個list給lock，不給其他的skb來使用access

以免發生race condition，而做完之後就再把list給unlock。



extern __inline__ void __skb_insert(struct sk_buff *newsk, struct sk_buff * prev, struct sk_buff *next,struct sk_buff_head * list)

- 功能：將newsk插到prev和next這兩個skb之間，而其所屬的skb list是list
- 參數：３個skb，一個skb list，newsk是要新插入的skb，而prev和next是屬於list的skb
- 回傳值：無
- 解釋：

操作方式如skb_insert的圖。

extern __inline__ void skb_append(struct sk_buff *old, struct sk_buff *newsk)

- 功能：Place a packet after a given packet in a list.
- 參數：２個skb，會將newsk插到old這個skb的後面
- 回傳值：無
- 解釋：

  這個函式會將old所屬的list給lock住，以防其他的skb來access skb list，造成race condition

  呼叫__skb_append，__skb__append利用__skb_insert來模擬出來。

  PS：skb_insert和skb_append操作模式都是一樣的，只是一個是插在old的前面，另一個是跟在old的後面

extern __inline__ void skb_unlink(struct sk_buff *skb)

- 功能：將skb從其所屬的skb list中解除連結
- 參數：１個skb，就是所要unlink的skb
- 回傳值：無
- 解釋：

  因為又要修改skb list，所以先把所屬的list給鎖住，以免造成其他的skb也access到

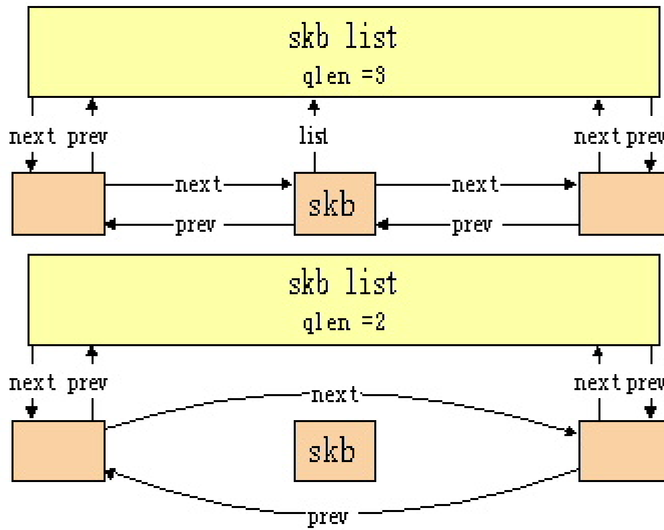
  而造成race condition，看看其所屬的list在不在，若在的話，就呼叫__skb_unlink真

  正的將這skb從list移除。

  解除list lock

extern __inline__ void __skb_unlink(struct sk_buff *skb, struct sk_buff_head *list)

- 功能：真正地將skb從skb list中解除連結

- 參數：skb和skb其所屬的skb list
- 回傳值：無
- 解釋：如右圖
    - 

extern __inline__ struct sk_buff *skb_peek_tail(struct sk_buff_head *list_)

- 功能：用來看看skb list的最後一個skb是否存在，以防這個list已經是空的，還對它進行dequeue的動作。
- 參數：一個skb list
- 回傳值：一個skb
- 解釋：

    利用struct sk_buff_head的成員prev取得skb list的最後一個skb，並看看這個skb是否等於他自己(list)，若是的話表示不存在，就回傳NULL

    若存在，則回傳所得到的skb。

extern __inline__ struct sk_buff *skb_peek_tail(struct sk_buff_head *list_)

- 功能：用來看看skb list的最前面的skb是否存在，以防這個list已經是空的，還對它進行dequeue的動作。
- 參數：一個skb list
- 回傳值：一個skb

- 解釋：

 利用struct sk_buff_head的成員next取得skb list的最前一個skb，並看看這個skb是否等於他自己(list)，若是的話表示不存在，就回傳NULL

 若存在，則回傳所得到的skb。

extern __inline__ struct sk_buff *skb_dequeue(struct sk_buff_head *list)

- 功能：從skb list的中，取出第一個skb出來，並且將這skb和skb list的連結都給斷掉。
- 參數：一個skb list
- 回傳值：一個skb
- 解釋：

