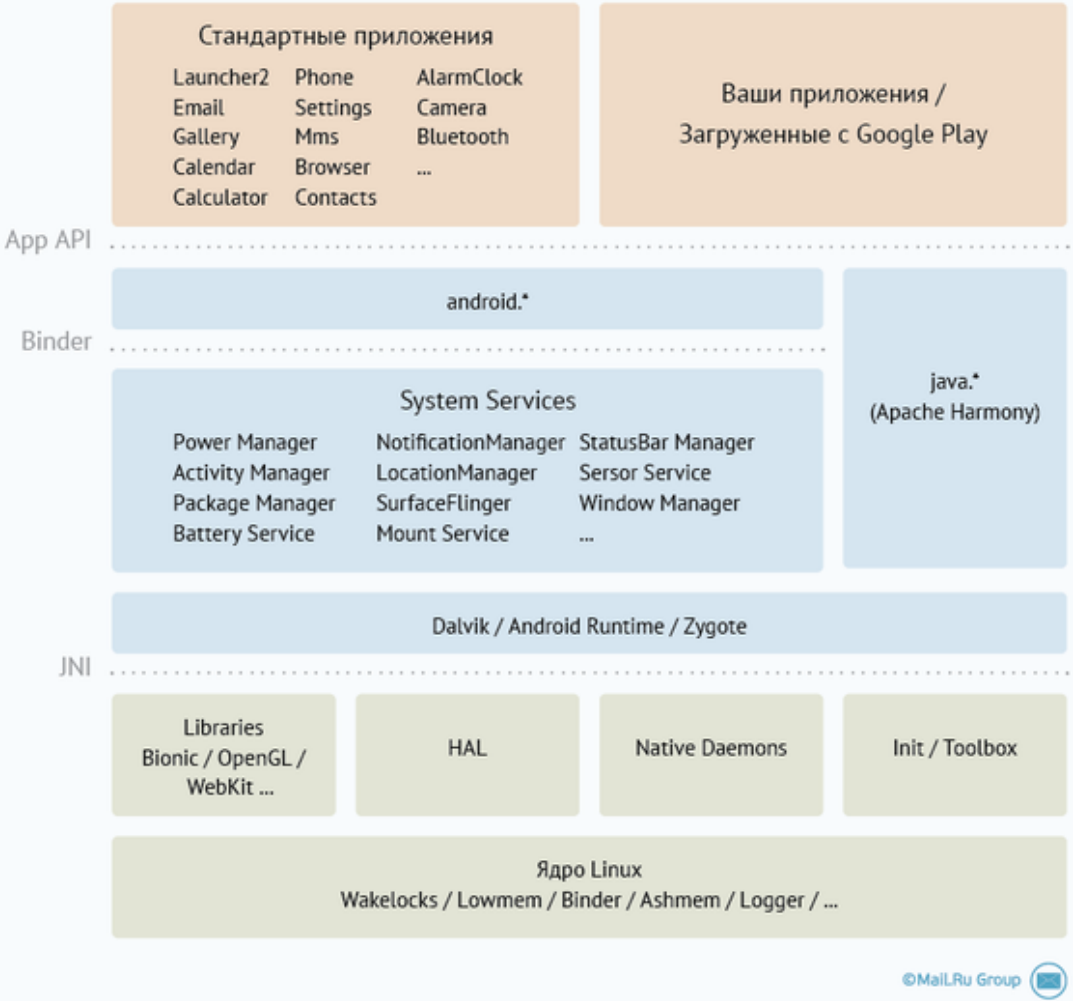
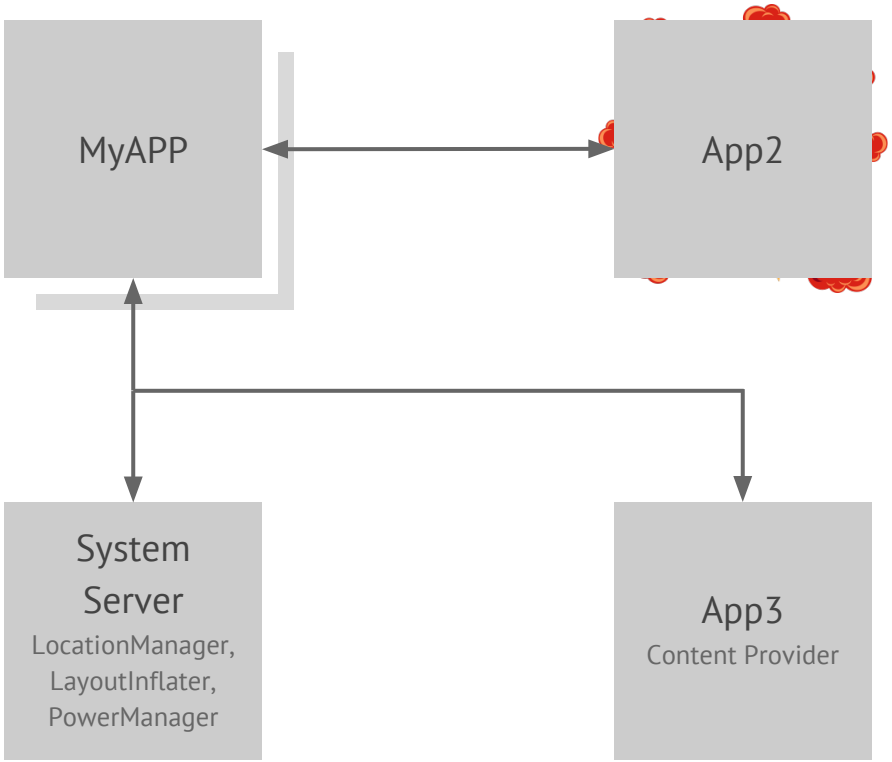
одноклассники

