Ruby I/O 機構の改善
— stdio considered harmful —

田中哲

akr@m17n.org

産業技術総合研究所 情報技術研究部門

2005–06–02

目的: Ruby I/O の問題修正

- Nonblocking I/O
- readpartial の提案
- スレッドの read 待ち
- EOF フラグ
- Solaris FILE 256 個制限
- 双方向ストリーム判定
- 読み込み・書き込みの連続
- ungetc で SEGV
- errno クリア

Nonblocking I/O

用途:

- write でプロセス全体のブロック防止 (他のスレッドをブロックさせない)
- signal handling

問題: stdio と組み合わせると書き込みデータが消える

stdio を使う限り修正困難

readpartial

用途: 読み込めるようになるまでブロックし、その時点で読み込めるだけ読み込む

問題: FILE 構造体の中身を覗かなければならない

ポータブルには出来ない

スレッドの **read** 待ち

用途: 読めないときは他のスレッドを動かす

問題: FILE 構造体の中身を覗かなければならない

ポータブルには出来ない

Solaris FILE 256 個制限

用途: 256 個以上ファイル・ソケットをオープンしたい

問題: FILE 構造体の fd メンバは un-signed char

stdio を使う限り修正困難

他の問題

- 双方向ストリームかどうかの自動判定に stdio が邪魔
- stdio の EOF フラグはポータブルでない
- 読み込み・書き込みの切替えが手間
- ungetc での SEGV を防ぐのが手間
- stdio が勝手に errno をクリア
- etc.

原因

1. stdio
2. Ruby
3. Unix

問題と解決

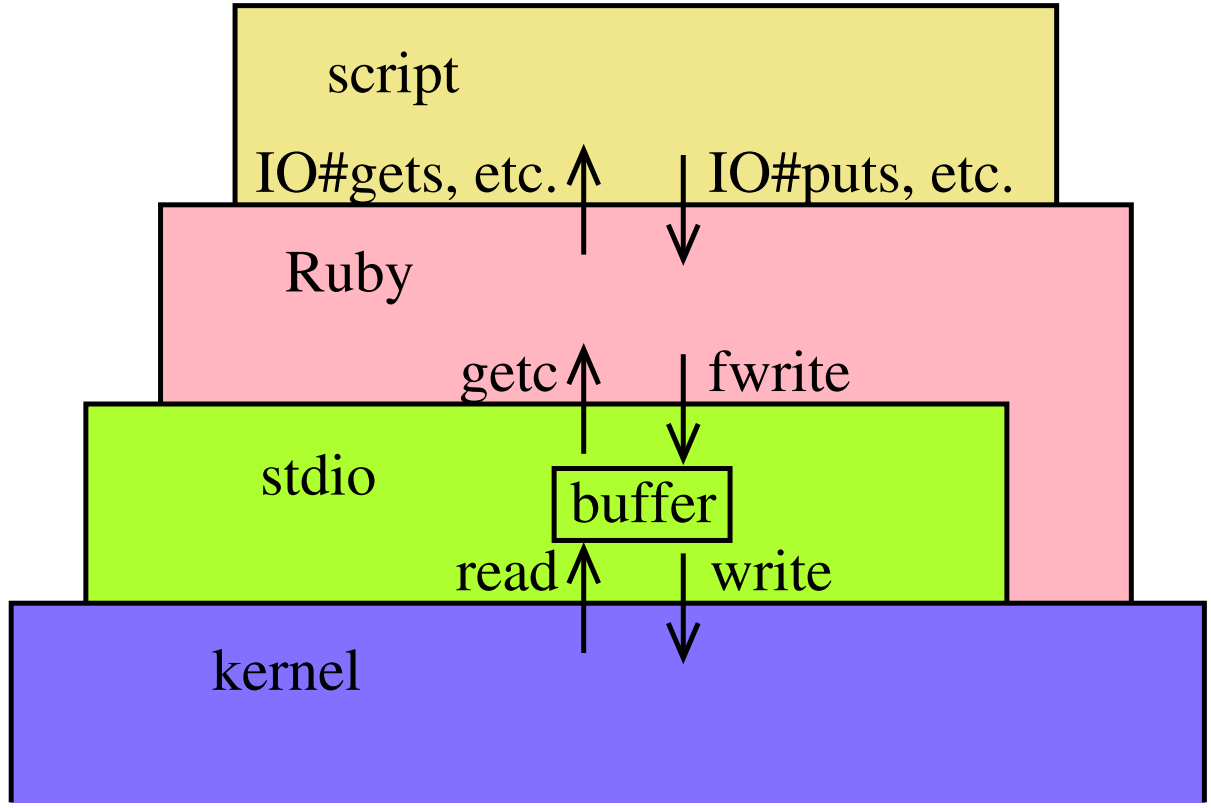
- 実装の問題 – stdio の品質
  - Ruby 1.9 開発版 : 使わない
  - Ruby 1.8 安定版 : だましだまし使う
- 仕様の問題 – Unix と Ruby の挙動
  - Ruby: 可能なところから変えていく
  - Unix: あきらめてがんばる

