# Android 内核以及 Android 源码的编译和 Binder 实验

@vonnyfly 冯力 lifeng1519@gmail.com

2013-11-01

## Contents



# 1 目标

1. 了解 Android 内核

2. 熟悉 Android 内核的编译
3. 熟悉 Android 源码的编译
4. 学习 Android 的进程间通信机制 ——Binder
5. 学习 native service 的创建

# 2 Android 内核

Android 是基于 Linux 内核的操作系统。虽然 Android 基于 Linux 内核，但是它与 Linux 之间还是有很大的差别，比如 Android 在 Linux 内核的基础上添加了自己所特有的驱动程序。

## 2.1 Android 为什么会选择 Linux

成熟的操作系统有很多，但是 Android 为什么选择采用 Linux 内核呢? 这就与 Linux 的一些特性有关了，这也是很多教材反复讲到的 linux 的重要特点。比如:

1. 强大的内存管理和进程管理方案
2. 基于权限的安全模式
3. 支持共享库
4. 经过认证的驱动模型
5. Linux 本身就是开源项目

更多关于上述特性的信息可以参考 Linux 2.6 版内核的官方文档，这便于我们在后面的学习中更好地理解 Android 所特有的功能特性。

## 2.2 Android 对 Linux 的改动

Android 对 linux 系统的改动主要有以下几个方面:

1. 它没有 glibc 支持由于 Android 最初用于一些便携的移动设备上，出于效率等方面的考虑，Android 并没有采用 glibc 作为 C 库，而是 Google 自己开发了一套 Bionic Libc 来代替 glibc。

2. 它并不包括一整套标准的 Linux 使用程序 Android 并没有完全照搬 Liunx 系统的内核，除了修正部分 Liunx 的 Bug 之外，还增加了不少内容，比如：它基于 ARM 构架增加的 Gold-Fish 平台，以及 yaffs2 FLASH 文件系统（如果学习了嵌入式的话就会知道 yaffs2 FLASH 文件系统已经在基于 linux 的很多嵌入式设备上采用了，技术已经非常成熟）等。

3. 它没有本地基于 X 服务的窗口系统什么是本地窗口系统呢? 本地窗口系统是指 GNU/Linux 上的 X 窗口系统，或者 Mac OX X 的 Quartz 等。不同的操作系统的窗口系统可能不一样，Android 并没有使用 (也不需要使用)Linux 的 X 窗口系统。

4. Android 专有的驱动程序除了上面这些不同点之外，Android 还对 Linux 设备驱动进行了增强，主要如下所示：

- Android Binder 基于 OpenBinder 框架的一个驱动，用于提供 Android 平台的进程间通信 (`InterProcess Communication, IPC`) 功能。源代码位于 `drivers/staging/android/binder.c`。
- Android 电源管理 (PM) 一个基于标准 Linux 电源管理系统的轻量级 Android 电源管理驱动，针对嵌入式设备做了很多优化。源代码位于：

  kernel/power/earlysuspend.c kernel/power/consoleearlysuspend.c kernel/power/fbearlysuspend.c kernel/power/wakelock.c kernel/power/userwakelock.c
- 低内存管理器 (Low Memory Killer) 比 Linux 的标准的 OOM(Out Of Memory) 机制更加灵活，它可以根据需要杀死进程以释放需要的内存。源代码位于 drivers/staging/ android/lowmemorykiller.c。
- 匿名共享内存 (Ashmem) 为进程间提供大块共享内存，同时为内核提供回收和管理这个内存的机制。源代码位于 mm/ashmem.c。
- Android Logger 一个轻量级的日志设备，用于抓取 Android 系统的各种日志。源代码位于 drivers/staging/android/logger.c。
- Android Alarm 提供了一个定时器，用于把设备从睡眠状态唤醒，同时它还提供了一个即使在设备睡眠时也会运行的时钟基准。源代码位于 drivers/rtc/alarm.c。
- USB Gadget 驱动一个基于标准 Linux USB gadget 驱动框架的设备驱动，Android 的 USB 驱动是基于 gaeget 框架的。源代码位于 drivers/usb/gadget/。

