Ruby - Bug #703

string output duplication occurs if the same file descriptor written to in different threads

11/01/2008 02:39 AM - rogerdpack (Roger Pack)

Status:	Closed		
Priority:	Normal		
Assignee:			
Target version:			
ruby -v:		**Backport:**	

Description

=begin
ruby-dev:32566

a = Thread.new { p '4'}
b = Thread.new { p '3' }
a.join
b.join

results in
"4"
"3"
"4"
"3"

because the buffer is modified simultaneously by two threads within io_fflush [I think].
=end

History

#1 - 11/03/2008 02:34 AM - gnufied (hemant kumar)

=begin
I do not see this problem, with:

ruby 1.9.0 (2008-09-30 revision 0) [i686-linux]

=end

#2 - 11/03/2008 09:06 AM - matz (Yukihiro Matsumoto)

=begin
Hi,

In message "Re: [ruby-core:19668] [Bug #703] string output duplication occurs if the same file descriptor written to in different threads"
on Sat, 1 Nov 2008 02:38:52 +0900, Roger Pack redmine@ruby-lang.org writes:

|ruby-dev:32566

It's not related to ruby-dev:32566, I think.

|a = Thread.new { p '4'}
|b = Thread.new { p '3' }
|a.join
|b.join
|
|results in
|"4"
|"3"
|"4"
|"3"
|
|because the buffer is modified simultaneously by two threads within io_fflush [I think].

It doesn't happen on either trunk nor 1.8.7. What's your version, and
platform?

```
matz.
```

=end

#3 - 11/03/2008 09:46 AM - nobu (Nobuyoshi Nakada)

=begin
Hi,

At Mon, 3 Nov 2008 09:04:37 +0900,
Yukihiro Matsumoto wrote in [ruby-core:19676]:

> It doesn't happen on either trunk nor 1.8.7. What's your version, and
> platform?

It could reproduced with full ruby, but not with miniruby nor
full ruby with disabling gems.

```
$ ./ruby -v bug-703.rb
ruby 1.9.0 (2008-11-03 revision 20092) [i686-linux]
"4"
"3"
"4"
"3"

$ ./ruby --disable-gems bug-703.rb
"4"
"3"
```

Interestingly, -v makes it occur again.

```
$ ./ruby -v --disable-gems bug-703.rb
ruby 1.9.0 (2008-11-03 revision 20092) [i686-linux]
"4"
"3"
"4"
"3"
```

But -v after --disable-gems doesn't.

```
$ ./ruby --disable-gems -v bug-703.rb
ruby 1.9.0 (2008-11-03 revision 20092) [i686-linux]
"4"
"3"
```

--
Nobu Nakada

=end

#4 - 11/04/2008 02:46 AM - nobu (Nobuyoshi Nakada)

=begin
Hi,

At Mon, 3 Nov 2008 09:44:44 +0900,
Nobuyoshi Nakada wrote in [ruby-core:19678]:

> > It doesn't happen on either trunk nor 1.8.7. What's your version, and
> > platform?
>
> It could reproduced with full ruby, but not with miniruby nor
> full ruby with disabling gems.

