Linux Fault Injection

第5回 The Linux Foundation Japan Symposium

美田晃伸
〈akinobu.mita@gmail.com〉

目次

- Linux Fault Injection の紹介
  - 仕組み，使い方，使用例
- 開発のきっかけ～マージまでの道のり
  - アイディアをくれた人々
- よくあるエラー処理の間違い

Fault injection とは

http://en.wikipedia.org/wiki/Fault_injection

- エラーをわざと起こす
  - エラー処理のようなあまり実行されない部分がテストできる
  - ソフトウェアのテストでコードカバレージを高める
  - ストレステストと組み合わせてソフトウェアの堅牢性を高める

Linux Fault Injection とは

- Fault injection のためのフレームワーク
  - エラーを意図的に発生させたい箇所に should_fail() を仕掛ける
  - エラーを発生させる確率やタイミングなどを debugfs でコントロールする
- あらかじめカーネルに仕掛けてある箇所
  - スラブ・アロケータ
  - ページ・アロケータ
  - I/O リクエスト処理

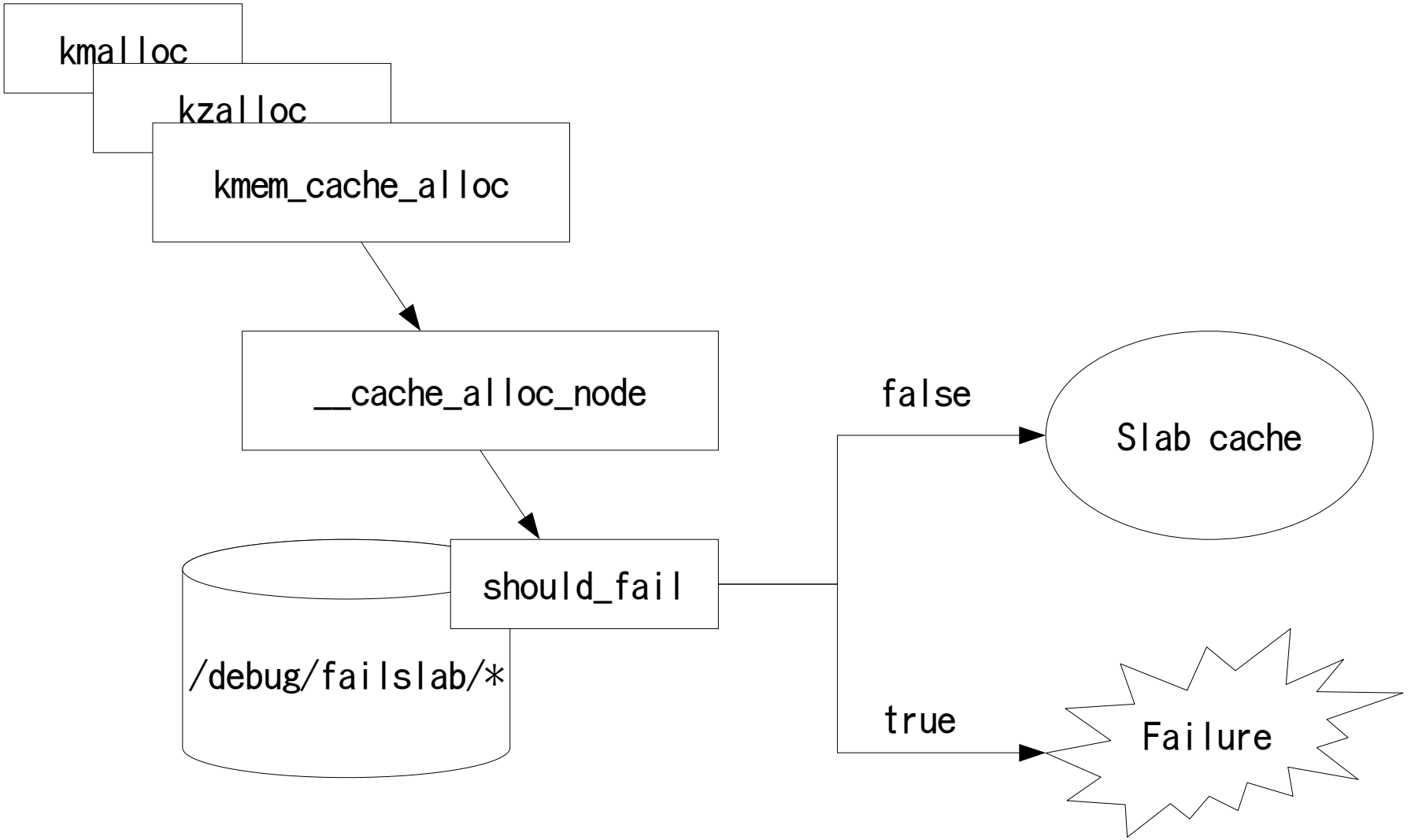
スラブ・アロケーションの失敗を埋め込む仕組み

```
static int should_failslab(struct kmem_cache *cachep, gfp_t flags)
{
    if (flags & __GFP_NOFAIL)
        return 0;

    return should_fail(&failslab.attr, obj_size(cachep));
}

static void *
__cache_alloc_node(struct kmem_cache *cachep, gfp_t flags, int nodeid,
                   void *caller)
{
    if (should_failslab(cachep, flags))
        return NULL;
    ...
```

