```
Xref: sparky sci.crypt:7943 sci.math:20530
Newsgroups: sci.crypt,sci.math
Path: sparky!uunet!destroyer!gatech!howland.reston.ans.net!zaphod.mps.ohio-state.ed
From: g...@koonda.acci.com.au (Greg Rose)
Subject: Re: "Card-shuffling" algorithms
Message-ID: <9306912.14607@mulga.cs.mu.OZ.AU>
Sender: ne...@cs.mu.OZ.AU
Organization: Australian Computing and Communications Institute
References: <9306312.14290@mulga.cs.mu.OZ.AU> <1578@eouk5.eoe.co.uk>
Date: Wed, 10 Mar 1993 02:41:52 GMT
Lines: 100

A Point of Order, Mr. Chairman!

In article <15...@eouk5.eoe.co.uk> aha...@eoe.co.uk (Andrew Haley) writes:
>Greg Rose (g...@nareen.acci.COM.AU) wrote:
>: >    for (n=0; n<3; n++)

As you can see from the doubled indentation, I didn't write this,
someone else did. (I really ought to keep a copy of my outgoing news
postings, then I'd be able to tell you who did, but I don't
want to.) Please check your attributions. I was quoting the program
from someone else's article.

>: >    {
>: >        r = lrand48() % 3;
>                  ^^^^^^^^^^^
>
>This is a bad way of generating a random number in the range 0 <= r < 4,
>because linear congruential generators such as lrand48() generate
>numbers with very nonrandom lower bits.  (This only occurs if you use
>a modulus which is a power of 2 in your random number generator.)  The
>best way to fix this is to use
>r = (lrand48() >> x) % 3
>where x is some power of two.

I don't see your point. 3 is not a power of 2, and all of the bits
returned are significant in computing the result. Now if it had been
'% 4', I might have commented on it myself.

(Sarcasm: You're right! It really IS a bad way of "generating a random
number in the range 0 <= r < 4", since it generates one in the range
0 <= r < 3. :-)

In fact, suppose x == 29 in the example you give above (given that
lrand48 returns 31 PR bits). You are trying to tell me that the four
remaining possible values, mod 3, give equally likely results? In
fact, 0 will be twice as likely as 1 or 2.

Sorry. I think you are equating the mod-by-3 operation with the
mask-with-3 operation, and they are not even nearly the same thing.

>To some extent lrand48() protects you from this by not returning the
>lower 16 bits of its internal state, but with a linear congruential
>generator you might as well use the most significant bits you can get.

Given that the multiplier (a = 0x5DEECE66D on my machine, note that
needs 35 bits to represent it) any low-order cycling behaviour will be
restricted to the bottom 13 or so bits, which are not returned. (This
is according to my intuition, if someone has a (dis)proof of this
```

please tell me.) I cannot see any reason to further restrict the interval, or to preferentially use some of the bits over others.

Technically, there IS a problem with the above generator. The example above demonstrates it. When the desired interval doesn't exactly divide the interval you already have, the smaller numbers are more likely than the larger ones. What you really have to do is throw away any result from the original generator that is too big. The worst case of this, by one estimate, is when you are trying to generate random numbers in an interval roughly 2/3 of the input interval. In this case, half of the numbers are twice as likely as the other half. What is worse is that merely looking at the resulting numbers (with an eyeball I mean) probably won't make this apparent. Try running a program that says

```
  printf("%ld\n", lrand48() % 1431655764L);
```

a few million times -- the results "look" random to me, but definitely are not.

Here is the routine I use to roll a random number in the interval [0, n) (I assume without attesting that "random" returns good PR numbers in the range [0, 2**31-1], which according to current theory and the documentation it should):

```
/*
 * Roll a random number [0..n-1].
 * Avoid the slight bias to low numbers.
 */
int
roll(int n)
{
    long        rval;
    long        biggest;
#define BIG 0x7FFFFFFFL /* biggest value returned by random() */

    if (n == 0)
        return 0;
    biggest = (BIG / n) * n;
    do {
        rval = random();
    } while (rval >= biggest);
    return rval % n;
}
```

I hope this has cleared things up a bit.

--
Greg Rose Australian Computing and Communications Institute
g...@acci.com.au +61 18 174 842
`Use of the standard phrase "HIJACKED" may be inadvisable' -- CAA