2018/4/15

|  |
| --- |
| 1. 上午把程序移植后始终不正确，直到晚上8点调通。 |
| 1. 排查配置项，经过对比DMA1，DMA1\_Stream寄存器发现不一样的地方修正为一样。 2. 后又对比发现波形不正确，后发现IO配置有问题：（1）速度选的太高（2）应该下拉 |
| GPIOI |
| GPIOI-2 |
| 3）以上解决发现能够正常收一个字节 |
| 4）有经过排查，发现源程序还有问题。 |
| 5）后示波器捕捉发现读取始终没有后面数据 |
| 6）经过一段时间思索。突然发现读写应该如何区分？ |
| 1. 后比对程序，发现读写不正确。   #else  uint8 spi\_data[3];  spi\_data[0] = (addrbsb & 0x00FF0000) >> 16;  spi\_data[1] = (addrbsb & 0x0000FF00) >> 8;  spi\_data[2] = (addrbsb & 0x000000F8) + 4;  spi\_dma\_write(spi\_data, buf, len);  #endif  IINCHIP\_ISR\_ENABLE(); // Interrupt Service Routine Enable |
| uint8 spi\_data[3];  spi\_data[0] = (addrbsb & 0x00FF0000) >> 16;  spi\_data[1] = (addrbsb & 0x0000FF00) >> 8;  spi\_data[2] = (addrbsb & 0x000000F8);  spi\_dma\_read(spi\_data, &data, 1); |
| 8）系两种程序，机械复制导致错误！ |

2018/4/14

|  |
| --- |
| 1. 抓包工具——高性能hub，交换机   2.底层组网公祖 |
| 3.目前工业用型号 |
| 4. |

2018/4/13

|  |
| --- |
| 1.轮询可以，设置中断不正常。 |
| 2.优化对比 |
| OP2 |
| OP0 |
|  |
|  |
|  |
| 两块板子均是这样 |
| 可以进图中断的板子 |
| 修改后 |
|  |
|  |
|  |
| 3.测试记录 |
| 交换机：UDP 100ms 一次 测试方式：内部定时器us， 使用示波器校准 |
| |  |  | | --- | --- | |  |  | | 100 | 658 | | 200 | 1048 | | 300 | 1430 | | 400 | 1820 | | 500 | 2200 | | 600 | 2587 | | 700 | 2974 | | 800 | 3364 | | 更改测试位置 | | | 800 | 1847 | | 1 | 179 | |
| 100字节    首次为 702us |
| 200字节 |
| 300字节 ，首次为1480us |
| 400字节 首次1871 |
| 500bytes |
| rt\_thread\_delay(100);  len = send\_len;  StartTimer();  INDICATE\_LED\_OFF();  sendto(0, dest\_buff, len, dest\_ip, remote\_port);    result = rt\_sem\_take(&w5500\_int\_sem, RT\_WAITING\_FOREVER);  if(getSn\_IR(0) & Sn\_IR\_RECV)  {  setSn\_IR(0, Sn\_IR\_RECV); // Sn\_IRµÄRECVÎ»ÖÃ1  }    // Êý¾Ý»Ø»·²âÊÔ³ÌÐò£ºÊý¾Ý´ÓÔ¶³ÌÉÏÎ»»ú·¢¸øW5500£¬W5500½ÓÊÕµ½Êý¾ÝºóÔÙ»Ø¸øÔ¶³ÌÉÏÎ»»ú  if((len=getSn\_RX\_RSR(0))>0)  {  memset(recv\_buff, 0, len+1);  recvfrom(0,recv\_buff, len, remote\_ip,&remote\_port); // W5500½ÓÊÕÀ´×ÔÔ¶³ÌÉÏÎ»»úµÄÊý¾Ý£¬²¢Í¨¹ýSPI·¢ËÍ¸øMCU  StopTimer();  INDICATE\_LED\_ON();  result = data\_check(len);  if (!result)  {  rt\_kprintf("T1, %d\r\n", TimeCn);  }  else  {  rt\_kprintf("ERROR, %d\r\n", result);  }  } |

2018/4/12

|  |
| --- |
| 1.花费大量时间 |
| 花费大量时间1个多小时，问题在于更新库版本不彻底！ |
|  |
| 1. ..\HALLIB\STM32F4xx\_HAL\_Driver\Inc\stm32f4xx\_ll\_exti.h(287): error: #134: expected a field name   由于重名所致，更改引用顺序即可 |
| 3.采用LL驱动正常 |

2018/4/11

|  |
| --- |
| 1.开始形成本文档 |
|  |
| |  | | --- | | 1.调试口对应 面对端子从左到右 | | 1. 调试口 U5- PC12/PD2 2. U4- PC10 tx/PC11 rx 3. U3- PB10 tx/PB11 rx 4. U1-PA9 tx/PA10 rx   5) HMI U6-PC7 rx/PC6 tx | |  | | 3. #if defined的使用  #if后面接的是一个宏。  #if defined (x)  ...code...  #endif  这个#if defined它不管里面的“x”的逻辑是“真”还是“假”它只管这个程序的前面的宏定义里面有没有定义“x”这个宏，如果定义了x这个宏，那么，编译器会编译中间的…code…否则不直接忽视中间的…code…代码。  另外 #if defined(x)也可以取反，也就用 #if !defined(x) | |  | |