スラブ・アロケーションの失敗を埋め込む図



使い方

- make menuconfig
  - Kernel hacking --->
    - [*] Fault-injection framework
      - [*] fault-injection capability for kmalloc
      - [*] fault-injection capability for alloc_pages()
      - [*] fault-injection capability for disk IO
      - [*] debugfs entries for fault-injection capabilities
        - [*] stacktrace filter for fault-injection capabilities

エラーの発生を debugfs でコントロールする

- /debug/failslab/*
  - スラブ・アロケーション失敗をコントロール
- /debug/fail_page_alloc/*
  - ページ・アロケーション失敗をコントロール
- /debug/fail_make_request/*
  - IO リクエスト失敗をコントロール

debugfs エントリの解説

- probability
  - エラーを発生させる確率［パーセント］
- interval
  - エラー毎の間隔
- times
  - エラーを発生させる最大回数
  - -1 にすると無制限

- require-start
- require-end
  - 指定したアドレス範囲のみエラーを発生させる
  - 特定のモジュールをテストしたい場合に役立つ
- process-filter
  - /proc/<pid>/make-it-fail が設定してあるプロセスのコンテキストのみエラーを発生させる

- ignore-gfp-wait
  - failslab と fail_page_alloc のみ
  - GFP_ATOMIC アロケーションだけ失敗させる
  - GFP_ATOMIC は GFP_KERNEL より失敗しやすい
- min-order
  - fail_page_alloc のみ
  - 指定したサイズ以上のページ・アロケーションのみ失敗させる [order]
  - 大きいサイズのページアロケーションほど失敗しやすい

（例1）モジュールのロード・アンロード中のスラブ・アロケーションを 5% の確率で失敗させる

```
echo Y > /debug/failslab/task-filter
echo N > /debug/failslab/ignore-gfp-wait
echo 5 > /debug/failslab/probability
echo 1 > /debug/failslab/interval
echo -1 > /debug/failslab/times

while true; do
    bash -c "echo 1 > /proc/self/make-it-fail && exec modprobe $test_module"
    bash -c "echo 1 > /proc/self/make-it-fail && exec modprobe -r $test_module"
done
```

