Full credit is given to the above companies including the OS that this PDF file was generated!

Red Hat Enterprise Linux Release 9.2 Manual Pages on 'initstate.3p' command

$ man initstate.3p

INITSTATE(3P) POSIX Programmer's Manual INITSTATE(3P)

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

initstate, random, setstate, srandom ? pseudo-random number functions

SYNOPSIS

```
#include <stdlib.h>
char *initstate(unsigned seed, char *state, size_t size);
long random(void);
char *setstate(char *state);
void srandom(unsigned seed);
```

DESCRIPTION

The random() function shall use a non-linear additive feedback random-number generator employing a default state array size of 31 long inte? gers to return successive pseudo-random numbers in the range from 0 to 231-1. The period of this random-number generator is approximately 16 x (231-1). The size of the state array determines the period of the ran? dom-number generator. Increasing the state array size shall increase the period.

With 256 bytes of state information, the period of the random-number

generator shall be greater than 269.

Like rand(), random() shall produce by default a sequence of numbers that can be duplicated by calling srandom() with 1 as the seed.

The srandom() function shall initialize the current state array using the value of seed.

The initstate() and setstate() functions handle restarting and changing random-number generators. The initstate() function allows a state ar? ray, pointed to by the state argument, to be initialized for future use. The size argument, which specifies the size in bytes of the state array, shall be used by initstate() to decide what type of random-num? ber generator to use; the larger the state array, the more random the numbers. Values for the amount of state information are 8, 32, 64, 128, and 256 bytes. Other values greater than 8 bytes are rounded down to the nearest one of these values. If initstate() is called with 8?size<32, then random() shall use a simple linear congruential random number generator. The seed argument specifies a starting point for the random-number sequence and provides for restarting at the same point.

The initstate() function shall return a pointer to the previous state information array.

If initstate() has not been called, then random() shall behave as though initstate() had been called with seed=1 and size=128.

Once a state has been initialized, setstate() allows switching between state arrays. The array defined by the state argument shall be used for further random-number generation until initstate() is called or set? state() is called again. The setstate() function shall return a pointer to the previous state array.

RETURN VALUE

If initstate() is called with size less than 8, it shall return NULL.

The random() function shall return the generated pseudo-random number.

The srandom() function shall not return a value.

Upon successful completion, initstate() and setstate() shall return a pointer to the previous state array; otherwise, a null pointer shall be returned.

ERRORS

No errors are defined.

The following sections are informative.

EXAMPLES

None.

APPLICATION USAGE

After initialization, a state array can be restarted at a different point in one of two ways:

1. The initstate() function can be used, with the desired seed, state array, and size of the array.
2. The setstate() function, with the desired state, can be used, fol? lowed by srandom() with the desired seed. The advantage of using both of these functions is that the size of the state array does not have to be saved once it is initialized.

Although some implementations of random() have written messages to standard error, such implementations do not conform to POSIX.1?2008.

Issue 5 restored the historical behavior of this function.

Threaded applications should use erand48(), nrand48(), or jrand48() in? stead of random() when an independent random number sequence in multi? ple threads is required.

These functions should be avoided whenever non-trivial requirements (including safety) have to be fulfilled.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

drand48(), rand()

The Base Definitions volume of POSIX.1?2017, <stdlib.h>

COPYRIGHT