stdio の機能

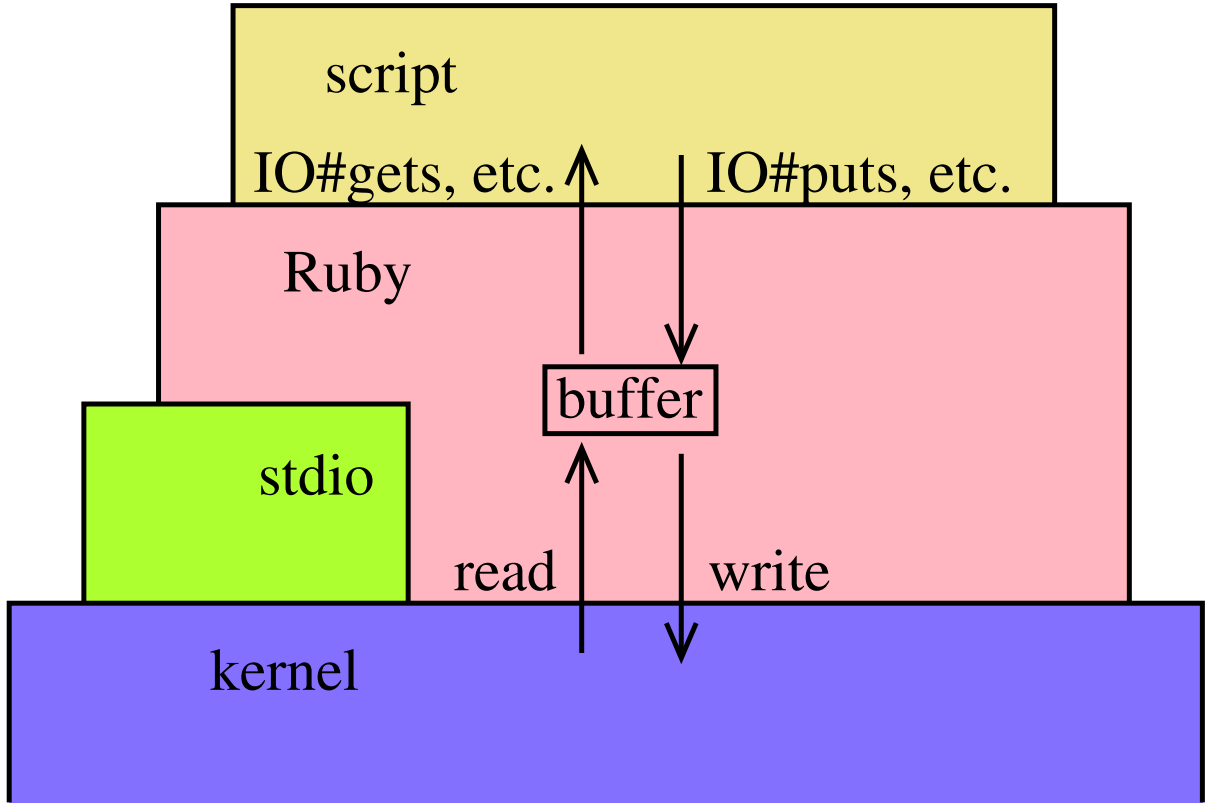
C の標準入出力ライブラリ

- FILE 構造体を使う I/O ルーチン
  (getc, fwrite, ungetc, etc.)
- 書式付き入出力 (printf, scanf)
- プロセス I/O (popen, pclose)
- etc.

