第十六讲：进程通信

第 3 节：Linux 信号机制

向勇、陈渝

清华大学计算机系

xyong,yuchen@tsinghua.edu.cn

2020 年 4 月 12 日

提纲

1. 第 3 节：Linux 信号机制
   - Signal Model
   - Signal Handler Control Flow
   - Signal handlers

Ref: Understanding the Linux Kernel
Signals and Inter-Process Communication

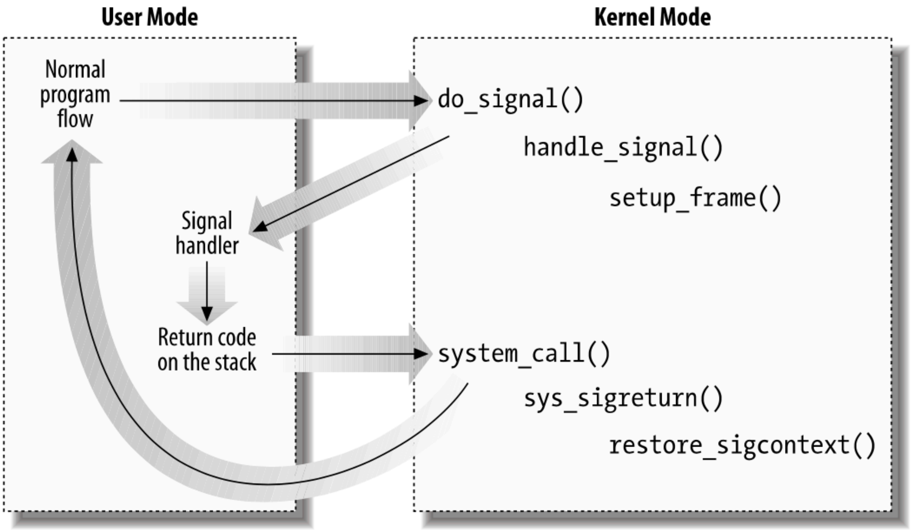
Signal Model

- Application registers handlers with signal() or sigaction()
- Send signals with kill() and friends
  - Or raised by hardware exception handlers in kernel
- Signal delivery jumps to signal handler
  - Irregular control flow, similar to an interrupt

Language Exceptions

- Signals are the underlying mechanism for Exceptions and catch blocks
- JVM or other runtime system sets signal handlers
- Signal handler causes execution to jump to the catch block

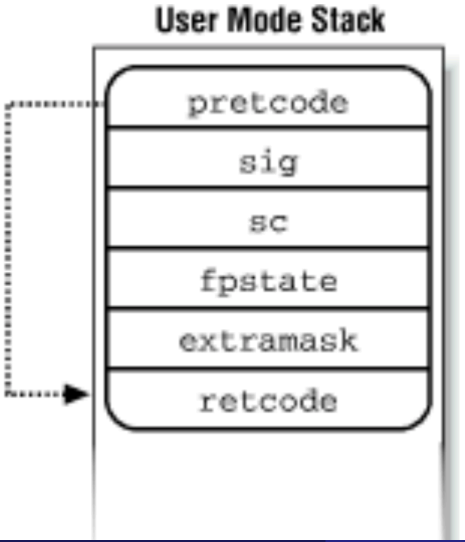
Signal Handler Control Flow



Alternate Stacks

- Signal handlers can execute on a different stack than program execution.
  - Set with sigaltstack() system call
- Like an interrupt handler, kernel pushes register state on interrupt stack
  - Return to kernel with sigreturn() system call
  - App can change its own on-stack register state!

Frame on the User Mode stack



- pretcode: Return address of the signal handler function
- sig: Signal number
- sc: Hardware context of the User Mode process
- fpstate: Floating point registers of the User Mode process
- extramask: Blocked real-time signals
- retcode: Eight-byte code issuing a sigreturn() system call

Signal trampoline & sigreturn() syscall

A small piece of assembly code to perform cleanup after handling the signal.

- Signal trampoline code calls sigreturn().
- sigreturn() undoes everything that was done in order to invoke the signal handler
    - Changing the process's signal mask, switching signal stacks
    - switches stacks, and restores the process's context
    - sigreturn() never returns
- Signal trampoline code lives either in the vDSO or in the C library.
    - vDSO (virtual dynamic shared object): a small shared library that the kernel automatically maps into the address space of all user-space applications.

Dealing With Asynchronous Signals In Multi Threaded Program

The first available thread gets the signal.

- Most handlers run on the thread's stack
- A handler can run on an alternate stack
- Thread in the kernel does not run the handler until it goes to userspace.

