본 문서는
http://letsgoustc.spaces.live.com/blog/cns!89AD27DFB5E249BA!443.entry
의 문서를 정리한다.

원저작자는 링크의 Steve guo 이며,
수정자는 nilakantha38@gmail.com 입니다.
수정/재배포 가능하나 원저작자/수정자 명시 바라며,
문서에 대한 의견 및 질문은 nilakantha38@gmail.com 으로 연락 바랍니다.

Service Manager Run 2
0. entry point : framework/base/cmds/service manager/service_manager.c, main() 2
1. binder_loop() 3

Get IServiceManager 5
0. entry point : frameworks/base/libs/binder/IServiceManager.cpp, defaultServiceManager() 5
1. ProcessState::self() 5
2. ProcessState::getContextObject() 6
2.1. ProcessState::getStrongProxyForHandle 6
2.2. BpBinder 7
2.2.1. IPCThreadState::incWeakHandle() 7
3. interface_cast(const sp& obj) 7
3.1. IServiceManager 8
3.2. BpServiceManager 9
4. 결론 9

Generate AudioFlinger Service 11
0. entry point : framework/base/media/mediaserver/main_mediaserver.cpp, main() 11
1. AudioFlinger::instantiate() 11
1.1. AudioFlinger 11
1.2. BBinder::transact() 12
1.3. BnAudioFlinger::onTransact() 13
2. IPCThreadState::joinThreadPool() 13
3. 결론 13

RPC Call IServiceManager::addService 14
0. BpServiceManager::addService() 14
1. Parcel::flatten_binder() 14
2. BpBinder::transact() 15
2.1. IPCThreadState::transact() 15
2.2. IPCThreadState::writeTransationData() 16
2.3. IPCThreadState::waitForResponse() 17
3. 결론 17

Transaction in Binder Kernel Driver 18
0. binder_open() 19
1. binder_ioctl() 21

1.1. binder_thread_write 22
1.1.1. binder_thread_write :: BC_INCREFS 23
1.1.2. binder_thread_write :: BC_TRANSACTION 25
1.1.2.1. binder_transaction() 26
1.2. binder_thread_read() 29
2. 결론 32

Service Manager Handle Add Service 32
0. return to service_manager 32
1. binder_parse() 33
2. svcmgr_handler() 34
3. do_add_service() 35
4. 결론 36

Get IAudioFlinger 36
0. entry point : frameworks/base/libs/binder/IServiceManager.cpp, BpServiceManager::getService() 37
1. BpServiceManager::checkService() 37
2. svcmgr_handler() 37
3. 결론 : 38

IAudioFlinger::setMode() 38
0. RPC 38
1. IPCThreadState::joinThreadPool 39
1.1. executeCommnad() 40
1.1.1. BnAudioFlinger::onTransact() 41
1.1.2. sendReply() 41
2. 결론 : 42

요약 **: 43**
booting 직후, 서비스 매니저의 등록과 AudioFlinger 의 서비스 등록 43
클라이언트 프로세스의 IServiceManager::getService() 44
클라이언트 프로세스의 IAudioFlinger::setMode() 44

Service Manager Run

부팅 직후**, init.rc** 에 의해 **service manager** 가 시작된다.

0. entry point : framework/base/cmds/service manager/ service_manager.c, main()

시작점은 frameworks/base/cmds/servicemanager/service_manager.c 의 main 함수이다.

```
int main(int argc, char **argv)
{
    struct binder_state *bs;
    void *svcmgr = BINDER_SERVICE_MANAGER;
```

```
    bs = binder_open(128*1024);

    if (binder_become_context_manager(bs)) {
        LOGE("cannot become context manager (%s)\n", strerror(errno));
        return -1;
    }

    svcmgr_handle = svcmgr;
    binder_loop(bs, svcmgr_handler);
    return 0;
}
```

```
struct binder_state
{
    int fd;
    void *mapped;
    unsigned mapsize;
};
```

struct binder_state *bs 를 관리하는데, bs = binder_open(128*1024) 호출을 한다.

binder_open() 에서는 bs->fd = open("dev/binder" O_RDWR) 를 한다.

이후, binder_become_context_manager(bs)를 통해

/dev/binder 에게 **BINCER_SET_CONTEXT_MGR ioctl** 을 날려 binder kernel driver 에게 /dev/binder 를 연 서비스 매니저가 서비스 매니저로 동작함을 알린다.

다음으로 svcmgr_handle 에 BINDER_SERVICE_MANAGER 를 넣어 주는데, 이것은 ((void*)0) 로 Define 되어 있다. 이것은 **service_manager 의 핸들**로 사용된다.
추후, 다른 프로세스에서는 service_manager 와 대화하기 위해 이 핸들을 가져야만 한다.

마지막으로 binder_loop(bs, svcmgr_handler) 를 호출하고 main 함수는 끝난다.

1. binder_loop()

binder_loop() 에선 struct binder_write_read bwr 을 통해 binder 에게 명령을 내리고, 명령을 받는다.

```
void binder_loop(struct binder_state *bs, binder_handler func)
{
    int res;
    struct binder_write_read bwr;
    unsigned readbuf[32];

    bwr.write_size = 0;
```

```
    bwr.write_consumed = 0;
    bwr.write_buffer = 0;

    readbuf[0] = BC_ENTER_LOOPER;
    binder_write(bs, readbuf, sizeof(unsigned));

    for (;;) {
        bwr.read_size = sizeof(readbuf);
        bwr.read_consumed = 0;
        bwr.read_buffer = (unsigned) readbuf;

        res = ioctl(bs->fd, BINDER_WRITE_READ, &bwr);

        if (res < 0) {
            LOGE("binder_loop: ioctl failed (%s)\n", strerror(errno));
            break;
        }

        res = binder_parse(bs, 0, readbuf, bwr.read_consumed, func);
        if (res == 0) {
            LOGE("binder_loop: unexpected reply?!\n");
            break;
        }
        if (res < 0) {
            LOGE("binder_loop: io error %d %s\n", res, strerror(errno));
            break;
        }
    }
}
```

초기에 BC_ENTER_LOOP 커맨드를 쓰고, 루프에 들어간다.

루프에서는 res = ioctl(bs->fd, BINDER_WRITE_READ, &bwr) 을 통해 지속적으로 **다른 process 로부터의 요청을 읽어들인다.**
일반적으로 여기서 **블록 되어 있으며, 바인더로부터 데이터가 들어온다면, 그 때 깨어날** 것이다.

읽어들였다면 binder_parse() 를 호출한다.

> binder_parse() 에서는 bwr.read_buffer 의 내용을 cmd 로 받아들여 BR_TRANSACTION, BR_REPLY, BR_DEAD_BINDER 등의 커맨드를 swtich~case 방식으로 처리한다.

이제, 서비스 매니저는 준비됐다.

Get IServiceManager

서비스 매니저와 대화하기 위해 IServiceManager 를 얻어와야 한다.

0. entry point : frameworks/base/libs/binder/ IServiceManager.cpp, defaultServiceManager()

IServiceManager 를 얻기 위해선 defaultServiceManager() 를 호출한다. 이는 frameworks/base/ libs/binder/IServiceManager.cpp 에 정의 되어 있다.
이 함수는 **sp<IServiceManager> 를 리턴**하며, param 은 받지 않는다.
여기선, libutil 에 정의 되어 있는 심볼이며, 프로세스 마다 unique 한 값을 갖는,
sp<IServiceManager> 타입의 gDefaultServiceManager 를 리턴한다.
처음엔 존재하지 않으므로, **ProcessState::self()->getContextObject(NULL) 을 return** 한다.

```
sp<IServiceManager> defaultServiceManager()
{
   if (gDefaultServiceManager != NULL) return gDefaultServiceManager;

   {
      AutoMutex _l(gDefaultServiceManagerLock);
      if (gDefaultServiceManager == NULL) {
         gDefaultServiceManager = interface_cast<IServiceManager>(
            ProcessState::self()->getContextObject(NULL));
      }
   }

   return gDefaultServiceManager;
}
```

1. ProcessState::self()

frameworks/base/libs/binder/ProcessState.cpp 에 정의 되어 있다.
Process 마다 단 하나의 ProcessState instance 를 갖는다.
slef() 에서는 **ProcessState 인스턴스인 gProcess 를 싱글턴으로 생성해 리턴**한다.
ProcessState 는 **생성자에서 /dev/binder 를 열어**, 그 file descriptor 를 mDriverFD 멤버변수로 갖는다.
ProcessState::ProcessState() : mDriverFD(open_driver())

2. ProcessState::getContextObject()

return getStrongProxyForHandle(0) 한다.
인자로 0을 주었다는 점을 기억하라. 0 은 **서비스 매니저에서 핸들로 등록**했던 값이다.(**#define BINDER_SERVICE_MANAGER ((void*)0)**)

2.1. ProcessState::getStrongProxyForHandle

```
sp<IBinder> ProcessState::getStrongProxyForHandle(int32_t handle)
{
    sp<IBinder> result;

    AutoMutex _l(mLock);

    handle_entry* e = lookupHandleLocked(handle);

    if (e != NULL) {
        // We need to create a new BpBinder if there isn't currently one, OR we
        // are unable to acquire a weak reference on this current one.  See comment
        // in getWeakProxyForHandle() for more info about this.
        IBinder* b = e->binder;
        if (b == NULL || !e->refs->attemptIncWeak(this)) {
            b = new BpBinder(handle);
            e->binder = b;
            if (b) e->refs = b->getWeakRefs();
            result = b;
        } else {
            // This little bit of nastyness is to allow us to add a primary
            // reference to the remote proxy when this team doesn't have one
            // but another team is sending the handle to us.
            result.force_set(b);
            e->refs->decWeak(this);
        }
    }

    return result;
}
```

인자로 받은 handle 값으로 lookupHandleLocked()를 호출해, **IBinder *binder** 와 **RefBase::Weakref_type* refs** 를 멤버로 갖는 구조체 handle_entry 를 얻어 온다.
lookupHandleLocked() 에서는, **vector<handle_entry>** 인 mHandleToObject 로부터 해당 핸들의 인덱스에 맞는 handle_entry 를 리턴한다.

이후, 해당 handle_entry 의 멤버 binder 를 리턴하는데, NULL 인 경우(처음),
BpBinder 를 생성(생성자에 인자로 받은 핸들 값 - 0 - 을 넣는다)해 리턴한다.

2.2. BpBinder

Base Proxy Binder. remote binder 객체를 위한 base proxy class 이다.
생성자에서는 **핸들 값을 받아 mHandle**, mAlive, mObitsSent 등의 멤버 변수를 초기화 하고, extendObjectLifetime() 을 호출해 레퍼런스 플래그에 OBJECT_LIFETIME_WEAK 를 설치하고, IPCThreadState::self()->incWeakHandle(handle) 을 한다.