ПРЕПАРИРУЕМ АНДРОИД

Никитин Алексей
Андроид разработчик в Одноклассниках


Стандартные приложения
Launcher2
Email
Gallery
Calendar
Calculator
Phone
Settings
Mms
Browser
Contacts
AlarmClock
Camera
Bluetooth
...
Ваши приложения /
Загруженные с Google Play
App API
android.*
Binder
System Services
Power Manager
Activity Manager
Package Manager
Battery Service
NotificationManager
LocationManager
SurfaceFlinger
Mount Service
StatusBar Manager
Sersor Service
Window Manager
...
java.*
(Apache Harmony)
Dalvik / Android Runtime / Zygote
JNI
Libraries
Bionic / OpenGL /
WebKit ...
HAL
Native Daemons
Init / Toolbox
Ядро Linux
Wakelocks / Lowmem / Binder / Ashmem / Logger / ...
©Mail.Ru Group

1. ВЗАИМОДЕЙСТВИЕ ПРОЦЕССОВ
2. НЕМНОГО О ПАМЯТИ
3. ПЕРВЫЕ 30 СЕКУНД ЖИЗНИ
4. ЭНЕРГОПОТРЕБЛЕНИЕ


MyAPP
App2
System
Server
LocationManager,
LayoutInflater,
PowerManager
App3
Content Provider

