Android专用驱动

Logger、Binder、Ashmem

罗升阳

http://weibo.com/shengyangluo

http://blog.csdn.net/luoshengyang

About Me

- 《老罗的Android之旅》博客作者
- 《Android系统源代码情景分析》书籍作者
- 博客：http://blog.csdn.net/Luoshengyang
- 微博：http://weibo.com/shengyangluo

Agenda

- Android专用驱动概述
- Android Logger驱动系统
- Android Binder驱动系统
- Android Ashmem驱动系统

Android专用驱动概述

- 以Linux驱动形式实现在内核空间
  - 不是为了驱动硬件设备工作
  - 而是为了获得特权管理系统
- 为Android Runtime Framework服务
  - 高效的日志服务
  - 高效的进程间通信机制
  - 以组件为目标的进程管理机制
  - ……
- 在整个系统中广泛和频繁地使用

Android Logger驱动系统

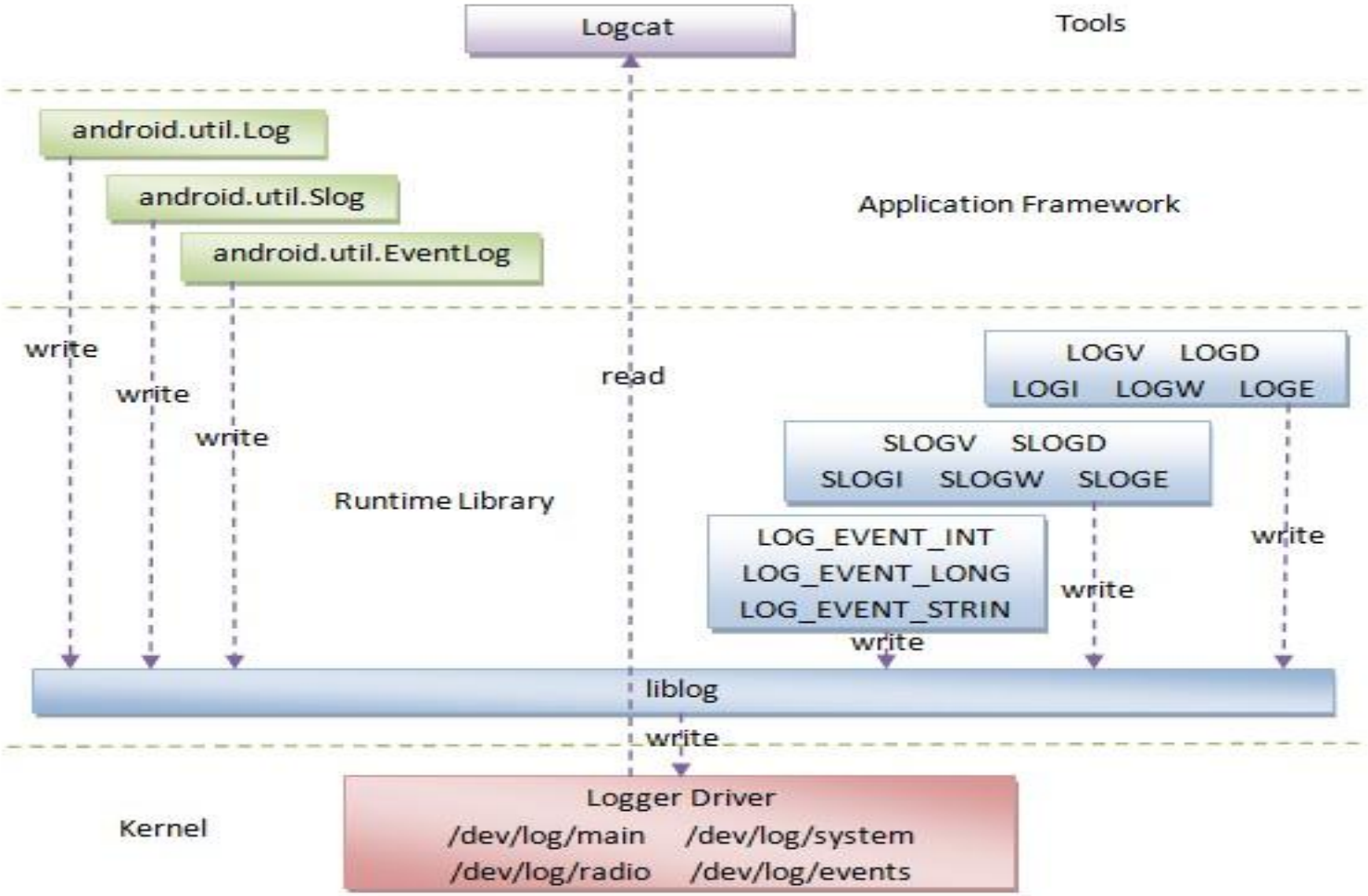
- 日志系统的作用和特点
  - 开发期间调试程序功能
  - 发布期间记录程序运行
  - 频繁和广泛地使用
- 传统日志系统以文件为输出
  - 从用户空间切换至内核空间
  - 在内核空间执行磁盘IO操作
- Android日志系统以内核缓冲区为输出
  - 从用户空间切换至内核空间
  - 直接内存操作
- Android日志系统更高效

Android Logger驱动系统(续)

- 以缓冲区为输出的日志系统要解决的问题
  - 不能占用过多的内存
  - 次要日志不能覆盖重要日志
  - 频繁写入的日志不能覆盖不频繁写入的日志
- 以上三个问题的解决方案
  - 环形缓冲区，新日志覆盖旧日志
  - 日志分类，不同类的日志写在不同的缓冲区

Android Logger驱动系统(续)

- 整体架构图



Android Logger驱动系统(续)

- 日志分类
  - Main，记录应用程序类型日志，次要，文本格式
    - /dev/log/main
  - System，记录系统类型日志，重要，文本格式
    - /dev/log/system
  - Radio，记录无线相关日志，频繁，文本格式
    - /dev/log/radio
  - Event，记录系统事件日志，特殊用途，二进制格式
    - /dev/log/events
- 每一类日志对应一个设备文件
  - Main：/dev/log/main
  - System： /dev/log/system
  - Radio：/dev/log/radio
  - Event：/dev/log/events
- 每一类日志对应一个环形缓冲区，大小256K

Android Logger驱动系统(续)

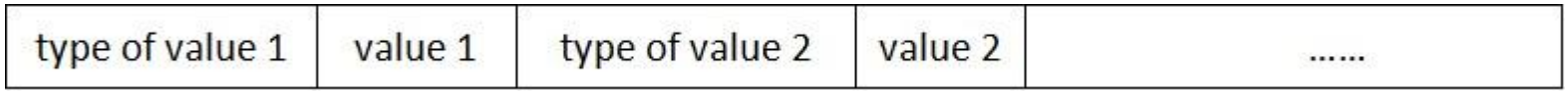
- 文本类型日志格式

- Priority，优先级，整数
  - VERBOSE、DEBUG、INFO、WARN、ERROR、FATAL
- Tag，标签，字符串
  - 自定义
- Msg，日志内容，字符串
  - 自定义

Android Logger驱动系统(续)

- 二进制类型日志格式

  - Tag，标签，整数
    - 自定义
  - Msg，日志内容，二进制数据块



    - Type：整数(1)，长整数(2)，字符串(3)，列表(4)
    - Value：自定义

Android Logger驱动系统(续)

- 二进制日志格式由/system/etc/event-log-tags文件描述

tag number	tag name	format for tag value

  - 日志标签tag number对应的文本描述为tag name
  - 日志内容格式

```
(name|data type[|data unit])[,(name|data type[|data unit]),.....]
```

    - name：名称
    - data type：数据类型
    - data unit：数据单位

Android Logger驱动系统(续)