2.2.1. IPCThreadState::incWeakHandle()

여기선 output buffer 인, Parcel 형 멤버 변수 mOut 에 mOut.writeInt32() 를 이용해 BC_INCREFS 와 handle 을 써넣는다.

3. interface_cast(const sp<IBinder>& obj)

이제, entry point 에서는 BpBinder 인스턴스를 받았다. 이걸 interface_cast() 로 변환 시킨 후, gDefaultService 에 집어 넣는다.
여기선 return IServiceManager::asInterface(obj) 를 한다.

```
sp<IServiceManager> defaultServiceManager()
{
   if (gDefaultServiceManager != NULL) return gDefaultServiceManager;

   {
      AutoMutex _l(gDefaultServiceManagerLock);
      if (gDefaultServiceManager == NULL) {
         gDefaultServiceManager = interface_cast<IServiceManager>(
            ProcessState::self()->getContextObject(NULL));
      }
   }

   return gDefaultServiceManager;
}
```

```
template<typename INTERFACE>
inline sp<INTERFACE> interface_cast(const sp<IBinder>& obj)
{
   return INTERFACE::asInterface(obj);
}
```

3.1. IServiceManager

IServiceManager 클래스는 IServiceManager.h 파일에서 구현되어 있다.

```
class IServiceManager : public IInterface
{
public:
   DECLARE_META_INTERFACE(ServiceManager);

   /**
    * Retrieve an existing service, blocking for a few seconds
    * if it doesn't yet exist.
    */
   virtual sp<IBinder>         getService( const String16& name) const = 0;

   /**
    * Retrieve an existing service, non-blocking.
    */
   virtual sp<IBinder>         checkService( const String16& name) const = 0;

   /**
    * Register a service.
    */
   virtual status_t            addService( const String16& name,
                                    const sp<IBinder>& service) = 0;

   /**
    * Return list of all existing services.
    */
   virtual Vector<String16>    listServices() = 0;

   enum {
      GET_SERVICE_TRANSACTION = IBinder::FIRST_CALL_TRANSACTION,
      CHECK_SERVICE_TRANSACTION,
      ADD_SERVICE_TRANSACTION,
      LIST_SERVICES_TRANSACTION,
   };
};
```

클래스 선언 부에서 DECLARE_META_INTERFACE(ServiceManager) 매크로를 사용한다.
매크로의 내용은 다음과 같다.
여기서 INTERFACE 는 곧 ServiceManager 이므로,
sp<IServiceManager> **asInterface**(const sp<IBinder>& obj); 라는 메소드를 선언한다.

```
#define DECLARE_META_INTERFACE(INTERFACE)                           \
   static const String16 descriptor;                                \
   static sp<I##INTERFACE> asInterface(const sp<IBinder>& obj);     \
   virtual const String16& getInterfaceDescriptor() const;          \
   I##INTERFACE();                                                  \
   virtual ~I##INTERFACE();
```

IServiceManager.cpp 에서는 IServiceManager.h 를 include 하며, 파일 내에서

IMPLEMENT_META_INTERFACE(ServiceManager, "android.os.IServiceManager")
를 통해
DECLARE_META_INTERFACE() 에서 정의한 멤버함수를 구현한다.
IMPLEMENT_META_INTERFACE 매크로의 내용은 다음과 같다.

```
#define IMPLEMENT_META_INTERFACE(INTERFACE, NAME)                    \
   const String16 I##INTERFACE::descriptor(NAME);                    \
   const String16& I##INTERFACE::getInterfaceDescriptor() const {    \
      return I##INTERFACE::descriptor;                               \
   }                                                                 \
   sp<I##INTERFACE> I##INTERFACE::asInterface(const sp<IBinder>& obj)  \
   {                                                                 \
      sp<I##INTERFACE> intr;                                         \
      if (obj != NULL) {                                             \
         intr = static_cast<I##INTERFACE*>(                          \
            obj->queryLocalInterface(                                \
                  I##INTERFACE::descriptor).get());                  \
         if (intr == NULL) {                                         \
            intr = new Bp##INTERFACE(obj);                           \
         }                                                           \
      }                                                              \
      return intr;                                                   \
   }                                                                 \
   I##INTERFACE::I##INTERFACE() { }                                  \
   I##INTERFACE::~I##INTERFACE() { }
```

INTERFACE 는 ServiceManager, NAME 은 "android.os.IServiceManager" 이므로,
IServiceManager::asInterface() 는, 인자로 받은 객체를 생성자의 인자(우리는 여기에 새로 생성한 **BpBinder** 를 넣었다.)로 하여 새로 생성한 **BpServiceManager 클래스의 인스턴스를 리턴**한다.
BpServiceManager 클래스의 생성자는 자신의 super class인 BpInterface<IServiceManager>() 를 호출하고, 여기선 또 자신의 super class인 BpRefBase() 를 호출하여, BpRefBase 의 **mRemote 에 인자 - 0을 핸들값으로 갖는 BpBinder - 를 집어 넣는다**.
결국, BpServiceManager 의 mRemote 는 핸들 값을 갖는 BpBinder 이다.

3.2. BpServiceManager

BpServiceManager 는 IServiceManager.cpp 에 구현 되어 있다.
BpInterface 를 상속하며, **getService, checkService, addService, listServices** 의 네개 메소드를 구현하는데,
구현 내부에서는 **Parcel 에 적당히 데이터를 써 넣은 후, mRemote - 이건 앞서 살펴 본, 핸들 0 를 갖는 BpBinder 이다. - 에게 transact()** 하는 방식이다.

4. 결론

결국, defaultServiceManager() 에서는 BpServiceManager 클래스를 받아 오게 된다.
최종적으로 **BpServiceManager** 는 원격의 **BnServiceManager** 를 위한 프록시로 동작하며,

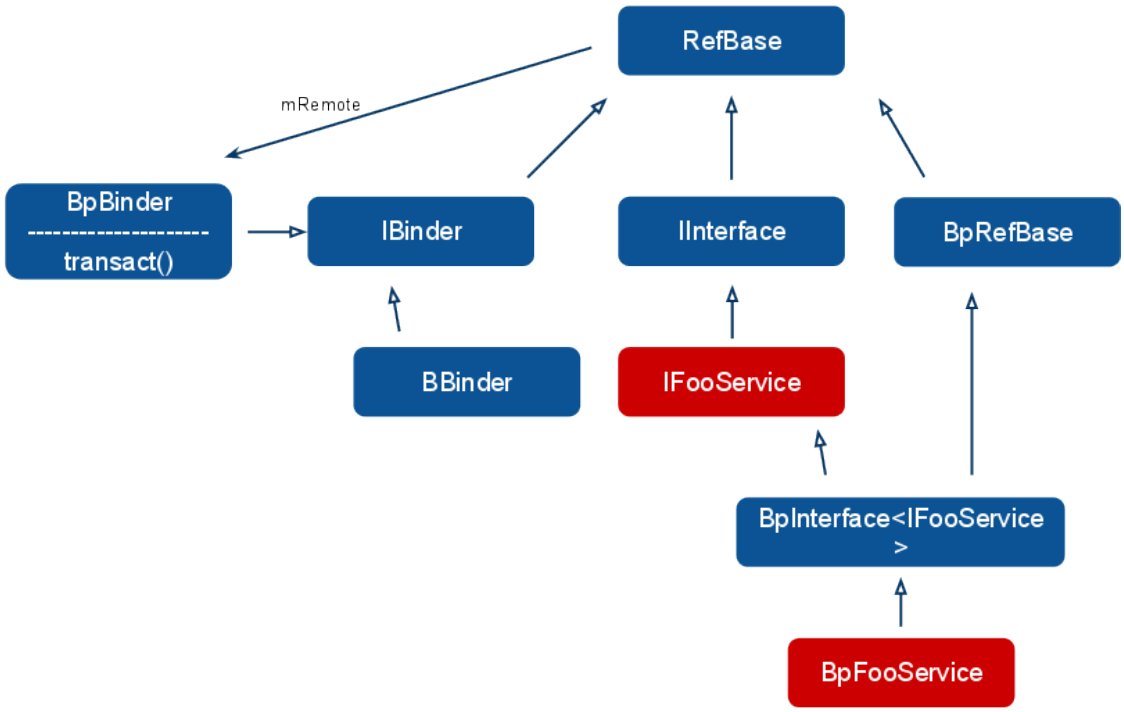
사용자가 IServiceManager 로 하게 되는 동작은 BpServiceManager 의 virtual 함수로 동작한다.

원격 객체를 위한 프록시 객체를 어떻게 얻어 오는지 알아 보았다.

IFunnyTest 와 같은, 나만의 서비스를 만들고자 한다면,
DECLARE_META_INTERFACE(FunnyTest) 와 IMPLEMENT_META_INTERFACE(FunnyTest, "my unique name") **매크로를 인터페이스 소스 파일에 넣고,**
BpFunnyTest **클래스를** 구현해야 한다.

이제, 우리는 서비스 매니저를 사용 할 수 있다.

we wanna get IFooService for FooService



Generate AudioFlinger Service

media_server 프로그램에서 audioFlinger service 를 시작한다.

