ClearCode

バグの直し方

実例を添えて

Kenji Okimoto

株式会社クリアコード

2012-09-16

自己紹介

- ✓ okkez (おっきーと読みます)
- ✓ Ruby 歴8年くらい
- ✓ るりま (2006-11から)
- ✓ Hiki (2008-09から)
- ✓ CRuby コミッタ (2011-12から)

自己紹介

- クリアコード (2010-11から)
  - 大規模メールシステム (milter manager)
  - GLib, GTK+ アプリケーション
  - Ruby, C/C++, etc.

お品書き

- よく知らないプログラムのバグの直し方
- 実例紹介

バグの直し方

- ✓ バグに気付く
- ✓ 再現方法を記録する
- ✓ 問題箇所を絞り込む
- ✓ 問題を修正する
- ✓ 修正できたことを確認する
- ✓ 壊していないことを確認する

再現方法を記録する

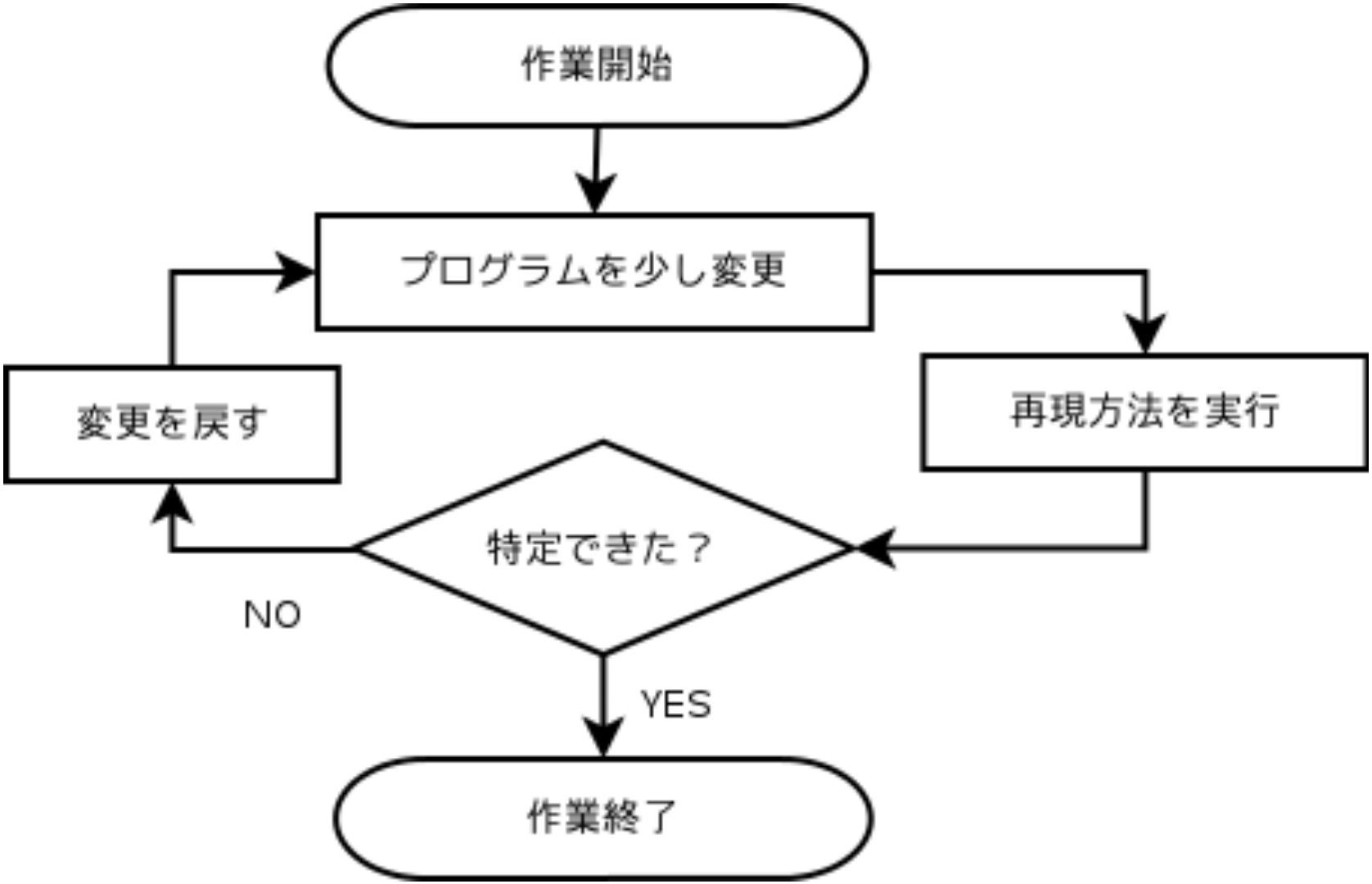
- 問題を再現できる
  - テストプログラムを作成する
  - 操作をメモする
  - データを作成する

問題箇所を絞り込む

対象のプログラムを

- ✓ よく知っている場合
  - ✓ 怪しい部分の目星を付ける
- ✓ よく知らない場合
  - ✓ 地道に作業して問題箇所を特定する

問題箇所を絞り込む



ポイント!!

