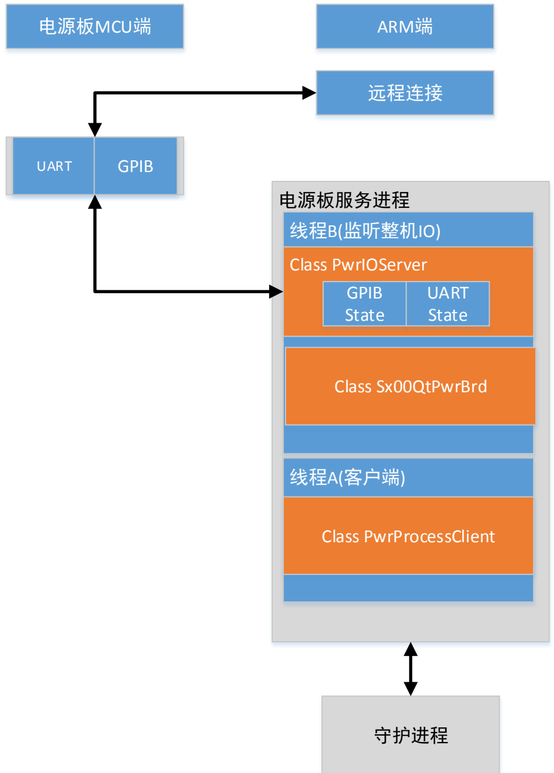

设计思路：

进程使用多线程方案，分为A、B两个线程。A线程负责电源板服务中电源板相关业务处理和SCPI指令传输相关所有功能；B线程负责电源板与ARM端IO监听服务相关所有功能。

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。

线程必要性阐述：

线程A：因为主要负责电源板业务处理和与守护进程进行数据传输，单独运行在线程中能保障数据传输和处理的实时性，也不会阻塞电源板与ARM的IO服务。

线程B：处理ARM与电源板IO服务，涉及到实际IO，需要单独运行在线程中保障IO服务的实时性。线程不能合并至线程A中，因为B线程中处理守护进程的数据时不应该阻塞电源板IO数据，同时电源板IO传输也不应该阻塞B线程中电源板业务的处理。

```
class PwrBoard
{
  public:

        SetTemp();

        GetTemp();

        SetFan();

  private:

        double temp;
        int fan;
};
```

```
class PwrTransManage
{
  public:

        SetUartTrans();

        SetGPIBTrans();

        GetTransIO();

  private:
        bool uartTrans;
};
```

```
class PwrDataParse
{
  public:
        Parse();
/* read from deamon */
        Read();

/* write to deamon */
        Write();

  signals:
        /* to user */
        Transparent();
};
```

```
class PwrIOServer
{
  public:
/* read from user */
        Read();
/* write to user */
        Write();

        Close();

        Listen();

  signals:
        Parse();
};
```

```
main()
{
        /* 当前主线程为线程A */
        pthread_t pidB;
        pthread_create(&pidB);

        connect_to_deamon();
        while(1)
        {
                PwrDataParse.Read();
                Parse();
                PwrDataParse.Write();
        }
        pthread_join(pidB);
}
```

```
threadB()
{
        Listen();

        while(1)
        {
                /* read from user */
                Read();

                emit DtaParse.Parse();

                /* write to user */
                Write();
        }
}
```