Ruby trunk - Bug #14413

`-n` and `-p` flags break when stdout is closed

01/28/2018 01:35 AM - josh.cheek (Josh Cheek)

Status:	Feedback		
Priority:	Normal		
Assignee:			
Target version:			
ruby -v:	ruby 2.5.0preview1 (2017-10-10 trunk 60153) [x86_64-darwin16]	**Backport:**	2.3: UNKNOWN, 2.4: UNKNOWN, 2.5: UNKNOWN

Description

Ruby generally works well within a pipeline. The -n and -p flags are incredibly useful. However, it is common practice to use programs like head and sed, which will close the pipe after completing their job. This is convenient, because often it limits an expensive amount of output to a workable subset. I can figure out the pipeline for a subset of input, and then remove the limiting function.

However, Ruby explodes with -e:1:inwrite': Broken pipe @ io_write - (Errno::EPIPE), when it writes the current line to stdout. When in a line oriented mode, and stdout closes, I think it should exit successfully (so it doesn't break bash scripts withpipe fail`set) and silently (no error message), hence marking this a bug report rather than a feature request.

I've attached a screenshot to show that this is how every other program works, that I tried.

```
git clone https://github.com/jquery/esprima
cd esprima/test/fixtures/
ls                                                                    | head -1 # ls
find . -type f -name '*json'                                          | head -1 # find
find . -type f -name '*json' | head -1 | xargs cat | jq .             | head -1 # jq
find . -type f -name '*json' | grep -v JSX | grep -v tokenize         | head -1 # grep
find . -type f -name '*json' | sed -E '/JSX|tokenize/d'               | head -1 # sed
find . -type f -name '*json' | awk '/JSX|tokenize/ { next }; { print }' | head -1 # awk
find . -type f -name '*json' | perl -e 'while(<>) { /JSX|tokenize/ || print }' | head -1 # perl
find . -type f -name '*json' | ruby -ne 'print unless /JSX|tokenize/'  | head -1 # ruby :(
```

History

#1 - 01/28/2018 01:46 AM - josh.cheek (Josh Cheek)

Actually, maybe it should choose this behavior when -e is passed, as well. I often use -e, standalone. Here's a video from just a week ago where I used it to filter output (though, as I watch it now, I realized I could have done the same thing with the -n flag). Looking at the output, it's easy to imagine myself passing that into further pipeline segments.

https://vimeo.com/251434577

#2 - 01/28/2018 01:08 PM - nobu (Nobuyoshi Nakada)

- Status changed from Open to Feedback

In common, SIGPIPE will terminate the process printing to closed pipe.
And few programs "exit successfully" in such case.

```
$ yes | cat | head -1; echo ${PIPESTATUS[@]}
y
141 141 0

$ yes | grep ^ | head -1; echo ${PIPESTATUS[@]}
y
141 141 0

$ yes | sed '' | head -1; echo ${PIPESTATUS[@]}
y
141 141 0
```

You can set SIGPIPE signal handler to system default to behave as above.

```
$ yes | ruby -pe 'BEGIN{trap(:PIPE,:SYSTEM_DEFAULT)}' | head -1; echo ${PIPESTATUS[@]}
```

```
y
141 141 0
```

Or require a small library to do it.

```
# sigpipe.rb
BEGIN{trap(:PIPE,:SYSTEM_DEFAULT)}

$ yes | ruby -r./sigpipe -pe '' | head -1; echo ${PIPESTATUS[@]}
y
141 141 0
```

#3 - 01/28/2018 02:48 PM - nobu (Nobuyoshi Nakada)

