SystemTap Tapset Reference Manual

SystemTap

SystemTap Tapset Reference Manual

by SystemTap

Table of Contents

Chapter 1. Introduction

SystemTap provides free software (GPL) infrastructure to simplify the gathering of information about the running Linux system. This assists diagnosis of a performance or functional problem. SystemTap eliminates the need for the developer to go through the tedious and disruptive instrument, recompile, install, and reboot sequence that may be otherwise required to collect data.

SystemTap provides a simple command line interface and scripting language for writing instrumentation for a live running kernel. The instrumentation makes extensive use of the probe points and functions provided in the *tapset* library. This document describes the various probe points and functions.

Chapter 2. Context Functions

The context functions provide additional information about where an event occurred. These functions can provide information such as a backtrace to where the event occurred and the current register values for the processor.

function::addr

function::addr — Address of the current probe point.

Synopsis

```
addr:long()
```

Arguments

None

Description

Returns the instruction pointer from the current probe's register state. Not all probe types have registers though, in which case zero is returned. The returned address is suitable for use with functions like `symname` and `symdata`.

function::asmlinkage

function::asmlinkage — Mark function as declared asmlinkage

Synopsis

```
asmlinkage()
```

Arguments

None

Description

Call this function before accessing arguments using the *_arg functions if the probed kernel function was declared asmlinkage in the source.

function::backtrace

function::backtrace — Hex backtrace of current kernel stack

Synopsis

```
backtrace:string()
```

Arguments

None

Description

This function returns a string of hex addresses that are a backtrace of the kernel stack. Output may be truncated as per maximum string length (MAXSTRINGLEN). See `ubacktrace` for user-space backtrace.

function::caller

function::caller — Return name and address of calling function

Synopsis

```
caller:string()
```

Arguments

None

Description

This function returns the address and name of the calling function. This is equivalent to calling: sprintf(“s 0xx”, symname(`caller_addr`), `caller_addr`)

function::caller_addr

function::caller_addr — Return caller address

Synopsis

```
caller_addr:long()
```

Arguments

None

Description

This function returns the address of the calling function.

function::callers

function::callers — Return first n elements of kernel stack backtrace

Synopsis

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

Arguments

n number of levels to descend in the stack (not counting the top level). If n is -1, print the entire stack.

Description

This function returns a string of the first n hex addresses from the backtrace of the kernel stack. Output may be truncated as per maximum string length (MAXSTRINGLEN).

function::cmdline_arg

function::cmdline_arg — Fetch a command line argument

Synopsis

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

Arguments

n Argument to get (zero is the command itself)

Description

Returns argument the requested argument from the current process or the empty string when there are not that many arguments or there is a problem retrieving the argument. Argument zero is traditionally the command itself.

function::cmdline_args

function::cmdline_args — Fetch command line arguments from current process

Synopsis

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

Arguments

n First argument to get (zero is the command itself)

m Last argument to get (or minus one for all arguments after n)

delim String to use to delimit arguments when more than one.

Description

Returns arguments from the current process starting with argument number n, up to argument m. If there are less than n arguments, or the arguments cannot be retrieved from the current process, the empty string is returned. If m is smaller than n then all arguments starting from argument n are returned. Argument zero is traditionally the command itself.

function::cmdline_str

function::cmdline_str — Fetch all command line arguments from current process

Synopsis

```
cmdline_str:string()
```

Arguments

None

Description

Returns all arguments from the current process delimited by spaces. Returns the empty string when the arguments cannot be retrieved.

function::cpu

function::cpu — Returns the current cpu number

Synopsis

```
cpu:long()
```

Arguments

None

Description

This function returns the current cpu number.

function::cpuid

function::cpuid — Returns the current cpu number

Synopsis

```
cpuid:long()
```

Arguments

None

Description

This function returns the current cpu number. Deprecated in SystemTap 1.4 and removed in SystemTap 1.5.

function::egid

function::egid — Returns the effective gid of a target process

Synopsis

```
egid:long()
```

Arguments

None

Description

This function returns the effective gid of a target process

function::env_var

function::env_var — Fetch environment variable from current process

Synopsis

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

Arguments

name Name of the environment variable to fetch

Description

Returns the contents of the specified environment value for the current process. If the variable isn't set an empty string is returned.

function::euid

function::euid — Return the effective uid of a target process

Synopsis

```
euid:long()
```

Arguments

None

Description

Returns the effective user ID of the target process.

function::execname

function::execname — Returns the execname of a target process (or group of processes)

Synopsis

```
execname:string()
```

Arguments

None

Description

Returns the execname of a target process (or group of processes).

function::fastcall

function::fastcall — Mark function as declared fastcall

Synopsis

```
fastcall()
```

Arguments

None

Description

Call this function before accessing arguments using the *_arg functions if the probed kernel function was declared fastcall in the source.

function::gid

function::gid — Returns the group ID of a target process

Synopsis

```
gid:long()
```

Arguments

None

Description

This function returns the group ID of a target process.

function::int_arg

function::int_arg — Return function argument as signed int

Synopsis

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

Arguments

n index of argument to return

Description

Return the value of argument n as a signed int (i.e., a 32-bit integer sign-extended to 64 bits).

function::is_myproc

function::is_myproc — Determines if the current probe point has occurred in the user's own process

Synopsis

```
is_myproc:long()
```

Arguments

None

Description

This function returns 1 if the current probe point has occurred in the user's own process.

function::is_return

function::is_return — Whether the current probe context is a return probe

Synopsis

```
is_return:long()
```

Arguments

None

Description

Returns 1 if the current probe context is a return probe, returns 0 otherwise.

function::long_arg

function::long_arg — Return function argument as signed long

Synopsis

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

Arguments

n index of argument to return

Description

Return the value of argument n as a signed long. On architectures where a long is 32 bits, the value is sign-extended to 64 bits.

function::longlong_arg

function::longlong_arg — Return function argument as 64-bit value

Synopsis

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

Arguments

n index of argument to return

Description

Return the value of argument n as a 64-bit value.

function::modname

function::modname — Return the kernel module name loaded at the address

Synopsis

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

Arguments

addr The address to map to a kernel module name

Description

Returns the module name associated with the given address if known. If not known it will return the string “<unknown>”. If the address was not in a kernel module, but in the kernel itself, then the string “kernel” will be returned.

function::module_name

function::module_name — The module name of the current script

Synopsis

```
module_name:string()
```

Arguments

None

Description

This function returns the name of the stap module. Either generated randomly (stap_[0-9a-f]+_[0-9a-f]+) or set by stap -m <module_name>.

function::pexecname

function::pexecname — Returns the execname of a target process's parent process

Synopsis

```
pexecname:string()
```

Arguments

None

Description

This function returns the execname of a target process's parent procces.

function::pgrp

function::pgrp — Returns the process group ID of the current process

Synopsis

```
pgrp:long()
```

Arguments

None

Description

This function returns the process group ID of the current process.

function::pid

function::pid — Returns the ID of a target process

Synopsis

```
pid:long()
```

Arguments

None

Description

This function returns the ID of a target process.

function::pid2execname

function::pid2execname — The name of the given process identifier

Synopsis

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

Arguments

pid process identifier

Description

Return the name of the given process id.

function::pid2task

function::pid2task — The task_struct of the given process identifier

Synopsis

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

Arguments

pid process identifier

Description

Return the task struct of the given process id.

function::pn

function::pn — Returns the active probe name

Synopsis

```
pn:string()
```

Arguments

None

Description

This function returns the script-level probe point associated with a currently running probe handler, including wild-card expansion effects. Context: The current probe point.

function::pointer_arg

function::pointer_arg — Return function argument as pointer value

Synopsis

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

Arguments

n index of argument to return

Description

Return the unsigned value of argument n, same as ulong_arg. Can be used with any type of pointer.

function::pp

function::pp — Returns the active probe point

Synopsis

```
pp:string()
```

Arguments

None

Description

This function returns the fully-resolved probe point associated with a currently running probe handler, including alias and wild-card expansion effects. Context: The current probe point.

function::ppfunc

function::ppfunc — Returns the function name parsed from `pp`

Synopsis

```
ppfunc:string()
```

Arguments

None

Description

This returns the function name from the current `pp`. Not all `pp` have functions in them, in which case "" is returned.

function::ppid

function::ppid — Returns the process ID of a target process’s parent process

Synopsis

```
ppid:long()
```

Arguments

None

Description

This function return the process ID of the target proccess’s parent process.

function::print_backtrace

function::print_backtrace — Print kernel stack back trace

Synopsis

```
print_backtrace()
```

Arguments

None

Description

This function is equivalent to print_stack(`backtrace`), except that deeper stack nesting may be supported. See print_ubacktrace for user-space backtrace. The function does not return a value.

function::print_regs

function::print_regs — Print a register dump

Synopsis

```
print_regs()
```

Arguments

None

Description

This function prints a register dump. Does nothing if no registers are available for the probe point.

function::print_stack

function::print_stack — Print out kernel stack from string

Synopsis

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

Arguments

stk String with list of hexadecimal addresses

Description

This function performs a symbolic lookup of the addresses in the given string, which is assumed to be the result of a prior call to `backtrace`.

Print one line per address, including the address, the name of the function containing the address, and an estimate of its position within that function. Return nothing.

NOTE

it is recommended to use `print_syms` instead of this function.

function::print_syms

function::print_syms — Print out kernel stack from string

Synopsis

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

Arguments

callers String with list of hexadecimal (kernel) addresses

Description

This function performs a symbolic lookup of the addresses in the given string, which are assumed to be the result of prior calls to `stack`, `callers`, and similar functions.

Prints one line per address, including the address, the name of the function containing the address, and an estimate of its position within that function, as obtained by `symdata`. Returns nothing.

function::print_ubacktrace

function::print_ubacktrace — Print stack back trace for current user-space task.

Synopsis

```
print_ubacktrace()
```

Arguments

None

Description

Equivalent to print_ustack(`ubacktrace`), except that deeper stack nesting may be supported. Returns nothing. See `print_backtrace` for kernel backtrace.

Note

To get (full) backtraces for user space applications and shared shared libraries not mentioned in the current script run stap with -d /path/to/exe-or-so and/or add --ldd to load all needed unwind data.

function::print_ubacktrace_brief

function::print_ubacktrace_brief — Print stack back trace for current user-space task.

Synopsis

```
print_ubacktrace_brief()
```

Arguments

None

Description

Equivalent to `print_ubacktrace`, but output for each symbol is shorter (just name and offset, or just the hex address of no symbol could be found).

Note

To get (full) backtraces for user space applications and shared shared libraries not mentioned in the current script run stap with -d /path/to/exe-or-so and/or add --ldd to load all needed unwind data.

function::print_ustack

function::print_ustack — Print out stack for the current task from string.

Synopsis

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

Arguments

stk String with list of hexadecimal addresses for the current task.

Description

Perform a symbolic lookup of the addresses in the given string, which is assumed to be the result of a prior call to `ubacktrace` for the current task.

Print one line per address, including the address, the name of the function containing the address, and an estimate of its position within that function. Return nothing.

NOTE

it is recommended to use `print_usyms` instead of this function.

function::print_usyms

function::print_usyms — Print out user stack from string

Synopsis

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

Arguments

callers String with list of hexadecimal (user) addresses

Description

This function performs a symbolic lookup of the addresses in the given string, which are assumed to be the result of prior calls to `ustack`, `ucallers`, and similar functions.

Prints one line per address, including the address, the name of the function containing the address, and an estimate of its position within that function, as obtained by `usymdata`. Returns nothing.

function::probe_type

function::probe_type — The low level probe handler type of the current probe.