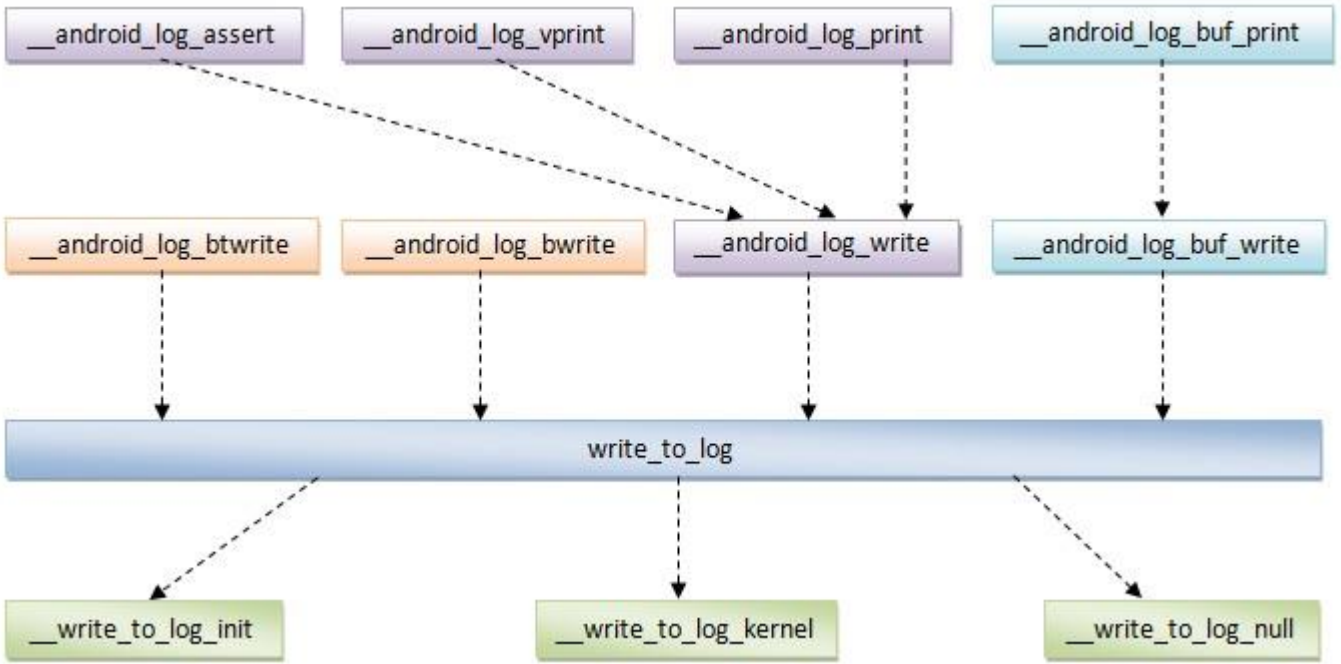
- 二进制日志格式描述示例

  2722 battery_level (level|1|6),(voltage|1|1),(temperature|1|1)

  - 日志2722表示日志标签值，battery_level是其对应的文件描述
  - 标签值等于2722的日志内容由三个值组成，它们分别是level、voltage和temperature
  - level 、voltage和temperature的数据类型均是整数(1)
  - level的单位是百分比(6)
  - voltage和temperature的单位均为对象数量(1)

Android Logger驱动系统(续)

- 用户空间日志库



Android Logger驱动系统(续)

- __android_log_assert、__android_log_vprint、__android_log_print
  - 写入类型为main的日志记录
- __android_log_btwrite、__android_log_bwrite
  - 写入类型为event的日志记录
- __android_log_buf_print
  - 写入任意类型的日志记录
- __android_log_write、__android_log_buf_write
  - 日志标签以“RIL”开头或者于“HTC_RIL”、“AT”、“GSM”、“STK”、“CDMA”、“PHONE”或“SMS”，那么会被认为是radio类型的日志记录
- write_to_log
  - 函数指针，负责最终的日志写入
  - 开始时指向__write_to_log_init
  - 初始化成功后指向__write_to_log_kernel
  - 初始化失败后指向__write_to_log_null

Android Logger驱动系统(续)

- C/C++日志写入接口
  - LOGV、LOGD、LOGI、LOGW、LOGE
  - SLOGV、SLOGD、SLOGI、SLOGW、SLOGE
  - LOG_EVENT_INT、LOG_EVENT_LONG、LOG_EVENT_STRING
- Java日志写入接口
  - android.util.Log
  - android.util.Slog
  - android.util.EventLog

Android Logger驱动系统(续)

- 日志读取工具-- logcat
  - adb logcat --help

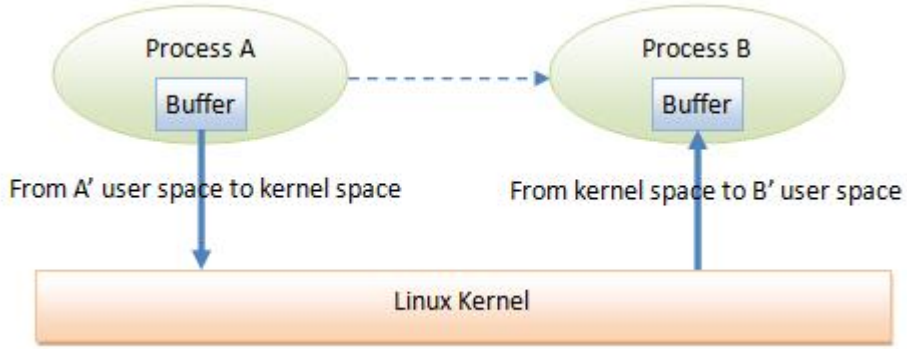
```
luo@ubuntu-11-04:~/Android$ adb logcat --help
Usage: logcat [options] [filterspecs]
options include:
  -s              Set default filter to silent.
                  Like specifying filterspec '*:s'
  -f <filename>   Log to file. Default to stdout
  -r [<kbytes>]   Rotate log every kbytes. (16 if unspecified). Requires -f
  -n <count>      Sets max number of rotated logs to <count>, default 4
  -v <format>     Sets the log print format, where <format> is one of:

                  brief process tag thread raw time threadtime long

  -c              clear (flush) the entire log and exit
  -d              dump the log and then exit (don't block)
  -t <count>      print only the most recent <count> lines (implies -d)
  -g              get the size of the log's ring buffer and exit
  -b <buffer>     request alternate ring buffer
                  ('main' (default), 'radio', 'events')
  -B              output the log in binary
filterspecs are a series of
  <tag>[:priority]
```

