Ruby - Bug #20682

Slave PTY output is lost after a child process exits in macOS

08/19/2024 01:16 AM - ono-max (Naoto Ono)

Status:	Closed		
Priority:	Normal		
Assignee:			
Target version:			
ruby -v:		**Backport:**	3.1: UNKNOWN, 3.2: UNKNOWN, 3.3: UNKNOWN

Description

According to Launchable, the following PTY tests are flaky only on macOS.

https://app.launchableinc.com/organizations/ruby/workspaces/ruby/data/test-paths/file%3Dtest%2Ftest_pty.rb%23%23%23class%3DTestPTY%23%23%23testcase%3Dtest_spawn_without_block
https://app.launchableinc.com/organizations/ruby/workspaces/ruby/data/test-paths/file%3Dtest%2Ftest_pty.rb%23%23%23class%3DTestPTY%23%23%23testcase%3Dtest_spawn_with_block
https://app.launchableinc.com/organizations/ruby/workspaces/ruby/data/test-paths/file%3Dtest%2Ftest_pty.rb%23%23%23class%3DTestPTY%23%23%23testcase%3Dtest_commandline
https://app.launchableinc.com/organizations/ruby/workspaces/ruby/data/test-paths/file%3Dtest%2Ftest_pty.rb%23%23%23class%3DTestPTY%23%23%23testcase%3Dtest_argv0

It's because the slave PTY output is lost after a child process exits in macOS. Here is the code to reproduce the problem.
When I remove sleep 3 from the code, "a" is returned.

```
require 'pty'

r, w, pid = PTY.spawn('ruby', '-e', 'puts "a"')
sleep 3
puts r.gets #=> Returns nil
```

Based on my investigation, this issue happens in the macOS side and it's almost same as https://github.com/pexpect/pexpect/issues/662.
The cause is described as follows in the ticket:

> // NOTE[macOS-S_CTTYREF]: On macOS, after a forkpty(), if the pty slave (child)
> // is closed before the pty master (parent) reads, the pty's buffer is cleared
> // thus the master (parent) reads nothing. This can happen if the child exits
> // before the parent has a chance to call master.read().
> //
> // This issue has been reported to Apple, but has not been resolved:
> // https://developer.apple.com/forums/thread/663632
> //
> // Work around this issue by opening /dev/tty then closing it. This ultimately
> // causes the child's exit() to flush the slave pty's output buffer in a
> // blocking way. This fixes the problem on macOS 13.2 in my testing.
> //
> // Here's how the workaround works in detail:
> //
> // If we open /dev/tty, it sets the S_CTTYREF flag on the process. This flag
> // remains set if we close the /dev/tty file descriptor.
> // https://github.com/apple-oss-distributions/xnu/blob/aca3beaa3dfbd42498b42c5e5ce20a938e6554e5/bsd/kern/tty_tty.c#L128
> // Additionally, opening /dev/tty retains a reference to the pty slave.
> // https://github.com/apple-oss-distributions/xnu/blob/aca3beaa3dfbd42498b42c5e5ce20a938e6554e5/bsd/kern/tty_tty.c#L147
> //
> // When the child process exits:
> //
> // 1. All open file descriptors (including stdin/stdout/stderr which are the pty
> //   slave) are closed. This does *not* drain unread pty slave output.
> //   * If S_CTTYREF was set, closing the file descriptors does not close the
> //   last reference to the pty slave, so no cleanup happens yet.
> //   * NOTE[macOS-pty-close-loss]: If S_CTTYREF was not set, closing the file

```
//     descriptors drops the last reference to the pty slave. Unread data is
//     dropped.
//
// 2. If the S_CTTYREF flag is set on the child process, the controlling
//    terminal (pty slave) is closed. XNU's ptsclose() ultimately calls
//    ttywait().
//
https://github.com/apple-oss-distributions/xnu/blob/aca3beaa3dfbd42498b42c5e5ce20a938e6554e5/bsd/kern/kern_exit.c#L2272
//    * ttywait() is the same as ioctl(slave, TIOCDRAIN); it blocks waiting for
//      output to be received.
//
https://github.com/apple-oss-distributions/xnu/blob/aca3beaa3dfbd42498b42c5e5ce20a938e6554e5/bsd/kern/tty.c#L1129-L1130
//    * NOTE[macOS-pty-waitpid-hang]: Because of the blocking ttywait(), the
//      process is in an exiting (but not zombie) state. waitpid() will hang.
//
//    * NOTE[macOS-pty-close-loss]: If the S_CTTYREF flag is not set on the
//      child process, ttywait() is not called, thus the pty slave does not
//      block waiting for the output to be received, and the output is dropped.
//      A well-behaving parent will use a poll() loop anyway, so this isn't a
//      problem. (It does make quick tests annoying to write though.)
//
// Demonstration of NOTE[macOS-pty-close-loss] (S_CTTYREF is not set before
// exit):
//
//     // On macOS, this program should report 'data = ""', demonstrating that
//     // writes are lost.
//
//     #include <stdlib.h>
//     #include <errno.h>
//     #include <stdio.h>
//     #include <string.h>
//     #include <unistd.h>
//     #include <util.h>
//
//     int main() {
//       int tty_fd;
//       pid_t pid = forkpty(&tty_fd, /*name=*/NULL, /*termp=*/NULL,
//                           /*winp=*/NULL);
//       if (pid == -1) { perror("forkpty"); abort(); }
//
//       if (pid == 0) {
//         // Child.
//         (void)write(STDOUT_FILENO, "y", 1);
//         exit(0);
//       } else {
//         // Parent.
//
//         // Cause the child to write() then exit(). exit() will drop written
//         // data.
//         sleep(1);
//
//         char buffer[10];
//         ssize_t rc = read(tty_fd, buffer, sizeof(buffer));
//         if (rc < 0) { perror("read"); abort(); }
//         fprintf(stderr, "data = "%.*s"\n", (int)rc, buffer);
//       }
//
//       return 0;
//     }
//
// Demonstration of NOTE[macOS-pty-waitpid-hang] (S_CTTYREF is set before exit):
//
//     // On macOS, this program should hang, demonstrating that the child
//     // process doesn't finish exiting.
//     //
```

```
//    // During the hang, observe that the child is in an exiting state ("E"):
//    //
//    //    $ ps -e -o pid,stat | grep 20125
//    //    20125 ?Es
//
//    #include <errno.h>
//    #include <fcntl.h>
//    #include <stdio.h>
//    #include <stdlib.h>
//    #include <string.h>
//    #include <unistd.h>
//    #include <util.h>
//
//    int main() {
//      int tty_fd;
//      pid_t pid = forkpty(&tty_fd, /*name=*/NULL, /*termp=*/NULL,
//                          /*winp=*/NULL);
//      if (pid == -1) { perror("forkpty"); abort(); }
//
//      if (pid == 0) {
//        // Child.
//        close(open("/dev/tty", O_WRONLY));
//        (void)write(STDOUT_FILENO, "y", 1);
//        exit(0);
//      } else {
//        // Parent.
//
//        fprintf(stderr, "child PID: %d\n", pid);
//
//        // This will hang because, despite the child being is an exiting
//        // state, the child is waiting for us to read().
//        pid_t rc = waitpid(pid, NULL, 0);
//        if (rc < 0) { perror("waitpid"); abort(); }
//      }
//
//      return 0;
//    }
```