Ruby I/O レイヤ (1.8 安定版)



Ruby I/O レイヤ (1.9 開発版)

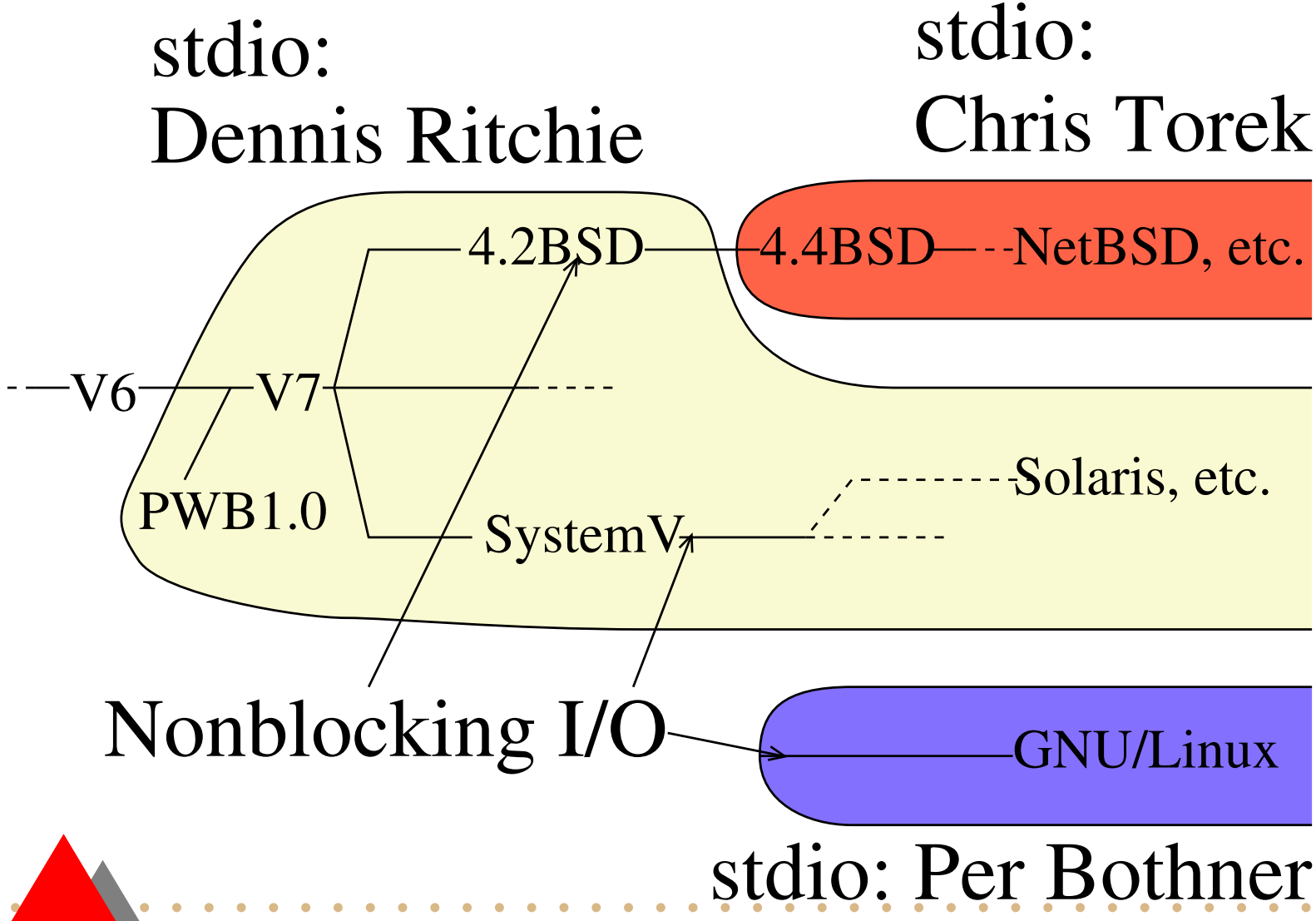


stdio 問題: 背景

標準入出力ライブラリは1975年頃に Dennis Ritchie が作成した。これは、Mike Lesk が書いたポータブルな入出力ライブラリの主要な改定版であった。驚くべきことであるが、15年以上にわたって標準入出力ライブラリはほとんど書き直されていない。