/bionic/libc/docs/SYSV-IPC.TXT

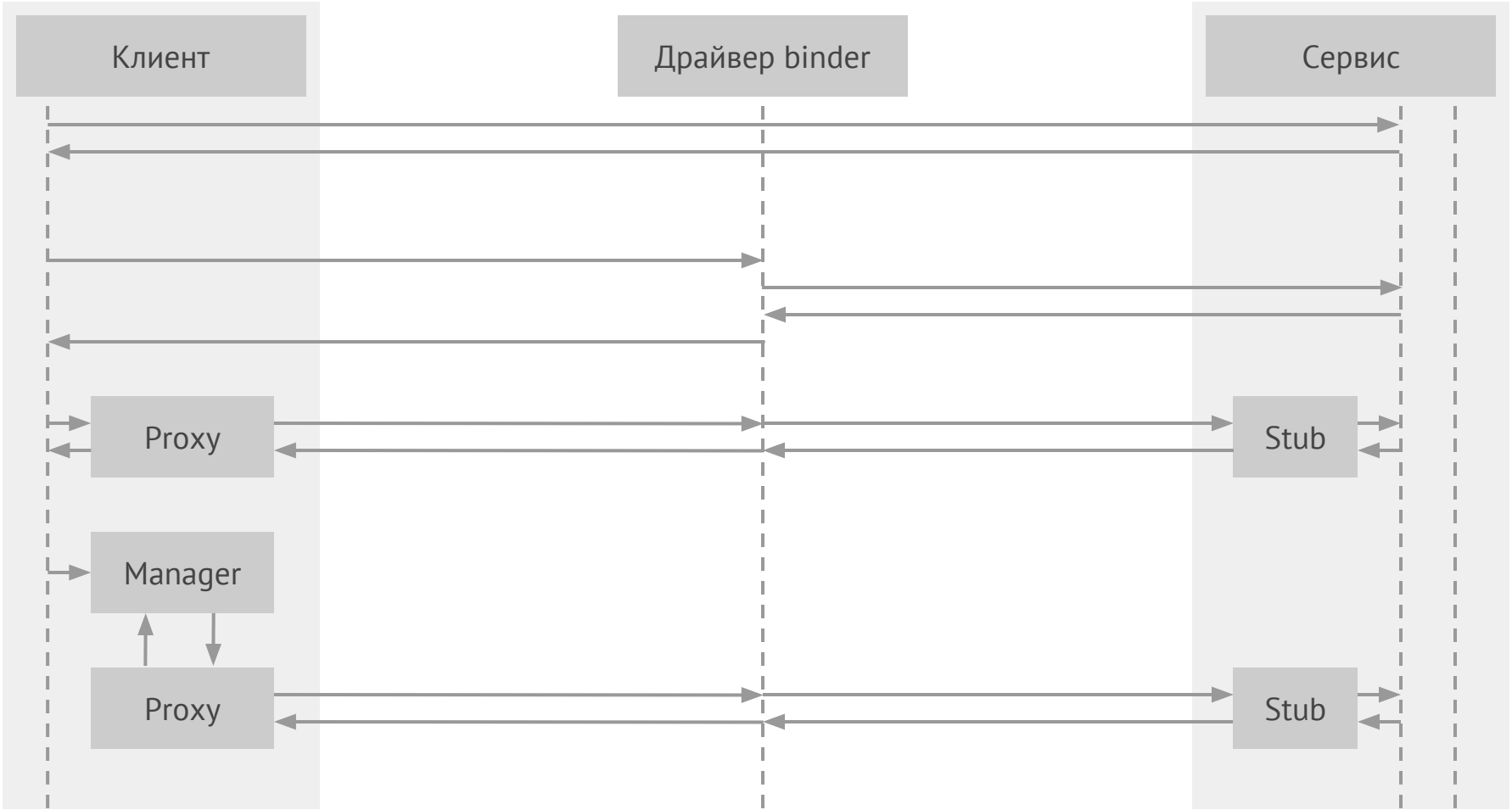
```
Android does not support System V IPCs, i.e. the facilities provided by the
following standard Posix headers:

  <sys/sem.h>   /* SysV semaphores */
  <sys/shm.h>   /* SysV shared memory segments */
  <sys/msg.h>   /* SysV message queues */
  <sys/ipc.h>   /* General IPC definitions */

The reason for this is due to the fact that, by design, they lead to global
kernel resource leakage.

For example, there is no way to automatically release a SysV semaphore
allocated in the kernel when:

- a buggy or malicious process exits
- a non-buggy and non-malicious process crashes or is explicitely killed.

Killing processes automatically to make room for new ones is an
important part of Android's application lifecycle implementation. This means
that, even assuming only non-buggy and non-malicious code, it is very likely
that over time, the kernel global tables used to implement SysV IPCs will fill
up.

At that point, strange failures are likely to occur and prevent programs that
use them to run properly until the next reboot of the system.

And we can't ignore potential malicious applications. As a proof of concept
here is a simple exploit that you can run on a standard Linux box today:
```

