ANDROID ISPOD HAUBE

RT-RK
COMPUTER BASED SYSTEMS

Android sistemski koncepti

RT-RK 1. Avgust – 12. Avgust 2016.

Agenda

- Init proces
- Android svojstva
- Binder
- JNI
- Ashmem
- Ograničenja

Agenda

- Init proces
- Android svojstva
- Binder
- JNI
- Ashmem
- Ograničenja

Init proces (1/4)

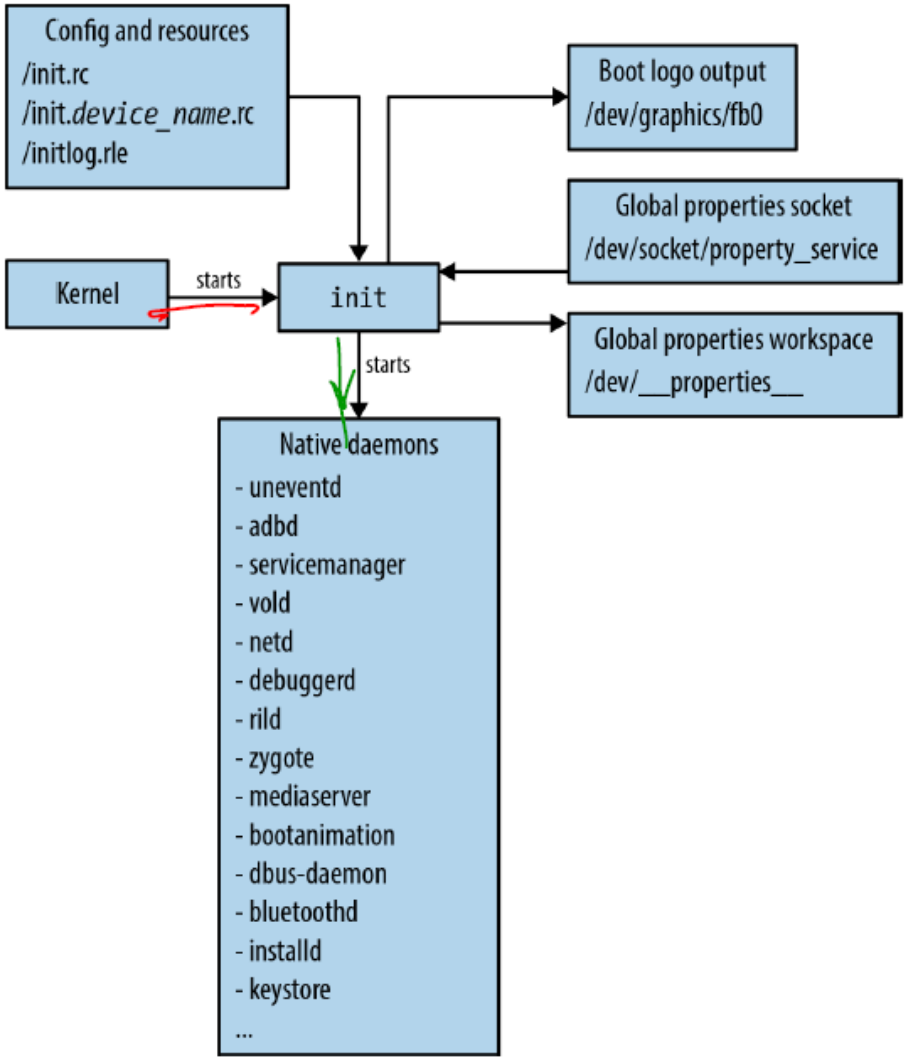
- Šta je init proces?
  - Init proces je progam koji pokreće Android sistem.
  - Pokreće se nakon uspešnog pokretanja Linux kernel-a
  - Prvi Android program. PID = 1
- Gde se nalazi?
  - Na razvojnoj platformi: ./system/core/init
  - Na uređaju: /init

Init proces (2/4)

```
[/mnt/hdd-02/jb]
vranic@gtv03-lin-$ $ adb shell ps
USER      PID   PPID  VSIZE  RSS     WCHAN    PC         NAME
root      1     0     8864   724   ffffffff 00000000 S /init
root      67    1     8860   588   ffffffff 00000000 S /sbin/ueventd
root      94    1     9324   712   ffffffff 00000000 S /system/bin/sh
logd      101   1     19280  1920  ffffffff 00000000 S /system/bin/logd
root      102   1     9832   288   ffffffff 00000000 S /sbin/healthd
root      103   1     10628  1252  ffffffff 00000000 S /system/bin/lmkd
system    104   1     9460   696   ffffffff 00000000 S /system/bin/servicemanager
root      105   1     18112  1856  ffffffff 00000000 S /system/bin/vold
system    106   1     167540 11192 ffffffff 00000000 t /system/bin/surfaceflinger
root      123   1     22836  1356  ffffffff 00000000 S /system/bin/netd
keystore  129   1     31988  2940  ffffffff 00000000 S /system/bin/keystore
root      152   1     966132 42988 ffffffff 00000000 S zygote
media_rw  153   1     15408  400   ffffffff 00000000 S /system/bin/sdcard
shell     155   1     16984  220   ffffffff 00000000 S /sbin/adbd
system    551   152   1108444 88312 ffffffff 00000000 S system_server
u0_a5     649   152   945452 29568 ffffffff 00000000 S android.process.media
u0_a18    686   152   998828 41324 ffffffff 00000000 S com.android.systemui
media_rw  785   1     15408  400   ffffffff 00000000 S /system/bin/sdcard
u0_a4     883   152   944296 24300 ffffffff 00000000 S com.android.defcontainer
u0_a11    939   152   1050432 37724 ffffffff 00000000 S com.google.android.katniss:interactor
dhcp      951   1     9352   664   ffffffff 00000000 S /system/bin/dhcpcd
u0_a12    1005  152   946116 28896 ffffffff 00000000 S com.google.android.leanback.ime
```

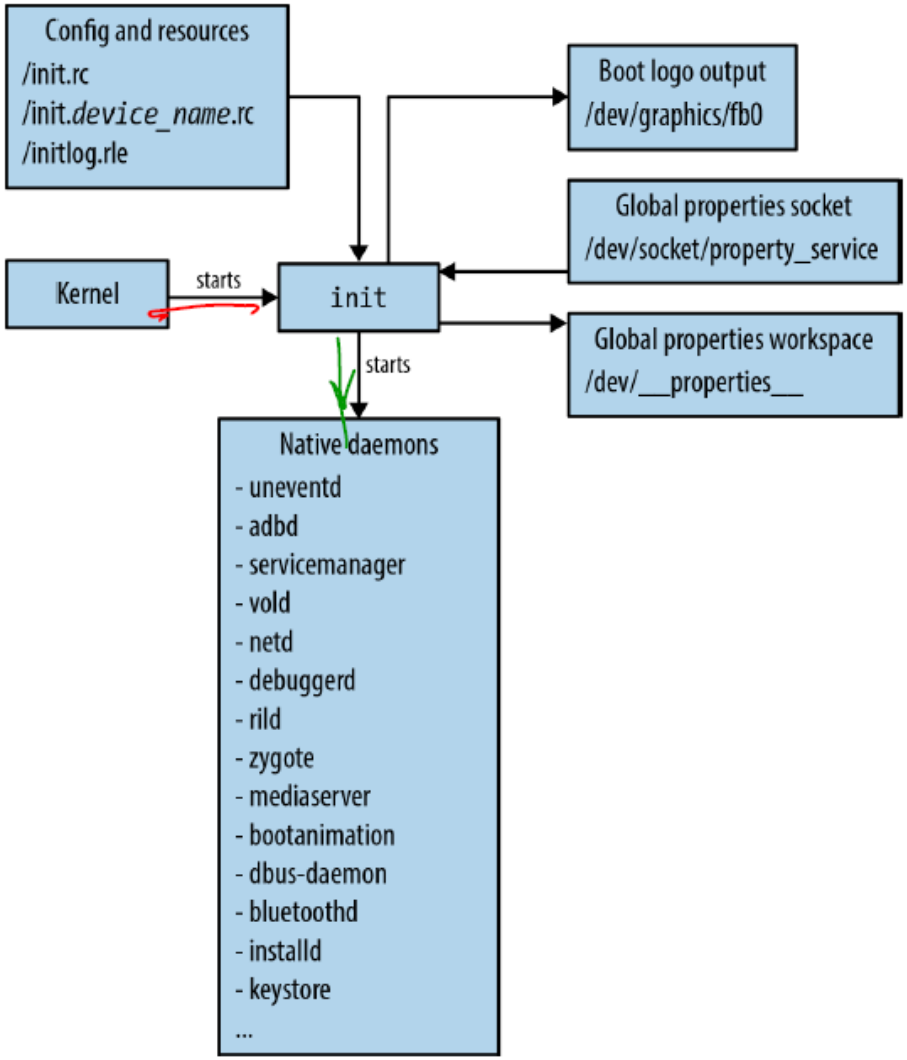
Init proces (3/4)

- Kako funkcioniše? (1/2)
  - Izvršava *.rc dateke: /init.rc i init.<device_name>.rc
    - Napisane u specifičnom formatu koji se naziva Android init jezik
    - Na lokalnom računaru init.rc datoteka se nalaze: ./system/core/rootdir
    - Na lokalnom računaru: ./system/core/rootdir/etc/init.goldfish.rc, ./device/htc/passion/init.mahimahi.rc, ./device/samsung/crespo/init.herring.rc
  - Pokreće system_property program i podešava okruženje za rad sa Android svojstvima
  - Pokreće boot animaciju
- Posledice izvršavanje *.rc datoteka
  - Pokreću se Android native servisi sa fork komandom
    - Podešava prava sa setgui i setuid
  - Učitavaju se particije
  - Prave se ključni deirektorijumi na uređaju: /dev, /proc i /sys



Init proces (4/4)

- Kako funkcioniše? (2/2)
  - Nakon pokretanja, u beskonačnog petlji:
    - Ponovo pokreće servise koji se sruše
    - Prati property utičnicu za svaki ulaz koji treba da bude obrađen
    - Prihvata key event-e i pokreće servise
- Za više informacija o init procesu i Android init jeziku pogledati: /system/core/init/readme.txt



Sintaksa Android init jezika

- Akcije/okidači, komande:

```
on <trigger>
    <command>
    <command>
    ...
```

- Servisi:

```
service <name> <pathname> [<arguments>]
    <option>
    <option>
    ...
```

- Ključna reč service u init jeziku nema nikakve veze sa sistemskim servisima niti sa servisima koje prave app developeri
- Ograničenje: naziv servisa ne sme biti duži od 16 karaktera
- Akcije i servisi moraju da imaju jedinstvene nazive

Okidači

- Okidači su uslovi koji kada se ispune manifestuju izvršavanje liste komandi
- Vrste okidača:
  - Predefinisani okidači
    - `early-init`
    - `init`
    - `early-fs`
    - `fs`
    - `post-fs`
    - `post-fs-data`
    - `early-boot`
    - `boot`
    - `charger`
  - Okidači koji se aktiviraju na promenu vrednosti property-a
    - `on property:<name>=<value>`

Predefinisani okidači: detaljnije

Okidač	Opis
`early-init`	Prva faza inicijalizacije. Koristi se za SELinux i Out Of Memory podešavanja.
`init`	Pravljenje file sistema, mount tačaka i upis u kernel promenjljive.
`early-fs`	Pokreće se pre nego što file system postane spreman za učitavanje.
`fs`	Učitavanje particija.
`post-fs`	Komande koje se izvršavaju nakon što je file system učitan.
`post-fs-data`	Komande koje se uvršavaja kada je data particija dekriptovana.
`early-boot`	Komande koje se izvršavaju nakon što je property service pokrenut.
`boot`	Normalno boot komande.
`charger`	Komande koje se izvršavaju kada je uređaj priključen na punjač.

Primer okidača

```
on property:persist.service.adb.enable=1
    start adbd

on property:persist.service.adb.enable=0
    stop adbd
```

Komande (1/2)

Komanda	Opis
`start <service>`	Pokretanje servisa.
`stop <service>`	Zaustavljanje servisa.
`restart <service>`	Ponovno pokretanje servisa.
`setprop <name> <value>`	Postavljanje Andorid property-a na neku vrednost.
`exec <path> [ <argument> ]*`	Izvršavanja programa. Nije preporučnjivo. Koristi init service umesto ove komande. Izbačeno u novijim verzijama.
`export <name> <value>`	Pravljenje i postavljenje varijable na neku vrednost.
`trigger <event>`	Pozivanje okidača.
`setenforce <value>`	Omogućavanje/onemogućavanje SE Linux-a.

Komande (2/2)

Komanda	Opis
`chdir <directory>`	Isto kao i cd komanda.
`chmod <octal-mode> <path>`	Promena pristupnih dozvola.
`chown <owner> <group> <path>`	Promena vlasništva.
`chroot <directory>`	Promena root direktorijuma.
`copy <path> <destination>`	Kopiranje datoteke.
`mkdir <path> [mode] [owner] [group]`	Pravljenje novog direktorijuma.
`rmdir <path>`	Brisanje datoteke/direktorijuma.
`insmod <path>`	Učitavanje kernel modula.
`mount <type> <device> <dir> [ <mount option> ]*`	Mountovanje uređaja na neki direktorijum.

Opcije

Opcija	Opis
`critical`	Ako se servis sruši pet puta, sistem će biti ponovno pokrenut u recovery režimu
`oneshot`	Pokreni servis samo jednom. Nemoj ga ponovo pokretati nakon rušenja.
`onrestart <command>`	Kada se servis ponovo pokrene izvrši određenu komandu.
`disabled`	Servis neće biti startovan automatski već ručno sa start komandom.
`keycodes <keycode> [ <key code> ]*`	Pokretanje servisa sa određenom kombinacijom ključeva.
`class <name>`	Klasa kojoj pripada servis.
`group <groupname> [ <group name> ]*`	Grupa kojoj pripada servis.
`user <username>`	Korisnik kome pripada servis.