It seems a mere timing and probability issue. And seems solved
by serializing write operations, as Roger figured out.

```
Index: gc.c
===================================================================
--- gc.c (revision 20101)
+++ gc.c (working copy)
@@ -1509,4 +1509,5 @@ gc_mark_children(rb_objspace_t *objspace
             gc_mark(objspace, obj->as.file.fptr->writeconv_pre_ecopts, lev);
             gc_mark(objspace, obj->as.file.fptr->encs.ecopts, lev);
```

```
+            gc_mark(objspace, obj->as.file.fptr->write_lock, lev);
         }
         break;
Index: io.c
===================================================================
--- io.c (revision 20101)
+++ io.c (working copy)
@@ -525,9 +525,27 @@ rb_write_internal(int fd, void *buf, siz
 }

+static long
+io_writable_length(rb_io_t *fptr, long l)
+{
+    if (PIPE_BUF < l &&
+        !rb_thread_alone() &&
+        wsplit_p(fptr)) {
+        l = PIPE_BUF;
+    }
+    return l;
+}
+
+static VALUE
+io_flush_buffer(VALUE arg)
+{
+    rb_io_t *fptr = (rb_io_t *)arg;
+    long l = io_writable_length(fptr, fptr->wbuf_len);
+    return rb_write_internal(fptr->fd, fptr->wbuf+fptr->wbuf_off, l);
+}
+
 static int
 io_fflush(rb_io_t *fptr)
 {
-    int r, l;
-    int wbuf_off, wbuf_len;
+    long r;

     rb_io_check_closed(fptr);
@@ -540,13 +558,5 @@ io_fflush(rb_io_t *fptr)
     if (fptr->wbuf_len == 0)
         return 0;
-    wbuf_off = fptr->wbuf_off;
-    wbuf_len = fptr->wbuf_len;
-    l = wbuf_len;
-    if (PIPE_BUF < l &&
-        !rb_thread_alone() &&
-        wsplit_p(fptr)) {
-        l = PIPE_BUF;
-    }
-    r = rb_write_internal(fptr->fd, fptr->wbuf+wbuf_off, l);
+    r = rb_mutex_synchronize(fptr->write_lock, io_flush_buffer, (VALUE)fptr);
     /* xxx: Other threads may modify wbuf.
      * A lock is required, definitely. */
@@ -732,9 +742,23 @@ make_writeconv(rb_io_t *fptr)

 /* writing functions */
+struct binwrite_arg {
+    rb_io_t *fptr;
+    VALUE str;
+    long offset;
+    long length;
+};
+
+static VALUE
+io_binwrite_string(VALUE arg)
+{
+    struct binwrite_arg *p = (struct binwrite_arg *)arg;
+    long l = io_writable_length(p->fptr, p->length);
+    return rb_write_internal(p->fptr->fd, RSTRING_PTR(p->str)+p->offset, l);
+}

 static long
 io_binwrite(VALUE str, rb_io_t *fptr, int nosync)
 {
-    long len, n, r, l, offset = 0;
+    long len, n, r, offset = 0;
```

```
     len = RSTRING_LEN(str);
@@ -745,7 +769,10 @@ io_binwrite(VALUE str, rb_io_t *fptr, in
         fptr->wbuf_capa = 8192;
         fptr->wbuf = ALLOC_N(char, fptr->wbuf_capa);
+ fptr->write_lock = rb_mutex_new();
     }
     if ((!nosync && (fptr->mode & (FMODE_SYNC|FMODE_TTY))) ||
         (fptr->wbuf && fptr->wbuf_capa <= fptr->wbuf_len + len)) {
+ struct binwrite_arg arg;
+
         /* xxx: use writev to avoid double write if available */
         if (fptr->wbuf_len && fptr->wbuf_len+len <= fptr->wbuf_capa) {
@@ -767,12 +794,10 @@ io_binwrite(VALUE str, rb_io_t *fptr, in
      rb_io_check_closed(fptr);
  }
+ arg.fptr = fptr;
+ arg.str = str;
+ arg.offset = offset;
       retry:
-        l = n;
-        if (PIPE_BUF < l &&
-            !rb_thread_alone() &&
-            wsplit_p(fptr)) {
-            l = PIPE_BUF;
-        }
- r = rb_write_internal(fptr->fd, RSTRING_PTR(str)+offset, l);
+ arg.length = n;
+ r = rb_mutex_synchronize(fptr->write_lock, io_binwrite_string, (VALUE)&arg);
  /* xxx: other threads may modify given string. */
         if (r == n) return len;
@@ -3040,4 +3065,17 @@ finish_writeconv(rb_io_t *fptr, int nora
 }

+struct finish_writeconv_arg {
+    rb_io_t *fptr;
+    int noraise;
+};
+
+static VALUE
+finish_writeconv_sync(VALUE arg)
+{
+    struct finish_writeconv_arg *p = (struct finish_writeconv_arg *)arg;
+    finish_writeconv(p->fptr, p->noraise);
+    return Qnil;
+}
+
 static void
 fptr_finalize(rb_io_t *fptr, int noraise)
@@ -3045,5 +3083,13 @@ fptr_finalize(rb_io_t *fptr, int noraise
     int ebadf = 0;
     if (fptr->writeconv) {
-        finish_writeconv(fptr, noraise);
+ if (fptr->write_lock) {
+     struct finish_writeconv_arg arg;
+     arg.fptr = fptr;
+     arg.noraise = noraise;
+     rb_mutex_synchronize(fptr->write_lock, finish_writeconv_sync, (VALUE)&arg);
+ }
+ else {
+     finish_writeconv(fptr, noraise);
+ }
     }
     if (fptr->wbuf_len) {
Index: include/ruby/io.h
===================================================================
--- include/ruby/io.h (revision 20101)
+++ include/ruby/io.h (working copy)
@@ -74,4 +74,5 @@ typedef struct rb_io_t {
     int writeconv_initialized;

+    VALUE write_lock;
 } rb_io_t;

@@ -134,4 +135,5 @@ typedef struct rb_io_t {
```

```
     fp->encs.ecflags = 0;\
     fp->encs.ecopts = Qnil;\
+    fp->write_lock = 0;\
 } while (0)
```

--
Nobu Nakada

=end

#5 - 11/08/2008 05:47 AM - nobu (Nobuyoshi Nakada)

- Status changed from Open to Closed

- % Done changed from 0 to 100

=begin
Applied in changeset r20144.
=end

#6 - 11/11/2008 02:38 PM - rogerdpack (Roger Pack)

=begin
Thank you that fixed it. I don't have OS X to be able to tell if the test named after ruby-dev:32566 is also fixed but wouldn't be surprised if it was.
Regards.
-=R
=end