Ruby trunk - Bug #2025

problem with pthread handling on non NPTL platform

08/31/2009 08:42 PM - Petr.Salinger@seznam.cz (Petr Salinger)

Status:	Closed		
Priority:	Normal		
Assignee:	mame (Yusuke Endoh)		
Target version:	2.0.0		
ruby -v:	1.9.1.243	**Backport:**	

Description

=begin
I tried to fix some testsuite failures on GNU/kFreeBSD,
http://bugs.debian.org//cgi-bin/bugreport.cgi?bug=542927.
I observed some problems in the pthread related code.
The hang in 1st test in
http://redmine.ruby-lang.org/issues/show/1525
also applies for us.

IMO, the ruby should try to work under any POSIX pthread
conforming implementation, not only NPTL.
The code audit in this area seems needed.

There are some problems with handling of fork()/exec().
There really should be reinitialization of locks in child,
the timer should be started using pthread_once(), the current
approach is fragile and might lead to start of more timer threads.
http://www.opengroup.org/onlinepubs/9699919799/functions/pthread_once.html

In general, I do not understand how code in thread_pthread.c:

static pthread_t timer_thread_id;
static pthread_cond_t timer_thread_cond = PTHREAD_COND_INITIALIZER;
static pthread_mutex_t timer_thread_lock = PTHREAD_MUTEX_INITIALIZER;
rb_thread_create_timer_thread()
thread_timer()

could survive correctly fork(), see also
http://www.opengroup.org/onlinepubs/009695399/functions/pthread_atfork.html

I really doubt the following code in process.c
for rb_f_fork(VALUE obj) is correct:

```
  switch (pid = rb_fork(0, 0, 0, Qnil)) {
    case 0:
```

#ifdef linux
after_exec();
#endif
rb_thread_atfork();
if (rb_block_given_p()) {
int status;

```
          rb_protect(rb_yield, Qundef, &status);
          ruby_stop(status);
      }
```

The conditional after_exec() shouldn't be here.
There is already "after_fork()" at line 2331,
which is executed for both parent and child.
The exception is when chfunc is not NULL,
then it is not executed at all.

The bug is timing dependent, i.e. there is a race condition.
Sometimes the child process would have 2 timer threads, sometimes
it would have the expected 1.

Only the probability of 2 is higher on linuxthreads compared to NPTL,
but it can happen under any pthread implementation.

Ruby should not use PTHREAD_CREATE_DETACHED and after that use pthread_join.
http://www.opengroup.org/onlinepubs/9699919799/functions/pthread_join.html:
"The behavior is undefined if the value specified by the thread argument
to pthread_join() does not refer to a joinable thread."

Ruby should use pthread_sigmask() instead of sigprocmask()
when available and so on.
http://www.opengroup.org/onlinepubs/9699919799/functions/pthread_sigmask.html:
"The use of the sigprocmask() function is unspecified in a

This would work correctly on both linuxthreads/NPTL and should on any
POSIX pthread conforming implementation.
Ideally, ruby would not require full conformance, but also
accept some known exceptions, like our getpid() difference.
=end

Related issues:

Related to Ruby trunk - Bug #1525: Deadlock in Ruby 1.9's VM caused by Condit...	**Closed**	**05/28/2009**

History

#1 - 04/23/2010 02:16 AM - mame (Yusuke Endoh)

- Category set to core

- Assignee set to mame (Yusuke Endoh)

=begin
Hi,

I tried to fix some testsuite failures on GNU/kFreeBSD,

Thank you for your investigation reports!

I show a patch before I answer to each report.
The following patch removes "after_exec()" and changes sigprocmask()
to pthread_sigmask().
"make test" and "make test-rubyspec" was passed on my Linux.