Default Signal handlers

- Signals have default handlers:
  - Ignore, kill, suspend, continue, dump core
  - These execute inside the kernel
- Installing a handler with signal()/sigaction() overrides the default
- A few (SIGKILL, SIGSTOP) cannot be overridden

Signal Delivery

- Send a signal == mark a pending signal in the task
  - And make runnable if blocked with TASK_INTERRUPTIBLE flag
- Check pending signals on return from interrupt or syscall
  - Deliver if pending

Nested Signals

- sigaction() API lets you specify this in detail
  - What signals are blocked (and delivered on sigreturn)
  - Similar to disabling hardware interrupts
- Blocking system calls inside of a signal handler are only safe with careful use of sigaction()

第十六讲：进程通信

第 5 节：D-Bus 机制

向勇、陈渝

清华大学计算机系

xyong,yuchen@tsinghua.edu.cn

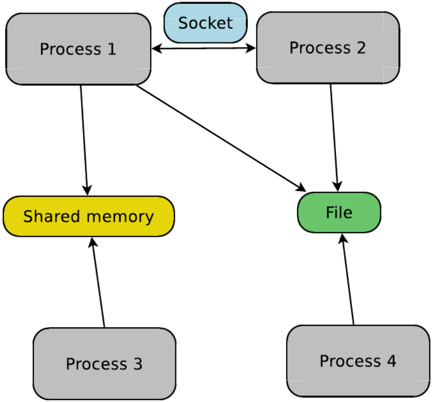
2020 年 4 月 12 日

D-Bus 介绍

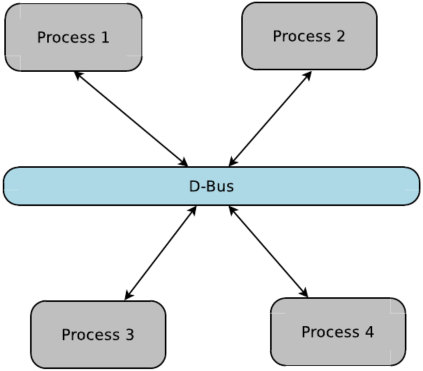
- 2002 年创建的一种进程间通信机制
- 是 freedesktop.org 项目的一部分
- 有 Redhat 和 FreeDesktop 社区维护
- 主要 Linux 桌面环境中的通信服务

D-Bus 介绍

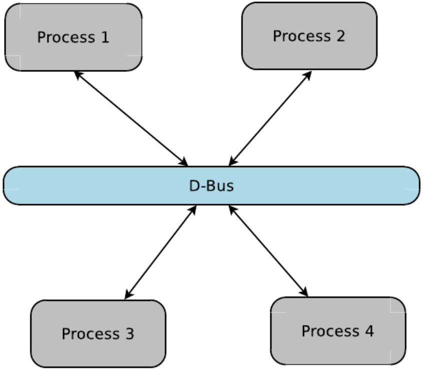
- 早期的 IPC 机制
- socket 相对使用广泛
- 某些机制已经很少使用



- 使用 socket 机制
- 提供了软件总线抽象
- 比传统 IPC 机制简单



D-Bus 介绍



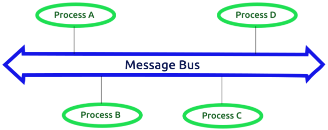
基本特征

- 高层次的 IPC
- Multicast & point-to-point
- OS/architecture/language 无关
- GNOME，KDE，xfce

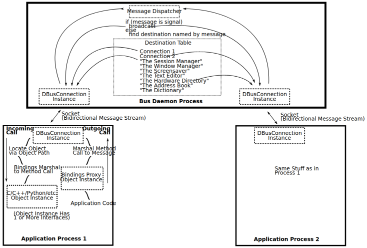
优势

- 低延迟：无 socket 的循环的等待
- 低开销：使用一个二进制的协议
- 高可用性：基于消息机制而不是字节流机制

D-Bus 运行机制



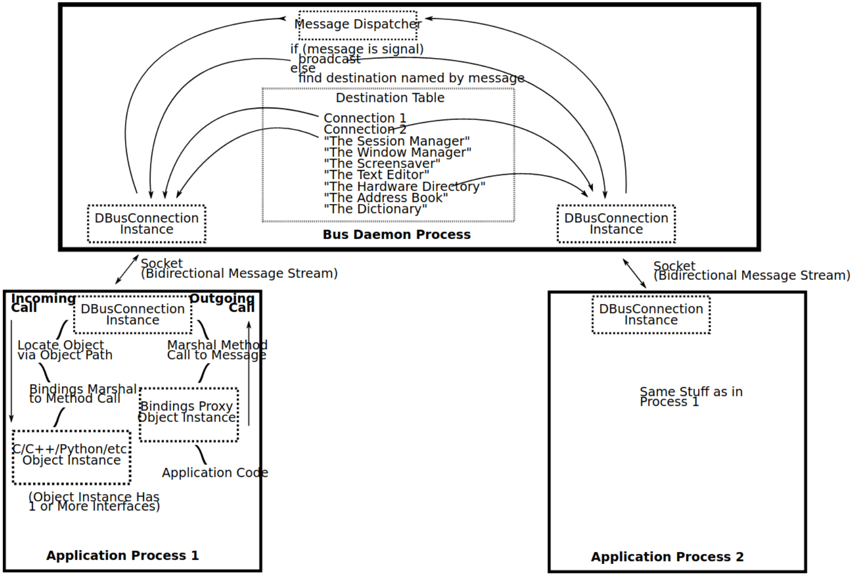
Interprocess communication using D-Bus



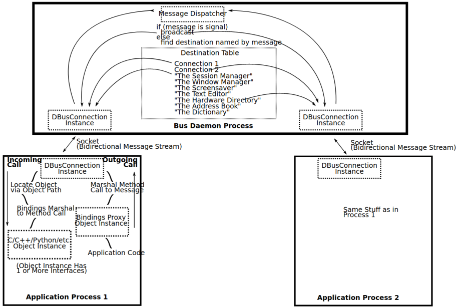
D-Bus 结构

- 一个库 libdbus，它允许两个应用程序相互连接并交换消息
- 一个消息总线守护程序 (daemon)，建立在 libdbus，多个应用程序可以连接到
- 包装程序库或基于特定应用程序框架的绑定
- System Bus & Session Bus
- 通过 policy 文件制定安全机制

