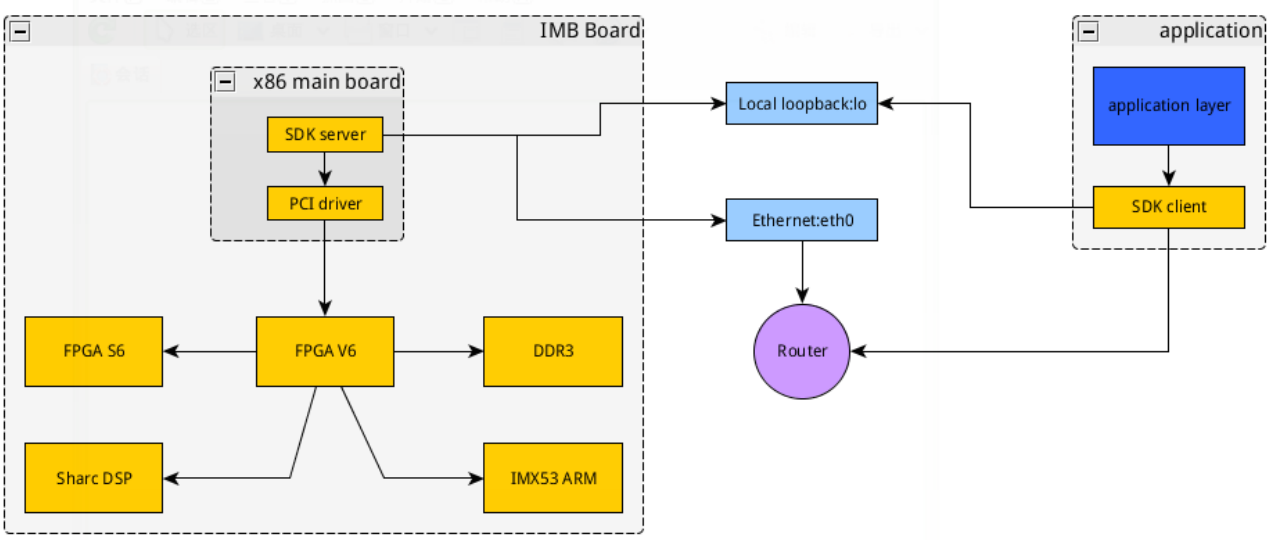
IMB 板卡 SDK 类库开发技术交流

辰星科技-电影终端部
张绍言
2014/6/1

一、应用场景：本地+网络



1、文档分析

根据 MikroM 公司的文档中列出的类名，得知 SDK 需要支持两种通信功能：

一种是本地通信，即调用 SDK 的应用程序部署在 IMB 板卡上，对应于文档中的 MvcDevice;

另一种是网络通信，即调用 SDK 的应用程序部署在随意一台 PC 机上，通过网络与 IMB 板卡通信进行远程控制及音视频数据传输，对应于文档中的 MvcNetDeviceIterator，该类需要提供 IMB 板卡的 IP 地址。

2、实物测试

通过对老外板卡的实物测试，发现老外的 linux 上开放了 ssh 服务，通过 ssh 客户端登录后，查后系统进程发现/usr/sbin/mvc2ipserver 进程，该进程就是板卡的服务端进程，以守护进程存在。

3、架构仿制

为了最大限度的适应(靠拢)MikroM 公司的规范，我将 SDK 的开发，分成两部分，一是 SDK 的服务器端，脱离 SDK 单独部署在 x86 CentOS 下;二是 SDK 的标准类库，该套类库就跟 MikroM 公司的文档《MVC20x API Programming interface for MVC 20x media block.pdf》最大程度的一致。

