Tencent 腾讯

Crash自动分析诊断系统

TencentOS (原tlinux) 团队

陈贺 heddchen@tencent.com
彭志光 zgpeng@tencent.com
2019.10.19

目录

1 背景介绍

内核精细化运营的痛点和难点

服务器稳定性对于业务至关重要

宕机 ➡ 延迟, 掉线, 访问失败, etc.

服务器数量上升, 宕机率不变, 宕机数上升

内核可用性99.9%,宕机率 0.1% ➡ 1,000,000 × 0.1% = 1,000

服务器

业务运维人员

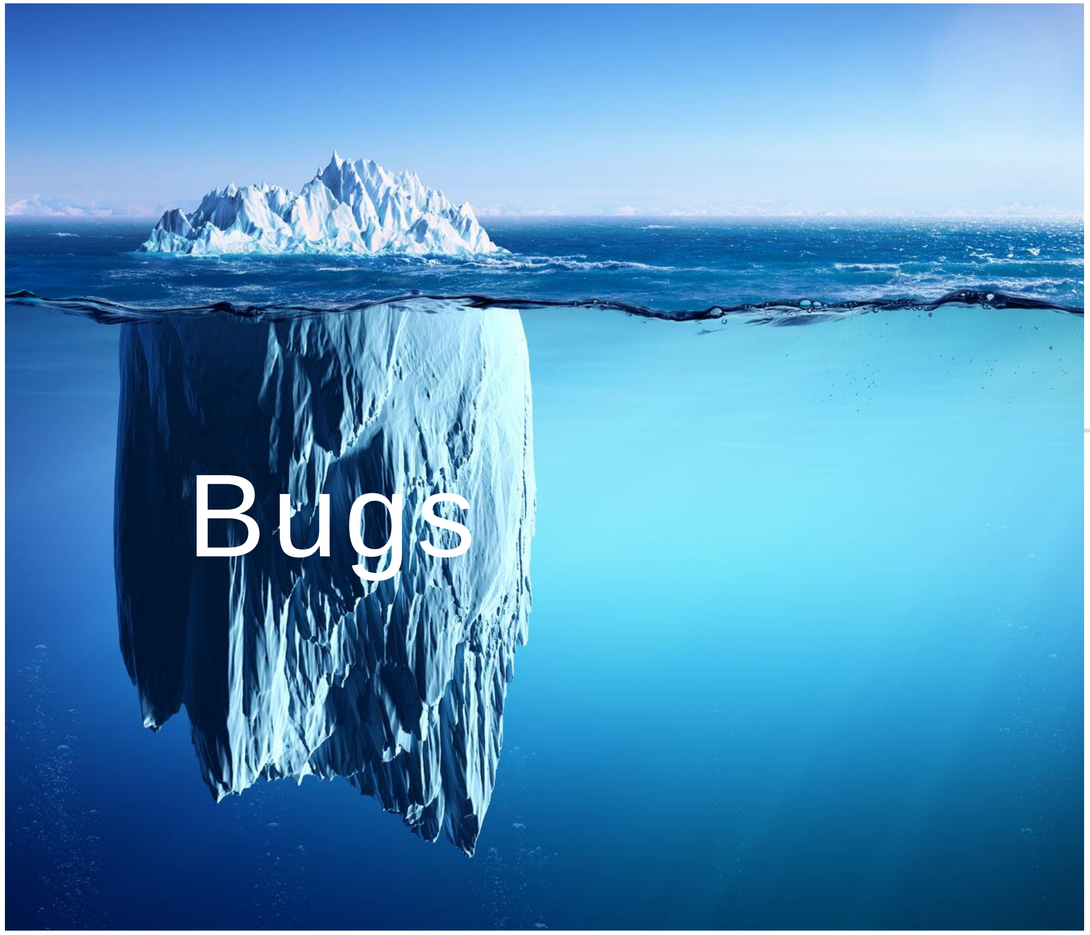
内核开发人员

宕机

- 宕机原因？
- 规避/解决方案？

- 人工分析耗时
- 已知问题重复分析
- 未知问题缺少问题跟踪途径

BUG宕机茫茫多，谁来解决?



LWN.net
News from the source

Content
Weekly Edition
Archives
Search
Kernel
Security
Distributions
Events calendar
Unread comments

LWN FAQ
Write for us

Edition
Return to the Front

tencent_os | Log out | (Subscriber)

Dealing with automated kernel bug reports

By **Jonathan Corbet**
September 15, 2019

Maintainers Summit

There is value in automatic testing systems, but they also present a problem of their own: how can one keep up with the high volume of bug reports that they generate? At the 2019 Linux Kernel Maintainers Summit, Shuah Khan ran a session dedicated to this issue. There was general agreement that the reports are hard to deal with, but not a lot of progress toward a solution.

Khan began by noting that one pervasive problem with these systems is classification: who should be responsible for a problem, what priority should it have, and is anybody working on it now? Turning to syzbot in particular, she said that getting the reproducer — the program that causes the reported problem to manifest itself — for any given report is a manual task, and that kernel developers tend to lose track of reproducers once the problem is fixed. It would be better, she said, to hang onto these reproducers and use them as regression tests going forward. She is looking into adding them to the kernel self-test infrastructure.

How can one keep up with the high volume of bug reports?

人工分析VMCORE复杂耗时

```
 #7 [ffff880fef2a3e80] dput at ffffffff8118bdad
 #8 [ffff880fef2a3eb0] __fput at ffffffff81176ecb
 #9 [ffff880fef2a3ef0] ____fput at ffffffff81176fde
#10 [ffff880fef2a3f00] task_work_run at ffffffff810687d7
#11 [ffff880fef2a3f30] do_notify_resume at ffffffff81002a19
#12 [ffff880fef2a3f50] int_signal at ffffffff81aa41aa
    RIP: 00007f15f9f26ea0  RSP: 00007ffc8adde2e8  RFLAGS: 00000246
    RAX: 0000000000000000  RBX: 00007f15fa6bc010  RCX: ffffffffffffffff
    RDX: 0000000000020000  RSI: 00007f15fa6bc010  RDI: 0000000000000006
    RBP: 0000000000020000   R8: 00007f15f9e85988   R9: 0000000000000011
    R10: 0000000000000007  R11: 0000000000000246  R12: 0000000000000000
    R13: 00007f15fa6bc010  R14: 0000000000000000  R15: 0000000000000006
    ORIG_RAX: 0000000000000003  CS: 0033  SS: 002b
```

异常栈分析

```
crash> dis dput
0xffffffff8118bd80 <dput>:      nopl   0x0(%rax,%rax,1) [FTRACE NOP]
0xffffffff8118bd85 <dput+0x5>:  push   %rbp
0xffffffff8118bd86 <dput+0x6>:  mov    %rsp,%rbp
0xffffffff8118bd89 <dput+0x9>:  push   %r14
0xffffffff8118bd8b <dput+0xb>:  push   %r13
0xffffffff8118bd8d <dput+0xd>:  push   %r12
0xffffffff8118bd8f <dput+0xf>:  push   %rbx
0xffffffff8118bd90 <dput+0x10>: mov    %rdi,%rbx
0xffffffff8118bd93 <dput+0x13>: nopl   0x0(%rax,%rax,1)
0xffffffff8118bd98 <dput+0x18>: test   %rbx,%rbx
0xffffffff8118bd9b <dput+0x1b>: je     0xffffffff8118be2e <dput+0xae>
0xffffffff8118bda1 <dput+0x21>: lea    0x5c(%rbx),%r13
0xffffffff8118bda5 <dput+0x25>: mov    %r13,%rdi
```

