

번역자 : 전호철 (<http://www.hybridego.net/>)

안드로이드 IPC 시스템이 어떻게 작동하는지 보기 위해 IAudioFlinger::serMode API 를 사용한다.
AudioFlinger는 media_server 프로그램의 서비스다.

Service Manager Run

sevice_manager는 다른 프로세스에게 sevice manager 서비스를 제공한다. 이것은 반드시 다른 어떤 서비스의 실행 보다 먼저 실행되어야만 한다.

```
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;
}
```

우선 /dev/binder 드라이버를 오픈하고 BINDER_SET_CONTEXT_MGR ioctl 을 호출해서 바인더 커널 드라이버가 manager처럼 작동하게 한다. 그런 다음 루프로 들어가서 다른 프로세스로부터 어떤 데이터가 오기를 기다리게 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 (;;) {
    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;
    }
}
}

```

BINDER_SERVICE_MANAGER 를 주의깊게 보라.

```

/* the one magic object */
#define BINDER_SERVICE_MANAGER ((void*) 0)

```

BINDER_SERVICE_MANAGER 는 service_manager를 위해 등록된 핸들이다. 다른 프로세스들은 반드시 이 핸들을 사용해서 service_manager와 통신해야 한다.

Get IServiceManager

IServiceManager 인스턴스를 얻기 위한 단 하나의 방법은 IServiceManager.cpp에 있는 defaultServiceManager를 호출하는 것이다.

```

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

    {
        AutoMutex _(gDefaultServiceManagerLock);

```

```

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

    return gDefaultServiceManager;
}

```

gDefaultServiceManager 는 libutil에 정의되어 있다. 그래서 어떤 프로그램이나 라이브러리가 libutil을 포함하고 있으면 이 심볼을 갖고 있을 것이다. 이것은 한 프로세스에 단 하나가 있다. 처음 gDefaultServiceManager는 NULL 값을 갖고 있다. 그래서 이것은 처음에 ProcessState::self()를 통해 ProcessState 인스턴스를 얻는다. 한 프로세스는 단 하나의 ProcessState 인스턴스를 갖는다. ProcessState 는 IPCThreadState를 사용하기위해 /dev/binder 드라이버를 오픈할것이다.

```

ProcessState::ProcessState()
    : mDriverFD(open_driver())

```

이제 우리는 ProcessState의 인스턴스 하나를 얻었다.

이제 getContextObject를 보자

```

sp<IBinder> ProcessState::getContextObject(const sp<IBinder>& caller)
{
    if (supportsProcesses()) {
        return getStrongProxyForHandle(0);
    } else {
        return getContextObject(String16("default"), caller);
    }
}

```

보드는 binder driver를 제공한다. 그래서 우리는 getStrongProxyForHandle 에 도착할것이다. (Handle 0는 service manager를 위해 예약되었다. 다음에 설명될 것이다.)

```

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

    AutoMutex _l(mLock);

    return result;
}

```

처음에 b 는 NULL로 한다. BpBinder는 remote binder object를 위한 base proxy class이다.

```

BpBinder::BpBinder(int32_t handle)
    : mHandle(handle)
    , mAlive(1)
    , mObitsSent(0)
    , mObituaries(NULL)
{
    LOGV("Creating BpBinder %p handle %d\n", this, mHandle);

    extendObjectLifetime(OBJECT_LIFETIME_WEAK);
    IPCThreadState::self()->incWeakHandle(handle);
}

```

IPCThreadState::incWeakHandle 은 output buffer에 있는 BC_INCREFS 명령을 추가한다.

```

void IPCThreadState::incWeakHandle(int32_t handle)
{
    LOG_REMOTE_REFS("IPCThreadState::incWeakHandle(%d)\n", handle);
    mOut.writeInt32(BC_INCREFS);
    mOut.writeInt32(handle);
}

```

이제 getContextObject 는 BpBinder 인스턴스를 리턴한다. interface_cast는 IInterface.h에 정의되어 있다.

```

inline sp<IServiceManager> interface_cast(const sp<IBinder>& obj)
{
    return IServiceManager::asInterface(obj);
}

```

IServiceManager의 정의 부분이다.