Primer servisa

```
# bugreport is triggered by holding down volume down,
volume up and power
service bugreport /system/bin/dumpstate -d -v -o
/sdcard/bugreports/bugreport
    disabled
    oneshot
    keycodes 114 115 116
```

Primer servisa (1/2)

```
#define LOG_TAG "ExampleService"
#include <utils/log.h>

int main(int argc, char **argv) {
    LOGI("Service started");
    int elapsed = 0;
    while(1) {
        sleep(3);
        elapsed += 3;
        ALOGI("Service elasped time is: %d", elapsed);
    }
}
```

Primer servisa (2/2)

```
on boot
    start exampleservice

service exampleservice /system/bin/exampleservice
    user exampleservice
    group exampleservice
    oneshot
    disabled
```

Agenda

- Init proces
- Android svojstva
- Binder
- JNI
- Ashmem
- Ograničenja

Android svojstva: pregled

- Šta su Andorid svojstva?
  - Jednostavan način za deljenje relativno stabilnih vrednosti globalno kroz sve slojeve Android-a
  - Pandam Windows Registery
- Glavna komponenta je property service
- Način za pristup property service-u
  - /dev/socket/propery_service
    - Servis za promenu vrednosti svojstva
  - /dev/__properties__
    - Read only datoteka – property “baza podataka”.
- Na koji način se čuvaju svojstva?
  - U vidu binarnog stabla
- Gde se nalazi kod?
  - Na lokalnom računaru: system/core/init/property_service

Android svojstva: pregled

- Zašto je `/dev/__properties__` read only?
  - Init proces napravi `/dev/__properties__` datoteka u read/write režimu
  - Memorijski mapira datoteku u SVOJ adresni prostor
  - Zatvori deskriptor datoteke
  - Otvori datoteku u read only režimu

- Posledice
  - Svojstva se čuvaju u RAM memoriji
  - Init proces može da menja vrednosti svojstva
  - Svi ostali procesi mogu da vrše čitanje, dok pisanje mogu da vrše kroz property_service
  - Na taj način se vrši kontrola pisanja

Android svojstva: pregled

- Odakle se učitavaju svojstva na uređaju?
  - /default.prop
  - /system/build.prop
  - /system/default.prop
  - /vendor/build.prop
  - /data/local.prop
  - /factory/factory.prop (izbačeno u novijim verzijama)
- Kontrola upisa
  - Do verzije Jelly Bean (uključujući i ovu verziju)
    - `property_perms` struktura
    - `system/core/init/property_service.c`
  - Novije verzije
    - Selinux
- Šta će se desiti ako proces nema prava menjanja vrednosti svojstva?

Android svojstva: pregled

```
/*
 * Properties are stored in a hybrid trie/binary tree structure.
 * Each property's name is delimited at '.' characters, and the tokens are put
 * into a trie structure.  Siblings at each level of the trie are stored in a
 * binary tree.  For instance, "ro.secure"="1" could be stored as follows:
 *
 * +-----+   children    +----+   children    +--------+
 * |     |-------------->| ro |-------------->| secure |
 * +-----+               +----+               +--------+
 *                       /    \                 /   |
 *                 left /      \ right   left /    |  prop   +===========+
 *                     v        v             v     +-------->| ro.secure |
 *                  +-----+   +-----+     +-----+             +-----------+
 *                  | net |   | sys |     | com |             |     1     |
 *                  +-----+   +-----+     +-----+             +===========+
```

Android svojstva: primer na starijim verzijama

```
/* White list of permissions for setting property services. */
struct {
        const char *prefix;
        unsigned int uid;
        unsigned int gid;
} property_perms[] = {
        { "net.rmnet0.", AID_RADIO, 0 },
        { "net.gprs.", AID_RADIO, 0 },
        { "net.ppp", AID_RADIO, 0 },
        { "ril.", AID_RADIO, 0 },
        { "gsm.", AID_RADIO, 0 },
        { "persist.radio", AID_RADIO, 0 },
...
        { "service.adb.root", AID_SHELL, 0 },
        { "persist.sys.", AID_SYSTEM, 0 },
        { "persist.service.", AID_SYSTEM, 0 },
        { "persist.security.", AID_SYSTEM, 0 },
        { NULL, 0, 0 }
};
```

Android svojstva: SELinux primer

- Korak 1: definicija domena
  - property.te

```
type pmf_prop, property_type;
```

- Korak 2: definicija svojstva pmf.*
  - property_contexts

```
pmf.                    u:object_r:pmf_prop:s0
```

- Korak 3: omogućavanje procesu da pristupi svojstvu
  - pmfprocess.te

```
allow pmfprocess pmf_prop:property_service set;
```

Android svojstva: specifična svojstva

Prefix	Opis
`ro.*`	Svojstva koja se mogu samo čistati, tj mogu se upisati samo jednom i čitati proizvoljan broj puta. Primer: `ro.hardware` i `ro.build.id`.
`persist.*`	Svojstva čija vrednost će biti sačuvana i nakon ponovnog pokretanja uređaja. Putanja za čuvanje je: /data/property/*. Primer: `cat /data/property/persist.sys.timezone Europe/Belgrade`
`ctl.*`	Svrha ctl svojstava je da pokrenu ili zaustave servise koji su vezani na njih. Primer, surfaceflinger: `property_set("ctl.start", "bootanim");`
`net.change.*`	Poslednja promena nad svojstvom iz net.* paketa.

Android svojstva: nova svojstva

- Build time
  - U product-u:
    - `PRODUCT_PROPERTY_OVERRIDES += NAME=VALUE`
    - `PRODUCT_PROPERTY_OVERRIDES += test.property=some_value`
    - `make`
    - Izmene završe u system/build.prop na razvojnom uređaju
  - init*.*.rc
    - `setprop NAME VALUE`
    - `make`
  - U system.prop: vendor/vendor_name/product_name
    - Izmene završe u system/build.prop na razvojnom uređaju
    - `make`
- Runtime
  - U build.prop: system/build.prop
    - Promena direktno na razvojnom uređaju
    - `adb shell chmod 644 system/build.prop`
  - U default.prop: data/local.prop
    - Promena direktno na razvojnom uređaju

Android svojstva: shell upotreba

- Preko adb shell-a:
  - `adb shell setprop NAME VALUE`
  - `adb shell getprop`
  - `adb shell getprop NAME`

  - `adb shell setprop status.battery.level 6`
  - `adb shell getprop status.battery.level`
  - `adb shell getprop | grep level`

Android svojstva: native kod upotreba

```
#include "cutils/properties.h"
LOCAL_SHARED_LIBRARIES += libcutils
```

```
int property_set(const char *key, const char *value);
```

- key: naziv property-a
- value: vrednost property-a
- Povratna vrednost: 0 za uspešno izvršenu operaciju , inače <0

```
int property_get(const char *key, char *value, const
char *default_value);
```

- key: naziv property-a
- value: vrednost property-a
- default_value: podrazumevana vrednost, u slučaju da property key ne postoji
- Povratna vrednost: dužina property-a

Android svojstva: Java kod upotreba

- frameworks\base\core\java\android\os\SystemProperties.java
- Sve funkcije su statične, tako da se mogu pozivati bez prevljenja konstruktora klase SystemProperties

Funkcija	Opis
`String get(String key)`	Dobavljanje tekstualno svojstva po ključu.
`String get(String key, String def)`	Dobvljanje tekstualnog svojstva po ključu. Ako svojstvo ne postoji, def vrednost će biti vraćena.
`int getInt(String key, int def)`	Dobavljanje celobrojnog svojstva po ključu. Ako svojstvo ne postoji, def vrednost će biti vraćena.
`boolean getBoolean(String key, boolean def)`	Dobavljanje boolean svojstva po ključu. Ako svojstvo ne postoji, def vrednost će biti vraćena.
`void set(String key, String val)`	Postavljanje svojstva sa ključem key na vrednost val.
`void addChangeCallback(Runnable callback)`	Registrovanje callback funkcije koja će biti pozvana ukoliko se promeni vrednost svojstva.

Agenda

- Init proces
- Android svojstva
- Binder
- JNI
- Ashmem
- Ograničenja

Android Binder: uvod

- Načini za razmenu poruka između 2 ili više procesa
  - Datoteke (uključujući i memorijski mapirane)
  - Signali
  - Utičnice
  - Semafori
  - Deljena memorija
  - Pipes
  - Intents, ContentProviders,
  - Binder

Android Binder: istorija

- Potiče od OpenBinder projekta:
  - Započet u Be. Inc. kao osnovni deo njihovog OS (~2001)
  - Preuzet od strane PalmSource
  - Prva implementacija u okviru Palm Cobalt OS
  - Palm prelazi na Linux, tako da je Binder portovan na Linux, i načinjen open-source (~2005)
  - Google preuzima razvojni tim OpenBinder i pridružuje ih Android timu
  - Isprva je korišćena OpenBinder verzija, a kasnije je kompletno napisan od početka, ~2008

Android Binder: šta?

- Šta je IPC?
  - Inter Process Communication: okruženje koje se koristi za razmenu podataka između procesa
- Šta je Binder?
  - IPC implementacija za Android
- Koristi se za:
  - Prosleđivanje poruka
  - Sinhronizaciju
  - Deljenje memorije
  - Udaljene funkcijske pozive (RPC – Remote Procedure Calls)
- Omogućuje sledeće koncepte:
  - Deljenje informacija
  - Modularnost
  - Bolju raspodelu privilegija
  - Izolaciju podataka
  - Ubrzanje performansi
  - Lakoću upotrebe
  - Stabilnost
- Predstavlja suštinsku komponentu Android platforme

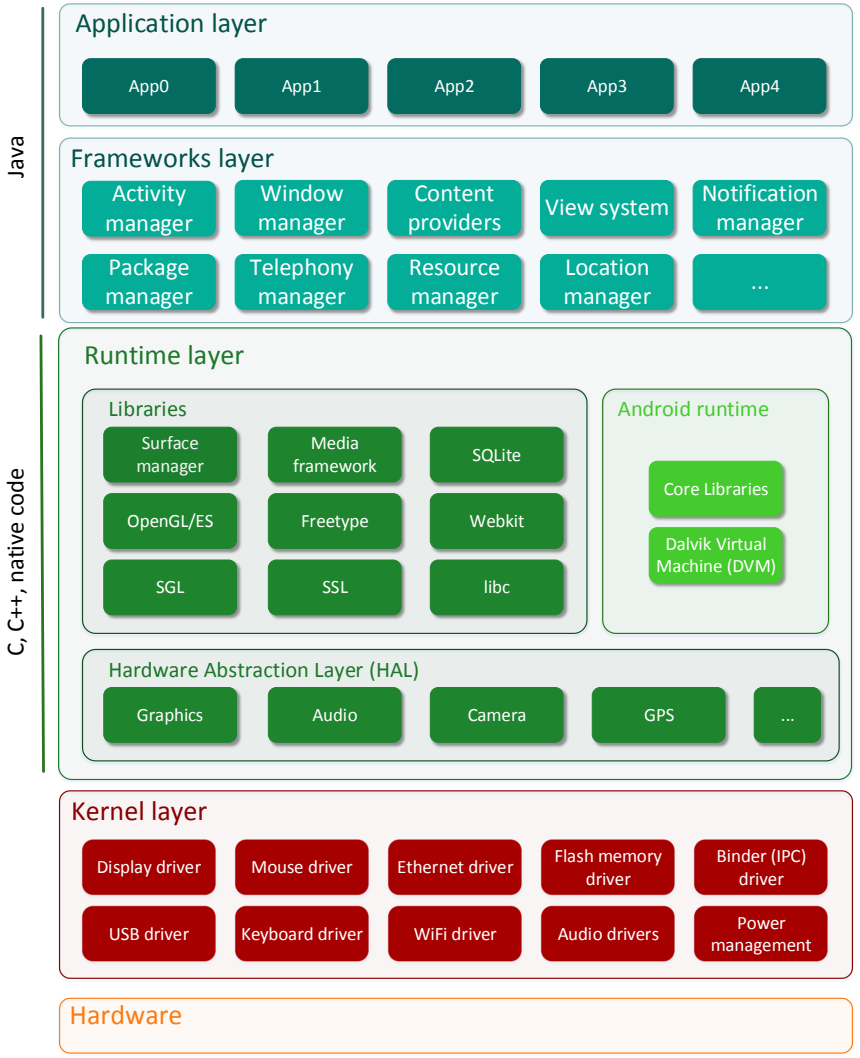
Android Binder: zašto?

- Android aplikacije i sistemski servisi se izvršavaju u **odvojenim** procesima, zbog:
  - Sigurnosti: Svaki proces se izvršava u svom odeljku (engl. *sandbox*) i sa jedinstvenim identitetom
  - Stabilnosti: Ako jedan proces doživi iznenadan prestanak rada, to ne utiče na druge procese u sistemu
  - Boljeg memorijskog upravljanja: „Nepotrebni“ procesi se uklanjaju da bi se oslobodili resursi (najviše memorija) za nove procese

- Ali i dalje moraju da komuniciraju međusobno i dele podatke!

Android Binder: zašto?

- Kako komunicirati između Java aplikacija?
- Kako komunicirati između izvršnih programa-servisa?
- Kako kominicirati između Java aplikacija i izvršnih program-servisa?
- Rešenje?
- Binder

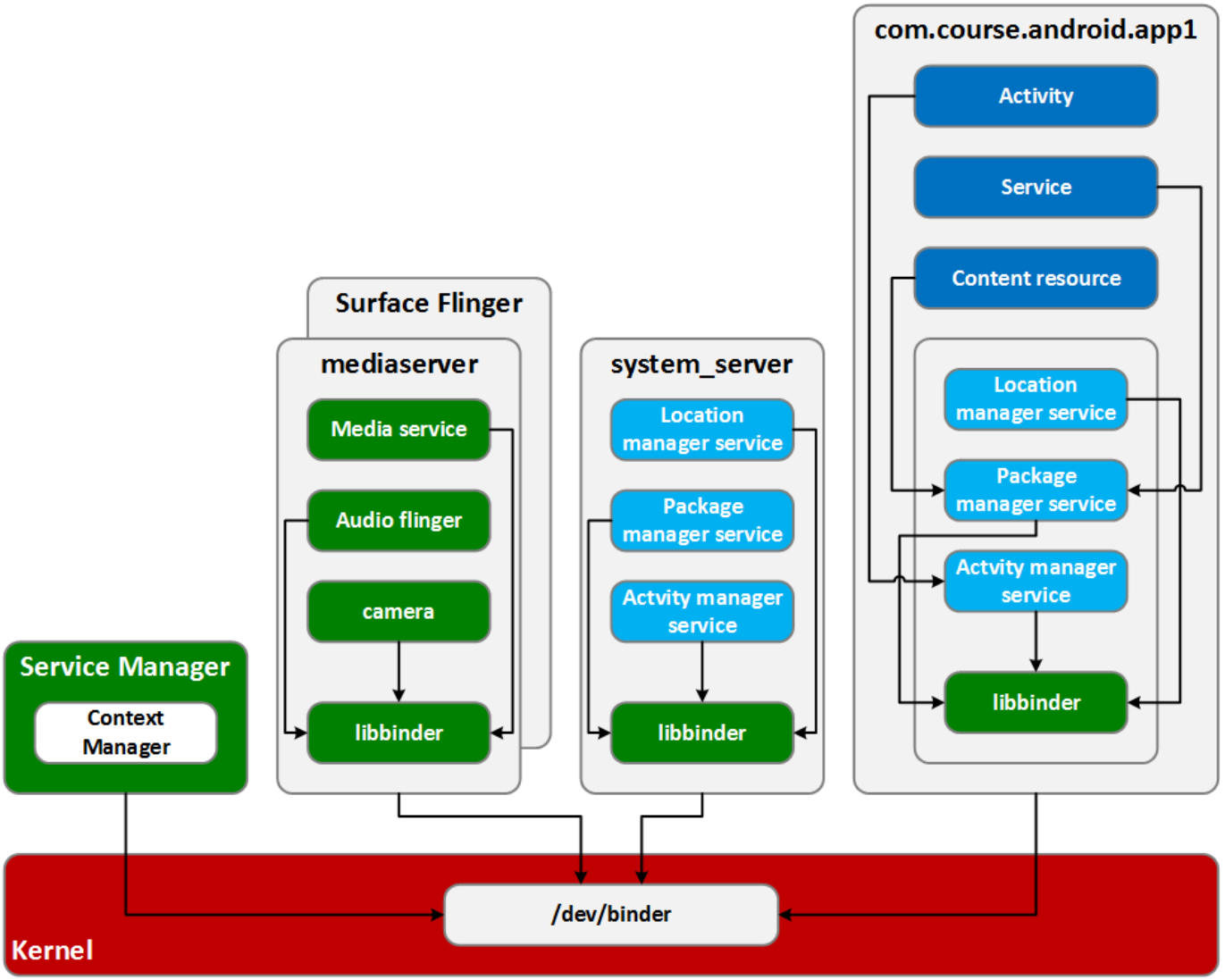


Android Binder: zašto?