Android Binder驱动系统

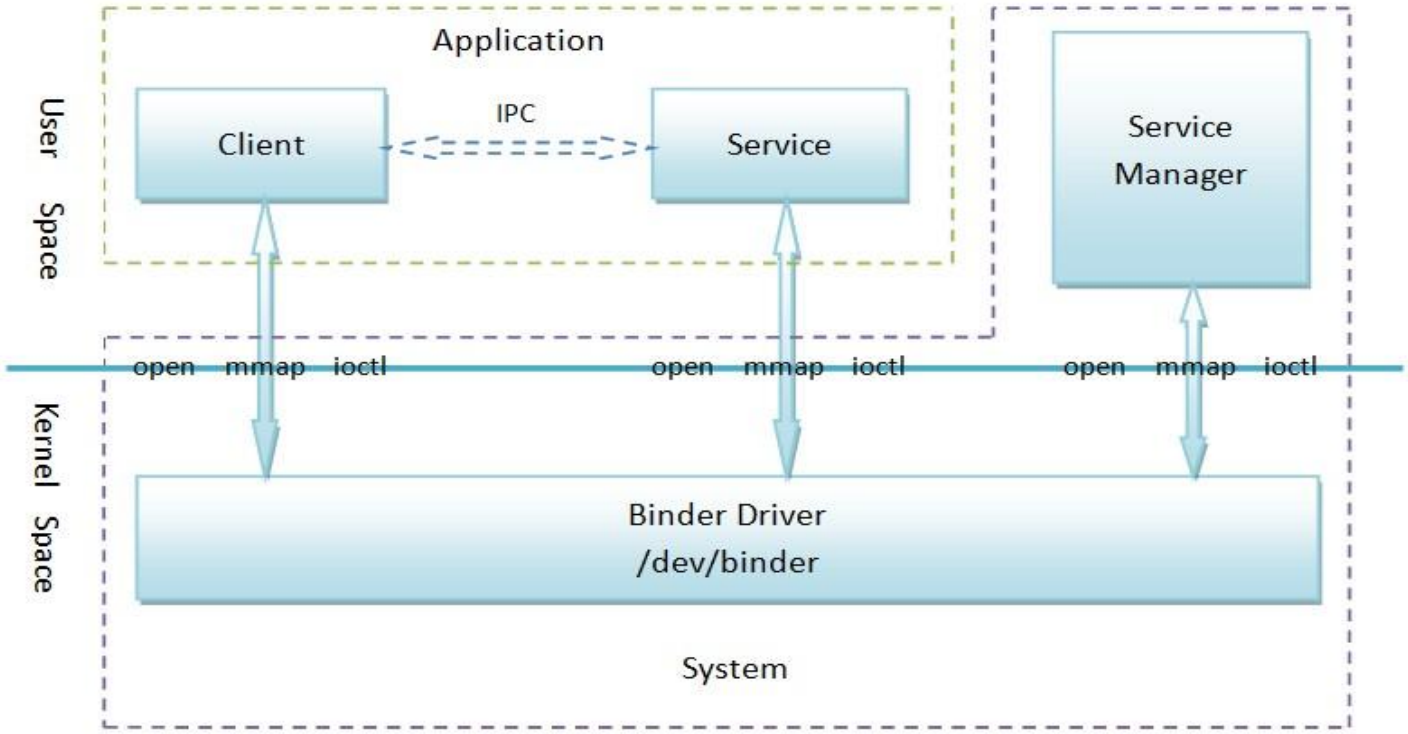
- 传统的IPC，例如Pipe和Socket，执行一次通信需要两次数据拷贝



- 内存共享机制虽然只需要执行一次数据拷贝，但是它需要结合其它IPC来做进程同步，效率同样不理想

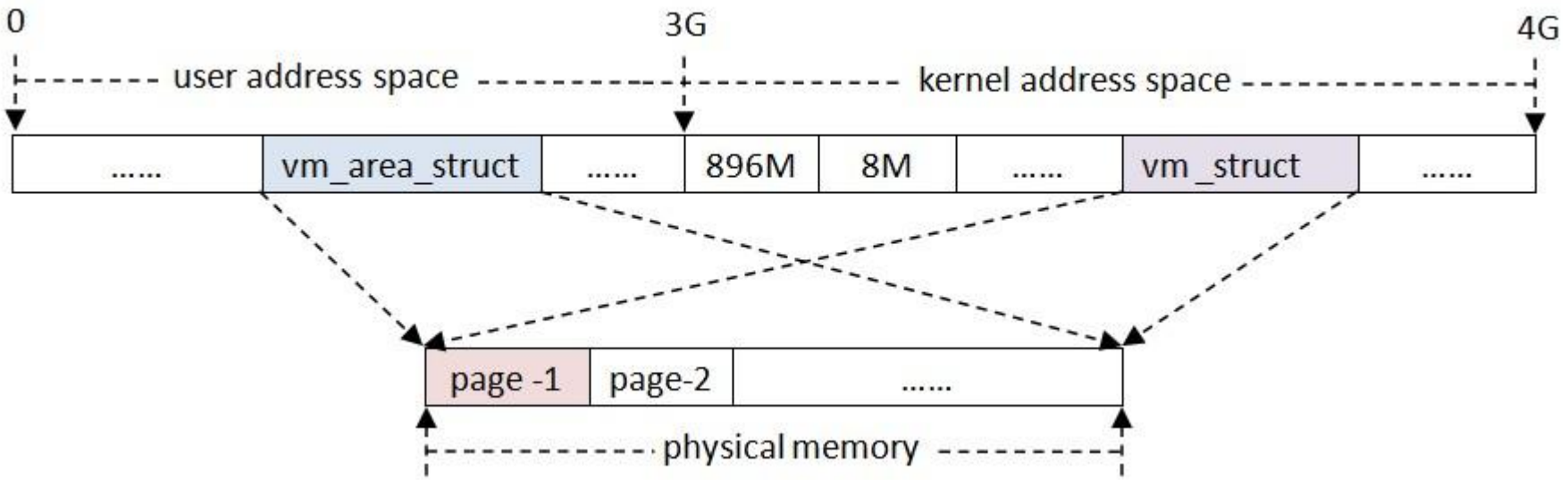
Android Binder驱动系统(续)

- Binder是一种高效且易用的IPC机制
  - 一次数据拷贝
  - Client/Server通信模型
  - 既可用作进程间通信，也可用作进程内通信



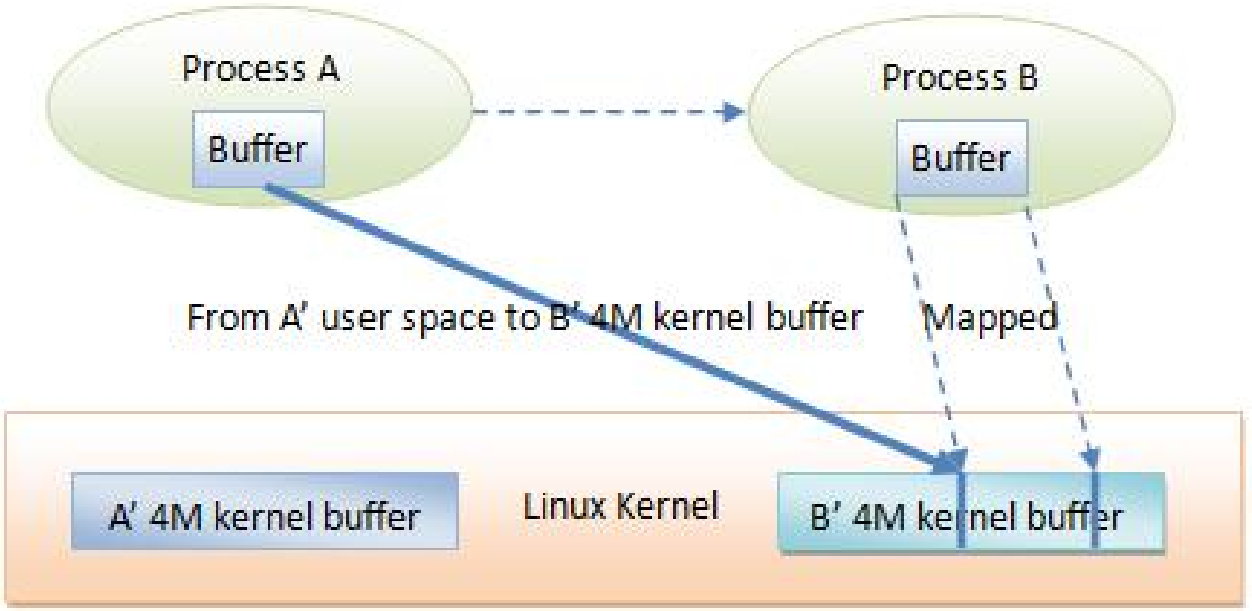
Android Binder驱动系统(续)

- Binder驱动为每一个进程分配4M的内核缓冲区，用作数据传输
  - 4M内核缓冲区所对应的物理页面是按需要分配的，一开始只有一个物理页被被映射
  - 4M内核缓冲区所对应的物理页面除了映射在内核空间之外，还会被映射在进程的用户空间



Android Binder驱动系统(续)

- 进程间的一次数据拷贝

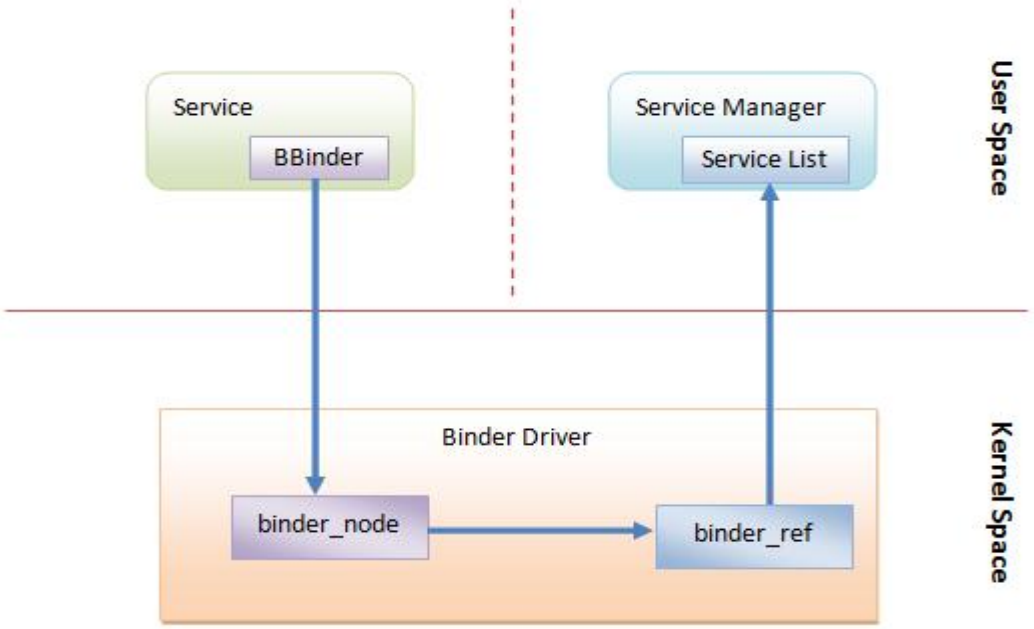


Android Binder驱动系统(续)