- Android Ram Console 为了提供调试功能，Android 允许将调试日志信息写入一个被称为 RAM Console 的设备里，它是一个基于 RAM 的 Buffer。源代码位于 drivers/staging/android / ram_console.c。
- Android timed device 提供了对设备进行定时控制的功能，目前支持 vibrator 和 LED 设备。源代码位于 drivers/staging/android /timed_output.c(timed_gpio.c)。
- Yaffs2 文件系统 Android 采用 Yaffs2 作为 MTD nand flash 文件系统，源代码位于 fs/yaffs2/目录下。Yaffs2 是一个快速稳定的应用于 NAND 和 NOR Flash 的跨平台的嵌入式设备文件系统，同其他 Flash 文件系统相比，Yaffs2 能使用更小的内存来保存其运行状态，因此它占用内存小。Yaffs2 的垃圾回收非常简单而且快速，因此能表现出更好的性能。Yaffs2 在大容量的 NAND Flash 上的性能表现尤为突出，非常适合大容量的 Flash 存储。

## 2.3 内核编译与运行

以下假设源码位于/home/whoami/src 目录下：1. 安装基本包：* 方式一: 在线安装

```
$ sudo apt-get -y install git gnupg flex bison gperf build-essential \
  zip curl libc6-dev libncurses5-dev:i386 x11proto-core-dev \
  libx11-dev:i386 libreadline6-dev:i386 libgl1-mesa-glx:i386 \
  libgl1-mesa-dev g++-multilib mingw32 tofrodos \
  python-markdown libxml2-utils xsltproc zlib1g-dev:i386 \
  gcc-4.4 g++-4.4 gcc-4.4-multilib g++-4.4-multilib
```

- 方式二: 离线安装所有的包已放在 archives 目录下

```
$ cd ~/src/archives/
sudo dpkg -i *.deb
```

2. 降低 gcc g++ 版本, 以及其它的符号链接修改:

```
$ sudo ln -s /usr/lib/i386-linux-gnu/mesa/libGL.so.1 /usr/lib/i386-linux-gnu/libGL.so
$ rm /usr/bin/{gcc,g++}
$ ln -s /usr/bin/{gcc-4.4,gcc}
$ ln -s /usr/bin/g++-{4.4,}
```

3. jdk 的安装

```
$ cd ~/src
$ ./jdk-6u25-linux-x64
$ sudo mkdir -p /usr/lib/jvm/java/
$ sudo mv jdk-6u25 /usr/lib/jvm/java/jdk1.6
```

4. 修改环境变量

- 方式一: 临时修改

```
# 添加交叉编译工具链到 PATH
$ cd ~/src/android-4.0.4/prebuilt/linux-x86/toolchain/arm-eabi-4.4.3/bin
$ export PATH=$(pwd):$PATH
$ cd -
```

- 方式二: 永久修改将 export 命令放到 bash 的配置文件里, 例如使用的 bash shell 的话对应的配置文件就是 ~/.bashrc, 使用的是 zsh shell 的话, 那么对应的配置文件就是 ~/.zshrc.

增加一下几行到 ~/.bashrc 里:

```
export ANDROID_SRC=~/src/android-4.0.4
export ANDROID_PRODUCT_OUT=$ANDROID_SRC/out/target/product/generic

#toolchain , prefix arm-eabi-
export PATH=$ANDROID_SRC/prebuilt/linux-x86/toolchain/arm-eabi-4.4.3/bin:$PATH

#android sdk
export ANDROID_HOME=~/src/android-sdk-linux
```

```
export PATH=$ANDROID_HOME/tools:$ANDROID_HOME/platform-tools:$PATH

#android ndk
export PATH=~/src/android-ndk:$PATH

#java
JAVA_HOME="/usr/lib/jvm/java/jdk1.6"
CLASSPATH=".:$JAVA_HOME/bin/lib"
export PATH=$JAVA_HOME/bin:$PATH
```

5. 编译

```
#!/bin/bash
#Author: @vonnyfly 冯力 ,lifeng1519@gmail.com
#    Liuyuxiao ,liuyuxiao123@gmail.com
#Todo: chmod a+x ./build.sh
#Run:   ./build.sh

# 添加交叉编译工具链到 PATH
export ANDROID_SRC=~/src/android-4.0.4
export ANDROID_PRODUCT_OUT=$ANDROID_SRC/out/target/product/generic
#toolchain , prefix arm-eabi-
export PATH=$ANDROID_SRC/prebuilt/linux-x86/toolchain/arm-eabi-4.4.3/bin:$PATH

# 添加 ARCH 等相关环境变量, 并编译
cd ~/src/goldfish
export ARCH=arm
export SUBARCH=arm
export CROSS_COMPILE=arm-eabi-
make clean
make goldfish_armv7_defconfig
make -j4

#emulator &
emulator -kernel ~/src/goldfish/arch/arm/boot/zImage 2>1 1>/dev/null &
```

```
sleep 20
adb shell cat /proc/version
```

