Lab02: Android/Linux Process Tree

NJU Fall 2018

截止日期：2018年10月30日23:59PM

Lab2的任务是对内核进行编程，增加一个新的系统调用ptree，并在Android设备模拟器和Linux虚拟机上使用该系统调用打印出指定的进程树信息。Lab2要求尝试Android和Linux平台并熟悉其开发环境。Android和Linux平台可以在许多不同的架构上运行，但我们将针对的特定平台是X86_64 CPU系列。

我们在提供的虚拟机中预先安装好了一个基于QEMU系统的Android模拟器。相应的Android SDK文件也已预先安装在虚拟机中。由于我们使用的是X86_64系列CPU，与通常PC中运行的CPU相同，所有的内核编译都可以在提供的Linux虚拟机中完成[1]。虚拟机用户名密码分别是nju和空格[2]。我们使用的内核版本是用于Android系统的android-goldfish-4.4-dev，可以在课程主页上下载[3]。Android是基于Linux内核的，只是在Linux主分支中添加了一些额外的附加功能，用上述Android的内核源码编译生成的内核同样适用Linux虚拟机或物理机（比如乙126的机器）。

1 实验要求

首先创建一个Android虚拟设备，然后编译新的4.4.124版本的Linux内核，并指定Android模拟设备从新编译的内核启动。

1.1 创建一个Android虚拟设备

我们在提供的虚拟机中已经预先创建了一个名为njuavd的Android虚拟设备(Android Virtual Device)。也可以自己使用android avd命令运行Android AVD管理器,创建一个Android虚拟设备，注意CPU/ABI需要选择X86_64。

设置AVD后，可以使用下述命令从终端启动它。

```
emulator @njuavd -show-kernel
```

需要注意的是，需要在虚拟机的虚拟化引擎设置中启用硬件加速才能启动Android AVD。AVD最多可能需要5分钟才能启动。如果AVD无法启动或

[1]当然你也可以使用你自己或乙126机房的Linux机器，需要自己安装Android模拟器

[2]虚拟机下载地址：http://210.28.133.11:21318/,下载Linux2018.ova后用VMware workstation等虚拟机软件即可打开

[3]内核下载地址是：http://cs.nju.edu.cn/zhangl/linux/goldfish.gz，也可以在官网下载git clone https://android.googlesource.com/kernel/goldfish.git

卡在启动画面中，请尝试重新启动主机。如果模拟器显示“系统没有响应”的黑屏，请单击等待几次。或者尝试以下方法：

1. AVD设置更大的内存RAM;
2. 向emulator命令传递其他命令行参数，例如：-gpu off和-wipe-data。[4]

模拟器运行后，就可以编译生成自定义内核了。下载内核源代码后解压，在arch/x86/configs目录中提供了默认的内核配置，为模拟器生成内核比较简单：

```
cd goldfish
make x86_64_ranchu_defconfig
make -j2
```

可以尝试使用-jN替换-j2，其中N是计算机上CPU数量的两倍。

最终编译生成的内核映像默认输出位置为：arch/x86/boot/bzImage。这个文件既可以用于引导Android模拟器。使用emulator命令使用自定义内核启动模拟器命令如下：

```
emulator @njuavd -kernel arch/x86/boot/bzImage -show-kernel
```

1.2 增加一个Linux系统调用

新的系统调用ptree应该有两个参数，并且其系统调用编号应为326。该系统调用的功能是以深度优先搜索的顺序返回进程树中的进程信息。这一步需要修改我们提供的Linux 4.4.124内核源码，编译生成新内核后在模拟器，虚拟机/物理机上运行。

ptree系统调用的函数声明如下：

```
int ptree(struct prinfo *buf, int *nr);
```

其中，struct prinfo应定义在prinfo.h头文件中，内容如下：

```
struct prinfo {
        pid_t parent_pid;      /* process id of parent */
        pid_t pid;             /* process id */
        pid_t first_child_pid; /* pid of youngest child */
        pid_t next_sibling_pid;/* pid of older sibling */
        long state;            /* current state of process */
        long uid;              /* user id of process owner */
        char comm[64];         /* name of program executed */
};
```