```

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,
};
};

```

DEVLARE_META_INTERFACE 매크로는 IInterface.h에 다음처럼 정의되어 있다. 그리고 이것은 다음 코드로 확장될 것이다.

```

    static const String16 descriptor;
    static sp<IServiceManager> asInterface(const sp<IBinder>& obj);
    virtual String16 getInterfaceDescriptor() const;

```

보는 것 처럼 DECLARE_META_INTERFACE 매크로는 두개의 함수를 선언하는데 IServiceManager.cpp에서 IMPLEMENT_META_INTERFACE 매크로에 의해 실행된다.

```
IMPLEMENT_META_INTERFACE(ServiceManager, "android.os.IServiceManager");
```

코드는 다음처럼 확장된다.

```

    const String16 IServiceManager::descriptor(NAME);
    String16 IServiceManager::getInterfaceDescriptor() const {
        return IServiceManager::descriptor;
    }
    sp<IServiceManager> IServiceManager::asInterface(const sp<IBinder>& obj)
    {
        sp<IServiceManager> intr;

```

```

    if (obj != NULL) {
        intr = static_cast<IServiceManager*>(
            obj->queryLocalInterface(
                IServiceManager::descriptor).get());
        if (intr == NULL) {
            intr = new BpServiceManager(obj);
        }
    }
    return intr;
}

```

그래서 IServiceManager::asInterface 는 BpServiceManager 인스턴스를 리턴한다. BpServiceManager 원격 BnServiceManager를 위한 proxy처럼 작동한다. IServiceManager의 연산은 실제로 BpServiceManager의 상응하는 가상 함수를 호출한다.

Summary:

이 섹션은 remote 오브젝트를 위한 proxy 오브젝트를 어떻게 얻을수 있는지 알려준다.

여러분의 서비스-IFunnyTest를 수행하려면 다음을 그대로 따라하면 된다.

- DECLARE_META_INTERFACE(FunnyTest) 매크로를 여러분의 header 파일에 넣는다.
- IMPLEMENT_META_INTERFACE(Funnytest, "your unuque name") 매크로를 여러분의 interface source file 에 넣는다.
- 여러분의 BpFunnyTest 클래스를 실행합니다.

Generate AudioFlinger Service

Media_server 프로그램은 AudioFlinger 서비스를 시작할 것입니다.

```
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();
    ProcessState::self()->startThreadPool();
    IPCThreadState::self()->joinThreadPool();
}

```

AudioFlinger 는 IServiceManager::addService를 호출하는 RPC Call 이다.

```

void AudioFlinger::instantiate() {
    defaultServiceManager()->addService(

```

```

        String16("media.audio_flinger"), new AudioFlinger());
    }

```

AudioFlinger 는 BnAudioFlinger로부터 상속받는다. 이것은 BnInterface 클래스의 템플릿이다.

```

class BnAudioFlinger : public BnInterface<IAudioFlinger>
{
public:
    virtual status_t onTransact( uint32_t code,
                                const Parcel& data,
                                Parcel* reply,
                                uint32_t flags = 0);
};

```

BnInterface 는 BBinder로부터 상속받는다.

```

template<typename INTERFACE>
class BnInterface : public INTERFACE, public BBinder
{
public:
    virtual sp<Interface> queryLocalInterface(const String16& _descriptor);
    virtual String16 getInterfaceDescriptor() const;
protected:
    virtual IBinder* onAsBinder();
};
template<typename INTERFACE>
IBinder* BnInterface<INTERFACE>::onAsBinder()
{
    return this;
}

```

BnInterface의 실행에 따르면 파라미터가 IServiceManager::addService 를 통과하는 것을 알 수 있는데 이것은 새로운 AudioFlinger 인스턴스의 주소이다. BBinder는 IBinder로 부터 파생되었고 이 transact 함수는 onTrasact 가상함수를 호출한다.

```

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;
}

```

가장 중요한 가상함수는 onTransact이다. BnAudioFlinger 가 가상함수를 실행했다. 시나리오 대로라면 우리는 단지 SET_MODE 분기에 대한 포커스만 필요할 것이다.