- Client/Server通信模型
  - Server进程启动时，将在本进程内运行的Service注册到Service Manager中，并且启动一个Binder线程池，用来接收Client进程请求
  - Client进程向Service Manager查询所需要的Service，并且获得一个Binder代理对象，通过该代理对象即可向Service发送请求

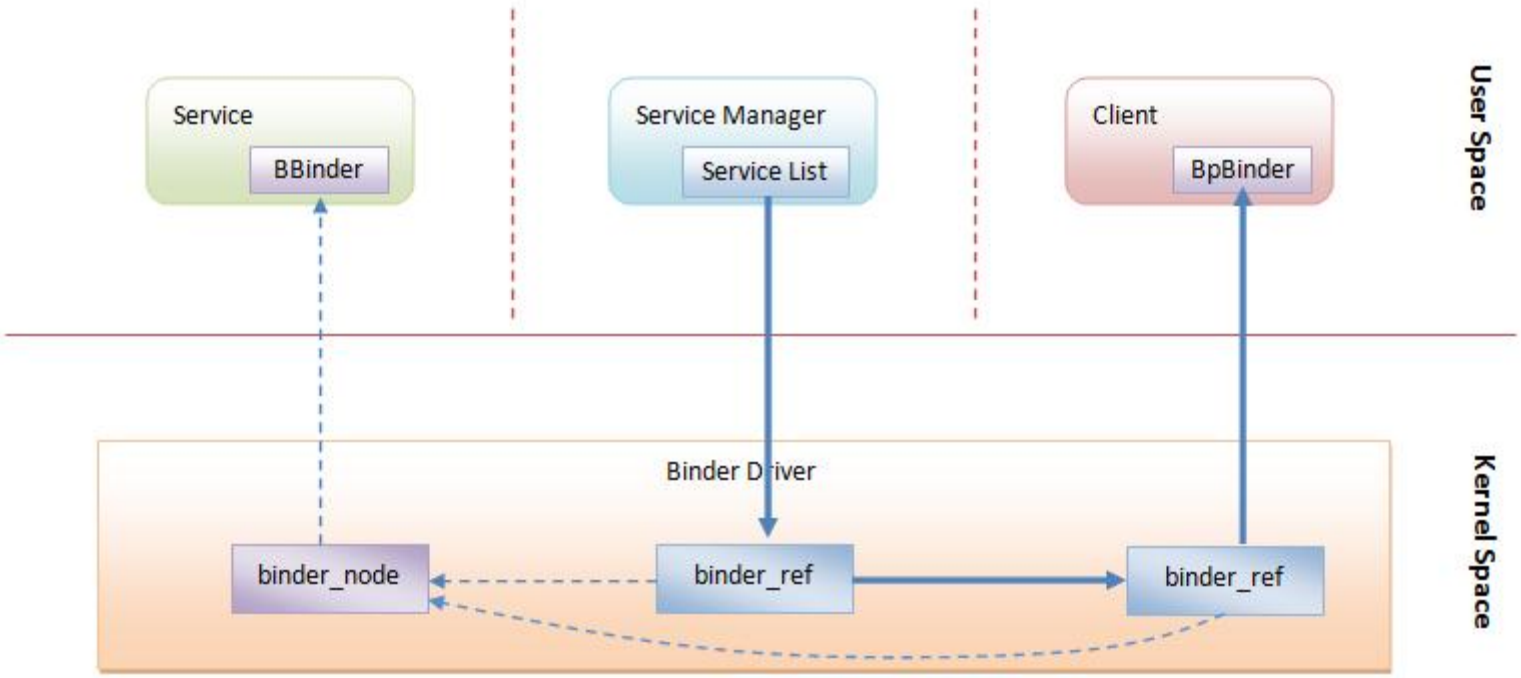
Android Binder驱动系统(续)

- Service注册
  - BBinder：Service在用户空间的描述
  - binder_node：Service在内核空间的描述
  - binder_ref：Service代理在内核空间的描述(一个整数句柄值)



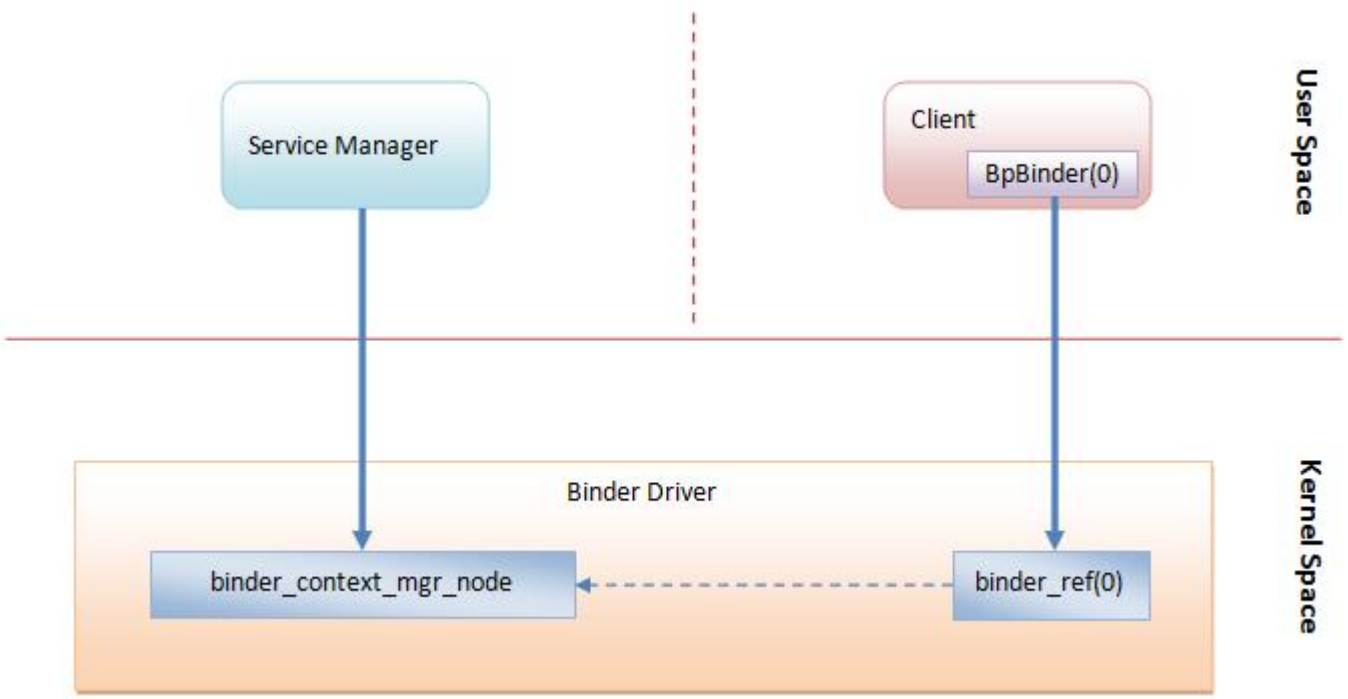
Android Binder驱动系统(续)

- Service代理获取
  - BpBinder：Service代理在用户空间的描述(一个整数句柄值)