In Ruby, PTY is implemented with fork() and posix_openpt() in macOS. I could reproduce the problem in the following script.

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <fcntl.h>
#include <sys/ioctl.h>

int main() {
    int master_fd, slave_fd;
    pid_t child_pid;
    char *slave_name;

    // Open a master pseudo-terminal
    master_fd = posix_openpt(O_RDWR | O_NOCTTY);
    if (master_fd == -1) {
        perror("posix_openpt");
        exit(1);
    }

    // Grant access to the slave pseudo-terminal
    if (grantpt(master_fd) == -1) {
        perror("grantpt");
        exit(1);
    }

    // Unlock the slave pseudo-terminal
    if (unlockpt(master_fd) == -1) {
```

```
        perror("unlockpt");
        exit(1);
    }

    // Get the name of the slave pseudo-terminal
    slave_name = ptsname(master_fd);
    if (slave_name == NULL) {
        perror("ptsname");
        exit(1);
    }

    // Fork a child process
    child_pid = fork();
    if (child_pid == -1) {
        perror("fork");
        exit(1);
    } else if (child_pid == 0) {
        // Child process

        // Open the slave pseudo-terminal
        slave_fd = open(slave_name, O_RDWR);
        if (slave_fd == -1) {
            perror("open");
            exit(1);
        }

        // Create a new session and process group
        if (setsid() == -1) {
            perror("setsid");
            exit(1);
        }

        // Set the controlling terminal for the child process
        if (ioctl(slave_fd, TIOCSCTTY, NULL) == -1) {
            perror("ioctl");
            exit(1);
        }

        // Duplicate the slave file descriptor to stdin, stdout, and stderr
        if (dup2(slave_fd, STDIN_FILENO) == -1) {
            perror("dup2");
            exit(1);
        }
        if (dup2(slave_fd, STDOUT_FILENO) == -1) {
            perror("dup2");
            exit(1);
        }
        if (dup2(slave_fd, STDERR_FILENO) == -1) {
            perror("dup2");
            exit(1);
        }
        // close(open("/dev/tty", O_WRONLY));