Synopsis

```
probe_type:string()
```

Arguments

None

Description

Returns a short string describing the low level probe handler type for the current probe point. This is for informational purposes only. Depending on the low level probe handler different context functions can or cannot provide information about the current event (for example some probe handlers only trigger in user space and have no associated kernel context). High-level probes might map to the same or different low-level probes (depending on systemtap version and/or kernel used).

function::probefunc

function::probefunc — Return the probe point's function name, if known

Synopsis

```
probefunc:string()
```

Arguments

None

Description

This function returns the name of the function being probed based on the current address, as computed by symname(`addr`) or usymname(`uaddr`) depending on probe context (whether the probe is a user probe or a kernel probe).

Please note

this function's behaviour differs between SystemTap 2.0 and earlier versions. Prior to 2.0, `probefunc` obtained the function name from the probe point string as returned by `pp`, and used the current address as a fallback.

function::probemod

function::probemod — Return the probe point's kernel module name

Synopsis

```
probemod:string()
```

Arguments

None

Description

This function returns the name of the kernel module containing the probe point, if known.

function::pstrace

function::pstrace — Chain of processes and pids back to init(1)

Synopsis

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

Arguments

task Pointer to task struct of process

Description

This function returns a string listing execname and pid for each process starting from *task* back to the process ancestor that init(1) spawned.

function::register

function::register — Return the signed value of the named CPU register

Synopsis

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

Arguments

name Name of the register to return

Description

Return the value of the named CPU register, as it was saved when the current probe point was hit. If the register is 32 bits, it is sign-extended to 64 bits.

For the i386 architecture, the following names are recognized. (name1/name2 indicates that name1 and name2 are alternative names for the same register.) 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.

For the x86_64 architecture, the following names are recognized: 64-bit registers: 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-bit registers: eax, ebp, ebx, ecx, edx, edi, edx, eip, esi, esp, flags/eflags, orig_eax; segment registers: xcs/cs, xss/ss.

For powerpc, the following names are recognized: r0, r1, ... r31, nip, msr, orig_gpr3, ctr, link, xer, ccr, softe, trap, dar, dsisr, result.

For s390x, the following names are recognized: r0, r1, ... r15, args, psw.mask, psw.addr, orig_gpr2, ilc, trap.

function::registers_valid

function::registers_valid — Determines validity of `register` and `u_register` in current context

Synopsis

```
registers_valid:long()
```

Arguments

None

Description

This function returns 1 if `register` and `u_register` can be used in the current context, or 0 otherwise. For example, `registers_valid` returns 0 when called from a begin or end probe.

function::regparm

function::regparm — Specify regparm value used to compile function

Synopsis

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

Arguments

n original regparm value

Description

Call this function with argument n before accessing function arguments using the *_arg function is the function was build with the gcc -mregparm=n option.

(The i386 kernel is built with \-mregparm=3, so systemtap considers regparm(3) the default for kernel functions on that architecture.) Only valid on i386 and x86_64 (when probing 32bit applications). Produces an error on other architectures.

function::remote_id

function::remote_id — The index of this instance in a remote execution.

Synopsis

```
remote_id:long()
```

Arguments

None

Description

This function returns a number 0..N, which is the unique index of this particular script execution from a swarm of “stap --remote A --remote B ...” runs, and is the same number “stap --remote-prefix” would print. The function returns -1 if the script was not launched with “stap --remote”, or if the remote staprun/stapsh are older than version 1.7.

function::remote_uri

function::remote_uri — The name of this instance in a remote execution.

Synopsis

```
remote_uri:string()
```

Arguments

None

Description

This function returns the remote host used to invoke this particular script execution from a swarm of “stap --remote” runs. It may not be unique among the swarm. The function returns an empty string if the script was not launched with “stap --remote”.

function::s32_arg

function::s32_arg — Return function argument as signed 32-bit value

Synopsis

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

Arguments

n index of argument to return

Description

Return the signed 32-bit value of argument n, same as int_arg.

function::s64_arg

function::s64_arg — Return function argument as signed 64-bit value

Synopsis

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

Arguments

n index of argument to return

Description

Return the signed 64-bit value of argument n, same as longlong_arg.

function::sid

function::sid — Returns the session ID of the current process

Synopsis

```
sid:long()
```

Arguments

None

Description

The session ID of a process is the process group ID of the session leader. Session ID is stored in the signal_struct since Kernel 2.6.0.

function::sprint_backtrace

function::sprint_backtrace — Return stack back trace as string

Synopsis

```
sprint_backtrace:string()
```

Arguments

None

Description

Returns a simple (kernel) backtrace. One line per address. Includes the symbol name (or hex address if symbol couldn't be resolved) and module name (if found). Includes the offset from the start of the function if found, otherwise the offset will be added to the module (if found, between brackets). Returns the backtrace as string (each line terminated by a newline character). Note that the returned stack will be truncated to MAXSTRINGLEN, to print fuller and richer stacks use `print_backtrace`. Equivalent to sprint_stack(`backtrace`), but more efficient (no need to translate between hex strings and final backtrace string).

function::sprint_stack

function::sprint_stack — Return stack for kernel addresses from string

Synopsis

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

Arguments

stk String with list of hexadecimal (kernel) addresses

Description

Perform a symbolic lookup of the addresses in the given string, which is assumed to be the result of a prior call to `backtrace`.

Returns a simple backtrace from the given hex string. One line per address. Includes the symbol name (or hex address if symbol couldn't be resolved) and module name (if found). Includes the offset from the start of the function if found, otherwise the offset will be added to the module (if found, between brackets). Returns the backtrace as string (each line terminated by a newline character). Note that the returned stack will be truncated to MAXSTRINGLEN, to print fuller and richer stacks use print_stack.

NOTE

it is recommended to use `sprint_syms` instead of this function.

function::sprint_syms

function::sprint_syms — Return stack for kernel addresses from string

Synopsis

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

Arguments

callers String with list of hexadecimal (kernel) addresses

Description

Perform a symbolic lookup of the addresses in the given string, which are assumed to be the result of a prior calls to `stack`, `callers`, and similar functions.

Returns a simple backtrace from the given hex string. One line per address. Includes the symbol name (or hex address if symbol couldn't be resolved) and module name (if found), as obtained from `symdata`. Includes the offset from the start of the function if found, otherwise the offset will be added to the module (if found, between brackets). Returns the backtrace as string (each line terminated by a newline character). Note that the returned stack will be truncated to MAXSTRINGLEN, to print fuller and richer stacks use `print_syms`.

function::sprint_ubacktrace

function::sprint_ubacktrace — Return stack back trace for current user-space task as string.

Synopsis

```
sprint_ubacktrace:string()
```

Arguments

None

Description

Returns a simple backtrace for the current task. One line per address. Includes the symbol name (or hex address if symbol couldn't be resolved) and module name (if found). Includes the offset from the start of the function if found, otherwise the offset will be added to the module (if found, between brackets). Returns the backtrace as string (each line terminated by a newline character). Note that the returned stack will be truncated to MAXSTRINGLEN, to print fuller and richer stacks use `print_ubacktrace`. Equivalent to sprint_ustack(`ubacktrace`), but more efficient (no need to translate between hex strings and final backtrace string).

Note

To get (full) backtraces for user space applications and shared shared libraries not mentioned in the current script run stap with -d /path/to/exe-or-so and/or add --ldd to load all needed unwind data.

function::sprint_ustack

function::sprint_ustack — Return stack for the current task from string.

Synopsis

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

Arguments

stk String with list of hexadecimal addresses for the current task.

Description

Perform a symbolic lookup of the addresses in the given string, which is assumed to be the result of a prior call to `ubacktrace` for the current task.

Returns a simple backtrace from the given hex string. One line per address. Includes the symbol name (or hex address if symbol couldn't be resolved) and module name (if found). Includes the offset from the start of the function if found, otherwise the offset will be added to the module (if found, between brackets). Returns the backtrace as string (each line terminated by a newline character). Note that the returned stack will be truncated to MAXSTRINGLEN, to print fuller and richer stacks use print_ustack.

NOTE

it is recommended to use `sprint_usyms` instead of this function.

function::sprint_usyms

function::sprint_usyms — Return stack for user addresses from string

Synopsis

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

Arguments

callers String with list of hexadecimal (user) addresses

Description

Perform a symbolic lookup of the addresses in the given string, which are assumed to be the result of a prior calls to `ustack`, `ucallers`, and similar functions.

Returns a simple backtrace from the given hex string. One line per address. Includes the symbol name (or hex address if symbol couldn't be resolved) and module name (if found), as obtained from `usymdata`. Includes the offset from the start of the function if found, otherwise the offset will be added to the module (if found, between brackets). Returns the backtrace as string (each line terminated by a newline character). Note that the returned stack will be truncated to MAXSTRINGLEN, to print fuller and richer stacks use `print_usyms`.

function::stack

function::stack — Return address at given depth of kernel stack backtrace

Synopsis

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

Arguments

n number of levels to descend in the stack.

Description

Performs a simple (kernel) backtrace, and returns the element at the specified position. The results of the backtrace itself are cached, so that the backtrace computation is performed at most once no matter how many times `stack` is called, or in what order.

function::stack_size

function::stack_size — Return the size of the kernel stack

Synopsis

```
stack_size:long()
```

Arguments

None

Description

This function returns the size of the kernel stack.

function::stack_unused

function::stack_unused — Returns the amount of kernel stack currently available

Synopsis

```
stack_unused:long()
```

Arguments

None

Description

This function determines how many bytes are currently available in the kernel stack.

function::stack_used

function::stack_used — Returns the amount of kernel stack used

Synopsis

```
stack_used:long()
```

Arguments

None

Description

This function determines how many bytes are currently used in the kernel stack.

function::stp_pid

function::stp_pid — The process id of the stapio process

Synopsis

```
stp_pid:long()
```

Arguments

None

Description

This function returns the process id of the stapio process that launched this script. There could be other SystemTap scripts and stapio processes running on the system.

function::symdata

function::symdata — Return the kernel symbol and module offset for the address

Synopsis

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

Arguments

addr The address to translate

Description

Returns the (function) symbol name associated with the given address if known, the offset from the start and size of the symbol, plus module name (between brackets). If symbol is unknown, but module is known, the offset inside the module, plus the size of the module is added. If any element is not known it will be omitted and if the symbol name is unknown it will return the hex string for the given address.

function::symname

function::symname — Return the kernel symbol associated with the given address

Synopsis

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

Arguments

addr The address to translate

Description

Returns the (function) symbol name associated with the given address if known. If not known it will return the hex string representation of addr.

function::target

function::target — Return the process ID of the target process

Synopsis

```
target:long()
```

Arguments

None

Description

This function returns the process ID of the target process. This is useful in conjunction with the -x PID or -c CMD command-line options to stap. An example of its use is to create scripts that filter on a specific process.