|  |
| --- |
| **RTT 启动过程分析** |
| 1.startup\_stm32f429xx.s |
| IMPORT SystemInit  IMPORT \_\_main  LDR R0, =SystemInit  BLX R0  LDR R0, =\_\_main  BX R0  ENDP |
| 1. int $Sub$$main(void) |
| 为了在进入主应用程序之前，完成系统初始化，可以使用$sub和$super函数标识符在进入主程序之前插入一个例程。这一机制可以在不改变源代码的情况下扩展函数的功能。 |
| int $Sub$$main(void)  {  rt\_hw\_interrupt\_disable();  rtthread\_startup();  return 0;  } |
| 3. rtthread\_startup();s |
| int rtthread\_startup(void)  {  rt\_hw\_interrupt\_disable();  /\* board level initalization  \* NOTE: please initialize heap inside board initialization.  \*/  rt\_hw\_board\_init();  /\* show RT-Thread version \*/  rt\_show\_version();  /\* timer system initialization \*/  rt\_system\_timer\_init();  /\* scheduler system initialization \*/  rt\_system\_scheduler\_init();  /\* create init\_thread \*/  rt\_application\_init();  /\* timer thread initialization \*/  rt\_system\_timer\_thread\_init();  /\* idle thread initialization \*/  rt\_thread\_idle\_init();  /\* start scheduler \*/  rt\_system\_scheduler\_start();  /\* never reach here \*/  return 0;  } |
| void rt\_hw\_board\_init()  {  HAL\_Init(); //³õÊ¼»¯HAL¿â  Stm32\_Clock\_Init(360,25,2,8); //ÉèÖÃÊ±ÖÓ,180Mhz  // rtthread tick configuration  // 2. Configure rtos tick and interrupt  SysTick\_Config(SystemCoreClock / RT\_TICK\_PER\_SECOND);    //´®¿Ú³õÊ¼»¯  uart\_init(115200);    delay\_init(180);  //³õÊ¼»¯LED  LED\_Init();  OSRunning=1;  /\* Call components board initial (use INIT\_BOARD\_EXPORT()) \*/  #ifdef RT\_USING\_COMPONENTS\_INIT  rt\_components\_board\_init();  #endif    #if defined(RT\_USING\_CONSOLE) && defined(RT\_USING\_DEVICE)  rt\_console\_set\_device(RT\_CONSOLE\_DEVICE\_NAME);  #endif    #if defined(RT\_USING\_USER\_MAIN) && defined(RT\_USING\_HEAP)  rt\_system\_heap\_init((void\*)HEAP\_BEGIN, (void\*)SRAM\_END);  #endif  } |
| void rt\_show\_version(void)  {  rt\_kprintf("\n \\ | /\n");  rt\_kprintf("- RT - Thread Operating System\n");  rt\_kprintf(" / | \\ %d.%d.%d build %s\n",  RT\_VERSION, RT\_SUBVERSION, RT\_REVISION, \_\_DATE\_\_);  rt\_kprintf(" 2006 - 2017 Copyright by rt-thread team\n");  } |
| void rt\_application\_init(void)  {  rt\_thread\_t tid;  #ifdef RT\_USING\_HEAP  tid = rt\_thread\_create("main", main\_thread\_entry, RT\_NULL,  RT\_MAIN\_THREAD\_STACK\_SIZE, RT\_THREAD\_PRIORITY\_MAX / 3, 20);  RT\_ASSERT(tid != RT\_NULL);  #else  rt\_err\_t result;  tid = &main\_thread;  result = rt\_thread\_init(tid, "main", main\_thread\_entry, RT\_NULL,  main\_stack, sizeof(main\_stack), RT\_THREAD\_PRIORITY\_MAX / 3, 20);  RT\_ASSERT(result == RT\_EOK);  #endif  rt\_thread\_startup(tid);  } |
