Ruby master - Bug #13885

Random.urandom と securerandom について

09/09/2017 03:40 PM - mame (Yusuke Endoh)

Status:	Closed		
Priority:	Normal		
Assignee:			
Target version:			
ruby -v:	ruby 2.5.0dev (2017-09-09 trunk 59792) [x86_64-linux]	**Backport:**	2.2: UNKNOWN, 2.3: UNKNOWN, 2.4: UNKNOWN

Description

遠藤です。

Random.urandom と securerandom の仕様について卜部さんと話していて、いくつか気になる点が出てきたので挙げておきます。

1. Random.urandom は、getrandom(2) や (/dev/urandom に対する) read(2) システムコールを 1 回しか呼ばないようです。よって、要求量が大きくて 1 回では文字列を準備できないとき（例えば Random.urandom(100_000_000) ）、失敗して nil を返します。これは意図的でしょうか。要求量が得られるまで（または他の要因で失敗するまで）繰り返し呼ぶべきではないでしょうか。
2. Random.urandom は失敗するとき、nil ではなく例外を投げるほうがよいのではないでしょうか。nil を返して実行が進んで幸せになるケースはあまり思い浮かびませんでした。
3. Random.urandom が nil を返す可能性があることは rdoc で明記されるべきではないでしょうか。（とりあえず r59803 で書き足しました。まずかったらすみません）
4. securerandom は、初回の呼び出しで Random.urandom が成功したらその後ずっと Random.urandom を使い、失敗したらずっと openssl を使うようになっています。これは本当に初回の呼び出しだけで振り分けるのでよいのでしょうか。具体的には、Random.urandom(0) は常に成功する（空文字列を返す）ので、実際には urandom が使えない環境でも urandom に固定される可能性があります。
5. 逆に、securerandom のバックエンドを動的に切り替えるようにしたら問題あるでしょうか。最初は成功していた Random.urandom が途中から失敗するようになったとき、openssl へフォールバックしても良いものでしょうか。また、Random.urandom が復活した場合、openssl から Random.urandom に戻すのは問題ないでしょうか。

Related issues:

Related to Ruby master - Bug #9569: SecureRandom should try /dev/urandom first	**Closed**

Associated revisions

Revision 59858 - 09/12/2017 01:57 PM - mame (Yusuke Endoh)

Random.urandom raises an exception instead of returning nil when failed

Early failure looks better in this case. Refs [Bugs #13885].

Revision 59858 - 09/12/2017 01:57 PM - mame (Yusuke Endoh)

Random.urandom raises an exception instead of returning nil when failed

Early failure looks better in this case. Refs [Bugs #13885].

Revision 59858 - 09/12/2017 01:57 PM - mame (Yusuke Endoh)

Random.urandom raises an exception instead of returning nil when failed

Early failure looks better in this case. Refs [Bugs #13885].

History

#1 - 09/10/2017 02:59 AM - kosaki (Motohiro KOSAKI)

小崎です
このへんあんまり詳しくないんですが。

> 1.Random.urandom は、getrandom(2) や (/dev/urandom に対する) read(2) システムコールを 1 回しか呼ばないようです。よって、要求量が大きくて 1 回では文字列を準備できないとき（例えば Random.urandom(100_000_000) ）、失敗して nil を返します。これは意図的でしょうか。要求量が得られるまで（または他の要因で失敗するまで）繰り返し呼ぶべきではないでしょうか。

そんな気がします。一瞬エントロピー枯渇について心配しましたが、Rubyの側で小さいurandomを大量に呼べばどうせ枯渇するんだし

> 2.Random.urandom は失敗するとき、nil ではなく例外を投げるほうがよいのではないでしょうか。nil を返して実行が進んで幸せになるケースはあまり思い浮かびませんでした。

同意します
どうせ実アプリケーションでurandom直接呼んでる人なんていないのだから気にせず変えてしまっていいと思います

> 4.securerandom は、初回の呼び出しで Random.urandom が成功したらその後ずっと Random.urandom を使い、失敗したらずっと openssl を使うようになっています。
> これは本当に初回の呼び出しだけで振り分けるのでよいのでしょうか。具体的には、Random.urandom(0) は常に成功する（空文字列を返す）ので、実際には urandom が
> 使えない環境でも urandom に固定される可能性があります。
> 逆に、securerandom のバックエンドを動的に切り替えるようにしたら問題あるでしょうか。最初は成功していた Random.urandom が途中から失敗するようになったとき、
> openssl へフォールバックしても良いものでしょうか。また、Random.urandom が復活した場合、openssl から Random.urandom に戻すのは問題ないでしょうか。

途中でurandomが使えなくなったり使えるようになったりってのは、1の長大な引数のケースぐらいしか思いつかないので、
固定でいいのではないでしょうか。
それはそれとしてRandom.urandom(0) の件はバグじゃないのかと思います。

#2 - 09/10/2017 12:24 PM - shyouhei (Shyouhei Urabe)

kosaki (Motohiro KOSAKI) wrote:

