Red Hat Enterprise Linux 7

SystemTap タップセットリファレンス

SystemTap スクリプトの最も一般的なタップセット定義

Last Updated: 2023-01-29

Red Hat Enterprise Linux 7 SystemTap タップセットリファレンス

SystemTap スクリプトの最も一般的なタップセット定義

Enter your first name here. Enter your surname here.
Enter your organisation's name here. Enter your organisational division here.
Enter your email address here.

法律上の通知

概要

このドキュメントは、プレビュー版としてのみ提供されています。これは開発中であり、大幅に変更される可能性があります。含まれる情報は不完全なものと考え、注意して使用してください。
tapset リファレンスガイドは、ユーザーが SystemTap スクリプトに適用できる最も一般的な tapset 定義について説明しています。

目次

第1章 はじめに

SystemTap はフリーソフトウェア (GPL) インフラストラクチャーを提供し、実行中の Linux システムに関する情報の収集を簡素化します。これにより、パフォーマンスまたは機能的な問題の診断が容易になります。SystemTap により、開発者はデータの収集に必要になる可能性のある未知で破壊的なインストルメント、再コンパイル、インストール、および再起動を行う必要がなくなります。

SystemTap では、ライブで実行中のカーネル用にインストルメンテーションを記述する簡単なコマンドラインインターフェイスとスクリプト言語を利用できます。このインストルメンテーションは、*tapset* ライブラリーで提供されるプローブポイントと関数を使用します。

簡単に説明すると、tapset は、カーネルサブシステムに関する知識を他のスクリプトで使用できる事前記述されたプローブおよび関数にカプセル化するスクリプトです。tapsets は C プログラムのライブラリーに類似しています。これらのライブラリーは、カーネルエリアの基礎となる詳細を非表示にし、カーネルの管理および監視に必要な主な情報を公開します。これらは通常、カーネルの専門家によって開発されます。

tapset は高レベルなデータとサブシステムの状態遷移を公開します。多くの場合、優れた tapset の開発者であれば、SystemTap ユーザーがカーネルサブシステムの低レベルの詳細をほとんど認識していないと想定しています。そのため、tapset の開発者は、通常の SystemTap ユーザーが有用で便利な SystemTap スクリプトを記述するのに役立つ tapset を記述します。

1.1. 本ガイドの目的

本ガイドは、SystemTap の最も有用で一般的な tapset エントリーを取り上げることを目的としています。また、適切な tapset 開発やドキュメントに関するガイドラインも記載しています。本ガイドに含まれる tapset の定義は、各 tapset ファイルのコードの適切にフォーマットされたコメントから自動的に抽出されます。そのため、本ガイドの定義に対する改訂は、それぞれの tapset ファイルに直接適用する必要があります。

第2章 TAPSET 開発ガイドライン

本章では、適切な tapset ドキュメントのアップストリームのガイドラインについて説明します。また、本ガイドで適切に定義されるように tapset を適切に記述する方法も含まれています。

2.1. TAPSET の記述

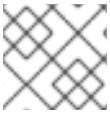
適切な tapset を記述するための最初の手順は、サブジェクトエリアの単純なモデルを作成することです。たとえば、プロセスサブシステムのモデルには以下が含まれる場合があります。

キーデータ

- プロセス ID
- 親プロセス ID
- プロセスグループ ID

状態遷移

- forked (フォーク)
- exec'd (実行)
- running (実行中)
- stopped (停止)
- terminated (終了)



備考

上記のリストは両方とも例であり、完全なリストではありません。

サブシステムの知識を使用して、モデルの要素を公開するプローブポイント (関数エントリーおよび終了) を検索し、それらのポイントのプローブエイリアスを定義します。一部の状態遷移は複数の場所で発生する可能性があることに注意してください。この場合、エイリアスはプローブを複数の場所に配置できます。

たとえば、プロセス exec は **do_execve ()** または **compat_do_execve ()** 関数のいずれかで発生する可能性があります。以下のエイリアスは、これらの関数の最初にプローブを挿入します。

```
probe kprocess.exec = kernel.function("do_execve"),
kernel.function("compat_do_execve")
{probe body}
```

プローブを可能な限り安定したインターフェイス (インターフェイスレベルで変更できない関数など) に配置してみてください。これにより、カーネルの変更により tapset が破損する可能性が低くなります。カーネルのバージョンまたはアーキテクチャーの依存関係を回避できない場合は、プリプロセッサー条件を使用します (詳細は、man ページの **stap(1)** を参照してください)。

プローブポイントで利用可能なキーデータでプローブボディーを入力します。関数エントリープローブは、関数に指定されたエントリーパラメーターにアクセスできますが、終了プローブはエントリーパラメーターおよび戻り値にアクセスできます。適切な場合はデータを意味のある形式に変換します (バイトをキロバイト、状態値は文字列など)。

補助関数を使用して一部のデータにアクセスしたり、変換する必要がある場合があります。補助関数は多くの場合、埋め込み C を使用して SystemTap 言語では実行できないことを行います。たとえば、一部のコンテキスト内の構造フィールドへのアクセス、リンクされた一覧などです。他の tapset で定義された補助関数を使用するか、独自の tapset を作成することができます。

以下の例では、新しいプロセスに対して **task_struct** のポインターを **copy_process()** に返します。新しいプロセスのプロセス ID は **task_pid()** ポインターを呼び出して、**task_struct** を渡すことで取得されます。この場合、補助関数は **task.stp** で定義されている埋め込み C 関数です。

```
probe kprocess.create = kernel.function("copy_process").return
{
  task = $return
  new_pid = task_pid(task)
}
```

すべての関数にプローブを作成することは推奨されません。多くの SystemTap ユーザーには必要ありませんし、理解する必要もありません。簡単で高レベルな tapset を維持します。

2.2. TAPSET の要素

以下のセクションでは、tapset を作成する最も重要な側面を説明します。ここでのコンテンツの大半は、SystemTap の tapset のアップストリームライブラリーへの貢献を希望する開発者に適しています。

2.2.1. tapset ファイル

tapset ファイルは SystemTap GIT ディレクトリー ***src*/tapset/** に保存されます。ほとんどの tapset ファイルはこのレベルで保持されます。特定のアーキテクチャーまたはカーネルバージョンでのみ機能するコードがある場合は、tapset を適切なサブディレクトリーに配置することを選択できます。

インストールされたタップセットは **/usr/share/systemtap/tapset/** または **/usr/local/share/systemtap/tapset にあり** ます。

個人用の tapset はどこにでも保存できます。ただし、SystemTap がこれらの tapset を使用できるようにするには、**-I *tapset_directory*** を使用して **stap** を呼び出す際に場所を指定する必要があります。

2.2.2. Namespace

プローブエイリアス名は、***tapset_name.probe_name*** の形式で指定する必要があります。たとえば、シグナルを送信するプローブは、**signal.send** とすることができます。

グローバルシンボル名 (プローブ、関数、および変数) は tapset 全体で一意である必要があります。これは、複数の tapset を使用するスクリプトで名前空間の競合を回避するのに役立ちます。これを確認するには、グローバルシンボルで tapset 固有の接頭辞を使用します。

内部シンボル名にはアンダースコア (_) を接頭辞として使用する必要があります。

2.2.3. コメントおよびドキュメント

すべてのプローブおよび関数には、目的、提供するデータ、および実行するコンテキストを記述するコメントブロック (割り込み、プロセスなど) が含まれている必要があります。コードの読み取りでは、不明瞭なエリアでコメントを使用します。

特殊形式のコメントは、多くの tapset から自動的に抽出され、本ガイドに含まれています。これにより、tapset の貢献者が tapset を作成し、*同じ場所で文書化するのに役立ちます。tapset の文書化に指定される書式は、以下のとおりです。*

```
/**
 * probe tapset.name - Short summary of what the tapset does.
 * @argument: Explanation of argument.
 * @argument2: Explanation of argument2. Probes can have multiple arguments.
 *
 * Context:
 * A brief explanation of the tapset context.
 * Note that the context should only be 1 paragraph short.
 *
 * Text that will appear under "Description."
 *
 * A new paragraph that will also appear under the heading "Description".
 *
 * Header:
 * A paragraph that will appear under the heading "Header".
 **/
```

以下に例を示します。

```
/**
 * probe vm.write_shared_copy- Page copy for shared page write.
 * @address: The address of the shared write.
 * @zero: Boolean indicating whether it is a zero page
 *        (can do a clear instead of a copy).
 *
 * Context:
 *  The process attempting the write.
 *
 *  Fires when a write to a shared page requires a page copy.  This is
 *  always preceded by a vm.shared_write.
 **/
```

*自動生成される **Synopsis** コンテンツをオーバーライドするには、以下を使用します。*

```
 * Synopsis:
 * New Synopsis string
 *
```

以下に例を示します。

```
/**
 * probe signal.handle - Fires when the signal handler is invoked
 * @sig: The signal number that invoked the signal handler
 *
 * Synopsis:
 * <programlisting>static int handle_signal(unsigned long sig, siginfo_t *info, struct k_sigaction *ka,
 * sigset_t *oldset, struct pt_regs * regs)</programlisting>
 */
```

*エントリーの **Synopsis** の内容を上書きしても必要なタグが自動的に作成されないため、この例では **<programlisting>** タグの使用を推奨しています。*

コメントの DocBook XML 出力を改善するため、コメントに以下の XML タグを使用することもできます。

- ***command***
- ***emphasis***
- ***programlisting***
- ***remark*** *(タグ付けされた文字列が、ドキュメントの Publican ベータビルドに表示されます)*

第3章 コンテキスト関数

コンテキスト関数では、イベントが発生した場所に関する追加情報を利用できます。これらの関数は、イベントが発生した場所へのバックトレースやプロセッサーの現在のレジスター値へのバックトレースなどの情報を提供します。

名前

function::addr – 現在のプローブポイントのアドレス。

概要

```
addr:long()
```

引数

なし

説明

現在のプローブのレジスタ状態から命令ポインターを返します。ただし、すべてのプローブタイプにレジスタがあるわけではなく、その場合はゼロが返されます。返されたアドレスは、次のような関数での使用に適しています**symname**と**symdata**.

名前

function::asmlinkage – 関数を宣言された asmlinkage としてマークする

概要

```
asmlinkage()
```

引数

なし

説明

プローブされたカーネル関数がソースで asmlinkage と宣言されている場合は、*_arg 関数を使用して引数にアクセスする前に、この関数を呼び出します。

名前

function::backtrace – 現在のカーネルスタックの 16 進バックトレース

概要

```
backtrace:string()
```

引数

なし

説明

この関数は、スタックのバックトレースである 16 進アドレスの文字列を返します。出力は文字列の最大長 (MAXSTRINGLEN) に従って切り捨てられることがあります。見る**ubacktrace**ユーザー空間のバックトレース用。

名前
function::caller — 呼び出し元関数の名前およびアドレスを返します。

概要

```
caller:string()
```

引数
なし
説明
この関数は呼び出し元関数のアドレスを返します。これは次の呼び出しと同じです: sprintf (「**s**0x**x**」、同名 (**caller_addr**)、**caller_addr**)

名前
function::caller_addr — 呼び出し元アドレスを返します。

概要

```
caller_addr:long()
```

引数
なし
説明
この関数は呼び出し元関数のアドレスを返します。

名前
function::callers — カーネルスタックバックトレースの最初の n 個の要素を返す

概要

```
callers:string(n:long)
```

引数

n

スタック内で下降するレベルの数 (トップレベルは数えません)。n が -1 の場合、スタック全体を出力します。

説明
この関数は、カーネルスタックのバックトレースから最初の n 個の 16 進アドレスの文字列を返します。出力は文字列の最大長 (MAXSTRINGLEN) に従って切り捨てられることがあります。

名前
function::cmdline_arg — コマンドライン引数を取得します。

概要

```
cmdline_arg:string(n:long)
```

引数

n

取得する引数 (ゼロはプログラム自体)

説明

現在のプロセスから要求された引数を返します。それほど多くの引数がない場合や、引数を取得できなかった場合は、空の文字列を返します。通常、引数 0 はコマンド自体です。

名前

function::cmdline_args – 現在のプロセスからコマンドライン引数を取得します。

概要

```
cmdline_args:string(n:long,m:long,delim:string)
```

引数

n

取得する最初の引数 (ゼロは通常、 プログラム自体です)

m

最後に取得する引数 (-1 は n の後の引数すべて)。

delim

複数の場合に引数を区切るために使用する文字列。

説明

現在のプロセスから、n から m までの引数を返します。n 未満の引数がある場合や、現在のプロセスから引数を取得できない場合は、空の文字列が返されます。m が n よりも小さい場合は、引数 n から始まるすべての引数が返されます。引数 0 は、通常コマンド自体です。

名前

function::cmdline_str – 現在のプロセスからすべてのコマンドライン引数を取得します。

概要

```
cmdline_str:string()
```

引数

なし

説明

現在のプロセスから、スペースで区切られたすべての引数を返します。引数を取得できない場合は空の文字列を返します。

名前
function::cpu — 現在の CPU 番号を返します。

概要

```
cpu:long()
```

引数
なし
説明
この関数は、現在の CPU 番号を返します。

名前
function::cpuid — 現在の CPU 番号を返します。

概要

```
cpuid:long()
```

引数
なし
説明
この関数は、現在の CPU 番号を返します。SystemTap 1.4 で廃止され、SystemTap 1.5 で削除されました。

名前
function::egid — ターゲットプロセスの実効 GID を返します。

概要

```
egid:long()
```

引数
なし
説明
この関数は、ターゲットプロセスの実効 GID を返します。

名前
function::env_var — 現在のプロセスから環境変数を取得します。

概要

```
env_var:string(name:string)
```

引数

name

取得する環境変数の名前。

説明

現在のプロセスの指定された環境値の内容を返します。変数が設定されていないと、空の文字列が返されます。

名前

function::euid – ターゲットプロセスの実効UID を返します。

概要

```
euid:long()
```

引数

なし

説明

ターゲットプロセスの実効ユーザーID を返します。

名前

function::execname – ターゲットプロセス (またはプロセスのグループ) の実行名を返します。

概要

```
execname:string()
```

引数

なし

説明

ターゲットプロセス (またはプロセスのグループ) の実行名を返します。

名前

function::fastcall – 関数を宣言された fastcall としてマークする

概要

```
fastcall()
```

引数

なし

説明

*プローブされたカーネル関数がソースで fastcall と宣言されている場合は、*_arg 関数を使用して引数にアクセスする前にこの関数を呼び出します。*

名前
function::gid – ターゲットプロセスのグループID を返します。

概要

```
gid:long()
```

引数
なし
説明
この関数は、ターゲットプロセスのグループID を返します。

名前
function::int_arg – 関数の引数を signed int として返す

概要

```
int_arg:long(n:long)
```

引数

n

返す引数のインデックス

説明
引数 n の値を signed int (つまり、64 ビットに符号拡張された 32 ビット整数) として返します。

名前
function::is_myproc – ユーザー独自のプロセスで現在のプローブポイントが発生したかどうかを判断します。

概要

```
is_myproc:long()
```

引数
なし
説明
ユーザー独自のプロセスで現在のプローブポイントが発生した場合、この関数は 1 を返します。

名前
function::is_return – 現在のプローブコンテキストが return プローブであるかどうかを指定します。

概要

```
is_return:long()
```

引数

なし

説明

現在のプローブコンテキストがreturn プローブである場合は1 を返し、それ以外の場合は 0 を返します。

名前

function::long_arg – 関数の引数を符号付き long として返す

概要

```
long_arg:long(n:long)
```

引数

n

返す引数のインデックス

説明

引数n の値を符号付き long として返します。long が32 ビットのアーキテクチャーでは、値は64 ビットに符号拡張されます。

名前

function::longlong_arg – 関数の引数を 64 ビット値として返す

概要

```
longlong_arg:long(n:long)
```

引数

n

返す引数のインデックス

説明

引数n の値を 64 ビット値として返します。

名前

function::modname – アドレスでロードされたカーネルモジュール名を返します。

概要

```
modname:string(addr:long)
```

引数

addr

カーネルモジュール名にマップするアドレス

説明

既知の場合、指定のアドレスに関連付けられたモジュール名を返します。不明な場合は、エラーが発生します。アドレスがカーネルモジュールではなく、カーネル自体にない場合は、文字列 「kernel」 を返します。

名前

function::module_name — 現在のスクリプトのモジュール名。

概要

```
module_name:string()
```

引数

なし

説明

この関数は、stap モジュールの名前を返します。ランダムに生成される (stap_[0-9a-f]+_[0-9a-f]+) または stap -m <module_name> で設定されます。

名前

function::module_size — 現在のスクリプトのモジュールサイズ

概要

```
module_size:string()
```

引数

なし

説明

この関数は、stap モジュールのさまざまなセクションのサイズを返します。

名前

function::ns_egid — ユーザー名前空間で見られるターゲットプロセスの有効な gid を返します。

概要

```
ns_egid:long()
```

引数

なし

説明

この関数は、ターゲットユーザー名前空間 (提供されている場合) または stap プロセス名前空間で見られるターゲットプロセスの有効な gid を返します。

名前

function::ns_euid – ユーザー名前空間に表示されるターゲットプロセスの実効ユーザー ID を返します。

概要

```
ns_euid:long()
```

引数

なし

説明

この関数は、ターゲットユーザーの名前空間 (提供されている場合) または stap プロセスの名前空間で見られる、ターゲットプロセスの有効なユーザー ID を返します。

名前

function::ns_gid – ユーザー名前空間に表示されるターゲットプロセスのグループ ID を返します。

概要

```
ns_gid:long()
```

引数

なし

説明

この関数は、ターゲットユーザー名前空間 (提供されている場合) または stap プロセス名前空間に表示されるターゲットプロセスのグループ ID を返します。

名前

function::ns_pgrp – pid 名前空間で見られる現在のプロセスのプロセスグループ ID を返します。

概要

```
ns_pgrp:long()
```

引数

なし

説明

この関数は、提供されている場合はターゲット pid 名前空間、または stap プロセス名前空間で見られる現在のプロセスのプロセスグループ ID を返します。

名前

function::ns_pid – pid 名前空間で見られるターゲットプロセスの ID を返します。

概要

```
ns_pid:long()
```

引数
なし
説明
この関数は、ターゲット pid 名前空間で見られるターゲットプロセスの ID を返します。

名前
function::ns_ppid – pid 名前空間で見られるターゲットプロセスの親プロセスのプロセス ID を返します。

概要

```
ns_ppid:long()
```

引数
なし
説明
この関数は、ターゲット pid 名前空間 (提供されている場合) または stap プロセス名前空間で見られる、ターゲットプロセスの親プロセスのプロセス ID を返します。

名前
function::ns_sid – pid 名前空間で見られる現在のプロセスのセッション ID を返します

概要

```
ns_sid:long()
```

引数
なし
説明
プロセスの名前空間認識セッション ID は、指定されている場合はターゲット pid 名前空間、または stap プロセス名前空間で見られるセッションリーダーのプロセスグループ ID です。Kernel 2.6.0 以降では、セッション ID は signal_struct に格納されます。

名前
function::ns_tid – pid 名前空間で見られるターゲットプロセスのスレッド ID を返します。

概要

```
ns_tid:long()
```

引数
なし

説明
この関数は、ターゲット pid 名前空間 (提供されている場合) または stap プロセス名前空間で見られるターゲットプロセスのスレッド ID を返します。

名前
function::ns_uid – ユーザー名前空間に表示されるターゲットプロセスのユーザー ID を返します。

概要

```
ns_uid:long()
```

引数
なし
説明
この関数は、ターゲットユーザー名前空間 (提供されている場合) または stap プロセス名前空間に表示されるターゲットプロセスのユーザー ID を返します。

名前
function::pexecname – ターゲットプロセスの親プロセスの実行名を返します。

概要

```
pexecname:string()
```

引数
なし
説明
この関数は、ターゲットプロセスの親プロセスの実行名を返します。

名前
function::pgrp – 現在のプロセスのプロセスグループ ID を返します。

概要

```
pgrp:long()
```

引数
なし
説明
この関数は、現在のプロセスのプロセスグループ ID を返します。

名前
function::pid – ターゲットプロセスの ID を返します。

概要

```
pid:long()
```

引数
なし
説明
この関数は、ターゲットプロセスの ID を返します。

名前
function::pid2execname — 指定のプロセス識別子の名前。

概要

```
pid2execname:string(pid:long)
```

引数

pid
プロセス識別子。

説明
指定のプロセス ID の名前を返します。

名前
function::pid2task — 指定のプロセス識別子の task_struct。

概要

```
pid2task:long(pid:long)
```

引数

pid
プロセス識別子。

説明
指定のプロセス ID の task 構造を返します。

名前
function::pn — アクティブなプローブ名を返します。

概要

```
pn:string()
```

引数

なし

説明

この関数は、現在実行しているプローブハンドラーに関連するスクリプトレベルのプローブポイントを返します (ワイルドカードによる拡張を含む)。コンテキスト: 現在のプローブポイント。

名前

function::pnlabel – プローブ名から解析されたラベル名を返します

概要

```
pnlabel:string()
```

引数

なし

説明

これは、スクリプトレベルのプローブポイントから解析されたラベル名を返します。この関数は、'.label' プローブポイントの本体から直接呼び出された場合 (つまり、エイリアスがない場合) にのみ機能します。

コンテキスト

現在のプローブポイント。

名前

function::pointer_arg – 関数の引数をポインター値として返す

概要

```
pointer_arg:long(n:long)
```

引数

n

返す引数のインデックス

説明

ulong_arg と同じように、引数 n の符号なしの値を返します。任意のタイプのポインターで使用できます。

名前

function::pp – アクティブなプローブポイントを返します。

概要

```
pp:string()
```

引数

なし

説明
この関数は、現在実行しているプローブハンドラーに関連する完全に解決されたプローブポイントを返します (エイリアスやワイルドカードによる拡張を含む)。コンテキスト: 現在のプローブポイント。

名前
function::ppfunc – から解析された関数名を返します **pp**

概要

```
ppfunc:string()
```

引数
なし
説明
これは、現在の関数名を返します**pp**.すべてではない**pp**その中に関数があり、その場合は "" が返されます。

名前
function::ppid – ターゲットプロセスの親プロセスのプロセス ID を返します。

概要

```
ppid:long()
```

引数
なし
説明
この関数は、ターゲットプロセスの親プロセスのプロセス ID を返します。

名前
function::print_backtrace – カーネルスタックバックトレースを出力する

概要

```
print_backtrace()
```

引数
なし
説明
この関数は print_stack (**backtrace**) と同等ですが、より深いスタックのネストがサポートされます。ユーザー空間のバックトレースについては、print_ubacktrace を参照してください。この関数は値を返しません。

名前
function::print_regs – レジスターダンプを出力します。

概要

```
print_regs()
```

引数

なし

説明

この関数はレジスターダンプを出力します。プローブポイントに使用できるレジスタがない場合は何もしません。

名前

function::print_stack – 文字列からカーネルスタックを出力します。

概要

```
print_stack(stk:string)
```

引数

stk

16 進アドレスのリストが含まれる文字列。

説明

この関数は、**backtrace** への事前呼び出しの結果となることが仮定される指定の文字列でアドレスのシンボリックルックアップを実行します。
アドレスごとに 1 行出力し、アドレス、アドレスが含まれる関数の名前、およびその関数内での推定位置が含まれます。戻り値はありません。

注記

使用することをお勧めします**print_syms**この関数の代わりに。

名前

function::print_syms – 文字列からカーネルスタックを出力します。

概要

```
print_syms(callers:string)
```

引数

発信者

16 進 (カーネル) アドレスのリストが含まれる文字列。

説明

この関数は、指定された文字列内のアドレスのシンボリックルックアップを実行します。これは、以前の呼び出しの結果であると想定されます。**stack**、**callers**、および同様の機能。
アドレスごとに 1 行出力し、アドレス、アドレスが含まれる関数の名前、およびその関数内での推定位置が含まれます。何も返しません。

名前
function::print_ubacktrace – 現在のユーザー空間タスクのスタックバックトレースを出力します。

概要

```
print_ubacktrace()
```

引数
なし
説明
print_ustack(**ubacktrace**) と同等ですが、より深いスタックのネストがサポートされます。何も返しません。見る**print_backtrace**カーネルのバックトレース用。
注記
ユーザー空間アプリケーションと現在のスクリプトに記述されていない共有ライブラリーの (完全な) バックトレースを取得するには、stap を -d /path/to/exe-or-so で実行するか、または --ldd を追加して必要なすべてのアンワインドデータをロードします。

名前
function::print_ubacktrace_brief – 現在のユーザー空間タスクのスタックバックトレースを出力します。

概要

```
print_ubacktrace_brief()
```

引数
なし
説明
print_ubacktrace と同様ですが、各シンボルの出力は短くなります (名前とオフセットのみ、もしくはシンボルなしの 16 進数アドレスのみ)。
備考
ユーザー空間アプリケーションと現在のスクリプトに記述されていない共有ライブラリーの (完全な) バックトレースを取得するには、stap を -d /path/to/exe-or-so で実行するか、または --ldd を追加して必要なすべてのアンワインドデータをロードします。

名前
function::print_ustack – 文字列から現在のタスクのスタックを出力します。

概要

```
print_ustack(stk:string)
```

引数

stk

現在のタスクの 16 進アドレスのリストが含まれる文字列。

説明
現在のタスクの **ubacktrace** への事前呼び出しの結果となることが仮定される指定の文字列でアドレスのシンボリックルックアップを実行します。
アドレスごとに1行出力し、アドレス、アドレスが含まれる関数の名前、およびその関数内での推定位置が含まれます。戻り値はありません。
注記
使用することをお勧めします**print_usyms**この関数の代わりに。

名前
function::print_usyms – 文字列からユーザースタックを出力する

概要

```
print_usyms(callers:string)
```

引数

発信者

16 進数 (ユーザー) アドレスのリストを含む文字列

説明
この関数は、指定された文字列内のアドレスのシンボリックルックアップを実行します。これは、以前の呼び出しの結果であると想定されます。**ustack**、**ucallers**、および同様の機能。
アドレスごとに1行出力し、アドレス、アドレスが含まれる関数の名前、およびその関数内での推定位置が含まれます。何も返しません。

名前
function::probe_type – 現在のプローブの低レベルプローブハンドラータイプ。

概要

```
probe_type:string()
```

引数
なし
説明
現在のプローブポイントの低レベルプローブハンドラータイプを説明する短い文字列を返します。これは情報提供のみを目的としています。低レベルのプローブハンドラーに応じて、さまざまなコンテキスト関数が現在のイベントに関する情報を提供できる場合と提供できない場合があります (たとえば、一部のプローブハンドラーはユーザー空間でのみトリガーされ、関連付けられたカーネルコンテキストはありません)。高レベルのプローブは、同じまたは異なる低レベルのプローブにマップされる場合があります (systemtap のバージョンや使用されているカーネルによって異なります)。

名前
function::probefunc – 既知の場合は、プローブポイントの関数名を返します。

概要

```
probefunc:string()
```

引数
なし
説明
この関数は、symname (によって計算された現在のアドレスに基づいてプローブされている関数の名前を返します。**addr**) または usymname (**uaddr**) プローブのコンテキスト (プローブがユーザープローブかカーネルプローブか) に応じて異なります。
注記
この関数の動作は、SystemTap 2.0 とそれ以前のバージョンで異なります。2.0 より前は、**probefunc**によって返されるプローブポイント文字列から関数名を取得しました。**pp**、現在のアドレスをフォールバックとして使用しました。
使用を検討してください**ppfunc**代わりは。

名前
function::probemod – プローブポイントのカーネルモジュール名を返します。

概要

```
probemod:string()
```

引数
なし
説明
既知の場合、この関数はプローブポイントが含まれるカーネルモジュールの名前を返します。

名前
function::pstrace – init (1) に戻るプロセスと pid のチェーン

概要

```
pstrace:string(task:long)
```

引数

task

プロセスのタスク構造体へのポインター

説明
この関数は、各プロセスの execname と pid をリストする文字列を、**タスク** から開始し、init (1) が生成したプロセスの祖先まで戻します。

名前
function::register – 指定された CPU レジスタの符号付きの値を返します

概要

```
register:long(name:string)
```

引数

name

返すレジスタの名前

説明

現在のプローブポイントにヒットしたときに保存された名前付き CPU レジスタの値を返します。レジスタが 32 ビットの場合は、64 ビットに符号拡張されます。
i386 アーキテクチャーでは、次の名前が認識されます。(name1/name2 は、name1 と name2 が同じレジスタの代替名であることを示します。) eax/ax、ebp/bp、ebx/bx、ecx/cx、edi/di、edx/dx、eflags/flags、eip/ip、esi/si、esp/sp、orig_eax/orig_ax、xcs/cs、xds/ds、xes/es、xfs/fs、xss/ss。
x86_64 アーキテクチャーでは、次の名前が認識されます。64 ビットレジスタ: r8、r9、r10、r11、r12、r13、r14、r15、rax/ax、rbp/bp、rbx/bx、rcx/cx、rdi/di、rdx/dx、rip/ip、rsi/si、rsp/sp; 32 ビットレジスタ: eax、ebp、ebx、ecx、edx、edi、edx、eip、esi、esp、flags/eflags、orig_eax; セグメントレジスタ: xcs/cs、xss/ss。
powerpc の場合、次の名前が認識されます: r0、r1、... r31、nip、msr、orig_gpr3、ctr、link、xer、ccr、softe、trap、dar、dsisr、result。
s390x の場合、次の名前が認識されます: r0、r1、... r15、args、psw.mask、psw.addr、orig_gpr2、ilc、trap。
AArch64 の場合、次の名前が認識されます: x0、x1、... x30、fp、lr、sp、pc、および orig_x0。

名前

function::registers_valid – 現在のコンテキストの **register** および **u_register** の有効性を決定します。

概要

```
registers_valid:long()
```

引数

なし

説明

この関数は、**register** と **u_register**が現在のコンテキストで使用できる場合に 1 を返します。そうでない場合は 0 を返します。たとえば、 は begin または end プローブから呼び出されると 0 を返します。

名前

function::regparm – 関数のコンパイルに使用する regparm 値を指定します

概要

```
regparm(n:long)
```

引数

n

元の regparm 値

説明

*_arg 関数を使用して関数引数にアクセスする前に、引数 n を指定してこの関数を呼び出します。関数は gcc -mregparm=n オプションでビルドされています。
(i386 カーネルは \-mregparm=3 で構築されているため、systemtap は regparm (3) をそのアーキテクチャーのカーネル関数のデフォルトと見なします。) i386 および x86_64 でのみ有効です (32 ビットアプリケーションをプローブする場合)。他のアーキテクチャーではエラーが発生します。

名前

function::remote_id – リモート実行におけるこのインスタンスのインデックス。

概要

```
remote_id:long()
```

引数

なし

説明

この関数は数値 0..N を返します。これは stap 「--remote A --remote B ... 」 実行の群れからのこの特定のスクリプト実行の一意のインデックスであり、stap 「--remote-prefix」 と同じ数値です。印刷します。スクリプトが stap 「--remote」 で起動されなかった場合、またはリモート staprun/stapsh がバージョン 1.7 より古い場合、関数は -1 を返します。

名前

function::remote_uri – リモート実行でのこのインスタンスの名前。

概要

```
remote_uri:string()
```

引数

なし

説明

この関数は、一連の 「stap --remote」 実行からこの特定のスクリプト実行を呼び出すために使用されるリモートホストを返します。群れの中でユニークではないかもしれません。スクリプトが 「stap --remote」 で起動されていない場合、関数は空の文字列を返します。

名前

function::s32_arg – 関数の引数を符号付き 32 ビット値として返す

概要

```
s32_arg:long(n:long)
```

引数

n

返す引数のインデックス

説明
int_arg と同じように、引数 n の符号付き 32 ビット値を返します。

名前
function::s64_arg – 関数の引数を符号付き 64 ビット値として返す

概要

```
s64_arg:long(n:long)
```

引数

n

返す引数のインデックス

説明
longlong_arg と同じように、引数 n の符号付き 64 ビット値を返します。

名前
function::sid – 現在のプロセスのセッション ID を返します。

概要

```
sid:long()
```

引数
なし
説明
プロセスのセッション ID は、セッションリーダーのプロセスグループ ID です。Kernel 2.6.0 以降では、セッション ID は signal_struct に格納されます。

名前
function::sprint_backtrace – スタックバックトレースを文字列として返します。

概要

```
sprint_backtrace:string()
```

引数
なし
説明
単純な (カーネル) バックトレースを返します。1 つのアドレスにつき 1 行。シンボル名 (シンボルが解決できない場合は 16 進アドレス) およびモジュール名 (見つかった場合) が含まれます。見つかった場合は、関数の開始点からのオフセットが含まれます。そうでなければ、オフセットはモジュールに追加されます (見つかった場合は括弧間)。文字列としてバックトレースを返します (改行文字で終端される各

*行)。返されたスタックは MAXSTRINGLEN に切り捨てられます。より完璧で優れたスタックを出力するには、**print_backtrace** を使用することに注意してください。Sprint_stack() と同等ですが、より効率的です (16 進文字列と最終バックトレース文字列間の変換は不要)。*

名前

function::sprint_stack — 文字列からカーネルアドレスのスタックを返します。

概要

```
sprint_stack:string(stk:string)
```

引数

stk

16 進 (カーネル) アドレスのリストが含まれる文字列。

説明

***backtrace** への事前呼び出しの結果となることが仮定される指定の文字列でアドレスのシンボリックルックアップを実行します。*
指定の 16 進文字列から単純なバックトレースを返します。1 つのアドレスにつき 1 行。シンボル名 (シンボルが解決できない場合は 16 進アドレス) およびモジュール名 (見つかった場合) が含まれます。見つかった場合は、関数の開始点からのオフセットが含まれます。そうでなければ、オフセットはモジュールに追加されます (見つかった場合は括弧間)。文字列としてバックトレースを返します (改行文字で終端される各行)。返されたスタックは MAXSTRINGLEN に切り捨てられます。より完璧で優れたスタックを出力するには、print_stack を使用することに注意してください。

注記

*使用することをお勧めします**sprint_syms**この関数の代わりに。*

名前

function::sprint_syms — 文字列からカーネルアドレスのスタックを返します。

概要

```
sprint_syms(callers:string)
```

引数

発信者

16 進 (カーネル) アドレスのリストが含まれる文字列。

説明

*指定された文字列内のアドレスのシンボリック検索を実行します。これは、以前の呼び出しの結果であると想定されます**stack**、**callers**、および同様の機能。*
*指定の 16 進文字列から単純なバックトレースを返します。1 つのアドレスにつき 1 行。から取得したシンボル名 (シンボルを解決できなかった場合は 16 進アドレス) とモジュール名 (見つかった場合) が含まれます。**symdata** .見つかった場合は、関数の開始点からのオフセットが含まれます。そうでなけれ*

ば、オフセットはモジュールに追加されます (見つかった場合は括弧間)。文字列としてバックトレースを返します (改行文字で終端される各行)。返されたスタックは MAXSTRINGLEN に切り捨てられます。より完璧で優れたスタックを出力するには、 を使用することに注意してください。

名前

function::sprint_ubacktrace – 現在のユーザー空間タスクのスタックバックトレースを文字列として返します。

概要

```
sprint_ubacktrace:string()
```

引数

なし

説明

*現在のタスクの単純なバックトレースを返します。1 つのアドレスにつき 1 行。シンボル名 (シンボルが解決できない場合は 16 進アドレス) およびモジュール名 (見つかった場合) が含まれます。見つかった場合は、関数の開始点からのオフセットが含まれます。そうでなければ、オフセットはモジュールに追加されます (見つかった場合は括弧間)。文字列としてバックトレースを返します (改行文字で終端される各行)。返されたスタックは MAXSTRINGLEN に切り捨てられます。より完璧で優れたスタックを出力するには、**print_ubacktrace** を使用することに注意してください。Sprint_ustack() と同等ですが、より効率的です (16 進文字列と最終バックトレース文字列間の変換は不要)。*

備考

ユーザー空間アプリケーションと現在のスクリプトに記述されていない共有ライブラリーの (完全な) バックトレースを取得するには、stap を -d /path/to/exe-or-so で実行するか、または --ldd を追加して必要なすべてのアンワインドデータをロードします。