0. entry point : framework/base/media/mediaserver/main_mediaserver.cpp, main()

media_server 의 시작점이다. 여기서 AudioFlinger::instantiate() 를 통해 AudioFlinger 서비스를 시작시킨다. 코드는 다음과 같다.

```
int main(int argc, char** argv)
{
    sp<ProcessState> proc(ProcessState::self());
    sp<IServiceManager> sm = defaultServiceManager();
    LOGI("ServiceManager: %p", sm.get());
    AudioFlinger::instantiate();
    MediaPlayerService::instantiate();
    CameraService::instantiate();
    AudioPolicyService::instantiate();
    ProcessState::self()->startThreadPool();
    IPCThreadState::self()->joinThreadPool();
}
```

1. AudioFlinger::instantiate()

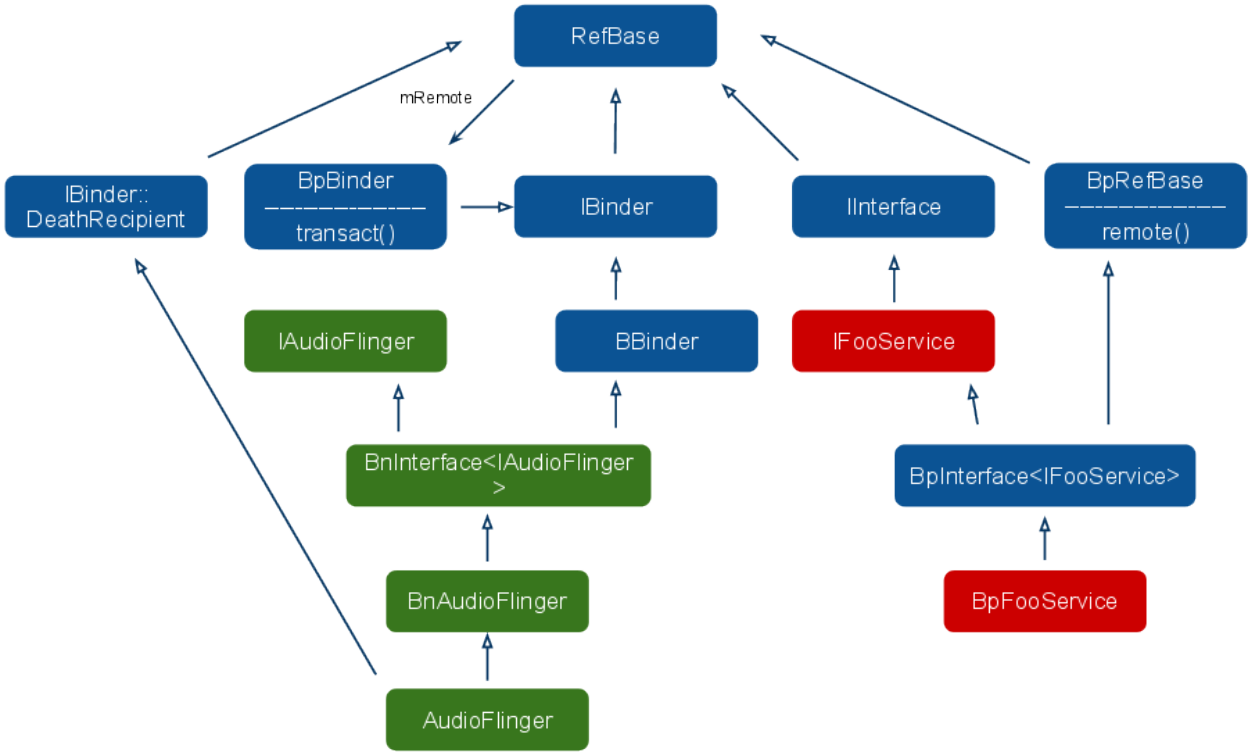
아까 만든 **service manager** 에게 **addService()** 요청을 한다.

```
void AudioFlinger::instantiate() {
    defaultServiceManager()->addService(
            String16("media.audio_flinger"), new AudioFlinger());
}
```

1.1. AudioFlinger

AudioFlinger 는 BnAudioFlinger 를 상속받는다.
BnAudioFlinger 는 BnInterface 를 상속받는다.
BnInterface 는 BBinder 를 상속받는다.
BBinder 는 IBinder 를 상속 받는다.

IAudioFlinger is one of IFooService.



1.2. BBinder::transact()

defaultServiceManager() 에 대해서는 앞서 확인했다.
mRemote->transact() 로 귀결되고, 바인더 드라이버를 거쳐 거쳐, BBinder::transact()로 데이터를 넘긴다.

```
status_t BBinder::transact(
    uint32_t code, const Parcel& data, Parcel* reply, uint32_t flags)
{
    data.setDataPosition(0);

    status_t err = NO_ERROR;
    switch (code) {
        case PING_TRANSACTION:
            reply->writeInt32(pingBinder());
            break;
        default:
            err = onTransact(code, data, reply, flags);
            break;
    }
```

```
    if (reply != NULL) {
        reply->setDataPosition(0);
    }

    return err;
}
```

transact 에서는 인자로 받은 code 값이 PING_TRANSACTION 이라면 reply 에 대답을 해주고,
ping 이 아닌 code 라면 onTransact() 를 호출한다.
onTransact() 는 BnAudioFlinger 에서 구현 되어 있다.

1.3. BnAudioFlinger::onTransact()

code 값에 의한 switch ~ case 방식으로 OPEN_RECORD, SAMPLE_RATE 등의 명령에 대한 동작을 구현하고 있다.
동작에 맞게, Parcel 인자로 넘겨받은 data 로부터 사용자의 입력을 받고 reply 에 대답을 써 넣는 방식이다.
default 에 대해서는 BBinder::onTransact() 로 넘긴다.

2. IPCThreadState::joinThreadPool()

instantiate 가 끝났다.
joinThreadPool() 에서는 **루프**를 돌며 talkWithDriver() 를 통해 다른 프로세스로부터의 통신을 받고, 동작한다.

> talkWithDriver() 에서는 mProcess->mDriverFD (ProcessState 생성자에서 /dev/binder 를 열고 디스크립터를 얻었다.) 를 통해 ioctl() 하여 입력을 받는다.

받은 입력에서 cmd 를 얻어 내고, **executeCommand()**를 통해 요청 사항을 처리하는데,
여기선 cmd 에 switch ~ case 방식으로, BR_ACQUIRE, BR_RELEASE 등등의 동작에 대해 처리한다.

> BR_TRANSACTION 의 경우, 사용자의 입력으로부터 **BBinder 를 얻어와, 해당 객체의 transact() 를 호출**한다.

3. 결론

이제, 나만의 서비스 IFunnyTest 를 만들고자 한다면, BnFunnyTest 클래스를 구현하고, 나만의 서비스를 돌린 후(instantiate), IPCThreadState::joinThreadPool 을 호출해야 한다.

RPC Call IServiceManager::addService

IServiceManager::addService() 는 결국 BpServiceManager::addService() 에 구현 되어 있다.

0. BpServiceManager::addService()

```
    virtual status_t addService(const String16& name, const sp<IBinder>& service)
    {
        Parcel data, reply;
        data.writeInterfaceToken(IServiceManager::getInterfaceDescriptor());
        data.writeString16(name);
        data.writeStrongBinder(service);
        status_t err = remote()->transact(ADD_SERVICE_TRANSACTION, data,
&reply);
        return err == NO_ERROR ? reply.readInt32() : err;
    }
```

data.writeStrongBinder() 를 통해, 등록하고자 하는 **service** 를 **Parcel** 에 집어 넣는다.
writeStrongBinder() 에선 flatten_binder 를 호출한다.

1. Parcel::flatten_binder()

서비스의 바인더를 직렬화 하고 **Binder command** 를 만든다.
등록하고자 하는 서비스의
flat_binder_object 변수를 만든 후,
type에 BINDER_TYPE_BINDER,
binder 에 등록하고자 하는 서비스 바인더의 getWeakRefs(),
cookie 에 등록하고자 하는 서비스 바인더(sp<IBinder>)를 넣는다.

이제, Parcel 인 data 에는
|"서비스 매니저 이름" | "추가하고자 하는 서비스 이름" | "flat_binder_object(type : BINDER_TYPE_BINDER, binder : 추가하고자 하는 서비스**(IBinder)**의 **weak reference, cookie :** 추가하고자 하는 서비스**(sp<IBinder>))" |**
의 형태를 띈다.

2. BpBinder::transact()

이제, **remote()->transact()** 를 통해 Parcel 에 집어넣은 서비스를 넘긴다.
remote() 는**, mRemote** 인 **BpBinder** 객체를 리턴한다.
여기선, IPCThreadState::self()->transact(mHandle, code, data, reply, flags) 를 한다.
mHandle 은, 서비스 매니저이니 0일 것이다.

2.1. IPCThreadState::transact()

```
status_t IPCThreadState::transact(int32_t handle,
                                  uint32_t code, const Parcel& data,
                                  Parcel* reply, uint32_t flags)
{
   status_t err = data.errorCheck();

    flags |= TF_ACCEPT_FDS;

   IF_LOG_TRANSACTIONS() {
      TextOutput::Bundle _b(alog);
+--  3 줄: alog << "BC_TRANSACTION thr " << (void*)pthread_self() << " / hand "--------
------------------------------------------------------------------
   }

   if (err == NO_ERROR) {
      LOG_ONEWAY(">>>> SEND from pid %d uid %d %s", getpid(), getuid(),
         (flags & TF_ONE_WAY) == 0 ? "READ REPLY" : "ONE WAY");
      err = writeTransactionData(BC_TRANSACTION, flags, handle, code, data,
NULL);
   }

+--  4 줄: if (err != NO_ERROR) {-----------------------------------------------------------
-------------------------------------------------------

   if ((flags & TF_ONE_WAY) == 0) {
      if (reply) {
         err = waitForResponse(reply);
      } else {
         Parcel fakeReply;
         err = waitForResponse(&fakeReply);
      }

      IF_LOG_TRANSACTIONS() {
+--  5 줄: TextOutput::Bundle _b(alog);-----------------------------------------------------
-------------------------------------------------------
      }
```

```
    } else {
        err = waitForResponse(NULL, NULL);
    }

    return err;
}
```

여기선 writeTransactionData(BC_TRANSACTION, flags, handle, code, data, NULL); 를 하고, waitForResponse() 하여 기다린다.

2.2. IPCThreadState::writeTransationData()

```
status_t IPCThreadState::writeTransactionData(int32_t cmd, uint32_t binderFlags,
    int32_t handle, uint32_t code, const Parcel& data, status_t* statusBuffer)
{
    binder_transaction_data tr;

    tr.target.handle = handle;
    tr.code = code;
    tr.flags = binderFlags;

    const status_t err = data.errorCheck();
    if (err == NO_ERROR) {
        tr.data_size = data.ipcDataSize();
        tr.data.ptr.buffer = data.ipcData();
        tr.offsets_size = data.ipcObjectsCount()*sizeof(size_t);
        tr.data.ptr.offsets = data.ipcObjects();
    } else if (statusBuffer) {
        tr.flags |= TF_STATUS_CODE;
        *statusBuffer = err;
        tr.data_size = sizeof(status_t);
        tr.data.ptr.buffer = statusBuffer;
        tr.offsets_size = 0;
        tr.data.ptr.offsets = NULL;
    } else {
        return (mLastError = err);
    }

    mOut.writeInt32(cmd);
    mOut.write(&tr, sizeof(tr));

    return NO_ERROR;
}
```

binder_transaction_data tr 에 handle 을 넣어 줌으로써, transaction target 이 어디가 될지를 명시해준다.

또, tr.code 에 인자로 전달받은 code 를 넣는다. 이 경우엔 앞서 addService() 에서 보낸
ADD_SERVICE_TRANSACTION 이 될 것이다.
실제 transaction 은 Parcel 인 mOut 에 Binder 에 보낼 커맨드와 데이터를 보낸다.

이제, mOut 의 형태는 다음과 같다.
| cmd(이 경우, BC_TRANSACTION) | binder_transaction_data |
여기서 binder_transaction_data 는,
| target handle | code(이 경우, ADD_SERVICE_TRANSACTION) | flags | data packet size | data packet |
여기서 data packet 은,
|"서비스 매니저 이름" | "추가하고자 하는 서비스 이름" | "flat_binder_object(type : BINDER_TYPE_BINDER, binder : 추가하고자 하는 서비스(IBinder)의 weak reference, cookie : 추가하고자 하는 서비스(sp<IBinder>))" |
의 형태를 갖는 Parcel 이다.