```

status_t BnAudioFlinger::onTransact(
    uint32_t code, const Parcel& data, Parcel* reply, uint32_t flags)
{
    switch(code) {
    case SET_MODE: {
        CHECK_INTERFACE(IAudioFlinger, data, reply);
        int mode = data.readInt32();
        reply->writeInt32( setMode(mode) );
        return NO_ERROR;
    } break;
    }
}

```

media_server 는 IPThreadState::joinThreadPool를 통해 루프로 들어갈 것이다. service_manager 는 talkWithDriver 에서 다른 프로세스로부터 데이터를 기다릴 것이다.

```

void IPThreadState::joinThreadPool(bool isMain)
{
    mOut.writeInt32(isMain ? BC_ENTER_LOOPER : BC_REGISTER_LOOPER);

    status_t result;
    do {
        int32_t cmd;
        result = talkWithDriver();
        if (result >= NO_ERROR) {
            size_t IN = mIn.dataAvail();
            if (IN < sizeof(int32_t)) continue;
            cmd = mIn.readInt32();
            result = executeCommand(cmd);
        }
    }
    // 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);

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

```

여러분의 IFunnyTest를 실행하려면 다음과 같이 하면 된다.

- 여러분의 BnFunnyTest 클래스를 실행하라
- 프로세스에서 여러분의 서비스가 실행중일 때 IPCThreadState::joinThreadPool 을 호출하여 binder 를 위한 루프를 시작한다.

RPC Call IServiceManager::addService

우리가 IServiceManager::addService 호출했다. 이것은 BpServiceManager::addService를 호출했다는 의미이다.

```

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;
}

```

Parcel은 간단하다. 우리는 이것을 연속적인 버퍼라고 생각할 수 있다. 서비스 파라미터가 BBinder 오브젝트(Bn 으로부터 파생된 AudioFlinger)를 가리키고 있다는 것을 주목해라.

```

status_t Parcel::writeStrongBinder(const sp<IBinder>& val)
{
    return flatten_binder(ProcessState::self(), val, this);
}

```

flatten_binder는 하나의 Binder 명령을 생성한다. BBinder가 지역 binder 객체이기 때문에, 코드는 빨간색으로 마크된 라인으로 갈라진다.

```

status_t flatten_binder(const sp<ProcessState>& proc,
    const sp<IBinder>& binder, Parcel* out)
{
    flat_binder_object obj;

```

```

obj.flags = 0x7f | FLAT_BINDER_FLAG_ACCEPTS_FDS;
if (binder != NULL) {
    IBinder *local = binder->localBinder();
    if (!local) {
        BpBinder *proxy = binder->remoteBinder();
        if (proxy == NULL) {
            LOGE("null proxy");
        }
        const int32_t handle = proxy ? proxy->handle() : 0;
        obj.type = BINDER_TYPE_HANDLE;
        obj.handle = handle;
        obj.cookie = NULL;
    } else {
        obj.type = BINDER_TYPE_BINDER;
        obj.binder = local->getWeakRefs();
        obj.cookie = local;
    }
}
return finish_flatten_binder(binder, obj, out);
}

```

빨간 부분을 주목하라. 지역 주소가 packet으로 들어간다.(이것은 다음에 쓰일것이다.) addService RPC 호출을 위한 패킷을 만든다음 BpServiceManager::addService는 BpBinder의 transact를 호출할것이다.

```

status_t BpBinder::transact(
    uint32_t code, const Parcel& data, Parcel* reply, uint32_t flags)
{
    // Once a binder has died, it will never come back to life.
    if (mAlive) {
        status_t status = IPCThreadState::self()->transact(
            mHandle, code, data, reply, flags);
        if (status == DEAD_OBJECT) mAlive = 0;
        return status;
    }
    return DEAD_OBJECT;
}

```

BpBinder는 IPCThreadState::transact를 호출하여 mHandle에 해당하는 바인더 오브젝트로 트랜잭션을 시작한다. 시나리오에서 mHandle 은 0 이다.

```

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 (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);
    }

    if (err != NO_ERROR) {
        if (reply) reply->setError(err);
        return (mLastError = err);
    }

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

```

IPCThreadState::transact 는 첫번째로 writeTransactionData를 호출하여 바인더커널드라이버를 위한 트랜잭션 structure를 조립한다.

다음 라인을 주목해라. 이 부분은 바인더커널드라이버가 트랜잭션 타겟을 확인하기 위해 매우 중요하다.

