

Linux crash dump 読み方入門

LinuxのKernel crash dumpからの
障害原因調査

2007年2月 29日
1.1版

ミラクル・リナックス株式会社
Asianux開発本部 吉田

AXcelerate Your Business

アジェンダ

- 対象、目的、前提知識
- dump取得ツール/解析ツール
- lcrashでのdump解析
- crashでのdump解析
- まとめ

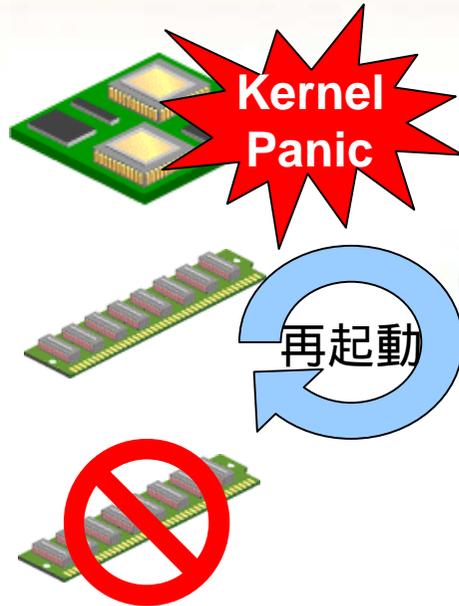
対象の方

Dump?

- 企業で業務にLinuxサーバを使っている/管理している方
- 企業にLinuxサーバを導入している方
- 対象は通常のIAサーバ+Linux

Dumpとは(1)

- サーバが停止！
しかし、ダンプを取得していない(設定していない)場合



Linuxが管理するデータ、プログラムは
全てメモリに存在

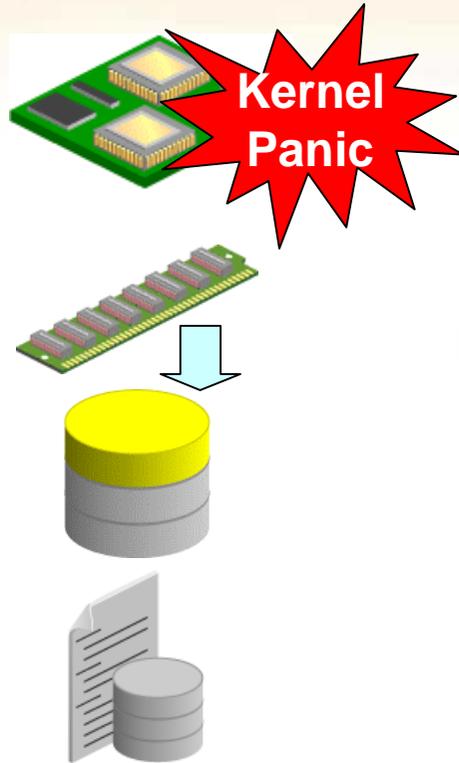
Linuxが
異常を起こし、再起動実施

再起動によってメモリは初期化され
手がかりなし

- 原因不明、対策不可

Dumpとは(2)

- サーバーは停止したが、ダンプが取得できた場合



Linuxが異常を起こした

Linuxが管理するデータ、プログラムは全てメモリに存在する。

ログに記録を残せない状態なので、強制的にクラッシュダンプを取得する。

ダンプファイル(Dump)が残るので原因究明が可能
対策を講じることができる。

Kernelpanic時のメモリの内容=>Dump

AXcelerate Your Business

Dumpを解析する目的(1)

- 業務サーバの安定性を高める
- Linuxサーバ(OS)の障害の現象パターン
 - Kernel Panic ← 今回のメイン

サーバが何か変なメッセージで止まっているぞ？

- フリーズ(ストール、ハングアップ、スローダウン)

解析が難しい

サーバが固まったぞ？
ものすごく遅いぞ？

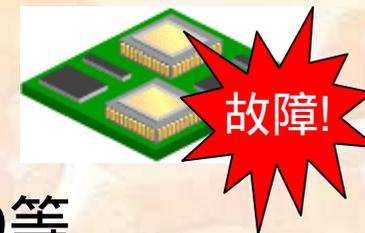
AXcelerate Your Business

Dumpを解析する目的(2)

- KernelPanicの原因

- ハードウェア障害

- マザー、CPU、メモリ、GPU、電源、DISK、RAID等



- Kernel bug

- (アプリケーションのバグ)

0除算エラー!

メモリ保護違反!

Oops!

AXcelerate Your Business

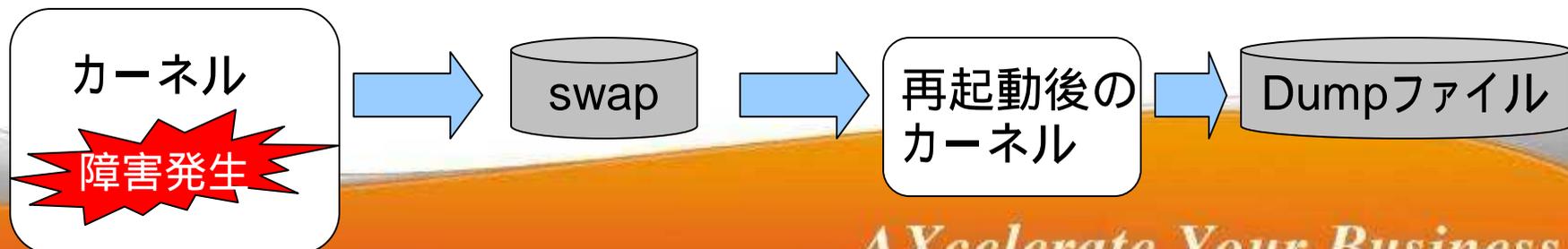
Dump取得ツールの種類

- LKCD(Linux Kernel Crash Dump)
 - MiracleLinux Ver2.1, Ver3
- diskdump, netdump
 - MiracleLinux Ver4, RHEL4等
- Kdump
 - Asianux Server 3, RHEL5等

最新のDump
取得ツール

LKCD

- LKCD
 - Swapパーティション等のdumpデバイスにdumpを出力
 - 再起動時に上記からdumpファイルに出力する
 - /var/log/dumpディレクトリ
 - 詳細は以下参照
 - <http://lkcd.sourceforge.net/>



AXcelerate Your Business

diskdump

- dump専用パーティションにdump

/etc/sysconfig/diskdumpに設定

```
# service diskdump initialformat
```

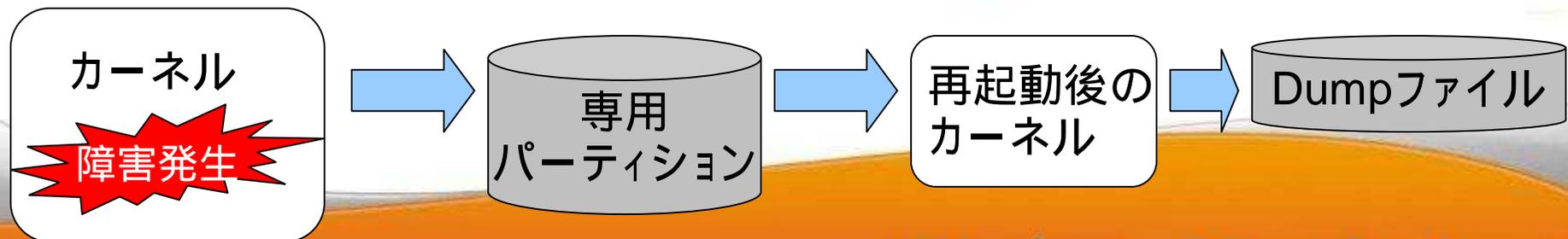
```
# service diskdump start
```

```
# chkconfig diskdump on
```

- 詳細はガイドを参照

<http://www.miraclelinux.com/technet/document/linux/ml40/index.html>

<http://www.miraclelinux.com/technet/document/linux/ml40/pdf/27.pdf>



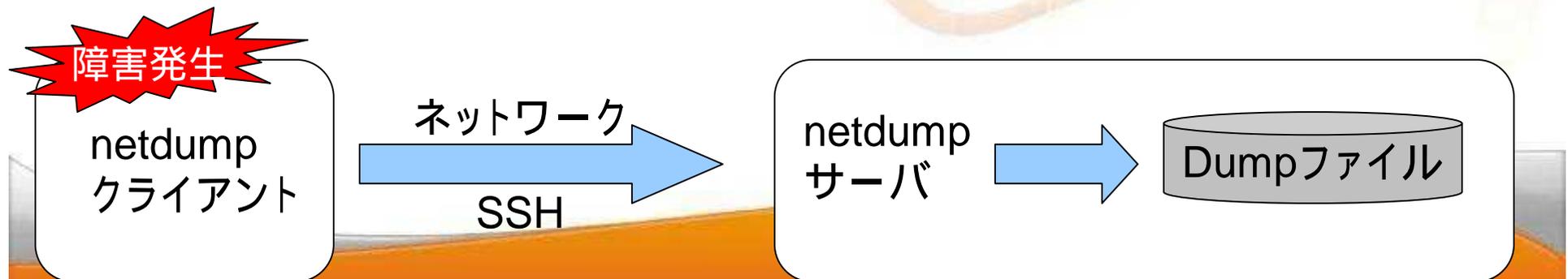
AXcelerate Your Business

netdump

- ssh経由で別マシンにdumpを採取
- 詳細はガイドを参照

<http://www.miraclelinux.com/technet/document/linux/ml40/index.html>

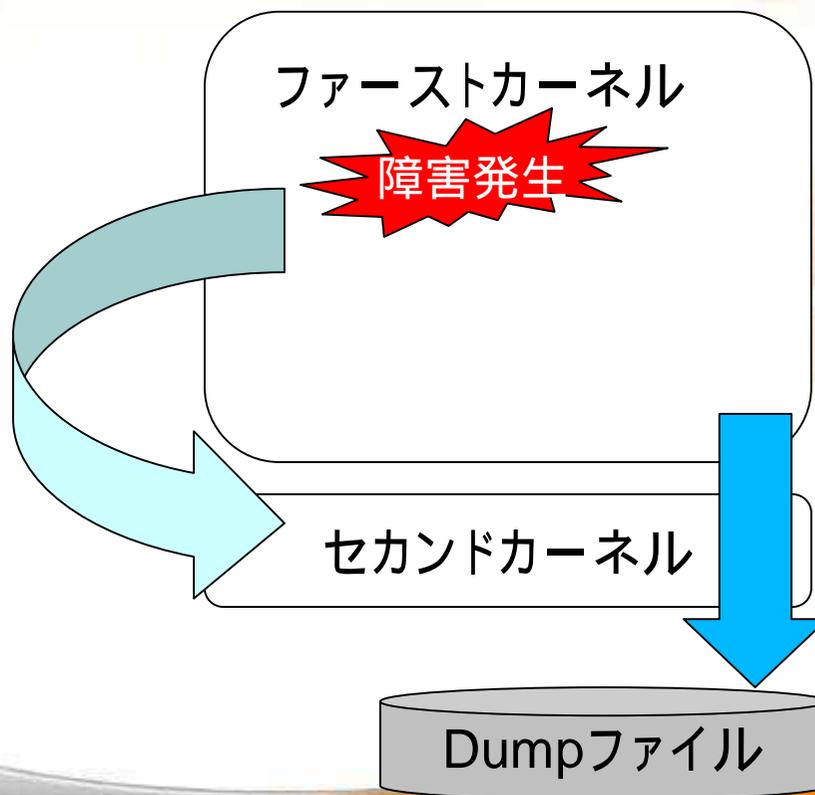
<http://www.miraclelinux.com/technet/document/linux/ml40/pdf/27.pdf>



AXcelerate Your Business

Kdump(1)

- 起動時にdump専用のkernelを準備する



1.起動時に予約した領域にdump専用カーネルを準備

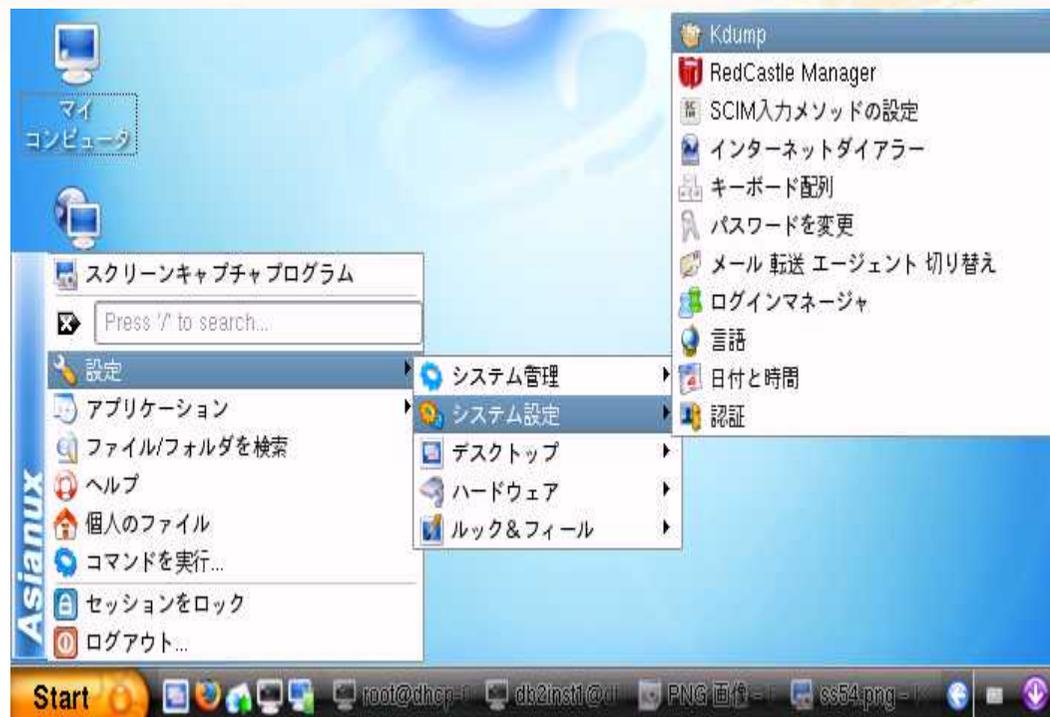
2.通常運用時はセカンドカーネルの領域は使用しない

3.障害を検知するとセカンドカーネルが起動される

4.セカンドカーネルがファーストカーネルのメモリ内容をdumpファイルへ保存

Kdump(2)

- 起動時にdump専用のkernelを準備する



- 設定後再起動

Kdump(3)

- /boot/grub/menu.lstを確認し、カーネルのオプションにcrashkernel= ~ の記述が追加されていることを確認

```
title Asianux Server 3 (2.6.18-8.10AX)
  root (hd0,0)
  kernel /boot/vmlinuz-2.6.18-8.10AX ro root=LABEL=/ crashkernel=128M@16M
  initrd /boot/initrd-2.6.18-8.10AX.img
```

障害に備えて事前に設定、確認する項目(1)

- panic時の動作の設定

- # cat /proc/sys/kernel/panic
- パニック時に自動rebootする秒数(デフォルト0:リブートしない)
- メリット:自動的に再起動する
- デメリット:panic時のメッセージ等が見えない
 - /var/log/message等にかかれていている可能性もある
 - netconsole,シリアルコンソールの利用も検討



障害に備えて事前に設定、確認する項目(2)

- oops時の動作の設定

- oops:kernel内部で何か異常だと判断された状態で出力

```
# cat /proc/sys/kernel/panic_on_oops
```

oops発生時にpanicを発生させる(1)

oops発生時に継続させる(0)

メリット:早期検知

デメリット:業務中断発生

障害発生

? ..

oops

Panic

障害に備えて事前に設定、確認する項目(3)

- nmi_watchdogの有効化

- カーネルのブートオプションに、nmi_watchdog=1、もしくはnmi_watchdog=2を追加
- /proc/interruptsでNMIの数値が増えるか確認
- 一部のストールを検知してOops メッセージを出力(panic_on_oops設定でpanic,dump)させる
- 注意:使えるマシンが限られる

- sysrqの有効化

- キーボードまたは/procインターフェイスからのsync,dump,再起動等を有効化
- /etc/sysctl.confにkernel.sysrq = 1を設定し、#sysctl -p
 - または# echo 1 > /proc/sys/kernel/sysrq

障害に備えて事前に設定、確認する項目(4)

- /varの空き容量
 - dumpが出力する度に、メモリ容量分のDiskを消費
 - LKCD : /var/log/dump
 - diskdump,kdump : /var/crash
 - netdump : netdumpサーバの/var/crash

Dumpテスト

- NMIスイッチ(ボタン)押下
- Sysrq-key の同時押し
 - Alt-SysRq-s 'S'ync
 - Alt-SysRq-c 'C'rash
 - Alt-SysRq-b re'B'oot
- Sysrq-trigger
 - # sync;sync;sync
 - # echo 1 > /proc/sys/kernel/sysrq
 - # echo c > /proc/sysrq-trigger



コンソールメッセージの例(1)

```
Process bash (pid: 4288, threadinfo 000001001b1ea000, task 000001001b1ea000)
Stack: ffffffff802667d9 000001001b1ea000 0000000000000002 000001001b1ea000
       0000000000000002 0000002a97d08000 ffffffff801d45a7 0000000000000000
       0000000000000000 000001001dce8a40
Call Trace:<fffffff802667d9>{__handle_sysrq+102} <fffffff801d45a7>{__handle_sysrq+43}
       <fffffff8018aa33>{vfs_write+207} <fffffff8018ab1b>{sys_write+10}
       <fffffff80110984>{LKST_ETYPE_SYSCALL_ENTRY_HEADER_hook+129}
Code: c6 04 25 00 00 00 00 00 c3 e9 e3 5d f2 ff e9 ef cb f2 ff 48
RIP <fffffff80266679>{sysrq_handle_crash+0} RSP <000001001b1ebec0>
CR2: 0000000000000000
<0>Kernel panic - not syncing: Oops
```

コンソールメッセージの例(2)

```
<fffffffffffa0184d58>{:ipv6:ip6_push_pending_frames+779}
<fffffffffffa0197714>{:ipv6:rawv6_sendmsg+2273} <ffffffffff802e59ab>{sys_sendmsg+271}
<ffffffffff80259684>{n_tty_receive_buf+2677} <ffffffffff80259684>{n_tty_receive_buf+2677}
<fffffffffffa01911f4>{:ipv6:ipv6_setsockopt+2252} <ffffffffff80259684>{n_tty_receive_buf+2677}
<ffffffffff802e59ab>{sys_sendmsg+454} <ffffffffff8017794a>{har
9}
<ffffffffff80135559>{default_wake_function+0} <ffffffffff80125
ult+509}
<ffffffffff802e6187>{sock_set_timeout+31} <ffffffffff801a12ce>
}
<ffffffffff80110984>{LKST_ETYPE_SYSCALL_ENTRY_HEADER_hook+12

Code: 48 8b 98 88 01 00 00 48 85 db 74 06 ff 83 10 01 00 00 48 8d
RIP <fffffffffffa01987d7>{:ipv6:icmpv6_send+1328} RSP <000001001a817
CR2: 00000000000000188
<0>Kernel panic - not syncing: Oops
```

AXcelerate Your Business

dumpで生成されるファイル

- LKCD
 - dump.x(ほぼサーバ実装のメモリサイズ分)
 - map.x(メモリマップ)
 - kerntypes.x等
- Kdump,diskdump,netdump
 - vmcore(ほぼサーバ実装のメモリサイズ分)

dump解析ツール

- lcrash
 - lkcdの形式に対応
 - MiracleLinux Ver2.1, Ver3.0
- crash
 - Kdump, diskdump, netdumpの形式に対応
 - MiracleLinux Ver4.0 Asianux Server 3
 - RHEL4,5等

dump解析に必要な知識

- IAプロセッサ(x86_64)の知識
- C言語、アセンブラの知識
- kernel構造の知識

CPUの知識

- プログラムのコードがCPUで実行される仕組み
- 対象が32bitか?x86_64か?
 - 32bit=i386 ~ i686,x86_64=AMD64,Intel 64(EMT64)
- レジスタ名称と役割

CPU

クロック

処理装置

レジスタ[(E|R)AX]

スタックポインタ [(E|R)SP]

命令ポインタ [(E|R)IP]

フラグレジスタ [(E|R)FLAGS] 等

メモリ

a1	実行命令1	a5 を r1に
a2	実行命令2	a6をr2に
a3	実行命令3	r1とr2を足してr2
a4	実行命令4	r2のデータをa7へ
a5	データ1 [1]	
a6	データ2 [1]	
a7	データ3 [a3]	

詳細はIntelが出しているガイドブック等を見る

C言語、アセンブラの知識

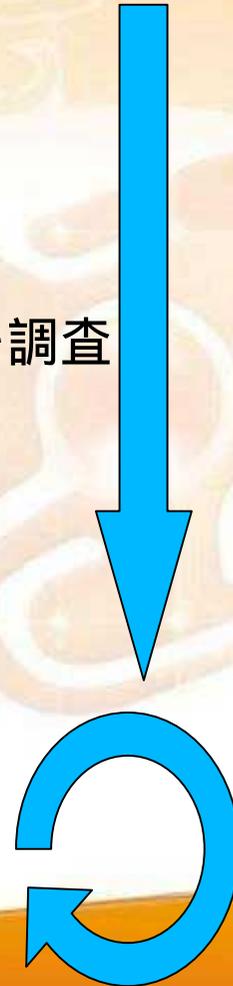
- C言語
 - GCCやプリプロセッサ、マクロの理解
- アセンブラ
 - ニーモニック

kernelの知識

- kernelの動作、該当機能
- kernelコードを追う
- kernelのパッケージ構成
- bugzillaやLKMLのアーカイブ

Dump解析の流れ(Icrash)

1. Dumpが生成された状況を確認
2. analysis.xが生成されていれば確認
3. Icrashでdumpを読み込む
4. メッセージ(stat,report)の調査
kernel内でどこで出しているか、ソースを調査
5. バックトレースの調査(bt)
どのプロセスが呼び出しているか
どこでpanicになったか
6. プロセスの状態調査(ps)
7. 逆アセンブル(dis)
怪しいプロセス、処理を逆アセンブル
8. 該当するソースを確認
9. メモリの値を確認(print 式)



実例1

- サーバが早朝4時位に再起動していました。
- その時間の動作プロセス
 - cron
 - whatis,locate等の更新
 - バックアップソフト
- 各種ログを調べると/var/log/dumpにdumpが残っていた。
 - LKCDのdump
- サーバはいままで安定稼働していました。

analysis.x(状態確認)

```
# less analysis.0
|
=====
STACK TRACE OF FAILING TASK
=====

=====
STACK TRACE FOR TASK: 0xea874000 (nvdevmgr)

  0 LKST_ETYPE_0_PANIC_HEADER [0xc01262e1]
TRACE ERROR 0x800000000
=====
```

实例1 lcrash 启动

```
# lcrash map.0 dump.0 kerntypes.0
```

```
DUMP INFORMATION:
```

```
architecture: i386  
  byte order: little  
  pointer size: 32  
bytes per word: 4  
  
kernel release: 2.4.9  
  memory size: 939524096 (0G 896M 0K 0Byte)  
num phys pages: 655340  
number of cpus: 2
```

AXcelerate Your Business

lcrash stat,report(状態確認)

```
>> stat↓
```

```
<4>Ops: 0000↓
```

```
<4>Kernel 2.4.9-e.25.50msmp↓
```

```
<4>CPU: 1↓
```

```
<4>EIP: 0010:[<c01c93fb>] Not tainted↓
```

```
<4>EFLAGS: 00010002↓
```

```
<4>EIP is at generic_unplug_device [kernel] 0xb↓ 問題の場所
```

```
<4>eax: 4545455d ebx: 4545455d ecx: 00000000 edx: d17e6bc0
```

```
<4>esi: 00000202 edi: e9b68040 ebp: ea875f24 esp: ea875eb8
```

```
<4>ds: 0018 es: 0018 ss: 0018↓
```

レジスタ

```
<4>Process nvdevmgr (pid: 17958, stackpage=ea875000)↓ プロセス
```

```
<4>Stack: d17e6bc0 e9b68064 e9b68040 ea875f24 f8a8f137 4545455d d17e6bc0
```

```
<4> dc2d8000 00000400 f8a901b0 00015f90 00000001 00000001 400161fa
```

```
<4> 400161fa f8a8ecc8 e9b68000 e9b68040 ea875f24 00015f90 00015f90 00015f90
```

```
<4>Call Trace: [<f8a8f137>] sg_common_write [sg] 0x247↓
```

```
<4>[<f8a901b0>] sg_cmd_done_bh [sg] 0x0↓
```

実行履歴

```
<4>[<f8a8ecc8>] sg_write [sg] 0x2d8↓
```

```
<4>[<c0160c76>] sys_write [kernel] 0x96↓
```

```
<4>[<c0110240>] LKST_ETYPE_SYSV_IPC_HEADER [kernel] 0x81↓
```

```
<4>[<c0108016>] LKST_ETYPE_SYSCALL_ENTRY_HEADER [kernel] 0x36↓
```

```
<4>↓
```

```
<4>↓
```

LKSTの動作ログ(今回は関係なさそう)

```
<4>Code: 8b 83 84 00 00 00 f0 fe 08 0f 88 4f f5 0c 00 8b 0d 70 bb 83 84 00 00 00 00
```

```
<4> <0>Kernel panic: not continuing↓
```

```
<4>↓
```

lcrash ps(プロセス状態)

```
>> ps
  ADDR      UID      PID      PPID     STATE      FLAGS CPU  NAME
=====
0xea874000   0    17958    1046      0    0x800100   -  nvdevmgr
```

lcrash bt(バックトレース)

```
>> bt
```

```
=====
```

```
STACK TRACE FOR TASK: 0xea874000(nvdevmgr)
```

```
0 die+242 [0xc01089a2]
1 do_page_fault+1121 [0xc0120061]
2 generic_unplug_device+11 [0xc01c93fb]
3 [scsi_mod]scsi_dispatch_cmd+501 [0xf8800995]
4 [scsi_mod]scsi_request_fn+1341 [0xf880a27d]
5 do_page_fault [0xc011fc00]
6 error_code+96 [0xc0108168]
7 generic_unplug_device+11 [0xc01c93fb]
8 [sg]sg_common_write+583 [0xf8a8f137]
9 [sg]sg_cmd_done_bh [0xf8a901b0]
10 [sg]sg_write+728 [0xf8a8ecc8]
11 sys_write+150 [0xc0160c76]
12 LKST_ETYPE_SYSV_IPC_HEADER+129 [0xc0110240]
13 LKST_ETYPE_SYSCALL_ENTRY_HEADER+54 [0xc0108016]
14 startup_32+43 [0xc010002b]
```

Dump処理
(今回は関係
なさそう)

問題発生

実行履歴

実行
順序

AXcelerate Your Business

暫定対処

- nvdevmgr=>バックアップソフト
 - panic時テープにバックアップ処理を動かしていた
 - sgデバイスはSCSIのデバイスドライバ
- バックアップ処理中にkernelのbugにあたった？
ハード障害？
- バックアップ処理を控えてもらう

dis(ディスアSEMBル)

- panicが発生したgeneric_unplug_device+11 の処理内容を確認

実行順序 ↓

```
<generic_unplug_device>:      pushl   %ebp
<generic_unplug_device+1>:    pushl   %edi
<generic_unplug_device+2>:    pushl   %esi
<generic_unplug_device+3>:    pushl   %ebx
<generic_unplug_device+4>:    movl    0x14(%esp,1),%ebx
<generic_unplug_device+8>:    pushf
<generic_unplug_device+9>:    popl   %esi
<generic_unplug_device+10>:   cli
<generic_unplug_device+11>:   movl    0x84(%ebx),%eax
<generic_unplug_device+17>:   lock  decb (%eax)
```

- メモリ内容をebxレジスタに保存して
- ebxレジスタの値を使用している部分でエラー

Icrash dis(例2)

- sg_common_writeのgeneric_unplug_deviceが呼び出されるまでの動作を確認

```
>> dis [sg]sg_common_write 200
```

```
<[sg]sg_common_write+564>:  pushl  %ebp
<[sg]sg_common_write+565>:  pushl  %ebx
<[sg]sg_common_write+566>:  call   0xf8800bb0 <[scsi_mod]scsi_do_req>
<[sg]sg_common_write+571>:  movl   0x4c(%ebx),%eax
<[sg]sg_common_write+574>:  addl   $0x18,%eax
<[sg]sg_common_write+577>:  pushl  %eax
<[sg]sg_common_write+578>:  call   0xc01c93f0 <generic_unplug_device>
<[sg]sg_common_write+583>:  addl   $0x20,%esp
```