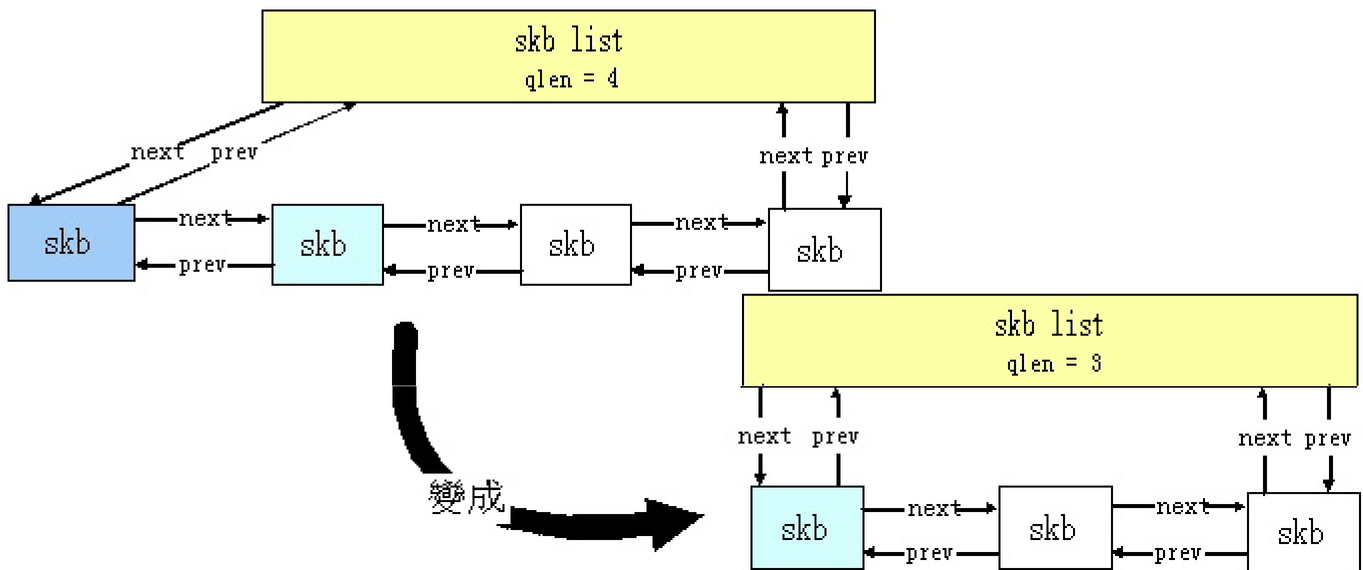
 因為又要修改skb list，所以先把所屬的list給鎖住，以免造成其他的skb也access到而造成race condition

 呼叫__skb_dequeue來做真正將連結斷掉的動作

extern __inline__ struct sk_buff *__skb_dequeue(struct sk_buff_head *list)

- 功能：從傳進來的skb list中，取出它的第一個skb並且將這skb跟這個list的所有連結給丟掉並傳回去。
- 參數：一個skb list
- 傳回值：一個skb
- 解釋：

 如圖



extern __inline__ struct sk_buff *skb_dequeue_tail(struct sk_buff_head *list)

- 功能：從skb list的中，取出尾端第後一個skb出來，並且將這skb和skb list的連結都給斷掉。
- 參數：一個skb list
- 回傳值：一個skb
- 解釋：

因為又要修改skb list，所以先把所屬的list給鎖住，以免造成其他的skb也access到而造成race condition

呼叫__skb_dequeue_tail來做真正將連結斷掉的動作

extern __inline__ struct sk_buff *__skb_dequeue_tail(struct sk_buff_head *list)