反汇编查找异常数据

异常栈分析, 入参推导,

反汇编查找异常数据, 内存搜索…

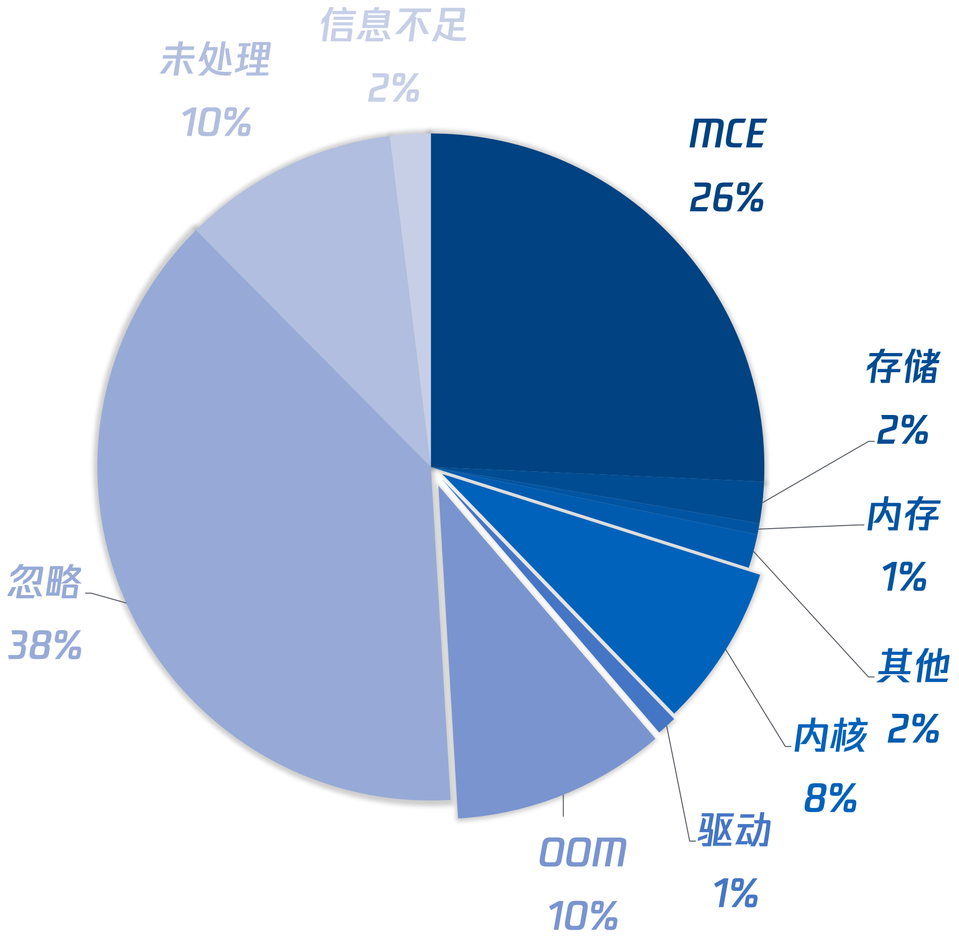
源码分析

故障原因定位, 复现验证

解决方案, 测试验证

1天 ~ 2周

内核宕机原因分布 – 无效项多



MCE, 磁盘故障, 内存故障,
第三方内核模块BUG,
老旧内核, 非常规内核

只关心内核BUG引起的宕机

1.已解决问题　2.已知未解决问题　3.新问题

无需人工介入　需要人工分析

早期宕机分析系统

收集dmesg日志进行分析, 关键字段匹配 + 机器学习.
能解决部分问题, 但是精度不够, 作用有限, 很多问题仅仅依靠dmesg无法处理

特征详情

基本信息

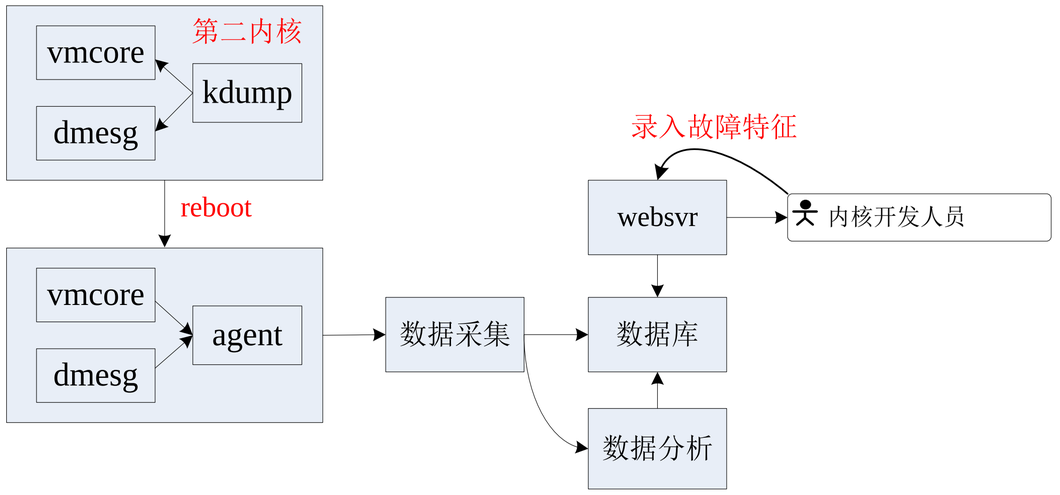
类型	优先级	创建时间	最后修改时间	最后修改人	编辑
软件故障-内核BUG	3	2016-06-28 15:00:57	2016-06-30 15:08:36	brookxu	

故障分析	编辑
hawkeye: intel_pstate_timer_func() 触发除0 bug :)	

特征值

算法	特征串	组id	成立条件	权重	#	#
regexp	RIP:\s+\d{4}:\[<\w+>\]\s+\[<\w+>\]\s+intel_pstate_timer_func	0	包含	0.7	编辑	删除
regexp	divide\s+error:\s+0{4}	0	包含	0.7	编辑	删除

添加



宕机处理 – Crash自动分析工具

自动化 已知问题自动分析, 未知问题建单人工定位跟进

准确性 基于dmesg转变为基于vmcore内存镜像判断

时效性 宕机发生重启后就地分析, 第一时间发送诊断结果和解决方案

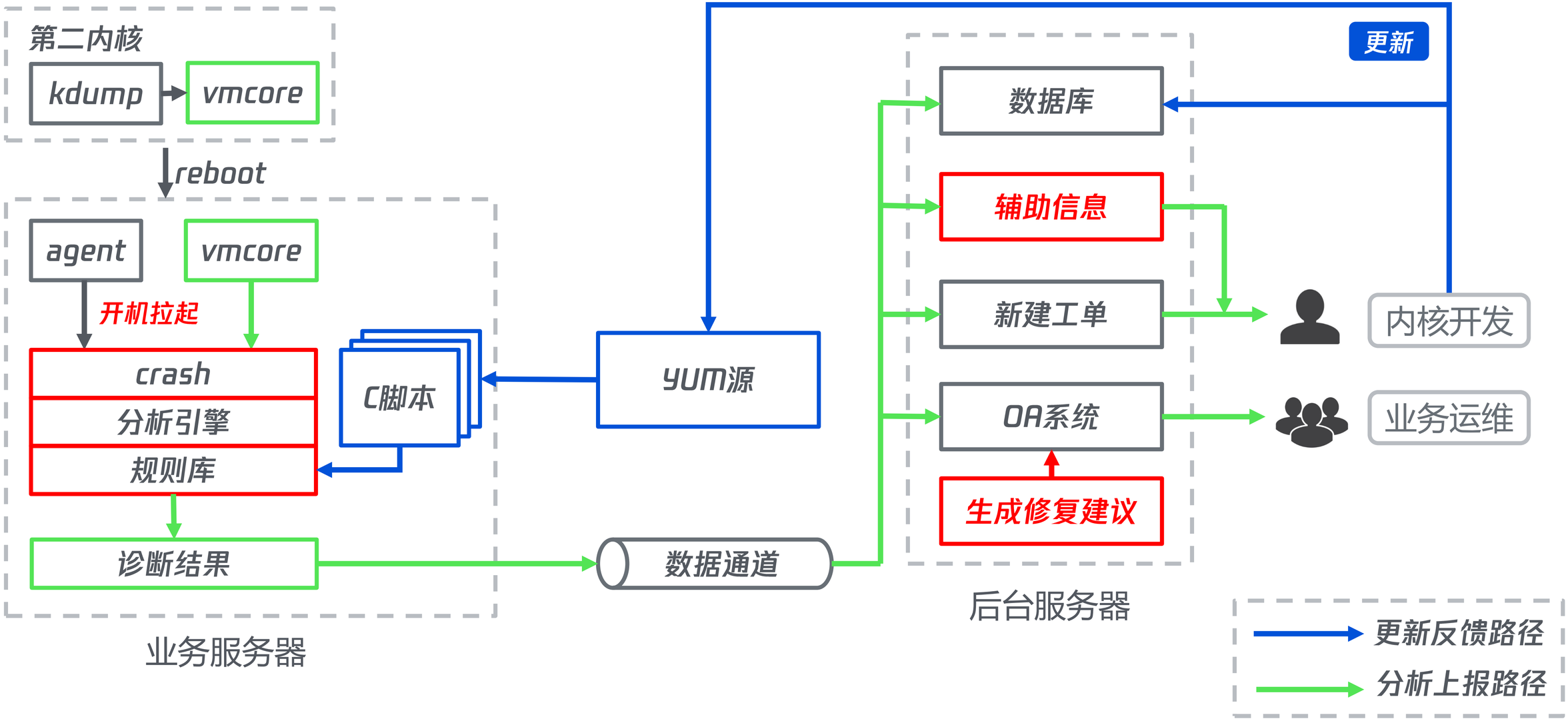
辅助性 未知问题提供问题相关信息辅助人工定位

内核运营模式转变

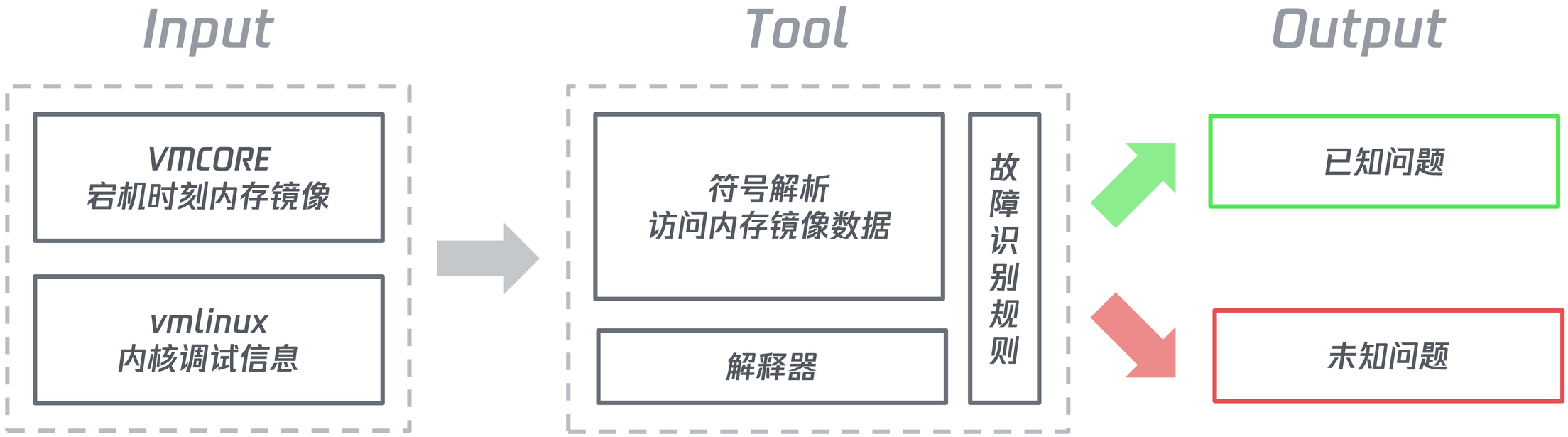
发生问题业务联系**被动**跟进解决 → **主动**处理上报跟踪问题

2 Crash自动分析系统概览

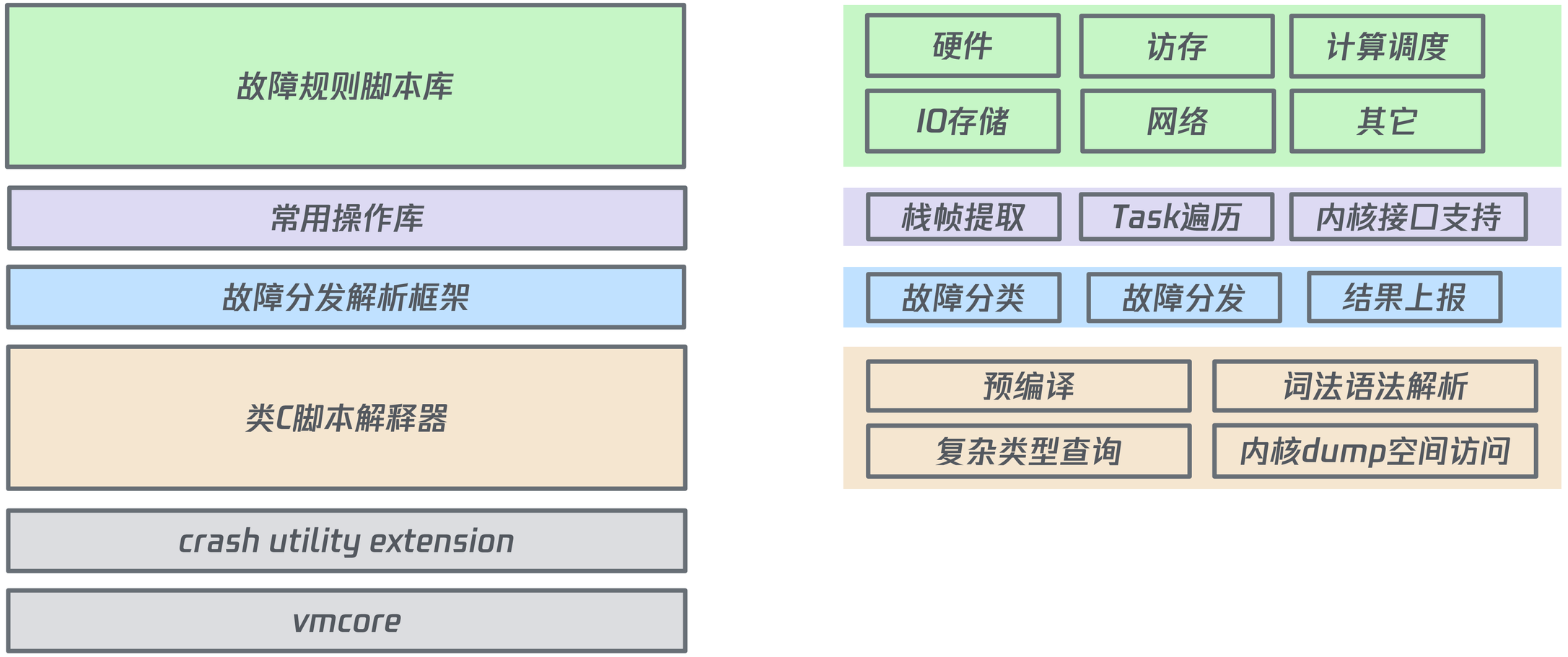
Crash自动分析系统处理流程



Crash自动分析系统原理



Crash自动分析工具结构

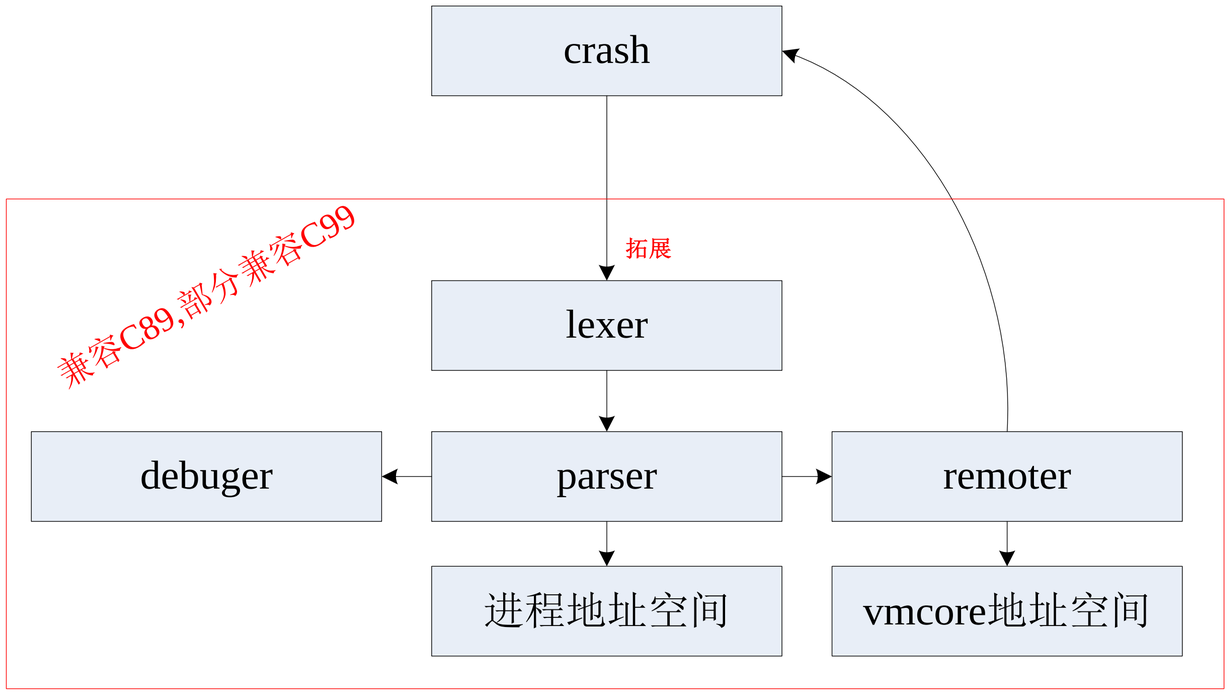


3 Crash自动分析系统实现

类C语言脚本解析器

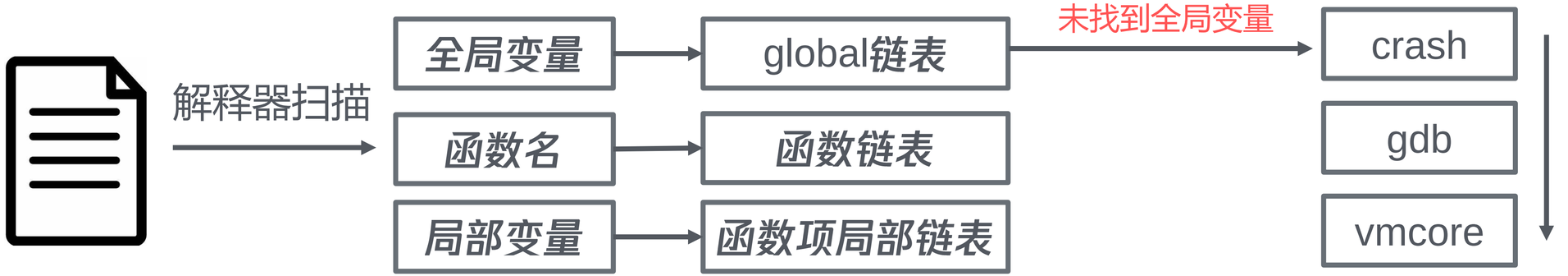
类C脚本解析器：

- 方便内核开发人员编写故障分析脚本
- 方便复用内核已有的数据结构
- 方便复用内核已有的逻辑

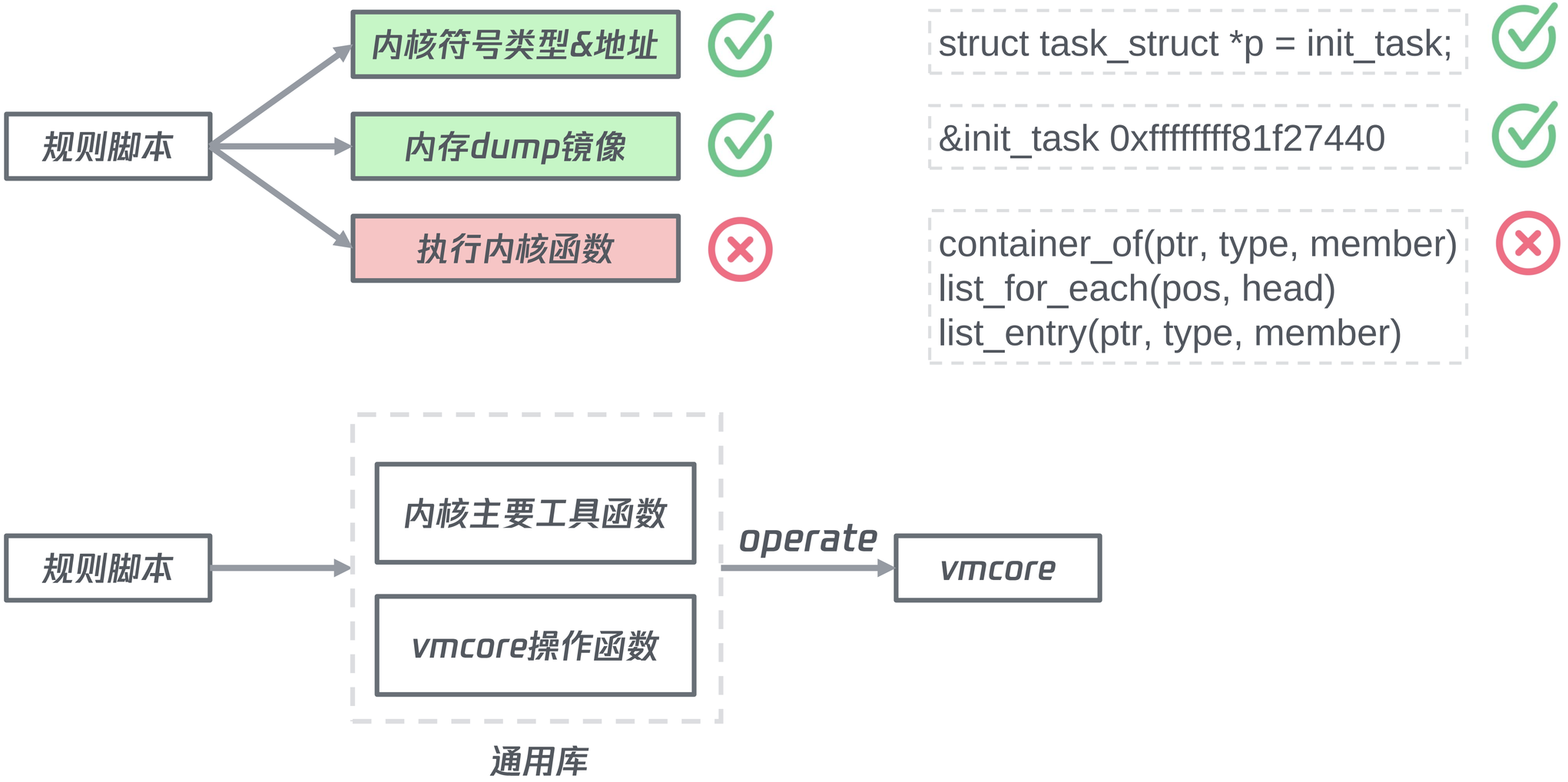


类C语言脚本解析器 – 解析进程空间与vmcore空间

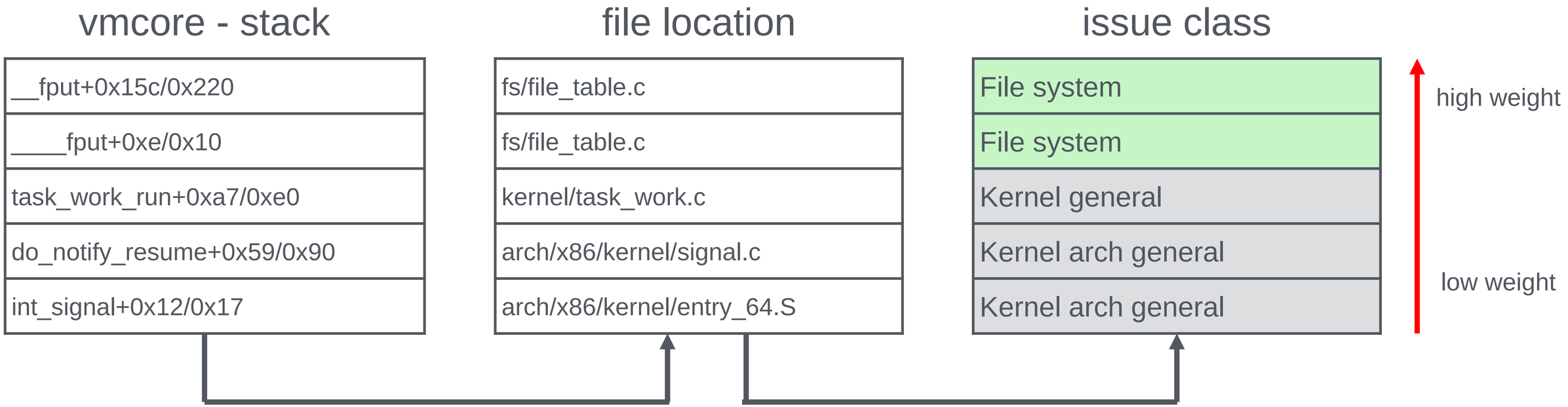

```
void foo()
{
        char c;
        struct task_struct *p = &init_task;
        …;
}
```



通用库支持


规则脚本
内核符号类型&地址
内存dump镜像
执行内核函数
struct task_struct *p = init_task;
&init_task 0xffffffff81f27440
container_of(ptr, type, member)
list_for_each(pos, head)
list_entry(ptr, type, member)
规则脚本
内核主要工具函数
vmcore操作函数
operate
vmcore
通用库

故障类型识别



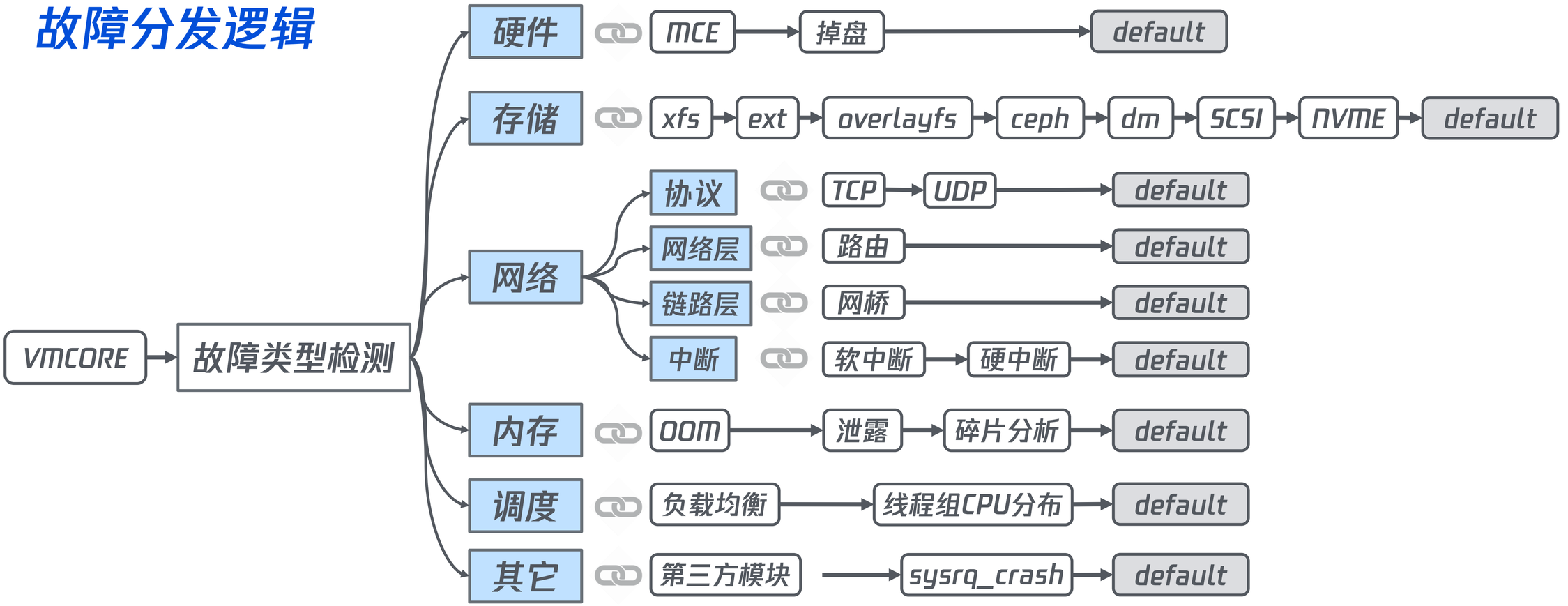
栈帧提取, 符号解析

.debug_info

文件-故障类型映射规则表

支持正则匹配

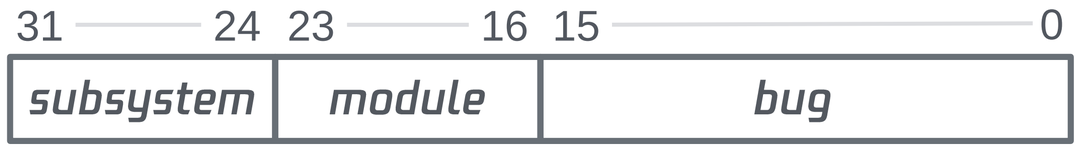
故障分发逻辑



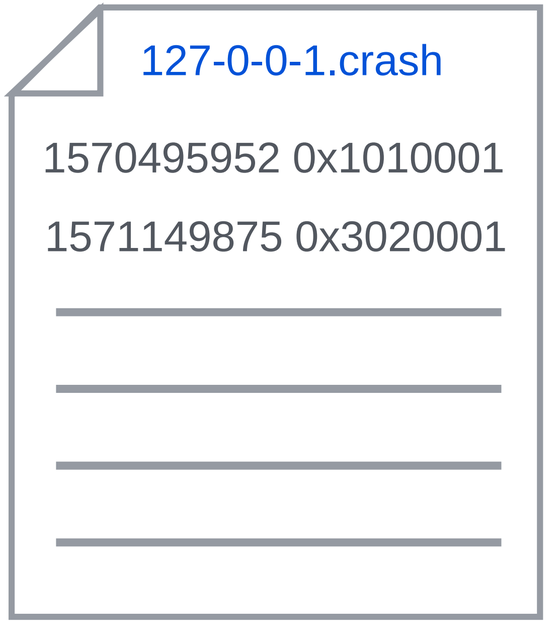
规则

- pre_fn：预过滤，匹配问题
- proc_fn：问题分析识别主体逻辑
- post_fn：识别失败收集辅助信息逻辑

BUG编号 & 预警

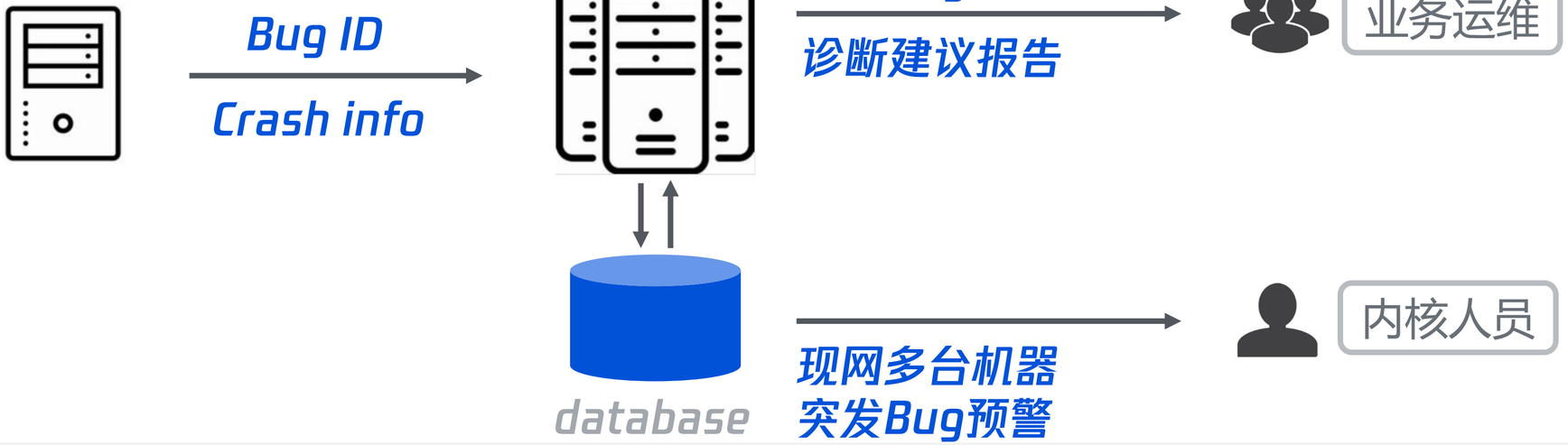


bug编码规则



1571149875 0x3020001

Timestamp
2019/10/15 22:31:15
subsystem
0x03 → file system
module
0x02 → xfs
bug
0x0001 → agi_agf_lock



未知问题的辅助分析

mutex死锁辅助信息

扫描所有进程得的栈，筛选出包含持有该mutex的进程，对于ABBA之类死锁，直接给出结论，
对于无法确定ABBA的，给出所有锁信息.

Soft lockup辅助信息

输出所有栈上持有的关联spin lock并且给出锁的信息，比如多少cpu尝试获取该所等，如果spin lock保护的是一个链表或者树，将整个链表或者树打印出来，用于辅助判断是否链表元素过多导致的soft lockup.

Slab泄露辅助信息

针对kmalloc-xx占用量异常的问题，根据size大小列出内核已经装载模块中所有对应size的结构信息，
并根据结构特征进行过滤，用于内核人员筛查.

Use after free辅助信息

通过栈回溯找到最近的有效数据结构，并给出哪一个字段被释放，方便分析人员重点关注
这个字段的分配和释放。

Etc.

基于已知问题可能出现的引发原因组合来生成类似未知问题的辅助信息收集策略

4 举例演示

sysrq主动触发宕机

```
int diag_other_sysrq_crash_proc_fn(void)
{
    unsigned long *ptr = get_cpu_ptr(kernel_stack, crashing_cpu);
    unsigned long kern_stack_end = ptr[0];
    unsigned long kern_stack = ptr[0] + 40 - THREAD_SIZE;
    unsigned long *irq_stack_ptr = get_cpu_ptr(irq_stack_ptr, crashing_cpu);
    unsigned long irq_stack_end = irq_stack_ptr[0];
    unsigned long irq_stack = irq_stack_ptr[0] - IRQ_STACK_SIZE;

    unsigned long rsp = crash_cpu_rsp(crashing_cpu);

    /* if we trigger alt-ctrl-c, we inside irq stack */
    if (rsp >= kern_stack && rsp < kern_stack_end ||
            rsp >= irq_stack && rsp < irq_stack_end) {
        struct pt_regs *regs = crash_cpu_regs(crashing_cpu);
        string ripsym = sym_name(regs->ip);

        if (ripsym == "sysrq_handle_crash") {
            /* ok, match!, gen descript msg */
            struct diag_bug *bug;
            string res;

            res = "我们检测到你名下的机器发生"+"宕机，系统初步诊断是";
            res += "主动宕机，一般是由于"+"用户态主动调用sysrq所置，";
            res += "如echo c > /proc/sysrq-trigger。";

            bug = diag_find_bug(MAKEID(SS_OTHER, 0, SYSRQ_CRASH));
            if (bug) {
                bug->descript = res;
                return 0;
            }
        }
    }
    return -1;
}
```

处理逻辑:

- 获取crash cpu栈的rip
- 判断rip是否处于sysrq_handle_crash函数中.

MCE故障分析脚本举例

```
int diag_hw_mce_proc_fn(void)
{
  struct diag_bug *bug;
  string res, msg;
  int i;

  res = "我们检测到你名下的机器发生"+"MCE, 系统初步诊断是";
  /* ok, we search mce_log and show mce_log item */
  /* which serverity is larger than SRAS */
  for (i = 0; i < MCE_LOG_LEN; i++) {
    struct mce *m = &mcelog.entry[i];
    if (!(m->status & MCI_STATUS_VAL))
      continue;
    if (m->status & MCI_STATUS_UC) {
      int ser = mce_severity(m);
      if (ser == MCE_PANIC_SEVERITY) {
        res += mce_diag(m);
        break;
      } else
        msg = mce_diag(m);
    }
  }
  /* mce panic, but serverity larger than MCE_PANIC_SEVERITY,
   * use last UC error message instead.
   */
  if (i == MCE_LOG_LEN)
    res += msg;

  res += "。MCE触发宕机通常是由于硬件发生了不可纠正错误，";
  res += "如果机器频繁发生此类错误，"+"建议提单检测机器硬件。";

  bug = diag_find_bug(MAKEID(SS_HW, HW_MCE, MCE_CPU));
  if (bug)
    bug->descript = res;
  return 0;
}
```

MCE的处理逻辑:

- 扫描crashing_cpu调用栈确定RSP
- 检查RSP是否处于MCE_STACK内
- 扫描mcelog队列获取严重度不小于PATNIC的mce
- 将mce的bank status寄存器转换为故障信息
- 追加故障处理建议

故障通知邮件

tlinux 宕机诊断报告

基本信息

- IP地址
- 宕机时间 2019-10-14-03:13
- 内核版本
- 硬件型号
- 业务部门
- 运维人员
- 所在机房

诊断信息

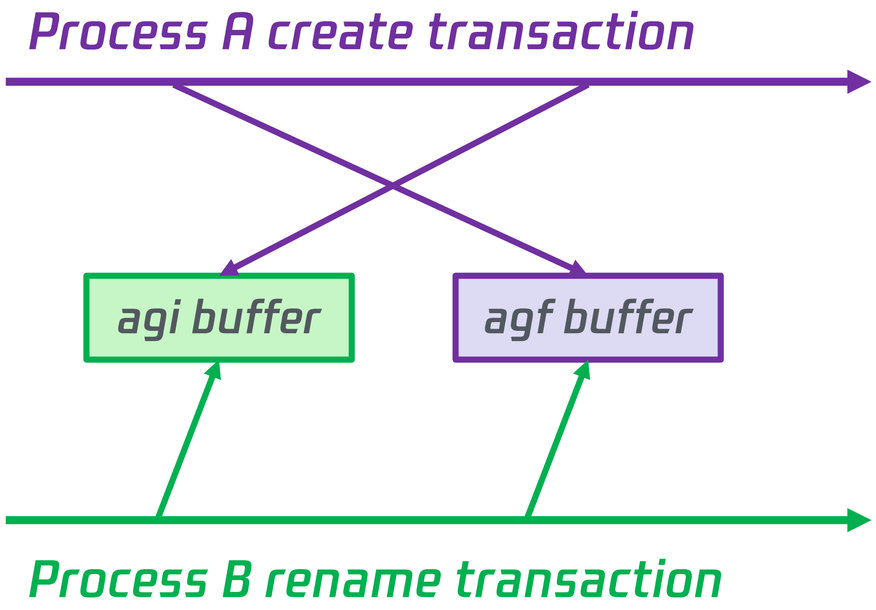
我们检测到你名下的机器发生MCE，系统初步诊断是内部数据校验出错。MCE触发宕机通常是由于硬件发生了不可纠正错误, 如果机器频繁发生此类错误，建议提单检测机器硬件。此信息为诊断系统自动生成, 如有不解之处, 请联系h_tlinux_helper。

宕机日志

```
[ 734.673583] acpi_pad_handle_notify, num_cpus: 0
[ 735.672372] acpi_pad_handle_notify, num_cpus: 0
[ 736.673685] acpi_pad_handle_notify, num_cpus: 0
[ 737.675726] acpi_pad_handle_notify, num_cpus: 0
[ 738.675691] acpi_pad_handle_notify, num_cpus: 0
[ 739.677012] acpi_pad_handle_notify, num_cpus: 0
[ 740.678182] acpi_pad_handle_notify, num_cpus: 0
[ 741.679361] acpi_pad_handle_notify, num_cpus: 0
[ 742.680599] acpi_pad_handle_notify, num_cpus: 0
[ 5167.355755] mce: [Hardware Error]: CPU 51: Machine Check Exception: 5 Bank 0: b2
000000000a0005
[ 5167.355840] mce: [Hardware Error]: RIP !INEXACT! 10: {pci_mmcfg_read+0x95/0xe0}
[ 5167.356046] mce: [Hardware Error]: TSC 1c238c0dcf669
[ 5167.356179] mce: [Hardware Error]: PROCESSOR 0:406f1 TIME 1570994040 SOCKET
1 APIC 35 microcode b000010
[ 5167.356261] mce: [Hardware Error]: Run the above through 'mcelog --ascii'
[ 5167.356338] mce: [Hardware Error]: CPU 23: Machine Check Exception: 5 Bank 0: b2
000000000a0005
```

XFS死锁检测

问题描述:
在docker场景下, 上层overlayfs会频繁给xfs下发rename请求, 在特定时刻, rename操作和create操作会相互持有agi和agf的buffer信号量, 造成ABBA型死锁.