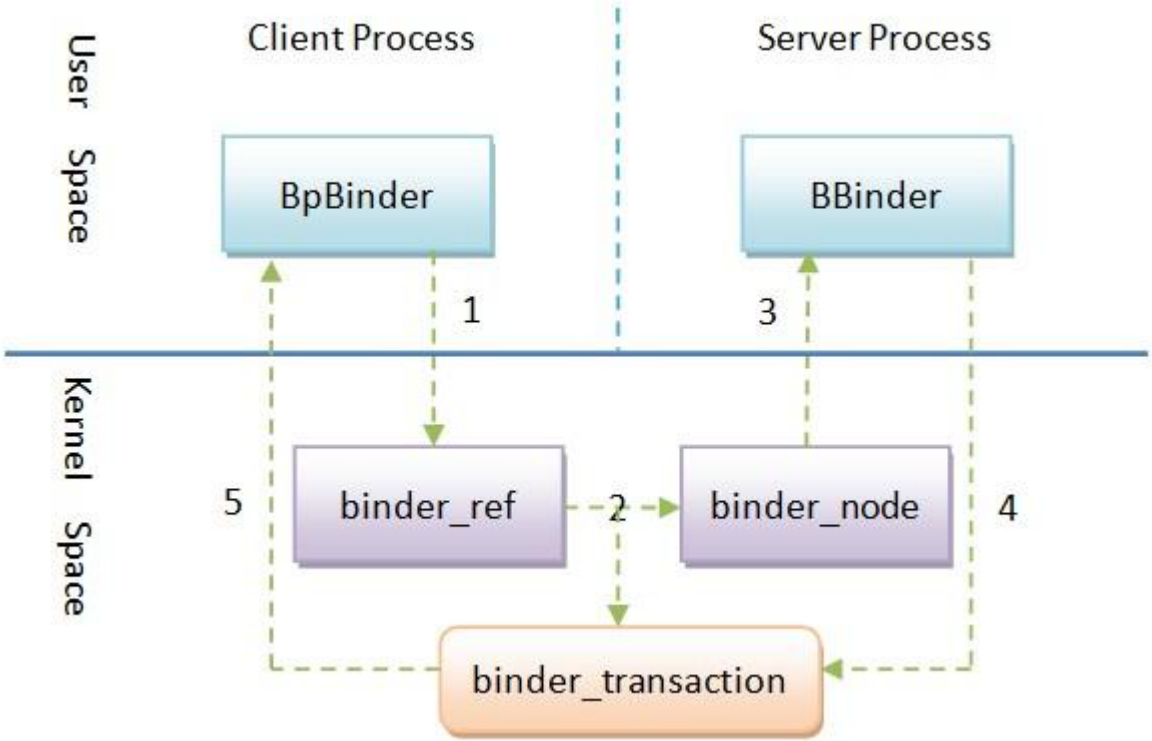
Android Binder驱动系统(续)

- Service Manager注册及其代理获得
  - 一个特殊Service，它的代理句柄值永远等于0



Android Binder驱动系统(续)

- Client和Server的通信过程



Android Binder驱动系统(续)

- binder_transaction：通信数据描述

```
struct binder_transaction {
        int debug_id;
        struct binder_work work;
        struct binder_thread *from;
        struct binder_transaction *from_parent;
        struct binder_proc *to_proc;
        struct binder_thread *to_thread;
        struct binder_transaction *to_parent;
        unsigned need_reply : 1;
        /*unsigned is_dead : 1;*/ /* not used at the moment */

        struct binder_buffer *buffer;
        unsigned int    code;
        unsigned int    fla
        long    priority;
        long    saved_prior
        uid_t   sender_euid
};
```

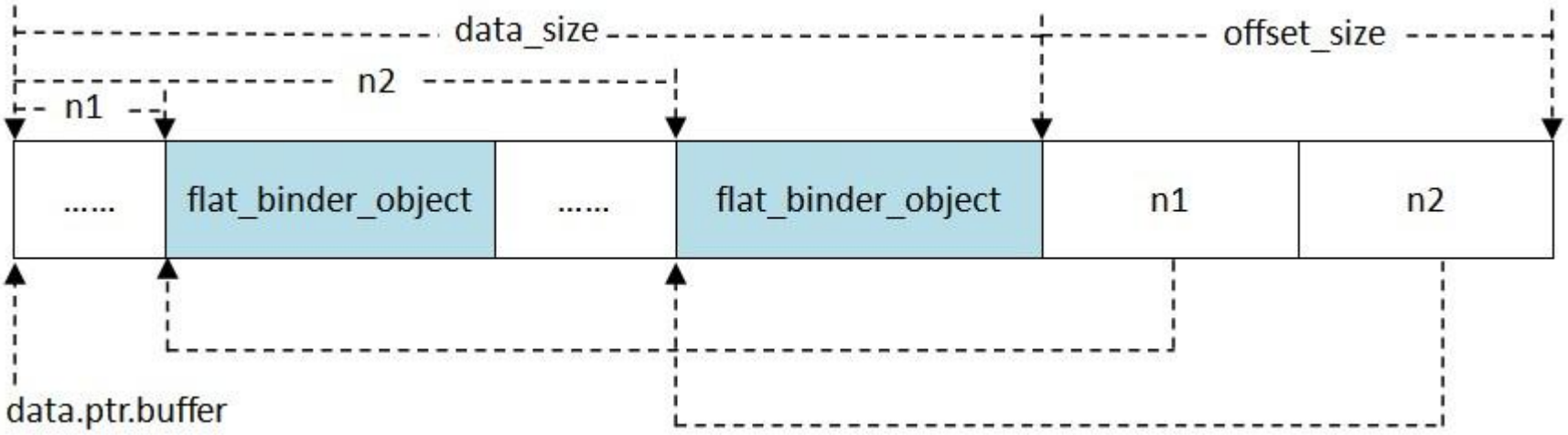
```
struct binder_buffer {
        struct list_head entry; /* free and allocated entries by addesss */
        struct rb_node rb_node; /* free entry by size or allocated entry */
                                /* by address */
        unsigned free : 1;
        unsigned allow_user_free : 1;
        unsigned async_transaction : 1;
        unsigned debug_id : 29;

        struct binder_transaction *transaction;

        struct binder_node *target_node;
        size_t data_size;
        size_t offsets_size;
        uint8_t data[0];
};
```

Android Binder驱动系统(续)

- Binder对象(flat_binder_object)的类型
  - BINDER_TYPE_BINDER
  - BINDER_TYPE_WEAK_BINDER
  - BINDER_TYPE_HANDLE
  - BINDER_TYPE_WEAK_HANDLE
  - BINDER_TYPE_FD



Android Binder驱动系统(续)

- BINDER_TYPE_BINDER和BINDER_TYPE_WEAK_BINDER类型的flat_binder_object传输发生在：
  - Server进程主动向Client进程发送Service (匿名Service)
  - Server进程向Service Manager进程注册Service
- BINDER_TYPE_HANDLE和BINDER_TYPE_WEAK_HANDLE类型的flat_binder_object传输发生在：
  - 一个Client向另外一个进程发送Service代理
- BINDER_TYPE_FD类型的flat_binder_object传输发生在：
  - 一个进程向另外一个进程发送文件描述符

Android Binder驱动系统(续)

- Binder驱动对类型BINDER_TYPE_BINDER 的Binder对象(flat_binder_object)的 处理：
  - 在源进程中找到对应的binder_node。如果不存在，则创建。
  - 根据上述binder_node在目标进程中找到对应的binder_ref。如果不存在，则创建。
  - 增加上述binder_ref的强引用计数和弱引用计数
  - 构造一个类型为BINDER_TYPE_HANDLE的flat_binder_object对象。
  - 将上述flat_binder_object对象发送给目标进程。

Android Binder驱动系统(续)

- Binder驱动对类型BINDER_TYPE_WEAK_BINDER的Binder对象(flat_binder_object)的 处理：
  - 在源进程中找到对应的binder_node。如果不存在，则创建。
  - 根据上述binder_node在目标进程中找到对应的binder_ref。如果不存在，则创建。
  - 增加上述binder_ref的弱引用计数。
  - 构造一个类型为BINDER_TYPE_WEAK_HANDLE的flat_binder_object对象。
  - 将上述flat_binder_object对象发送给目标进程。

Android Binder驱动系统(续)

- Binder驱动对类型BINDER_TYPE_HANDLE 的Binder对象(flat_binder_object)的 处理：
  - 在源进程中找到对应的binder_ref。
    - 如果上述binder_ref所引用的binder_node所在进程就是目标进程：
      - 增加上述binder_node的强引用计数和弱引用计数
      - 构造一个类型为BINDER_TYPE_BINDER的flat_binder_object
      - 将上述flat_binder_object发送给目标进程
    - 如果上述binder_ref所引用的binder_node所在进程不是目标进程：
      - 为目标进程创建一个binder_ref，该binder_ref与上述binder_ref引用的是同一个binder_node
      - 增加上述新创建的binder_ref的强引用计数和弱引用计数
      - 构造一个类型为BINDER_TYPE_HANDLE的flat_binder_object
      - 将上述flat_binder_object发送给目标进程

Android Binder驱动系统(续)

- Binder驱动对类型BINDER_TYPE_WEAK_HANDLE 的Binder对象(flat_binder_object)的 处理：
  - 在源进程中找到对应的binder_ref。
    - 如果上述binder_ref所引用的binder_node所在进程就是目标进程：
      - 增加上述binder_node的弱引用计数
      - 构造一个类型为BINDER_TYPE_WEAK_BINDER的flat_binder_object
      - 将上述flat_binder_object发送给目标进程
    - 如果上述binder_ref所引用的binder_node所在进程不是目标进程：
      - 为目标进程创建一个binder_ref，该binder_ref与上述binder_ref引用的是同一个binder_node
      - 增加上述新创建的binder_ref的弱引用计数
      - 构造一个类型为BINDER_TYPE_WEAK_HANDLE的flat_binder_object
      - 将上述flat_binder_object发送给目标进程

