/dev/random和/dev/urandom是unix系统提供的产生随机数的设备，很多应用都需要使用random设备提供的随机数，比如ssh keys, SSL keys, TCP/IP sequence numbers等等。

而random设备的random pool是从基于中断的IRQS里面取值，IRQS跟一些特殊的硬件绑定，基于这些硬件的interrupts将会提供给random设备。

linux下我们可以用cat /proc/interrupts 查看哪些设备绑定了irq

[oracle@test oracle]$ cat /proc/interrupts  
CPU0 CPU1 CPU2 CPU3  
0: 135 0 0 301721845 IO-APIC-edge timer  
1: 0 0 0 54 IO-APIC-edge keyboard  
2: 0 0 0 0 XT-PIC cascade  
8: 0 0 0 1 IO-APIC-edge rtc  
14: 0 0 0 2 IO-APIC-edge ide0  
16: 0 0 0 40 IO-APIC-level usb-uhci  
18: 0 0 0 0 IO-APIC-level usb-uhci  
19: 0 0 0 0 IO-APIC-level usb-uhci  
23: 0 0 0 2257 IO-APIC-level ide2, ehci-hcd  
25: 0 0 0 29 IO-APIC-level ioc1  
26: 0 0 0 29 IO-APIC-level ioc0  
48: 0 0 0 2151823972 IO-APIC-level eth0  
82: 0 0 0 339588855 IO-APIC-level megaraid  
NMI: 0 0 0 0  
LOC: 301720634 301720634 301720634 301720633  
ERR: 0  
MIS: 0

来模拟一个从/dev/random取值但是/dev/random取不到足够值的情况，这时候取值的进程将会等待，直到得到足够的random 值。

[root@csdba ~]# time dd if=/dev/random of=1.dmp bs=1024k count=100

这时dd的进程将会hang住等待足够的random值

trace dd的进程在做什么

[root@csdba ~]# ps -ef|grep dd  
root 4749 4156 0 17:11 pts/1 00:00:00 dd if /dev/random of 1.dmp bs 1024k count 100  
root 4753 4634 0 17:12 pts/3 00:00:00 grep dd

[root@csdba ~]# strace -p 4749  
Process 4749 attached - interrupt to quit  
read(0,

dd的进程在等待读取足够的随机数，从上面我们可以看到eth0 网卡绑定了IRQS,那么我们进行网路传输的时候网卡将会提供很多interrupt 给/dev/random,既然这样，我们试着scp给这台机器一个文件看看。

[oracle@csdbc oracle]$ scp 1.dmp [root@10.0.100.115:/](mailto:root@10.0.100.115:/)  
[root@10.0.100.115’s](mailto:root@10.0.100.115%E2%80%99s) password:  
1.dmp 100% |\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

再回到原来窗口看strace，可以发现strace已经结束，也就是dd已经完成

read(0, “E\347\276m\274\33\227-IZ5\330\264\245\260z\326\’\376\266″…, 1048576) = 44  
write(1, “E\347\276m\274\33\227-IZ5\330\264\245\260z\326\’\376\266″…, 44) = 44  
munmap(0xb7cb5000, 1060864) = 0  
open(”/usr/share/locale/locale.alias”, O\_RDONLY) = 3  
fstat64(3, {st\_mode=S\_IFREG|0644, st\_size=2528, …}) = 0  
mmap2(NULL, 4096, PROT\_READ|PROT\_WRITE, MAP\_PRIVATE|MAP\_ANONYMOUS, -1, 0) = 0xb7db7000  
read(3, “# Locale name alias data base.\n#”…, 4096) = 2528  
read(3, “”, 4096) = 0  
close(3) = 0  
munmap(0xb7db7000, 4096) = 0  
open(”/usr/share/locale/en\_US/LC\_MESSAGES/coreutils.mo”, O\_RDONLY) = -1 ENOENT (No such file or directory)  
open(”/usr/share/locale/en/LC\_MESSAGES/coreutils.mo”, O\_RDONLY) = -1 ENOENT (No such file or directory)  
write(2, “0+100 records in\n”, 17) = 17  
write(2, “0+100 records out\n”, 18) = 18  
close(0) = 0  
close(1) = 0  
exit\_group(0) = ?  
Process 4749 detached

再看看dd的窗口

[root@csdba ~]# time dd if=/dev/random of=1.dmp bs=1024k count=100  
0+100 records in  
0+100 records out  
real 4m19.484s  
user 0m0.000s  
sys 0m0.004s

果然dd已经结束，这个过程也验证了/dev/random是基于IRQS产生随机数的。

看过/dev/random后再来看看/dev/urandom

[root@csdba ~]# time dd if=/dev/urandom of=1.dmp bs=1024k count=100  
100+0 records in  
100+0 records out  
real 0m21.758s  
user 0m0.002s  
sys 0m21.752s

很明显/dev/urandom和/dev/random有区别，/dev/urandom不受interrupts的限制，即使没有足够的interrupt它也能通过 random number generator产生足够的输出值，所以它不会导致dd hang.