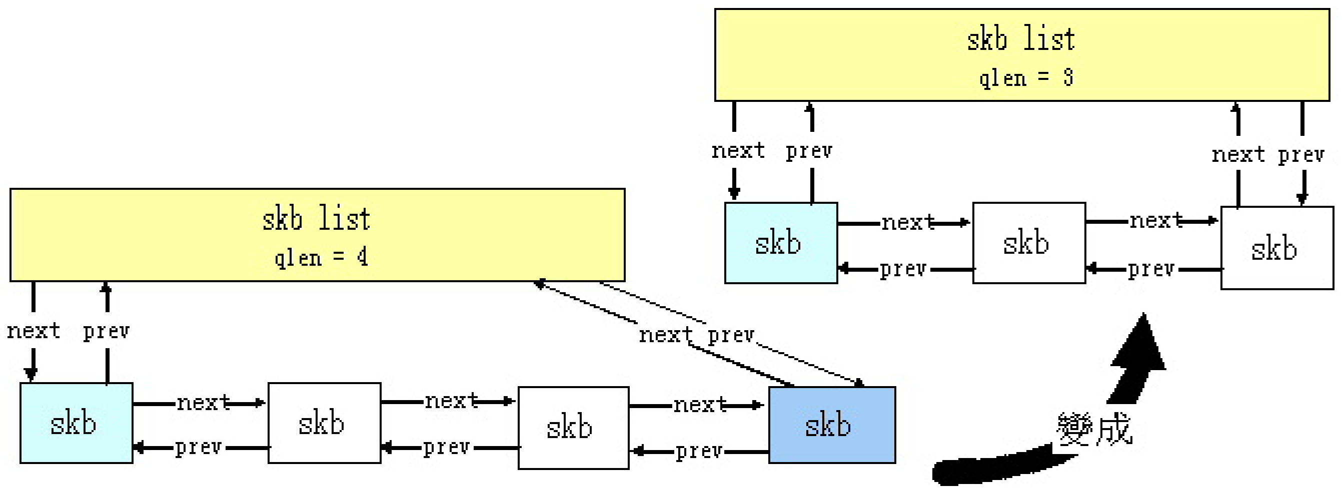
- 功能：從傳進來的skb list中，取出它的尾端最後一個skb並且將這skb跟這個list的所有連結給丟掉並傳回去
- 參數：一個skb list
- 傳回值：一個skb
- 解釋：

利用skb_peek_tail來找到最

後一個skb。

再利用skb_unlink來解除

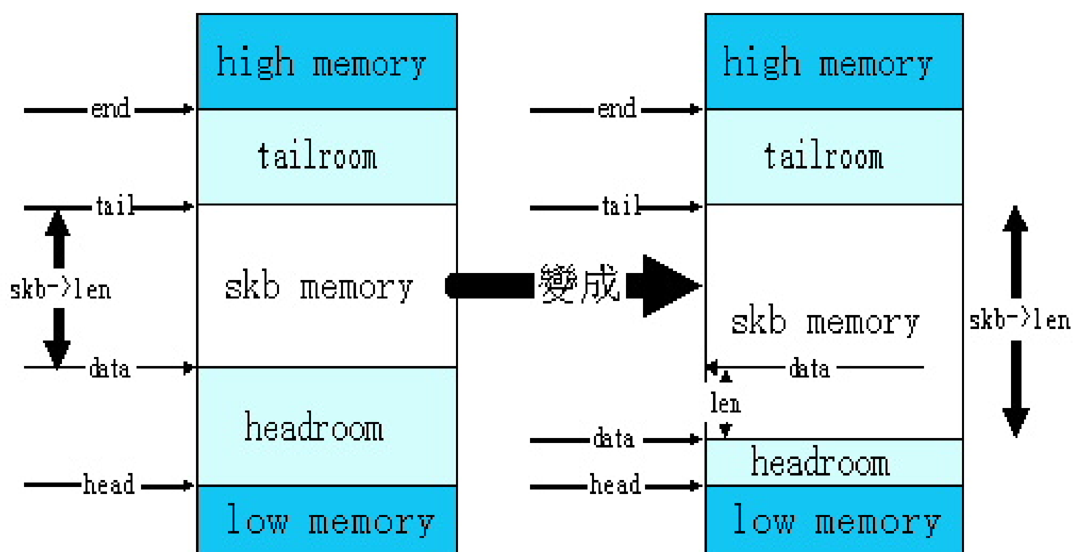
連結，如圖



extern __inline__ unsigned char *skb_push(struct sk_buff *skb, unsigned int len)

- 功能：在不造成skb_under_panic的情況下，將skb的data的指標往下移(增加資料區的可用空間)
- 參數：一個skb、所要增加資料區空間的長度len
- 傳回值：經過調整過後的skb->data的指標
- 解釋：

經過檢查後不會造成skb_under_panic的情況時，就會將skb的data的指標往下移，情形如下圖



extern __inline__ unsigned char *__skb_push(struct sk_buff *skb, unsigned int len)