# 3 Android IPC——Binder

Android 是一个包含内核、中间件、关键应用的移动设备操作系统。由于 Andriod 系统构建在 Linux 内核之上，那么理论上 Linux 本身的 IPC 方式是支持的，不过 Bionic 并没有完全实现它们。Bionic 实现了的 IPC 包括：POSIX 共享内存、POSIX 信号量、管道、Socket，但是 SYSTEM V IPC 及 POSIX 消息队列并不支持，因为它们可能产生全局内核资源泄漏。同时，传统的 IPC 会增加进程的开销并导致安全漏洞]。因此为了适应嵌入式设备，Android 系统增加了一个新的 IPC 通信方式 ——Binder。

## 3.1 Binder 用户层的原理与实现

Binder 涉及到 kernel 驱动、框架层和应用层，是整个 Android 系统的核心。守护进程 Service Manager 管理系统中的 Service（也即 Binder 实体），负责进程间的数据交换。各进程通过 Binder 访问同一块共享内存, 以达到数据通信的机制。从应用层的角度看，进程通过访问数据守护进程获取用于数据交换的程序框架接口，调用并通过接口共享数据。而其他进程要访问数据，也只需与程序框架接口进行交互，简化了应用开发。一次使用 Service 的完整通信过程如图所示：

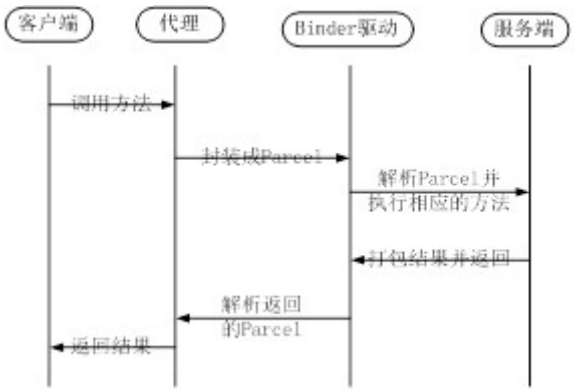


Figure 1: 一个完整的 Binder 通信过程

Android Binder 基于 OpenBinder 来实现的，它基于 C/S 通信模式。在用户空间，提供服务的进程叫做 Server 进程，具体提供服务的组件叫做 Service 组件。请求服务的进程叫做 Client 进程。一个 Server 进程可以同时运行多个 Service 来对外提供服务，一个 cilent 也可以同时发动多个请求来访问服务。Server 和 Client 通过 Binder 虚拟设备驱动来进行数据传递达到交换数据的目的。Binder 驱动程序提供了标准文件访问接口（设备文件/dev/binder），使得用户可以使用 Posix 文件 API(open/ioctl) 来访问设备。同时 Binder 通过进程 Service Manager 来管理系统所有的 Service，使得 Client 能够发现 Service 的存在，并向它发出请求成为可能。当 Server 的 Service 组件在启动的时候，会主动的向 ServiceManager 注册自己，然后，Client 就可以通过 Service 的名字向 ServiceMananger 查询特定 Service 的存在，获取 Service 的一个唯一引用，以便将来向 Service 发出请求。因此，ServiceManager 作为一个 Service 管理者的同时，也扮演了一个 Service 的功能。同时一个 Service 在启动之初也作为一个 Client 向 ServiceManager 请求 “注册” 服务。

