2018/9/12

|  |
| --- |
| 1.直接丢失 |
| 直接造成0x10 stNum丢失 |
| 试验5次，发现13,16过流，17只有少数执行隔离，其它时候不执行，极有可能是数据丢失所致。先优化接收。 |
| 1. 有时发送也丢失 |

2018/9/11

|  |
| --- |
| 1.死机复现，需要控制全局列表的生成，一旦生成，除非特殊情况，禁止更新。 |
|  |
| 2.死机 |
| psr: 0x81000000  pc: 0x0809f066(rt\_malloc)  lr: 0x080a0c4d(rt\_sem\_take)  r12: 0xc06e9fa4  r03: 0x00000024  r02: 0xc1effff4  r01: 0xc0600000  r00: 0xd9e71987  hard fault on thread: erx  thread pri status sp stack size max used left tick error  ------- --- ------- ---------- ---------- ------ ---------- ---  goRe 5 ready 0x00000088 0x00000800 16% 0x0000000a 000  ieds 10 ready 0x0000010c 0x000007ec 19% 0x00000001 -02  distpro 4 ready 0x00000064 0x00000400 52% 0x00000002 -02  SIMSW 2 suspend 0x00000064 0x00000400 17% 0x00000013 000  udpser 2 suspend 0x000000bc 0x00000c00 53% 0x00000008 000  tshell 20 suspend 0x000000e4 0x00001000 05% 0x00000008 000  61850 14 suspend 0x00000074 0x00002800 66% 0x00000003 000  idle 28 ready 0x0000005c 0x00001000 11% 0x0000000f -02  iec104 23 ready 0x00000104 0x00001000 06% 0x00000012 000  iec101 24 ready 0x00000104 0x00001000 07% 0x00000013 000  watch 18 ready 0x000000e4 0x00001f40 03% 0x00000003 000 |
|  |
| pc: 0x08073dc6（SerachPowerConditionList）  lr: 0x0809bb7f（rt\_malloc）  r12: 0x00000000  r03: 0xc0924b4c  r02: 0x00000000  r01: 0xc0924b28  r00: 0x00000008  hard fault on thread: connect  thread pri status sp stack size max used left tick error  ------- --- ------- ---------- ---------- ------ ---------- ---  goRe 5 suspend 0x000000b0 0x00000800 18% 0x00000012 000  ieds 10 suspend 0x00000110 0x000007ec 19% 0x0000000d 000  connect 15 ready 0x0000018c 0x00000400 46% 0x00000002 000  distpro 4 suspend 0x00000064 0x00000400 28% 0x00000012 000  SIMSW 2 suspend 0x00000064 0x00000400 17% 0x00000014 000  udpser 2 suspend 0x0000006c 0x00000c00 54% 0x00000004 000  tshell 20 suspend 0x000000e4 0x00001000 10% 0x0000000a 000  61850 14 suspend 0x00000074 0x00002800 66% 0x00000004 000  idle 28 ready 0x0000005c 0x00001000 11% 0x0000000e -02  iec104 23 ready 0x00000104 0x00001000 08% 0x00000005 000  iec101 24 ready 0x00000104 0x00001000 08% 0x00000005 000  watch 18 ready 0x000000e4 0x00001f40 03% 0x0000000d 000  protect 9 suspend 0x000000e4 0x00001000 07% 0x0000000d 000  cal 8 suspend 0x00000104 0x00001000 11% 0x00000002 000  tcpip 12 suspend 0x000000a4 0x00000400 33% 0x00000003 000  etx 11 suspend 0x00000078 0x00000400 11% 0x00000010 000  erx 3 suspend 0x00000078 0x00000400 21% 0x0000000e 000  tidle 31 ready 0x00000048 0x00000400 07% 0x00000010 000  timer 0 suspend 0x00000060 0x00000200 71% 0x00000009 000 |
| 3.合并在一起会超时 |
| 开机 |
|  |
| pc: 0x0804679e  lr: 0x0804678b（BreadthFirstPath）  r12: 0x0000000b  r03: 0x2002bddc  r02: 0x2002bdd8  r01: 0xc06ea088  r00: 0x05050505  hard fault on thread: connect  thread pri status sp stack size max used left tick error  ------- --- ------- ---------- ---------- ------ ---------- ---  goRe 5 suspend 0x000000ac 0x00000800 17% 0x00000010 000  ieds 10 suspend 0x00000110 0x000007ec 19% 0x00000002 000  connect 15 ready 0x00000184 0x00000400 52% 0x00000005 000  distpro 4 suspend 0x00000064 0x00000400 28% 0x0000000b 000  SIMSW 2 suspend 0x00000064 0x00000400 17% 0x00000014 000  udpser 2 suspend 0x0000006c 0x00000c00 53% 0x00000009 000  tshell 20 suspend 0x000000e4 0x00001000 05% 0x00000006 000  61850 14 suspend 0x00000074 0x00002800 66% 0x00000013 000  idle 28 suspend 0x0000005c 0x00001000 11% 0x00000013 000  iec104 23 ready 0x00000104 0x00001000 08% 0x0000000f 000  iec101 24 ready 0x00000104 0x00001000 07% 0x00000010 000  watch 18 ready 0x000000dc 0x00001f40 03% 0x0000000e 000  protect 9 suspend 0x000000e4 0x00001000 07% 0x00000002 000  cal 8 suspend 0x00000104 0x00001000 11% 0x00000004 000  tcpip 12 suspend 0x000000a4 0x00000400 33% 0x00000008 000  etx 11 suspend 0x00000078 0x00000400 11% 0x0000000f 000  erx 3 suspend 0x00000078 0x00000400 20% 0x00000006 000  tidle 31 ready 0x00000048 0x00000400 07% 0x0000000c 000  timer 0 suspend 0x00000060 0x00000200 71% 0x00000009 000 |
| ListInit(ss->connect.path + 1, FREE);此处为null |
| 1. 数据设置和使用，需要留意使用时不能修改的数据。 2. 所有链表，尤其是global链表一旦形成不能修改删除。 3. snap操作， |

2018/9/10

|  |
| --- |
| 1. |
|  |
| 造成为0的可能性分析。  为0，查找list不为空，而且出现cp->hopsNumber <=1 的现象。  情况1：生成list时就已经，插入空  情况2：在读取时恰好删除列表。——如果不复归不应该出现此现象。 |
| MEMCPY(cp, list\_data(m\_foreach), sizeof(ConnectPath));  [修改目的和源错误  新建数据结构和函数一定要仔细判别！仔细验证，不可侥幸！ |
| 2.connectPath |
| connectPath: id:0xc0a80a08, hops:1, isUpdated:0x0, cap:0, switchNum:4. |
| cp->switchNum = sw->distributionArea->powerArea[1].switchNum; |
| 系power索引理解错误所致。  本质上是大意，没有充分测试快照里面的值。 |
| 3. 8号为什么出现多次死机 |
| 4.   1. 构思合并connect任务与分布式任务，减少同步复杂性。 2. 构思合并W5500中断接收与任务处理 3. Connect任务不执行完成意味着分布式保护不能运行，其是串行前后关系。 |
|  |
| 5.错误，进行一次故障切除复位后 |
| 出现以后可以获取但是其灯不再正常闪烁了！  出现以下错误！ |

2018/9/6

|  |
| --- |
| 1.目前交互信息 |
| 1. 更新路径信息需要与分布式保护相互配合，时序上可能出现先更新联络信息后进行保护保护判断的情况，造成遗漏。！一旦检测到故障信息或者接收到故障信息此时，应该挂起判断线程，直到退出是重新启动！   以上方法不具有灵活性，引入快照结构，触发后保存判断是采用快照信息，按照需要逐步扩展。 |
| 什么样的信息需要快照保存  1.connectPath与connect信息 |
|  |

2019/9/3

|  |
| --- |
| 1.一直识别不了发现头文件有重名！！！！！与dfs里面的重名 |
| ..\LibIEC\libIEC61850\src\mms\iso\_mms\asn1c |
| 2. RT\_USING\_MEMHEAP\_AS\_HEAP |

2018/9/2

|  |
| --- |
| 1.4,5优先级 |
| 1）53/21   1. (27+47) / (1+34) |
| 2.调整优先级 |
| 1. /16ms 2. （1+56）/(1+24） 3. /(14+26) 4. /(8+18) 5. /(8+30) 6. /(1+16) 7. /(7+31) 8. /(8+26) 9. (0+31)/ 10. /(8+21) |
| 3.取消2248节拍 |
| 1. （1+34）/（1+2） 2. /(0+9) 3. /(8+6) 4. /(8+4) 5. /(8+18) 6. /(1+15) 7. /(1+20） 8. /(1+21) 9. /(8+13) 10. /(0+21) |
| 4.单测试1，过流 |
| 1）0+15  2）0+9  3）1+9  4)0+15  5)1+15 |
| 5. 50ms时限，恢复其它 |
| 1. /(0+41) 2. /(1+40) 3. /(8+12) 4. /(7+22) 5. /(1+40) 6. /(7+24) 7. /(0+30) 8. /(8+21) 9. /(0+30) 10. /(0+30) |
| 5.单纯50ms，取消收集完整判断 |
| 1）50  2）58  3）44  4）7+50  5）7+50  6）1+50  7）1+50  8）1+50  9）8+50  10）7+50 |

2018/9/1

|  |
| --- |
| 1.为什么节点3一直有获取问题！！Arp请求 |
|  |
|  |

2018/8/31

|  |
| --- |
| 1.打印输出 |
| 需要使用参考开关。switchRef |
| 2.对于故障信号，直到复归，否则不能清除。 |

2018/8/30

|  |
| --- |
| 1.整治打印以利于输出 |
| 2.错误输出.  rt\_mb\_send(MacRawReciveMb, (rt\_uint32\_t)(pPacket)), error: -3. |
| 1. 解决复归问题。复归需要同时复归。   复归方式：收到复归命令以后，屏蔽接收3s以后，再重新打开接收。 |
|  |

2018/8/29

|  |
| --- |
| 1.注意不是一个网段，通过交换机不能直连，需要修改掩码。 |
| 2.屏蔽空闲文件操作，避免被打断，造成错误。 |
|  |
|  |

2018/8/28

|  |
| --- |
| 1. |
| >tftp\_get("192.168.10.111","//sojo", "stu-goose.txt" ) |
| 2. 8口交换机，有时候不能传播goose数据。 |
| 3.弄了1个小时，没调通VLAN |
| 4.  #if 0  /\* Priority tag - IEEE 802.1Q \*/  self->buffer[bufPos++] = 0x81;  self->buffer[bufPos++] = 0x00; |

2018/8/27

|  |
| --- |
| 1.sram占用 |
| |  |  |  |  | | --- | --- | --- | --- | | point\_table\_config | 4900 | 19K | .data | | point\_table\_config | 1a78 | 6.7K | .bss | | multi\_thread | 31 |  | .data | | multi\_thread | 32e8 | 13K | .bss | | common\_data | 6634 | 26k | .bss | | calculator | 1fb0 | 8K | .bss | | drv\_eth | 3194 | 12.6K | .bss | | memp | 6c7f | 27.7K | .bss | |  |  |  |  | |
| FixedValueCfg1 0x20002fec Data 6776 point\_table\_config.o(.data) |
| 2.为了减少内存占用做如下修改  1）#define RT\_LWIP\_PBUF\_NUM 8 |
| tftp\_get("192.168.10.111", "//sojo", "stu.cfg") |
| 1. 内存分错误 2. ic->indexTrans = (uint16\_t\*)GLOBAL\_CALLOC(sizeof(uint8\_t), daCount);//转换索引数组   不匹配导致一系列错误！Hard\_fault |

2018/8/25

|  |
| --- |
| 1.添加前 |
| Total RO Size (Code + RO Data) 528052 ( 515.68kB)  Total RW Size (RW Data + ZI Data) 196328 ( 191.73kB)  Total ROM Size (Code + RO Data + RW Data) 543524 ( 530.79kB) |
| #ifdef IED\_DATA\_ENABLE\_OTHER |
| Ref文件，显然过大放弃！  一个逻辑节点如GGIO17 涉及467个逻辑变量（16个数字输入，8个alm，就24个数字输入） |
| 2.先更新站点信息，再更新goose |

2018/8/24

|  |
| --- |
| 1.修改原生的goose发布机制。 |
| Goose变化机制分析 |
| IedServer\_updateBooleanAttributeValue（Update）  -->  checkForChangedTriggers(self, dataAttribute);  --> |
| static inline void  checkForChangedTriggers(IedServer self, DataAttribute\* dataAttribute)  {  #if (CONFIG\_IEC61850\_REPORT\_SERVICE == 1) || (CONFIG\_INCLUDE\_GOOSE\_SUPPORT == 1)  if (dataAttribute->triggerOptions & TRG\_OPT\_DATA\_CHANGED) {  #if (CONFIG\_INCLUDE\_GOOSE\_SUPPORT == 1)  MmsMapping\_triggerGooseObservers(self->mmsMapping, dataAttribute->mmsValue);  #endif  #if (CONFIG\_IEC61850\_REPORT\_SERVICE == 1)  MmsMapping\_triggerReportObservers(self->mmsMapping, dataAttribute->mmsValue,  REPORT\_CONTROL\_VALUE\_CHANGED);  #endif  #if (CONFIG\_IEC61850\_LOG\_SERVICE == 1)  MmsMapping\_triggerLogging(self->mmsMapping, dataAttribute->mmsValue,  LOG\_CONTROL\_VALUE\_CHANGED);  #endif  }  else if (dataAttribute->triggerOptions & TRG\_OPT\_QUALITY\_CHANGED) {  #if (CONFIG\_INCLUDE\_GOOSE\_SUPPORT == 1)  MmsMapping\_triggerGooseObservers(self->mmsMapping, dataAttribute->mmsValue);  #endif  #if (CONFIG\_IEC61850\_REPORT\_SERVICE == 1)  MmsMapping\_triggerReportObservers(self->mmsMapping, dataAttribute->mmsValue,  REPORT\_CONTROL\_QUALITY\_CHANGED);  #endif  #if (CONFIG\_IEC61850\_LOG\_SERVICE == 1)  MmsMapping\_triggerLogging(self->mmsMapping, dataAttribute->mmsValue,  LOG\_CONTROL\_QUALITY\_CHANGED);  #endif  }  #endif /\* (CONFIG\_IEC61850\_REPORT\_SERVICE== 1) || (CONFIG\_INCLUDE\_GOOSE\_SUPPORT == 1) \*/  } |
| 在上方 |
| void  MmsMapping\_triggerGooseObservers(MmsMapping\* self, MmsValue\* value)  {  LinkedList element = self->gseControls;  while ((element = LinkedList\_getNext(element)) != NULL) {  MmsGooseControlBlock gcb = (MmsGooseControlBlock) element->data;  if (MmsGooseControlBlock\_isEnabled(gcb)) {  DataSet\* dataSet = MmsGooseControlBlock\_getDataSet(gcb);    if (DataSet\_isMemberValue(dataSet, value, NULL)) {  MmsGooseControlBlock\_observedObjectChanged(gcb);  }    }  }  }  //从控制块链表中查找使能的goose控制块，并判断指定参数是否为数据集成员，若是则进入。MmsGooseControlBlock\_observedObjectChanged |
|  |
| void  MmsGooseControlBlock\_observedObjectChanged(MmsGooseControlBlock self)  {  #if (CONFIG\_MMS\_THREADLESS\_STACK != 1)  Semaphore\_wait(self->publisherMutex);  #endif    uint64\_t currentTime = GoosePublisher\_increaseStNum(self->publisher);  self->retransmissionsLeft = CONFIG\_GOOSE\_EVENT\_RETRANSMISSION\_COUNT;  if (self->retransmissionsLeft > 0) {  self->nextPublishTime = currentTime + self->minTime;  GoosePublisher\_setTimeAllowedToLive(self->publisher, self->minTime \* 3);  }  else {  self->nextPublishTime = currentTime + self->maxTime;  GoosePublisher\_setTimeAllowedToLive(self->publisher, self->maxTime \* 3);  }    GoosePublisher\_publish(self->publisher, self->dataSetValues);    #if (CONFIG\_MMS\_THREADLESS\_STACK != 1)  Semaphore\_post(self->publisherMutex);  #endif    }  以上对于数据集有变化的情况，进行了一下几点处理：   1. 增加stNum 2. 设置GoosePublisher\_setTimeAllowedToLive， 时间 3. 判断重发次数，以及下一次重发间隔。 4. 由数据集发送出去。 |
|  |
| 关于goose周期性发送时间，以及IedServer分析 |
| **#if** (CONFIG\_MMS\_SINGLE\_THREADED == 1)  **static** **void**  **singleThreadedServerThread**(**void**\* parameter)  {  IedServer self = (IedServer) parameter;  MmsMapping\* mmsMapping = self->mmsMapping;  bool running = true;  mmsMapping->reportThreadFinished = false;  mmsMapping->reportThreadRunning = true;  **if** (DEBUG\_IED\_SERVER)  **printf**("IED\_SERVER: server thread started!\n");  **while** (running) {  **if** (IedServer\_waitReady(self, 25) > 0)  MmsServer\_handleIncomingMessages(self->mmsServer);  IedServer\_performPeriodicTasks(self);  Thread\_sleep(1);  running = mmsMapping->reportThreadRunning;  }  **if** (DEBUG\_IED\_SERVER)  **printf**("IED\_SERVER: server thread finished!\n");  mmsMapping->reportThreadFinished = true;  }  次任务是一个循环时间为1ms的调度任务。主任务为IedServer\_performPeriodicTasks(self); |
| -->  processPeriodicTasks() |
| **static** **void**  **processPeriodicTasks**(MmsMapping\* self)  {  uint64\_t currentTimeInMs = Hal\_getTimeInMs();  **#if** (CONFIG\_INCLUDE\_GOOSE\_SUPPORT == 1)  GOOSE\_processGooseEvents(self, currentTimeInMs);  **#endif**  **#if** (CONFIG\_IEC61850\_CONTROL\_SERVICE == 1)  Control\_processControlActions(self, currentTimeInMs);  **#endif**  **#if** (CONFIG\_IEC61850\_REPORT\_SERVICE == 1)  Reporting\_processReportEvents(self, currentTimeInMs);  **#endif**  **#if** (CONFIG\_IEC61850\_SETTING\_GROUPS == 1)  MmsMapping\_checkForSettingGroupReservationTimeouts(self, currentTimeInMs);  **#endif**  **#if** (CONFIG\_IEC61850\_LOG\_SERVICE == 1)  Logging\_processIntegrityLogs(self, currentTimeInMs);  **#endif**  /\* handle low priority MMS backgound tasks (like file upload...) \*/  MmsServer\_handleBackgroundTasks(self->mmsServer);  } |
|  |
| **static** **void**  **GOOSE\_processGooseEvents**(MmsMapping\* self, uint64\_t currentTimeInMs)  {  LinkedList element = LinkedList\_getNext(self->gseControls);  **while** (element != NULL) {  MmsGooseControlBlock mmsGCB = (MmsGooseControlBlock) element->data;  **if** (MmsGooseControlBlock\_isEnabled(mmsGCB)) {  MmsGooseControlBlock\_checkAndPublish(mmsGCB, currentTimeInMs);  }  element = LinkedList\_getNext(element);  }  } |
|  |
| **void**  **MmsGooseControlBlock\_checkAndPublish**(MmsGooseControlBlock self, uint64\_t currentTime)  {  **if** (currentTime >= self->nextPublishTime) {  **#if** (CONFIG\_MMS\_THREADLESS\_STACK != 1)  Semaphore\_wait(self->publisherMutex);  **#endif**  GoosePublisher\_publish(self->publisher, self->dataSetValues);  **if** (self->retransmissionsLeft > 0) {  self->nextPublishTime = currentTime + self->minTime;  **if** (self->retransmissionsLeft > 1)  GoosePublisher\_setTimeAllowedToLive(self->publisher, self->minTime \* 3);  **else**  GoosePublisher\_setTimeAllowedToLive(self->publisher, self->maxTime \* 3);  self->retransmissionsLeft--;  }  **else** {  GoosePublisher\_setTimeAllowedToLive(self->publisher, self->maxTime \* 3);  self->nextPublishTime = currentTime + self->maxTime;  }  **#if** (CONFIG\_MMS\_THREADLESS\_STACK != 1)  Semaphore\_post(self->publisherMutex);  **#endif**  }  **else** **if** ((self->nextPublishTime - currentTime) > ((uint32\_t) self->maxTime \* 2)) {  self->nextPublishTime = currentTime + self->minTime;  }  }  以上任务完成如下功能   1. 判断下次发送时间是否到达，若到达进行发布 2. 设置下次的存货时间 3. 设置下次的发布时间 |
| 由以上分析可知，程序中原始的变化发布程序，将多种服务集中在一起，采用主动触发，任务轮询的方式发送。 |
| 为了提高goose的实时性，平衡任务开销，需要将goose作为单独的高优先级处理，其它服务降低优先级、轮询间隔。 |
| 具体执行步骤：  1）屏蔽触发更新服务的goose服务即MmsMapping\_triggerGooseObservers  2）取消循环任务内的周期即**GOOSE\_processGooseEvents**  **3）修改**MmsGooseControlBlock\_observedObjectChanged，使其与之前设计好的周期性发送功能融合。  4）初始化beat |

2018/8/23

|  |
| --- |
| 1.通信与逻辑各司其职，通信只用来更新数据。其它交给逻辑。 |
|  |

2018/8/22

|  |
| --- |
| 1.IAR要比MDK省内存 |
| 13:00-13:47--重新回到MDK上，在编译器上不做太多折腾了。 |
|  |
|  |

2018/8/21

|  |
| --- |
| 1.如何让MmsValue 转变为普通值。 |
| 通过控制块进行转换 |
|  |

2018/8/16

|  |
| --- |
| 1.将任务优先级抬升到tcp 4-5，不丢包了。测试20分钟。 |
| 2. Goose发布机制不能使用自带的机制，其是查询，对于保证固定节拍不利。 |
| 3.至少保证保护和计算任务。 |
| 4.单独调试 |
| 5.发送间隔 |
| for (i = 0; i < 4; i++) {  //Thread\_sleep();  if (GoosePublisher\_publish(publisher, dataSetValues) == -1) {  *printf*("Error sending message!\n");  }    }  100次没有丢  4变成8，就发生last3,current7丢包。否则没有提示。 |
| ppID: 0x1001, valid, st:82 , sq: 0!  duty time: 3923  AppID: 0x1001, valid, st:83 , sq: 0!  duty time: 3918  AppID: 0x1001, valid, st:84 , sq: 0!  duty time: 3898  AppID: 0x1001, valid, st:85 , sq: 0!  duty time: 3920  AppID: 0x1001, valid, st:86 , sq: 0!  duty time: 3915  AppID: 0x1001, valid, st:87 , sq: 0!  duty time: 3919  接收时间，改变任务优先级也没效果。  打印影响。    以上为当前任务优先级 |
| 实际接收处理时间  duty time: 783  duty time: 810  duty time: 811  接收优先级调到2， 时间不确定性加大！  有可能lwip发送任务也影响时间 |
| 不加显示，没法看结果，加显示，时间又不准。 |
| 对于单独一帧稳定在780us，就比较短小了。 |
| 1000次接收，处理正常 |
| 时间变短，应该是有所覆盖  for (int k = 0 ; k < 200; k++)  {  for (i = 0; i < 8; i++) {  //Thread\_sleep();  if (GoosePublisher\_publish(publisher, dataSetValues) == -1) {  *printf*("Error sending message!\n");  }    }  GoosePublisher\_increaseStNum(publisher);  Thread\_sleep(50);    }  接收过程是没有打印，尽在最后打印输出。 |
| 红线具有很强规律性，说明最大接收能力是突发的4帧。    重新测试结果一致。 |
| 发送加1ms延时（上位机）    最大接收是6帧，8帧丢一帧 |
| 减少包长度 1267 变成458    最大接收是5帧。 |
| 5.为甚么从1400us 降到800us！！ |
| 1. 调整优先级，以太网接收中断     这次倒是没丢，但是只到105。 105\*8 = 840，这样后面的去哪了？？？？？？？？ |
| 把发送间隔提高到100ms，则不再出现丢失现象 |
| 以上左侧是8帧连续发生  以上右侧是1帧数据  785\*8 = 6280us。 |
| 从测试结果看也可以不丢包，也可以不丢包。 |
|  |
| 7.结果 |
| 结果是-3  #define RT\_EFULL 3 /\*\*< The resource is full \*/  说明缓冲已满。100个用完了。。。 |
| 即使改成连续16个发送时间并没有明显变化。 |
| 单独将接收优先级      时间减少，但是丢包啊，似乎是优先级重复所致 |
| 修改优先级    不再掉包，似乎时间已经没有参考价值。 |
| 调整计时方式以后稳定在800us，至此可以得出结论。  接收可以保证：连续接收多帧，可以不丢失，但会推迟时间。 |
| 16帧连续发送，显然接收力不从心，有丢包现象，连续接收10帧。 |
| 短报也不出现丢失。时间也较短，能计算出来。（457字节） |
| 即使32此连续发送也不丢。 |
| for (i = 0; i < 1100; i++) {  //Thread\_sleep(50);  if (GoosePublisher\_publish(publisher, dataSetValues) == -1) {  *printf*("Error sending message!\n");  }    }    即使1000此也可以不丢失，说明这种短帧可以即时解析。这是上位机电脑的速度。 |
|  |
| 10.注意二维数组指针的申请！ |
| 11.改tcp软件校验位硬件校验 |
| 1. 先按照目前情况进行调试，进行下一步   经调试，偶尔会出现如上情况 |
| RT\_LWIP\_ETHTHREAD\_MBOX\_SIZE 32 |
| 增大邮箱尺寸，期望不超标 |