Kernel BUG

```
FAULT_INJECTION: forcing a failure

Call Trace:
 [<ffffffff80321c8a>] random32+0x1c/0x20
 [<ffffffff80326d10>] should_fail+0xc5/0x101
 [<ffffffff8028c07e>] should_failslab+0x37/0x40
 [<ffffffff8028d36a>] kmem_cache_zalloc+0x1a/0x10a
 [<ffffffff802d9c35>] __sysfs_new_dirent+0x1d/0x5f
 [<ffffffff802d9d01>] __sysfs_make_dirent+0x1c/0x84
 [<ffffffff802d9d84>] sysfs_make_dirent+0x1b/0x37
 [<ffffffff802dad93>] sysfs_create_link+0xfd/0x146
 [<ffffffff802517b6>] sys_init_module+0xd45/0x16d8
 [<ffffffff88127000>] :mii:mii_ethtool_sset+0x0/0x210
 [<ffffffff80209d0e>] system_call+0x7e/0x83

e100: Intel(R) PRO/100 Network Driver, 3.5.17-k4-NAPI
e100: Copyright(c) 1999-2006 Intel Corporation
list_add corruption. prev->next should be next (ffffffff8059ab48), but was fffff
fff8848f658. (prev=ffffffff8848f658).
------------[ cut here ]------------
kernel BUG at lib/list_debug.c:33!
invalid opcode: 0000 [1] SMP
```

（例2）あるモジュールからのページ・アロケーションを 1000 回ごとに失敗させる

```
cat /sys/module/$test_module/sections/.text > /debug/fail_page_alloc/require-start
cat /sys/module/$test_module/sections/.data > /debug/fail_page_alloc/require-end

echo N > /debug/fail_page_alloc/task-filter
echo Y > /debug/fail_page_alloc/ignore-gfp-wait
echo 1 > /debug/fail_page_alloc/min-order
echo 100 > /debug/fail_page_alloc/probability
echo 1000 > /debug/fail_page_alloc/interval
echo -1 > /debug/fail_page_alloc/times
```

マージされるまでの道のり

1. failmalloc に触発される
   - http://www.nongnu.org/failmalloc/
2. failmalloc for slab allocator
   - http://lkml.org/lkml/2006/8/13/36
3. フィードバックを取り入れて改良を重ねる
   - たくさんカーネルバグが検出できるようになる
4. バグフィックスをして役に立つことを実証
5. v2.6.20 にマージ

アイディアをくれた人々

- Andrew Morton
  - IO リクエスト失敗
  - エラーの発生を動的にコントロール可能にする
- Andi Kleen
  - 指定したプロセスのコンテキストのみエラーを発生させる
  - 指定したアドレス範囲のみエラーを発生させる
- Don Mills
  - エラー発生時にスタックトレースをダンプさせる
  - ドキュメンテーション

よくあるエラー処理の間違い

- エラーチェック抜け
- IS_ERR() チェックと NULL チェックの誤用
- 返り値の間違い
- ロールバックしわすれ

エラーチェック抜け

```
dev = alloc_ieee80211(sizeof(struct ieee80211softmac_device) + sizeof_priv);
softmac = ieee80211_priv(dev);
...
```

1. alloc_ieee80211() はメモリ・アロケーション に失敗すると NULL を返す
2. NULL チェックしていない
3. NULL ポインター参照でクラッシュする

IS_ERR() と NULL チェックの誤用

```
rbu_device = platform_device_register_simple("dell_rbu", -1, NULL, 0);
if (!rbu_device) { // --> IS_ERR(rbu_device)
    <エラー処理>
}

rc = sysfs_create_bin_file(&rbu_device->dev.kobj, &rbu_data_attr);
```

1. platform_register_simple() の返り値を IS_ERR() でチェックしていない
2. エラー処理が実行されない
3. sys_create_bin_file() に初期化に失敗した引数が渡されて実行されてクラッシュする

返り値の間違い

```
static int __init init_spkm3_module(void)
{
    ...
    status = gss_mech_register(&gss_spkm3_mech);
    if (status)
        printk("Failed to register spkm3 gss mechanism!\n");
    return 0;
}
module_init(init_spkm3_module);
```

1. モジュールの初期化関数でエラーが発生したにもかかわらず、返り値が 0（成功）
2. モジュール・ロードが成功する
3. モジュール・アンロードするとクラッシュする

ロールバックしわすれ

```
static int __init init_sr(void)
{
        int rc;

        rc = register_blkdev(SCSI_CDROM_MAJOR, "sr");
        if (rc)
                return rc;
        return scsi_register_driver(&sr_template.gendrv);
}
module_init(init_sr);
```

1. scsi_register_driver() が失敗した場合
2. その手前で割り当てたメジャー番号を unregister_blkdev() で開放するのを忘れている