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 'rand.3p' command

$ man rand.3p

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

rand, rand_r, srand ? pseudo-random number generator

SYNOPSIS

```
#include <stdlib.h>
int rand(void);
int rand_r(unsigned *seed);
void srand(unsigned seed);
```

DESCRIPTION

For rand() and srand(): The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the re? quirements described here and the ISO C standard is unintentional. This volume of POSIX.1?2017 defers to the ISO C standard.

The rand() function shall compute a sequence of pseudo-random integers in the range [0,{RAND_MAX}] with a period of at least 232.

The rand() function need not be thread-safe.

The rand_r() function shall compute a sequence of pseudo-random inte? gers in the range [0,{RAND_MAX}]. (The value of the {RAND_MAX} macro

shall be at least 32767.)

If rand_r() is called with the same initial value for the object pointed to by seed and that object is not modified between successive returns and calls to rand_r(), the same sequence shall be generated.

The srand() function uses the argument as a seed for a new sequence of pseudo-random numbers to be returned by subsequent calls to rand(). If srand() is then called with the same seed value, the sequence of pseudo-random numbers shall be repeated. If rand() is called before any calls to srand() are made, the same sequence shall be generated as when srand() is first called with a seed value of 1.

The implementation shall behave as if no function defined in this vol?ume of POSIX.1?2017 calls rand() or srand().

RETURN VALUE

The rand() function shall return the next pseudo-random number in the sequence.

The rand_r() function shall return a pseudo-random integer.

The srand() function shall not return a value.

ERRORS

No errors are defined.

The following sections are informative.

EXAMPLES

Generating a Pseudo-Random Number Sequence

The following example demonstrates how to generate a sequence of pseudo-random numbers.

```
#include <stdio.h>
#include <stdlib.h>
...
    long count, i;
    char *keystr;
    int elementlen, len;
    char c;
...
/* Initial random number generator. */
```

```
srand(1);
/* Create keys using only lowercase characters */
len = 0;
for (i=0; i<count; i++) {
    while (len < elementlen) {
        c = (char) (rand() % 128);
        if (islower(c))
            keystr[len++] = c;
    }
    keystr[len] = '\0';
    printf("%s Element%0*ld\n", keystr, elementlen, i);
    len = 0;
}
```

Generating the Same Sequence on Different Machines

The following code defines a pair of functions that could be incorporated into applications wishing to ensure that the same sequence of numbers is generated across different machines.

```
static unsigned long next = 1;
int myrand(void)  /* RAND_MAX assumed to be 32767. */
{
    next = next * 1103515245 + 12345;
    return((unsigned)(next/65536) % 32768);
}
void mysrand(unsigned seed)
{
    next = seed;
}
```

APPLICATION USAGE

The drand48() and random() functions provide much more elaborate pseudo-random number generators.

The limitations on the amount of state that can be carried between one function call and another mean the rand_r() function can never be implemented in a way which satisfies all of the requirements on a pseudo-

random number generator.

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

RATIONALE

The ISO C standard rand() and srand() functions allow per-process pseudo-random streams shared by all threads. Those two functions need not change, but there has to be mutual-exclusion that prevents inter? ference between two threads concurrently accessing the random number generator.

With regard to rand(), there are two different behaviors that may be wanted in a multi-threaded program:

1. A single per-process sequence of pseudo-random numbers that is shared by all threads that call rand()
2. A different sequence of pseudo-random numbers for each thread that calls rand()

This is provided by the modified thread-safe function based on whether the seed value is global to the entire process or local to each thread.

This does not address the known deficiencies of the rand() function im? plementations, which have been approached by maintaining more state. In effect, this specifies new thread-safe forms of a deficient function.

FUTURE DIRECTIONS

The rand_r() function may be removed in a future version.

SEE ALSO

drand48(), initstate()

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

COPYRIGHT

is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see https://www.ker?nel.org/doc/man-pages/reporting_bugs.html .