名前

function::sprint_ustack – 文字列から現在のタスクのスタックを返します。

概要

```
sprint_ustack:string(stk:string)
```

引数

stk

現在のタスクの 16 進アドレスのリストが含まれる文字列。

説明

*現在のタスクの **ubacktrace** への事前呼び出しの結果となることが仮定される指定の文字列でアドレスのシンボリックルックアップを実行します。*
指定の 16 進文字列から単純なバックトレースを返します。1 つのアドレスにつき 1 行。シンボル名 (シンボルが解決できない場合は 16 進アドレス) およびモジュール名 (見つかった場合) が含まれます。見つかった場合は、関数の開始点からのオフセットが含まれます。そうでなければ、オフセットはモジュールに追加されます (見つかった場合は括弧間)。文字列としてバックトレースを返します (改行文字で終端される各行)。返されたスタックは MAXSTRINGLEN に切り捨てられます。より完璧で優れたスタックを出力するには、print_ustack を使用することに注意してください。

注記

*使用することをお勧めします**sprint_usyms**この関数の代わりに。*

名前

function::sprint_usyms – 文字列からユーザーアドレスのスタックを返す

概要

```
sprint_usyms(callers:string)
```

引数

発信者

16 進数 (ユーザー) アドレスのリストを含む文字列

説明

*指定された文字列内のアドレスのシンボリック検索を実行します。これは、以前の呼び出しの結果であると想定されます**ustack**、**ucallers**、および同様の機能。*
*指定の 16 進文字列から単純なバックトレースを返します。1 つのアドレスにつき 1 行。から取得したシンボル名 (シンボルを解決できなかった場合は 16 進アドレス) とモジュール名 (見つかった場合) が含まれます。**usymdata**.見つかった場合は、関数の開始点からのオフセットが含まれます。そうでなければ、オフセットはモジュールに追加されます (見つかった場合は括弧間)。文字列としてバックトレースを返します (改行文字で終端される各行)。返されたスタックは MAXSTRINGLEN に切り捨てられます。より完璧で優れたスタックを出力するには、 を使用することに注意してください。*

名前

function::stack – カーネルスタックバックトレースの特定の深さでアドレスを返す

概要

```
stack:long(n:long)
```

引数

n

スタックで下降するレベル数。

説明

*単純な (カーネル) バックトレースを実行し、指定された位置にある要素を返します。バックトレース自体の結果はキャッシュされるため、バックトレースの計算は何度実行しても最大 1 回です。**stack**呼び出されるか、またはどの順序で呼び出されます。*

名前

function::stack_size – カーネルスタックのサイズを返します。

概要

```
stack_size:long()
```

引数
なし
説明
この関数は、カーネルスタックのサイズを返します。

名前
function::stack_unused – 現在使用可能なカーネルスタックの容量を返します。

概要

```
stack_unused:long()
```

引数
なし
説明
この関数は、カーネルスタックの現在の空き容量 (バイト数) を判断します。

名前
function::stack_used – 使用されたカーネルスタックの容量を返します。

概要

```
stack_used:long()
```

引数
なし
説明
この関数は、カーネルスタックで現在使用されている容量 (バイト数) を判断します。

名前
function::stp_pid – stapio プロセスのプロセス ID。

概要

```
stp_pid:long()
```

引数
なし
説明
この関数は、このスクリプトを起動した stapio プロセスのプロセス ID を返します。システム上では、その他の SystemTap スクリプトや stapio プロセスが実行されている可能性があります。

名前
function::symdata – アドレスのカーネルシンボルとモジュールオフセットを返します。

概要

```
symdata:string(addr:long)
```

引数

addr

変換するアドレス。

説明
既知の場合、指定のアドレスに関連する (関数) シンボル名、シンボルの開始およびサイズ、およびモジュール名 (括弧間) を返します。シンボルが不明でもモジュールが分かっている場合は、モジュール内のオフセットとモジュールのサイズが追加されます。要素が不明な場合は省略され、シンボル名が不明な場合は、指定のアドレスの 16 進文字列を返します。

名前
function::symfile – 指定されたアドレスのファイル名を返します。

概要

```
symfile:string(addr:long)
```

引数

addr

変換するアドレス。

説明
既知の場合、指定されたアドレスのファイル名を返します。ファイル名が見つからない場合は、アドレスの 16 進文字列表現が返されます。

名前
function::symfileline – アドレスのファイル名と行番号を返します。

概要

```
symfileline:string(addr:long)
```

引数

addr

変換するアドレス。

説明
既知の場合、指定されたアドレスのファイル名と (おおよその) 行番号を返します。ファイル名または行番号が見つからない場合は、アドレスの 16 進文字列表現が返されます。

名前
function::symline – アドレスの行番号を返します。

概要

```
symline:string(addr:long)
```

引数

addr

変換するアドレス。

説明
既知の場合、指定されたアドレスの (おおよその) 行番号を返します。行番号が見つからない場合は、アドレスの 16 進文字列表現が返されます。

名前
function::symname – 指定のアドレスに関連するカーネルシンボルを返します。

概要

```
symname:string(addr:long)
```

引数

addr

変換するアドレス。

説明
既知の場合、指定のアドレスに関連する (関数) シンボル名を返します。分からない場合は、addr の 16 進文字列表記を返します。

名前
function::target – ターゲットプロセスのプロセス ID を返します。

概要

```
target:long()
```

引数
なし

説明

この関数は、ターゲットプロセスのプロセス ID を返します。これは、stap に対する -x PID または -c CMD コマンドラインオプションと併用する場合に便利です。その使用例として、特定のプロセスでフィルターするスクリプトを作成できます。
-x < ピッド >**target** -x で指定された pid を返します
target -c で指定された実行済みコマンドの pid を返します

名前

function::task_ancestry — 指定されたタスクの祖先

概要

```
task_ancestry:string(task:long,with_time:long)
```

引数

task

task_struct ポインター

with_time

1 に設定すると、プロセスの開始時刻も出力されます (起動時刻からのデルタとして与えられます)。

説明

指定されたタスクの祖先を 「grandparent_process=>parent_process=>process」 の形式で返します。

名前

function::task_backtrace — 任意タスクの 16 進バックトレース。

概要

```
task_backtrace:string(task:long)
```

引数

task

task_struct へのポインター

説明

この関数は、特定タスクのスタックのバックトレースである 16 進アドレスの文字列を返します。出力は文字列の最大長に従って切り捨てられることがあります。SystemTap 1.6 で廃止されました。

名前

function::task_cpu — タスクのスケジュールされた CPU。

概要

-

```
task_cpu:long(task:long)
```

引数

task

task_struct ポインター

説明

この関数は、指定タスクのスケジュールされた CPU を返します。

名前

function::task_current – 現在のタスクの現在の task_struct。

概要

```
task_current:long()
```

引数

なし

説明

この関数は、現在のプロセスを表す task_struct を返します。このアドレスは、タスク固有のデータを抽出するためにさまざまな task_*() 関数に渡すことができます。

名前

function::task_cwd_path – タスクの現在の作業ディレクトリーのパス構造体ポインターを取得します

概要

```
task_cwd_path:long(task:long)
```

引数

task

task_struct ポインター

名前

function::task_egid – タスクの実効グループ識別子。

概要

```
task_egid:long(task:long)
```

引数

task

task_struct ポインター

説明

この関数は、指定タスクの実効グループID を返します。

名前

function::task_euid – タスクの実効ユーザー識別子。

概要

```
task_euid:long(task:long)
```

引数

task

task_struct ポインター

説明

この関数は、指定タスクの実効ユーザーID を返します。

名前

function::task_exe_file – タスクの実行可能ファイルのファイル構造体ポインターを取得します

概要

```
task_exe_file:long(task:long)
```

引数

task

task_struct ポインター

名前

function::task_execname – タスクの名前。

概要

```
task_execname:string(task:long)
```

引数

task

task_struct ポインター

説明
指定タスクの名前を返します。

名前
function::task_fd_lookup – タスクの fd のファイル構造体を取得する

概要

```
task_fd_lookup:long(task:long,fd:long)
```

引数

task

task_struct ポインター

fd

ファイル記述子番号。

説明
タスクのファイル記述子のファイル構造体ポインターを返します。

名前
function::task_gid – タスクのグループ識別子。

概要

```
task_gid:long(task:long)
```

引数

task

task_struct ポインター

説明
この関数は、指定タスクのグループ ID を返します。

名前
function::task_max_file_handles – タスクのオープンファイルの最大数。

概要

```
task_max_file_handles:long(task:long)
```

引数

task

task

task_struct ポインター

説明

この関数は、指定タスクのファイルハンドラーの最大数を返します。

名前

function::task_nice – タスクの nice 値。

概要

```
task_nice:long(task:long)
```

引数

task

task_struct ポインター

説明

この関数は、指定タスクの nice 値を返します。

名前

function::task_ns_egid – タスクの実効グループ識別子。

概要

```
task_ns_egid:long(task:long)
```

引数

task

task_struct ポインター

説明

この関数は、指定タスクの実効グループ ID を返します。

名前

function::task_ns_euid – タスクの実効ユーザー識別子。

概要

```
task_ns_euid:long(task:long)
```

引数

task

task

task_struct ポインター

説明

この関数は、指定タスクの実効ユーザー ID を返します。

名前

function::task_ns_gid — 名前空間で見られるタスクのグループ識別子

概要

```
task_ns_gid:long(task:long)
```

引数

task

task_struct ポインター

説明

この関数は、指定されたユーザー名前空間で見られる、指定されたタスクのグループ ID を返します。

名前

function::task_ns_pid — タスクのプロセス識別子。

概要

```
task_ns_pid:long(task:long)
```

引数

task

task_struct ポインター

説明

この関数は、指定された pid 名前空間に基づいて、指定されたタスクのプロセス ID を返します。

名前

function::task_ns_tid — 名前空間で見られるタスクのスレッド識別子

概要

```
task_ns_tid:long(task:long)
```

引数

task

task_struct ポインター

説明

この関数は、pid 名前空間で見られるように、指定されたタスクのスレッド ID を返します。

名前

function::task_ns_uid – タスクのユーザー識別子。

概要

```
task_ns_uid:long(task:long)
```

引数

task

task_struct ポインター

説明

この関数は、指定タスクのユーザー ID を返します。

名前

function::task_open_file_handles – タスクのオープンファイルの数。

概要

```
task_open_file_handles:long(task:long)
```

引数

task

task_struct ポインター

説明

この関数は、指定タスクのオープンファイルハンドラーの数を返します。

名前

function::task_parent – 親タスクの task_struct。

概要

```
task_parent:long(task:long)
```

引数

task

task_struct ポインター

説明

この関数は、指定タスクの親 task_struct を返します。このアドレスは、タスク固有のデータを抽出するためにさまざまな task_*() 関数に渡すことができます。

名前

function::task_pid — タスクのプロセス識別子。

概要

```
task_pid:long(task:long)
```

引数

task

task_struct ポインター

説明

この関数は、指定タスクのプロセス ID を返します。

名前

function::task_prio — タスクの優先度の値。

概要

```
task_prio:long(task:long)
```

引数

task

task_struct ポインター

説明

この関数は、指定タスクの優先度の値を返します。

名前

function::task_state — タスクの状態。

概要

```
task_state:long(task:long)
```

引数

task

task_struct ポインター

説明

指定タスクの状態を返します (TASK_RUNNING (0)、TASK_INTERRUPTIBLE (1)、TASK_UNINTERRUPTIBLE (2)、TASK_STOPPED (4)、TASK_TRACED (8)、EXIT_ZOMBIE (16)、EXIT_DEAD (32) のいずれか)。

名前

function::task_tid – タスクのスレッド識別子。

概要

```
task_tid:long(task:long)
```

引数

task

task_struct ポインター

説明

この関数は、指定タスクのスレッド ID を返します。

名前

function::task_uid – タスクのユーザー識別子。

概要

```
task_uid:long(task:long)
```

引数

task

task_struct ポインター

説明

この関数は、指定タスクのユーザー ID を返します。

名前

function::tid – ターゲットプロセスのスレッド ID を返します。

概要

```
tid:long()
```

引数
なし
説明
この関数はターゲットプロセスのスレッドID を返します。

名前
function::u32_arg – 関数の引数を符号なし 32 ビット値として返す

概要

```
u32_arg:long(n:long)
```

引数

n

返す引数のインデックス

説明
uint_arg と同じ、引数 n の符号なし 32 ビット値を返します。

名前
function::u64_arg – 関数の引数を符号なし 64 ビット値として返す

概要

```
u64_arg:long(n:long)
```

引数

n

返す引数のインデックス

説明
ulonglong_arg と同様に、引数 n の符号なし 64 ビット値を返します。

名前
function::u_register – 指定された CPU レジスタの符号なし値を返します

概要

```
u_register:long(name:string)
```

引数

name

返すレジスタの名前

説明

register (name) と同じですが、レジスタが 32 ビット幅の場合は 64 ビットにゼロ拡張されます。

名前

function::uaddr — 現在実行しているタスクのユーザー空間アドレス

概要

```
uaddr:long()
```

引数

なし

説明

*プローブの発生時に現在のタスクが存在していたユーザー空間のアドレスを返します。現在実行中のタスクがユーザー空間スレッドではない場合やアドレスが見つからない場合は、ゼロが返されます。現在のタスクが **usymname** または **usymdata** と組み合わされる場所を確認するために使用できます。多くのタスクは、カーネルを入力した VDSO にあります。*

名前

function::ubacktrace — 現在のユーザー空間タスクスタックの 16 進バックトレース。

概要

```
ubacktrace:string()
```

引数

なし

説明

*現在のタスクのスタックのバックトレースである 16 進アドレスの文字列を返します。出力は文字列の最大長に従って切り捨てられることがあります。現在のプローブポイントがユーザーのバックトレースを判断できない場合は空の文字列を返します。見る**backtrace**カーネルのトレースバック用。*

注記

ユーザー空間アプリケーションと現在のスクリプトに記述されていない共有ライブラリーの (完全な) バックトレースを取得するには、stap を -d /path/to/exe-or-so で実行するか、または --ldd を追加して必要なすべてのアンワインドデータをロードします。

名前

function::ucallers — ユーザースタックバックトレースの最初の n 個の要素を返す

概要

```
ucallers:string(n:long)
```

引数

n

スタック内で下降するレベルの数 (トップレベルは数えません)。n が -1 の場合、スタック全体を出力します。

説明

この関数は、ユーザースタックのバックトレースから最初の n 個の 16 進アドレスの文字列を返します。出力は文字列の最大長 (MAXSTRINGLEN) に従って切り捨てられることがあります。

注記

ユーザー空間アプリケーションと現在のスクリプトに記述されていない共有ライブラリーの (完全な) バックトレースを取得するには、stap を -d /path/to/exe-or-so で実行するか、または --ldd を追加して必要なすべてのアンワインドデータをロードします。

名前

function::uid – ターゲットプロセスのユーザー ID を返します。

概要

```
uid:long()
```

引数

なし

説明

この関数は、ターゲットプロセスのユーザー ID を返します。

名前

function::uint_arg – 関数の引数を unsigned int として返す

概要

```
uint_arg:long(n:long)
```

引数

n

返す引数のインデックス

説明

引数 n の値を unsigned int (つまり、64 ビットにゼロ拡張された 32 ビット整数) として返します。

名前

function::ulong_arg – 関数の引数を unsigned long として返す

概要

```
ulong_arg:long(n:long)
```

引数

n

返す引数のインデックス

説明

引数 n の値を unsigned long として返します。long が 32 ビットのアーキテクチャーでは、値は 64 ビットにゼロ拡張されます。

名前

function::ulonglong_arg — 関数の引数を 64 ビット値として返す

概要

```
ulonglong_arg:long(n:long)
```

引数

n

返す引数のインデックス

説明

引数 n の値を 64 ビット値として返します。(longlong_arg と同じ。)

名前

function::umodname — ユーザーモジュールの (短い) 名前を返します。

概要

```
umodname:string(addr:long)
```

引数

addr

ユーザー空間アドレス

説明

指定されたアドレスが含まれる現在のタスクのユーザー空間モジュールの短い名前を返します。アドレスが (マップされた) モジュールにない場合、または何らかの理由でモジュールが見つからない場合にエラーを報告します。

名前

function::user_mode — プローブポイントがユーザーモードで発生するかどうかを判断します。

概要

```
user_mode:long()
```

引数

なし

説明

プローブポイントがユーザーモードで発生した場合は1 を返します。

名前

function::ustack – ユーザースタックバックトレースの特定の深さでアドレスを返します

概要

```
ustack:long(n:long)
```

引数

n

スタックで下降するレベル数。

説明

単純な (ユーザー空間) バックトレースを実行し、指定された位置にある要素を返します。バックトレース自体の結果はキャッシュされるため、バックトレースの計算は何度実行しても最大1 回です。**ustack**呼び出されるか、またはどの順序で呼び出されます。

名前

function::usymdata – アドレスのシンボルとモジュールオフセットを返します。

概要

```
usymdata:string(addr:long)
```

引数

addr

変換するアドレス。

説明

既知の場合は、現在のタスクの指定のアドレスに関連する (関数) シンボル名、開始からのオフセット、シンボルのサイズ、およびモジュール名 (括弧間) を返します。シンボルが不明でもモジュールが分かっている場合は、モジュール内のオフセットとモジュールのサイズが追加されます。要素が不明な場合は省略され、シンボル名が不明な場合は、指定のアドレスの16 進文字列を返します。

名前

function::usymfile – 指定されたアドレスのファイル名を返します。

概要

```
usymfile:string(addr:long)
```

引数

addr

変換するアドレス。

説明

既知の場合、指定されたアドレスのファイル名を返します。ファイル名が見つからない場合は、アドレスの 16 進文字列表現が返されます。

名前

function::usymfileline — アドレスのファイル名と行番号を返します。

概要

```
usymfileline:string(addr:long)
```

引数

addr

変換するアドレス。

説明

既知の場合、指定されたアドレスのファイル名と (おおよその) 行番号を返します。ファイル名または行番号が見つからない場合は、アドレスの 16 進文字列表現が返されます。

名前

function::usymline — アドレスの行番号を返します。

概要

```
usymline:string(addr:long)
```

引数

addr

変換するアドレス。

説明

既知の場合、指定されたアドレスの (おおよその) 行番号を返します。行番号が見つからない場合は、アドレスの 16 進文字列表現が返されます。

名前

function::usymname — 現在のタスクでのアドレスのシンボルを返します。

概要

```
usymname:string(addr:long)
```

引数

addr

変換するアドレス。

説明

既知の場合、指定のアドレスに関連する (関数) シンボル名を返します。分からない場合は、addr の 16 進文字列表記を返します。

第4章 タイムスタンプ関数

各タイムスタンプ関数は、関数が実行されるタイミングを示す値を返します。返された値は、イベント発生時、イベントの順序付けの指定、または 2 つのタイムスタンプ間で経過した時間の算出に使用することができます。

名前
function::HZ – Kernel HZ

概要

```
HZ:long()
```

引数
なし
説明
この関数は、jiffies 値の増加率に対応するカーネル HZ マクロの値を返します。

名前
function::cpu_clock_ms – 指定された CPU のクロックのミリ秒数

概要

```
cpu_clock_ms:long(cpu:long)
```

引数

cpu

読み取るプロセッサーのクロック

説明
この関数は、指定された CPU のクロックのミリ秒数を返します。これは、同じ CPU での比較では常に単調ですが、CPU 間で多少のずれが生じる可能性があります (約 1 時間以内)。

名前
function::cpu_clock_ns – 指定された CPU のクロックのナノ秒数

概要

```
cpu_clock_ns:long(cpu:long)
```

引数

cpu

読み取るプロセッサーのクロック

説明

この関数は、指定された CPU のクロックのナノ秒数を返します。これは、同じ CPU での比較では常に単調ですが、CPU 間で多少のずれが生じる可能性があります (約1 時間以内)。

名前

function::cpu_clock_s – 指定された CPU のクロックの秒数

概要

```
cpu_clock_s:long(cpu:long)
```

引数

cpu

読み取るプロセッサーのクロック

説明

この関数は、指定された CPU のクロックの秒数を返します。これは、同じ CPU での比較では常に単調ですが、CPU 間で多少のずれが生じる可能性があります (約1 時間以内)。

名前

function::cpu_clock_us – 指定された CPU のクロックのマイクロ秒数

概要

```
cpu_clock_us:long(cpu:long)
```

引数

cpu

読み取るプロセッサーのクロック

説明

この関数は、指定された CPU のクロックのマイクロ秒数を返します。これは、同じ CPU での比較では常に単調ですが、CPU 間で多少のずれが生じる可能性があります (約1 時間以内)。

名前

function::delete_stopwatch – 既存のストップウォッチを削除する

概要

```
delete_stopwatch(name:string)
```

引数

name

ストップウォッチの名前

説明

*ストップウォッチ **名** を削除します。*

名前

function::get_cycles – プロセッサーサイクル数

概要

```
get_cycles:long()
```

引数

なし

説明

この関数はプロセッサーサイクルカウンター値を返します (ある場合)。それ以外の場合はゼロを返します。サイクルカウンターは、各プロセッサーでフリーランで、同期されていない状態です。そのため、異なるプロセッサーの get_cycles 関数の結果を比較することでは、イベントの順序を判断できません。

名前

function::gettimeofday_ms – UNIX エポックからの経過時間 (ミリ秒)

概要

```
gettimeofday_ms:long()
```

引数

なし

説明

この関数は、UNIX エポックからの経過時間をミリ秒数で返します。

名前

function::gettimeofday_ns – UNIX エポックからの経過時間 (ナノ秒)

概要

```
gettimeofday_ns:long()
```

引数

なし

説明

この関数は、UNIX エポックからの経過時間をナノ秒数で返します。

名前

function::gettimeofday_s – UNIX エポックからの経過時間 (秒)

概要

```
gettimeofday_s:long()
```

引数

なし

説明

この関数は、UNIX エポックからの経過時間を秒数で返します。

名前

function::gettimeofday_us – UNIX エポックからの経過時間 (マイクロ秒)

概要

```
gettimeofday_us:long()
```

引数

なし

説明

この関数は、UNIX エポックからの経過時間をマイクロ秒数で返します。

名前

function::jiffies – カーネル jiffies カウント

概要

```
jiffies:long()
```

引数

なし

説明

*この関数は、カーネル jiffies 変数の値を返します。この値は、タイマー割り込みによって定期的にインクリメントされ、32 ビットまたは 64 ビット境界をラップする場合があります。**HZ** を参照してください。*

名前

function::local_clock_ms – ローカル CPU のクロックのミリ秒数

概要

```
local_clock_ms:long()
```

引数

なし

説明
この関数は、ローカル CPU のクロックのミリ秒数を返します。これは、同じ CPU での比較では常に単調ですが、CPU 間で多少のずれが生じる可能性があります (約1 時間以内)。

名前
function::local_clock_ns – ローカル CPU のクロックのナノ秒数

概要

```
local_clock_ns:long()
```

引数
なし
説明
この関数は、ローカル CPU のクロックのナノ秒数を返します。これは、同じ CPU での比較では常に単調ですが、CPU 間で多少のずれが生じる可能性があります (約1 時間以内)。

名前
function::local_clock_s – ローカル CPU のクロックの秒数

概要

```
local_clock_s:long()
```

引数
なし
説明
この関数は、ローカル CPU のクロックの秒数を返します。これは、同じ CPU での比較では常に単調ですが、CPU 間で多少のずれが生じる可能性があります (約1 時間以内)。

名前
function::local_clock_us – ローカル CPU のクロックのマイクロ秒数

概要

```
local_clock_us:long()
```

引数
なし
説明
この関数は、ローカル CPU のクロックのマイクロ秒数を返します。これは、同じ CPU での比較では常に単調ですが、CPU 間で多少のずれが生じる可能性があります (約1 時間以内)。

名前
function::read_stopwatch_ms – ストップウォッチのミリ秒単位の時間を読み取ります

概要

```
read_stopwatch_ms:long(name:string)
```

引数

name

ストップウォッチ名

説明

stopwatch **name** の時間をミリ秒単位で返します。現在存在しない場合は、ストップウォッチ **名** を作成します。

名前

function::read_stopwatch_ns – ストップウォッチのナノ秒単位の時間を読み取ります

概要

```
read_stopwatch_ns:long(name:string)
```

引数

name

ストップウォッチ名

説明

stopwatch **name** の時間をナノ秒単位で返します。現在存在しない場合は、ストップウォッチ **名** を作成します。

名前

function::read_stopwatch_s – ストップウォッチの時間を秒単位で読み取ります

概要

```
read_stopwatch_s:long(name:string)
```

引数

name

ストップウォッチ名

説明

ストップウォッチ **name** の時間を秒単位で返します。現在存在しない場合は、ストップウォッチ **名** を作成します。

名前
function::read_stopwatch_us – ストップウォッチのマイクロ秒単位の時間を読み取ります

概要

```
read_stopwatch_us:long(name:string)
```

引数

name
ストップウォッチ名

説明
stopwatch **name** の時間をマイクロ秒単位で返します。現在存在しない場合は、ストップウォッチ **名** を作成します。

名前
function::start_stopwatch – ストップウォッチを開始する

概要

```
start_stopwatch(name:string)
```

引数

name
ストップウォッチの名前

説明
ストップウォッチ **名** を開始します。現在存在しない場合は、ストップウォッチ **名** を作成します。

名前
function::stop_stopwatch – ストップウォッチを止める

概要

```
stop_stopwatch(name:string)
```

引数

name
ストップウォッチの名前

説明
ストップウォッチ **名** .現在存在しない場合は、ストップウォッチ **名** を作成します。

第5章 時間効用関数

ユーティリティー関数は、タイムスタンプ関数 gettimeofday_s() によって返されるエポックからの経過時間 (秒数) を、人が判読できる日付/時間の文字列に変換します。

名前

function::ctime – エポックからの経過時間 (秒単位) を、人が判読できる日付/時間の文字列に変換します。

概要

```
ctime:string(epochsecs:long)
```

引数

epochsecs

エポックからの経過時間 (**gettimeofday_s** によって返された場合)。

説明

gettimeofday_s によって返されるエポックからの経過時間 (秒単位) の引数を取ります。フォームの文字列を返します。
「Wed Jun 30 21:49:08 1993」
文字列は常に 24 文字ちょうどになります。時間がかなり過去 (エポックからの 32 ビットオフセットで表すことができる秒数よりも前) の場合、エラーが発生します (これは、try/catch で回避できます)。時間が無理に遠い未来の場合も、エラーが発生します。
エポック (ゼロ) は以下になります。
「Thu Jan 1 00:00:00 1970」
epochsecs -2147483648 に対応する、ctime によって提供される最も初期の日付は、「Fri Dec 13 20:45:52 1901」 です。epochsecs 2147483647 に対応する ctime によって提供される最新の日付は 「Tue Jan 19 03:14:07 2038」 です。
曜日の省略形は 'Sun'、'Mon'、'Tue'、'Wed'、'Thu'、'Fri'、および 'Sat' です。月の省略形は 'Jan'、'Feb'、'Mar'、'Apr'、'Jun'、'Jul'、'Aug'、'Sep'、'Oct'、'Nov'、および 'Dec' です。
実際の C ライブラリー **ctime** 関数は、この関数が文字列の最後に改行文字 (\n) を挿入することに注意してください。また、カーネルにはタイムゾーンの概念がないため、時間は常に GMT で返されます。

名前

function::tz_ctime – エポックからの秒数を、ローカルタイムゾーンを使用して人間が読める日付/時刻文字列に変換します

概要

```
tz_ctime(epochsecs:)
```

引数

epochsecs

エポックからの経過時間 (**gettimeofday_s** によって返された場合)。

説明

gettimeofday_s *によって返されるエポックからの経過時間 (秒単位) の引数を取ります。と同じ形式の文字列を返します* ***ctime*** *、ただし、ローカルタイムゾーンのエポックタイムをオフセットし、ローカルタイムゾーンの名前を追加します。紐の長さが異なる場合がございます。タイムゾーン情報は、スクリプトの起動時にのみ staprun によって渡されます。*

名前

function::tz_gmtoff – ローカルタイムゾーンのオフセットを返す

概要

```
tz_gmtoff()
```

引数

なし

説明

スクリプトの起動時にのみ staprun によって渡されるローカルタイムゾーンオフセット (UTC から西への秒数) を返します。

名前

function::tz_name – ローカルタイムゾーン名を返す

概要

```
tz_name()
```

引数

なし

説明

スクリプトの起動時にのみ staprun によって渡されるローカルタイムゾーン名を返します。

第6章 シェルコマンド関数

シェルコマンドをキューに入れるためのユーティリティー関数。

名前

function::system – システムにコマンドを発行する

概要

```
system(cmd:string)
```

引数

cmd

システムに発行するコマンド

説明

この関数は、システムでコマンドを実行します。コマンドは、現在のプローブが完了した後、バックグラウンドで開始されます。コマンドは、stap または staprun コマンドを実行しているユーザーと同じ UID で実行されます。

第7章 メモリー TAPSET

この種類のプローブポイントは、メモリー関連のイベントをプローブしたり、現在のプロセスのメモリー使用量を照会するために使用されます。これには、以下のプローブポイントが含まれます。

名前

function::addr_to_node – NUMA システム内で指定のアドレスが属するノードを返します。

概要

```
addr_to_node:long(addr:long)
```

引数

addr

障害が発生しているメモリーアクセスのアドレス

説明

この関数はアドレスを受け取り、NUMA システム内で指定アドレスが属するノードを返します。

名前

function::bytes_to_string – 指定バイトの人が判読できる文字列。

概要

```
bytes_to_string:string(bytes:long)
```

引数

bytes

変換するバイト数。

説明

バイト数 (最大 1024 バイト)、'K' が末尾のキロバイト数 (1024K 未満)、'M' が末尾のメガバイト数 (1024M 未満)、または 'G' が末尾のギガバイト数を表す文字列を返します。K、M、または G が付き、数値が 100 未満の場合は、. と、その残りが含まれます。返される文字列は 5 文字です (9999G バイトを超える場合や負の値である場合以外は最初に空白文字が挿入されます)。

名前

function::mem_page_size – このアーキテクチャーのページのバイト数。

概要

```
mem_page_size:long()
```

引数

なし

名前
function::pages_to_string – ページを人が判読できる文字列に変換します。

概要

```
pages_to_string:string(pages:long)
```

引数

pages

変換するページ数。

説明
ページを **page_size** で乗算してバイト数を取得し、**bytes_to_string** の結果を返します。

名前
function::proc_mem_data – ページ単位のプログラムデータサイズ (データ + スタック)。

概要

```
proc_mem_data:long()
```

引数
なし
説明
ページ単位の現在のプロセスデータサイズ (データ + スタック) を返します。現在のプロセスがない場合や、ページ数を取得できなかった場合はゼロを返します。

名前
function::proc_mem_data_pid – ページ単位のプログラムデータサイズ (データ + スタック)。

概要

```
proc_mem_data_pid:long(pid:long)
```

引数

pid

確認するプロセスの PID。

説明
ページ単位の指定のプロセスデータサイズ (データ + スタック) を返します。プロセスがない場合や、ページ数を取得できなかった場合はゼロを返します。

名前
function::proc_mem_rss – ページ単位のプログラムレジデント設定サイズ。

概要

```
proc_mem_rss:long()
```

引数
なし
説明
現在のプロセスのページ単位のレジデント設定サイズを返します。現在のプロセスがない場合や、ページ数を取得できなかった場合はゼロを返します。

名前
function::proc_mem_rss_pid – ページ単位のプログラムレジデント設定サイズ。

概要

```
proc_mem_rss_pid:long(pid:long)
```

引数

pid

確認するプロセスのPID。

説明
指定プロセスのページ単位のレジデント設定サイズを返します。指定のプロセスがない場合や、ページ数を取得できなかった場合はゼロを返します。

名前
function::proc_mem_shr – プログラム共有ページ(共有マッピングより)。

概要

```
proc_mem_shr:long()
```

引数
なし
説明
現在のプロセスの共有ページ(共有マッピングより) を返します。現在のプロセスがない場合や、ページ数を取得できなかった場合はゼロを返します。

名前
function::proc_mem_shr_pid – プログラム共有ページ(共有マッピングより)。

概要

```
proc_mem_shr_pid:long(pid:long)
```

引数

pid

確認するプロセスのPID。

説明

指定プロセスの共有ページ(共有マッピングより) を返します。プロセスがない場合や、ページ数を取得できなかった場合はゼロを返します。

名前

function::proc_mem_size – ページ単位のプログラム仮想メモリーサイズの合計。

概要

```
proc_mem_size:long()
```

引数

なし

説明

現在のプロセスのページ単位の仮想メモリーサイズ合計を返します。現在のプロセスがない場合や、ページ数を取得できなかった場合はゼロを返します。

名前

function::proc_mem_size_pid – ページ単位のプログラム仮想メモリーサイズの合計。

概要

```
proc_mem_size_pid:long(pid:long)
```

引数

pid

確認するプロセスのPID。

説明

指定プロセスのページ単位の仮想メモリーサイズ合計を返します。指定のプロセスがない場合や、ページ数を取得できなかった場合はゼロを返します。

名前

function::proc_mem_string – 人が判読できる文字列で示された現在のproc メモリー使用量。

概要

```
proc_mem_string:string()
```

引数
なし
説明
現在のプロセスによって使用されるメモリーのサイズ、rss、shr、txt、およびデータを表示する、人が判読できる文字列を返します。例: 「size: 301m, rss: 11m, shr: 8m, txt: 52k, data: 2248k 」

名前
function::proc_mem_string_pid – 人が判読できる文字列で示されたプロセスメモリー使用量。

概要

```
proc_mem_string_pid:string(pid:long)
```

引数

pid

確認するプロセスのPID。

説明
指定のプロセスによって使用されるメモリーのサイズ、rss、shr、txt、およびデータを表示する、人が判読できる文字列を返します。例: 「size: 301m, rss: 11m, shr: 8m, txt: 52k, data: 2248k 」

名前
function::proc_mem_txt – ページ単位のプログラムテキスト (コード) サイズ。

概要

```
proc_mem_txt:long()
```

引数
なし
説明
ページ単位の現在のプロセステキスト (コード) サイズを返します。現在のプロセスがない場合や、ページ数を取得できなかった場合はゼロを返します。

名前
function::proc_mem_txt_pid – ページ単位のプログラムテキスト (コード) サイズ。

概要

```
proc_mem_txt_pid:long(pid:long)
```

引数

pid

確認するプロセスのPID。

説明

ページ単位の指定のプロセステキスト(コード)サイズを返します。プロセスがない場合や、ページ数を取得できなかった場合はゼロを返します。

名前

function::vm_fault_contains – ページフォールトの理由の戻り値をテストします。

概要

```
vm_fault_contains:long(value:long,test:long)
```

引数

value

vm.page_fault.return によって返される fault_type

test

テストするフォールトタイプ(VM_FAULT_OOM または同等)

