Alicia Version 1.0.0
使用解説書

UNIADEX, Ltd.

目次

1 はじめに

1.1 Aliciaとは

Aliciaは、既存のLinuxカーネルダンプ解析ツールであるCrash（将来はlcrash、GDBにも対応予定）をラッピングしたダンプ解析ツールであり、Perl Shellを実装する。これにより、Perl言語での解析スクリプト(LDAS)の作成と実行､Alicia標準関数によるメモリの参照､既存のCrashコマンドでの解析、などが実現される。

また、解析スクリプトを残し、解析者同士で共有することにより、解析時間の短縮や、カーネル教育用の資料としても利用できる。

1.2 このマニュアルの表記規則

このマニュアルは、以下の表記規則に従って記載されている。

alicia>

Aliciaのプロンプトを表す。これに続けて表記がある場合、それはAliciaに入力するコマンドを表す。

[shell]$
[shell]#

Shellのプロンプトを表す。$の場合は一般ユーザ、#の場合はスーパーユーザを表す。

[ ]

構文の説明での角括弧は、省略可能な部分を表す。

italic

斜体は構文の説明で、定型ではなく、ユーザ自身が指定する部分を表す。

また、メッセージの章の斜体は、定型ではなく状況により可変する部分を表す。

その他

・各コマンド、LDASの書式説明では引数を( )で囲む表記をしてあるが、例題などでは( )で囲む表記はしていない。Aliciaで実際にコマンドを入力するときは、( )や、サブルーチンを明示する&記号などは、Perlの規則に従って省略可能である。

2 動作環境

2.1 Aliciaの動作推奨環境

Aliciaは、以下の環境で動作を確認、推奨している。他の環境での動作は保証しない。

Crashレベル	3.8-5
Perlレベル	5.8.0
ディストリビューション	MIRACLE LINUX V3.0 - Asianux Inside

2.2 解析可能なダンプ形式

Aliciaでは、Crashで解析可能なダンプを参照することが出来る。参考資料として、Aliciaの検証で使用したダンプを採取したシステムを以下に示す。

CPUタイプ	IA32
CPU数	2～32
メモリ容量	4GB～16GB

カーネルレベル	2.4.21 (SMP)
ディストリビューション	MIRACLE LINUX V3.0 - Asianux Inside
ダンプフォーマット	LKCDによって採取されたダンプ

3 Aliciaのインストールとアンインストール

3.1 Aliciaに必要なソフトウェア

Aliciaを動作させる為には、PerlのTerm::ReadKeyモジュールと、Term::ReadLine::Perlモジュール、そして既存のダンプ解析ツールCrashが必要となる。Aliciaをインストールするマシンに、前述のソフトウェアがインストールされていない場合、Alicia起動前に各ソフトウェアをインストールしておく事。

3.2 インストール

以下のAlicia配布用ファイルを適当なディレクトリにダウンロードして、インストールを実行する。

Alicia-*version*.tar.gz
※ *version* : Alicia.pmのバージョン

以下の手順で、Aliciaがインストールされる。インストールするディレクトリを変更したい場合は、「3.1.1 インストール先ディレクトリの変更」を参照のこと。

```
[shell]$ tar -xzvf Alicia-version.tar.gz      配布ファイルの解凍
[shell]$ cd Alicia-version                    解凍先のディレクトリに移動
[shell]$ perl Makefile.PL                         Makefileの作成
rootユーザに変更
[shell]# make
[shell]# make install
```

インストール先のディレクトリは、システムに依存する。以下のコマンドで、インストールに使用されるディレクトリを確認できる。

```
[shell]# perl '-V:install.*'
```

上記コマンドで表示される変数のうち、Aliciaはインストール時に以下の3つの変数に指定されているディレクトリを使用する。

① installsitebin - Alicia実行ファイル(alicia)がインストールされるディレクトリ
② installsitelib - Alicia.pm CRASH.pm WPSH.pm およびAliciaディレクトリが
インストールされるディレクトリ

③ installsiteman3 - Aliciaモジュール用manファイル（Alicia.3pm）
CRASHモジュール用manファイル(CRASH.3pm)
WPSHモジュール用manファイル(WPSH.3pm)
がインストールされるディレクトリ

LDASスクリプトとAlicia設定ファイルは、デフォルトでは以下のディレクトリにインストールされる。

/usr/share/ldas	LDASスクリプト（複数）
/etc	Alicia設定ファイル(alicia.conf)

インストールされたファイルは、.packlistファイルに記録される。
（例）
/usr/lib/perl5/site_perl/5.8.0/i386-linux-thread-multi/auto/Alicia/.packlist
※ /usr/lib/perl5/site_perl/5.8/0/i386-linux-thread-multi は、perlのバージョン、システムによって異なる。

3.2.1 インストール先ディレクトリの変更

(1) インストールディレクトリの変更

Makefile作成時にprefixを指定することにより、インストールディレクトリを変更する事ができる。

（例） /home/alicia以下に、Alicia実行ファイル、モジュール、manファイルを作成する。

```
[shell]$ perl MakefIle.PL prefix =/home/alicia
```