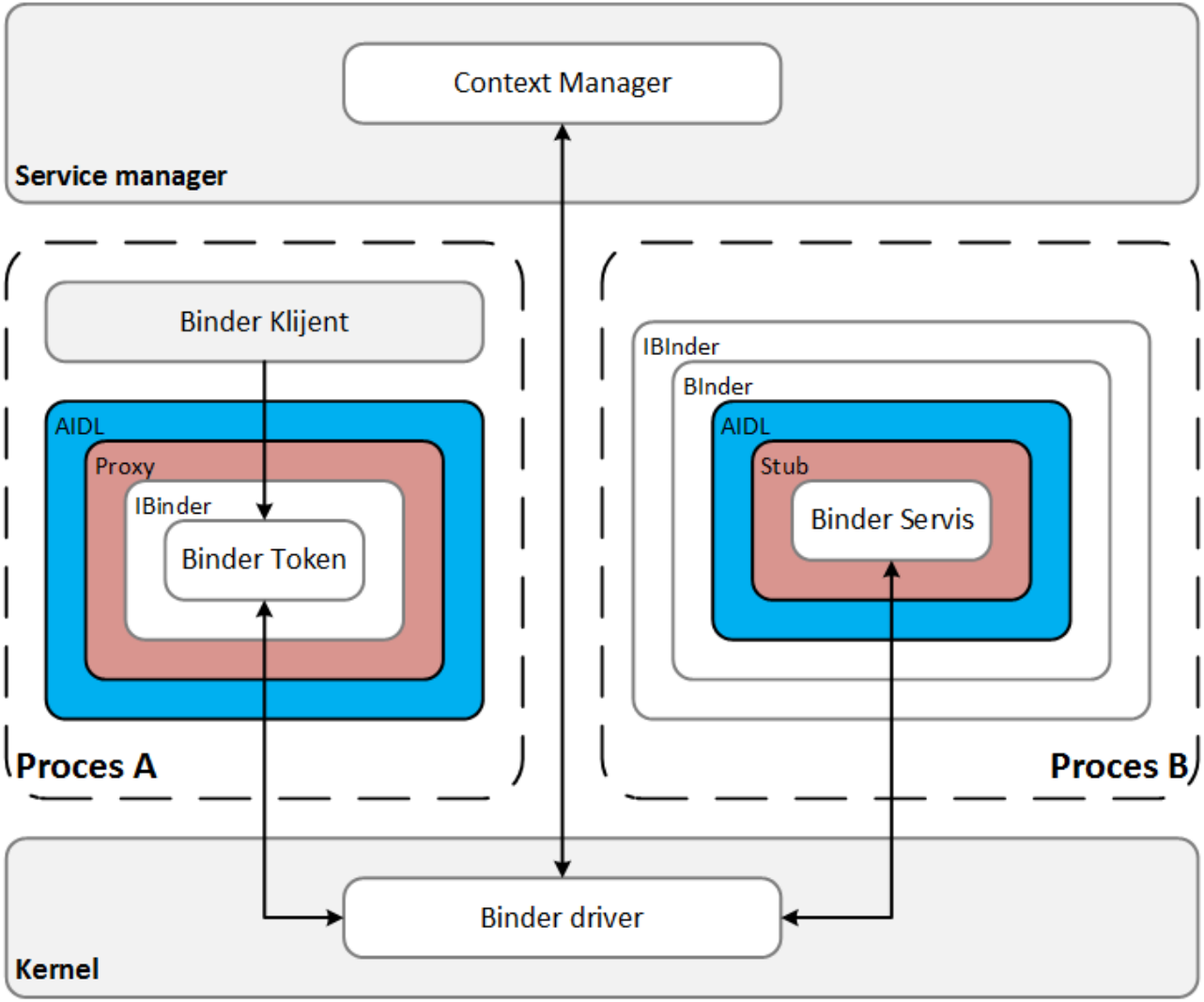
- Prednosti:
  - Serijalizacija i de-serijalizacija većine standardnih tipova (engl. *marshalling*) – skraćeno „prosleđivanje“ objekata
  - Mod lokalnog izvršavanja (bez IPC/prosleđivanja podataka) ukoliko su klijent i servis u okviru istog procesa

- Mane:
  - Klijent-server komunikacija bazirana na razmeni poruka – nije dobro prilagođen za razmenu tokova podataka (engl. *streaming*)
  - Nije deo (definisan od strane) nekog standarda – npr. Posix

Android Binder: kako?

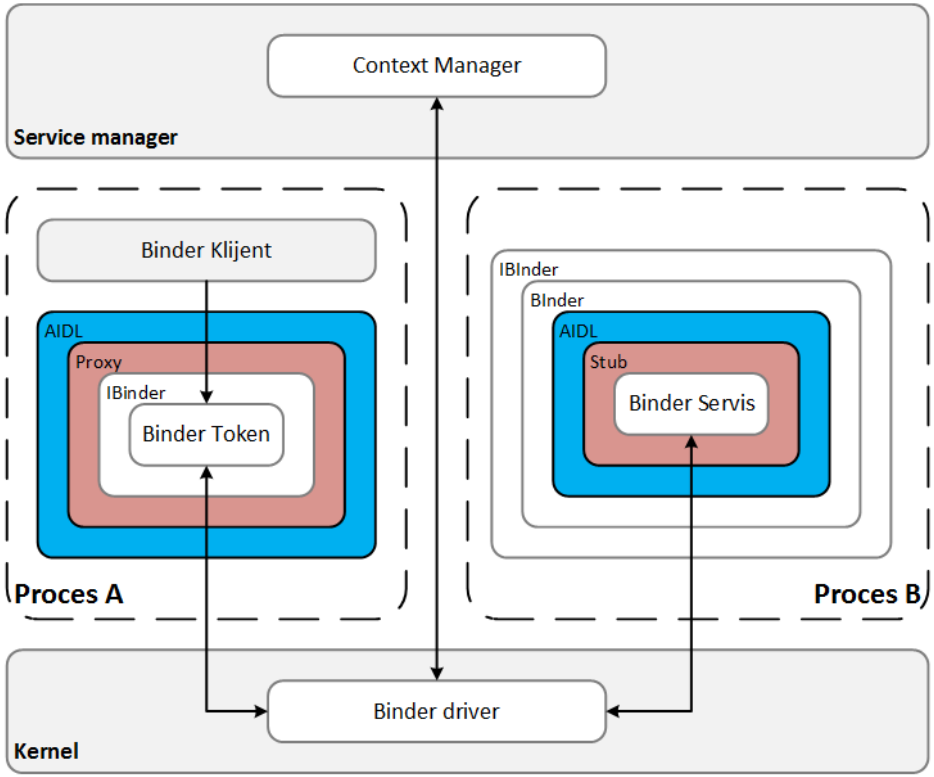


Binder: terminologija (1/4)



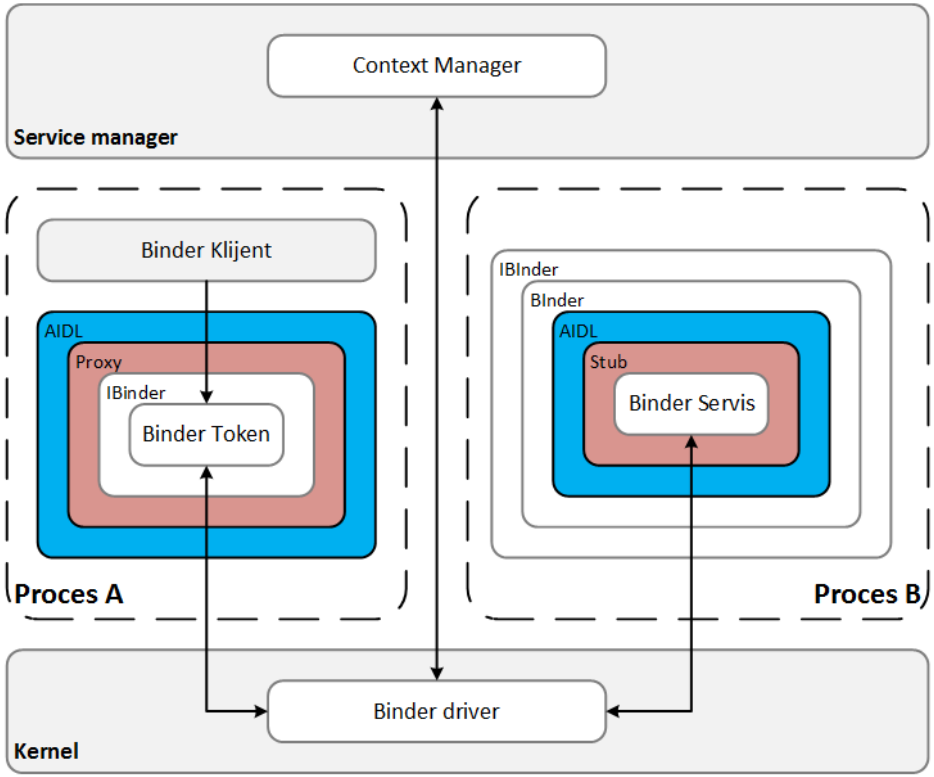
Binder: terminologija (2/4)

Pojam	Opis
Binder (okruženje)	Sveukupna IPC arhitektura
Binder Driver	Uslužilac na nivou Linux kernela koji opslužuje komunikaciju između procesa
Binder Protocol	Protokol niskog nivoa (ioctl-baziran) koji se koristi za pristup Binder uslužiocu
IBinderInterface	Metode koje Binder objektni moraju da implementiraju
AIDL	Jezik za opis operacijama IBinder sprezi.
Binder (Object)	Generička implementacija IBinder sprege
Binder Token	Apstraktna 32-bitna celobrojna vrednost koja na jedinstveni način identifikuje Binder objekat u sistemu (u okviru svih procesa)
Binder Service	Implementacija Binder (objekta) koji implementira operaciju
Binder Client	Objekat koji želi da koristi ponašanje ponuđeno od strane Binder servisa



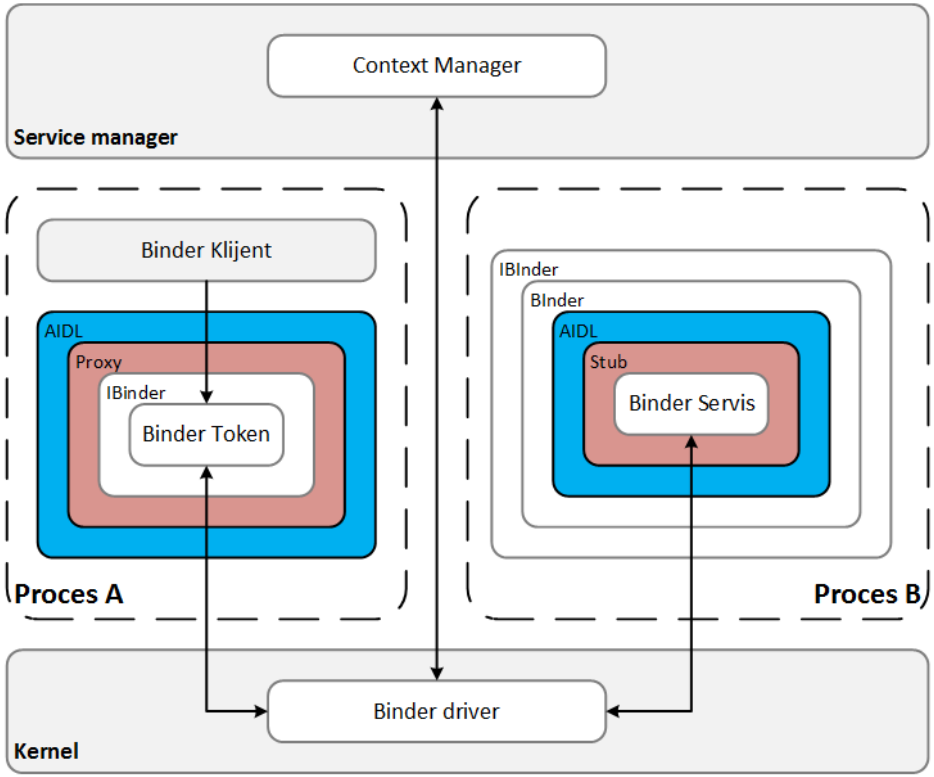
Binder: terminologija (3/4)

Pojam	Opis
Binder Transaction	Akt poziva operacije (metode) na udaljenom Binder objektu, koji može da uključuje slanje/prijem podataka, putem Binder protokola
Parcel	Kontejner za poruke (podaci ili reference na objekte) koje se mogu poslati putem IBinder sprege
Marshalling	Serijalizacija podataka (proces pretvaranja struktura „visokog“ nivoa u parcele) sa ciljem ugrađivanja u Binder transakciju
Unmarshalling	Obrnut proces od Marshalling-a



Binder: terminologija (4/4)

Pojam	Opis
Proxy	Implementacija AIDL sprege koja de/serijalizuje podatke i mapira pozive metoda sa transakcijama koje su podnete putem IBinder reference na Binder objekat. **Bp** implementacija u native realizaciji.
Stub	Delimična implementacija AIDL sprege koja mapira transakcije na Binder servis metode, dok de/serijalizuje podatke **Bn** implementacija u native realizaciji.
Context Manager (iliti `servicemanager`)	Poseban Binder objekat, sa poznatim Handle objektom (registrovan sa handle 0), koji se koristi za registraciju/pretragu servisa za druge Binder objekte (naziv -> handle mapiranje)