I'm afraid that this patch breaks Ruby on other platforms, such as
os x, Solaris, etc. Could you (or anyone) test the patch?

diff --git a/process.c b/process.c
index e566e9b..4f68d69 100644
--- a/process.c
+++ b/process.c
@@ -2620,9 +2620,6 @@ rb_f_fork(VALUE obj)

```
  switch (pid = rb_fork(0, 0, 0, Qnil)) {
    case 0:
```

-#ifdef linux

- after_exec(); -#endif rb_thread_atfork(); if (rb_block_given_p()) { int status; diff --git a/signal.c b/signal.c index 3fe1633..92e5d35 100644 --- a/signal.c +++ b/signal.c @@ -888,11 +888,7 @@ static VALUE trap_ensure(struct trap_arg *arg) { /* enable interrupt */ -#ifdef HAVE_SIGPROCMASK
- sigprocmask(SIG_SETMASK, &arg->mask, NULL); -#else
- sigsetmask(arg->mask); -#endif
- pthread_sigmask(SIG_SETMASK, &arg->mask, NULL); trap_last_mask = arg->mask; return 0; } @@ -902,11 +898,7 @@ void rb_trap_restore_mask(void) { #if USE_TRAP_MASK -# ifdef HAVE_SIGPROCMASK
- sigprocmask(SIG_SETMASK, &trap_last_mask, NULL); -# else
- sigsetmask(trap_last_mask); -# endif
- pthread_sigmask(SIG_SETMASK, &trap_last_mask, NULL); #endif }

@@ -966,12 +958,8 @@ sig_trap(int argc, VALUE *argv)

}
#if USE_TRAP_MASK
/ disable interrupt */
-# ifdef HAVE_SIGPROCMASK
sigfillset(&arg.mask);

- sigprocmask(SIG_BLOCK, &arg.mask, &arg.mask); -# else
- arg.mask = sigblock(~0); -# endif

- pthread_sigmask(SIG_BLOCK, &arg.mask, &arg.mask);

  return rb_ensure(trap, (VALUE)&arg, trap_ensure, (VALUE)&arg);
  #else
  @@ -1026,12 +1014,8 @@ init_sigchld(int sig)

#if USE_TRAP_MASK
/* disable interrupt */
-# ifdef HAVE_SIGPROCMASK
sigfillset(&mask);

- sigprocmask(SIG_BLOCK, &mask, &mask); -# else
- mask = sigblock(~0); -# endif

- pthread_sigmask(SIG_BLOCK, &mask, &mask);
  #endif

  oldfunc = ruby_signal(sig, SIG_DFL);
  @@ -1042,13 +1026,8 @@ init_sigchld(int sig)
  }

#if USE_TRAP_MASK
-#ifdef HAVE_SIGPROCMASK
sigdelset(&mask, sig);

- sigprocmask(SIG_SETMASK, &mask, NULL); -#else
- mask &= ~sigmask(sig);
- sigsetmask(mask); -#endif
- pthread_sigmask(SIG_SETMASK, &mask, NULL); trap_last_mask = mask; #endif }

  Sometimes the child process would have 2 timer threads, sometimes
  it would have the expected 1.

I confirmed that issue on Linux.

The following patch logs when timer thread wakes up.

diff --git a/thread_pthread.c b/thread_pthread.c
index e6295db..51ed234 100644
--- a/thread_pthread.c
+++ b/thread_pthread.c
@@ -762,6 +762,7 @@ thread_timer(void *dummy)
#define WAIT_FOR_10MS() native_cond_timedwait(&timer_thread_cond, &timer_thread_lock, get_ts(&ts, PER_NANO/100))
while (system_working > 0) {
int err = WAIT_FOR_10MS();

- printf("pid: %d, tid: %p, timer thread tick\n", getpid(), pthread_self()); if (err == ETIMEDOUT); else if (err == 0 || err == EINTR) { if (rb_signal_buff_size() == 0) break;