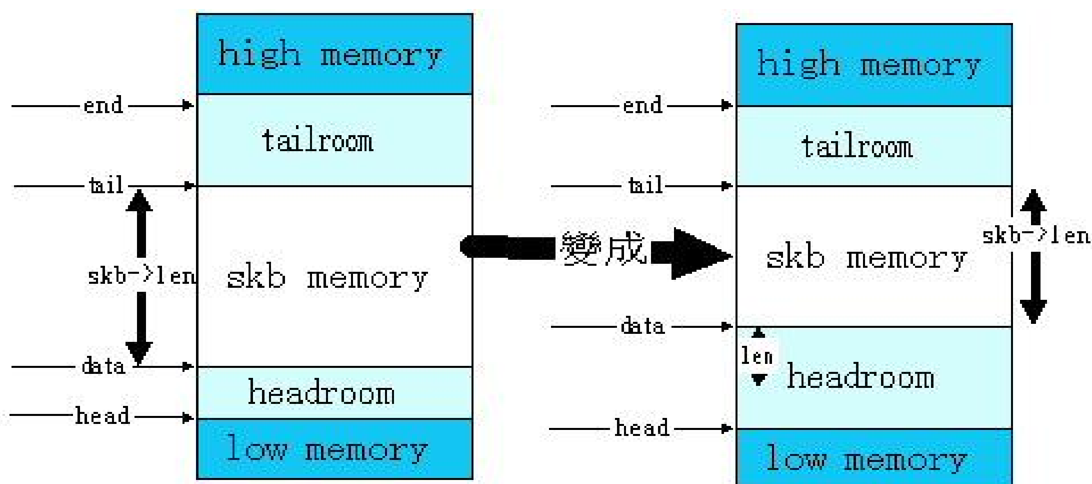
- 功能：將skb的data的指標往下移(增加資料區的可用空間)
- 參數：一個skb、所要增加資料區空間的長度len
- 傳回值：經過調整過後的skb->data的指標
- 解釋：

功能和skb_push是一樣的，但是它不會檢查會不會造成skb_under_panic的情況時，就將skb的data的指標往下移

extern __inline__ unsigned char * skb_pull(struct sk_buff *skb, unsigned int len)

- 功能：將skb的data指標往上移（減少資料區的可用空間）
- 參數：一個skb、所要增加資料區空間的長度len
- 傳回值：經過調整過後的skb->data的指標
- 解釋：

會先檢查傳進來的len是否大於skb->len，若大於表示是錯誤的將不執行pull的動作，執行情形如下圖



extern __inline__ char *__skb_pull(struct sk_buff *skb, unsigned int len)

- 功能：將skb的data指標往上移（減少資料區的可用空間）
- 參數：一個skb、所要增加資料區空間的長度len
- 傳回值：經過調整過後的skb->data的指標
- 解釋：

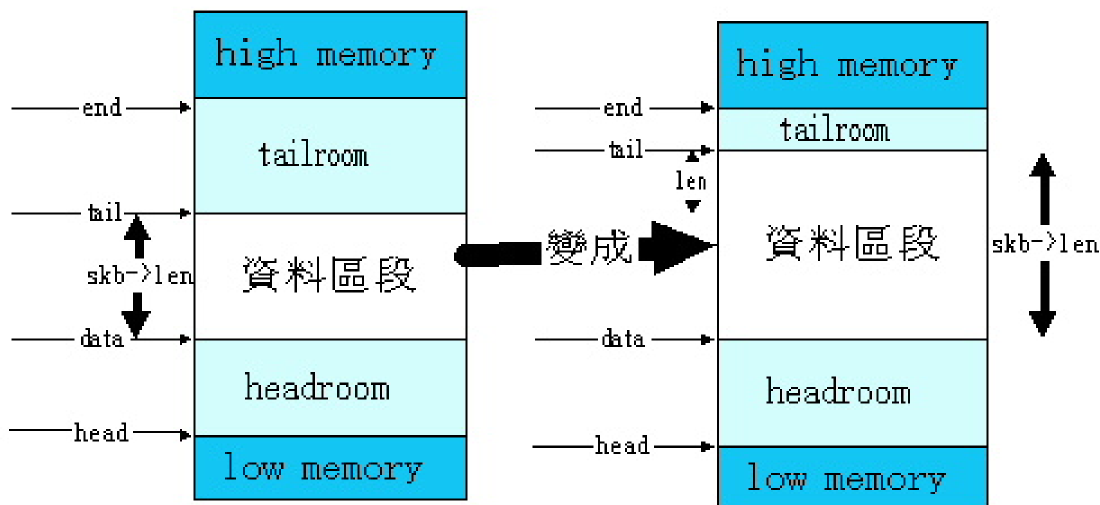
功能和skb_pull一樣，但是不會先檢查傳進來的len是否大於skb-

>len

extern __inline__ unsigned char *skb_put(struct sk_buff *skb, unsigned int len)

