NAME

random, urandom − kernel random number source devices

SYNOPSIS

#include <linux/random.h>

int ioctl(*fd***, RND***r equest***,** *param***);**

DESCRIPTION

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. The file*/de v/random* has major device number 1 and minor device number 8. The file */de v/urandom* has major device number 1 and minor device number 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.

Linux 3.17 and later provides the simpler and safer **getrandom**(2) interface which requires no special files; see the **getrandom**(2) manual page for details.

When read, the */de v/urandom* device returns random bytes using a pseudorandom number generator seeded from the entropy pool. Reads from this device do not block (i.e., the CPU is not yielded), but can incur an appreciable delay when requesting large amounts of data.

When read during early boot time, */de v/urandom* may return data prior to the entropy pool being initialized. If this is of concern in your application, use**getrandom**(2) or */de v/random* instead.

The */dev/random* device is a legacy interface which dates back to a time where the cryptographic primitives used in the implementation of */dev/urandom* were not widely trusted. It will return random bytes only within the estimated number of bits of fresh noise in the entropy pool, blocking if necessary. */dev/random* is suitable for applications that need high quality randomness, and can afford indeterminate delays.

When the entropy pool is empty, reads from */dev/random* will block until additional environmental noise is gathered. Since Linux 5.6, the**O_NONBLOCK** flag is ignored as */de v/random* will no longer block except during early boot process. In earlier versions, if **open**(2) is called for */de v/random* with the **O_NONBLOCK** flag, a subsequent **read**(2) will not block if the requested number of bytes is not available. Instead, the available bytes are returned. If no byte is available, **read**(2) will return −1 and *errno* will be set to **EAGAIN**.

The **O_NONBLOCK** flag has no effect when opening */de v/urandom*. When calling**r ead**(2) for the device */dev/urandom*, reads of up to 256 bytes will return as many bytes as are requested and will not be interrupted by a signal handler. Reads with a buffer over this limit may return less than the requested number of bytes or fail with the error **EINTR**, if interrupted by a signal handler.

Since Linux 3.16, a **read**(2) from */de v/urandom* will return at most 32 MB. A **read**(2) from */de v/random* will return at most 512 bytes (340 bytes before Linux 2.6.12).

Writing to */dev/random* or */dev/urandom* will update the entropy pool with the data written, but this will not result in a higher entropy count. This means that it will impact the contents read from both files, but it will not make reads from */dev/random* faster.

Usage

The */dev/random* interface is considered a legacy interface, and */de v/urandom* is preferred and sufficient in all use cases, with the exception of applications which require randomness during early boot time; for these applications, **getrandom**(2) must be used instead, because it will block until the entropy pool is initialized.

If a seed file is saved across reboots as recommended below, the output is cryptographically secure against attackers without local root access as soon as it is reloaded in the boot sequence, and perfectly adequate for network encryption session keys. (All major Linux distributions have saved the seed file across reboots since 2000 at least.) Since reads from */de v/random* may block, users will usually want to open it in nonblocking mode (or perform a read with timeout), and provide some sort of user notification if the desired entropy is not immediately available.

Configuration

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

```
mknod −m 666 /dev/random c 1 8
mknod −m 666 /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 lines to an appropriate script which is run during the Linux system start-up sequence:

```
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 $poolfile ] && bits=$(cat $poolfile) || bits=4096
bytes=$(expr $bits / 8)
dd if=/dev/urandom of=$random_seed count=1 bs=$bytes
```

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

```
# 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 $poolfile ] && bits=$(cat $poolfile) || bits=4096
bytes=$(expr $bits / 8)
dd if=/dev/urandom of=$random_seed count=1 bs=$bytes
```

In the above examples, we assume Linux 2.6.0 or later, where */proc/sys/kernel/random/poolsize* returns the size of the entropy pool in bits (see below).

/proc interfaces

The files in the directory */proc/sys/kernel/random* (present since Linux 2.3.16) provide additional information about the */dev/random* device:

entropy_avail

This read-only file gives the available entropy, in bits. This will be a number in the range 0 to 4096.

poolsize

This file gives the size of the entropy pool. The semantics of this file vary across kernel versions:

Linux 2.4:

This file gives the size of the entropy pool in *bytes*. Normally, this file will have the value 512, but it is writable, and can be changed to any value for which an algorithm is available. The choices are 32, 64, 128, 256, 512, 1024, or 2048.

Linux 2.6 and later:
This file is read-only, and gives the size of the entropy pool in *bits*. It contains the value 4096.

read_wakeup_threshold
This file contains the number of bits of entropy required for waking up processes that sleep waiting for entropy from */dev/random*. The default is 64.

write_wakeup_threshold
This file contains the number of bits of entropy below which we wake up processes that do a **select**(2) or **poll**(2) for write access to */dev/random*. These values can be changed by writing to the files.

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

ioctl(2) interface
The following **ioctl**(2) requests are defined on file descriptors connected to either */dev/random* or */dev/urandom*. All requests performed will interact with the input entropy pool impacting both */dev/random* and */dev/urandom*. The**CAP_SYS_ADMIN** capability is required for all requests e xcept **RNDGETENTCNT**.

RNDGETENTCNT
Retrieve the entropy count of the input pool, the contents will be the same as the *entropy_avail* file under proc. The result will be stored in the int pointed to by the argument.

RNDADDTOENTCNT
Increment or decrement the entropy count of the input pool by the value pointed to by the argument.

RNDGETPOOL
Removed in Linux 2.6.9.

RNDADDENTROPY
Add some additional entropy to the input pool, incrementing the entropy count. This differs from writing to */dev/random* or */dev/urandom*, which only adds some data but does not increment the entropy count. The following structure is used:

```
struct rand_pool_info {
    int    entropy_count;
    int    buf_size;
    __u32  buf[0];
};
```

Here *entropy_count* is the value added to (or subtracted from) the entropy count, and *buf* is the buffer of size *buf_size* which gets added to the entropy pool.

RNDZAPENTCNT, **RNDCLEARPOOL**
Zero the entropy count of all pools and add some system data (such as wall clock) to the pools.

FILES
/dev/random
/dev/urandom

NOTES
For an overview and comparison of the various interfaces that can be used to obtain randomness, see **random**(7).

BUGS
During early boot time, reads from */dev/urandom* may return data prior to the entropy pool being initialized.

SEE ALSO

mknod(1), **getrandom**(2), **random**(7)

RFC 1750, "Randomness Recommendations for Security"