[4]更多参数可见https://developer.android.com/studio/run/emulator-commandline

ptree系统调用中的第一个参数buf指向一个进程数据的缓冲区，nr指向此缓冲区的大小（条目数）。ptree系统调用最多将nr条进程树数据复制到缓冲区buf，并保存实际复制的条目数nr。

ptree系统调用的返回值应为系统中所有进程的总数。注意返回值可能会大于复制的实际条目数，例如，如果缓冲区只能容纳300个条目但进程树中有1000个条目，则返回值应该是1000，虽然只复制了300条。

系统调用的代码应该处理可能发生的错误，例如：

1. 如果buf或nr为null，或者条目数小于1，应返回-EINVAL。
2. 如果buf或nr在可访问的地址空间之外，应返回-EFAULT。

注意：进程描述符task_struct结构的类型定义在文件include/linux/sched.h的第1528行。Linux使用双向链表来维护所有task_struct的列表。遍历该链表时，需要对该链表进行加锁防止其他内核线程的并发访问。为此，内核使用一个特殊的锁，即tasklist_lock，来确保一致性。使用的方法时在开始遍历之前获取此锁，在遍历完成时释放该锁。在释放锁之前，可能无法执行任何可能导致睡眠的操作，例如内存分配，从内核复制数据和复制数据到内核等。获取释放锁的具体代码如下：

```
read_lock(tasklist_lock);
...
...
read_unlock(&tasklist_lock);
```

尽量避免自己实现链表，使用Linux内核提供的通用链表，具体使用方法见文件include/linux/list.h。

1.3 测试你的系统调用

定义好ptree系统调用之后，编写C程序调用ptree函数。根据ptree函数的返回结果，以深度优先搜索DFS的顺序打印整个进程树。使用\t缩进区分parent进程和child进程。程序的输出语句应如下：

```
printf(/* correct number of \t */);
printf("%s,%d,%ld,%d,%d,%d,%d\n", p.comm, p.pid, p.state,
      p.parent_pid, p.first_child_pid, p.next_sibling_pid, p.uid);
```

使用ptree的方法是通过系统调用syscall，可以使用man syscall了解该系统调用的具体信息，下面给出了使用syscall调用的一个样例。

```
#include <stdio.h>

int main(void)
{
    int param;
    int ret= syscall(356, param);
    printf("ptree returned %d.\n",ret);
```

```
    return ret;
}
```

系统调用测试程序的样例输出如下。

```
swapper/0,0,0,0,1,0,0
        init,1,1,0,556,2,0
                ueventd,556,1,1,0,807,0
                logd,807,1,1,0,808,1036
                debuggerd,808,1,1,818,809,0
                ...
                main,846,1,1,976,847,0
                        system_server,976,1,846,0,1171,1000
                        putmethod.latin,1171,1,846,0,1177,10039
                        ndroid.systemui,1177,1,846,0,1332,10019
                        m.android.phone,1332,1,846,0,1348,1001
                        ...
                main,847,1,1,0,848,0
                audioserver,848,1,1,0,849,1041
                cameraserver,849,1,1,0,850,1047
                ...
        kthreadd,2,1,0,3,0,0
                ksoftirqd/0,3,1,2,0,4,0
                kworker/0:0,4,1,2,0,5,0
                kworker/0:0H,5,1,2,0,6,0
                kworker/u2:0,6,1,2,0,7,0
                migration/0,7,1,2,0,8,0
                rcu_preempt,8,1,2,0,9,0
                ...
```

1.4 在Android模拟器上运行

编译生成测试程序后，须使用Android Debug Bridge（adb）程序将其移动到Android模拟器的文件系统中。使用自定义内核启动Android虚拟设备后，可以通过以下命令检查AVD设备的状态:

```
adb devices
```

要将文件移动到模拟器的/data/misc目录（默认可读写），输入:

```
adb root
adb push local_file /data/misc
```

要使用Android系统的shell，输入

```
adb shell
```

进入Android的shell之后即可执行你的测试程序，也可以直接从adb调用程序：

```
adb shell /data/misc/exe_name
```

要从模拟器中提取文件/path/in/emulator：

```
adb pull /path/in/emulator /local/path
```

更多可见adb help。

1.5 内核模块形式添加系统调用

除了直接修改内核源码并编译，另一种方法是使用内核模块。内核模块可以根据需要在操作系统运行时进行动态加载和卸载，使用内核模块形式添加系统调用的优点是不需要重新编译内核。