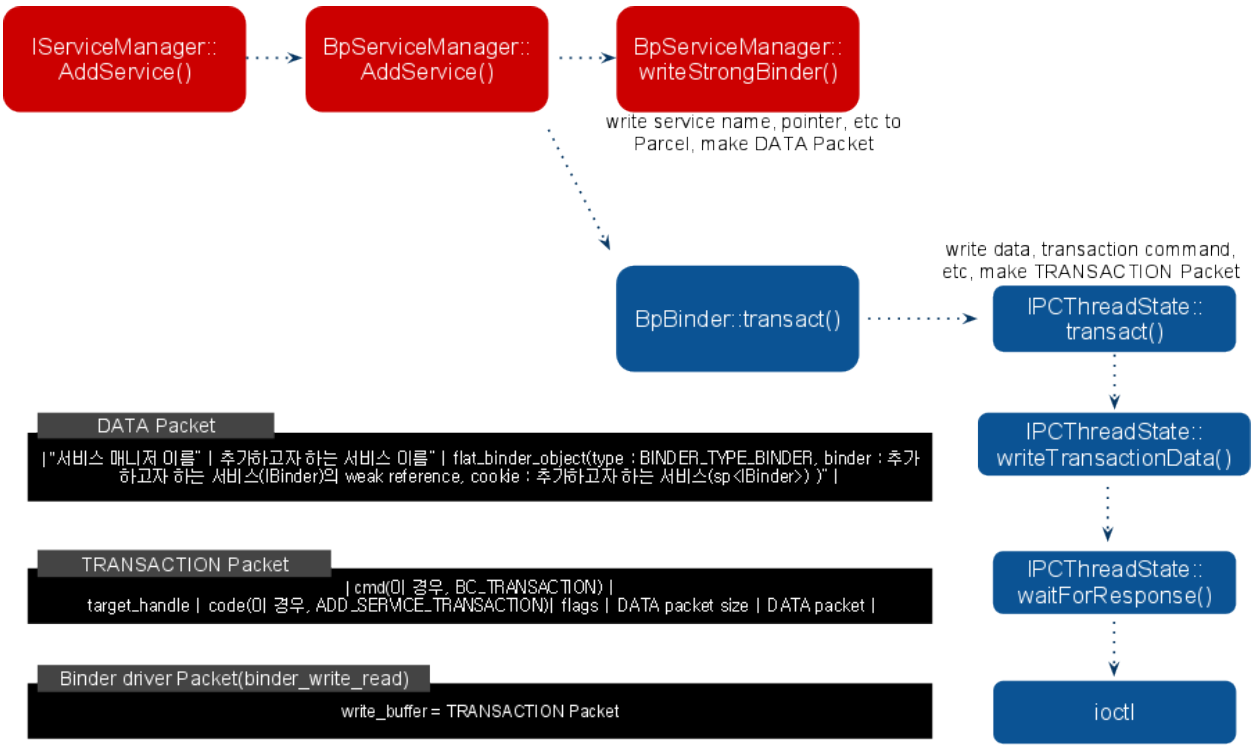
2.3. IPCThreadState::waitForResponse()

다음으로, waitForResponse() 에서는 talkWithDriver()를 호출하는데,
 talkWithDriver() 에서는 binder_write_read 구조체 변수를 만들어, 앞서 커맨드와 데이터를 써 둔 Parcel, **mOut 의 값을 bwr.write_buffer 에 넣고**서는
/dev/binder 에게 **BINDER_WRITE_READ ioctl** 을 날려서 드라이버에게 실제 데이터를 보낸다.

```
if (ioctl(mProcess->mDriverFD, BINDER_WRITE_READ, &bwr) >= 0)
```

3. 결론

Proxy 객체(Bp 시리즈) 에서 RPC 를 위한, 커맨드와 데이터를 담은 데이터 패킷을 만들고,
BINDER_WRITE_READ ioctl 호출을 통해 데이터 패킷을 transaction 패킷으로 만들고,
이를 binder_read_write 구조체에 넣어서는 실제 binder 커널 드라이버에게 넘긴다.
패킷은 RPC의 경우, BC_TRANSACTION 패킷 타입을 갖는다.
나만의 서비스를 만들고자 한다면, IServiceManager::addSerivce 를 통해, 서비스 매니저에게 자신의 서비스를 등록해야 한다.



Transaction in Binder Kernel Driver

이번엔 바인더 드라이버 내부를 분석한다.
드라이버 소스는 kernel/drivers/staging/android/binder.c 에 있다.

바인더 드라이버는 /dev/binder 디바이스 파일에의 접근을 제어하게 되는데,
이는, static 으로 실행되는 **device_initcall()** 에서 등록한 콜백 함수 포인터 binder_init 에서
misc_register() 를 하여 binder_fops 를 등록하여
어느 동작에 어느 바인더 드라이버 함수가 사용될지 등록한다.

```
static struct file_operations binder_fops = {
    .owner = THIS_MODULE,
    .poll = binder_poll,
    .unlocked_ioctl = binder_ioctl,
    .mmap = binder_mmap,
    .open = binder_open,
    .flush = binder_flush,
    .release = binder_release,
};
```

binder 디바이스 파일은 /dev/binder 이지만, **/proc/binder/ 하위에서는 binder 디바이스 파일의 상태를 확인 할 수 있다.**
여기엔 transaction_log, failed_transaction_log, transactions, stats, state, proc/ 이 존재한다.

0. binder_open()

어떤 프로세스가 /dev/binder 를 open 하게 되면, binder_open() 이 호출된다.

```
static int binder_open(struct inode *nodp, struct file *filp)
{
   struct binder_proc *proc;

   if (binder_debug_mask & BINDER_DEBUG_OPEN_CLOSE)
      printk(KERN_INFO "binder_open: %d:%d\n", current->group_leader->pid, current-
>pid);

   proc = kzalloc(sizeof(*proc), GFP_KERNEL);
   if (proc == NULL)
      return -ENOMEM;
   get_task_struct(current);
   proc->tsk = current;
   INIT_LIST_HEAD(&proc->todo);
   init_waitqueue_head(&proc->wait);
   proc->default_priority = task_nice(current);
   mutex_lock(&binder_lock);
   binder_stats.obj_created[BINDER_STAT_PROC]++;
   hlist_add_head(&proc->proc_node, &binder_procs);
   proc->pid = current->group_leader->pid;
   INIT_LIST_HEAD(&proc->delivered_death);
   filp->private_data = proc;
   mutex_unlock(&binder_lock);

   if (binder_proc_dir_entry_proc) {
      char strbuf[11];
      snprintf(strbuf, sizeof(strbuf), "%u", proc->pid);
      remove_proc_entry(strbuf, binder_proc_dir_entry_proc);
      create_proc_read_entry(strbuf, S_IRUGO, binder_proc_dir_entry_proc,
binder_read_proc_proc, proc);
   }

   return 0;
}
```

여기선 binder_proc 구조체를 생성해, 구조체의 tsk, default_priority, pid 등을 집어 넣고,
binder 참조 횟수 증가 시키고, **binder_procs 링크드 리스트 구조(binder_procs)에 생성한 구조체의 proc_node 를 추가**시킨다.

마지막으로, **open 하기 위해 사용한 파일 구조체 포인터의 멤버 private_data 에 binder_proc 구조체를 집어넣는다.**

이제, 해당 파일 포인터를 사용하는 클라이언트 프로세스가 ioctl 을 하게 되면,
인자로 넘어오는 **파일 포인터에서 private_data 멤버를 얻어와, 해당 클라이언트 프로세스에게 할당한 binder_proc 을 얻어와**서,
해당 클라이언트 프로세스의 tsk, pid, binder 참조 횟수 등을 알 수 있다.

struct file 은 다음과 같다.

```
struct file {
   /*
    * fu_list becomes invalid after file_free is called and queued via
    * fu_rcuhead for RCU freeing
    */
   union {
      struct list_head    fu_list;
      struct rcu_head     fu_rcuhead;
   } f_u;
   struct path     f_path;
#define f_dentry    f_path.dentry
#define f_vfsmnt    f_path.mnt
   const struct file_operations    *f_op;
   atomic_long_t       f_count;
   unsigned int        f_flags;
   fmode_t         f_mode;
   loff_t          f_pos;
   struct fown_struct  f_owner;
   const struct cred   *f_cred;
   struct file_ra_state    f_ra;

   u64         f_version;
#ifdef CONFIG_SECURITY
   void            *f_security;
#endif
   /* needed for tty driver, and maybe others */
   void            *private_data;

#ifdef CONFIG_EPOLL
   /* Used by fs/eventpoll.c to link all the hooks to this file */
   struct list_head    f_ep_links;
   spinlock_t      f_ep_lock;
#endif /* #ifdef CONFIG_EPOLL */
   struct address_space    *f_mapping;
#ifdef CONFIG_DEBUG_WRITECOUNT
   unsigned long f_mnt_write_state;
#endif
};
```

struct binder_proc 의 구조는 아래와 같다.

```
struct binder_proc {
    struct hlist_node proc_node;
    struct rb_root threads;
    struct rb_root nodes;
    struct rb_root refs_by_desc;
    struct rb_root refs_by_node;
    int pid;
    struct vm_area_struct *vma;
    struct task_struct *tsk;
    struct files_struct *files;
    struct hlist_node deferred_work_node;
    int deferred_work;
    void *buffer;
    ptrdiff_t user_buffer_offset;

    struct list_head buffers;
    struct rb_root free_buffers;
    struct rb_root allocated_buffers;
    size_t free_async_space;

    struct page **pages;
    size_t buffer_size;
    uint32_t buffer_free;
    struct list_head todo;
    wait_queue_head_t wait;
    struct binder_stats stats;
    struct list_head delivered_death;
    int max_threads;
    int requested_threads;
    int requested_threads_started;
    int ready_threads;
    long default_priority;
};
```

1. binder_ioctl()

어떤 프로세스가 /dev/binder 에게 ioctl 을 할 때마다 binder_ioctl 이 호출된다. 인자로 파일 포인터와 커맨드, 아규먼트를 받는다.
앞서 talkWithDriver 에서 알아봤듯 **마지막 아규먼트는 곧 사용자가 보내는 binder_write_read 구조체**였다.