2018/8/15

|  |
| --- |
| 1.发送总耗时 |
| duty time: 3279  duty time: 4511  duty time: 4511  duty time: 4529  duty time: 4556  duty time: 4521  duty time: 4606  duty time: 4514  duty time: 4576  duty time: 4520  duty time: 4524  duty time: 4513 |
| 2.电脑端MAC如果不设置 全FFFFFFF，无法发到网卡。 |
|  |

2018/8/14

|  |
| --- |
| <DAI name="cVal" sAddr="11301"/>  没有生成短地址 |
| Before  435 294 bytes of readonly code memory  64 799 bytes of readonly data memory  173 367 bytes of readwrite data memory |
| after  472 518 bytes of readonly code memory  90 366 bytes of readonly data memory  178 995 bytes of readwrite data memory |
| 2. 缺少PTUV1 |

2018/8/13

|  |
| --- |
| 1. 几种变量生命  RO-data 表示 程序定义的常量const temp; RW-data 表示 已初始化的全局变量 ZI-data 表示 未初始化的全局变量 |
| 加入所有模型以后  static\_model.o 184 1 412 46 084 |
|  |
| 之前 |
|  |
| MEAS <GSE ldInst="LD0" cbName="gcbMeasureValues" des="电压电流等测量值"> |
| GC(gcbRemote remote Rmote 1 0 1000 3000 );  GC(gcbMeasureValues analog MeasureValues 1 0 1000 3000 ){  PA(4 1 4098 ffffffffffff);  } |
| 空格很害人，不能有  <GSE ldInst="LD0" cbName="gcbRemote " des="遥信"> |
| 3.finsh msh |
| Mesh  //#define FINSH\_USING\_MSH  //#define FINSH\_USING\_MSH\_ONLY    Finsh    占用空间确实小一些，但是一些功能也没有了 |
|  |
| 4.列出空间大小 |
| ls("//")---这是正确的  finsh />mkdir("//sojo//s")  0, 0x00000000  /sojo/s>tftp\_get("192.168.10.111", "//sojo//s", "stu.bin") |
| tftp\_get("192.168.10.111", "//sojo", "stu.bin") |
| rt\_snprintf((char\*)tftp\_buffer, sizeof(tftp\_buffer),  "%s//%s", dir, filename);  写失败的原因！ |
| tftp\_get("192.168.10.111", "//sojo","test\_goose.cfg") |
| 挂载文件系统稍微慢一些/需要加延时！！！！！！！！ |
|  |
| 5. 获取设备列表超时，但是在上位机一切正常！ |
| error:Socket\_write, line: 298:  lwip\_send <0  COTP: sending message failed!  COTP: message transmission finished (fragments=1, return=1)  Recive:  lwip\_send Error: 0xfffffff9  error:Socket\_write, line: 298:  lwip\_send <0, size: 1736  EWOULDBLOCK, /\* ERR\_WOULDBLOCK -7 Operation would block. \*/  改增  **lwip\_send\_ex**  **仅仅改了最后一句 为了获取错误码！**  **return** (err == ERR\_OK ? (**int**)written : err);  现在看协议栈的细节处理很好，考虑全面，比自己写的好！ |
| 增加上位机的接收时间不能解决 |

2018/8/11

|  |
| --- |
| 1.hex |
|  |
| 2. Icf配置 |
|  |
| 3.debuger 下载 |
|  |
| 4.输出项--调试信息，否则不能加断点（影响效率400-450），对mdk无太多影响还是530 |
|  |
|  |

2018/8/9

|  |
| --- |
| **1.昨天错误系 lwip select select等综合造成的影响！！！！！！！！！！！** |
|  |
| **2.接收或者发送错误，他们底层写的很好！！！改动反而是狗尾续貂！** |
|  |

2018/8/9

|  |
| --- |
| 1.调试错误 |
| Bind lw\_bind |
|  |
| 2.接收两包以后，后面不正常 |
| Recive:  3 0 0 16 TPKT: read 4 bytes from socket  TPKT: header complete (msg size = 22)  Recive:  11 E0 0 0 0 1 0 C0 1 D C2 2 0 1 C1 2 0 1 TPKT: message complete (size = 22)  COTP: option: c0 len: 01  COTP: requested TPDU size: 8192  COTP: option: c2 len: 02  COTP: option: c1 len: 02  ISO\_SERVER: COTP connection indication  COTP: send TPDU size: 8192  3 0 0 16 11 D0 0 1 0 1 0 C0 1 D C2 2 0 1 C1 2 0 1 |
| 以上，接收两包以后就收不到了。也没有再进入接收！ |
|  |
| readFromSocket  CotpConnection\_readToTpktBuffer  IsoConnection\_handleTcpConnection  handleClientConnections |
| handleIsoConnectionsThreadless  IsoServer\_processIncomingMessages  / |
| handleIsoConnections isoServerThread |
| singleThreadedServerThread |

2018/8/8

|  |
| --- |
|  |
| Program Size: Code=304298 RO-data=57206 RW-data=23332 ZI-data=136156 |
| 加入服务之后  Program Size: Code=381218 RO-data=62090 RW-data=54888 ZI-data=136280 |
|  |
| 时间测试 |
| IedServer\_updateFloatAttributeValue 414us |
|  |
| StopWatchInit();  StopWatchStart();  #if (CONFIG\_INCLUDE\_GOOSE\_SUPPORT == 1)  MmsMapping\_triggerGooseObservers(self->mmsMapping, dataAttribute->mmsValue);  #endif  StopWatchStop();  409us |
| void  Ethernet\_sendPacket(EthernetSocket ethSocket, uint8\_t\* buffer, int packetSize)  {  StopWatchInit();  StopWatchStart();  EthernetOutput(buffer, packetSize);  //MacRawOutput(buffer, packetSize);  StopWatchStop();  }  8u-15us |
|  |
| if (DataSet\_isMemberValue(dataSet, value, NULL)) {  StopWatchInit();  StopWatchStart();  MmsGooseControlBlock\_observedObjectChanged(gcb);  StopWatchStop(); |
| 431us |
|  |
| MmsGooseControlBlock\_observedObjectChanged(MmsGooseControlBlock self)  {  #if (CONFIG\_MMS\_THREADLESS\_STACK != 1)  Semaphore\_wait(self->publisherMutex);  #endif  StopWatchInit();  StopWatchStart();  uint64\_t currentTime = GoosePublisher\_increaseStNum(self->publisher);  self->retransmissionsLeft = CONFIG\_GOOSE\_EVENT\_RETRANSMISSION\_COUNT;  if (self->retransmissionsLeft > 0) {  self->nextPublishTime = currentTime + self->minTime;  GoosePublisher\_setTimeAllowedToLive(self->publisher, self->minTime \* 3);  }  else {  self->nextPublishTime = currentTime + self->maxTime;  GoosePublisher\_setTimeAllowedToLive(self->publisher, self->maxTime \* 3);  }  StopWatchStop();  GoosePublisher\_publish(self->publisher, self->dataSetValues);  #if (CONFIG\_MMS\_THREADLESS\_STACK != 1)  Semaphore\_post(self->publisherMutex);  #endif  }  5us |
|  |
| StopWatchInit();  StopWatchStart();  #if (CONFIG\_MMS\_THREADLESS\_STACK != 1)  Semaphore\_wait(self->publisherMutex);  #endif  StopWatchStop();  uint64\_t currentTime = GoosePublisher\_increaseStNum(self->publisher);  self->retransmissionsLeft = CONFIG\_GOOSE\_EVENT\_RETRANSMISSION\_COUNT;  if (self->retransmissionsLeft > 0) {  self->nextPublishTime = currentTime + self->minTime;  GoosePublisher\_setTimeAllowedToLive(self->publisher, self->minTime \* 3);  }  else {  self->nextPublishTime = currentTime + self->maxTime;  GoosePublisher\_setTimeAllowedToLive(self->publisher, self->maxTime \* 3);  }  GoosePublisher\_publish(self->publisher, self->dataSetValues);  StopWatchInit();  StopWatchStart();  #if (CONFIG\_MMS\_THREADLESS\_STACK != 1)  Semaphore\_post(self->publisherMutex);  #endif  StopWatchStop();  }  1-2us 偶尔 465us |
|  |
| 信号量改互斥问题依旧 |
|  |
| MmsGooseControlBlock\_observedObjectChanged(MmsGooseControlBlock self)  {  #if (CONFIG\_MMS\_THREADLESS\_STACK != 1)  Semaphore\_wait(self->publisherMutex);  #endif  StopWatchInit();  StopWatchStart();    uint64\_t currentTime = GoosePublisher\_increaseStNum(self->publisher);  self->retransmissionsLeft = CONFIG\_GOOSE\_EVENT\_RETRANSMISSION\_COUNT;  if (self->retransmissionsLeft > 0) {  self->nextPublishTime = currentTime + self->minTime;  GoosePublisher\_setTimeAllowedToLive(self->publisher, self->minTime \* 3);  }  else {  self->nextPublishTime = currentTime + self->maxTime;  GoosePublisher\_setTimeAllowedToLive(self->publisher, self->maxTime \* 3);  }  GoosePublisher\_publish(self->publisher, self->dataSetValues);    StopWatchStop();  #if (CONFIG\_MMS\_THREADLESS\_STACK != 1)  Semaphore\_post(self->publisherMutex);  #endif    }  356-396us |
| 地方 |
|  |
| StopWatchInit();  StopWatchStart();  GoosePublisher\_publish(self->publisher, self->dataSetValues);  StopWatchStop();  393us |
|  |
| StopWatchInit();  StopWatchStart();  uint8\_t\* buffer = self->buffer + self->payloadStart;  size\_t maxPayloadSize = GOOSE\_MAX\_MESSAGE\_SIZE - self->payloadStart;  int32\_t payloadLength = createGoosePayload(self, dataSet, buffer, maxPayloadSize);  self->sqNum++;  if (payloadLength == -1)  return -1;  int lengthIndex = self->lengthField;  size\_t gooseLength = payloadLength + 8;  self->buffer[lengthIndex] = gooseLength / 256;  self->buffer[lengthIndex + 1] = gooseLength & 0xff;      if (DEBUG\_GOOSE\_PUBLISHER)  printf("GOOSE\_PUBLISHER: send GOOSE message\n");  StopWatchStop();  383us |
| 以上是最大的开销！！传输字节约333字节  137字节约7u3s |

2018/8/6

|  |
| --- |
| 1. Stm32f429 VLAN   不经过设置，默认情况下，程序可以收到VLAN Tag，发送不行。需要设置。 |
| 2. 1309字节接收  1），整个是750us到2408us之间，融合到实际程序中。接收优先级为4. |
| 2）需要发送接收以及tcp处理到提升方能保证时间稳定在746-783之间。 |
| 3）发送接收同时提高，不改变tcp优先级，延时100次测试出现一次12609，其它在746-783之间 |
| 4）重新测试，最低770 其它为840-876之间 |
| 5）重新测试，最低740-787之间 之间。100us从何而来？ |
| 1. 另外把优先级调低到保护计算一下，出现问题。   综上应该是发送优先级与计算任务共同影响所致。 |
| 先按以上测试近试验，后期完善接收控制，需要引入中间层，脱离干扰。  至此接收与发送正常。 |
|  |
| 3.常用词汇 |
| 860-74 |
| LPHD 物理装置信息 |

2018/8/4

|  |
| --- |
| 1.接收错误，不单单之前有现在有 |
| receive frame faild  receive frame faild  receive frame faild  Ping 时间不稳定  和优化灭有关系，关闭优化问题依旧。    去除MicroLIb，恢复到1-2ms以内 |
| /\* avoid the heap and heap-using library functions supplied by arm \*/  #pragma import(\_\_use\_no\_heap)  void \*malloc(int n)  {  return rt\_malloc(n);  }  RTM\_EXP  mem\_std.c 对malloc进行了重写 |

2018/8/3

|  |
| --- |
| 1.下一步策略 |
| 1）Using interface eth0  duty time: 55  duty time: 55  duty time: 55 |
| 0优化  duty time: 61  duty time: 62  duty time: 62  Using interface eth0  duty time: 62  duty time: 62 |
| Using interface eth0  duty time: 150  duty time: 150  duty time: 150 |
| 2.启动引导错误 |
| // GooseSubscriber\_setListener(subscriber, gooseListener, NULL);  GooseReceiver\_addSubscriber(receiver, subscriber);  // GooseReceiver\_start(receiver); |
| //GooseSubscriber\_setListener(subscriber, gooseListener, NULL);  GooseSubscriber\_setListener(subscriber, NULL, NULL); |
| MmsValue\_printToBuffer 此函数 |
| 换成MicroLib就好了，是部分函数没有实现所致。 |
|  |
| 三级优化+时间 |
| duty time: 129  GOOSE event:  stNum: 1 sqNum: 0 |
| duty time: 129  GOOSE event:  stNum: 1 sqNum: 0  timeToLive: 0  timestamp: 1533277710.410  {1234,1903?00???.000Z,5678}  duty time: 76  GOOSE event:  stNum: 1 sqNum: 0  timeToLive: 0  timestamp: 1533277753.477  {1234,1903?00???.000Z,5678}  duty time: 75  GOOSE event:  stNum: 2 sqNum: 0  timeToLive: 0  timestamp: 1533277753.577  {1234,1903?00???.000Z,5678}  duty time: 75 |
| MACRAW Open Sucess!  duty time: 117  GOOSE event:  stNum: 1 sqNum: 0  timeToLive: 0  timestamp: 1533277985.587  {1234,1903?00???.000Z,5678}  duty time: 64  GOOSE event:  stNum: 2 sqNum: 0  timeToLive: 0  timestamp: 1533277985.687  {1234,1903?00???.000Z,5678}  duty time: 64  GOOSE event:  stNum: 3 sqNum: 0  timeToLive: 0  timestamp: 1533277986.788  {1234,1903?00???.000Z,5678}  duty time: 64 |
| 复制时间10us 201字节 |
| 三级优化以后 |
| duty time: 98  GOOSE event:  stNum: 1 sqNum: 0  timeToLive: 0  timestamp: 1533278550.477  {1234,194403030106?.000Z,5678}  GOOSE event:  stNum: 1 sqNum: 0  timeToLive: 0  timestamp: 1533278550.477  {1234,194403030106?.000Z,5678}  duty time: 55  GOOSE event:  stNum: 4 sqNum: 0  timeToLive: 0  timestamp: 1533278552.777  {1234,194403030106?.000Z,5678}  duty time: 55  GOOSE event:  stNum: 1 sqNum: 0  timeToLive: 0  timestamp: 1533278561.210  {1234,194403030106?.000Z,5678}  duty time: 55  GOOSE event:  stNum: 3 sqNum: 0  timeToLive: 0  timestamp: 1533278562.410  {1234,194403030106?.000Z,5678}  duty time: 55  GOOSE event:  stNum: 4 sqNum: 0  timeToLive: 0  timestamp: 1533278563.510  {1234,194403030106?.000Z,5678} |
| 1333字节 797us，首次时间更长4ms-2ms |
| 去除复制时间是719us |
|  |
| 2.三级优化加时间 |
|  |
|  |
| microlib对比 |
|  |
|  |
| 全开优化时间基本相同719us。以后采用二级优化，加优化时间 |

2018/8/2

|  |
| --- |
| 1 之前分布式之上，开启dp83848，测得延时很惊人！ |
|  |
| 取消分布式任务 |
|  |
| MAC:Address: Stmicroe\_08:27:3b (00:80:e1:08:27:3b) |
| 1. 普通交换机，少4个字节 8100 8000 |
|  |
| 换交换机还是不行，后直接插拔网线发现是因为，经过交换机被滤掉了！ |

2018/7/31

|  |
| --- |
| 1.检查所有列表与自定义队列等的插入删除，避免冲突问题。 |
| 2.有灯不正常闪烁的，在不能有效找到拓扑结构以后。系W5500初始化失败所致。 |
| 3.rt\_kprintf("\n DATA:%s,%s, %s, %d\n", \_\_DATE\_\_, \_\_TIME\_\_, \_\_FUNCTION\_\_, \_\_LINE\_\_); 需要重新编译此页 |
| 4.不报错  rt\_err\_t err = rt\_mutex\_init (&udp\_mutex, "udp\_mutex", RT\_IPC\_FLAG\_FIFO );  if (err != RT\_EOK)  {  while(1);//TODO:此种有问题  } |

2018/7/30

|  |
| --- |
| 1.联络路径信息重复 |
| 1）出现重复问题。 |
|  |
| 2.异常错误 |
| psr: 0x81000000  pc: 0x080606f4  lr: 0x080938ff 具有重复性  r12: 0x00000000  r03: 0x00000000  r02: 0xffffffff  r01: 0xc06e665c  r00: 0x06478048  hard fault on thread: udpser  thread pri status sp stack size max used left tick error  ------- --- ------- ---------- ---------- ------ ---------- ---  tshell 20 ready 0x00000154 0x00000800 16% 0x00000009 000  connect 15 ready 0x000000a8 0x00000400 36% 0x00000014 -02  distpro 5 ready 0x00000088 0x00000400 23% 0x00000012 -02  SIMSW 3 suspend 0x00000080 0x00000400 25% 0x0000000c 000  udpser 5 ready 0x00000100 0x00000800 18% 0x0000000e 000  idle 28 ready 0x00000044 0x00001000 20% 0x0000000b 000  iec104 23 ready 0x0000011c 0x00001000 06% 0x00000014 000  iec101 24 ready 0x0000011c 0x00001000 09% 0x00000004 000  dp83848 21 ready 0x000002f4 0x00001000 23% 0x00000014 000  watch 18 ready 0x000000fc 0x00001f40 03% 0x0000000f 000  protect 9 ready 0x000000fc 0x00001000 06% 0x00000013 000  cal 8 ready 0x0000011c 0x00001000 10% 0x00000012 000  tcpip 10 ready 0x000000cc 0x00000400 37% 0x00000006 -02  etx 12 suspend 0x0000008c 0x00000400 13% 0x00000010 000  erx 12 suspend 0x0000008c 0x00000400 13% 0x00000010 000  tidle 31 ready 0x00000054 0x00000400 08% 0x00000008 000  timer 0 suspend 0x0000005c 0x00000200 17% 0x00000009 000  init 16 close 0x000000f4 0x00001000 45% 0x00000002 000 |
| ErrorCode GetSwitchList(const ListDouble\* listTopology, ListDouble\* listSwitch)  {  uint8\_t size = list\_size(listTopology);  ListElment\* element = list\_head(listTopology);  for (uint8\_t i = 0; i < size; i++)  {  uint8\_t num = GET\_TOPOLOGY\_ELEMENT(element)->switchNum;  for (uint8\_t k = 0; k < num; k++)  {  ListInsertNext(listSwitch, NULL, (GET\_TOPOLOGY\_ELEMENT(element)->switchCollect + k));    }  element = element->next;  } |
|  |
| 0x65 显然是错误的 |
|  |
|  |
|  |
| else if (isSet == SET\_PATH\_CONNECT) //已经存在则不进行重复添加  {  pid->hopsNumber = hops; //更新跳数  pid->timeStamp.updateTime = GetTime();（更新tick时改变size）  pid->timeStamp.isValid = true;  return ERROR\_OK\_NULL;  } |
| 结论：pid未赋值错误！ |
| 1. 联络开关判别，由于轮询时间周期1S，而整个动作过程仅100ms左右，故障信息来不及处理。就已经被覆盖掉了。   方式1：闭锁一个是闭锁条件前移到接收。  方式2：添加其它启动条件。  先按方式1进行。 |
| 4.极可能又陷入SPI循环中！ |

2018/7/26

|  |
| --- |
| 1.联络路径判别以后不能有效更新 |
| 此处和里面的实现机制相关，只在变位时更新。实际上不知道开关什么时候变位。需要检测。  本质上需要维持开关信息的互传与更新机制。包括上线问题。 |
| 2.升级驱动应该全面升级，切忌遗漏 |

2018/7/25

|  |
| --- |
| 1.关于中点dll赋值丢失问题 |
| 128-512不能正常，赋值丢失，后改为static恢复 |
|  |

2018/7/24

|  |
| --- |
| 1.W5500 比实际数据多8字节 |
| thread pri status sp stack size max used left tick error  ------- --- ------- ---------- ---------- ------ ---------- ---  connect 15 ready 0x00000074 0x00000400 39% 0x00000005 000  distpro 5 suspend 0x00000084 0x00000400 22% 0x00000012 000  SIMSW 3 suspend 0x0000007c 0x00000400 25% 0x00000012 000  tshell 20 suspend 0x00000154 0x00000800 16% 0x00000008 000  udpser 5 suspend 0x00000090 0x00000800 69% 0x00000010 000  idle 28 ready 0x000000d4 0x00001000 25% 0x0000000b 000  iec104 23 ready 0x0000011c 0x00001000 06% 0x0000000c 000  iec101 24 ready 0x0000011c 0x00001000 06% 0x00000013 000  dp83848 21 ready 0x000002f8 0x00001000 23% 0x0000000e 000 |
| 增加stacksize |
| 3.FRAM 地址 |
| common\_data.h |
| /\* FRAM----------------------------------------------------------------------\*/  #define ADDR\_FRAM\_MONITOR 0x00000 // FRAM自检  #define ADDR\_FRAM\_UPDATE 0x00001 // 在线更新 2字节  #define ADDR\_FRAM\_START\_NUM 0x00003 // 开机次数一个字节  #define ADDR\_FRAM\_CURRENT\_SN 0x00004 // 当前定值区号 1个字节  #define ADDR\_FRAM\_CALI\_FACTOR 0x00050 // 校准系数存储起始地址 0x50  #define ADDR\_FRAM\_AREA0 0x00100 // FRAM参数存储起始地址 运行参数0区 0x600  #define ADDR\_FRAM\_AREA1 0x00700 // FRAM参数存储起始地址 定值区一 0x600  #define ADDR\_FRAM\_AREA2 0x00D00 // FRAM参数存储起始地址 定值区二 0x600  #define ADDR\_FRAM\_SOE 0x01300 // FRAM参数存储起始地址 SOE // 0x1400  #define ADDR\_FRAM\_SOE\_NEW 0x02700 // FRAM参数存储起始地址 SOE // 0x1400  #define ADDR\_FRAM\_GRID 0x0A000 // 网架起始地址 0x2000 |
| 1）.先find 获取句柄device\_fram  2）然后 rt\_device\_write(device\_fram, ADDR\_FRAM\_MONITOR, &flag, 1); |
| 4.nanoPB 不同结构可能不同 |
| 65 A6 89 00 A6 65 01 0A 6F 0A 09 02 85 00 0A 0A 81 94 A0 85 0C 82 94 A0 85 0C 10 01 1A 21 08 81 94 A0 85 0C 10 41 18 01 2A 01 02 30 00 38 00 40 00 48 00 50 02 58 00 6A 02 01 00 72 02 01 00 22 06 08 00 20 01 28 00 3A 12 08 01 10 01 18 00 20 00 28 81 94 A0 85 0C 30 00 38 00 42 36 08 81 94 A0 85 0C 10 21 18 21 20 21 28 00 30 00 38 00 40 00 48 AC 02 50 96 01 58 C8 01 60 90 03 68 90 03 70 64 78 90 03 80 01 64 88 01 F4 03 90 01 F4 03 98 01 01 00 BD 51 1A |