名前

probe::vm.brk – brk が要求されたときに実行されます(例: ヒープのリサイズ)。

概要

```
vm.brk
```

値

name

プローブポイントの名前

address

要求されたアドレス

長さ

メモリーセグメントの長さ

コンテキスト

brk を呼び出すプロセス。

名前

probe::vm.kfree – kfree が要求されたときに実行されます。

概要

```
vm.kfree
```

値

name

プローブポイントの名前

ptr

kmalloc によって返された割り当て済み kmemory へのポインター。

caller_function

呼び出し元関数の名前。

call_site

この kmemory 関数を呼び出す関数のアドレス。

名前

probe::vm.kmalloc – kmalloc が要求されたときに実行されます。

概要

```
vm.kmalloc
```

値

gfp_flags

割り当てる kmemory のタイプ。

bytes_req

要求されたバイト。

name

プローブポイントの名前

ptr

割り当てられた kmemory へのポインター

bytes_alloc

割り当てられたバイト。

caller_function

呼び出し元関数の名前。

gfp_flag_name

割り当てる kmemory のタイプ(文字列形式)。

call_site

kmemory 関数のアドレス。

名前

probe::vm.kmalloc_node – kmalloc_node が要求されたときに実行されます。

概要

```
vm.kmalloc_node
```

値

caller_function

呼び出し元関数の名前。

gfp_flag_name

割り当てる kmemory のタイプ(文字列形式)。

call_site

この kmemory 関数を呼び出す関数のアドレス。

gfp_flags

割り当てる kmemory のタイプ。

bytes_req

要求されたバイト。

name

プローブポイントの名前

ptr

割り当てられた kmemory へのポインター

bytes_alloc

割り当てられたバイト。

名前

probe::vm.kmem_cache_alloc – kmem_cache_alloc が要求されたときに発生します

概要

```
vm.kmem_cache_alloc
```

値

bytes_alloc

割り当てられたバイト。

ptr

割り当てられた kmemory へのポインター

name

プローブポイントの名前

bytes_req

要求されたバイト。

gfp_flags

割り当てる kmemory のタイプ。

caller_function

呼び出し元関数の名前。

gfp_flag_name

割り当てる kmemory のタイプ(文字列形式)。

call_site

この kmemory 関数を呼び出す関数のアドレス。

名前

probe::vm.kmem_cache_alloc_node – kmem_cache_alloc_node が要求されたときに発生します

概要

```
vm.kmem_cache_alloc_node
```

値

gfp_flags

割り当てる kmemory のタイプ。

name

プローブポイントの名前

bytes_req

要求されたバイト。

ptr

割り当てられた kmemory へのポインター

bytes_alloc

割り当てられたバイト。

caller_function

呼び出し元関数の名前。

call_site

この kmemory 関数を呼び出す関数のアドレス。

gfp_flag_name

割り当てる kmemory のタイプ(文字列形式)。

名前

probe::vm.kmem_cache_free – kmem_cache_free が要求されたときに発生します

概要

```
vm.kmem_cache_free
```

値

caller_function

呼び出し元関数の名前。

call_site

この kmemory 関数を呼び出す関数のアドレス。

ptr

kmem_cache によって返された割り当て済み kmemory へのポインター。

name

プローブポイントの名前

名前
probe::vm.mmap — mmap が要求されたときに実行されます。

概要

```
vm.mmap
```

値

name
プローブポイントの名前

長さ
メモリーセグメントの長さ

address
要求されたアドレス

コンテキスト
mmap を呼び出すプロセス。

名前
probe::vm.munmap — munmap が要求されたときに実行されます。

概要

```
vm.munmap
```

値

長さ
メモリーセグメントの長さ

address
要求されたアドレス

name
プローブポイントの名前

コンテキスト
munmap を呼び出すプロセス。

名前
probe::vm.oom_kill — OOM Killer によって終了されるスレッドが選択されたときに実行されます。

概要

vm.oom_kill

値

name
プローブポイントの名前

task
kill されるタスク。

コンテキスト
余分なメモリーを消費しようとし、OOM を引き起こしたプロセス。

名前
probe::vm.pagefault — ページフォールトの発生を記録します。

概要

vm.pagefault

値

address
障害が発生しているメモリーアクセスのアドレス (例: ページフォールトの原因となったアドレス)。

write_access
読み取りまたは書き込みアクセスであるかを示します。1 は書き込み、0 は読み取りを示します。

name
プローブポイントの名前

コンテキスト
障害を引き起こしたプロセス。

名前
probe::vm.pagefault.return — 発生した障害のタイプを示します。

概要

vm.pagefault.return

値

name

プローブポイントの名前

fault_type

メモリー不足による障害は 0 (VM_FAULT_OOM)、小さな障害は 2 (VM_FAULT_MINOR)、大きな障害は 3 (VM_FAULT_MAJOR) を返します。障害がいずれにも該当しない場合は 1 (VM_FAULT_SIGBUS) を返します。

名前

probe::vm.write_shared – 共有ページへの書き込みを試行します。

概要

```
vm.write_shared
```

値

address

共有書き込みのアドレス。

name

プローブポイントの名前

コンテキスト

コンテキストは、書き込みを試行するプロセスです。

説明

プロセスが共有ページへの書き込みを試みる際に実行されます。コピーが必要な場合は、その後に vm.write_shared_copy が続きます。

名前

probe::vm.write_shared_copy – 共有ページ書き込みのページコピー。

概要

```
vm.write_shared_copy
```

値

zero

ゼロページであるかどうかを示すブール値 (コピーの代わりに消去を実行可能)。

name

プローブポイントの名前

address

共有書き込みのアドレス。

コンテキスト

書き込みを試行するプロセス。

説明

共有ページへの書き込みでページのコピーが必要な場合に実行されます。これには、常に vm.write_shared が先行します。

第8章 タスク時間 TAPSET

この tapset は、ユーティリティー関数を定義して現在のタスクの時間に関するプロパティーを問い合わせ、これらをミリ秒単位で人が判読できる文字列に変換します。

名前

function::cputime_to_msecs — 指定の CPU 時間をミリ秒に変換します。

概要

```
cputime_to_msecs:long(cputime:long)
```

引数

cputime

ミリ秒に変換する時間。

名前

function::cputime_to_string — 人が判読できる文字列で示された指定の CPU 時間。

概要

```
cputime_to_string:string(cputime:long)
```

引数

cputime

変換する時間。

説明

msec_to_string (cputime_to_msecs (cputime) の呼び出しと同等です。

名前

function::cputime_to_usecs — 指定された cputime をマイクロ秒に変換します

概要

```
cputime_to_usecs:long(cputime:long)
```

引数

cputime

マイクロ秒に変換する時間。

名前
function::msecs_to_string – 人が判読できる文字列で示された指定のミリ秒。

概要

```
msecs_to_string:string(msecs:long)
```

引数

msecs
変換するミリ秒数。

説明
ミリ秒数を人が判読できる文字列で返します。この文字列の形式は 「XmY.ZZZs」 で、X は分数、Y は秒数、ZZZ はミリ秒数を示します。

名前
function::nsecs_to_string – 与えられたナノ秒の人間が読める文字列

概要

```
nsecs_to_string:string(nsecs:long)
```

引数

nsecs
変換するナノ秒数。

説明
「XmY.ZZZZZZs」 で設定される人間が読める文字列としてナノ秒数を表す文字列を返します。ここで、X は分数、Y は秒数、ZZZZZZZZZ はナノ秒数です。

名前
function::task_start_time – 指定されたタスクの開始時刻

概要

```
task_start_time:long(tid:long)
```

引数

tid()
指定タスクのスレッド ID。

説明

指定されたタスクの起動時刻からのナノ秒単位の開始時刻を返します。タスクが存在しない場合は 0 を返します。

名前

function::task_stime – 現在のタスクのシステム時間。

概要

```
task_stime:long()
```

引数

なし

説明

現在のタスクのシステム時間を cputime で返します。このプロセスの他のタスクが使用した時間や、このタスクの子が費やした時間は含まれません。

名前

function::task_stime_tid – 指定タスクのシステム時間。

概要

```
task_stime_tid:long(tid:long)
```

引数

tid()

指定タスクのスレッド ID。

説明

指定タスクのシステム時間を cputime で返します。タスクが存在しない場合はゼロを返します。このプロセスの他のタスクが使用した時間や、このタスクの子が費やした時間は含まれません。

名前

function::task_time_string – 人が判読できる文字列で示されたタスク時間の使用量。

概要

```
task_time_string:string()
```

引数

なし

説明

現在のタスクがこれまでに使用したユーザーおよびシステム時間を示す、人が判読可能な文字列を返します。例: 「usr: 0m12.908s, sys: 1m6.851s」

名前
function::task_time_string_tid – 人が判読できる文字列で示されたタスク時間の使用量。

概要

```
task_time_string_tid:string(tid:long)
```

引数

tid()
指定タスクのスレッドID。

説明
指定のタスクがこれまでに使用したユーザーおよびシステム時間を示す、人が判読可能な文字列を返します。例: 「usr: 0m12.908s, sys: 1m6.851s」

名前
function::task_utime – 現在のタスクのユーザー時間。

概要

```
task_utime:long()
```

引数
なし
説明
現在のタスクのユーザー時間を cputime で返します。このプロセスの他のタスクが使用した時間や、このタスクの子が費やした時間は含まれません。

名前
function::task_utime_tid – 指定タスクのユーザー時間。

概要

```
task_utime_tid:long(tid:long)
```

引数

tid()
指定タスクのスレッドID。

説明
指定タスクのユーザー時間を cputime で返します。タスクが存在しない場合はゼロを返します。このプロセスの他のタスクが使用した時間や、このタスクの子が費やした時間は含まれません。

名前

function::usecs_to_string – 与えられたマイクロ秒の人間が読める文字列

概要

```
usecs_to_string:string(usecs:long)
```

引数

usecs

変換するマイクロ秒数。

説明

ミリ秒数を人が判読できる文字列で返します。この文字列の形式は 「XmY.ZZZs」 で、X は分数、Y は秒数、ZZZ はミリ秒数を示します。

第9章 スケジューラータップセット

この一連のプローブポイントは、タスクスケジューラーのアクティビティーをプローブするために使用されます。以下のプローブポイントが含まれます。

名前

probe::scheduler.balance – より多くの作業を見つけようとしている CPU。

概要

```
scheduler.balance
```

値

name

プローブポイントの名前

コンテキスト

より多くの仕事を探している CPU。

名前

probe::scheduler.cpu_off – プロセスが CPU での実行を停止しようとしています

概要

```
scheduler.cpu_off
```

値

task_prev

CPU を離れるプロセス (現在と同じ)

idle

現在のプロセスがアイドル状態かどうかを示すブール値

name

プローブポイントの名前

task_next

現在のものを置き換えるプロセス

コンテキスト

CPU を離れるプロセス。

名前

probe::scheduler.cpu_on — プロセスは CPU で実行を開始しています

概要

```
scheduler.cpu_on
```

値

idle

- 現在のプロセスがアイドルプロセスかどうかを示すブール値

task_prev

この CPU で以前に実行されていたプロセス

name

プローブポイントの名前

コンテキスト

再開プロセス。

名前

probe::scheduler.ctxswitch — コンテキストスイッチが発生しています。

概要

```
scheduler.ctxswitch
```

値

prev_tid

スイッチアウトするプロセスの TID

name

プローブポイントの名前

next_tid

切り替えられるプロセスの TID

prev_pid

切り替えるプロセスの PID

prevtsk_state

切り替えられるプロセスの状態

next_pid

切り替えるプロセスの PID

nexttsk_state
切り替えられるプロセスの状態

prev_priority
切り替えるプロセスの優先度

next_priority
切り替えるプロセスの優先度

prev_task_name
切り替えるプロセスの名前

next_task_name
切り替えるプロセスの名前

名前

probe::scheduler.kthread_stop – kthread_create によって作成されたスレッドが停止中です

概要

```
scheduler.kthread_stop
```

値

thread_pid
停止中のスレッドのPID

thread_priority
スレッドの優先度

名前

probe::scheduler.kthread_stop.return – kthread が停止し、戻り値を取得します

概要

```
scheduler.kthread_stop.return
```

値

return_value
スレッド停止後の戻り値

name

プローブポイントの名前

名前

probe::scheduler.migrate – CPU 間で移行中のタスク

概要

```
scheduler.migrate
```

値

priority

移行されるタスクの優先度

cpu_to

宛先 CPU

cpu_from

元の CPU

task

移行中のプロセス

name

プローブポイントの名前

pid

移行中のタスクの PID

名前

probe::scheduler.process_exit – プロセスの終了

概要

```
scheduler.process_exit
```

値

name

プローブポイントの名前

pid

終了するプロセスの PID

priority
終了するプロセスの優先度

名前
probe::scheduler.process_fork – 分岐したプロセス

概要

```
scheduler.process_fork
```

値

name
プローブポイントの名前

parent_pid
親プロセスのPID

child_pid
子プロセスのPID

名前
probe::scheduler.process_free – プロセスのデータ構造を解放するスケジューラー

概要

```
scheduler.process_free
```

値

name
プローブポイントの名前

pid
解放されるプロセスのPID

priority
解放されるプロセスの優先度

名前
probe::scheduler.process_wait – プロセスの待機を開始するスケジューラー

概要

```
scheduler.process_wait
```

値

name
プローブポイントの名前

pid
プロセススケジューラーが待機している PID

名前
probe::scheduler.signal_send – 信号を送る

概要

```
scheduler.signal_send
```

値

pid
シグナルを送信するプロセスの pid

name
プローブポイントの名前

signal_number
信号番号

名前
probe::scheduler.tick – スケジューラーの内部ティック、プロセスのタイムスライスアカウンティングが更新されます

概要

```
scheduler.tick
```

値

idle
現在のプロセスがアイドル状態かどうかを示すブール値

name

プローブポイントの名前

コンテキスト

アカウンティングが更新されるプロセス。

名前

probe::scheduler.wait_task – タスクのスケジュール解除を待機中 (非アクティブになる)

概要

```
scheduler.wait_task
```

値

task_pid

スケジューラーが待機しているタスクの PID

name

プローブポイントの名前

task_priority

タスクの優先順位

名前

probe::scheduler.wakeup – タスクが起床

概要

```
scheduler.wakeup
```

値

task_tid

起こされているタスクの tid

task_priority

起床するタスクの優先度

task_cpu

起床中のタスクの CPU

task_pid

起床中のタスクの PID

name
プローブポイントの名前

task_state
起床中のタスクの状態

名前

probe::scheduler.wakeup_new — 新しく作成されたタスクが初めて起床

概要

```
scheduler.wakeup_new
```

値

name
プローブポイントの名前

task_state
起きたタスクの状態

task_pid
起床した新タスクの PID

task_tid
ウェイクアップされた新しいタスクの TID

task_priority
新しいタスクの優先度

task_cpu
タスクの cpu が起きました

第10章 IO スケジューラーおよびブロック IO TAPSET

この種類のプローブポイントは、ブロック IO レイヤーおよび IO スケジューラーの活動をプローブするために使用されます。以下のプローブポイントが含まれます。

名前

probe::ioblock.end – ブロック I/O 転送が完了したときに必ず実行されます。

概要

```
ioblock.end
```

値

name

プローブポイントの名前

sector

バイオ全体の開始セクター

hw_segments

物理および DMA 再マッピングハードウェア結合が実行された後のセグメント数

phys_segments

物理アドレスの合体が実行された後の、このバイオのセグメント数。

flags

以下を参照してください BIO_UPTODATE 0 I/O 完了後に OK BIO_RW_BLOCK 1 RW_AHEAD が設定され、読み取り/書き込みがブロックされますユーザーページ BIO_EOPNOTSUPP 7 はサポートされていません

devname

ブロックデバイス名

bytes_done

転送されたバイト数

error

成功時

size

合計サイズ (バイト)

idx

bio ベクトル配列へのオフセット

vcnt

この I/O 要求を設定する配列要素 (ページ、オフセット、長さ) の数を表す bio vector count

いの

マップされたファイルの i ノード番号

rw

読み取り/書き込み要求のバイナリートレース

コンテキスト

プロセスは転送が行われたことを示します。

名前

probe::ioblock.request — 汎用的なブロック I/O 要求を行ったときに必ず実行されます。

概要

```
ioblock.request
```

値

sector

バイオ全体の開始セクター

name

プローブポイントの名前

devname

ブロックデバイス名

phys_segments

物理アドレスの結合が実行された後の、この略歴のセグメント数

flags

以下を参照してください BIO_UPTODATE 0 I/O 完了後に OK BIO_RW_BLOCK 1 RW_AHEAD が設定され、読み取り/書き込みがブロックされますユーザーページ BIO_EOPNOTSUPP 7 はサポートされていません

hw_segments

物理および DMA 再マッピングハードウェア結合が実行された後のセグメント数

bdev_contains

パーティションを含むデバイスオブジェクトを指します (バイオ構造がパーティションを表す場合)

vcnt

この I/O 要求を設定する配列要素 (ページ、オフセット、長さ) の数を表す bio vector count

idx

bio ベクトル配列へのオフセット

bdev

ターゲットブロックデバイス

p_start_sect

デバイスのパーティション構造の開始セクタを指します

size

合計サイズ (バイト)

いの

マップされたファイルの i ノード番号

rw

読み取り/書き込み要求のバイナリートレース

コンテキスト

ブロック I/O 要求を行うプロセス

名前

probe::ioblock_trace.bounce – ブロック I/O 要求の少なくとも 1 つのページに対してバッファーバウンスが必要なときに必ず実行されます。

概要

```
ioblock_trace.bounce
```

値

q

この略歴がキューに入れられたリクエストキュー。

size

合計サイズ (バイト)

vcnt

この I/O 要求を設定する配列要素 (ページ、オフセット、長さ) の数を表す bio vector count

idx

*bio ベクトル配列 **phys_segments** へのオフセット - 物理アドレス結合が実行された後のこの bio 内のセグメントの数。*

bdev

ターゲットブロックデバイス

p_start_sect

デバイスのパーティション構造の開始セクタを指します

いの

マップされたファイルの i ノード番号

rw

読み取り/書き込み要求のバイナリートレース

name

プローブポイントの名前

sector

バイオ全体の開始セクター

bdev_contains

パーティションを含むデバイスオブジェクトを指します (バイオ構造がパーティションを表す場合)

devname

バッファーバウンスが必要なデバイス。

flags

以下を参照してください BIO_UPTODATE 0 I/O 完了後に OK BIO_RW_BLOCK 1 RW_AHEAD が設定され、読み取り/書き込みがブロックされますユーザーページ BIO_EOPNOTSUPP 7 はサポートされていません

bytes_done

転送されたバイト数

コンテキスト

ブロック I/O 要求を作成するプロセス。

名前

probe::ioblock_trace.end – ブロック I/O 転送が完了したときに必ず実行されます。

概要

```
ioblock_trace.end
```

値

bdev_contains

パーティションを含むデバイスオブジェクトを指します (バイオ構造がパーティションを表す場合)

flags

以下を参照してください BIO_UPTODATE 0 I/O 完了後に OK BIO_RW_BLOCK 1 RW_AHEAD が設定され、読み取り/書き込みがブロックされますユーザーページ BIO_EOPNOTSUPP 7 はサポートされていません

devname

ブロックデバイス名

bytes_done

転送されたバイト数

name

プローブポイントの名前

sector

バイオ全体の開始セクター

いの

マップされたファイルの i ノード番号

rw

読み取り/書き込み要求のバイナリートレース

size

合計サイズ (バイト)

q

この略歴がキューに入れられたリクエストキュー。

idx

*bio ベクトル配列 **phys_segments** へのオフセット - 物理アドレス結合が実行された後のこの bio 内のセグメントの数。*

vcnt

この I/O 要求を設定する配列要素 (ページ、オフセット、長さ) の数を表す bio vector count

bdev

ターゲットブロックデバイス

p_start_sect

デバイスのパーティション構造の開始セクタを指します

コンテキスト

プロセスは転送が行われたことを示します。

名前

probe::ioblock_trace.request – bio に対して汎用的なブロック I/O 要求が作成されるときに実行されます。

概要

```
ioblock_trace.request
```

値

q

この略歴がキューに入れられたリクエストキュー。

size

合計サイズ (バイト)

idx

bio ベクトル配列 **phys_segments** へのオフセット - 物理アドレス結合が実行された後のこの bio 内のセグメントの数。

vcnt

この I/O 要求を設定する配列要素 (ページ、オフセット、長さ) の数を表す bio vector count

bdev

ターゲットブロックデバイス

p_start_sect

デバイスのパーティション構造の開始セクタを指します

いの

マップされたファイルの i ノード番号

rw

読み取り/書き込み要求のバイナリートレース

name

プローブポイントの名前

sector

バイオ全体の開始セクター

bdev_contains

パーティションを含むデバイスオブジェクトを指します (バイオ構造がパーティションを表す場合)

devname

ブロックデバイス名

flags

以下を参照してください BIO_UPTODATE 0 I/O 完了後に OK BIO_RW_BLOCK 1 RW_AHEAD が設定され、読み取り/書き込みがブロックされますユーザーページ BIO_EOPNOTSUPP 7 はサポートされていません

bytes_done

転送されたバイト数

コンテキスト

ブロック I/O 要求を行うプロセス

名前

probe::ioscheduler.elv_add_request – 要求が要求キューに追加されたことを示すプローブ。

概要

```
ioscheduler.elv_add_request
```

値

rq

要求のアドレス。

q

要求キューのポインター。

elevator_name

現在有効になっている I/O エレベーターのタイプ。

disk_major

要求のディスクメジャー番号。

disk_minor

要求のディスクマイナー番号。

rq_flags

要求フラグ。

名前

probe::ioscheduler.elv_add_request.kp – 要求が要求キューに追加されたことを示す kprobe ベースのプローブ。

概要

```
ioscheduler.elv_add_request.kp
```

値

disk_major
要求のディスクメジャー番号。

disk_minor
要求のディスクマイナー番号。

rq_flags
要求フラグ。

elevator_name
現在有効になっている I/O エレベーターのタイプ。

q
要求キューへのポインター。

rq
要求のアドレス。

name
プローブポイントの名前

名前
probe::ioscheduler.elv_add_request.tp – 要求が要求キューに追加されたことを示す tracepoint ベースのプローブ。

概要

```
ioscheduler.elv_add_request.tp
```

値

q
要求キューのポインター。

elevator_name
現在有効になっている I/O エレベーターのタイプ。

name
プローブポイントの名前

rq
要求のアドレス。

disk_major
要求のディスクメジャー番号。

disk_minor
要求のディスクマイナー番号。

rq_flags
要求フラグ。

名前

probe::ioscheduler.elv_completed_request – 要求が完了すると実行されます

概要

```
ioscheduler.elv_completed_request
```

値

name
プローブポイントの名前

rq
要求のアドレス。

elevator_name
現在有効になっている I/O エレベーターのタイプ。

disk_major
要求のディスクメジャー番号。

disk_minor
要求のディスクマイナー番号。

rq_flags
要求フラグ。

名前

probe::ioscheduler.elv_next_request – 要求キューから要求が取得されたときに実行されます。

概要

```
ioscheduler.elv_next_request
```

値

elevator_name

現在有効になっている I/O エレベーターのタイプ。

name

プローブポイントの名前

名前

probe::ioscheduler.elv_next_request.return — 要求の取得によってシグナルが返されると実行されます。

概要

```
ioscheduler.elv_next_request.return
```

値

disk_major

要求のディスクメジャー番号。

disk_minor

要求のディスクマイナー番号。

rq_flags

要求フラグ。

rq

要求のアドレス。

name

プローブポイントの名前

名前

probe::ioscheduler_trace.elv_abort_request — 要求が中止されると実行されます。

概要

```
ioscheduler_trace.elv_abort_request
```

値

disk_major

要求のディスクメジャー番号。

disk_minor
要求のディスクマイナー番号。

rq_flags
要求フラグ。

elevator_name
現在有効になっている I/O エレベーターのタイプ。

rq
要求のアドレス。

name
プローブポイントの名前

名前
probe::ioscheduler_trace.elv_completed_request – 要求が完了すると実行されます。

概要

```
ioscheduler_trace.elv_completed_request
```

値

elevator_name
現在有効になっている I/O エレベーターのタイプ。

rq
要求のアドレス。

name
プローブポイントの名前

rq_flags
要求フラグ。

disk_minor
要求のディスクマイナー番号。

disk_major
要求のディスクメジャー番号。

説明
完了済み。

名前
probe::ioscheduler_trace.elv_issue_request – 要求が完了すると実行されます。

概要

```
ioscheduler_trace.elv_issue_request
```

値

rq_flags
要求フラグ。

disk_minor
要求のディスクマイナー番号。

disk_major
要求のディスクメジャー番号。

elevator_name
現在有効になっている I/O エレベーターのタイプ。

rq
要求のアドレス。

name
プローブポイントの名前

説明
スケジュール済み。

名前
probe::ioscheduler_trace.elv_requeue_request – 要求が完了すると実行されます。

概要

```
ioscheduler_trace.elv_requeue_request
```

値

rq
要求のアドレス。

name
プローブポイントの名前

elevator_name

現在有効になっている I/O エレベーターのタイプ。

rq_flags
要求フラグ。

disk_minor
要求のディスクマイナー番号。

disk_major
要求のディスクメジャー番号。

説明
ハードウェアがこれ以上要求を受け入れることができないときにキューに戻されます。

名前
probe::ioscheduler_trace.plug – 要求キューがプラグされると実行されます。

概要

```
ioscheduler_trace.plug
```

値

rq_queue
要求キュー

name
プローブポイントの名前

説明
キュー内の要求はブロックドライバーで処理できません。

名前
probe::ioscheduler_trace.unplug_io – 要求キューがアンプラグされると実行されます。

概要

```
ioscheduler_trace.unplug_io
```

値

name
プローブポイントの名前

rq_queue

要求キュー

説明

キュー内の保留中の要求の数がしきい値を超えたとき、またはキューがプラグされたときにアクティブ化されたタイマーの終了時。

名前

probe::ioscheduler_trace.unplug_timer – アンプラグタイマーが関連付けられたときに実行されます。

概要

```
ioscheduler_trace.unplug_timer
```

値

rq_queue

要求キュー

name

プローブポイントの名前

説明

要求キューの終了時。

第11章 SCSI TAPSET

この種類のプローブポイントは、SCSI アクティビティーをプローブするために使用されます。以下のプローブポイントが含まれます。

名前

probe::scsi.iocompleted – ブロックデバイス I/O 要求の完了処理を行っている SCSI 中間層

概要

```
scsi.iocompleted
```

値

device_state

デバイスの現在の状態

dev_id

SCSI デバイス ID

req_addr

現在の構造要求ポインター (数字)

data_direction_str

データの方向 (文字列)

device_state_str

デバイスの現在の状態 (文字列)

lun

LUN 番号

goodbytes

完了済みバイト数

data_direction

data_direction は、このコマンドがデバイスから送信されるか、デバイスに送信されるかを指定します。

channel

チャネル番号

host_no

ホスト番号

名前

probe::scsi.iodispatching – SCSI 中間層でディスパッチされるローレベル SCSI コマンド

概要

```
scsi.iodispatching
```

値

device_state

デバイスの現在の状態

request_bufflen

要求バッファーの長さ

request_buffer

要求バッファーアドレス

dev_id

SCSI デバイス ID

data_direction_str

データの方向 (文字列)

req_addr

現在の構造要求ポインター (数字)

device_state_str

デバイスの現在の状態 (文字列)

lun

LUN 番号

data_direction

data_direction は、このコマンドがデバイス 0 (DMA_BIDIRECTIONAL)、1 (DMA_TO_DEVICE)、2 (DMA_FROM_DEVICE)、3 (DMA_NONE) から送信されるか、デバイスに送信されるかを指定します。

channel

チャネル番号

host_no

ホスト番号

名前

probe::scsi.iodone – 低レベルドライバーによって完了され、実行済みキューに格納される SCSI コマンド。

概要

```
scsi.iodone
```

値

device_state

デバイスの現在の状態

data_direction_str

データの方向 (文字列)

req_addr

現在の構造要求ポインター (数字)

dev_id

SCSI デバイス ID

lun

LUN 番号

scsi_timer_pending

タイマーがこの要求で保留中の場合は 1

device_state_str

デバイスの現在の状態 (文字列)

host_no

ホスト番号

channel

チャネル番号

data_direction

data_direction は、このコマンドがデバイスから送信されるか、デバイスに送信されるかを指定します。

名前

probe::scsi.ioentry – SCSI 中間層要求の作成

概要

scsi.ioentry

値

req_addr

現在の構造要求ポインター (数字)

disk_major

ディスクのメジャー番号 (未指定の場合は -1)

device_state_str

デバイスの現在の状態 (文字列)

disk_minor

ディスクのマイナー番号 (未指定の場合は -1)

device_state

デバイスの現在の状態

名前

probe::scsi.ioexecute — 中間層 SCSI 要求の作成および結果の待機

概要

scsi.ioexecute

値

host_no

ホスト番号

channel

チャネル番号

data_direction

data_direction は、このコマンドがデバイスから送信されるか、デバイスに送信されるかを指定します。

lun

LUN 番号

retries

要求を再試行する回数

device_state_str

デバイスの現在の状態 (文字列)

data_direction_str
データの方向 (文字列)

dev_id
SCSI デバイス ID

request_buffer
データバッファーアドレス

request_bufflen
データバッファーの長さ

device_state
デバイスの現在の状態

timeout
要求タイムアウト (秒単位)

名前

probe::scsi.set_state – SCSI デバイス状態の変更を要求します。

概要

```
scsi.set_state
```

値

state
デバイスの新しい状態

old_state
デバイスの現在の状態

dev_id
SCSI デバイス ID

state_str
デバイスの新しい状態 (文字列)

old_state_str
デバイスの現在の状態 (文字列)

lun

LUN 番号

channel

チャネル番号

host_no

ホスト番号

第12章 TTY TAPSET

この種類のプローブポイントは、TTY(Teletype) のアクティビティーをプローブするために使用されます。以下のプローブポイントが含まれます。

名前

probe::tty.init – tty が初期化される際に呼び出されます。

概要

```
tty.init
```

値

name

ドライバー.dev_name の名前

module

モジュール名

driver_name

ドライバー名

名前

probe::tty.ioctl – ioctl が tty に要求される際に呼び出されます。

概要

```
tty.ioctl
```

値

arg:

ioctl 引数

name

ファイル名

cmd

ioctl コマンド

名前

probe::tty.open – tty が開かれると呼び出されます。

概要

```
tty.open
```

値

inode_state
inode 状態

file_mode
ファイルモード

inode 番号
inode 番号

file_flags
ファイルフラグ

file_name
ファイル名

inode_flags
inode フラグ

名前

probe::tty.poll – tty デバイスがポーリングされる際に呼び出されます。

概要

```
tty.poll
```

値

file_name
tty ファイル名

wait_key
待機キューキー

名前

probe::tty.read – tty 行が読み込まれる際に呼び出されます。

概要

```
tty.read
```

値

file_name

tty に関連するファイル名

driver_name

ドライバー名

nr

読み込まれる文字数

buffer

文字を受信するバッファー

名前

probe::tty.receive – tty がメッセージを受信すると呼び出されます。

概要

```
tty.receive
```

値

driver_name

ドライバー名

count

受信する文字数

index

tty インデックス

cp

受信したバッファー

id

tty id

name

モジュールファイルの名前

fp

フラグバッファー

名前

probe::tty.register – tty デバイスが登録されると呼び出されます。

概要

```
tty.register
```

値

name

ドライバー.dev_name の名前

module

モジュール名

index

要求される tty インデックスです。

driver_name

ドライバー名

名前

probe::tty.release – tty が閉じると呼び出されます。

概要

```
tty.release
```

値

inode_flags

inode フラグ

file_flags

ファイルフラグ

file_name

ファイル名

inode_state

inode 状態

inode 番号

inode 番号

file_mode
ファイルモード

名前
probe::tty.resize – 端末のサイズが変更される際に呼び出されます。

概要

```
tty.resize
```

値

new_row
新規の行の値

old_row
古い行の値

name
tty 名

new_col
新規の列の値

old_xpixel
古い xpixel

old_col
古い列の値

new_xpixel
新規の xpixel 値

old_ypixel
古い ypixel

new_ypixel
新規の ypixel 値

名前
probe::tty.unregister – tty デバイスの登録が解除されると呼び出されます。

概要

```
tty.unregister
```

値

name
ドライバー.dev_name の名前

module
モジュール名

index
要求される tty インデックスです。

driver_name
ドライバー名

名前
probe::tty.write – tty 行に書き込みます。

概要

```
tty.write
```

値

nr
文字数

buffer
書き込まれるバッファー

file_name
tty に関連するファイル名

driver_name
ドライバー名

第13章 割り込み要求 (IRQ) タップセット

この一連のプローブポイントは、割り込み要求 (IRQ) アクティビティーをプローブするために使用されます。以下のプローブポイントが含まれます。

名前

probe::irq_handler.entry – 割り込みハンドラーの実行開始

概要

```
irq_handler.entry
```

値

next_irqaction

共有割り込みの次の irqaction へのポインター

thread_fn

スレッド化された割り込みの割り込みハンドラー関数

THREAD

スレッド化された割り込みのスレッドポインター

thread_flags

スレッドに関連するフラグ

irq

irq 番号

flags_str

IRQ フラグのシンボリック文字列表現

dev_name

デバイスの名前

action

*この割り込み番号の struct irqaction**

dir

proc/irq/NN/name エントリーへのポインター

flags

IRQ ハンドラーのフラグ

dev_id

デバイスを識別するための Cookie

handler
割り込みハンドラー関数

名前

probe::irq_handler.exit – 割り込みハンドラーの実行完了

概要

```
irq_handler.exit
```

値

flags_str
IRQ フラグのシンボリック文字列表現

dev_name
デバイスの名前

ret
ハンドラーの戻り値

action
*構造体 irqaction**

thread_fn
スレッド化された割り込みの割り込みハンドラー関数

next_irqaction
共有割り込みの次の irqaction へのポインター

THREAD
スレッド化された割り込みのスレッドポインター

thread_flags
スレッドに関連するフラグ

irq
割り込み番号

handler
実行された割り込みハンドラー関数

flags
IRQ ハンドラーのフラグ

dir

proc/irq/NN/name エントリーへのポインター

dev_id

デバイスを識別するための Cookie

名前

probe::softirq.entry – 保留中の softirq 開始のハンドラーの実行

概要

```
softirq.entry
```

値

action

実行しようとしている softirq ハンドラーへのポインター

vec_nr

softirq ベクトル番号

vec

softirq_action vector

h

*現在保留中の softirq の struct softirq_action**

名前

probe::softirq.exit – 保留中の softirq のハンドラーの実行が完了しました

概要

```
softirq.exit
```

値

vec_nr

softirq ベクトル番号

action

実行を終了したばかりの softirq ハンドラーへのポインター

h

*直前に実行された softirq の struct softirq_action**

vec

softirq_action vector

名前

probe::workqueue.create – 新しいワークキューの作成

概要

```
workqueue.create
```

値

wq_thread

ワークキュースレッドの task_struct

cpu

ワーカースレッドが作成される cpu

名前

probe::workqueue.destroy – ワークキューの破棄

概要

```
workqueue.destroy
```

値

wq_thread

ワークキュースレッドの task_struct

名前

probe::workqueue.execute – 据え置き作業の実行

概要

```
workqueue.execute
```

値

wq_thread

ワークキュースレッドの task_struct

work_func

ハンドラー関数へのポインター

work

work_struct 実行中*

名前

probe::workqueue.insert – ワークキューでの作業のキューイング

概要

```
workqueue.insert
```

値

wq_thread

ワークキュースレッドの task_struct

work_func

ハンドラー関数へのポインター

work

work_struct がキューに入れられている*

第14章 ネットワーキングTAPSET

この種類のプローブポイントは、ネットワークデバイスおよびプロトコル層の活動をプローブするために使用されます。

名前

function::format_ipaddr – IP アドレスの文字列表現を返します

概要

```
format_ipaddr:string(addr:long,family:long)
```

引数

addr

IP アドレス

family

IP アドレスファミリー (AF_INET または AF_INET6 のいずれか)

名前

function::htonl – 32 ビットの長さをホストからネットワークの順序に変換します

概要

```
htonl:long(x:long)
```

引数

x

変換する値

名前

function::htonll – ホストからネットワーク順序への 64 ビット long long の変換

概要

```
htonll:long(x:long)
```

引数

x

変換する値

名前
function::htons – ホストからネットワーク順序への16 ビット short の変換

概要

```
htons:long(x:long)
```

引数

x
変換する値

名前
function::ip_ntop – IPv4 アドレスの文字列表現を返します

概要

```
ip_ntop:string(addr:long)
```

引数

addr
整数で表されたIPv4 アドレス

名前
function::ntohl – 32 ビットの長さをネットワークからホストの順序に変換します

概要

```
ntohl:long(x:long)
```

引数

x
変換する値

名前
function::ntohll – 64 ビットのlong long をネットワークからホストの順序に変換します

概要

```
ntohll:long(x:long)
```

引数

x

変換する値

名前

function::ntohs – 16 ビットの short をネットワークからホストの順序に変換します

概要

```
ntohs:long(x:long)
```

引数

x

変換する値

名前

probe::netdev.change_mac – netdev_name により MAC が変更されたときに呼び出されます。

概要

```
netdev.change_mac
```

値

mac_len

MAC 長

old_mac

現在の MAC アドレス

dev_name

MAC が変更されるデバイス

new_mac

新しい MAC アドレス

名前

probe::netdev.change_mtu – netdev MTU が変更されたときに呼び出されます。

概要

```
netdev.change_mtu
```

値

old_mtu

現在の MTU

new_mtu

新しい MTU

dev_name

MTU が変更されたデバイス

名前

probe::netdev.change_rx_flag – デバイスの RX フラグが変更されたときに呼び出されます。

概要

```
netdev.change_rx_flag
```

値

flags

新しいフラグ

dev_name

変更されるデバイス

名前

probe::netdev.close – デバイスが閉じられたときに呼び出されます。

概要

```
netdev.close
```

値

dev_name

閉じられるデバイス

名前

probe::netdev.get_stats – デバイス統計の問い合わせが行われるときに呼び出されます。

概要

```
netdev.get_stats
```

値

dev_name

統計を提供するデバイス

名前

probe::netdev.hard_transmit – デバイスが TX (ハード) に移動するときに呼び出されます。

概要

```
netdev.hard_transmit
```

値

原寸大

送信されるデータのサイズ

dev_name

送信するようスケジュールされたデバイス

protocol

送信で使用されるプロトコル

長さ

送信バッファーの長さ。

名前

probe::netdev.ioctl – デバイスで IOCTL が発生したときに呼び出されます。

概要

```
netdev.ioctl
```

値

arg:

IOCTL 引数 (通常は netdev インターフェイス)

cmd

IOCTL 要求

名前

probe::netdev.open – デバイスが開いたときに呼び出されます。

概要

```
netdev.open
```

値

dev_name

開くデバイス

名前

probe::netdev.receive – ネットワークデバイスから受信されたデータ

概要

```
netdev.receive
```

値

長さ

受信バッファーの長さ。

protocol

受信されるパケットのプロトコル。

dev_name

デバイスの名前 (例: eth0、ath1)。

名前

probe::netdev.register – デバイスが登録されたときに呼び出されます。

概要

```
netdev.register
```

値

dev_name

登録されるデバイス

名前

probe::netdev.rx – デバイスがパケットを受信するときに呼び出されます。

概要

```
netdev.rx
```

値

dev_name

パケットを受信したデバイス。

protocol

パケットプロトコル

名前

probe::netdev.set_promiscuity – デバイスのプロミスキャスモードが開始されたとき、またはプロミスキャスモードが終了したときに呼び出されます。

概要

```
netdev.set_promiscuity
```

値

dev_name

プロミスキャスモードが開始される、またはプロミスキャスモードが終了するデバイス。

enable

デバイスでプロミスキャスモードが開始される場合

株式会社 です。

プロミスキャスモードオープナーの数をカウントします。

disable

デバイスがpromiscuity モードを脱退する場合

名前

probe::netdev.transmit – ネットワークデバイスの送信バッファー

概要

```
netdev.transmit
```

値

protocol

このパケットのプロトコル (include/linux/if_ether.h で定義されます)。

長さ

送信バッファーの長さ。

原寸大

送信されるデータのサイズ

dev_name

デバイスの名前 (例: eth0、ath1)。

名前

probe::netdev.unregister – デバイスの登録が解除されると呼び出されます。

概要

```
netdev.unregister
```

値

dev_name

登録解除されるデバイス

名前

probe::netfilter.arp.forward – - ARP パケットが転送されるたびに呼び出される

概要

```
netfilter.arp.forward
```

値

ar_hln

ハードウェアアドレスの長さ

nf_stop

停止判定を表すために使用される定数

outdev_name

パケットがルーティングされるネットワークデバイスの名前 (既知の場合)

ar_tha

Ethernet+IP のみ (ar_pro==0x800): ターゲットハードウェア (MAC) アドレス

nf_accept

受け入れる評決を示すために使用される定数

ar_data

ARP パケットデータ領域のアドレス (ヘッダーの後)

indev_name

パケットを受信したネットワークデバイスの名前 (既知の場合)

arphdr

ARP ヘッダーのアドレス

outdev

出力デバイスを表す net_device のアドレス、不明の場合は 0

nf_repeat

反復評決を示すために使用される定数

長さ

パケットバッファーの内容の長さ (バイト単位)

nf_stolen

盗まれた評決を示すために使用される定数

ar_pln

プロトコルアドレスの長さ

pf

プロトコルファミリー -- 常に 「arp」

ar_sha

Ethernet+IP のみ (ar_pro==0x800): ソースハードウェア (MAC) アドレス

インデブ

入力デバイスを表す net_device のアドレス、不明の場合は 0

nf_drop

ドロップ判定を示すために使用される定数

ar_pro
プロトコルアドレスのフォーマット

ar_sip
Ethernet+IP のみ (ar_pro==0x800): ソース IP アドレス

ar_tip
Ethernet+IP のみ (ar_pro==0x800): ターゲット IP アドレス

ar_hrd
ハードウェアアドレスのフォーマット

nf_queue
キューの判定を示すために使用される定数

ar_op
ARP オペコード (コマンド)

名前

probe::netfilter.arp.in – - 着信 ARP パケットごとに呼び出される

概要

```
netfilter.arp.in
```

値

ar_hln
ハードウェアアドレスの長さ

nf_stop
停止判定を表すために使用される定数

nf_accept
受け入れる評決を示すために使用される定数

ar_tha
Ethernet+IP のみ (ar_pro==0x800): ターゲットハードウェア (MAC) アドレス

ar_data
ARP パケットデータ領域のアドレス (ヘッダーの後)

outdev_name

パケットがルーティングされるネットワークデバイスの名前 (既知の場合)

outdev

出力デバイスを表す net_device のアドレス、不明の場合は 0

nf_repeat

反復評決を示すために使用される定数

arphdr

ARP ヘッダーのアドレス

indev_name

パケットを受信したネットワークデバイスの名前 (既知の場合)

nf_stolen

盗まれた評決を示すために使用される定数

長さ

パケットバッファーの内容の長さ (バイト単位)

ar_pln

プロトコルアドレスの長さ

ar_sha

Ethernet+IP のみ (ar_pro==0x800): ソースハードウェア (MAC) アドレス

pf

プロトコルファミリー -- 常に 「arp」

nf_drop

ドロップ判定を示すために使用される定数

ar_pro

プロトコルアドレスのフォーマット

ar_sip

Ethernet+IP のみ (ar_pro==0x800): ソース IP アドレス

インデブ

入力デバイスを表す net_device のアドレス、不明の場合は 0

ar_tip

Ethernet+IP のみ (ar_pro==0x800): ターゲット IP アドレス

ar_hrd

ハードウェアアドレスのフォーマット

ar_op

ARP オペコード (コマンド)

nf_queue

キューの判定を示すために使用される定数

名前

probe::netfilter.arp.out – - 発信 ARP パケットごとに呼び出される

概要

```
netfilter.arp.out
```

値

ar_tip

Ethernet+IP のみ (ar_pro==0x800): ターゲット IP アドレス

nf_drop

ドロップ判定を示すために使用される定数

ar_pro

プロトコルアドレスのフォーマット

ar_sip

Ethernet+IP のみ (ar_pro==0x800): ソース IP アドレス

インデブ

入力デバイスを表す net_device のアドレス、不明の場合は 0

ar_sha

Ethernet+IP のみ (ar_pro==0x800): ソースハードウェア (MAC) アドレス

pf

プロトコルファミリー -- 常に 「arp」

ar_op

ARP オペコード (コマンド)

nf_queue

キューの判定を示すために使用される定数

ar_hrd

ハードウェアアドレスのフォーマット

nf_accept
受け入れる評決を示すために使用される定数

ar_data
ARP パケットデータ領域のアドレス (ヘッダーの後)

ar_tha
Ethernet+IP のみ (ar_pro==0x800): ターゲットハードウェア (MAC) アドレス

outdev_name
パケットがルーティングされるネットワークデバイスの名前 (既知の場合)

nf_stop
停止判定を表すために使用される定数

ar_hln
ハードウェアアドレスの長さ

ar_pln
プロトコルアドレスの長さ

nf_stolen
盗まれた評決を示すために使用される定数

長さ
パケットバッファーの内容の長さ (バイト単位)

outdev
出力デバイスを表す net_device のアドレス、不明の場合は 0

nf_repeat
反復評決を示すために使用される定数

arphdr
ARP ヘッダーのアドレス

indev_name
パケットを受信したネットワークデバイスの名前 (既知の場合)

名前

probe::netfilter.bridge.forward – 他のコンピューター宛ての着信ブリッジングパケットで呼び出されます

概要

-

netfilter.bridge.forward

値

br_fd

1/256 秒の転送遅延

nf_queue

キューの判定を示すために使用される定数

brhdr

ブリッジヘッダーのアドレス

br_mac

ブリッジMAC アドレス

インデブ

入力デバイスを表すnet_device のアドレス、不明の場合は 0

br_msg

メッセージの経過時間 (1/256 秒)

nf_drop

ドロップ判定を示すために使用される定数

llcproto_stp

ブリッジスパニングツリープロトコルパケットを示すために使用される定数

pf

プロトコルファミリー -- 常に 「ブリッジ」

br_vid

プロトコルバージョン識別子

indev_name

パケットを受信したネットワークデバイスの名前 (既知の場合)

br_poid

ポート識別子

outdev

出力デバイスを表すnet_device のアドレス、不明の場合は 0

nf_repeat

反復評決を示すために使用される定数

llcpdu

LLC プロトコルデータユニットのアドレス

長さ
パケットバッファーの内容の長さ (バイト単位)

nf_stolen
盗まれた評決を示すために使用される定数

br_cost
送信ブリッジからルートまでの総コスト

nf_stop
停止判定を表すために使用される定数

br_type
BPDU タイプ

br_max
1/256 秒の最大年齢

br_htime
1/256 秒のハロータイム

protocol
パケットプロトコル

br_bid
橋の正体

br_rmac
ルートブリッジの MAC アドレス

br_prid
プロトコル識別子

outdev_name
パケットがルーティングされるネットワークデバイスの名前 (既知の場合)

br_flags
BPDU フラグ

nf_accept
受け入れる評決を示すために使用される定数

br_rid
ルートブリッジのアイデンティティ

名前

probe::netfilter.bridge.local_in – ローカルコンピューター宛てのブリッジングパケットで呼び出される

概要

```
netfilter.bridge.local_in
```

値

llcproto_stp

ブリッジスパニングツリープロトコルパケットを示すために使用される定数

pf

プロトコルファミリー-- 常に 「ブリッジ」

nf_drop

ドロップ判定を示すために使用される定数

br_msg

メッセージの経過時間 (1/256 秒)

インデブ

入力デバイスを表す net_device のアドレス、不明の場合は 0

nf_queue

キューの判定を示すために使用される定数

br_fd

1/256 秒の転送遅延

br_mac

ブリッジMAC アドレス

brhdr

ブリッジヘッダーのアドレス

br_rid

ルートブリッジのアイデンティティ

nf_accept

受け入れる評決を示すために使用される定数

outdev_name

パケットがルーティングされるネットワークデバイスの名前 (既知の場合)

br_flags

BPDU フラグ

br_prid
プロトコル識別子

br_htime
1/256 秒のハロータイム

protocol
パケットプロトコル

br_bid
橋の正体

br_rmac
ルートブリッジの MAC アドレス

br_max
1/256 秒の最大年齢

br_type
BPDU タイプ

nf_stop
停止判定を表すために使用される定数

br_cost
送信ブリッジからルートまでの総コスト

nf_stolen
盗まれた評決を示すために使用される定数

長さ
パケットバッファーの内容の長さ (バイト単位)

llcpdu
LLC プロトコルデータユニットのアドレス

outdev
出力デバイスを表す net_device のアドレス、不明の場合は 0

nf_repeat
反復評決を示すために使用される定数

indev_name
パケットを受信したネットワークデバイスの名前 (既知の場合)

br_poid

ポート識別子

br_vid

プロトコルバージョン識別子

名前

probe::netfilter.bridge.local_out — ローカルプロセスからのブリッジングパケットで呼び出されます

概要

```
netfilter.bridge.local_out
```

値

インデブ

入力デバイスを表す net_device のアドレス、不明の場合は 0

br_msg

メッセージの経過時間 (1/256 秒)

nf_drop

ドロップ判定を示すために使用される定数

llcproto_stp

ブリッジスパニングツリープロトコルパケットを示すために使用される定数

pf

プロトコルファミリー -- 常に 「ブリッジ」

br_fd

1/256 秒の転送遅延

nf_queue

キューの判定を示すために使用される定数

brhdr

ブリッジヘッダーのアドレス

br_mac

ブリッジMAC アドレス

br_flags

BPDU フラグ

outdev_name

パケットがルーティングされるネットワークデバイスの名前 (既知の場合)

nf_accept

受け入れる評決を示すために使用される定数

br_rid

ルートブリッジのアイデンティティ

nf_stop

停止判定を表すために使用される定数

br_type

BPDU タイプ

br_max

1/256 秒の最大年齢

protocol

パケットプロトコル

br_htime

1/256 秒のハロータイム

br_bid

橋の正体

br_rmac

ルートブリッジの MAC アドレス

br_prid

プロトコル識別子

llcpdu

LLC プロトコルデータユニットのアドレス

長さ

パケットバッファーの内容の長さ (バイト単位)

nf_stolen

盗まれた評決を示すために使用される定数

br_cost

送信ブリッジからルートまでの総コスト

br_vid

プロトコルバージョン識別子

indev_name
パケットを受信したネットワークデバイスの名前 (既知の場合)

br_poid
ポート識別子

outdev
出力デバイスを表す net_device のアドレス、不明の場合は 0

nf_repeat
反復評決を示すために使用される定数

名前

probe::netfilter.bridge.post_routing – - ブリッジングパケットが回線に到達する前に呼び出される

概要

```
netfilter.bridge.post_routing
```

値

llcproto_stp
ブリッジスパニングツリープロトコルパケットを示すために使用される定数

pf
プロトコルファミリー -- 常に 「ブリッジ」

インデブ
入力デバイスを表す net_device のアドレス、不明の場合は 0

nf_drop
ドロップ判定を示すために使用される定数

br_msg
メッセージの経過時間 (1/256 秒)

nf_queue
キューの判定を示すために使用される定数

br_mac
ブリッジMAC アドレス

br_fd

1/256 秒の転送遅延

brhdr
ブリッジヘッダーのアドレス

br_htime
1/256 秒のハロータイム

br_bid
橋の正体

br_rmac
ルートブリッジの MAC アドレス

protocol
パケットプロトコル

br_prid
プロトコル識別子

br_type
BPDU タイプ

nf_stop
停止判定を表すために使用される定数

br_max
1/256 秒の最大年齢

br_rid
ルートブリッジのアイデンティティ

br_flags
BPDU フラグ

outdev_name
パケットがルーティングされるネットワークデバイスの名前 (既知の場合)

nf_accept
受け入れる評決を示すために使用される定数

indev_name
パケットを受信したネットワークデバイスの名前 (既知の場合)

br_poid
ポート識別子

outdev

出力デバイスを表す net_device のアドレス、不明の場合は 0

nf_repeat

反復評決を示すために使用される定数

br_vid

プロトコルバージョン識別子

長さ

パケットバッファーの内容の長さ (バイト単位)

nf_stolen

盗まれた評決を示すために使用される定数

br_cost

送信ブリッジからルートまでの総コスト

llcpdu

LLC プロトコルデータユニットのアドレス

名前

probe::netfilter.bridge.pre_routing – - ブリッジングパケットがルーティングされる前に呼び出される

概要

```
netfilter.bridge.pre_routing
```

値

llcproto_stp

ブリッジスパニングツリープロトコルパケットを示すために使用される定数

pf

プロトコルファミリー-- 常に 「ブリッジ」

nf_drop

ドロップ判定を示すために使用される定数

br_msg

メッセージの経過時間 (1/256 秒)

インデブ

入力デバイスを表す net_device のアドレス、不明の場合は 0

brhdr
ブリッジヘッダーのアドレス

nf_queue
キューの判定を示すために使用される定数

br_fd
1/256 秒の転送遅延

br_mac
ブリッジMAC アドレス

br_rid
ルートブリッジのアイデンティティ

nf_accept
受け入れる評決を示すために使用される定数

br_flags
BPDU フラグ

outdev_name
パケットがルーティングされるネットワークデバイスの名前(既知の場合)

br_prid
プロトコル識別子

br_rmac
ルートブリッジのMAC アドレス

br_htime
1/256 秒のハロータイム

br_bid
橋の正体

protocol
パケットプロトコル

br_max
1/256 秒の最大年齢

br_type
BPDU タイプ

nf_stop

停止判定を表すために使用される定数

br_cost
送信ブリッジからルートまでの総コスト

nf_stolen
盗まれた評決を示すために使用される定数

長さ
パケットバッファーの内容の長さ (バイト単位)

llcpdu
LLC プロトコルデータユニットのアドレス

outdev
出力デバイスを表す net_device のアドレス、不明の場合は 0

nf_repeat
反復評決を示すために使用される定数

indev_name
パケットを受信したネットワークデバイスの名前 (既知の場合)

br_poid
ポート識別子

br_vid
プロトコルバージョン識別子

名前

probe::netfilter.ip.forward – 他のコンピューター宛ての着信 IP パケットで呼び出される

概要

> *netfilter.ip.forward*

値

saddr
ソース IP アドレスを表す文字列

sport
TCP または UDP ソースポート (ipv4 のみ)

daddr

宛先 IP アドレスを表す文字列

pf
プロトコルファミリー -- 「ipv4」 または 「ipv6」 のいずれか

インデブ
入力デバイスを表す net_device のアドレス、不明の場合は 0

nf_drop
ドロップ判定を示すために使用される定数

nf_queue
キューの判定を示すために使用される定数

dport
TCP または UDP 宛先ポート (ipv4 のみ)

iphdr
IP ヘッダーのアドレス

フィン
TCP FIN フラグ (プロトコルが TCP の場合、ipv4 のみ)

確認
TCP ACK フラグ (プロトコルが TCP の場合。ipv4 のみ)

シン
TCP SYN フラグ (プロトコルが TCP の場合、ipv4 のみ)

ipproto_udp
パケットプロトコルが UDP であることを示すために使用される定数

outdev_name
パケットがルーティングされるネットワークデバイスの名前 (既知の場合)

nf_accept
受け入れる評決を示すために使用される定数

rst
TCP RST フラグ (プロトコルが TCP の場合、ipv4 のみ)

protocol
ドライバーからのパケットプロトコル (ipv4 のみ)

nf_stop
停止判定を表すために使用される定数

長さ

パケットバッファーの内容の長さ (バイト単位)

nf_stolen

盗まれた評決を示すために使用される定数

促す

TCP URG フラグ (プロトコルが TCP の場合、ipv4 のみ)

psh

TCP PSH フラグ (プロトコルが TCP の場合、ipv4 のみ)

ipproto_tcp

パケットプロトコルが TCP であることを示すために使用される定数

indev_name

パケットを受信したネットワークデバイスの名前 (既知の場合)

family

IP アドレスファミリー

outdev

出力デバイスを表す net_device のアドレス、不明の場合は 0

nf_repeat

反復評決を示すために使用される定数

名前

probe::netfilter.ip.local_in – ローカルコンピューター宛ての着信 IP パケットで呼び出される

概要

```
netfilter.ip.local_in
```

値

nf_stolen

盗まれた評決を示すために使用される定数

長さ

パケットバッファーの内容の長さ (バイト単位)

促す

TCP URG フラグ (プロトコルが TCP の場合、ipv4 のみ)

psh

TCP PSH フラグ (プロトコルが TCP の場合、ipv4 のみ)

nf_repeat

反復評決を示すために使用される定数

family

IP アドレスファミリー

outdev

出力デバイスを表す net_device のアドレス、不明の場合は 0

ipproto_tcp

パケットプロトコルが TCP であることを示すために使用される定数

indev_name

パケットを受信したネットワークデバイスの名前 (既知の場合)

nf_accept

受け入れる評決を示すために使用される定数

outdev_name

パケットがルーティングされるネットワークデバイスの名前 (既知の場合)

protocol

ドライバーからのパケットプロトコル (ipv4 のみ)

rst

TCP RST フラグ (プロトコルが TCP の場合、ipv4 のみ)

nf_stop

停止判定を表すために使用される定数

nf_queue

キューの判定を示すために使用される定数

dport

TCP または UDP 宛先ポート (ipv4 のみ)

iphdr

IP ヘッダーのアドレス

フィン

TCP FIN フラグ (プロトコルが TCP の場合、ipv4 のみ)

シン

TCP SYN フラグ (プロトコルが TCP の場合、ipv4 のみ)

確認

TCP ACK フラグ (プロトコルが TCP の場合。ipv4 のみ)

ipproto_udp

パケットプロトコルが UDP であることを示すために使用される定数

saddr

ソース IP アドレスを表す文字列

sport

TCP または UDP ソースポート (ipv4 のみ)

pf

プロトコルファミリー -- 「ipv4」 または 「ipv6」 のいずれか

daddr

宛先 IP アドレスを表す文字列

nf_drop

ドロップ判定を示すために使用される定数

インデブ

入力デバイスを表す net_device のアドレス、不明の場合は 0

名前

probe::netfilter.ip.local_out – 発信 IP パケットで呼び出される

概要

```
netfilter.ip.local_out
```

値

dport

TCP または UDP 宛先ポート (ipv4 のみ)

nf_queue

キューの判定を示すために使用される定数

シン

TCP SYN フラグ (プロトコルが TCP の場合、ipv4 のみ)

ipproto_udp

パケットプロトコルが UDP であることを示すために使用される定数

確認
TCP ACK フラグ (プロトコルが TCP の場合。ipv4 のみ)

フィン
TCP FIN フラグ (プロトコルが TCP の場合、ipv4 のみ)

iphdr
IP ヘッダーのアドレス

saddr
ソース IP アドレスを表す文字列

sport
TCP または UDP ソースポート (ipv4 のみ)

インデブ
入力デバイスを表す net_device のアドレス、不明の場合は 0

nf_drop
ドロップ判定を示すために使用される定数

daddr
宛先 IP アドレスを表す文字列

pf
プロトコルファミリー -- 「ipv4」 または 「ipv6」 のいずれか

psh
TCP PSH フラグ (プロトコルが TCP の場合、ipv4 のみ)

促す
TCP URG フラグ (プロトコルが TCP の場合、ipv4 のみ)

長さ
パケットバッファーの内容の長さ (バイト単位)

nf_stolen
盗まれた評決を示すために使用される定数

ipproto_tcp
パケットプロトコルが TCP であることを示すために使用される定数

indev_name
パケットを受信したネットワークデバイスの名前 (既知の場合)

nf_repeat

反復評決を示すために使用される定数

family

IP アドレスファミリー

outdev

出力デバイスを表す net_device のアドレス、不明の場合は 0

outdev_name

パケットがルーティングされるネットワークデバイスの名前 (既知の場合)

nf_accept

受け入れる評決を示すために使用される定数

nf_stop

停止判定を表すために使用される定数

rst

TCP RST フラグ (プロトコルが TCP の場合、ipv4 のみ)

protocol

ドライバーからのパケットプロトコル (ipv4 のみ)

名前

probe::netfilter.ip.post_routing – 発信 IP パケットがコンピューターを離れる直前に呼び出されます

概要

```
netfilter.ip.post_routing
```

値

family

IP アドレスファミリー

outdev

出力デバイスを表す net_device のアドレス、不明の場合は 0

nf_repeat

反復評決を示すために使用される定数

ipproto_tcp

パケットプロトコルが TCP であることを示すために使用される定数

indev_name

パケットを受信したネットワークデバイスの名前 (既知の場合)

nf_stolen

盗まれた評決を示すために使用される定数

長さ

パケットバッファーの内容の長さ (バイト単位)

促す

TCP URG フラグ (プロトコルが TCP の場合、ipv4 のみ)

psh

TCP PSH フラグ (プロトコルが TCP の場合、ipv4 のみ)

rst

TCP RST フラグ (プロトコルが TCP の場合、ipv4 のみ)

protocol

ドライバーからのパケットプロトコル (ipv4 のみ)

nf_stop

停止判定を表すために使用される定数

nf_accept

受け入れる評決を示すために使用される定数

outdev_name

パケットがルーティングされるネットワークデバイスの名前 (既知の場合)

iphdr

IP ヘッダーのアドレス

フィン

TCP FIN フラグ (プロトコルが TCP の場合、ipv4 のみ)

シン

TCP SYN フラグ (プロトコルが TCP の場合、ipv4 のみ)

ipproto_udp

パケットプロトコルが UDP であることを示すために使用される定数

確認

TCP ACK フラグ (プロトコルが TCP の場合。ipv4 のみ)

nf_queue

キューの判定を示すために使用される定数

dport

TCP または UDP 宛先ポート (ipv4 のみ)

pf

プロトコルファミリー -- 「ipv4」 または 「ipv6」 のいずれか

daddr

宛先 IP アドレスを表す文字列

nf_drop

ドロップ判定を示すために使用される定数

インデブ

入力デバイスを表す net_device のアドレス、不明の場合は 0

saddr

ソース IP アドレスを表す文字列

sport

TCP または UDP ソースポート (ipv4 のみ)

名前

probe::netfilter.ip.pre_routing – IP パケットがルーティングされる前に呼び出されます

概要

```
netfilter.ip.pre_routing
```

値

インデブ

入力デバイスを表す net_device のアドレス、不明の場合は 0

nf_drop

ドロップ判定を示すために使用される定数

daddr

宛先 IP アドレスを表す文字列

pf

プロトコルファミリー - ipv4 または ipv6 のいずれか

sport

TCP または UDP ソースポート (ipv4 のみ)

saddr

ソース IP アドレスを表す文字列

シン

TCP SYN フラグ (プロトコルが TCP の場合、ipv4 のみ)

ipproto_udp

パケットプロトコルが UDP であることを示すために使用される定数

確認

TCP ACK フラグ (プロトコルが TCP の場合。ipv4 のみ)

iphdr

IP ヘッダーのアドレス

フィン

TCP FIN フラグ (プロトコルが TCP の場合、ipv4 のみ)

dport

TCP または UDP 宛先ポート (ipv4 のみ)

nf_queue

キューの判定を示すために使用される定数

nf_stop

停止判定を表すために使用される定数

rst

TCP RST フラグ (プロトコルが TCP の場合、ipv4 のみ)

protocol

ドライバーからのパケットプロトコル (ipv4 のみ)

outdev_name

パケットがルーティングされるネットワークデバイスの名前 (既知の場合)

nf_accept

受け入れる評決を示すために使用される定数

indev_name

パケットを受信したネットワークデバイスの名前 (既知の場合)

ipproto_tcp

パケットプロトコルが TCP であることを示すために使用される定数

family

IP アドレスファミリー

nf_repeat

反復評決を示すために使用される定数

outdev

出力デバイスを表す net_device のアドレス、不明の場合は 0

psh

TCP PSH フラグ (プロトコルが TCP の場合、ipv4 のみ)

促す

TCP URG フラグ (プロトコルが TCP の場合、ipv4 のみ)

長さ

パケットバッファーの内容の長さ (バイト単位)

nf_stolen

盗まれた評決を示すために使用される定数

名前

probe::sunrpc.clnt.bind_new_program – 新しい RPC プログラムを既存のクライアントにバインドする

概要

```
sunrpc.clnt.bind_new_program
```

値

progname

新しい RPC プログラムの名前

old_prog

古い RPC プログラムの数

ヴァース

新しい RPC プログラムのバージョン

servername

サーバーのマシン名

old_vers

古い RPC プログラムのバージョン

old_progname
古い RPC プログラムの名前

prog
新しい RPC プログラムの数

名前

probe::sunrpc.clnt.call_async – 非同期 RPC 呼び出しを行う

概要

```
sunrpc.clnt.call_async
```

値

progname
RPC プログラム名

prot
IP プロトコル番号

/proc/
この RPC 呼び出しの手続き番号

procname
この RPC 呼び出しのプロシージャ名

ヴァース
RPC プログラムのバージョン番号

flags
flags

servername
サーバーのマシン名

xid
現在の送信 ID

port
ポート番号

prog
RPC プログラム番号

dead

このクライアントが放棄されたかどうか

名前

probe::sunrpc.clnt.call_sync – 同期 RPC 呼び出しを行う

概要

```
sunrpc.clnt.call_sync
```

値

xid

現在の送信 ID

servername

サーバーのマシン名

flags

flags

dead

このクライアントが放棄されたかどうか

prog

RPC プログラム番号

port

ポート番号

prot

IP プロトコル番号

progname

RPC プログラム名

ヴァース

RPC プログラムのバージョン番号

/proc/

この RPC 呼び出しの手続き番号

procname

この RPC 呼び出しのプロシージャ名

名前

probe::sunrpc.clnt.clone_client – RPC クライアント構造を複製する

概要

```
sunrpc.clnt.clone_client
```

値

authflavor

認証フレーバー

port

ポート番号

progname

RPC プログラム名

servername

サーバーのマシン名

prot

IP プロトコル番号

prog

RPC プログラム番号

ヴァース

RPC プログラムのバージョン番号

名前

probe::sunrpc.clnt.create_client – RPC クライアントを作成する

概要

```
sunrpc.clnt.create_client
```

値

servername

サーバーのマシン名

prot

IP プロトコル番号

authflavor
認証フレーバー

port
ポート番号

progname
RPC プログラム名

ヴァース
RPC プログラムのバージョン番号

prog
RPC プログラム番号

名前

probe::sunrpc.clnt.restart_call – 非同期 RPC 呼び出しを再開する

概要

```
sunrpc.clnt.restart_call
```

値

servername
サーバーのマシン名

tk_priority
タスクの優先度

xid
送信 ID

prog
RPC プログラム番号

tk_runstate
タスクの実行ステータス

tk_pid
タスクのデバッグ支援

tk_flags
タスクフラグ

名前

probe::sunrpc.clnt.shutdown_client — RPC クライアントをシャットダウンする

概要

```
sunrpc.clnt.shutdown_client
```

値

om_queue

xmit のためにキューに入れられた jiffies

クローン

クローンの数

ヴァース

RPC プログラムのバージョン番号

om_rtt

RPC RTT jiffies

om_execute

RPC 実行の瞬間

rpccnt

RPC 呼び出しの数

progname

RPC プログラム名

authflavor

認証フレーバー

prot

IP プロトコル番号

prog

RPC プログラム番号

om_bytes_recv

のバイト数

om_bytes_sent

バイト数

port

ポート番号

om_ntrans
RPC 送信の数

netreconn
再接続の数

om_ops
操作回数

tasks
参照数

servername
サーバーのマシン名

名前

probe::sunrpc.sched.delay – RPC タスクを遅らせる

概要

```
sunrpc.sched.delay
```

値

prog
RPC 呼び出しのプログラム番号

xid
RPC 呼び出しの送信 ID

delay
遅れた時間

ヴァース
RPC 呼び出しのプログラムバージョン

tk_flags
タスクのフラグ

tk_pid
タスクのデバッグ ID

prot

RPC 呼び出しの IP プロトコル

名前

probe::sunrpc.sched.execute – RPC スケジューラーを実行します

概要

```
sunrpc.sched.execute
```

値

tk_pid

タスクのデバッグ ID

prot

RPC 呼び出しの IP プロトコル

ヴァース

RPC 呼び出しのプログラムバージョン

tk_flags

タスクのフラグ

xid

RPC 呼び出しの送信 ID

prog

RPC 呼び出しのプログラム番号

名前

probe::sunrpc.sched.new_task – 指定されたクライアントの新しいタスクを作成します

概要

```
sunrpc.sched.new_task
```

値

xid

RPC 呼び出しの送信 ID

prog

RPC 呼び出しのプログラム番号

prot

RPC 呼び出しの IP プロトコル

ヴァース

RPC 呼び出しのプログラムバージョン

tk_flags

タスクのフラグ

名前

probe::sunrpc.sched.release_task – タスクに関連付けられたすべてのリソースを解放する

概要

```
sunrpc.sched.release_task
```

値

prot

RPC 呼び出しの IP プロトコル

tk_flags

タスクのフラグ

ヴァース

RPC 呼び出しのプログラムバージョン

xid

RPC 呼び出しの送信 ID

prog

RPC 呼び出しのプログラム番号

説明

***rpc_release_task** 関数が特定のカーネルで見つからない場合があります。したがって、見つからない場合は、すべてに対して -1 を返します。*

名前

probe::sunrpc.svc.create – RPC サービスを作成する

概要

```
sunrpc.svc.create
```

値

バフサイズ
バッファーサイズ

pg_nvers
サポートされているバージョンの数

progname
プログラムの名前

prog
プログラムの番号

名前
probe::sunrpc.svc.destroy – RPC サービスを破棄する

概要

```
sunrpc.svc.destroy
```

値

sv_nrthreads
同時スレッドの数

sv_name
サービス名

sv_prog
プログラムの番号

rpcbadauth
認証失敗のためにドロップされたリクエストの数

rpcbadfmt
不正なフォーマットのためにドロップされたリクエストの数

rpccnt
有効な RPC リクエストの数

sv_progname
プログラムの名前

netcnt

受信した RPC リクエストの数

nettcpconn
受け入れられた TCP 接続の数

名前

probe::sunrpc.svc.drop – RPC 要求をドロップする

概要

```
sunrpc.svc.drop
```

値

rq_xid
リクエストの送信 ID

sv_name
サービス名

rq_prot
リクエストの IP プロトコル

peer_ip
リクエストの送信元のピアアドレス

rq_proc
リクエストの手続き番号

rq_vers
リクエスト内のプログラムバージョン

rq_prog
リクエスト内のプログラム番号

名前

probe::sunrpc.svc.process – RPC リクエストを処理する

概要

```
sunrpc.svc.process
```

値

rq_prog
リクエスト内のプログラム番号

rq_vers
リクエスト内のプログラムバージョン

peer_ip
リクエストの送信元のピアアドレス

rq_proc
リクエストの手続き番号

sv_prog
プログラムの番号

rq_prot
リクエストの IP プロトコル

sv_name
サービス名

rq_xid
リクエストの送信 ID

sv_nrthreads
同時スレッドの数

名前

probe::sunrpc.svc.recv – 任意のソケットで次の RPC リクエストをリッスンします

概要

```
sunrpc.svc.recv
```

値

sv_nrthreads
同時スレッドの数

sv_name
サービス名

sv_prog
プログラムの番号

timeout
データ待ちのタイムアウト

名前
probe::sunrpc.svc.register – RPC サービスをローカルポートマッパーに登録する

概要

```
sunrpc.svc.register
```

値

sv_name
サービス名

prog
プログラムの番号

port
ポート番号

progname
プログラムの名前

prot
IP プロトコル番号

説明
proto *と* ***port*** *が両方とも 0 の場合、サービスの登録を解除します。*

名前
probe::sunrpc.svc.send – RPC クライアントに返信を返す

概要

```
sunrpc.svc.send
```

値

rq_vers
リクエスト内のプログラムバージョン

rq_prog
リクエスト内のプログラム番号

rq_prot
リクエストの IP プロトコル

sv_name
サービス名

rq_xid
リクエストの送信 ID

peer_ip
リクエストの送信元のピアアドレス

rq_proc
リクエストの手続き番号

名前

probe::tcp.disconnect – TCP ソケットの接続解除

概要

```
tcp.disconnect
```

値

flags
TCP フラグ (FIN など)

daddr
宛先 IP アドレスを表す文字列

sport
TCP ソースポート。

family
IP アドレスファミリー

name
このプローブの名前

saddr
ソース IP アドレスを表す文字列

dport
TCP 宛先ポート。

sock

ネットワークソケット。

コンテキスト

TCP を接続解除するプロセス

名前

probe::tcp.disconnect.return – TCP ソケットの接続解除が完了しました。

概要

```
tcp.disconnect.return
```

値

name

このプローブの名前

ret

エラーコード (0: エラーなし)