인자로 받은 **파일 포인터로부터 binder_proc 포인터를 얻어 오고(binder_proc *proc = filp->private_data)**,
여기서 thread 또한 얻어온다.

```
    case BINDER_WRITE_READ: {
        struct binder_write_read bwr;
        if (size != sizeof(struct binder_write_read)) {
            ret = -EINVAL;
```

```
            goto err;
        }
        if (copy_from_user(&bwr, ubuf, sizeof(bwr))) {
            ret = -EFAULT;
            goto err;
        }
        if (binder_debug_mask & BINDER_DEBUG_READ_WRITE)
            printk(KERN_INFO "binder: %d:%d write %ld at %08lx, read %ld at %08lx\n",
                  proc->pid, thread->pid, bwr.write_size, bwr.write_buffer, bwr.read_size,
bwr.read_buffer);
        if (bwr.write_size > 0) {
            ret = binder_thread_write(proc, thread, (void __user *)bwr.write_buffer,
bwr.write_size, &bwr.write_consumed);
            if (ret < 0) {
                bwr.read_consumed = 0;
                if (copy_to_user(ubuf, &bwr, sizeof(bwr)))
                    ret = -EFAULT;
                goto err;
            }
        }
        if (bwr.read_size > 0) {
            ret = binder_thread_read(proc, thread, (void __user *)bwr.read_buffer,
bwr.read_size, &bwr.read_consumed, filp->f_flags & O_NONBLOCK);
            if (!list_empty(&proc->todo))
                wake_up_interruptible(&proc->wait);
            if (ret < 0) {
                if (copy_to_user(ubuf, &bwr, sizeof(bwr)))
                    ret = -EFAULT;
                goto err;
            }
        }
        if (binder_debug_mask & BINDER_DEBUG_READ_WRITE)
            printk(KERN_INFO "binder: %d:%d wrote %ld of %ld, read return %ld of %ld\n",
                  proc->pid, thread->pid, bwr.write_consumed, bwr.write_size,
bwr.read_consumed, bwr.read_size);
        if (copy_to_user(ubuf, &bwr, sizeof(bwr))) {
            ret = -EFAULT;
            goto err;
        }
        break;
    }
```

먼저, 사용자가 보낸 binder_write_read 구조체를 읽어 들이고는, write 명령을 먼저 처리한다.

1.1. binder_thread_write

binder_thread_write 에서는 write 버퍼로부터 사용자 커맨드를 받아, 해당 커맨드를 수행한다.

```
int
```

```
binder_thread_write(struct binder_proc *proc, struct binder_thread *thread,
        void __user *buffer, int size, signed long *consumed)
{
   uint32_t cmd;
   void __user *ptr = buffer + *consumed;
   void __user *end = buffer + size;

   while (ptr < end && thread->return_error == BR_OK) {
      if (get_user(cmd, (uint32_t __user *)ptr))
         return -EFAULT;
      ptr += sizeof(uint32_t);
      if (_IOC_NR(cmd) < ARRAY_SIZE(binder_stats.bc)) {
         binder_stats.bc[_IOC_NR(cmd)]++;
         proc->stats.bc[_IOC_NR(cmd)]++;
         thread->stats.bc[_IOC_NR(cmd)]++;
      }
      switch (cmd) {
      case BC_INCREFS:
      case BC_ACQUIRE:
      case BC_RELEASE:
      case BC_DECREFS: {

         ...

      default:
         printk(KERN_ERR "binder: %d:%d unknown command %d\n", proc->pid, thread-
>pid, cmd);
         return -EINVAL;
      }
      *consumed = ptr - buffer;
   }
   return 0;
}
```

1.1.1. binder_thread_write :: BC_INCREFS

BC_INCREFS 에 대한 부분을 보자.

```
      switch (cmd) {
      case BC_INCREFS:
      case BC_ACQUIRE:
      case BC_RELEASE:
      case BC_DECREFS: {
         uint32_t target;
         struct binder_ref *ref;
         const char *debug_string;

         if (get_user(target, (uint32_t __user *)ptr))
            return -EFAULT;
```

```
        ptr += sizeof(uint32_t);
        if (target == 0 && binder_context_mgr_node &&
            (cmd == BC_INCREFS || cmd == BC_ACQUIRE)) {
            ref = binder_get_ref_for_node(proc,
                      binder_context_mgr_node);
            if (ref->desc != target) {
                binder_user_error("binder: %d:"
                    "%d tried to acquire "
                    "reference to desc 0, "
                    "got %d instead\n",
                    proc->pid, thread->pid,
                    ref->desc);
            }
        } else
            ref = binder_get_ref(proc, target);
        if (ref == NULL) {
            binder_user_error("binder: %d:%d refcou"
                "nt change on invalid ref %d\n",
                proc->pid, thread->pid, target);
            break;
        }
        switch (cmd) {
        case BC_INCREFS:
            debug_string = "IncRefs";
            binder_inc_ref(ref, 0, NULL);
            break;
        case BC_ACQUIRE:

            ...

        default:
            debug_string = "DecRefs";
            binder_dec_ref(ref, 0);
            break;
        }
        if (binder_debug_mask & BINDER_DEBUG_USER_REFS)
            printk(KERN_INFO "binder: %d:%d %s ref %d desc %d s %d w %d for node
%d\n",
                    proc->pid, thread->pid, debug_string, ref->debug_id, ref->desc, ref-
>strong, ref->weak, ref->node->debug_id);
        break;
    }
```

system_manager 가 생성 초기에 BINDER_SET_CONTEXT_MGR ioctl 을 날렸고(이 때 binder_node 구조체인 **binder_context_mgr_node**를 생성한다)
binder 드라이버와 대화 하는 건 서비스 매니저이니, target 은 0일 것이다.
때문에, 여기서 동작은 사용자 binder_write_read 구조체로부터 reference 를 얻어오고, 이를 이용해 binder_inc_ref() 를 이용해, weak reference(두번째 인자가 0라면 weak reference 증가시킨다) 참조회수를 증가시킨다.

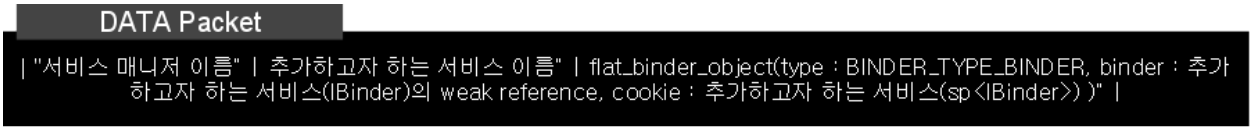
1.1.2. binder_thread_write :: BC_TRANSACTION

다음으로, BC_TRANSACTION 부분을 보겠다.

```
        case BC_TRANSACTION:
        case BC_REPLY: {
            struct binder_transaction_data tr;

            if (copy_from_user(&tr, ptr, sizeof(tr)))
                return -EFAULT;
            ptr += sizeof(tr);
            binder_transaction(proc, thread, &tr, cmd == BC_REPLY);
            break;
        }
```

binder_read_write 구조체로 받은 사용자 입력의 write_buffer 에서 binder_transaction_data 구조체로 **Transaction Packet** 을 얻어와, binder_transaction 에게 넘긴다.
결국 binder_transaction 에서 모든 일을 한다.
binder_transaction_data 와 그림의 TRANSACTION Packet 의 구조가 동일함에 주목하자.

```
struct binder_transaction_data {
    /* The first two are only used for bcTRANSACTION and brTRANSACTION,
     * identifying the target and contents of the transaction.
     */
    union {
        size_t  handle; /* target descriptor of command transaction */
        void    *ptr;   /* target descriptor of return transaction */
    } target;
    void        *cookie;    /* target object cookie */
    unsigned int    code;       /* transaction command */

    /* General information about the transaction. */
    unsigned int    flags;
    pid_t       sender_pid;
    uid_t       sender_euid;
    size_t      data_size;  /* number of bytes of data */
    size_t      offsets_size;   /* number of bytes of offsets */

    /* If this transaction is inline, the data immediately
     * follows here; otherwise, it ends with a pointer to
     * the data buffer.
     */
    union {
        struct {
            /* transaction data */
            const void  *buffer;
            /* offsets from buffer to flat_binder_object structs */
            const void  *offsets;
        } ptr;
        uint8_t buf[8];
    } data;
};
```



1.1.2.1. binder_transaction()

인자로 받은 binder_transaction_data 인 tr 로부터 클라이언트가 Parcel 에 집어넣은, 요청 사항과 데이터(DATA Packet)을 읽어들여 처리한다.

먼저, target handle 이 0 이니, if ~ else 에 의해 다음의 코드가 실행되어
target_node(binder_context_mgr_node) 를 얻어 오고,
 target_node(binder_context_mgr_node) 에서 target_proc을 얻어 오고,
 인자로 받았던 binder_thread 인 thread(filep 에 넣어 둔 binder_proc 으로부터 만들었다) 로부터 transaction_stack 을 얻어 오고,
 stack 내용을 모두 돌면서 target_proc 에 맞는 binder_thread* target_thread 를 얻어 온다 (stack->from).

```
    if (tr->target.hanlde) {
        ...
    } else {
        target_node = binder_context_mgr_node;
        if (target_node == NULL) {
            return_error = BR_DEAD_REPLY;
            goto err_no_context_mgr_node;
        }
    }
    e->to_node = target_node->debug_id;
    target_proc = target_node->proc;
    if (target_proc == NULL) {
        return_error = BR_DEAD_REPLY;
        goto err_dead_binder;
    }
    if (!(tr->flags & TF_ONE_WAY) && thread->transaction_stack) {
        struct binder_transaction *tmp;
        tmp = thread->transaction_stack;
        if (tmp->to_thread != thread) {
```

```
            binder_user_error("binder: %d:%d got new "
                "transaction with bad transaction stack"
                ", transaction %d has target %d:%d\n",
                proc->pid, thread->pid, tmp->debug_id,
                tmp->to_proc ? tmp->to_proc->pid : 0,
                tmp->to_thread ?
                tmp->to_thread->pid : 0);
            return_error = BR_FAILED_REPLY;
            goto err_bad_call_stack;
        }
        while (tmp) {
            if (tmp->from && tmp->from->proc == target_proc)
                target_thread = tmp->from;
            tmp = tmp->from_parent;
        }
    }
```

다음으로, TRANSACTION Packet 으로부터 DATA Packet 을 얻어오고,
DATA Packet 의 type(커맨드와 같다) 에 따라 switch ~ case 로 처리한다.
여기서, **BINDER_TYPE_BINDER**(addService() 시, transact 하는 DATA Packet에 type 을 BINDER_TYPE_BINDER 로 넣었다) 의 경우를 살펴 본다.