2018/7/23

|  |
| --- |
| 1.数据如何固化到FRAM中？ |
| 2.关于通信UDP设置问题。 |
| 1. 定义常规：5555 / 维护端口4321 |
| 1. 首次配置如何处理，配置关系 |

2018/7/21

|  |
| --- |
| 1.COAP 运用的可行性！ |
| 2.//INIT\_COMPONENT\_EXPORT(rt\_multi\_common\_data\_init) 临时屏蔽 |
| 3.串口数据出现数据丢失的问题。——添加超时，接收关闭中断，DMA缓冲 |

2018/7/20

|  |
| --- |
| 1.转供电超时 |
| 11:55:57 Send: 65 A6 06 00 A6 65 00 00 AA 00 30 01 0D 0A A8 C0 04 43 7A 1A  11:55:57 Recive: FROM：A0D, TO:FFFF, FUNCODE:7, ValidData:01 0D 0A A8 C0 02 01 00 00  11:55:57 Recive: FROM：A0D, TO:FFFF, FUNCODE:8, ValidData:01 0D 0A A8 C0 00  11:55:57 Recive: FROM：A0D, TO:FFFF, FUNCODE:C, ValidData:01 0D 0A A8 C0 00  11:56:03 Send: 65 A6 06 00 A6 65 00 00 AA 00 30 01 0D 0A A8 C0 08 46 7A 1A  11:56:04 Recive: FROM：A0D, TO:FFFF, FUNCODE:C, ValidData:01 0D 0A A8 C0 01  11:56:09 Recive: FROM：A0D, TO:FFFF, FUNCODE:7, ValidData:01 0D 0A A8 C0 02 02 00 00  11:56:09 Recive: FROM：A0D, TO:FFFF, FUNCODE:7, ValidData:01 0D 0A A8 C0 02 02 00 00  11:56:09 Recive: FROM：A0D, TO:FFFF, FUNCODE:8, ValidData:01 0D 0A A8 C0 00  11:56:09 Recive: FROM：A0D, TO:FFFF, FUNCODE:C, ValidData:01 0D 0A A8 C0 01    11:56:15 Recive: FROM：A0B, TO:A0C, FUNCODE:B, ValidData:0B 0A A8 C0 02 01  11:56:15 Recive: FROM：A0B, TO:A0D, FUNCODE:B, ValidData:0B 0A A8 C0 02 04  11:56:15 Recive: FROM：A0B, TO:A10, FUNCODE:B, ValidData:0B 0A A8 C0 02 03  11:56:15 Recive: FROM：A0B, TO:A11, FUNCODE:B, ValidData:0B 0A A8 C0 02 02  11:56:15 Recive: FROM：A0B, TO:A13, FUNCODE:B, ValidData:0B 0A A8 C0 02 01  11:56:15 Recive: FROM：A04, TO:A01, FUNCODE:B, ValidData:04 0A A8 C0 02 01  11:56:15 Recive: FROM：A04, TO:A0D, FUNCODE:B, ValidData:04 0A A8 C0 02 02  11:56:15 Recive: FROM：A04, TO:A0F, FUNCODE:B, ValidData:04 0A A8 C0 02 01 |
|  |

2018/7/19

|  |
| --- |
| 1.AddStation 命令 有内存泄漏情况，在重复添加时——已改近 |
|  |
|  |

2017/7/17

|  |
| --- |
| 嵌入式操作 |
| 1.乱码溢出 |
| ID: C0A80A13, FAULT: 2锛?TICK:79967.  STATION:C0A80A01, Code: 9. Tick: 80546  thread:router\_self stack overflow |
| 2.内存超了，超过110K |
| 1. 单片机内，轮询导致有些得不到执行，导致错误。 |
| 节点13,16，过流，但是16已结判断结束了，但是13还没处理到，导致错误。  解决这个问题：1.对于来回数据复制接口需要调整。避免低效率复制。2.调整优先级和频次  3.需要提供与上位机设置的优先级，避免设置被打断，从而引入错乱模式 |
| 1. 突然进入     Printf问题，以前发生过 |
| 5.无法响应 |
| 14:37:52 Recive: FROM：A09, TO:A0B, FUNCODE:2, ValidData:01 0B 0A A8 C0 01  14:37:54 Recive: FROM：A02, TO:A0B, FUNCODE:2, ValidData:01 0B 0A A8 C0 01  14:37:56 Recive: FROM：A02, TO:A0B, FUNCODE:2, ValidData:01 0B 0A A8 C0 01  14:37:58 Recive: FROM：A02, TO:A0B, FUNCODE:2, ValidData:01 0B 0A A8 C0 01  14:38:00 Recive: FROM：A02, TO:A0B, FUNCODE:2, ValidData:01 0B 0A A8 C0 01  14:38:02 Recive: FROM：A13, TO:A0B, FUNCODE:2, ValidData:01 0B 0A A8 C0 01  14:38:04 Recive: FROM：A13, TO:A0B, FUNCODE:2, ValidData:01 0B 0A A8 C0 01  14:38:06 Recive: FROM：A13, TO:A0B, FUNCODE:2, ValidData:01 0B 0A A8 C0 01  14:38:08 Recive: FROM：A13, TO:A0B, FUNCODE:2, ValidData:01 0B 0A A8 C0 01 |
| 6.收到数据 |
| 65 A6 06 00 A6 65 0E 0A 0B 0A 02 01 0B 0A A8 C0 01 A3 40 1A 0A A8 C0 0C 0A A8 C0 13 0A A8 C0 62 09 83 07 1A 65 0B 0A 13 0A 03 01 01 21 00 0B 0A A8 C0 01 01 0B 0A A8 C0 45 02 04 09 0A A8 C0 0A 0A A8 C0 0C 0A |
| 65 A6 06 00 A6 65 0F 0A 0B 0A 02 01 0B 0A A8 C0 01 5F 44 1A 0A A8 C0 0C 0A A8 C0 13 0A A8 C0 62 09 4A 96 1A 65 0B 0A 02 0A 03 01 01 21 00 0B 0A A8 C0 01 01 0B 0A A8 C0 45 02 04 09 0A A8 C0 0A 0A A8 C0 0C 0A |
| 19:33:01 Recive: FROM：A0A, TO:A0B, FUNCODE:2, ValidData:01 0B 0A A8 C0 01 |
|  |
| 19:33:01 Recive: FROM：A0A, TO:A0B, FUNCODE:2, ValidData:01 0B 0A A8 C0 01 |
| 65 A6 06 00 A6 65 09 0A 0B 0A 02 01 0B 0A A8 C0 01 D7 5A 1A 0A A8 C0 0C 0A A8 C0 13 0A A8 C0 62 09 7F F6 1A 65 0B 0A 11 0A 03 01 01 21 00 0B 0A A8 C0 01 01 0B 0A A8 C0 45 02 04 09 0A A8 C0 0A 0A A8 C0 0C 0A |
| 0A A8 C0 0C 0A A8 C0 13 0A A8 C0 62 09 65 46 1A 65 0B 0A 12 0A 03 01 01 21 00 0B 0A A8 C0 01 01 0B 0A A8 C0 45 02 04 09 0A A8 C0 0A 0A A8 C0 0C 0A  65 A6 06 00 A6 65 0A 0A 0B 0A 02 01 0B 0A A8 C0 01 93 55 1A |
| head与end不一致 |

2018/7/16

|  |
| --- |
|  |
| C0A80A11: ID: C0A80A11, FAULT: 1， TICK:210040.  11, FAULT: C0A80A10: set fault.  C0A80A0D: set fault.  C0A80A11,Fault TransmitMessage, TICK:210041.  C0A80A11 :FROM: 21 TO: 22. TICK: 210041.  ID: C0A80A10, FAULT: 1， TICK:210045.  ID: C0A80A0D, FAULT: 1， TICK:210046.  C0A80A13,Trigger TransmitMessage, TICK:210052.  C0A80A13 :FROM: 21 TO: 25. TICK: 210052.  C0A80A12,Trigger TransmitMessage, TICK:210053.  C0A80A12 :FROM: 21 TO: 25. TICK: 210054.  C0A80A11 :FROM: 22 TO: 23. TICK: 210058.  C0A80A11,treatment open, TICK:210058.  C0A80A11 :FROM: 23 TO: 24. TICK: 210058.  C0A80A10,Fault TransmitMessage, TICK:210058.  C0A80A10 :FROM: 21 TO: 22. TICK: 210058.  C0A80A10 :FROM: 22 TO: 23. TICK: 210059.  C0A80A10,treatment open, TICK:210059.  C0A80A10 :FROM: 23 TO: 24. TICK: 210059.  C0A80A0D,Fault TransmitMessage, TICK:210059.  C0A80A0D :FROM: 21 TO: 22. TICK: 210059.  C0A80A0F,Trigger TransmitMessage, TICK:210074.  C0A80A0F :FROM: 21 TO: 25. TICK: 210074.  C0A80A0E,Trigger TransmitMessage, TICK:210074.  C0A80A0E :FROM: 21 TO: 25. TICK: 210074.  C0A80A0D :FROM: 22 TO: 23. TICK: 210084.  C0A80A0D :FROM: 23 TO: 26. TICK: 210084.  ID: C0A80A11, FAULT: 2， TICK:210096.  ID: C0A80A11, SwitchState: 2. TICK:210096  ID: C0A80A10, FAULT: 2， TICK:210096.  ID: C0A80A10, SwitchState: 2. TICK:210096  C0A80A11,Removal Sucess, TICK:210104.  C0A80A11 :FROM: 24 TO: 91. TICK: 210104.  C0A80A11 :FROM: 91 TO: FF. TICK: 210104.  C0A80A10,Removal Sucess, TICK:210104.  C0A80A10 :FROM: 24 TO: 91. TICK: 210104.  C0A80A10 :FROM: 91 TO: FF. TICK: 210104.  C0A80A0D :FROM: 26 TO: 91. TICK: 210114.  C0A80A0D :FROM: 91 TO: FF. TICK: 210114.  C0A80A13 :FROM: 25 TO: 51. TICK: 210215.  C0A80A12 :FROM: 25 TO: 31. TICK: 210215.  C0A80A12 :FROM: 31 TO: FF. TICK: 210216.  C0A80A0F :FROM: 25 TO: 31. TICK: 210236.  C0A80A0F :FROM: 31 TO: FF. TICK: 210236.  C0A80A0E :FROM: 25 TO: 31. TICK: 210236.  C0A80A0E :FROM: 31 TO: FF. TICK: 210236.  ID: C0A80A13, SwitchState: 2. TICK:210266  C0A80A13,Insulate SUCCESS, TICK:210266.  C0A80A13 :FROM: 51 TO: 71. TICK: 210266.  C0A80A0B,Meet In TransferPower Mode, TICK:210276.  C0A80A0B :FROM: 21 TO: 72. TICK: 210276.  C0A80A13,Transfer Master Send Cmd., TICK:210286.  C0A80A13 :FROM: 71 TO: 91. TICK: 210286.  C0A80A13 :FROM: 91 TO: FF. TICK: 210290.  C0A80A0B,Transfer Close Operate, TICK:210290.  ID: C0A80A0B, SwitchState: 1. TICK:210346  C0A80A0B,Transfer Success, TICK:210350.  C0A80A0B :FROM: 72 TO: 91. TICK: 210350.  C0A80A0B :FROM: 91 TO: FF. TICK: 210350.  状态不一样。问题所在？ |
| 确实有溢出错误。 |
| 调大缓冲从1024到2048，首次正确。 |
| 复归后，不正确。16,17均动作 |
| ID: C0A80A11, FAULT: 1， TICK:26390.  ID: C0A80A10, FAULT: 1， TICK:26390.  ID: C0A80A0D, FAULT: 1， TICK:26390.  C0A80A11,Fault TransmitMessage, TICK: 26398.  C0A80A11 :FROM: 21 TO: 22. TICK: 26398.  C0A80A10,Fault TransmitMessage, TICK: 26398.  C0A80A10 :FROM: 21 TO: 22. TICK: 26398.  C0A80A0D,Fault TransmitMessage, TICK: 26398.  C0A80A0D :FROM: 21 TO: 22. TICK: 26398.  C0A80A13,Trigger TransmitMessage, TICK: 26408.  C0A80A13 :FROM: 21 TO: 25. TICK: 26408.  C0A80A12,Trigger TransmitMessage, TICK: 26408.  C0A80A12 :FROM: 21 TO: 25. TICK: 26408.  C0A80A11 :FROM: 22 TO: 23. TICK: 26408.  C0A80A11,treatment open, TICK: 26408.  C0A80A11 :FROM: 23 TO: 24. TICK: 26408.  C0A80A10 :FROM: 22 TO: 23. TICK: 26408.  C0A80A10,treatment open, TICK: 26408.  C0A80A10 :FROM: 23 TO: 24. TICK: 26408.  C0A80A0F,Trigger TransmitMessage, TICK: 26409.  C0A80A0F :FROM: 21 TO: 25. TICK: 26409.  C0A80A0E,Trigger TransmitMessage, TICK: 26409.  C0A80A0E :FROM: 21 TO: 25. TICK: 26409.  C0A80A0D :FROM: 22 TO: 23. TICK: 26409.  C0A80A0D :FROM: 23 TO: 26. TICK: 26409.  ID: C0A80A11, FAULT: 2， TICK:26445.  ID: C0A80A11, SwitchState: 2. TICK:26445  ID: C0A80A10, FAULT: 2， TICK:26445.  ID: C0A80A10, SwitchState: 2. TICK:26445  C0A80A11,Removal Sucess, TICK: 26449.  C0A80A11 :FROM: 24 TO: 91. TICK: 26449.  C0A80A11 :FROM: 91 TO: FF. TICK: 26449.  C0A80A10,Removal Sucess, TICK: 26449.  C0A80A10 :FROM: 24 TO: 91. TICK: 26449.  C0A80A10 :FROM: 91 TO: FF. TICK: 26449.  C0A80A0D :FROM: 26 TO: 91. TICK: 26459.  C0A80A0D :FROM: 91 TO: FF. TICK: 26459.  C0A80A13 :FROM: 25 TO: 51. TICK: 26569.  C0A80A12 :FROM: 25 TO: 31. TICK: 26569.  C0A80A12 :FROM: 31 TO: FF. TICK: 26569.  C0A80A0F :FROM: 25 TO: 51. TICK: 26569.  C0A80A0E :FROM: 25 TO: 31. TICK: 26569.  C0A80A0E :FROM: 31 TO: FF. TICK: 26569.  ID: C0A80A13, SwitchState: 2. TICK:26615  C0A80A13,Insulate SUCCESS, TICK: 26619.  C0A80A13 :FROM: 51 TO: 71. TICK: 26619.  C0A80A0B,Meet In TransferPower Mode, TICK: 26629. |
| 以上情况，首次上电正确，重新复位后发生错乱，应该是状态位未能有效清空所致。 |
| 对于故障触发后，增加重新复位标识。  switchProperty->distributionArea->StartUpdate(switchProperty);// 重新复位 |
| 对于接收处理。   |  |  |  |  | | --- | --- | --- | --- | | 条件 | 逻辑 | | 结果 | | 有配电区域 | **&&** | **&&** | UpdatePowerArea（）  **！**isGatherCalculateCompleted **-->**  GatherCompletedAndJudgeFaultArea（）  IsAlreayExitedFault（） | | 有故障信息 | **||** | | 处在故障处理状态IsRun | |
| 统一复归（19个点，2018缓冲），会导致队列溢出，需要一部分一部分处理 |
| 3.缺失状态F2故障，切除拒动，不能进行隔离。 |
| 地方 |
| C0A80A02: set fault.  C0A80A01: set fault.  ID: C0A80A02, FAULT: 1， TICK:27921.  ID: C0A80A01, FAULT: 1， TICK:27921.  C0A80A02,Fault TransmitMessage, TICK: 27929.  C0A80A02 :FROM: 21 TO: 22. TICK: 27929.  C0A80A01,Fault TransmitMessage, TICK: 27929.  C0A80A01 :FROM: 21 TO: 22. TICK: 27929.  C0A80A04,Trigger TransmitMessage, TICK: 27939.  C0A80A04 :FROM: 21 TO: 25. TICK: 27939.  C0A80A03,Trigger TransmitMessage, TICK: 27939.  C0A80A03 :FROM: 21 TO: 25. TICK: 27939.  C0A80A02 :FROM: 22 TO: 23. TICK: 27950.  C0A80A02,treatment open, TICK: 27950.  C0A80A02 :FROM: 23 TO: 24. TICK: 27950.  C0A80A01 :FROM: 22 TO: 23. TICK: 27950.  C0A80A01 :FROM: 23 TO: 26. TICK: 27951.  C0A80A02,reject open, TICK: 28052.  C0A80A01,backup open because of Reject, TICK: 28064.  C0A80A01 :FROM: 26 TO: 27. TICK: 28064.  ID: C0A80A01, FAULT: 2， TICK:28101.  ID: C0A80A01, SwitchState: 2. TICK:28102  C0A80A04 :FROM: 25 TO: 31. TICK: 28104.  C0A80A04 :FROM: 31 TO: FF. TICK: 28104.  C0A80A03 :FROM: 25 TO: 31. TICK: 28104.  C0A80A03 :FROM: 31 TO: FF. TICK: 28104.  C0A80A01,Backup: Removal Sucess, TICK: 28105.  C0A80A01 :FROM: 27 TO: 91. TICK: 28105.  C0A80A01 :FROM: 91 TO: FF. TICK: 28105.  C0A80A02,Removal Sucess: Backup , TICK: 28117.  C0A80A02 :FROM: 24 TO: 92. TICK: 28118.  C0A80A02 :FROM: 92 TO: FF. TICK: 28118. |
| 4.故障状态出现一次未知？ ——单侧失压故障，未知 |
| 5.1，2置过流，2拒分 |
| ID: C0A80A02, FAULT: 1， TICK:434856.  ID: C0A80A01, FAULT: 1， TICK:434856.  C0A80A02,Fault TransmitMessage, TICK:434862.  C0A80A02 :FROM: 21 TO: 22. TICK: 434862.  C0A80A01,Fault TransmitMessage, TICK:434862.  C0A80A01 :FROM: 21 TO: 22. TICK: 434864.  C0A80A04,Trigger TransmitMessage, TICK:434877.  C0A80A04 :FROM: 21 TO: 25. TICK: 434877.  C0A80A03,Trigger TransmitMessage, TICK:434877.  C0A80A03 :FROM: 21 TO: 25. TICK: 434877.  C0A80A02 :FROM: 22 TO: 23. TICK: 434887.  C0A80A02,treatment open, TICK:434887.  C0A80A02 :FROM: 23 TO: 24. TICK: 434887.  C0A80A01 :FROM: 22 TO: 23. TICK: 434887.  C0A80A01 :FROM: 23 TO: 26. TICK: 434887.  ID: C0A80A02, FAULT: 2， TICK:434926.  ID: C0A80A02, SwitchState: 2. TICK:434926  C0A80A02,Removal Sucess, TICK:434927.  C0A80A02 :FROM: 24 TO: 91. TICK: 434927.  C0A80A02 :FROM: 91 TO: FF. TICK: 434927.  C0A80A01 :FROM: 26 TO: 91. TICK: 434937.  C0A80A01 :FROM: 91 TO: FF. TICK: 434937.  C0A80A04 :FROM: 25 TO: 31. TICK: 435037.  C0A80A04 :FROM: 31 TO: FF. TICK: 435037.  C0A80A03 :FROM: 25 TO: 51. TICK: 435037.  C0A80A03,Insulate area backup, TICK:435047.  ID: C0A80A03, SwitchState: 2. TICK:435086  C0A80A03,Insulate SUCCESS, TICK:435088.  C0A80A03 :FROM: 51 TO: 71. TICK: 435088.  C0A80A08,Meet In TransferPower Mode, TICK:435098.  C0A80A08 :FROM: 21 TO: 72. TICK: 435098.  C0A80A03,Transfer Master Send Cmd., TICK:435108.  C0A80A03 :FROM: 71 TO: 91. TICK: 435108.  C0A80A03 :FROM: 91 TO: FF. TICK: 435108. |
| 13:11:44 Recive: FROM：A08, TO:A03, FUNCODE:D, ValidData:01 08 0A A8 C0 08 0A A8 C0 【容量信息】  13:11:44 Recive: FROM：A03, TO:A08, FUNCODE:D, ValidData:02 08 0A A8 C0 【启动转供电】  13:11:44 Recive: FROM：A03, TO:A08, FUNCODE:D, ValidData:03 08 0A A8 C0 【不进行转供电】  13:11:44 Recive: FROM：A03, TO:FFFF, FUNCODE:7, ValidData:01 03 0A A8 C0 02 02 00 00  13:11:44 Recive: FROM：A03, TO:FFFF, FUNCODE:7, ValidData:01 03 0A A8 C0 02 02 00 00  13:11:44 Recive: FROM：A03, TO:FFFF, FUNCODE:8, ValidData:02 03 0A A8 C0 00 01 0A A8 C0 00  13:11:44 Recive: FROM：A03, TO:FFFF, FUNCODE:C, ValidData:01 03 0A A8 C0 01 |
| 6. 切除逻辑不正确 |
| ID: C0A80A02, FAULT: 1， TICK:21922.  ID: C0A80A01, FAULT: 1， TICK:21922.  C0A80A02,Fault TransmitMessage, TICK: 21928.  C0A80A02 :FROM: 21 TO: 22. TICK: 21928.  C0A80A01,Fault TransmitMessage, TICK: 21928.  C0A80A01 :FROM: 21 TO: 22. TICK: 21928.  C0A80A04,Trigger TransmitMessage, TICK: 21938.  C0A80A04 :FROM: 21 TO: 25. TICK: 21938.  C0A80A03,Trigger TransmitMessage, TICK: 21938.  C0A80A03 :FROM: 21 TO: 25. TICK: 21938.  C0A80A02 :FROM: 22 TO: 23. TICK: 21948.  C0A80A02,treatment open, TICK: 21948.  C0A80A02 :FROM: 23 TO: 24. TICK: 21948.  C0A80A01 :FROM: 22 TO: 23. TICK: 21948.  C0A80A01 :FROM: 23 TO: 26. TICK: 21948.  ID: C0A80A02, FAULT: 2， TICK:21987.  ID: C0A80A02, SwitchState: 2. TICK:21987  C0A80A02,Removal Sucess, TICK: 21988.  C0A80A02 :FROM: 24 TO: 91. TICK: 21988.  C0A80A02 :FROM: 91 TO: FF. TICK: 21988.  C0A80A01 :FROM: 26 TO: 91. TICK: 21998.  C0A80A01 :FROM: 91 TO: FF. TICK: 21998.  C0A80A04 :FROM: 25 TO: 31. TICK: 22098.  C0A80A04 :FROM: 31 TO: FF. TICK: 22098.  C0A80A03 :FROM: 25 TO: 51. TICK: 22098.  C0A80A03,Insulate area backup, TICK: 22108.【此处不正确，无法区分拒分与正常情况】  ID: C0A80A03, SwitchState: 2. TICK:22147  C0A80A03,Insulate SUCCESS, TICK: 22148.  C0A80A03 :FROM: 51 TO: 71. TICK: 22148.  C0A80A08,Meet In TransferPower Mode, TICK: 22158.  C0A80A08 :FROM: 21 TO: 72. TICK: 22158.  C0A80A03,Transfer Master Send Cmd., TICK: 22168.  C0A80A03 :FROM: 71 TO: 91. TICK: 22168.  C0A80A03 :FROM: 91 TO: FF. TICK: 22168.  C0A80A08,Transfer Close Operate, TICK: 22178.  ID: C0A80A08, SwitchState: 1. TICK:22237  C0A80A08,Transfer Success, TICK: 22238. |
| 13:49:19 Recive: FROM：A02, TO:FFFF, FUNCODE:8, ValidData:02 02 0A A8 C0 01 01 0A A8 C0 01  13:49:19 Recive: FROM：A02, TO:FFFF, FUNCODE:7, ValidData:01 02 0A A8 C0 02 02 00 00  13:49:19 Recive: FROM：A02, TO:FFFF, FUNCODE:7, ValidData:01 02 0A A8 C0 02 02 00 00  **13:49:19 Recive: FROM：A02, TO:FFFF, FUNCODE:8, ValidData:02 02 0A A8 C0 01 01 0A A8 C0 01**  13:49:19 Recive: FROM：A02, TO:FFFF, FUNCODE:C, ValidData:01 02 0A A8 C0 00  13:49:19 Recive: FROM：A01, TO:FFFF, FUNCODE:7, ValidData:01 01 0A A8 C0 01 01 00 00  13:49:19 Recive: FROM：A01, TO:FFFF, FUNCODE:7, ValidData:01 01 0A A8 C0 01 01 00 00  13:49:19 Recive: FROM：A01, TO:FFFF, FUNCODE:8, ValidData:01 01 0A A8 C0 01  13:49:19 Recive: FROM：A01, TO:FFFF, FUNCODE:C, ValidData:01 01 0A A8 C0 00  13:49:20 Recive: FROM：A04, TO:FFFF, FUNCODE:7, ValidData:01 04 0A A8 C0 02 02 00 00  13:49:20 Recive: FROM：A03, TO:FFFF, FUNCODE:C, ValidData:01 03 0A A8 C0 01  13:49:20 Recive: FROM：A08, TO:A03, FUNCODE:D, ValidData:01 08 0A A8 C0 08 0A A8 C0  13:49:20 Recive: FROM：A03, TO:A08, FUNCODE:D, ValidData:02 08 0A A8 C0 |
| 7.多个链表错误再次引发异常 |
| 1. 配电区域为故障路径   if (error == ERROR\_OK\_NULL)  {  if (i == 0)//只针对首个进行标注。其它只标记配电区域  {  find->removalType = type;  }  //配电区域为链路  (find->distributionArea != NULL) ? (find->distributionArea->SignRemovalMessage(find)) : (find);  } |
| 1. 1,3,5分闸跳跃故障 |
| C0A80A05,Fault TransmitMessage, TICK: 30905.  C0A80A05 :FROM: 21 TO: 22. TICK: 30905.  C0A80A03,Fault TransmitMessage, TICK: 30905.  C0A80A03 :FROM: 21 TO: 22. TICK: 30905.  C0A80A01,Fault TransmitMessage, TICK: 30905.  C0A80A01 :FROM: 21 TO: 22. TICK: 30906.  C0A80A08,Trigger TransmitMessage, TICK: 30916.  C0A80A08 :FROM: 21 TO: 25. TICK: 30916.  C0A80A07,Trigger TransmitMessage, TICK: 30919.  C0A80A07 :FROM: 21 TO: 25. TICK: 30920.  C0A80A06,Trigger TransmitMessage, TICK: 30920.  C0A80A06 :FROM: 21 TO: 25. TICK: 30920.  C0A80A04,Trigger TransmitMessage, TICK: 30920.  C0A80A04 :FROM: 21 TO: 25. TICK: 30920.  C0A80A02,Trigger TransmitMessage, TICK: 30920.  C0A80A02 :FROM: 21 TO: 25. TICK: 30920.  C0A80A05 :FROM: 22 TO: 23. TICK: 30930.  C0A80A05,treatment open, TICK: 30932.  C0A80A05 :FROM: 23 TO: 24. TICK: 30932.  C0A80A03 :FROM: 22 TO: 23. TICK: 30932.  C0A80A03 :FROM: 23 TO: 26. TICK: 30932.  C0A80A01 :FROM: 22 TO: 23. TICK: 30932.  C0A80A01 :FROM: 23 TO: 26. TICK: 30932.  ID: C0A80A05, FAULT: 2， TICK:30970.  ID: C0A80A05, SwitchState: 2. TICK:30970  C0A80A05,Removal Sucess, TICK: 30972.  C0A80A05 :FROM: 24 TO: 91. TICK: 30972.  C0A80A05 :FROM: 91 TO: FF. TICK: 30972.  C0A80A08 :FROM: 25 TO: 31. TICK: 31082.  C0A80A08 :FROM: 31 TO: FF. TICK: 31082.  C0A80A07 :FROM: 25 TO: 31. TICK: 31083.  C0A80A07 :FROM: 31 TO: FF. TICK: 31083.  C0A80A06 :FROM: 25 TO: 31. TICK: 31083.  C0A80A06 :FROM: 31 TO: FF. TICK: 31083.  C0A80A04 :FROM: 25 TO: 31. TICK: 31083.  C0A80A04 :FROM: 31 TO: FF. TICK: 31086.  C0A80A02 :FROM: 25 TO: 31. TICK: 31087.  C0A80A02 :FROM: 31 TO: FF. TICK: 31087.  C0A80A03,backup overtime open, TICK: 31337.  C0A80A03 :FROM: 26 TO: 27. TICK: 31337.  C0A80A01,backup overtime open, TICK: 31337.  C0A80A01 :FROM: 26 TO: 27. TICK: 31337.  ID: C0A80A03, FAULT: 2， TICK:31376.  ID: C0A80A03, SwitchState: 2. TICK:31376  ID: C0A80A01, FAULT: 2， TICK:31376.  ID: C0A80A01, SwitchState: 2. TICK:31376  C0A80A03,Backup: Removal Sucess, TICK: 31377.  C0A80A03 :FROM: 27 TO: 91. TICK: 31377.  C0A80A03 :FROM: 91 TO: FF. TICK: 31377.  C0A80A01,Backup: Removal Sucess, TICK: 31378.  C0A80A01 :FROM: 27 TO: 91. TICK: 31378.  C0A80A01 :FROM: 91 TO: FF. TICK: 31379. |
| 10,。 F14故障分析Insulate area backup，  添加到电源开关的距离。 |
| C0A80A0D: set fault.  C0A80A10: set fault.  ID: C0A80A10, FAULT: 1， TICK:29004.  ID: C0A80A0D, FAULT: 1， TICK:29004.  C0A80A10,Fault TransmitMessage, TICK: 29010.  C0A80A10 :FROM: 21 TO: 22. TICK: 29010.  C0A80A0D,Fault TransmitMessage, TICK: 29010.  C0A80A0D :FROM: 21 TO: 22. TICK: 29010.  C0A80A11,Trigger TransmitMessage, TICK: 29020.  C0A80A11 :FROM: 21 TO: 25. TICK: 29020.  C0A80A0E,Trigger TransmitMessage, TICK: 29020.  C0A80A0E :FROM: 21 TO: 25. TICK: 29020.  C0A80A0F,Trigger TransmitMessage, TICK: 29020.  C0A80A0F :FROM: 21 TO: 25. TICK: 29020.  C0A80A10 :FROM: 22 TO: 23. TICK: 29030.  C0A80A10,treatment open, TICK: 29030.  C0A80A10 :FROM: 23 TO: 24. TICK: 29030.  C0A80A0D :FROM: 22 TO: 23. TICK: 29030.  C0A80A0D :FROM: 23 TO: 26. TICK: 29030.  C0A80A10,reject open, TICK: 29130.  C0A80A0D,backup open because of Reject, TICK: 29140.  C0A80A0D :FROM: 26 TO: 27. TICK: 29140.  ID: C0A80A0D, FAULT: 2， TICK:29179.  ID: C0A80A0D, SwitchState: 2. TICK:29179  C0A80A11 :FROM: 25 TO: 51. TICK: 29181.  C0A80A0E :FROM: 25 TO: 31. TICK: 29181.  C0A80A0E :FROM: 31 TO: FF. TICK: 29181.  C0A80A0D,Backup: Removal Sucess, TICK: 29181.  C0A80A0D :FROM: 27 TO: 91. TICK: 29181.  C0A80A0D :FROM: 91 TO: FF. TICK: 29181.  C0A80A0F :FROM: 25 TO: 51. TICK: 29182.  C0A80A10,Removal Sucess: Backup , TICK: 29192.  C0A80A10 :FROM: 24 TO: 92. TICK: 29192.  C0A80A10 :FROM: 92 TO: FF. TICK: 29192.  C0A80A0F,Insulate area backup, TICK: 29192.  ID: C0A80A0F, SwitchState: 2. TICK:29229  C0A80A0F,Insulate SUCCESS, TICK: 29232.  C0A80A0F :FROM: 51 TO: 71. TICK: 29232.  ID: C0A80A11, SwitchState: 2. TICK:29239  C0A80A0D,It's path connected., TICK: 29245.  C0A80A01,It's path connected., TICK: 29245.  C0A80A11,Insulate SUCCESS, TICK: 29246.  C0A80A11 :FROM: 51 TO: 71. TICK: 29246.  C0A80A0B,Meet In TransferPower Mode, TICK: 29258.  C0A80A0B :FROM: 21 TO: 72. TICK: 29258.  C0A80A11,Transfer Master Send Cmd., TICK: 29268.  C0A80A11 :FROM: 71 TO: 91. TICK: 29268.  C0A80A11 :FROM: 91 TO: FF. TICK: 29268.  C0A80A0B,Transfer Close Operate, TICK: 29279.  ID: C0A80A0B, SwitchState: 1. TICK:29339  C0A80A0B,Transfer Success, TICK: 29339.  C0A80A0B :FROM: 72 TO: 91. TICK: 29339.  C0A80A0B :FROM: 91 TO: FF. TICK: 29339.  C0A80A0F,Transfer Master Failure: receive overTime, TICK: 29739.  C0A80A0F :FROM: 71 TO: 91. TICK: 29739.  C0A80A0F :FROM: 91 TO: FF. TICK: 29739. |
| 11.三点联动出现反复问题 |

