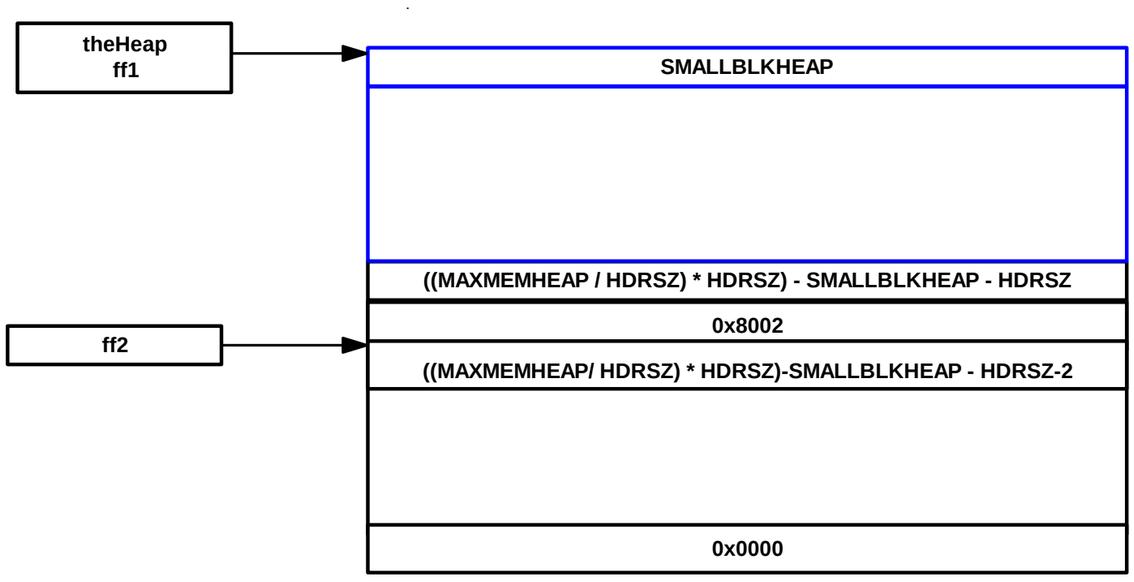
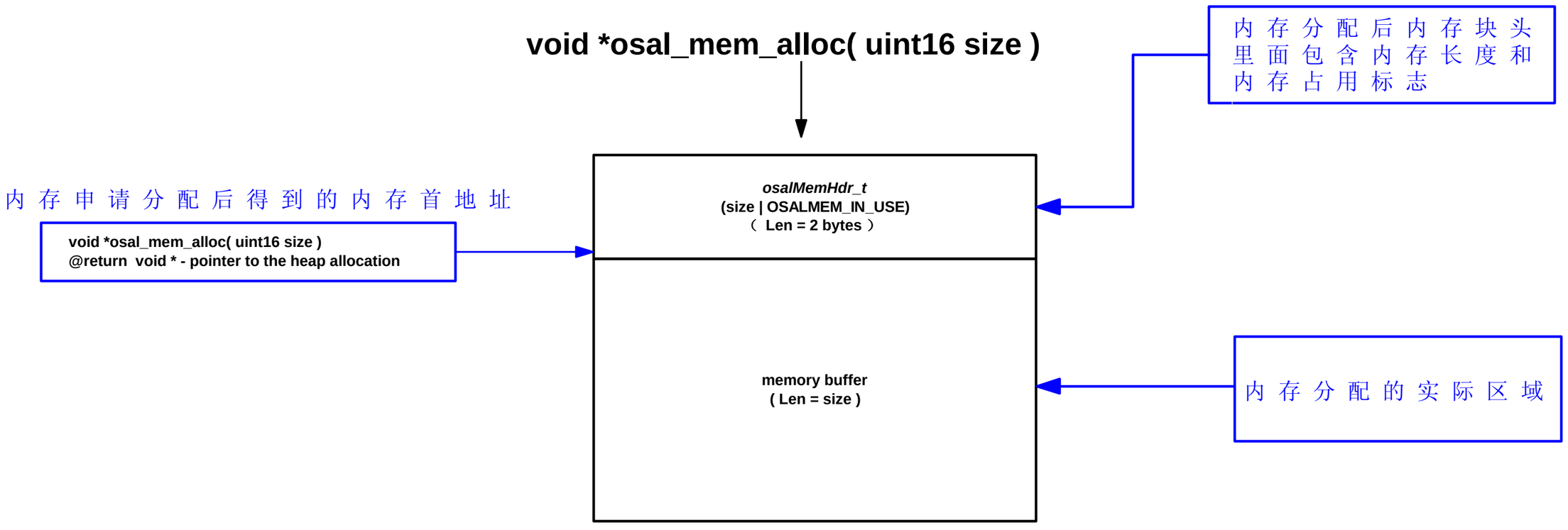
OSAL层 学 习 研 究