-x <pid> `target` returns the pid specified by -x

-c <command> `target` returns the pid for the executed command specified by -c

function::task_backtrace

function::task_backtrace — Hex backtrace of an arbitrary task

Synopsis

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

Arguments

task pointer to task_struct

Description

This function returns a string of hex addresses that are a backtrace of the stack of a particular task Output may be truncated as per maximum string length. Deprecated in SystemTap 1.6.

function::task_cpu

function::task_cpu — The scheduled cpu of the task

Synopsis

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

Arguments

task task_struct pointer

Description

This function returns the scheduled cpu for the given task.

function::task_current

function::task_current — The current task_struct of the current task

Synopsis

```
task_current:long()
```

Arguments

None

Description

This function returns the task_struct representing the current process. This address can be passed to the various task_*() functions to extract more task-specific data.

function::task_egid

function::task_egid — The effective group identifier of the task

Synopsis

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

Arguments

task task_struct pointer

Description

This function returns the effective group id of the given task.

function::task_euid

function::task_euid — The effective user identifier of the task

Synopsis

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

Arguments

task task_struct pointer

Description

This function returns the effective user id of the given task.

function::task_execname

function::task_execname — The name of the task

Synopsis

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

Arguments

task task_struct pointer

Description

Return the name of the given task.

function::task_gid

function::task_gid — The group identifier of the task

Synopsis

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

Arguments

task task_struct pointer

Description

This function returns the group id of the given task.

function::task_max_file_handles

function::task_max_file_handles — The max number of open files for the task

Synopsis

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

Arguments

task task_struct pointer

Description

This function returns the maximum number of file handlers for the given task.

function::task_nice

function::task_nice — The nice value of the task

Synopsis

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

Arguments

task task_struct pointer

Description

This function returns the nice value of the given task.

function::task_open_file_handles

function::task_open_file_handles — The number of open files of the task

Synopsis

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

Arguments

task task_struct pointer

Description

This function returns the number of open file handlers for the given task.

function::task_parent

function::task_parent — The task_struct of the parent task

Synopsis

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

Arguments

task task_struct pointer

Description

This function returns the parent task_struct of the given task. This address can be passed to the various task_*() functions to extract more task-specific data.

function::task_pid

function::task_pid — The process identifier of the task

Synopsis

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

Arguments

task task_struct pointer

Description

This fucntion returns the process id of the given task.

function::task_prio

function::task_prio — The priority value of the task

Synopsis

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

Arguments

task task_struct pointer

Description

This function returns the priority value of the given task.

function::task_state

function::task_state — The state of the task

Synopsis

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

Arguments

task task_struct pointer

Description

Return the state of the given task, one of: TASK_RUNNING (0), TASK_INTERRUPTIBLE (1), TASK_UNINTERRUPTIBLE (2), TASK_STOPPED (4), TASK_TRACED (8), EXIT_ZOMBIE (16), or EXIT_DEAD (32).

function::task_tid

function::task_tid — The thread identifier of the task

Synopsis

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

Arguments

task task_struct pointer

Description

This function returns the thread id of the given task.

function::task_uid

function::task_uid — The user identifier of the task

Synopsis

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

Arguments

task task_struct pointer

Description

This function returns the user id of the given task.

function::tid

function::tid — Returns the thread ID of a target process

Synopsis

```
tid:long()
```

Arguments

None

Description

This function returns the thread ID of the target process.

function::u32_arg

function::u32_arg — Return function argument as unsigned 32-bit value

Synopsis

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

Arguments

n index of argument to return

Description

Return the unsigned 32-bit value of argument n, same as uint_arg.

function::u64_arg

function::u64_arg — Return function argument as unsigned 64-bit value

Synopsis

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

Arguments

n index of argument to return

Description

Return the unsigned 64-bit value of argument n, same as ulonglong_arg.

function::u_register

function::u_register — Return the unsigned value of the named CPU register

Synopsis

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

Arguments

name Name of the register to return

Description

Same as register(name), except that if the register is 32 bits wide, it is zero-extended to 64 bits.

function::uaddr

function::uaddr — User space address of current running task

Synopsis

```
uaddr:long()
```

Arguments

None

Description

Returns the address in userspace that the current task was at when the probe occurred. When the current running task isn't a user space thread, or the address cannot be found, zero is returned. Can be used to see where the current task is combined with `usymname` or `usymdata`. Often the task will be in the VDSO where it entered the kernel.

function::ubacktrace

function::ubacktrace — Hex backtrace of current user-space task stack.

Synopsis

```
ubacktrace:string()
```

Arguments

None

Description

Return a string of hex addresses that are a backtrace of the stack of the current task. Output may be truncated as per maximum string length. Returns empty string when current probe point cannot determine user backtrace. See `backtrace` for kernel traceback.

Note

To get (full) backtraces for user space applications and shared shared libraries not mentioned in the current script run stap with -d /path/to/exe-or-so and/or add --ldd to load all needed unwind data.

function::ucallers

function::ucallers — Return first n elements of user stack backtrace

Synopsis

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

Arguments

n number of levels to descend in the stack (not counting the top level). If n is -1, print the entire stack.

Description

This function returns a string of the first n hex addresses from the backtrace of the user stack. Output may be truncated as per maximum string length (MAXSTRINGLEN).

Note

To get (full) backtraces for user space applications and shared shared libraries not mentioned in the current script run stap with -d /path/to/exe-or-so and/or add --ldd to load all needed unwind data.

function::uid

function::uid — Returns the user ID of a target process

Synopsis

```
uid:long()
```

Arguments

None

Description

This function returns the user ID of the target process.

function::uint_arg

function::uint_arg — Return function argument as unsigned int

Synopsis

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

Arguments

n index of argument to return

Description

Return the value of argument n as an unsigned int (i.e., a 32-bit integer zero-extended to 64 bits).

function::ulong_arg

function::ulong_arg — Return function argument as unsigned long

Synopsis

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

Arguments

n index of argument to return

Description

Return the value of argument n as an unsigned long. On architectures where a long is 32 bits, the value is zero-extended to 64 bits.

function::ulonglong_arg

function::ulonglong_arg — Return function argument as 64-bit value

Synopsis

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

Arguments

n index of argument to return

Description

Return the value of argument n as a 64-bit value. (Same as longlong_arg.)

function::umodname

function::umodname — Returns the (short) name of the user module.

Synopsis

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

Arguments

addr User-space address

Description

Returns the short name of the user space module for the current task that that the given address is part of. Returns “<unknown>” when the address isn’t in a (mapped in) module, or the module cannot be found for some reason.

function::user_mode

function::user_mode — Determines if probe point occurs in user-mode

Synopsis

```
user_mode:long()
```

Arguments

None

Description

Return 1 if the probe point occurred in user-mode.

function::ustack

function::ustack — Return address at given depth of user stack backtrace

Synopsis

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

Arguments

n number of levels to descend in the stack.

Description

Performs a simple (user space) backtrace, and returns the element at the specified position. The results of the backtrace itself are cached, so that the backtrace computation is performed at most once no matter how many times `ustack` is called, or in what order.

function::usymdata

function::usymdata — Return the symbol and module offset of an address.

Synopsis

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

Arguments

addr The address to translate.

Description

Returns the (function) symbol name associated with the given address in the current task if known, the offset from the start and the size of the symbol, plus the module name (between brackets). If symbol is unknown, but module is known, the offset inside the module, plus the size of the module is added. If any element is not known it will be omitted and if the symbol name is unknown it will return the hex string for the given address.

function::usymname

function::usymname — Return the symbol of an address in the current task.

Synopsis

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

Arguments

addr The address to translate.

Description

Returns the (function) symbol name associated with the given address if known. If not known it will return the hex string representation of addr.

Chapter 3. Timestamp Functions

Each timestamp function returns a value to indicate when a function is executed. These returned values can then be used to indicate when an event occurred, provide an ordering for events, or compute the amount of time elapsed between two time stamps.

function::HZ

function::HZ — Kernel HZ

Synopsis

```
HZ:long()
```

Arguments

None

Description

This function returns the value of the kernel HZ macro, which corresponds to the rate of increase of the jiffies value.

function::cpu_clock_ms

function::cpu_clock_ms — Number of milliseconds on the given cpu's clock

Synopsis

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

Arguments

cpu Which processor's clock to read

Description

This function returns the number of milliseconds on the given cpu's clock. This is always monotonic comparing on the same cpu, but may have some drift between cpus (within about a jiffy).

function::cpu_clock_ns

function::cpu_clock_ns — Number of nanoseconds on the given cpu's clock

Synopsis

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

Arguments

cpu Which processor's clock to read

Description

This function returns the number of nanoseconds on the given cpu's clock. This is always monotonic comparing on the same cpu, but may have some drift between cpus (within about a jiffy).

function::cpu_clock_s

function::cpu_clock_s — Number of seconds on the given cpu's clock

Synopsis

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

Arguments

cpu Which processor's clock to read

Description

This function returns the number of seconds on the given cpu's clock. This is always monotonic comparing on the same cpu, but may have some drift between cpus (within about a jiffy).

function::cpu_clock_us

function::cpu_clock_us — Number of microseconds on the given cpu's clock

Synopsis

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

Arguments

cpu Which processor's clock to read

Description

This function returns the number of microseconds on the given cpu's clock. This is always monotonic comparing on the same cpu, but may have some drift between cpus (within about a jiffy).

function::delete_stopwatch

function::delete_stopwatch — Remove an existing stopwatch

Synopsis

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

Arguments

name the stopwatch name

Description

Remove stopwatch *name*.

function::get_cycles

function::get_cycles — Processor cycle count

Synopsis

```
get_cycles:long()
```

Arguments

None

Description

This function returns the processor cycle counter value if available, else it returns zero. The cycle counter is free running and unsynchronized on each processor. Thus, the order of events cannot determined by comparing the results of the get_cycles function on different processors.

function::gettimeofday_ms

function::gettimeofday_ms — Number of milliseconds since UNIX epoch

Synopsis

```
gettimeofday_ms:long()
```

Arguments

None

Description

This function returns the number of milliseconds since the UNIX epoch.

function::gettimeofday_ns

function::gettimeofday_ns — Number of nanoseconds since UNIX epoch

Synopsis

```
gettimeofday_ns:long()
```

Arguments

None

Description

This function returns the number of nanoseconds since the UNIX epoch.

function::gettimeofday_s

function::gettimeofday_s — Number of seconds since UNIX epoch

Synopsis

```
gettimeofday_s:long()
```

Arguments

None

Description

This function returns the number of seconds since the UNIX epoch.

function::gettimeofday_us

function::gettimeofday_us — Number of microseconds since UNIX epoch

Synopsis

```
gettimeofday_us:long()
```

Arguments

None

Description

This function returns the number of microseconds since the UNIX epoch.

function::jiffies

function::jiffies — Kernel jiffies count

Synopsis

```
jiffies:long()
```

Arguments

None

Description

This function returns the value of the kernel jiffies variable. This value is incremented periodically by timer interrupts, and may wrap around a 32-bit or 64-bit boundary. See `HZ`.

function::local_clock_ms

function::local_clock_ms — Number of milliseconds on the local cpu's clock

Synopsis

```
local_clock_ms:long()
```

Arguments

None

Description

This function returns the number of milliseconds on the local cpu's clock. This is always monotonic comparing on the same cpu, but may have some drift between cpus (within about a jiffy).

function::local_clock_ns

function::local_clock_ns — Number of nanoseconds on the local cpu's clock

Synopsis

```
local_clock_ns:long()
```

Arguments

None

Description

This function returns the number of nanoseconds on the local cpu's clock. This is always monotonic comparing on the same cpu, but may have some drift between cpus (within about a jiffy).

function::local_clock_s

function::local_clock_s — Number of seconds on the local cpu's clock

Synopsis

```
local_clock_s:long()
```

Arguments

None

Description

This function returns the number of seconds on the local cpu's clock. This is always monotonic comparing on the same cpu, but may have some drift between cpus (within about a jiffy).

function::local_clock_us

function::local_clock_us — Number of microseconds on the local cpu's clock

Synopsis

```
local_clock_us:long()
```

Arguments

None

Description

This function returns the number of microseconds on the local cpu's clock. This is always monotonic comparing on the same cpu, but may have some drift between cpus (within about a jiffy).

function::read_stopwatch_ms

function::read_stopwatch_ms — Reads the time in milliseconds for a stopwatch

Synopsis

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

Arguments

name stopwatch name

Description

Returns time in milliseconds for stopwatch *name*. Creates stopwatch *name* if it does not currently exist.

function::read_stopwatch_ns

function::read_stopwatch_ns — Reads the time in nanoseconds for a stopwatch

Synopsis

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

Arguments

name stopwatch name

Description

Returns time in nanoseconds for stopwatch *name*. Creates stopwatch *name* if it does not currently exist.

function::read_stopwatch_s

function::read_stopwatch_s — Reads the time in seconds for a stopwatch

Synopsis

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

Arguments

name stopwatch name

Description

Returns time in seconds for stopwatch *name*. Creates stopwatch *name* if it does not currently exist.

function::read_stopwatch_us

function::read_stopwatch_us — Reads the time in microseconds for a stopwatch

Synopsis

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

Arguments

name stopwatch name

Description

Returns time in microseconds for stopwatch *name*. Creates stopwatch *name* if it does not currently exist.

function::start_stopwatch

function::start_stopwatch — Start a stopwatch

Synopsis

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

Arguments

name the stopwatch name

Description

Start stopwatch *name*. Creates stopwatch *name* if it does not currently exist.

function::stop_stopwatch

function::stop_stopwatch — Stop a stopwatch

Synopsis

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

Arguments

name the stopwatch name

Description

Stop stopwatch *name*. Creates stopwatch *name* if it does not currently exist.