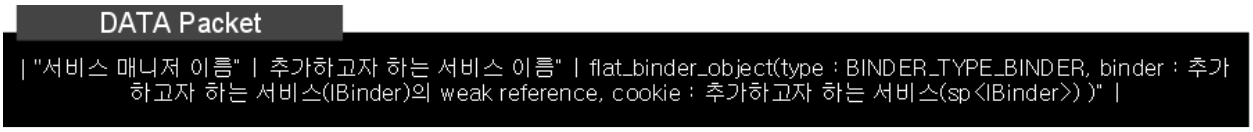
```
    fp = (struct flat_binder_object *)(t->buffer->data + *offp);
    switch (fp->type) {
    case BINDER_TYPE_BINDER:
    case BINDER_TYPE_WEAK_BINDER: {
        struct binder_ref *ref;
        struct binder_node *node = binder_get_node(proc, fp->binder);
        if (node == NULL) {
            node = binder_new_node(proc, fp->binder, fp->cookie);
            if (node == NULL) {
                return_error = BR_FAILED_REPLY;
                goto err_binder_new_node_failed;
            }
            node->min_priority = fp->flags & FLAT_BINDER_FLAG_PRIORITY_MASK;
            node->accept_fds = !!(fp->flags & FLAT_BINDER_FLAG_ACCEPTS_FDS);
        }
        if (fp->cookie != node->cookie) {
            binder_user_error("binder: %d:%d sending u%p "
                "node %d, cookie mismatch %p != %p\n",
                proc->pid, thread->pid,
                fp->binder, node->debug_id,
                fp->cookie, node->cookie);
            goto err_binder_get_ref_for_node_failed;
        }
        ref = binder_get_ref_for_node(target_proc, node);
        if (ref == NULL) {
            return_error = BR_FAILED_REPLY;
            goto err_binder_get_ref_for_node_failed;
        }
```

```
            if (fp->type == BINDER_TYPE_BINDER)
                fp->type = BINDER_TYPE_HANDLE;
            else
                fp->type = BINDER_TYPE_WEAK_HANDLE;
            fp->handle = ref->desc;
            binder_inc_ref(ref, fp->type == BINDER_TYPE_HANDLE, &thread->todo);
            if (binder_debug_mask & BINDER_DEBUG_TRANSACTION)
                printk(KERN_INFO "        node %d u%p -> ref %d desc %d\n",
                       node->debug_id, node->ptr, ref->debug_id, ref->desc);
        } break;
```

t 는 binder_transaction 구조체 포인터이다.
BINDER_TYPE_BINDER 타입인 경우, binder_node 를 새로 만드는데, 이 때 생성 함수에 클라이언트가 flatten 해 넣어둔 객체를 인자로 넘긴다.(cookie 에 IBinder 가 들어 있다)

DATA Packet 이 struct flat_binder_object 와 유사함을 주목하자.

```
struct flat_binder_object {
    /* 8 bytes for large_flat_header. */
    unsigned long       type;
    unsigned long       flags;

    /* 8 bytes of data. */
    union {
        void        *binder;    /* local object */
        signed long handle;     /* remote object */
    };

    /* extra data associated with local object */
    void            *cookie;
};
```



마지막으로, 다음의 코드가 실행된다.

```
t->work.type = BINDER_WORK_TRANSACTION;
list_add_tail(&t->work.entry, target_list);
tcomplete->type = BINDER_WORK_TRANSACTION_COMPLETE;
list_add_tail(&tcomplete->entry, &thread->todo);
if (target_wait)
    wake_up_interruptible(target_wait);
return;
```

t->work 는 struct binder_work 로, list_head 구조체 멤버 하나와 enum 으로 type 하나를 가지고 있다.
target_list*(**target_node->async_list**) 에 이번 트랜잭션의 엔트리(struct list_head)를 넣은 후, wake_up_interruptible 로 binder_thread_read 의 대기 중인 thread 를 실행 시킨다.

target_list = &target_thread->todo; target_wait = &target_thread->wait; 로 앞서 값이 들어가 있다.

이제, target thread 가 깨어나 일을 처리할 것이다.

1.2. binder_thread_read()

binder_ioctl() 에서, BINDER_WRITE_READ 명령에 대해, 인자로 받은 bind_read_write 구조체의 read_size 가 있는 경우에 대해, binder_thread_read() 호출을 통해 read 커맨드에 대한 동작을 해준다.

```
ret = binder_thread_read(proc, thread, (void __user *)bwr.read_buffer,
bwr.read_size, &bwr.read_consumed, filp->f_flags & O_NONBLOCK);
```

실행 직후, 에러 처리 후 인자로 받은 binder_proc 의 대기 중인 쓰레드에서 할 일이 생길 때까지 블락되어 있는다.
서비스 매니저는, 최초 실행 후 들어갔던 루프에서 read command 를 보냈으므로, 여기에 멈춰 있을 것이다.

```
    if (wait_for_proc_work) {
        if (!(thread->looper & (BINDER_LOOPER_STATE_REGISTERED |
                    BINDER_LOOPER_STATE_ENTERED))) {
&amp;nbsp;&amp;nbsp;          binder_user_error("binder: %d:%d ERROR: Thread waiting "
                "for process work before calling BC_REGISTER_"
                "LOOPER or BC_ENTER_LOOPER (state %x)\n",
                proc->pid, thread->pid, thread->looper);
            wait_event_interruptible(binder_user_error_wait, binder_stop_on_user_error <
2);
        }
```

```
        binder_set_nice(proc->default_priority);
        if (non_block) {
            if (!binder_has_proc_work(proc, thread))
                ret = -EAGAIN;
        } else
            ret = wait_event_interruptible_exclusive(proc->wait,
binder_has_proc_work(proc, thread));
    }
```

media server 가 이걸 깨웠다(binder 에 write 동작을 했다).
이어서, 루프를 돌며 읽어들인 데이터를 가지고 thread->todo 리스트, proc->todo 리스트에 대해 **binder_work 구조체를 얻어 온 후**,
해당 구조체의 type 값에 따라 switch ~ case 로 처리를 해준다.

아까 binder_transaction() 에서 target_list 에 **BINDER_WORK_TRANSACTION** 으로 work type 을 넣었다.

```
    while (1) {
        uint32_t cmd;
        struct binder_transaction_data tr;
        struct binder_work *w;
        struct binder_transaction *t = NULL;

        if (!list_empty(&thread->todo))
            w = list_first_entry(&thread->todo, struct binder_work, entry);
        else if (!list_empty(&proc->todo) && wait_for_proc_work)
            w = list_first_entry(&proc->todo, struct binder_work, entry);
        else {
            if (ptr - buffer == 4 && !(thread->looper &
BINDER_LOOPER_STATE_NEED_RETURN)) /* no data added */
                goto retry;
            break;
        }

        if (end - ptr < sizeof(tr) + 4)
            break;

        switch (w->type) {
        case BINDER_WORK_TRANSACTION: {
            t = container_of(w, struct binder_transaction, work);
        } break;
        case BINDER_WORK_TRANSACTION_COMPLETE: {
            cmd = BR_TRANSACTION_COMPLETE;
            if (put_user(cmd, (uint32_t __user *)ptr))
                return -EFAULT;
            ptr += sizeof(uint32_t);

            binder_stat_br(proc, thread, cmd);
            if (binder_debug_mask & BINDER_DEBUG_TRANSACTION_COMPLETE)
                printk(KERN_INFO "binder: %d:%d BR_TRANSACTION_COMPLETE\n",
                       proc->pid, thread->pid);
```

```
            list_del(&w->entry);
            kfree(w);
            binder_stats.obj_deleted[BINDER_STAT_TRANSACTION_COMPLETE]++;
        } break;
        case BINDER_WORK_NODE: {
```

binder_transaction 구조체를 가져오고 swtich 를 빠져나온다.
binder_transaction 구조체는 다음과 같다.

```
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;
};
```

type 에 따른 switch ~ case 이후(여전히 루프의 안이다),
클라이언트의 write buffer 내용을, 서비스 매니저의 read buffer 에 써 넣는다. 아까 꺼낸 binder_transaction 구조체 t 를 사용함에 유의하자.
디바이스 드라이버는 자신의 프로세스를 갖지 않는다. 클라이언트 프로세스에서 직접 커널 영역으로 들어가 디바이스 드라이버의 함수를 사용한다.
여기서 **유저 영역과 커널 영역 사이의 메모리를 오가기 위해 copy_to_user(), copy_from_user() 를 사용**한다.

```
        tr.data_size = t->buffer->data_size;
        tr.offsets_size = t->buffer->offsets_size;
        tr.data.ptr.buffer = (void *)t->buffer->data + proc->user_buffer_offset;
        tr.data.ptr.offsets = tr.data.ptr.buffer + ALIGN(t->buffer->data_size, sizeof(void *));

        if (put_user(cmd, (uint32_t __user *)ptr))
            return -EFAULT;
        ptr += sizeof(uint32_t);
        if (copy_to_user(ptr, &tr, sizeof(tr)))
            return -EFAULT;
        ptr += sizeof(tr);
```

TR 행위를 해준 후, 아무 신호도 주지 않는다는 데 주목하자. 서버가 알아서 결과를 찾아가야 한다.

2. 결론

이상으로, client 부터 server 까지의 데이터 플로우를 알아 보았다.

Service Manager Handle Add Service

다시 프로세스 레벨로 돌아온다.

0. return to service_manager

이제, 서비스 매니저는 binder_loop() 내에서의 루프에서 AudioFlinger 의 addService() 에 의해 바인더 디바이스 드라이버로부터 데이터를 읽어들였다.
이제 ioctl() 이 리턴했으므로, 블록이 풀리고, 다음의 binder_parse() 를 호출하게 된다.

```
void binder_loop(struct binder_state *bs, binder_handler func)
{
   int res;
   struct binder_write_read bwr;
   unsigned readbuf[32];

   bwr.write_size = 0;
   bwr.write_consumed = 0;
   bwr.write_buffer = 0;

   readbuf[0] = BC_ENTER_LOOPER;
   binder_write(bs, readbuf, sizeof(unsigned));

   for (;;) {
+--  4 줄: bwr.read_size = sizeof(readbuf);-----------------------------------------------------
-------------------------------------------------------
      res = ioctl(bs->fd, BINDER_WRITE_READ, &bwr);
+--  6 줄: if (res < 0) {-------------------------------------------------------------------------
-----------------------------------------------------
```

```
      res = binder_parse(bs, 0, readbuf, bwr.read_consumed, func);
+---  8 줄: if (res == 0) {----------------------------------------------------------------------
-------------------------------------------------
   }
}
```

1. binder_parse()

읽어들인 커맨드(readbuf 로 받았고, binder_parse() 에서는 ptr 라는 param 으로 받는다.)에 따라 switch ~ case 로 처리한다.
여기선 BR_TRANSACTION 으로 들어왔을 것이다.

```
      case BR_TRANSACTION: {
         struct binder_txn *txn = (void *) ptr;
         if ((end - ptr) * sizeof(uint32_t) < sizeof(struct binder_txn)) {
            LOGE("parse: txn too small!\n");
            return -1;
         }
         binder_dump_txn(txn);
         if (func) {
            unsigned rdata[256/4];
            struct binder_io msg;
            struct binder_io reply;
            int res;

            bio_init(&reply, rdata, sizeof(rdata), 4);
            bio_init_from_txn(&msg, txn);
            res = func(bs, txn, &msg, &reply);
            binder_send_reply(bs, &reply, txn->data, res);
         }
         ptr += sizeof(*txn) / sizeof(uint32_t);
         break;
      }
```

결국, binder_handler 를 호출한다. 여기서 struct binder_txn 은, 바인더 드라이버의 binder_transaction_data 구조체와 동일하다.

```
struct binder_txn
{
   void *target;
   void *cookie;
   uint32_t code;
   uint32_t flags;

   uint32_t sender_pid;
```

```
    uint32_t sender_euid;

    uint32_t data_size;
    uint32_t offs_size;
    void *data;
    void *offs;
};
```

2. svcmgr_handler()

func 는 service_manager.c 의 main 함수에서 svcmgr_handler() 의 함수 포인터를 넣어 주었다. 이제, service_manager.c 의 svcmgr_handler() 함수가 실행된다.