2018/7/14

|  |
| --- |
|  |
| C0A80A11,Fault TransmitMessage, TICK: 28466.  C0A80A11 :FROM: 21 TO: 22. TICK: 28466.  C0A80A10,Fault TransmitMessage, TICK: 28466.  C0A80A10 :FROM: 21 TO: 22. TICK: 28466.  C0A80A0D,Fault TransmitMessage, TICK: 28467.  C0A80A0D :FROM: 21 TO: 22. TICK: 28467.  C0A80A13,Trigger TransmitMessage, TICK: 28501.  C0A80A13 :FROM: 21 TO: 25. TICK: 28501.  C0A80A12,Trigger TransmitMessage, TICK: 28501.  C0A80A12 :FROM: 21 TO: 25. TICK: 28502.  C0A80A0F,Trigger TransmitMessage, TICK: 28503.  C0A80A0F :FROM: 21 TO: 25. TICK: 28503.  C0A80A0E,Trigger TransmitMessage, TICK: 28503.  C0A80A0E :FROM: 21 TO: 25. TICK: 28503.  C0A80A0D :FROM: 22 TO: 23. TICK: 28553.  C0A80A0D :FROM: 23 TO: 26. TICK: 28553.  C0A80A11 :FROM: 22 TO: 23. TICK: 28564.  C0A80A11,treatment open, TICK: 28565.  C0A80A11 :FROM: 23 TO: 24. TICK: 28565.  C0A80A10 :FROM: 22 TO: 23. TICK: 28565.  C0A80A10 :FROM: 23 TO: 26. TICK: 28565.  ID: C0A80A11, FAULT: 2， TICK:28601.  ID: C0A80A11, SwitchState: 2. TICK:28601  C0A80A11,Removal Sucess, TICK: 28606.  C0A80A11 :FROM: 24 TO: 91. TICK: 28606.  C0A80A11 :FROM: 91 TO: FF. TICK: 28606.  C0A80A10 :FROM: 26 TO: 91. TICK: 28697.  C0A80A10 :FROM: 91 TO: FF. TICK: 28697.  C0A80A0D :FROM: 26 TO: 91. TICK: 28697.  C0A80A0D :FROM: 91 TO: FF. TICK: 28698.  C0A80A13 :FROM: 25 TO: 32. TICK: 28825.  C0A80A13,OverTime TransmitMessage, TICK: 28829.  C0A80A13 :FROM: 32 TO: 91. TICK: 28829.  C0A80A13 :FROM: 91 TO: FF. TICK: 28829.  C0A80A12 :FROM: 25 TO: 32. TICK: 28830.  C0A80A12,OverTime TransmitMessage, TICK: 28830.  C0A80A12 :FROM: 32 TO: 91. TICK: 28830.  C0A80A12 :FROM: 91 TO: FF. TICK: 28830.  C0A80A0F :FROM: 25 TO: 32. TICK: 28888.  C0A80A0F,OverTime TransmitMessage, TICK: 28888.  C0A80A0F :FROM: 32 TO: 91. TICK: 28888.  C0A80A0F :FROM: 91 TO: FF. TICK: 28889.  C0A80A0E :FROM: 25 TO: 32. TICK: 28889.  C0A80A0E,OverTime TransmitMessage, TICK: 28890.  C0A80A0E :FROM: 32 TO: 91. TICK: 28890.  C0A80A0E :FROM: 91 TO: FF. TICK: 28890. |
| 后先判断1,35，在判断16,17，变好了 |

2018/7/13

|  |
| --- |
| 1.转供电错误 |
| 11:52:37 Recive: FROM：A02, TO:FFFF, FUNCODE:7, ValidData:01 02 0A A8 C0 02 01 00 00  11:52:37 Recive: FROM：A05, TO:FFFF, FUNCODE:C, ValidData:01 05 0A A8 C0 01  11:52:37 Recive: FROM：A08, TO:A05, FUNCODE:D, ValidData:01 05 0A A8 C0 05 0A A8 C0  11:52:42 Recive: FROM：A05, TO:FFFF, FUNCODE:7, ValidData:01 05 0A A8 C0 02 02 00 00  11:52:42 Recive: FROM：A05, TO:FFFF, FUNCODE:7, ValidData:01 05 0A A8 C0 02 02 00 00 |
| 2.关于全部复归问题 |
| 对于单机复归，比较方便。但是对于多机复归的本质是需要更新的状态量的同步。  1）列出需要同步的状态信息量。  2）同步的目的是维持信息的有效性。  3）需要对状态量进行“变化更新”，“周期更新”，并且需要避开特殊时期。 |
| error:\_TransferPowerS  error:\_StationExecuteFunctioncode, line: 264:  StationExecuteFunctioncode: error code: 360F614  ncode:  error code: 360F6 Transfer\_ControlSend Error: 326F6EC  error:\_FaultDealStateCenter, line: 128:  326F824 :FaultRemovalStateCenter ERROR : 326F6F8. |
| 14:36:33 Recive: FROM：A05, TO:FFFF, FUNCODE:C, ValidData:01 05 0A A8 C0 01  14:36:33 Recive: FROM：A08, TO:A05, FUNCODE:D, ValidData:01 08 0A A8 C0 08 0A A8 C0  14:37:01 Recive: FROM：A05, TO:A08, FUNCODE:D, ValidData:02 08 0A A8 C0  14:37:14 Recive: FROM：A05, TO:FFFF, FUNCODE:7, ValidData:01 05 0A A8 C0 02 02 00 00  14:37:14 Recive: FROM：A05, TO:FFFF, FUNCODE:7, ValidData:01 05 0A A8 C0 02 02 00 00  14:37:17 Recive: FROM：A08, TO:FFFF, FUNCODE:7, ValidData:01 08 0A A8 C0 02 02 00 00 |
| ID: C0A80A05, SwitchState: 2. TICK:26950  C0A80A05,Insulate SUCCESS, TICK: 26950.  C0A80A05 :FROM: 51 TO: 71. TICK: 26950.  C0A80A08,Meet In TransferPower Mode, TICK: 26960.  C0A80A08 :FROM: 21 TO: 72. TICK: 26960.  C0A80A05,Transfer Master Send Cmd., TICK: 27163.  C0A80A05 :FROM: 71 TO: 91. TICK: 27163.  C0A80A05 :FROM: 91 TO: FF. TICK: 27163.  C0A80A08,Transfer Over Cancer, TICK: 27463.  C0A80A08 :FROM: 72 TO: 91. TICK: 27463.  C0A80A08 :FROM: 91 TO: FF. TICK: 27463. |

2018/7/12

|  |
| --- |
| 1.关于联络开关判断时机的问题？ |
| 1）检测到链路上有故障发生，---->此时闭锁--->故障消失（切除成功） |
| 2.问题记录——路径错误+链表错误 |
|  |
|  |
| int list\_rem\_next(List \*list,ListElmt \*element,void \*\*data)  {  ListElmt \*old\_element;    /\*Do not allow removal from an empty list. \*/  if(list\_size(list) == 0)  return -1;    /\*Remove the element from the list. \*/  if(element == NULL)  {  /\*Handle removal from the head of the list. \*/  \*data = list->head->*data*;  old\_element = list->head;  list->head = list->head->*next*;    //previous  if (list->head != NULL)  {  list->head->*prev* = old\_element->prev;  }  if(list\_size(list) == 1)  list->tail = NULL;  }  else  {  /\*Handle removal from somewhere other than the head. \*/  if(element->*next* == NULL)  return -1;    \*data = element->*next*->*data*;  old\_element = element->*next*;  element->*next* = element->*next*->*next*;    if (element->*next* == NULL)  {  list->tail = element;  }  else  {  //previous  element->*next*->*prev* = old\_element->prev;  }  }  /\*FREE the storage allocated by the abstract datatype.\*/  FREE(old\_element);  /\*Adjust the size of the list account for the removed element. \*/  list->*size*--;  return 0;  } |
| 3.出现组合错误，合环问题导致大量判别数据 |
| C0A80A0D,It's path connected., TICK: 40301.  C0A80A04,It's path connected., TICK: 40301.  C0A80A04,It's path connected., TICK: 40301.  C0A80A01,It's path connected., TICK: 40301.  C0A80A01,It's path connected., TICK: 40302.  C0A80A01,It's path connected., TICK: 40302.  C0A80A01,It's path connected., TICK: 40302.  C0A80A01,It's path connected., TICK: 40302.  C0A80A0F,It's path connected., TICK: 40302.  C0A80A0F,It's path connected., TICK: 40302.  C0A80A0D,It's path connected., TICK: 40305.  C0A80A0D,It's path connected., TICK: 40305.  C0A80A05,It's path connected., TICK: 40305.  C0A80A05,It's path connected., TICK: 40305.  C0A80A05,It's path connected., TICK: 40305.  C0A80A05,It's path connected., TICK: 40306.  C0A80A04,It's path connected., TICK: 40306.  C0A80A04,It's path connected., TICK: 40306.  C0A80A03,It's path connected., TICK: 40306.  C0A80A03,It's path connected., TICK: 40306.  C0A80A03,It's path connected., TICK: 40306.  C0A80A01,It's path connected., TICK: 40306.  C0A80A01,It's path connected., TICK: 40306.  C0A80A0F,It's path connected., TICK: 40306.  C0A80A0F,It's path connected., TICK: 40307.  C0A80A04,It's path connected., TICK: 40310.  C0A80A04,It's path connected., TICK: 40310.  C0A80A03,It's path connected., TICK: 40310.  C0A80A03,It's path connected., TICK: 40310.  没有更新开关状态导致状态是遗留状态。底层上需要实现维护数据的有效性，定时更新。 |
| 5.需要整理所有通讯数据类型定义与形式 |
| 6.转供电冲突，电源开关进线失压。 |

2018/7/10

|  |
| --- |
| 1. 要仔细分清各个变量传递关系。调度关系，变量传递关系   最近这两天在上面吃不少亏！  如果在数据接收部分判断，对于故障发起点没有问题，~~但是对于其他店，收不到自己的故障信息，无法进行判断。~~  如果在分布式判断，对于三点以上的由于优先级问题可能导致冲突。正在计算是，被打断重新计算。  考虑不能状态位之间的配合。  如果在解析中处理。 |
| 1. 监控状态  |  |  |  |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | --- | --- | --- | | ID | 类型 | 合分位 | 邻居 | 容量 | 故障 | 操作 | 超时 | 配电区域 | |  |  |  |  |  |  |  |  |  | | 合闸超时时间 | 分闸超时时间 | 后备保护启动时间 |  |  |  |  |  |  | |
| 3.对于末梢开关，直接分闸不再扩散，就没有隔离过程。 |
|  |
| 4. 1-3-5-7 5，7拒分 |
| Virtual Node:0, ExecuteFunctioncode, Functioncode: 30.  ID: C0A80A07, FAULT: 1， TICK:56850.  ID: C0A80A05, FAULT: 1， TICK:56850.  ID: C0A80A03, FAULT: 1， TICK:56850.  ID: C0A80A01, FAULT: 1， TICK:56850.  C0A80A07,Fault TransmitMessage, TICK: 56859.  C0A80A07 :FROM: 21 TO: 22. TICK: 56859.  C0A80A05,Fault TransmitMessage, TICK: 56859.  C0A80A05 :FROM: 21 TO: 22. TICK: 56859.  C0A80A03,Fault TransmitMessage, TICK: 56874.  C0A80A03 :FROM: 21 TO: 22. TICK: 56874.  C0A80A03 :FROM: 22 TO: 23. TICK: 56874.  C0A80A03 :FROM: 23 TO: 26. TICK: 56874.  C0A80A01,Fault TransmitMessage, TICK: 56875.  C0A80A01 :FROM: 21 TO: 22. TICK: 56875.  C0A80A08,Trigger TransmitMessage, TICK: 56888.  C0A80A08 :FROM: 21 TO: 25. TICK: 56888.  C0A80A06,Trigger TransmitMessage, TICK: 56889.  C0A80A06 :FROM: 21 TO: 25. TICK: 56889.  C0A80A04,Trigger TransmitMessage, TICK: 56889.  C0A80A04 :FROM: 21 TO: 25. TICK: 56889.  C0A80A02,Trigger TransmitMessage, TICK: 56889.  C0A80A02 :FROM: 21 TO: 25. TICK: 56890.  C0A80A07 :FROM: 22 TO: 23. TICK: 56914.  C0A80A07,treatment open, TICK: 56914.  C0A80A07 :FROM: 23 TO: 24. TICK: 56914.  C0A80A05 :FROM: 22 TO: 23. TICK: 56915.  C0A80A05 :FROM: 23 TO: 26. TICK: 56915.  C0A80A01 :FROM: 22 TO: 23. TICK: 56916.  C0A80A01 :FROM: 23 TO: 26. TICK: 56916.  ID: C0A80A07, FAULT: 2， TICK:56951.  ID: C0A80A07, SwitchState: 2. TICK:56951  C0A80A07,Removal Sucess, TICK: 56957.  C0A80A07 :FROM: 24 TO: 91. TICK: 56957.  C0A80A07 :FROM: 91 TO: FF. TICK: 56957.  C0A80A05 :FROM: 26 TO: 91. TICK: 56967.  C0A80A05 :FROM: 91 TO: FF. TICK: 56967.  C0A80A08 :FROM: 25 TO: 31. TICK: 57057.  C0A80A08 :FROM: 31 TO: FF. TICK: 57058.  C0A80A06 :FROM: 25 TO: 31. TICK: 57058.  C0A80A06 :FROM: 31 TO: FF. TICK: 57058.  C0A80A04 :FROM: 25 TO: 31. TICK: 57058.  C0A80A04 :FROM: 31 TO: FF. TICK: 57058.  C0A80A02 :FROM: 25 TO: 31. TICK: 57058.  C0A80A02 :FROM: 31 TO: FF. TICK: 57059.  C0A80A03,backup overtime open, TICK: 57282.  C0A80A03 :FROM: 26 TO: 27. TICK: 57282.  ID: C0A80A03, FAULT: 2， TICK:57321.  ID: C0A80A03, SwitchState: 2. TICK:57322  C0A80A03,Backup: Removal Sucess, TICK: 57322.  C0A80A03 :FROM: 27 TO: 91. TICK: 57322.  C0A80A03 :FROM: 91 TO: FF. TICK: 57322.  C0A80A01,backup overtime open, TICK: 57323.  C0A80A01 :FROM: 26 TO: 27. TICK: 57323.  ID: C0A80A01, FAULT: 2， TICK:57362.  ID: C0A80A01, SwitchState: 2. TICK:57362  C0A80A01,Backup: Removal Sucess, TICK: 57363.  C0A80A01 :FROM: 27 TO: 91. TICK: 57363.  C0A80A01 :FROM: 91 TO: FF. TICK: 57363.  C0A80A0C,It's path connected., TICK: 57421. |
|  |
| 5. switchProperty 是 0xC0A80A03。 |
|  |