下面是一个hello.c文件的内容，示范了如何利用内核模块实现一个新的系统调用hello()。

```c
#include <linux/module.h>
#include <linux/kernel.h>
#include <linux/syscalls.h>

MODULE_LICENSE("GPL");
#define __NR_hello 326

static int (*oldcall)(void);

static unsigned long **find_sys_call_table(void) {
    unsigned long int offset = PAGE_OFFSET;
    unsigned long **sct;

    while (offset < ULLONG_MAX) {
        sct = (unsigned long **)offset;

        if (sct[__NR_close] == (unsigned long *) sys_close) {
            printk(KERN_INFO "Found syscall table at address: 0x%02lX",
                   (unsigned long) sct);
            return sct;
        }

        offset += sizeof(void *);
    }
```

```
    return NULL;
}
static long sys_hello(void)
{
    printk(KERN_INFO "hello, world!\n");
    return 0;
}

static int addsyscall_init(void) {
    long *syscall = (long *)find_sys_call_table();
    oldcall = (int(*)(void))(syscall[__NR_hello]);
    write_cr0 (read_cr0 () & (~ 0x10000));
    syscall[__NR_hello] = (unsigned long) sys_hello;
    write_cr0 (read_cr0 () | 0x10000);
    printk(KERN_INFO "module load\n");
    return 0;
}

static void addsyscall_exit(void) {
    long *syscall = (long *) find_sys_call_table();
    write_cr0 (read_cr0 () & (~ 0x10000));
    syscall[__NR_hello] = (unsigned long) oldcall;
    write_cr0 (read_cr0 () | 0x10000);
    printk(KERN_INFO "module exit\n");
}

module_init(addsyscall_init);
module_exit(addsyscall_exit);
```

假设内核源码位置在~/goldfish，新建Makefile内容如下。

```
obj-m := hello.o
KDIR := ~/goldfish

all:
        make -C $(KDIR) M=$(PWD) modules

clean:
        make -C $(KDIR) M=$(PWD) clean
```

在虚拟机/物理机上使用make all命令即可进行内核模块编译，完成后可得到模块文件hello.ko。

使用内核模块的方法非常简单，首先将模块hello.ko复制到Android模拟器中

```
adb push hello.ko /data/misc
```

然后在Android模拟器中使用insmod命令安装模块

```
adb shell
cd /data/misc
insmod hello.ko
```

此时使用dmesg命令或者在Android模拟器中看到内核输出module loads即说明内核模块安装成功。通过lsmod命令也可以看到系统已经安装的内核模块。

模块使用完毕后，使用rmmod命令可以卸载模块

```
rmmod hello.ko
```

2 Linux虚拟机上添加系统调用

由于Android goldfish同样适用于Linux虚拟机和物理机，我们同样可以在PC机上测试ptree系统调用。

PC机上大部分步骤都是一样的，只有编译内核和安装内核稍有区别。编译内核的步骤如下，

```
cd goldfish
cp /boot/config-`uname -r`* .config
make menuconfig
make -j2
```

这里我们使用PC机系统当前的配置，make menuconfig之后会弹出图形界面选择配置，此时直接退出保存即可。

安装编译生成的内核步骤如下。

```
sudo make modules_install
sudo make install
```

安装完成后重启，即可在Grub启动界面选择从新内核启动。此时可以运行你的测试程序。

3 讨论

使用Android系统和桌面Linux进行多次测试，比较桌面Linux和Android系统的进程树的区别。注意在Android平台上有一个名为zygote的进程（或者叫做main），记录该进程的pid，通过运行ps和ptree简要回答下面几个问题:

1. zygote进程的目的是什么?
2. zygote二进制可执行文件在哪里?
3. 和桌面Linux系统相比，分析Android使用zygote进程的目的。

4 评分标准

1. 系统调用实现：60分
2. 系统调用测试程序：20分
3. 内核模块：10分
4. 分析比较Android和Linux进程树的区别：10分

5 注意事项

1. 遵循Linux内核编码风格，利用checkpatch脚本检查代码规范`https://www.kernel.org/doc/Documentation/process/coding-style.rst`
2. Linux Cross Referencer(LXR)可以比较方便地查看现有的系统调用是如何实现的，如`https://elixir.bootlin.com/linux/v4.4.124/source` 。参考内核源码文件kernel/sched/core.c 和kernel/timer.c 中实现的系统调用。
3. Android模拟器下载及使用方法见：http://developer.android.com