Service、Client 和 Service manager 的关系如图所示：

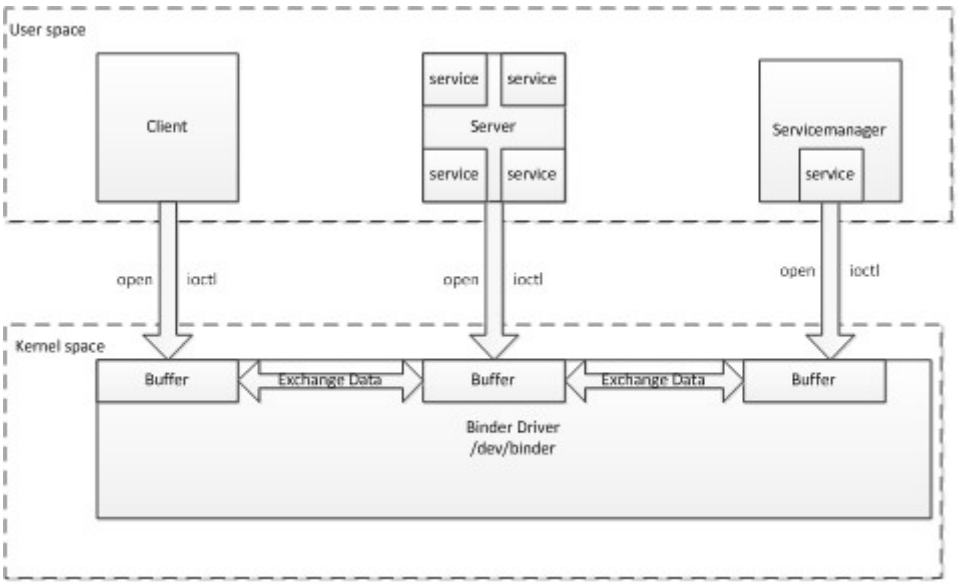


Figure 2: Service 、 Client、 ServiceManager 和驱动的关系

Binder 在用户空间（Framework 层）提供了一个 libbinder 动态链接库给系统其它进程使用。在架构上,Binder 采用了代理设计模式, 任何与 Service 进行通信的部件都有一个 Service 代理。正由于代理的存在，使得 Client 产生一个错觉，Service 似乎在本机上运行。这种 “透明” 代理的效果使得开发更加

清晰。开发者如果先定义一个 Interface，然后在 Service 端，实现这个接口，也就是 BnInterface 类，实现功能，同时实现 BpInterface，将数据打包通过驱动传递给 Service 来调用 Service 的实现。

在上层 Java 应用通过 Jni 调用 Service 之后，实际的调用流程如图所示。用户层的 Binder 基于 C++ 的模板技巧以及继承来完成 Service 接口的定义以及实现，良好的架构使得编写 Service 非常容易。

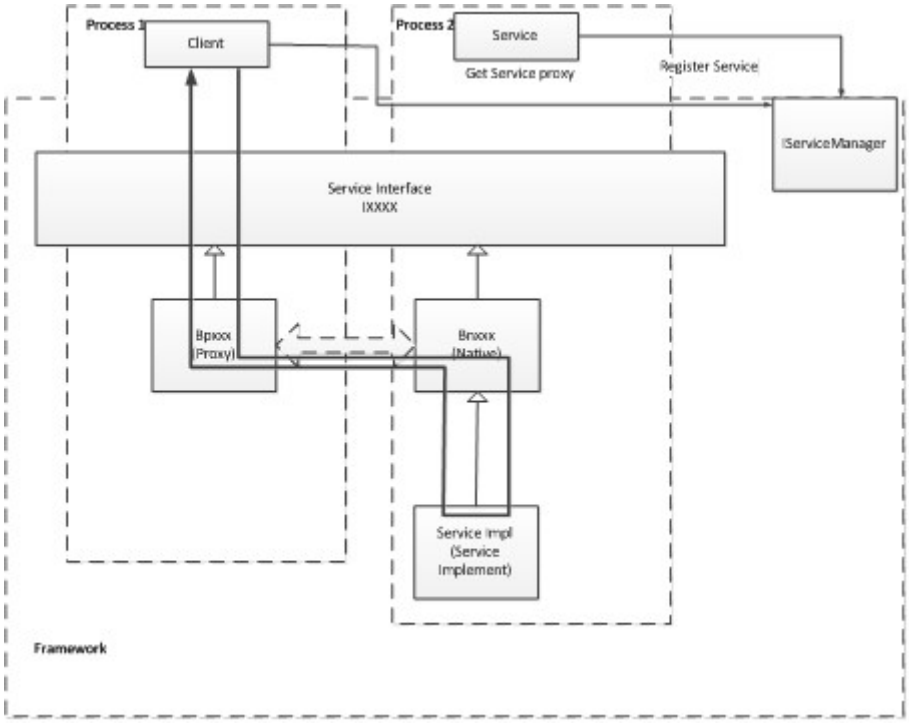


Figure 3: Binder 调用流

## 3.2 Binder 驱动层的原理及内存模型

由于 Binder 驱动是一个字符设备驱动，因此提供了设备文件的读写能力。同时为了操作的统一，使用 ioctl 作为统一的操作入口。使用不同的命令，来实现不同的调用。在 Binder 驱动的实现中，我们最感兴趣的是驱动内存模型。在 Binder 的通信中，数据不是等长，也就是说不能提前预测，同时还要跟用户空间进行共享，因而对缓冲区的管理的设计非常重要。Binder 在接收缓冲区内存管理上采用一次映射，按需分配，精细管理的方式。一

次映射。在用户空间打开 Binder 设备后，立即进行的操作是调用 mmap 函数。在驱动的实现就是在内核空间的 VMALLOC 区映射一块虚拟地址到该进程的虚拟用户空间。具体如图所示。