Chapter 4. Time utility functions

Utility functions to turn seconds since the epoch (as returned by the timestamp function gettimeofday_s()) into a human readable date/time strings.

function::ctime

function::ctime — Convert seconds since epoch into human readable date/time string

Synopsis

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

Arguments

epochsecs Number of seconds since epoch (as returned by `gettimeofday_s`)

Description

Takes an argument of seconds since the epoch as returned by `gettimeofday_s`. Returns a string of the form

"Wed Jun 30 21:49:08 1993"

The string will always be exactly 24 characters. If the time would be unreasonable far in the past (before what can be represented with a 32 bit offset in seconds from the epoch) the returned string will be "a long, long time ago...". If the time would be unreasonable far in the future the returned string will be "far far in the future..." (both these strings are also 24 characters wide).

Note that the epoch (zero) corresponds to

"Thu Jan 1 00:00:00 1970"

The earliest full date given by ctime, corresponding to epochsecs -2147483648 is "Fri Dec 13 20:45:52 1901". The latest full date given by ctime, corresponding to epochsecs 2147483647 is "Tue Jan 19 03:14:07 2038".

The abbreviations for the days of the week are 'Sun', 'Mon', 'Tue', 'Wed', 'Thu', 'Fri', and 'Sat'. The abbreviations for the months are 'Jan', 'Feb', 'Mar', 'Apr', 'May', 'Jun', 'Jul', 'Aug', 'Sep', 'Oct', 'Nov', and 'Dec'.

Note that the real C library `ctime` function puts a newline ('\n') character at the end of the string that this function does not. Also note that since the kernel has no concept of timezones, the returned time is always in GMT.

function::tz_ctime

function::tz_ctime — Convert seconds since epoch into human readable date/time string, with local time zone

Synopsis

```
tz_ctime(epochsecs:)
```

Arguments

epochsecs number of seconds since epoch (as returned by `gettimeofday_s`)

Description

Takes an argument of seconds since the epoch as returned by `gettimeofday_s`. Returns a string of the same form as `ctime`, but offsets the epoch time for the local time zone, and appends the name of the local time zone. The string length may vary. The time zone information is passed by staprun at script startup only.

function::tz_gmtoff

function::tz_gmtoff — Return local time zone offset

Synopsis

```
tz_gmtoff()
```

Arguments

None

Description

Returns the local time zone offset (seconds west of UTC), as passed by staprun at script startup only.

function::tz_name

function::tz_name — Return local time zone name

Synopsis

```
tz_name()
```

Arguments

None

Description

Returns the local time zone name, as passed by staprun at script startup only.

Chapter 5. Shell command functions

Utility functions to enqueue shell commands.

function::system

function::system — Issue a command to the system

Synopsis

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

Arguments

cmd the command to issue to the system

Description

This function runs a command on the system. The command is started in the background some time after the current probe completes. The command is run with the same UID as the user running the stap or staprun command.

Chapter 6. Memory Tapset

This family of probe points is used to probe memory-related events or query the memory usage of the current process. It contains the following probe points:

function::addr_to_node

function::addr_to_node — Returns which node a given address belongs to within a NUMA system

Synopsis

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

Arguments

addr the address of the faulting memory access

Description

This function accepts an address, and returns the node that the given address belongs to in a NUMA system.

function::bytes_to_string

function::bytes_to_string — Human readable string for given bytes

Synopsis

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

Arguments

bytes Number of bytes to translate.

Description

Returns a string representing the number of bytes (up to 1024 bytes), the number of kilobytes (when less than 1024K) postfixed by 'K', the number of megabytes (when less than 1024M) postfixed by 'M' or the number of gigabytes postfixed by 'G'. If representing K, M or G, and the number is amount is less than 100, it includes a '.' plus the remainer. The returned string will be 5 characters wide (padding with whitespace at the front) unless negative or representing more than 9999G bytes.

function::mem_page_size

function::mem_page_size — Number of bytes in a page for this architecture

Synopsis

```
mem_page_size:long()
```

Arguments

None

function::pages_to_string

function::pages_to_string — Turns pages into a human readable string

Synopsis

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

Arguments

pages Number of pages to translate.

Description

Multiplies pages by `page_size` to get the number of bytes and returns the result of `bytes_to_string`.

function::proc_mem_data

function::proc_mem_data — Program data size (data + stack) in pages

Synopsis

```
proc_mem_data:long()
```

Arguments

None

Description

Returns the current process data size (data + stack) in pages, or zero when there is no current process or the number of pages couldn't be retrieved.

function::proc_mem_data_pid

function::proc_mem_data_pid — Program data size (data + stack) in pages

Synopsis

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

Arguments

pid The pid of process to examine

Description

Returns the given process data size (data + stack) in pages, or zero when the process doesn’t exist or the number of pages couldn’t be retrieved.

function::proc_mem_rss

function::proc_mem_rss — Program resident set size in pages

Synopsis

```
proc_mem_rss:long()
```

Arguments

None

Description

Returns the resident set size in pages of the current process, or zero when there is no current process or the number of pages couldn't be retrieved.

function::proc_mem_rss_pid

function::proc_mem_rss_pid — Program resident set size in pages

Synopsis

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

Arguments

pid The pid of process to examine

Description

Returns the resident set size in pages of the given process, or zero when the process doesn't exist or the number of pages couldn't be retrieved.

function::proc_mem_shr

function::proc_mem_shr — Program shared pages (from shared mappings)

Synopsis

```
proc_mem_shr:long()
```

Arguments

None

Description

Returns the shared pages (from shared mappings) of the current process, or zero when there is no current process or the number of pages couldn't be retrieved.

function::proc_mem_shr_pid

function::proc_mem_shr_pid — Program shared pages (from shared mappings)

Synopsis

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

Arguments

pid The pid of process to examine

Description

Returns the shared pages (from shared mappings) of the given process, or zero when the process doesn't exist or the number of pages couldn't be retrieved.

function::proc_mem_size

function::proc_mem_size — Total program virtual memory size in pages

Synopsis

```
proc_mem_size:long()
```

Arguments

None

Description

Returns the total virtual memory size in pages of the current process, or zero when there is no current process or the number of pages couldn't be retrieved.

function::proc_mem_size_pid

function::proc_mem_size_pid — Total program virtual memory size in pages

Synopsis

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

Arguments

pid The pid of process to examine

Description

Returns the total virtual memory size in pages of the given process, or zero when that process doesn't exist or the number of pages couldn't be retrieved.

function::proc_mem_string

function::proc_mem_string — Human readable string of current proc memory usage

Synopsis

```
proc_mem_string:string()
```

Arguments

None

Description

Returns a human readable string showing the size, rss, shr, txt and data of the memory used by the current process. For example “size: 301m, rss: 11m, shr: 8m, txt: 52k, data: 2248k”.

function::proc_mem_string_pid

function::proc_mem_string_pid — Human readable string of process memory usage

Synopsis

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

Arguments

pid The pid of process to examine

Description

Returns a human readable string showing the size, rss, shr, txt and data of the memory used by the given process. For example “size: 301m, rss: 11m, shr: 8m, txt: 52k, data: 2248k”.

function::proc_mem_txt

function::proc_mem_txt — Program text (code) size in pages

Synopsis

```
proc_mem_txt:long()
```

Arguments

None

Description

Returns the current process text (code) size in pages, or zero when there is no current process or the number of pages couldn't be retrieved.

function::proc_mem_txt_pid

function::proc_mem_txt_pid — Program text (code) size in pages

Synopsis

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

Arguments

pid The pid of process to examine

Description

Returns the given process text (code) size in pages, or zero when the process doesn’t exist or the number of pages couldn’t be retrieved.

function::vm_fault_contains

function::vm_fault_contains — Test return value for page fault reason

Synopsis

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

Arguments

value the fault_type returned by vm.page_fault.return

test the type of fault to test for (VM_FAULT_OOM or similar)

probe::vm.brk

probe::vm.brk — Fires when a brk is requested (i.e. the heap will be resized)

Synopsis

```
vm.brk
```

Values

length	the length of the memory segment
name	name of the probe point
address	the requested address

Context

The process calling brk.

probe::vm.kfree

probe::vm.kfree — Fires when kfree is requested

Synopsis

```
vm.kfree
```

Values

ptr	pointer to the kmemory allocated which is returned by kmalloc
caller_function	name of the caller function.
call_site	address of the function calling this kmemory function
name	name of the probe point

probe::vm.kmalloc

probe::vm.kmalloc — Fires when kmalloc is requested

Synopsis

```
vm.kmalloc
```

Values

ptr	pointer to the kmemory allocated
caller_function	name of the caller function
call_site	address of the kmemory function
gfp_flag_name	type of kmemory to allocate (in String format)
name	name of the probe point
bytes_req	requested Bytes
bytes_alloc	allocated Bytes
gfp_flags	type of kmemory to allocate

probe::vm.kmalloc_node

probe::vm.kmalloc_node — Fires when kmalloc_node is requested

Synopsis

```
vm.kmalloc_node
```

Values

ptr	pointer to the kmemory allocated
caller_function	name of the caller function
call_site	address of the function caling this kmemory function
gfp_flag_name	type of kmemory to allocate(in string format)
name	name of the probe point
bytes_req	requested Bytes
bytes_alloc	allocated Bytes
gfp_flags	type of kmemory to allocate

probe::vm.kmem_cache_alloc

probe::vm.kmem_cache_alloc — Fires when kmem_cache_alloc is requested

Synopsis

```
vm.kmem_cache_alloc
```

Values

ptr	pointer to the kmemory allocated
caller_function	name of the caller function.
call_site	address of the function calling this kmemory function.
gfp_flag_name	type of kmemory to allocate(in string format)
name	name of the probe point
bytes_req	requested Bytes
bytes_alloc	allocated Bytes
gfp_flags	type of kmemory to allocate

probe::vm.kmem_cache_alloc_node

probe::vm.kmem_cache_alloc_node — Fires when kmem_cache_alloc_node is requested

Synopsis

```
vm.kmem_cache_alloc_node
```

Values

ptr	pointer to the kmemory allocated
caller_function	name of the caller function
call_site	address of the function calling this kmemory function
gfp_flag_name	type of kmemory to allocate(in string format)
name	name of the probe point
bytes_req	requested Bytes
bytes_alloc	allocated Bytes
gfp_flags	type of kmemory to allocate

probe::vm.kmem_cache_free

probe::vm.kmem_cache_free — Fires when kmem_cache_free is requested

Synopsis

```
vm.kmem_cache_free
```

Values

ptr
: Pointer to the kmemory allocated which is returned by kmem_cache

caller_function
: Name of the caller function.

call_site
: Address of the function calling this kmemory function

name
: Name of the probe point

probe::vm.mmap

probe::vm.mmap — Fires when an mmap is requested

Synopsis

```
vm.mmap
```

Values

length	the length of the memory segment
name	name of the probe point
address	the requested address

Context

The process calling mmap.

probe::vm.munmap

probe::vm.munmap — Fires when an munmap is requested

Synopsis

```
vm.munmap
```

Values

length	the length of the memory segment
name	name of the probe point
address	the requested address

Context

The process calling munmap.

probe::vm.oom_kill

probe::vm.oom_kill — Fires when a thread is selected for termination by the OOM killer

Synopsis

```
vm.oom_kill
```

Values

name name of the probe point

task the task being killed

Context

The process that tried to consume excessive memory, and thus triggered the OOM.

probe::vm.pagefault

probe::vm.pagefault — Records that a page fault occurred

Synopsis

```
vm.pagefault
```

Values

write_access
: indicates whether this was a write or read access; 1 indicates a write, while 0 indicates a read

name
: name of the probe point

address
: the address of the faulting memory access; i.e. the address that caused the page fault

Context

The process which triggered the fault

probe::vm.pagefault.return

probe::vm.pagefault.return — Indicates what type of fault occurred

Synopsis

```
vm.pagefault.return
```

Values

name
name of the probe point

fault_type
returns either 0 (VM_FAULT_OOM) for out of memory faults, 2 (VM_FAULT_MINOR) for minor faults, 3 (VM_FAULT_MAJOR) for major faults, or 1 (VM_FAULT_SIGBUS) if the fault was neither OOM, minor fault, nor major fault.

probe::vm.write_shared

probe::vm.write_shared — Attempts at writing to a shared page

Synopsis

```
vm.write_shared
```

Values

name	name of the probe point
address	the address of the shared write

Context

The context is the process attempting the write.