2018/7/9

|  |
| --- |
| 1、distributionArea->isGatherCompleted 值为0x28？？？ |
| 2、需要附加信息，传递所有变量，以备监控使用 |
| 3、通讯优先级，处理优先级关系！ |
| 4、  Virtual Node:0, ExecuteFunctioncode, Functioncode: 30  ID: C0A80A03, FAULT: 1， TICK:12509.  ID: C0A80A01, FAULT: 1， TICK:12509.  C0A80A03,Fault TransmitMessage, TICK: 12513.  C0A80A03 :FROM: 21 TO: 22. TICK: 12513.  C0A80A01,Fault TransmitMessage, TICK: 12513.  C0A80A01 :FROM: 21 TO: 22. TICK: 12513.  C0A80A05,Trigger TransmitMessage, TICK: 12524.  C0A80A05 :FROM: 21 TO: 25. TICK: 12524.  C0A80A04,Trigger TransmitMessage, TICK: 12524.  C0A80A04 :FROM: 21 TO: 25. TICK: 12524.  C0A80A02,Trigger TransmitMessage, TICK: 12525.  C0A80A02 :FROM: 21 TO: 25. TICK: 12525.  C0A80A03 :FROM: 22 TO: 23. TICK: 12535.  C0A80A03 :FROM: 23 TO: 26. TICK: 12535.  C0A80A01 :FROM: 22 TO: 23. TICK: 12535.  C0A80A01 :FROM: 23 TO: 26. TICK: 12535.  C0A80A04 :FROM: 25 TO: 31. TICK: 12685.  C0A80A04 :FROM: 31 TO: FF. TICK: 12687.  C0A80A03 :FROM: 26 TO: 27. TICK: 12687.  C0A80A02 :FROM: 25 TO: 31. TICK: 12687.  C0A80A02 :FROM: 31 TO: FF. TICK: 12687.  C0A80A01 :FROM: 26 TO: 27. TICK: 12687.  ID: C0A80A03, FAULT: 2， TICK:12725.  ID: C0A80A03, SwitchState: 2. TICK:12727  ID: C0A80A01, FAULT: 2， TICK:12727.  ID: C0A80A01, SwitchState: 2. TICK:12727  C0A80A03,Backup: Removal Sucess, TICK: 12728.  C0A80A03 :FROM: 27 TO: 91. TICK: 12728.  C0A80A03 :FROM: 91 TO: FF. TICK: 12728.  C0A80A01,Backup: Removal Sucess, TICK: 12728.  C0A80A01 :FROM: 27 TO: 91. TICK: 12728.  C0A80A01 :FROM: 91 TO: FF. TICK: 12728.  C0A80A05 :FROM: 25 TO: 31. TICK: 12738.  C0A80A05 :FROM: 31 TO: FF. TICK: 12738.  C0A80A0F,It's not path connected., TICK: 13165. |

2018/7/8

|  |
| --- |
| 1.首次隔离成功记录 |
| total memory: 8388580  used memory : 97344  maximum allocated memory: 97344  msh />Virtual Node:0, ExecuteFunctioncode, Functioncode: 3  ID: C0A80A03, FAULT: 1.  ID: C0A80A01, FAULT: 1.  C0A80A03,Fault TransmitMessage, TICK: 13683.  C0A80A03 :FROM: 21 TO: 22. TICK: 13683.  C0A80A01,Fault TransmitMessage, TICK: 13683.  C0A80A01 :FROM: 21 TO: 22. TICK: 13683.  C0A80A05,Trigger TransmitMessage, TICK: 13693.  C0A80A05 :FROM: 21 TO: 23. TICK: 13693.  C0A80A04,Trigger TransmitMessage, TICK: 13693.  C0A80A04 :FROM: 21 TO: 23. TICK: 13693.  C0A80A02,Trigger TransmitMessage, TICK: 13693.  C0A80A02 :FROM: 21 TO: 23. TICK: 13693.  C0A80A03 :FROM: 22 TO: 24. TICK: 13703.  C0A80A03 :FROM: 24 TO: 25. TICK: 13703.  C0A80A03 :FROM: 25 TO: 27. TICK: 13703.  C0A80A01 :FROM: 22 TO: 24. TICK: 13703.  C0A80A01 :FROM: 24 TO: 25. TICK: 13703.  C0A80A01 :FROM: 25 TO: 27. TICK: 13703.  ID: C0A80A03, FAULT: 2.  ID: C0A80A03, SwitchState: 2.  ID: C0A80A01, FAULT: 2.  ID: C0A80A01, SwitchState: 2.  C0A80A03,Backup Removal Sucess, TICK: 13753.  C0A80A03 :FROM: 27 TO: 91. TICK: 13753.  C0A80A03 :FROM: 91 TO: FF. TICK: 13753.  C0A80A01,Backup Removal Sucess, TICK: 13753.  C0A80A01 :FROM: 27 TO: 91. TICK: 13753.  C0A80A01 :FROM: 91 TO: FF. TICK: 13753.  C0A80A05 :FROM: 23 TO: 51. TICK: 13853.  C0A80A04 :FROM: 23 TO: 31. TICK: 13853.  C0A80A04 :FROM: 31 TO: FF. TICK: 13853.  C0A80A02 :FROM: 23 TO: 31. TICK: 13853.  C0A80A02 :FROM: 31 TO: FF. TICK: 13853.  C0A80A05,Insulate SUCCESS, TICK: 13929.  C0A80A05 :FROM: 51 TO: 91. TICK: 13929.  C0A80A05 :FROM: 91 TO: FF. TICK: 13929.  ID: C0A80A05, SwitchState: 2. |
| 2.内存使用情况分析 |
| memory total: 8388580, used:99552, maxused: 99552  memory total: 8388580, used:100368, maxused: 100368 （+816）  memory total: 8388580, used:100276, maxused: 100732 （-92）  memory total: 8388580, used:100008, maxused: 100732 （-268） +456  ID:C0A80A01, isn't connect swtich.  memory total: 8388580, used:100008, maxused: 100732  memory total: 8388580, used:100464, maxused: 100732 （+456）  memory total: 8388580, used:100468, maxused: 101120 （+4）  memory total: 8388580, used:100104, maxused: 101120 （-364） +96  ID:C0A80A08, isn't connect swtich.  memory total: 8388580, used:100104, maxused: 101120  memory total: 8388580, used:100560, maxused: 101120 （+456）  memory total: 8388580, used:100580, maxused: 101232 （+20）  memory total: 8388580, used:100200, maxused: 101232 （-380） （+96）  ID:C0A80A04, isn't connect swtich.  memory total: 8388580, used:100200, maxused: 101232  memory total: 8388580, used:100656, maxused: 101232 （+456）  memory total: 8388580, used:100580, maxused: 101232 （-76）  memory total: 8388580, used:100296, maxused: 101232 （-284） （+96）  ID:C0A80A01, isn't connect swtich.  memory total: 8388580, used:100296, maxused: 101232  memory total: 8388580, used:100752, maxused: 101232 （+456）  memory total: 8388580, used:100780, maxused: 101432 （+28）  memory total: 8388580, used:100392, maxused: 101432 （-380） （+96）  ID:C0A80A08, isn't connect swtich.  memory total: 8388580, used:100392, maxused: 101432  memory total: 8388580, used:100848, maxused: 101432 （+456）  memory total: 8388580, used:100892, maxused: 101544 （+44）  memory total: 8388580, used:100488, maxused: 101544 （-404） （+96）  ID:C0A80A04, isn't connect swtich.  memory total: 8388580, used:100488, maxused: 101544  memory total: 8388580, used:100944, maxused: 101544 （+456）  memory total: 8388580, used:100892, maxused: 101544 （-52）  memory total: 8388580, used:100584, maxused: 101544 （-308） （+96）  ID:C0A80A01, isn't connect swtich.  memory total: 8388580, used:100584, maxused: 101544  memory total: 8388580, used:101040, maxused: 101544 （+456）  memory total: 8388580, used:101092, maxused: 101744 （+52）  memory total: 8388580, used:100708, maxused: 101744 （-384） （+124）  ID:C0A80A08, isn't connect swtich.  memory total: 8388580, used:100708, maxused: 101744  memory total: 8388580, used:101164, maxused: 101744 （+456）  memory total: 8388580, used:101204, maxused: 101856 （+40）  memory total: 8388580, used:100804, maxused: 101856 （-400） （+96）  ID:C0A80A04, isn't connect swtich. |
| 更详细判断 |
| msh />  memory total: 8388580, used:99760, maxused: 99796  memory total: 8388580, used:99760, maxused: 99796  memory total: 8388580, used:100216, maxused: 100216 （+456）  memory total: 8388580, used:100676, maxused: 100872 （+460）  memory total: 8388580, used:100220, maxused: 100872 （-456）  memory total: 8388580, used:100220, maxused: 100872 （0）  memory total: 8388580, used:99856, maxused: 100872 （-364） （+96）  ID:C0A80A08, isn't connect swtich.  memory total: 8388580, used:99856, maxused: 100872  memory total: 8388580, used:99856, maxused: 100872  memory total: 8388580, used:100312, maxused: 100872  memory total: 8388580, used:100788, maxused: 100984  memory total: 8388580, used:100332, maxused: 100984  memory total: 8388580, used:100332, maxused: 100984  memory total: 8388580, used:99952, maxused: 100984  ID:C0A80A04, isn't connect swtich.  memory total: 8388580, used:99952, maxused: 100984  memory total: 8388580, used:99952, maxused: 100984  memory total: 8388580, used:100408, maxused: 100984  memory total: 8388580, used:100788, maxused: 100984  memory total: 8388580, used:100332, maxused: 100984  memory total: 8388580, used:100332, maxused: 100984  memory total: 8388580, used:100048, maxused: 100984  ID:C0A80A01, isn't connect swtich.  memory total: 8388580, used:100048, maxused: 100984  memory total: 8388580, used:100048, maxused: 100984  memory total: 8388580, used:100504, maxused: 100984 （+456）  memory total: 8388580, used:100988, maxused: 101184 （+484）  memory total: 8388580, used:100532, maxused: 101184 （-456）  memory total: 8388580, used:100532, maxused: 101184 0  memory total: 8388580, used:100144, maxused: 101184 （-388） （+96）  ID:C0A80A08, isn't connect swtich.  memory total: 8388580, used:100144, maxused: 101184  memory total: 8388580, used:100144, maxused: 101184  memory total: 8388580, used:100600, maxused: 101184  memory total: 8388580, used:101100, maxused: 101296  memory total: 8388580, used:100644, maxused: 101296  memory total: 8388580, used:100644, maxused: 101296  memory total: 8388580, used:100240, maxused: 101296  ID:C0A80A04, isn't connect swtich.  添加比较 |
| memory total: 8388580, used:99840, maxused: 99840  memory total: 8388580, used:99840, maxused: 99840  memory total: 8388580, used:100296, maxused: 100296  memory total: 8388580, used:100328, maxused: 100328 （+32）  FreeBFSHelper: start:memory total: 8388580, used:100524, maxused: 100524 （+196）  FreeBFSHelper: memory total: 8388580, used:100328, maxused: 100524 （-196）  memory total: 8388580, used:100360, maxused: 100524 （+32）  FreeBFSHelper: start:memory total: 8388580, used:100700, maxused: 100700 （+340）  memory total: 8388580, used:100756, maxused: 100756 【释放不是问题】 （+56）[32+24]  FreeBFSHelper: start:memory total: 8388580, used:100952, maxused: 100952  FreeBFSHelper: memory total: 8388580, used:100756, maxused: 100952  memory total: 8388580, used:100756, maxused: 100952  memory total: 8388580, used:100300, maxused: 100952 (-456)  memory total: 8388580, used:100300, maxused: 100952  memory total: 8388580, used:99936, maxused: 100952 (-364) (+96) = 32\*3 这是原因！  ID:C0A80A08, isn't connect swtich.  memory total: 8388580, used:99936, maxused: 100952  memory total: 8388580, used:99936, maxused: 100952  memory total: 8388580, used:100392, maxused: 100952  memory total: 8388580, used:100424, maxused: 100952  FreeBFSHelper: start:memory total: 8388580, used:100780, maxused: 100952  memory total: 8388580, used:100836, maxused: 100952  FreeBFSHelper: start:memory total: 8388580, used:101032, maxused: 101032  FreeBFSHelper: memory total: 8388580, used:100836, maxused: 101032  memory total: 8388580, used:100868, maxused: 101032  FreeBFSHelper: start:memory total: 8388580, used:101064, maxused: 101064  FreeBFSHelper: memory total: 8388580, used:100868, maxused: 101064  memory total: 8388580, used:100868, maxused: 101064  memory total: 8388580, used:100412, maxused: 101064  memory total: 8388580, used:100412, maxused: 101064  memory total: 8388580, used:100032, maxused: 101064  ID:C0A80A04, isn't connect swtich.  memory total: 8388580, used:100032, maxused: 101064  memory total: 8388580, used:100032, maxused: 101064  memory total: 8388580, used:100488, maxused: 101064  memory total: 8388580, used:100520, maxused: 101064  FreeBFSHelper: start:memory total: 8388580, used:100716, maxused: 101064  FreeBFSHelper: memory total: 8388580, used:100520, maxused: 101064  memory total: 8388580, used:100552, maxused: 101064  FreeBFSHelper: start:memory total: 8388580, used:100748, maxused: 101064  FreeBFSHelper: memory total: 8388580, used:100552, maxused: 101064  memory total: 8388580, used:100584, maxused: 101064  FreeBFSHelper: start:memory total: 8388580, used:100844, maxused: 101064  memory total: 8388580, used:100868, maxused: 101064  memory total: 8388580, used:100412, maxused: 101064  memory total: 8388580, used:100412, maxused: 101064  memory total: 8388580, used:100128, maxused: 101064  ID:C0A80A01, isn't connect swtich.  memory total: 8388580, used:100128, maxused: 101064  memory total: 8388580, used:100128, maxused: 101064  memory total: 8388580, used:100584, maxused: 101064  memory total: 8388580, used:100616, maxused: 101064  FreeBFSHelper: start:memory total: 8388580, used:100820, maxused: 101064  FreeBFSHelper: memory total: 8388580, used:100616, maxused: 101064  memory total: 8388580, used:100648, maxused: 101064  FreeBFSHelper: start:memory total: 8388580, used:101012, maxused: 101064  memory total: 8388580, used:101068, maxused: 101068  FreeBFSHelper: start:memory total: 8388580, used:101264, maxused: 101264  FreeBFSHelper: memory total: 8388580, used:101068, maxused: 101264  memory total: 8388580, used:101068, maxused: 101264  memory total: 8388580, used:100612, maxused: 101264  memory total: 8388580, used:100612, maxused: 101264  memory total: 8388580, used:100224, maxused: 101264  ID:C0A80A08, isn't connect swtich.  memory total: 8388580, used:100224, maxused: 101264  memory total: 8388580, used:100224, maxused: 101264  memory total: 8388580, used:100680, maxused: 101264  memory total: 8388580, used:100712, maxused: 101264  FreeBFSHelper: start:memory total: 8388580, used:101092, maxused: 101264  memory total: 8388580, used:101148, maxused: 101264  FreeBFSHelper: start:memory total: 8388580, used:101344, maxused: 101344  FreeBFSHelper: memory total: 8388580, used:101148, maxused: 101344  memory total: 8388580, used:101180, maxused: 101344  FreeBFSHelper: start:memory total: 8388580, used:101376, maxused: 101376  FreeBFSHelper: memory total: 8388580, used:101180, maxused: 101376  memory total: 8388580, used:101180, maxused: 101376  memory total: 8388580, used:100724, maxused: 101376  memory total: 8388580, used:100724, maxused: 101376  memory total: 8388580, used:100320, maxused: 101376  ID:C0A80A04, isn't connect swtich.  memory total: 8388580, used:100320, maxused: 101376  memory total: 8388580, used:100320, maxused: 101376  memory total: 8388580, used:100776, maxused: 101376  memory total: 8388580, used:100808, maxused: 101376  FreeBFSHelper: start:memory total: 8388580, used:101024, maxused: 101376  FreeBFSHelper: memory total: 8388580, used:100808, maxused: 101376  memory total: 8388580, used:100840, maxused: 101376  FreeBFSHelper: start:memory total: 8388580, used:101072, maxused: 101376  FreeBFSHelper: memory total: 8388580, used:100840, maxused: 101376  memory total: 8388580, used:100872, maxused: 101376  FreeBFSHelper: start:memory total: 8388580, used:101156, maxused: 101376  memory total: 8388580, used:101180, maxused: 101376  memory total: 8388580, used:100724, maxused: 101376  memory total: 8388580, used:100724, maxused: 101376  memory total: 8388580, used:100416, maxused: 101376  ID:C0A80A01, isn't connect swtich. |
| 后经查已经分配重复分配导致！  \*helper = (BFSHelper\*)MALLOC(sizeof(BFSHelper));  if (\*helper == NULL)  {  return ERROR\_MALLOC;  } |
|  |
| 1. Dsds   ID: C0A80A10, FAULT: 1.  ID: C0A80A0D, FAULT: 1.  C0A80A10,Fault TransmitMessage, TICK: 77807.  C0A80A10 :FROM: 21 TO: 22. TICK: 77807.  C0A80A0D,Fault TransmitMessage, TICK: 77807.  C0A80A0D :FROM: 21 TO: 22. TICK: 77807.  C0A80A11,Trigger TransmitMessage, TICK: 77817.  C0A80A11 :FROM: 21 TO: 23. TICK: 77817.  C0A80A0F,Trigger TransmitMessage, TICK: 77817.  C0A80A0F :FROM: 21 TO: 23. TICK: 77817.  C0A80A0E,Trigger TransmitMessage, TICK: 77817.  C0A80A0E :FROM: 21 TO: 23. TICK: 77817.  C0A80A10 :FROM: 22 TO: 24. TICK: 77827.  C0A80A10 :FROM: 24 TO: 25. TICK: 77827.  C0A80A10 :FROM: 25 TO: 27. TICK: 77827.  C0A80A0D :FROM: 22 TO: 24. TICK: 77827.  C0A80A0D :FROM: 24 TO: 25. TICK: 77827.  C0A80A0D :FROM: 25 TO: 27. TICK: 77827.  ID: C0A80A10, FAULT: 2.  ID: C0A80A10, SwitchState: 2.  ID: C0A80A0D, FAULT: 2.  ID: C0A80A0D, SwitchState: 2.  C0A80A10,Backup Removal Sucess, TICK: 77877.  C0A80A10 :FROM: 27 TO: 91. TICK: 77877.  C0A80A10 :FROM: 91 TO: FF. TICK: 77877.  C0A80A0D,Backup Removal Sucess, TICK: 77877.  C0A80A0D :FROM: 27 TO: 91. TICK: 77877.  C0A80A0D :FROM: 91 TO: FF. TICK: 77877.  ID:C0A80A11, isn't connect swtich.  ID:C0A80A10, isn't connect swtich.  ID:C0A80A0D, isn't connect swtich.  ID:C0A80A0B, isn't connect swtich.  ID:C0A80A08, isn't connect swtich.  ID:C0A80A04, isn't connect swtich.  ID:C0A80A01, isn't connect swtich.  C0A80A11 :FROM: 23 TO: 31. TICK: 77977.  C0A80A11 :FROM: 31 TO: FF. TICK: 77977.  C0A80A0F :FROM: 23 TO: 31. TICK: 77977.  C0A80A0F :FROM: 31 TO: FF. TICK: 77977.  C0A80A0E :FROM: 23 TO: 31. TICK: 77977.  C0A80A0E :FROM: 31 TO: FF. TICK: 77977.  ID:C0A80A11, isn't connect swtich. |
|  |
| 4.理清判断故障与收集，以及后备保护的需要之间的关系 |