ただし、LDASスクリプトとAlicia設定ファイルalicia.confのインストール先は、この操作では変更できない。以下、(2)、(3)を参照のこと。

(2) LDASスクリプト

配布ファイル内の設定ファイルalicia.confに記述されている「ldasDir」を、ユーザ任意のディレクトリに書き換える。

変更前

```
ldasDir = /usr/share/ldas;
```

変更例

```
ldasDir = /etc/local/ldas;
```

(3) Alicia設定ファイルのインストール先の変更

Makefile.PL 5行目の$CONFDIRを、ユーザ任意のディレクトリに書き換える。

変更前

```
$CONFDIR="/etc";
```

変更例

```
$CONFDIR="/usr";
```

3.3 アンインストール

アンインストールするスクリプトは提供していないが、以下のAlicia関連のファイルを削除することができる。

① .packlistファイルに記述されているファイル

② コンフィグファイル（デフォルトは/etc/alicia.conf）

③ LDASディレクトリ（デフォルトは/usr/share/ldas）

④ .packlistファイル

```
[shell]# RMF='cat /usr/lib/perl5/site_perl/5.8.0/i386-linux-multi/auto/Alicia/.packlist'
[shell]# for i in $RMF
> do
>   rm -rf $i
> done
[shell]# rm -rf /usr/share/ldas
[shell]# rm -rf /etc/alicia.conf
[shell]# rm -rf /usr/lib/perl5/site_perl/5.8.0/i386-linux-multi/auto/Alicia/.packlist
```

4 Aliciaの起動～終了

4.1 起動

起動プログラム'alicia'を引数付きで実行する。

[shell]$ alicia *mode* [*modeに従った引数*]

mode	-crash	AliciaをCrashモードで起動する。
	-lcrash	Version1.0.0では未対応。
	-gdb	Version1.0.0では未対応。
	-v	Aliciaのバージョン情報を表示する。
modeに従った引数		各モード（Crash、lcrash、gdb）を起動する際の引数

（例） AliciaをCrashモードで起動する場合

```
[shell]$ alicia -crash -s /var/log/dump/map.0 /var/log/dump/debug /var/log/dump/dump.0
```

Alicia起動後、Please wait for opening dump ...のメッセージが出力され、しばらくするとAliciaは自身のプロンプトを出力し、ユーザからの入力を待つ。

4.2 基本的な操作

4.2.1 コマンドの入力

コマンドは、プロンプト'alicia>'に続けて入力する。Aliciaは、Perl Shellを実装している為、Perlの関数、四則演算などをそのまま使用する事が可能である。逆に、入力の末尾にセミコロンが必要など、基本的にPerlの構文に従わなければならないので注意すること。

4.2.2 複数行の入力（コメントによる方法）

Aliciaでは、複数行の入力をサポートする。その方法のひとつとして、コメントを利用する方法がある。以下に具体例を示す。

```
alicia> print 1;#        末尾のセミコロンの後にコメント'#'を付ける
> print 2;#              継続行のプロンプトは'>'のみとなる
> print 3;               最終行にはコメント'#'を付けない
123                      入力した3行が実行される
```

4.2.3 複数行の入力（ヒアドキュメント形式）

もうひとつの方法として、Perlのヒアドキュメントのように複数行を入力する方法がある。以下に具体例を示す。

```
alicia> <<EOF          <<に続けて、任意のキーワードを指定
> print 1;             1行目を入力
> print 2;             2行目を入力
> print 3;             3行目を入力
> EOF                  キーワードを入力
123                    入力した3行が実行される
```

4.3 コマンドライン編集機能

GNU Readlineライブラリと同じインタフェースを実装しており、ヒストリ機能、コマンド名補完機能などを提供する。この機能の提供にはCPANに登録されているTerm::ReadLine::Perlモジュールを利用している。ここでは特に便利な機能を掲載する。

4.3.1 ヒストリ

カーソルキーの↑↓、及びCTRL+p、CTRL+nで、ユーザが入力したコマンドのヒストリをたどることが出来る。ヒストリとして保存されるコマンドの数はユーザ毎に最新200個で、alicia.conf（各ユーザごとの設定ファイル）によって決定される。

4.3.2 コマンド名補完

TABキーにより、コマンド名、及びファイル名の補完機能が利用できる。

4.3.3 リバース検索

CTRL+rにより、コマンドヒストリに対して、リバース検索をかけることができる。

4.3.4 その他のコマンドライン編集

CTRL+aでカーソルを先頭に移動できる。

CTRL+eでカーソルを最後尾に移動できる。

CTRL+wで、現在の入力イメージを切り取る。

CTRL+yで、CTRL+wにより切り取ったイメージを貼り付ける。

4.4 終了

exit関数にて終了する（「5.1.9 exit関数」を参照のこと）。もしくは、Crashシームレスコマンドのquit（省略形 q）コマンドでも終了できる。

5 コマンド

5.1 標準関数

Aliciaに実装されている関数について説明する。

5.1.1 pass_through関数

■ 機能

Crashコマンドを実行する。

■ 書式

pass_through('*crash-command*')