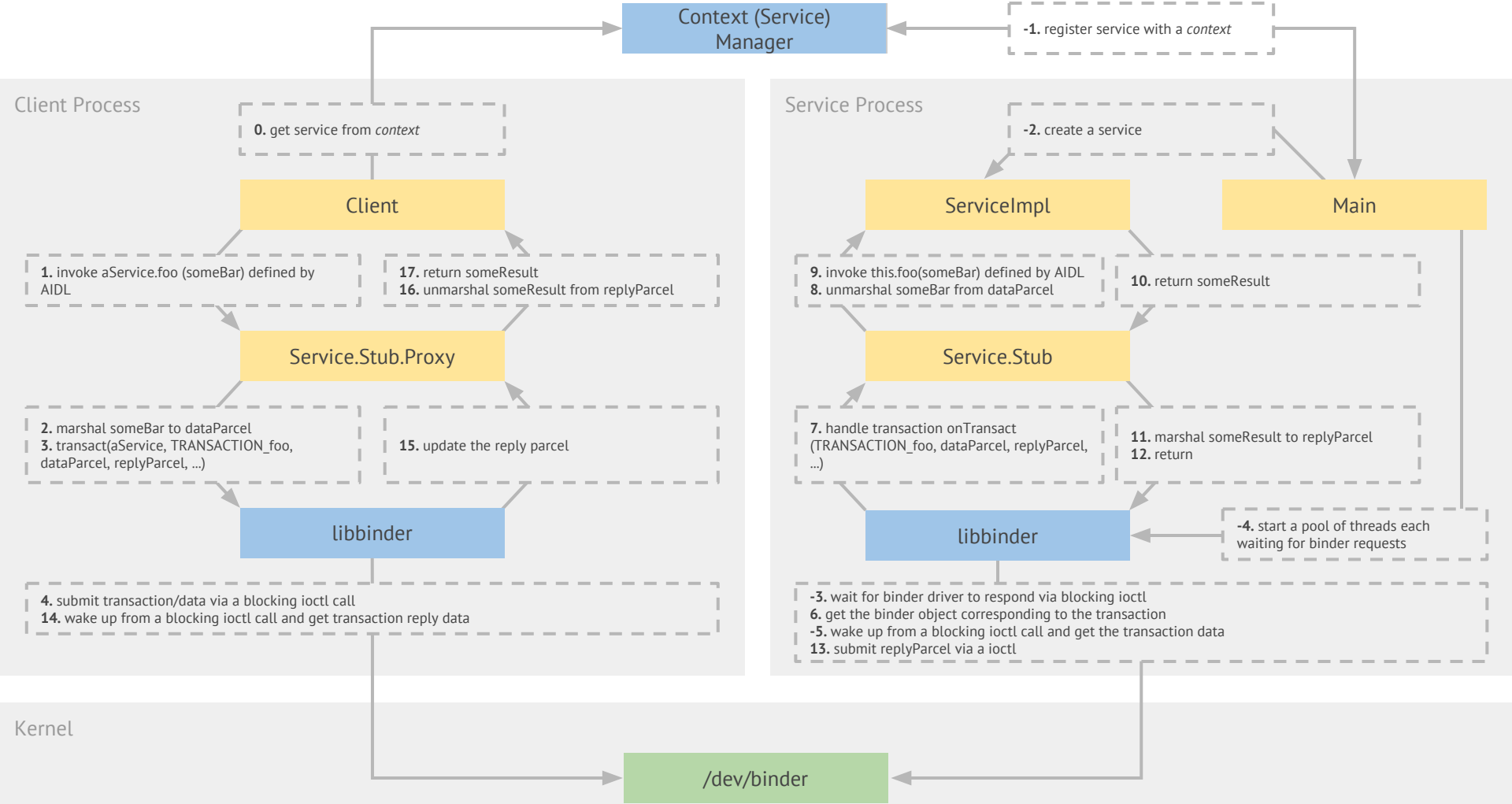
Binder

- Death notification
- Управление потоками
  - Pool потоков
  - синхронные/асинхронные вызовы
- UID/PID отправителя/получателя.
  Вы можете понять, разрешено ли ему это.