```

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;
}

```

그러면 `waitForResponse`는 `talkWithDriver`를 호출하여 `BINDER_WRITE_READ` ioctl을 가져온다.

```

#ifdef HAVE_ANDROID_OS
    if (ioctl(mProcess->mDriverFD, BINDER_WRITE_READ, &bwr) >= 0)
        err = NO_ERROR;
    else
        err = -errno;
#else

```

그러면 이제 트랜잭션 데이터가 바인더커널드라이버로 배달되었다.

Summary:

Proxy 오브젝트는 RPC 호출을 위해 필요한 패킷을 생성한다. 그리고 바인더커널드라이버에 패킷을 쓰기 위해 `BINDER_WRITE_READ`를 불러낸다. 그 패킷은 형식이 정의된 패킷이다. RPC호출을 위해서 `BC_TRANSACTION` 패킷 타입을 사용한다.

여러분의 `IFunnyTest`를 실행하려면 다음을 따라해라.

- 여러분의 서비스가 실행중인 프로세스에서 `IServiceManager::addService`를 호출해서 `servier_manager`에 서비스를 등록해라.

Transaction in Binder Kernel Driver

어떤 프로세스가 /dev/binder 드라이버를 열 때, 상응하는 binder_proc structure는 binder_open에서 할당될 것이다.

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

    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);
        create_proc_read_entry(strbuf, S_IRUGO, binder_proc_dir_entry_proc,
                               binder_read_proc_proc, proc);
    }
    return 0;
}
```

그래서 어떤 ioctl이 도착했을 때, 드라이버는 프로세스 정보를 알 수 있다. 트랜잭션 데이터는 BINDER_WRITE_READ ioctl을 통해 전송된다.

```
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 (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 (copy_to_user(ubuf, &bwr, sizeof(bwr))) {
    ret = -EFAULT;
    goto err;
}
break;
}

```

Driver first handles는 쓰고, 읽는다. binder_thread_write를 보자. binder_thread_write의 핵심은 write 버퍼의 명령어를 parse 하고, 해당 커맨드를 실행하는 루프이다.

```

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 ***:

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

```

우리는 두개의 커맨드가 시나리오에 관련됨을 볼수 있다. 그 중 하나는 BC_INCREFs 이다.

```

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);
    } 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;
    }
    break;
}

```

시나리오에서 target은 0이 될것이라고 우리가 전에 언급했던 것을 기억해라 system_manager 가 BINDER_SET_CONTEXT_MGR ioctl을 호출할 때 binder_context_mgr_mode 는 handle 0 을 나타내도록 생성된다. 그래서 여기서는 단지 binder_context_mgr_node 노드를 약한 참조 하는 것을 증가시킬 뿐이다.

```
binder_context_mgr_node = binder_new_node(proc, NULL);
```

다른 한가지는 BC_TRANSACTION 이다.

```
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;
  }
```

만일 패킷이 하나의 BINDER_TYPE_BINDER 객체를 갖고 있다면 binder_transaction은 하나의 새로운 바인더 노드를 생성한다.

```
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);
      if (node == NULL) {
        return_error = BR_FAILED_REPLY;
        goto err_binder_new_node_failed;
      }
      node->cookie = fp->cookie;
    }
    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;

```

binder_transaction 은 타겟이 handle 0 이라는 것을 알 것이다. 그래서 이것은 target_node, target_proc과 target_thread를 얻기 위해서 다음 분기를 실행한다.

```

} 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 (!(tr->flags & TF_ONE_WAY) && thread->transaction_stack) {
    struct binder_transaction *tmp;
    tmp = thread->transaction_stack;
    while (tmp) {
        if (tmp->from && tmp->from->proc == target_proc)
            target_thread = tmp->from;
        tmp = tmp->from_parent;
    }
}

```

마침내 binder_transaction은 이 request를 리스트에 집어넣을 것이고 binder_thread_read 의 waiting thread를 깨운다.

```

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);

