Binder

提出问题：谈一谈binder原理，和实现拷贝一次的流程。

解析：问题点： 1、用户态、内核态

2、linux进程通信原理

3、引申binder为何只需要一次拷贝

binder是一种跨进程通信。

Linux传统通信方式：IPC进程间通信

跨进程通信涉及到的概念：

1、进程隔离

2、进程空间划分：用户空间/内核空间

3、系统调用：用户态，内核态

一、进程隔离：

进程与进程之间内存不共享。a无法访问b，所以要采用特殊通信机制，IPC，言外之意就是内核、用户存在隔离状态。

二、进程空间划分：用户空间，内核空间

现在操作系统采用虚拟存储，所以拥有寻址空间（虚拟存储空间）32位系统有2^32大小，4G。操作系统独立于普通应用程序。保证内核安全要使用内核空间和用户空间的形式。最高字节的1G给内核，低字节给各进程使用

三、系统调用：用户态和内核态

用户需要访问内核资源，文件操作，网络访问 要借助系统调用来实现。（唯一方式）进程工作在用户空间即用户态，内核态则类似。

进程执行系统调用陷入内核代码执行，称为内核状态。

具体实现：

1：从当前进程获取内核栈提取ss0和esp0信息

2：使用ss0和esp0指向的内核栈将当前进程cs，eip，eflags，ss，esp信息保存，这个过程完成了用户栈到内核栈的切换过程。同时保存被执行的程序的下条指令

3：由中断向量检索得到中断处理程序cs，eip信息装入相应寄存器，开始执行中断处理程序，这时就转到了内核态的程序执行。

四、linux下传统IPC通信原理

消息发送到缓存区，系统调用进入内核态，内核程序开辟内核空间（内核缓存区）。调用copy\_from\_user，从用户空间拷贝到内核空间中。同样，接受进程数据会从自己用户空间开辟缓存，内核调用copy\_to\_user拷贝到用户缓存。

存在两个问题：

性能低，一次要两次数据拷贝

结合数据缓存由接收进程提供，但并不知道需多大空间来存放传递数据。

Binder跨进程通信

动态内核可加载模块&&内存映射

动态内核可加载模块：

loadable kernel module，LKM。模块具有独立功能的程序，运行时会链接到内核作为一部分运行。通信会通过内核模块作为桥梁进行。

内存映射：

内存映射是通过mmap（）实现，操作系统的内存映射方法。用户空间一块区域映射到内核空间，建立联系，用户修改可反映到内核。内核修改也可反映到用户。减少数据拷贝次数，实现高效互动。

实现原理：

通过 物理介质（文件系统）mmap（）映射。

进程中用户是不能直接和物理设备打交道，但三把磁盘数据读取到进程区域需要两次拷贝，mmap（）作用在物理介质和用户空间之间建立映射。减少数据拷贝，直接用内存读写代替io读写，提高文件读取效率。

binder是在内核空间创建数据接收的缓存空间。

完整通信流程：

binder驱动在内核空间创建一个数据接收缓存区

在内核空间开辟一块内核缓存区，建立内核缓存区和内核数据接收缓存区的映射。以及内核中数据缓存区和接收进程用户空间地址的映射关系；

发送方进程通过系统调用将数据复制到内核缓存区，由于内存缓存区和接收进程的用户空间存在内存映射，因此相当于把数据发送到接收进程的用户空间，这样便完成了一次进程间通信。

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

Binder通信模型：

客户端，服务端。管理端的驱动，C/S架构

通信过程：

进程使用BINDERSETCONTEXT\_MGR 命令通过binder驱动注册servicemanager。

server通过驱动向sm中注册binder实体。表明对外提供服务。驱动为这个binder创建位于内核中实体节点以及sm对实体的引用。将名字新建引用打包传给sm，sm填入查找表。

client通过名字，在binder的帮助下从sm获取对binder实体的引用，通过引用就能实现和server进程通信。

通信中的代理模式

驱动会返回一个obj一样的代理对象，并不会返回实体。

binder的意义：

1、是一种进程间通信的机制

2、是一种binder的实体对象

3、远程代理

4、跨进程传输对象，并且会自动完成代理对象和本地对象的转换。

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/