- 功能：在不造成skb_over_panic的情況下，將skb的tail的指標往上移(增加資料區的可用空間)
- 參數：一個skb、所要增加資料區空間的長度len
- 傳回值：經過調整過後的skb->tail的指標
- 解釋：

經過檢查後不會造成skb_over_panic的情況時，就會將skb的tail的指標往上移，情形如下圖



extern __inline__ unsigned char *__skb_put(struct sk_buff *skb, unsigned int len)

- 功能：將skb的tail的指標往上移(增加資料區的可用空間)
- 參數：一個skb、所要增加資料區空間的長度len
- 傳回值：經過調整過後的skb->tail的指標
- 解釋：

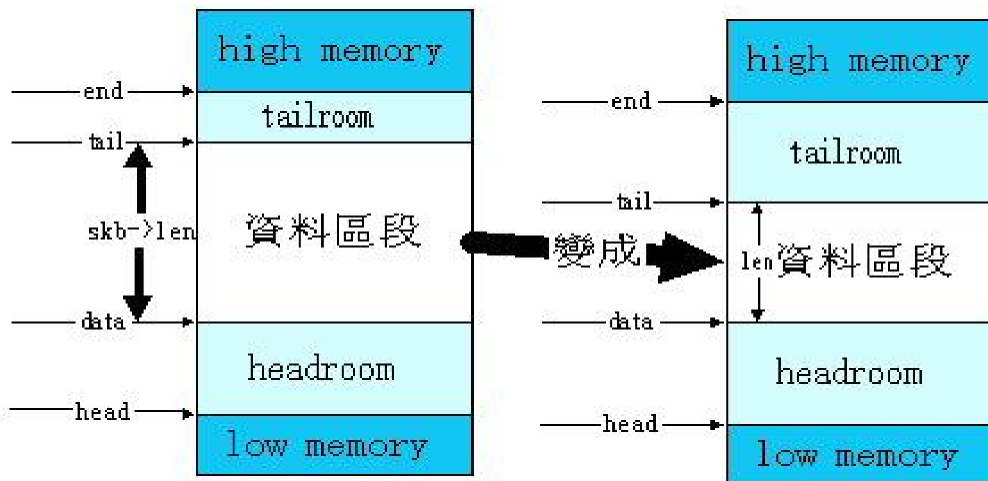
和skb_put的功能是一樣的，只是它不會檢查會不會造成skb_over_panic的情況。

extern __inline__ void skb_trim(struct sk_buff *skb, unsigned int len)

- 功能：將skb->data和skb->tail之間的長度，修剪縮小到所要的len的長度
- 參數：一個skb、所要資料區空間的長度len

- 傳回值：無
- 解釋：

  先skb->data和skb->tail之間的長度，若這長度大於所要求的長度len的話，就修剪成所要的len長度，而修剪的方式是將skb->tail往下移，造成資料區段縮小到所要的len，若小於或等於的話就不做任何事。示意圖如下



extern __inline__ void __skb_trim(struct sk_buff *skb, unsigned int len)

- 功能：將skb->data和skb->tail之間的長度，修剪縮小到所要的len的長度
- 參數：一個skb、所要資料區空間的長度len
- 傳回值：無
- 解釋：

  功能和skb_trim是一樣的，只是這個函式不做任何的檢查就直接做修剪的動作了

extern __inline__ void skb_orphan(struct sk_buff *skb)

- 功能：若解構式存在，則起動解構來解除skbuff
- 參數：一個skb
- 傳回值：無
- 解釋：

  若解構式存在，則起動解構來解除skbuff

  將skb->destructor和skb->sk設為null

extern __inline__ void skb_queue_purge(struct sk_buff_head *list)