2018/7/7

|  |
| --- |
| 1.  switch 0 id:0xc0a80a0d ip: 192:168:10:13  neighrbour num 3  neighrbour 0 id:0xc0a80a0e, ip: 192:168:10:14  neighrbour 1 id:0xc0a80a0f, ip: 192:168:10:15  neighrbour 2 id:0xc0a80a10, ip: 192:168:10:16  memory total: 8388580, used:70148, maxused: 70180  STATION:C0A80A0E, Code: 1. Tick: 23865  reserialize error, error code:{ 23 }  toplogy id: 0xc0a80a0e, ip: 192:168:10:14  toplogy type: 0x1  toplogy switch num: 0x1  switch 0 id:0xc0a80a0e ip: 192:168:10:14  neighrbour num 3  neighrbour 0 id:0xc0a80a0d, ip: 192:168:10:13  neighrbour 1 id:0xc0a80a0f, ip: 192:168:10:15  neighrbour 2 id:0xc0a80a10, ip: 192:168:10:16  memory total: 8388580, used:70148, maxused: 70180  STATION:C0A80A0F, Code: 1. Tick: 23875 |
|  |
| 15:09:42 Recive: FROM：A08, TO:A0E, FUNCODE:1, ValidData:01 0E 0A A8 C0 01  15:09:42 Recive: FROM：A08, TO:A0F, FUNCODE:1, ValidData:01 0F 0A A8 C0 01  15:09:42 Recive: FROM：A08, TO:A10, FUNCODE:1, ValidData:01 10 0A A8 C0 01  15:09:42 Recive: FROM：A08, TO:A11, FUNCODE:1, ValidData:01 11 0A A8 C0 01  15:09:42 Recive: FROM：A08, TO:A12, FUNCODE:1, ValidData:01 12 0A A8 C0 01  15:09:43 Recive: FROM：A08, TO:A13, FUNCODE:1, ValidData:01 13 0A A8 C0 01  数据错误  return MakeSimpleMessage(GET\_MESSAGE, id, (*uint8\_t*)cmd, packet);  修改错误 |
| 3. 需要多次获取，才能得到错误信息GET\_MESSAGE |
| 4.  m\_foreach 是 nullptr。 |
| ErrorCode SetConnectPath(*uint32\_t* id, PathConnected isSet, StationTopology\* toplogy)  {  CHECK\_POINT\_RETURN\_LOG(toplogy, NULL, ERROR\_NULL\_PTR, 0);  List\* list = &(toplogy->connectPath);    FOR\_EARCH\_LIST\_START(list);  *uint32\_t* getID = \*(*uint32\_t*\*)list\_data(m\_foreach);  if (getID == id)  {  if (isSet == CANCER\_PATH\_CONNECT)  {  *uint32\_t*\* pid;  list\_rem\_next(list, m\_foreach->*prev*, &pid);  SafeFree(pid);  return ERROR\_NULL;  }  }  FOR\_EARCH\_LIST\_END(); |
| 跳闸顺序  C0A80A13: CONTROL\_REMOVAL\_RESET.  ID: C0A80A03, FAULT: 1.  ID: C0A80A01, FAULT: 1.  C0A80A03,Fault TransmitMessage, TICK: 24221.  C0A80A03 :FROM: 21 TO: 22. TICK: 24222.  C0A80A01,Fault TransmitMessage, TICK: 24222.  C0A80A01 :FROM: 21 TO: 22. TICK: 24222.  C0A80A05,Trigger TransmitMessage, TICK: 24245.  C0A80A05 :FROM: 21 TO: 23. TICK: 24245.  C0A80A04,Trigger TransmitMessage, TICK: 24245.  C0A80A04 :FROM: 21 TO: 23. TICK: 24245.  C0A80A02,Trigger TransmitMessage, TICK: 24247.  C0A80A02 :FROM: 21 TO: 23. TICK: 24247.  C0A80A03 :FROM: 22 TO: 24. TICK: 24257.  C0A80A03 :FROM: 24 TO: 26. TICK: 24257.  C0A80A01 :FROM: 22 TO: 24. TICK: 24257.  C0A80A01 :FROM: 24 TO: 25. TICK: 24257.  C0A80A01 :FROM: 25 TO: 27. TICK: 24257.  ID: C0A80A03, FAULT: 2.  ID: C0A80A03, SwitchState: 2.  ID: C0A80A01, FAULT: 2.  ID: C0A80A01, SwitchState: 2.  C0A80A03 :FROM: 26 TO: 92. TICK: 24308.  C0A80A03 :FROM: 92 TO: FF. TICK: 24309.  C0A80A01 :FROM: 27 TO: 91. TICK: 24309.  C0A80A01 :FROM: 91 TO: FF. TICK: 24309.  C0A80A05 :FROM: 23 TO: 31. TICK: 24409.  C0A80A05 :FROM: 31 TO: FF. TICK: 24409.  C0A80A04 :FROM: 23 TO: 31. TICK: 24409.  C0A80A04 :FROM: 31 TO: FF. TICK: 24409.  C0A80A02 :FROM: 23 TO: 31. TICK: 24409.  C0A80A02 :FROM: 31 TO: FF. TICK: 24409.  memory total: 8388580, used:98052, maxused: 98840  memory total: 8388580, used:C0A80A0F,It's not path connected., TICK: 25050.  C0A80A0D,It's not path connected., TICK: 25051.  C0A80A01,It's not path connected., TICK: 25051.  98148, maxused: 98840 |
| 5.异常 |
| 0x013789C9 处有未经处理的异常(在 vs.exe 中): 0xC0000005: 读取位置 0x00000001 时发生访问冲突。  如有适用于此异常的处理程序，该程序便可安全地继续运行。 |
| ErrorCode FindSwitchNodeByID(const List\* listNeighboorSwitch, *uint32\_t* id, SwitchProperty\*\* find)  {  CHECK\_POINT\_RETURN\_LOG(listNeighboorSwitch, NULL, ERROR\_NULL\_PTR, 0);    FOR\_EARCH\_LIST\_START(listNeighboorSwitch);  if (GET\_SWITCH\_ELEMENT(m\_foreach)->id == id)  {  \*find = GET\_SWITCH\_ELEMENT(m\_foreach);  return ERROR\_NULL;  }  FOR\_EARCH\_LIST\_END();  return ERROR\_UNFIND;  } |
|  |
| 修改后再次出现问题！ |
| 6.添加已经覆盖 |
| C0A80A01 :FROM: 25 TO: 27. TICK: 13048.  ID: C0A80A03, FAULT: 2.  ID: C0A80A03, SwitchState: 2.  ID: C0A80A01, FAULT: 2.  ID: C0A80A01, SwitchState: 2.  C0A80A03 :FROM: 27 TO: 91. TICK: 13099.  C0A80A03 :FROM: 91 TO: FF. TICK: 13099.  C0A80A01 :FROM: 27 TO: 91. TICK: 13099.  C0A80A01 :FROM: 91 TO: FF. TICK: 13105.  C0A80A0F,It's not path connected., TICK: 13132.  C0A80A0D,It's path connected., TICK: 13133.  C0A80A0C,It's path connected., TICK: 13133.  C0A80A09,It's not path connected., TICK: 13133.  C0A80A05,It's not path connected., TICK: 13134.  C0A80A03,It's not path connected., TICK: 13134.  C0A80A01,It's path connected., TICK: 13138.  C0A80A01,It's not path connected., TICK: 13140.  toplogy id: 0xc0a80a06, ip: 192:168:10:6  toplogy type: 0x1  toplogy switch num: 0x1  switch 0 id:0xc0a80a06 ip: 192:168:10:6  neighrbour num 3  neighrbour 0 id:0xc0a80a05, ip: 192:168:10:5  neighrbour 1 id:0xc0a80a07, ip: 192:168:10:7  neighrbour 2 id:0xc0a80a08, ip: 192:168:10:8  memory total: 8388580, used:97152, maxused: 97404  toplogy id: 0xc0a80a07, ip: 192:168:10:7  toplogy type: 0x1  toplogy switch num: 0x1  switch 0 id:0xc0a80a07 ip: 192:168:10:7  neighrbour num 3  neighrbour 0 id:0xc0a80a05, ip: 192:168:10:5  neighrbour 1 id:0xc0a80a06, ip: 192:168:10:6  neighrbour 2 id:0xc0a80a08, ip: 192:168:10:8  memory total: 8388580, used:97328, maxused: 97404  C0A80A05 :FROM: 23 TO: 31. TICK: 13186.  C0A80A05 :FROM: 31 TO: FF. TICK: 13186.  C0A80A04 :FROM: 23 TO: 31. TICK: 13186.  C0A80A04 :FROM: 31 TO: FF. TICK: 13186.  C0A80A02 :FROM: 23 TO: 31. TICK: 13186.  C0A80A02 :FROM: 31 TO: FF. TICK: 13186.  toplogy id: 0xc0a80a08, ip: 192:168:10:8 |
| C0A80A01 :FROM: 24 TO: 25. TICK: 19470.  C0A80A01 :FROM: 25 TO: 27. TICK: 19470.  ID: C0A80A03, FAULT: 2.  ID: C0A80A03, SwitchState: 2.  ID: C0A80A01, FAULT: 2.  ID: C0A80A01, SwitchState: 2.  C0A80A03 :FROM: 27 TO: 91. TICK: 19521.  C0A80A03 :FROM: 91 TO: FF. TICK: 19521.  C0A80A01 :FROM: 27 TO: 91. TICK: 19521.  C0A80A01 :FROM: 91 TO: FF. TICK: 19522.  memory total: 8388580, used:103760, maxused: 104576  memory total: 8388580, used:103872, maxused: 104576  ID:C0A80A03, isn't connect swtich.  memory total: 8388580, used:103872, maxused: 104576  memory total: 8388580, used:103968, maxused: 104672  ID:C0A80A01, isn't connect swtich.  C0A80A0F,It's not path connected., TICK: 19589.  C0A80A0D,It's path connected., TICK: 19589.  C0A80A0C,It's path connected., TICK: 19589.  C0A80A09,It's not path connected., TICK: 19590.  C0A80A05,It's not path connected., TICK: 19590.  C0A80A03,It's not path connected., TICK: 19590.  C0A80A01,It's path connected., TICK: 19592.  C0A80A01,It's not path connected., TICK: 19592.  C0A80A05 :FROM: 23 TO: 31. TICK: 19612.  C0A80A05 :FROM: 31 TO: FF. TICK: 19612.  C0A80A04 :FROM: 23 TO: 31. TICK: 19612.  C0A80A04 :FROM: 31 TO: FF. TICK: 19613.  C0A80A02 :FROM: 23 TO: 31. TICK: 19613.  C0A80A02 :FROM: 31 TO: FF. TICK: 19615.  memory total: 8388580, used:103584, maxused: 104672 |
| 新问题：在运行保护时，不能更新连接路径，否则会导致判别错误。 |

2018/7/6

|  |
| --- |
| 1.错误提示 |
| Local Node Message:  toplogy id: 0xc0a80a12, ip: 192:168:10:18  toplogy type: 0x1  toplogy switch num: 0x1  switch 0 id:0xc0a80a12 ip: 192:168:10:18  neighrbour num 2  neighrbour 0 id:0xc0a80a11, ip: 192:168:10:17  neighrbour 1 id:0xc0a80a13, ip: 192:168:10:19  memory total: 8388580, used:89240, maxused: 89240  Neighbour Node Message:  switch id:0xc0a80a13 ip: 192:168:10:19  neighrbour count: 3  neighrbour 0 id:0xc0a80a11, ip: 192:168:10:17  neighrbour 1 id:0xc0a80a12, ip: 192:168:10:18  neighrbour 2 id:0xc0a80a0b, ip: 192:168:10:11  switch id:0xc0a80a11 ip: 192:168:10:17  neighrbour count: 3  neighrbour 0 id:0xc0a80a10, ip: 192:168:10:16  neighrbour 1 id:0xc0a80a12, ip: 192:168:10:18  neighrbour 2 id:0xc0a80a13, ip: 192:168:10:19  to free a bad data block:  mem: 0x007f5d20, used flag: 0, magic code: 0x1ea0  (mem->used) assertion failed at function:rt\_free, line number:589 |
|  |
| if (areaNum == 1)  {  SwitchProperty\* deleteElement;  ListElmt\* element = list\_head(listArea);  ListElmt\* lastElement = element;  *uint8\_t* size = list\_size(listArea);  for(; size-- ;)  {  if (element != NULL)  {  if (GET\_SWITCH\_ELEMENT(element)->state == SWITCH\_OPEN)  {  if (list\_is\_head(listArea, lastElement))  {  list\_rem\_next(listArea, NULL, (void\*\*)&deleteElement);  }  else  {  list\_rem\_next(listArea, lastElement, (void\*\*)&deleteElement);  }  }  lastElement = element;  element = element->*next*;  }  } |
| 0x002B698F 处有未经处理的异常(在 vs.exe 中): 0xC0000005: 读取位置 0x00000008 时发生访问冲突。  如有适用于此异常的处理程序，该程序便可安全地继续运行。 |

2018/7/5

|  |
| --- |
| 1.数组超标 |
|  |
| Run-Time Check Failure #2 - Stack around the variable 'queueBuffer' was corrupted.  数组溢出，由 10到32正常 |
| 1. 内存泄漏   多了120字节 |

2018/7/4

|  |
| --- |
| 1. |
| helper size : 17.  C0A80A14,C0A80A13,C0A80A12,C0A80A10,C0A80A0F,C0A80A0E,C0A80A0D,C0A80A0C,C0A80A0A,C0A80A09,C0A80A06,C0A80A05,C0A80A03,C0A80A02,C0A80A01,C0A80A07,C0A80A08,  helper edgeTo:  0,0,0,0,0,0,0,9,9,16,16,16,11,12,12,16,0,  helper path:  16,11,12,14,  memory total: 8388580, used:16900, maxused: 16900  【C0A80A08，C0A80A05，C0A80A03, C0A80A01】  helper size : 17.  C0A80A0C,C0A80A14,C0A80A13,C0A80A12,C0A80A10,C0A80A0F,C0A80A0E,C0A80A0D,C0A80A0A  ,C0A80A09,C0A80A06,C0A80A05,C0A80A03,C0A80A02,C0A80A01,C0A80A07,C0A80A08,  helper edgeTo:  9,0,0,0,0,0,0,0,9,16,16,16,11,12,12,16,0,  helper path:  16,9,0,  【C0A80A08，C0A80A09,C0A80A12】 |
|  |

2018/7/3

|  |
| --- |
| 1. 关于W5500 UDP接收调试纪要 |
| 1) 6/28 晚上开始进行调试（19:00--22:00）  当时与向然进行联调，主要是熟悉程序逻辑，加入测试函数，后台打印输出，测试任务占比。  对于单独的终端接收，没有ms事件的情况，能够正常接收，且不丢包（60000个/ms）.但是只要加入1 ms事件就丢包，而且是大面积丢失，即使拉大ms间隔，如5ms。接收打印对其有影响需要改到后台中。    从占空比上看，正常工作下。占比为16%。    一旦有接收任务 16% + 43%（增加）总共占比59% |
| 有接收，有ms事件的情况。说明进入即退出即重新进入。 |
| 1. 后测试中断数与接收数不一致，按照推理，应该后者大于前者。实际要小很多。   到22:00 今天到此为止。  今天考虑几个问题：（1）加入ms事件对于CPU负载的影响。（2）执行当前任务时执行调度，会不会调度任务。——不会，调度时只向优先级高的任务调度。相等等级不调度。  （3）需要重点处理中断数与接收数不一致的情况。 |
| 3）2018/7/2 9:00--21:00  理清思路，发现  ）把清除标志位放置于中断中发现老是进入无休止发送等待，试了几下遂放弃。  ）后关中断，发现直接死机，原因不明。  ）后加入do {}while（）增加响应速度，有起色，甚至出现连续多次测试不掉包的现象。但后来测试有出现丢包。丢包数量有所下降。  ）测试重新上电，首次一般能通过后来不正常，怀疑驱动。  ）16:00 感觉驱动有问题，对照半天，确实有漏洞，后修改。感觉不放心，又想把自己的驱动移植过来，移植一部分，发现大量名称重复函数，遂放弃。  ）包括之前发现死机发生在发送标志位时候，怀疑重复调用，加检测不起效果。  ）添加关闭接收中断  ）偶然发现通讯关闭Close。发现少breakler  ）在中断里清除标志位，一切正常，后发现不在中断清除有问题。 |

2018/6/30

|  |
| --- |
| 1.建立跟踪日志体系，先修路！ |
| 2.异常信息获取以后，再添加一个通用提示（字符串信息），基本上皆可以囊括大部分情况。 |
|  |
|  |

2018/6/27

|  |
| --- |
| 1.昨天问题整理描述：在VC++仿真正常，但是在嵌入式式除了语法更为严格以外的其它错误，以输出打印异常，定时器异常居多，应该是底层配置冲突导致。有些问题以前就有如heap over stack。 |
| void rt\_malloc\_sethook(void (\*hook)(void \*ptr, rt\_uint32\_t size)); |
| void rt\_malloc\_sethook(void (\*hook)(void \*ptr, rt\_size\_t size))  修改了头文件定义。  其它通过，distribution 没有通过 |
| 问题依旧。。。 |
| 就是在这翻车。。 |
| 取消进入临界区设置，能够正常打印输出。重复了之前的问题 |
| Local Node Message:  toplogy id: 0xc0psr: 0x00000000  pc: 0xfffffffe  lr: 0x20002956  r12: 0x20001b20  r03: 0xffffffff  r02: 0x00000002  r01: 0x200028ff  r00: 0x0000000a  hard fault on thread: switch |
| //rt\_enter\_critical();  // while(\*str!='\0')  // {  // if(\*str=='\n')  // {  // USART6->DR = (u8) '\r';  // while((USART6->SR&0X40)==0);  // }  // USART6->DR =\*str++;  // while((USART6->SR&0X40)==0);  // }  // rt\_exit\_critical(); |
| 以上注销，会导致  #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  停留在黄色位置 |
| #define RT\_USING\_HOOK 加上  调大堆栈256-1024 能正常使用了。。。 |
| 教训：1.简单升级不能彻底解决问题2.11-3.03，需要注意配置情况   1. 配置情况具有关联性！ 2. 分析好情况，这次证明了基本语法和方式是没有问题的！ |
|  |
| 1. 测试需要   1）计算时间，整理后备保护正确与否 |
|  |
| 3.出现一个多次下载bug，重复问题 |
| ****无锁环形队列是什么**** |
| 1. 建立综合测试监控通道   1）使用nanopb，具体作为数据载体。  2）适适用于分布式多点通信，能够灵活适应485,232，以太网组网  3）对需要信息进行代号化，减少资源占用。 |
| 1. 代号化信息   1）拓扑信息  2）状态信息  3）中间过程记录，日志系统！  日志系统内容：  rt\_kprintf("DATA:%s,file: %s, %s, %d\n",\_\_DATE\_\_, \_\_FILE\_\_, \_\_FUNCTION\_\_, \_\_LINE\_\_);   1. 错误记录：   函数（字符串），行数（整形），自定义错误代码（整形），tick（可选） |

2018/6/26

|  |
| --- |
| 1.有效数据 |
| 通讯更改处理机制以后，速度提高很多，之前轮询延时太大。还以为是系统问题。 |
| 2.通信的优先级问题，需要处理不同通信优先级问题，以在处理重要任务时，屏蔽其他任务。 |
| 1. 全局邻居列表数量不对？对于状态信息更新，应该从全局里面找还是邻居里面找？   对于普通状态信息，应该从全局以便掌握全体状态。二队故障判别？ |
| 4.需要添加一个检验区 |
| 5.const 需要从底层开始 |
| 6.嵌入式异常 |
| thread router start.  thread comself start.  thread communication start.  psr: 0x61000000  pc: 0xdeadbeee  lr: 0x0800623d  r12: 0x00000000  r03: 0x00000005  r02: 0x20000054  r01: 0x10000000  r00: 0x00000000 |
| 两断电之间进入中断。取消LL\_mDelay(10)正常 |
| 打印输出异常。 |
| psr: 0x00000040  pc: 0x20002672  lr: 0x20003430  r12: 0x00000000  r03: 0x00000001  r02: 0x20000054  r01: 0x10000000  r00: 0x00000000  hard fault on thread: comself  加大串口缓冲加大到1024问题依旧  加大Comself缓冲问题依旧  开启堆检测  psr: 0x00000040  pc: 0x200025f2  lr: 0x20003430  r12: 0x00000000  r03: 0x00000001  r02: 0x20000054  r01: 0x10000000  r00: 0x00000000  hard fault on thread: comself  \0  \ | /  thread:heap stack overflow  \0 |
| 改成100K  thread:heap stack overflow  \0sr: 0x00000000  thread: stack overflow  \0sr: 0x00000000  thread: stack overflow  关了stack over check 可以正常打印了！！！ |
| toplogy id: 0xc0a80a01, ip: 192:168:10:1  psr: 0x41000000  pc: 0x20002754  lr: 0x08006349  r12: 0xfffffffe  r03: 0x000000c8  r02: 0x00000000  r01: 0x00004000  r00: 0x00000000  hard fault on thread: switch  又是打印前后异常！ |

2018/6/25

|  |
| --- |
| 1.先修路，把状态量回显弄好。 |
| 2.为了能够提供复用率，需要减去不要的函数。 |
| 3.bool 需要改名，可能与C++实际冲突，可以考虑使用stdbool |
| 5. Dll 需要使用realeas 64bit版本 |
|  |
| 3.中哪一个的参数 实现多种传输！ |
| [StructLayout(*LayoutKind*.*Sequential*)]  public struct ComselfMessage  {  public *UInt16* sourceAddress; //地址  public *UInt16* destAdrress; //目的地址  public byte funcode; //功能代码  public *UInt16* datalen; //数据长度  //[MarshalAs(UnmanagedType.ByValArray, SizeConst = 200)]  //public byte[] validData;//有效数据最长为200 #define FRAME\_MESSAGE\_MAX 200  public *IntPtr* validDataPtr;  public byte completeFlag; //0-未完成  } |

2018/6/23

|  |
| --- |
| 1.2-2-2 阵型通讯记录 |
| 65 A6 09 00 A6 65 01 0A FF FF 07 01 01 0A A8 C0 01 01 00 00 92 24 1A  65 A6 09 00 A6 65 01 0A FF FF 07 01 01 0A A8 C0 01 01 00 00 92 24 1A  65 A6 09 00 A6 65 03 0A FF FF 07 01 03 0A A8 C0 02 01 00 00 CD A6 1A  65 A6 09 00 A6 65 02 0A FF FF 07 01 02 0A A8 C0 02 01 00 00 C0 67 1A  65 A6 09 00 A6 65 03 0A FF FF 07 01 03 0A A8 C0 02 01 00 00 CD A6 1A  65 A6 09 00 A6 65 02 0A FF FF 07 01 02 0A A8 C0 02 01 00 00 C0 67 1A  和昨天一样，现在是两遍？为什么？  超时发送 |
| 65 A6 09 00 A6 65 01 0A FF FF 07 01 01 0A A8 C0 01 01 00 00 92 24 1A  65 A6 09 00 A6 65 01 0A FF FF 07 01 01 0A A8 C0 01 01 00 02 53 A5 1A  65 A6 09 00 A6 65 03 0A FF FF 07 01 03 0A A8 C0 02 01 00 00 CD A6 1A  65 A6 09 00 A6 65 02 0A FF FF 07 01 02 0A A8 C0 02 01 00 00 C0 67 1A  65 A6 09 00 A6 65 03 0A FF FF 07 01 03 0A A8 C0 02 01 00 00 CD A6 1A  65 A6 09 00 A6 65 02 0A FF FF 07 01 02 0A A8 C0 02 01 00 00 C0 67 1A |
| 65 A6 09 00 A6 65 01 0A FF FF 07 01 01 0A A8 C0 01 01 00 00 92 24 1A  65 A6 09 00 A6 65 03 0A FF FF 07 01 03 0A A8 C0 02 01 00 00 CD A6 1A  65 A6 09 00 A6 65 02 0A FF FF 07 01 02 0A A8 C0 02 01 00 00 C0 67 1A  65 A6 09 00 A6 65 01 0A FF FF 07 01 01 0A A8 C0 01 01 00 02 53 A5 1A  65 A6 09 00 A6 65 03 0A FF FF 07 01 03 0A A8 C0 02 01 00 02 0C 27 1A  65 A6 09 00 A6 65 02 0A FF FF 07 01 02 0A A8 C0 02 01 00 02 01 E6 1A |
| 需要上位机实时呈现下面状态。。。 |
|  |
| 正确的故障处理方式：  65 A6 09 00 A6 65 01 0A FF FF 07 01 01 0A A8 C0 01 01 00 00 92 24 1A  65 A6 09 00 A6 65 03 0A FF FF 07 01 03 0A A8 C0 02 01 00 00 CD A6 1A  65 A6 09 00 A6 65 02 0A FF FF 07 01 02 0A A8 C0 02 01 00 00 C0 67 1A |