■ 引数

crash-command	オプションや引数を全て含むCrashのコマンド。 ただし、eval、exit、foreach、repeatコマンドは実行できない。

■ 戻り値

Crashコマンドの実行結果をそのまま文字列で返す。
Crashコマンドがエラーだった場合、エラーメッセージを文字列で返す（pass_through自体はコマンドエラー等にはならない事に注意する）。

■ 例

```
alicia> print pass_through 'rd f46ec000 8';
f46ec000:  00000000 00000100 00000000 c0000000   ................
f46ec010:  c0401d20 00000001 00000000 ffffffff    .@............
```

5.1.2 kernel関数

■ 機能

メモリの内容を、アドレス、構造体名、メンバ名を使用して参照する。

■ 書式

kernel('*address*', '*structure*', '*member*[.*member*]'[, '*type*'])

■ 引数

address	構造体の先頭アドレス。 プレフィックスに'0x'は付けても付けなくても良い。
structure	カーネル構造体名。
member[.*member*]	メンバ名。
type	メンバのC言語における型。 省略可能であり、指定した場合は指定した型にキャストする。

■ 戻り値

メンバのメモリ上の内容を返す。
エラーの場合はundefを返す。

■ 例

```
alicia> print kernel 'f46ec000','task_struct','comm';
"bash\000\000ty\000\000\000\000\000\000\000"

alicia> print kernel 'f46ec000','task_struct','comm','char *';
bash

alicia> print kernel 'f46ec000','task_struct','tasks';
{
  next = 0xf4596060,
  prev = 0xf47da060
}

alicia> print kernel 'f46ec000','task_struct','tasks.next';
0xf4596060
```

5.1.3 get_mem関数

■ 機能

指定したアドレスの内容を参照する。

■ 書式

get_mem('*address*'[, *number*])

■ 引数

address　　メモリ上のアドレス。
　　プレフィックスに'0x'は付けても付けなくても良い。
number　　参照する個数。
　　省略可能で、省略した場合は1になる。

■ 戻り値

メモリ上のアドレスの内容を返す。*number*で2以上を指定した場合、配列で内容を返す。
エラーの場合はundefを返す。

■ 例

```
alicia> print get_mem 'f46ec000';
0x00000000

alicia> @var = get_mem 'f46ec000',4;
alicia> foreach (@var) { print "$_\n" };
0x00000000
0x00000100
0x00000000
0xc0000000
```

5.1.4 get_addr関数

■ 機能

指定したカーネル変数のアドレスを参照する。

■ 書式

get_addr('*kernel_variable*')

■ 引数

kernel_variable　　カーネル変数名。

■ 戻り値

指定したカーネル変数が格納されるメモリ上のアドレスを返す。

■ 例

```
alicia> print get_addr 'jiffies';
```

0xc0507580

5.1.5 get_value関数

■ 機能

指定したカーネル変数の内容を参照する。

■ 書式

get_value('*kernel_variable*')

■ 引数

kernel_variable　　カーネル変数名。

■ 戻り値

指定したカーネル変数の内容を返す。
エラーの場合、undefを返す。

■ 例

```
alicia> print get_value 'jiffies';
746556

alicia> print get_value 'banner';
"<6>NET4: Unix domain sockets 1.0/SMP for Linux NET4.0.\n"

alicia> print get_value 'sysrq_ctls';
{
  enabled = 1,
  keycode = 84,
  sticky = 0,
  timer = 10
}
```

5.1.6 get_increment関数

■ 機能

開いているダンプのワードサイズ（バイト）を得る。

■ 書式

get_increment()

■ 引数
なし

■ 戻り値
1ワードのバイト数。
※ハードウェアに依存する。IA32なら4、IA64なら8が返る。

■ 例
```
alicia> print get_increment;
4

alicia> print ((get_increment) * 4);
16
```

5.1.7 less関数

■ 機能
引数をlessページャで参照する。

■ 書式
less('*argument*')

■ 引数
argument　lessページャで参照したい文字列。
引数に指定できるのは任意の文字列などで、
変数や、結果を返す関数でも良い。

■ 戻り値
なし

■ 例
```
alicia> $task = pass_through 'struct task_struct';
alicia> less $task;

alicia> less pass_through 'bt';
```

5.1.8 more関数

■ 機能

引数をmoreページャで参照する。

■ 書式

more('*argument*')

■ 引数

argument　moreページャで参照したい文字列。
引数に指定できるのは任意の文字列などで、
変数や、結果を返す関数でも良い。

■ 戻り値

なし

■ 例

```
alicia> $task = pass_through 'struct task_struct';
alicia> more $task;

alicia> more pass_through 'bt';
```

5.1.9 exit関数

■ 機能

Aliciaを終了する。

■ 書式

exit

■ 引数

なし

■ 戻り値

なし

■ 例

```
alicia> exit;
```

5.1.10 load関数

■ 機能

LDASをファイルからローディングして、Aliciaで使用可能にする。

■ 書式

load('*filename*')

■ 引数

filename　/usr/share/ldas（LDAS保存ディレクトリ）、またはカレントディレクトリに保存したLDASファイルのファイル名。
※フルパス、または相対パスでLDASファイルを指定すれば、上記以外のディレクトリでもLDASを読み込み可

■ 戻り値

読み込みに成功すれば1、失敗すればundefが返る。

■ 例

```
alicia> load 'example.ldas';
alicia> load '/home/uniadex/example.ldas';
```