W. Richard Stevens, 1992

Unix の系譜と stdio



stdio v.s. Nonblocking I/O

書き込んでいないデータが消える

1. fd を Nonblocking I/O に設定
2. fwrite
3. fflush
4. write
5. EAGAIN エラー
6. バッファが空になる

fflush (V7, 部分)

```
/* バッファを空にする */
iop->_ptr = base;
iop->_cnt = BUFSIZ;

if (write(fileno(iop), base, n)
      != n) {
   /* 全部書き込めなければ失敗 */
   iop->_flag |= _IOERR;
   return(EOF);
}
```

fflush (4.4BSD, 部分)

```
/* バッファを空にする */
fp->_p = p;
fp->_w = t & (__SLBF|__SNBF) ? 0 :
for (; n > 0; n -= t, p += t) {
  t = (*fp->_write)(fp->_cookie,
                    (char *)p, n);
  if (t <= 0) {
    /* write のエラーで失敗 */
    fp->_flags |= __SERR;
    return (EOF);
  }
}
```

fflush (glibc, 部分)

```
/* partial write 処理および
   エラーでフラグを立てる */
count = _IO_SYSWRITE (fp, data, to
...
/* バッファを空にする */
fp->_IO_write_base = fp->_IO_write
```

fflush の問題

- write が失敗してもバッファは空になる
  データが書き込まれなくてもデータは
  なくなる (V7, 4.4BSD, glibc)
- write の結果で、エラーと partial write
  を区別していない
  途中まで書き込んだあとに続きを書き
  込まない (V7)

データ蒸発問題への対応

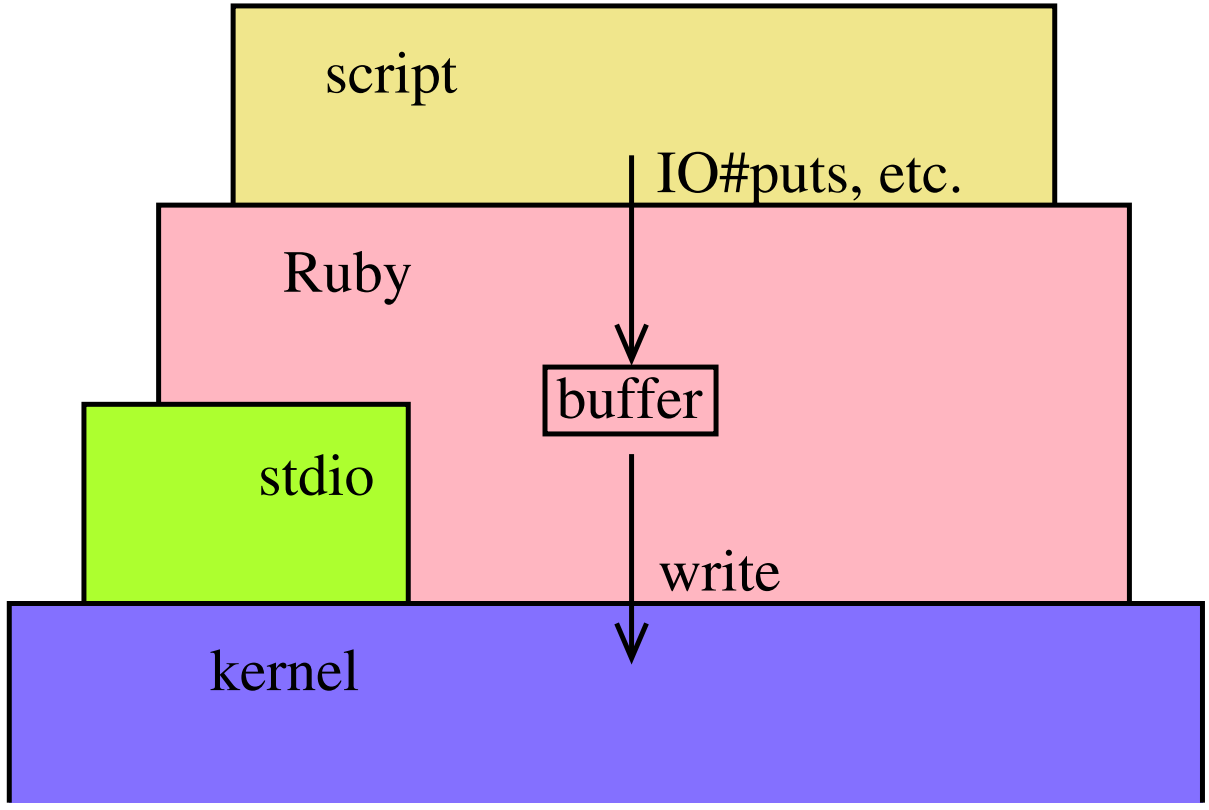
Ruby 1.9 stdio を捨てる
write を直接呼び出す
EAGAIN エラーではデータを捨てない