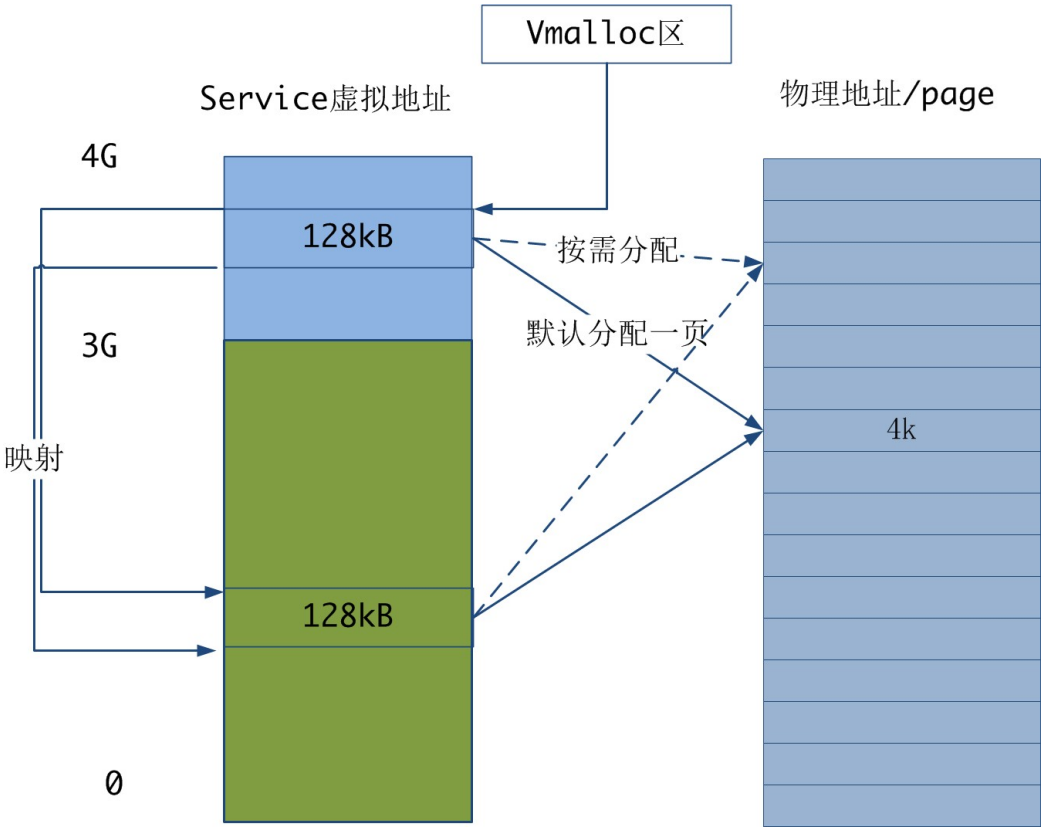


Figure 4: Binder 内存映射

- 按需分配。在映射完一块区域后，Binder 并没有立即分配所有的物理页面，而是只分配一页。当这一页用完之后，会触发 pagefault，然后来分配页面，再和页表建立关联。
- 精细管理。当一个 Binder 通信进入到驱动之后，驱动根据数据负载的大小，从该进程的虚拟内存区分割一块相应大小的区域，然后将其添加到已分配的链表里来进行跟踪，剩下的继续放入未分配链表。同理，一次通信结束后，相应的区域要被释放，重新放回未分配链表，同时要进行相应的合并等操作。

现在注意看数据的流动。如图所示。Binder 通信由用户空间发起，第一步要将数据从用户空间拷贝到内核空间。为了提高效率，Binder 直接将数据拷贝到目标进程的接收缓冲区中。由于采用页面映射的原理，此时，目标进程的用户空间可以看到这个数据。Binder 一步到位，实现了数据的传递。但是，为了实现这个目的，Binder 必须付出巨大的妥协，那就是全局锁问题。我们在下一节讲到。

