Ruby master - Feature #17148

stdbuf(1) support

09/04/2020 09:54 AM - os (Shigeki OHARA)

Status:	Open
Priority:	Normal
Assignee:	
Target version:	

Description

概要

Ruby インタープリターを stdbuf(1) による stdout/stderr の出力バッファ制御に対応させたい。

背景

UNIX のフィルターコマンドの多くは、標準出力の先が端末かどうかによって出力バッファリングの制御を行っており、パイプを多段に繋げているとリアルタイムな出力が行われず問題になることが多々あります。

コマンドによってはバッファリング制御するオプションを設けることでバッファリングの問題を回避することができますが、必ずしもそのようなオプションがあるとは限りません。

そこで Linux (GNU coreutils) や FreeBSD には stdbuf(1) というコマンドが用意されており、

```
# vmstat 1 | stdbuf -o L awk '$2 > 1 || $3 > 1' | cat -n
```

のようにコマンドを引数に指定して起動することにより、そのコマンドの出力のバッファリングを制御することができます。

この stdbuf(1) のしくみですが、まず libstdbuf.so という共有ライブラリが用意してあり、ここには環境変数で指定したバッファリングの設定を元に stdin/stdout/stderr を setvbuf() するコードが定義してあります。

そして stdbuf(1) はバッファリングの設定値と LD_PRELOAD=libstdbuf.so を環境変数にセットしたうえで、指定したコマンドを起動します。

すると、指定したコマンドは所与の設定にて setvbuf() された状態で動作することになります。

ちなみに、NetBSD では stdbuf(1) コマンドは用意されていませんが、stdio に環境変数を参照して setvbuf(3) する機能が組み込まれており、所定の環境変数をセットすることで同様の効果が得られます。

翻って (現在の) Ruby の STDIN/STDOUT/STDERR は、stdio そのまま利用しているわけではないので stdbuf(1) を使用することはできません。

提案

Ruby インタープリターの初期化時に stdbuf(1) がセットした環境変数を読み取って、libstdbuf.so と同様のバッファリング制御を Ruby でエミュレーションする機能を提案します。

C ではなく Ruby のコードですみませんが、実装のイメージとしては以下のような感じとなります。

```
% cat stdbuf.rb
def stdbuf(env = ENV)
  case RUBY_PLATFORM
  when /netbsd/i
    stdbuf_all = env['STDBUF']
    {
      'STDBUF0' => STDIN,
      'STDBUF1' => STDOUT,
      'STDBUF2' => STDERR,
    }.each do |key, io|
      next unless value = env[key] || value = stdbuf_all
      case value
      when 'U', 'u', 'L', 'l', '0'
        io.sync = true
      when 'F', 'f', /\A\d+\z/
        io.sync = false
      end
    end
```

```
  else  # Linux (GNU coreutils), FreeBSD, etc...
    return if !env.key?('LD_PRELOAD') || /\blibstdbuf.so\b/ !~ env['LD_PRELOAD']
    {
      '_STDBUF_I' => STDIN,
      '_STDBUF_O' => STDOUT,
      '_STDBUF_E' => STDERR,
    }.each do |key, io|
      next unless value = env[key]
      case value
      when '0', 'L'
        io.sync = true
      when 'B', /\A\d+(?:[kKMGTPEZY]B?)?\z/
        io.sync = false
      end
    end
  end
end

stdbuf(ENV)
% ruby -I. -rstdbuf -e'loop{puts Time.now; sleep 1}' | cat
^C-e:1:in `sleep': Interrupt
        from -e:1:in `block in <main>'
        from -e:1:in `loop'
        from -e:1:in `<main>'

% stdbuf -o 0 ruby -I. -rstdbuf -e'loop{puts Time.now; sleep 1}' | cat
2018-02-23 12:43:05 +0900
2018-02-23 12:43:06 +0900
2018-02-23 12:43:07 +0900
```

libstdbuf.so が利用しているのと同じ環境変数を、 Ruby の側でも自前で読み取り、バッファリングの制御を行うイメージです。

議論・課題

- Ruby インタープリター本体に組み込む必要があるか?
  - 上記コード例のように require などする方向もあるかもしれませんが、
  - 既存のスクリプトに手を入れなくて良いのが stdbuf のメリットなので、
  - Ruby インタープリターに組み込むのが良いかと思います。
- コード例は IO#sync= でバッファリング制御を雑に行っています
  - 実用上これでも多くのユースケースをカバーできている気もします
  - オリジナルの libstdbuf.so 相当の細かい制御ができるとより良いとは思います
- 互換性
  - 私の確認したのは GNU coreutils (Linux, Cygwin), FreeBSD, NetBSD のみです
  - 他にもあるかもしれません
  - ビルド時に stdbuf に対応しているか判断する必要があるかもしれません

History

#1 - 09/04/2020 03:42 PM - nobu (Nobuyoshi Nakada)

バッファリングのモードを直接知る方法ってないですよねぇ。

```
diff --git a/io.c b/io.c
index 0d6e2178573..f69aa14934d 100644
--- a/io.c
+++ b/io.c
@@ -8162,6 +8162,35 @@ prep_stdio(FILE *f, int fmode, VALUE klass, const char *path)
     return io;
 }

+static void
+prep_flush_mode(VALUE io)
+{
+#ifndef _WIN32                  /* FD-base system only */
+    rb_io_t *fptr = RFILE(io)->fptr;
+    int savefd = dup(fptr->fd);
+    int pipefds[2] = {-1, -1};
+    if (savefd == -1) return;
+    if (pipe(pipefds) == 0) {
```

```
+         fd_set rfdset;
+         struct timeval zero = {0, 0};
+         FILE *f = fptr->stdio_file;
+         int failed = dup2(pipefds[1], fptr->fd) == -1;
+         (void)close(pipefds[1]);
+         if (!failed && fputc('\n', f) != EOF) { /* _IONBF or _IOLBF */
+             FD_ZERO(&rfdset);
+             FD_SET(pipefds[0], &rfdset);
+             if (select(pipefds[0] + 1, &rfdset, NULL, NULL, &zero) == 1) {
+                 fptr->mode |= FMODE_SYNC;
+             }
+             fflush(f);      /* fpurge(f) */
+             dup2(savefd, fptr->fd);
+         }
+         (void)close(pipefds[0]);
+     }
+     (void)close(savefd);
+#endif
+}
+
 VALUE
 rb_io_prep_stdin(void)
 {
@@ -13572,6 +13601,7 @@ Init_IO(void)
     rb_stdin  = rb_io_prep_stdin();
     rb_stdout = rb_io_prep_stdout();
     rb_stderr = rb_io_prep_stderr();
+    prep_flush_mode(rb_stdout);

     rb_global_variable(&rb_stdin);
     rb_global_variable(&rb_stdout);
```

#2 - 09/07/2020 04:12 AM - os (Shigeki OHARA)

なるほど、そんなやり方が……。

それだと stdbuf の有無など環境の差異を気にしなくて良さそうですね。