Description

Fires when a process attempts to write to a shared page. If a copy is necessary, this will be followed by a vm.write_shared_copy.

probe::vm.write_shared_copy

probe::vm.write_shared_copy — Page copy for shared page write

Synopsis

```
vm.write_shared_copy
```

Values

name Name of the probe point

zero boolean indicating whether it is a zero page (can do a clear instead of a copy)

address The address of the shared write

Context

The process attempting the write.

Description

Fires when a write to a shared page requires a page copy. This is always preceded by a vm.shared_write.

Chapter 7. Task Time Tapset

This tapset defines utility functions to query time related properties of the current tasks, translate those in miliseconds and human readable strings.

function::cputime_to_msecs

function::cputime_to_msecs — Translates the given cputime into milliseconds

Synopsis

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

Arguments

cputime Time to convert to milliseconds.

function::cputime_to_string

function::cputime_to_string — Human readable string for given cputime

Synopsis

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

Arguments

cputime Time to translate.

Description

Equivalent to calling: msec_to_string (cputime_to_msecs (cputime).

function::msecs_to_string

function::msecs_to_string — Human readable string for given milliseconds

Synopsis

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

Arguments

msecs Number of milliseconds to translate.

Description

Returns a string representing the number of milliseconds as a human readable string consisting of "XmY.ZZZs", where X is the number of minutes, Y is the number of seconds and ZZZ is the number of milliseconds.

function::task_stime

function::task_stime — System time of the current task

Synopsis

```
task_stime:long()
```

Arguments

None

Description

Returns the system time of the current task in cputime. Does not include any time used by other tasks in this process, nor does it include any time of the children of this task.

function::task_stime_tid

function::task_stime_tid — System time of the given task

Synopsis

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

Arguments

tid Thread id of the given task

Description

Returns the system time of the given task in cputime, or zero if the task doesn't exist. Does not include any time used by other tasks in this process, nor does it include any time of the children of this task.

function::task_time_string

function::task_time_string — Human readable string of task time usage

Synopsis

```
task_time_string:string()
```

Arguments

None

Description

Returns a human readable string showing the user and system time the current task has used up to now. For example “usr: 0m12.908s, sys: 1m6.851s”.

function::task_time_string_tid

function::task_time_string_tid — Human readable string of task time usage

Synopsis

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

Arguments

tid Thread id of the given task

Description

Returns a human readable string showing the user and system time the given task has used up to now. For example “usr: 0m12.908s, sys: 1m6.851s”.

function::task_utime

function::task_utime — User time of the current task

Synopsis

```
task_utime:long()
```

Arguments

None

Description

Returns the user time of the current task in cputime. Does not include any time used by other tasks in this process, nor does it include any time of the children of this task.

function::task_utime_tid

function::task_utime_tid — User time of the given task

Synopsis

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

Arguments

tid Thread id of the given task

Description

Returns the user time of the given task in cputime, or zero if the task doesn't exist. Does not include any time used by other tasks in this process, nor does it include any time of the children of this task.

Chapter 8. Scheduler Tapset

This family of probe points is used to probe the task scheduler activities. It contains the following probe points:

probe::scheduler.balance

probe::scheduler.balance — A cpu attempting to find more work.

Synopsis

```
scheduler.balance
```

Values

name name of the probe point

Context

The cpu looking for more work.

probe::scheduler.cpu_off

probe::scheduler.cpu_off — Process is about to stop running on a cpu

Synopsis

```
scheduler.cpu_off
```

Values

task_prev	the process leaving the cpu (same as current)
name	name of the probe point
idle	boolean indicating whether current is the idle process
task_next	the process replacing current

Context

The process leaving the cpu.

probe::scheduler.cpu_on

probe::scheduler.cpu_on — Process is beginning execution on a cpu

Synopsis

```
scheduler.cpu_on
```

Values

task_prev	the process that was previously running on this cpu
name	name of the probe point
idle	- boolean indicating whether current is the idle process

Context

The resuming process.

probe::scheduler.ctxswitch

probe::scheduler.ctxswitch — A context switch is occuring.

Synopsis

```
scheduler.ctxswitch
```

Values

prev_pid	The PID of the process to be switched out
name	name of the probe point
next_task_name	The name of the process to be switched in
nexttsk_state	the state of the process to be switched in
prev_priority	The priority of the process to be switched out
next_pid	The PID of the process to be switched in
next_priority	The priority of the process to be switched in
prevtsk_state	the state of the process to be switched out
next_tid	The TID of the process to be switched in
prev_task_name	The name of the process to be switched out
prev_tid	The TID of the process to be switched out

probe::scheduler.kthread_stop

probe::scheduler.kthread_stop — A thread created by kthread_create is being stopped

Synopsis

```
scheduler.kthread_stop
```

Values

thread_priority	priority of the thread
thread_pid	PID of the thread being stopped

probe::scheduler.kthread_stop.return

probe::scheduler.kthread_stop.return — A kthread is stopped and gets the return value

Synopsis

```
scheduler.kthread_stop.return
```

Values

return_value	return value after stopping the thread
name	name of the probe point

probe::scheduler.migrate

probe::scheduler.migrate — Task migrating across cpus

Synopsis

```
scheduler.migrate
```

Values

priority	priority of the task being migrated
cpu_from	the original cpu
name	name of the probe point
task	the process that is being migrated
cpu_to	the destination cpu
pid	PID of the task being migrated

probe::scheduler.process_exit

probe::scheduler.process_exit — Process exiting

Synopsis

```
scheduler.process_exit
```

Values

priority	priority of the process exiting
name	name of the probe point
pid	PID of the process exiting

probe::scheduler.process_fork

probe::scheduler.process_fork — Process forked

Synopsis

```
scheduler.process_fork
```

Values

name	name of the probe point
parent_pid	PID of the parent process
child_pid	PID of the child process

probe::scheduler.process_free

probe::scheduler.process_free — Scheduler freeing a data structure for a process

Synopsis

```
scheduler.process_free
```

Values

priority	priority of the process getting freed
name	name of the probe point
pid	PID of the process getting freed

probe::scheduler.process_wait

probe::scheduler.process_wait — Scheduler starting to wait on a process

Synopsis

```
scheduler.process_wait
```

Values

name
: name of the probe point

pid
: PID of the process scheduler is waiting on

probe::scheduler.signal_send

probe::scheduler.signal_send — Sending a signal

Synopsis

```
scheduler.signal_send
```

Values

signal_number	signal number
name	name of the probe point
pid	pid of the process sending signal

probe::scheduler.tick

probe::scheduler.tick — Schedulers internal tick, a processes timeslice accounting is updated

Synopsis

```
scheduler.tick
```

Values

name name of the probe point

idle boolean indicating whether current is the idle process

Context

The process whose accounting will be updated.

probe::scheduler.wait_task

probe::scheduler.wait_task — Waiting on a task to unschedule (become inactive)

Synopsis

```
scheduler.wait_task
```

Values

name	name of the probe point
task_pid	PID of the task the scheduler is waiting on
task_priority	priority of the task

probe::scheduler.wakeup

probe::scheduler.wakeup — Task is woken up

Synopsis

```
scheduler.wakeup
```

Values

task_cpu	cpu of the task being woken up
name	name of the probe point
task_pid	PID of the task being woken up
task_priority	priority of the task being woken up
task_state	state of the task being woken up
task_tid	tid of the task being woken up

probe::scheduler.wakeup_new

probe::scheduler.wakeup_new — Newly created task is woken up for the first time

Synopsis

```
scheduler.wakeup_new
```

Values

task_cpu	cpu of the task woken up
name	name of the probe point
task_pid	PID of the new task woken up
task_priority	priority of the new task
task_state	state of the task woken up
task_tid	TID of the new task woken up

Chapter 9. IO Scheduler and block IO Tapset

This family of probe points is used to probe block IO layer and IO scheduler activities. It contains the following probe points:

probe::ioblock.end

probe::ioblock.end — Fires whenever a block I/O transfer is complete.

Synopsis

```
ioblock.end
```

Values

None

Description

name - name of the probe point *devname* - block device name *ino* - i-node number of the mapped file *bytes_done* - number of bytes transferred *sector* - beginning sector for the entire bio *flags* - see below BIO_UPTODATE 0 ok after I/O completion BIO_RW_BLOCK 1 RW_AHEAD set, and read/write would block BIO_EOF 2 out-out-bounds error BIO_SEG_VALID 3 nr_hw_seg valid BIO_CLONED 4 doesn't own data BIO_BOUNCED 5 bio is a bounce bio BIO_USER_MAPPED 6 contains user pages BIO_EOPNOTSUPP 7 not supported *error* - 0 on success *rw* - binary trace for read/write request *vcnt* - bio vector count which represents number of array element (page, offset, length) which makes up this I/O request *idx* - offset into the bio vector array *phys_segments* - number of segments in this bio after physical address coalescing is performed. *hw_segments* - number of segments after physical and DMA remapping hardware coalescing is performed *size* - total size in bytes

Context

The process signals the transfer is done.

probe::ioblock.request

probe::ioblock.request — Fires whenever making a generic block I/O request.

Synopsis

```
ioblock.request
```

Values

None

Description

name - name of the probe point *devname* - block device name *ino* - i-node number of the mapped file *sector* - beginning sector for the entire bio *flags* - see below BIO_UPTODATE 0 ok after I/O completion BIO_RW_BLOCK 1 RW_AHEAD set, and read/write would block BIO_EOF 2 out-out-bounds error BIO_SEG_VALID 3 nr_hw_seg valid BIO_CLONED 4 doesn't own data BIO_BOUNCED 5 bio is a bounce bio BIO_USER_MAPPED 6 contains user pages BIO_EOPNOTSUPP 7 not supported

rw - binary trace for read/write request *vcnt* - bio vector count which represents number of array element (page, offset, length) which make up this I/O request *idx* - offset into the bio vector array *phys_segments* - number of segments in this bio after physical address coalescing is performed *hw_segments* - number of segments after physical and DMA remapping hardware coalescing is performed *size* - total size in bytes *bdev* - target block device *bdev_contains* - points to the device object which contains the partition (when bio structure represents a partition) *p_start_sect* - points to the start sector of the partition structure of the device

Context

The process makes block I/O request

probe::ioblock_trace.bounce

probe::ioblock_trace.bounce — Fires whenever a buffer bounce is needed for at least one page of a block IO request.