実行順序

- generic_unplug_deviceを呼ぶ前にscsi_do_reqを呼び出している。

ソースの確認(sg.c)

```
static int sg_common_write(Sg_fd * sfp, Sg_request * sr
                          unsigned char * cmd, int ti
{
```

```
    scsi_do_req(SRpnt, (void *)cmd,
                (void *)SRpnt->sr_buffer, hp->dxfer_len,
                sg_cmd_done_bh, timeout, SG_DEFAULT_RETRIES);
    /* dxfer len overwrites SRpnt->sr buflen, hence need for
    generic_unplug_device(&SRpnt->sr_device->request_queue);
    return 0;
}
```

```
<[sg]sg_common_write+564>:  pushl   %ebp
<[sg]sg_common_write+565>:  pushl   %ebx
<[sg]sg_common_write+566>:  call    0xf8800bb0 <[scsi_mod]scsi_do_req>
<[sg]sg_common_write+571>:  movl    0x4c(%ebx),%eax
<[sg]sg_common_write+574>:  addl    $0x18,%eax
<[sg]sg_common_write+577>:  pushl   %eax
<[sg]sg_common_write+578>:  call    0xc01c93f0 <generic_unplug_device>
<[sg]sg_common_write+583>:  addl    $0x20,%esp
```

事例/ソースを調べる(1)

- 該当するソース(src.rpm)を展開し、関連する部分のソースを比較、今回の例では、sg.cを最新版カーネルと比べる
 - 対象のカーネルは2.4.9-e.25.50mlなので、2.4.9系を使っているkernelを調査
 - RHEL2.1
 - 現状の最新は2.4.9-e.72、そのsg.cと比較すると、差分がある
 - 差分についてのコメントを見ると
 - * Mon Jun 9 2003 Tom Coughlan <coughlan@redhat.com>
 - - Fix bug 75669, **SG driver kernel oops due to a null pointer**
 - in sg_common_write. Fixed based on upstream. Patch added to scsifixes.patch.

事例/ソースを調べる(1)

- 2.4.9-e.72のspecファイルによると
- =====
- * Mon Jun 9 2003 Tom Coughlan <coughlan@redhat.com>
- - Fix bug 75669, **SG driver kernel oops due to a null pointer**
- **in sg_common_write**. Fixed based on upstream. Patch added to
- scsifixes.patch.
- =====

バグ情報を検索

- SPECを参考にbugzillaを見ると以下が該当する修正
https://bugzilla.redhat.com/show_bug.cgi?id=75669

```
+   q = &SRpnt->sr_device->request_queue;
   SRpnt->sr_request.rq_dev = sdp->i_rdev;
   SRpnt->sr_request.rq_status = RQ_ACTIVE;
   SRpnt->sr_sense_buffer[0] = 0;
@@ -715,7 +717,8 @@
           (void *)SRpnt->sr_buffer, hp->dxfer_len,
           sg_cmd_done_bh, timeout, SG_DEFAULT_RETRIES);
   /* dxfer_len overwrites SRpnt->sr_bufflen, hence need for
-   generic_unplug_device(&SRpnt->sr_device->request_queue);
+   generic_unplug_device(q);
```

事例1の原因と対処

- ユーザから、ハードウェア障害だったとの連絡
- Kernelに潜在的な問題がある可能性があるため、Kernel開発チームに情報を上げて終了

crashでのdumpの解析(1)

- 準備するファイル

- debug版vmlinux(debug版kernelパッケージ)
 - dump時のkernelと同一バージョンのdebug版のkernelのファイルが必要
 - 例) kernel-PAE-debuginfo-2.6.18-8.10AX.i686.rpm
 - kernel-debuginfo-2.6.18-8.10AX.x86_64.rpm
- System.map(通常のkernelパッケージに含まれる)
 - dump時のkernelが使用していたファイル
- vmcore(dump本体:メモリイメージ):/var/crash/日付/vmcore

debug版kernelが必要

AXcelerate Your Business

crashでのdumpの解析(2)

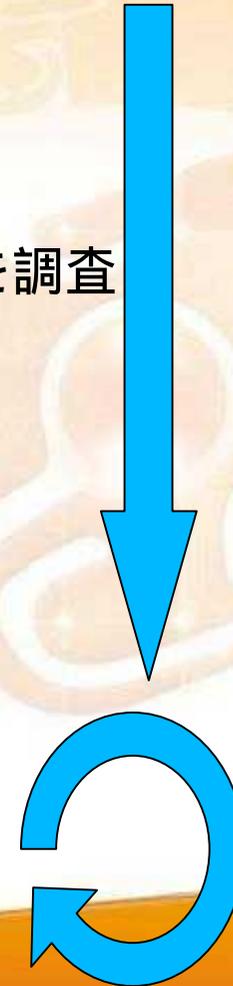
- 注意事項
 - dump発生マシンと別ディストリビューションでも (crashが動くなら) dump解析は可能
 - ただし、dump出力したマシンとアーキテクチャ (32bit,x86_64等)の違うマシンでは解析ができない

同一アーキテクチャのマシンが必要

AXcelerate Your Business

Dump解析の流れ(crash)

1. Dumpが生成された状況を確認
2. crashでdumpを読み込む
3. メッセージ(log)の調査
kernel内でどこで出しているか、ソースを調査
4. バックトレースの調査(bt)
どのプロセスが呼び出しているか
どこでpanicになったか
5. プロセスの状態調査(ps)
6. 逆アセンブル(dis)
怪しいプロセス、処理を逆アセンブル
7. 該当するソースを確認(gdb list)
8. メモリの値を確認(struct xx アドレス)



実例2

- カーネルモジュールを作成してテスト中に
panic発生 

实例(1) crash起動

```
# crash System.map vmlinux vmcore
```

```
LOAD AVERAGE: 0.00, 0.00, 0.00
TASKS: 62
NODENAME: localhost.localdomain
RELEASE: 2.6.9-42.7AXcustom
VERSION: #11 SMP Wed Feb 13 13:27:54 JST 2008
MACHINE: x86_64 (1862 Mhz)
MEMORY: 1 GB
PANIC: "Oops: 0002 [1] SMP " (check log for details)
PID: 4122
COMMAND: "run"
TASK: 1003d9d5030 [THREAD_INFO: 10039496000]
CPU: 0
STATE: TASK_RUNNING (PANIC)
```

log(カーネルメッセージ)

```
crash> log
```

```
Unable to handle kernel NULL pointer dereference at 0000000000000000 RIP:  
<ffffffff801a4a0a> [proc_pid_shmcdump_write+173]  
PML4 3a91d067 PGD 39ba0067 PMD 0  
Oops: 0002 [1] SMP
```

- アクセス例外、nullpointerを使ったため、panicになっている

bt(バックトレース)

```
crash> bt
PID: 4122  TASK: 1003d9d5030      CPU: 0   COMMAND: "run"
#0 [10039497d00] start_disk_dump at ffffffff8014636d
#1 [10039497d30] try_crashdump at ffffffff8014b66d
#2 [10039497d40] do_page_fault at ffffffff8012393f
#3 [10039497d80] __up_write at ffffffff801e6d65
#4 [10039497dd0] handle_mm_fault at ffffffff8016a6f2
#5 [10039497e20] error_exit at ffffffff80110d91
[exception RIP: proc_pid_shmcdump_write+173]
RIP: ffffffff801a4a0a  RSP: 0000010039497ed8  RFLAGS: 00010246
RAX: 000001003961d000  RBX: 000001003debd0a8  RCX: 0000000000000000
RDX: 0000000000000000  RSI: 0000000000000000  RDI: 000001003961d002
RBP: 000001003debd080  R8: 0000000000000001  R9: 0000002a9557b3e0
R10: 0000000000000000  R11: 0000000000000246  R12: 0000000000000002
R13: 0000000000000000  R14: 0000002a95557000  R15: 0000010039497f50
ORIG_RAX: ffffffffffffffff  CS: 0010  SS: 0018
#6 [10039497ed0] proc_pid_shmcdump_write at ffffffff801a49f0
#7 [10039497f10] vfs_write at ffffffff80179d70
#8 [10039497f40] sys_write at ffffffff80179e58
#9 [10039497f80] system_call at ffffffff8011026a
```

dumpの動作ログ
(今回は関係なさそう)

問題の関数

ファイルシステム周り
から呼ばれています。

実行
順序

dis (ディスアセンブル)

```
crash> dis proc_pid_shmcdump_write+173  
0xffffffff801a4a0a <proc_pid_shmcdump_write+173>:  mov    %rax,0x0
```

gdb list (ソースのファイル確認)

```
crash> dis proc_pid_shmcdump_write+173  
0xffffffff801a4a0a <proc_pid_shmcdump_write+173>:    mov    %rax,0x0
```

```
crash> gdb list proc_pid_shmcdump_write  
106      in fs/shmcdump.c
```

事例2の原因

- 作成したCのソース中のバグ
- モジュールの中でメモリ確保されていない変数を使用していました。

shmcdump.cのソースコード(一部)

```
char **p = 0;  
...  
*p = ...
```

- 対処: ソースコード修正

まとめ

- kernel panicが発生したときにdumpを取れるようにしておけば、再発防止がしやすい

- dump取得のツールも進化しています



- kernel panicの原因はダンプとソースを当てることで解決できる

Linuxはオープンソース



AXcelerate Your Business

質疑応答/参考資料

- Miracle Linux Support サイト
 - <http://www.miraclelinux.com/support/>
- MiracleLinux Asianux製品マニュアル、ガイド
 - <http://www.miraclelinux.com/support/?q=node/101>
- みらくるblog カーネルダンプ
 - <http://blog.miraclelinux.com/uraura/cat324471/index.html>
- Alicia(Advanced Linux Crash-dump Interactive Analyzer)
 - <http://alicia.sourceforge.net/>
- intelリファレンスマニュアル
 - <http://www.intel.co.jp/jp/download/index.htm>
- google等検索エンジン

3/3 リニューアル!
(プレオープン)