> 1.Random.urandom は、getrandom(2) や (/dev/urandom に対する) read(2) システムコールを 1 回しか呼ばないようです。よって、要求量が大きくて 1 回では文字列を準備できないとき（例えば
> Random.urandom(100_000_000) ）、失敗して nil を返します。これは意図的でしょうか。要求量が得られるまで（または他の要因で失敗するまで）繰り返し呼ぶべきではないでしょうか。

そんな気がします。一瞬エントロピー枯渇について心配しましたが、Rubyの側で小さいurandomを大量に呼べばどうせ枯渇するんだし

昔のLinuxのmanに "Users should be very economical in the amount of seed material that they read from /dev/urandom"
とか書いてあったのがループしない経緯ではないかと想像しています。その記述はなくなったので、今となってはループしない理由はないんではないかなと思いますがいかがでしょうか。

(まあループの中でエントロピー吸い出すなら当然ブロッキングIOということになるわけで、GVL放した方がいいんじゃないのとか思わなくもないのでパッチはやや大きそうではあると感じますが)

> 2.Random.urandom は失敗するとき、nil ではなく例外を投げるほうがよいのではないでしょうか。nil を返して実行が進んで幸せになるケースはあまり思い浮かびませんでした。

同意します
どうせ実アプリケーションでurandom直接呼んでる人なんていないのだから気にせず変えてしまっていいと思います

Random.urandomという名前になったのは2.5からなので、今ならまだ実アプリケーションがどうとか悩む必要もなく、いきなり変えてOKのはずです。

> 4.securerandom は、初回の呼び出しで Random.urandom が成功したらその後ずっと Random.urandom を使い、失敗したらずっと openssl を使うようになっています。
> これは本当に初回の呼び出しだけで振り分けるのでよいのでしょうか。具体的には、Random.urandom(0) は常に成功する（空文字列を返す）ので、実際には urandom が
> 使えない環境でも urandom に固定される可能性があります。
> 逆に、securerandom のバックエンドを動的に切り替えるようにしたら問題あるでしょうか。最初は成功していた Random.urandom が途中から失敗するようになったとき、
> openssl へフォールバックしても良いものでしょうか。また、Random.urandom が復活した場合、openssl から Random.urandom に戻すのは問題ないでしょうか。

> 途中でurandomが使えなくなったり使えるようになったりってのは、1の長大な引数のケースぐらいしか思いつかないので、
> 固定でいいのではないでしょうか。

0の件は後述。Random.urandomがきちんとループして要求されたバイト数まで読むようになった場合、Random.urandomの実行結果は以下の2パターンのいずれかになるはずです。

- 成功して、必要なだけのデータが入手できる
- 失敗して、例外が上がる

さて、この整理が済んだ後、urandomが「折に触れて成功したり失敗したりする」というのは考えられるでしょうか。なんか失敗する時はずっと失敗するのではないかという気がします。なので、バックエンドの切り替えが固定かどうかはさほど重要な論点ではなくなる気がしています。

> それはそれとしてRandom.urandom(0) の件はバグじゃないのかと思います。

これに関してはまず、urandom(0)で何が返るべきかですが、空文字列だと思います。

同様の例としてread(0)で何が返るべきかという問題が「APIデザインケーススタディ」の21ページ付近に載っているので読むといいと思うんですが、要は可変長の文字列が欲しいという場合がありえて、意味のある引数として0が渡ってくる可能性がある。そこで戻り値にnilを返しうることにしてしまうと、呼び出し側でnilかどうか毎回チェックせねばならず面倒なわけです。なのでこれは食う文字列であるべき。

一方で、そうすると、初回の引数が0だったときにSecureRandomが意図しない動きになってしまうじゃないかという話はあって、これは回避する必要がありそうです。思いつくのは、

- たとえ引数が0であってもシステムコールやデバイスファイル読み出しを律儀に行い、きちんと例外をあげていく
- urandomとは別に現在利用可能な乱数元の一覧を得るAPIを新設する

くらいですが、後者はやや話が大きいので、前者かなあと思います。

#3 - 09/10/2017 12:43 PM - shyouhei (Shyouhei Urabe)

- Related to Bug #9569: SecureRandom should try /dev/urandom first added

#4 - 09/10/2017 02:52 PM - mame (Yusuke Endoh)

shyouhei (Shyouhei Urabe) wrote:

> (まあループの中でエントロピー吸い出すなら当然ブロッキングIOということになるわけで、GVL放した方がいいんじゃないのとか思わなくもないのでパッチはやや大きそうではあると感じますが)

/dev/urandom はブロックしない（少なくとも最近の OS なら）、という前提があっても、GVL 放すべきですかね?