5.1.11 help関数

■ 機能

Alicia、及びCrashのヘルプをlessページャで参照する。

■ 書式

help [*command*]

■ 引数

command　Alicia、及びCrashのコマンド。
commandを「crash」とすると、crashのヘルプ（コマンド未指定）が表示される。

■ 戻り値

なし

■ 例

```
alicia> help;

    USAGE:
      $program [-v¦--version]
      $program [-crash] arguments for crash

      crash arguments:
      [-h [opt]][-v][-s][-i file][-d num] [-S] [mapfile] [namelist] [dumpfile]

    DESCRIPTION :
        ALICIA is Dump analysis tool of Linux running on Perl environment.
             :
             :

alicia> help get_mem;

  get_mem function
     This function returns the value of the address specified by the argument.
     If two or more numerical values are specified by the 2nd argument,
     the continuous value will be returned to arrangement.
     When the 2nd argument is omitted, 1 is specified tacitly.

     Form:
     get_mem ("address"[,number]);
             :
             :

alicia> help crash;

*             files         mod           runq          union
alias         foreach       mount         search        vm
ascii         fuser         net           set           vtop
```

```
bt              gdb             p               sig             waitq
                :
                :

alicia> help bt;

NAME
  bt - backtrace

SYNOPSIS
  bt [-a¦-r¦-t¦-l¦-e¦-E¦-f] [-R ref] [ -I ip ] [-S sp] [pid ¦ taskp]

DESCRIPTION
  Display a kernel stack backtrace.  If no arguments are given, the stack
  trace of the current context will be displayed.
                :
                :
```

5.2 Crashコマンド

Aliciaでは、Crashのコマンドをシームレスに実行できる。実行するには、Aliciaのプロンプトに続けて、Crashのコマンドをオプションや引数を含めて、そのまま入力すれば良い。

ただし、入力したコマンドの末尾に';'セミコロンを付けなくてはならない、などの留意点があり、以下に示す。

※ Crashコマンドをシームレスに実行する際の留意点

- 末尾にセミコロンが必要
- サブルーチン、及びLDASに含めることは出来ない
  （サブルーチン、LDAS内でCrashコマンドを実行する場合はpass_through関数で代用可）
- Perlの構文と違い、1行に2個以上のコマンドを入力できない
- パイプで出力結果を渡せるのは、lessとmoreのみ
- Crashの、引数なしのhelpコマンドは、シームレスに実行できない
  （引数なしでhelpを実行すると、Aliciaのヘルプになる。
  Crashのhelpコマンドは「5.1.11 help関数」を参照の事）
- Crashコマンドの、*、eval、foreach、repeatは実行できない
- Crashコマンドのstructにて、「struct」を省略することはできない

例：

```
alicia> dis schedule 5;
0xc0129820 <schedule>:  push   %ebp
0xc0129821 <schedule+1>:        mov    %esp,%ebp
0xc0129823 <schedule+3>:        push   %edi
0xc0129824 <schedule+4>:        push   %esi
0xc0129825 <schedule+5>:        push   %ebx

alicia> struct task_struct | less;
                :
                :
```

task_structの内容がlessページャで開かれる

5.3 Shellコマンド

Aliciaでは、コマンドの1文字目に'!'（エクスクラメーション）を入力すると、Shellコマンドとして実行する。この場合は、末尾のセミコロンは不要。ただし、サブルーチン、及びLDASに含めることは出来ない。

例：

```
alicia> !ps ax
  PID TTY      STAT   TIME COMMAND
    1 ?        S      0:32 init
    2 ?        SW     0:00 [migration/0]
    3 ?        SW     0:00 [migration/1]
    4 ?        SW     0:00 [migration/2]
    5 ?        SW     0:00 [migration/3]
    6 ?        SW     3:30 [keventd]
    7 ?        SWN    0:00 [ksoftirqd/0]
    8 ?        SWN    0:00 [ksoftirqd/1]
    9 ?        SWN    0:00 [ksoftirqd/2]
   10 ?        SWN    0:00 [ksoftirqd/3]
                :
                :
```

6 LDAS

LDASとは、Perlのサブルーチンであり、ダンプの解析手順をスクリプト（サブルーチン）化して保存・実行が可能となる。

6.1 LDASのロード

LDASはload関数により、ファイルから読み込んでAliciaで使用可能になる。load関数に関しては、「5.1.10 load関数」を参照のこと。

6.2 LDASの実行

通常のPerlサブルーチンと同様に、サブルーチン名（LDAS名）を指定する。LDASにより、必要ならば引数を指定する。

6.3 LDASの作成

事前に通常のファイルにサブルーチンを書いておくことでLDASを作成できる。

以下に簡単なLDASの例を示す。

例：example.ldas

```
sub example {
    my $args = shift;
    print get_mem("$args");
};
1;
```

ファイル名とサブルーチン名は必ずしも同一でなくて良い。また、ファイルの拡張子も特に問わない。ファイルの末尾に1;は必要なので注意すること（LDASの書式は、Perlの通常のサブルーチンに従わなければならない）。