Binder

- Передача объектов через несколько процессов, подсчёт ссылок
- Передача дескрипторов файлов
- Поддержка рекурсии
- Плюшки для нас:
  - AIDL
  - Поддержка базовых простых типов
  - Работа с транзакциями через Stub и Proxy
- Вызовы внутри одного процесса — как на локальных объектах.


Клиент
Драйвер binder
Сервис
Proxy
Stub
Manager
Proxy
Stub


Context (Service) Manager
-1. register service with a context
Client Process
0. get service from context
Client
1. invoke aService.foo (someBar) defined by AIDL
17. return someResult
16. unmarshal someResult from replyParcel
Service.Stub.Proxy
2. marshal someBar to dataParcel
3. transact(aService, TRANSACTION_foo, dataParcel, replyParcel, ...)
15. update the reply parcel
libbinder
4. submit transaction/data via a blocking ioctl call
14. wake up from a blocking ioctl call and get transaction reply data
Service Process
-2. create a service
ServiceImpl
Main
9. invoke this.foo(someBar) defined by AIDL
8. unmarshal someBar from dataParcel
10. return someResult
Service.Stub
7. handle transaction onTransact (TRANSACTION_foo, dataParcel, replyParcel, ...)
11. marshal someResult to replyParcel
12. return
libbinder
-4. start a pool of threads each waiting for binder requests
-3. wait for binder driver to respond via blocking ioctl
6. get the binder object corresponding to the transaction
-5. wake up from a blocking ioctl call and get the transaction data
13. submit replyParcel via a ioctl
Kernel
/dev/binder

Binder

binder.h

```
struct binder_write_read {
    signed long write_size;
    signed long write_consumed;
    unsigned long write_buffer;
    signed long read_size;
    signed long read_consumed;
    unsigned long read_buffer;
};
```

IPCThreadState.cpp

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

binder.c

```
switch (cmd) {
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;
    }
    binder_debug(BINDER_DEBUG_READ_WRITE,
```

Service Manager

Service Manager - token == 0 ()

Клиент:

- получает ContextManager
- Запрашивает token для интересующего сервиса

Сервисы:

- регистрируются в ContextManager-е через addService

service_manager.c

```
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)) {
        ALOGE("cannot become context manager (%s)\n", strerror
(errno));
        return -1;
    }

    svcmgr_handle = svcmgr;
    binder_loop(bs, svcmgr_handler);
    return 0;
}
```

Service Manager

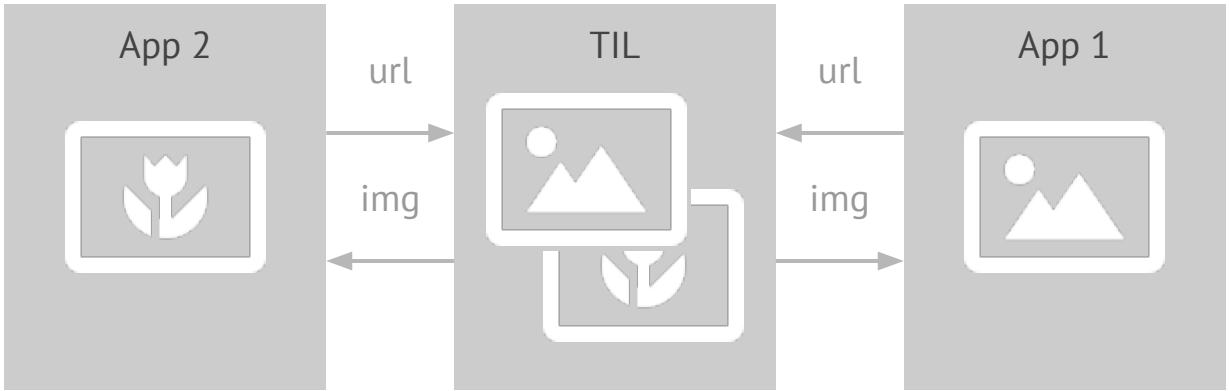
ActivityManagerService

```
public static void setSystemProcess() {
  try {
    ActivityManagerService m = mSelf;
    ServiceManager.addService("activity", m, true);
    ServiceManager.addService("meminfo", new MemBinder(m));
    ServiceManager.addService("gfxinfo", new GraphicsBinder(m));
    ServiceManager.addService("dbinfo", new DbBinder(m));
    if (MONITOR_CPU_USAGE) {
      ServiceManager.addService("cpuinfo", new CpuBinder(m));
    }

    ServiceManager.addService("permission", new PermissionController(m));

    ApplicationInfo info =
      mSelf.mContext.getPackageManager().getApplicationInfo(
        "android", STOCK_PM_FLAGS);
```