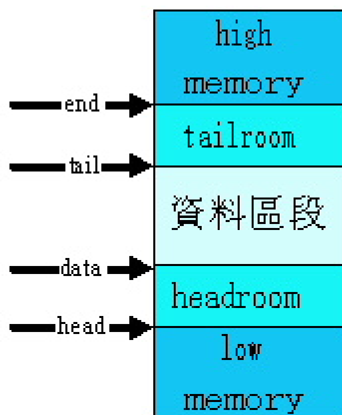
- 功能：將skb list中的skb一個個拿出來給free掉
- 參數：一個skb list
- 傳回值：無
- 解釋：

  利用skb_dequeue將 list中的所有的skb，一個個拿出來並使用kfree_skb來將skb給free掉

extern __inline__ int skb_headroom(struct sk_buff *skb)

- 功能：用來詢問skb所管理的memory，其headroom有多大
- 參數：一個skb
- 傳回值：headroom大小
- 解釋：

  詢問skb所管理的memory中，head到data之間的空間有多大，情形如圖的headroom



extern __inline__ int skb_tailroom(struct sk_buff *skb)

- 功能：用來詢問skb所管理的memory，其tailroom有多大
- 參數：一個skb
- 傳回值：tailroom大小
- 解釋：

  詢問skb所管理的memory中，tail到end之間的空間有多大，情形

如skb_headroom附圖的tailroom

extern __inline__ struct sk_buff *skb_unshare(struct sk_buff *skb, int pri)

- 功能：依照所要的設定的privilege(pri)來設定skb
- 參數：原本的skb，所要的privilege
- 回傳值：改好的skb
- 解釋：

先看看原本的skb是不是被clone過了，如果被clone過了就馬上回傳原本的skb

若沒有被clone過的話，就將pri當作是gfp_mask做為skb_copy的參數來重新拷貝一分skb

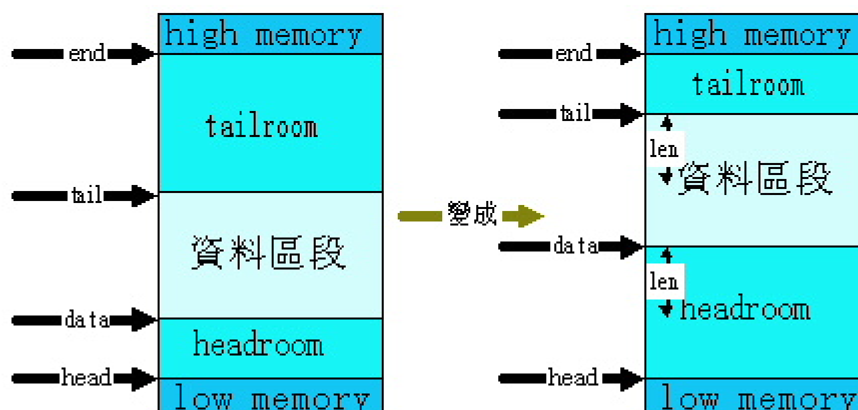
然後將原本的free掉，之後將新拷貝好的給回傳回去。

extern __inline__ void skb_reserve(struct sk_buff *skb, unsigned int len)

- 功能：將整個資料區段往tailroom移個len的空間，故tailroom減少空間len，而headroom得到多出的空間len
- 參數：一個skb、所要何留的大小len
- 傳回值：無
- 解釋：

這個函式在headroom那個區域，data指標往上移len

在tailroom那個區域，tail指標往上移len



extern __inline__ struct sk_buff *dev_alloc_skb(unsigned int length)

- 功能：可依照device的不同，而給定不同的的長度，來造出一個skb，並且會保留16byte給device來用
- 參數：所要的長度
- 傳回值：一個skb
- 解釋：