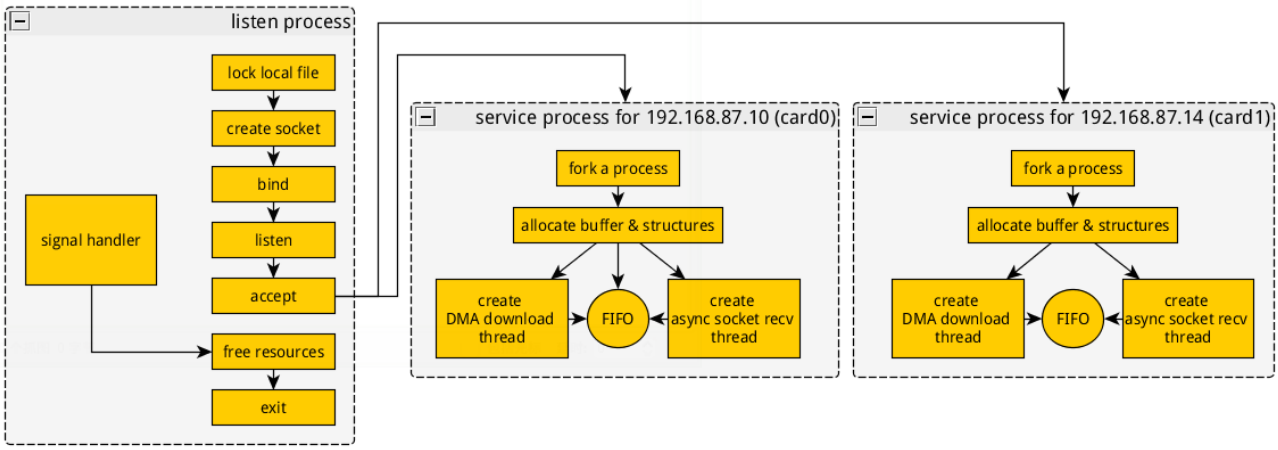
4、通信说明

本地通信：server 与 client 之间通过本地回环接口 lo 进行通信;