Ruby 1.8 可能なら stdio を迂回
sync mode では write を直接呼び出す
EAGAIN エラーではデータを捨てない

write を直接呼び出す

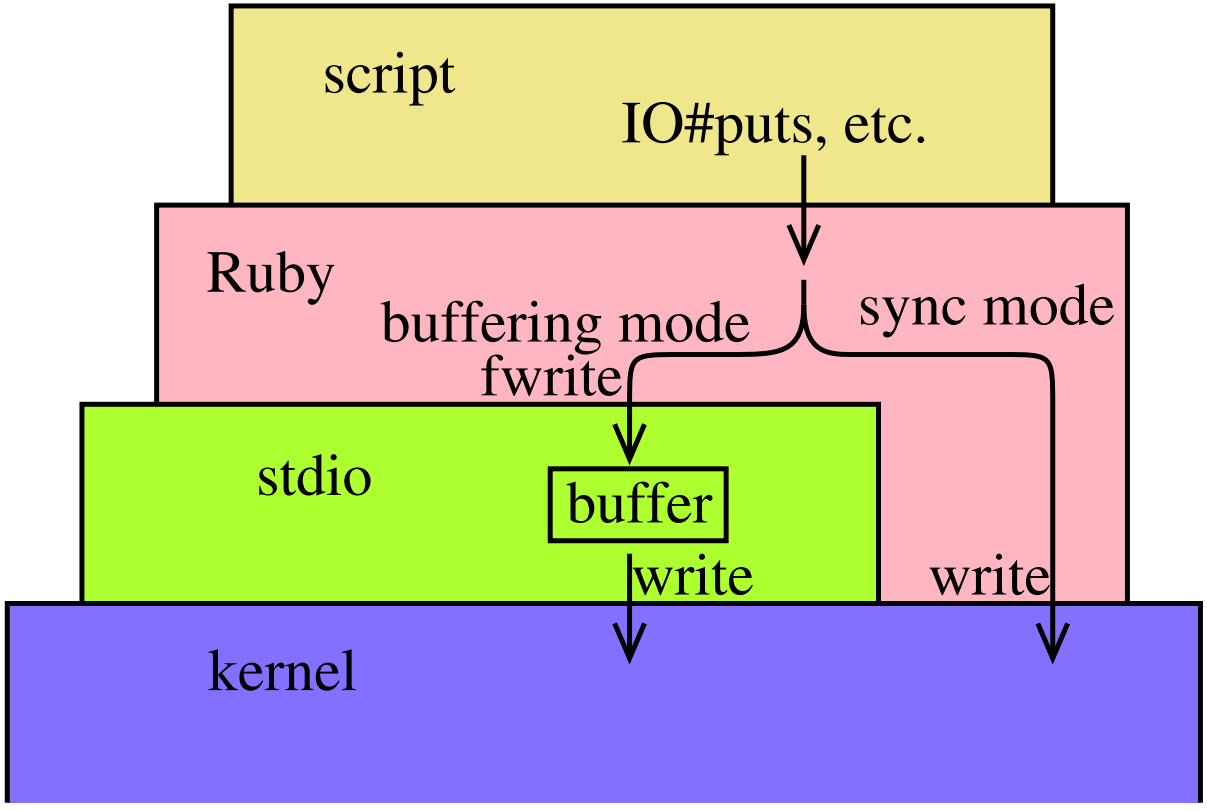
データ蒸発問題: Ruby 1.9

write を直接呼び出す



データ蒸発問題: Ruby 1.8

sync mode で write を直接呼び出す



データ蒸発問題: 現状

Ruby 1.9 修正済み

Ruby 1.8 部分的に修正済み

- sync mode は問題なし
- buffering mode は問題あり
- socket 等はデフォルトでは sync mode
- 必要なタイミングで stdio のバッファの破棄を行う

readpartial/スレッドの read 待ち

要求:

- その時点で読み出せるだけ読みだしたい (readpartial)
- 即座に読めないならスレッドを context switch して他のスレッドを動かす (スレッドの read 待ち)
- ポータブルに実装したい

即座に読み出せるかどうか判断したい

getc はブロックするか?

「読み込みバッファが空でない」or
「select して readable」

||

「getc はブロックしない」

読み込みバッファは空?

stdio には読み込みバッファが空かどうかを判定する関数がない

getc はブロックするか?

「select して readable」

↓

「getc はブロックしない」

- 1byte 単位に select すれば判断できる
- 効率が悪すぎる
  (1byte 毎にシステムコールが必要)

非現実的

FILE 構造体の中身を覗く

システム	バッファにデータがあるか?
V7	`f->_cnt > 0`
4.4BSD	`f->_r > 0`
glibc	`f->_IO_read_ptr != f->_IO_read_end`
	etc.

- バッファが空でなければシステムコールを使わずにブロックしないことを確認できる
- ポータブルではない (例: 64bit Solaris では動かない)

スレッドの **read** 待ち

- Ruby は (以前から) FILE 構造体の中身を覗いてスレッドの read 待ちを扱う
- ポータブルでないため、64bit Solaris など動かない環境がある

64bit Solaris の FILE 構造体:

```
struct __FILE_TAG {
  long    __pad[16];
};
```

スレッドの **read** 待ちの改善

- Ruby 1.9 では読み込みバッファが空かどうかポータブルに判定できる
- 64bit Solaris を含めて問題なく動く

readpartial

1byte 以上読み出せるようになるまで待ち、その時点で読み出せるだけ読み出す

必要な状況:

- 読み出す長さが不明
- 読み出すデータの終端も不明
- 行単位読み込みや固定長読み込みも併用したい

readpartial の用途

- telnet.rb - 読んでからプロンプトを調べる
- port forwarder - 読んだ端から転送する
- cvs protocol - 圧縮された 1 行を読む
- etc.

readpartial 実装

1. 読み込みバッファにデータがあったらそのデータを読む
2. 読み込みバッファにデータがなかったら read システムコールで読む

バッファが空かどうかの判定が必要

- Ruby 1.8 では FILE 構造体を覗く
- Ruby 1.9 ではポータブルに実装できる

Solaris FILE 256 個制限

```
struct __FILE_TAG /* needs to be h
{
 ssize_t          _cnt;    /* number o
 unsigned char *_ptr;      /* next cha
 unsigned char *_base;     /* the buff
 unsigned char _flag;      /* the stat
 unsigned char _file;      /* UNIX Sys
 ...
};
```

$0 \leq$ fd $<$ 256 しか扱えない

Ruby 1.9 では FILE 構造体は必要ないか ぎり使わない

改善結果

	旧 Ruby 1.8	新 Ruby 1.8	Ruby 1.9
スレッドの read 待ち	△	△	○
errno をクリアする stdio 実装	△	△	○
ungetc	△	△	○
双方向ストリーム	△	△	○
読み込み・書き込みの切替え	△	△	○
Nonblocking write におけるデータ消滅	×	△	○
Nonblocking IO#read の動作	×	×	○
readpartial の提案・実装	×	○	○
Solaris の 256 個制限	×	×	○
EOF フラグ	×	△	○
拡張ライブラリの非互換性	○	○	△
テキストモード	△	△	△
popen のポータビリティ	○	○	○
Windows 上の双方向 popen	○	○	×
stdio を使うライブラリとの協調	○	○	×

今後の予定

- 安全な signal handling
- バウンダリを考慮したバッファリングで高速化
- FD_SETSIZE (a C10K problem)

まとめ

- stdio のバッファリングは使わない
- Nonblocking I/O が使えるようになる
- 様々な問題が解決する