D-Bus 运行机制



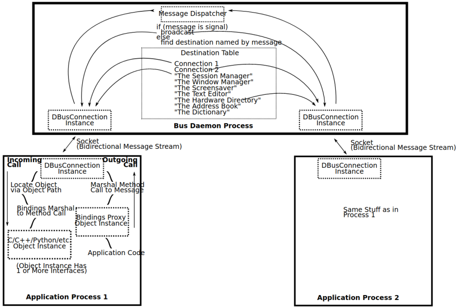
Message Dispatcher
if (message is signal)
broadcast
else
find destination named by message
Destination Table
Connection 1
Connection 2
"The Session Manager"
"The Window Manager"
"The Screensaver"
"The Text Editor"
"The Hardware Directory"
"The Address Book"
"The Dictionary"
DBusConnection Instance
DBusConnection Instance
Bus Daemon Process
Socket
(Bidirectional Message Stream)
Socket
(Bidirectional Message Stream)
Incoming Call
DBusConnection Instance
Outgoing Call
Locate Object via Object Path
Marshal Method Call to Message
Bindings Marshal to Method Call
Bindings Proxy Object Instance
C/C++/Python/etc. Object Instance
Application Code
(Object Instance Has 1 or More Interfaces)
Application Process 1
DBusConnection Instance
Same Stuff as in Process 1
Application Process 2



D-Bus 的总线 bus

- 相当于 D-BUS 的通信链路，应用之间通过总线进行通信。应用在总线上寻找 service
- 系统总线 System Bus：用于 kernel，系统应用/服务
- 任务总线 Session Bus：用于 gnome/kde 等应用通信

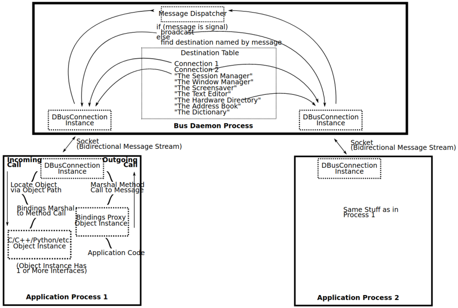
D-Bus 运行机制



D-Bus 的服务 service

- 服务是提供 IPC API 的程序，每个服务都有一个 reverse domain name 结构的标识名称
- org.freedesktop.NetworkManager 对应系统总线上的 NetworkManager
- org.freedesktop.login1 对应系统总线上的 systemd-logind

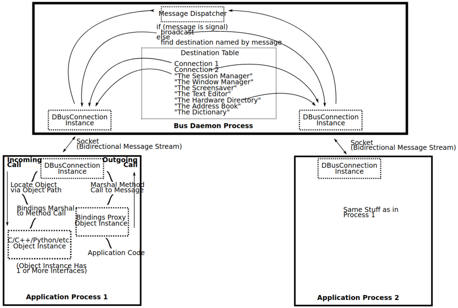
D-Bus 运行机制



D-Bus 的对象 (object)

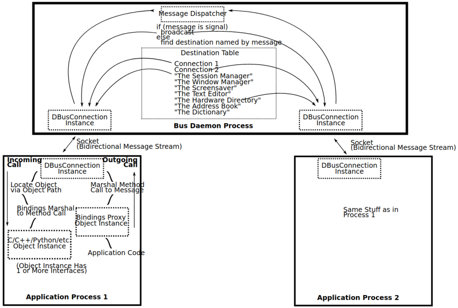
- 相当于通信的地址，每个 service 的 object 都通过 object path 来标识
- object path 类似文件系统的路径
- 如/org/freedesktop/login1 是服务 org.freedesktop.login1 的 manager 对象的路径

D-Bus 运行机制



D-Bus 的接口

- D-Bus 接口定义了 D-Bus 对象支持的方法 method 和信号 signal
- 每个 object 包含一个或者多个 interfaces



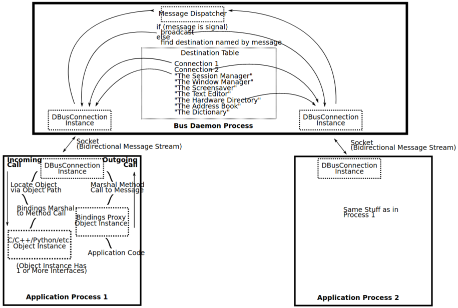
D-Bus 的方法 (method)

- D-Bus 方法可以接受任意数量的参数，并且可以返回任意数量的值，包括任何值。

D-Bus 的信号 (signal)

- D-Bus 信号提供了一对多的发布-订阅机制
- 与方法返回值类似，D-Bus 信号可能包含任意数量的数据
- 与方法不同，信号是完全异步的，并且可以随时由 D-Bus 对象发出