        // Close the original slave file descriptor
        close(slave_fd);

        // Execute a shell or other program
        (void)write(STDOUT_FILENO, "y", 1);
        exit(1);
    } else {
        sleep(5);
        char buffer[10];
        ssize_t rc = read(master_fd, buffer, sizeof(buffer));
        if (rc < 0)
        {
            perror("read");
            abort();
```

```
        }
        fprintf(stderr, "data = \"%.*s\"\n", (int)rc, buffer);
        // Clean up
        close(master_fd);
    }

    return 0;
}
```

Associated revisions

Revision db4ea95219f045b8e855afae0b6d28027ec8d3eb - 02/28/2025 09:32 AM - ono-max (Naoto Ono)

[Bug #20682] Add sleep 0.1 to stabilize flaky failures on macOS (#12829)

[Bug #20682] Add sleep 0.1 to stabilize flaky failures on macOS

Revision db4ea95219f045b8e855afae0b6d28027ec8d3eb - 02/28/2025 09:32 AM - ono-max (Naoto Ono)

[Bug #20682] Add sleep 0.1 to stabilize flaky failures on macOS (#12829)

[Bug #20682] Add sleep 0.1 to stabilize flaky failures on macOS

Revision db4ea952 - 02/28/2025 09:32 AM - ono-max (Naoto Ono)

[Bug #20682] Add sleep 0.1 to stabilize flaky failures on macOS (#12829)

[Bug #20682] Add sleep 0.1 to stabilize flaky failures on macOS

History

#1 - 08/19/2024 01:21 AM - ono-max (Naoto Ono)

I created the PR: https://github.com/ruby/ruby/pull/11404

This change works in my macOS environment.

#2 - 08/19/2024 01:51 AM - mame (Yusuke Endoh)

The PR looks good to me.

@ono-max (Naoto Ono) Do you want to merge it yourself? If you are willing, I'd like to propose you as a committer.

#3 - 08/19/2024 01:53 AM - ono-max (Naoto Ono)

mame (Yusuke Endoh) wrote in #note-2:

> The PR looks good to me.
>
> @ono-max (Naoto Ono) Do you want to merge it yourself? If you are willing, I'd like to propose you as a committer.

Sure! I would be delighted to accept a committer.

#4 - 08/22/2024 07:54 AM - ono-max (Naoto Ono)

@mame (Yusuke Endoh) and I have noticed that there is a problem in the workaround solution: https://github.com/ruby/ruby/pull/11404

This program hangs in Process.wait(pid).

```
require 'pty'
_, _, pid = PTY.spawn('ruby', '-e', 'puts "a"; puts "b"')
Process.waitpid(pid)
```

On the other hand, this program does not hang and exits successfully.

```
require 'pty'
r, _, pid = PTY.spawn('ruby', '-e', 'puts "a"; puts "b"')
puts r.gets #=> "a\n"
Process.waitpid(pid)
```

From the above result, a user who uses PTY has to read from the IO at least once.

#5 - 08/22/2024 08:00 AM - ono-max (Naoto Ono)

Here is the summary that we know so far.

Current PTY's behavior

The slave PTY output is lost after a child process exits in macOS

Workaround's behavior

The program hangs if a user writes a code which does not read the output as follows:

```
require 'pty'
_, _, pid = PTY.spawn('ruby', '-e', 'puts "a"; puts "b"')
Process.waitpid(pid)
```

#6 - 08/22/2024 08:03 AM - ono-max (Naoto Ono)

@akr (Akira Tanaka)

I'd appreciate any ideas you may have.

#7 - 01/27/2025 08:17 AM - nobu (Nobuyoshi Nakada)

ono-max (Naoto Ono) wrote in #note-4:

> This program hangs in Process.wait(pid).
>
> ```
> require 'pty'
> _, _, pid = PTY.spawn('ruby', '-e', 'puts "a"; puts "b"')
> Process.waitpid(pid)
> ```

Now I tried this, and it exited immediately, on arm64 macOS 14.7.2 23H311.

However, the code in the description still returns nil.

```
require 'pty'

r, w, pid = PTY.spawn('ruby', '-e', 'puts "a"')
sleep 3
puts r.gets #=> Returns nil
```

#8 - 02/28/2025 11:04 AM - ono-max (Naoto Ono)

- Status changed from Open to Closed

Applied in changeset git|db4ea95219f045b8e855afae0b6d28027ec8d3eb.

[Bug #20682] Add sleep 0.1 to stabilize flaky failures on macOS (#12829)

[Bug #20682] Add sleep 0.1 to stabilize flaky failures on macOS