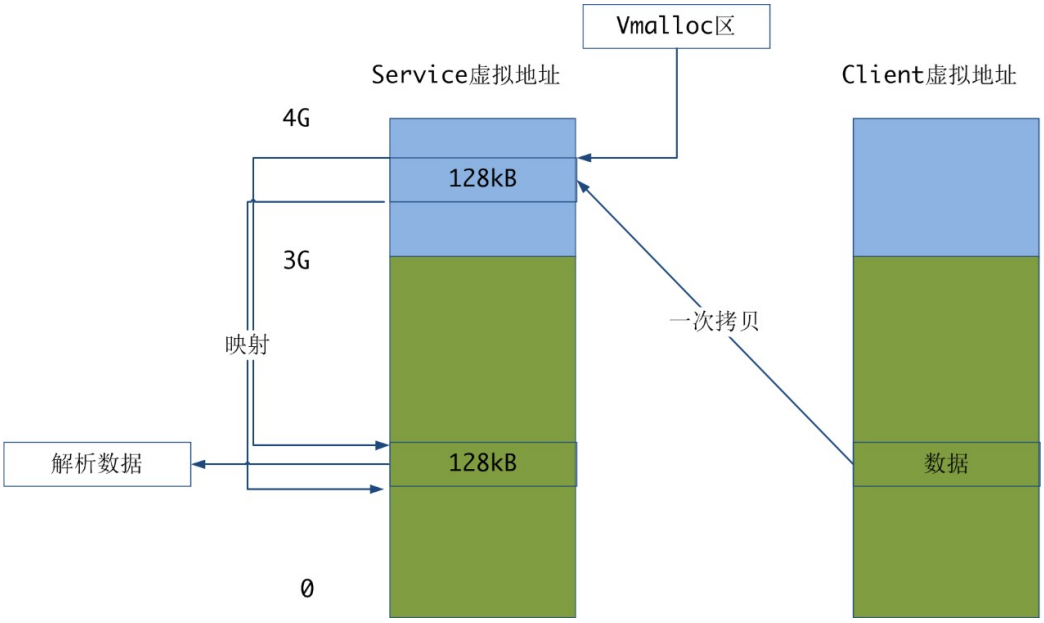


Figure 5: Binder 数据传递

## 3.3 Binder 存在的问题

Binder 使用了复杂的内存管理方式实现了一次数据拷贝，但是为了达到这个目的，也必须做出性能上的妥协。具体说来，由于涉及到跨进程的空间操作，使得操作的过程中几乎不能被打断，这些操作只能串行执行，发挥不了多核的能力。Binder 驱动存在以下几个并发问题：

1. 构造 transaction 对象。分配目标进程的内存涉及多并发。
2. 拷贝数据到目标进程。访问目标进程内存涉及多并发
3. 修改目标进程数据（解析 flat binder）。访问目标进程内存、查找引用和节点操作涉及多并发。（此时 fp->binder 返回引用号）为了处理并