Android Binder驱动系统(续)

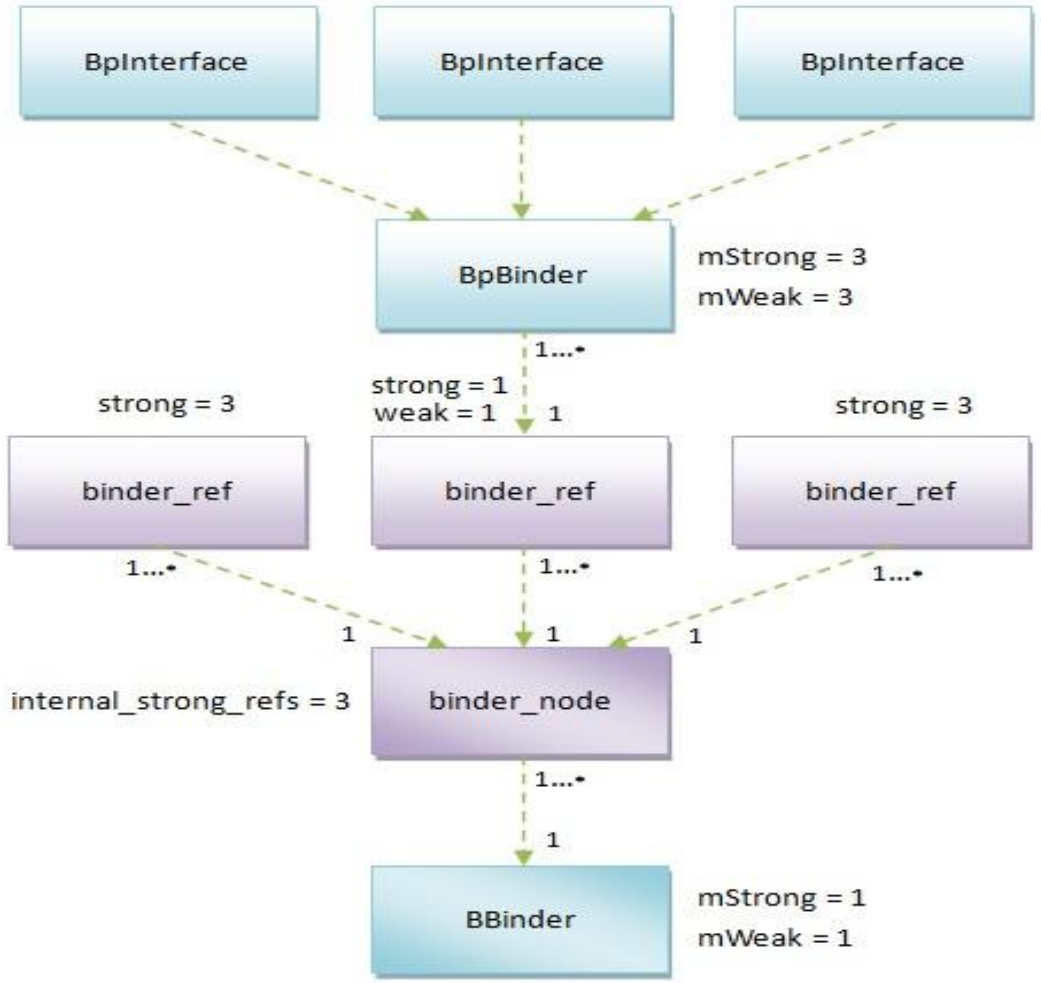
- Binder驱动对类型BINDER_TYPE_FD 的Binder对象(flat_binder_object)的 处理：
  - 在源进程中找到对应的struct file结构体
  - 将上述struct file结构体 保存在目标进程的打开文件列表中
  - 构造一个类型为BINDER_TYPE_FD的flat_binder_object
  - 将上述flat_binder_object发送给目标进程

Android Binder驱动系统(续)

- Binder对象的引用计数
  - BBinder：位于用户空间，通过智能指针管理，有mStrong和mWeak两个引用计数
  - BpBinder：位于用户空间，通过智能指针管理，有mStrong和mWeak两个引用计数
  - binder_node：位于内核空间，有internal_strong_refs、local_weak_refs和local_strong_refs三个引用计数，以及一个binder_ref引用列表refs
  - binder_ref：位于内核空间，有strong和weak两个引用计数

Android Binder驱动系统(续)

- Binder对象之间的引用关系



Android Binder驱动系统(续)

- Binder对象引用关系小结
  - BBinder被binder_node引用
  - binder_node被binder_ref引用
  - binder_ref被BpBinder引用
  - BBinder和BpBinder运行在用户空间
  - binder_node和binder_ref运行在内核空间
- 内核空间足够健壮，保证binder_node和binder_ref不会异常销毁
- 用户空间不够健壮，BBinder和BpBinder可能会异常销毁
- BpBinder异常销毁不会引发致命问题，但是BBinder异常销毁会引发致命问题
- 需要有一种方式来监控BBinder和BpBinder的异常销毁

Android Binder驱动系统(续)

- Binder对象异常销毁监控
  - 所有执行Binder IPC的进程都需要打开/dev/binder文件
  - 进程异常退出的时候，内核保证会释放未正常关闭的它打开的/dev/binder文件，即调用与/dev/binder文件所关联的release回调函数
  - Binder驱动通过实现/dev/binder文件的release回调函数即可监控Binder对象的异常销毁，进而执行清理工作

Android Binder驱动系统(续)

- BBinder异常销毁的时候，不单止Binder驱动需要执行清理工作，引用了它的BpBinder所在的Client进程也需要执行清理工作
- 需要有一种BBinder死亡通知机制
  - Client进程从Binder驱动获得一个BpBinder
  - Client进程向Binder驱动注册一个死亡通知，该死亡通知与上述BpBinder所引用的BBinder相关联
  - Binder驱动监控到BBinder所在进程异常退出的时候，检查该BBinder是否注册有死亡通知
  - Binder驱动向注册的死亡通知所关联的BpBinder所运行在的Client进程发送通知
  - Client进程执行相应的清理工作

Android Binder驱动系统(续)

- Binder线程池
  - 在Server进程中，Client进程发送过来的Binder请求由Binder线程进行处理
  - 每一个Server进程在启动的时候都会创建一个Binder线程池，并且向里面注册一个Binder线程
  - 之后Server进程可以无限地向Binder线程池注册新的Binder线程
  - Binder驱动发现Server进程没有空间的Binder线程时，会主动向Server进程请求注册新的Binder线程
  - Binder驱动主动请求Server进程注册新的Binder线程的数量可以由Server进程设置，默认是16

Android Binder驱动系统(续)

- Binder线程调度机制
  - 在Binder驱动中，每一个Server进程都有一个todo list，用来保存Client进程发送过来的请求，这些请求可以由其Binder线程池中的任意一个空闲线程处理
  - 在Binder驱动中，每一个Binder线程也有一个todo list，用来保存Client进程发送过来的请求，这些请求只可以由该Binder线程处理
  - Binder线程没事做的时候，就睡眠在Binder驱动中，直至它所属的Server进程的todo list或者它自己的todo list有新的请求为止
  - 每当Binder驱动将Client进程发送过来的请求保存在Server进程的todo list时，都会唤醒其Binder线程池的空闲Binder线程池，并且让其中一个来处理
  - 每当Binder驱动将Client进程发送过来的请求保存在Binder线程的todo list时，都会将其唤醒来处理

Android Binder驱动系统(续)

- 默认情况下， Client进程发送过来的请求都是保存在Server 进程的todo list中，然而有一种特殊情况：
  - 源进程P1的线程A向目标进程发起了一个请求T1，该请求被分发给目标进程P2的线程B处理
  - 源进程P1的线程A等待目标进程P2的线程B处理处理完成请求T1
  - 目标进程P2的线程B在处理请求T1的过程中，需要向源进程P1发起另一个请求T2
  - 源进程P1除了线程A是处于空闲等待状态之外，还有另外一个线程C处于空闲等待状态
- 这时候请求T2应该分发给线程A处理，还是线程C处理？
  - 如果分发给线程C处理，则线程A仍然是处于空闲等待状态
  - 如果分发给线程A处理，则线程 C可以处理其它新的请求

Android Binder驱动系统(续)