网络通信：server 与 client 之间通过以太网接口 eth0 进行通信。

（得益于 OS 提供的 lo 接口，使得 server 与 client 可在同一机器上运行。使得我的工作简化了，可将 2 种应用场景合二为一。）

二、服务端总体架构

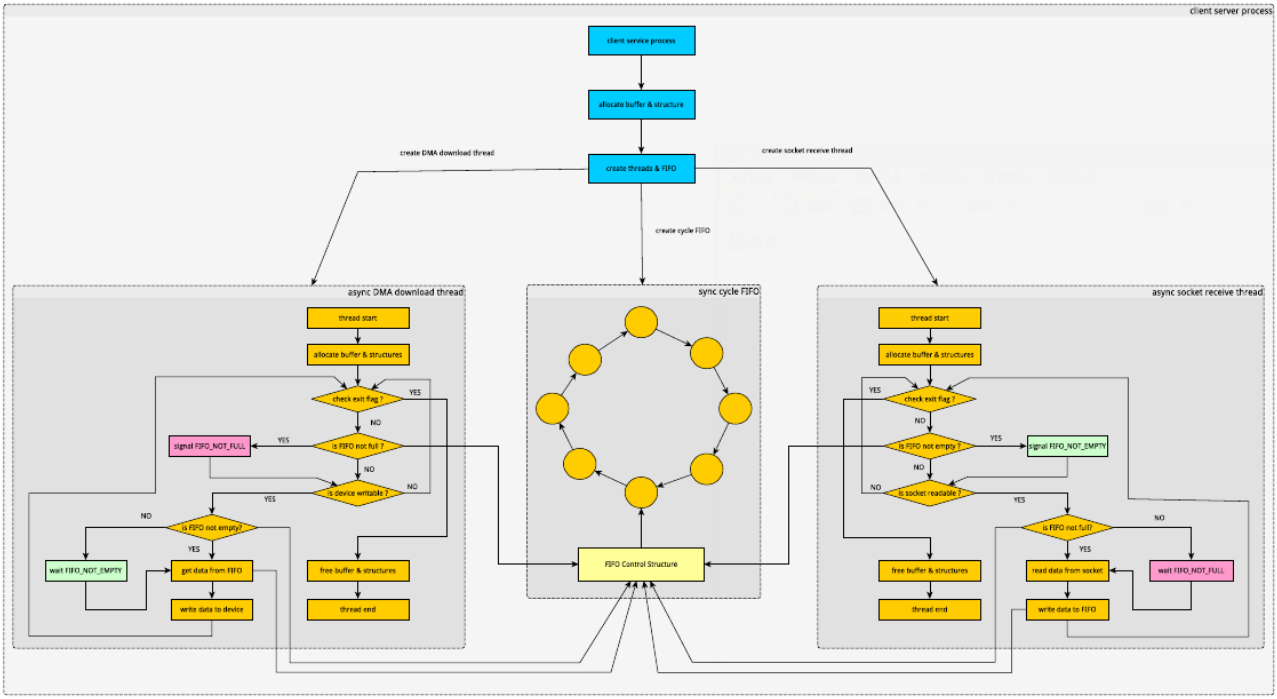


服务端总体上采用多进程+多线程的方法来实现，监听进程(listen process)作为主进程，当接收到来自 SDK 客户端的连接请求时，分叉一个服务进程(service process)专门用于处理跟此客户端的数据通信处理。服务进程内部又创建 2 个子线程 DMA 数据下发线程、异步 socket 读取线程和一个环形 FIFO。

服务端支持多块 IMB 板卡访问，通过设备文件名进行区分。驱动程序加载时，会自动枚举当前 PCI 总线上挂接的板卡，然后在/dev/mvc_card 目录下设备文件。例当前枚举到 2 块板卡，则创建设备文件如下：