Synopsis

```
ioblock_trace.bounce
```

Values

None

Description

name - name of the probe point *q* - request queue on which this bio was queued. *devname* - device for which a buffer bounce was needed. *ino* - i-node number of the mapped file *bytes_done* - number of bytes transferred *sector* - beginning sector for the entire bio *flags* - see below BIO_UPTODATE 0 ok after I/O completion BIO_RW_BLOCK 1 RW_AHEAD set, and read/write would block BIO_EOF 2 out-out-bounds error BIO_SEG_VALID 3 nr_hw_seg valid BIO_CLONED 4 doesn't own data BIO_BOUNCED 5 bio is a bounce bio BIO_USER_MAPPED 6 contains user pages BIO_EOPNOTSUPP 7 not supported *rw* - binary trace for read/write request *vcnt* - bio vector count which represents number of array element (page, offset, length) which makes up this I/O request *idx* - offset into the bio vector array *phys_segments* - number of segments in this bio after physical address coalescing is performed. *size* - total size in bytes *bdev* - target block device *bdev_contains* - points to the device object which contains the partition (when bio structure represents a partition) *p_start_sect* - points to the start sector of the partition structure of the device

Context

The process creating a block IO request.

probe::ioblock_trace.end

probe::ioblock_trace.end — Fires whenever a block I/O transfer is complete.

Synopsis

```
ioblock_trace.end
```

Values

None

Description

name - name of the probe point *q* - request queue on which this bio was queued. *devname* - block device name *ino* - i-node number of the mapped file *bytes_done* - number of bytes transferred *sector* - beginning sector for the entire bio *flags* - see below BIO_UPTODATE 0 ok after I/O completion BIO_RW_BLOCK 1 RW_AHEAD set, and read/write would block BIO_EOF 2 out-out-bounds error BIO_SEG_VALID 3 nr_hw_seg valid BIO_CLONED 4 doesn't own data BIO_BOUNCED 5 bio is a bounce bio BIO_USER_MAPPED 6 contains user pages BIO_EOPNOTSUPP 7 not supported

rw - binary trace for read/write request *vcnt* - bio vector count which represents number of array element (page, offset, length) which makes up this I/O request *idx* - offset into the bio vector array *phys_segments* - number of segments in this bio after physical address coalescing is performed. *size* - total size in bytes *bdev* - target block device *bdev_contains* - points to the device object which contains the partition (when bio structure represents a partition) *p_start_sect* - points to the start sector of the partition structure of the device

Context

The process signals the transfer is done.

probe::ioblock_trace.request

probe::ioblock_trace.request — Fires just as a generic block I/O request is created for a bio.

Synopsis

```
ioblock_trace.request
```

Values

None

Description

name - name of the probe point *q* - request queue on which this bio was queued. *devname* - block device name *ino* - i-node number of the mapped file *bytes_done* - number of bytes transferred *sector* - beginning sector for the entire bio *flags* - see below BIO_UPTODATE 0 ok after I/O completion BIO_RW_BLOCK 1 RW_AHEAD set, and read/write would block BIO_EOF 2 out-out-bounds error BIO_SEG_VALID 3 nr_hw_seg valid BIO_CLONED 4 doesn't own data BIO_BOUNCED 5 bio is a bounce bio BIO_USER_MAPPED 6 contains user pages BIO_EOPNOTSUPP 7 not supported

rw - binary trace for read/write request *vcnt* - bio vector count which represents number of array element (page, offset, length) which make up this I/O request *idx* - offset into the bio vector array *phys_segments* - number of segments in this bio after physical address coalescing is performed. *size* - total size in bytes *bdev* - target block device *bdev_contains* - points to the device object which contains the partition (when bio structure represents a partition) *p_start_sect* - points to the start sector of the partition structure of the device

Context

The process makes block I/O request

probe::ioscheduler.elv_add_request

probe::ioscheduler.elv_add_request — probe to indicate request is added to the request queue.

Synopsis

```
ioscheduler.elv_add_request
```

Values

disk_major	Disk major no of request.
rq	Address of request.
q	Pointer to request queue.
elevator_name	The type of I/O elevator currently enabled.
disk_minor	Disk minor number of request.
rq_flags	Request flags.

probe::ioscheduler.elv_add_request.kp

probe::ioscheduler.elv_add_request.kp — kprobe based probe to indicate that a request was added to the request queue

Synopsis

```
ioscheduler.elv_add_request.kp
```

Values

disk_major	Disk major number of the request
rq	Address of the request
q	pointer to request queue
name	Name of the probe point
elevator_name	The type of I/O elevator currently enabled
disk_minor	Disk minor number of the request
rq_flags	Request flags

probe::ioscheduler.elv_add_request.tp

probe::ioscheduler.elv_add_request.tp — tracepoint based probe to indicate a request is added to the request queue.

Synopsis

```
ioscheduler.elv_add_request.tp
```

Values

disk_major	Disk major no of request.
rq	Address of request.
q	Pointer to request queue.
name	Name of the probe point
elevator_name	The type of I/O elevator currently enabled.
disk_minor	Disk minor number of request.
rq_flags	Request flags.

probe::ioscheduler.elv_completed_request

probe::ioscheduler.elv_completed_request — Fires when a request is completed

Synopsis

```
ioscheduler.elv_completed_request
```

Values

disk_major	Disk major number of the request
rq	Address of the request
name	Name of the probe point
elevator_name	The type of I/O elevator currently enabled
disk_minor	Disk minor number of the request
rq_flags	Request flags

probe::ioscheduler.elv_next_request

probe::ioscheduler.elv_next_request — Fires when a request is retrieved from the request queue

Synopsis

```
ioscheduler.elv_next_request
```

Values

name	Name of the probe point
elevator_name	The type of I/O elevator currently enabled

probe::ioscheduler.elv_next_request.return

probe::ioscheduler.elv_next_request.return — Fires when a request retrieval issues a return signal

Synopsis

```
ioscheduler.elv_next_request.return
```

Values

disk_major	Disk major number of the request
rq	Address of the request
name	Name of the probe point
disk_minor	Disk minor number of the request
rq_flags	Request flags

probe::ioscheduler_trace.elv_abort_request

probe::ioscheduler_trace.elv_abort_request — Fires when a request is aborted.

Synopsis

```
ioscheduler_trace.elv_abort_request
```

Values

disk_major	Disk major no of request.
rq	Address of request.
name	Name of the probe point
elevator_name	The type of I/O elevator currently enabled.
disk_minor	Disk minor number of request.
rq_flags	Request flags.

probe::ioscheduler_trace.elv_completed_request

probe::ioscheduler_trace.elv_completed_request — Fires when a request is

Synopsis

```
ioscheduler_trace.elv_completed_request
```

Values

disk_major
: Disk major no of request.

rq
: Address of request.

name
: Name of the probe point

elevator_name
: The type of I/O elevator currently enabled.

disk_minor
: Disk minor number of request.

rq_flags
: Request flags.

Description

completed.

probe::ioscheduler_trace.elv_issue_request

probe::ioscheduler_trace.elv_issue_request — Fires when a request is

Synopsis

```
ioscheduler_trace.elv_issue_request
```

Values

disk_major	Disk major no of request.
rq	Address of request.
name	Name of the probe point
elevator_name	The type of I/O elevator currently enabled.
disk_minor	Disk minor number of request.
rq_flags	Request flags.

Description

scheduled.

probe::ioscheduler_trace.elv_requeue_request

probe::ioscheduler_trace.elv_requeue_request — Fires when a request is

Synopsis

```
ioscheduler_trace.elv_requeue_request
```

Values

disk_major	Disk major no of request.
rq	Address of request.
name	Name of the probe point
elevator_name	The type of I/O elevator currently enabled.
disk_minor	Disk minor number of request.
rq_flags	Request flags.

Description

put back on the queue, when the hadware cannot accept more requests.

probe::ioscheduler_trace.plug

probe::ioscheduler_trace.plug — Fires when a request queue is plugged;

Synopsis

```
ioscheduler_trace.plug
```

Values

name	Name of the probe point
rq_queue	request queue

Description

ie, requests in the queue cannot be serviced by block driver.

probe::ioscheduler_trace.unplug_io

probe::ioscheduler_trace.unplug_io — Fires when a request queue is unplugged;

Synopsis

```
ioscheduler_trace.unplug_io
```

Values

name	Name of the probe point
rq_queue	request queue

Description

Either, when number of pending requests in the queue exceeds threshold or, upon expiration of timer that was activated when queue was plugged.

probe::ioscheduler_trace.unplug_timer

probe::ioscheduler_trace.unplug_timer — Fires when unplug timer associated

Synopsis

```
ioscheduler_trace.unplug_timer
```

Values

name	Name of the probe point
rq_queue	request queue

Description

with a request queue expires.

Chapter 10. SCSI Tapset

This family of probe points is used to probe SCSI activities. It contains the following probe points:

probe::scsi.iocompleted

probe::scsi.iocompleted — SCSI mid-layer running the completion processing for block device I/O requests

Synopsis

```
scsi.iocompleted
```

Values

device_state_str	The current state of the device, as a string
dev_id	The scsi device id
channel	The channel number
data_direction	The data_direction specifies whether this command is from/to the device
lun	The lun number
host_no	The host number
data_direction_str	Data direction, as a string
device_state	The current state of the device
req_addr	The current struct request pointer, as a number
goodbytes	The bytes completed

probe::scsi.iodispatching

probe::scsi.iodispatching — SCSI mid-layer dispatched low-level SCSI command

Synopsis

```
scsi.iodispatching
```

Values

device_state_str
: The current state of the device, as a string

dev_id
: The scsi device id

channel
: The channel number

data_direction
: The data_direction specifies whether this command is from/to the device 0 (DMA_BIDIRECTIONAL), 1 (DMA_TO_DEVICE), 2 (DMA_FROM_DEVICE), 3 (DMA_NONE)

lun
: The lun number

request_bufflen
: The request buffer length

host_no
: The host number

device_state
: The current state of the device

data_direction_str
: Data direction, as a string

req_addr
: The current struct request pointer, as a number

request_buffer
: The request buffer address

probe::scsi.iodone

probe::scsi.iodone — SCSI command completed by low level driver and enqueued into the done queue.