여기선 binder_txn 구조체의 code 값에 대해 switch ~ case 로 처리를 한다. 이 경우, AudioFlinger 의 addService() 에서 보냈던 SVC_MGR_ADD_SERVICE 일 것이다.

```
int svcmgr_handler(struct binder_state *bs,
                   struct binder_txn *txn,
                   struct binder_io *msg,
                   struct binder_io *reply)
{
    struct svcinfo *si;
    uint16_t *s;
    unsigned len;
    void *ptr;

//    LOGI("target=%p code=%d pid=%d uid=%d\n",
//         txn->target, txn->code, txn->sender_pid, txn->sender_euid);

    if (txn->target != svcmgr_handle)
        return -1;

    s = bio_get_string16(msg, &len);

    if ((len != (sizeof(svcmgr_id) / 2)) ||
        memcmp(svcmgr_id, s, sizeof(svcmgr_id))) {
        fprintf(stderr,"invalid id %s\n", str8(s));
        return -1;
    }

    switch(txn->code) {
    case SVC_MGR_GET_SERVICE:
+--  8 줄: case SVC_MGR_CHECK_SERVICE:-----------------------------------------------------
-------------------------------------------------------------
    case SVC_MGR_ADD_SERVICE:
        s = bio_get_string16(msg, &len);
        ptr = bio_get_ref(msg);
        if (do_add_service(bs, s, len, ptr, txn->sender_euid)
```

```
            return -1;
        break;

+-- 12 줄: case SVC_MGR_LIST_SERVICES: {-----------------------------------------------
-------------------------------------------------------------
    default:
        LOGE("unknown code %d\n", txn->code);
        return -1;
    }

    bio_put_uint32(reply, 0);
    return 0;
}
```

SVC_MGR_ADD_SERVICE transaction code 에 대해, do_add_service() 함수를 호출해 실제 서비스 등록을 한다.
do_add_service() 를 보기 전에, binder_io 로부터 해당 binder 의 레퍼런스를 얻어 오는 bio_get_ref() 는 flatten_binder() 의 반대 동작을 한다.

3. do_add_service()

```
int do_add_service(struct binder_state *bs,
                   uint16_t *s, unsigned len,
                   void *ptr, unsigned uid)
{
    struct svcinfo *si;
//    LOGI("add_service('%s',%p) uid=%d\n", str8(s), ptr, uid);

    if (!ptr || (len == 0) || (len > 127))
        return -1;

    if (!svc_can_register(uid, s)) {
+--  3 줄: LOGE("add_service('%s',%p) uid=%d - PERMISSION DENIED\n",----------------
-------------------------------------------------------------
    }

    si = find_svc(s, len);
    if (si) {
        if (si->ptr) {
+--  3 줄: LOGE("add_service('%s',%p) uid=%d - ALREADY REGISTERED\n",--------------
-------------------------------------------------------------
        }
        si->ptr = ptr;
    } else {
        si = malloc(sizeof(*si) + (len + 1) * sizeof(uint16_t));
        if (!si) {
            LOGE("add_service('%s',%p) uid=%d - OUT OF MEMORY\n",
                 str8(s), ptr, uid);
            return -1;
```

```
        }
        si->ptr = ptr;
        si->len = len;
        memcpy(si->name, s, (len + 1) * sizeof(uint16_t));
        si->name[len] = '\0';
        si->death.func = svcinfo_death;
        si->death.ptr = si;
        si->next = svclist;
        svclist = si;
    }

    binder_acquire(bs, ptr);
    binder_link_to_death(bs, ptr, &si->death);
    return 0;
}
```

service info 를 만들어 해당 정보를 모두 집어 넣고, struct svcinfo * 인 svclist 의 리스트에 집어넣는다.

마지막으로 binder_acquire 에서 BC_ACQUIRE 로 커맨드를 넣고, binder_write() 한다.
binder_write() 에서는 binder_write_read 구조체를 만들고, ioctl 한다.
이제, 바인더 드라이버의 binder_ioctl() 에서 write 명령에 대한 부분에서 받아 처리 할 것이다.

4. 결론

이상으로 서비스 매니저가 시작되고, 사용자의 addService 요청에 대해 바인더 드라이버를 통해 해당 이름과 서비스의 정보를 등록하는 과정을 분석했다.

Get IAudioFlinger

서비스 매니저가 활성화 되었고, AudioFlinger 가 서비스 매니저에 등록되었다.
이제, 클라이언트에서는 서비스매니저로부터 해당 서비스의 인터페이스를 얻어오면 된다.
서비스를 얻어오기 위해선 IServiceManager::getSerive() 메소드를 사용한다.

0. entry point : frameworks/base/libs/binder/ IServiceManager.cpp, BpServiceManager::getService()

```
virtual sp<IBinder> getService(const String16& name) const
{
    unsigned n;
    for (n = 0; n < 5; n++){
        sp<IBinder> svc = checkService(name);
        if (svc != NULL) return svc;
        LOGI("Waiting for sevice %s...\n", String8(name).string());
        sleep(1);
    }
    return NULL;
}
```

이름을 가지고 checkService()를 통해 서비스 인터페이스를 얻어오고, 얻지 못할 경우 1초 쉬어 가면서 다섯번까지 시도한다.

1. BpServiceManager::checkService()

```
virtual sp<IBinder> checkService( const String16& name) const
{
    Parcel data, reply;
    data.writeInterfaceToken(IServiceManager::getInterfaceDescriptor());
    data.writeString16(name);
    remote()->transact(CHECK_SERVICE_TRANSACTION, data, &reply);
    return reply.readStrongBinder();
}
```

transact() 는 앞서 살펴 봤듯이, BpBinder::transact() 를 호출,
IPCThreadState::transact() 를 호출하여 Parcel 에 데이터를 넣고,
waitForResponse() 에서 talkWithDriver() 를 호출, ioctl 한다.

2. svcmgr_handler()

transact() 한 내용은 바인더 드라이버를 거쳐, 서비스 매니저의 svcmgr_handler 로 돌아올 것이다.
여기서 switch ~ case 를 돌게 되는데,
앞서 checkService() 에선 CHECK_SERVICE_TRANSACTION 을 넣었다. 이 값은 2다.
service_manager.c 에서는 SVC_MGR_CHECK_SERVICE 가 2로, 같은 값을 갖는다.

```
    switch(txn->code) {
    case SVC_MGR_GET_SERVICE:
    case SVC_MGR_CHECK_SERVICE:
        s = bio_get_string16(msg, &len);
        ptr = do_find_service(bs, s, len);
```

```
        if (!ptr)
            break;
        bio_put_ref(reply, ptr);
        return 0;
```

do_find_service() 는 find_svc()를 호출하고, 여기서 아까 addService() 에서 받은 서비스를 넣어 뒀던 리스트를 루프로 돌며 이름에 맞는 svcinfo 구조체를 찾아 리턴한다.

```
struct svcinfo *find_svc(uint16_t *s16, unsigned len)
{
    struct svcinfo *si;

    for (si = svclist; si; si = si->next) {
        if ((len == si->len) &&
            !memcmp(s16, si->name, len * sizeof(uint16_t))) {
            return si;
        }
    }
    return 0;
}
```

3. 결론 :

이제 슬슬 바인더의 일반적인 동작이 드러난다.

IAudioFlinger::setMode()

마지막으로, setMode() 와 같은, 일반적 RPC 의 경로를 분석한다.

0. RPC

IAudioFlinger::setMode() 는, 결국 BpAudioFlinger::setMode() 를 호출할 것이다.

```
    virtual status_t setMode(int mode)
    {
```

```
        Parcel data, reply;
        data.writeInterfaceToken(IAudioFlinger::getInterfaceDescriptor());
        data.writeInt32(mode);
        remote()->transact(SET_MODE, data, &reply);
        return reply.readInt32();
    }
```

getService() 와 별 다를 바가 없다. 그냥 Parcel 에 넣고 binder driver 에서 트랜잭션 데이터를 받아 switch ~ case 로 사용할 상수인 SET_MODE 를 넣고서는 걍 transact() 한다.

1. IPCThreadState::joinThreadPool

바인더 드라이버에선 미디어서버 프로세스의 리드 쓰레드를 깨울 것이다.
미디어 서버는 서비스 등록 된 후, executeCommand() 에서, 루프를 돌면서 talkWithDriver() 에서 에서 멈춰 있었다.

```
void IPCThreadState::joinThreadPool(bool isMain)
{
+-- 30 줄: LOG_THREADPOOL("**** THREAD %p (PID %d) IS JOINING THE THREAD
POOL\n", (void*)pthread_self(), getpid());-----------------------------------
        result = talkWithDriver();
        if (result >= NO_ERROR) {
            size_t IN = mIn.dataAvail();
            if (IN < sizeof(int32_t)) continue;
            cmd = mIn.readInt32();
            IF_LOG_COMMANDS() {
                alog << "Processing top-level Command: "
                    << getReturnString(cmd) << endl;
            }


            result = executeCommand(cmd);
        }

        // After executing the command, ensure that the thread is returned to the
        // default cgroup and priority before rejoining the pool.  This is a failsafe
        // in case the command implementation failed to properly restore the thread's
        // scheduling parameters upon completion.
        int my_id;
#ifdef HAVE_GETTID
        my_id = gettid();
#else
        my_id = getpid();
#endif
        if (!set_sched_policy(my_id, SP_FOREGROUND)) {
            // success; reset the priority as well
```

```
            setpriority(PRIO_PROCESS, my_id, ANDROID_PRIORITY_NORMAL);
        }

        // Let this thread exit the thread pool if it is no longer
        // needed and it is not the main process thread.
        if(result == TIMED_OUT && !isMain) {
            break;
        }
    } while (result != -ECONNREFUSED && result != -EBADF);

    LOG_THREADPOOL("**** THREAD %p (PID %d) IS LEAVING THE THREAD POOL
err=%p\n",
        (void*)pthread_self(), getpid(), (void*)result);

    mOut.writeInt32(BC_EXIT_LOOPER);
    talkWithDriver(false);
}
```

talkWithDriver() 가 리턴하여 깨어난 AudioFlinger 의 reader 쓰레드는 executeCommand() 로 해당 커맨드를 동작한다.

