

# Contents

<b>1</b>	<b>IPC</b>	<b>2</b>
<b>2</b>	<b>Binder</b>	<b>2</b>
2.1	Facilities . . . . .	2
2.2	Implementation . . . . .	3
2.3	Transactions . . . . .	4
<b>3</b>	<b>Defining Remotely Bound Services</b>	<b>5</b>
<b>4</b>	<b>IPC Abstraction</b>	<b>5</b>
<b>5</b>	<b>Binder Objects and Tokens</b>	<b>5</b>
<b>6</b>	<b>ServiceManager</b>	<b>6</b>
<b>7</b>	<b>Binder Security</b>	<b>6</b>

# 1 IPC

Inter-process communication. Each process has its own address space. This provides data isolation and prevents direct interaction between different processes. How can we communicate with a Service, or send an Intent? Use **Binder**.

- Underpins most Android communication, i.e. when we use various system capabilities
- Kernel driver: provides lightweight RPC (remote procedure calls), data passing. C.f. Linux/Unix signals / pipes / sockets etc. Reading and writing Parcels between processes. Process, user ID authority / trust
- Per-process thread pool for handling requests
- Synchronous calls between processes

## 2 Binder

### 2.1 Facilities

- Calls: Simple inter-process messaging system. One-way, two-way
- Identifying: PID, UID
- Notification: Link to death, Leaked Service connections
- Managing: Reference counting, object mapping across processes, sleeping and waking worker threads
- Indirect functionality: As a token, sharing fd (file descriptor) to shared memory area

## 2.2 Implementation

## API for apps

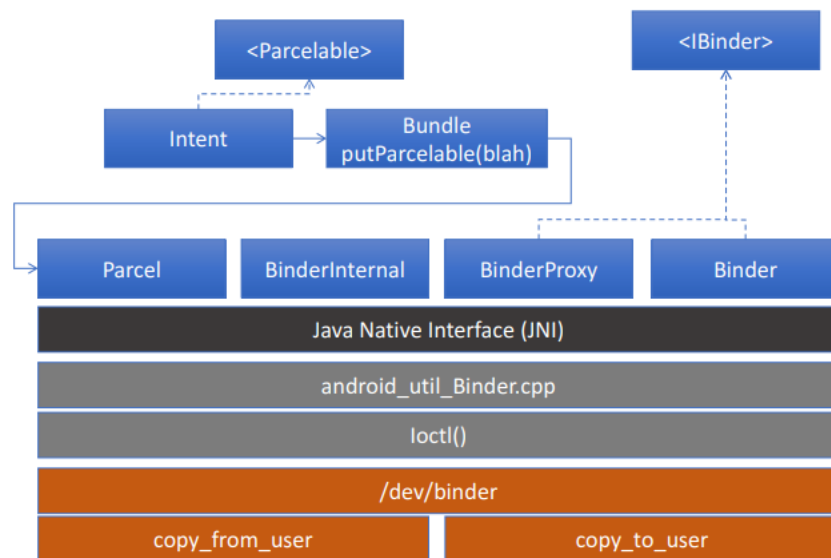
- AIDL
- Java API wrapper
  - Exposes the IBinder interface
  - Wraps the middleware layer
  - Parcelable object marshalling interface

## Native Middleware

- Implements the user space (i.e. within a process) facilities of the Binder framework
- Marshalling and unmarshalling of specific data to primitives
- Provides interaction with the Binder kernel driver