- Binder线程的事务堆栈

```
struct binder_thread {
        struct binder_proc *proc;
        struct rb_node rb_node;
        int pid;
        int looper;
        struct binder_transaction *transaction_stack;
        struct list_head todo;
        uint32_t return_error; /* Write failed, return error code in read b
uf */
        uint32_t return_error2; /* Write f
*/
                /* buffer. Used when sendi
 */
                /* we are also waiting on
        wait_queue_head_t wait;
        struct binder_stats stats;
};
```

```
struct binder_transaction {
        int debug_id;
        struct binder_work work;
        struct binder_thread *from;
        struct binder_transaction *from_parent;
        struct binder_proc *to_proc;
        struct binder_thread *to_thread;
        struct binder_transaction *to_parent;
        unsigned need_reply : 1;
        /*unsigned is_dead : 1;*/ /* not used at the moment */

        struct binder_buffer *buffer;
        unsigned int    code;
        unsigned int    flags;
        long    priority;
        long    saved_priority;
        uid_t   sender_euid;
};
```

Android Binder驱动系统(续)

- T1：From P1 to P2
  - BC_TRANSACTION:
    - T1->from_parent = NULL
    - T1->from = Thread(A)
    - Thread(A)->transaction_stack = T1
    - *Thread(A)->proc = P1*
  - BR_TRANSACTION:
    - T1->to_parent = NULL
    - T1->to_thread = Thread(B)
    - *Thread(B)-> transaction_stack = T1*
- T2：*From P2 to P1*
  - BC_TRANSACTION:
    - T2->form_parent = T1
    - T2->from = Thread(B)
    - Thread(B)->transaction_stack = T2
    - Thread(B)->proc = P2
  - BR_TRANSACTION
    - **T2->to_parent = T1**
    - *T2->to_thread = Thread(A)*
    - **Thread(A)->transaction_stack = T2**

Android Binder驱动系统(续)

- 同步请求优先于异步请求
  - 同一时刻，一个BBinder只能处理一个异步请求
  - 第一个异步请求将被保存在目标进程的todo list中
  - 第一个异步请求未被处理前，其它的异步请求将被保存在对应的 binder_node的async todo list中
  - 第一个异步请求被处理之后，第二个异步请求将从binder_node的async todo list转移至目标进程的todo list等待处理
  - 依次类推……
  - 此外，所有同时进行的异步请求所占用的内核缓冲区大小不超过目标进程的总内核缓冲区大小的一半

Android Binder驱动系统(续)

- 与请求相关的三个线程优先级
  - 源线程的优先级
  - 目标线程的优先级
  - 目标binder_node的最小优先级（注册Service时设置）
- Binder线程处理同步请求时的优先级
  - 取{源线程，目标binder_node，目标线程}的最高优先级
  - 保证目标线程以不低于源线程优先级或者binder_node最小优先级的优先级运行
- Binder线程处理异步请求时的优先级
  - 取{目标binder_node，目标线程}的最高优先级
  - 保证目标线程以不低于binder_node最小优先级的优先级运行

Android Binder驱动系统(续)

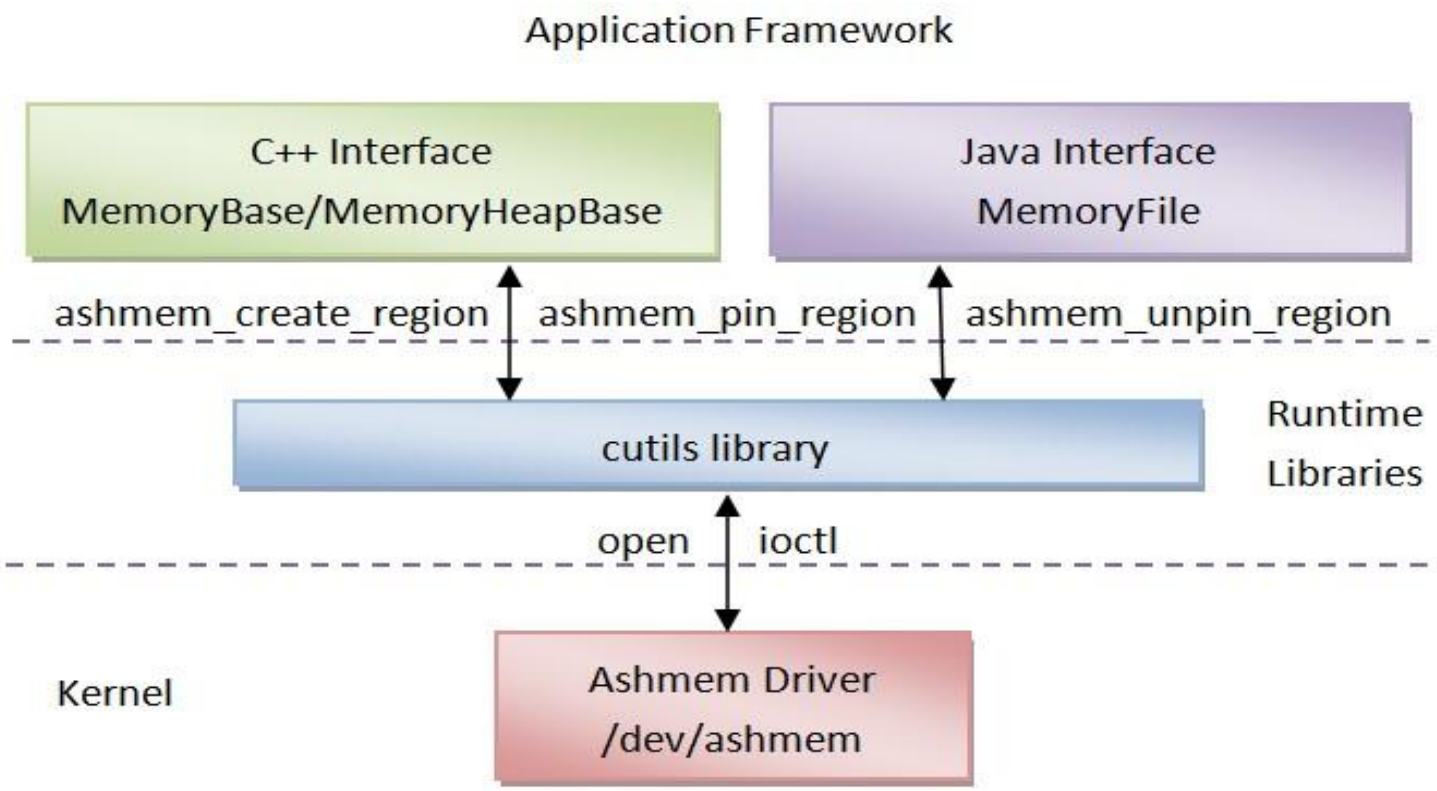
- Binder进程间通信机制的Java接口
  - 每一个Java层的Binder本地对象(Binder)在C++层都对应有一个JavaBBinder对象，后者是从C++层的BBinder继承下来的
  - 每一个Java层的Binder代理对象(BinderProxy)在C++层都对应有一个BpBinder对象
  - 于是Java层的Binder进程间通信实际上就是通过C++层的BpBinder和BBinder来进行的，与C++层的Binder进程间通信 一致

Android Ashmem驱动系统

- 传统的Linux共享内存机制
  - System V：shmget、shmctl、shmat、shmdt
    - 用一个整数ID来标志一块共享内存
    - 不能动态释放部分共享内存
  - Posix：shm_open、ftruncate、mmap、munmap
    - 用一个文件描述符来标一块共享内存
    - 不能动态释放部分共享内存
- Android匿名共享内存
  - open、ioctl、mmap、munmap
    - 用一个文件描述符来标一块共享内存
    - 能动态释放部分共享内存

Android Ashmem驱动系统(续)

- 整体架构



Android Ashmem驱动系统(续)

- Ashmem驱动程序
  - 与System V的共享内存一样，都是基于临时文件系统(tmpfs)来实现的，也就是每一块共享内存都对应有一个临时文件
  - 可以通过IO控制命令ASHMEM_PIN对部分共享内存进行锁定
  - 可以通过IO控制命令ASHMEM_UNPIN对部分共享内存进行解锁
  - 没有锁定的部分共享内存，在系统内存紧张时会被回收

Android Ashmem驱动系统(续)

- C访问接口
  - ashmem_create_region：创建匿名共享内存
    - open /dev/ashmem
    - mmap /dev/ashmem
  - ashmem_pin_region：锁定部分匿名共享内存
    - ioctl ASHMEM_PIN
  - ashmem_unpin_region：解锁部分匿名共享内存
    - ioctl ASHMEM_UNPIN