ServiceManagerProxy (в ServiceManagerNative)

```
public IBinder getService(String name) throws RemoteException {
  Parcel data = Parcel.obtain();
  Parcel reply = Parcel.obtain();
  data.writeInterfaceToken(IServiceManager.descriptor);
  data.writeString(name);
  mRemote.transact(GET_SERVICE_TRANSACTION, data, reply, 0);
  IBinder binder = reply.readStrongBinder();
  reply.recycle();
  data.recycle();
  return binder;
}
```

Tough Image Loader



Немного о Serializable

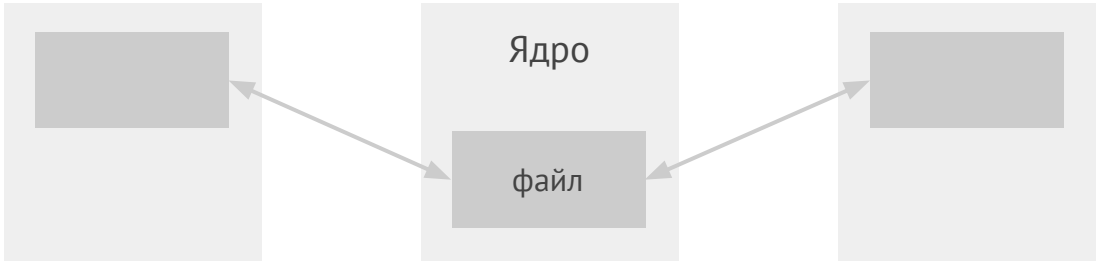
- Его нет в Си
- Parcel не содержит “лишних” данных

Пара других особенностей Binder

- Размер транзакции — 1Мб
- Количество потоков — 15
- Не подходит для потоковой передачи данных

1. ВЗАИМОДЕЙСТВИЕ ПРОЦЕССОВ
2. НЕМНОГО О ПАМЯТИ
3. ПЕРВЫЕ 30 СЕКУНД ЖИЗНИ
4. ЭНЕРГОПОТРЕБЛЕНИЕ

AshMem




```
int fd = ashmem_create_region("SharedRegionName", size);
address = mmap(NULL, size, PROT_READ PROT_WRITE, MAP_SHARED, fd, 0); O
```

- свобождается при закрытии всех процессов
- Page pinning

Low Memory Killer

Обычно (mm/oom_kill.c#select_bad_process, badness):

Выбор наилучшего кандидата:

- Наибольшее использование памяти
- Меньшее время жизни
- Количество дочерних процессов
- Поблажки корневым процессам
- Пользовательский коэффициент (oomadj)

Low Memory Killer

- Не обязательно дожидаться нехватки памяти
- Другие критерии оценки “важности” (видимость пользователю, ...)

lowmemorykiller.c

Приоритеты от -17 до 15 (ProcessList.java)

