真価を起動する
UNIADEX
Your IT Force

実践！Linuxカーネルパニック時の対応とダンプ解析
～「Alicia」のご紹介とデモ～

2006年3月

OSC2006

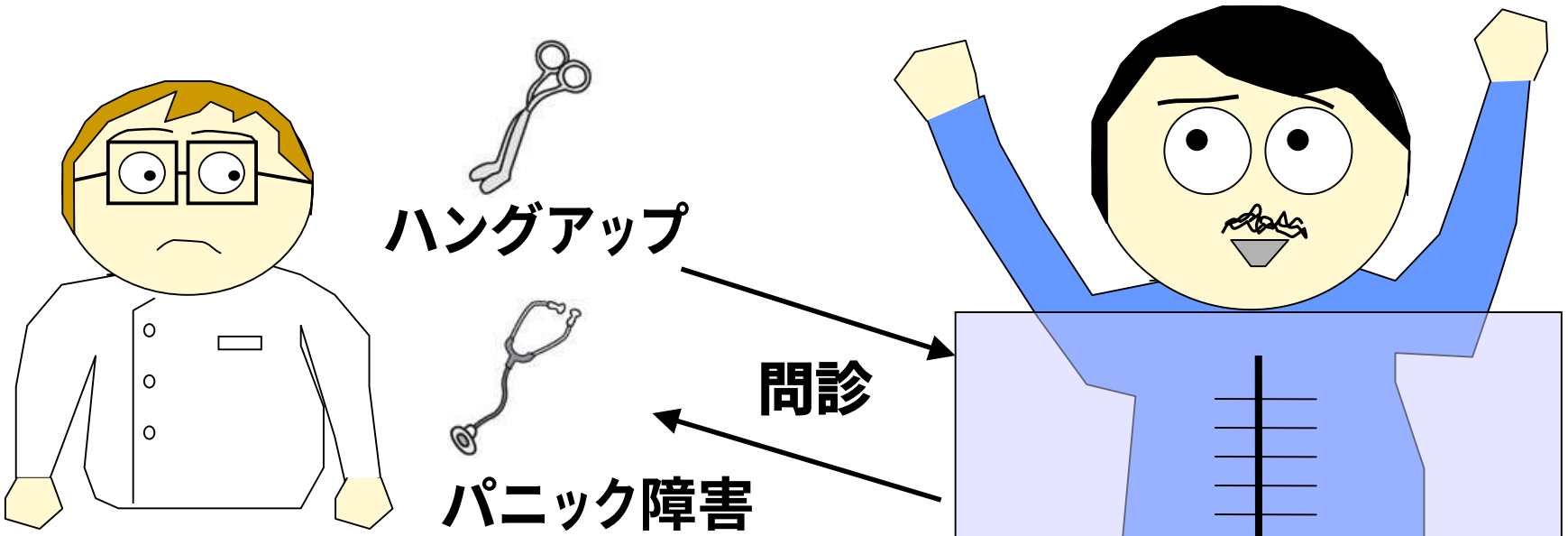
Aliciaとは?

その前に

カーネルクラッシュダンプとは？

- クラッシュ時のメモリの内容
- アプリケーションのコアダンプと大きくは違いがない
- 人間の世界ではレントゲンが近い

身近な例では



ダンプ解析とは？

- なぜ事件が起こったのか
- 事件に至るまでの背景の調査
- 聞き込みも時には必要
- 経験(解析パターン)重要
- 一時的な記憶喪失者との対話

1歳児がなぜ泣いてるかの解析パターンにも似ている。愛情が必要。

いきなり後ろから殴られた患者の方がよいか？

フライトレコーダーがあればもっとよいのだが。。

クラッシュダンプ解析手順

- ダイイングメッセージ(*log_buf)
- トレースバック(stack trace)
- アセンブルリスト or ソースコード
- レジスタ　＆　スタック
- /var/log/*
- bugzilla or LKML

どこで失踪？

現地調査。
「FIXME」なんて文字を見つけるかも。

それでもだめなら公開捜査？

ダンプ解析ツール: GDB (1/7)

- The GNU Source-Level Debugger
- バックトレース (backtrace)
- データの検査 (print)
- レジスタ (info register)
- メモリの検査 (x)
- その他(disas, ptype, list)
- マクロ (define 〜 end)

ダンプ解析ツール: GDB (2/7)

バックトレース (backtrace)

```
(gdb) bt
#0  0x08048374 in write_buf (tsp=0xbfeab330) at app.c:12
#1  0x0804839e in main_loop () at app.c:22
#2  0x080483e9 in main (argc=1, argv=0xbfeab3d4) at
    app.c:34
```

ダンプ解析ツール: GDB (3/7)

データの検査 (print)

```
(gdb) p *tsp
$2 = {flags = 17, p = 0x0, comm = 0x80484d4
    "this_is_command"}
(gdb) p *tsp->p
$3 = 0 '¥0'
(gdb) p write_buf
$4 = {char *(struct test_struct *)} 0x8048368 <write_buf>
```

ダンプ解析ツール: GDB (4/7)

レジスタ (info register)

```
(gdb) info reg
eax            0x0      0
esp            0xbfeab2fc       0xbfeab2fc
ebp            0xbfeab300       0xbfeab300
eip            0x8048374        0x8048374
(gdb) p/x $esp
$6 = 0xbfeab2fc
```

ダンプ解析ツール: GDB (5/7)

メモリの検査 (x)

```
(gdb) x/4 0xbfeab300
0xbfeab300:    0xbfeab318    0x0804839e    0xbfeab330
    0xffffffff
```

ダンプ解析ツール: GDB (6/7)

- その他(disas, ptype, list)

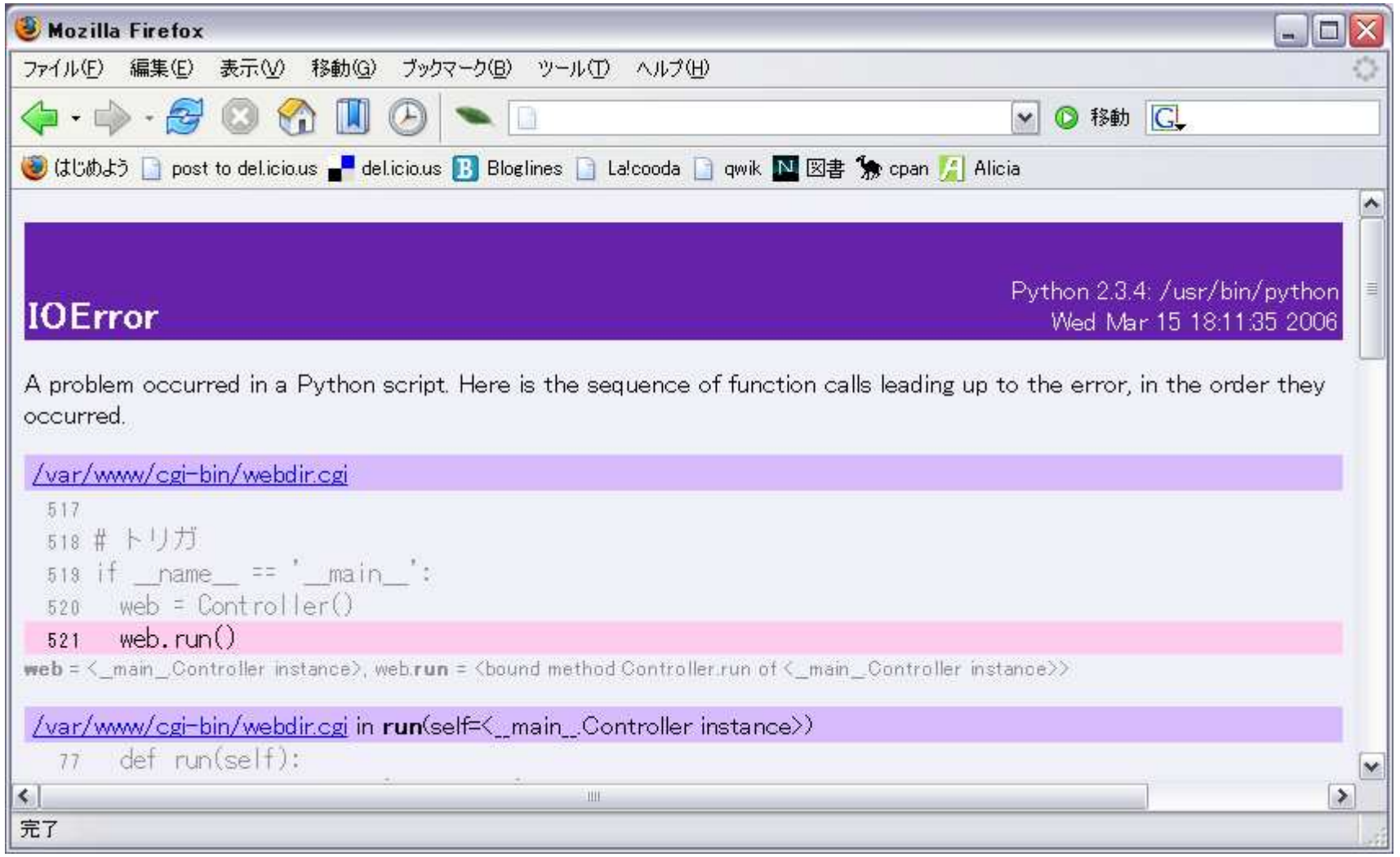
```
(gdb) disas write_buf
x08048368 <write_buf+0>:        push   %ebp
0x08048369 <write_buf+1>:       mov    %esp,%ebp
(gdb) ptype gtsp->comm
type = char *
(gdb) list write_buf
9       char *write_buf(struct test_struct *tsp)
10      {
```