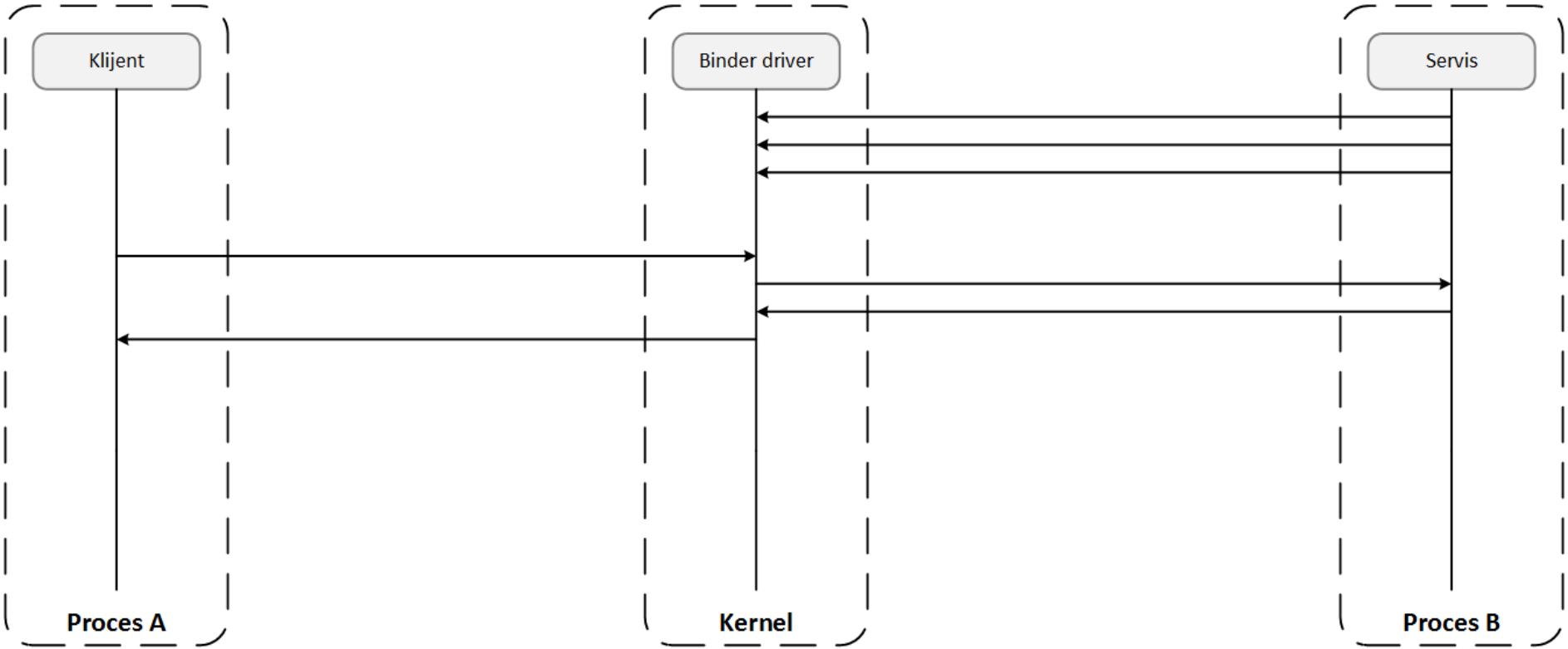


Binder: kako?

- Što se klijenta tiče, on jednostavno želi da „koristi“ servis:

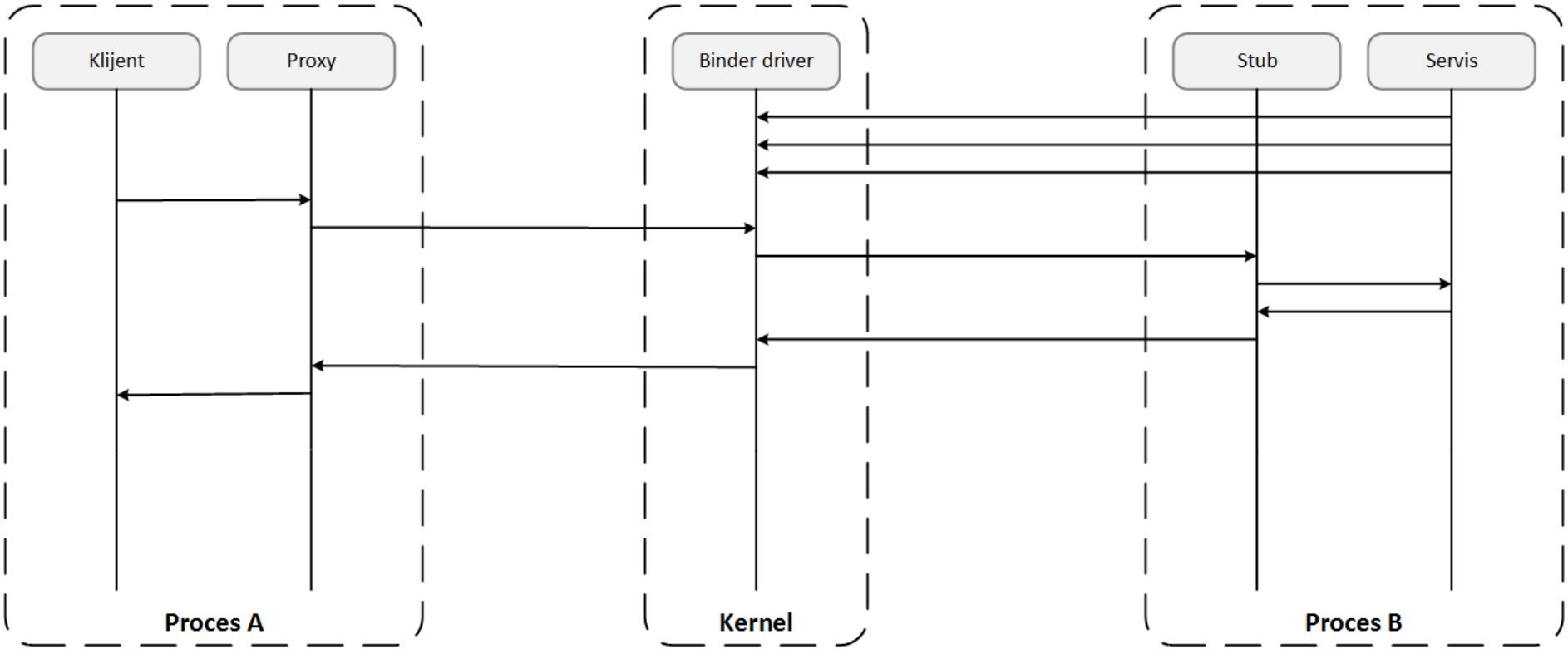
Binder: kako?

- Procesi ne mogu međusobno da direktno pozivaju metode (ili pišu/čitaju podatke). Ali kernel to može, stoga oni koriste Binder uslužilac:



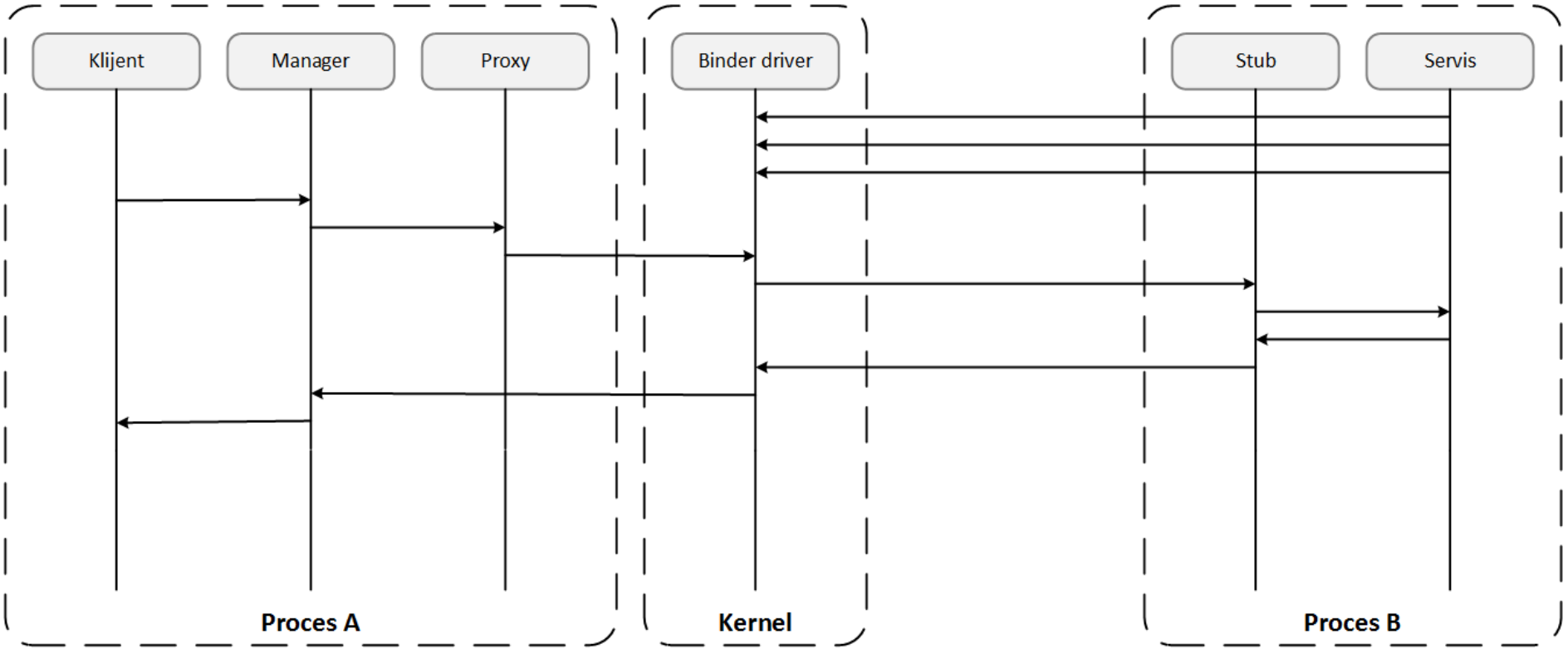
Binder: kako?

- Da bi se Binder protokol što vise sakrio od klijenata i servera, oni koriste Proxy i Stub mehanizme:



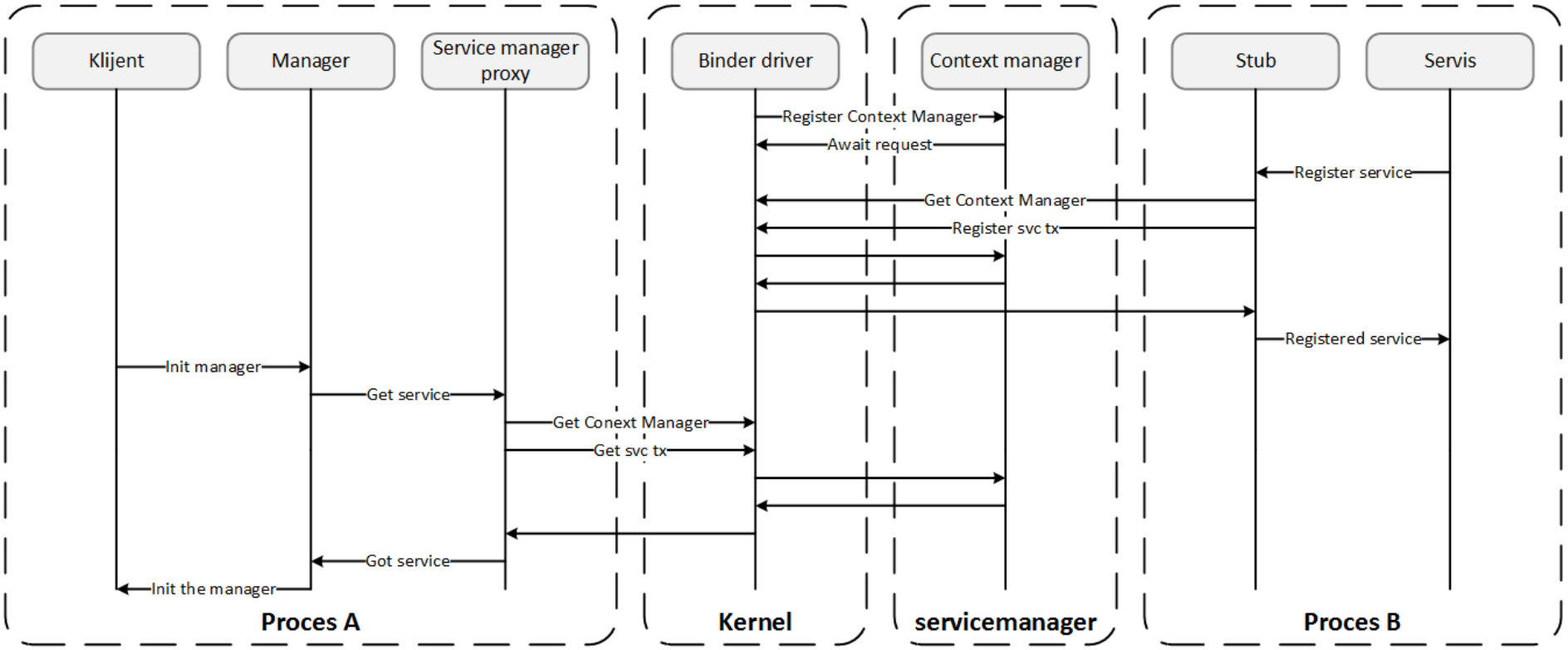
Binder: kako?

- Proxy i Stub delovi za Java baziranu IPC komunikaciju se mogu automatski generisati pomoću `aidl` alata. Većina klijenata ni ne želi da zna da koriste IPC, a posebno Binder, stoga se oslanjaju na `Manager` objekte koji apstrakuju tu kompleksnost od njih. Ovo je posebno tačno za sistemske servise (koji obznanjuju samo delove svog API ka klijentima kroz `Manager` objekte)



Binder: kako?