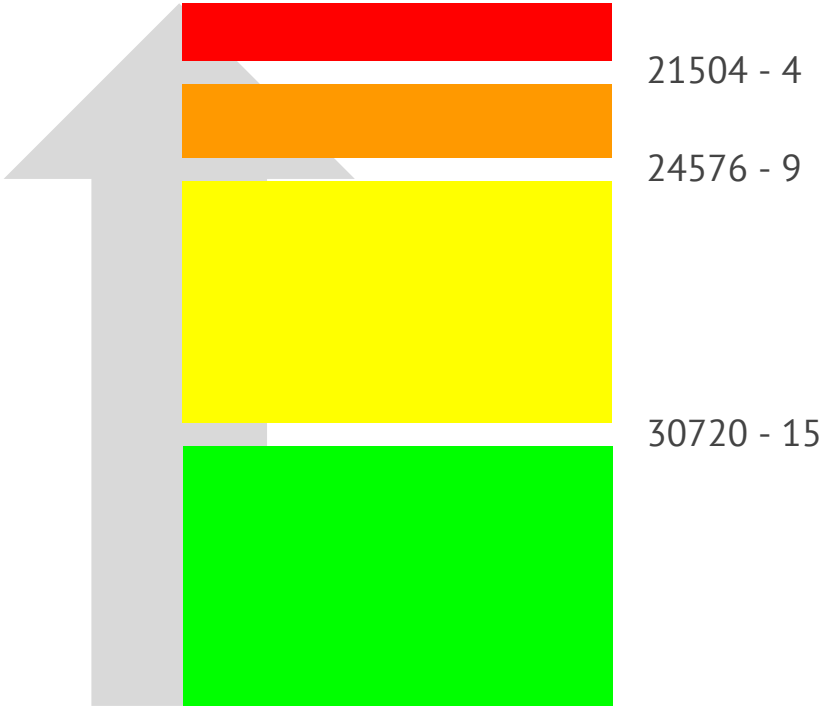
/sys/module/lowmemorykiller/parameters/adj
0,1,2,4,9,15

/sys/module/lowmemorykiller/parameters/minfree
12288,15360,18432,21504,24576,30720

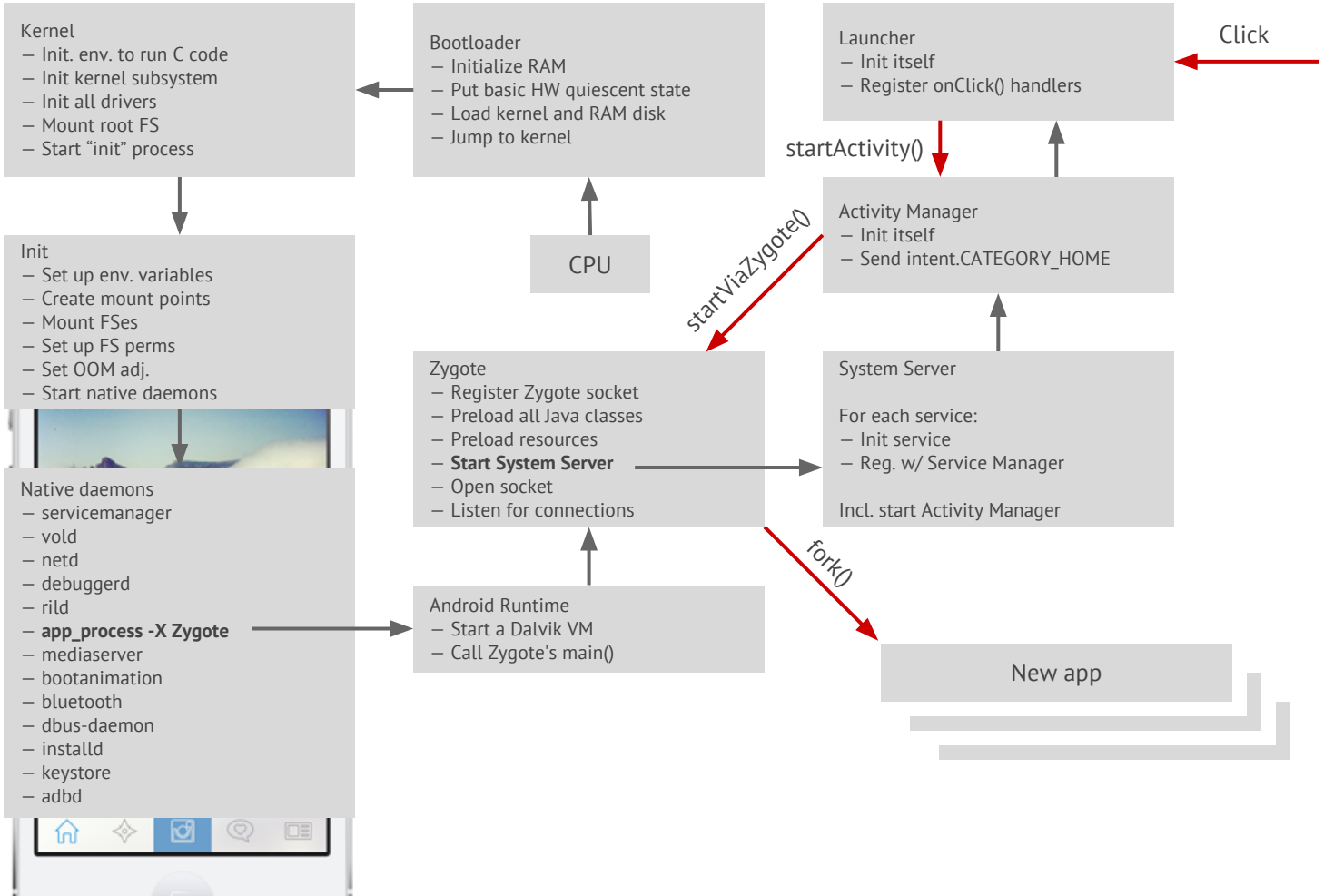
ProcessList — задаёт значения в зависимости от разрешение экрана и количества ОЗУ устройства (при старте ActivityManagerService)