コンテキスト

TCP を接続解除するプロセス

名前

probe::tcp.receive – TCP パケットが受信されたときに呼び出されます。

概要

```
tcp.receive
```

値

psh

TCP PSH フラグ

確認

TCP ACK フラグ

daddr

宛先 IP アドレスを表す文字列

シン

TCP SYN フラグ

rst
TCP RST フラグ

sport
TCP ソースポート。

protocol
ドライバーからのパケットプロトコル

促す
TCP URG フラグ

name
プローブポイントの名前

family
IP アドレスファミリー

フィン
TCP FIN フラグ

saddr
ソース IP アドレスを表す文字列

iphdr
IP ヘッダーアドレス

dport
TCP 宛先ポート。

名前

probe::tcp.recvmsg – TCP メッセージの受信。

概要

```
tcp.recvmsg
```

値

daddr
宛先 IP アドレスを表す文字列

sport
TCP ソースポート。

size

受信バイト数。

name

このプローブの名前

family

IP アドレスファミリー

saddr

ソース IP アドレスを表す文字列

sock

ネットワークソケット。

dport

TCP 宛先ポート。

コンテキスト

TCP メッセージを受信するプロセス。

名前

probe::tcp.recvmsg.return – TCP メッセージの受信が完了しました。

概要

```
tcp.recvmsg.return
```

値

saddr

ソース IP アドレスを表す文字列

dport

TCP 宛先ポート。

daddr

宛先 IP アドレスを表す文字列

size

エラーが発生した場合の受信バイト数またはエラーコード。

sport

TCP ソースポート。

family

IP アドレスファミリー

name

このプローブの名前

コンテキスト

TCP メッセージを受信するプロセス。

名前

probe::tcp.sendmsg – TCP メッセージの送信。

概要

```
tcp.sendmsg
```

値

family

IP アドレスファミリー

sock

ネットワークソケット。

name

このプローブの名前

size

送信バイト数。

コンテキスト

TCP メッセージを送信するプロセス。

名前

probe::tcp.sendmsg.return – TCP メッセージの送信が行われました。

概要

```
tcp.sendmsg.return
```

値

name

このプローブの名前

size

エラーが発生した場合の送信バイト数またはエラーコード。

コンテキスト

TCP メッセージを送信するプロセス。

名前

*probe::tcp.setsockopt – **setsockopt** に呼び出します*

概要

> *tcp.setsockopt*

値

optstr

optname を人間が判読できる形式に解決します。

name

このプローブの名前

family

IP アドレスファミリー

level

ソケットオプションが処理されるレベル。

optname

TCP ソケットオプション (TCP_NODELAY や TCP_MAXSEG など)

sock

ネットワークソケット。

optlen

***setsockopt** の値のアクセスに使用*

コンテキスト

setsockopt を呼び出すプロセス。

名前

*probe::tcp.setsockopt.return – **setsockopt** からの戻り値*

概要

> *tcp.setsockopt.return*

値

ret

エラーコード (0: エラーなし)

name

このプローブの名前

コンテキスト

setsockopt を呼び出すプロセス。

名前

probe::udp.disconnect – プロセスが UDP の接続解除を要求したときに実行されます。

概要

```
udp.disconnect
```

値

daddr

宛先 IP アドレスを表す文字列

sock

プロセスにより使用されるネットワークソケット

saddr

ソース IP アドレスを表す文字列

sport

UDP ソースポート

flags

フラグ (FIN など)

dport

TCP 宛先ポート。

name

このプローブの名前

family

IP アドレスファミリー

コンテキスト

UDP の接続解除を要求するプロセス

名前
probe::udp.disconnect.return – UDP が正常に接続解除されました。

概要

```
udp.disconnect.return
```

値

saddr
ソース IP アドレスを表す文字列

sport
UDP ソースポート

dport
TCP 宛先ポート。

family
IP アドレスファミリー

name
このプローブの名前

daddr
宛先 IP アドレスを表す文字列

ret
エラーコード (0: エラーなし)

コンテキスト
UDP の接続解除を要求したプロセス

名前
probe::udp.recvmsg – UDP メッセージが受信されたときに必ず実行されます

概要

```
udp.recvmsg
```

値

size
プロセスが受信したバイト数

sock
プロセスにより使用されるネットワークソケット

daddr
宛先 IP アドレスを表す文字列

family
IP アドレスファミリー

name
このプローブの名前

dport
TCP 宛先ポート。

saddr
ソース IP アドレスを表す文字列

sport
UDP ソースポート

コンテキスト
UDP メッセージを受信したプロセス

名前
probe::udp.recvmsg.return – UDP メッセージの受信試行が完了したときに必ず実行されます

概要

```
udp.recvmsg.return
```

値

name
このプローブの名前

family
IP アドレスファミリー

dport
TCP 宛先ポート。

saddr
ソース IP アドレスを表す文字列

sport

UDP ソースポート

size

プロセスが受信したバイト数

daddr

宛先 IP アドレスを表す文字列

コンテキスト

UDP メッセージを受信したプロセス

名前

probe::udp.sendmsg – プロセスが UDP メッセージを送信するときに必ず実行されます。

概要

> *udp.sendmsg*

値

daddr

宛先 IP アドレスを表す文字列

sock

プロセスにより使用されるネットワークソケット

size

プロセスが送信したバイト数

saddr

ソース IP アドレスを表す文字列

sport

UDP ソースポート

family

IP アドレスファミリー

name

このプローブの名前

dport

TCP 宛先ポート。

コンテキスト

UDP メッセージを送信したプロセス

名前

probe::udp.sendmsg.return – UDP メッセージの送信試行が完了したときに必ず実行されます

概要

```
udp.sendmsg.return
```

値

size

プロセスが送信したバイト数

name

このプローブの名前

コンテキスト

UDP メッセージを送信したプロセス

第15章 ソケット TAPSET

この種類のプローブポイントは、ソケットの活動をプローブするために使用されます。以下のプローブポイントが含まれます。

名前

function::inet_get_ip_source – カーネルソケットの IP ソースアドレス文字列を提供する

概要

```
inet_get_ip_source:string(sock:long)
```

引数

sock

カーネルソケットへのポインター

名前

function::inet_get_local_port – カーネルソケットのローカルポート番号を指定する

概要

```
inet_get_local_port:long(sock:long)
```

引数

sock

カーネルソケットへのポインター

名前

function::sock_fam_num2str – プロトコルファミリー番号が指定される場合、文字列表示を返します。

概要

```
sock_fam_num2str:string(family:long)
```

引数

family

ファミリー番号。

名前

function::sock_fam_str2num – プロトコルファミリー名 (文字列) を指定すると、対応するプロトコルファミリー番号を返します

概要

```
sock_fam_str2num:long(family:string)
```

引数

family

ファミリー名。

名前

function::sock_prot_num2str – プロトコル番号が指定される場合、文字列表現を返します。

概要

```
sock_prot_num2str:string(proto:long)
```

引数

proto

プロトコル番号。

名前

function::sock_prot_str2num – プロトコル名 (文字列) が指定される場合、対応するプロトコル番号を返します。

概要

```
sock_prot_str2num:long(proto:string)
```

引数

proto

プロトコル名。

名前

function::sock_state_num2str – ソケット状態番号が指定される場合、文字列表現を返します。

概要

```
sock_state_num2str:string(state:long)
```

引数

state

状態番号。

名前

function::sock_state_str2num – ソケット状態文字列が指定される場合、対応する状態番号を返します。

概要

```
sock_state_str2num:long(state:string)
```

引数

state

状態名。

名前

probe::socket.aio_read – **sock_aio_read** からのメッセージの受信

概要

```
socket.aio_read
```

値

flags

ソケットフラグ値

type

ソケットタイプの値。

size

メッセージサイズ (バイト単位)

family

プロトコルファミリー値

name

このプローブの名前

protocol
プロトコル値

state
ソケット状態値

コンテキスト
メッセージの送信者
説明
***sock_aio_read** 関数からの、ソケットでのメッセージ受信の開始時に実行されます。*

名前
*probe::socket.aio_read.return – **sock_aio_read** から受信したメッセージの結論*

概要

```
socket.aio_read.return
```

値

family
プロトコルファミリー値

protocol
プロトコル値

name
このプローブの名前

state
ソケット状態値

success
正常に受信されたかどうか (1 = yes、0 = no)。(1 = yes、0 = no)

flags
ソケットフラグ値

type
ソケットタイプの値。

size
受信されるメッセージのサイズ (バイト単位) またはエラーコード (success = 0 の場合)

コンテキスト

メッセージ受信者。

説明

***sock_aio_read** 関数からの、ソケットでのメッセージ受信の終了時に実行されます。*

名前

*probe::socket.aio_write – メッセージは **sock_aio_write** から送信されます*

概要

```
socket.aio_write
```

値

flags

ソケットフラグ値

type

ソケットタイプの値。

size

メッセージサイズ(バイト単位)

family

プロトコルファミリー値

protocol

プロトコル値

name

このプローブの名前

state

ソケット状態値

コンテキスト

メッセージの送信者

説明

***sock_aio_write** 関数から、ソケットでのメッセージ送信の開始時に実行されます。*

名前

*probe::socket.aio_write.return – **sock_aio_write** からメッセージの結論を送信*

概要

```
socket.aio_write.return
```

値

state
ソケット状態値

success
正常に受信されたかどうか (1 = yes、0 = no)。(1 = yes、0 = no)

family
プロトコルファミリー値

protocol
プロトコル値

name
このプローブの名前

type
ソケットタイプの値。

size
受信されるメッセージのサイズ (バイト単位) またはエラーコード (success = 0 の場合)

flags
ソケットフラグ値

コンテキスト
メッセージ受信者。
説明
sock_aio_write 関数から、ソケットでのメッセージ送信の終了時に実行されます。

名前
probe::socket.close – ソケットを閉じます

概要

```
socket.close
```

値

type
ソケットタイプの値。

flags
ソケットフラグ値

state

ソケット状態値

family

プロトコルファミリー値

name

このプローブの名前

protocol

プロトコル値

コンテキスト

リクエスター(ユーザープロセスまたはカーネル)

説明

ソケットのクローズ開始時に実行されます。

名前

probe::socket.close.return – ソケットのクローズから返します。

概要

```
socket.close.return
```

値

name

このプローブの名前

コンテキスト

リクエスター(ユーザープロセスまたはカーネル)

説明

ソケットのクローズ終了時に実行されます。

名前

probe::socket.create – ソケットの作成。

概要

```
socket.create
```

値

type

ソケットタイプの値。

name

このプローブの名前

protocol

プロトコル値

family

プロトコルファミリー値

requester

ユーザープロセスまたはカーネルによる要求 (1 = kernel, 0 = user)

コンテキスト

リクエスター (リクエスター変数を参照)

説明

ソケットの作成開始時に実行されます。

名前

probe::socket.create.return – ソケットの作成から返します。

概要

```
socket.create.return
```

値

success

ソケットが正常に作成されたかどうか(1 = yes、0 = no)

family

プロトコルファミリー値

requester

ユーザープロセスまたはカーネルによる要求 (1 = kernel, 0 = user)

name

このプローブの名前

protocol

プロトコル値

type

ソケットタイプの値。

err

success == 0 の場合のエラーコード

コンテキスト
リクエスター(ユーザープロセスまたはカーネル)
説明
ソケットの作成終了時に実行されます

名前
*probe::socket.read_iter – **sock_read_iter** からのメッセージの受信*

概要

```
socket.read_iter
```

値

state
ソケット状態値

protocol
プロトコル値

name
このプローブの名前

family
プロトコルファミリー値

size
メッセージサイズ(バイト単位)

type
ソケットタイプの値。

flags
ソケットフラグ値

コンテキスト
メッセージの送信者
説明
***sock_read_iter** 関数からの、ソケットでのメッセージ受信の開始時に実行されます。*

名前
*probe::socket.read_iter.return – **sock_read_iter** から受信したメッセージの結論*

概要

socket.read_iter.return

値

flags
ソケットフラグ値

type
ソケットタイプの値。

size
受信されるメッセージのサイズ (バイト単位) またはエラーコード (success = 0 の場合)

family
プロトコルファミリー値

name
このプローブの名前

protocol
プロトコル値

state
ソケット状態値

success
正常に受信されたかどうか (1 = yes、0 = no)。(1 = yes、0 = no)

コンテキスト
メッセージ受信者。
説明
***sock_read_iter** 関数からの、ソケットでのメッセージ受信の終了時に実行されます。*

名前
probe::socket.readv – ***sock_readv** からのメッセージの受信*

概要

socket.readv

値

state
ソケット状態値

family

プロトコルファミリー値

protocol
プロトコル値

name
このプローブの名前

type
ソケットタイプの値。

size
メッセージサイズ (バイト単位)

flags
ソケットフラグ値

コンテキスト
メッセージの送信者
説明
***sock_readv** 関数からの、ソケットでのメッセージ受信の開始時に実行されます。*

名前
*probe::socket.readv.return – **sock_readv** からのメッセージの受信の結論*

概要

```
socket.readv.return
```

値

name
このプローブの名前

protocol
プロトコル値

family
プロトコルファミリー値

success
正常に受信されたかどうか (1 = yes、0 = no)。(1 = yes、0 = no)

state
ソケット状態値

flags
ソケットフラグ値

size
受信されるメッセージのサイズ (バイト単位) またはエラーコード (success = 0 の場合)

type
ソケットタイプの値。

コンテキスト
メッセージ受信者。
説明
***sock_readv** 関数からの、ソケットでのメッセージ受信の終了時に実行されます。*

名前
probe::socket.receive – ソケットで受信されたメッセージ

概要

```
socket.receive
```

値

name
このプローブの名前

protocol
プロトコル値

family
プロトコルファミリー値

success
正常に送信されたか ?(1 = yes、0 = no)

state
ソケット状態値

flags
ソケットフラグ値

size
受信されるメッセージのサイズ (バイト単位) またはエラーコード (success = 0 の場合)

type
ソケットタイプの値。

コンテキスト
メッセージ受信者

名前
probe::socket.recvmsg – メッセージをソケットで受信中です。

概要

```
socket.recvmsg
```

値

family
プロトコルファミリー値

name
このプローブの名前

protocol
プロトコル値

state
ソケット状態値

flags
ソケットフラグ値

type
ソケットタイプの値。

size
メッセージサイズ (バイト単位)

コンテキスト
メッセージ受信者。
説明
sock_recvmsg 関数からの、ソケットでのメッセージ受信の開始時に実行されます。

名前
probe::socket.recvmsg.return – ソケットで受信中のメッセージから返します

概要

```
socket.recvmsg.return
```

値

family
プロトコルファミリー値

name
このプローブの名前

protocol
プロトコル値

state
ソケット状態値

success
正常に受信されたかどうか (1 = yes、0 = no)。(1 = yes、0 = no)

flags
ソケットフラグ値

type
ソケットタイプの値。

size
受信されるメッセージのサイズ (バイト単位) またはエラーコード (success = 0 の場合)

コンテキスト
メッセージ受信者。
説明
***sock_recvmsg** 関数からの、ソケットでのメッセージ受信の終了時に実行されます。*

名前
probe::socket.send – ソケットで送信されたメッセージ。

概要

```
socket.send
```

値

flags
ソケットフラグ値

size
送信されるメッセージのサイズ (バイト単位) またはエラーコード (success = 0 の場合)

type
ソケットタイプの値。

protocol
プロトコル値

name
このプローブの名前

family
プロトコルファミリー値

success
正常に送信されたか?(1 = yes、0 = no)

state
ソケット状態値

コンテキスト
メッセージの送信者

名前
probe::socket.sendmsg – メッセージをソケットで送信中です。

概要

> *socket.sendmsg*

値

family
プロトコルファミリー値

name
このプローブの名前

protocol
プロトコル値

state
ソケット状態値

flags
ソケットフラグ値

type

ソケットタイプの値。

size
メッセージサイズ (バイト単位)

コンテキスト
メッセージの送信者
説明
***sock_sendmsg** 関数から、ソケットでのメッセージ送信の開始時に実行されます。*

名前
probe::socket.sendmsg.return – socket.sendmsg から返します。

概要

```
socket.sendmsg.return
```

値

type
ソケットタイプの値。

size
送信されるメッセージのサイズ (バイト単位) またはエラーコード (success = 0 の場合)

flags
ソケットフラグ値

state
ソケット状態値

success
正常に送信されたか ?(1 = yes、0 = no)

family
プロトコルファミリー値

protocol
プロトコル値

name
このプローブの名前

コンテキスト
メッセージの送信者。

説明
sock_sendmsg 関数から、ソケットでのメッセージ送信の終了時に実行されます。

名前
probe::socket.write_iter – メッセージは **sock_write_iter** から送信されます

概要

```
socket.write_iter
```

値

state
ソケット状態値

family
プロトコルファミリー値

protocol
プロトコル値

name
このプローブの名前

type
ソケットタイプの値。

size
メッセージサイズ (バイト単位)

flags
ソケットフラグ値

コンテキスト
メッセージの送信者
説明
sock_write_iter 関数から、ソケットでのメッセージ送信の開始時に実行されます。

名前
probe::socket.write_iter.return – **sock_write_iter** からメッセージの結論を送信

概要

```
socket.write_iter.return
```

値

type
ソケットタイプの値。

size
受信されるメッセージのサイズ (バイト単位) またはエラーコード (success = 0 の場合)

flags
ソケットフラグ値

state
ソケット状態値

success
正常に受信されたかどうか (1 = yes、0 = no)。(1 = yes、0 = no)

family
プロトコルファミリー値

protocol
プロトコル値

name
このプローブの名前

コンテキスト
メッセージ受信者。
説明
sock_write_iter 関数から、ソケットでのメッセージ送信の終了時に実行されます。

名前
probe::socket.writev – **socket_writev** から送信したメッセージ

概要

```
socket.writev
```

値

state
ソケット状態値

protocol
プロトコル値

name

このプローブの名前

family

プロトコルファミリー値

size

メッセージサイズ (バイト単位)

type

ソケットタイプの値。

flags

ソケットフラグ値

コンテキスト

メッセージの送信者

説明

***sock_writev** 関数から、ソケットでのメッセージ送信の開始時に実行されます。*

名前

*probe::socket.writev.return – **socket_writev** から送信したメッセージの結論*

概要

```
socket.writev.return
```

値

success

正常に送信されたか ?(1 = yes、0 = no)

state

ソケット状態値

name

このプローブの名前

protocol

プロトコル値

family

プロトコルファミリー値

size

送信されるメッセージのサイズ (バイト単位) またはエラーコード (success = 0 の場合)

type

ソケットタイプの値。

flags

ソケットフラグ値

コンテキスト

メッセージ受信者。

説明

***sock_writev** 関数から、ソケットでのメッセージ送信の終了時に実行されます。*

第16章 SNMP 情報タップセット

この一連のプローブポイントは、ソケットアクティビティーをプローブしてSNMP タイプ情報を提供するために使用されます。これには、次の関数とプローブポイントが含まれています。

名前

function::ipmib_filter_key – ipmib.* プローブのデフォルトのフィルター関数

概要

```
ipmib_filter_key:long(skb:long,op:long,SourceIsLocal:long)
```

引数

skb

構造体 sk_buff へのポインター

op

skb がフィルターを通過した場合にカウントされる値

ソースはローカル

1 はローカル操作、0 は非ローカル操作です。

説明

この関数はデフォルトのフィルター関数です。ユーザーは、この関数を独自のものに置き換えることができます。ユーザー指定のフィルター関数は、**skb** の値に基づいてインデックスキーを返します。戻り値 0 は、この特定の **skb** をカウントしないことを意味します。

名前

function::ipmib_get_proto – プロトコル値を取得する

概要

```
ipmib_get_proto:long(skb:long)
```

引数

skb

構造体 sk_buff へのポインター

説明

skb からプロトコル値を返します。

名前

function::ipmib_local_addr – ローカル IP アドレスを取得する

概要

```
ipmib_local_addr:long(skb:long,SourceIsLocal:long)
```

引数

skb
構造体 sk_buff へのポインター

ソースはローカル
ローカル操作かどうかを示すフラグ

説明

ローカル IP アドレス **skb** を返します。

名前

function::ipmib_remote_addr – リモート IP アドレスを取得する

概要

```
ipmib_remote_addr:long(skb:long,SourceIsLocal:long)
```

引数

skb
構造体 sk_buff へのポインター

ソースはローカル
ローカル操作かどうかを示すフラグ

説明

skb からリモート IP アドレスを返します。

名前

function::ipmib_tcp_local_port – ローカル TCP ポートを取得する

概要

```
ipmib_tcp_local_port:long(skb:long,SourceIsLocal:long)
```

引数

skb
構造体 sk_buff へのポインター

ソースはローカル

ローカル操作かどうかを示すフラグ

説明

***skb** からローカル tcp ポートを返します。*

名前

function::ipmib_tcp_remote_port – リモート TCP ポートを取得する

概要

```
ipmib_tcp_remote_port:long(skb:long,SourceIsLocal:long)
```

引数

skb

構造体 sk_buff へのポインター

ソースはローカル

ローカル操作かどうかを示すフラグ

説明

***skb** からリモート tcp ポートを返します。*

名前

function::linuxmib_filter_key – linuxmib. プローブのデフォルトのフィルター関数*

概要

```
linuxmib_filter_key:long(sk:long,op:long)
```

引数

sk

構造体 sock へのポインター

op

***sk** がフィルターを通過した場合にカウントされる値*

説明

*この関数はデフォルトのフィルター関数です。ユーザーは、この関数を独自のものに置き換えることができます。ユーザー指定のフィルター関数は、**sk** の値に基づいてインデックスキーを返します。戻り値 0 は、この特定の **sk** をカウントしないことを意味します。*

名前

function::tcpmib_filter_key – tcpmib. プローブのデフォルトのフィルター関数*

概要

```
tcpmib_filter_key:long(sk:long,op:long)
```

引数

sk

操作対象の構造体 sock へのポインター

op

sk *がフィルターを通過した場合にカウントされる値*

説明

この関数はデフォルトのフィルター関数です。ユーザーは、この関数を独自のものに置き換えることができます。ユーザー指定のフィルター関数は、 ***sk*** *の値に基づいてインデックスキーを返します。戻り値 0 は、この特定の* ***sk*** *をカウントしないことを意味します。*

名前

function::tcpmib_get_state – ソケットの状態を取得する

概要

```
tcpmib_get_state:long(sk:long)
```

引数

sk

構造体 sock へのポインター

説明

構造体 sock から sk_state を返します。

名前

function::tcpmib_local_addr – 送信元アドレスを取得する

概要

```
tcpmib_local_addr:long(sk:long)
```

引数

sk

構造体 inet_sock へのポインター

説明
ホスト順で struct inet_sock から saddr を返します。

名前
function::tcpmib_local_port – ローカルポートを取得する

概要

```
tcpmib_local_port:long(sk:long)
```

引数

sk
構造体 inet_sock へのポインター

説明
ホスト順で struct inet_sock からスポーツを返します。

名前
function::tcpmib_remote_addr – リモートアドレスを取得する

概要

```
tcpmib_remote_addr:long(sk:long)
```

引数

sk
構造体 inet_sock へのポインター

説明
ホスト順に struct inet_sock から daddr を返します。

名前
function::tcpmib_remote_port – リモートポートを取得する

概要

```
tcpmib_remote_port:long(sk:long)
```

引数

sk
構造体 inet_sock へのポインター

説明
ホスト順に struct inet_sock から dport を返します。

名前
probe::ipmib.ForwDatagrams – 転送されたパケットを数える

概要

```
ipmib.ForwDatagrams
```

値

op
カウンターに追加する値 (デフォルト値 1)

skb
作用する構造体 sk_buff へのポインター

説明
skb が指すパケットは、関数によってフィルタリングされます。**ipmib_filter_key** .パケットが通過した場合、フィルターはグローバル **ForwDatagrams** (SNMP の MIB IPSTATS_MIB_OUTFORWDATAGRAMS に相当) でカウントされます。

名前
probe::ipmib.FragFails – フラグメント化に失敗したデータグラムのカウント

概要

```
ipmib.FragFails
```

値

op
カウンターに追加する値 (デフォルト値 1)

skb
作用する構造体 sk_buff へのポインター

説明
skb が指すパケットは、関数によってフィルタリングされます。**ipmib_filter_key** .パケットが通過すると、フィルターはグローバル **FragFails** (SNMP の MIB IPSTATS_MIB_FRAGFAILS に相当) でカウントされます。

名前
probe::ipmib.FragOKs – 正常にフラグメント化されたデータグラムをカウントする

概要

```
ipmib.FragOKs
```

値

skb

作用する構造体 sk_buff へのポインター

op

カウンターに追加する値 (デフォルト値 1)

説明

skb が指すパケットは、関数によってフィルタリングされます。**ipmib_filter_key** .パケットが通過した場合、フィルターはグローバル **FragOK** でカウントされます (SNMP の MIB IPSTATS_MIB_FRAGOKS に相当)。

名前

probe::ipmib.InAddrErrors – アドレスが正しくない到着パケットをカウントする

概要

```
ipmib.InAddrErrors
```

値

skb

作用する構造体 sk_buff へのポインター

op

カウンターに追加する値 (デフォルト値 1)

説明

skb が指すパケットは、関数によってフィルタリングされます。**ipmib_filter_key** .パケットが通過すると、フィルターはグローバル **InAddrErrors** (SNMP の MIB IPSTATS_MIB_INADDRERRORS に相当) でカウントされます。

名前

probe::ipmib.InDiscards – 破棄されたインバウンドパケットのカウント

概要

```
ipmib.InDiscards
```

値

op

.

カウンターに追加する値 (デフォルト値 1)

skb

作用する構造体 sk_buff へのポインター

説明

skb *が指すパケットは、関数によってフィルタリングされます。* ***ipmib_filter_key*** *.パケットが通過した場合、フィルターはグローバル* ***InDiscards*** *(SNMP の MIB STATS_MIB_INDISCARDS に相当) でカウントされます。*

名前

probe::ipmib.InNoRoutes – 一致するソケットがない到着パケットをカウントする

概要

```
ipmib.InNoRoutes
```

値

op

カウンターに追加する値 (デフォルト値 1)

skb

作用する構造体 sk_buff へのポインター

説明

skb *が指すパケットは、関数によってフィルタリングされます。* ***ipmib_filter_key*** *.パケットが通過すると、フィルターはグローバル* ***InNoRoutes*** *(SNMP の MIB IPSTATS_MIB_INNOROUTES に相当) でカウントされます。*

名前

probe::ipmib.InReceives – 到着パケットをカウントする

概要

```
ipmib.InReceives
```

値

skb

作用する構造体 sk_buff へのポインター

op

カウンターに追加する値 (デフォルト値 1)

説明

skb が指すパケットは、関数によってフィルタリングされます。**ipmib_filter_key** .パケットが通過すると、フィルターはグローバル **InReceives** でカウントされます (SNMP の MIB IPSTATS_MIB_INRECEIVES に相当)。

名前

probe::ipmib.InUnknownProtos – バインドされていないプロトコルで到着パケットをカウントする

概要

```
ipmib.InUnknownProtos
```

値

skb

作用する構造体 sk_buff へのポインター

op

カウンターに追加する値 (デフォルト値 1)

説明

skb が指すパケットは、関数によってフィルタリングされます。**ipmib_filter_key** .パケットが通過すると、フィルターはグローバル **InUnknownProtos** (SNMP の MIB IPSTATS_MIB_INUNKNOWNPROTOS に相当) でカウントされます。

名前

probe::ipmib.OutRequests – パケット送信要求をカウントする

概要

```
ipmib.OutRequests
```

値

skb

作用する構造体 sk_buff へのポインター

op

カウンターに追加する値 (デフォルト値 1)

説明

skb が指すパケットは、関数によってフィルタリングされます。**ipmib_filter_key** .パケットが通過すると、フィルターはグローバル **OutRequests** (SNMP の MIB IPSTATS_MIB_OUTREQUESTS に相当) でカウントされます。

名前
probe::ipmib.ReasmReqds — パケットフラグメントの再設定要求の数をカウントする

概要

```
ipmib.ReasmReqds
```

値

op
カウンターに追加する値 (デフォルト値 1)

skb
作用する構造体 sk_buff へのポインター

説明
skb が指すパケットは、関数によってフィルタリングされます。**ipmib_filter_key** .パケットが通過すると、フィルターはグローバル **ReasmReqds** (SNMP の MIB IPSTATS_MIB_REASMREQDS に相当) でカウントされます。

名前
probe::ipmib.ReasmTimeout — 再設定タイムアウトのカウント

概要

```
ipmib.ReasmTimeout
```

値

op
カウンターに追加する値 (デフォルト値 1)

skb
作用する構造体 sk_buff へのポインター

説明
skb が指すパケットは、関数によってフィルタリングされます。**ipmib_filter_key** .パケットが通過すると、フィルターはグローバル **ReasmTimeout** でカウントされます (SNMP の MIB IPSTATS_MIB_REASMTIMEOUT に相当)。

名前
probe::linuxmib.DelayedACKs — 遅延 ACK の数

概要

```
linuxmib.DelayedACKs
```

値

op

カウンターに追加する値 (デフォルト値 1)

sk

操作対象の構造体 sock へのポインター

説明

skb が指すパケットは、関数によってフィルタリングされます。**linuxmib_filter_key** .パケットが通過した場合、フィルターはグローバル **DelayedACKs** (SNMP の MIB LINUX_MIB_DELAYEDACKS に相当) でカウントされます。

名前

probe::linuxmib.ListenDrops – ドロップされた接続要求の回数

概要

```
linuxmib.ListenDrops
```

値

sk

操作対象の構造体 sock へのポインター

op

カウンターに追加する値 (デフォルト値 1)

説明

skb が指すパケットは、関数によってフィルタリングされます。**linuxmib_filter_key** .パケットが通過すると、フィルターはグローバルな **ListenDrops** (SNMP の MIB LINUX_MIB_LISTENDROPS に相当) でカウントされます。

名前

probe::linuxmib.ListenOverflows – リッスンキューがオーバーフローした回数

概要

```
linuxmib.ListenOverflows
```

値

sk

操作対象の構造体 sock へのポインター

op

カウンターに追加する値 (デフォルト値 1)

説明

skb *が指すパケットは、関数によってフィルタリングされます。* ***linuxmib_filter_key*** *.パケットが通過した場合、フィルターはグローバルな* ***ListenOverflows*** *(SNMP の MIB LINUX_MIB_LISTENOVERFLOWS に相当) でカウントされます。*

名前

probe::linuxmib.TCPMemoryPressures – メモリー不足が使用された回数

概要

```
linuxmib.TCPMemoryPressures
```

値

sk

操作対象の構造体 sock へのポインター

op

カウンターに追加する値 (デフォルト値 1)

説明

skb *が指すパケットは、関数によってフィルタリングされます。* ***linuxmib_filter_key*** *.パケットが通過すると、フィルターはグローバル* ***TCPMemoryPressures*** *(SNMP の MIB LINUX_MIB_TCPMEMORYPRESSURES に相当) でカウントされます。*

名前

probe::tcpmib.ActiveOpens – ソケットのアクティブな開始をカウントする

概要

```
tcpmib.ActiveOpens
```

値

op

カウンターに追加する値 (デフォルト値 1)

sk

操作対象の構造体 sock へのポインター

説明

skb *が指すパケットは、関数によってフィルタリングされます。* ***tcpmib_filter_key*** *.パケットが通過した場合、フィルターはグローバル* ***ActiveOpens*** *(SNMP の MIB TCP_MIB_ACTIVEOPENS に相当) でカウントされます。*

名前
probe::tcpmib.AttemptFails – ソケットを開こうとして失敗した回数を数える

概要

```
tcpmib.AttemptFails
```

値

op

カウンターに追加する値 (デフォルト値 1)

sk

操作対象の構造体 sock へのポインター

説明
skb が指すパケットは、関数によってフィルタリングされます。**tcpmib_filter_key**.パケットが通過した場合、フィルターはグローバル **AttemptFails** (SNMP の MIB TCP_MIB_ATTEMPTFAILS に相当) でカウントされます。

名前
probe::tcpmib.CurrEstab – 開いているソケットの数を更新する

概要

```
tcpmib.CurrEstab
```

値

sk

操作対象の構造体 sock へのポインター

op

カウンターに追加する値 (デフォルト値 1)

説明
skb が指すパケットは、関数によってフィルタリングされます。**tcpmib_filter_key**.パケットが通過すると、フィルターはグローバル **CurrEstab** でカウントされます (SNMP の MIB TCP_MIB_CURRESTAB と同等)。

名前
probe::tcpmib.EstabResets – ソケットのリセットを数える

概要

tcpmib.EstabResets

値

sk

操作対象の構造体 sock へのポインター

op

カウンターに追加する値 (デフォルト値 1)

説明

skb *が指すパケットは、関数によってフィルタリングされます。* ***tcpmib_filter_key*** *.パケットが通過すると、フィルターはグローバル* ***EstabResets*** *(SNMP の MIB TCP_MIB_ESTABRESETS に相当) でカウントされます。*

名前

probe::tcpmib.InSegs – 着信 tcp セグメントをカウントする

概要

tcpmib.InSegs

値

op

カウンターに追加する値 (デフォルト値 1)

sk

操作対象の構造体 sock へのポインター

説明

skb *が指すパケットは、関数によってフィルタリングされます。* ***tcpmib_filter_key*** *(または* ***ipmib_filter_key*** *TCP v4 の場合)。パケットが通過した場合、フィルターはグローバル* ***InSeg*** *でカウントされます (SNMP の MIB TCP_MIB_INSEGS に相当)*

名前

probe::tcpmib.OutRsts – リセットパケットの送信をカウントする

概要

tcpmib.OutRsts

値

sk

操作対象の構造体 sock へのポインター

op

カウンターに追加する値 (デフォルト値 1)

説明

***skb** が指すパケットは、関数によってフィルタリングされます。**tcpmib_filter_key**.パケットが通過すると、フィルターはグローバル **OutRsts** (SNMP の MIB TCP_MIB_OUTRSTS に相当) でカウントされます。*

名前

probe::tcpmib.OutSegs – TCP セグメントの送信をカウントする

概要

```
tcpmib.OutSegs
```

値

sk

操作対象の構造体 sock へのポインター

op

カウンターに追加する値 (デフォルト値 1)

説明

***skb** が指すパケットは、関数によってフィルタリングされます。**tcpmib_filter_key**.パケットが通過すると、フィルターはグローバル **OutSegs** (SNMP の MIB TCP_MIB_OUTSEGS に相当) でカウントされます。*

名前

probe::tcpmib.PassiveOpens – ソケットの受動的な作成を数える

概要

```
tcpmib.PassiveOpens
```

値

sk

操作対象の構造体 sock へのポインター

op

カウンターに追加する値 (デフォルト値 1)

説明

skb *が指すパケットは、関数によってフィルタリングされます。* ***tcpmib_filter_key*** *.パケットが通過すると、フィルターはグローバル* ***PassiveOpens*** *(SNMP の MIB TCP_MIB_PASSIVEOPENS に相当) でカウントされます。*

名前

probe::tcpmib.RetransSegs – TCP セグメントの再送信をカウントする

概要

```
tcpmib.RetransSegs
```

値

op

カウンターに追加する値 (デフォルト値 1)

sk

操作対象の構造体 sock へのポインター

説明

skb *が指すパケットは、関数によってフィルタリングされます。* ***tcpmib_filter_key*** *.パケットが通過した場合、フィルターはグローバル* ***RetransSegs*** *(SNMP の MIB TCP_MIB_RETRANSSEGS に相当) でカウントされます。*

第17章 カーネルプロセス TAPSET

この種類のプローブポイントは、プロセス関連のアクティビティーをプローブするために使用されます。以下のプローブポイントが含まれます。

名前

function::get_loadavg_index – 指定した間隔の負荷平均を取得する

概要

```
get_loadavg_index:long(indx:long)
```

引数

indx

キャプチャする負荷平均間隔。

説明

この関数は、指定された間隔での負荷平均を返します。3 つの負荷平均値 1、5、および 15 分平均は、avenrun 配列のインデックス 0、1、および 2 に対応します。linux/sched.h を参照してください。負荷平均の切り捨てられた整数部分が返されることに注意してください。指定されたインデックスが範囲外の場合、エラーメッセージと例外が出力されます。

名前

function::sprint_loadavg – きれいに印刷された負荷平均を報告する

概要

```
sprint_loadavg:string()
```

引数

なし

説明

1 分、5 分、15 分の負荷平均の通常の形式で 3 つの 10 進数を含む文字列を返します。

名前

function::target_set_pid – pid はターゲットプロセスから派生していますか?