发带来的副作用，Binder 驱动粗鲁的使用一个 binder 全局锁来做到互斥。从代码来看，大部分操作都是在获取 binder 全局锁下进行的。

## 3.4 Binder 使用实例

由于 Android 的 Binder 最底层需要访问到内部未导出的类，所以 ndk 编译不过去以下实例，需要放到内核源码里编译。

以下例子用来测试 Binder 的性能。

binderAddInts.cpp 文件

```
#include <cerrno>
#include <grp.h>
#include <iostream>
#include <libgen.h>
#include <time.h>
#include <unistd.h>

#include <sys/syscall.h>
#include <sys/time.h>
#include <sys/types.h>
#include <sys/wait.h>

//#include <binder/IPCThreadState.h>
//#include <binder/ProcessState.h>
//#include <binder/IServiceManager.h>

#include ”include/IPCThreadState.h”
#include ”include/ProcessState.h”
#include ”include/IServiceManager.h”
#include <utils/Log.h>
#include <testUtil.h>

using namespace android;
using namespace std;

const int unbound = -1; // Indicator for a thread not bound to a specific CPU
```

```
String16 serviceName("test.binderAddInts");

struct options {
    int serverCPU;
    int clientCPU;
    unsigned int iterations;
    float        iterDelay; // End of iteration delay in seconds
} options = { // Set defaults
    unbound, // Server CPU
    unbound, // Client CPU
    1000,    // Iterations
    1e-3,    // End of iteration delay
};

class AddIntsService : public BBinder
{
  public:
    AddIntsService(int cpu = unbound);
    virtual ~AddIntsService() {};

    enum command {
        ADD_INTS = 0x120,
    };

    virtual status_t onTransact(uint32_t code,
                                const Parcel& data, Parcel* reply,
                                uint32_t flags = 0);

  private:
    int cpu_;
};

// File scope function prototypes
static void server(void);
static void client(void);
static void bindCPU(unsigned int cpu);
static ostream &operator<<(ostream &stream, const String16& str);
```

```
static ostream &operator<<(ostream &stream, const cpu_set_t& set);

int main(int argc, char *argv[])
{
    int rv;

    // Determine CPUs available for use.
    // This testcase limits its self to using CPUs that were
    // available at the start of the benchmark.
    cpu_set_t availCPUs;
   if ((rv = sched_getaffinity(0, sizeof(availCPUs), &availCPUs)) != 0) {
        cerr << "sched_getaffinity failure, rv: " << rv
            << " errno: " << errno << endl;
        exit(1);
    }

    // Parse command line arguments
    int opt;
    while ((opt = getopt(argc, argv, "s:c:n:d:?")) != -1) {
        char *chptr; // character pointer for command-line parsing

        switch (opt) {
        case 'c': // client CPU
        case 's': { // server CPU
            // Parse the CPU number
            int cpu = strtoul(optarg, &chptr, 10);
            if (*chptr != '\0') {
                cerr << "Invalid cpu specified for -" << (char) opt
                    << " option of: " << optarg << endl;
                exit(2);
            }

            // Is the CPU available?
            if (!CPU_ISSET(cpu, &availCPUs)) {
           cerr << "CPU " << optarg << " not currently available" << endl;
                cerr << "  Available CPUs: " << availCPUs << endl;
                exit(3);
            }
```

```
            // Record the choice
        *((opt == 'c') ? &options.clientCPU : &options.serverCPU) = cpu;
            break;
        }

        case 'n': // iterations
            options.iterations = strtoul(optarg, &chptr, 10);
            if (*chptr != '\0') {
           cerr << "Invalid iterations specified of: " << optarg << endl;
                exit(4);
            }
            if (options.iterations < 1) {
                cerr << "Less than 1 iteration specified by: "
                    << optarg << endl;
                exit(5);
            }
            break;

        case 'd': // Delay between each iteration
            options.iterDelay = strtod(optarg, &chptr);
            if ((*chptr != '\0') || (options.iterDelay < 0.0)) {
             cerr << "Invalid delay specified of: " << optarg << endl;
                exit(6);
            }
            break;

        case '?':
        default:
            cerr << basename(argv[0]) << " [options]" << endl;
            cerr << "  options:" << endl;
            cerr << "    -s cpu - server CPU number" << endl;
            cerr << "    -c cpu - client CPU number" << endl;
            cerr << "    -n num - iterations" << endl;
        cerr << "   -d time - delay after operation in seconds" << endl;
            exit(((optopt == 0) || (optopt == '?')) ? 0 : 7);
        }
    }
```

```
// Display selected options
cout << "serverCPU: ";
if (options.serverCPU == unbound) {
    cout << " unbound";
} else {
    cout << options.serverCPU;
}
cout << endl;
cout << "clientCPU: ";
if (options.clientCPU == unbound) {
    cout << " unbound";
} else {
    cout << options.clientCPU;
}
cout << endl;
cout << "iterations: " << options.iterations << endl;
cout << "iterDelay: " << options.iterDelay << endl;

// Fork client, use this process as server
fflush(stdout);
switch (pid_t pid = fork()) {
case 0: // Child
    client();
    return 0;

default: // Parent
    server();

    // Wait for all children to end
    do {
        int stat;
        rv = wait(&stat);
        if ((rv == -1) && (errno == ECHILD)) { break; }
        if (rv == -1) {
            cerr << "wait failed, rv: " << rv << " errno: "
                << errno << endl;
            perror(NULL);
```

```
                exit(8);
            }
        } while (1);
        return 0;

    case -1: // Error
        exit(9);