D-Bus 运行机制



D-Bus 的方法 (method) 的执行流程

- 应用调用代理上的方法，代理将构造一个方法调用消息给远端的进程
- 方法调用消息发送到 bus daemon 中
- bus daemon 查找目标的 bus name，如果找到，就把这个方法发送到该进程中
- 在 dbus 高层接口中，会先检测并转换成对应的对象的方法，然后再将应答结果转换成应答消息发给 daemon
- bus daemon 接受到应答消息，将把应答消息直接发给发出调用消息的进程

第十六讲：进程通信

第 6 节：Binder 机制

向勇、陈渝

清华大学计算机系

xyong,yuchen@tsinghua.edu.cn

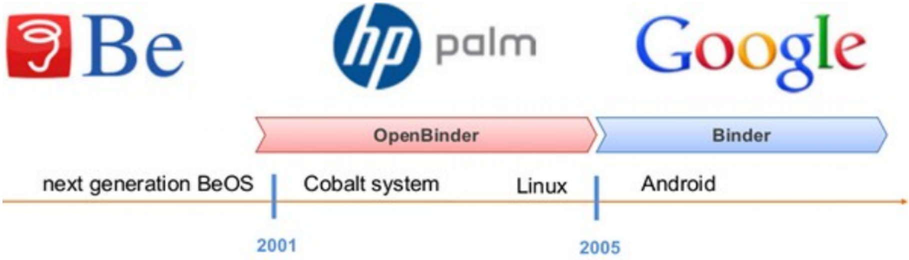
2020 年 4 月 12 日

背景介绍

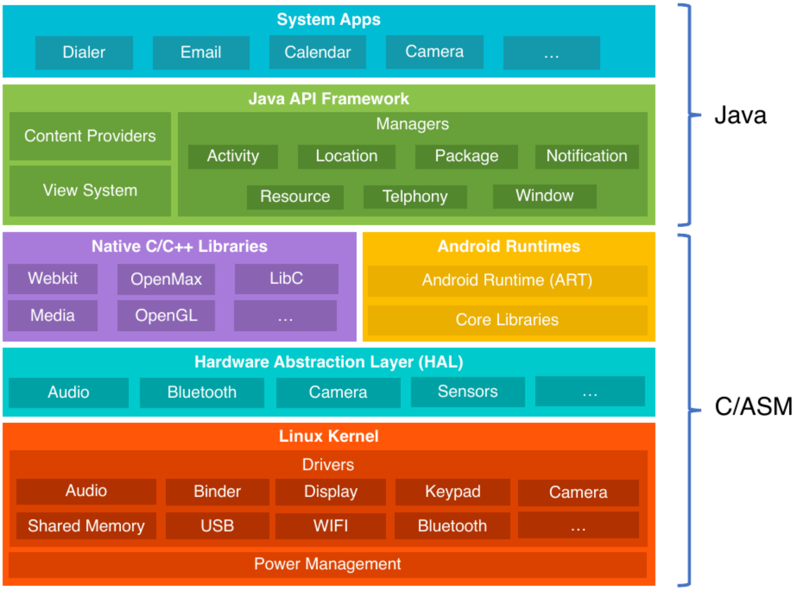

BeOS max
personal edition
The Operating System


palm

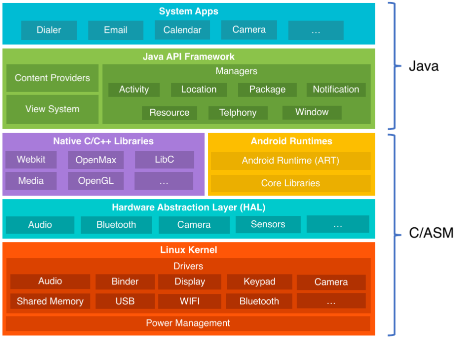

ANDROID
Alpha 09/2008
Beta 02/2009
Cupcake 04/2009
Donut 09/2009
Eclair 10/2009
Froyo 05/2010
Gingerbread 12/2010
Honeycomb 02/2011
Ice Cream Sandwich 10/2011
Jelly Bean 08/2012
KitKat 09/2013
Lollipop 06/2014
Marshmallow 10/2015
Nougat 08/2016
Oreo 08/2017


Be
hp palm
Google
OpenBinder
Binder
next generation BeOS
Cobalt system
Linux
Android
2001
2005

背景介绍


System Apps
Dialer
Email
Calendar
Camera
...
Java API Framework
Content Providers
View System
Managers
Activity
Location
Package
Notification
Resource
Telphony
Window
Native C/C++ Libraries
Webkit
OpenMax
LibC
Media
OpenGL
...
Android Runtimes
Android Runtime (ART)
Core Libraries
Hardware Abstraction Layer (HAL)
Audio
Bluetooth
Camera
Sensors
...
Linux Kernel
Drivers
Audio
Binder
Display
Keypad
Camera
Shared Memory
USB
WIFI
Bluetooth
...
Power Management
Java
C/ASM

背景介绍



Linux kernel v.s. Android

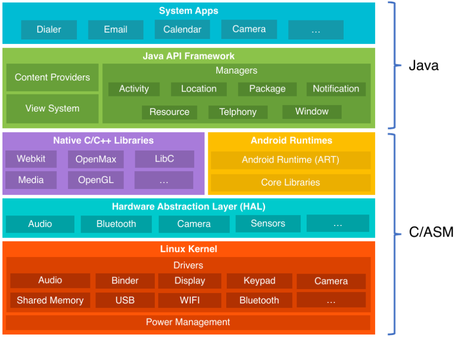
- binder – 新的 IPC 机制
- ashmem – 新的 shared memory 机制
- logger
- ……