/dev/mvc_card/card0
/dev/mvc_card/card1

三、服务端服务进程架构



服务进程被创建后，先分配相关的控制结构体及申请缓冲区，创建 FIFO，然后创建 2 个工作线程：异步 DMA 下发线程和异步 SOCKET 网络接收线程。

异步 SOCKET 网络接收线程：

异步轮询 socket 接口是否可读，当可读时，读取数据至临时缓冲区，然后进行帧头和帧尾序列检测，当检测到一个完整的帧时，抽取本地数据包结构并写入 FIFO。

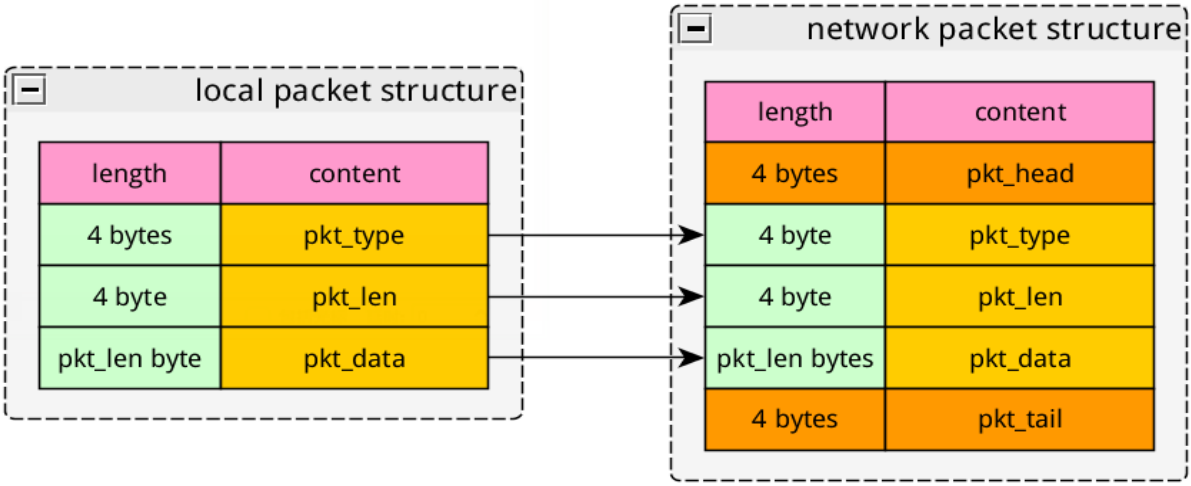
异步 DMA 下发线程：

异步轮询 device 接口是否可写，当可写时，从 FIFO 中获取一个本地数据包结构，然后写入到设备文件中。

注：

双线程对 FIFO 进行操作时，由互斥锁进行临界保护。

四、数据包结构



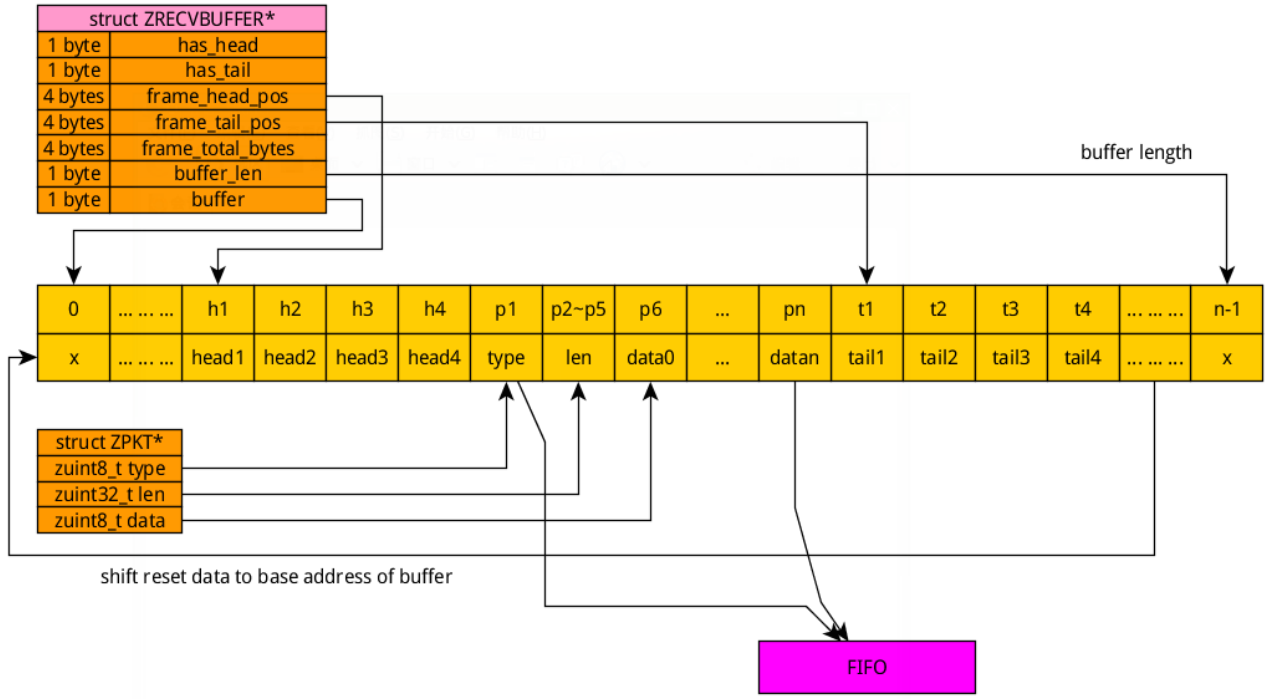
为了方便的进行数据的传输，定义了 2 种数据包结构，一种是本地数据包结构，用于本地的存储、解析等;另一种是网络数据包结构，用于在网络上进行传输。

由于采用 TCP 流式传输方式，数据包大小不一，并不能保证数据包在一次全部发送或全部接收完毕，协议栈可能会将其分包发送，所以在每个本地数据包传输之间，增加了 4 字节的帧头和 4 字节的帧尾。接收方检测到帧头认为是一个数据包的开始，检测到帧尾认为是一个数据包的结束。

注：

在 32 位机上，保证 4 字节对齐。

五、流式数据解析（粘包处理)



1、帧头+帧尾

由于 TCP 流式 socket 数据通信，并不能保证硬件一次性的将所有的数据都发送完毕，所以这里手工增加了帧头和帧尾。帧头为 4 个字节，固定特殊字符；帧尾也为 4 个字节，固定特殊字符。

2、接收

流式数据解析由 ZRECVBUFFER 结构体进行控制，frame_total_bytes 记录了自上一帧完整数据处理后的余下的数据的长度，当异步接收线程轮询到 socket 可读时，则从 socket 中尽可能多的读取数据，暂存在临时 buffer 中，增加 frame_total_bytes 接收总字节计数器。

3、检测帧头和帧尾

从 buffer 的首地址[0]开始到[frame_total_bytes-3]开始检测，是否包含帆头，若以下条件满足：

buffer[index] ==FRM_HEAD1 &&
buffer[index+1] ==FRM_HEAD2 &&
buffer[index+2] ==FRM_HEAD3 &&
buffer[index+3] ==FRM_HEAD4
则表示检测到帧头，置标志位 has_head=1，记录帧头索引值，frame_head_pos=index。

