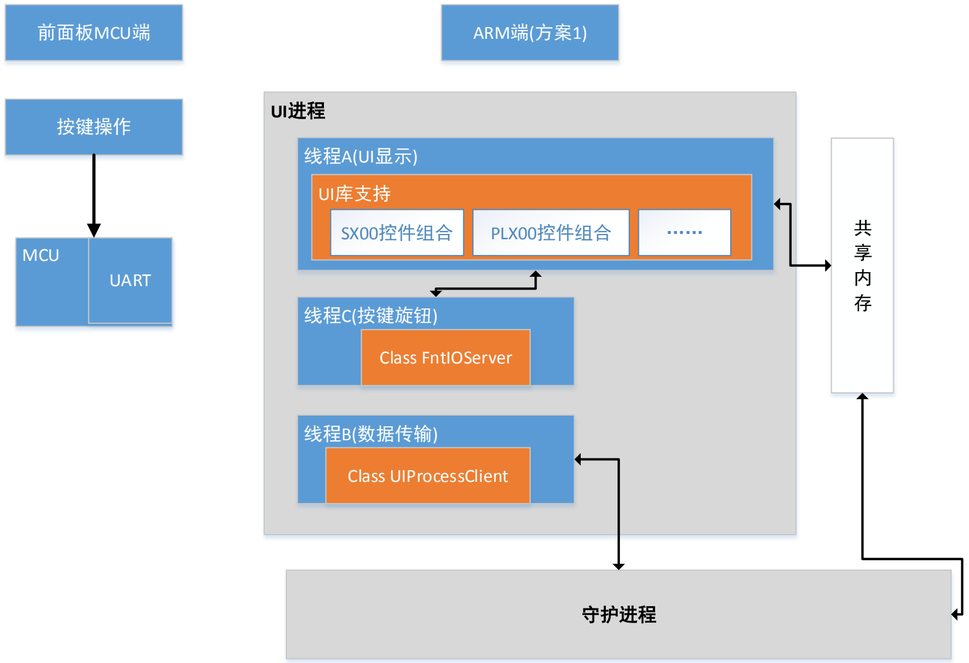

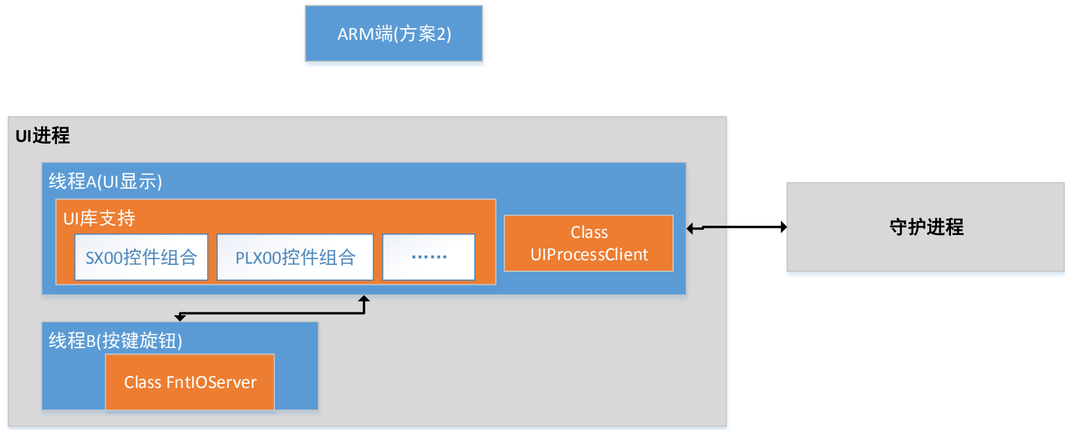

方案1设计思路：

进程使用多线程方案，分为A、B、C三个线程。A线程负责UI展示所有功能；B线程负责SCPI缓存和数据传输所有功能；C线程负责前面板与ARM之间的IO所有功能。

多线程方案能保障各功能模块独立运行互不干扰，也能保障进程运行的实时性，各模块的错误不会引发其他模块无法工作导致进程无法工作。

UI线程所有下发指令数据通过线程C给守护进程，所有需要实时获取的数据直接从共享内存中获取，提高数据显示实时性，守护进程将所有需要展示给UI的数据直接写入共享内存。

C线程主要用来给UI进程提供与守护进程传输格式数据的支持，方便后期拓展和协议数据的传输，对于所有UI显示和实时性较高的数据均通过共享内存交互。

线程必要性阐述：

线程A：因为UI界面应该独立于任何非UI的操作，且也不应该被任何非UI操作阻塞，UI的阻塞也不应该影响其他逻辑的正常执行，因此需要单独运行在独立线程中。

线程B：对于UI界面产生的业务处理和数据处理操作应该单独执行，保障UI界面的流畅性和处理的实时性；作为和守护进程通信的客户端也应该实时接收来自守护进程的数据，不应该因其他UI操作或界面阻塞而导致数据丢失。

线程C：线程C中涉及到独立IO，因此需要单独运行在线程中，时刻监听IO数据并处理。该线程不能合并至A线程中，因为UI可能阻塞；线程不能合并至B线程中，线程B可能频繁跟UI交互导致线程C通信不及时。

方案优点：

各线程功能独立，线程之间耦合度低，容易拓展和根据需求改变结构。

方案缺点：

UI和守护进程间通信涉及到多线程之间数据交互，需要借助共享内存和信号槽，在信号槽机制下数据传输时延大

对于实时性高的应用功能无法满足，且对于界面联动响应不好。

方案2设计思路：

进程使用多线程方案，分为A、B两个个线程。A线程负责UI展示所有功能；B线程负责前面板与ARM之间的IO所有功能。

多线程方案能保障各功能模块独立运行互不干扰，也能保障进程运行的实时性，各模块的错误不会引发其他模块无法工作导致进程无法工作。

A线程中所有数据支持由本线程中通信模块(UIProcessClient)支撑，直接与守护进程进行数据交互并通过UI展示，能提高数据实时性。

A线程中通信模块主要是指通过与守护进程建立socket通信，专门负责UI线程数据与守护进程之间的数据传输。

线程必要性阐述：

线程A：因为UI界面应该独立于任何非UI的操作，且也不应该被任何非UI操作阻塞，UI的阻塞也不应该影响其他逻辑的正常执行，因此需要单独运行在独立线程中。线程A中加入通信模块，负责与守护进程通信。

线程B：线程B中涉及到独立IO，因此需要单独运行在线程中，时刻监听IO数据并处理。该线程不能合并至A线程中，因为UI可能阻塞。

方案优点：

在UI线程中集成通信模块，能保证UI和守护进程的通信时延较低，提高UI运行实时性。

方案缺点：

UI线程中功能复杂，包含除UI之外的通信功能，可能导致需求拓展和更改较复杂。线程之间功能不独立，耦合度较高。

备注：

此方案中没有直接将UI线程中的通信模块换成共享内存，让守护进程直接与UI全部通过共享内存交互，而不使用通信模块是考虑到UI进程可能与守护进程之间有不同数据格式的通信，或者数据通信有顺序执行且不被覆盖的可能，共享内存无法实现格式数据的缓存，因此使用通信模块代替纯共享内存方式。

结论：基于组内讨论，方案2相对于方案1更适合当前产品的拓展和维护，因此UI进程最终选用方案2作为最终结论。