概要

```
target_set_pid(pid:)
```

引数

pid

照会するプロセスの pid

説明
この関数は、指定されたプロセス ID が「ターゲットセット」 内にあるかどうか、つまり最上位の子孫であるかどうかを返します。**target**処理する。

名前
function::target_set_report – ターゲットセットに関するレポートを印刷する

概要

```
target_set_report()
```

引数
なし
説明
この関数は、ターゲットセット内のプロセスとその祖先に関するレポートを出力します。

名前
probe::kprocess.create – 新しいプロセスまたはスレッドが正常に作成されるたびに起動します

概要

```
kprocess.create
```

値

new_tid

新しく作成されたタスクの TID

new_pid

新規に作成されたプロセスの PID。

コンテキスト
作成されたプロセスの親。
説明
フォーク (またはシステムコールのいずれか) または新しいカーネルスレッドの結果、新規プロセスが正常に作成されたときに必ず実行されます。

名前
probe::kprocess.exec – 新しいプログラムの実行の試行。

概要

```
kprocess.exec
```

値

filename

新しい実行可能ファイルのパス。

name

システムコールの名前 (「execve」) (SystemTap v2.5+)

args

0 番目の引数を含む、新しい実行可能ファイルに渡す引数 (SystemTap v2.5+)

argstr

0 番目の引数を除く、渡す引数が続くファイル名を含む文字列 (SystemTap v2.5+)

コンテキスト

exec の呼び出し元。

説明

プロセスが新規プログラムの実行を試みるたびに実行されます。SystemTap v2.5+ では syscall.execve プローブにエイリアスされています。

名前

probe::kprocess.exec_complete — 新しいプログラムの実行から返されます。

概要

```
kprocess.exec_complete
```

値

retstr

errno の文字列表現 (SystemTap v2.5+)

success

実行が成功したかどうかを示すブール値。

errno

実行の結果返されたエラー番号。

name

システムコールの名前 (「execve」) (SystemTap v2.5+)

コンテキスト

成功した場合は、新しい実行可能ファイルのコンテキスト。失敗した場合は、呼び出し元のコンテキストに留まります。

説明

実行呼び出しの完了時に実行されます。SystemTap v2.5+ では syscall.execve.return プローブにエイリアスされています。

名前
probe::kprocess.exit – プロセスを終了します。

概要

```
kprocess.exit
```

値

code
プロセスの終了コード。

コンテキスト
終了するプロセス。
説明
プロセスが終了すると実行されます。この後で kprocess.release が常に実行されます。ただし、プロセスがゾンビ状態で待機している場合、kprocess.release の実行は遅延することがあります。

名前
probe::kprocess.release – リリースされるプロセス。

概要

```
kprocess.release
```

値

released_tid
解放されるタスクの TID

task
リリースされるプロセスのタスクハンドル。

released_pid
リリースされるプロセスの PID。

pid
*互換性のために **released_pid** と同じ (非推奨)*

コンテキスト
親のコンテキスト (このプロセスの終了を通知したい場合。それ以外の場合は、プロセス自体のコンテキスト)。
説明
プロセスがカーネルからリリースされると実行されます。これは kprocess.exit に従うことがありますが、プロセスが異常状態で待機している場合に若干遅れる可能性があります。

名前
probe::kprocess.start — 新規プロセスを開始します。

概要

```
kprocess.start
```

値
なし
コンテキスト
新規に作成されたプロセス。
説明
新規プロセスが実行を開始する直前に実行されます。

第18章 シグナル TAPSET

この種類のプローブポイントは、シグナルアクティビティーをプローブするために使用されます。以下のプローブポイントが含まれます。

名前

function::get_sa_flags – sa_flags の数値を返します

概要

```
get_sa_flags:long(act:long)
```

引数

行為

クエリーする sigaction のアドレス。

名前

function::get_sa_handler – sa_handler の数値を返します

概要

```
get_sa_handler:long(act:long)
```

引数

行為

クエリーする sigaction のアドレス。

名前

function::is_sig_blocked – シグナルが現在ブロックされている場合は 1 を返し、ブロックされていない場合は 0 を返します。

概要

```
is_sig_blocked:long(task:long,sig:long)
```

引数

task

クエリーする task_struct のアドレス。

SIG

テストするシグナル番号。

名前
function::sa_flags_str – sa_flags の文字列表現を返します

概要

```
sa_flags_str:string(sa_flags:long)
```

引数

sa_flags
文字列に変換するフラグのセット。

名前
function::sa_handler_str – sa_handler の文字列表現を返します

概要

```
sa_handler_str(handler:)
```

引数

handler
文字列に変換する sa_handler。

説明
sa_handler の文字列表現を返します。SIG_DFL、SIG_IGN、SIG_ERR のいずれでもない場合は、ハンドラーのアドレスを返します。

名前
function::signal_str – シグナル番号の文字列表現を返します

概要

```
signal_str(num:)
```

引数

num
文字列に変換するシグナル番号。

名前

function::sigset_mask_str – sigset の文字列表現を返します

概要

```
sigset_mask_str:string(mask:long)
```

引数

mask
文字列に変換する sigset。

名前
probe::signal.check_ignored – シグナルが無視されたことを確認します。

概要

```
signal.check_ignored
```

値

sig_pid
シグナルを受信するプロセスの PID

SIG
シグナルの番号

sig_name
シグナルの文字列表現

pid_name
シグナルを受信するプロセスの名前

名前
probe::signal.check_ignored.return – シグナルが無視されたことの確認が完了しました。

概要

```
signal.check_ignored.return
```

値

name
プローブポイントの名前

retstr
値を文字列として返します。

名前

probe::signal.checkperm – 送信されたシグナルについての確認が実行されます。

概要

```
signal.checkperm
```

値

pid_name
シグナルを受信するプロセスの名前

task
シグナル受信者のタスクハンドル

sig_name
シグナルの文字列表現

sinfo
siginfo 構造のアドレス

name
プローブポイントの名前

SIG
シグナルの番号

si_code
シグナルタイプを示します。

sig_pid
シグナルを受信するプロセスの PID

名前

probe::signal.checkperm.return – 送信されたシグナルの確認実行が完了しました。

概要

```
signal.checkperm.return
```

値

retstr
値を文字列として返します。

name
プローブポイントの名前

名前
probe::signal.do_action – シグナルアクションを調査または変更します。

概要

```
signal.do_action
```

値

sigact_addr
シグナルに関連付けられた新しい sigaction 構造のアドレス。

sig_name
シグナルの文字列表現

sa_mask
シグナルの新しいマスク。

sa_handler
シグナルの新しいハンドラー。

oldsigact_addr
シグナルに関連付けられた古い sigaction 構造のアドレス

SIG
調査または変更するシグナル。

name
プローブポイントの名前

名前
probe::signal.do_action.return – シグナルアクションの調査または変更が完了しました。

概要

```
signal.do_action.return
```

値

retstr
値を文字列として返します。

name
プローブポイントの名前

名前
probe::signal.flush – タスクのすべての保留中シグナルを破棄します。

概要

```
signal.flush
```

値

task
破棄を実行するプロセスのタスクハンドラー

pid_name
破棄を実行するタスクに関連付けられたプロセスの名前

name
プローブポイントの名前

sig_pid
破棄を実行するタスクに関連付けられたプロセスの PID

名前
probe::signal.force_segv – IGSEGV の送信を強制実行します

概要

```
signal.force_segv
```

値

sig_name
シグナルの文字列表現

pid_name
シグナルを受信するプロセスの名前

sig_pid
シグナルを受信するプロセスの PID

name
プローブポイントの名前

SIG
シグナルの番号

名前
probe::signal.force_segv.return — SIGSEGV の送信の強制実行が完了しました。

概要

```
signal.force_segv.return
```

値

retstr
値を文字列として返します。

name
プローブポイントの名前

名前
probe::signal.handle — 呼び出されるシグナルハンドラー。

概要

```
signal.handle
```

値

name
プローブポイントの名前

SIG
シグナルハンドラーを呼び出したシグナル番号。

sinfo

siginfo テーブルのアドレス。

ka_addr

シグナルに関連付けられた k_sigaction テーブルのアドレス

sig_mode

シグナルがユーザーモードシグナルであるか、またはカーネルモードシグナルであるかを示します。

sig_code

siginfo シグナルの si_code 値。

登録

カーネルモードのスタック領域のアドレス (SystemTap 2.1 で非推奨)

oldset_addr

ブロックされたシグナルのビットマスク配列のアドレス (SystemTap 2.1 で非推奨)

sig_name

シグナルの文字列表現

名前

probe::signal.handle.return – シグナルハンドラーの呼び出しが完了しました。

概要

```
signal.handle.return
```

値

retstr

値を文字列として返します。

name

プローブポイントの名前

説明

(SystemTap 2.1 で非推奨)

名前

probe::signal.pending – 保留中シグナルの調査

概要

```
signal.pending
```

値

name
プローブポイントの名前

sigset_size
ユーザー空間シグナルセットのサイズ

sigset_add
ユーザー空間シグナルセットのアドレス (sigset_t)

説明
このプローブは、特定のスレッドへの配信を保留しているシグナルのセットを調べるために使用されます。これは通常、do_sigpending カーネル関数を実行する際に発生します。

名前
probe::signal.pending.return – 保留中のシグナルの調査が完了しました。

概要

```
signal.pending.return
```

値

name
プローブポイントの名前

retstr
値を文字列として返します。

名前
probe::signal.procmask – ブロックされたシグナルを調査または変更します。

概要

```
signal.procmask
```

値

name
プローブポイントの名前

sigset

sigset_t に設定する実際の値 (正しいか ?)。

how

ブロックされたシグナルを変更する方法を示します。値は SIG_BLOCK=0 (シグナルをブロックする場合)、SIG_UNBLOCK=1 (シグナルをブロック解除する場合)、および SIG_SETMASK=2 (シグナルマスクを設定する場合) です。

sigset_addr

実装するシグナルセット (sigset_t) のアドレス。

oldsigset_addr

シグナルセット (sigset_t) の古いアドレス。

名前

probe::signal.procmask.return – ブロックされたシグナルの調査または変更が完了しました。

概要

```
signal.procmask.return
```

値

retstr

値を文字列として返します。

name

プローブポイントの名前

名前

probe::signal.send – プロセスに送信されるシグナル。

概要

```
signal.send
```

値

send2queue

シグナルが既存の sigqueue に送信されるかどうかを示します (SystemTap 2.1 で非推奨)

pid_name

シグナル受信者の名前

task

シグナル受信者のタスクハンドル

sig_name
シグナルの文字列表現

sinfo
siginfo 構造のアドレス

shared
シグナルがスレッドグループで共有されているかどうかを示します。

si_code
シグナルタイプを示します。

name
シグナルを送信するために使用される関数の名前。

SIG
シグナルの番号

sig_pid
シグナルを受信するプロセスの PID

コンテキスト
シグナルの送信者。

名前
probe::signal.send.return – 完了したプロセスに送信されるシグナル (SystemTap 2.1 で非推奨)

概要

```
signal.send.return
```

値

shared
送信されたシグナルがスレッドグループで共有されているかどうかを示します。

name
シグナルを送信するために使用される関数の名前。

retstr
__group_send_sig_info、specific_send_sig_info、または send_sigqueue への戻り値

send2queue
送信されたシグナルが既存の sigqueue に送信されたかどうかを示します。

コンテキスト

シグナルの送信者 (正しいか ?)。

説明

__group_send_sig_info と specific_send_sig_info の戻り値は以下のとおりです。
0 -- シグナルはプロセスに正常に送信されました。つまり、(1) シグナルは受信プロセスによって無視されました。(2) これは非 RT シグナルであり、システムにはすでにキューに入れられたシグナルがあります。(3) 受信プロセスの sigqueue にシグナルが正常に追加されました。
*-EAGAIN -- 受信側プロセスの sigqueue はオーバーフロー状態です。シグナルは RT であり、**kill** 以外の関数を使用しているユーザーによって送信されました。*
send_group_sigqueue と send_sigqueue の戻り値は以下のとおりです。
0 -- シグナルは受信側プロセスの sigqueue に正常に追加されました。または、SI_TIMER エントリーがすでにキューに格納されています (この場合、オーバーランした数は単純に増分されます)。
1 -- シグナルが受信側プロセスによって無視されました。
-1 -- (send_sigqueue のみ) タスクは終了中とマークされ、 posix_timer_event をグループリーダーにリダイレクトすることが許可されます。*

名前

probe::signal.send_sig_queue – シグナルをプロセスのキューに格納します。

概要

```
signal.send_sig_queue
```

値

SIG

キューに格納されたシグナル

name

プローブポイントの名前

sig_pid

シグナルがキューに格納されるプロセスの PID

pid_name

シグナルがキューに格納されるプロセスの名前

sig_name

シグナルの文字列表現

sigqueue_addr

シグナルキューのアドレス

名前

probe::signal.send_sig_queue.return – プロセスのキューへのシグナルの格納が完了しました。

概要

```
signal.send_sig_queue.return
```

値

retstr
値を文字列として返します。

name
プローブポイントの名前

名前
probe::signal.sys_tgkill – スレッドグループに kill シグナルを送信します。

概要

```
signal.sys_tgkill
```

値

sig_pid
kill シグナルを受信するスレッドの PID

SIG
プロセスに送信される特定の kill シグナル。

name
プローブポイントの名前

pid_name
シグナル受信者の名前

sig_name
シグナルの文字列表現

tgid
kill シグナルを受信するスレッドのスレッドグループ ID

task
シグナル受信者のタスクハンドル

説明
tgkill 呼び出しは tkill に似ています。ただし、呼び出し元は、シグナルを送信するスレッドのスレッドグループ ID を指定できます。これにより、TID の再利用を回避できます。

名前

probe::signal.sys_tgkill.return – スレッドグループへの kill シグナルの送信が完了しました。

概要

```
signal.sys_tgkill.return
```

値

name

プローブポイントの名前

retstr

__group_send_sig_info に対する戻り値。

名前

probe::signal.sys_tkill – スレッドに kill シグナルを送信します。

概要

```
signal.sys_tkill
```

値

sig_pid

kill シグナルを受信するプロセスの PID

SIG

プロセスに送信される特定のシグナル

name

プローブポイントの名前

pid_name

シグナル受信者の名前

sig_name

シグナルの文字列表現

task

シグナル受信者のタスクハンドル

説明

tkill 呼び出しは kill(2) と似ていますが、特定のスレッドグループ内のプロセスを対象にすることができます。このようなプロセスは、一意のスレッド ID (TID) でターゲットとなります。

名前

probe::signal.syskill – プロセスに kill シグナルを送信します。

概要

```
signal.syskill
```

値

sig_pid

シグナルを受信するプロセスの PID

SIG

プロセスに送信される特定のシグナル

name

プローブポイントの名前

pid_name

シグナル受信者の名前

sig_name

シグナルの文字列表現

task

シグナル受信者のタスクハンドル

名前

probe::signal.syskill.return – kill シグナルの送信が完了しました。

概要

```
signal.syskill.return
```

値

なし

名前

probe::signal.systkill.return – スレッドへの kill シグナルの送信が完了しました。

概要

signal.systkill.return

値

retstr

__group_send_sig_info に対する戻り値。

name

プローブポイントの名前

名前

probe::signal.wakeup – シグナルによりウェイクするスリープ状態のプロセス

概要

signal.wakeup

値

pid_name

ウェイクするプロセスの名前。

再開

STOPPED または TRACED 状態のタスクをウェイクアップするかどうかを示します。

state_mask

ウェイクするタスク状態のマスクを示す文字列表現。可能な値は、TASK_INTERRUPTIBLE、TASK_STOPPED、TASK_TRACED、TASK_WAKEKILL、および TASK_INTERRUPTIBLE です。

sig_pid

ウェイクするプロセスの PID

第19章 ERRNO タップセット

この一連の関数は、errno 数値を処理するために使用されます。次の機能が含まれています。

名前

function::errno_str – エラーコードに関連付けられた記号文字列

概要

```
errno_str:string(err:long)
```

引数

err

受信したエラー番号

説明

この関数は、数値 2 の場合は ENOENT、3333 などの範囲外の値の場合は E#3333 など、エラーコードに関連付けられたシンボリック文字列を返します。

名前

function::return_str – 戻り値を文字列としてフォーマットします

概要

```
return_str:string(format:long,ret:long)
```

引数

format

戻り値の型のベース値を決定する変数

ret

戻り値 (通常**$return**)

説明

この関数は syscall タップセットによって使用され、文字列を返します。format を 10 進数の場合は 1、16 進数の場合は 2、8 進数の場合は 3 に設定します。
この関数は優先されることに注意してください**returnstr**.

名前

function::returnstr – 戻り値を文字列としてフォーマットします

概要

```
returnstr:string(format:long)
```

引数

format

戻り値の型のベース値を決定する変数

説明

この関数は nd_syscall タップセットで使用され、文字列を返します。format を 10 進数の場合は 1、16 進数の場合は 2、8 進数の場合は 3 に設定します。
この関数は、dwarfless プローブ (つまり、'kprobe.function (「foo 」)') でのみ使用する必要があることに注意してください。他のプローブは使用する必要があります**return_str**.

名前

function::returnval – プローブされた関数の可能な戻り値

概要

```
returnval:long()
```

引数

なし

説明

関数値が通常返されるレジスタの値を返します。プローブで使用できます。**$return**利用できません。これは実際の戻り値の推測にすぎず、完全に間違っている可能性があります。通常、ドワーフレスプローブでのみ使用されます。

第20章 RLIMIT タップセット

この一連の関数は、リソース制限 (RLIMIT_*) を定義し、対応するリソース制限の数を返す文字列を処理するために使用されます。次の機能が含まれています。

名前

function::rlimit_from_str – リソース制限コードに関連付けられた記号文字列

概要

```
rlimit_from_str:long(lim_str:string)
```

引数

lim_str

limit の文字列表現

説明

この関数は、文字列 RLIMIT_CPU の場合は 0、範囲外の値の場合は -1 など、指定された文字列に関連付けられた数値を返します。

第21章 デバイスタップセット

この一連の関数は、カーネルおよびユーザー空間のデバイス番号を処理するために使用されます。次の機能が含まれています。

名前

function::MAJOR – カーネルデバイス番号からメジャーデバイス番号を抽出 (kdev_t)

概要

```
MAJOR:long(dev:long)
```

引数

dev

問い合わせるカーネルデバイス番号。

名前

function::MINOR – カーネルデバイス番号からマイナーデバイス番号を抽出 (kdev_t)

概要

```
MINOR:long(dev:long)
```

引数

dev

問い合わせるカーネルデバイス番号。

名前

function::MKDEV – カーネルデバイス番号 (kdev_t) と比較できる値を作成します

概要

```
MKDEV:long(major:long,minor:long)
```

引数

major

意図したメジャーデバイス番号。

minor

意図したマイナーデバイス番号。

名前

function::usrdev2kerndev – ユーザー空間のデバイス番号をカーネルで使用される形式に変換します

概要

```
usrdev2kerndev:long(dev:long)
```

引数

dev

ユーザー空間形式のデバイス番号。

第22章 DIRECTORY-ENTRY (DENTRY) TAPSET

この種類の関数は、ファイルまたは完全パス名にカーネル VFS ディレクトリーのエントリーポインターをマップするために使用されます。

名前

function::d_name – dirent 名を取得します。

概要

```
d_name:string(dentry:long)
```

引数

dentry

dentry へのポインター。

説明

dirent 名 (パスベース名) を返します。

名前

function::d_path – 完全 nameidata パスを取得します。

概要

```
d_path:string(nd:long)
```

引数

nd

nameidata へのポインター。

説明

kernel d_path 関数などの完全 dirent 名を返します (root への完全パス)。

名前

function::fullpath_struct_file – フルパスを取得する

概要

```
fullpath_struct_file:string(task:long,file:long)
```

引数

task

task_struct ポインター

file

「構造体ファイル」 へのポインター。

説明

kernel d_path 関数などの完全 dirent 名を返します (root への完全パス)。

名前

function::fullpath_struct_nameidata — 完全 nameidata パスを取得します。

概要

```
fullpath_struct_nameidata(nd:)
```

引数

nd

「構造体 nameidata」 へのポインター。

説明

カーネル (および systemtap-tapset) d_path 関数のように、「/」 を使用して、完全な dirent 名 (ルートへの完全パス) を返します。

名前

function::fullpath_struct_path — フルパスを取得する

概要

```
fullpath_struct_path:string(path:long)
```

引数

path

「構造体パスへ」 のポインター。

説明

kernel d_path 関数などの完全 dirent 名を返します (root への完全パス)。

名前

function::inode_name — inode 名を取得する

概要

```
inode_name:string(inode:long)
```

引数

inode

i ノードへのポインター。

説明

指定された inode に関連付けられた最初のパスベース名を返します。

名前

function::inode_path – inode へのパスを取得する

概要

```
inode_path:string(inode:long)
```

引数

inode

i ノードへのポインター。

説明

指定された inode に関連付けられたフルパスを返します。

名前

function::real_mount – 構造体マウントポインターを取得します

概要

```
real_mount:long(vfsmnt:long)
```

引数

vfsmnt

struct vfsmount へのポインター

説明

struct vfsmount ポインターの struct mount ポインター値を返します。

名前

function::reverse_path_walk – 完全 dirent パスを取得します。

概要

```
reverse_path_walk:string(dentry:long)
```

引数

dentry

dentry へのポインター。

説明

パス名を返します (マウントポイントへの部分パス)

名前

function::task_dentry_path – 完全 dentry パスを取得します。

概要

```
task_dentry_path:string(task:long,dentry:long,vfsmnt:long)
```

引数

task

task_struct ポインター

dentry

direntry ポインター。

vfsmnt

vfsmnt ポインター。

説明

kernel d_path 関数などの完全 dirent 名を返します (root への完全パス)。

第23章 ロギングTAPSET

この種類の関数は、単純なメッセージ文字列を各種の宛先に送信するために使用されます。

名前

function::assert – アサーションを評価する

概要

```
assert(expression:,msg:)
```

引数

expression

評価する式

msg

フォーマットされたメッセージ文字列。

説明

この関数は式をチェックし、式がゼロに評価された場合、現在実行中のプローブを中止します。用途**error**try{} catch{} でキャッチされる可能性があります。

名前

function::error – エラーメッセージを送信します。

概要

```
error(msg:string)
```

引数

msg

フォーマットされたメッセージ文字列。

説明

暗黙的な行末が追加されます。staprun は文字列 「ERROR:」 の前に付けられます。エラーメッセージを送信すると、現在実行中のプローブが中止します。MAXERRORS パラメーターによっては、 がトリガーされる可能性があります。

名前

function::exit – プローブスクリプトのシャットダウンを開始します。

概要

```
exit()
```

引数
なし
説明
これにより、スクリプトのシャットダウンを開始する要求のみをキューに入れます。新規プローブは実行されません (「end」 プローブを除く)。現在実行中のプローブはすべて、その作業を完了する可能性があります。

名前
function::ftrace – ftrace リングバッファーにメッセージを送信します。

概要

```
ftrace(msg:string)
```

引数

msg

フォーマットされたメッセージ文字列。

説明
ftrace リングバッファーが設定され、利用可能な場合は、メッセージについては /debugfs/tracing/trace を参照してください。そうでない場合は、メッセージは警告なしで破棄される可能性があります。暗黙的な行末が追加されます。

名前
function::log – 共通トレースバッファーに行を送信します。

概要

```
log(msg:string)
```

引数

msg

フォーマットされたメッセージ文字列。

説明
この関数はデータをログに記録します。ログはメッセージをすぐに staprun と、一括トランスポート (relayfs) (使用している場合) に送信します。最後の文字が改行でない場合は、追加されます。この関数は printf ほど効率的ではありません。緊急メッセージにのみ使用してください。

名前
function::printk – メッセージをカーネルトレースバッファーに送信する

概要

```
printk(level:long,msg:string)
```

引数

level

重大度レベルの整数 (0=KERN_EMERG ...7=KERN_DEBUG)

msg

フォーマットされたメッセージ文字列。

説明

指定された重大度でテキスト行をカーネル dmesg/console に出力します。暗黙的な行末が追加されます。この関数は、すべてのカーネルプローブコンテキストから安全に呼び出されるとは限らないため、guru モードのみに制限されます。

名前

function::warn – 警告ストリームに行を送信します。

概要

```
warn(msg:string)
```

引数

msg

フォーマットされたメッセージ文字列。

説明

この関数は警告メッセージをすぐに staprun に送信します。また、一括トランスポート (relayfs) (使用されている場合) にわたり送信されます。最後の文字が改行でない場合は、その文字が追加されます。

第24章 キュー統計タップセット

この関数ファミリーは、キューイングシステムのパフォーマンスを追跡するために使用されます。

名前
function::qs_done — 仕上げ依頼記録機能

概要

```
qs_done(qname:string)
```

引数

QName
終了したサービスの名前

説明
この関数は、指定されたキューからのリクエストの処理が完了したことを記録します。

名前
function::qs_run — 待ち行列からサービス中への移動を記録する機能

概要

```
qs_run(qname:string)
```

引数

QName
移動および開始されるサービスの名前

説明
この関数は、以前にキューに入れられたリクエストが指定された待機キューから削除され、現在処理中であることを記録します。

名前
function::qs_wait — エンキュー要求を記録する機能

概要

```
qs_wait(qname:string)
```

引数

QName
エンキューを要求しているキューの名前

説明
この関数は、指定されたキュー名に対して新しいリクエストがキューに入れられたことを記録します。

名前
function::qsq_blocked – リクエストが待機キューにあった時間を返します

概要

```
qsq_blocked:long(qname:string,scale:long)
```

引数

QName

Queue Name*

scale

間隔分数を考慮する変数のスケール

説明
この関数は、1 つ以上の要求が待機キューにあった経過時間の割合を返します。

名前
function::qsq_print – 指定されたキューの統計の行を出力します

概要

```
qsq_print(qname:string)
```

引数

QName

Queue Name*

説明
この関数は、次を含む行を出力します。
指定されたキューの統計
キュー名、1 秒あたりの平均要求率、平均待機キュー長、待機キューの平均時間、要求を処理する平均時間、待機キューが使用された時間の割合、および要求時間の割合サービスされていました。

名前
function::qsq_service_time – リクエストサービスあたりの時間

概要

```
qsq_service_time:long(qname:string,scale:long)
```

引数

QName

Queue Name*

scale

間隔分数を考慮する変数のスケール

説明

この関数は、待機キューから削除されたリクエストを処理するのに必要な平均時間をマイクロ秒単位で返します。

名前

function::qsq_start — キューの統計をリセットする関数

概要

```
qsq_start(qname:string)
```

引数

QName

終了したサービスの名前

説明

この関数は、指定されたキューの統計カウンターをリセットし、関数が呼び出された瞬間から追跡を再開します。この関数は、キューを初期化するためにも使用されます。

名前

function::qsq_throughput — 単位時間あたりのリクエスト数

概要

```
qsq_throughput:long(qname:string,scale:long)
```

引数

QName

Queue Name*

scale

間隔分数を考慮する変数のスケール

説明

この関数は、マイクロ秒ごとに処理されたリクエストの平均数を返します。

名前
function::qsq_utilization — リクエストが処理された時間の割合

概要

```
qsq_utilization:long(qname:string,scale:long)
```

引数

QName

*Queue Name**

scale

間隔分数を考慮する変数のスケール

説明
この関数は、少なくとも1つの要求が処理されていた平均時間をマイクロ秒単位で返します。

名前
function::qsq_wait_queue_length — 待ち行列の長さ

概要

```
qsq_wait_queue_length:long(qname:string,scale:long)
```

引数

QName

*Queue Name**

scale

間隔分数を考慮する変数のスケール

説明
この関数は、待機キューの平均長を返します

名前
function::qsq_wait_time — キュー内の時間+ リクエストごとのサービス

概要

```
qsq_wait_time:long(qname:string,scale:long)
```

引数

QName

*Queue Name**

scale

間隔分数を考慮する変数のスケール

説明

*この関数は、リクエストが処理されるまでにかかった平均時間をマイクロ秒単位で返します (**qs_wait**に**qa_done**)。*

第25章 ランダム関数 TAPSET

以下の関数は乱数の生成を行います。

名前

function::randint – [0,n) の範囲のランダム数を返します。

概要

```
randint:long(n:long)
```

引数

n

範囲の上限を超える数字 (2**20 を超えない)。

第26章 文字列およびデータ取得関数 TAPSET

アドレスに基づいてカーネルまたはユーザー空間プログラムから文字列およびその他のプリミティブタイプを取得する関数。すべての文字列は MAXSTRINGLEN で指定される最大長です。

名前

function::atomic_long_read – カーネルメモリーからアトミック long 変数を取得します

概要

```
atomic_long_read:long(addr:long)
```

引数

addr

アトミック long 変数へのポインター

説明

アトミック long 変数の読み取りを安全に実行します。これは、カーネル設定で ATOMIC_LONG_INIT が設定されていないカーネルでは NOP になります。

名前

function::atomic_read – カーネルメモリーからアトミック変数を取得します

概要

```
atomic_read:long(addr:long)
```

引数

addr

アトミック変数へのポインター

説明

アトミック変数の読み取りを安全に実行します。

名前

function::kernel_char – カーネルメモリーに保存された char 値を取得します。

概要

```
kernel_char:long(addr:long)
```

引数

addr

char 値の取得元のカーネルアドレス。

説明

指定のカーネルメモリーアドレスから char 値を返します。指定アドレスからの読み取りに失敗する場合にエラーを報告します。

名前

function::kernel_int – カーネルメモリーに保存される int 値を取得します。

概要

```
kernel_int:long(addr:long)
```

引数

addr

int の取得元のカーネルアドレスです。

説明

指定のカーネルメモリーアドレスから int 値を返します。指定アドレスからの読み取りに失敗する場合にエラーを報告します。

名前

function::kernel_long – カーネルメモリーに保存された long 値を取得します。

概要

```
kernel_long:long(addr:long)
```

引数

addr

long 値の取得元のカーネルアドレス。

説明

指定のカーネルメモリーアドレスから long 値を返します。指定アドレスからの読み取りに失敗する場合にエラーを報告します。

名前

function::kernel_pointer – カーネルメモリーに保存されるポインター値を取得します。

概要

```
kernel_pointer:long(addr:long)
```

引数

addr

ポインターの取得元のカーネルアドレス。

説明

指定のカーネルメモリーアドレスからポインター値を返します。指定アドレスからの読み取りに失敗する場合にエラーを報告します。

名前

function::kernel_short – カーネルメモリーに保存される short 値を取得します。

概要

```
kernel_short:long(addr:long)
```

引数

addr

short 値の取得元のカーネルアドレス。

説明

指定のカーネルメモリーアドレスから short 値を返します。指定アドレスからの読み取りに失敗する場合にエラーを報告します。

名前

function::kernel_string – カーネルメモリーから文字列を取得します。

概要

```
kernel_string:string(addr:long)
```

引数

addr

文字列の取得元のカーネルアドレス。

説明

この関数は、指定のカーネルメモリーアドレスから NULL 終端 C 文字列を返します。文字列のコピー障害のエラーを報告します。

名前

function::kernel_string2 – 代替エラー文字列と共にカーネルメモリーから文字列を取得します。

概要

```
kernel_string2:string(addr:long,err_msg:string)
```

引数

addr

文字列の取得元のカーネルアドレス。

err_msg

データが利用できない場合に返すエラーメッセージ。

説明

この関数は、指定のカーネルメモリーアドレスから NULL 終端 C 文字列を返します。文字列のコピー障害の所定のエラーメッセージを報告します。

名前

function::kernel_string2_utf16 – 代替エラー文字列と共にカーネルメモリーから文字列を取得します。

概要

```
kernel_string2_utf16:string(addr:long,err_msg:string)
```

引数

addr

文字列の取得元のカーネルアドレス。

err_msg

データが利用できない場合に返すエラーメッセージ。

説明

この関数は、指定されたカーネルメモリーアドレスにある UTF-16 文字列から変換された null で終了する UTF-8 文字列を返します。文字列のコピーエラーまたは変換エラーについて、指定されたエラーメッセージを報告します。

名前

function::kernel_string2_utf32 – 代替エラー文字列と共にカーネルメモリーから文字列を取得します。

概要

```
kernel_string2_utf32:string(addr:long,err_msg:string)
```

引数

addr

文字列の取得元のカーネルアドレス。

err_msg

データが利用できない場合に返すエラーメッセージ。

説明

この関数は、指定されたカーネルメモリーアドレスにある UTF-32 文字列から変換された null で終了する UTF-8 文字列を返します。文字列のコピーエラーまたは変換エラーについて、指定されたエラーメッセージを報告します。

名前

function::kernel_string_n – カーネルメモリーから所定の長さの文字列を取得します。

概要

```
kernel_string_n:string(addr:long,n:long)
```

引数

addr

文字列の取得元のカーネルアドレス。

n

文字列の最大長 (null で終了していない場合)。

説明

指定のカーネルメモリーアドレスから最大長の C 文字列を返します。文字列のコピー障害のエラーを報告します。

名前

function::kernel_string_quoted – カーネルメモリーから文字列を取得して引用します

概要

```
kernel_string_quoted:string(addr:long)
```

引数

addr

文字列を取得するカーネルメモリーアドレス

説明

指定のユーザー空間メモリーアドレスから NULL 終端 C 文字列を返します。出力不可能な ASCII 文字は、返される文字列の対応するエスケープシーケンスに置き換えられます。文字列は二重引用符で囲まれることに注意してください。指定されたアドレスでカーネルメモリーデータにアクセスできない場合は、アドレス自体が二重引用符なしの文字列として返されます。

名前
function::kernel_string_quoted_utf16 — 指定されたカーネル UTF-16 文字列を引用します。

概要

```
kernel_string_quoted_utf16:string(addr:long)
```

引数

addr
文字列の取得元のカーネルアドレス。

説明
*この関数は、string_quoted による引用と、**kernel_string_utf16** による UTF-16 デコードを **組み合わせ**ます。*

名前
function::kernel_string_quoted_utf32 — 指定された UTF-32 カーネル文字列を引用します。

概要

```
kernel_string_quoted_utf32:string(addr:long)
```

引数

addr
文字列の取得元のカーネルアドレス。

説明
*この関数は、string_quoted による引用と、**kernel_string_utf32** による UTF-32 デコードを **組み合わせ**ます。*

名前
function::kernel_string_utf16 — カーネルメモリーから UTF-16 文字列を取得します

概要

```
kernel_string_utf16:string(addr:long)
```

引数

addr
文字列の取得元のカーネルアドレス。

説明

この関数は、指定されたカーネルメモリーアドレスにある UTF-16 文字列から変換された null で終了する UTF-8 文字列を返します。文字列のコピーエラーまたは変換エラーでエラーを報告します。

名前

function::kernel_string_utf32 – カーネルメモリーから UTF-32 文字列を取得します

概要

```
kernel_string_utf32:string(addr:long)
```

引数

addr

文字列の取得元のカーネルアドレス。

説明

この関数は、指定されたカーネルメモリーアドレスにある UTF-32 文字列から変換された null で終了する UTF-8 文字列を返します。文字列のコピーエラーまたは変換エラーでエラーを報告します。

名前

function::user_char – ユーザー空間に保存された char 値を取得します。

概要

```
user_char:long(addr:long)
```

引数

addr

char の取得元のユーザー空間アドレス。

説明

指定のユーザー空間アドレスから char 値を返します。ユーザー空間データにアクセスできない場合はゼロを返します。

名前

function::user_char_warn – ユーザー空間に保存された char 値を取得します。

概要

```
user_char_warn:long(addr:long)
```

引数

addr

char の取得元のユーザー空間アドレス。

説明

指定のユーザー空間アドレスから char 値を返します。ユーザー空間データにアクセスできない場合はゼロを返し、障害に関して警告します (ただし、中止しません)。

名前

function::user_int – ユーザー空間に保存された int 値を取得します。

概要

```
user_int:long(addr:long)
```

引数

addr

int 値の取得元のユーザー空間アドレス。

説明

指定のユーザー空間アドレスから int 値を返します。ユーザー空間データにアクセスできない場合はゼロを返します。

名前

function::user_int16 – ユーザー空間に格納されている 16 ビット整数値を取得します

概要

```
user_int16:long(addr:long)
```

引数

addr

16 ビット整数を取得するユーザー空間アドレス

説明

指定されたユーザー空間アドレスから 16 ビット整数値を返します。ユーザー空間データにアクセスできない場合はゼロを返します。

名前

function::user_int32 – ユーザー空間に格納されている 32 ビット整数値を取得します

概要

```
user_int32:long(addr:long)
```

引数

addr

32 ビット整数を取得するユーザー空間アドレス

説明

指定されたユーザー空間アドレスから 32 ビット整数値を返します。ユーザー空間データにアクセスできない場合はゼロを返します。

名前

function::user_int64 – ユーザー空間に格納されている 64 ビット整数値を取得します

概要

```
user_int64:long(addr:long)
```

引数

addr

64 ビット整数を取得するユーザー空間アドレス

説明

指定されたユーザー空間アドレスから 64 ビット整数値を返します。ユーザー空間データにアクセスできない場合はゼロを返します。

名前

function::user_int8 – ユーザー空間に格納されている 8 ビット整数値を取得します

概要

```
user_int8:long(addr:long)
```

引数

addr

8 ビット整数を取得するユーザー空間アドレス

説明

指定されたユーザー空間アドレスから 8 ビット整数値を返します。ユーザー空間データにアクセスできない場合はゼロを返します。

名前

function::user_int_warn – ユーザー空間に保存された int 値を取得します。

概要

```
user_int_warn:long(addr:long)
```

引数

addr

int 値の取得元のユーザー空間アドレス。

説明

指定のユーザー空間アドレスから int 値を返します。ユーザー空間データにアクセスできない場合はゼロを返し、障害に関して警告します (ただし、中止しません)。

名前

function::user_long – ユーザー空間に保存された long 値を取得します。

概要

```
user_long:long(addr:long)
```

引数

addr

long 値の取得元のユーザー空間アドレス。

説明

指定のユーザー空間アドレスから long 値を返します。ユーザー空間データにアクセスできない場合はゼロを返します。long のサイズは、現在のユーザー空間タスクのアーキテクチャーによって異なることに注意してください (64/32 ビット互換タスクの両方をサポートするアーキテクチャーの場合)。