从 buffer 的尾地址[frame_head_pos]到[frame_total_bytes]开始检测，是否包含帧尾，若以下条件满足：

buffer[index] ==FRM_TAIL1 &&
buffer[index+1] ==FRM_TAIL2 &&
buffer[index+2] ==FRM_TAIL3 &&
buffer[index+3] ==FRM_TAIL4
则表示检测到帧尾，置标志位 has_tail=1，记录帧尾索引值，frame_tail_pos=index。

若
has_head==0 && has_tail==0，无效帧数据，直接抛弃缓冲区所有数据；
has_head==1 && has_tail==0，一帧未传完，暂不处理，等待下一次读到新数据时，再解析；
has_head==0 && has_tail==1，无效帧数据，直接抛弃缓冲区所有数据；
has_head==1 && has_tail==1，检测到完整帧数据，将数据拷贝到 FIFO 中，然后将缓冲区中余下的数据移至缓冲区首地址，重置接收总字节数计数器 frame_total_bytes。

循环处理临时缓冲区中所有的完整帧，前移不完整帧数据，等待下一次数据接收拼成完善的一帧。

六、服务端配置

1、配置文件

服务端软件支持配置文件，进程启动时，若存在默认的配置文件则读取进入初始化，否则使用程序内置的参数值。默认配置文件为/etc/zsdkd_server.conf。支持的配置选项暂定如下：

```
#********************network features**********************
#listen on which port ? Default is 1987.
LISTEN_PORT=1987
#max connected clients limit. Default is 1.
MAX_CLIENTS_NUM=1

#********************cycle FIFO features*******************
#how many packets can FIFO buffer ? Default is 6.
FIFO_DEPTH=10
#packet buffer size in FIFO. Default is 2MB.
FIFO_PKT_BUFSIZE=2048

#********************FPGA related************************
#key to active FPGA decoder IP core
LIC_KEY_MSB=0x57ba6d39
LIC_KEY_LSB=0x17e3958f
```

配置文件中以#开始的行为注释行，程序不解析，配置选项由 “key=value” 的样式进行存储。

2、实例单一化

服务器只在当时 OS 中启用一个实例进程，通过程序启动时加锁 PID 文件来实现的。程序启动后创建 PID 文件，默认为/var/run/zsdkd_server.pid，并将自身 PID 写入其中，然后对该文件加锁。

当另一进程启动时，尝试对 PID 文件进行加锁就会失败，则说明系统中已经有一实例正在运行，进程直接报错退出即可。

3、进程控制脚本

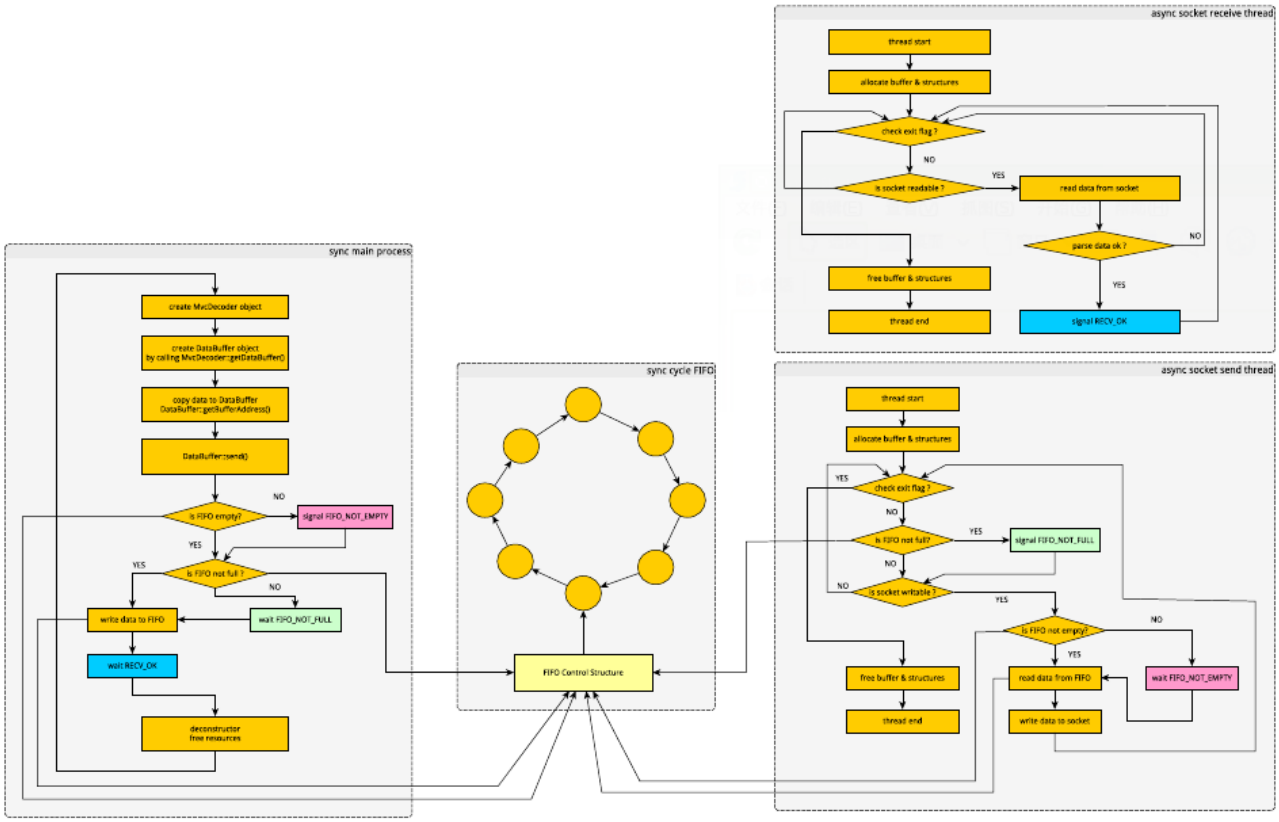
服务器端程序开机时，要以守护进程的形式启动；调试时需要停止、重新启动等。所以这里将提供常规 linux 程序都有的控制脚本。