尚、例の場合はひとつのファイルにひとつのサブルーチンが書かれているが、ひとつのファイル内に複数のサブルーチンを書くことも可能である。

このサンプルを実行するには、以下の様にload関数でLDASを読み込み、実行すれば良い。

```
alicia> load 'example.ldas';

alicia> example 'f46ec000';
0x00000000
```

6.4 Shellエスケープを利用して任意のエディタでLDASの作成・修正を行う

エクスクラメーションによるShellコマンドの実行（「5.3 Shellコマンド」参照）で、Alicia起動中にLDASを作成・修正する事も可能。以下の例はviエディタを使用してLDASの作成・修正を行う場合。

```
alicia> !vi example.ldas
```

起動したエディタを終了すると、制御はAliciaに戻る。
尚、修正したLDASは、load関数にて再読み込みしなければ、修正内容は反映されない。

7 留意事項・制限事項

7.1 留意事項

7.1.1 make clean実行時にMakefile.oldが残る

make cleanを実行すると、Makefile.oldが残る。これはPerlのmake cleanの仕様である。

7.1.2 TAB補完の挙動

ファイル名補完時、以下の現象が発生する。

・シンボリックリンクに'@'（アットマーク）が付く

・ディレクトリ名に' '（スペース）が付く

・実行可能ファイルに'*'（アスタリスク）が付く

7.1.3 表示結果の自動改行

Aliciaのプロンプトにて、Perlのprint文など何らかの表示を行った場合、表示内容の末尾に改行コードが付いていなくても、次のプロンプト'alicia >'は改行されて出力される。

7.2 制限事項

7.2.1 読み込んだLDASのTAB補完

load関数で読み込んだLDASに対する、コマンド名補完機能は効かない。

7.2.2 Crash拡張コマンド

Crashのextendコマンドによる拡張コマンドは、登録、削除は可能だが実行する場合、シームレスに実行できない。extendコマンドにより登録された拡張コマンドを実行するには、pass_through関数を使用する必要がある。

また、extendコマンドにより登録された拡張コマンドに、コマンド名補完機能は効かない。

7.2.3 特殊変数の使用

Aliciaのコマンドライン上で特殊変数を使用しても、内容は保証されない。

7.2.4 パイプとリダイレクト

Aliciaで、標準出力への出力結果をパイプやリダイレクトで標準出力以外へ出力することは出来ない。

ただし、Crashシームレスコマンドのみ出力結果をless、及びmoreにパイプできる（「5.2 Crashコマンド」を参照のこと）。

8 メッセージ

A pipe cannot be carried out to '*xxxxx*'.
Crashシームレスコマンドで、出力結果をパイプにて*xxxxx*には渡せない。

Can't create your alicia directory: *direcroty* : *reason*
何らかの理由*reason*により、ユーザのAlicia用ディレクトリ*directory*が作成できない。

Can't copy *systemconfig* to *userconfig* : *reason*
何らかの理由*reason*により、Aliciaのシステムデフォルトコンフィグファイル*systemconfig*を、ユーザのコンフィグファイル*userconfig*にコピーできない。

Can't dup status = *errorstatus* : *reason*
何らかの理由*reason*により、子プロセスの標準出力を親の入出力用パイプに割り当てることができない。

Can't fork status = *errorstatus* : *reason*
何らかの理由*reason*により、子プロセスを生成できない。

Can't open config file: *configfile* : *reason*
何らかの理由*reason*により、コンフィグファイル*configfile*が開けない。

Child does NOT receive the TERM signal!!
子プロセスがTERMシグナルを受けた。

crash command not found.
You must install crash
既存ダンプ解析ツール"Crash"がインストールされていない。

CRASH::get_addr : No symbol "*variable*" in current context at *subroutine* line *n*.
get_addr関数で指定された変数*variable*は存在しない。

CRASH::get_addr : The variable name is not specified at *subroutine* line *n*.
get_addr関数で、引数の指定が無い。あるいは、指定した引数が空白文字のみだった。

CRASH::get_mem : Invalid address: *address* at *subroutine* line *n*.

get_mem関数に引数として指定したアドレス*address*が不正である。

CRASH::get_mem : Invalid kernel ¦ user virtual address: *address* at *subroutine* line *n*.
get_mem関数で指定されたアドレス*address*は、参照出来ない。

CRASH::get_mem : Invalid number: *number* at *subroutine* line *n*.
get_mem関数の第2引数である、参照する個数*number*の指定が不正である。

CRASH::get_mem : The address is not specified at *subroutine* line *n*.
get_mem関数に引数が指定されていない。あるいは指定した引数が、空白文字のみだった。

CRASH::get_value : No symbol "*variable*" in current context at *subroutine* line *n*.
get_value関数で指定された変数*variable*は存在しない。

CRASH::get_value : The variable name is not specified at *subroutine* line *n*.
get_value関数で、引数の指定が無い。あるいは、指定した引数が空白文字のみだった。

CRASH::kernel : Cannot access memory at address *address* at *subroutine* line *n*.
kernel関数で引数として指定したアドレス*address*にはアクセス出来ない。

CRASH::kernel : Invalid address: *address* at *subroutine* line *n*.
kernel関数で引数として指定したアドレス*address*が不正である。

