# Assignment #4 - Mini-Binder

System Programming (Autumn 2025) — Dec 10, 2025

## Background

> **OpenBinder**
>
> **OpenBinder**는 프로세스 간 통신을 위한 시스템이다. Be Inc.와 Palm, Inc.에서 개발되었으며 구글에서 개발한 안드로이드 운영체제에서 현재 사용되는 바인더 프레임워크의 기반이 되었다.
>
> OpenBinder는 프로세스가 다른 스레드에 의해 호출될 수 있는 인터페이스를 제공하도록 허용한다. 각 프로세스는 이러한 요청을 처리하는 데 사용될 수 있는 스레드 풀을 유지한다. OpenBinder는 참조 횟수 계산, 원래 스레드로 다시 재귀하는 것, 그리고 프로세스 간 통신 자체를 처리한다. OpenBinder의 리눅스 버전에서 통신은 주어진 파일 디스크립터에 대한 ioctl을 사용하여 커널 드라이버와 통신함으로써 이루어진다.
>
> OpenBinder 리눅스 버전의 커널 측 구성 요소는 2015년 2월 8일에 출시된 커널 버전 3.19의 리눅스 커널 메인라인에 병합되었다.

ko: https://ko.wikipedia.org/wiki/오픈바인더

en: https://wikipedia.org/wiki/OpenBinder

## Goal

이번 과제의 목표는 **MiniBinder 예제 코드를 통해 IPC의 호출스택을 분석**하면서, 커널 관점에서 IPC가 어떻게 흘러가는지 이해하는 것입니다. 하나의 RPC 요청이 **client → kernel → server → kernel → client** 로 왕복하는 과정을 호출스택 기준으로 추적·정리하고, 이 과정에서 **IPC를 처리하기 위한 커널 측 요구사항과 동작 방식**을 분석합니다

## Mini-Binder

`Binder`는 다른 프로세스에서 제공해주는 기능(서비스)를 가리키도록 묶어둔 인터페이스 객체이다. 이 하나의 객체를 바인더 오브젝트라고도 부른다. Minibinder 프로젝트에서는 바인더 오브젝트는 2가지 요소로 구성되어 있다.

- `token` : 사람이 식별할 수 있도록 제공된 문자열 식별자.
- `handle` : 커널이 내부적으로 식별하기 위한 정수형 식별자.

아래의 코드는 MiniBinder 프로젝트에서 Binder를 통해 RPC를 수행하는 예를 보여준다.

```
// client side code (user-space)
token_t t;
// 'some_service'라는 이름을 토큰으로 변환
str2tok("some_service", &t);

bind( ... , &t, &binder);
/** 이제 'b' 가 'some_service'라는
 *  다른 프로세스의 서비스를 가리키는
 *  바인더 오브젝트(proxy)가 된다.
 */

// binder ipc call
transact(binder, TRANSACT_EXAMPLE, ... );
```

```
// server side code (user-space)
token_t t;
// 'some_service'라는 이름을 토큰으로 변환
str2tok("some_service", &t);

binder_init( ... , t, &binder);
binder_register(&binder, on_transact);
/** 이제 'b'가 some_service를 제공하는
 *  바인더 오브젝트(stub)가 된다.
 */
// binder ipc handle
on_transact( ... ) {
        switch() {
          case TRANSACT_EXAMPLE:
          // RPC Handling
        }
}
```

1. 서버에서 먼저 “some_service”라는 토큰으로 stub을 등록한다.
2. 이후, 클라이언트에서 “some_service”라는 토큰으로 바인더 오브젝트를 받도록 요청한다 (`bind`).
3. 바인더 드라이버는 클라이언트에게 some_service 토큰으로 등록된 바인더를 찾아, 바인더 오브젝트를 리턴한다.
4. 이후 클라이언트에서는 바인더 오브젝트를 통해, RPC를 수행할 수 있다.
    - 해당 RPC는 IPC를 통해 전달되어 서버에서 직접 실행되고, 실행 결과가 리턴된다.

> **TA Comment.**
> 실제 Binder 기반 RPC는 객체지향 언어 위에서 구현되는 것이 일반적이지만,
> MiniBinder에서는 C 언어로 제한적으로만 구현되었습니다.

## Requirements

### *R1.* Code Analysis

MiniBinder 프로젝트를 직접 빌드·실행해보고, **한 번의 RPC 요청이 어떻게 왕복하는지 호출스택을 분석한다**.

1. 호출스택 분석
    - 먼저, 클라이언트의 RPC 요청 (`transact` 호출)로 부터 시작하여, 라이브러리 계층에서 함수가 어떤 순서로 호출되는지, 그리고 각 함수가 내부적으로 어떤 커널 호출로 이어지는지 추적한다.
    - 이후, 커널 모듈의 코드와 서버 코드에서 위와 동일한 작업을 진행한다.
2. **IPC 관점에서의 메커니즘 분석**
    - 유저 공간과 커널 공간을 오가는 과정에서
        - 어떤 커널 API가 사용되는지,
        - zero-copy를 위해 **어떤 데이터는 copy 기반이고, 어떤 데이터는 mmap(shared buffer) 기반인지**를 코드 근거와 함께 설명한다.
    - 위 내용을 바탕으로, IPC를 지원하기 위해 만족해야 하는 **커널 측 요구사항(라우팅, 동기화, 메모리 공유 방식 등)** 을 정리한다.