```

이제 binder_thread_read 를 보자. service_manager 가 동작할 때, 하나의 request가 배달될때까지 여기에서 기다리고 있을 것이다.

```
ret = wait_event_interruptible_exclusive(proc->wait, binder_has_proc_work(proc, thread));
```

media_server 로부터 미리 썼기 때문에 프로세스는 깨어났고 실행 됐다. 다음 코드는 media_server의 write 버퍼의 데이터를 system_menager의 read 버퍼로 복사한다.

```
tr.data_size = t->buffer->data_size;
tr.offsets_size = t->buffer->offsets_size;
tr.data.ptr.buffer = (void *)((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);
```

Summary:

이 섹션은 RPC 호출의 클라이언트 사이드에서 서버사이드로의 데이터 흐름을 설명했다.

Service Manager Handle Add Service

지금 까지 service_manager는 media_server로부터 BR_TRANSACTION의 패킷을 얻었다. 그래서 이것은 binder_parser를 호출해서 패킷을 조작한다.

```
case BR_TRANSACTION: {
    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;
}
```

binder_parser는 svc_mgr_handler를 호출하여 BR_TRANSACTION 패킷을 분석하고 이것은 BpServerManager의

프로세스를 예약한다. 다음 binder_txn 구조체는 실제로 binder_transaction_data 구조체와 같다. 우리의 시나리오상 트랜잭션 코드는 SVC_MGR_ADD_SERVICE이다.

```
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;

    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_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;
    }
```

그래서 service_manager는 하나의 서비스를 s라고 이름 짓고 실행할 것이다. 그리고 bio_get_ref를 통해 object 정보를 얻는다.

```
void *bio_get_ref(struct binder_io *bio)
{
    struct binder_object *obj;

    obj = _bio_get_obj(bio);
    if (!obj)
        return 0;

    if (obj->type == BINDER_TYPE_HANDLE)
```

```

        return obj->pointer;

    return 0;
}

```

bio_get_ref는 flatten_binder의 작업을 예약한다. do_add_service는 마지막에 BC_ACQUIRE를 호출하여 ptr에 의해 나타나는 객체에 대한 강한참조(strong reference)를 얻는다.

Summary:

이 섹션은 service manager에 어떻게 서비스가 추가되는지 보였다.

여러분의 IFunnyTest 서비스를 실행하고 싶다면 다음과 같이 해라.

- service_manager의 허가된 service list에 여러분의 service name을 추가해라.

Get IAudioFlinger

서비스 인터페이스를 얻는 유일한 방법은 IServiceManager::getService를 통한 것이다. 예를 들면 여기 있는 이 메소드는 AudioSystem이 하나의 IAudioFlinger를 얻기 위한 것이다.

```

// establish binder interface to AudioFlinger service
const sp<IAudioFlinger> & AudioSystem::get_audio_flinger()
{
    Mutex::Autolock _(gLock);
    if (gAudioFlinger.get() == 0) {
        sp<IServiceManager> sm = defaultServiceManager();
        sp<IBinder> binder;
        do {
            binder = sm->getService(String16("media.audio_flinger"));
            if (binder != 0)
                break;
            LOGW("AudioFlinger not published, waiting...");
            usleep(500000); // 0.5 s
        } while(true);
        gAudioFlinger = interface_cast<IAudioFlinger>(binder);
    }
    LOGE_IF(gAudioFlinger==0, "no AudioFlinger!?");
    return gAudioFlinger;
}

```

IServiceManager::getService 는 BpServiceManger::getService로 호출된다.

```

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;
}

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();
}

```

분석하기 전에 호출은 마지막으로 바인더커널드라이버를 통해 service_manager 프로세스에서 제어된다.

```

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;
}

```

그러면 service_manager는 media_server에 의해 설정된 이전 핸들로 응답한다. (이것은 AudioFlinger의 인스턴스 주소이다. 그러면 BpServiceManger::checkService는 remote()->tracsact 호출로부터 리턴될 것이다. 그리고 IServiceManager를 위해 분석한 것은 새로운 다른 BpBinder 인스턴스가 해당하는 핸들을 service_manager로부터 리턴할 것이다. interface_case<IAudioFlinger>(binder)는 결국 하나의 BpAudioFlinger 인스턴스를 리턴할 것이다.