| void rt\_system\_timer\_thread\_init(void)  {  #ifdef RT\_USING\_TIMER\_SOFT  int i;  for (i = 0;  i < sizeof(rt\_soft\_timer\_list)/sizeof(rt\_soft\_timer\_list[0]);  i++)  {  rt\_list\_init(rt\_soft\_timer\_list+i);  }  /\* start software timer thread \*/  rt\_thread\_init(&timer\_thread,  "timer",  rt\_thread\_timer\_entry,  RT\_NULL,  &timer\_thread\_stack[0],  sizeof(timer\_thread\_stack),  RT\_TIMER\_THREAD\_PRIO,  10);  /\* startup \*/  rt\_thread\_startup(&timer\_thread);  #endif  } |
| void rt\_thread\_idle\_init(void)  {  /\* initialize thread \*/  rt\_thread\_init(&idle,  "tidle",  rt\_thread\_idle\_entry,  RT\_NULL,  &rt\_thread\_stack[0],  sizeof(rt\_thread\_stack),  RT\_THREAD\_PRIORITY\_MAX - 1,  32);  /\* startup \*/  rt\_thread\_startup(&idle);  }  static void rt\_thread\_idle\_entry(void \*parameter)  {  while (1)  {  #ifdef RT\_USING\_IDLE\_HOOK  if (rt\_thread\_idle\_hook != RT\_NULL)  {  rt\_thread\_idle\_hook();  }  #endif  rt\_thread\_idle\_excute();  }  } |
| 1. main.c |
| 1. int main(void) 2. { 3. // ´´½¨¾²Ì¬Ïß³Ì 4. rt\_thread\_init(&led0\_thread, //Ïß³Ì¿ØÖÆ¿é 5. "led0", //Ïß³ÌÃû×Ö£¬ÔÚshellÀïÃæ¿ÉÒÔ¿´µ½ 6. led0\_thread\_entry, //Ïß³ÌÈë¿Úº¯Êý 7. RT\_NULL, //Ïß³ÌÈë¿Úº¯Êý²ÎÊý 8. &rt\_led0\_thread\_stack[0], //Ïß³ÌÕ»ÆðÊ¼µØÖ· 9. sizeof(rt\_led0\_thread\_stack), //Ïß³ÌÕ»´óÐ¡ 10. 3, //Ïß³ÌµÄÓÅÏÈ¼¶ 11. 20); //Ïß³ÌÊ±¼äÆ¬ 13. rt\_thread\_startup(&led0\_thread); //Æô¶¯Ïß³Ìled0\_thread£¬¿ªÆôµ÷¶È 15. // ´´½¨¾²Ì¬Ïß³Ì 16. rt\_thread\_init(&led1\_thread, //Ïß³Ì¿ØÖÆ¿é 17. "led1", //Ïß³ÌÃû×Ö£¬ÔÚshellÀïÃæ¿ÉÒÔ¿´µ½ 18. led1\_thread\_entry, //Ïß³ÌÈë¿Úº¯Êý 19. RT\_NULL, //Ïß³ÌÈë¿Úº¯Êý²ÎÊý 20. &rt\_led1\_thread\_stack[0], //Ïß³ÌÕ»ÆðÊ¼µØÖ· 21. sizeof(rt\_led1\_thread\_stack), //Ïß³ÌÕ»´óÐ¡ 22. 3, //Ïß³ÌµÄÓÅÏÈ¼¶ 23. 20); 25. rt\_thread\_startup(&led1\_thread); //Æô¶¯Ïß³Ìled1\_thread£¬¿ªÆôµ÷¶È 27. } |

|  |
| --- |
| **RT\_DEBUG 分析** |
| D:\Program Files (x86)\keil523\ARM\PACK\rt-thread\rtthread\2.1.1\src\kservice.c(1285) : #ifdef RT\_DEBUG  #ifdef RT\_DEBUG  /\* RT\_ASSERT(EX)'s hook \*/  void (\*rt\_assert\_hook)(const char\* ex, const char\* func, rt\_size\_t line);  /\*\*  \* This function will set a hook function to RT\_ASSERT(EX). It will run when the expression is false.  \*  \* @param hook the hook function  \*/  void rt\_assert\_set\_hook(void (\*hook)(const char\* ex, const char\* func, rt\_size\_t line)) {  rt\_assert\_hook = hook;  }  /\*\*  \* The RT\_ASSERT function.  \*  \* @param ex the assertion condition string  \* @param func the function name when assertion.  \* @param line the file line number when assertion.  \*/  void rt\_assert\_handler(const char\* ex\_string, const char\* func, rt\_size\_t line)  {  volatile char dummy = 0;  if (rt\_assert\_hook == RT\_NULL)  {  #ifdef RT\_USING\_MODULE  if (rt\_module\_self() != RT\_NULL)  {  /\* unload assertion module \*/  rt\_module\_unload(rt\_module\_self());  /\* re-schedule \*/  rt\_schedule();  }  else  #endif  {  rt\_kprintf("(%s) assertion failed at function:%s, line number:%d \n", ex\_string, func, line);  while (dummy == 0);  }  }  else  {  rt\_assert\_hook(ex\_string, func, line);  }  }  RTM\_EXPORT(rt\_assert\_handler);  #endif /\* RT\_DEBUG \*/ |
| D:\Program Files (x86)\keil523\ARM\PACK\rt-thread\rtthread\2.1.1\include\rtthread.h(533) : #ifdef RT\_DEBUG  #ifdef RT\_DEBUG  extern void (\*rt\_assert\_hook)(const char\* ex, const char\* func, rt\_size\_t line);  void rt\_assert\_set\_hook(void (\*hook)(const char\* ex, const char\* func, rt\_size\_t line));  void rt\_assert\_handler(const char\* ex, const char\* func, rt\_size\_t line);  #endif /\* RT\_DEBUG \*/ |
| D:\Program Files (x86)\keil523\ARM\PACK\rt-thread\rtthread\2.1.1\include\rtdebug.h(27) : #ifdef RT\_DEBUG  #ifndef \_\_RTDEBUG\_H\_\_  #define \_\_RTDEBUG\_H\_\_  #include <rtconfig.h>  /\* Using this macro to control all kernel debug features. \*/  #ifdef RT\_DEBUG  /\* Turn on some of these (set to non-zero) to debug kernel \*/  #ifndef RT\_DEBUG\_MEM  #define RT\_DEBUG\_MEM 0  #endif  #ifndef RT\_DEBUG\_MEMHEAP  #define RT\_DEBUG\_MEMHEAP 0  #endif  #ifndef RT\_DEBUG\_MODULE  #define RT\_DEBUG\_MODULE 0  #endif  #ifndef RT\_DEBUG\_SCHEDULER  #define RT\_DEBUG\_SCHEDULER 0  #endif  #ifndef RT\_DEBUG\_SLAB  #define RT\_DEBUG\_SLAB 0  #endif  #ifndef RT\_DEBUG\_THREAD  #define RT\_DEBUG\_THREAD 0  #endif  #ifndef RT\_DEBUG\_TIMER  #define RT\_DEBUG\_TIMER 0  #endif  #ifndef RT\_DEBUG\_IRQ  #define RT\_DEBUG\_IRQ 0  #endif  #ifndef RT\_DEBUG\_IPC  #define RT\_DEBUG\_IPC 0  #endif  #ifndef RT\_DEBUG\_INIT  #define RT\_DEBUG\_INIT 0  #endif  /\* Turn on this to enable context check \*/  #ifndef RT\_DEBUG\_CONTEXT\_CHECK  #define RT\_DEBUG\_CONTEXT\_CHECK 1  #endif  #define RT\_DEBUG\_LOG(type, message) \  do \  { \  if (type) \  rt\_kprintf message; \  } \  while (0)  #define RT\_ASSERT(EX) \  if (!(EX)) \  { \  rt\_assert\_handler(#EX, \_\_FUNCTION\_\_, \_\_LINE\_\_); \  }  /\* Macro to check current context \*/  #if RT\_DEBUG\_CONTEXT\_CHECK  #define RT\_DEBUG\_NOT\_IN\_INTERRUPT \  do \  { \  rt\_base\_t level; \  level = rt\_hw\_interrupt\_disable(); \  if (rt\_interrupt\_get\_nest() != 0) \  { \  rt\_kprintf("Function[%s] shall not used in ISR\n", \_\_FUNCTION\_\_); \  RT\_ASSERT(0) \  } \  rt\_hw\_interrupt\_enable(level); \  } \  while (0)  /\* "In thread context" means:  \* 1) the scheduler has been started  \* 2) not in interrupt context.  \*/  #define RT\_DEBUG\_IN\_THREAD\_CONTEXT \  do \  { \  rt\_base\_t level; \  level = rt\_hw\_interrupt\_disable(); \  if (rt\_thread\_self() == RT\_NULL) \  { \  rt\_kprintf("Function[%s] shall not be used before scheduler start\n", \  \_\_FUNCTION\_\_); \  RT\_ASSERT(0) \  } \  RT\_DEBUG\_NOT\_IN\_INTERRUPT; \  rt\_hw\_interrupt\_enable(level); \  } \  while (0)  #else  #define RT\_DEBUG\_NOT\_IN\_INTERRUPT  #define RT\_DEBUG\_IN\_THREAD\_CONTEXT  #endif  #else /\* RT\_DEBUG \*/  #define RT\_ASSERT(EX)  #define RT\_DEBUG\_LOG(type, message)  #define RT\_DEBUG\_NOT\_IN\_INTERRUPT  #define RT\_DEBUG\_IN\_THREAD\_CONTEXT  #endif /\* RT\_DEBUG \*/  #endif /\* \_\_RTDEBUG\_H\_\_ \*/ |
| **以 RT\_DEBUG\_MEM 为例** |
| void rt\_system\_heap\_init(void \*begin\_addr, void \*end\_addr)  {  struct heap\_mem \*mem;  rt\_uint32\_t begin\_align = RT\_ALIGN((rt\_uint32\_t)begin\_addr, RT\_ALIGN\_SIZE);  rt\_uint32\_t end\_align = RT\_ALIGN\_DOWN((rt\_uint32\_t)end\_addr, RT\_ALIGN\_SIZE);  RT\_DEBUG\_NOT\_IN\_INTERRUPT;  /\* alignment addr \*/  if ((end\_align > (2 \* SIZEOF\_STRUCT\_MEM)) &&  ((end\_align - 2 \* SIZEOF\_STRUCT\_MEM) >= begin\_align))  {  /\* calculate the aligned memory size \*/  mem\_size\_aligned = end\_align - begin\_align - 2 \* SIZEOF\_STRUCT\_MEM;  }  else  {  rt\_kprintf("mem init, error begin address 0x%x, and end address 0x%x\n",  (rt\_uint32\_t)begin\_addr, (rt\_uint32\_t)end\_addr);  return;  }  /\* point to begin address of heap \*/  heap\_ptr = (rt\_uint8\_t \*)begin\_align;  RT\_DEBUG\_LOG(RT\_DEBUG\_MEM, ("mem init, heap begin address 0x%x, size %d\n",  (rt\_uint32\_t)heap\_ptr, mem\_size\_aligned));  /\* initialize the start of the heap \*/  mem = (struct heap\_mem \*)heap\_ptr;  mem->magic = HEAP\_MAGIC;  mem->next = mem\_size\_aligned + SIZEOF\_STRUCT\_MEM;  mem->prev = 0;  mem->used = 0;  /\* initialize the end of the heap \*/  heap\_end = (struct heap\_mem \*)&heap\_ptr[mem->next];  heap\_end->magic = HEAP\_MAGIC;  heap\_end->used = 1;  heap\_end->next = mem\_size\_aligned + SIZEOF\_STRUCT\_MEM;  heap\_end->prev = mem\_size\_aligned + SIZEOF\_STRUCT\_MEM;  rt\_sem\_init(&heap\_sem, "heap", 1, RT\_IPC\_FLAG\_FIFO);  /\* initialize the lowest-free pointer to the start of the heap \*/  lfree = (struct heap\_mem \*)heap\_ptr;  } |
| void \*rt\_malloc(rt\_size\_t size)  {  rt\_size\_t ptr, ptr2;  struct heap\_mem \*mem, \*mem2;  RT\_DEBUG\_NOT\_IN\_INTERRUPT;  if (size == 0)  return RT\_NULL;  if (size != RT\_ALIGN(size, RT\_ALIGN\_SIZE))  RT\_DEBUG\_LOG(RT\_DEBUG\_MEM, ("malloc size %d, but align to %d\n",  size, RT\_ALIGN(size, RT\_ALIGN\_SIZE)));  else  RT\_DEBUG\_LOG(RT\_DEBUG\_MEM, ("malloc size %d\n", size));  /\* alignment size \*/  size = RT\_ALIGN(size, RT\_ALIGN\_SIZE);  if (size > mem\_size\_aligned)  {  RT\_DEBUG\_LOG(RT\_DEBUG\_MEM, ("no memory\n"));  return RT\_NULL;  }  /\* every data block must be at least MIN\_SIZE\_ALIGNED long \*/  if (size < MIN\_SIZE\_ALIGNED)  size = MIN\_SIZE\_ALIGNED;  /\* take memory semaphore \*/  rt\_sem\_take(&heap\_sem, RT\_WAITING\_FOREVER);  for (ptr = (rt\_uint8\_t \*)lfree - heap\_ptr;  ptr < mem\_size\_aligned - size;  ptr = ((struct heap\_mem \*)&heap\_ptr[ptr])->next)  {  mem = (struct heap\_mem \*)&heap\_ptr[ptr];  if ((!mem->used) && (mem->next - (ptr + SIZEOF\_STRUCT\_MEM)) >= size)  {  /\* mem is not used and at least perfect fit is possible:  \* mem->next - (ptr + SIZEOF\_STRUCT\_MEM) gives us the 'user data size' of mem \*/  if (mem->next - (ptr + SIZEOF\_STRUCT\_MEM) >=  (size + SIZEOF\_STRUCT\_MEM + MIN\_SIZE\_ALIGNED))  {  /\* (in addition to the above, we test if another struct heap\_mem (SIZEOF\_STRUCT\_MEM) containing  \* at least MIN\_SIZE\_ALIGNED of data also fits in the 'user data space' of 'mem')  \* -> split large block, create empty remainder,  \* remainder must be large enough to contain MIN\_SIZE\_ALIGNED data: if  \* mem->next - (ptr + (2\*SIZEOF\_STRUCT\_MEM)) == size,  \* struct heap\_mem would fit in but no data between mem2 and mem2->next  \* @todo we could leave out MIN\_SIZE\_ALIGNED. We would create an empty  \* region that couldn't hold data, but when mem->next gets freed,  \* the 2 regions would be combined, resulting in more free memory  \*/  ptr2 = ptr + SIZEOF\_STRUCT\_MEM + size;  /\* create mem2 struct \*/  mem2 = (struct heap\_mem \*)&heap\_ptr[ptr2];  mem2->magic = HEAP\_MAGIC;  mem2->used = 0;  mem2->next = mem->next;  mem2->prev = ptr;  /\* and insert it between mem and mem->next \*/  mem->next = ptr2;  mem->used = 1;  if (mem2->next != mem\_size\_aligned + SIZEOF\_STRUCT\_MEM)  {  ((struct heap\_mem \*)&heap\_ptr[mem2->next])->prev = ptr2;  }  #ifdef RT\_MEM\_STATS  used\_mem += (size + SIZEOF\_STRUCT\_MEM);  if (max\_mem < used\_mem)  max\_mem = used\_mem;  #endif  }  else  {  /\* (a mem2 struct does no fit into the user data space of mem and mem->next will always  \* be used at this point: if not we have 2 unused structs in a row, plug\_holes should have  \* take care of this).  \* -> near fit or excact fit: do not split, no mem2 creation  \* also can't move mem->next directly behind mem, since mem->next  \* will always be used at this point!  \*/  mem->used = 1;  #ifdef RT\_MEM\_STATS  used\_mem += mem->next - ((rt\_uint8\_t\*)mem - heap\_ptr);  if (max\_mem < used\_mem)  max\_mem = used\_mem;  #endif  }  /\* set memory block magic \*/  mem->magic = HEAP\_MAGIC;  if (mem == lfree)  {  /\* Find next free block after mem and update lowest free pointer \*/  while (lfree->used && lfree != heap\_end)  lfree = (struct heap\_mem \*)&heap\_ptr[lfree->next];  RT\_ASSERT(((lfree == heap\_end) || (!lfree->used)));  }  rt\_sem\_release(&heap\_sem);  RT\_ASSERT((rt\_uint32\_t)mem + SIZEOF\_STRUCT\_MEM + size <= (rt\_uint32\_t)heap\_end);  RT\_ASSERT((rt\_uint32\_t)((rt\_uint8\_t \*)mem + SIZEOF\_STRUCT\_MEM) % RT\_ALIGN\_SIZE == 0);  RT\_ASSERT((((rt\_uint32\_t)mem) & (RT\_ALIGN\_SIZE-1)) == 0);  RT\_DEBUG\_LOG(RT\_DEBUG\_MEM,  ("allocate memory at 0x%x, size: %d\n",  (rt\_uint32\_t)((rt\_uint8\_t \*)mem + SIZEOF\_STRUCT\_MEM),  (rt\_uint32\_t)(mem->next - ((rt\_uint8\_t \*)mem - heap\_ptr))));  RT\_OBJECT\_HOOK\_CALL(rt\_malloc\_hook,  (((void \*)((rt\_uint8\_t \*)mem + SIZEOF\_STRUCT\_MEM)), size));  /\* return the memory data except mem struct \*/  return (rt\_uint8\_t \*)mem + SIZEOF\_STRUCT\_MEM;  }  }  rt\_sem\_release(&heap\_sem);  return RT\_NULL;  } |
| void rt\_interrupt\_enter(void)  {  rt\_base\_t level;  RT\_DEBUG\_LOG(RT\_DEBUG\_IRQ, ("irq coming..., irq nest:%d\n",  rt\_interrupt\_nest));  level = rt\_hw\_interrupt\_disable();  rt\_interrupt\_nest ++;  RT\_OBJECT\_HOOK\_CALL(rt\_interrupt\_enter\_hook,());  rt\_hw\_interrupt\_enable(level);  }  RTM\_EXPORT(rt\_interrupt\_enter); |

|  |
| --- |
| RT\_USING\_OVERFLOW\_CHECK |
| #ifdef RT\_USING\_OVERFLOW\_CHECK  static void \_rt\_scheduler\_stack\_check(struct rt\_thread \*thread)  {  RT\_ASSERT(thread != RT\_NULL);  if (\*((rt\_uint8\_t \*)thread->stack\_addr) != '#' ||  (rt\_uint32\_t)thread->sp <= (rt\_uint32\_t)thread->stack\_addr ||  (rt\_uint32\_t)thread->sp >  (rt\_uint32\_t)thread->stack\_addr + (rt\_uint32\_t)thread->stack\_size)  {  rt\_uint32\_t level;  rt\_kprintf("thread:%s stack overflow\n", thread->name);  #ifdef RT\_USING\_FINSH  {  extern long list\_thread(void);  list\_thread();  }  #endif  level = rt\_hw\_interrupt\_disable();  while (level);  }  else if ((rt\_uint32\_t)thread->sp <= ((rt\_uint32\_t)thread->stack\_addr + 32))  {  rt\_kprintf("warning: %s stack is close to end of stack address.\n",  thread->name);  }  }  #endif |
| void rt\_schedule(void)  {  /\* switch to new thread \*/  RT\_DEBUG\_LOG(RT\_DEBUG\_SCHEDULER,  ("[%d]switch to priority#%d "  "thread:%.\*s(sp:0x%p), "  "from thread:%.\*s(sp: 0x%p)\n",  rt\_interrupt\_nest, highest\_ready\_priority,  RT\_NAME\_MAX, to\_thread->name, to\_thread->sp,  RT\_NAME\_MAX, from\_thread->name, from\_thread->sp));  #ifdef RT\_USING\_OVERFLOW\_CHECK  \_rt\_scheduler\_stack\_check(to\_thread);  #endif  }  /\* enable interrupt \*/  rt\_hw\_interrupt\_enable(level);  } |
|  |