了 解 TI Zigbee OSAL 首 先 要 了 解 OSAL层 是 如 何 使 用 内 存 的 ， 内 存 是 根 据 使 用 量 大 概 估 算 定 义 的 一 个 数 组 。 详 细 描 述 如 下 ：
首 先 ， 该 数 据 块 尺 寸 必 须 小 于 32768字 节 。




```
typedef uint8   halDataAlign_t;

typedef uint16  osalMemHdr_t;

#define SMALLBLKHEAP    232

#define HDRSZ  ( (sizeof ( halDataAlign_t ) > sizeof( osalMemHdr_t )) ? \
                 sizeof ( halDataAlign_t ) : sizeof( osalMemHdr_t ) )

static halDataAlign_t _theHeap[ MAXMEMHEAP / sizeof( halDataAlign_t ) ];

static byte *theHeap = (byte *)_theHeap;
```

```
void osal_mem_init( void )
{
  osalMemHdr_t *tmp;

#if ( OSALMEM_PROFILER )
  osal_memset( theHeap, OSALMEM_INIT, MAXMEMHEAP );
#endif

  // Setup a NULL block at the end of the heap for fast comparisons with zero.
  tmp = (osalMemHdr_t *)theHeap + (MAXMEMHEAP / HDRSZ) - 1;
  *tmp = 0;

  // Setup a small-block bucket.
  tmp = (osalMemHdr_t *)theHeap;
  *tmp = SMALLBLKHEAP;

  // Setup the wilderness.
  tmp = (osalMemHdr_t *)theHeap + (SMALLBLKHEAP / HDRSZ);
  *tmp = ((MAXMEMHEAP / HDRSZ) * HDRSZ) - SMALLBLKHEAP - HDRSZ;

#if ( OSALMEM_GUARD )
  ready = OSALMEM_READY;
#endif

  // Setup a NULL block that is never freed so that the small-block bucket
  // is never coalesced with the wilderness.
  ff1 = tmp;
  ff2 = osal_mem_alloc( 0 );
  ff1 = (osalMemHdr_t *)theHeap;

#if ( OSALMEM_METRICS )
  /* Start with the small-block bucket and the wilderness - don't count the
   * end-of-heap NULL block nor the end-of-small-block NULL block.
   */
  blkCnt = blkFree = 2;
#endif
}
```



OSAl 是 一 个 死 循 环 查 询 ， 查 询 Task List ，
当 一 个 任 务 控 制 块 中 event 不 为 0 的 时 候
认 为 该 任 务 就 绪 ， 可 以 运 行 了

任 务 控 制 块 数 据 结 构

osalTaskRec_t

struct osalTaskRec *next;
pTaskInitFn pfnInit;
pTaskEventHandlerFn pfnEventProcessor;
byte taskID;
byte taskPriority;
uint16 events;

```
typedef void (*pTaskInitFn)( unsigned char task_id );

typedef unsigned short (*pTaskEventHandlerFn)( unsigned char task_id, unsigned short event );

typedef struct osalTaskRec
{
  struct osalTaskRec  *next;
  pTaskInitFn         pfnInit;                  //任 务 初 始 化 函 数
  pTaskEventHandlerFn  pfnEventProcessor;  //任 务 事 件 函 数
  byte                taskID;
  byte                taskPriority;
  uint16               events;

} osalTaskRec_t;
```

void osalTaskInit(void)

任 务 控 制 部 分 用 到 的 3个 全 局 变 量

osalTaskRec_t *tasksHead;	osalTaskRec_t *activeTask;	byte taskIDs;
NULL	NULL	0

```
void osal_start_system( void )
{
  uint16 events;
  uint16 retEvents;
  byte activity;
  halIntState_t intState;

  // Forever Loop
#if !defined ( ZBIT )
  for(;;)
#endif
  {

    /* This replaces MT_SerialPoll() and osal_check_timer() */
    Hal_ProcessPoll();

    activity = false;

    activeTask = osalNextActiveTask();

    if ( activeTask )
    {
      HAL_ENTER_CRITICAL_SECTION(intState);
      events = activeTask->events;
      // Clear the Events for this task
      activeTask->events = 0;
      HAL_EXIT_CRITICAL_SECTION(intState);

      if ( events != 0 )
      {
        // Call the task to process the event(s)
        if ( activeTask->pfnEventProcessor )
        {
          retEvents = (activeTask->pfnEventProcessor)( activeTask->taskID, events );

          // Add back unprocessed events to the current task
          HAL_ENTER_CRITICAL_SECTION(intState);
          activeTask->events |= retEvents;
          HAL_EXIT_CRITICAL_SECTION(intState);

          activity = true;
        }
      }
    }

    // Complete pass through all task events with no activity?
    if ( activity == false )
    {
#if defined( POWER_SAVING )
      // Put the processor/system into sleep
      osal_pwrmgr_powerconserve();
#endif
    }
  }
}
```