    }

    return 0;
}

static void server(void)
{
    int rv;

    // Add the service
    sp<ProcessState> proc(ProcessState::self());
    sp<IServiceManager> sm = defaultServiceManager();
    if ((rv = sm->addService(serviceName,
        new AddIntsService(options.serverCPU))) != 0) {
        cerr << "addService " << serviceName << " failed, rv: " << rv
            << " errno: " << errno << endl;
    }

    // Start threads to handle server work
    proc->startThreadPool();
}

static void client(void)
{
    int rv;
    sp<IServiceManager> sm = defaultServiceManager();
   double min = FLT_MAX, max = 0.0, total = 0.0; // Time in seconds for all
                                                  // the IPC calls.

    // If needed bind to client CPU
    if (options.clientCPU != unbound) { bindCPU(options.clientCPU); }
```

```
// Attach to service
sp<IBinder> binder;
do {
    binder = sm->getService(serviceName);
    if (binder != 0) break;
    cout << serviceName << ” not published, waiting...” << endl;
    usleep(500000); // 0.5 s
} while(true);

// Perform the IPC operations
for (unsigned int iter = 0; iter < options.iterations; iter++) {
    Parcel send, reply;

    // Create parcel to be sent.  Will use the iteration cound
    // and the iteration count + 3 as the two integer values
    // to be sent.
    int val1 = iter;
    int val2 = iter + 3;
    int expected = val1 + val2;  // Expect to get the sum back
    send.writeInt32(val1);
    send.writeInt32(val2);

    // Send the parcel, while timing how long it takes for
    // the answer to return.
    struct timespec start;
    clock_gettime(CLOCK_MONOTONIC, &start);
    if ((rv = binder->transact(AddIntsService::ADD_INTS,
        send, &reply)) != 0) {
        cerr << ”binder->transact failed, rv: ” << rv
            << ” errno: ” << errno << endl;
        exit(10);
    }
    struct timespec current;
    clock_gettime(CLOCK_MONOTONIC, &current);

   // Calculate how long this operation took and update the stats
    struct timespec deltaTimespec = tsDelta(&start, &current);
```

```
        double delta = ts2double(&deltaTimespec);
        min = (delta < min) ? delta : min;
        max = (delta > max) ? delta : max;
        total += delta;
        int result = reply.readInt32();
        if (result != (int) (iter + iter + 3)) {
           cerr << "Unexpected result for iteration " << iter << endl;
            cerr << "  result: " << result << endl;
            cerr << "expected: " << expected << endl;
        }

     if (options.iterDelay > 0.0) { testDelaySpin(options.iterDelay); }
    }

    // Display the results
    cout << "Time per iteration min: " << min
        << " avg: " << (total / options.iterations)
        << " max: " << max
        << endl;
}

AddIntsService::AddIntsService(int cpu): cpu_(cpu) {
    if (cpu != unbound) { bindCPU(cpu); }
};

// Server function that handles parcels received from the client
status_t AddIntsService::onTransact(uint32_t code, const Parcel &data,
                                    Parcel* reply, uint32_t flags) {
    int val1, val2;
    status_t rv(0);
    int cpu;

    // If server bound to a particular CPU, check that
    // were executing on that CPU.
    if (cpu_ != unbound) {
        cpu = sched_getcpu();
        if (cpu != cpu_) {
          cerr << "server onTransact on CPU " << cpu << " expected CPU "
```

```
                << cpu_ << endl;
            exit(20);
        }
    }

    // Perform the requested operation
    switch (code) {
    case ADD_INTS:
        val1 = data.readInt32();
        val2 = data.readInt32();
        reply->writeInt32(val1 + val2);
        break;

    default:
     cerr << "server onTransact unknown code, code: " << code << endl;
      exit(21);
    }

    return rv;
}

static void bindCPU(unsigned int cpu)
{
    int rv;
    cpu_set_t cpuset;

    CPU_ZERO(&cpuset);
    CPU_SET(cpu, &cpuset);
    rv = sched_setaffinity(0, sizeof(cpuset), &cpuset);

    if (rv != 0) {
     cerr << "bindCPU failed, rv: " << rv << " errno: " << errno << endl;
        perror(NULL);
        exit(30);
    }
}

static ostream &operator<<(ostream &stream, const String16& str)
```

```
{
    for (unsigned int n1 = 0; n1 < str.size(); n1++) {
        if ((str[n1] > 0x20) && (str[n1] < 0x80)) {
            stream << (char) str[n1];
        } else {
            stream << '~';
        }
    }

    return stream;
}

static ostream &operator<<(ostream &stream, const cpu_set_t& set)
{
    for (unsigned int n1 = 0; n1 < CPU_SETSIZE; n1++) {
        if (CPU_ISSET(n1, &set)) {
            if (n1 != 0) { stream << ' '; }
            stream << n1;
        }
    }

    return stream;
}
```

对应的 Android.mk 文件

```
LOCAL_PATH:= $(call my-dir)

include $(CLEAR_VARS)
LOCAL_MODULE_TAGS := eng tests
LOCAL_MODULE_PATH := $(TARGET_OUT_DATA)/nativebenchmark

LOCAL_STATIC_LIBRARIES += \
    libgtest \
    libgtest_main \
    libtestUtil
```

```
LOCAL_SHARED_LIBRARIES += \
    libutils \
    libstlport \
    libbinder

LOCAL_C_INCLUDES += \
    bionic \
    bionic/libstdc++/include \
    external/stlport/stlport \
    external/gtest/include \
    system/extras/tests/include \
    frameworks/base/include

LOCAL_MODULE := binderAddInts
LOCAL_SRC_FILES := binderAddInts.cpp
include $(BUILD_EXECUTABLE)
```

将其所有的文件放到 android 源码目录下的 `external/binder_test/`下。

# 4 参考文献

1. http://source.android.com/
2. http://54.250.154.181:8000/