Kernel driver

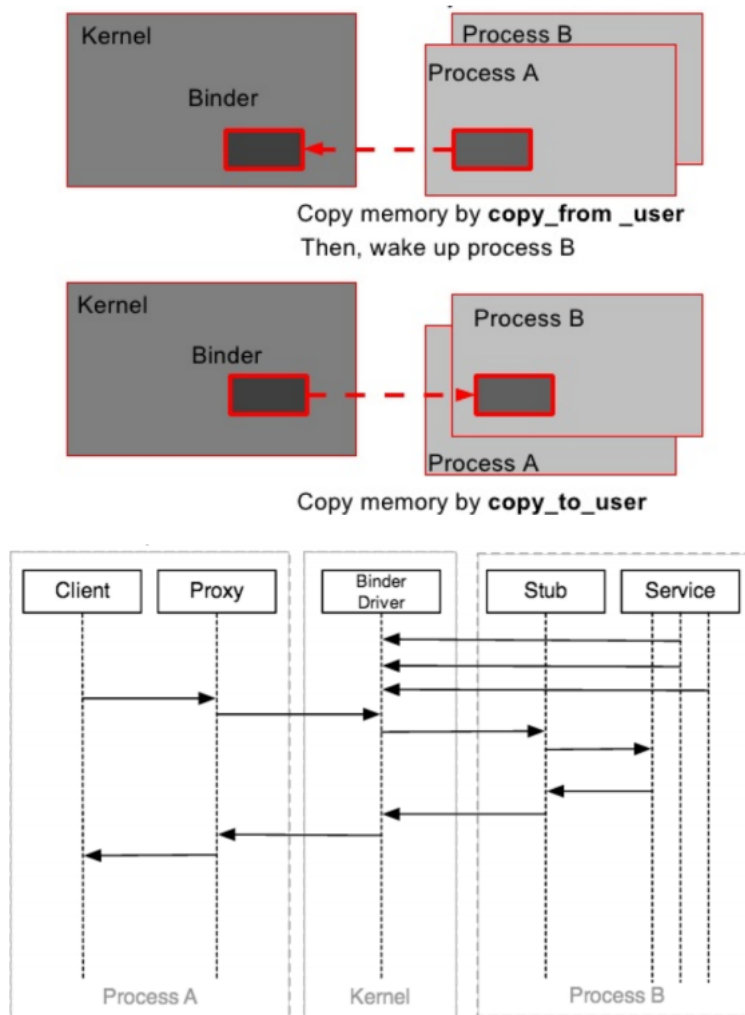
- Supports ioctl system calls from the middleware
- Supports cross-process file operations, memory mapping
- Thread pool for each service application for IPC
- Mapping of objects between processes via `copy_from_user`, `copy_to_user`



## 2.3 Transactions

A transaction between processes: `IBinder.transact -> Binder.onTransact()`. Binder maintains a pool of transaction threads in each process

- To dispatch all IPCs coming from other processes
- If process A calls process B
  - Calling thread in A blocks in `transact()`
  - Sends the transaction to process B
  - Next available thread in B receives the incoming transaction, calls `onTransact()` on the target object, replies with the resultant Parcel
  - Thread in process A returns, resumes execution
- Reliant on Service (B) responding in a timely manner
  - Hence catching remote exceptions, transaction failures
  - Developer defined worker threads
  - Handling multiple calls from multiple transaction threads



## 3 Defining Remotely Bound Services

Using the Android Interface Definition Language (AIDL)

- Specify an interface for the service functionality
- Generates a proxy object
- To be used locally as if the remote service was not remote
- Generates a stub implementation
- The remote side of the transaction
- Generates a communication protocol
- Parcelling and unparcelling steps

Similar to Java interface definitions. Has label method parameters for efficiency

- in: transferred to the remote method
- out: returned to the caller
- inout: both in and out
- oneway: asynchronous

Permitted types

- Java primitive types, Lists, Maps
- Other AIDL-generated interfaces
- Classes implementing the Parcelable protocol

## 4 IPC Abstraction

Intent

- Highest level abstraction
- Asynchronous message passing

**Inter-process** method invocation by using AIDL (Android Interface Definition Language). **Binder**: kernel driver. **Ashmem**: Shared memory.

