1. 在MiniTask中IPC和RPC何时释放内存？

* **IPC：**在MiniTask中使用到了IPC通信方式的两种：同步IPC和异步IPC（在Minitask中，story3的平滑使用异步IPC，story5的热拔插功能使用同步IPC）

IPC同步发送与异步发送的**区别**在于发送方**释放内存的代码位置**也就是**时机不同**，但也是遵循谁申请谁释放的原则：

• 异步发送时，IPC\_Send发送**失败**，**就地**释放内存，发送**成功**后就不管了，接收到发送完成的消息时，Minitask在任务队列中由**发送方**释放该内存

• 同步发送时，在IPC\_Send返回时（无论发送成功与否），发送方就地释放内存，同步体现在必须**等待**IPC\_Send**返回**

* **RPC：**在MiniTask的story2中，主控下发命令至接口板使用的是**同步RPC，**这个涉及到**发送消息**和**回应消息**内存释放的问题：

• 发送消息：在Mini\_CHB\_SendCmd函数里面，使用IPC\_RPC\_AllocMsg分配内存，消息内存由RPC模块释放，也就是函数IPC\_RPC\_SyncCall调用完后，内存即被释放

• 回应消息：申请内存由**接收方**申请（具体体现在*Mini\_LCL\_LpuCmdDispatch()*），由RPC模块释放**接收方申请**的内存，当回应消息pstRcvData指向的内存非空时，处理完回应消息后（DealRpcRcv函数之后），由回调函数中（Mini\_CHB\_SendCmd中）释放该内存块（该内存块是RPC模块发送数据时的内存）

1. 使用消息队列和回调函数的区别及优缺点？

创建IPC通道可以使用消息队列或者回调函数的方式，只是两种方式之后的处理方式有稍微的区别；IPC模块**发送结束通知**和**接受报文通知**都采用了**任务机制**，如果采用了回调函数的方式，则通知处理过程的任务占用了IPC的任务。IPC是公用模块，IPC任务被占用必然导致不能及时处理其他模块的IPC请求，导致IPC性能的下降。

创建IPC通道：1. 回调函数中**只进行必要的处理**，并且尽可能的**简单**；2. 非必要处理部分通过**消息队列的方式**交给模块自己的任务处理。

1. 对于同步和异步IPC的消息码和控制码有区别吗，它们分别什么作用？

消息码是区分不同消息的凭证，也是唯一ID，模块根据不同的消息码，进入不同的处理分支，比如IPC报文头中的不同IPC通信消息码，根据消息码进入分支，然后进行处理。

1. 产生了消息码必须要回应吗？

如果是异步，丢失了，可以不用回应。

1. 命令行的实现机制，比如Set命令，它的流程是怎么样的，它的数据流怎么传递，注册完之后系统大概是怎么处理的？

主要**流程**是：用户输入命令行 –> 命令行解析 –> 将参数打包 –> 模块获取参数处理

若是set命令，在参数解析完之后，将对应的数组的值进行设置，也就是流程的最后一步；注册完之后，根据注册模板，将命令存入命令池模板，通过关键字进行调用相应的命令。

1. 思考下一个完整的交换机的系统需要什么story功能，为什么需要这些功能，客户有这样的需求吗？还是伪需求？

比如Buildrun功能，当用户配置完常用配置之后，如果遇到意外情况，主控断电或者重启，这时候，客户就需要Buildrun功能，恢复以前的保存的**配置**

1. MiniTask中，用了很多注册，不管是命令行，还是RPC、IPC，请问这些注册是怎么实现的，注册这个动作在系统中意味着什么？

注册在我理解的过程，应该是系统分配相应的资源，如IPC通道的注册，若使用的是消息队列的方式，会分配消息队列资源；注册相当于为在线程池中分配相应的线程和系统资源。

1. 如果未释放IPC、或RPC申请的内存，会发生什么后果（除了内存泄漏）？

如果未释放申请的内存，极端情况会使IPC、RPC模块失效，造成通信阻塞，最后造成系统卡死崩溃，重启，造成严重后果。

1. IPC中使用到了消息队列，请问消息任务是怎么切换的？

在IPC消息队列中，是通过循环VOS\_Que\_Read读取队列中的消息，如果有消息，进入消息处理分支；而IPC任务是在一个死循环中，当任务启动时，独占CPU，并且不支持任务抢占；因此必须通过放权，释放CPU资源。