名前

function::user_long_warn – ユーザー空間に保存された long 値を取得します。

概要

```
user_long_warn:long(addr:long)
```

引数

addr

long 値の取得元のユーザー空間アドレス。

説明

指定のユーザー空間アドレスから long 値を返します。ユーザー空間データにアクセスできない場合はゼロを返し、障害に関して警告します (ただし、中止しません)。long のサイズは、現在のユーザー空間タスクのアーキテクチャーによって異なることに注意してください (64/32 ビット互換タスクの両方をサポートするアーキテクチャーの場合)。

名前

function::user_short – ユーザー空間に保存された short 値を取得します。

概要

```
user_short:long(addr:long)
```

引数

addr

short 値の取得元のユーザー空間アドレスです。

説明

指定のユーザー空間アドレスから short 値を返します。ユーザー空間データにアクセスできない場合はゼロを返します。

名前

function::user_short_warn – ユーザー空間に保存された short 値を取得します。

概要

```
user_short_warn:long(addr:long)
```

引数

addr

short 値の取得元のユーザー空間アドレスです。

説明

指定のユーザー空間アドレスから short 値を返します。ユーザー空間データにアクセスできない場合はゼロを返し、障害に関して警告します (ただし、中止しません)。

名前

function::user_string – ユーザー空間から文字列を取得します。

概要

```
user_string:string(addr:long)
```

引数

addr

文字列の取得元のユーザー空間アドレス。

説明

指定のユーザー空間メモリーアドレスから NULL 終端 C 文字列を返します。ユーザー空間データにアクセスできないまれなケースでエラーを報告します。

名前

function::user_string2 – 代替エラー文字列と共にユーザー空間から文字列を取得します。

概要

```
user_string2:string(addr:long,err_msg:string)
```

引数

addr

文字列の取得元のユーザー空間アドレス。

err_msg

データが利用できない場合に返すエラーメッセージ。

説明

指定のユーザー空間メモリーアドレスから NULL 終端 C 文字列を返します。ユーザー空間データにアクセスできない稀なケースで所定のエラーメッセージを報告します。

名前

function::user_string2_n_warn – 代替警告文字列を使用してユーザー空間から文字列を取得します

概要

```
user_string2_n_warn:string(addr:long,n:long,warn_msg:string)
```

引数

addr

文字列の取得元のユーザー空間アドレス。

n

文字列の最大長 (null で終了していない場合)。

warn_msg

データが利用できない場合に返される警告メッセージ

説明

指定のユーザー空間メモリーアドレスから C 文字列の n 文字までを返します。ユーザー空間データにアクセスできないまれなケースで、指定された警告メッセージを報告し、失敗について警告します (ただし、中止はしません)。

名前

function::user_string2_utf16 – 別のエラー文字列を使用してユーザーメモリーから UTF-16 文字列を取得します

概要

```
user_string2_utf16:string(addr:long,err_msg:string)
```

引数

addr

文字列を取得するユーザーのアドレス

err_msg

データが利用できない場合に返すエラーメッセージ。

説明

この関数は、指定されたユーザーメモリーアドレスにある UTF-16 文字列から変換された、NULL で終了する UTF-8 文字列を返します。文字列のコピーエラーまたは変換エラーについて、指定されたエラーメッセージを報告します。

名前

function::user_string2_utf32 – 別のエラー文字列を使用してユーザーメモリーから UTF-32 文字列を取得します

概要

```
user_string2_utf32:string(addr:long,err_msg:string)
```

引数

addr

文字列を取得するユーザーのアドレス

err_msg

データが利用できない場合に返すエラーメッセージ。

説明

この関数は、指定されたユーザーメモリーアドレスにある UTF-32 文字列から変換された、NULL で終了する UTF-8 文字列を返します。文字列のコピーエラーまたは変換エラーについて、指定されたエラーメッセージを報告します。

名前

function::user_string2_warn – 代替警告文字列を使用してユーザー空間から文字列を取得します

概要

```
user_string2_warn:string(addr:long,warn_msg:string)
```

引数

addr

文字列の取得元のユーザー空間アドレス。

warn_msg

データが利用できない場合に返される警告メッセージ

説明

指定のユーザー空間メモリーアドレスから NULL 終端 C 文字列を返します。ユーザー空間データにアクセスできないまれなケースで、指定された警告メッセージを報告し、失敗について警告します (ただし、中止はしません)。

名前

function::user_string_n – ユーザー空間から所定の長さの文字列を取得します。

概要

```
user_string_n:string(addr:long,n:long)
```

引数

addr

文字列の取得元のユーザー空間アドレス。

n

文字列の最大長 (null で終了していない場合)。

説明

指定のユーザー空間アドレスから最大長の C 文字列を返します。指定されたアドレスでユーザー空間データにアクセスできないまれなケースでエラーを報告します。

名前

function::user_string_n2 – ユーザー空間から所定の長さの文字列を取得します。

概要

```
user_string_n2:string(addr:long,n:long,err_msg:string)
```

引数

addr

文字列の取得元のユーザー空間アドレス。

n

文字列の最大長 (null で終了していない場合)。

err_msg

データが利用できない場合に返すエラーメッセージ。

説明

指定のユーザー空間アドレスから最大長の C 文字列を返します。指定のアドレスでユーザー空間データにアクセスできない稀なケースで指定のエラーメッセージ文字列を返します。

名前

function::user_string_n2_quoted – ユーザー空間から文字列を取得し、引用符で囲みます。

概要

```
user_string_n2_quoted:string(addr:long,inlen:long,outlen:long)
```

引数

addr

文字列の取得元のユーザー空間アドレス。

インレン

読み取る文字列の最大長 (null で終了していない場合)

アウトレン

出力文字列の最大長

説明

指定されたユーザー空間メモリーアドレスから最大 inlen 文字の C 文字列を読み取り、最大 outlen 文字を返します。出力できない ASCII 文字は、返された文字列内の対応するエスケープシーケンスに置き換えられます。文字列は二重引用符で囲まれることに注意してください。指定されたアドレスでユーザー空間データにアクセスできないというまれなケースでは、アドレス自体が二重引用符なしの文字列として返されます。

名前

function::user_string_n_quoted – ユーザー空間から文字列を取得し、引用符で囲みます。

概要

```
user_string_n_quoted:string(addr:long,n:long)
```

引数

addr

文字列の取得元のユーザー空間アドレス。

n

文字列の最大長 (null で終了していない場合)。

説明

指定のユーザー空間メモリーアドレスから n 文字の 終端 C 文字列を返します。出力不可能な ASCII 文字は、返される文字列の対応するエスケープシーケンスに置き換えられます。文字列は二重引用符で囲まれることに注意してください。指定されたアドレスでユーザー空間データにアクセスできないというまれなケースでは、アドレス自体が二重引用符なしの文字列として返されます。

名前

function::user_string_n_warn – ユーザー空間から文字列を取得します。

概要

```
user_string_n_warn:string(addr:long,n:long)
```

引数

addr

文字列の取得元のユーザー空間アドレス。

n

文字列の最大長 (null で終了していない場合)。

説明

指定のユーザー空間メモリーアドレスから C 文字列の n 文字までを返します。ユーザー空間データにアクセスできない場合に、まれなケースで 「<unknown>」 を報告し、障害に関して警告します (ただし、中止しません)。

名前

function::user_string_quoted – ユーザー空間から文字列を取得し、引用符で囲みます。

概要

```
user_string_quoted:string(addr:long)
```

引数

addr

文字列の取得元のユーザー空間アドレス。

説明

指定のユーザー空間メモリーアドレスから NULL 終端 C 文字列を返します。出力不可能な ASCII 文字は、返される文字列の対応するエスケープシーケンスに置き換えられます。文字列は二重引用符で囲まれることに注意してください。指定されたアドレスでユーザー空間データにアクセスできないというまれなケースでは、アドレス自体が二重引用符なしの文字列として返されます。

名前

function::user_string_quoted_utf16 – 指定されたユーザー UTF-16 文字列を引用します。

概要

```
user_string_quoted_utf16:string(addr:long)
```

引数

addr
文字列を取得するユーザーのアドレス

説明

この関数は、string_quoted による引用と **user_string_utf16** による UTF-16 デコードを **組み合わせ** ます。

名前

function::user_string_quoted_utf32 – 指定されたユーザー UTF-32 文字列を引用します。

概要

```
user_string_quoted_utf32:string(addr:long)
```

引数

addr
文字列を取得するユーザーのアドレス

説明

この関数は、string_quoted による引用と **user_string_utf32** による UTF-32 デコードを **組み合わせ** ます。

名前

function::user_string_utf16 – ユーザーメモリーから UTF-16 文字列を取得します

概要

```
user_string_utf16:string(addr:long)
```

引数

addr
文字列を取得するユーザーのアドレス

説明

この関数は、指定されたユーザーメモリーアドレスにある UTF-16 文字列から変換された、NULL で終了する UTF-8 文字列を返します。文字列のコピーエラーまたは変換エラーでエラーを報告します。

名前
function::user_string_utf32 – ユーザーメモリーから UTF-32 文字列を取得します

概要

```
user_string_utf32:string(addr:long)
```

引数

addr

文字列を取得するユーザーのアドレス

説明
この関数は、指定されたユーザーメモリーアドレスにある UTF-32 文字列から変換された、NULL で終了する UTF-8 文字列を返します。文字列のコピーエラーまたは変換エラーでエラーを報告します。

名前
function::user_string_warn – ユーザー空間から文字列を取得します。

概要

```
user_string_warn:string(addr:long)
```

引数

addr

文字列の取得元のユーザー空間アドレス。

説明
指定のユーザー空間メモリーアドレスから NULL 終端 C 文字列を返します。ユーザー空間データにアクセスできない場合に、まれなケースで <unknown> を報告し、障害に関して警告します (ただし、中止しません)。

名前
function::user_uint16 – ユーザー空間に格納されている符号なし 16 ビット整数値を取得します

概要

```
user_uint16:long(addr:long)
```

引数

addr

符号なし 16 ビット整数を取得するユーザー空間アドレス

説明

指定されたユーザー空間アドレスから符号なし 16 ビット整数値を返します。ユーザー空間データにアクセスできない場合はゼロを返します。

名前

function::user_uint32 – ユーザー空間に格納されている符号なし 32 ビット整数値を取得します

概要

```
user_uint32:long(addr:long)
```

引数

addr

符号なし 32 ビット整数を取得するためのユーザー空間アドレス

説明

指定されたユーザー空間アドレスから符号なし 32 ビット整数値を返します。ユーザー空間データにアクセスできない場合はゼロを返します。

名前

function::user_uint64 – ユーザー空間に格納されている符号なし 64 ビット整数値を取得します

概要

```
user_uint64:long(addr:long)
```

引数

addr

符号なし 64 ビット整数を取得するユーザー空間アドレス

説明

指定されたユーザー空間アドレスから符号なし 64 ビット整数値を返します。ユーザー空間データにアクセスできない場合はゼロを返します。

名前

function::user_uint8 – ユーザー空間に格納されている符号なし 8 ビット整数値を取得します

概要

```
user_uint8:long(addr:long)
```

引数

addr

符号なし 8 ビット整数を取得するユーザー空間アドレス

説明

指定されたユーザー空間アドレスから符号なし 8 ビット整数値を返します。ユーザー空間データにアクセスできない場合はゼロを返します。

名前

function::user_ulong – ユーザー空間に格納されている unsigned long 値を取得します

概要

```
user_ulong:long(addr:long)
```

引数

addr

unsigned long を取得するユーザー空間アドレス

説明

指定されたユーザー空間アドレスから unsigned long 値を返します。ユーザー空間データにアクセスできない場合はゼロを返します。long のサイズは、現在のユーザー空間タスクのアーキテクチャーによって異なることに注意してください (64/32 ビット互換タスクの両方をサポートするアーキテクチャーの場合)。

名前

function::user_ulong_warn – ユーザー空間に格納されている unsigned long 値を取得します

概要

```
user_ulong_warn:long(addr:long)
```

引数

addr

unsigned long を取得するユーザー空間アドレス

説明

指定されたユーザー空間アドレスから unsigned long 値を返します。ユーザー空間データにアクセスできない場合はゼロを返し、障害に関して警告します (ただし、中止しません)。long のサイズは、現在のユーザー空間タスクのアーキテクチャーによって異なることに注意してください (64/32 ビット互換タスクの両方をサポートするアーキテクチャーの場合)。

名前

function::user_ushort – ユーザー空間に格納されている unsigned short 値を取得します

概要

```
user_ushort:long(addr:long)
```

引数

addr

unsigned short を取得するユーザー空間アドレス

説明

指定されたユーザー空間アドレスから unsigned short 値を返します。ユーザー空間データにアクセスできない場合はゼロを返します。

名前

function::user_ushort_warn – ユーザー空間に格納されている unsigned short 値を取得します

概要

```
user_ushort_warn:long(addr:long)
```

引数

addr

unsigned short を取得するユーザー空間アドレス

説明

指定されたユーザー空間アドレスから unsigned short 値を返します。ユーザー空間データにアクセスできない場合はゼロを返し、障害に関して警告します (ただし、中止しません)。

第27章 文字列とデータの書き込み関数 TAPSET

*SystemTap guru モードを使用して、障害をシミュレートすることにより、カーネルコードでのエラー処理をテストできます。このタップセットの関数は、カーネルのメモリー内のプリミティブ型に書き込む標準的な方法を提供します。このタップセットのすべての関数では、グルモード (**-g**) を使用する必要があります。*

名前

function::set_kernel_char – char 値をカーネルメモリーに書き込みます。

概要

```
set_kernel_char(addr:long,val:long)
```

引数

addr

文字を書き込むカーネルアドレス

val

書き込む文字

説明

指定されたカーネルメモリーアドレスに char 値を書き込みます。指定されたアドレスへの書き込みが失敗したときにエラーを報告します。guru モード (-g) を使用する必要があります。

名前

function::set_kernel_int – int 値をカーネルメモリーに書き込みます

概要

```
set_kernel_int(addr:long,val:long)
```

引数

addr

int を書き込むカーネルアドレス

val

書き込む int

説明

指定されたカーネルメモリーアドレスに int 値を書き込みます。指定されたアドレスへの書き込みが失敗したときにエラーを報告します。guru モード (-g) を使用する必要があります。

名前

function::set_kernel_long – long 値をカーネルメモリーに書き込みます

概要

```
set_kernel_long(addr:long,val:long)
```

引数

addr

long を書き込むカーネルアドレス

val

書かれる長さ

説明

指定されたカーネルメモリーアドレスに long 値を書き込みます。指定されたアドレスへの書き込みが失敗したときにエラーを報告します。guru モード (-g) を使用する必要があります。

名前

function::set_kernel_pointer – ポインター値をカーネルメモリーに書き込みます。

概要

```
set_kernel_pointer(addr:long,val:long)
```

引数

addr

ポインターを書き込むカーネルアドレス

val

書き込むポインター

説明

指定されたカーネルメモリーアドレスにポインター値を書き込みます。指定されたアドレスへの書き込みが失敗したときにエラーを報告します。guru モード (-g) を使用する必要があります。

名前

function::set_kernel_short – 短い値をカーネルメモリーに書き込みます

概要

```
set_kernel_short(addr:long,val:long)
```

引数

addr

short を書き込むカーネルアドレス

val

書かれる短編

説明

指定されたカーネルメモリーアドレスに short 値を書き込みます。指定されたアドレスへの書き込みが失敗したときにエラーを報告します。guru モード (-g) を使用する必要があります。

名前

function::set_kernel_string – 文字列をカーネルメモリーに書き込みます

概要

```
set_kernel_string(addr:long,val:string)
```

引数

addr

文字列を書き込むカーネルアドレス

val

書き込む文字列

説明

指定された文字列を指定されたカーネルメモリーアドレスに書き込みます。文字列のコピー障害のエラーを報告します。guru モード (-g) を使用する必要があります。

名前

function::set_kernel_string_n – 指定された長さの文字列をカーネルメモリーに書き込みます

概要

```
set_kernel_string_n(addr:long,n:long,val:string)
```

引数

addr

文字列を書き込むカーネルアドレス

n

文字列の最大長

val

書き込む文字列

説明

指定された文字列を指定された最大長まで、指定されたカーネルメモリーアドレスに書き込みます。文字列のコピー障害のエラーを報告します。guru モード (-g) を使用する必要があります。

第28章 GURU タップセット

*障害を挿入したり可観測性を向上させるために、システムの動作を意図的に妨害する関数。このタップセットのすべての関数では、グルモード (**-g**) を使用する必要があります。*

名前

function::mdelay – ミリ秒の遅延

概要

```
mdelay(ms:long)
```

引数

ミリ秒

遅延するミリ秒数。

説明

この関数は、数ミリ秒のビジー遅延をプローブハンドラーに挿入します。達人モードが必要です。

名前

function::panic – パニックを引き起こす

概要

```
panic(msg:string)
```

引数

msg

*カーネルに渡すメッセージ**panic**関数*

説明

この関数は、ユーザー指定のパニックメッセージで、実行中のカーネルの即時パニックをトリガーします。達人モードが必要です。

名前

function::raise – 現在のスレッドでシグナルを発生させる

概要

```
raise(signo:long)
```

引数

signo

信号番号

説明

*この関数は、指定された生のチェックされていないシグナル番号を使用して、現在のスレッドでカーネル send_sig ルーチンを呼び出します。次の場合、エラーが発生する可能性があります**send_sig**失敗した。達人モードが必要です。*

名前

function::udelay — マイクロ秒の遅延

概要

```
udelay(us:long)
```

引数

us

遅延するマイクロ秒数。

説明

この関数は、数マイクロ秒のビジー遅延をプローブハンドラーに挿入します。達人モードが必要です。

第29章 標準的な文字列関数のコレクション

長さ、サブ文字列の取得、個別の文字の取得、文字列の検索、エスケープ、トークン化および文字列の long への変換を実行する関数です。

名前

function::isdigit – 数字をチェックします。

概要

```
isdigit:long(str:string)
```

引数

str

チェックする文字列。

説明

文字列の最初の文字として数字 (0 から 9) があるかどうかを確認します。true の場合はゼロ以外の値を返し、false の場合はゼロを返します。

名前

function::isinstr – 文字列が別の文字列のサブ文字列かどうかを返します。

概要

```
isinstr:long(s1:string,s2:string)
```

引数

s1

検索する文字列。

s2

検索するサブ文字列。

説明

この関数は、文字列 s1 に s2 が含まれる場合に 1 を返します。そうでない場合はゼロを返します。

名前

function::str_replace – str_replace は、サブ文字列のすべてのインスタンスを別のものに置き換えます。

概要

```
str_replace:string(prnt_str:string,srch_str:string,rplc_str:string)
```

引数

prnt_str

検索し、置換する文字列。

srch_str

prnt_str 文字列で検索するために使用されるサブ文字列。

rplc_str

srch_str を置き換えるために使用されるサブ文字列。

説明

この関数は、サブ文字列が置換された所定の文字列を返します。

名前

function::string_quoted – 指定された文字列を引用します

概要

```
string_quoted:string(str:string)
```

引数

str

文字列の取得元のカーネルアドレス。

説明

指定された文字列の引用符付き文字列バージョンを返します。返された文字列では、印刷できない ASCII 文字が対応するエスケープシーケンスに置き換えられています。文字列は二重引用符で囲まれることに注意してください。

名前

function::stringat – 文字列の所定位置の文字を返します。

概要

```
stringat:long(str:string,pos:long)
```

引数

str

文字の取得元の文字列。

pos

文字を取得する位置 (最初の文字は 0)

説明
この関数は、文字列の指定の位置にある文字を返します。または、文字列に多くの文字がない場合はゼロを返します。pos が範囲外の場合、エラーを報告します。

名前
function::strlen – 文字列の長さを返します。

概要

```
strlen:long(s:string)
```

引数

s

文字列。

説明
この関数は、ゼロから MAXSTRINGLEN までに設定できる文字列の長さを返します。

名前
function::strtol – strtol - 文字列を long に変換します。

概要

```
strtol:long(str:string,base:long)
```

引数

str

変換する文字列。

base

使用するベース。

説明
この関数は、数字の文字列表現を整数に変換します。ベースパラメーターは文字列に想定する数値のベースを示します (例: 16 進数の場合は 16、8 進数の場合は 8、バイナリーは 2)。

名前
function::substr – サブ文字列を返します。

概要

```
substr:string(str:string,start:long,length:long)
```

引数

str

サブ文字列の取得元の文字列

start

抽出された文字列の開始位置 (最初の文字は 0)

長さ

返す文字列の長さ。

説明

指定された文字列の指定された開始位置にある指定された長さの部分文字列を返します (元の文字列の長さが start + length より小さい場合、または長さが MAXSTRINGLEN より大きい場合は、それより小さくなります)。

名前

function::text_str – 文字列の出力できない文字をエスケープします。

概要

```
text_str:string(input:string)
```

引数

input

エスケープする文字列。

説明

この関数は文字列引数を受け入れ、出力不可能なすべての ASCII 文字は、返される文字列の対応するエスケープシーケンスに置き換えられます。

名前

function::text_strn – 文字列の出力できない文字をエスケープします。

概要

```
text_strn:string(input:string,len:long,quoted:long)
```

引数

input

エスケープする文字列。

len

返される文字列の最大長 (0 は MAXSTRINGLEN を意味します)

引用した

文字列を二重引用符で囲みます。入力文字列が切り捨てられる場合は、2 つ目の引用符の後に 「...」 が使用されます。

説明

この関数は指定された長さの文字列を受け入れ、出力不可能なすべてのASCII 文字は、返される文字列の対応するエスケープシーケンスに置き換えられます。

名前

function::tokenize – 文字列の次の空でないトークンを返します。

概要

```
tokenize:string(input:string,delim:string)
```

引数

input

*トークン化する文字列。NULL の場合は、**tokenize** を行う直前の呼び出しで渡された文字列の空でないトークンを返します。*

delim

トークンを区切る文字セット。

説明

この関数は、トークンが delim 文字列の文字で区切られる指定の入力文字列の次の空でないトークンを返します。入力文字列が空でない場合、最初のトークンを返します。入力文字列がNULL の場合は、トークン化する直前のへの呼び出しで渡された文字列の次のトークンを返します。区切り文字が見つからない場合は、残りの入力文字列全体が返されます。使用可能なトークンがなくなると、空が返されます。

第30章 ANSI 制御文字をログで使用するためのユーティリティー関数

ansi 制御文字を使用してロギングするユーティリティー関数。これにより、カーソル位置と文字色出力、ログメッセージの属性を変更できます。

名前

function::ansi_clear_screen – カーソルを左上に移動し、画面をクリアします。

概要

```
ansi_clear_screen()
```

引数

なし

説明

カーソルを左上に移動するために ansi コードを送信し、カーソル位置から終了位置までの画面をクリアするために ansi コードを送信します。

名前

function::ansi_cursor_hide – カーソルを非表示にします。

概要

```
ansi_cursor_hide()
```

引数

なし

説明

カーソルを非表示にするために ansi コードを送信します。

名前

function::ansi_cursor_move – カーソルを新規の座標に移動します。

概要

```
ansi_cursor_move(x:long,y:long)
```

引数

x

カーソルの移動先の行。

y

カーソルの移動先の列。

説明
カーソルを x 行および y 列に配置するために ansi コードを送信します。座標は 1 から始まります。(1,1) は左上隅です。

名前
function::ansi_cursor_restore – 以前に保存されたカーソル位置を復元します。

概要

```
ansi_cursor_restore()
```

引数
なし
説明
ansi_cursor_save *で以前に保存した現在のカーソル位置を復元するために ansi コードを送信します。*

名前
function::ansi_cursor_save – カーソル位置を保存します。

概要

```
ansi_cursor_save()
```

引数
なし
説明
現在のカーソル位置を保存するために ansi コードを送信します。

名前
function::ansi_cursor_show – カーソルを表示します。

概要

```
ansi_cursor_show()
```

引数
なし
説明
カーソルを表示するために ansi コードを送信します。

名前
function::ansi_new_line – カーソルを新しい行に移動します。

概要

```
ansi_new_line()
```

引数
なし
説明
改行の ANSI コードを送信します。

名前
function::ansi_reset_color – Select Graphic Rendition モードをリセットします。

概要

```
ansi_reset_color()
```

引数
なし
説明
前景色、背景色および色属性をデフォルト値にリセットするために ansi コードを送信します。

名前
function::ansi_set_color – ansi Select Graphic Rendition モードを設定します。

概要

```
ansi_set_color(fg:long)
```

引数

fg
設定する前景色。

説明
指定のフォグラウンドカラー用に Select Graphic Rendition モードの ansi コードを送信します。黒 (30)、青 (34)、緑 (32)、シアン (36)、赤 (31)、紫 (35)、緑 (33)、緑 (37)。

名前
function::ansi_set_color2 – ansi Select Graphic Rendition モードを設定します。

概要

```
ansi_set_color2(fg:long,bg:long)
```

引数

fg

設定する前景色。

bg

設定する背景色。

説明

前景色 (黒 (30)、青 (34)、緑 (32)、シアン (36)、赤 (31)、紫 (35)、茶 (33)、灰 (37)) および背景色 (黒 (40)、赤 (41)、緑 (42)、黄 (43)、青 (44)、紫 (45)、シアン (46)、白 (47)) を指定する Select Graphic Rendition モードの ANSI コードを送信します。

名前

function::ansi_set_color3 – ansi Select Graphic Rendition モードを設定します。

概要

```
ansi_set_color3(fg:long,bg:long,attr:long)
```

引数

fg

設定する前景色。

bg

設定する背景色。

attr

設定する色属性。

説明

前景色 (黒 (30)、青 (34)、緑 (32)、シアン (36)、赤 (31)、紫 (35)、茶 (33)、灰 (37))、および背景色 (黒 (40)、赤 (41)、緑 (42)、黄 (43)、青 (44)、紫 (45)、シアン (46)、白 (47))、およびすべての属性を解除 (0)、太字 (1)、下線 (4)、点滅 (5)、高速点滅 (6)、 反転 (7) などの色属性を指定する Select Graphic Rendition モードの ANSI コードを送信します。

名前

function::indent – インデントするスペースの量を返します

概要

```
indent:string(delta:long)
```

引数

delta

各呼び出しで追加/削除されたスペースの量

説明

この関数は、適切なインデントを含む文字列を返します。小さな正または一致する負のデルタで呼び出します。thread_indent 関数とは異なり、インデントはスレッドごとに個々のインデント値を追跡しません。

名前

function::indent_depth – グローバルな入れ子の深さを返します

概要

```
indent_depth:long(delta:long)
```

引数

delta

各呼び出しで追加/削除された深さの量

説明

この関数は、適切なインデントの数値を返します。**indent**.小さな正または一致する負のデルタで呼び出します。thread_indent_depth 関数とは異なり、インデントはスレッドごとに個々のインデント値を追跡しません。

名前

function::thread_indent – 現在のタスク情報を含むスペースの量を返します

概要

```
thread_indent:string(delta:long)
```

引数

delta

各呼び出しで追加/削除されたスペースの量

説明

この関数は、スレッドに適切なインデントを含む文字列を返します。小さな正または一致する負のデルタで呼び出します。これがインデントの実際の最も外側の初期レベルである場合、関数は相対タイムスタンプベースをゼロにリセットします。タイムスタンプは __indent_timestamp 関数によって提供されるとおりで、デフォルトではマイクロ秒単位で測定されます。

名前

function::thread_indent_depth – 現在のタスクのネストされた深さを返します

概要

```
thread_indent_depth:long(delta:long)
```

引数

delta

各呼び出しで追加/削除された深さの量

説明

この関数は、最も外側の初期レベルから始まるネストされた関数呼び出しの深さに等しい整数を返します。この関数は、ネストされた関数呼び出しが長いトレースでスペース (空白によって消費される) を節約するのに役立ちます。この関数を同様の方法で使用します。**thread_indent** つまり、call-probe では thread_indent_depth (1) を使用し、return-probe では thread_indent_depth (-1) を使用します。

第31章 SYSTEMTAP トランスレータタップセット

*この一連のユーザー空間プローブポイントは、SystemTap トランスレーター (**stap**) および実行コマンド (**staprun**) の動作をプローブするために使用されます。タップセットには、SystemTap のさまざまなフェーズと SystemTap のインストルメンテーションキャッシュの管理を監視するためのプローブが含まれています。以下のプローブポイントが含まれます。*

名前

probe::stap.cache_add_mod – キャッシュへのカーネル計測モジュールの追加

概要

```
stap.cache_add_mod
```

値

dest_path

.ko ファイルのパス (ファイル名を含む)

source_path

.ko ファイルのパス (ファイル名を含む)

説明

ファイルが実際に移動される直前に発生します。注: 移動に失敗した場合、cache_add_src と cache_add_nss は起動しません。

名前

probe::stap.cache_add_nss – NSS (Network Security Services) 情報をキャッシュに追加する

概要

```
stap.cache_add_nss
```

値

source_path

.sgn ファイルのパス (ファイル名を含む)

dest_path

.sgn ファイルのパス (ファイル名を含む)

説明

ファイルが実際に移動される直前に発生します。注: stap は NSS サポートでコンパイルする必要があります。カーネルモジュールの移動が失敗した場合、このプローブは起動しません。

名前

probe::stap.cache_add_src – キャッシュへの C コード変換の追加

概要

```
stap.cache_add_src
```

値

dest_path

.c ファイルが移動するパス (ファイル名を含む)

source_path

.c ファイルのパス (ファイル名を含む)

説明

ファイルが実際に移動される直前に発生します。注: カーネルモジュールの移動が失敗した場合、このプローブは起動しません。

名前

probe::stap.cache_clean – stap キャッシュからのファイルの削除

概要

```
stap.cache_clean
```

値

path

削除される .ko/.c ファイルへのパス

説明

モジュール/ ソースファイルのリンクを解除する呼び出しの直前に発生します。

名前

probe::stap.cache_get – stap キャッシュにアイテムが見つかりました

概要

```
stap.cache_get
```

値

module_path

.ko カーネルモジュールファイルのパス

source_path

.c ソースファイルのパス

説明

キャッシュグラブが成功すると、get_from_cache が返される直前に発生します。

名前

probe::stap.pass0 – stap pass0 の開始 (コマンドライン引数の解析)

概要

```
stap.pass0
```

値

session

systemtap_session 変数

説明

pass0 は、コマンドライン引数が解析された後に起動します。

名前

probe::stap.pass0.end – stap pass0 の完了 (コマンドライン引数の解析)

概要

```
stap.pass0.end
```

値

session

systemtap_session 変数

説明

*pass0.end の直前に発生します**gettimeofday**pass1 を呼び出します。*

名前

probe::stap.pass1.end – stap pass1 の完了 (スクリプトの解析)

概要

```
stap.pass1.end
```

値

session

systemtap_session 変数

説明

pass1.end は、s.last_pass = 1 の場合、クリーンアップへのジャンプの直前に発生します。

名前

probe::stap.pass1a — stap pass1 の開始 (ユーザースクリプトの解析)

概要

```
stap.pass1a
```

値

session

systemtap_session 変数

説明

*pass1a は、への呼び出しの直後に起動します**gettimeofday**、ユーザースクリプトが解析される前。*

名前

probe::stap.pass1b — stap pass1 の開始 (ライブラリースクリプトの解析)

概要

```
stap.pass1b
```

値

session

systemtap_session 変数

説明

pass1b は、ライブラリースクリプトが解析される直前に起動します。

名前

probe::stap.pass2 — stap pass2 の開始 (詳細)

概要

```
stap.pass2
```

値

session

systemtap_session 変数

説明

*pass2 は、 への呼び出しの直後に発生します**gettimeofday**、semantic_pass の呼び出しの直前。*

名前

probe::stap.pass2.end – 完成した stap pass2(精緻化)

概要

```
stap.pass2.end
```

値

session

systemtap_session 変数

説明

pass2.end は、s.last_pass = 2 の場合、クリーンアップへのジャンプの直前に発生します

名前

probe::stap.pass3 – stap pass3 の開始 (C への翻訳)

概要

```
stap.pass3
```

値

session

systemtap_session 変数

説明

*pass3 は、 への呼び出しの直後に発生します**gettimeofday**、translate_pass の呼び出しの直前。*

名前

probe::stap.pass3.end – stap パス 3 の完了 (C への翻訳)

概要

```
stap.pass3.end
```

値

session

systemtap_session 変数

説明

pass3.end は、s.last_pass = 3 の場合、クリーンアップへのジャンプの直前に発生します。

名前

probe::stap.pass4 – st pass4 の開始 (C コードをカーネルモジュールにコンパイル)

概要

```
stap.pass4
```

値

session

systemtap_session 変数

説明

*pass4 は、への呼び出しの直後に発生します**gettimeofday**、compile_pass の呼び出しの直前。*

名前

probe::stap.pass4.end – stap pass4 の完了 (C コードをカーネルモジュールにコンパイル)

概要

```
stap.pass4.end
```

値

session

systemtap_session 変数

説明

pass4.end は、s.last_pass = 4 の場合、クリーンアップへのジャンプの直前に発生します

名前

probe::stap.pass5 – stap pass5 の開始 (インストルメンテーションの実行)

概要

```
stap.pass5
```

値

session

systemtap_session 変数

説明

*pass5 は、への呼び出しの直後に発生します**gettimeofday**、run_pass の呼び出しの直前。*

名前

probe::stap.pass5.end – stap パス 5 の完了 (インストルメンテーションの実行)

概要

```
stap.pass5.end
```

値

session

systemtap_session 変数

説明

pass5.end は cleanup ラベルの直前に発生します

名前

probe::stap.pass6 – stap pass6 の開始 (クリーンアップ)

概要

```
stap.pass6
```

値

session

systemtap_session 変数

説明

pass6 は cleanup ラベルの直後に発生します。基本的には pass5.end と同じ場所です。

名前

probe::stap.pass6.end – 完成した stap pass6 (クリーンアップ)

概要

```
stap.pass6.end
```

値

session

systemtap_session 変数

説明

main が戻る直前に pass6.end が発火します。

名前

probe::stap.system – stap からのコマンドの開始

概要

```
stap.system
```

値

command

posix_spawn によって実行されるコマンド文字列 (sh -c <str> として)

説明

stap_system コマンドのエントリーで起動します。

名前

probe::stap.system.return – stap からのコマンドを終了しました

概要

```
stap.system.return
```

値

ret

生成されたプロセスでの waitpid の実行に関連付けられた戻りコード。ゼロ以外の値はエラーを示します

説明

waitpid の後、stap_system 関数が戻る直前に発生します。

名前

probe::stap.system.spawn – stap によって生成された新しいプロセス

概要

```
stap.system.spawn
```

値

ret

ret

posix_spawn からの戻り値

pid

生成されたプロセスの pid

説明

posix_spawn の呼び出しの直後に発生します。

名前

probe::stapio.receive_control_message – コントロールメッセージを受信しました

概要

```
stapio.receive_control_message
```

値

len

データブロブの長さ (バイト単位)

data

制御メッセージとして送信されるデータのバイナリー blob への ptr

type

送信されるメッセージのタイプ。runtime/transport/transport_msgs.h で定義

説明

メッセージが受信された直後、処理される前に発生します。

名前

probe::staprun.insert_module – SystemTap 計測モジュールの挿入

概要

```
staprun.insert_module
```

値

path

挿入しようとしている .ko カーネルモジュールへのフルパス

説明

モジュールを挿入する呼び出しの直前に発生します。

名前
probe::staprun.remove_module – SystemTap 計測モジュールの削除

概要

```
staprun.remove_module
```

値

name

削除する stap モジュール名 (.ko 拡張子なし)

説明
モジュールを削除する呼び出しの直前に発生します。

名前
probe::staprun.send_control_message – 制御メッセージの送信

概要

```
staprun.send_control_message
```

値

type

送信されるメッセージのタイプ。runtime/transport/transport_msgs.h で定義

data

制御メッセージとして送信されるデータのバイナリー blob への ptr

len

データブロブの長さ (バイト単位)

説明
send_request 関数の開始時に発生します。

第32章 ネットワークファイルストレージタップセット

この一連のプローブポイントは、ネットワークファイルストレージの機能と操作をプローブするために使用されます。

名前

function::nfsderror – nfsd エラー番号を文字列に変換する

概要

```
nfsderror:string(err:long)
```

引数

err

エラー番号

説明

この関数は、関数に渡されたエラー番号の文字列を返します。

名前

probe::nfs.aop.readpage – ページを同期的に読み取る NFS クライアント

概要

```
nfs.aop.readpage
```

値

size

この実行で読み取られるページ数

i_flag

ファイルフラグ

file

ファイル引数

いの

inode 番号

i_size

ファイルの長さ (バイト)

dev

デバイス識別子。

rsize
読み取りサイズ(バイト単位)

__page
ページのアドレス

sb_flag
スーパーブロックフラグ

page_index
マッピング内のオフセット。ページ識別子とページフレーム内の位置識別子を使用できます。

説明
ページを読み込んで、前の非同期読み取り操作が失敗した場合にのみ起動します

名前
probe::nfs.aop.readpages – 複数のページを読み取る NFS クライアント

概要

```
nfs.aop.readpages
```

値

nr_pages
この実行で読み取ろうとしたページ数

いの
inode 番号

file
フィルプ引数

size
この実行で読み取ろうとしたページ数

rsize
読み取りサイズ(バイト単位)

dev
デバイス識別子。

rpages
読み取りサイズ(ページ数)

説明
先読みの方法で起動し、一度に複数のページを読み取ります

名前
probe::nfs.aop.release_page – NFS クライアントのリリースページ

概要

```
nfs.aop.release_page
```

値

size
リリースページ

いの
inode 番号

dev
デバイス識別子。

__page
ページのアドレス

page_index
マッピング内のオフセット。ページ識別子とページフレーム内の位置識別子を使用できます。

説明
NFS で解放操作を行うときに発生します。

名前
probe::nfs.aop.set_page_dirty – NFS クライアントがページをダーティとしてマーク

概要

```
nfs.aop.set_page_dirty
```

値

__page
ページのアドレス

page_flag
ページフラグ

説明

このプローブは、一般的な__set_page_dirty_nobuffers 関数に接続します。したがって、このプローブは、NFS クライアントに加えて、他の多くのファイルシステムで起動します。

名前

probe::nfs.aop.write_begin – NFS クライアントがデータの書き込みを開始

概要

```
nfs.aop.write_begin
```

値

__page

ページのアドレス

page_index

マッピング内のオフセット。ページ識別子とページフレーム内の位置識別子を使用できます。

size

書き込みバイト

上記を以下のように変更します。

この書き込み操作の終了アドレス

いの

inode 番号

offset

この書き込み操作の開始アドレス

dev

デバイス識別子。

説明

nfs で書き込み操作が発生したときに発生します。書き込み用のページを用意し、そのページに対応するリクエストを探します。1 つがあり、それが別のファイルに属している場合、ページに何かをコピーしようとする前に、それをフラッシュします。既存のドロップされたページからのリクエストが見つかった場合も同じことを行います

名前

probe::nfs.aop.write_end – NFS クライアントがデータの書き込みを完了

概要

```
nfs.aop.write_end
```

値

sb_flag
スーパーブロックフラグ

__page
ページのアドレス

page_index
マッピング内のオフセット。ページ識別子とページフレーム内の位置識別子を使用できます。

上記を以下のように変更します。
この書き込み操作の終了アドレス

いの
inode 番号

i_flag
ファイルフラグ

size
書き込みバイト

dev
デバイス識別子。

offset
この書き込み操作の開始アドレス

i_size
ファイルの長さ (バイト)

説明

多くの場合、prepare_write の後に、nfs で書き込み操作を行うときに発生します
NFS ファイルのキャッシュされたページを更新し、場合によっては書き込みます。

名前

probe::nfs.aop.writepage – マップされたページを NFS サーバーに書き込む NFS クライアント

概要

```
nfs.aop.writepage
```

値

wsize

書き込みサイズ

size
この実行で書き込まれるページ数

i_flag
ファイルフラグ

for_kupdate
kupdate の書き戻しかどうかを示す writeback_control のフラグ

いの
inode 番号

i_size
ファイルの長さ (バイト)

dev
デバイス識別子。

for_reclaim
ページアロケータから呼び出されたかどうかを示す writeback_control のフラグ

__page
ページのアドレス

sb_flag
スーパーブロックフラグ

page_index
マッピング内のオフセット。ページ識別子とページフレーム内の位置識別子を使用できます。

i_state
inode 状態フラグ

説明

*wb の優先順位は、フラグ **for_reclaim** および **for_kupdate** によって決定されます。*

名前

probe::nfs.aop.writepages – NFS クライアントがいくつかのダーティページを NFS サーバーに書き込んでいる

概要

```
nfs.aop.writepages
```

値

for_reclaim
ページアロケータから呼び出されたかどうかを示す writeback_control のフラグ

wpages
書き込みサイズ (ページ単位)

nr_to_write
この実行で書き込みを試みたページ数

for_kupdate
kupdate の書き戻しかどうかを示す writeback_control のフラグ

いの
inode 番号

size
この実行で書き込みを試みたページ数

wsize
書き込みサイズ

dev
デバイス識別子。

説明
wb の優先順位は、フラグ **for_reclaim** および **for_kupdate** によって決定されます。

名前
probe::nfs.fop.aio_read – NFS クライアント aio_read ファイル操作

概要

```
nfs.fop.aio_read
```

値

いの
inode 番号

cache_time
この i ノードの読み取りキャッシュを開始したとき

file_name
file:name

バフ

ユーザー空間の buf のアドレス

dev

デバイス識別子。

pos

ファイルの現在位置

attrtimeo

キャッシュされた情報が有効であると想定される期間。jiffies - read_cache_jiffies > attrtimeo の場合、この i ノードのキャッシュされた属性を再検証する必要があります。

count

読み取りバイト

parent_name

親ディレクトリー名

cache_valid

キャッシュ関連のビットマスクフラグ

名前

probe::nfs.fop.aio_write – NFS クライアント aio_write ファイル操作

概要

```
nfs.fop.aio_write
```

値

count

読み取りバイト

parent_name

親ディレクトリー名

いの

inode 番号

file_name

file:name

バフ

ユーザー空間の buf のアドレス

dev
デバイス識別子。

pos
ファイルのオフセット

名前
probe::nfs.fop.check_flags – NFS クライアントチェックフラグ操作

概要

```
nfs.fop.check_flags
```

値

flag
ファイルフラグ

名前
probe::nfs.fop.flush – NFS クライアントフラッシュファイル操作

概要

```
nfs.fop.flush
```

値

ndirty
ダーティページ数

いの
inode 番号

mode
ファイルモード

dev
デバイス識別子。

名前
probe::nfs.fop.fsync – NFS クライアントの fsync 操作

概要

nfs.fop.fsync

値

ndirty
ダーティーページ数

いの
inode 番号

dev
デバイス識別子。

名前
probe::nfs.fop.llseek – NFS クライアント llseek 操作

概要

nfs.fop.llseek

値

いの
inode 番号

whence
求める位置

dev
デバイス識別子。

offset
ファイルのオフセットが再配置されます

whence_str
シークする位置のシンボリック文字列表現

名前
probe::nfs.fop.lock – NFS クライアントファイルロック操作

概要

nfs.fop.lock

値

fl_start
ロックされた領域の開始オフセット

いの
inode 番号

fl_flag
glock フラグ

i_mode
ファイルの種類とアクセス権

dev
デバイス識別子。

fl_end
ロックされた領域の終了オフセット

fl_type
ロックタイプ

cmd
コマンド引数

名前
probe::nfs.fop.mmap – NFS クライアントの mmap 操作

概要

nfs.fop.mmap

値

attrtimeo
キャッシュされた情報が有効であると想定される期間。jiffies - read_cache_jiffies > attrtimeo の場合、この i ノードのキャッシュされた属性を再検証する必要があります。

vm_end
vm_mm 内の終了アドレスの後の最初のバイト

dev

デバイス識別子。

バフ

ユーザー空間の buf のアドレス

vm_flag

vm フラグ

cache_time

この i ノードの読み取りキャッシュを開始したとき

file_name

file:name

いの

inode 番号

cache_valid

キャッシュ関連のビットマスクフラグ

parent_name

親ディレクトリー名

vm_start

vm_mm 内の開始アドレス

名前

probe::nfs.fop.open – NFS クライアントファイルのオープン操作

概要

```
nfs.fop.open
```

値

flag

ファイルフラグ

i_size

ファイルの長さ (バイト)

dev

デバイス識別子。

file_name

file:name

いの

inode 番号

名前

probe::nfs.fop.read – NFS クライアント読み取り操作

概要

```
nfs.fop.read
```

値

devname

ブロックデバイス名

説明

SystemTap は vfs.do_sync_read プローブを使用してこのプローブを実装し、その結果、NFS クライアントの読み取り操作以外の操作を取得します。

名前

probe::nfs.fop.read_iter – NFS クライアント read_iter ファイル操作

概要

```
nfs.fop.read_iter
```

値

いの

inode 番号

file_name

file:name

cache_time

この i ノードの読み取りキャッシュを開始したとき

pos

ファイルの現在位置

dev

デバイス識別子。

attrtimeo

キャッシュされた情報が有効であると想定される期間。jiffies - read_cache_jiffies > attrtimeo の場合、この i ノードのキャッシュされた属性を再検証する必要があります。

count

読み取りバイト

parent_name

親ディレクトリー名

cache_valid

キャッシュ関連のビットマスクフラグ

名前

probe::nfs.fop.release — NFS クライアントリリースページの操作

概要

```
nfs.fop.release
```

値

いの

inode 番号

dev

デバイス識別子。

mode

ファイルモード

名前

probe::nfs.fop.sendfile — NFS クライアントのファイル送信操作

概要

```
nfs.fop.sendfile
```

値

cache_valid

キャッシュ関連のビットマスクフラグ

ppos

ファイルの現在位置

count

読み取りバイト

dev

デバイス識別子。

attrtimeo

キャッシュされた情報が有効であると想定される期間。jiffies - read_cache_jiffies > attrtimeo の場合、この i ノードのキャッシュされた属性を再検証する必要があります。

いの

inode 番号

cache_time

この i ノードの読み取りキャッシュを開始したとき

名前

probe::nfs.fop.write – NFS クライアントの書き込み操作

概要

```
nfs.fop.write
```

値

devname

ブロックデバイス名

説明

SystemTap は vfs.do_sync_write プローブを使用してこのプローブを実装し、その結果、NFS クライアントの書き込み操作以外の操作を取得します。

名前

probe::nfs.fop.write_iter – NFS クライアント write_iter ファイル操作

概要

```
nfs.fop.write_iter
```

値

parent_name

親ディレクトリー名

count

読み取りバイト

pos

ファイルのオフセット

dev

デバイス識別子。

file_name

file:name

いの

inode 番号

名前

probe::nfs.proc.commit – サーバー上でデータをコミットする NFS クライアント

概要

```
nfs.proc.commit
```

値

size

この実行でバイトを読み取る

prot

転送プロトコル

version

NFS バージョン 3

server_ip

サーバーの IP アドレス

bitmask1

このファイルシステムでサポートされている一連の属性を表す V4 ビットマスク

offset

ファイルオフセット

bitmask0

このファイルシステムでサポートされている一連の属性を表す V4 ビットマスク

説明

すべての nfs.proc.commit カーネル関数は、2006 年 12 月のカーネルコミット 200baa で削除されたため、これらのプローブは Linux 2.6.21 以降のカーネルには存在しません。
クライアントがバッファーリングされたデータをディスクに書き込むときに発生します。バッファーリングされたデータは、以前にクライアントによって非同期に書き込まれます。コミット機能は同期的に機能します。このプローブポイントは、NFSv2 には存在しません。

名前

probe::nfs.proc.commit_done — commit RPC タスクに対する NFS クライアントの応答

概要

```
nfs.proc.commit_done
```

値

status

最後の操作の結果

server_ip

サーバーの IP アドレス

prot

転送プロトコル

version

NFS バージョン 3

count

コミットされたバイト数

Valid

fattr->valid、有効なフィールドを示します

timestamp

リースの更新に使用される V4 タイムスタンプ

説明

コミット RPC タスクへの応答を受信したとき、または何らかのコミット操作エラー (タイムアウトまたはソケットシャットダウン) が発生したときに発生します。

名前

probe::nfs.proc.commit_setup — コミット RPC タスクをセットアップする NFS クライアント

概要

nfs.proc.commit_setup

値

version

NFS バージョン 3

count

このコミットのバイト数

prot

転送プロトコル

server_ip

サーバーの IP アドレス

bitmask1

このファイルシステムでサポートされている一連の属性を表す V4 ビットマスク

bitmask0

このファイルシステムでサポートされている一連の属性を表す V4 ビットマスク

offset

ファイルオフセット

size

このコミットのバイト数

説明

commit_setup 関数は、コミット RPC タスクをセットアップするために使用されます。実際のコミット操作を行っていません。NFSv2 には存在しません。

名前

probe::nfs.proc.create – サーバー上でファイルを作成する NFS クライアント

概要

nfs.proc.create

値

server_ip

サーバーの IP アドレス

prot

転送プロトコル

version

NFS バージョン (この機能はすべての NFS バージョンで使用されます)

filename

file:name

fh

親ディレクトリーのファイルハンドル

ファイルレン

ファイル名の長さ

flag

作成モードを示します (NFSv3 および NFSv4 のみ)

名前

probe::nfs.proc.handle_exception – NFSv4 例外を処理する NFS クライアント

概要

```
nfs.proc.handle_exception
```

値

errorcode

エラーの種類を示します

説明

これは、NFSv4 のプロセスのエラー処理ルーチンです。

名前

probe::nfs.proc.lookup – NFS クライアントがサーバー上のファイルを開く/検索する

概要

```
nfs.proc.lookup
```

値

bitmask1

このファイルシステムでサポートされている一連の属性を表す V4 ビットマスク

bitmask0

このファイルシステムでサポートされている一連の属性を表すV4 ビットマスク

filename

クライアントがサーバー上で開く/検索するファイルの名前

server_ip

サーバーのIP アドレス

prot

転送プロトコル

name_len

ファイル名の長さ

version

NFS バージョン3

名前

probe::nfs.proc.open – NFS クライアントは、ファイルの読み取り/書き込みコンテキスト情報を割り当てます

概要

```
nfs.proc.open
```

値

flag

ファイルフラグ

filename

file:name

version

NFS バージョン(この機能はすべてのNFS バージョンで使用されます)

prot

転送プロトコル

mode

ファイルモード

server_ip

サーバーのIP アドレス

説明

ファイルの読み取り/書き込みコンテキスト情報を割り当てます

名前

probe::nfs.proc.read – NFS クライアントがサーバーからファイルを同期的に読み取る

概要

```
nfs.proc.read
```

値

offset

ファイルオフセット

server_ip

サーバーの IP アドレス

flags

rpc_init_task 関数で task->tk_flags を設定するために使用

prot

転送プロトコル

count

この実行でバイトを読み取る

version

NFS バージョン 3

説明

すべての nfs.proc.read カーネル関数は、2006 年 12 月のカーネルコミット 8e0969 で削除されたため、これらのプローブは Linux 2.6.21 以降のカーネルには存在しません。

名前

probe::nfs.proc.read_done – 読み取り RPC タスクに対する NFS クライアントの応答

概要

```
nfs.proc.read_done
```

値

timestamp

リースの更新に使用される V4 タイムスタンプ

prot

転送プロトコル

count

読み取られたバイト数

version

NFS バージョン 3

status

最後の操作の結果

server_ip

サーバーの IP アドレス

説明

読み取り RPC タスクへの応答を受信したとき、または何らかの読み取りエラー (タイムアウトまたはソケットのシャットダウン) が発生したときに発生します。

名前

probe::nfs.proc.read_setup – 読み取り RPC タスクをセットアップする NFS クライアント

概要

```
nfs.proc.read_setup
```

値

offset

ファイルオフセット

server_ip

サーバーの IP アドレス

prot

転送プロトコル

version

NFS バージョン 3

count

この実行でバイトを読み取る

size

この実行でバイトを読み取る

説明

read_setup 関数は、読み取り RPC タスクをセットアップするために使用されます。実際の読み取り操作は行っていません。

名前

probe::nfs.proc.release – NFS クライアントは、ファイルの読み取り/書き込みコンテキスト情報を解放します

概要

```
nfs.proc.release
```

値

flag

ファイルフラグ

filename

file:name

prot

転送プロトコル

version

NFS バージョン (この機能はすべての NFS バージョンで使用されます)

mode

ファイルモード

server_ip

サーバーの IP アドレス

説明

ファイルの読み取り/書き込みコンテキスト情報を解放する

名前

probe::nfs.proc.remove – NFS クライアントがサーバー上のファイルを削除する

概要

```
nfs.proc.remove
```

値

prot

転送プロトコル

version

NFS バージョン (この機能はすべての NFS バージョンで使用されます)

server_ip

サーバーの IP アドレス

ファイルレン

ファイル名の長さ

filename

file:name

fh

親ディレクトリーのファイルハンドル

名前

probe::nfs.proc.rename – NFS クライアントがサーバー上のファイルの名前を変更する

概要

```
nfs.proc.rename
```

値

new_fh

新しい親ディレクトリーのファイルハンドル

new_filelen

新しいファイル名の長さ

old_name

古いファイル名

version

NFS バージョン (この機能はすべての NFS バージョンで使用されます)

old_fh

古い親ディレクトリーのファイルハンドル

prot

転送プロトコル

new_name

新しいファイル名

old_filelen
古いファイル名の長さ

server_ip
サーバーの IP アドレス

名前
probe::nfs.proc.rename_done — 名前変更 RPC タスクに対する NFS クライアントの応答

概要

```
nfs.proc.rename_done
```

値

timestamp
リースの更新に使用される V4 タイムスタンプ

status
最後の操作の結果

server_ip
サーバーの IP アドレス

prot
転送プロトコル

version
NFS バージョン 3

old_fh
古い親ディレクトリーのファイルハンドル

new_fh
新しい親ディレクトリーのファイルハンドル

説明
名前変更 RPC タスクへの応答を受信したとき、または名前変更エラーが発生したとき (タイムアウトまたはソケットシャットダウン) に起動します。

名前
probe::nfs.proc.rename_setup — 名前変更 RPC タスクをセットアップする NFS クライアント

概要

nfs.proc.rename_setup

値

fh

親ディレクトリーのファイルハンドル

prot

転送プロトコル

version

NFS バージョン 3

server_ip

サーバーの IP アドレス

説明

rename_setup 関数は、名前変更 RPC タスクをセットアップするために使用されます。実際の名前変更操作は行っていません。

名前

probe::nfs.proc.write – NFS クライアントがファイルをサーバーに同期的に書き込む

概要

nfs.proc.write

値

size

この実行でバイトを読み取る

flags

rpc_init_task 関数で task->tk_flags を設定するために使用

prot

転送プロトコル

version

NFS バージョン 3

bitmask1

このファイルシステムでサポートされている一連の属性を表す V4 ビットマスク

offset

ファイルオフセット

bitmask0

このファイルシステムでサポートされている一連の属性を表す V4 ビットマスク

server_ip

サーバーの IP アドレス

説明

すべての nfs.proc.write カーネル関数は、2006 年 12 月のカーネルコミット 200baa で削除されたため、これらのプローブは Linux 2.6.21 以降のカーネルには存在しません。

名前

probe::nfs.proc.write_done — 書き込み RPC タスクに対する NFS クライアントの応答

概要

```
nfs.proc.write_done
```

値

server_ip

サーバーの IP アドレス

status

最後の操作の結果

version

NFS バージョン 3

count

書き込まれたバイト数

prot

転送プロトコル

Valid

fattr->valid、有効なフィールドを示します

timestamp

リースの更新に使用される V4 タイムスタンプ

説明

書き込み RPC タスクへの応答を受信したとき、または何らかの書き込みエラー (タイムアウトまたはソケットのシャットダウン) が発生したときに発生します。

名前

probe::nfs.proc.write_setup – 書き込み RPC タスクをセットアップする NFS クライアント

概要

```
nfs.proc.write_setup
```

値

size

この実行で書き込まれたバイト数

prot

転送プロトコル

version

NFS バージョン 3

count

この実行で書き込まれたバイト数

bitmask0

このファイルシステムでサポートされている一連の属性を表す V4 ビットマスク

bitmask1

このファイルシステムでサポートされている一連の属性を表す V4 ビットマスク

offset

ファイルオフセット

how

args.stable の設定に使用されます。安定した値は次のようになります: NFS_UNSTABLE,NFS_DATA_SYNC,NFS_FILE_SYNC (nfs.proc3.write_setup および nfs.proc4.write_setup 内)

server_ip

サーバーの IP アドレス

説明

write_setup 関数は、書き込み RPC タスクをセットアップするために使用されます。実際の書き込み操作は行っていません。

名前

probe::nfsd.close – NFS サーバーがクライアントのファイルを閉じています

概要

```
nfsd.close
```

-

値

filename

file:name

説明

このプローブポイントは、4.2 以降のカーネルには存在しません。

名前

probe::nfsd.commit — 保留中のすべての書き込みを安定したストレージにコミットする NFS サーバー

概要

```
nfsd.commit
```

値

fh

ファイルハンドル (最初の部分はファイルハンドルの長さです)

flag

この実行が同期操作かどうかを示します

offset

ファイルのオフセット

size

読み取りバイト

count

読み取りバイト

client_ip

クライアントの IP アドレス

名前

probe::nfsd.create — クライアント用のファイル (regular、dir、device、fifo) を作成する NFS サーバー

概要

```
nfsd.create
```

値

fh

ファイルハンドル (最初の部分はファイルハンドルの長さです)

iap_valid

属性フラグ

ファイルレン

ファイル名の長さ

type

ファイルの種類 (regular、dir、device、fifo ...)

filename

file:name

iap_mode

ファイルアクセスモード

client_ip

クライアントの IP アドレス

説明

nfsd は、このプローブポイントの代わりに nfsd_create_v3 を呼び出すことがあります。

名前

probe::nfsd.createv3 – NFS サーバーが通常のファイルを作成するか、クライアントのファイル属性を設定する

概要

```
nfsd.createv3
```

値

iap_mode

ファイルアクセスモード

filename

file:name

client_ip

クライアントの IP アドレス

fh

ファイルハンドル (最初の部分はファイルハンドルの長さです)

createmode

作成モード。可能な値は次のとおりです: NFS3_CREATE_EXCLUSIVE、NFS3_CREATE_UNCHECKED、または NFS3_CREATE_GUARDED

ファイルレン

ファイル名の長さ

iap_valid

属性フラグ

verifier

ファイル属性 (atime、mtime、mode)。CREATE_EXCLUSIVE のファイル属性をリセットするために使用されます

truncp

trunp 引数。ファイルを切り詰める必要があるかどうかを示します

説明

このプローブポイントは、op_claim_type が NFS4_OPEN_CLAIM_NULL の場合に、nfsd3_proc_create および nfsd4_open によってのみ呼び出されます。

名前

probe::nfsd.dispatch – NFS サーバーがクライアントから操作を受け取る

概要

```
nfsd.dispatch
```

値

xid

送信 ID

version

NFS バージョン 3

proto

転送プロトコル

/proc/

手続き番号

client_ip

クライアントの IP アドレス

prog

プログラム番号

名前

probe::nfsd.lookup – NFS サーバーがクライアントのファイルを開いているか、ファイルを検索しています

概要

```
nfsd.lookup
```

値

filename

file:name

client_ip

クライアントの IP アドレス

fh

親ディレクトリーのファイルハンドル (最初の部分はファイルハンドルの長さ)

ファイルレン

ファイル名の長さ

名前

probe::nfsd.open – クライアントのファイルを開く NFS サーバー

概要

```
nfsd.open
```

値

fh

ファイルハンドル (最初の部分はファイルハンドルの長さです)

type

ファイルのタイプ (通常のファイルまたはディレクトリー)

access

オープンのタイプを示します (読み取り/書き込み/コミット/読み取りディレクトリー...)

client_ip

クライアントの IP アドレス

名前

probe::nfsd.proc.commit – クライアントのコミット操作を実行する NFS サーバー

概要

```
nfsd.proc.commit
```

値

count

読み取りバイト

client_ip

クライアントの IP アドレス

proto

転送プロトコル

size

読み取りバイト

version

NFS バージョン 3

uid

リクエスタのユーザー ID

offset

ファイルのオフセット

gid

リクエスタのグループ ID

fh

ファイルハンドル (最初の部分はファイルハンドルの長さです)

名前

probe::nfsd.proc.create – クライアント用のファイルを作成する NFS サーバー

概要

nfsd.proc.create

値

proto

転送プロトコル

filename

file:name

client_ip

クライアントのIP アドレス

uid

リクエスタのユーザーID

version

NFS バージョン3

gid

リクエスタのグループID

fh

ファイルハンドル(最初の部分はファイルハンドルの長さです)

ファイルレン

ファイル名の長さ

名前

probe::nfsd.proc.lookup – NFS サーバーがクライアントのファイルを開いているか検索しています

概要

nfsd.proc.lookup

値

fh

親ディレクトリーのファイルハンドル(最初の部分はファイルハンドルの長さ)

gid

リクエスタのグループID

ファイルレン

ファイル名の長さ

uid

リクエスタのユーザー ID

version

NFS バージョン 3

proto

転送プロトコル

filename

file:name

client_ip

クライアントの IP アドレス

名前

probe::nfsd.proc.read – クライアント用の NFS サーバー読み取りファイル

概要

```
nfsd.proc.read
```

値

size

読み取りバイト

vec

struct kvec、カーネルアドレスに buf アドレスと各バッファーの長さを含む

version

NFS バージョン 3

uid

リクエスタのユーザー ID

count

読み取りバイト

client_ip

クライアントの IP アドレス

proto

転送プロトコル

offset

ファイルのオフセット

gid

リクエスタのグループID

vlen

ブロックを読む

fh

ファイルハンドル(最初の部分はファイルハンドルの長さです)

名前

probe::nfsd.proc.remove – クライアントのファイルを削除するNFS サーバー

概要

```
nfsd.proc.remove
```

値

gid

リクエスタのグループID

fh

ファイルハンドル(最初の部分はファイルハンドルの長さです)

ファイルレン

ファイル名の長さ

uid

リクエスタのユーザーID

version

NFS バージョン3

proto

転送プロトコル

filename

file:name

client_ip

クライアントのIP アドレス

名前

probe::nfsd.proc.rename — クライアントのファイルの名前を変更する NFS サーバー

概要

```
nfsd.proc.rename
```

値

uid

リクエスタのユーザー ID

tfh

新しいパスのファイルハンドラー

tname

新しいファイル名

filename

古いファイル名

client_ip

クライアントの IP アドレス

フレン

古いファイル名の長さ

gid

リクエスタのグループ ID

fh

古いパスのファイルハンドラー

トレン

新しいファイル名の長さ

名前

probe::nfsd.proc.write — クライアントのファイルにデータを書き込む NFS サーバー

概要

```
nfsd.proc.write
```

値

offset
ファイルのオフセット

gid
リクエスタのグループID

vlen
ブロックを読む

fh
ファイルハンドル(最初の部分はファイルハンドルの長さです)

size
読み取りバイト

vec
struct kvec、カーネルアドレスにbufアドレスと各バッファーの長さを含む

stable
argp->stable

version
NFSバージョン3

uid
リクエスタのユーザーID

count
読み取りバイト

client_ip
クライアントのIPアドレス

proto
転送プロトコル

名前

probe::nfsd.read – クライアントのファイルからデータを読み取るNFSサーバー

概要

```
nfsd.read
```

値

offset
ファイルのオフセット

vlen
ブロックを読む

file
引数 file は、ファイルが開かれているかどうかを示します。

fh
ファイルハンドル (最初の部分はファイルハンドルの長さです)

count
読み取りバイト

client_ip
クライアントの IP アドレス

size
読み取りバイト

vec
struct kvec、カーネルアドレスに buf アドレスと各バッファーの長さを含む

名前

probe::nfsd.rename – クライアントのファイルの名前を変更する NFS サーバー

概要

```
nfsd.rename
```

値

トレン
新しいファイル名の長さ

fh
古いパスのファイルハンドラー

フレン
古いファイル名の長さ

client_ip
クライアントの IP アドレス

filename

古いファイル名

tname

新しいファイル名

tfh

新しいパスのファイルハンドラー

名前

probe::nfsd.unlink – クライアントのファイルまたはディレクトリーを削除する NFS サーバー

概要

```
nfsd.unlink
```

値

ファイルレン

ファイル名の長さ

fh

ファイルハンドル (最初の部分はファイルハンドルの長さです)

type

ファイルの種類 (ファイルまたはディレクトリー)

client_ip

クライアントの IP アドレス

filename

file:name

名前

probe::nfsd.write – クライアントのファイルにデータを書き込む NFS サーバー

概要

```
nfsd.write
```

値

offset

ファイルのオフセット

fh

ファイルハンドル (最初の部分はファイルハンドルの長さです)

vlen

ブロックを読む

file

引数 file は、ファイルが開かれているかどうかを示します。

client_ip

クライアントの IP アドレス

count

読み取りバイト

size

読み取りバイト

vec

struct kvec、カーネルアドレスに buf アドレスと各バッファーの長さを含む

第33章 投機

この一連の関数は、投機的に情報を記録し、SystemTap スクリプトの後半で情報をコミットまたは破棄する機能を提供します。

名前

function::commit – 投機バッファーに関連するすべての出力を書き出す

概要

```
commit(id:long)
```

引数

id

情報を格納するバッファーの

説明

id のすべての出力を、投機バッファーに入力された順序で出力します。**speculative**.

名前

function::discard – 投機バッファーに関連するすべての出力を破棄します

概要

```
discard(id:long)
```

引数

id

情報を格納するバッファーの

名前

function::speculate – 後で出力できるように文字列を保存する

概要

```
speculate(id:long,output:string)
```

引数

id

情報を格納するバッファーID

output

コミット時に書き出す文字列

説明

id の投機的バッファーに文字列を追加します。

名前

function::speculation — 投機的な出力用に新しい ID を割り当てます

概要

```
speculation:long()
```

引数

なし

説明

*の**speculation**関数は、新しい投機バッファーが必要なときに呼び出されます。投機的な出力の ID を返します。複数のスレッドが同時に推測されている可能性があります。この ID は、スレッドを分離しておくために他の投機関数によって使用されます。*

第34章 JSON タップセット

この一連のプローブポイント、関数、およびマクロは、データを JSON 形式で出力するために使用されます。これには、次のプローブポイント、関数、およびマクロが含まれています。

名前

function::json_add_array – 配列を追加する

概要

```
json_add_array:long(name:string,description:string)
```

引数

name

配列の名前。

description

配列の説明。空の文字列を使用できます。

説明

*この関数は配列を追加し、必要なものをすべて設定します。配列には、追加された他のメトリックが含まれます**json_add_array_numeric_metric**また**json_add_array_string_metric**.*

名前

function::json_add_array_numeric_metric – 数値メトリックを配列に追加する

概要

```
json_add_array_numeric_metric:long(array_name:string,metric_name:string,metric_description:string,metric_units:string)
```

引数

array_name

数値メトリックを追加する配列の名前。

metric_name

数値メトリックの名前。

metric_description

メトリックの説明。空の文字列を使用できます。

metric_units

メートル単位。空の文字列を使用できます。

説明
この関数は、数値メトリックを配列に追加し、必要なものをすべて設定します。

名前
function::json_add_array_string_metric – 文字列メトリックを配列に追加する

概要

```
json_add_array_string_metric:long(array_name:string,metric_name:string,metric_description:string)
```

引数

array_name
文字列メトリックを追加する配列の名前。

metric_name
文字列メトリックの名前。

metric_description
メトリックの説明。空の文字列を使用できます。

説明
この関数は、配列に文字列メトリックを追加し、必要なものをすべて設定します。

名前
function::json_add_numeric_metric – 数値指標を追加する

概要

```
json_add_numeric_metric:long(name:string,description:string,units:string)
```

引数

name
数値メトリックの名前。

description
メトリックの説明。空の文字列を使用できます。

units
メートル単位。空の文字列を使用できます。

説明
この関数は数値メトリクスを追加し、必要なものをすべて設定します。

名前
function::json_add_string_metric – 文字列指標を追加する

概要

```
json_add_string_metric:long(name:string,description:string)
```

引数

name
文字列メトリックの名前。

description
メトリックの説明。空の文字列を使用できます。

説明
この関数は文字列メトリックを追加し、必要なものをすべて設定します。

名前
function::json_set_prefix – メトリック 接頭辞を設定します。

概要

```
json_set_prefix:long(prefix:string)
```

引数

prefix
使用する接頭辞名。

説明
この関数は、メトリック階層のベースの名前である 「接頭辞」 を設定します。この関数の呼び出しはオプションです。デフォルトでは、systemtap モジュールの名前が使用されます。

名前
macro::json_output_array_numeric_value – メトリックの数値を配列で出力します。

概要

```
@json_output_array_numeric_value(array_name,array_index,metric_name,value)
```

引数

array_name
配列の名前。

array_index

数値を格納する場所を示す配列インデックス (文字列として)。

metric_name

数値メトリックの名前。

value

出力する数値。

説明

*json_output_array_numeric_value マクロは、ユーザースクリプトの json_data プローブから呼び出され、配列内のメトリックの数値を出力するように設計されています。このメトリックは、**json_add_array_numeric_metric**.*

名前

macro::json_output_array_string_value – メトリックの文字列値を配列で出力します。

概要

```
@json_output_array_string_value(array_name,array_index,metric_name,value)
```

引数

array_name

配列の名前。

array_index

文字列値を格納する場所を示す (文字列としての) 配列インデックス。

metric_name

文字列メトリックの名前。

value

出力する文字列値。

説明

*json_output_array_string_value マクロは、ユーザースクリプトの json_data プローブから呼び出されて、配列内のメトリックの文字列値を出力するように設計されています。このメトリックは、**json_add_array_string_metric**.*

名前

macro::json_output_data_end – json 出力を終了します。

概要

```
@json_output_data_end()
```

引数
なし
説明
json_output_data_end マクロは、ユーザーのスクリプトの json_data プローブから呼び出されるように設計されています。JSON 出力の終わりを示します。

名前
macro::json_output_data_start – json 出力を開始します。

概要

```
@json_output_data_start()
```

引数
なし
説明
json_output_data_start マクロは、ユーザーのスクリプトの json_data プローブから呼び出されるように設計されています。JSON 出力の開始を示します。

名前
macro::json_output_numeric_value – 数値を出力します。

概要

```
@json_output_numeric_value(name,value)
```

引数

name

数値メトリックの名前。

value

出力する数値。

説明
json_output_numeric_value マクロは、メトリックの数値を出力するために、ユーザースクリプトの json_data プローブから呼び出されるように設計されています。このメトリックは、***json_add_numeric_metric***.

名前
macro::json_output_string_value – 文字列値を出力します。

概要

```
@json_output_string_value(name,value)
```

引数

name

文字列メトリックの名前。

value

出力する文字列値。

説明

*json_output_string_value マクロは、メトリックの文字列値を出力するために、ユーザーのスクリプトの json_data プローブから呼び出されるように設計されています。このメトリック は、**json_add_string_metric**.*

名前

probe::json_data – リーダーが JSON データを必要とするたびに発生します。

概要

```
json_data
```

値

なし

コンテキスト

*このプローブは、JSON データが読み取られる直前に起動します。このプローブはデータを収集し、次のマクロを呼び出してデータを JSON 形式で出力する必要があります。初め、@**json_output_data_start**呼び出す必要があります。その呼び出しの後に、次の1 つ以上が続きます (データ項目ごとに1 つの呼び出し)。**json_output_string_value**、@**json_output_numeric_value**、@**json_output_array_string_value**、と @**json_output_array_numeric_value**. ついに @**json_output_data_end**呼び出す必要があります。*

第35章 出力ファイル切り替え TAPSET

出力ファイルを切り替えられるユーティリティー機能。

名前

function::switch_file – 次の出力ファイルに切り替える

概要

```
switch_file()
```

引数

なし

説明

この関数は stapio プロセスにシグナルを送信し、出力がファイルに送信されたときに次の出力ファイルにローテーションするように指示します。

付録A 更新履歴

改訂 7-6 **Tue Oct 30 2018** **Vladimír Slávik**
Release for Red Hat Enterprise Linux 7.6 GA.

改訂 7-5 **Tue Jan 09 2018** **Vladimír Slávik**
Release for Red Hat Enterprise Linux 7.5 Beta.

改訂 7-4 **Wed Jul 26 2017** **Vladimír Slávik**
Release for Red Hat Enterprise Linux 7.4.

改訂 1-4 **Wed Oct 19 2016** **Robert Krátký**
Release for Red Hat Enterprise Linux 7.3.

改訂 1-2 **Thu Mar 10 2016** **Robert Kratky**
Release for Red Hat Enterprise Linux 7.0.

改訂 1-2 **Thu Nov 11 2015** **Robert Kratky**
Release for Red Hat Enterprise Linux 7.2.