With this patch applied,

$ ./miniruby -e '
trap(:USR1) { p :signaled }
if pid = fork
Process.kill(:USR1, pid)
Process.wait(pid)
else
sleep 0.1
p "#$$ wait start"
sleep 3
p "#$$ wait end"

end
'
pid: 1015, tid: 0xb7b77b90, timer thread tick
pid: 1017, tid: 0xb7b77b90, timer thread tick
:signaled
pid: 1015, tid: 0xb7b77b90, timer thread tick
pid: 1017, tid: 0xb7fddb90, timer thread tick
pid: 1017, tid: 0xb7b77b90, timer thread tick
snip
pid: 1015, tid: 0xb7b77b90, timer thread tick
pid: 1017, tid: 0xb7b77b90, timer thread tick
pid: 1017, tid: 0xb7fddb90, timer thread tick
"1017 wait start"
pid: 1015, tid: 0xb7b77b90, timer thread tick
pid: 1017, tid: 0xb7fddb90, timer thread tick
pid: 1017, tid: 0xb7b77b90, timer thread tick
snip
pid: 1017, tid: 0xb7fddb90, timer thread tick
pid: 1015, tid: 0xb7b77b90, timer thread tick
pid: 1017, tid: 0xb7b77b90, timer thread tick
pid: 1017, tid: 0xb7fddb90, timer thread tick
pid: 1017, tid: 0xb7b77b90, timer thread tick
pid: 1015, tid: 0xb7b77b90, timer thread tick
"1017 wait end"
pid: 1017, tid: 0xb7fddb90, timer thread tick
pid: 1015, tid: 0xb7b77b90, timer thread tick

The sub-process whose pid is 1017 has two timer threads whose tids
are 0xb7fddb90 and 0xb7b77b90.

I agree the cause is "after_exec()" in rb_f_fork(). It should be
removed.

> Ruby should not use PTHREAD_CREATE_DETACHED and after that use pthread_join.

As far as I know, pthread_join is not used against thread created
with PTHREAD_CREATE_DETACHED. Ruby uses pthread_join against only
timer thread which is not created with PTHREAD_CREATE_DETACHED, I
think.
Did you actually see any detached thread are pthread_join'ed?

> Ruby should use pthread_sigmask() instead of sigprocmask()
> when available and so on.

Agreed.

Thanks once again!

--
Yusuke Endoh mame@tsg.ne.jp
=end

#2 - 04/23/2010 11:12 PM - Petr.Salinger@seznam.cz (Petr Salinger)

=begin
Hello.

1) for sigprocmask part, please just replace sigprocmask() by pthread_sigmask() and leave
possibility to use sigsetmask(). Ideally also replace the check and HAVE_SIGPROCMASK.
I doubt there will be a platform with sigprocmask without pthread_sigmask available.

2) PTHREAD_CREATE_DETACHED is used in native_thread_create(), and native_thread_join() calls pthread_join(th, 0) - at least in current Debian 1.9.1.378-2

3) please could you add fix also for
http://bugs.debian.org/cgi-bin/bugreport.cgi?bug=560293

=end

#3 - 04/24/2010 12:31 AM - mame (Yusuke Endoh)

=begin
Hi,

Thank you for your quick response, in spite of my very late
response!

2010/4/23 Petr Salinger redmine@ruby-lang.org:

> 1) for sigprocmask part, please just replace sigprocmask() by pthread_sigmask() and leave
> possibility to use sigsetmask(). Ideally also replace the check and HAVE_SIGPROCMASK.
> I doubt there will be a platform with sigprocmask without pthread_sigmask available.

pthread_sigmask is already used in rb_disable_interrupt and
rb_enable_interrupt. But AFAIK, there is no bug report about
build error because of absence of pthread_sigmask.
So I think we don't have to worry about it.

> 2) PTHREAD_CREATE_DETACHED is used in native_thread_create(), and native_thread_join() calls ?pthread_join(th, 0) - at least in current Debian 1.9.1.378-2

Yes, but native_thread_join is not used against a thread
created by native_thread_create.
native_thread_join is used just against timer thread which
is created by pthread_create, not by native_thread_create.

> 3) please could you add fix also for
> http://bugs.debian.org/cgi-bin/bugreport.cgi?bug=560293

Okay. Will do.

--
Yusuke Endoh mame@tsg.ne.jp

=end

#4 - 04/24/2010 12:47 AM - mame (Yusuke Endoh)

- Status changed from Open to Closed

- % Done changed from 0 to 100

=begin
This issue was solved with changeset r27464.
Petr, thank you for reporting this issue.
Your contribution to Ruby is greatly appreciated.
May Ruby be with you.

=end