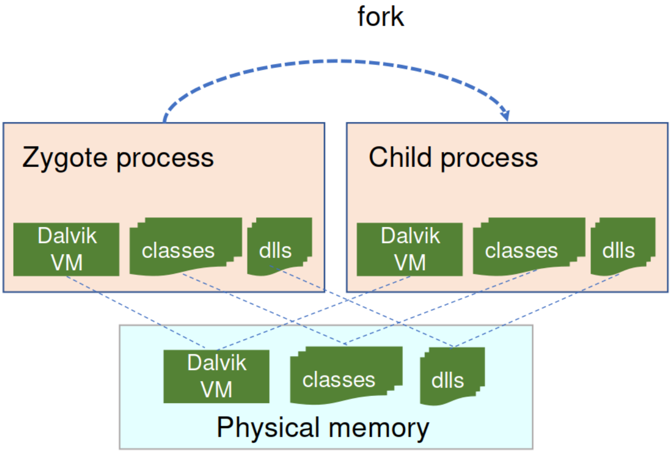
Binder: Android's Solution

"In the Android platform, the binder is used for nearly everything that happens across processes in the core platform. " –Dianne Hackborn https://lkml.org/lkml/2009/6/25/3

背景介绍

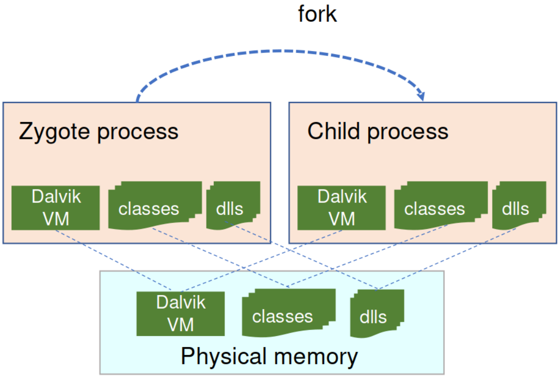
Android 进程



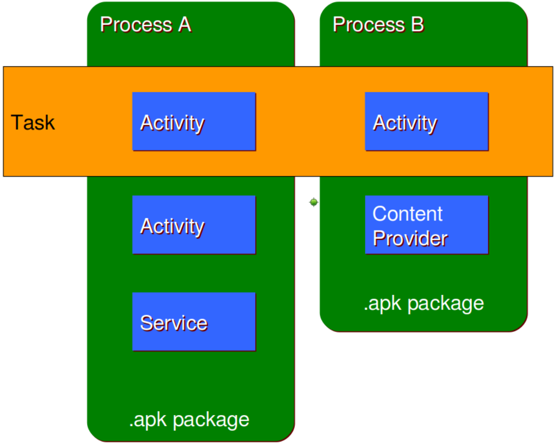


背景介绍

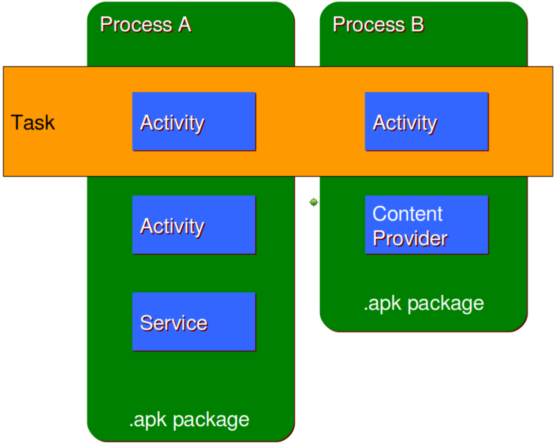
Android 进程



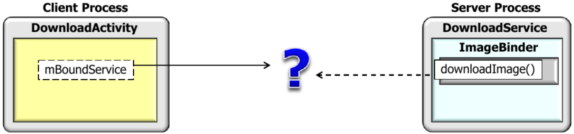
Android 任务 task



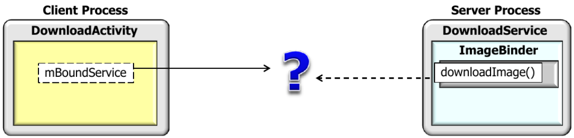
Android 任务 task

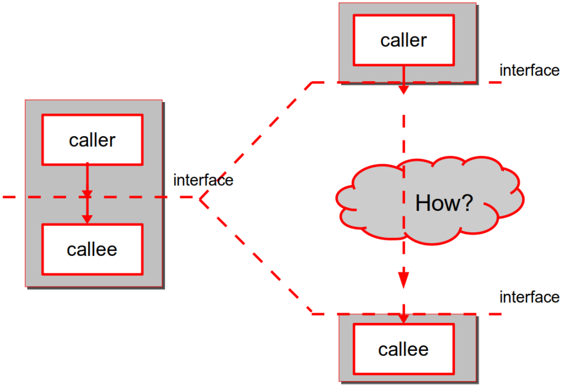


Android 任务如何跨进程交互?