HOME_APP_ADJ = 6

lowmemorykiller.c



1. ВЗАИМОДЕЙСТВИЕ ПРОЦЕССОВ
2. НЕМНОГО О ПАМЯТИ
3. ПЕРВЫЕ 30 СЕКУНД ЖИЗНИ
4. ЭНЕРГОПОТРЕБЛЕНИЕ


Kernel
– Init. env. to run C code
– Init kernel subsystem
– Init all drivers
– Mount root FS
– Start “init” process
Bootloader
– Initialize RAM
– Put basic HW quiescent state
– Load kernel and RAM disk
– Jump to kernel
Launcher
– Init itself
– Register onClick() handlers
Click
startActivity()
Activity Manager
– Init itself
– Send intent.CATEGORY_HOME
CPU
startViaZygote()
Init
– Set up env. variables
– Create mount points
– Mount FSes
– Set up FS perms
– Set OOM adj.
– Start native daemons
Zygote
– Register Zygote socket
– Preload all Java classes
– Preload resources
– Start System Server
– Open socket
– Listen for connections
System Server
For each service:
– Init service
– Reg. w/ Service Manager
Incl. start Activity Manager
Native daemons
– servicemanager
– vold
– netd
– debuggerd
– rild
– app_process -X Zygote
– mediaserver
– bootanimation
– bluetooth
– dbus-daemon
– installd
– keystore
– adbd
Android Runtime
– Start a Dalvik VM
– Call Zygote's main()
fork()
New app

1. ВЗАИМОДЕЙСТВИЕ ПРОЦЕССОВ
2. НЕМНОГО О ПАМЯТИ
3. ПЕРВЫЕ 30 СЕКУНД ЖИЗНИ
4. ЭНЕРГОПОТРЕБЛЕНИЕ

Wake Locks


★★★★★
Galaxy Note II (t03g)
looks to me like some people hacked up some ad-hoc trick for their own local need
★★★★★
Версия: 3.6
Galaxy S4 (ja3g)
2 000
hink that this wakelock stuff is in “can't be used properly” area on Rusty's scale of nasty interfaces
★★★★★
Версия: 3.6
Galaxy S3 (m0)
fayzulla akbarov
Исправьте свои ошибки.Это фуфло а не приложение разработчики все хуже вы чем занимаетесь? После осмотра видео приходиться выйти с приложение!!!??

Управление питанием

- ~~Earlysuspend~~

  градации состояний сна и работы
- Autosleep

  Пытаемся уснуть, если нет Wake Lock-ов.
- PowerManagerService

Литература

1. Karim Yagbmour “Embedded Android”
2. Alexandar Gargenta “Deep Dive into Android IPC/Binder Framework”
3. lkml.org
4. elinux.org
5. lwn.net
6. anatomyofandroid.com
7. kernel.org

Спасибо! Свои мысли, вопросы и отзывы вы можете присылать мне на alexey.nikitin@corp.mail.ru