先用alloc_skb來配置一個length+16的skb之後，如果配置成功的話，再利用skb_reserve來保留空間以後放device的data。

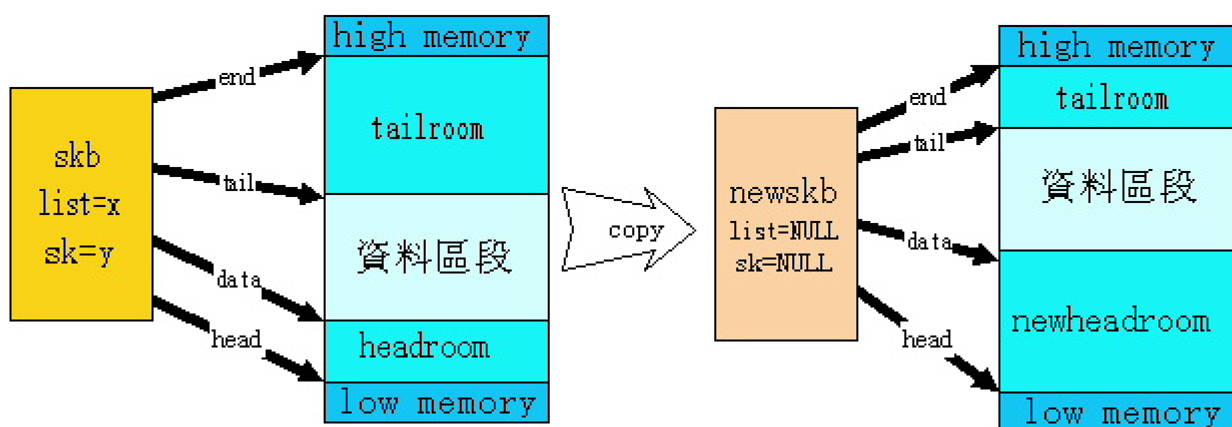
struct sk_buff *skb_realloc_headroom(struct sk_buff *skb, int newheadroom)

- 功能：因為需要比較大的headroom時，這時就可以呼叫這個函式來得到一個比較大的headroom的skb
- 參數：原本的skb、所需headroom的大小
- 傳回值：擁有所需大小的headroom的skb
- 解釋：

先利用skb_headroom求得原本skb headroom的大小

使用alloc_skb配置一塊可以管理skb->truesize+newheadroom-headroom這樣大memory的skb

再用skb_reserve保留newheadroom的空間



使用memory將原本skb的資料區段，拷貝到newskb的資料區段

n->list=NULL;沒有加入任何

list n->sk=NULL;沒有屬於任何sock

n->priority=skb->priority;priority和舊的skbuff一樣

n->protocol=skb->protocol;protocol也和舊的skbuff一樣

n->dev=skb->dev;所要output或input的的device也和舊的skbuff一樣

n->dst=dst_clone(skb->dst);所要傳的目的位址也和舊的skbuff一樣

n->h.raw=skb->h.raw+offset;transport layer也和舊的skbuff一樣

n->nh.raw=skb->nh.raw+offset;network layer也和舊的skbuff一樣

n->mac.raw=skb->mac.raw+offset;mac layer也和舊的skbuff一樣

memcpy(n->cb, skb->cb, sizeof(skb->cb));ip option也和舊的skbuff一樣

n->used=skb->used;是否有被使用過的狀況也和舊的skbuff一樣

n->is_clone=0;不是屬於clone後的產品 atomic_set(&n->users, 1);目前屬於一個使用者來用這新的skb

n->pkt_type=skb->pkt_type;pkt_type也和舊的skbuff一樣

n->stamp=skb->stamp; n->destructor = NULL;

n->security=skb->security;新的skb的security也和舊的skbuff一樣

這個函式最主要就是為了要得到一個新的memory，且這塊memory的headroom是所想要的headroom大小，所以有被拷貝的不只是skb還有skb所管理的memory

struct sk_buff *skb_copy(struct sk_buff *skb, int gfp_mask)

- 功能：拷貝一分一模一樣的skb和其管理的memory
- 參數：原來的skb，和gfp MASK
- 回傳值：新拷貝出來的skb
- 解釋：

  用alloc_skb配置一塊skb->end-skb->head大小的memory，用來給新拷貝出來的skb來管理

  newskb用skb_reserve何留跟原本skb一樣的headroom(skb->data-skb->head)

  使用memcpy將skb->head到skb->end之間的資料，拷貝到newskb的newkskb->head到newskb->end之間

  newskb->csum = skb->csum;

  newskb->list=NULL;

  newskb->sk=NULL;

  newskb->dev=skb->dev;

  newskb->priority=skb->priority;

  newskb->protocol=skb->protocol;

  newskb->dst=dst_clone(skb->dst);

  newskb->h.raw=skb->h.raw+offset;

  newskb->nh.raw=skb->nh.raw+offset;

  newskb->mac.raw=skb->mac.raw+offset;

  memcpy(newskb->cb, skb->cb, sizeof(skb->cb));

  newskb->used=skb->used;

  newskb->is_clone=0; 不是clone品

atomic_set(&newskb->users, 1);只有一個user在使用這個skb

newskb->pkt_type=skb->pkt_type;packet type和原本的skb一樣

newskb->stamp=skb->stamp;time stamp跟原本的一樣

newskb->destructor = NULL;

newskb->security=skb->security;security level跟原本的一樣

回傳newskb

extern __inline__ struct sk_buff *skb_cow(struct sk_buff *skb, unsigned int headroom)

