NAME

random, srandom, initstate, setstate – better random number generator; routines for changing generators

SYNOPSIS

long random( );

int srandom(unsigned *seed***);**

char ∗**initstate(unsigned** *seed***, char** ∗*state***, int** *n***);**

char ∗**setstate(char** ∗*state***);**

MT-LEVEL

See the **NOTES** section of this page.

DESCRIPTION

random( ) uses a non-linear additive feedback random number generator employing a default table of size 31 long integers to return successive pseudo-random numbers in the range from 0 to $2^{31}-1$. The period of this random number generator is very large, approximately $16\times(2^{31}-1)$.

random( ) and **srandom( )** have (almost) the same calling sequence and initialization properties as **rand( )** and **srand( )** (see **rand**(3C)). The difference is that **rand**(3C) produces a much less random sequence—in fact, the low dozen bits generated by rand go through a cyclic pattern. All the bits generated by **random( )** are usable. For example,

random( )&01

will produce a random binary value.

Unlike **srand( )**, **srandom( )** does not return the old seed because the amount of state information used is much more than a single word. Two other routines are provided to deal with restarting/changing random number generators. Like **rand**(3C), however, **random( )** will, by default, produce a sequence of numbers that can be duplicated by calling **srandom( )** with **1** as the seed.

The **initstate( )** routine allows a state array, passed in as an argument, to be initialized for future use. *n* specifies the size of *state* in bytes. **initstate( )** uses *n* to decide how sophisticated a random number generator it should use—the more state, the better the random numbers will be. Current "optimal" values for the amount of state information are 32, 64, 128, and 256 bytes. If the amount of state information is less than 32 bytes, a simple linear congruential random number generator is used. Using less than 8 bytes causes an error. The seed for the initialization (which specifies a starting point for the random number sequence, and provides for restarting at the same point) is also an argument. **initstate( )** returns a pointer to the previous state information array.

Once a state has been initialized, the **setstate( )** routine provides for rapid switching between states. **setstate( )** returns a pointer to the previous state array; its argument state array is used for further random number generation until the next call to **initstate( )** or **setstate( )**.

Once a state array has been initialized, it may be restarted at a different point either by calling **initstate( )** (with the desired seed, the state array, and its size) or by calling both **setstate( )** (with the state array) and **srandom( )** (with the desired seed). The advantage of calling both **setstate( )** and **srandom( )** is that the size of the state array does not have to be remembered after it is initialized.

With 256 bytes of state information, the period of the random number generator is greater than 2^{69}, which should be sufficient for most purposes.

RETURN VALUES

If **initstate( )** is called with less than 8 bytes of state information, or if **setstate( )** detects that the state information has been garbled, error messages are printed on the standard error output.

EXAMPLES

```
/* Initialize an array and pass it in to initstate. */
static long state1[32] = {
        3,
        0x9a319039, 0x32d9c024, 0x9b663182, 0x5da1f342,
        0x7449e56b, 0xbeb1dbb0, 0xab5c5918, 0x946554fd,
        0x8c2e680f, 0xeb3d799f, 0xb11ee0b7, 0x2d436b86,
        0xda672e2a, 0x1588ca88, 0xe369735d, 0x904f35f7,
        0xd7158fd6, 0x6fa6f051, 0x616e6b96, 0xac94efdc,
        0xde3b81e0, 0xdf0a6fb5, 0xf103bc02, 0x48f340fb,
        0x36413f93, 0xc622c298, 0xf5a42ab8, 0x8a88d77b,
        0xf5ad9d0e, 0x8999220b, 0x27fb47b9
        };
main( ) {
        unsigned seed;
        int n;
        seed = 1;
        n = 128;
        initstate(seed, state1, n);
        setstate(state1);
        printf("%d0,random( ));
}
```

SEE ALSO

drand48(3C), **rand**(3C)

NOTES

random( ) and **srandom( )** are unsafe in multi-thread applications.

Use of these interfaces in multi-thread applications is unsupported.

random( ) and **srandom( )** function at about two-thirds the speed of **rand**(3C).