***Prototype：VOID VOS\_T\_RunDelay(ULONG ulRunMillisecs, ULONG ulDelayMillisecs)***

在for循环中***VOS\_T\_RunDelay(100, 1000);***是运行100微秒，放权1000微秒（释放资源）。

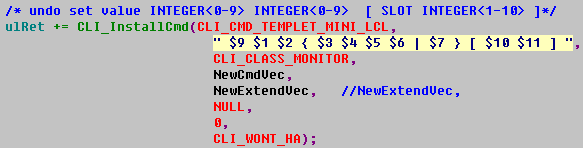
1. 讲讲补丁实现的机制，冷补丁和热补丁的区别，为什么会有这种区别，以及什么场景下分别使用冷补丁和热补丁？

**实现机制：**当前系统的补丁的做法是根据需要打补丁函数的符号名（也就是函数指针）找到其地址，然后修改此函数开始部分的指针（也就是函数的地址），让其跳转到对应的补丁函数位置去执行。

**区别：**冷补丁与热补丁最大的区别是对业务的影响大小，冷补丁需要reboot，在交换机中，需要执行命令reboot fast；热补丁不需要，只需要patch \*\* all run。冷补丁是一个bin文件，即一个可执行文件，和基础版本没有本质上的区别。所以冷补丁升级，其实是把老版本的bin文件替换，加载新bin文件的过程，只有复位才能做到新bin文件的加载运行。热补丁不是一个可执行文件。

**使用场景：**对于要求长时间不间断工作的设备，当发现软件有缺陷时，在不中断业务即不需要重启软件模块的情况下用新代码来替换旧代码来解决这些缺陷，使用热补丁，在minitask中可以体现的部分是不在init初始化中的函数。为了解决软件已知缺陷，通过新文件覆盖旧文件或新增文件，需要中断业务并重启软件模块来解决这些缺陷或者实现新需求，使用冷补丁，需要冷补丁的是在整个任务起来时，只初始化依次的函数，也就是在init中的函数。

1. 讲讲命令安装函数中参数代表的含义，CLI\_WONT\_HA是什么含义？



**第一个参数***CLI\_CMD\_TEMPLET\_MINI\_LCL*是命令模板，也就是所注册的视图范围第二个参数是所注册的线索表达式；

第三个参数是命令的优先级和级别

第四、五个参数表示的是命令的描述向量和拓展向量，也就是在CLI\_NewDefineCmdElement（）中申请的向量

最后一个参数CLI\_WONT\_HA表示的是不会触发系统本身集成的HA功能，比如实备，批备等，CLI\_WILL\_HA刚好相反，会自动触发相应的功能。

1. HA作用响应角色是谁，比如热拔插？HotNotify()这个函数的作用是什么？

**热插拔是当有接口板插入时**，主控板将数据下发到接口板，所以响应角色是主控板，平滑是将接口板的数据取出然后主控板进行处理，比对数据，如果出现数据不一致的情况下，发出警告，所以响应角色依旧是主控板；对于批备来讲，收到HA模块发来的批备信息（备板插入触发），主控板处理，将数组值批量备份到备板，备板收到消息后，将数组值拷贝到自己的数组中，响应角色是主控板和备板，对于时备来讲，主板设置数组信息，调用realtime backsend，备板收到实备消息后，进行处理将数据拷贝到自己数组，因此响应角色是备板。

1. 在做Minitask中出现问题，你是怎么定位的？

**Minitask代码比较偏底层**，代码不能放vs，一步步往下走debug，刚开始很不习惯，后面和师父和周围的同事交流，也有专门的debug方式，用的最多的是打log，但由于还没实现功能，师父建议我先使用断言定位问题。

刚开始的第一种方式是，打断言，然后利用callstack找出断点及异常点，再进行排查代码块问题。

然后做了story4，Debugging功能后，就使用Debugging功能，在响应的异常及想要看到的变量的值答应出来，看是否异常，这就相当于在程序里面加printf看了，而且现在信息中心的log模块比较完备，支持event、trap和log级别的信息。

1. 平滑启动以后，当新主板和接口板信息不一致时，以哪个信息为准

答辩时回答是以接口板为准，因为新主板起来以后，其实是不清楚新主板的信息是否准确，而接口板上信息因为热插拔存在的原因，应该和主板的数据保持一致。答辩后私下讨论发现应该以实际情况为准，及需要新主板数据还是接口板数据，两者数据都能够成为实际数据，而正确数据应该还是老主板数据。

1. ulRelt = IPC\_Create(TID\_IPC\_MINI\_WHW, IPC\_NOTI\_VIA\_QUEUE, g\_WHW\_ulMiniQueId, NULL);函数中各个参数的作用。

TID\_IPC\_MINI\_WHW 公共模块中所注册的消息通道号、

IPC\_NOTI\_VIA\_QUEUE 表示以消息队列方式传递信息(问题可延伸至消息队列和回调的区别)、

g\_WHW\_ulMiniQueId 消息队列号、

NULL 回调函数