Synopsis

```
scsi.iodone
```

Values

device_state_str The current state of the device, as a string

dev_id The scsi device id

channel The channel number

data_direction The data_direction specifies whether this command is from/to the device.

lun The lun number

host_no The host number

data_direction_str Data direction, as a string

device_state The current state of the device

scsi_timer_pending 1 if a timer is pending on this request

req_addr The current struct request pointer, as a number

probe::scsi.ioentry

probe::scsi.ioentry — Prepares a SCSI mid-layer request

Synopsis

```
scsi.ioentry
```

Values

disk_major	The major number of the disk (-1 if no information)
device_state_str	The current state of the device, as a string
device_state	The current state of the device
req_addr	The current struct request pointer, as a number
disk_minor	The minor number of the disk (-1 if no information)

probe::scsi.ioexecute

probe::scsi.ioexecute — Create mid-layer SCSI request and wait for the result

Synopsis

```
scsi.ioexecute
```

Values

retries	Number of times to retry request
device_state_str	The current state of the device, as a string
dev_id	The scsi device id
channel	The channel number
data_direction	The data_direction specifies whether this command is from/to the device.
lun	The lun number
timeout	Request timeout in seconds
request_bufflen	The data buffer buffer length
host_no	The host number
data_direction_str	Data direction, as a string
device_state	The current state of the device
request_buffer	The data buffer address

probe::scsi.set_state

probe::scsi.set_state — Order SCSI device state change

Synopsis

```
scsi.set_state
```

Values

state_str	The new state of the device, as a string
dev_id	The scsi device id
channel	The channel number
state	The new state of the device
old_state_str	The current state of the device, as a string
lun	The lun number
old_state	The current state of the device
host_no	The host number

Chapter 11. TTY Tapset

This family of probe points is used to probe TTY (Teletype) activities. It contains the following probe points:

probe::tty.init

probe::tty.init — Called when a tty is being initalized

Synopsis

```
tty.init
```

Values

driver_name	the driver name
name	the driver .dev_name name
module	the module name

probe::tty.ioctl

probe::tty.ioctl — called when a ioctl is request to the tty

Synopsis

```
tty.ioctl
```

Values

cmd	the ioctl command
arg	the ioctl argument
name	the file name

probe::tty.open

probe::tty.open — Called when a tty is opened

Synopsis

```
tty.open
```

Values

inode_state	the inode state
file_name	the file name
file_mode	the file mode
file_flags	the file flags
inode_number	the inode number
inode_flags	the inode flags

probe::tty.poll

probe::tty.poll — Called when a tty device is being polled

Synopsis

```
tty.poll
```

Values

file_name	the tty file name
wait_key	the wait queue key

probe::tty.read

probe::tty.read — called when a tty line will be read

Synopsis

```
tty.read
```

Values

driver_name	the driver name
buffer	the buffer that will receive the characters
file_name	the file name lreated to the tty
nr	The amount of characters to be read

probe::tty.receive

probe::tty.receive — called when a tty receives a message

Synopsis

```
tty.receive
```

Values

driver_name	the driver name
count	The amount of characters received
name	the name of the module file
fp	The flag buffer
cp	the buffer that was received
index	The tty Index
id	the tty id

probe::tty.register

probe::tty.register — Called when a tty device is registred

Synopsis

```
tty.register
```

Values

driver_name	the driver name
name	the driver .dev_name name
index	the tty index requested
module	the module name

probe::tty.release

probe::tty.release — Called when the tty is closed

Synopsis

```
tty.release
```

Values

inode_state	the inode state
file_name	the file name
file_mode	the file mode
file_flags	the file flags
inode_number	the inode number
inode_flags	the inode flags

probe::tty.resize

probe::tty.resize — Called when a terminal resize happens

Synopsis

```
tty.resize
```

Values

new_ypixel	the new ypixel value
old_col	the old col value
old_xpixel	the old xpixel
old_ypixel	the old ypixel
name	the tty name
old_row	the old row value
new_row	the new row value
new_xpixel	the new xpixel value
new_col	the new col value

probe::tty.unregister

probe::tty.unregister — Called when a tty device is being unregistered

Synopsis

```
tty.unregister
```

Values

driver_name	the driver name
name	the driver .dev_name name
index	the tty index requested
module	the module name

probe::tty.write

probe::tty.write — write to the tty line

Synopsis

```
tty.write
```

Values

driver_name	the driver name
buffer	the buffer that will be written
file_name	the file name lreated to the tty
nr	The amount of characters

Chapter 12. Interrupt Request (IRQ) Tapset

This family of probe points is used to probe interrupt request (IRQ) activities. It contains the following probe points:

probe::irq_handler.entry

probe::irq_handler.entry — Execution of interrupt handler starting

Synopsis

```
irq_handler.entry
```

Values

dev_name	name of device
flags	Flags for IRQ handler
dev_id	Cookie to identify device
dir	pointer to the proc/irq/NN/name entry
irq	irq number
next_irqaction	pointer to next irqaction for shared interrupts
thread_flags	Flags related to thread
thread	thread pointer for threaded interrupts
thread_fn	interrupt handler function for threaded interrupts
handler	interrupt handler function
flags_str	symbolic string representation of IRQ flags
action	struct irqaction* for this interrupt num

probe::irq_handler.exit

probe::irq_handler.exit — Execution of interrupt handler completed

Synopsis

```
irq_handler.exit
```

Values

dev_name	name of device
ret	return value of the handler
flags	flags for IRQ handler
dev_id	Cookie to identify device
dir	pointer to the proc/irq/NN/name entry
next_irqaction	pointer to next irqaction for shared interrupts
irq	interrupt number
thread_flags	Flags related to thread
thread	thread pointer for threaded interrupts
thread_fn	interrupt handler function for threaded interrupts
flags_str	symbolic string representation of IRQ flags
handler	interrupt handler function that was executed
action	struct irqaction*

probe::softirq.entry

probe::softirq.entry — Execution of handler for a pending softirq starting

Synopsis

```
softirq.entry
```

Values

vec	softirq_action vector
h	struct softirq_action* for current pending softirq
vec_nr	softirq vector number
action	pointer to softirq handler just about to execute

probe::softirq.exit

probe::softirq.exit — Execution of handler for a pending softirq completed

Synopsis

```
softirq.exit
```

Values

vec	softirq_action vector
h	struct softirq_action* for just executed softirq
vec_nr	softirq vector number
action	pointer to softirq handler that just finished execution

probe::workqueue.create

probe::workqueue.create — Creating a new workqueue

Synopsis

```
workqueue.create
```

Values

wq_thread	task_struct of the workqueue thread
cpu	cpu for which the worker thread is created

probe::workqueue.destroy

probe::workqueue.destroy — Destroying workqueue

Synopsis

```
workqueue.destroy
```

Values

wq_thread task_struct of the workqueue thread

probe::workqueue.execute

probe::workqueue.execute — Executing deferred work

Synopsis

```
workqueue.execute
```

Values

wq_thread	task_struct of the workqueue thread
work_func	pointer to handler function
work	work_struct* being executed

probe::workqueue.insert

probe::workqueue.insert — Queuing work on a workqueue

Synopsis

```
workqueue.insert
```

Values

wq_thread	task_struct of the workqueue thread
work_func	pointer to handler function
work	work_struct* being queued

Chapter 13. Networking Tapset

This family of probe points is used to probe the activities of the network device and protocol layers.

function::format_ipaddr

function::format_ipaddr — Returns a string representation for an IP address

Synopsis

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

Arguments

addr
: the IP address

family
: the IP address family (either AF_INET or AF_INET6)

function::htonl

function::htonl — Convert 32-bit long from host to network order

Synopsis

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

Arguments

x Value to convert

function::htonll

function::htonll — Convert 64-bit long long from host to network order

Synopsis

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

Arguments

x Value to convert

function::htons

function::htons — Convert 16-bit short from host to network order

Synopsis

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

Arguments

x Value to convert

function::ip_ntop

function::ip_ntop — Returns a string representation for an IPv4 address

Synopsis

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

Arguments

addr the IPv4 address represented as an integer

function::ntohl

function::ntohl — Convert 32-bit long from network to host order

Synopsis

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

Arguments

x Value to convert

function::ntohll

function::ntohll — Convert 64-bit long long from network to host order

Synopsis

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

Arguments

x Value to convert

function::ntohs

function::ntohs — Convert 16-bit short from network to host order

Synopsis

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

Arguments

x Value to convert

probe::netdev.change_mac

probe::netdev.change_mac — Called when the netdev_name has the MAC changed

Synopsis

```
netdev.change_mac
```

Values

dev_name	The device that will have the MAC changed
new_mac	The new MAC address
mac_len	The MAC length
old_mac	The current MAC address

probe::netdev.change_mtu

probe::netdev.change_mtu — Called when the netdev MTU is changed

Synopsis

```
netdev.change_mtu
```

Values

dev_name The device that will have the MTU changed

new_mtu The new MTU

old_mtu The current MTU

probe::netdev.change_rx_flag

probe::netdev.change_rx_flag — Called when the device RX flag will be changed

Synopsis

```
netdev.change_rx_flag
```

Values

dev_name The device that will be changed

flags The new flags

probe::netdev.close

probe::netdev.close — Called when the device is closed

Synopsis

```
netdev.close
```

Values

dev_name	The device that is going to be closed

probe::netdev.get_stats

probe::netdev.get_stats — Called when someone asks the device statistics

Synopsis

```
netdev.get_stats
```

Values

dev_name The device that is going to provide the statistics

probe::netdev.hard_transmit

probe::netdev.hard_transmit — Called when the devices is going to TX (hard)

Synopsis

```
netdev.hard_transmit
```

Values

protocol	The protocol used in the transmission
dev_name	The device scheduled to transmit
length	The length of the transmit buffer.
truesize	The size of the data to be transmitted.

probe::netdev.ioctl

probe::netdev.ioctl — Called when the device suffers an IOCTL

Synopsis

```
netdev.ioctl
```

Values

cmd The IOCTL request

arg The IOCTL argument (usually the netdev interface)

probe::netdev.open

probe::netdev.open — Called when the device is opened

Synopsis

```
netdev.open
```

Values

dev_name The device that is going to be opened

probe::netdev.receive

probe::netdev.receive — Data received from network device.

Synopsis

```
netdev.receive
```

Values

protocol	Protocol of received packet.
dev_name	The name of the device. e.g: eth0, ath1.
length	The length of the receiving buffer.

probe::netdev.register

probe::netdev.register — Called when the device is registered

Synopsis

```
netdev.register
```

Values

dev_name
: The device that is going to be registered

probe::netdev.rx

probe::netdev.rx — Called when the device is going to receive a packet

Synopsis

```
netdev.rx
```

Values

protocol	The packet protocol
dev_name	The device received the packet

probe::netdev.set_promiscuity

probe::netdev.set_promiscuity — Called when the device enters/leaves promiscuity

Synopsis

```
netdev.set_promiscuity
```

Values

dev_name	The device that is entering/leaving promiscuity mode
enable	If the device is entering promiscuity mode
inc	Count the number of promiscuity openers
disable	If the device is leaving promiscuity mode

probe::netdev.transmit

probe::netdev.transmit — Network device transmitting buffer

Synopsis

```
netdev.transmit
```

Values

protocol The protocol of this packet(defined in include/linux/if_ether.h).

dev_name The name of the device. e.g: eth0, ath1.

length The length of the transmit buffer.

truesize The size of the data to be transmitted.

probe::netdev.unregister

probe::netdev.unregister — Called when the device is being unregistered

Synopsis

```
netdev.unregister
```

Values

dev_name The device that is going to be unregistered

probe::netfilter.arp.forward

probe::netfilter.arp.forward — - Called for each ARP packet to be forwarded

Synopsis

```
netfilter.arp.forward
```

Values

indev	Address of net_device representing input device, 0 if unknown
nf_accept	Constant used to signify an ’accept’ verdict
ar_sha	Ethernet+IP only (ar_pro==0x800): source hardware (MAC) address
pf	Protocol family -- always “arp”
ar_sip	Ethernet+IP only (ar_pro==0x800): source IP address
nf_queue	Constant used to signify a ’queue’ verdict
ar_op	ARP opcode (command)
nf_stolen	Constant used to signify a ’stolen’ verdict
outdev_name	Name of network device packet will be routed to (if known)
nf_drop	Constant used to signify a ’drop’ verdict
ar_hln	Length of hardware address
ar_pro	Format of protocol address
ar_pln	Length of protocol address
ar_tip	Ethernet+IP only (ar_pro==0x800): target IP address
nf_stop	Constant used to signify a ’stop’ verdict
arphdr	Address of ARP header
length	The length of the packet buffer contents, in bytes
outdev	Address of net_device representing output device, 0 if unknown
ar_tha	Ethernet+IP only (ar_pro==0x800): target hardware (MAC) address
ar_data	Address of ARP packet data region (after the header)
nf_repeat	Constant used to signify a ’repeat’ verdict
indev_name	Name of network device packet was received on (if known)
ar_hrd	Format of hardware address