> なんか失敗する時はずっと失敗するのではないかという気がします。

小崎さんと卜部さんの 2 人がそういうなら、大丈夫な気がしてきました。

- たとえ引数が0であってもシステムコールやデバイスファイル読み出しを律儀に行い、きちんと例外をあげていく

これはちょっと安心できない気がします。0 バイトの読み出しに対して何もせずに成功を返すというのは Ruby に限らずありがちな実装だと思うので、getrandom(2) システムコールが同じことをしても驚かないです（仮に今の実装がそうなっていなくても、man などで明記されていない限り、将来的に変わるかもしれない）。

Random.urandom(0) の挙動は今のまま変えず、securerandom は初回の読み出しで（要求が 0 バイトであろうとも）必ず 1 バイト以上読み出してみる、というのが手軽な対策だと思ってます。異論なければ以下のパッチをコミットしようと思いますが、どうでしょうか。

```
diff --git a/lib/securerandom.rb b/lib/securerandom.rb
index 6a5720c44e..e20591a64f 100644
--- a/lib/securerandom.rb
+++ b/lib/securerandom.rb
@@ -52,7 +52,7 @@ def bytes(n)
     end

     def gen_random(n)
-      ret = Random.urandom(n)
+      ret = Random.urandom(1)
       if ret.nil?
         begin
           require 'openssl'
@@ -67,10 +67,6 @@ class << self
           end
           return gen_random(n)
         end
-      elsif ret.length != n
-        raise NotImplementedError, \
-              "Unexpected partial read from random device: " \
-              "only #{ret.length} for #{n} bytes"
       else
         @rng_chooser.synchronize do
           class << self
```

#5 - 09/11/2017 01:29 AM - shyouhei (Shyouhei Urabe)

mame (Yusuke Endoh) wrote:

> shyouhei (Shyouhei Urabe) wrote:
>
> > (まあループの中でエントロピー吸い出すなら当然ブロッキングIOということになるわけで、GVL放した方がいいんじゃないのとか思わなくもないのでパッチはやや大きそうではあると感じますが)
>
> /dev/urandom はブロックしない（少なくとも最近の OS なら）、という前提があっても、GVL 放すべきですかね?

それってO_NBLOCKのディスクリプタをビジーループで読み込みってことですよね。ユーザーランドからへんにビジーループするよりカーネル側で適切に生成されてくるのをユーザーランド側はすなおにブロッキングIOしたほうがトータルのコンテキストスイッチが少なそうな気がするんですが、まあ気がするだけで定量的評価ではないです。

> - たとえ引数が0であってもシステムコールやデバイスファイル読み出しを律儀に行い、きちんと例外をあげていく
>
> これはちょっと安心できない気がします。0 バイトの読み出しに対して何もせずに成功を返すというのは Ruby に限らずありがちな実装だと思うので、getrandom(2) システムコールが同じことをしても驚かないです（仮に今の実装がそうなっていなくても、man などで明記されていない限り、将来的に変わるかもしれない）。
>
> Random.urandom(0) の挙動は今のまま変えず、securerandom は初回の読み出しで（要求が 0 バイトであろうとも）必ず 1 バイト以上読み出してみる、というのが手軽な対策だと思ってます。異論なければ以下のパッチをコミットしようと思いますが、どうでしょうか。

これは特に反対ありません。

#6 - 09/11/2017 12:46 PM - mame (Yusuke Endoh)

shyouhei (Shyouhei Urabe) wrote:

> これは特に反対ありません。

ありがとうございます。r59840 でコミットしました。（コミットログにチケット番号書き忘れた……）

#7 - 09/14/2017 11:40 AM - mame (Yusuke Endoh)

特に反対もなかったので、nil を返すのではなく RuntimeError を投げるようにしました（r59858）。

残るはシステムコールをループする点だけですが、

> それってO_NBLOCKのディスクリプタをビジーループで読み込みってことですよね。ユーザーランドからへんにビジーループするよりカーネル側で適切に生成されてくるのをユーザーランド側はすなおにブロッキングIOしたほうがトータルのコンテキストスイッチが少なそうな気がするんですが、まあ気がするだけで定量的評価ではないです。

これはちょっとよくわかりませんでした。ブロッキング IO にしても、getrandom(2) や read(2)
システムコールは途中で終わる可能性がある（つまりカーネル側で要求量きっかり生成してくれることは保証されない）ので、ユーザランドでのループはどのみち必要です。

GVL を放すかどうかは、放してもいいとは思いますが、

- getrandom(2) などがブロックすることはない（はず）
- 普通のユースケース（数十バイト程度の urandom 取得）では、ループが必要になることもほぼない

ということで、オーバーヘッドになるだけのような気はしてます。
あと、この辺の関数はスレッド周りが初期化されていないプロセス起動時にも呼ばれるので、rb_thread_call_without_gvl が単純に呼べず、さらにパッチがでかくなりそうでした。

#8 - 12/16/2017 12:43 AM - mame (Yusuke Endoh)

- Status changed from Open to Closed

開発者会議で議論し、GVL 開放せずにループで良いだろうということになりました。
そしてさきほど r61292 で修正したので閉じます。（コミットログに書き忘れました。すみません）