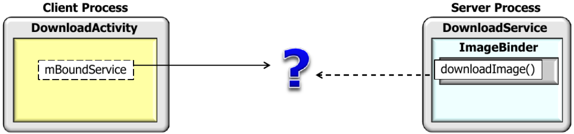
Android 任务如何跨进程交互?

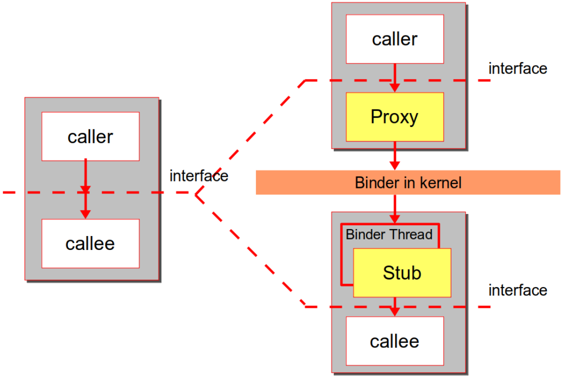




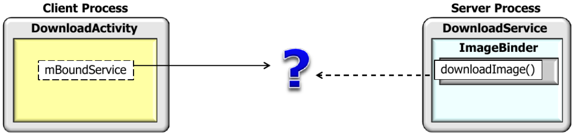
Binder 机制

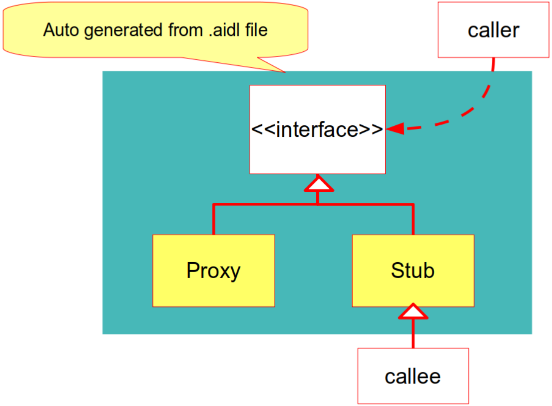
Android 任务如何跨进程交互?



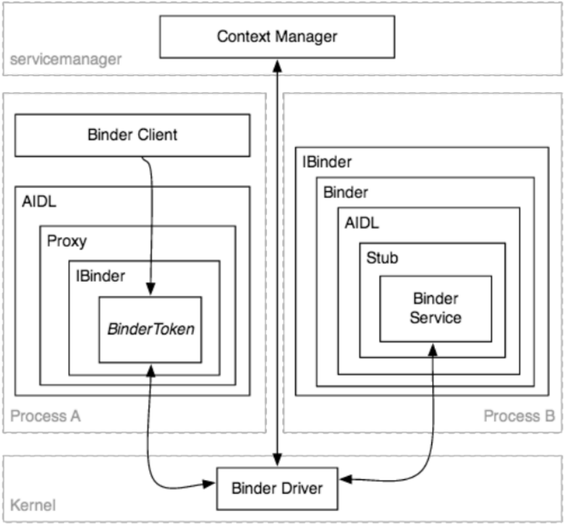


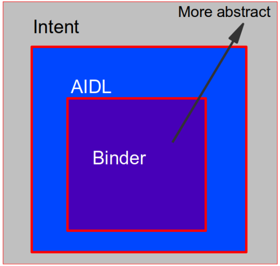
Android 任务如何跨进程交互?





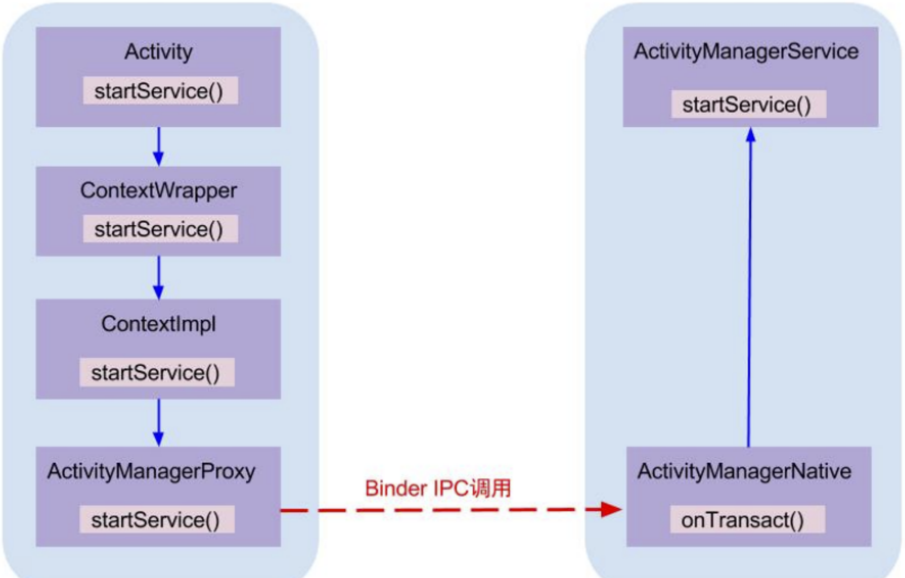
Binder 机制





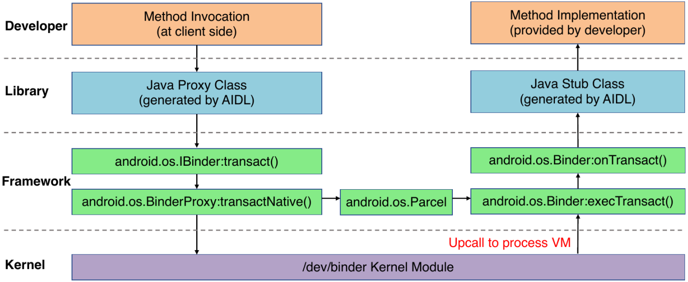
- Intent – 最高层的 IPC 抽象
- AIDL – Android Interface Definition Language
- binder: kernel driver
- ashmem: shared memory