probe::netfilter.arp.in

probe::netfilter.arp.in — - Called for each incoming ARP packet

Synopsis

```
netfilter.arp.in
```

Values

indev	Address of net_device representing input device, 0 if unknown
nf_accept	Constant used to signify an 'accept' verdict
ar_sha	Ethernet+IP only (ar_pro==0x800): source hardware (MAC) address
pf	Protocol family -- always "arp"
ar_sip	Ethernet+IP only (ar_pro==0x800): source IP address
nf_queue	Constant used to signify a 'queue' verdict
ar_op	ARP opcode (command)
nf_stolen	Constant used to signify a 'stolen' verdict
outdev_name	Name of network device packet will be routed to (if known)
nf_drop	Constant used to signify a 'drop' verdict
ar_hln	Length of hardware address
ar_pro	Format of protocol address
ar_pln	Length of protocol address
ar_tip	Ethernet+IP only (ar_pro==0x800): target IP address
nf_stop	Constant used to signify a 'stop' verdict
arphdr	Address of ARP header
length	The length of the packet buffer contents, in bytes
outdev	Address of net_device representing output device, 0 if unknown
ar_tha	Ethernet+IP only (ar_pro==0x800): target hardware (MAC) address
ar_data	Address of ARP packet data region (after the header)
nf_repeat	Constant used to signify a 'repeat' verdict
indev_name	Name of network device packet was received on (if known)
ar_hrd	Format of hardware address

probe::netfilter.arp.out

probe::netfilter.arp.out — - Called for each outgoing ARP packet

Synopsis

```
netfilter.arp.out
```

Values

indev	Address of net_device representing input device, 0 if unknown
nf_accept	Constant used to signify an 'accept' verdict
ar_sha	Ethernet+IP only (ar_pro==0x800): source hardware (MAC) address
pf	Protocol family -- always "arp"
ar_sip	Ethernet+IP only (ar_pro==0x800): source IP address
nf_queue	Constant used to signify a 'queue' verdict
ar_op	ARP opcode (command)
nf_stolen	Constant used to signify a 'stolen' verdict
outdev_name	Name of network device packet will be routed to (if known)
nf_drop	Constant used to signify a 'drop' verdict
ar_hln	Length of hardware address
ar_pro	Format of protocol address
ar_pln	Length of protocol address
ar_tip	Ethernet+IP only (ar_pro==0x800): target IP address
nf_stop	Constant used to signify a 'stop' verdict
arphdr	Address of ARP header
length	The length of the packet buffer contents, in bytes
outdev	Address of net_device representing output device, 0 if unknown
ar_tha	Ethernet+IP only (ar_pro==0x800): target hardware (MAC) address
ar_data	Address of ARP packet data region (after the header)
nf_repeat	Constant used to signify a 'repeat' verdict
indev_name	Name of network device packet was received on (if known)
ar_hrd	Format of hardware address

probe::netfilter.bridge.forward

probe::netfilter.bridge.forward — Called on an incoming bridging packet destined for some other computer

Synopsis

```
netfilter.bridge.forward
```

Values

indev	Address of net_device representing input device, 0 if unknown
nf_accept	Constant used to signify an 'accept' verdict
pf	Protocol family -- always "bridge"
nf_queue	Constant used to signify a 'queue' verdict
nf_stolen	Constant used to signify a 'stolen' verdict
outdev_name	Name of network device packet will be routed to (if known)
nf_drop	Constant used to signify a 'drop' verdict
nf_stop	Constant used to signify a 'stop' verdict
length	The length of the packet buffer contents, in bytes
outdev	Address of net_device representing output device, 0 if unknown
nf_repeat	Constant used to signify a 'repeat' verdict
indev_name	Name of network device packet was received on (if known)

probe::netfilter.bridge.local_in

probe::netfilter.bridge.local_in — Called on a bridging packet destined for the local computer

Synopsis

```
netfilter.bridge.local_in
```

Values

indev	Address of net_device representing input device, 0 if unknown
nf_accept	Constant used to signify an 'accept' verdict
pf	Protocol family -- always "bridge"
nf_queue	Constant used to signify a 'queue' verdict
nf_stolen	Constant used to signify a 'stolen' verdict
outdev_name	Name of network device packet will be routed to (if known)
nf_drop	Constant used to signify a 'drop' verdict
nf_stop	Constant used to signify a 'stop' verdict
length	The length of the packet buffer contents, in bytes
outdev	Address of net_device representing output device, 0 if unknown
nf_repeat	Constant used to signify a 'repeat' verdict
indev_name	Name of network device packet was received on (if known)

probe::netfilter.bridge.local_out

probe::netfilter.bridge.local_out — Called on a bridging packet coming from a local process

Synopsis

```
netfilter.bridge.local_out
```

Values

indev	Address of net_device representing input device, 0 if unknown
nf_accept	Constant used to signify an 'accept' verdict
pf	Protocol family -- always "bridge"
nf_queue	Constant used to signify a 'queue' verdict
nf_stolen	Constant used to signify a 'stolen' verdict
outdev_name	Name of network device packet will be routed to (if known)
nf_drop	Constant used to signify a 'drop' verdict
nf_stop	Constant used to signify a 'stop' verdict
length	The length of the packet buffer contents, in bytes
outdev	Address of net_device representing output device, 0 if unknown
nf_repeat	Constant used to signify a 'repeat' verdict
indev_name	Name of network device packet was received on (if known)

probe::netfilter.bridge.post_routing

probe::netfilter.bridge.post_routing — - Called before a bridging packet hits the wire

Synopsis

```
netfilter.bridge.post_routing
```

Values

indev	Address of net_device representing input device, 0 if unknown
nf_accept	Constant used to signify an 'accept' verdict
pf	Protocol family -- always “bridge”
nf_queue	Constant used to signify a 'queue' verdict
nf_stolen	Constant used to signify a 'stolen' verdict
outdev_name	Name of network device packet will be routed to (if known)
nf_drop	Constant used to signify a 'drop' verdict
nf_stop	Constant used to signify a 'stop' verdict
length	The length of the packet buffer contents, in bytes
outdev	Address of net_device representing output device, 0 if unknown
nf_repeat	Constant used to signify a 'repeat' verdict
indev_name	Name of network device packet was received on (if known)

probe::netfilter.bridge.pre_routing

probe::netfilter.bridge.pre_routing — - Called before a bridging packet is routed

Synopsis

```
netfilter.bridge.pre_routing
```

Values

indev	Address of net_device representing input device, 0 if unknown
nf_accept	Constant used to signify an 'accept' verdict
pf	Protocol family -- always “bridge”
nf_queue	Constant used to signify a 'queue' verdict
nf_stolen	Constant used to signify a 'stolen' verdict
outdev_name	Name of network device packet will be routed to (if known)
nf_drop	Constant used to signify a 'drop' verdict
nf_stop	Constant used to signify a 'stop' verdict
length	The length of the packet buffer contents, in bytes
outdev	Address of net_device representing output device, 0 if unknown
nf_repeat	Constant used to signify a 'repeat' verdict
indev_name	Name of network device packet was received on (if known)

probe::netfilter.ip.forward

probe::netfilter.ip.forward — Called on an incoming IP packet addressed to some other computer

Synopsis

```
netfilter.ip.forward
```

Values

urg	TCP URG flag (if protocol is TCP; ipv4 only)
protocol	Packet protocol from driver (ipv4 only)
indev	Address of net_device representing input device, 0 if unknown
nf_accept	Constant used to signify an ’accept’ verdict
pf	Protocol family -- either “ipv4” or “ipv6”
ipproto_tcp	Constant used to signify that the packet protocol is TCP
rst	TCP RST flag (if protocol is TCP; ipv4 only)
dport	TCP or UDP destination port (ipv4 only)
nf_queue	Constant used to signify a ’queue’ verdict
nf_stolen	Constant used to signify a ’stolen’ verdict
ipproto_udp	Constant used to signify that the packet protocol is UDP
outdev_name	Name of network device packet will be routed to (if known)
ack	TCP ACK flag (if protocol is TCP; ipv4 only)
fin	TCP FIN flag (if protocol is TCP; ipv4 only)
nf_drop	Constant used to signify a ’drop’ verdict
psh	TCP PSH flag (if protocol is TCP; ipv4 only)
saddr	A string representing the source IP address
nf_stop	Constant used to signify a ’stop’ verdict
length	The length of the packet buffer contents, in bytes
daddr	A string representing the destination IP address
outdev	Address of net_device representing output device, 0 if unknown
syn	TCP SYN flag (if protocol is TCP; ipv4 only)
sport	TCP or UDP source port (ipv4 only)
nf_repeat	Constant used to signify a ’repeat’ verdict
indev_name	Name of network device packet was received on (if known)
family	IP address family
iphdr	Address of IP header

probe::netfilter.ip.local_in

probe::netfilter.ip.local_in — Called on an incoming IP packet addressed to the local computer

Synopsis

```
netfilter.ip.local_in
```

Values

urg	TCP URG flag (if protocol is TCP; ipv4 only)
protocol	Packet protocol from driver (ipv4 only)
indev	Address of net_device representing input device, 0 if unknown
nf_accept	Constant used to signify an ’accept’ verdict
pf	Protocol family -- either “ipv4” or “ipv6”
ipproto_tcp	Constant used to signify that the packet protocol is TCP
rst	TCP RST flag (if protocol is TCP; ipv4 only)
dport	TCP or UDP destination port (ipv4 only)
nf_queue	Constant used to signify a ’queue’ verdict
nf_stolen	Constant used to signify a ’stolen’ verdict
ipproto_udp	Constant used to signify that the packet protocol is UDP
outdev_name	Name of network device packet will be routed to (if known)
ack	TCP ACK flag (if protocol is TCP; ipv4 only)
fin	TCP FIN flag (if protocol is TCP; ipv4 only)
nf_drop	Constant used to signify a ’drop’ verdict
psh	TCP PSH flag (if protocol is TCP; ipv4 only)
saddr	A string representing the source IP address
nf_stop	Constant used to signify a ’stop’ verdict
length	The length of the packet buffer contents, in bytes
daddr	A string representing the destination IP address
outdev	Address of net_device representing output device, 0 if unknown
syn	TCP SYN flag (if protocol is TCP; ipv4 only)
sport	TCP or UDP source port (ipv4 only)
nf_repeat	Constant used to signify a ’repeat’ verdict
indev_name	Name of network device packet was received on (if known)
family	IP address family
iphdr	Address of IP header

probe::netfilter.ip.local_out

probe::netfilter.ip.local_out — Called on an outgoing IP packet

Synopsis

```
netfilter.ip.local_out
```

Values

urg	TCP URG flag (if protocol is TCP; ipv4 only)
protocol	Packet protocol from driver (ipv4 only)
indev	Address of net_device representing input device, 0 if unknown
nf_accept	Constant used to signify an 'accept' verdict
pf	Protocol family -- either “ipv4” or “ipv6”
ipproto_tcp	Constant used to signify that the packet protocol is TCP
rst	TCP RST flag (if protocol is TCP; ipv4 only)
dport	TCP or UDP destination port (ipv4 only)
nf_queue	Constant used to signify a 'queue' verdict
nf_stolen	Constant used to signify a 'stolen' verdict
ipproto_udp	Constant used to signify that the packet protocol is UDP
outdev_name	Name of network device packet will be routed to (if known)
ack	TCP ACK flag (if protocol is TCP; ipv4 only)
fin	TCP FIN flag (if protocol is TCP; ipv4 only)
nf_drop	Constant used to signify a 'drop' verdict
psh	TCP PSH flag (if protocol is TCP; ipv4 only)
saddr	A string representing the source IP address
nf_stop	Constant used to signify a 'stop' verdict
length	The length of the packet buffer contents, in bytes
daddr	A string representing the destination IP address
outdev	Address of net_device representing output device, 0 if unknown
syn	TCP SYN flag (if protocol is TCP; ipv4 only)
sport	TCP or UDP source port (ipv4 only)
nf_repeat	Constant used to signify a 'repeat' verdict
indev_name	Name of network device packet was received on (if known)
family	IP address family
iphdr	Address of IP header

probe::netfilter.ip.post_routing

probe::netfilter.ip.post_routing — