第二章 基础组件

内存管理

LwIP 内存管理部分（mem.h mem.c）比较灵活，支持多种分配策略，有运行时库自带的内存分配(MEM_LIBC_MALLOC)，有内存池分配(MEM_USE_POOLS)，有动态内存堆分配，这些分配策略可以通过宏定义来更改。在嵌入式系统里面，C 运行时库自带的内存分配一般情况下很少用，更多的是后面二者，下面就这两种分配策略进行简单的分析：

动态内存堆分配

其原理就是在一个事先定义好大小的内存块中进行管理，其内存分配的策略是采用最快合适（First Fit）方式，只要找到一个比所请求的内存大的空闲块，就从中切割出合适的块，并把剩余的部分返回到动态内存堆中。在分配的内存块前大约有 12 字节会存放内存分配器管理用的私有数据，该数据区不能被用户程序修改，否则导致致命问题。

批注 [user1]: 是 First Fit 还是 Next Fit 需要再仔细阅读代码确认。

内存释放的过程是相反的过程，但分配器会查看该节点前后相邻的内存块是否空闲，如果空闲则合并成一个大的内存空闲块。

采用这种分配策略，其优点就是内存浪费小，比较简单，适合用于小内存的管理，其缺点就是如果频繁的动态分配和释放，可能会造成严重的内存碎片，如果在碎片情况严重的话，可能会导致内存分配不成功。对于动态内存的使用，比较推荐的方法就是分配->释放->分配->释放，这种使用方法能够减少内存碎片。

mem_init( ) 内存堆的初始化，主要是告知内存堆的起止地址，以及初始化空闲列表，由 lwip 初始化时自己调用，该接口为内部私有接口，不对用户层开放。

mem_malloc( ) 申请分配内存。将总共需要的字节数作为参数传递给该函数，返回值是指向最新分配的内存的指针，而如果内存没有分配好，则返回值是 NULL，分配的空间大

小会收到内存对齐的影响，可能会比申请的略大。返回的内存是“没有“初始化的。这块内存可能包含任何随机的垃圾，你可以马上用有效数据或者至少是用零来初始化这块内存。内存的分配和释放，不能在中断函数里面进行。内存堆是全局变量，因此内存的申请、释放操作做了线程安全保护，如果有多个线程在同时进行内存申请和释放，那么可能会因为信号量的等待而导致申请耗时较长。

mem_calloc( ) 是对 mem_malloc( )函数的简单包装，他有两个参数，分别为元素的数目和每个元素的大小，这两个参数的乘积就是要分配的内存空间的大小，与 mem_malloc()不同的是它会把动态分配的内存清零。有经验的程序员更喜欢使用 mem_malloc ()，因为这样的话新分配内存的内容就不会有什么问题，调用 mem_malloc ()肯定会清 0，并且可以避免调用 memset()。

mem_realloc( ) 函数

Opt.h 文件的宏 MEM_SIZE 是表示初始内存堆的大小。

动态内存池分配

说实话，我也不知道这个说法对不对，反正从源代码里面看，前者叫 heap，后者叫 pool。欢迎专家指正。

需要启用宏 MEM_USE_POOLS， 动态内存分配方式只能在内存池与内存堆中二选一。他们对外部的接口都是一样，只不过内部工作原理不太一样。

（….）

动态内存池管理

LwIP 为内部的一些结构设计了专用的内存池，比如 netconn，协议控制块，数据包等，这些都是在 memp.c/memp.h 里用内存池进行管理的。

这个模块里面 LwIP 把 C 语言的宏用到了极致，它大量采用了 C 语言的宏特性，设计上面也非常的精妙，看上去也很优雅，不过对于初学者来说猛的看上去很头大，下面就且听我给你介绍，我们先看几个静态变量数组：

memp_memory[]

这是内存池容器，他的大小由编译期决定，他是各个组件的结构用量的累加。

我们来看代码：

```
145 /** This is the actual memory used by the pools. */
146 static u8_t memp_memory[MEM_ALIGNMENT - 1
147 #define LWIP_MEMPOOL(name,num,size,desc) + ( (num) * (MEMP_SIZE + MEMP_ALIGN_SIZE(size) ) )
148 #include "lwip/memp_std.h"
149 ];
```

这是内存池的具体定义，通过 147 行，我们可以看出内存池的大小由各个组件的 num*size 的累加。如下图所示，每个组件的 num 就是下列阴影区的宏定义(在 memp_std.h 文件)

```
33 #if LWIP_RAW
34 LWIP_MEMPOOL(RAW_PCB,     MEMP_NUM_RAW_PCB,     sizeof(struct raw_pcb),     "RAW_PCB")
35 #endif /* LWIP_RAW */
36
37 #if LWIP_UDP
38 LWIP_MEMPOOL(UDP_PCB,     MEMP_NUM_UDP_PCB,     sizeof(struct udp_pcb),     "UDP_PCB")
39 #endif /* LWIP_UDP */
```

size 的大小就是各个结构的大小，如下图阴影区所示。

```
33 #if LWIP_RAW
34 LWIP_MEMPOOL(RAW_PCB,     MEMP_NUM_RAW_PCB,     sizeof(struct raw_pcb),     "RAW_PCB")
35 #endif /* LWIP_RAW */
36
37 #if LWIP_UDP
38 LWIP_MEMPOOL(UDP_PCB,     MEMP_NUM_UDP_PCB,     sizeof(struct udp_pcb),     "UDP_PCB")
39 #endif /* LWIP_UDP */
```

整个内存池的大小又可以根据组件的需要而调整，比如，如果你不需要 UDP，那么只要把 LWIP_UDP 定义为 0。用宏定义来实现用起来方便，改起来容易，就是看起来头大。

memp_num

这个静态数组用于保存各个组件的成员数目，与 memp_memory 类似也是用宏实现的。

memp_sizes

这个静态数组用于保存各个组件的结构大小，与 memp_memory 类似也是用宏实现的。

memp_init

内存池的初始化，主要是为每种内存池建立链表 memp_tab，其链表是逆序的，此外，如果有统计功能使能的话，也把记录了各种内存池的数目。

memp_malloc

如果相应的 memp_tab 链表还有空闲的节点，则从中切出一个节点返回，否则返回空。

memp_free

把释放的节点添加到相应的链表 memp_tab 头上。

pbuf

pbuf 是 lwIP 包的内部表示，被设计为最小化栈的特殊需要。pbufs 类似于 BSD 实现中的 mbufs。pbuf 结构支持为包内容动态分配内存和让包数据驻留在静态内存中。pbufs 能被一个称为 pbuf 链的链接到一个链表中，以至一个包能跨越多个 pbufs。

pbufs 有三种类型:PBUF_RAM,PBUF_ROM 和 PBUF_POOL。