Android Ashmem驱动系统(续)

- C++访问接口
  - MemoryHeapBase
    - 用来访问一块匿名共享内存
    - Binder Service，实现IMemoryHeap接口
    - 对应的Binder代理为BpMemoryHeap
  - MemoryBase： Binder Service
    - 在MemoryHeapBase的基础上实现，用来访问一整块匿名共享内存的其中一部分
    - Binder Service ，实现IMemory接口
    - 对应的Binder代理为BpMemory

Android Ashmem驱动系统(续)

- Java访问接口
  - android.os.MemoryFile

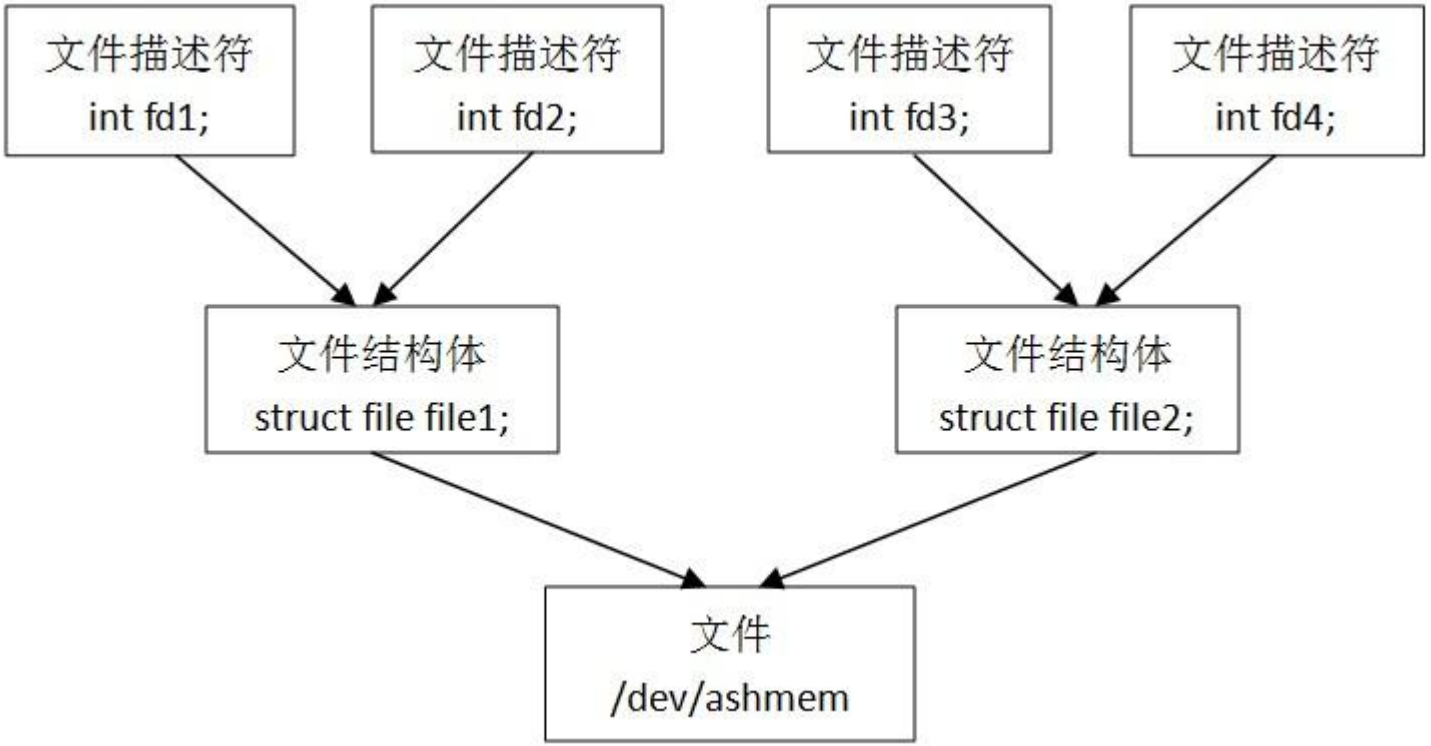
```
public MemoryFile(String name, int length) throws IOException {
    mLength = length;
    mFD = native_open(name, length);
    mAddress = native_mmap(mFD, length, PROT_READ | PROT_WRITE);
    mOwnsRegion = true;
}

public MemoryFile(FileDescriptor fd, int length, String mode) throws IOException {
    if (fd == null) {
        throw new NullPointerException("File descriptor is null.");
    }
    if (!isMemoryFile(fd)) {
        throw new IllegalArgumentException("Not a memory file.");
    }
    mLength = length;
    mFD = fd;
    mAddress = native_mmap(mFD, length, modeToProt(mode));
    mOwnsRegion = false;
}
```

  - Android 2.3之后，第二个构造函数已不存在，因此MemoryFile只能作为内存文件使用

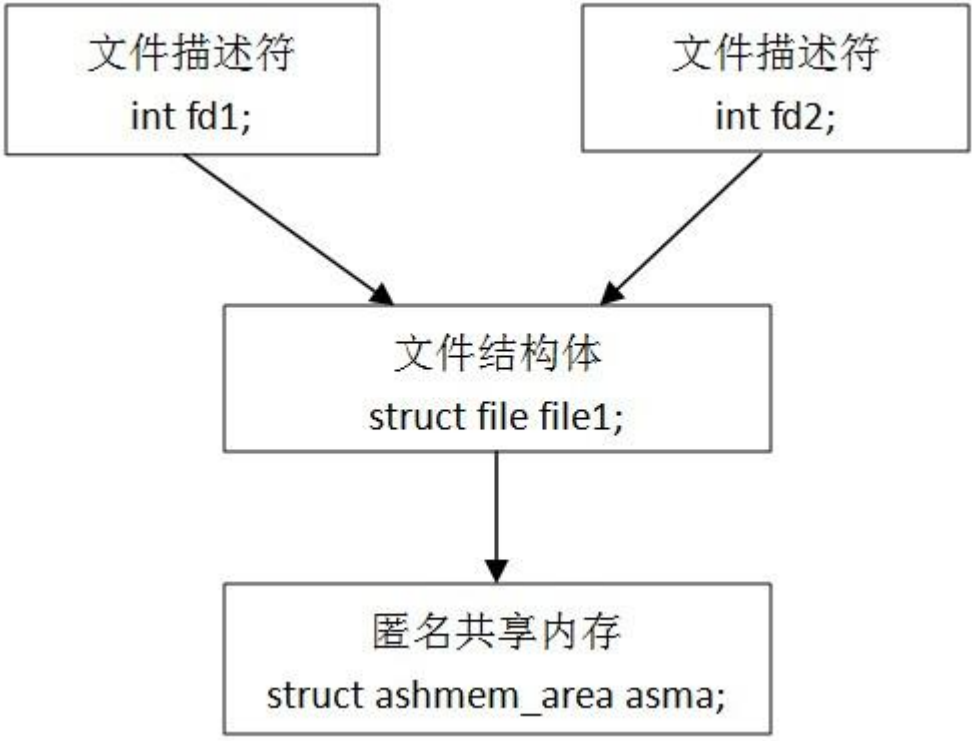
Android Ashmem驱动系统(续)

- Ashmem进程间共享原理--文件、文件结构体、文件描述符的关系



Android Ashmem驱动系统(续)

- Ashmem进程间共享原理--两个在不同进程中的文件描述符对应同一个指向设备文件/dev/ashmem的文件结构体



Android Ashmem驱动系统(续)

- 可以通过Binder IPC在进程间进行传递和共享 – flat_binder_object(BINDER_TYPE_FD)

```
static void
binder_transaction(struct binder_proc *proc, struct binder_thread *thread,
    struct binder_transaction_data *tr, int reply)
{
    ......
    for (; offp < off_end; offp++) {
        struct flat_binder_object *fp;
        ......
        fp = (struct flat_binder_object *)(t->buffer->data + *offp);
        switch (fp->type) {
        ......
        case BINDER_TYPE_FD: {
            int target_fd;
            struct file *file;
            ......
            file = fget(fp->handle);
            ......
            target_fd = task_get_unused_fd_flags(target_proc, O_CLOEXEC);
            ......
            task_fd_install(target_proc, target_fd, file);
            ......
            fp->handle = target_fd;
        } break;
        ......
        }
    }
    ......
}
```

Q&A

Thank You