CRASH::kernel : No struct type named *structname* at *subroutine* line *n*.
kernel関数で引数として指定した構造体名*structname*が不正である。

CRASH::kernel : There is no member named *member* at *subroutine* line *n*.
kernel関数で引数として指定したメンバ名*member*が不正である。

CRASH::kernel : Undefined argument at *subroutine* line *n*.
kernel関数で、引数としてアドレス、構造体名、またはメンバ名の指定が無い。

CRASH::pass_through : Crash command '*command*' is not supported at *subroutine* line *n*.
pass_through関数に引数として指定したコマンド*command*は、Aliciaではサポートしていない。

CRASH::pass_through : The argument error of the help command at *subroutine* line *n*.

pass_through関数でhelpコマンドの引数に'ENDOFOUTPUTSTREAMFROMCRASH'を与えることは出来ない。

CRASH::pass_through : There is no argument of a crash function at *subroutine* line *n*.

pass_through関数に引数が指定されていない。あるいは指定した引数が、空白文字のみだった。

history file cannot open for load

ヒストリファイルを読み込み用に開く事が出来ない。

history file cannot open for save

ヒストリファイルを保存用に開く事が出来ない。

Invalid tag name in *file*

コンフィグファイル*file*に不正なタグ名が含まれている。

***sysconfig* is not found. Please re-install alicia...**

システムデフォルトAliciaコンフィグファイル*sysconfig*がない。もう一度Aliciaをインストールする必要がある。

load: Can't load ldas file "*file*": No such file

load関数で指定された*file*は、存在しない。

load: "*file*": not a ldas format

load関数で指定された*file*は、LDASファイルでは無い。

load: The file name was not specified.

load関数にて、loadするLDASファイル名の指定が無い。

Only crash tool can be supported

Alicia起動時に'crash'以外のモードが指定された。

9 Aliciaで作成されるファイル

9.1 コンフィグファイル

各ユーザごとのコンフィグが記録され、$HOME/.alicia/alicia.conf に保存される。

9.2 コマンドヒストリファイル

各ユーザごとのコマンドヒストリが記録され、$HOME/.alicia/alicia_history に保存される。

付録1 LDASのサンプル

Aliciaには、LDASのサンプルが標準で付属する。

付録1.1 list_h

■ 機能

環状のlist_headリンクをたどりリンク上のすべての構造体のアドレス配列を返す。

■ 書式

list_h('*address*', '*structure*', '*member*')

■ 引数

address	構造体のアドレス。
structure	構造体名。
member	メンバ名。

■ 戻り値

リンクに参加している構造体のアドレスを配列で返す。
返される配列の先頭は構造体の個数となる。

■ 例

```
alicia> @allprocess = list_h '0xe9eb2000', 'task_struct', 'tasks';
alicia> foreach (@allprocess) { print "$_\n" };
2134
0xf10b2000
0xf1c5e000
0xeac1a000
0xf0f00000
0xf36ae000
0xed878000
0xf349c000
0xf3f04000
0xc6010000
        :
        :
```

付録1.2 list_c

■ 機能

親の構造体の配下の子の構造体のlist_headリンクをたどりリンク上のすべての構造体のアドレス配列を返す。

■ 書式

list_c('*address*', '*owner_structure*', '*owner_member*',
'*child_structure*', '*child_member*')

■ 引数

address	親構造体のアドレス。
owner_structure	親構造体名。
owner_member	親メンバ名。
child_structure	子構造体名。
child_member	子メンバ名。

■ 戻り値

リンクに参加している構造体のアドレスが配列で返される。
返される配列の先頭は構造体の個数となる。

■ 例

```
alicia> @list = list_c '0xf45fe180','inode','i_dirty_data_buffers',
'buffer_head','b_inode_buffers';
alicia> foreach (@list) { print "$_\n" };
2
0xe9b6ce14
0xeb0f5b8c
```

付録1.3 list_s

■ 機能

入力されたアドレスからのlist_headリンクをたどり、リンク上のすべての構造体のアドレス配列を返す。

■ 書式

list_s('*address*', '*structure*', '*member*')

■ 引数

address　　構造体のアドレス。
structure　　構造体名。
member　　メンバ名。

■ 戻り値

リンクに参加している構造体のアドレスが配列で返される。
返される配列の先頭は構造体の個数となる。

■ 例

```
alicia> @list = list_s '0xf79d9dd0','dentry','d_hash';
alicia> foreach (@list) { print "$_\n" };
1
0xf4602200
```

付録1.4 list_foreach

■ 機能

入力された配列内のアドレスの構造体のアドレスと特定のメンバの値を一覧表示する。

■ 書式

list_foreach('*structure*', '*member*', *@array*)

■ 引数

structure　　構造体名。
member　　メンバ名。
@array　　先頭にエントリ数を持つ構造体のアドレスのエントリ。

■ 戻り値

なし

■ 例

```
alicia> @dhash = list_s '0xf79d9dd0','dentry','d_hash';
alicia> list_foreach 'dentry','d_name.name',@dhash;
addr       d_name.name
0xf4602200 logfile.suite7
               :
```

```
                    :
```

付録1.5 list_task_vma

■ 機能

入力されたプロセスの仮想メモリ空間を一覧表示する。

■ 書式

list_task_vma('*address*')

■ 引数

address　　task_struct構造体のアドレス。

■ 戻り値

なし

■ 例

```
alicia> list_task_vma '0xe9eb2000';
task e9eb2000
  comm multitask
  pid 29207
  mm 0xeba0c380
    mm.pgd 0xed9c5800
    mm.mm_count.counter 1
start of vm_next list from 0xeb4047e8
  address      vm_start     vm_end       vm_flags
  0xeb4047e8    0x8048000    0x8058000       0x1875
  0xeef0d8f8    0x8058000    0x8059000     0x101873
                          :
                          :
  0xeaa07584   0xb7600000   0xb7601000     0x100873
  0xeff0e2dc   0xbfff8000   0xc0000000     0x100177
end of vm_area_struct.vm_next list
```

付録1.6 task_dentries

■ 機能

入力されたプロセスのアタッチしているすべてのファイルのdentry構造体のアドレスの配列を

返す。

■ 書式

task_dentries('*address*')

■ 引数

address　　task_struct構造体のアドレス。

■ 戻り値

アタッチされているファイルのdentry構造体のアドレスが配列で返される。
返される配列の先頭は構造体の個数となる。

■ 例

```
alicia> @dirs = task_dentries '0xe9eb2000';
alicia> list_foreach 'dentry','d_name.name', @dirs;
addr       d_name.name
0xec659b80 0
0xec659b80 0
0xec659b80 0
0xf4602200 logfile.suite7
```

付録1.7 dentry_parents

■ 機能

入力されたdentryの親のdentryを/(ルート)までたどりdenty構造体のアドレスの配列を返す。

■ 書式

dentry_parents('*address*')

■ 引数

address　　dentry構造体のアドレス。

■ 戻り値

/(ルート・ディレクトリ)までの上位のディレクトリのdentry構造体のアドレスが配列で返される。
返される配列の先頭は構造体の個数となる。

■ 例

```
alicia> @dparents = dentry_parents '0xf4602200';
alicia> list_foreach 'dentry','d_name.name', @dparents;
addr       d_name.name
0xf4602200 logfile.suite7
0xf46c2980 s7110
0xf46c2a00 src
0xf6118f00 usr
0xf6184680 /
```

付録1.8 get_dentry

■ 機能

入力されたフルパスのディレクトリまたはファイルのdentry構造体のアドレスを返す。

■ 書式

get_dentry('*full-filepath*')

■ 引数

full-filepath　　フルパス名。

■ 戻り値

ディレクトリ、またはファイルのdentry構造体のアドレス。

■ 例

```
alicia> print get_dentry '/usr/src/s7110/logfile.suite7';
0xf4602200
```

付録1.9 get_task

■ 機能

入力されたPIDを持つtask_struct構造体のアドレスを返す。

■ 書式

get_task('*pid_number*')

■ 引数

pid_number　　pid番号。

■ 戻り値

入力されたPIDを持つtask_struct構造体のアドレスが返される。

■ 例

```
alicia> $mytask = get_task 29207;
alicia> print kernel $mytask, 'task_struct', 'comm', 'char *';
multitask
```

付録1.10 ts

■ 機能

インタラクティブにtask_struct構造体のメンバの値を表示できる。

■ 書式

ts(['*address*'])

■ 引数

address	task_structの先頭アドレス。 省略した場合は、init_task_unionのアドレスとなる。

■ 戻り値

なし

■ 詳細

tsを実行すると、以下のようなプロンプトが出力される。

```
(c03b6000) >
```

()内は現在参照中のtask_structの先頭アドレスが表示され、tsはユーザの入力を待つ。ここで、ユーザは以下のコマンドなどを入力できる。

n	tasks.nextチェインをたどり、次のtask_structを参照する
p	tasks.prevチェインをたどり、前のtask_structを参照する
q	tsを終了する（Aliciaに戻る）
0xnnnnnnnn	プレフィックス0x付きの8桁の16進数で、参照したいtask_structの 先頭アドレスを指定する

メンバ名　　現在参照しているtask_struct内の任意のメンバの値を表示する
　　入力されたメンバが存在しない場合、入力された文字列を含む
　　メンバ名をすべて表示する
$nnn　　最後に表示したメンバの値を、ユーザ指定の任意の変数$nnnに代入する
　　この変数はts終了後にもAliciaで有効となる

■ 例

```
alicia> ts;     --- ts起動。アドレスは未指定なのでinit_task_unionのアドレスになる。
(c04a6000) > comm       --- メンバ名commの内容を参照する。
"swapper\000\000\000\000\000\000\000\000"
(c04a6000) > n          --- 次のtask_structを参照する。
(f7f70000) > comm       --- プロンプトのアドレスが次のtask_structのアドレスになる。
                            メンバ名commの内容を参照する。
"init\000er\000\000\000\000\000\000\000\000"
(f7f70000) > 0xf46ec000--- task_structの先頭アドレスを指定する。

(f46ec000) > comm       --- プロンプトのアドレスが指定したアドレスになる。
                            メンバ名commの内容を参照する。
"bash\000\000ty\000\000\000\000\000\000\000"
(f46ec000) > $ccc       --- 変数$cccに直前の参照内容を代入する。

(f46ec000) > co         --- メンバ名coを参照する。
CRASH::kernel : There is no member named co at ts.ldas line 37.
exit_code               --- メンバcoなんて居ないのでエラーとなり、
comm[16]                    メンバ名に'co'を含むメンバが全て表示される。
link_count
total_link_count

(f46ec000) > q          --- tsを終了する。

alicia> print $ccc;     --- 通常のAliciaに戻って、ts内で代入した$cccを参照する。
"bash\000\000ty\000\000\000\000\000\000\000"
alicia>
```