- Kako klijent da dozna Handle objekat od servisa sa kojim želi da komunicira?
- Sve što treba da uradi je da pita `servicemanager` (Binder-ov CONTEXT_MGR) i da se nada da se servis registrovao kod istog



Binder: otkrivanje raspoloživih servisa

- Da bi se dobila lista svih (trenutno) pokrenutih i registrovanih servisa kod `servicemanager`, potrebno je uraditi sledeće:

```
[/mnt/hdd-02/jb]
vranic@gtv03-lin-$ $ adb shell service list
Found 71 services:
0 sip: [android.net.sip.ISipService]
1 phone: [com.android.internal.telephony.ITelephony]
…
20 location: [android.location.ILocationManager]
…
55 activity: [android.app.IActivityManager]
56 package: [android.content.pm.IPackageManager]
…
67 SurfaceFlinger: [android.ui.ISurfaceComposer]
68 media.camera: [android.hardware.ICameraService]
69 media.player: [android.media.IMediaPlayerService]
70 media.audio_flinger: [android.media.IAudioFlinger]
```

Binder: AIDL

- Šta?
  - *Android Interface Definition Language* je jezik specifičan za Android kojim se definiše Binder bazirane sprege servisa
- Kako?
  - AIDL prati sintaksu Java sprega i dozvoljava deklaraciju metoda od interesa
  - Svaki Binder bazirani servis je definisan u posebnom `.aidl` fajlu, koji se najčešće naziva po sledećem šablonu: `I`*`Foo`*`Service.aidl`
- `aidl` alat, koje je deo Android SDK, se koristi za generisanje prave Java sprege (zajedno sa Stub-om koji obezbeđuje Androidov `android.os.IBinder`) za svaki `.aidl` fajl
- Gde se nalazi generisani kod?
  - out\target\common\obj\JAVA_LIBRARIES*_intermediates\src\src*
  - out\target\common\obj\JAVA_LIBRARIES\BinderExampleCommon_intermediates\src\src\com\course\android\

Binder: AIDL primer, izvorni kod

```
package com.example.app;

import com.example.app.Bar;

interface IFooService {

    void save(inout Bar bar);

    Bar getById(int id);

    void delete(in Bar bar);
}
```

Binder: AIDL primer, generisan kod (1/3)

```
package com.example.app;
public interface IFooService extends android.os.IInterface
{
    public static abstract class Stub extends android.os.Binder
        implements com.example.app.IFooService {
        …
        public static com.example.app.IFooService asInterface(
            android.os.IBinder obj) {
            …
            return new com.example.app.IFooService.Stub.Proxy(obj);
        }
        …
```

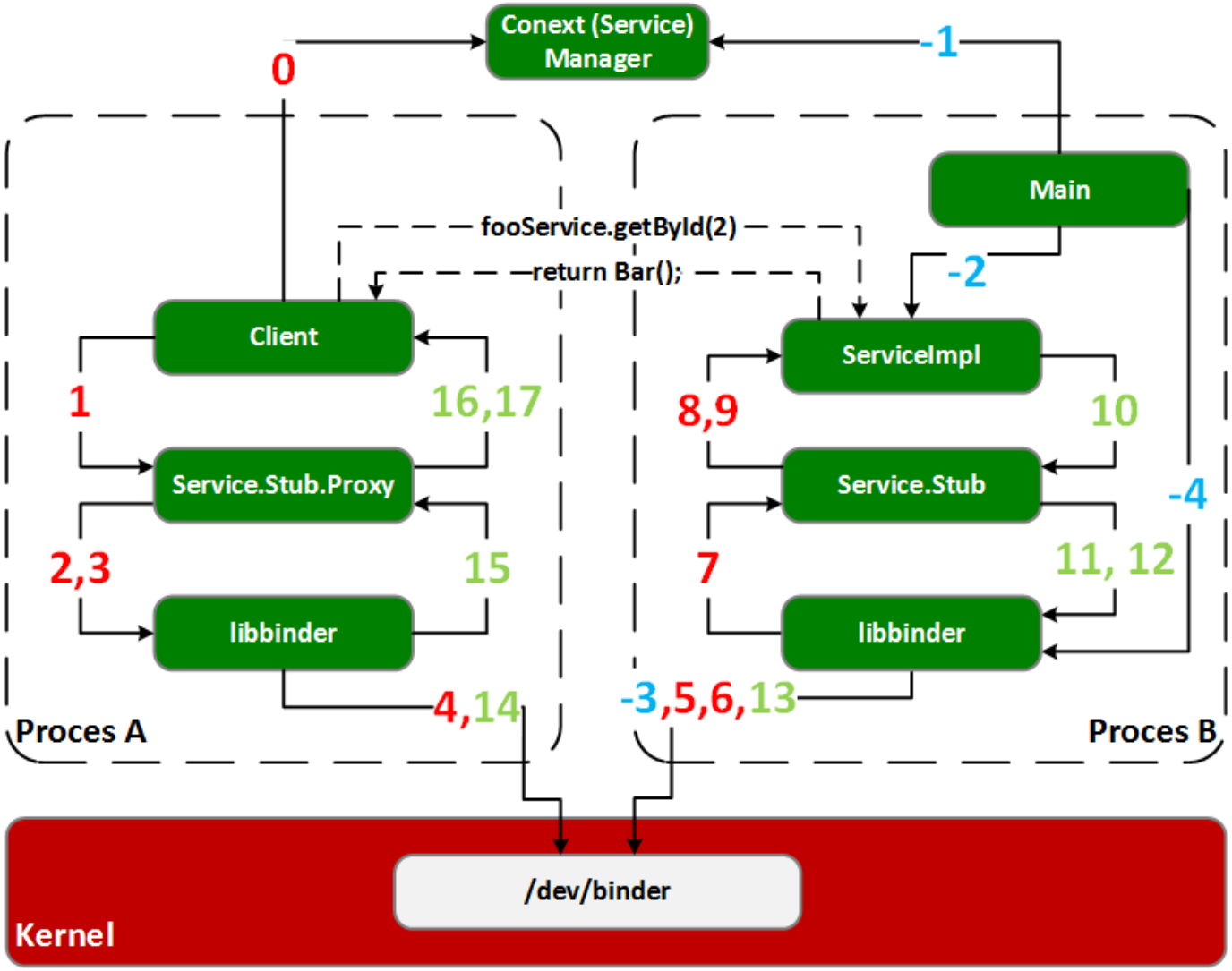
Binder: AIDL primer, generisan kod (2/3)

```
…
        public boolean onTransact(int code, android.os.Parcel data,
android.os.Parcel reply, int flags) throws android.os.RemoteException {
            switch (code) {
                …
                case TRANSACTION_save: {
                    …
                    com.example.app.Bar _arg0;
                    …
                    _arg0 =
com.example.app.Bar.CREATOR.createFromParcel(data);
                    this.save(_arg0);
                    …
                }
                …
            }
            …
        }
```

Binder: AIDL primer, generisan kod (3/3)

```
…
        private static class Proxy implements com.example.app.IFooService {
            private android.os.IBinder mRemote;
            …
            public void save(com.example.app.Bar bar) throws
android.os.RemoteException {
                …
                android.os.Parcel _data = android.os.Parcel.obtain();
                …
                bar.writeToParcel(_data, 0);
                …
                mRemote.transact(Stub.TRANSACTION_save, _data, _reply, 0);
                …
            }
        }
    }
    void save(com.example.app.Bar bar) throws android.os.RemoteException;
    com.example.app.Bar getById(int id) throws android.os.RemoteException;
    void delete(com.example.app.Bar bar) throws android.os.RemoteException;
}
```

Binder: životni vek (1/4)



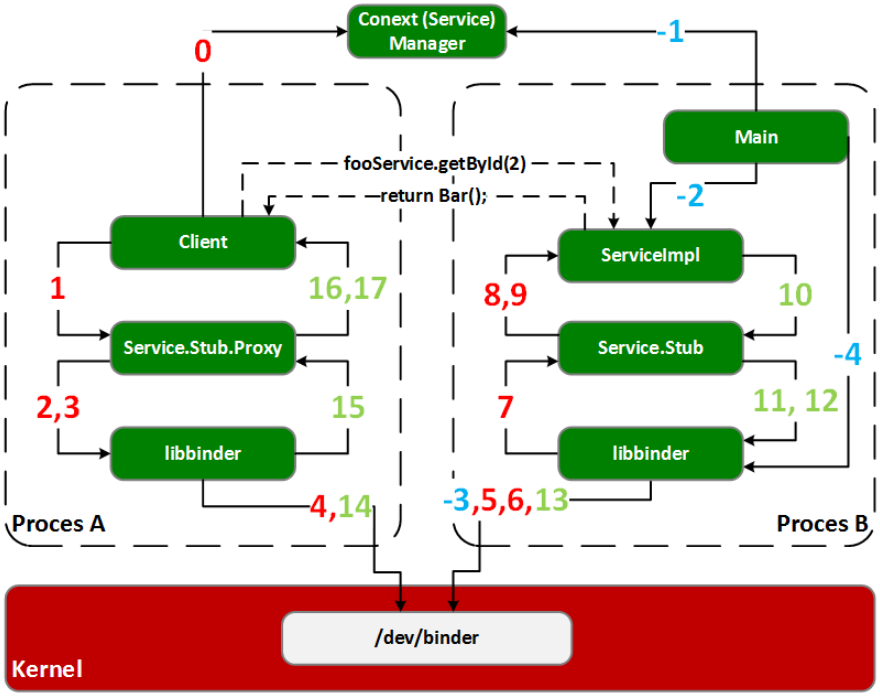
Binder: životni vek (2/4)

-1. registracija servisa

-2. pravljenje servisa

-3. čekanje na binder da odgovori preko blokirajućeg ioctl poziva

-4. pokretanje niti koje će čekati za novi binder zahtev



Binder: životni vek (3/4)

0. dobavljanje servisa

1. pozivanje funkcije definisane preko AIDL-a, npr fooService.getById(id)

2. Serijalizacija id u Parcel

3. transact(fooService, TRANSACTION_GET_BY_ID, dataParcel, replyParcel, ...)

4. Prenos transakcije preko blokirajućeg ioctl poziva

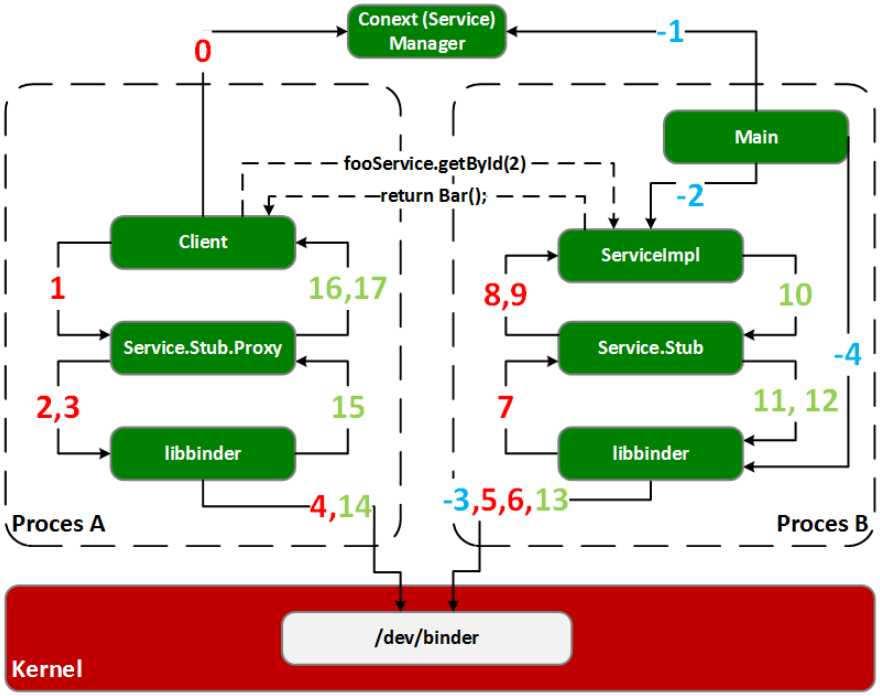
5. Buđenje iz blokirajućeg ioctl poziva i dobavljanje podataka iz transakcije

6. Dobavnjanje binder objekta

7. onTransact(TRANSACTION_GET_BY_ID, dataParcel, replyParcel, ...)

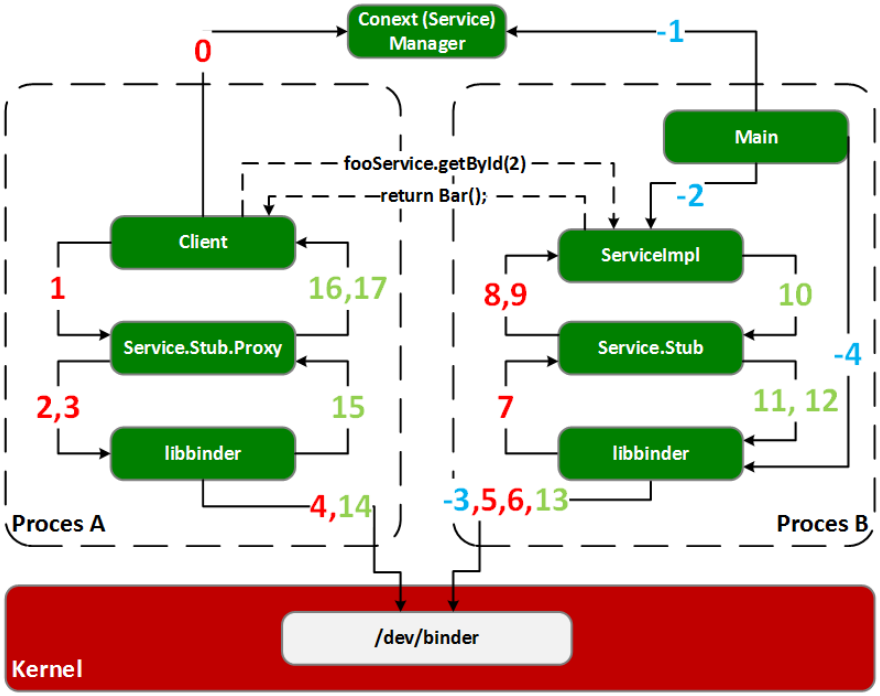
8. Deserijalizacija

9. Pozivanje same funkcije u implementaciji servisa



Binder: životni vek (4/4)

10. Povratna vrednost
11. Serijalizacija povratne vrednosti
12. Povratak u libbinder
13. Slanje povratne vrednosti preko ioctl
14. Buđenje iz blokirajućeg ioctl poziva i slanje povratne vrednosti
15. Osvežavanje replyParcel s apovratnom vrednošću
16. Deserijalizacija
17. Vraćanje povratne vrednosti klijentu



Android Binder, Java Parcel

Funkcija	Opis
(write|read)*	Upisivanje i čitanje primitivnog tipa: boolean, byte, double, float, int, long ili String. Primer: writeInt(5) i int i = readInt();
(write|read) *Array	Upisivanje i čitanje niza primitivnog tipa: boolean, byte, char, double, float, int, long, string ili SparseBoolean. Primer: writeIntArray(new int[]{1,2,3}) i int [] ia = readIntArray();;
(write|read)Parcerable	Upisavinje ičitanje klase koja implementira interfejs Parcerable.
(write|read)Bundle	Upisivanje i čitanje Bundle objekta.
(write|read)StrongBinder()	Upisivanje i čitanje IBinder objekta. Ovde se ne prenosi stvarni sadržaj, već referenca.
(write|read)FileDescriptor	Upisivanje i čitanje file descriptor-a.
(write|read)(Value|Array|List|ArrayList|Map|SparseArray)	Upis i čitanje ne tipiziranih kontejnera.

Android Binder, Native Parcel

Funkcija	Opis
(write|read)*	Upisivanje i čitanje primitivnih tipova: int32, uint32, int64, uint64, float, double, Cstring, String8 i String16.
write|read)FileDescriptor	Upisivanje i čitanje file descriptor-a.
(write|read)StrongBinder	Upisivanje i čitanje binder objekta.
(write|read)	Upisivanje i čitanje niza void * tipa.
(write|read)Inplace	Zauzimanje ili čitanje iz memorije.
(write|read)WeakBinder	Upisivanje i čitanje binder objekta.
(write|read)	Flattenable ili LightFlattenable
(write|read)NativeHandler	

Android Binder, primeri

- Integracije se razlikuje od toga gde se binder koristi
- Java aplikacija
  - Integracija je mnogo lakša
  - AIDL
- Native programi – servisi
  - Nema podrške za AIDL u Andorid SDK
    - Postoje open source alati koji mogu da napave C++ Binder kod od AIDL datoteke
    - https://github.com/iFeelSmart/AidlToCpp
    - https://github.com/iFeelSmart/AidlDataTypeGen
  - Potrebno je napisati Proxy i Stub kod

Android Binder, primeri

- **Java servis – Java klijent**
- Java servis – Java klijent + callback

- **Native servis – native klijent**
- **Native servis – Java klijent**

- Native servis – native klijent + callback
- Native service – Java klijent + callback

Android Binder, primeri

- Java servis – Java klijent
- Java servis – Java klijent + callback

- Native servis – native klijent
- Native servis – Java klijent

- Native servis – native klijent + callback
- Native service – Java klijent + callback

Android Binder: Java servis – Java klijent

- Interface

```
package com.course.android;

interface IExampleService {
          long functionExample(long input);
}
```

Android Binder: Java servis – Java klijent

- Servis Android manifest

```
<service android:name="com.course.android.ExampleService">
        <intent-filter>
                <action android:name="com.course.android.IExampleService" />
        </intent-filter>
</service>
```

Android Binder: Java servis – Java klijent

- Servis klasa

```
public class ExampleService extends Service {
        private static final String TAG = "ExampleService";
        private IExampleServiceImpl mService;

        @Override public void onCreate() {
                super.onCreate();
                this.mService = new IExampleServiceImpl();
        }

        @Override public IBinder onBind(Intent intent) {
                Log.d(TAG, "[onBind]");
                return mService;
        }

        @Override public void onDestroy() {
                this.mService = null;
                super.onDestroy();
        }
}
```

Android Binder: Java servis – Java klijent

- Implementacija servisa

```
public class IExampleServiceImpl extends IExampleService.Stub {
        private static final String TAG = "IExampleServiceImpl";

        public long functionExample(long input) {
                Log.d(TAG, "[functionExample][input: " + input + "]");
                return input + 1;
        }
}
```

Android Binder: Java servis – Java klijent

- Nasljeđivanje ServiceConnection

```
public class ExampleActivity extends Activity implements ServiceConnection {
        private static final String TAG = "ExampleActivity";
        private IExampleService mService;
```

- Povezivanje na servis

```
Intent serviceIntent = new Intent("com.course.android.IExampleService");
serviceIntent = createExplicitFromImplicitIntent(this, serviceIntent);
if (!bindService(serviceIntent, this, Context.BIND_AUTO_CREATE)) {
        Log.w(TAG, "Failed to bind to service");
}
```

Android Binder: Java servis – Java klijent

- Implementacija ServiceConnection

```
public void onServiceConnected(ComponentName name, IBinder service) {
        Log.d(TAG, "[onServiceConnected][" + name + "]");
        mService = IExampleService.Stub.asInterface(service);
        try {
                long ret = mService.functionExample(1);
                Log.d(TAG, "[onServiceConnected][fun output: " + ret + "]");
        } catch (RemoteException e) {
                e.printStackTrace();
        }
}

public void onServiceDisconnected(ComponentName name) {
        Log.d(TAG, "[onServiceDisconnected][" + name + "]");
        mService = null;
}
```

Android Binder, primeri

- Java servis – Java klijent
- Java servis – Java klijent + callback

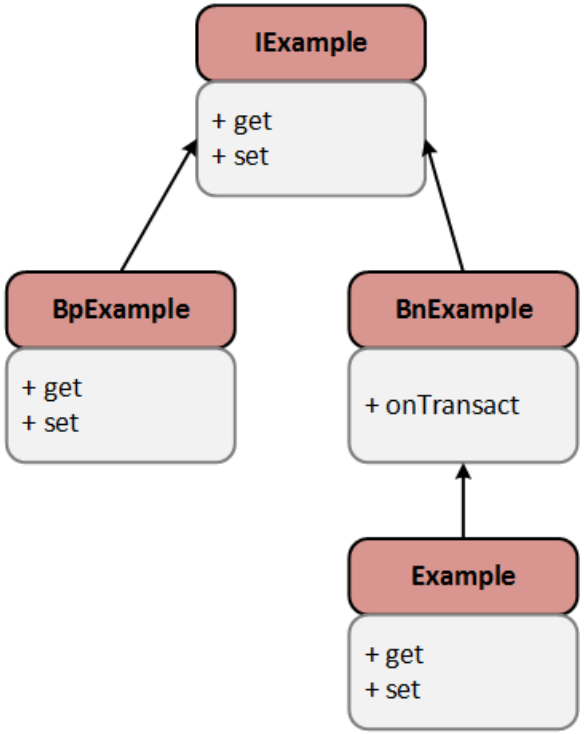
- Native servis – native klijent
- Native servis – Java klijent

- Native servis – native klijent + callback
- Native service – Java klijent + callback

Android Binder: Native servis – Native klijent

- Interface: IExample.h

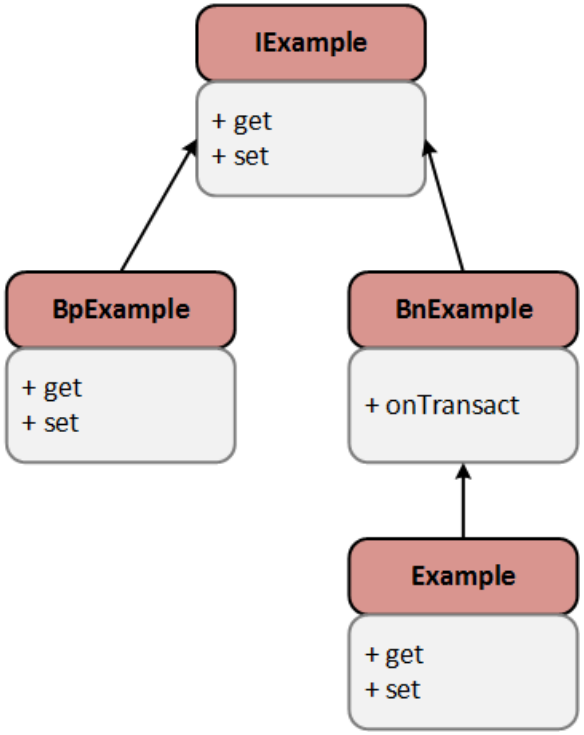
```
#pragma once

#include <binder/IInterface.h>
#include <binder/IBinder.h>
#include <utils/String16.h>

namespace android {
class IExample : public IInterface {
public:
          enum {
                    GET = IBinder::FIRST_CALL_TRANSACTION,
                    SET
          };

          virtual int get() = 0;
          virtual void set(int arg) = 0;

          DECLARE_META_INTERFACE(Example);
};
}
```



Android Binder: Native servis – Native klijent

- Interface: IExample.cpp

```
#include "Example.h"
#include "BpExample.h"

using namespace android;

IMPLEMENT_META_INTERFACE(Example,
"com.course.android.IExample");

}
```



Android binder: DECLARE_META_INTERFACE

- frameworks/native/include/binder/IInterface.h

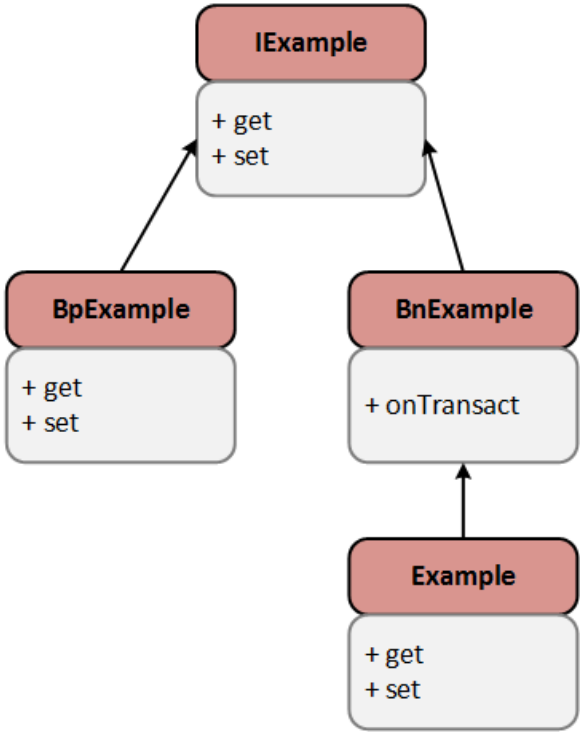
```
#define DECLARE_META_INTERFACE(INTERFACE)                               \
    static const android::String16 descriptor;                          \
    static android::sp<I##INTERFACE> asInterface(                       \
            const android::sp<android::IBinder>& obj);                  \
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

Android binder: IMPLEMENT_META_INTERFACE

```
#define IMPLEMENT_META_INTERFACE(INTERFACE, NAME)                       \
    const android::String16 I##INTERFACE::descriptor(NAME);             \
    const android::String16&                                            \
            I##INTERFACE::getInterfaceDescriptor() const {              \
        return I##INTERFACE::descriptor;                                \
    }                                                                   \
    android::sp<I##INTERFACE> I##INTERFACE::asInterface(                \
            const android::sp<android::IBinder>& obj)                   \
    {                                                                   \
        android::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() { }                                   \
```

Android Binder: Native servis – Native klijent

- Servis: implementacija Bn (Binder Native-Stub)
  - BnExample.h

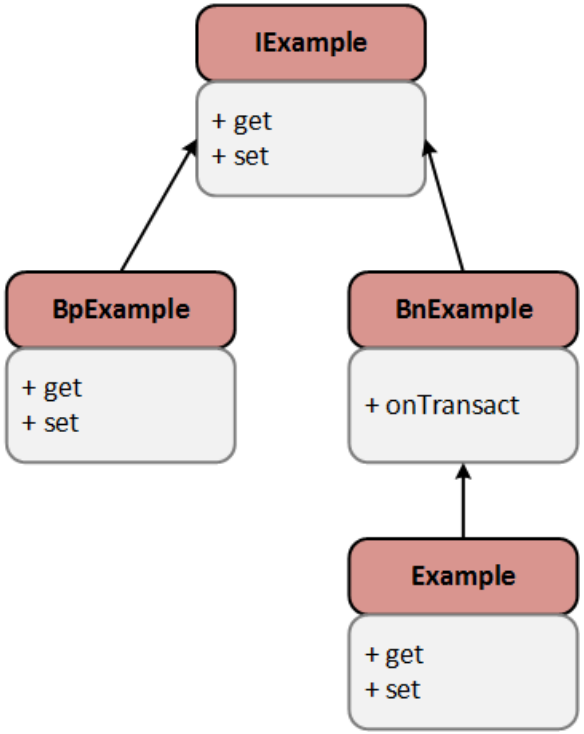



```
#pragma once

#include <binder/IInterface.h>
#include <binder/IBinder.h>
#include <utils/String16.h>

#include "IExample.h”

namespace android {

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

}
```

Android Binder: Native servis – Native klijent

- Servis: implementacija Bn (Binder Native-Stub)
  - BnExample.cpp

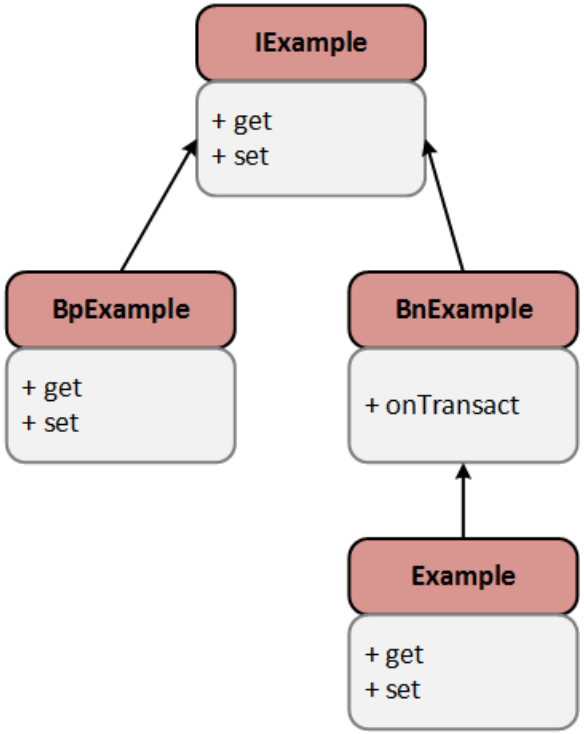
```
#include "BnExample.h”
#include "IExample.h”

using namespace android;

status_t BnExample::onTransact(uint32_t code,
const Parcel& data, Parcel* reply, uint32_t flags) {
    switch(code){
        case SET: {
            CHECK_INTERFACE(IExample, data, reply);
            int arg = data.readInt32();
            set(arg);
            reply->writeNoException();
            break;
        }
        case GET : {
            CHECK_INTERFACE(IExample, data, reply);
            int retVal = get();
            reply->writeNoException();
            reply->writeInt32(retVal);
            break;
        }
        default:
            return BBinder::onTransact(code, data, reply, flags);
    }
    return NO_ERROR;
}
```



Android Binder: Native servis – Native klijent

- Servis: sama implementacija funkcija
  - Example.h

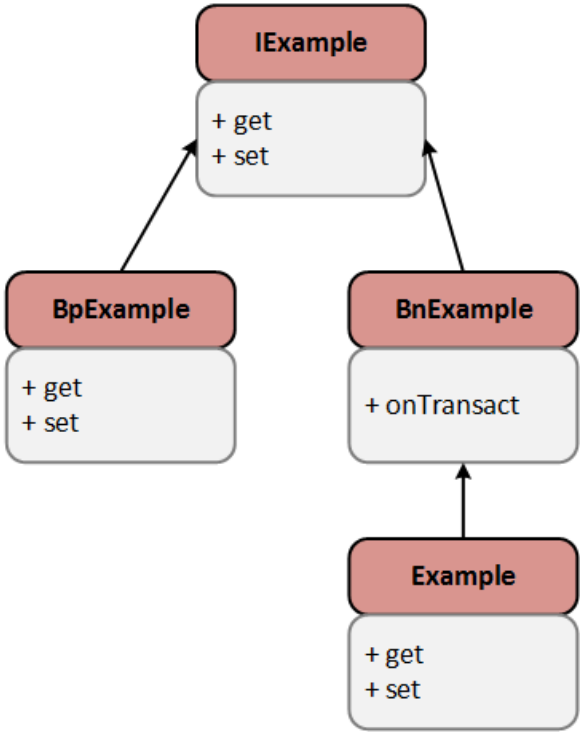
```
#pragma once