另外还有一个很有意思的事情

当我们在linux上执行time dd if=/dev/random of=1.dmp bs=1024k count=100，dd虽然会hang住，但是可以用ctrl+c或kill命令中止dd，这是因为linux的dd会工作在用户模式。

但是我们遇到过在aix上执行 dd if=/dev/random of=1.dmp bs=1024k count=100，dd进程会导致占用一个单独的cpu，当dd读取一个bs的时候它进入内核模式，它将不能被ctrl+c或kill －9中断，只有当它读完一个bs后会把控制权返回给用户，这样的话就面临着如果bs值设置过大的时候会导致dd进程从/dev/random里面读不到足够的值而一直处于内核模式，dd的进程将会一直消耗cpu直到/dev/random提供了新的值。由于dd命令在aix和linux处理的方式不同，所以在aix上多采用/dev/urandom来产生随机数

=========================

在编译内核时或者生成 gpg 密钥时，会让系统给出随机参数，以便生成密钥。往往靠键盘和鼠标的输入是远不够的，我都快把键盘和鼠标给敲烂了都没有够， 比打真三还激烈啊。系统包里面有这样的软件来生成随机参数的

软件包名：  rng-utils

首先可以看一下当前系统现有的随机数有多少， 你可以看看狂敲之后，究竟有多少值，以此也可能看你的操作是否还娴熟

#cat /proc/sys/kernel/random/entropy\_avail

如果太少，就用这个来生成，吧

#rngd -r /dev/urandom  -o /dev/random  -f -t 1

运行几秒之后， 差不多也有四位数了。

现在应该够了

/proc/sys/kernel/random 下面存放了random相关的一些东西，如果entropy\_avail 等

**Can't open /dev/random的解决方法和一点/dev/random的资料【by joyride】**

一. 解决“Can't open /dev/random”的问题

今天在移植appweb时遇到一个错误：Can't open /dev/random。搞了半天终于解决了。

如果我们使用的linux版本（1.3.30以上）支持random，那么运行下面的命令（必需以root身份运行）

mknod -m 644 /dev/random c 1 8  
mknod -m 644 /dev/urandom c 1 9  
chown root:root /dev/random /dev/urandom

即可在/dev下创建出两个设备文件random和urandom，上面的问题“Can't open /dev/random”解决。

经试验验证，设备random和urandom是必需成对创建（虽然只报告了Can't open /dev/random错误）。若没有创建urandom而只创建了random，仍然还是报告Can't open /dev/random错误。

二. 以下是关于/dev/random的一点资料

英文部分转自<http://blog.csdn.net/wonder4/archive/2008/03/18/2194411.aspx>，中文部分是我自己翻译的，水平有限，尽力而译（紫色文字是我自己加的，英文原文中没有；红色部分表示根据理解翻译的，不是直译过来的）。

The character special files /dev/random and /dev/urandom (present since Linux 1.3.30) provide an interface to the kernel's random number generator. File /dev/random has major device number 1 and minor device number 8. File /dev/urandom has major device number 1 and minor device number 9.

特殊字符文件/dev/random和/dev/urandom（linux从1.3.30内核起支持）提供了一个与linux内核中的随机数发生器的接口。文件/dev/random的主设备号是1，次设备号是8；文件/dev/urandom的主设备号是1，次设备号是9。

The random number generator gathers environmental noise from device drivers and other sources into an entropy pool. The generator also keeps an estimate of the number of bits of noise in the entropy pool. From this entropy pool random numbers are created.

（太专业了，看不懂。大概的意思是：随机数发生器搜集一些环境噪声（从网上搜到的：例如，键盘驱动程序收集两个按键之间时间的信息）放到熵池中，同时熵池对噪声的位数进行估计，然后熵池用这些随机信息来生成随机数。）

When read, the /dev/random device will only return random bytes within the estimated number of bits of noise in the entropy pool. /dev/random should be suitable for uses that need very high quality randomness such as one-time pad or key generation. When the entropy pool is empty, reads from /dev/random will block until additional environmental noise is gathered.

