Android OS Lab1 - 系统调用

随着人们对智能手机的依赖逐渐加重，手机上敏感信息的保护也因此变得非常重要。学术和工业上近年来都对此有非常深入的探索，例如 2011 年 SOSP 会议上的这篇论文实现了一个 virtual phone 的概念。我们这次 Lab 的目标是实现其中一部分内容：通过限制进程间的访问权限来提高隐私保护能力。安卓系统中的进程之间某些不必要的相互唤醒，增加了系统的负担，降低了用户体验。因此，本 Lab 旨在通过修改内核，对安卓系统的进程通信进行一定的限制。

大概的思路是为每一个进程赋予一个颜色("color")，不同的颜色代表不同的运行环境(例如，work for home)。只允许相同颜色的进程之间互相通信。实现这个目标需要两个步骤：添加进程颜色和控制进程间通信。对于前者，我们要实现两个定制的系统调用，分别用来获取和修改指定进程的颜色；后者则需要我们修改 Android 中最重要的 IPC 机制：Binder。Binder 为 Android 应用提供了系统服务的接口，如 contact list, email, phone dialer 等。网络上有许多关于 Binder 的介绍，例如这里，这里和这里。

Part 1: 环境搭建

第一部分的工作主要是熟悉 Android Kernel 的编译环境。我们可以大致地将 Android 系统分为两部分：底层的 Linux Kernel 和上层的 Framework。Kernel 部分基本上与 PC 上的 Linux Kernel 相同，但针对 Android 平台做了一些定制；而 Framework 也就是我们常说的 Android Open Source Project (AOSP)，由 Google 开发和维护。**我们的 Lab 只需要在 Kernel 之中完成**。

虽然现在基本上所有的商业手机都是采用 arm 架构，但是 Google 还是提供了各种架构的源码。在这里，考虑到大部分的 PC 都使用 x86 体系结构，为了提高模拟器的运行效率，我们选择 64 位的 x86 来完成 Lab。如果想要在真机上做测试，则需要编译不同的 AOSP 和内核版本(代码都相同，但需要 checkout 不同的 branch)。

```
# Step 1: Make a folder for our course
mkdir AndroidCourse
cd AndroidCourse

# Step 2: Download emulator and cross-compiler tool
git clone https://aosp.tuna.tsinghua.edu.cn/platform/prebuilts/gcc/linux-x86/x86/x86_64-linux-android-4.9
git clone https://aosp.tuna.tsinghua.edu.cn/platform/prebuilts/android-emulator

# Step 3: Download three imgs from course website and put them under ./AndroidCourse/imgs

# Step 4: Download Kernel source
git clone https://aosp.tuna.tsinghua.edu.cn/kernel/goldfish.git
git checkout android-goldfish-3.10-m-dev

# Step 5: Build the kernel
export ARCH=x86_64 CROSS_COMPILE=../x86_64-linux-android-4.9/bin/x86_64-linux-android-
make ARCH=x86_64 x86_64_emu_defconfig
make CC="${CROSS_COMPILE}gcc -mno-android" bzImage -j8

# Step 6: Run emulator
./android-emulator/linux-x86_64/emulator64-x86 -kernel goldfish/arch/x86_64/boot/bzImage -system imgs/system.img -
ramdisk imgs/ramdisk.img -data imgs/userdata.img -sysdir ./imgs/ -show-kernel -verbose -memory 1024
```

总体的流程为如上六步，其中第二步和第四步我们用到了清华大学的 AOSP

镜像服务（Google 的在大陆不可访问）。第三部的 imgs 可以直接从我们的教学网上或者 ftp 服务器上下载。第五步中我们用到了交叉编译器。

Part 2: 添加系统调用

为 Android Kernel 添加两个系统调用，使得用户可以获取和修改进程的颜色。每一个进程都有自己的颜色，记录在 task_struct 结构体(include/linux/sched.h) 中，其默认值为 0，类型是 u_int16_t(include/linux/types.h)。系统调用的实现添加在 kernel/color.c 中，其结构如下：

```
0   /* nr_pids contains the number of entries in
1      the pids, colors, and the retval arrays. The colors array contains the
2      color to assign to each pid from the corresponding position of
3      the pids array. Returns 0 if all set color requests
4      succeed. Otherwise, The array retval contains per-request
5      error codes -EINVAL for an invalid pid, or 0 on success.
6    */
7   int setcolors(int nr_pids, pid_t *pids, u_int16_t *colors, int *retval);
8
9   /* Gets the colors of the processes
10     contained in the pids array. Returns 0 if all set color requests
11     succeed. Otherwise, an error code is returned. The array
12     retval contains per-request error codes: -EINVAL for an
13     invalid pid, or 0 on success.
14   */
15  int getcolors(int nr_pids, pid_t *pids, u_int16_t *colors, int *retval);
```