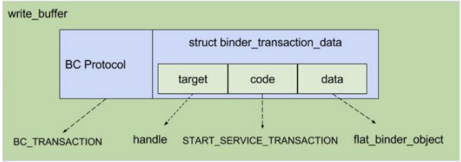
Binder 机制

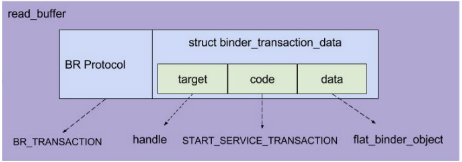


Activity
startService()
ContextWrapper
startService()
ContextImpl
startService()
ActivityManagerProxy
startService()
Binder IPC调用
ActivityManagerService
startService()
ActivityManagerNative
onTransact()

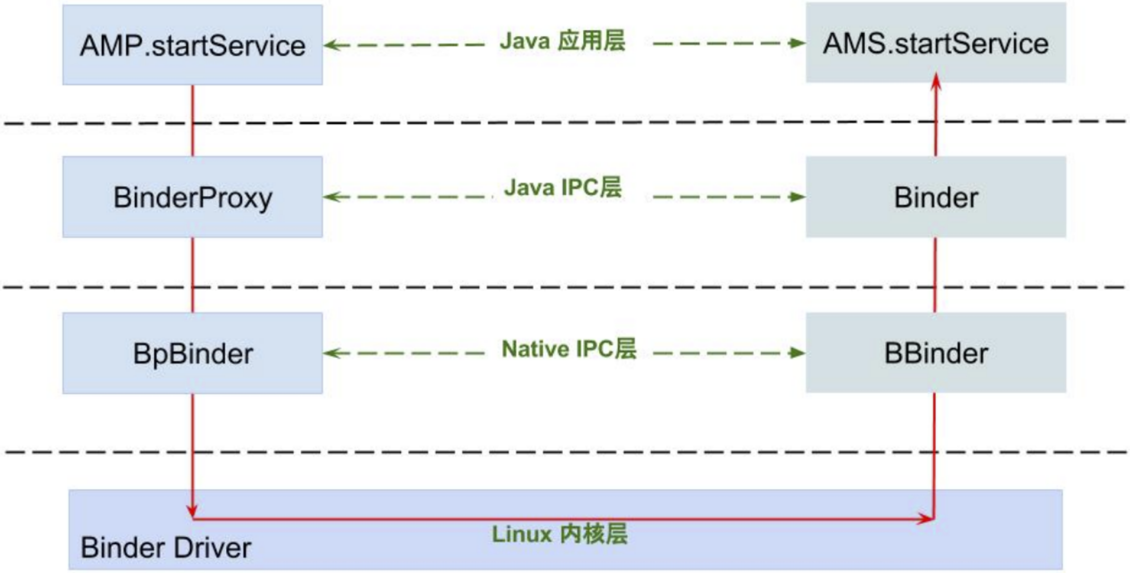
Binder 机制


Developer
Method Invocation
(at client side)
Method Implementation
(provided by developer)
Library
Java Proxy Class
(generated by AIDL)
Java Stub Class
(generated by AIDL)
Framework
android.os.IBinder:transact()
android.os.Binder:onTransact()
android.os.BinderProxy:transactNative()
android.os.Parcel
android.os.Binder:execTransact()
Upcall to process VM
Kernel
/dev/binder Kernel Module

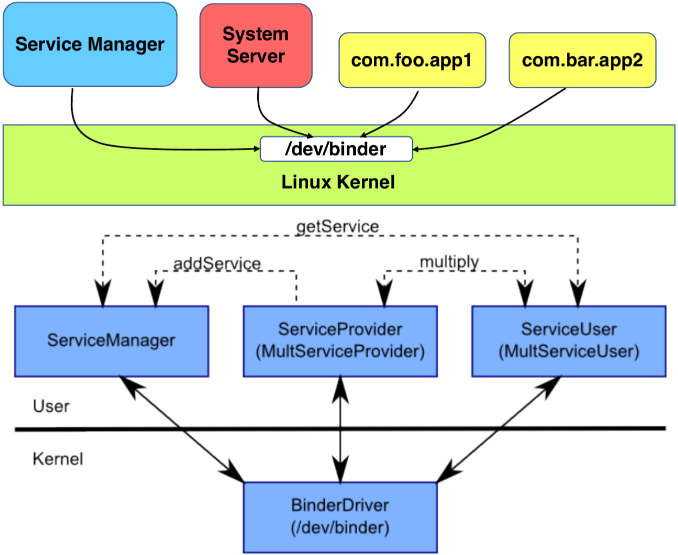

write_buffer
BC Protocol
struct binder_transaction_data
target
code
data
BC_TRANSACTION
handle
START_SERVICE_TRANSACTION
flat_binder_object


read_buffer
BR Protocol
struct binder_transaction_data
target
code
data
BR_TRANSACTION
handle
START_SERVICE_TRANSACTION
flat_binder_object

