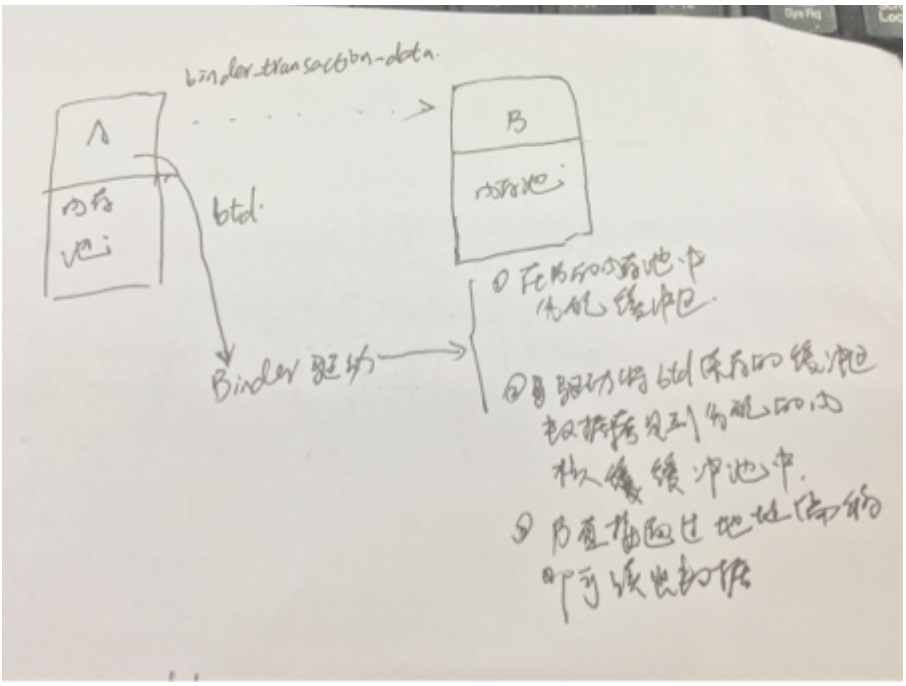
5.1 Binder设备文件打开,映射，以内核缓冲区管理

1. 打开过程
   1) 一个需要使用binder通信的进程，首先要打开设备文件/dev/binder/, 此时binder_open函数被调用，该函数中将为该进程创建一个binder_proc.
   2) 创建的binder_proc将被保存在参数filp的private_data中。因为进程调用open函数打开/dev/binder函数后，内核会返回一个文件描述符给进程，而该文件描述符是与filp 所指向的打开文件结构体关联在一起的，因此后面以这个文件描述符为参数调用其他关键函数来与Binder驱动交互，内核就会将与该文件描述符相关联的打开文件结构体传递给Binder驱动，Binder驱动就可以通过filp的private_data成员取出该进程的Binder_proc.
2. 设备文件内存映射过程
   1). /dev/binder是一个虚拟的设备，将它映射到进程的地址空间的目的不是为了获取其内容，而数据为了给进程分配内核缓冲区，该缓冲区被用来传输进程间通信数据。
   2). Binder驱动程序为进程分配的内核缓冲区在用户空间只读，不能拷贝，并禁止设置可能进行的写操作标志位设置。
   3). Bindder驱动分配的内核缓冲区有两个虚地址，一个用户空间地址，由参数vma指向的vm_area_struct描述；另外一个是内核空间地址，由变量area指向的vm_struct结构体描述。
   4) 在binder_mmap函数中，”proc->free_async_space = proc->buffer_size/2“来设定了最大可以用于异步事务的内核缓冲区为总的内核缓冲区大小的一半。
3. 内核缓冲去管理
   1) 物理内存的分配以页为单位，但是一次使用却不是一页为单位，因此，驱动为进程维护了一个内核缓冲区池，缓冲区池中的每一块内存都用结构体binder_buffer描述，并保存到一个列表中。
   2)内存缓冲区分配时机： 当一个进程以命令BC_TRANSACTION和HC_REPLAY向另外一个进程传递数据时，Binder驱动需要将这些数据从用户空间拷贝到内核空间，然后再传递给目标进程，此时驱动就需要在目标进程的内存池中分配小块内核缓冲区来保存这些数据。
   3)非常重要的说明：