2018/6/22

|  |
| --- |
| 1. 1,2分析 |
| 65 A6 06 00 A6 65 01 0A 02 0A 02 01 02 0A A8 C0 01 E6 C6 1A  65 A6 06 00 A6 65 02 0A 01 0A 02 01 01 0A A8 C0 01 52 99 1A  65 A6 06 00 A6 65 01 0A 02 0A 02 01 02 0A A8 C0 01 E6 C6 1A  65 A6 17 00 A6 65 02 0A 01 0A 03 01 01 15 00 02 0A A8 C0 01 01 02 0A A8 C0 42 01 01 01 0A A8 C0 B6 04 3F 9B 1A  65 A6 17 00 A6 65 01 0A 02 0A 03 01 01 15 00 01 0A A8 C0 01 01 01 0A A8 C0 41 01 01 02 0A A8 C0 B4 04 9B C2 1A |
| 2. 2-2-2，相互通信正常 |
| 65 A6 06 00 A6 65 03 0A 01 0A 02 01 01 0A A8 C0 01 AE 9D 1A  65 A6 06 00 A6 65 03 0A 02 0A 02 01 02 0A A8 C0 01 5E CD 1A  65 A6 06 00 A6 65 02 0A 01 0A 02 01 01 0A A8 C0 01 52 99 1A  65 A6 06 00 A6 65 02 0A 03 0A 02 01 03 0A A8 C0 01 F2 F9 1A  65 A6 06 00 A6 65 01 0A 02 0A 02 01 02 0A A8 C0 01 E6 C6 1A  65 A6 06 00 A6 65 01 0A 03 0A 02 01 03 0A A8 C0 01 B6 F6 1A  65 A6 1B 00 A6 65 01 0A 03 0A 03 01 01 19 00 01 0A A8 C0 01 01 01 0A A8 C0 41 01 02 02 0A A8 C0 03 0A A8 C0 2E 06 0D 72 1A  65 A6 1B 00 A6 65 03 0A 02 0A 03 01 01 19 00 03 0A A8 C0 01 01 03 0A A8 C0 43 01 02 01 0A A8 C0 02 0A A8 C0 32 06 00 B4 1A  65 A6 1B 00 A6 65 02 0A 03 0A 03 01 01 19 00 02 0A A8 C0 01 01 02 0A A8 C0 42 01 02 01 0A A8 C0 03 0A A8 C0 30 06 C0 2A 1A  65 A6 1B 00 A6 65 02 0A 01 0A 03 01 01 19 00 02 0A A8 C0 01 01 02 0A A8 C0 42 01 02 01 0A A8 C0 03 0A A8 C0 30 06 60 28 1A  65 A6 1B 00 A6 65 01 0A 02 0A 03 01 01 19 00 01 0A A8 C0 01 01 01 0A A8 C0 41 01 02 02 0A A8 C0 03 0A A8 C0 2E 06 5D 73 1A  65 A6 1B 00 A6 65 03 0A 01 0A 03 01 01 19 00 03 0A A8 C0 01 01 03 0A A8 C0 43 01 02 01 0A A8 C0 02 0A A8 C0 32 06 F0 B7 1A |
| 3.在之上添加一个 |
| 65 A6 06 00 A6 65 04 0A 01 0A 02 01 01 0A A8 C0 01 DA 87 1A  65 A6 06 00 A6 65 04 0A 02 0A 02 01 02 0A A8 C0 01 2A D7 1A  65 A6 06 00 A6 65 04 0A 03 0A 02 01 03 0A A8 C0 01 7A E7 1A  65 A6 1B 00 A6 65 01 0A 04 0A 03 01 01 19 00 01 0A A8 C0 01 01 01 0A A8 C0 41 01 02 02 0A A8 C0 03 0A A8 C0 2E 06 FD 76 1A  65 A6 1B 00 A6 65 02 0A 04 0A 03 01 01 19 00 02 0A A8 C0 01 01 02 0A A8 C0 42 01 02 01 0A A8 C0 03 0A A8 C0 30 06 30 2E 1A  65 A6 1B 00 A6 65 03 0A 04 0A 03 01 01 19 00 03 0A A8 C0 01 01 03 0A A8 C0 43 01 02 01 0A A8 C0 02 0A A8 C0 32 06 A0 B1 1A |
| 4. 3-3-3-3队列 |
| 65 A6 06 00 A6 65 04 0A 01 0A 02 01 01 0A A8 C0 01 DA 87 1A  65 A6 06 00 A6 65 04 0A 02 0A 02 01 02 0A A8 C0 01 2A D7 1A  65 A6 06 00 A6 65 04 0A 03 0A 02 01 03 0A A8 C0 01 7A E7 1A  65 A6 06 00 A6 65 03 0A 01 0A 02 01 01 0A A8 C0 01 AE 9D 1A  65 A6 06 00 A6 65 03 0A 02 0A 02 01 02 0A A8 C0 01 5E CD 1A  65 A6 06 00 A6 65 03 0A 04 0A 02 01 04 0A A8 C0 01 FE 6E 1A  65 A6 06 00 A6 65 02 0A 01 0A 02 01 01 0A A8 C0 01 52 99 1A  65 A6 06 00 A6 65 02 0A 03 0A 02 01 03 0A A8 C0 01 F2 F9 1A  65 A6 06 00 A6 65 02 0A 04 0A 02 01 04 0A A8 C0 01 02 6A 1A  65 A6 06 00 A6 65 01 0A 02 0A 02 01 02 0A A8 C0 01 E6 C6 1A  65 A6 06 00 A6 65 01 0A 03 0A 02 01 03 0A A8 C0 01 B6 F6 1A  65 A6 06 00 A6 65 01 0A 04 0A 02 01 04 0A A8 C0 01 46 65 1A  65 A6 1F 00 A6 65 01 0A 04 0A 03 01 01 1D 00 01 0A A8 C0 01 01 01 0A A8 C0 41 01 03 02 0A A8 C0 03 0A A8 C0 04 0A A8 C0 A9 07 60 F3 1A  65 A6 1F 00 A6 65 02 0A 04 0A 03 01 01 1D 00 02 0A A8 C0 01 01 02 0A A8 C0 42 01 03 01 0A A8 C0 03 0A A8 C0 04 0A A8 C0 AB 07 21 73 1A  65 A6 1F 00 A6 65 04 0A 03 0A 03 01 01 1D 00 04 0A A8 C0 01 01 04 0A A8 C0 45 01 03 01 0A A8 C0 02 0A A8 C0 03 0A A8 C0 B0 07 75 7D 1A  65 A6 1F 00 A6 65 03 0A 04 0A 03 01 01 1D 00 03 0A A8 C0 01 01 03 0A A8 C0 43 01 03 01 0A A8 C0 02 0A A8 C0 04 0A A8 C0 AD 07 5A 5F 1A  65 A6 1F 00 A6 65 01 0A 03 0A 03 01 01 1D 00 01 0A A8 C0 01 01 01 0A A8 C0 41 01 03 02 0A A8 C0 03 0A A8 C0 04 0A A8 C0 A9 07 63 F2 1A  65 A6 1F 00 A6 65 03 0A 02 0A 03 01 01 1D 00 03 0A A8 C0 01 01 03 0A A8 C0 43 01 03 01 0A A8 C0 02 0A A8 C0 04 0A A8 C0 AD 07 B4 5F 1A  65 A6 1F 00 A6 65 02 0A 03 0A 03 01 01 1D 00 02 0A A8 C0 01 01 02 0A A8 C0 42 01 03 01 0A A8 C0 03 0A A8 C0 04 0A A8 C0 AB 07 22 72 1A  65 A6 1F 00 A6 65 04 0A 02 0A 03 01 01 1D 00 04 0A A8 C0 01 01 04 0A A8 C0 45 01 03 01 0A A8 C0 02 0A A8 C0 03 0A A8 C0 B0 07 98 7C 1A  65 A6 1F 00 A6 65 02 0A 01 0A 03 01 01 1D 00 02 0A A8 C0 01 01 02 0A A8 C0 42 01 03 01 0A A8 C0 03 0A A8 C0 04 0A A8 C0 AB 07 B8 73 1A  65 A6 1F 00 A6 65 01 0A 02 0A 03 01 01 1D 00 01 0A A8 C0 01 01 01 0A A8 C0 41 01 03 02 0A A8 C0 03 0A A8 C0 04 0A A8 C0 A9 07 8E F3 1A  65 A6 1F 00 A6 65 03 0A 01 0A 03 01 01 1D 00 03 0A A8 C0 01 01 03 0A A8 C0 43 01 03 01 0A A8 C0 02 0A A8 C0 04 0A A8 C0 AD 07 C3 5F 1A  65 A6 1F 00 A6 65 04 0A 01 0A 03 01 01 1D 00 04 0A A8 C0 01 01 04 0A A8 C0 45 01 03 01 0A A8 C0 02 0A A8 C0 03 0A A8 C0 B0 07 EF 7C 1A  65 A6 06 00 A6 65 03 0A 01 0A 02 01 01 0A A8 C0 01 AE 9D 1A  65 A6 06 00 A6 65 03 0A 02 0A 02 01 02 0A A8 C0 01 5E CD 1A  65 A6 06 00 A6 65 03 0A 04 0A 02 01 04 0A A8 C0 01 FE 6E 1A  65 A6 06 00 A6 65 02 0A 01 0A 02 01 01 0A A8 C0 01 52 99 1A  65 A6 06 00 A6 65 02 0A 03 0A 02 01 03 0A A8 C0 01 F2 F9 1A  65 A6 06 00 A6 65 02 0A 04 0A 02 01 04 0A A8 C0 01 02 6A 1A  65 A6 06 00 A6 65 01 0A 02 0A 02 01 02 0A A8 C0 01 E6 C6 1A  65 A6 06 00 A6 65 01 0A 03 0A 02 01 03 0A A8 C0 01 B6 F6 1A  65 A6 06 00 A6 65 01 0A 04 0A 02 01 04 0A A8 C0 01 46 65 1A  65 A6 06 00 A6 65 01 0A 02 0A 02 01 02 0A A8 C0 01 E6 C6 1A  65 A6 06 00 A6 65 01 0A 03 0A 02 01 03 0A A8 C0 01 B6 F6 1A  65 A6 06 00 A6 65 01 0A 04 0A 02 01 04 0A A8 C0 01 46 65 1A  65 A6 1F 00 A6 65 01 0A 03 0A 03 01 01 1D 00 01 0A A8 C0 01 01 01 0A A8 C0 41 01 03 02 0A A8 C0 03 0A A8 C0 04 0A A8 C0 A9 07 63 F2 1A  65 A6 1F 00 A6 65 03 0A 02 0A 03 01 01 1D 00 03 0A A8 C0 01 01 03 0A A8 C0 43 01 03 01 0A A8 C0 02 0A A8 C0 04 0A A8 C0 AD 07 B4 5F 1A  65 A6 1F 00 A6 65 02 0A 03 0A 03 01 01 1D 00 02 0A A8 C0 01 01 02 0A A8 C0 42 01 03 01 0A A8 C0 03 0A A8 C0 04 0A A8 C0 AB 07 22 72 1A  65 A6 1F 00 A6 65 04 0A 03 0A 03 01 01 1D 00 04 0A A8 C0 01 01 04 0A A8 C0 45 01 03 01 0A A8 C0 02 0A A8 C0 03 0A A8 C0 B0 07 75 7D 1A  65 A6 1F 00 A6 65 02 0A 01 0A 03 01 01 1D 00 02 0A A8 C0 01 01 02 0A A8 C0 42 01 03 01 0A A8 C0 03 0A A8 C0 04 0A A8 C0 AB 07 B8 73 1A  65 A6 1F 00 A6 65 01 0A 02 0A 03 01 01 1D 00 01 0A A8 C0 01 01 01 0A A8 C0 41 01 03 02 0A A8 C0 03 0A A8 C0 04 0A A8 C0 A9 07 8E F3 1A  65 A6 1F 00 A6 65 03 0A 01 0A 03 01 01 1D 00 03 0A A8 C0 01 01 03 0A A8 C0 43 01 03 01 0A A8 C0 02 0A A8 C0 04 0A A8 C0 AD 07 C3 5F 1A  65 A6 1F 00 A6 65 04 0A 02 0A 03 01 01 1D 00 04 0A A8 C0 01 01 04 0A A8 C0 45 01 03 01 0A A8 C0 02 0A A8 C0 03 0A A8 C0 B0 07 98 7C 1A  65 A6 1F 00 A6 65 02 0A 01 0A 03 01 01 1D 00 02 0A A8 C0 01 01 02 0A A8 C0 42 01 03 01 0A A8 C0 03 0A A8 C0 04 0A A8 C0 AB 07 B8 73 1A  65 A6 1F 00 A6 65 03 0A 01 0A 03 01 01 1D 00 03 0A A8 C0 01 01 03 0A A8 C0 43 01 03 01 0A A8 C0 02 0A A8 C0 04 0A A8 C0 AD 07 C3 5F 1A  65 A6 1F 00 A6 65 04 0A 01 0A 03 01 01 1D 00 04 0A A8 C0 01 01 04 0A A8 C0 45 01 03 01 0A A8 C0 02 0A A8 C0 03 0A A8 C0 B0 07 EF 7C 1A  65 A6 1F 00 A6 65 04 0A 01 0A 03 01 01 1D 00 04 0A A8 C0 01 01 04 0A A8 C0 45 01 03 01 0A A8 C0 02 0A A8 C0 03 0A A8 C0 B0 07 EF 7C 1A |
| 以上出现重复情况，正常情况下，不应该重复，后拉长TIck从1000改成3000后正常 |
| 6.更新节点时候，需要退出分布式 |
| 7.状态信息获取  65 A6 09 00 A6 65 01 0A FF FF 07 01 01 0A A8 C0 01 01 00 00 92 24 1A |
| 8.关于任务，一定确保不能让其长时间占用，不然会让其它程序一直挂起，饿死。 |
| 9.关于缓冲信息流，可以考虑采用事件通知的经典模式，既减少轮询消耗，又能提高效率。实现稍复杂。 |
| 10.故障判断启动后的通信分析 |
| 65 A6 09 00 A6 65 01 0A FF FF 07 01 01 0A A8 C0 01 01 00 00 92 24 1A  65 A6 09 00 A6 65 01 0A FF FF 07 01 01 0A A8 C0 01 01 00 00 92 24 1A  65 A6 09 00 A6 65 04 0A FF FF 07 01 04 0A A8 C0 02 01 00 00 EC E1 1A  65 A6 09 00 A6 65 04 0A FF FF 07 01 04 0A A8 C0 02 01 00 00 EC E1 1A  只有1,4右发送 |
| 11.注意保护启动的时机 |
| 12.置过流结果 |
| ID: C0A80A01, FAULT: 1.  C0A80A01 :CURRENT: A1 TO: A2.  C0A80A01 :CURRENT: A2 TO: A8.  C0A80A01 :CURRENT: A2 TO: C3.  STATION:C0A80A06, ExecuteFunctioncode : 7.  STATION:C0A80A05, ExecuteFunctioncode : 7.  STATION:C0A80A04, ExecuteFunctioncode : 7.  STATION:C0A80A03, ExecuteFunctioncode : 7.  STATION:C0A80A02, ExecuteFunctioncode : 7.  C0A80A04 :CURRENT: A1 TO: A3.  STATION:C0A80A06, ExecuteFunctioncode : 7.  STATION:C0A80A05, ExecuteFunctioncode : 7.  STATION:C0A80A04, ExecuteFunctioncode : 7.  STATION:C0A80A03, ExecuteFunctioncode : 7.  STATION:C0A80A02, ExecuteFunctioncode : 7.  STATION:C0A80A06, ExecuteFunctioncode : 7.  STATION:C0A80A05, ExecuteFunctioncode : 7.  STATION:C0A80A03, ExecuteFunctioncode : 7.  STATION:C0A80A02, ExecuteFunctioncode : 7.  C0A80A04 :CURRENT: A3 TO: A8.  C0A80A04 :CURRENT: A3 TO: C3.  STATION:C0A80A06, ExecuteFunctioncode : 7.  STATION:C0A80A05, ExecuteFunctioncode : 7.  STATION:C0A80A03, ExecuteFunctioncode : 7.  STATION:C0A80A02, ExecuteFunctioncode : 7.  STATION:C0A80A01, ExecuteFunctioncode : 7.  STATION:C0A80A01, ExecuteFunctioncode : 7. |
| 65 A6 09 00 A6 65 01 0A FF FF 07 01 01 0A A8 C0 01 01 00 00 92 24 1A  65 A6 09 00 A6 65 01 0A FF FF 07 01 01 0A A8 C0 01 01 00 00 92 24 1A  65 A6 09 00 A6 65 04 0A FF FF 07 01 04 0A A8 C0 02 01 00 00 EC E1 1A  65 A6 09 00 A6 65 04 0A FF FF 07 01 04 0A A8 C0 02 01 00 00 EC E1 1A |
| 13. 数据 |
| 65 A6 09 00 A6 65 06 0A FF FF 07 01 06 0A A8 C0 01 01 00 00 B3 63 1A  65 A6 09 00 A6 65 06 0A FF FF 07 01 06 0A A8 C0 01 01 00 00 B3 63 1A |
| 65 A6 09 00 A6 65 01 0A FF FF 07 01 01 0A A8 C0 01 01 00 00 92 24 1A  65 A6 09 00 A6 65 01 0A FF FF 07 01 01 0A A8 C0 01 01 00 00 92 24 1A |

2018/6/21

|  |
| --- |
| 1.数据分析 |
| 65 A6 12 00 A6 65 02 0A 02 01 02 0A A8 C0 01 02 0A **1A** 65 A6 12 00 A6 65 03 0A 02 01 03 0A A8 C0 01 03 0A 1A 65 A6 12 00 A6 65 04 0A 02 01 04 0A A8 C0 01 04 0A 1A 65 A6 12 00 A6 65 02 0A 02 01 02 0A A8 C0 01 02 0A 1A 65 A6 12 00 A6 65 03 0A 02 01 03 0A A8 C0 01 03 0A 1A 65 A6 12 00 A6 65 04 0A 02 01 04 0A A8 C0 01 04 0A 1A 65 A6 12 00 A6 65 02 0A 02 01 02 0A A8 C0 01 02 0A 1A 65 A6 12 00 A6 65 03 0A 02 01 03 0A A8 C0 01 03 0A 1A 65 A6 12 00 A6 65 04 0A 02 01 04 0A A8 C0 01 04 0A 1A 65 A6 12 00 A6 65 02 0A 02 01 02 0A A8 C0 01 02 0A 1A 65 A6 12 00 A6 65 03 0A 02 01 03 0A A8 C0 01 03 0A 1A 65 A6 12 00 A6 65 04 0A 02 01 04 0A A8 C0 01 04 0A 1A 65 A6 12 00 A6 65 02 0A 02 01 02 0A A8 C0 01 02 0A 1A 65 A6 12 00 A6 65 03 0A 02 01 03 0A A8 C0 01 03 0A 1A 65 A6 12 00 A6 65 04 0A 02 01 04 0A A8 C0 01 04 0A 1A 65 A6 12 00 A6 65 02 0A 02 01 02 0A A8 C0 01 02 0A 1A 65 A6 12 00 A6 65 03 0A 02 01 03 0A A8 C0 01 03 0A 1A |
| 65 A6 12 00 A6 65 01 0A 02 01 01 0A A8 C0 01 01 0A 1A 65 A6 12 00 A6 65 02 0A 02 01 02 0A A8 C0 01 02 0A 1A 65 A6 12 00 A6 65 03 0A 02 01 03 0A A8 C0 01 03 0A 1A 65 A6 12 00 A6 65 0F 0A 02 01 0F 0A A8 C0 01 0F 0A 1A 65 A |
| 花了很长时间才发现，长度赋值错误。 |
| 2. 帧中需要包含发送地址，以便处理。 |
| 65 A6 1D 00 A6 65 FF FF 03 1D 00 02 0A A8 C0 01 01 02 0A A8 C0 42 01 03 01 0A A8 C0 03 0A A8 C0 04 0A A8 C0 AB 07 A7 26 1A  65 A6 1D 00 A6 65 FF FF 03 1D 00 01 0A A8 C0 01 01 01 0A A8 C0 41 01 03 02 0A A8 C0 03 0A A8 C0 04 0A A8 C0 A9 07 C0 E6 1A |
| 3.  \ | /  - RT - Thread Operating System  / | \ 3.1.0 build Jun 21 2018  2006 - 2018 Copyright by rt-thread team  Hello RT-Thread!  File System on root initialized!  File System on sd initialization failed!  thread distribution\_thread\_entry start.  thread simulation start.  thread router start.  thread comself start.  thread communication start.  init usart4.  thread gain\_thread\_entry start.  msh />Virtual Node:0, ExecuteFunctioncode, Functioncode: 40.  toplogy id: 0xc0a80a01, ip: 192:168:10:1  toplogy type: 0x1  toplogy switch num: 0x1  switch 0 id:0xc0a80a01 ip: 192:168:10:1  neighrbour num 3  neighrbour 0 id:0xc0a80a02, ip: 192:168:10:2  neighrbour 1 id:0xc0a80a03, ip: 192:168:10:3  neighrbour 2 id:0xc0a80a04, ip: 192:168:10:4  memory total: 8388580, used:11904, maxused: 11904  ID: C0A80A01, FAULT: 2.  Virtual Node:0, ExecuteFunctioncode, Functioncode: 40.  toplogy id: 0xc0a80a02, ip: 192:168:10:2  toplogy type: 0x1  toplogy switch num: 0x1  switch 0 id:0xc0a80a02 ip: 192:168:10:2  neighrbour num 3  neighrbour 0 id:0xc0a80a01, ip: 192:168:10:1  neighrbour 1 id:0xc0a80a03, ip: 192:168:10:3  neighrbour 2 id:0xc0a80a04, ip: 192:168:10:4  memory total: 8388580, used:14876, maxused: 14876  ID: C0A80A02, FAULT: 2.  STATION:C0A80A02, ExecuteFunctioncode, Functioncode: 2.  STATION:C0A80A01, ExecuteFunctioncode, Functioncode: 2.  STATION:C0A80A02, ExecuteFunctioncode, Functioncode: 3.  toplogy id: 0xc0a80a01, ip: 192:168:10:1  toplogy type: 0x1  toplogy switch num: 0x1  switch 0 id:0xc0a80a01 ip: 192:168:10:1  neighrbour num 3  neighrbour 0 id:0xc0a80a02, ip: 192:168:10:2  neighrbour 1 id:0xc0a80a03, ip: 192:168:10:3  neighrbour 2 id:0xc0a80a04, ip: 192:168:10:4  memory total: 8388580, used:15912, maxused: 15912  STATION:C0A80A01, ExecuteFunctioncode, Functioncode: 3.  toplogy id: 0xc0a80a02, ip: 192:168:10:2  toplogy type: 0x1  toplogy switch num: 0x1  switch 0 id:0xc0a80a02 ip: 192:168:10:2  neighrbour num 3  neighrbour 0 id:0xc0a80a01, ip: 192:168:10:1  neighrbour 1 id:0xc0a80a03, ip: 192:168:10:3  neighrbour 2 id:0xc0a80a04, ip: 192:168:10:4  memory total: 8388580, used:16072, maxused: 16072  STATION:C0A80A02, ExecuteFunctioncode, Functioncode: 2.  STATION:C0A80A01, ExecuteFunctioncode, Functioncode: 2.  STATION:C0A80A02, ExecuteFunctioncode, Functioncode: 3.  FreeTopologyMemory before memory total: 8388580, used:16376, maxused: 16400  FreeTopologyMemory after memory total: 8388580, used:16240, maxused: 16400  toplogy id: 0xc0a80a01, ip: 192:168:10:1  toplogy type: 0x1  toplogy switch num: 0x1  switch 0 id:0xc0a80a01 ip: 192:168:10:1  neighrbour num 3  neighrbour 0 id:0xc0a80a02, ip: 192:168:10:2  neighrbour 1 id:0xc0a80a03, ip: 192:168:10:3  neighrbour 2 id:0xc0a80a04, ip: 192:168:10:4  memory total: 8388580, used:16264, maxused: 16400  STATION:C0A80A01, ExecuteFunctioncode, Functioncode: 3.  FreeTopologyMemory before memory total: 8388580, used:16376, maxused: 16400  FreeTopologyMemory after memory total: 8388580, used:16240, maxused: 16400  toplogy id: 0xc0a80a02, ip: 192:168:10:2  toplogy type: 0x1  toplogy switch num: 0x1  switch 0 id:0xc0a80a02 ip: 192:168:10:2 |
| 4. 发送 |
| 65 A6 1D 00 A6 65 00 00 AA 00 01 1D 00 01 0A A8 C0 01 01 01 0A A8 C0 41 01 03 02 0A A8 C0 03 0A A8 C0 04 0A A8 C0 A9 07 BF 10 1A |
| 65 A6 08 00 A6 65 2C F5 03 0A 02 01 01 0A A8 C0 01 ED A4 1A  65 A6 08 00 A6 65 2C F5 03 0A 02 01 02 0A A8 C0 01 ED E0 1A  65 A6 08 00 A6 65 2C F5 03 0A 02 01 04 0A A8 C0 01 ED 68 1A |
| 65 A6 08 00 A6 65 03 0A 01 0A 02 01 01 0A A8 C0 01 AA 13 1A  65 A6 08 00 A6 65 03 0A 02 0A 02 01 02 0A A8 C0 01 5A 43 1A  65 A6 08 00 A6 65 03 0A 04 0A 02 01 04 0A A8 C0 01 FA E0 1A |
| A6 65 01 0A 02 0A 02 01 02 0A A8 C0 01 E6 C6 1A  65 A6 06 00 A6 65 01 0A 03 0A 02 01 03 0A A8 C0 01 B6 F6 1A 65 A6 06 00 A6 65 01 0A 04 0A 02 01 04 0A A8 C0 01 46 65 1A 65 A6 1F 00 A6 65 02 0A 03 0A 03 01 01 02 0A A8 C0 01 01 02 0A A8 C0 42 01 03 01 0A A8 C0 03 0A A8 C0 04 0A A8 C0 AB 07 00 00 8C F7 1A 65 A6 1F 00 A6 65 01 0A 03 0A 03 01 01 01 0A A8 C0 01 01 01 0A A8 C0 41 01 03 02 0A A8 C0 03 0A A8 C0 04 0A A8 C0 A9 07 00 00 40 A6 1A 65 A6 23 00 A6 65 03 0A 01 0A 03 01 01 03 0A A8 C0 01 01 03 0A A8 C0 43 01 04 01 0A A8 C0 02 0A A8 C0 04 0A A8 C0 05 0A A8 C0 29 09 00 00 CD 2D 1A 65 A6 23 00 A6 65 04 0A 01 0A 03 01 01 04 0A A8 C0 01 01 04 0A A8 C0 45 01 04 01 0A A8 C0 02 0A A8 C0 03 0A A8 C0 05 0A A8 C0 2C 09 00 00 1C CC 1A 65  A6 1F 00 A6 65 02 0A 01 0A 03 01 01 02 0A A8 C0 01 01 02 0A A8 C0 42 01 03 01 0A A8 C0 03 0A A8 C0 04 0A A8 C0 AB 07 00 00 16 F6 1A |
| 5. 挂起的程序 |
| RT\_ETIMEOUT 2 属于超时 |