Summary:

IServiceManager를 얻는 것 처럼, 하지만 이번에 이것은 service_manager로부터 핸들을 얻는 것이 필요하다. While for IServiceManager it always use handle 0.

RPC Call IAudioFlinger::SetMode

만일 우리가 AAA 프로세스에서 IAudioFlinger::SetMode를 호출하면 사실 BpAudioFlinget::setMode 를 호출한것이다.

```

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();
}

```

IServiceManager::addService의 분석처럼 이 함수는 하나의 패킷을 생성하고 바인더커널드라이버에 이것을 쏜다. 그리고 읽기 응답을 기다린다. 한가지 다른점은 이때 타켓 핸들은 media_server 프로세스의 어떤 주소를 가리킨다는 것이다.

Handle IAudioFlinger::SetMode

바인더커널드라이버는 결국 media_server 프로세스의 Read 쓰레드를 깨울 것이다. 이것은 IPCThreadState::joinThreadPool에서 실행된다. 코드를 다시 보자.

```

void IPCThreadState::joinThreadPool(bool isMain)
{
    mOut.writeInt32(isMain ? BC_ENTER_LOOPER : BC_REGISTER_LOOPER);

    status_t result;
    do {
        int32_t cmd;
        result = talkWithDriver();
        if (result >= NO_ERROR) {
            size_t IN = mIn.dataAvail();
            if (IN < sizeof(int32_t)) continue;
            cmd = mIn.readInt32();
            result = executeCommand(cmd);
        }

        // 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);

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

```

```
}
```

이번에 `talkWithDriver`는 `BpServiceManager::serMode` 가 생성한 패킷을 리턴할 것이다. 그리고 `excuteCommand`는 커맨드를 처리할 것이다. 이 시나리오에서 커맨드는 `BR_TRANSACTION` 이다.

```
case BR_TRANSACTION:
```

```
{
    binder_transaction_data tr;

    Parcel reply;
    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 {
        const status_t error = the_context_object->transact(tr.code, buffer, &reply, 0);
        if (error < NO_ERROR) reply.setError(error);
    }

    if ((tr.flags & TF_ONE_WAY) == 0) {
        LOG_ONeway("Sending reply to %d!", mCallingPid);
        sendReply(reply, 0);
    } else {
        LOG_ONeway("NOT sending reply to %d!", mCallingPid);
    }
}
break;
```

가장 중요한 두줄은 빨간색으로 표시했다. 여기에서 바인더커널드라이버로부터 하나의 주소를 얻고 `BBinder` 포인터로 보낸다. (`IServiceManager::addService`를 호출 했을 때 바인더커널드라이버로 넣은 주소) `AudioFlinger`는 `BBinder`로부터 전달된다는 것을 기억해라. 이 포인터는 사실 우리의 `AudioFlinger` 인스턴스의 포인터와 같은 것이다. 그래서 다음 `transact` 호출은 결국 우리의 `BnAudioFlinger`의 `onTransact` 가상 함수를 호출할 것이다.

```
status_t BnAudioFlinger::onTransact(
    uint32_t code, const Parcel& data, Parcel* reply, uint32_t flags)
{
    case SET_MODE: {
        CHECK_INTERFACE(IAudioFlinger, data, reply);
        int mode = data.readInt32();
        reply->writeInt32( setMode(mode) );
        return NO_ERROR;
    }
}
```

```

    } break;
}

```

그러면 응답은 sendReply를 통해 써질 것이다.