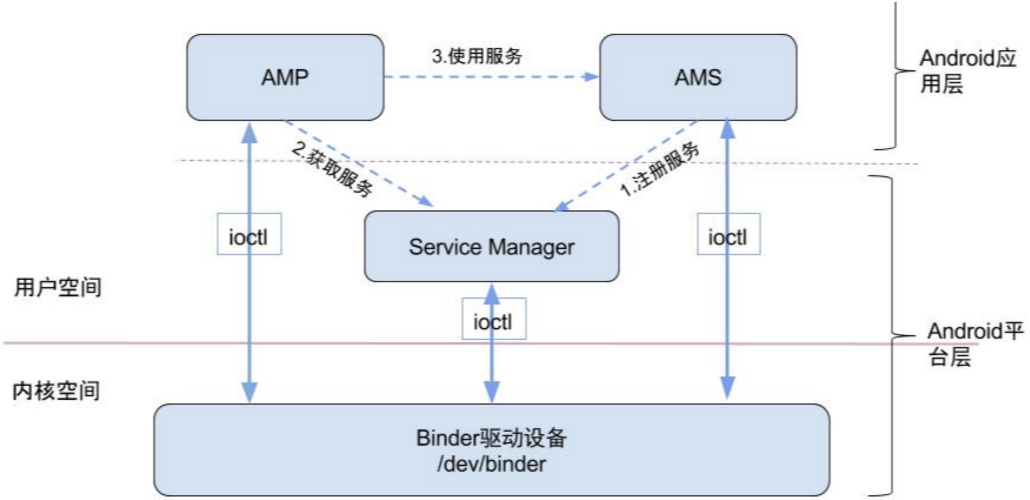
Binder 机制


AMP.startService
Java 应用层
AMS.startService
BinderProxy
Java IPC层
Binder
BpBinder
Native IPC层
BBinder
Binder Driver
Linux 内核层

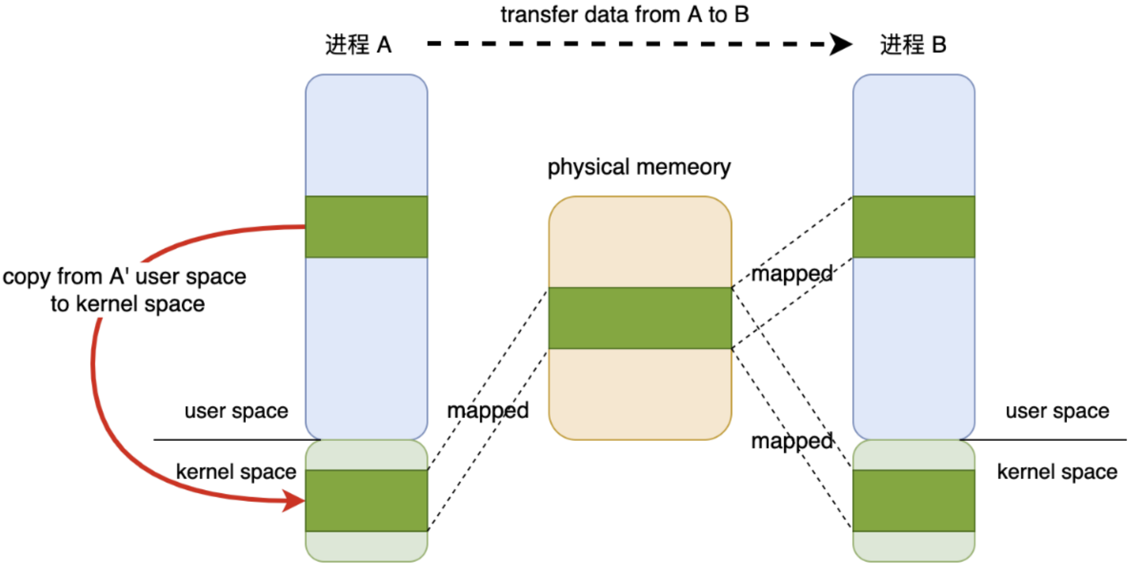
Binder 机制


Service Manager
System Server
com.foo.app1
com.bar.app2
/dev/binder
Linux Kernel
getService
addService
multiply
ServiceManager
ServiceProvider
(MultServiceProvider)
ServiceUser
(MultServiceUser)
User
Kernel
BinderDriver
(/dev/binder)

Binder 机制


AMP
3.使用服务
AMS
Android应用层
2.获取服务
1.注册服务
ioctl
Service Manager
ioctl
用户空间
ioctl
Android平台层
内核空间
Binder驱动设备
/dev/binder

Binder 机制 – 一次拷贝



参考

- Android IPC Mechanism，Jim Huang (黃敬群), 2012
- Lec20: CS 318 Principles of Operating Systems, Ryan Huang, 2018
- Deep Dive into Android IPC/Binder Framework at Android Builders Summit, Aleksandar Gargenta, 2013
- 彻底理解 Android Binder 通信架构, Gityuan, 2016