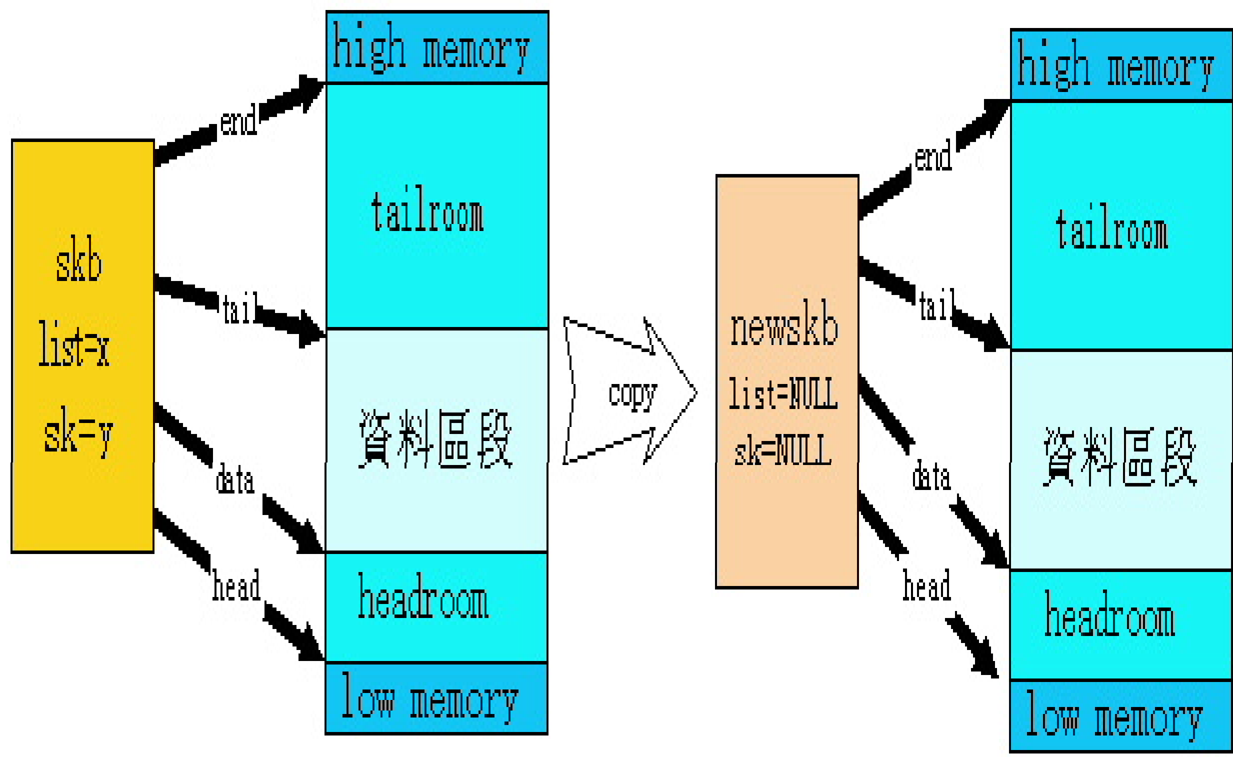
- 功能：會看看headroom是否有大於skb的headroom，若有大於就重造一個skb和其memory並取代原本的
- 參數：一個用來判斷的skb，所要的headroom
- 回傳值：一個skb
- 解釋：

將所要的headroom變成16的倍數，因為這樣比較好管理

如果headroom大於原本skb的headroom或者是這個原本的skb是被clone過了的話，

就呼叫skb_realloc_headroom來重造一個skb叫skb2，將原本skb給free掉，用skb2取代skb

回傳skb


skb
list=x
sk=y
end
tail
data
head
high memory
tailroom
資料區段
headroom
low memory
copy
newskb
list=NULL
sk=NULL
end
tail
data
head
high memory
tailroom
資料區段
headroom
low memory