Task List

tasksHead

任 务 列 表 是 按 任 务 优 先 级 排 列 的 ， TaskPriority越 大 优 先 级 越 高 ， 任 务 控 制 块 越 靠 前 。

struct osalTaskRec *next;
pTaskInitFn pfnInit;
pTaskEventHandlerFn pfnEventProcessor;
byte taskID;
byte taskPriority;
uint16 events;

osalTaskRec_t

struct osalTaskRec *next;
pTaskInitFn pfnInit;
pTaskEventHandlerFn pfnEventProcessor;
byte taskID;
byte taskPriority;
uint16 events;

. . .

struct osalTaskRec *next;
pTaskInitFn pfnInit;
pTaskEventHandlerFn pfnEventProcessor;
byte taskID;
byte taskPriority;
uint16 events;

NULL

```
void osalTaskAdd( pTaskInitFn pfnInit,
          pTaskEventHandlerFn pfnEventProcessor,
          byte taskPriority)
{
  osalTaskRec_t *newTask;
  osalTaskRec_t *srchTask;
  osalTaskRec_t **ptr;

  newTask = osal_mem_alloc( sizeof( osalTaskRec_t ) );
  if ( newTask )
  {
    // Fill in new task
    newTask->pfnInit          = pfnInit;
    newTask->pfnEventProcessor = pfnEventProcessor;
    newTask->taskID           = taskIDs++;
    newTask->taskPriority     = taskPriority;
    newTask->events           = 0;
    newTask->next             = (osalTaskRec_t *)NULL;

    // 'ptr' is the address of the pointer to the new task when the new task is
    // inserted. Initially it is set to address of 'tasksHead' in case the new
    // task is higher priority than the existing head or the queue is empty.
    ptr     = &tasksHead;
    srchTask = tasksHead;
    while (srchTask)  {
      if (newTask->taskPriority > srchTask->taskPriority)  {
        // insert here. New task has a higher priority than the task
        // with which is being compared and a lower or equal priority
        // to any task that precedes it.
        newTask->next = srchTask;
        *ptr         = newTask;
        return;
      }
      // set 'ptr' to address of the pointer to 'next' in the current
      // (soon to be previous) task control block
      ptr     = &srchTask->next;
      srchTask = srchTask->next;
    }

    // We're at the end of the current queue. New task is not higher
    // priority than any other already in the list. Make it the tail.
    // (It is also the head if the queue was initially empty.)
    *ptr = newTask;
  }
  return;
}
```

```
osalTaskRec_t *osalNextActiveTask( void )
{
  osalTaskRec_t *srchTask;

  // Start at the beginning
  srchTask = tasksHead;

  // When found or not
  while ( srchTask )  {
    if (srchTask->events)  {
      // task is highest priority that is ready
      return srchTask;
    }
    srchTask = srchTask->next;
  }
  return NULL;
}
```

消 息 头 数 据 结 构

osal_msg_hdr_t

void *next;
uint16 len;
byte dest_id;

```
typedef struct
{
  void   *next;
  uint16 len;
  byte   dest_id;
} osal_msg_hdr_t;
```

事 件 头 数 据 结 构

osal_event_hdr_t

uint8 event;
uint8 status;

```
typedef struct
{
  uint8  event;
  uint8  status;
} osal_event_hdr_t;
```

typedef void * osal_msg_q_t

osal_msg_q_t osal_qHead;

消 息 队 列 头 指 针

```
byte osal_init_system( void )
{
  ...

  // Initialize the message queue
  osal_qHead = NULL;//NULL初 始 化

  ...
}
```

消 息 内 存 空 间 分 配 申 请 函 数

```
byte * osal_msg_allocate( uint16 len )
{
  osal_msg_hdr_t *hdr;

  if ( len == 0 )
    return ( NULL );

  hdr = (osal_msg_hdr_t *) osal_mem_alloc( (short)(len + sizeof( osal_msg_hdr_t )) );
  if ( hdr )
  {
    hdr->next = NULL;
    hdr->len = len;
    hdr->dest_id = TASK_NO_TASK;

#if defined( OSAL_TOTAL_MEM )
    osal_msg_cnt++;
#endif
    return ( (byte *) (hdr + 1) );
  }
  else
    return ( NULL );
}
```

执 行 消 息 分 配 后
得 到 的 消 息 首 地 址

byte * osal_msg_allocate(uint16 len)

↓

message Buffer

NULL
len
TASK_NO_TASK
message payload(Len = n)

消 息 传 递 流 程 图

注 :SYS_EVENT_MSG 是 系 统 占 用 的 一 个 事 件 标 志 位

```
#define SYS_EVENT_MSG           0x8000  // A message is waiting event
```