/etc/init.d/zsdks_server start|stop|restart

进程的停止，是通过 shell 的 kill 程序发送信号给服务器端程序，服务器端程序在信号处理程序中通知所有的子进程退出，子进程再通知所有的子线程退出。

进程的重启，是先让进程停止，然后再启动。

七、客户端 SDK 架构



客户端的 SDK 从总体上划分为四部分，这四部分由三个线程和一个环形 FIFO 组成，各线程间通过互斥锁限制对临界区的访问：

1、同步主线程：

由 SDK 类库自身进程调用，将 DataBuffer 中的数据写入到环形 FIFO 中；

2、环形 FIFO：

暂存需要发送到远程服务端的数据；

3、异步发送线程：

异步轮询远程 socket 是否可写，当可写时，从 FIFO 中取出一个本地数据包，然后加上帧头和帧尾，打包成网络数据包，通过 socket 发送出去；

4、异步接收线程

异步轮询远程 socket 是否写读，当可读时，读取数据包，暂存在临时 buffer 中，然后进入序列检测，当检测到一个完整的网络帧时，剥去帧头和帧尾解析后，发出相应的信号，唤醒同步主线程继续执行。

八、网络帧类型

网络帧类型定义了服务器与客户端之间通信时可用的功能帧，用于标识一帧数据的具体意义，暂时定义如下：

1、读 FPGA 寄存器
2、写 FPGA 寄存器
3、传输音视频数据
4、获取 IMB 板卡数量
5、获取服务端所在主机的 CPU、内存等信息

(注：随着 SDK 类库开发工作的推进，帧类型会逐步增加。)

九、进度计划表

时间	任务	子任务	当前进度	备注
06/03~06/06	服务器端开发	总体架构设计	已完成，100%	正常
		协议处理	已开始，50%	读写寄存器 DMA 数据传输 获取 CPU 信息 获取内存信息 枚举板卡信息 心跳包探测
		底层驱动接口对接	未开始	
06/09~06/13	客户端 SDK 开发	总体架构设计	已完成，100%	正常
		协议处理	已开始，50%	
		类库完善	已经开始，60%	复用以前的 将底层驱动接口改为 socket 接口
06/16~06/20	C/S+驱动联调	稳定可靠性测试	未开始	
		性能优化测试	未开始	
06/23~06/27	上层应用程序联调	接口对接调试	未开始	
06/30~xxxx	维护及 bug 修改	维护及 bug 修改	未开始	

/*the end of file*/