2018/6/20

|  |
| --- |
| 1.除了最高优先级，其他全部挂起不能执行。 |
| 2.router之上正常之下全部挂起 |
| 修改互斥为rt-thread 依旧不正确。 |
| Residen正好为临界值，就出现死区 |
| 1. 输入list\_thread     0x0F421328 (ucrtbased.dll)处(位于 vs.exe 中)引发的异常。  0xC0000005: 读取位置 0x7268745F 时发生访问冲突。 |
| 4关于配置信息，要求分开存放逐个获取，要么是统一固化存放。 |
| runState 与nextState的区别。 |

2018/6/19

|  |
| --- |
| 1.  VC++  引发了未经处理的异常:读取访问权限冲突。  thread 是 0xFFFFFFEC。  引发了异常: 读取访问权限冲突。  thread 是 0xFFFFFFEC。 |
| \ | /  - RT - Thread Operating System  / | \ 3.1.0 build Jun 14 2018  2006 - 2018 Copyright by rt-thread team  Hello RT-Thread!  File System on root initialized!  File System on sd initialization failed!  thread distribution\_thread\_entry start.  thread simulation start.  thread router start.  thread comself start.  thread communication start.  init usart4.  (rx\_fifo != RT\_NULL) assertion failed at function:rt\_hw\_serial\_isr, l |
| 3. 下发当前配置项时候，与此同时增加站点.并将本站点作为本地站点，可以节省大量判断。 |
| 1. void PrintTopologyMessage(const TopologyMessage\* topologyMessage)   值改变？  void PrintTopologyMessage(const TopologyMessage\* topologyMessage)  {  *uint32\_t* id = topologyMessage->id;  rt\_kprintf("toplogy id: 0x%x, ip: %d:%d:%d:%d\n", id, GET\_N\_BYTE(id, 3), GET\_N\_BYTE(id, 2), GET\_N\_BYTE(id, 1), GET\_N\_BYTE(id, 0));  rt\_kprintf("toplogy type: 0x%x\n", topologyMessage->type);  rt\_kprintf("toplogy switch num: 0x%x\n", topologyMessage->switchNum);  for(*uint8\_t* i = 0; i < topologyMessage->switchNum; i++)  {  id = (topologyMessage->switchCollect[i]).id;  rt\_kprintf("switch %d id:0x%x ip: %d:%d:%d:%d\n",i, id, GET\_N\_BYTE(id, 3), GET\_N\_BYTE(id, 2), GET\_N\_BYTE(id,1), GET\_N\_BYTE(id, 0));  rt\_kprintf("neighrbour num %d \n", (topologyMessage->switchCollect[i]).neighbourNum);  for (*uint8\_t* k = 0; k < (topologyMessage->switchCollect[i]).neighbourNum; k++)  {  id = (topologyMessage->switchCollect[i]).neighbourCollect[k];  rt\_kprintf("neighrbour %d id:0x%x, ip: %d:%d:%d:%d\n",k, id, GET\_N\_BYTE(id, 3), GET\_N\_BYTE(id, 2), GET\_N\_BYTE(id, 1), GET\_N\_BYTE(id, 0));  }  }  *uint32\_t* total;  *uint32\_t* used;  *uint32\_t* maxused;  rt\_memory\_info(&total, &used, &maxused);  rt\_kprintf("memory total: %d, used:%d, maxused: %d\n", total, used, maxused);  } |
| 5. 试了若干方法不行，然后改mdk开发，mdk 编译器错误比vc设置的严格。 |
| 6.全部转码为UTF-8 目前为止一切正常。 |

2018/6/16

|  |
| --- |
| 1. 应对extern 执行更为严格的参数检查，static是以前为保证。 |
| 2.需要在初始化时，增加站点就立即设置本站点信息，方便其它引用。需要提供本拓扑节点所有信息，主要包括邻居信息。 |
| switch 0 id:0xc0a80a01 ip: 192:168:10:1  neighrbour num 3  neighrbour 0 id:0xc0a80a02, ip: 192:168:10:2  neighrbour 1 id:0xc0a80a03, ip: 192:168:10:3  neighrbour 2 id:0xc0a80a04, ip: 192:168:10:4  memory total: 32744, used:7312, maxused: 7312  FreeTopologyMemory before memory total: 32744, used:7288, maxused: 7312  FreeTopologyMemory after memory total: 32744, used:7188, maxused: 7312  toplogy id: 0xc0a80a01, ip: 192:168:10:1  FaultRemovalStateCenter ERROR : 0toplogy type: 0x1  FaultRemovalStateCenter ERROR : 0toplogy switch num: 0x1  FaultRemovalStateCenter ERROR : 0switch 0 id:0x23232323 ip: 35:35:35:35  FaultRemovalStateCenter ERROR : 0neighrbour num 35  FaultRemovalStateCenter ERROR : 0psr: 0x21000000  pc: 0x08002498  lr: 0x08004bab  r12: 0xfffffffd  r03: 0x00000001  r02: 0x20000054  r01: 0x00000023  r00: 0x23232323  hard fault on thread: comself  首次即错误 |
| toplogy id: 0xc0a80a01, ip: 192:168:10:1  toplogy type: 0x1  toplogy switch num: 0x1  switch 0 id:0x23232323 ip: 35:35:35:35  neighrbour num 35  psr: 0x21000000  pc: 0x08002498  lr: 0x08004ba7  r12: 0xfffffffd  r03: 0x00000001  r02: 0x20000054  r01: 0x00000023  r00: 0x23232323  hard fault on thread: comself |
| 应该说，MDK和VC++ 是同样的问题 |

2018/6/12

|  |
| --- |
| 1. 上位机问题   下发相应设置时出现。    经排查发现    此处为null. |
|  |

|  |
| --- |
| 2.通讯下发死机问题 |
| thread communication start.  thread blink start.  psr: 0x01000000  pc: 0x2b3cf4dc  lr: 0x08002261  r12: 0x00c00000  r03: 0x20002720  r02: 0x2b3cf4dd  r01: 0x000000aa  r00: 0x20003538  hard fault on thread: communic€      初始化不正常。 |
|  |
| 2.MEMSET 问题malloc后需要清零，防止意外 |
|  |
| 按照自定义协议，一旦出错，后面将全部出错，地址容易混用。  在使用101和自定义之间暂时使用自定义 |

2018/6/11

|  |
| --- |
| 1.psr: 0x41000000  pc: 0x0800315c  lr: 0x08003157  r12: 0x20003dd0  r03: 0x00000bc8  r02: 0x00000003  r01: 0x4d9f7a89  r00: 0x20003ee8  hard fault on thread: main    /RemovalHandleInit(handle, station->topology.localTopology->switchCollect);  此部分未初始化，不正确 |
| thread communication start.  psr: 0x01000000  pc: 0x08003384  lr: 0x08002a6d  r12: 0x00000000  r03: 0x00000001  r02: 0x20000058  r01: 0x10000000  r00: 0x09478048  hard fault on thread: switch  ListElmt\* element = list\_head(list);  uint8\_t size = list\_size(list);  【此处应该重新幅值】  //循环更新模拟开关状态  do  {  for (uint8\_t i = 0; i < size; i++)  {  SimulationSation\* station = (SimulationSation\*)(element->data);  if (station != NULL)  {  SwitchRunStateSimulation(station);  }  else  {  rt\_kprintf("imulationSation\* station = NULL.\n");  break;  }  element = element->next;  }  rt\_thread\_delay(1);  }while(1); |
|  |
| thread communication start.  manger->router.FindMemberById ERROR : 11g\_VirtualNode is NULL, ERROR : 11thread communication start.  init usart4.  thread simulation start.  thread router start.  thread communication start.  psr: 0x20000000  pc: 0xbeef0000  lr: 0x08000c35  r12: 0x00000000  r03: 0x00000001  r02: 0xbeef0000  r01: 0x200022f0  r00: 0x34761ec6  hard fault on thread: comself    未有效初始化 |

2018/6/2

|  |
| --- |
| 1.指针幅值不正确 |
| if (areaNum++ < POWER\_AREA\_NUM)  {  listArea += areaNum;  }  else  {  return ERROR\_OVER\_LIMIT;  }  }  }  while((element = element->next) != NULL);  DistributionAreaDeal\_SpecialDeal(local->switchCollect, listArea, areaNum);  上述两处冲突 |
|  |

2018/6/1

|  |
| --- |
| 1. 内存释放过程中的不一致现象，   内存大小应该相等，但是经过一番释放后增加了4个字节。 |
| \0  \ | /  - RT - Thread Operating System  / | \ 3.0.3 build May 31 2018  2006 - 2018 Copyright by rt-thread team  thread communication start.  init usart4.  thread blink start.  toplogy id: 0xc0a80a01, ip: 192:168:10:1  toplogy type: 0x1  toplogy switch num: 0x1  switch 0 id:0xc0a80a01 ip: 192:168:10:1  neighrbour num 3  neighrbour 0 id:0xc0a80a02, ip: 192:168:10:2  neighrbour 1 id:0xc0a80a03, ip: 192:168:10:3  neighrbour 2 id:0xc0a80a04, ip: 192:168:10:4  memory total: 32744, used:92, maxused: 400  toplogy id: 0xc0a80a02, ip: 192:168:10:2  toplogy type: 0x1  toplogy switch num: 0x1  switch 0 id:0xc0a80a02 ip: 192:168:10:2  neighrbour num 3  neighrbour 0 id:0xc0a80a01, ip: 192:168:10:1  neighrbour 1 id:0xc0a80a03, ip: 192:168:10:3  neighrbour 2 id:0xc0a80a04, ip: 192:168:10:4  memory total: 32744, used:208, maxused: 400  toplogy id: 0xc0a80a03, ip: 192:168:10:3  toplogy type: 0x1  toplogy switch num: 0x1  switch 0 id:0xc0a80a03 ip: 192:168:10:3  neighrbour num 4  neighrbour 0 id:0xc0a80a01, ip: 192:168:10:1  neighrbour 1 id:0xc0a80a02, ip: 192:168:10:2  neighrbour 2 id:0xc0a80a04, ip: 192:168:10:4  neighrbour 3 id:0xc0a80a05, ip: 192:168:10:5  memory total: 32744, used:328, maxused: 400  toplogy id: 0xc0a80a04, ip: 192:168:10:4  toplogy type: 0x1  toplogy switch num: 0x1  switch 0 id:0xc0a80a04 ip: 192:168:10:4  neighrbour num 4  neighrbour 0 id:0xc0a80a01, ip: 192:168:10:1  neighrbour 1 id:0xc0a80a02, ip: 192:168:10:2  neighrbour 2 id:0xc0a80a03, ip: 192:168:10:3  neighrbour 3 id:0xc0a80a0f, ip: 192:168:10:15  memory total: 32744, used:448, maxused: 448【后面重复删除应该一样】  toplogy id: 0xc0a80a04, ip: 192:168:10:4  toplogy type: 0x1  toplogy switch num: 0x1  switch 0 id:0xc0a80a04 ip: 192:168:10:4  neighrbour num 4  neighrbour 0 id:0xc0a80a01, ip: 192:168:10:1  neighrbour 1 id:0xc0a80a02, ip: 192:168:10:2  neighrbour 2 id:0xc0a80a03, ip: 192:168:10:3  neighrbour 3 id:0xc0a80a0f, ip: 192:168:10:15  memory total: 32744, used:568, maxused: 568  FreeTopologyMemory before memory total: 32744, used:544, maxused: 568  FreeTopologyMemory after memory total: 32744, used:448, maxused: 568  toplogy id: 0xc0a80a01, ip: 192:168:10:1  toplogy type: 0x1  toplogy switch num: 0x1  switch 0 id:0xc0a80a01 ip: 192:168:10:1  neighrbour num 3  neighrbour 0 id:0xc0a80a02, ip: 192:168:10:2  neighrbour 1 id:0xc0a80a03, ip: 192:168:10:3  neighrbour 2 id:0xc0a80a04, ip: 192:168:10:4  memory total: 32744, used:568, maxused: 568  FreeTopologyMemory before memory total: 32744, used:544, maxused: 568  FreeTopologyMemory after memory total: 32744, used:452, maxused: 568  toplogy id: 0xc0a80a02, ip: 192:168:10:2  toplogy type: 0x1  toplogy switch num: 0x1  switch 0 id:0xc0a80a02 ip: 192:168:10:2  neighrbour num 3  neighrbour 0 id:0xc0a80a01, ip: 192:168:10:1  neighrbour 1 id:0xc0a80a03, ip: 192:168:10:3  neighrbour 2 id:0xc0a80a04, ip: 192:168:10:4  memory total: 32744, used:568, maxused: 568  FreeTopologyMemory before memory total: 32744, used:544, maxused: 568  FreeTopologyMemory after memory total: 32744, used:452, maxused: 568  toplogy id: 0xc0a80a03, ip: 192:168:10:3  toplogy type: 0x1  toplogy switch num: 0x1  switch 0 id:0xc0a80a03 ip: 192:168:10:3  neighrbour num 4  neighrbour 0 id:0xc0a80a01, ip: 192:168:10:1  neighrbour 1 id:0xc0a80a02, ip: 192:168:10:2  neighrbour 2 id:0xc0a80a04, ip: 192:168:10:4  neighrbour 3 id:0xc0a80a05, ip: 192:168:10:5  memory total: 32744, used:572, maxused: 572  FreeTopologyMemory before memory total: 32744, used:548, maxused: 572  FreeTopologyMemory after memory total: 32744, used:452, maxused: 572 |
|  |

2018/5/31

|  |
| --- |
| 1.注意配置项的最大长度 |
| 2.添加邻居信息写了两遍。。。。！！！ |
| 3.需要注意处理，功能码有了，配套数据为空的情况！ |

2018/5/30

|  |
| --- |
| 1.error begin address 0x20002490, and end address 0x20002000 |
| #define SRAM\_END (HEAP\_BEGIN + SRAM\_SIZE \* 1024)//(0x20000000 + SRAM\_SIZE \* 1024) |
| 2 mem: 0x2000248c, used flag: 2048, magic code: 0x2df7  ighrboor 2 id:-1062729212, ip: 192:168:10:4  to free a bad data block:  mem: 0x2000248c, used flag: 2048, magic code: 0x2df7  psr: 0x81000200  pc: 0x08001ab6  lr: 0x08001d35  r12: 0x00000032  r03: 0x200024a0  r02: 0x20002498  r01: 0x2800528e  r00: 0x2000248c  hard fault on thread: communic€ |
| FREE((\*topology)->switchCollect->neighbourCollect);  (\*topology)->switchCollect->neighbourCollect = 0;  FREE((\*topology)->switchCollect);  (\*topology)->switchCollect = 0;  FREE(\*topology);  \*topology = 0; |
| 3. toplogy id: 4660, ip: 0:0:18:52 ID赋值始终不正确 |
| 上电复位 |
| 1. 下载一次定值 |
| 应该相等实际没有相等  re\_topology = (TopologyMessage\* )MALLOC(sizeof(TopologyMessage));  \*topology = re\_topology ;    if( \*topology == NULL) //简单的指针检测  {  return ERROR\_MALLOC;  }  MEMSET(\*topology, 0, sizeof(TopologyMessage)); |
|  |
| 解决方法  TopologyMessage\* topologyMessage;  TopologyMessage\* tempTopology;  不应该为  TopologyMessage\*\* topologyMessage;  TopologyMessage\*\* tempTopology;  二重指针改变指向，在函数内部需要3重指针！ |
| 这也把赋值不正确的问题予以解决。看似是不相关的两个问题，实际上是一个。 |
| 1. typedef unsigned int uint32\_t; stdint.h   typedef unsigned long rt\_uint32\_t; /\*\*< 32bit unsigned integer type \*/  以上两种定义不一致。  经实测  sizeof(uint32\_t): 4, sizeof(rt\_uint32\_t):4 |
| 1. 赋值不正确   list\_ins\_next(&NeighborTopologyMessageList, NULL, &topologyMessage);  PrintTopologyMessage((TopologyMessage\*)(list\_head(&NeighborTopologyMessageList)->data));  输出为乱码。  后仔细观察  list\_rem\_next(&NeighborTopologyMessageList, element, (void\*\*)&tempTopology);//移除  FreeTopologyMemory(&tempTopology);//释放  list\_ins\_next(&NeighborTopologyMessageList, element, topologyMessage);  int list\_rem\_next(List \*list,ListElmt \*element,void \*\*data)  int list\_ins\_next(List \*list, ListElmt \*element, const void \*data)  “一味地”未仔细注意其中的区别，直接强制转换使用。 |
|  |
| 5.内存分配跟踪 |
| \ | /  - RT - Thread Operating System  / | \ 3.0.3 build May 30 2018  2006 - 2018 Copyright by rt-thread team  thread communication start.  init usart4.  thread blink start.  toplogy id: 0xc0a80a01, ip: 192:168:10:1  toplogy type: 0x1  toplogy switch num: 0x1  switch 0 id:0xc0a80a01 ip: 192:168:10:1  neighrboor num 3  neighrboor 0 id:0xc0a80a02, ip: 192:168:10:2  neighrboor 1 id:0xc0a80a03, ip: 192:168:10:3  neighrboor 2 id:0xc0a80a04, ip: 192:168:10:4  memory total: 32744, used:92, maxused: 400【首次下发】  toplogy id: 0xc0a80a01, ip: 192:168:10:1  toplogy type: 0x1  toplogy switch num: 0x1  switch 0 id:0xc0a80a01 ip: 192:168:10:1  neighrboor num 3  neighrboor 0 id:0xc0a80a02, ip: 192:168:10:2  neighrboor 1 id:0xc0a80a03, ip: 192:168:10:3  neighrboor 2 id:0xc0a80a04, ip: 192:168:10:4  memory total: 32744, used:208, maxused: 400【2次 重复下发】  FreeTopologyMemory before memory total: 32744, used:184, maxused: 400  FreeTopologyMemory after memory total: 32744, used:92, maxused: 400【恢复92】  toplogy id: 0xc0a80a01, ip: 192:168:10:1  toplogy type: 0x1  toplogy switch num: 0x1  switch 0 id:0xc0a80a01 ip: 192:168:10:1  neighrboor num 3  neighrboor 0 id:0xc0a80a02, ip: 192:168:10:2  neighrboor 1 id:0xc0a80a03, ip: 192:168:10:3  neighrboor 2 id:0xc0a80a04, ip: 192:168:10:4  memory total: 32744, used:208, maxused: 400【3次 重复下发】  FreeTopologyMemory before memory total: 32744, used:184, maxused: 400  FreeTopologyMemory after memory total: 32744, used:92, maxused: 400【恢复92】 |
| toplogy id: 0xc0a80a02, ip: 192:168:10:2  toplogy type: 0x1  toplogy switch num: 0x1  switch 0 id:0xc0a80a02 ip: 192:168:10:2  neighrboor num 3  neighrboor 0 id:0xc0a80a01, ip: 192:168:10:1  neighrboor 1 id:0xc0a80a03, ip: 192:168:10:3  neighrboor 2 id:0xc0a80a04, ip: 192:168:10:4  memory total: 32744, used:208, maxused: 400【第2组 第1次下发】  toplogy id: 0xc0a80a02, ip: 192:168:10:2  toplogy type: 0x1  toplogy switch num: 0x1  switch 0 id:0xc0a80a02 ip: 192:168:10:2  neighrboor num 3  neighrboor 0 id:0xc0a80a01, ip: 192:168:10:1  neighrboor 1 id:0xc0a80a03, ip: 192:168:10:3  neighrboor 2 id:0xc0a80a04, ip: 192:168:10:4  memory total: 32744, used:324, maxused: 400  FreeTopologyMemory before memory total: 32744, used:300, maxused: 400  FreeTopologyMemory after memory total: 32744, used:208, maxused: 400【第2组 第2次下发】  toplogy id: 0xc0a80a02, ip: 192:168:10:2  toplogy type: 0x1  toplogy switch num: 0x1  switch 0 id:0xc0a80a02 ip: 192:168:10:2  neighrboor num 3  neighrboor 0 id:0xc0a80a01, ip: 192:168:10:1  neighrboor 1 id:0xc0a80a03, ip: 192:168:10:3  neighrboor 2 id:0xc0a80a04, ip: 192:168:10:4  memory total: 32744, used:324, maxused: 400  FreeTopologyMemory before memory total: 32744, used:300, maxused: 400  FreeTopologyMemory after memory total: 32744, used:208, maxused: 400【第2组 第3次下发】 |
| toplogy id: 0xc0a80a01, ip: 192:168:10:1【第3组重新下载第1组】  toplogy type: 0x1  toplogy switch num: 0x1  switch 0 id:0xc0a80a01 ip: 192:168:10:1  neighrboor num 3  neighrboor 0 id:0xc0a80a02, ip: 192:168:10:2  neighrboor 1 id:0xc0a80a03, ip: 192:168:10:3  neighrboor 2 id:0xc0a80a04, ip: 192:168:10:4  memory total: 32744, used:324, maxused: 400  toplogy id: 0xc0a80a01, ip: 192:168:10:1  toplogy type: 0x1  toplogy switch num: 0x1  switch 0 id:0xc0a80a01 ip: 192:168:10:1  neighrboor num 3  neighrboor 0 id:0xc0a80a02, ip: 192:168:10:2  neighrboor 1 id:0xc0a80a03, ip: 192:168:10:3  neighrboor 2 id:0xc0a80a04, ip: 192:168:10:4  memory total: 32744, used:440, maxused: 440  FreeTopologyMemory before memory total: 32744, used:416, maxused: 440  FreeTopologyMemory after memory total: 32744, used:324, maxused: 440  【理论上讲，重新下载第1组，内存占用总量应该不变】  后整理与查找程序，有12字节摆动，其他一切正常。 |

2018/5/28

|  |
| --- |
| 1.串口对应 面对板子正面，5.08端子朝上，从左到右 |
| 1. U4- PC10/PC11 ---U14(芯片) 2. U5- PC12/PD2-----U21（芯片） 3. U6-PC6(T)/PC7(R) ---U27(芯片)（新） |
| 以前 |

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);  } |
|  |