> *プログラムの変更を少しずつ行うことによって、一歩ずつ着実に問題の解決に向かっていく*

ポイント!!

> *プログラムの変更を行うたびに、「今回の変更では何を調べたいのか」を意識して作業する*

問題箇所を特定できたら

- ✓ 問題を修正する

修正できたことを確認

- ✓ 再現方法を実行
- ✓ 問題が再現しないことを確認
- ✓ 修正できた!!

実例紹介

- Ruby のメモリリーク
  - http://bugs.ruby-lang.org/issues/5688
  - http://www.clear-code.com/blog/2011/12/6.html

デモ

- ✓ バグに気付く
- ✓ 再現方法を記録する
- ✓ 問題箇所を絞り込む
- ✓ 問題を修正する
- ✓ 修正できたことを確認する
- ✓ 壊していないことを確認する

事前情報

- Solaris10 で発生
  - Ruby 1.9.2-p180 で発生
  - mtrace や valgrind が使えない
  - spawn だけでなく fork でも発生
- 慣れている Debian では発生しない

再現コード

```
ARGV[0].to_i.times do |n|
  fork{ exit! }
  #spawn{ exit! }
  GC.start if n % 100 == 0
end
```

fork()でもspawn()でもメモリリークする

便利スクリプト

```
#!/bin/bash
/tmp/a/bin/ruby ./fork-exit.rb 300000 &
pid=$!
echo fork-exit:$pid
trap "kill $pid; exit" INT TERM
count=0
ps -o pid,ppid,vsz,rss,args | head -1
prev=""
while true; do
  current=`ps -p ${pid} -o pid,ppid,vsz,rss,args | grep fork-exit.rb`
  if test "$current" != "$prev"; then
      echo "$current"
  fi
  prev=$current
  sleep 1
done
```

便利スクリプトがあると便利!!

Step 1

spawn のコールグラフ

```
spawn
  -> rb_f_spawn
    -> rb_spawn_process
      -> rb_fork_err
```

Step 1

fork のコールグラフ

```
fork
 -> rb_f_fork
  -> rb_fork
   -> rb_fork_err
```

Step 2

spawn でも
fork でも
rb_fork_err
を呼んでいる

Step 3 - 処理を確認する

rb_fork_err の中身を確認

```
before_fork()
fork()
if (!pid) { ... }
after_fork()
```

if(!pid){...}は子プロセスの処理なので無視する

Step 3 - 処理を変更する

- before_fork() の前で return -1
- before_fork() の後で return -1
- fork() の後で return pid
- after_fork() の後で return pid
  - 修正前と同じ

Step 4

- ✓ before_fork() の前 ... ビルドエラー
- ✓ before_fork() の後 ... ビルドエラー
- ✓ fork() の後で ... メモリリークしない!

after_fork() に問題があることがわかった!!

Step 5 - after_fork()

- GET_THREAD()->thrown_errinfo = 0
  - 値をセットしているだけ
- rb_thread_reset_timer_thread()
  - 値をセットしてるだけ

Step 5 - after_fork()

- rb_thread_start_timer_thread()
  - rb_thread_create_timer_thread()
- forked_child = 0
  - 値をセットしてるだけ
- rb_disable_interrupt()
  - 値をセットしてるだけ

Step 6

- ✓ pthread_attr_init(3) を読む
  - ✓ pthread_attr_destroy() もある
- ✓ pthread_create(3) を読む
  - ✓ pthread_create()の引数がわかる

Step 7

pthread_create の第2引数を NULLにすると...

Step 7

メモリリーク
しなくなる!!

Step 8

pthread_attr_init
してるけど
pthread_attr_destroy
してないことに気付く

Step 9

pthread_attr_destroy を追加して試すと...

Step 10

メモリリーク
しなくなる

パッチ

```
diff --git a/thread_pthread.c b/thread_pthread.c
index 4746aaa..ab7bdf9 100644
--- a/thread_pthread.c
+++ b/thread_pthread.c
@@ -835,6 +835,7 @@ rb_thread_create_timer_thread(void)
     }
      native_cond_wait(&timer_thread_cond, &timer_thread_lock);
      native_mutex_unlock(&timer_thread_lock);
+     pthread_attr_destroy(&attr);
     }
      rb_disable_interrupt(); /* only timer thread recieve signal */
 }
```

Step 11

パッチを作ったら
make test-all
を流してE,Fが増えていない
ことを確認する

まとめ

- ✓ バグに気付く
- ✓ 再現方法を記録する
- ✓ 問題箇所を絞り込む
- ✓ 問題を修正する
- ✓ 修正できたことを確認する
- ✓ 壊していないことを確認する

ポイント!!

問題箇所は
一歩ずつ着実に
絞り込む

ポイント!!

> *問題箇所を特定できたらバグの修正は八割以上できたも同然です*

次の一歩

- 水平展開
  - 似たようなバグがあるかも。。。