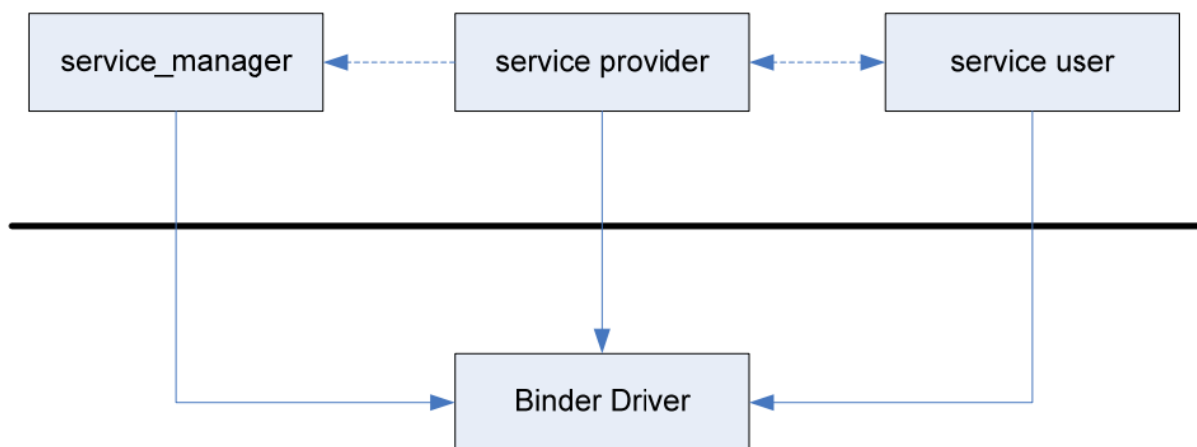
```

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);
}

```

이것은 결국 바인더커널드라이버에 쓸 것이다. 그러면 커널드라이버는 AAA의 읽기 스레드를 깨울꺼울 것이다.



이 그림에서 안드로이드 IPC 시스템의 전반적 아키텍처를 보여준다. 여기에는 네가지 주요 블록을 있다.

- Binder Driver
이것은 IPC 시스템의 핵심이다. 이것은 service proviser와 service user 간에 데이터를 전달한다.
- service provider
이것은 몇가지의 서비스를 제공한다. 이것은 바인더드라이버로부터 받은 RPC 호출 데이터를 파스 파스할 것. 그리고 실제 동작을 한다.
- service_manager
이것은 특별한 service provider이다. 이것은 다른 service provider를 위해 service manager 서비스를 제공한다.

다음은 예를 든 시나리오의 주요 제어 흐름 리스트 이다.

1. 처음에는 service_manager를 실행시키고 이것은 special node 0를 바인더 드라이버에 등록할 것이다.

2. media_server 는 special node 0 를 위해 하나의 IServiceManager proxy 오브젝트를 얻는다.
3. media_server RPC 는 IServiceManager::AddService를 호출해서 IAudioFlinger 서비스에 추가한다. 이 호출은 node 0으로 발송된다. 이것은 바인더드라이버로 데이터를 보낼 것이다.
4. 바인더드라이버는 node 0을 위한 데이터를 통지하고 이 데이터는 한 오브젝트를 묶은 명령어를 포함한다. 그래서 이것은 IAudioFlinger 서비스를 위한 다른 하나의 노드를 생성하고 데이터를 service_manager로 보낸다.
5. service_manager는 바인더드라이버로부터 데이터를 읽고 IServiceManager:addService RPC 호출을 처리한다.
6. 다른 프로세스 P는 special node 0 을 위해 IServiceManager proxy 오브젝트를 얻어온다.
7. P RPC는 IServiceManager::getService를 호출하여 IAudioFlinger 서비스를 얻어온다. 이 호출은 node 0 으로 전달된다. 이것은 바인더드라이버로 데이터를 보낼것이다.
8. 바인더드라이버는 node 0 을 위한 데이터를 알려준다 그래서 이것은 데이터를 service_manager로 전달할 것이다.
9. service_manager는 바인더드라이버로부터 데이터를 읽는다. 그리고 IServiceManager::getService RPC 호출 을 처리하고 IAudioFlinger 서비스를 나타내는 node A를 리턴한다.
10. P RPC는 IAudioFlinger::setMode를 호출한다. 지금 이 호출은 node A로 전달될것이다.
11. 바인더드라이버는 데이터는 node A를 위한 것이라고 알려준다. 그래서 데이터를 media_server로 전달할 것이다.
12. media_server 는 바인더 드라이버로부터 데이터를 읽는다. IAudioFlinger::setMode RPC 호출을 조작 하고 바인더드라이버로 응답데이터를 보낸다.
13. 바인더드라이버는 P로 응답을 전달한다.
14. P는 바인더드라이버로부터 데이터를 읽고 결국 응답을 얻는다.