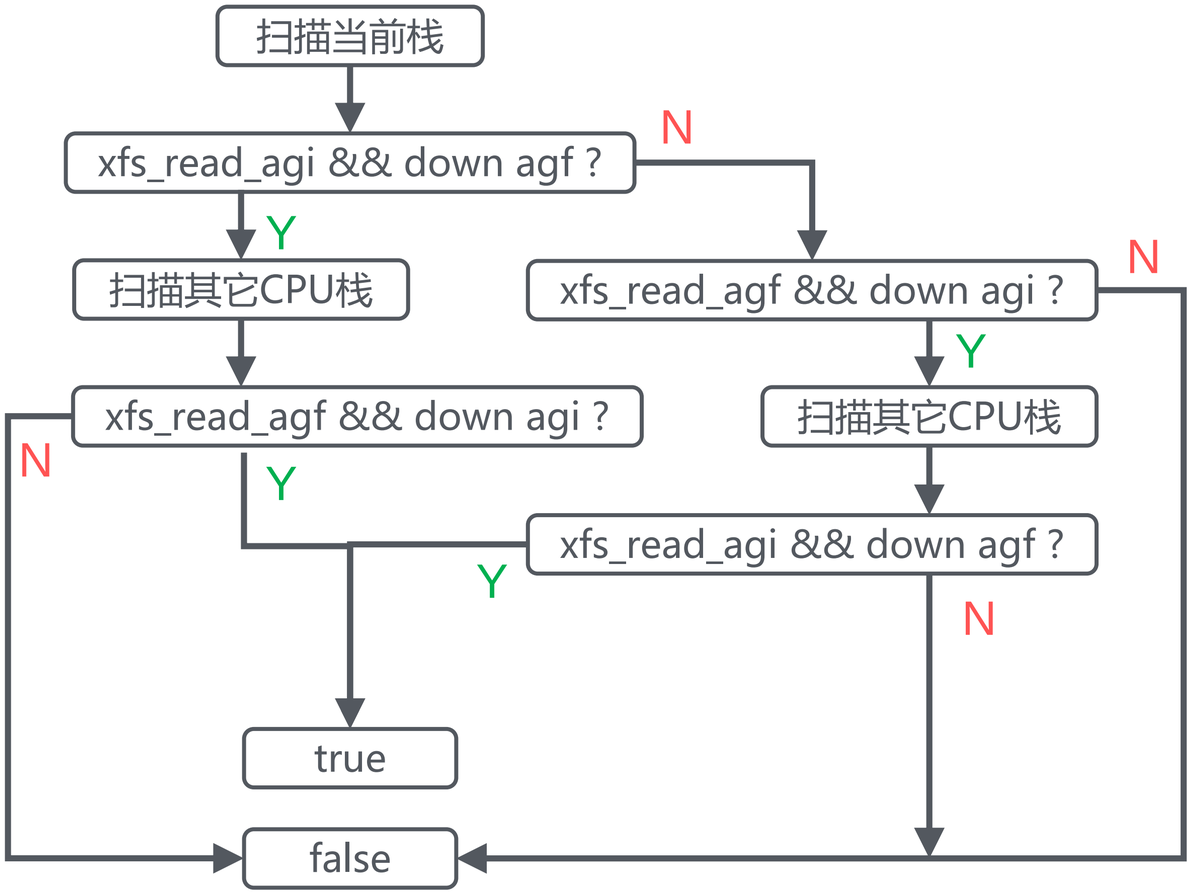
Process A

```
? __schedule+0x2bd/0x620
 schedule+0x33/0x90
 schedule_timeout+0x17d/0x290
 __down_common+0xef/0x125
 ? xfs_buf_find+0x215/0x6c0 [xfs]
 down+0x3b/0x50
 xfs_buf_lock+0x34/0xf0 [xfs]
 xfs_buf_find+0x215/0x6c0 [xfs]
 xfs_buf_get_map+0x37/0x230 [xfs]
 xfs_buf_read_map+0x29/0x190 [xfs]
 xfs_trans_read_buf_map+0x13d/0x520 [xfs]
 xfs_read_agf+0xa6/0x180 [xfs]
 ? schedule_timeout+0x17d/0x290
 xfs_alloc_read_agf+0x52/0x1f0 [xfs]
 xfs_alloc_fix_freelist+0x432/0x590 [xfs]
 ? down+0x3b/0x50
 ? xfs_buf_lock+0x34/0xf0 [xfs]
 ? xfs_buf_find+0x215/0x6c0 [xfs]
 xfs_alloc_vextent+0x301/0x6c0 [xfs]
 xfs_ialloc_ag_alloc+0x182/0x700 [xfs]
 ? _xfs_trans_bjoin+0x72/0xf0 [xfs]
 xfs_dialloc+0x116/0x290 [xfs]
 xfs_ialloc+0x6d/0x5e0 [xfs]
 ? xfs_log_reserve+0x165/0x280 [xfs]
 xfs_dir_ialloc+0x8c/0x240 [xfs]
 xfs_create+0x35a/0x610 [xfs]
 xfs_generic_create+0x1f1/0x2f0 [xfs]
```

Process B

```
? __schedule+0x2bd/0x620
? xfs_bmapi_allocate+0x245/0x380 [xfs]
schedule+0x33/0x90
schedule_timeout+0x17d/0x290
? xfs_buf_find+0x1fd/0x6c0 [xfs]
__down_common+0xef/0x125
? xfs_buf_get_map+0x37/0x230 [xfs]
? xfs_buf_find+0x215/0x6c0 [xfs]
down+0x3b/0x50
xfs_buf_lock+0x34/0xf0 [xfs]
xfs_buf_find+0x215/0x6c0 [xfs]
xfs_buf_get_map+0x37/0x230 [xfs]
xfs_buf_read_map+0x29/0x190 [xfs]
xfs_trans_read_buf_map+0x13d/0x520 [xfs]
xfs_read_agi+0xa8/0x160 [xfs]
xfs_iunlink_remove+0x6f/0x2a0 [xfs]
? current_time+0x46/0x80
? xfs_trans_ichgtime+0x39/0xb0 [xfs]
xfs_rename+0x57a/0xae0 [xfs]
xfs_vn_rename+0xe4/0x150 [xfs]
```

XFS死锁检测



commit bc56ad8c74b8588685c2875de0df8ab6974828ef
xfs: Fix deadlock between AGI and AGF with RENAME_WHITEOUT

tlinux 宕机诊断报告

基本信息

- IP地址
- 宕机时间 2019-10-11 14:01
- 内核版本
- 硬件型号
- 业务部门
- 运维人员
- 所在机房

诊断信息

我们检测到你名下的机器发生xfs agi agf死锁问题，之后的内核已经修复该问题，当前内核版本为，建议下载热补丁_xfs_fix_agi_afg_deadlock修复此问题。在过去的30天内该问题共计发生2次，建议尽早修复避免影响业务。如有任何问题，欢迎咨询TencentOS团队。

宕机日志

```
[53235.139975] [] schedule+0x29/0x70
[53235.139977] [] schedule_timeout+0x199/0x2c0
[53235.139990] [] ? xfs_bmbt_get_all+0x18/0x20 [xfs]
[53235.139993] [] __down_common+0x9e/0xf8
[53235.140003] [] ? _xfs_buf_find+0x1ee/0x3c0 [xfs]
[53235.140006] [] __down+0x1d/0x1f
[53235.140009] [] down+0x41/0x50
[53235.140019] [] xfs_buf_lock+0x3d/0x120 [xfs]
[53235.140029] [] _xfs_buf_find+0x1ee/0x3c0 [xfs]
[53235.140039] [] xfs_buf_get_map+0x2a/0x190 [xfs]
[53235.140050] [] xfs_buf_read_map+0x2c/0x130 [xfs]
[53235.140065] [] xfs_trans_read_buf_map+0x2e9/0x590 [xfs]
[53235.140079] [] xfs_read_agi+0xb0/0x120 [xfs]
[53235.140092] [] xfs_ialloc_read_agi+0x1c/0xc0 [xfs]
[53235.140103] [] xfs_dialloc+0xe8/0x290 [xfs]
[53235.140116] [] xfs_ialloc+0x7a/0x760 [xfs]
[53235.140130] [] ? xfs_trans_mod_dquot+0x75/0x2d0 [xfs]
[53235.140142] [] xfs_dir_ialloc+0xac/0x2e0 [xfs]
[53235.140154] [] xfs_create+0x3f9/0x6d0 [xfs]
[53235.140165] [] xfs_vn_mknod+0xb9/0x230 [xfs]
[53235.140176] [] xfs_vn_create+0x13/0x20 [xfs]
[53235.140179] [] vfs_create+0x8c/0x110
```

5 未来工作展望

- 推出更方便bug分析的脚本语言
- 更智能化的自动分析
- 在无法自动分析得出结论时，做最大化的辅助分析
- 支持更完善的内核接口
- 更高效的脚本运行效率
- 开源

TencentOS团队开源地址
http://mirrors.tencent.com/tlinux/

故障分析可以自动化

已知问题全自动化处理

未知问题辅助处理实现半自动化

6 Q & A

Thanks