A patch to exit with SIGPIPE when EPIPE if -n or -p option is given.

```
diff --git i/error.c w/error.c
index 7cbbf101e0..2f454d01da 100644
--- i/error.c
+++ w/error.c
@@ -53,6 +53,7 @@ int rb_str_end_with_asciichar(VALUE str, int c);
 VALUE rb_eEAGAIN;
 VALUE rb_eEWOULDBLOCK;
 VALUE rb_eEINPROGRESS;
+VALUE rb_eEPIPE;
 static VALUE rb_mWarning;
 static VALUE rb_cWarningBuffer;

@@ -1816,6 +1817,9 @@ set_syserr(int n, const char *name)
       case EINPROGRESS:
         rb_eEINPROGRESS = error;
         break;
+      case EPIPE:
+        rb_eEPIPE = error;
+        break;
     }

     rb_define_const(error, "Errno", INT2NUM(n));
diff --git i/parse.y w/parse.y
index c8acac5d2c..57dbc7f7db 100644
--- i/parse.y
+++ w/parse.y
@@ -10571,11 +10571,13 @@ rb_parser_warn_location(VALUE vparser, int warn)
     p->warn_location = warn;
 }

+extern VALUE rb_eEPIPE;
 static NODE *
 parser_append_options(struct parser_params *p, NODE *node)
 {
     static const YYLTYPE default_location = {{1, 0}, {1, 0}};
     const YYLTYPE *const LOC = &default_location;
+    int ignore_epipe = p->do_print || p->do_loop;

     if (p->do_print) {
     NODE *print = NEW_FCALL(rb_intern("print"),
@@ -10584,6 +10586,19 @@ parser_append_options(struct parser_params *p, NODE *node)
     node = block_append(p, node, print);
     }

+    if (ignore_epipe) {
+    /* rescue Errno::EPIPE then raise SignalException.new(SIGPIPE) */
+    VALUE signo = INT2FIX(SIGPIPE);
+    VALUE exc = rb_class_new_instance(1, &signo, rb_eSignal);
+    NODE *res = NEW_RESBODY(NEW_LIST(NEW_LIT(rb_eEPIPE, LOC), LOC),
+                NEW_FCALL(rb_intern("raise"),
+                      NEW_LIST(NEW_LIT(exc, LOC), LOC),
+                      LOC),
+                0,
+                LOC);
+    node = NEW_RESCUE(node, res, 0, LOC);
+    }
+
     if (p->do_loop) {
     if (p->do_split) {
```

```
        NODE *split = NEW_GASGN(rb_intern("$F"),
```

#4 - 01/28/2018 03:57 PM - josh.cheek (Josh Cheek)

nobu (Nobuyoshi Nakada) wrote:

> A patch to exit with SIGPIPE when EPIPE if -n or -p option is given.

□□ ♥□ □□

#5 - 01/29/2018 01:05 AM - nobu (Nobuyoshi Nakada)

It didn't work with END {}.

```
# silent_epipe.rb
# -*- frozen-string-literal :true -*-
BEGIN {
  if Errno.const_defined?("EPIPE") and Signal.list["PIPE"]
    class SystemCallError
      prepend Module.new {
        sigpipe = SignalException.new("PIPE").freeze
        define_method(:exception) {Errno::EPIPE === self ? sigpipe : super}
      }
    end
  end
}
```

```
$ yes | ruby -w -r./silent_epipe -pe 'END{STDERR.puts :END}' | head -1; echo ${PIPESTATUS[@]}
y
END
141 141 0
```

#6 - 01/29/2018 02:17 AM - nobu (Nobuyoshi Nakada)

It may be better to translate Errno::EPIPE to SIGPIPE only in STDOUT.write.

```
# silent_epipe.rb
# -*- frozen-string-literal :true -*-
BEGIN {
  if Errno.const_defined?("EPIPE") and Signal.list["PIPE"]
    class << STDOUT
      prepend Module.new {
        def write(*)
          super
        rescue Errno::EPIPE
          raise SignalException.new("PIPE")
        end
      }
    end
  end
}
```

Files

Screen Shot 2018-01-27 at 7.27.49 PM.png	458 KB	01/28/2018	josh.cheek (Josh Cheek)