Ruby master - Bug #12689

Thread isolation of $~ and $_

08/19/2016 06:37 AM - headius (Charles Nutter)

Status:	Open		
Priority:	Normal		
Assignee:			
Target version:			
ruby -v:		**Backport:**	2.1: UNKNOWN, 2.2: UNKNOWN, 2.3: UNKNOWN

Description

We are debating what is correct behavior now, and what should be correct behavior in the future, for the thread-visibility of the special variables %~ and $_

We have several examples from https://github.com/jruby/jruby/issues/3031 that seem to exhibit conflicting behavior...or at least the behavior is unexpected in many cases.

```
$ ruby23 -e 'p = proc { p $~; "foo" =~ /foo/ }; Thread.new {p.call}.join; Thread.new{p.call}.join'
nil
nil

$ ruby23 -e 'def foo; proc { p $~; "foo" =~ /foo/ }; end; p = foo; Thread.new {p.call}.join; Threa
d.new{p.call}.join'
nil
#<MatchData "foo">

$ ruby23 -e 'p = proc { p $~; "foo" =~ /foo/ }; def foo(p); Thread.new {p.call}.join; Thread.new{p
.call}.join; end; foo(p)'
nil
#<MatchData "foo">

$ ruby23 -e 'class Foo; P = proc { p $~; "foo" =~ /foo/ }; def foo; Thread.new {P.call}.join; Thre
ad.new{P.call}.join; end; end; Foo.new.foo'
nil
#<MatchData "foo">

$ ruby23 -e 'def foo; p = proc { p $~; "foo" =~ /foo/ }; Thread.new {p.call}.join; Thread.new{p.ca
ll}.join; end; foo'
nil
nil

$ ruby23 -e 'def foo; p = proc { p $~; "foo" =~ /foo/ }; bar(p); end; def bar(p); Thread.new {p.ca
ll}.join; Thread.new{p.call}.join; end; foo'
nil
#<MatchData "foo">
```

These cases exhibit some oddities in whether $~ (and presumably $_) are shared across threads.

The immediate thought is that they should be both frame and thread-local...but ko1 points out that such a change would break cases like this:

```
def foo
  /foo/ =~ 'foo'
  Proc.new{
    p $~
  }
end

Thread.new{
  foo.call
}.join
```

So there's a clear conflict here. Users sometimes expect the $~ value to be shared across threads (at least for read, as in ko1's example) and sometimes do not want it shared at all (as in the case of https://github.com/jruby/jruby/issues/3031

Now we discuss.

Related issues:

Related to Ruby master - Bug #8444: Regexp vars $~ and friends are not thread...	**Open**

History

#1 - 08/19/2016 06:39 AM - headius (Charles Nutter)

To clarify the one-liners' behavior: when the thread's top-level frame is the same as a proc's frame that it calls, it will see thread-local values. When the proc's frame is not the top-level frame for the thread, the memory location for $~ will be shared across all threads.

#2 - 08/19/2016 09:53 AM - Eregon (Benoit Daloze)

Maybe $~ is always set in the surrounding method frame, but never in a block frame?
There is still a lot of weird cases to explain though.

#3 - 08/19/2016 10:17 AM - darix (Marcus Rückert)

I wonder, if moving away from those special $ variables to explicit match objects wouldn't be a possible solution to this.

#4 - 08/19/2016 06:12 PM - headius (Charles Nutter)

Marcus Rückert wrote:

> I wonder, if moving away from those special $ variables to explicit match objects wouldn't be a possible solution to this.

If you always use the returned MatchData then you can avoid these problems. This only affects consumers of the implicit $~ variable.

Unfortunately, that also includes some core *methods* that access the $_ variable, so there's possibility of steppping on threading even if you never use the implicit variables in your code.

#5 - 08/22/2016 11:06 AM - darix (Marcus Rückert)

That's why i would deprecate the $ variables and make people use match objects all the time.

I mean the stdlib even has code that reads

```
matchdata = $~
```

That feels just wrong.

Maybe 2.4 could start issue warnings about using $ variables and 3.0 removes them?

#6 - 09/07/2016 07:40 AM - naruse (Yui NARUSE)

Below example shows 2nd thread overwrites 1st thread's regexp match result.

```
% ruby -e
'P = proc {|s| p [s, $~]; sleep 1; /foo.*/=~s; sleep 1; p [s,$~] }; def foo; Thread.new{P.call("foobar")}; sle
ep 0.2; Thread.new{P.call("foo")}; end; foo;sleep 5'
["foobar", nil]
["foo", nil]
["foobar", #<MatchData "foo">]
["foo", #<MatchData "foo">]
```

This example doesn't happen above phenomena different from above one.

```
% ruby -e
'P = proc {|s| p [s, $~]; sleep 1; /foo.*/=~s; sleep 1; p [s,$~] }; Thread.new{P.call("foobar")}; sleep 0.2; T
hread.new{P.call("foo")}; sleep 5'
["foobar", nil]
["foo", nil]
["foobar", #<MatchData "foobar">]
["foo", #<MatchData "foo">]
```

#7 - 11/27/2017 10:49 PM - headius (Charles Nutter)

We've had another report in JRuby about this behavior. In this case, two threads doing String#split step on each others backrefs because they share a backref frame: https://github.com/jruby/jruby/issues/4868

This case can't even be avoided. Even if you don't use $~ there are threading issues. These may come into play for MRI in either split or other methods that consume backref and lastline, but they'll certainly be a problem for all parallel-threaded implementations that wish to be compatible.

#8 - 11/28/2017 08:32 PM - Eregon (Benoit Daloze)

FWIW, TruffleRuby always stores the MatchData $? in a thread-local storage per frame.
It seems to work fine so far and seems to cause no real-world incompatibilities.

#9 - 11/29/2017 03:00 AM - ko1 (Koichi Sasada)

Eregon (Benoit Daloze) wrote:

> FWIW, TruffleRuby always stores the MatchData $? in a thread-local storage per frame.
> It seems to work fine so far and seems to cause no real-world incompatibilities.

Each frame has a map (thread -> MachData)?

#10 - 12/02/2017 09:37 PM - Eregon (Benoit Daloze)

ko1 (Koichi Sasada) wrote:

> Each frame has a map (thread -> MatchData)?

Conceptually yes, but it is allocated lazily and it specializes for being accessed by a single thread.
A Java ThreadLocal is used in the general case when more than one thread stores a MatchData in a frame.
https://github.com/graalvm/truffleruby/blob/vm-enterprise-0.29/src/main/java/org/truffleruby/language/threadlocal/ThreadAndFrameLocalStorage.java

#11 - 07/25/2019 05:49 PM - jeremyevans0 (Jeremy Evans)

- Related to Bug #8444: Regexp vars $~ and friends are not thread local added