### *R2.* Discussion

아래의 주제에 대하여 토의한 보고서를 작성한다.

1. Binder는 다른 IPC 시스템들과 비교하였을 때 실제로 **더 빠른** IPC 성능을 보인다고 생각한다면 그 이유는 무엇인지, 그렇지 않다고 생각하신다면, 그럼에도 불구하고 Binder를 사용하는 이유는 무엇인지에 대해 토론하십시오.
2. 실제 안드로이드에서 사용되는 Binder 드라이버는 송수신 데이터를 직렬화하는 과정에서 **4바이트 정렬(4-byte alignment)** 을 수행합니다. 이는 IPC 수행 과정에서 아키텍처의 워드 단위가 4바이트이기 때문에 의미 있는 성능 향상을 가져온다는 주장과도 연결됩니다. 이 주장에 동의한다면 그러한 **성능 향상이 발생하는 원리**에 대하여, 그렇지 않다고 생각하신다면 그럼에도 **4바이트 정렬을 사용하는 이유**에 대하여 토론하십시오.
3. 현대의 소프트웨어에서는 양방향 RPC를 사용하는 경우가 많지만, MiniBinder는 한 시점에 **단 한 번의 트랜잭션만** 처리하도록 설계되어 있습니다. 이는 동시에 진행되는 양방향 RPC를 지원하기 어렵다는 것을 의미합니다. 특히, A 프로세스가 B 프로세스에게 RPC를 요청해 **블로킹된 상태**에서, B 프로세스가 다시 A 프로세스에게 RPC를 요청하는 상황을 가정하면, A는 이미 대기 상태이므로 B의 요청을 처리할 수 없고, 결국 시스템은 **교착 상태(Deadlock)** 에 빠지게 됩니다. 왜 이러한 제약이 발생하는지, 그리고 이 문제를 해결하여 **양방향 RPC를 안전하게 지원**하려면
    - 프로세스 / 스레드 구조
    - 인터럽트에 따른 프로세스 상태 전이

    등 어떤 측면에서 설계적 고민이 필요한지, **특히 Interrupt 관점**을 중심으로 토론하십시오.

4. 실제 MiniBinder 설계에서는 **요청(Request)은 copy_from_user로 커널 내부 구조체에 복사**하고, **응답(Reply)은 mmap된 shared buffer에서 직접 읽는 pseudo zero-copy** 형태로 동작합니다. 이와 같은 설계가 **어느 구간까지를 "zero-copy"라고 부를 수 있는지**, 그리고 단순 copy_from_user / copy_to_user 기반 IPC와 비교했을 때의 장단점을 중심으로, "zero-copy IPC 설계"가 실제 시스템에서 어느 정도까지 의미가 있는지 토론하십시오.
5. MiniBinder의 설계에서는 IPC 요청 시, IPC 메세지를 해석하기 위해 **유저-커널간 메모리 복사**를 수행합니다. 반면에, 실제 데이터는 메세지에 포인터나 오프셋만을 담아 메타데이터 형식으로 전달하고, 이를 기반으로 제로카피 방식으로 접근하도록 설계되어 있습니다. 이는 "복사를 진행하지 않는다"는 제로카피의 철학과 모순되는 설계방식에 해당합니다. 이러한 구조가 **성능 향상 측면에서 여전히 유의미하다고 생각한다면 그 이유를**, 그렇지 않다고 생각한다면 어떤 다른 설계가 더 적절하다고 보는지에 대해 토론하십시오.

## Submission

**제출 요구사항**

- **제출 기한 - 12월 24일 (수) 11:59 PM**
- **제출 파일**
  - 작성한 보고서를 자신의 학번을 이름으로 하여 PDF 파일로 제출한다. (보고서 양식은 자유)
    (e.g. 202224210.pdf)

## Grading Criteria

본 과제는 총 30점으로, 다음 기준을 토대로 채점을 진행합니다.

- Code Analysis 보고서 **(18)**
  - 호출스택 분석 (6)
    - 호출스택 분석이 MiniBinder와 적지 않게 상이한 경우, 예외없이 0점.
  - 메커니즘 분석 (12)
- Discussion 보고서 **(12)**
  - 5개의 토의를 전부 완료했음 (6)
  - 5개의 토의에서 전부 우수하게 답변함 (6)

**기타 감점사항**

- **plagiarism** : 과제에 AI(GPT, Gemini 등) 사용 및 치팅(카피)가 **심히 의심되는** 경우, 예외없이 0점
- 제출 형식이 잘못된 경우, 예외없이 -1점
- 지각제출 시 예외없이 기본 -1점, 시간당 -1점

> **TA Comment**
>
> 본 과제의 답안은 **수업에서 다룬 내용**을 기반으로, 그 위에서 한 단계 정도 심화하여 생각한 수준을 기대합니다.
>
> 다시 말해, 수업과 직접적인 관련이 없는 외부 지식(생성형 AI가 생성한 일반론, 무분별한 인터넷 검색 내용, 과도하게 추상적인 설명 등)을 길게 나열한 답안은 감점될 수 있습니다.
>
> 다만, 다른 과목에서 배운 내용이나 참고 서적의 이론을 바탕으로, 이를 본인의 언어로 정리하여 답안을 작성하는 경우는 참작 대상이 될 수 있습니다.
>
> 어느 정도까지를 "적절한 심화"로 볼지는 별도로 공개하지 않습니다.