付録2 LDASのサンプルを使用した解析例

ここでは、LDASのサンプルとAliciaの標準関数、Crashシームレスコマンドを使用して、カーネルクラッシュ直前の/var/log/messagesにまだ書き込まれていないイメージを採取する解析手順例を示す。

（Step 1.）LDAS get_dentryを使用して、/var/log/messagesのdentry構造体のアドレスを得る

```
alicia> $dent = get_dentry '/var/log/messages';
```

（Step 2.）kernel関数を使用して、dentry構造体のメンバd_inodeの値を得る

```
alicia> $inode = kernel $dent, 'dentry', 'd_inode';
```

（Step 3.）LDAS list_cを使用して、inode構造体の配下のbuffer_head構造体のlist_headリンクをたどり、リンク上の全ての構造体のアドレス配列を得る

```
alicia> $os = 'inode';
alicia> $om = 'i_dirty_data_buffers';
alicia> $cs = 'buffer_head';
alicia> $cm = 'b_inode_buffers';
alicia> @bhs = list_c $inode,$os,$om,$cs,$cm;
```

（Step 4.）LDAS list_foreachを使用して、@bhsをアドレスとする構造体buffer_headのメンバb_pageの値を表示する

```
alicia> list_foreach 'buffer_head','b_page',@bhs;
addr       b_page
0xf10fc7c0 0xc29b515c
```

（Step 5.）kernel関数を使用して、page構造体のメンバvirtualの値を得る

```
alicia> print kernel '0xc29b515c','page','virtual';
0xff081000
```

（Step 6.）Crashコマンド rdを使用して、アドレスff081000から1ページ分のメモリを表示する

```
alicia> rd ff081000 1000;
ff081000:  69686320 7920646c 676e756f 6f207265    child younger o
ff081010:  7265646c 74634f0a 20392020 303a3130   lder.Oct  9 01:0
ff081020:  30333a39 6e6c6d20 6b203178 656e7265   9:30 mlnx1 kerne
ff081030:  69203a6c 2074696e 20202020 20202020   l: init
```

```
ff081040:  43205320 42463430 20303832 20202020    S C04FB280
ff081050:  20202034 20312020 20202020 20203020   4     1      0
ff081060:  34332020 20202020 32202020 20202020     34       2
ff081070:  28202020 4c544f4e 4f0a2942 20207463      (NOTLB).Oct
ff081080:  31302039 3a39303a 6d203033 31786e6c   9 01:09:30 mlnx1
ff081090:  72656b20 3a6c656e 6c614320 7254206c    kernel: Call Tr
ff0810a0:  3a656361 5b202020 3130633c 37623932   ace:   [<c0129b7
                        :
                        :
ff081f80:  00000000 00000000 00000000 00000000   ................
ff081f90:  00000000 00000000 00000000 00000000   ................
```

付録3 リファレンス

Alicia	当使用解説書で解説するCrashをラッピングしたダンプ解析ツール。 または、Alicia起動用プログラム。
alicia_history	Aliciaコマンド履歴ファイル。
alicia.conf	Alicia設定ファイル。システムデフォルトファイルと、ユーザ固有ファイルがある。
Alicia.pm	Aliciaモジュール。Alicia本体。
Crash	既存のダンプ解析ツール。Aliciaは、このCrashをラッピングする。
Crash拡張コマンド	Crashのextendコマンドにより読み込まれたコマンド。
CRASH.pm	CRASHモジュール。主にCrashへのアクセスを受け持つ。
LDAS	Aliciaで読み込み、実行可能な解析手順スクリプト。 実態はPerlのサブルーチン。
LKCD	Linux Kernel Crash Dumpsの略。kernelクラッシュ時のダンプをディスクに採取する機能。詳細は以下のURLを参照のこと。 http://sourceforge.net/projects/lkcd/
mode	Aliciaの動作モード。 ※Version1.0.0では、Crashモードしか使えない。
TAB補完	Aliciaが実装するコマンド名及びファイル名補完機能。
WPSH.pm	WPSHモジュール。主にユーザインタフェースを受け持つ。
シームレス	Crashのコマンドをそのまま実行できる機能。
ダンプ	プログラムやカーネルが不正終了した時のメモリの内容が保存されたファイル。
ヒストリ	Aliciaが実装するコマンド履歴。
標準関数	Aliciaで実装されている関数。
リバース検索	GNU Readlineの書式に従ったコマンド後方検索。

Aliciaは、「独立行政法人 情報処理推進機構 オープンソースソフトウェア活用基盤整備事業」に係る委託業務の一環として開発したものです。