当读取/dev/random设备时将从熵池中返回一个随机数（return random bytes 我理解为返回的随机数由多个随机字节组成）。/dev/random应该适用于高性能的应用，如一次性密码或用于生成密匙等。一旦熵池为空时，读取/dev/random的操作将被阻塞，直到有新的环境噪声被（随机数发生器）捕获（，然后被放入熵池以产生出新的随机数）。

A read from the /dev/urandom device will not block waiting for more entropy. As a result, if there is not sufficient entropy in the entropy pool, the returned values are theoretically vulnerable to a cryptographic attack on the algorithms used by the driver. Knowledge of how to do this is not available in the current non-classified literature, but it is theoretically possible that such an attack may exist. If this is a concern in your application, use /dev/random instead.

（还是太专业，不太懂。大概意思是：/dev/urandom是/dev/random的非阻塞版本，即当熵池中没有可取的随机数时，读取/dev/urandom设备仍然会返回一个随机数，只是该随机数的安全性理论上不高。如果应用对安全性要求很高，那么应该使用/dev/random。）

***Configuring***

If your system does not have /dev/random and /dev/urandom created already, they can be created with the following commands:

如果你的系统没有/dev/random和/dev/urandom，那么可以通过下面的命令来创建：

mknod -m 644 /dev/random c 1 8  
mknod -m 644 /dev/urandom c 1 9  
chown root:root /dev/random /dev/urandom

When a Linux system starts up without much operator interaction, the entropy pool may be in a fairly predictable state. This reduces the actual amount of noise in the entropy pool below the estimate. In order to counteract this effect, it helps to carry entropy pool information across shut-downs and start-ups. To do this, add the following lines to an appropriate script which is run during the Linux system start-up sequence:

在linux系统启动过程中，如果没有足够多的交互处理，那么熵池中的噪声就很少（可能低于估计随机数需要的数量），于是熵池产生随机数也容易被言中。为了处理这个问题，熵池可以从关机和启动过程来取得噪声。实现的方法是加入下面的行到一个合适的linux启动脚本中：

    echo "Initializing random number generator..."  
    random\_seed=/var/run/random-seed  
    # Carry a random seed from start-up to start-up  
    # Load and then save the whole entropy pool  
    if [ -f $random\_seed ]; then  
        cat $random\_seed >/dev/urandom  
    else  
        touch $random\_seed  
    fi  
    chmod 600 $random\_seed  
    poolfile=/proc/sys/kernel/random/poolsize  
    [ -r $poolfile ] && bytes='cat $poolfile' || bytes=512  
    dd if=/dev/urandom of=$random\_seed count=1 bs=$bytes

Also, add the following lines in an appropriate script which is run during the Linux system shutdown:

加入系列行到一个合适的linux系统的关机执行脚本中：

    # Carry a random seed from shut-down to start-up  
    # Save the whole entropy pool  
    echo "Saving random seed..."  
    random\_seed=/var/run/random-seed  
    touch $random\_seed  
    chmod 600 $random\_seed  
    poolfile=/proc/sys/kernel/random/poolsize  
    [ -r $poolfile ] && bytes='cat $poolfile' || bytes=512  
    dd if=/dev/urandom of=$random\_seed count=1 bs=$bytes

下面是关于/dev/random在proc目录下的一个接口。目录/proc/sys/kernel/random（linux 2.3.16版开始）下的文件提供了另外一个与/dev/random设备的接口。水平有限，就不再翻译了。

***Proc Interface***

The files in the directory /proc/sys/kernel/random (present since 2.3.16) provide an additional interface to the /dev/random device.

The read-only file entropy\_avail gives the available entropy. Normally, this will be 4096 (bits), a full entropy pool.

The file poolsize gives the size of the entropy pool. Normally, this will be 512 (bytes). It can be changed to any value for which an algorithm is available. Currently the choices are 32, 64, 128, 256, 512, 1024, 2048.

The file read\_wakeup\_threshold contains the number of bits of entropy required for waking up processes that sleep waiting for entropy from /dev/random. The default is 64. The file write\_wakeup\_threshold contains the number of bits of entropy below which we wake up processes that do a select() or poll() for write access to /dev/random. These values can be changed by writing to the files.

The read-only files uuid and boot\_id contain random strings like 6fd5a44b-35f4-4ad4-a9b9-6b9be13e1fe9. The former is generated afresh for each read, the latter was generated once.

***Files***

/dev/random  
/dev/urandom

***Author***

The kernel's random number generator was written by Theodore Ts'o ().