要求和提示：

- 虽然我们的 Lab 是基于 x86_64 的平台，但要求同学们的系统调用注册在多个架构上(arm, i386)。注意，不同架构上的系统调用号(System Call Number)是不同的，所以多用宏(#define)来描述调用号。
- 永远不要信任用户空间传入的参数，尤其是指针！Kernel 需要检查这些参数，防止因为恶意程序的调用而导致内核系统崩溃。阅读 copy_from_user()和 copy_to_user()两个函数的使用说明，如果发现参数不合法，系统调用应返回 EFAULT。
- 只有 root(or sudo)权限的用户才能修改进程的颜色，否则，返回-EACCES。这里可以通过 task_struct 获得进程的用户权限，也可以使用内核已经提供的函数如 current_uid()。
- 同一个进程下的所有线程都应该有同样的颜色（了解 Linux 中进程和线程的区别）。可以使用 task_struct 中的 tgid 来获取进程组信息。
- 颜色信息会通过 fork 和 clone 系统调用传递到子进程，但不包括 vfork。

在实现的时候可以参考其他系统调用例如 getpid。记得添加相应的 c 文件到 Makefile 中和 git repository 中。Kernel 提供了很多有用的方法和宏，例如使用 find_task_by_pid()可以通过进程号来找到该进程的 task_struct，在本次 Lab 中会非常有用。

Part 3: 修改 Binder

这个部分要求同学们修改 Android Binder，使得不同颜色的进程间无法通过 Binder 通信——除非他们中的某一个颜色为 0.

Binder 的主要实现代码在 drivers/staging/android/binder.c 和 drivers/staging/android/binder.h 中。系统的代码会非常繁琐，需要考虑到容错，兼容等等问题。我们不需要读懂 Binder 的所有代码，只需要理解其中的关键部分，也就是做权限检查的函数：binder_transaction。用户程序通过 IOCTL 调用这个函数，建立与其他进程的通信。关于 IOCTL 的部分，我们之后还会深入地学习。

权限检查的结果有两种：允许通信和禁止通信。如果为后者，系统应当返回相应的错误码，而不是简答地让程序崩溃，这可能会导致用户程序试图重复建立通信。仔细阅读 Binder 中的代码，可以看到很多其他系统错误是被如何正确处理的。

如何测试 Binder 的定制是否正确？可以自己手动编写程序，或者利用 Android 本身带有的一些通信进程。例如：给进程 com.android.calendar (calendar app)和 com.android.providers.calendar (calendar provider)赋予不同的颜色，可以导致日历程序无法获取当前日期而崩溃。

Part 4: 测试程序

这个部分要求同学们编写三个用户空间的程序，测试内核的修改是否正确。这些程序的格式都应是 Android 模拟器里的可执行程序。

- *setcolors* 通过命令行获取 2N 个参数，其中 2i+1 和 2i+2 个参数分别表示进程名和颜色。程序的执行结果是修改相应的进程颜色。例如执行 setcolors /system/bin/sh 1 com.android.email 2 com.android.phone 2 会将 system/bin/sh 的颜色修改为 1，其它两个进程类似地设置为 2。
- *getcolors* 类似地，通过命令行获取 N 个进程名作为参数，然后输出这些进程当前的颜色。例如执行 getcolors /system/bin/sh com.process.acore com.android.email 会输出这两个进程的颜色。
- *forktest* 的执行格式如下：

  forktest delay fork|vfork|clone cmdline。

  该程序首先读入参数，然后睡眠 delay 秒（考虑 libc 中的 sleep 函数）。唤醒之后，取决于输入参数，它会执行 fork(), vfork(), clone()中的一个，然后再通过 exec 执行剩下的命令行参数 cmdline。例如，在运行 forktest 10 vfork forktest 10 fork sleep 10，会首先睡眠 10 秒，然后执行 vfork，递归调用 forktest，再 10 秒之后 fork 出一个只睡眠 10 秒的进程。

和系统内核一样，用户程序也应该检查参数是否合法。例如在 setcolors 中检查输入的参数个数，输入的进程名是否存在，输入的颜色是否是正确的数字格式等等，并做相应的输出提示。