> 在前面的5.1.1小节中介绍命令协议BinderDriverCommandProtocol时提到，当一个进程使用命令协议BC_TRANSACTION或者BC_REPLY来与Binder驱动程序交互时，它会从用户空间传递一个binder_transaction_data结构体给Binder驱动程序，而在binder_transaction_data结构体中，有一个数据缓冲区和一个偏移数组缓冲区，这两个缓冲区的内容就是需要拷贝到目标进程的内核缓冲区中的。

generated by haroopad



4)查询内核缓冲区

5.1.5.2 释放内核缓冲区

当一个进程处理完成Binder驱动程序给它发送的返回协议BR_TRANSACTION或者BR_REPLY之后，它就会使用命令协议BC_FREE_BUFFER来通知Binder驱动程序释放相应的内核缓冲区，以免浪费系统内存。

在Binder驱动程序中，释放内核缓冲区的操作是由函数binder_free_buf实现的，它的定义如下所示。

kernel/goldfish/drivers/staging/android/binder.c

```
01 static void binder_free_buf(
02         struct binder_proc *proc, struct binder_buffer *buffer)
03 {
04         size_t size, buffer_size;
05
06         buffer_size = binder_buffer_size(proc, buffer);
07
08         size = ALIGN(buffer->data_size, sizeof(void *)) +
09                 ALIGN(buffer->offsets_size, sizeof(void *));
10         ......
11
12         if (buffer->async_transaction) {
13                 proc->free_async_space += size + sizeof(struct binder_buffer);
14                 ......
15         }
16
17         binder_update_page_range(proc, 0,
18                 (void *)PAGE_ALIGN((uintptr_t)buffer->data),
19                 (void *)(((uintptr_t)buffer->data + buffer_size) & PAGE_MASK),
20                 NULL);
21         rb_erase(&buffer->rb_node, &proc->allocated_buffers);
22         buffer->free = 1;
```

generated by haroopad

构体，然后才可以释放它所占用的物理页面。因此，Binder驱动程序提供了函数binder_buffer_lookup根据一个用户空间地址来查询一个内核缓冲区，它的实现如下所示。

kernel/goldfish/drivers/staging/android/binder.c

```
01 static struct binder_buffer *binder_buffer_lookup(
02         struct binder_proc *proc, void __user *user_ptr)
03 {
04         struct rb_node *n = proc->allocated_buffers.rb_node;
05         struct binder_buffer *buffer;
06         struct binder_buffer *kern_ptr;
07
08         kern_ptr = user_ptr - proc->user_buffer_offset
09                 - offsetof(struct binder_buffer, data);
10
11         while (n) {
12                 buffer = rb_entry(n, struct binder_buffer, rb_node);
13                 BUG_ON(buffer->free);
14
15                 if (kern_ptr < buffer)
16                         n = n->rb_left;
17                 else if (kern_ptr > buffer)
18                         n = n->rb_right;
19                 else
20                         return buffer;
21         }
22         return NULL;
23 }
```

Binder驱动程序将一个内核缓冲区的数据传递给目标进程时，它只把数据缓冲区的用户地址传递给目标进程，即将一个binder_buffer结构体的成员变量data所指向的一块数据缓冲区的用户地址传递给目标进程。了解了这个背景知识之后，我们就可以知道，参数user_ptr是一个用户空间地址，并且它指向一个binder_buffer结构体的成员变量data的地址。

第8行和第9行中的子表达式user_ptr – offsetof(struct binder_buffer, data)用来计算一个binder_buffer结构体的用户空间地址，接着再减去目标进程proc的成员变量user_buffer_offset，就得到用户空间地址user_ptr所对应的binder_buffer结构体的内核空间地址了，最后将它保存在变量kern_ptr中。

前面计算得到的binder_buffer结构体的内核空间地址不一定指向一个有效的内核缓冲区，因此，函数就需要对它进行验证。只有验证通过后，才可以将它返回给调用者。在前面的5.1.1小节中介绍进程结构体binder_proc时提到，一个进程的已经分配物理页面的内核缓冲区，是以它的内核空间地址作为关键字保存在进程的已分配内核缓冲区红黑树allocated_buffers中的，因此，如果函数能够根据前面计算得到的binder_buffer结构体的内核空间地址在这个红黑树中找到一个对应的节点，那么就说明该binder_buffer结构体是指向一个有效的内核缓冲区的。第11行到第21行的while循环便是在目标进程proc的已分配内核缓冲区红黑树allocated_buffers中查找与内核空间地址kern_ptr对应的节点。如果能找到，那么第20行就将该节点所对应的一个binder_buffer结构体返回给调用者；否则，第22行就返回一个NULL值给调用者，表示找不到与用户空间地址user_ptr对应的内核缓冲区。

generated by haroopad