`std::rand::random::<Talk>()`

Huon Wilson

December 18, 2014

http://huonw.github.io/rand-dec14

Digital Randomness

A sequence of bits, e.g.

$$11011110\ 11111000\ 01001010\ 00111100\ldots,$$

A sequence of bits, e.g.

$$\underbrace{11011110}_{222}\underbrace{11111000}_{248}\underbrace{01001010}_{74}\underbrace{00111100}_{60}\ldots,$$

Usually generated/consumed in chunks.

Why?

Lots of uses for randomness:

- simulations: scientific, testing
- games: shuffling cards, collecting loot
- security: keys, session IDs

All want “high quality” random numbers.

What is quality?

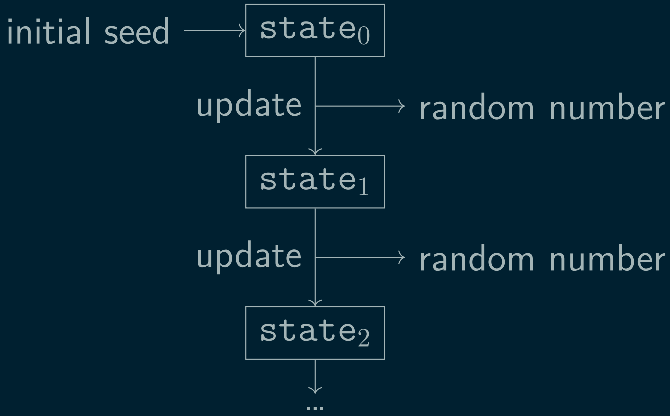
It depends!

Usually:

- uniformity: every bit has 50% chance of being 0 or 1
- unpredictability: the value of a bit can't be guessed base on the value of others

How can a deterministic machine be random?

Conventional computer RNGs follow patterns.



The seed controls *which* pattern.

Compute the seed (or state), and you know the full stream.

RNGs for cryptography need to ensure the seed/state is hard to compute. (Or be true random number generators, e.g. measure nuclear decay.)

Bad: XorShift. Good: ChaCha.

Rust

Rust

Thread-safety (by default)

Often a pervasive use of a single global RNG. Languages like C, R, Julia (recently improved in e.g. JuliaLang/julia#8832).

Automatically guaranteed this isn't a problem in Rust!

Rust

SIMD: dSFMT

```
__m128i v, w, x, y, z;

// ...
x = a->si;
z = _mm_slli_epi64(x, DSFMT_SL1);
z = _mm_xor_si128(z, b->si);
y = _mm_xor_si128(y, z);

v = _mm_srli_epi64(y, DSFMT_SR);
w = _mm_and_si128(y, sse2_param_mask.i128);
v = _mm_xor_si128(v, x);
v = _mm_xor_si128(v, w);
r->si = v;
u->si = y;
```

http://www.math.sci.hiroshima-u.ac.jp/~ m-mat/MT/SFMT/

```c
__m128i v, w, x, y, z;

// ...
x = a->si;
z = _mm_slli_epi64(x, DSFMT_SL1);
z = _mm_xor_si128(z, b->si);
y = _mm_xor_si128(y, z);

v = _mm_srli_epi64(y, DSFMT_SR);
w = _mm_and_si128(y, sse2_param_mask.i128);
v = _mm_xor_si128(v, x);
v = _mm_xor_si128(v, w);
r->si = v;
u->si = y;
```

http://www.math.sci.hiroshima-u.ac.jp/~ m-mat/MT/SFMT/

```rust
// ...
let y = (a << SSE2_SL) ^ b ^ y;
let v = (y >> SSE2_SR) ^ (y & SSE2_PARAMS_MASK) ^ a;

*r = v;
*u = y;
```

https://github.com/Grieverheart/dsfmt-rs

Creates essentially the same ASM. Benchmark:

```
let mut rng: dsfmt::DSFMTRng = SeedableRng::from_seed(12345u32);

let mut sum = 0_f64;
for _ in range(0u32, 1_000_000_000) {
    sum += rng.gen()
}

println!("{}", sum)
```

```
C
500014293.513722
User time: 1.86s
Rust
500014293.513722
User time: 1.93s
```

Rust

Traits

```
impl Rand for u8
impl Rand for u16
// ...
```

Get an number with a random value:

```
use std::rand;

let x: u8 = rand::random();
let y: u16 = rand::random();
```

```
impl Rand for XorShiftRng
impl Rand for ChaChaRng
// ...
```

Get an RNG with a random seed:

```
use std::rand;

let x: rand::XorShiftRng = rand::random();
let y: rand::ChaChaRng = rand::random();
```

Rust

Community!

E.g.

- Careful analysis of documentation/use of `/dev/[u]random`
- Implement Bernstein's ChaCha RNG (http://cr.yp.to/chacha.html, sneves: #17387)
- Update `std::rand` to use the new, better `getrandom(2)` syscall on Linux, when available (strcat and klutzy: #18664)

Questions?