ダンプ解析ツール: GDB (7/7)

マクロ (define 〜 end)

```
(gdb) define spdump
Type commands for definition of "spdump".
End with a line saying just "end".
>x/4 $sp
>end
(gdb) spdump
0xbfeab2fc:     0xbfeab380      0xbfeab318      0x0804839e
    0xbfeab330
```

最近のWebアプリケーションでは



DEMO-0 (1/2)

- GDBを実際に使ってみる
- 「クラッシュダンプ解析手順」に従って解析

  実行画面

  Stacktrace

  assemble list

  register & stack

DEMO-0 補足(2/2)

メモリの内容

色のついたところがスタックの一部。

0x80495ec	0xbfeab330	gtsp		
:				
0xbfeab2fc	0xbfeab380	現在のesp	sub $0x4,%esp	
0xbfeab300	0xbfeab318	現在のebp, 以前のebp	push %ebp	
0xbfeab304	0x0804839e	以前のesp, 戻り番地	call	
0xbfeab308	0xbfeab330	0x8(%ebp), 引数	tsp ポインタ	write_buf(gtsp)
:				
0xbfeab330	0x00000011	tsp	flags	
	0x00000000		p	
	0x080484d4		comm	

ダンプ解析ツール: lcrash

- LKCD付属の解析ユーティリティ
- LKCD(Linux Kernel Crash Dump)
- Linuxカーネルダンプに特化
- たくさんのコマンド群
- デザイン(仕様)が懲りすぎの面も
- sial(Cインタプリタ:Simple Image Access Language)

ダンプ解析ツール: crash

- Red Hat®製SVR4 UNIX crash+GDB
- Linuxカーネルダンプに特化
- GDBのコマンド使用可(ソースに含まれる)
- 動的ライブラリでコマンド拡張
- 最新のカーネルに素早く対応
- プロセス情報等のキャッシング

Aliciaとは?

Advanced

LInux

Crash-dump

Interactive

Analyzer

ネーミングの由来はid:yoshiokさんからセクシーな名前がよいとの助言を頂いたからである。

Alicia全体図

新Linux クラッシュダンプ解析ツール

ノウハウ蓄積:

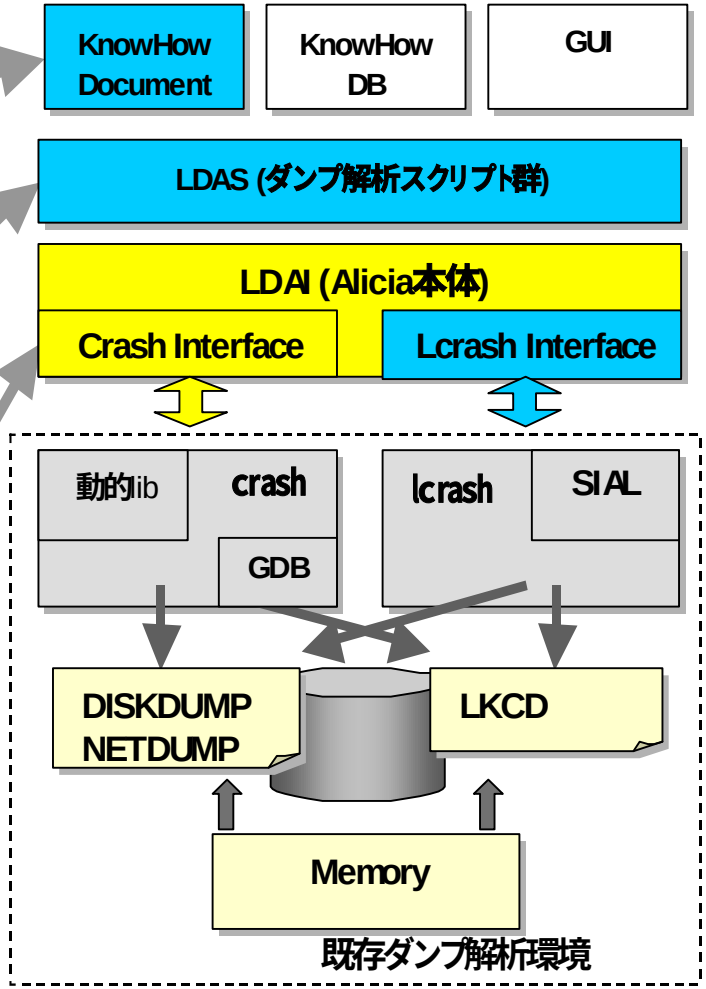
LDASはインタプリタであり、可視性がよく、そのままDump解析のノウハウとして蓄積可能

LDAS -- ダンプ解析スクリプト群 --
(Linux Dump Analysis Script)

Dump解析で必要とされた手順をスクリプト化することで同種のトラブルを迅速に解析

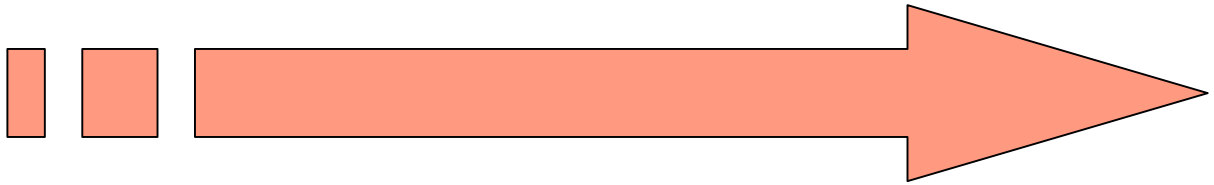
LDAI -- Alicia本体 --
(Linux Dump Analysis Interface)

既存Dump解析ツール(Crash、Lcrash)のコマンド群に対する入出力インターフェース、および既存Dump解析ツールを介してCrash Dumpへのアクセスを実現 (Alicia API)



ダンプ解析ツール: Alicia

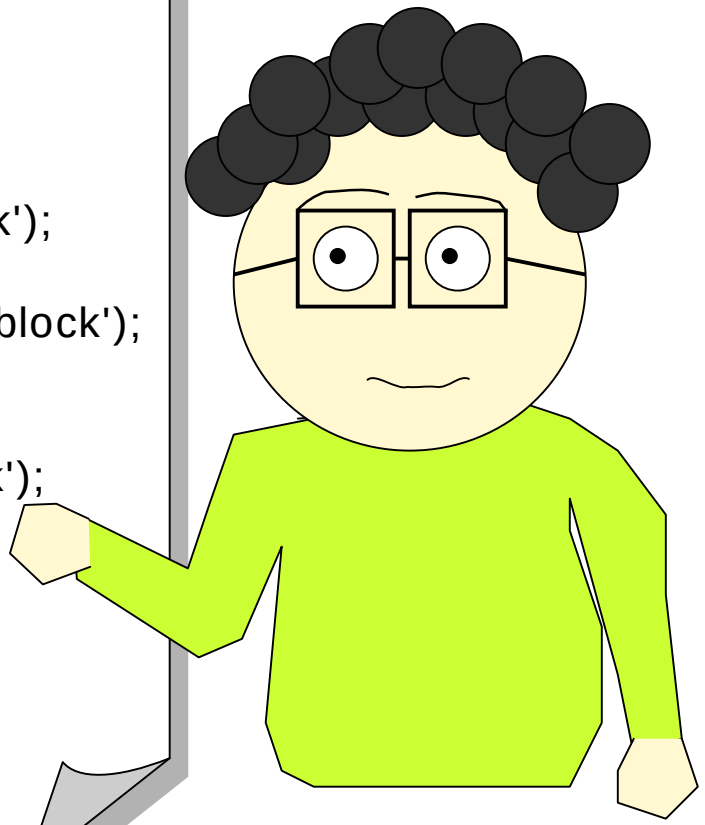
- Linux Kernel ダンプを解析する
- ダンプ解析環境を向上させる
- ダンプ解析のUI担当
- Perl言語による解析Plugin作成
- ダンプ解析のLibrary化を実現
- Debug手順のScript化(Perl)



もう少し分かりやすく：Good~~ク~~ハッキングをメモ

もう少し分かりやすく：Perlでメモ！

```
sub lock_status {
    use List;
    list_for_each(¥@lists, 'file_lock_list');
    foreach $list (@lists) {
        $fl = list_entry($list, 'struct file_lock','fl_link');
        _print_lock_status($fl, $i, "");
        my $fl_block =  _get_member_addr($fl,'fl_block');
        list_for_each(¥@blists, $fl_block);
        foreach $bl (@blists) {
            $bfl = list_entry($bl, 'file_lock', 'fl_block');
            _print_lock_status($bfl, $i, "->");
        }
        $i++;
    }
    return wantarray ? @out : join("¥n", @out);
}
```



なぜPerlなのか?

- LightWeight言語としての歴史が長い
- ユーザ層が広い
- TMTOWTDI(There's More Than One Way To Do It.) やり方は一つではない
- メインフレーマにとって扱いやすい
- ダンプ解析という泥臭い作業にも向いている
- 楽器でいうとGuitarである

初心者にも。老若男女。伴奏やオーケストラにも。ライトハンドでも歯でも。ネジっても壊しても。様々なスタイルで。豊富なエフェクターやアンプで。ジャンルを問わず。

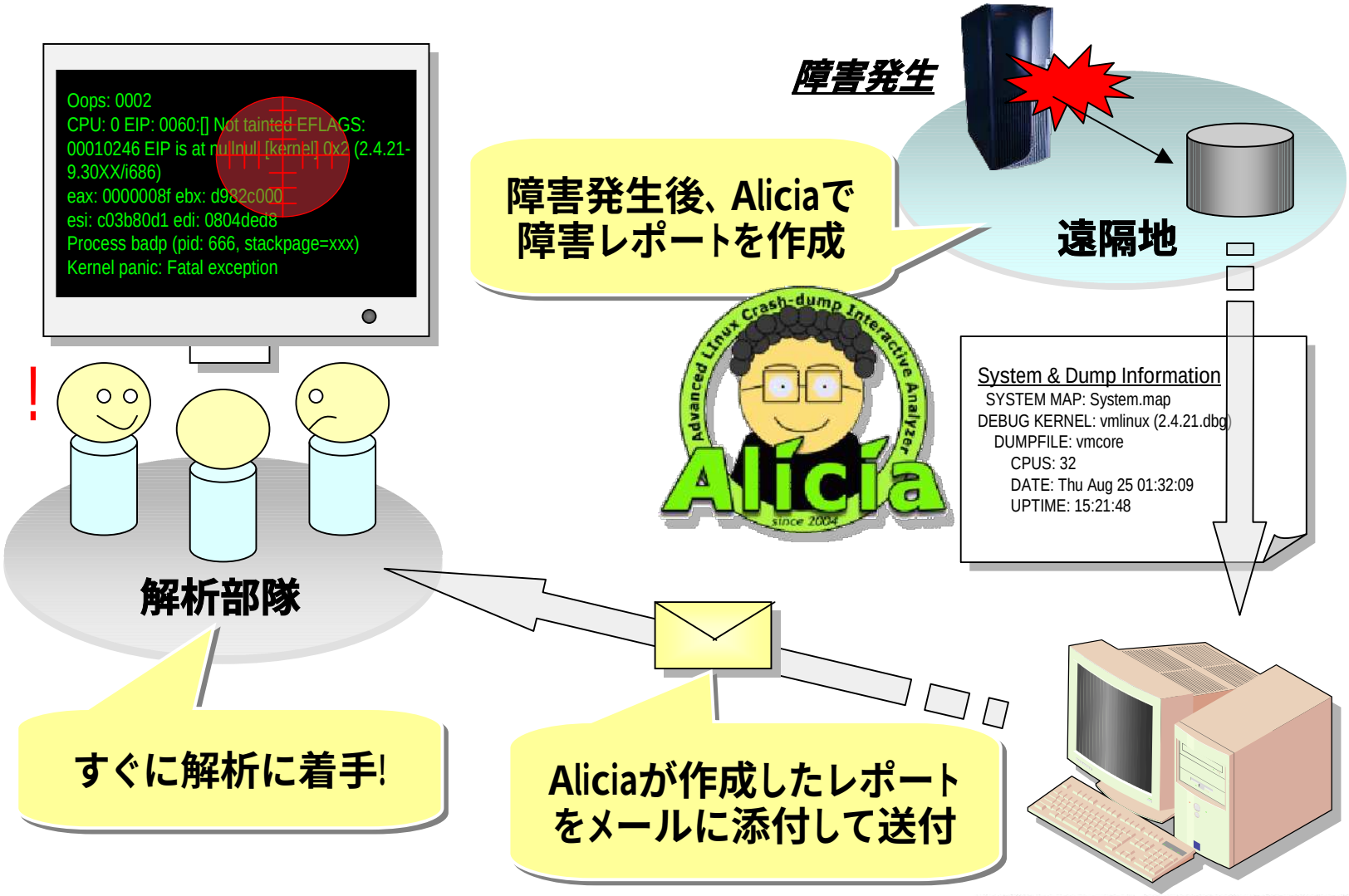
Alicia Internals

- CLインターフェースにTerm::ReadLine::Perl
- Engineとのやり取りはパイプ
- Pluginは単なるPerlモジュール
- 名前空間のぶつかり等の考慮がない
- API名がかっこ悪い(低位レイヤーです)

DEMO1

- crashのcommandの実行
- 操作感 (perlに包まれている感じ)
- Linux Dump Analysis Script (LDAS)作成
- 初期Dump解析command “report”

初期Dump解析Script



DEMO2 (1/6)

- 実際のクラッシュダンプを解析
- 「クラッシュダンプ解析手順」に従って解析

  log_buf

  Stacktrace

  assemble list

  register & stack

DEMO2 (2/6)

■ダイイングメッセージ抜粋

Unable to handle kernel **NULL pointer dereference** at virtual address **00000010**

EIP: 0060:[<**f8d4e2b9**>] Not tainted

EIP is at svc_register [sunrpc] 0x19

eax: **00000000** ebx: c6fadb00 esp: **d3259d74**

Process rpc.nfsd

DEMO2 (3/6)

■ スタックトレース

EIP is at svc_register [sunrpc] 0x1

[<f8d511f0>] svc_setup_socket [sunrpc] 0x2b0

[<f8d5131c>] svc_create_socket [sunrpc] 0xcc

[<f8d517bd>] svc_makesock_Rsmp_3c4d7d21 [sunrpc]

[<f8d8f1cd>] nfsd_svc [nfsd] 0x16d (0xd3259e6c)

[<f8d8fca3>] handle_sys_nfsservctl [nfsd] 0x253

DEMO2 (4/6)

■ アセンブルリスト

```
0xf8d4e2a0:    push   %ebp
0xf8d4e2a1:    xor    %ebp,%ebp
0xf8d4e2a3:    push   %edi
0xf8d4e2a4:    push   %esi
0xf8d4e2a5:    push   %ebx
0xf8d4e2a6:    sub    $0x38,%esp
0xf8d4e2a9:    mov    0x4c(%esp),%eax
0xf8d4e2ad:    testb  $0x2,0xf8d5ccc9
0xf8d4e2b4:    movzwl 0x54(%esp),%edi
0xf8d4e2b9:    mov    0x10(%eax),%esi
```

0x10積み上げ

引数1(構造体ポインタ)をeax(=0x0)

構造体メンバ(offset=0x10)にアクセス

DEMO2 (5/6)

スタック

0xd3259d40	c6fadb00	KERNEL-EFRAME	ebx	
	00000000		eax	
0xd3259d68	**f8d4e2b9**	svc_register+0x19	eip	(pt_regs.eip)
	00000060		xcs	
	00010246		elags	
0xd3259d74	00000048	svc_registerスタック	esp (pt_regs.esp)	0x48 0x00
	000001f0		xss	0x44
:				
	d3258000		task_struct	
	c6fadb00	ebx		0x10
	d4578c80	esi		0x0c
	d4cdc2b4	edi	push %edi	0x08 0x40
	00000000	ebp	push %ebp	0x04 0x44
0xd3259dbc	**f8d511f0**	svc_setup_socket 0x2b0	pt_regs.esp + *(esp)	0xd3259d74+0x48
	00000000	%ebp	*serv	0x4c
	00000006	0x33b(%esi)	proto	
	00000801	%eax	port=2049	

DEMO2 (6/6)

■ソースコード

```
struct svc_serv {
0x0   struct list_head  sv_threads, sv_sockets;
0x10  struct svc_program *    sv_program;
……}
svc_setup_socket(struct svc_serv *serv,……)
    *errp = svc_register(serv, ……)
svc_register(struct svc_serv *serv, ……)
    progp = serv->sv_program;
```

最新版Alicia-1.1.4では

- GDBのWrapper(coredump解析のみ)
- GDBのマクロをインラインに記述可能
- 構造体へのアクセッサモジュール導入

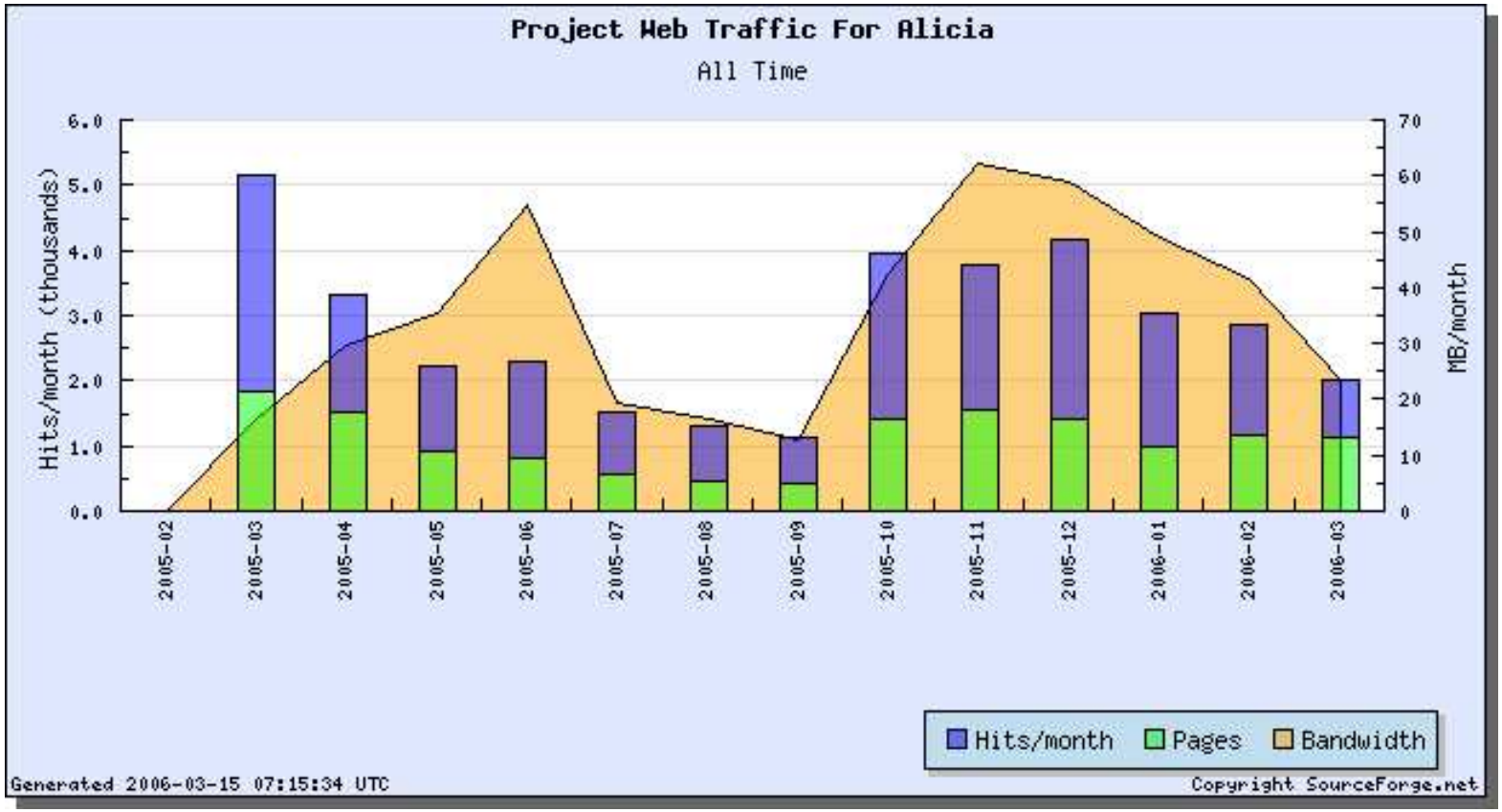
```
class ‘task_struct’, qw{ flags comm tasks}
$t = task_struct(dde8853c)
print $t->flags
print $t->dump
```

TODO

- LDASの充実
- Core EngineをGDB or Originalで実装
- 構造体・Symbol名のSuggest機能
- GUI
- Emacs
- LDASのDistribution siteの構築
- Alicia2.0

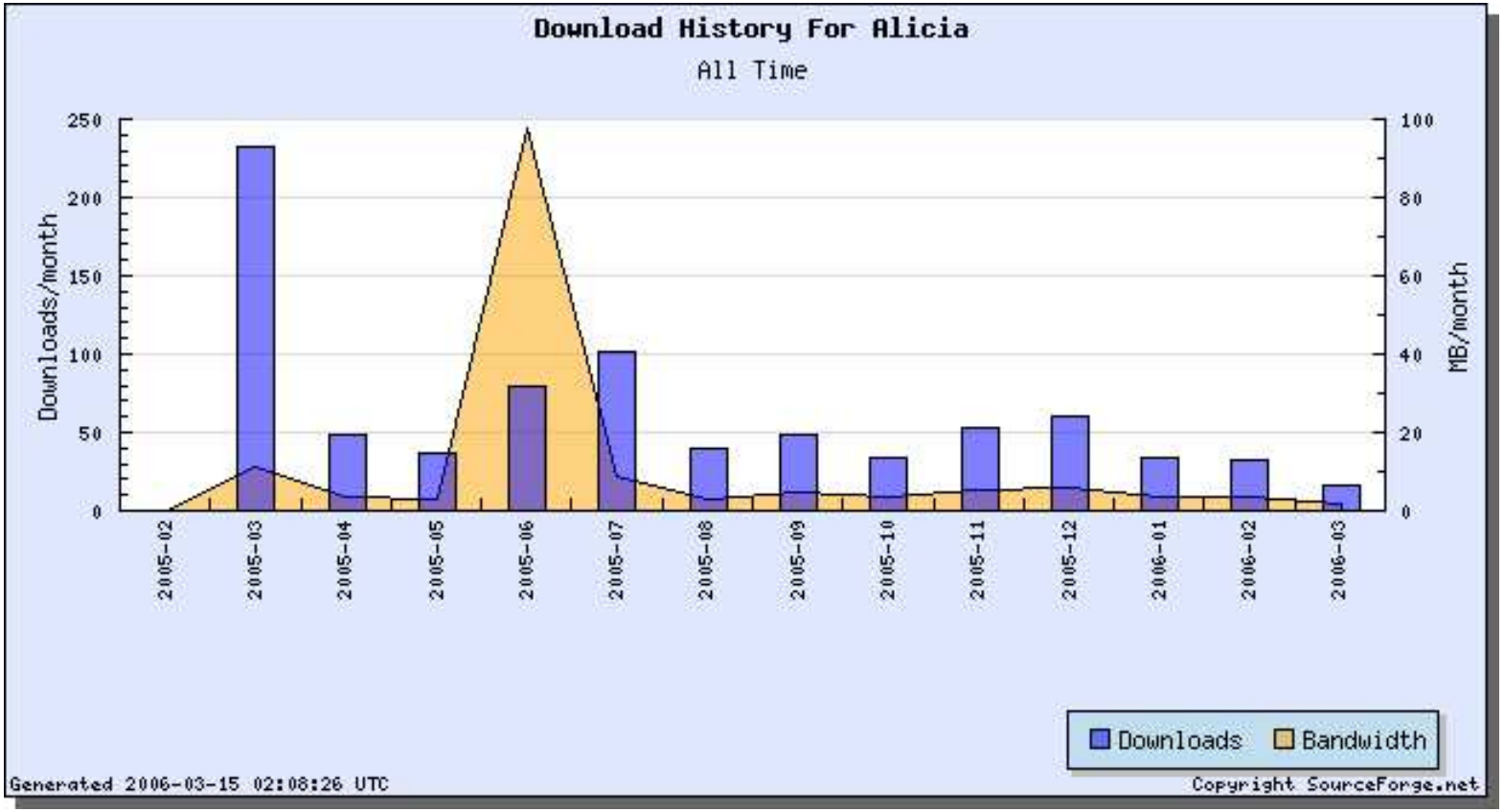
Access数：sourceforge.net (1/2)

- **Project Web Traffic: 36,674 (2006/03/14)**



Access数：sourceforge.net (2/2)

- **Download History: 計 816 (2006/03/14)**



URL

- Project at sourceforge.net
  http://sourceforge.net/projects/alicia/
- IPA
  http://www.ipa.go.jp/software/open/forum/
- ThinkIT
  http://www.thinkit.co.jp/free/tech/12/1/1.html
- Mailing List
  Alicia-users@lists.sourceforge.net
- Blog(予定)
  http://www.tyzoh.jp

Q & A

www.uniadex.co.jp

真価を起動する UNIADEX Your IT Force

ふろく