内容目录

[一 输入系统\_必备Linux编程知识 1](#__RefHeading___Toc248_2054555503)

# 一 输入系统\_必备Linux编程知识

(1) 键盘即插即用=》怎么检测键盘的接入与拔出

1) hotplug: 内核发现键盘插入/拔出=》启动hotplug进程=》发送消息给输入系统；ps: android没有用这个方式

2) inotify: 输入系统使用inotify检测目录/dev/input

(2) 对于多键盘，使用的是epoll

## 1.1 inotify和epoll

### 1.1.1 inotify

监测目录 / 文件的变化，参考代码: frameworks\native\services\inputflinger\EventHub.cpp

APP\_0006\_inotify\_epoll/inotify.c

gcc -o inotify inotify.c

mkdir tmp

./inotify tmp &

echo > tmp/1

echo > tmp/2

rm tmp/1 tmp/2

### 1.1.2 epoll

检测多个文件有无数据供读出，有无空间供写入

APP\_0006\_inotify\_epoll/epoll.c

gcc -o epoll epoll.c

mkdir tmp

mkfifo tmp/1 tmp/2 tmp/3

./epoll tmp/1 tmp/2 tmp/3 &

echo aaa > tmp/1

echo bbb > tmp/2

使用fifo是，我们的epoll程序是reader

echo aa > tmp/1 是writer

a. int tmpFd = open(argv[i], O\_RDONLY|O\_NONBLOCK);//会导致无限循环

如果reader以 O\_RDONLY|O\_NONBLOCK打开FIFO文件，

当writer写入数据时，epoll\_wait会立刻返回；

当writer关闭FIFO之后，reader再次调用epoll\_wait，它也会立刻返回(原因是EPPLLHUP，描述符被挂断)

b. int tmpFd = open(argv[i], O\_RDWR);

如果reader以 O\_RDWR打开FIFO文件

当writer写入数据时，epoll\_wait会立刻返回；

当writer关闭FIFO之后，reader再次调用epoll\_wait，它并不会立刻返回，而是继续等待有数据

### 1.1.3 inotify + epoll

编写 inotify\_epoll.c，用它来监测tmp/目录: 有文件被创建/删除，有文件可读出数据

a. 当在tmp/下创建文件时，会立刻监测到，并且使用epoll监测该文件

b. 当文件有数据时，读出数据

c. 当tmp/下文件被删除时，会立刻监测到，并且把它从epoll中移除不再监测

inotify\_epoll.c

gcc -o inotify\_epoll inotify\_epoll.c

mkdir tmp

./inotify\_epoll tmp/ &

mkfifo tmp/1 tmp/2 tmp/3

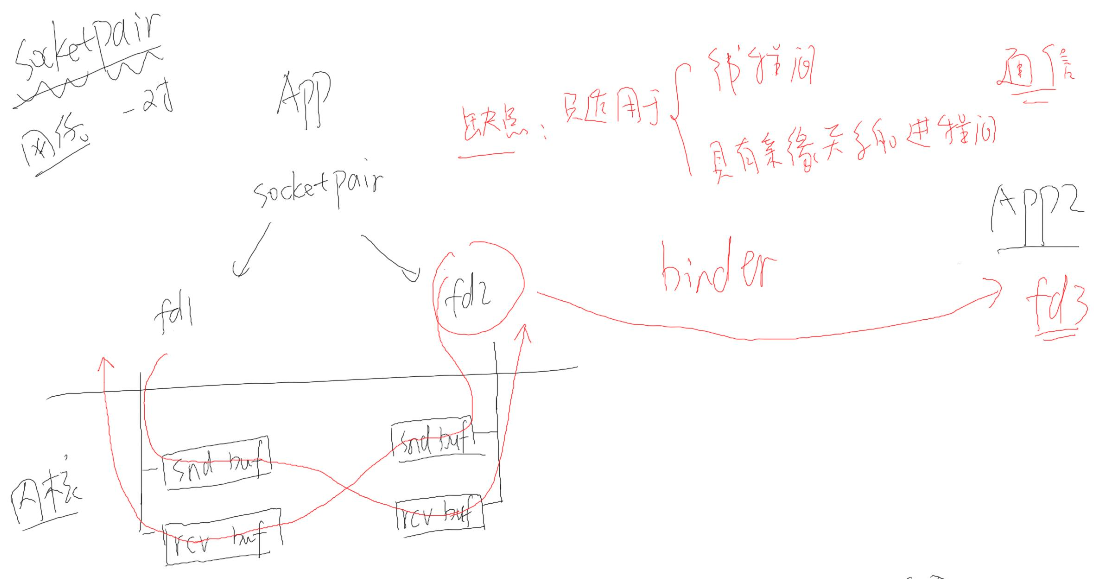
echo aaa > tmp/1

echo bbb > tmp/2

rm tmp/3

## 1.2 socketpair

APP\_0007\_socketpair\_binder/socketpair.c，亲缘关系是指fork



### 1.3 任意进程双向通信(socketpair+binder)

APP1打开1.txt得到句柄fd，在kernel中task\_struct来描述一个线程(进程)，

struct task\_struct {

struct files\_struct \*files;

{

struct fdtable \_\_rcu \*fdt;