#include <BnExample.h>

namespace android {

class Example : public BnExample {
public:
    Example();
    ~Example();

    int get();
    void set(int arg);
};
}
```



Android Binder: Native servis – Native klijent

- Servis: sama implementacija funkcija
  - Example.cpp

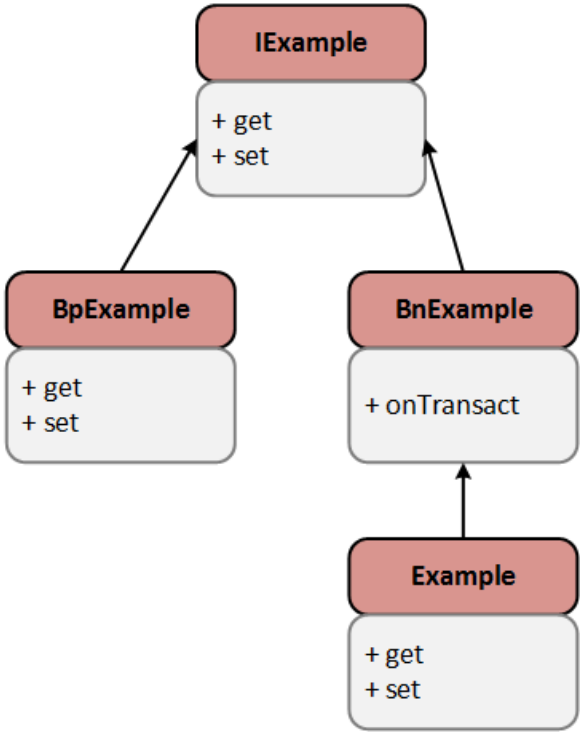
```
#include "Example.h"

#define LOG_TAG " ExampleService"
using namespace android;

Example:: Example() {
    ALOGD("Initializing Example Service...");
}

Example::~ Example() {
    // Here should do disconnection
}

int Example::get() {
    return 1;
}

void Example::set(int arg) {
}
```



Android Binder: Native servis – Native klijent

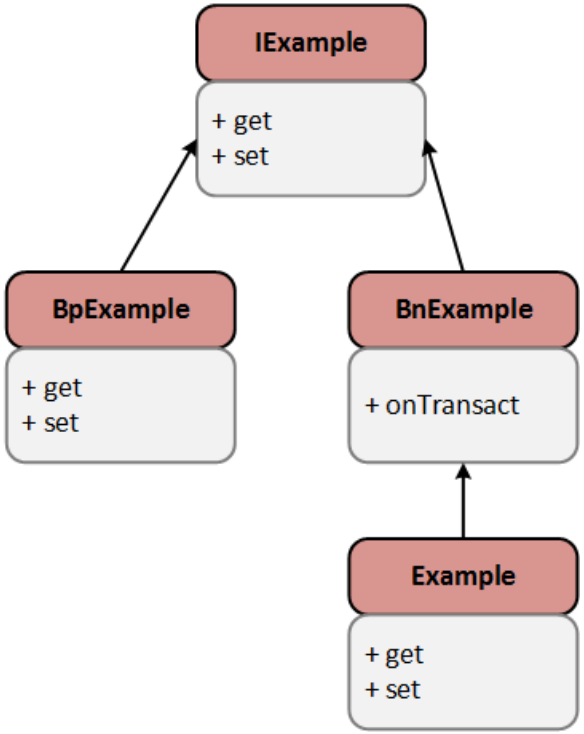
- Servis: main

```
#include <binder/IServiceManager.h>
#include <binder/IPCThreadState.h>
#include <utils/String16.h>
#include <Example.h>

using namespace android;

int main(int argc, char** argv) {
    defaultServiceManager()->
addService(String16("Example", new Example());
    android::ProcessState::self()->startThreadPool();
    IPCThreadState::self()->joinThreadPool();
    return 0;
}
```



Android Binder: Native servis – Native klijent

- Klijent: implementacija Bp (Binder Proxy)
  - BpExample.h

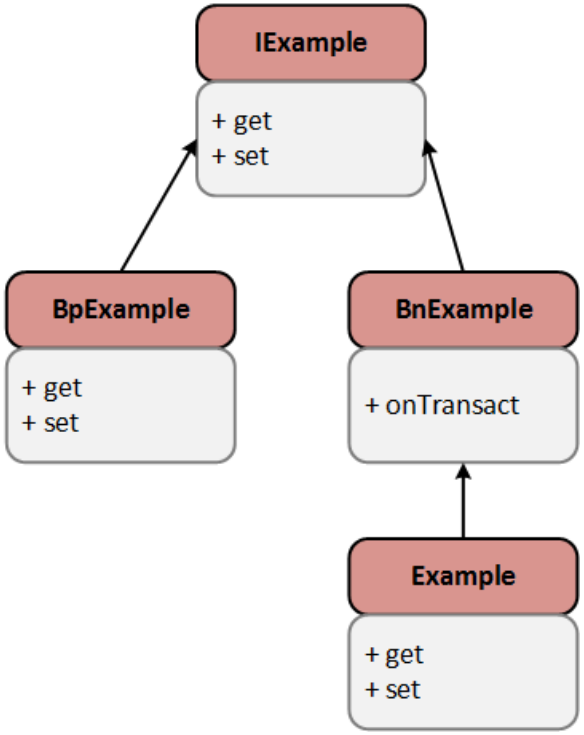



```
#pragma once

#include <binder/IInterface.h>
#include <binder/Parcel.h>
#include <utils/StrongPointer.h>
#include "IExample.h"

namespace android {

class BpExample : public BpInterface<IExample> {
public:
        BpExample(const sp<IBinder>& impl);

        virtual int get();
        virtual void set(int arg);
};
```

Android Binder: Native servis – Native klijent

- Klijent: implementacija Bp (Binder Proxy)
  - BpExample.cpp

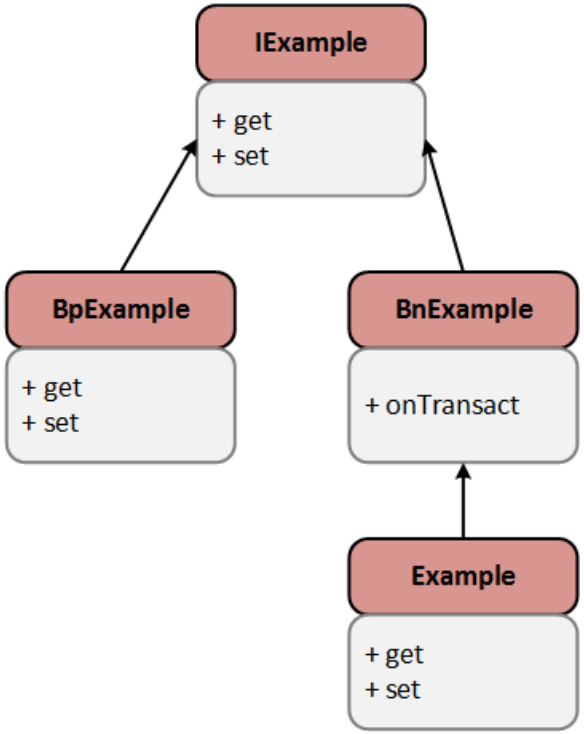
```
#include <BpExample.h>
#include <IExample.h>

using namespace android;

BpExample::BpExample(const sp<IBinder>& impl) : BpInterface<IExample
    ALOGD("BpExample()");
}

int BpExample::get()
{
          Parcel data, reply;    int retVal = -1;
          data.writeInterfaceToken(IExample::getInterfaceDescriptor
          remote()->transact(GET, data, &reply);
          reply.readInt32();
          retVal = reply.readInt32();
          return retVal;
}

void BpExample::set(int arg){
          Parcel data, reply;
          data.writeInterfaceToken(IExample::getInterfaceDescriptor());
          data.writeInt32(arg);
          remote()->transact(SET, data, &reply);
          reply.readInt32();
}
}
```



Android Binder: Native servis – Native klijent

- Klijent: test program

```
#include <IExample.h>
#include <binder/IServiceManager.h>
#include <binder/IBinder.h>

#define LOG_TAG "ExampleClient"

using namespace android;

int main() {
        sp<IServiceManager> smanager =
defaultServiceManager();

        sp<IBinder> binder =
smanager->getService(String16("Example"));

        sp<IExample> example = interface_cast <IExample> (binder);

        int testValue = example->get();
        ALOGI("Test value: %d", testValue);
}
```



Android Binder, primeri

- Java servis – Java klijent
- Java servis – Java klijent + callback

- Native servis – native klijent
- Native servis – Java klijent

- Native servis – native klijent + callback
- Native service – Java klijent + callback

Android Binder: Native servis – Java klijent

- Interface: IExample.aidl
- Hint: IExample.cpp
- IMPLEMENT_META_INTERFACE(Example, "com.course.android.IExample");}

```
package com.course.android;

interface IExample{
        int get();

        void set(int arg);
}
```

Android Binder: Native servis – Java klijent

- Klijent main

```
private IExample example = null;
IBinder binder = (IBinder)
ServiceManager.getService("Example");
if (binder != null) {
    example = IExample.Stub.asInterface(binder);
} else {
    Log.e(TAG, "[bind example][NOK]");
    return;
}
Log.d(TAG, "[bind to example][ok]");

try {
    Log.d(TAG, "[example service get][" + example.get() + "]");
    example.set(1);
} catch (RemoteException re) {
    re.printStackTrace();
}
```

Android Binder: Složeni tipovi?

- Pitanje: void set(**MyParcelable** arg) preko Binder-a?
- Odgovor: Parcelable interface

Android Binder: Složeni tipovi?

Parcerable: MyParcelable.aidl

```
package com.course.android;

parcelable MyParcelable;
```

Android Binder: Složeni tipovi?

- MyParcelable.java

```
package com.course.android;

public class MyParcelable implements Parcelable {
     private int mData1;
     private long mData2;

     public void writeToParcel(Parcel out, int flags) {
          out.writeInt(mData1);
          out.writeLong(mData2);
     }

     private MyParcelable(Parcel in) {
          mData1 = in.readInt();
          mData2 = in.readLong();
     }
```

Android Binder: Složeni tipovi?

```
public class MyParcelable implements Parcelable {
...
    public int describeContents() {
        return 0;
    }

    public static final Parcelable.Creator<MyParcelable> CREATOR
            = new Parcelable.Creator<MyParcelable>() {
        public MyParcelable createFromParcel(Parcel in) {
            return new MyParcelable(in);
        }

        public MyParcelable[] newArray(int size) {
            return new MyParcelable[size];
        }
    };
}
```

Android Binder: Složeni tipovi?

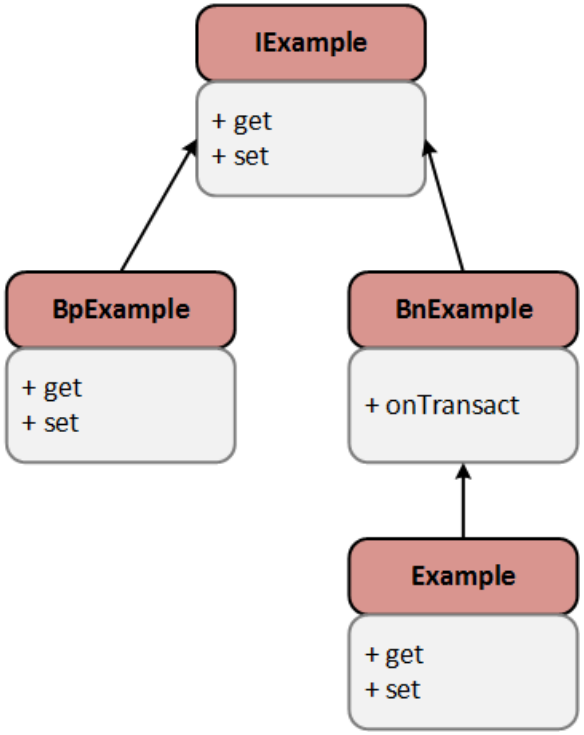
- Interface: IExample.aidl

```
package com.course.android;

import com.course.android.MyParcelable;

interface IExample{
        int get();

        void set(MyParcelable arg);
}
```

Android Binder: Složeni tipovi?

- Interface: IExample.h

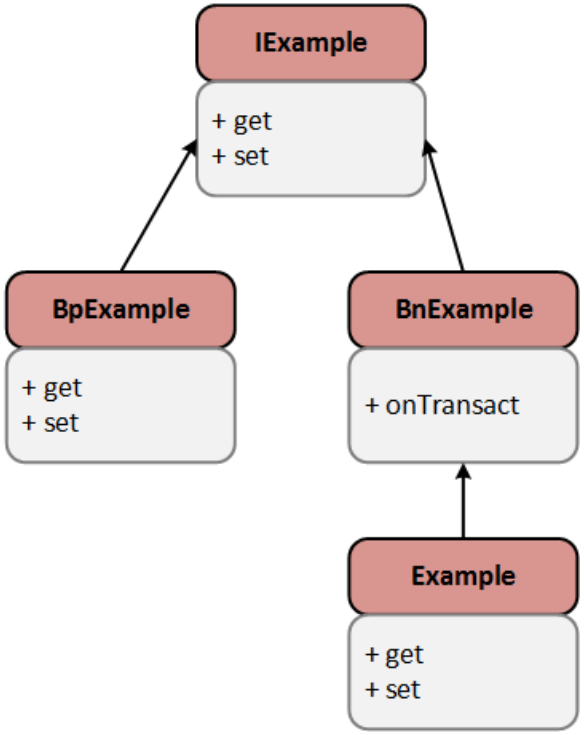
```
#pragma once

#include <binder/IInterface.h>
#include <binder/IBinder.h>
#include <utils/String16.h>

namespace android {
class IExample : public IInterface {
public:
        enum {
                GET = IBinder::FIRST_CALL_TRANSACTION,
                SET
        };

        virtual int get() = 0;
        virtual void set(int arg1, long arg2) = 0;