1.1. executeCommnad()

cmd 만을 받아, switch ~ case 로 해결한다.
바인더 드라이버로부터 깨어난 직후이니, 여기서 커맨드는 BR_TRANSACTION 일 것이다.

```
    case BR_TRANSACTION:
        {
            binder_transaction_data tr;
            result = mIn.read(&tr, sizeof(tr));
+--  2 줄: LOG_ASSERT(result == NO_ERROR,----------------------------------------------
------------------------------------------------------------
            if (result != NO_ERROR) break;

            Parcel buffer;
            buffer.ipcSetDataReference(
                reinterpret_cast<const uint8_t*>(tr.data.ptr.buffer),
                tr.data_size,
                reinterpret_cast<const size_t*>(tr.data.ptr.offsets),
                tr.offsets_size/sizeof(size_t), freeBuffer, this);

            const pid_t origPid = mCallingPid;
            const uid_t origUid = mCallingUid;

            mCallingPid = tr.sender_pid;
            mCallingUid = tr.sender_euid;

            //LOGI(">>>> TRANSACT from pid %d uid %d\n", mCallingPid, mCallingUid);
```

```
        Parcel reply;
        IF_LOG_TRANSACTIONS() {
            TextOutput::Bundle _b(alog);
+--  8 줄: alog << "BR_TRANSACTION thr " << (void*)pthread_self()----------------------
------------------------------------------------------------
        }
        if (tr.target.ptr) {
            sp<BBinder> b((BBinder*)tr.cookie);
            const status_t error = b->transact(tr.code, buffer, &reply, 0);
            if (error < NO_ERROR) reply.setError(error);

        } else {
```

tr.cookie 를 BBinder* 로 변환시켰다. tr.cookie 는 addService 시에 바인더 커널에 넣었던 값이다.
이제, tr.cookie 는 BBinder* 이고, 이것은 곧 우리의 AudioFlinger 의 interface 이다.
BBinder::transact() 에서는 onTransact()를 호출한다.
이제, b->transact()는 BnAudioFlinger 의 onTransact() 가상함수를 호출한 것과 같다.

1.1.1. BnAudioFlinger::onTransact()

여기선 코드에 대해 switch ~ case 로 동작한다.

```
    case SET_MODE: {
        CHECK_INTERFACE(IAudioFlinger, data, reply);
        int mode = data.readInt32();
        reply->writeInt32( setMode(mode) );
        return NO_ERROR;
    } break;
```

클라이언트가 요구한 mode 값(아규먼트)을 읽어들여 setMode() 해준다.
이는 AudioFlinger::setMode() 로 호출되어, AudioFlinger 에서 적당히 처리를 해 줄 것이다.

1.1.2. sendReply()

다시 IPCThreadState::executeCommand() 로 돌아온다. BR_TRANSACTION 케이스에선, reply Parcel 을 작성 한 후, 마지막으로 sendReply(reply) 를 호출한다.

```
status_t IPCThreadState::sendReply(const Parcel& reply, uint32_t flags)
{
    status_t err;
    status_t statusBuffer;
    err = writeTransactionData(BC_REPLY, flags, -1, 0, reply, &statusBuffer);
    if (err < NO_ERROR) return err;

    return waitForResponse(NULL, NULL);
}
```

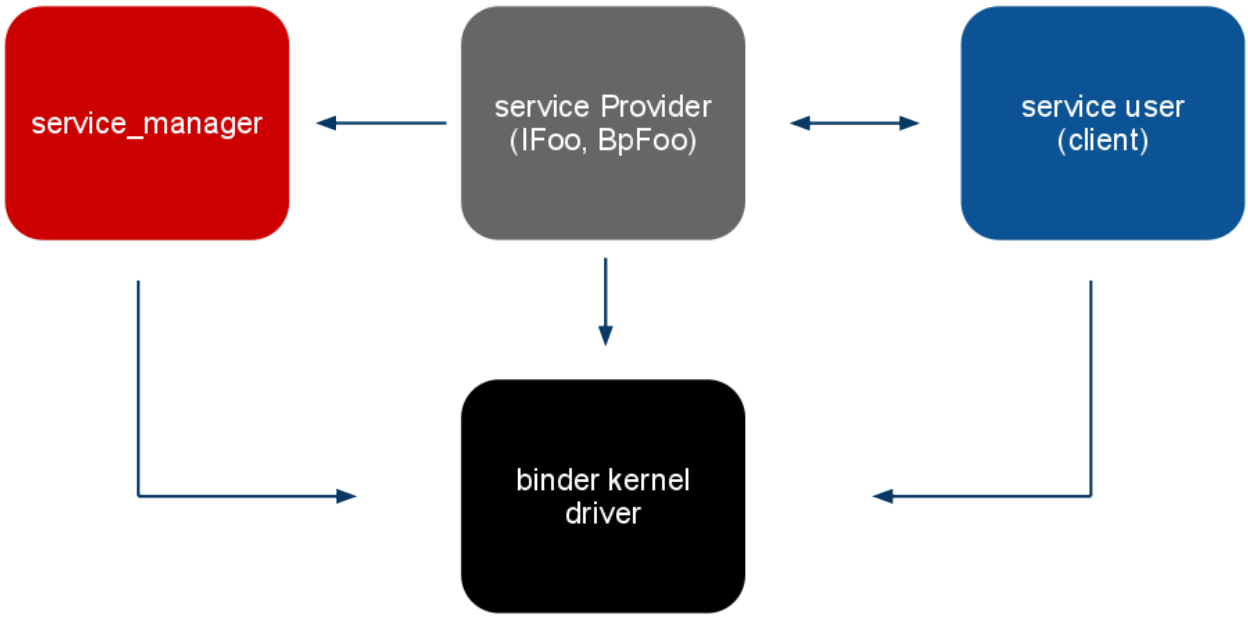
이제, repy 데이터는 바인더 드라이버에게 날아가고,
바인더 드라이버는 최초 IAudioFlinger::getService()::setMode() 를 호출했던 클라이언트 프로세스의 read 쓰레드를 깨울 것이다.

2. 결론 :

클라이언트의 요청 > IFoo 의 메소드를 호출하고, 이는 BpFoo 의 메소드로 이동해, transact를 한다.
바인더 드라이버는 BnFoo 를 깨우고, BnFoo 는 결국 Foo 이므로, Foo 의 동작을 하고, 마지막으로 reply 를 해준다.

결국 요점이자 중요한 점은 Binder 드라이버는 해당 요청에 대한 리더 쓰레드를 정확하게 깨워 준다는 거다.
즉, 클라이언트는 클라이언트 쓰레드에서 IFoo 를 얻어와 메소드를 호출하고, 이는 바인더 드라이버를 거쳐 Foo 의 쓰레드에서 해당 동작을 하고, 그 결과를 바인더 드라이버를 통해 클라이언트 쓰레드로 넘겨준다.

요약 :



결국, binder 드라이버 에서 주요 컴포넌트는 위의 네개 블록으로 나눌 수 있다.

- Binder kernel driver
  - service provider 와 service user 사이에 데이터를 전송한다. 데이터를 요구하는 쓰레드는 sleep 하게 되고, 데이터를 받을 때 binder 드라이버에 의해 깨어나게 된다.
- service provider
  - 서비스 인터페이스를 제공한다. 바인더 드라이버로부터 받은 RPC 호출 데이터를 파싱한다.
- service_manager
  - special service provider. service provider 의 service provider 라 할 수 있다.
- service user
  - service provider 에게 remote call 을 요청한다.

우리가 그렸던 시나리오, IAudioFlinger::setMode 의 전체 과정을 간략하게 정리 해 본다.

1. booting 직후, 서비스 매니저의 등록과 AudioFlinger 의 서비스 등록

1. booting 직후, service manager 가 node 0로 바인더 드라이버에 자신을 등록한다.
2. 미디어 서비스 프로세스가 실행되고, defaultServiceManager()에 노드 0를 넣어 IServiceManager proxy 객체를 얻어 온다.
3. 미디어 서버 프로세스가 **IServiceManager::addService()** 를 호출, 바인더 드라이버에게 데이터를 보낸다.
4. 바인더 드라이버는 노드 0을 가지고 바인드 되고 싶어 하는 데이터를 받게 되므로, **IAudioFlinger 서비스를 위해 특정 노드를 생성**하고 서비스 매니저로 데이터를 보낸다.

5. 서비스 매니저는 바인더 드라이버로부터 데이터를 받고, svcmgr_handler() 에서 이를 처리해 **IServiceManager::addService RPC 콜의 실제 동작**을 한다.

2. 클라이언트 프로세스의 IServiceManager::getService()

1. 클라이언트 프로세스는 노드 0으로 IServiceManager proxy 객체를 얻어온다.
2. 클라이언트 프로세스는 IAudioFlinger 서비스를 얻어 오기 위해 **IServiceManager::getService()** 를 호출한다. 이 요청은 노드 0을 가지고 바인더 드라이버에게 전송된다.
3. 바인더 드라이버는 노드 번호를 보고 서비스 매니저에게 데이터를 보내 깨운다.
4. 서비스 매니저는 바인더 드라이버로부터 데이터를 읽고, getService() 동작을 하여 클라이언트가 요구한 IAudioFlinger 서비스의 노드 번호를 바인더 드라이버에게 reply 한다.
5. 바인더 드라이버는 reply 받은 데이터를 클라이언트 프로세스에게 보낸다.

3. 클라이언트 프로세스의 IAudioFlinger::setMode()

1. 클라이언트 프로세스는 얻어온 IAudioFlinger 를 가지고 setMode() 를 호출한다. 이제 요청은 IAudioFlinger 서비스의 노드 번호를 가지고 바인더 드라이버에게 날아간다.
2. 바인더 드라이버는 들어온 데이터의 노드 번호를 보고, 미디어 서버 프로세스로 데이터를 넘긴다.
3. 미디어 서버는 데이터를 받고 setMode 에 대한 처리를 해주고, 바인더 드라이버에게 결과를 reply 한다.
4. 바인더 드라이버는 결과를 클라이언트 프로세스에게 reply 한다.
5. 클라이언트 프로세스는 마침내 결과를 받는다.