{

struct file \_\_rcu \*\*fd; /\* current fd array \*/

}

}

}

使用binder传输文件句柄

(1) APP1 open file得到fd1

(2) 通过binder驱动，通过fd1得到file，file = APP1->files->fdt->fd[fd1]

(3) 从APP2的files->fdt->fd取出空项fd2，让它指向该file，APP2->files->fdt->fd[fd2] = file

(4) APP1通过fd1，APP2通过fd2可访问同一个文件

Input/APP\_0004\_Binder\_CPP\_App/

git pull origin

git checkout v5 // v5, use binder to transfer file descriptor

test\_server.cpp

/\* usage : test\_server <file> \*/

int main(int argc, char \*\*argv)

{

int fd;

if (argc == 2)

fd = open(argv[1], O\_RDWR);

…

sm->addService(String16("hello"), new BnHelloService(fd));

…

}

IHelloService.h

class BnHelloService: public BnInterface<IHelloService>

{

private:

int fd;

public:

virtual int get\_fd(void);

…

BnHelloService(int fd);

};

BnHelloService.cpp

status\_t BnHelloService::onTransact( uint32\_t code,

const Parcel& data,

Parcel\* reply,

uint32\_t flags)

{

…

case HELLO\_SVR\_CMD\_GET\_FD: {

int fd = this->get\_fd();

reply->writeInt32(0); /\* no exception \*/

/\* 参考:

\* frameworks\base\core\jni\android\_view\_InputChannel.cpp

\* android\_view\_InputChannel\_nativeWriteToParcel

\*/

reply->writeDupFileDescriptor(fd);

return NO\_ERROR;

} break;

…

}

int BnHelloService::get\_fd(void)

{

return fd;

}

BpHelloService.cpp

int get\_fd(void)

{

/\* 构造/发送数据 \*/

Parcel data, reply;

int exception;

data.writeInt32(0);

data.writeString16(String16("IHelloService"));

remote()->transact(HELLO\_SVR\_CMD\_GET\_FD, data, &reply);

exception = reply.readInt32();

if (exception)

return -1;

else

{

/\* 参考:

\* frameworks\base\core\jni\android\_view\_InputChannel.cpp

\* android\_view\_InputChannel\_nativeReadFromParcel

\*/

int rawFd = reply.readFileDescriptor();

return dup(rawFd);//rawFd被析构时，要用dup来保证fd不被关闭，而不是直接return rawFd

}

}

test\_client.cpp

/\* 调用Service的函数 \*/

int fd = service->get\_fd();

ALOGI("client call get\_fd = %d", fd);

//while (1) sleep(10);

lseek(fd, 0, SEEK\_SET);

char buf[500];

int len = read(fd, buf, 500);

buf[len] = '\0';

ALOGI("client read file: %s", buf);

编译

mmm frameworks/testing/APP\_0004\_Binder\_CPP\_App

执行test\_server

sm6150\_au:/proc/3287/fd # cat /sdcard/1.txt

hello, fd!

sm6150\_au:/sdcard # test\_server /sdcard/1.txt

sm6150\_au:/proc/3287/fd # ls -l

total 0

...

lrwx------ 1 root root 64 2019-07-03 21:01 3 -> /storage/emulated/0/1.txt

...

执行test\_client

sm6150\_au:/proc/3287/fd # test\_client readfile

sm6150\_au:/proc/3287/fd # logcat | grep TestService

07-03 21:03:47.162 3341 3341 I TestService: client call get\_fd = 5

07-03 21:03:47.162 3341 3341 I TestService: client read file: hello, fd!

sm6150\_au:/proc/3341/fd # ls -l

total 0

lrwx------ 1 root root 64 2019-07-03 21:10 0 -> /dev/pts/1

lrwx------ 1 root root 64 2019-07-03 21:10 1 -> /dev/pts/1

lrwx------ 1 root root 64 2019-07-03 21:10 2 -> /dev/pts/1

lrwx------ 1 root root 64 2019-07-03 21:10 3 -> /dev/binder

lrwx------ 1 root root 64 2019-07-03 21:10 4 -> socket:[66320]

lrwx------ 1 root root 64 2019-07-03 21:10 5 -> /storage/emulated/0/1.txt

原理

reply->writeDupFileDescriptor(fd);

→ obj.hdr.type = BINDER\_TYPE\_FD;

→ case BINDER\_TYPE\_FD: {

struct binder\_fd\_object \*fp = to\_binder\_fd\_object(hdr);

target\_fd = binder\_translate\_fd(fp->fd, t, thread, in\_reply\_to);

}

→ file = fget(fd);//从当前进程里面，根据fd得到一个file结构体

target\_fd = task\_get\_unused\_fd\_flags(target\_proc, O\_CLOEXEC);//从目标进程里面，获得一个没有使用的文件句柄

task\_fd\_install(target\_proc, target\_fd, file);//把file安装到目标进程的target\_fd上面去

课后作业: 支持双向通信的程序 v6

第一次:

git clone https://github.com/weidongshan/APP\_0004\_Binder\_CPP\_App.git

更新:

git pull origin

取出指定版本:

git checkout v6 // v6, use binder and socketpair for bidirectional transfer，将socketpair的fd通过binder传递给client

编译命令与v5相同