- Passed as file descriptor objects by Binder
- The USB service gives a specific USB device to an app without giving it unrestricted access

## 5 Binder Objects and Tokens

Binder Object

- An object that can be accessed through the Binder framework
- Implements the IBinder interface
- A unique identity maintained across processes
  - Allocated by the Binder driver. Cannot be duplicated. Binder objects maintain a unique ID even when parcelled
  - A 32 bit handle maintained by the kernel

Process A creates a binder object <- references memory directly

- Passes it to process B <- referenced by handle
- Passes it to process C <- referenced by handle

## Capability-based security model

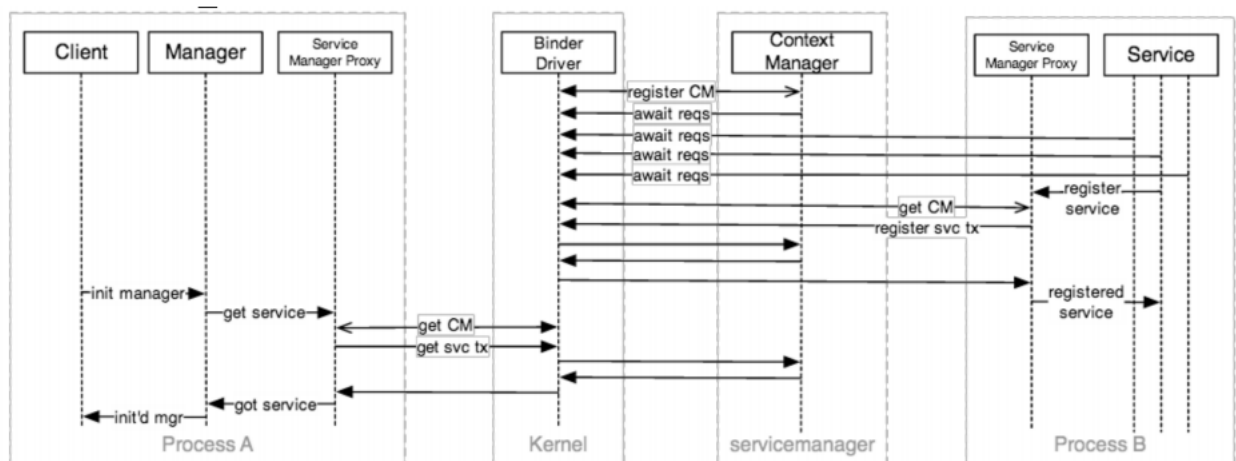
- Processes are granted access to a particular resource by giving them a capability in the form of the binder object. Binder object as token.
- The possession of a token grants the owning process full access to the Binder object enabling it to perform Binder transactions on the target object
- The only way to communicate with a Binder object is to be given a reference to it

## 6 ServiceManager

A single context manager that maintains references to Binder objects

- Implemented as ServiceManager
- Also hosts many system services within its process
- A Binder instance with a known Binder handle (0)
- Knows about other remote services
  - The first to be registered with Binder
  - Only trusted system services allowed to register. System, radio, media

Client does not know the token of remote Binder. Only the Binder interface **knows its own address**. Binder submits a service name and its Binder token to the ServiceManager via IPC. Client retrieves remote service Binder handle with service name and communicates with remote service



## 7 Binder

### 7.1 Security

Binder doesn't deal with security

- Enables a trusted execution environment
- Transactions via the kernel
- Client identity managed by the kernel
- `Binder.getCallingUid()`, `Binder.getCallingPid()`
- UID / PID included in each transaction

Access controlled in two ways

- Limit who can obtain a reference to a Binder object. Has interface reference security. Client cannot guess address of a service without going via the Service Manager.

- Check caller identity before performing an action on the Binder object
- Service asks package manager about UID permissions
- Check whether it holds a permission we want to enforce via `PackageManager.getPackageInfo(...)`

## 7.2 Performance

### Explicit limitations

- Transactional buffer size 1Mb per process for all concurrent transactions in a process
- Many moderately sized transactions could also exhaust its limit. Arguments and return values are too large
- Keep transaction data small

### Implicit limitations

- Data is copied: duplication of memory resources
- Native binary marshalling: better than reflection based serialization, still has overhead of parcel marshalling
- Not ideal for large data-streams, but good enough for window / activity / surface management -& graphics
- Pass file descriptors to shared memory regions (ashmem - Android shared memory)

**placeholder**