        DECLARE_META_INTERFACE(Example);
};
}
```



Android Binder: Složeni tipovi?

- Servis: implementacija Bn (Binder Native-Stub)
  - BnExample.cpp

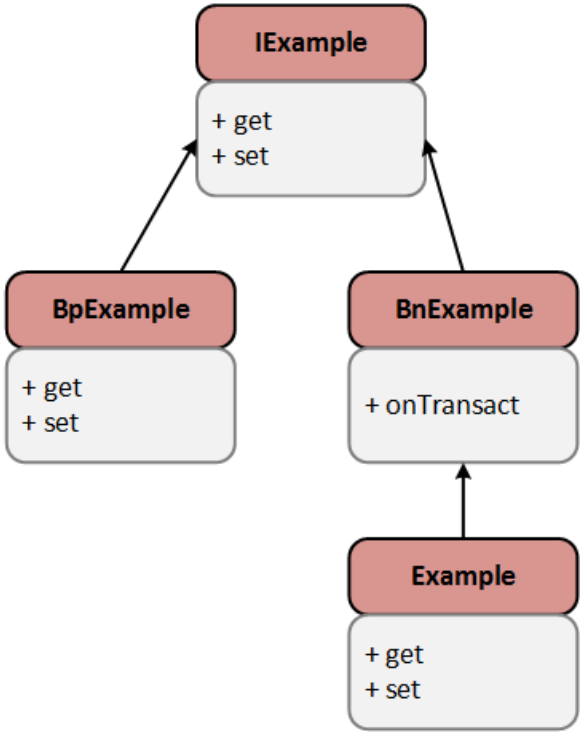
```
#include "BnExample.h"
#include "IExample.h"

using namespace android;

status_t BnExample::onTransact(uint32_t code,
const Parcel& data, Parcel* reply, uint32_t flags) {
    switch(code){
        case SET: {
            CHECK_INTERFACE(IExample, data, reply);
            int arg1 = data.readInt32();
            long arg2 = (long)data.readInt64();
            set(arg1, arg2);
            reply->writeNoException();
            break;
        }
...
        default:
            return BBinder::onTransact(code, data, reply, flags);
    }
    return NO_ERROR;
}
```



Android Binder: Složeni tipovi?

- Servis: sama implementacija funkcija
  - Example.h

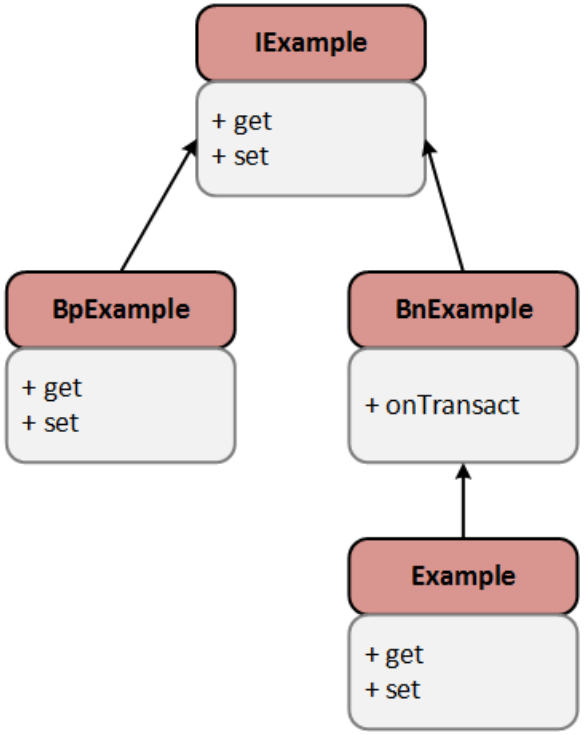
```
#pragma once

#include <BnExample.h>

namespace android {

class Example : public BnExample {
public:
    Example();
    ~Example();

    int get();
    void set(int arg1, long arg2);
};
}
```



Android Binder: Složeni tipovi?

- Servis: sama implementacija funkcija
  - Example.cpp

```
#include “Example.h”

#define LOG_TAG " ExampleService”
using namespace android;

Example:: Example() {
    ALOGD("Initializing Example Service...");
}

Example::~ Example() {
    // Here should do disconnection
}

int Example::get() {
    return 1;
}

void Example::set(int arg1, long arg2) {
}
```



Agenda

- Init proces
- Android svojstva
- Binder
- JNI
- Ashmem
- Ograničenja

JNI – Šta je JNI?

- Java Native Interface (JNI)
- Native – platformski zavisna progamska rešenja (C/C++)
- Programska sprega Java programskog jezika za pristup platformski zavisnim programskim rešenjima

JNI – Upotreba

- Upotreba postojećih platformski zavisnih programskih rešenja
- Pristup fizičkim komponentama sistema
- Optimizacija programskog rešenja

JNI – Java implementacija

- JNI podrazumeva deklaraciju metoda Java programske klase koje će biti implementirane u okviru native programskog rešenja
- Native metode označavaju se uz pomoć **native** modifikatora
- Klasa sa native metodama može sadržati Java metode
- Native metode su ravnopravne sa Java metodama i pozivaju se na isti način

```
package rtrk.pnrs.jniexample;

public class FibonacciNative {
    public native int get(int n);
}
```

JNI – Native implementacija

- Implementacija native metoda podrazumeva definiciju funkcija koje podležu određenim pravilima
- Naziv funkcije mora odgovarati punom nazivu programske klase i metode
- Prvi parametar funkcije mora biti kontekst virtuelne mašine (**JNIEnv**)
- Drugi parametar funkcije mora biti kontekst instance date programske klase (**jobject**)
- Obavezna je upotreba tipova podataka definisanih jni.h zaglavljem (**jint, jlong, jfloat, jstring, ...**)

```
jint Java_rtrk_pnrs_jniexample_FibonacciNative_get
     (JNIEnv *, jobject, jint);
```

JNI – javah

- Da bi se smanjio broj grešaka pri definiciji native funkcija na raspolaganju je **javah** alat
- Uz pomoć javah alata generišu se deklaracije native funkcija
- Opšti oblik:

```
javah -o <output_file> -classpath <path> <classes>
```

```
javah -o rtrk_pnrs_jniexample_FibonacciNative.h
  -classpath out/target/common/obj/APPS/
  JNIExample_intermediates/classes.jar
  rtrk.pnrs.jniexample.FibonacciNative
```

JNI – Učitavanje deljene biblioteke

- Implementacija native metoda mora biti deo deljene programske biblioteke
- Deljena programska biblioteka mora biti eksplicitno učitana u okviru klase čije metode implementira
- Pri učitavanju naziv deljene biblioteke navodi se bez lib prefiksa i .so ekstenzije

```
static {
    System.loadLibrary("fibonacci");
}
```

JNI - Ograničenja

- Platfomska zavisnost
  - Platformski zavisne celine čine celo programsko rešenje platformski zavisnim
  - Zbog platformske zavisnosti JNI se najčešće upotrebljava za realizaciju programskih rešenja koja ulaze u sastav operativnog sistema
  - U slučaju pojedinačnih aplikacija potrebno je obezbediti implementaciju za svaku podržanu platformu
- Kompleksnost native implementacije pristupa delovima Java rešenja
  - Podrazumevani tok poziva je iz Jave u native
  - Pozivi Java metoda iz native dela rešenja omogućen je jedino refleksijom
- Slaba programska sprega između Java i native rešenja

Agenda

- Init proces
- Android svojstva
- Binder
- JNI
- Ashmem
- Ograničenja

Ashmem

- Šta?
  - Ashmem je Andorid-ovo rešenje za deljenje memorije između više procesa
- Zašto?
  - System V, rešenje od koga je Ashmem najverovatnije potekao je mogao da deli memoriju samo između procesa koji su u srodstvu (fork)
  - Brojanje referenci (process killer)
- Kako?
  - Ashmem je realizovan u samom Linux kernelu. Deo je Android Linux kernel poboljšanja.
- Osobine:
  - Koristi virtualnu susednu memoriju.
    - Za razliku od ashmem-a, u ranijim verzijama Android-a je bio pmem rešenje koje je koristilo fizički susednu memoriju.

Ashmem

- Problemi:
  - Prilikom zauzimanja ashmem memorije potrebno je navesti naziv zauzetog regiona
  - Ukoliko se u drugom procesu zauzme region sa istim nazivom, on neće pokazivati na istu memoriju koja je u prethodnom koraku zauzeta
- Rešenje:
  - Prilikom zauzimanja memorije dobija se file descriptor koji predstavlja jedinstveni indentifikator deljenog regiona
  - Proslediti preko binder veze file descriptor drugom procesu koji će na osnovu njega mapirati kod sebe isti deljeni region
- Primer upotrebe:
  - Teletext use case
  - PVR root prava

Ashmem

Funkcija	Opis
`int ashmem_create_region(const char *name, size_t size);`	Zauzima potencijalnu deljenu memoriju velićine size i naziva je name. Povratna vrednost funkcije je file descriptor.
`int ashmem_set_prot_region(int fd, int prot);`	Dodeljuje prava pristupa memoriji: PROT_EXEC, PROT_READ, PROT_WRITE i PROT_NONE. Moguće je koristiti konkatenaciju sa ili.
`int ashmem_pin_region(int fd, size_t offset, size_t len);`	Pozivanjem ove funkcije memorija vezana za file descriptor neće moći da se oslobodi od strane kernel-a u slučaju OOM (Out Of Memory).
`int ashmem_unpin_region(int fd, size_t offset, size_t len);`	Pozivanjem ove funkcije memorija vezana za file descriptor moći će da se oslobodi od strane kernel-a u slučaju OOM.
`int ashmem_get_size_region(int fd);`	Vraća informaciju koliko memorije je zauzeto.
`void *mmap(void *addr, size_t length, int prot, int flags, int fd, off_t offset);`	Virtualno povezivanje user i kernel memorijskog prostora. Ono što se upiše u jedan memorijski prostor momentalno je vidljivo i u drugom. Više informacija: http://man7.org/linux/man-pages/man2/mmap.2.html
`int munmap(void *addr, size_t length);`	Poništavanje virtualnog povezivanje izmeću user i kernel memorijskog prostora.

Ashmem: primer

- Native primer: servis

```
size_t len = 4096;
int fd = ashmem_create_region("Ashmem region name", len);
ashmem_set_prot_region(fd, PROT_READ | PROT_WRITE);
void* ptr = mmap(NULL, len, PROT_READ | PROT_WRITE, MAP_SHARED, fd, 0);
ashmem_set_prot_region(fd, PROT_READ);

// write into ptr for len as desired
...
munmap(ptr, len);
close(fd);
```

- Native primer: klijent

```
void* ptr = mmap(NULL, len, PROT_READ, MAP_SHARED, fd, 0);
// read from ptr up to len as desired
...
munmap(ptr, len);
```

Ashmem: primer

- Java primer
- MemoryFile je samo Java omotač oko native ashmem implementacije

```
import android.os.MemoryFile;

private MemoryFile mMemoryFile = new MemoryFile("ashmem-java-region", 720 * 576 *
4);
```

Ashmem: primer

- Primeri ako se ashmem region pravi u Java aplikaciji
  - Java – native preko JNI-a
  - Java – native preko binder-a
- Primeri ako se ashmem region se pravi u native aplikaciji
  - Kako podesiti fd u MemoryFile klasi?

Ashmem: primer, Java – native, jni

- Java

```
public native boolean setMemoryFile(MemoryFile mf);
```

- JNI

```
static jboolean jni_setMemoryFile(JNIEnv* env, jclass clz, jobject jmf) {
        jclass clsMF = env->FindClass("android/os/MemoryFile");
        jfieldID fldFD = env->GetFieldID(clsMF, "mFD",
"Ljava/io/FileDescriptor;");
        jobject objFD = env->GetObjectField(jmf, fldFD);

        jclass clsFD = env->FindClass("java/io/FileDescriptor");
        fldFD = env->GetFieldID(clsFD, "descriptor", "I");

        jint fd = env->GetIntField(objFD, fldFD);


void* ptr = mmap(NULL, len, PROT_READ | PROT_WRITE, MAP_SHARED, fd, 0);
// write, read memory
...
munmap(ptr, len);
```

Ashmem: primer, Java – native, binder

- Application: AIDL

```
package com.course.android.nativeservice;
import android.os.ParcelFileDescriptor;

interface INativeService {

     int get();

     void set(in ParcelFileDescriptor arg);
}
```

- Application: Java

```
private INativeService nativeService = null;

...
MemoryFile mf = new MemoryFile("java-ashmem-region", 1024);

Field f = mf.getClass().getDeclaredField("mFD"); // NoSuchFieldException
f.setAccessible(true);
FileDescriptor fileDescriptor = (FileDescriptor) f.get(mf); // IllegalAccessException

ParcelFileDescriptor pfd = ParcelFileDescriptor.dup(fileDescriptor);
nativeService.set(pfd);
```

Ashmem: primer, Java – native, binder

- Native

```
status_t BnNativeService::onTransact(uint32_t code, const Parcel& data,
        Parcel* reply, uint32_t flags) {
    switch (code) {
    case SET: {
        CHECK_INTERFACE(INativeService, data, reply);
        int outcommchannel;
        int fd= data.readParcelFileDescriptor(outcommchannel);
        set(fd);
        reply->writeNoException();
        break;
...
    }


void NativeService::set(int fd) {
        void* ptr = mmap(NULL, len, PROT_READ | PROT_WRITE, MAP_SHARED, fd, 0);
        // write, read memory
        ...
        munmap(ptr, len);
}
```

Agenda

- Init proces
- Android svojstva
- Binder
- JNI
- Ashmem
- Ograničenja

Ograničenja

- Binder podržava maksimalno **15** Binder niti, po jednom procesu

```
…
static int open_driver() {
    int fd = open("/dev/binder", O_RDWR);
    if (fd >= 0) {
        …
        size_t maxThreads = 15;
        result = ioctl(fd, BINDER_SET_MAX_THREADS, &maxThreads);
        …
    } else {
        …
    }
    return fd;
}
…
```

Ograničenja

- Binder limitira svoj bafer za transakcije na **1 MB** po procesu, za sve konkurentne transakcije
  - Ukoliko su argumenti/povratne vrednosti prevelike da bi stale u ovaj bafer, dobije se poseban `TransactionTooLargeException` izuzetak
  - Pošto je u pitanju bafer koji se deli za sve transakcije u okviru procesa, čak i transakcije koje nisu velike, ukoliko ih ima puno mogu dovesti do ovog problema
  - Kada se ovaj problem desi, ne možemo znati da li nismo uspeli da pošaljemo transakciju, ili da primimo odgovor
  - Držite veličinu podataka koja se šalje transakcijama malom, i koristite umesto toga deljenu memoriju – `ashmem` (gde je to moguće)

Kraj

- Pitanja?

Literatura

- Pap dr Ištvan, Lukić dr Nemanja, "*Projektovanje i arhitekture softverskih sistema – Sistemi zasnovani na Androidu*", 2015.
- Aleksandar Gargenta, "*Deep Dive into Android IPC/Binder Framework*", Android Builders Summit, 2013.
- Thorsten Schreiber, "*Android Binder - Android Interprocess Communication*", 2011.
- https://sites.google.com/site/io/an-introduction-to-android
- http://www.it.iitb.ac.in/frg/wiki/images/5/5b/113050076_Rajesh_Proddutturi_week6_presentation_4_2012-08-11.pdf
- http://www.androidenea.com/2010/03/share-memory-using-ashmem-and-binder-in.html
- Jonathan Levin, Android internals: A confectioner's Cookbook, Volume I: The Power User's View, 11/2015.

Contact us

RT-RK Institute for Computer Based Systems
Narodnog fronta 23a
21000 Novi Sad
Serbia

www.rt-rk.com
info@rt-rk.com

RT - RK
COMPUTER BASED SYSTEMS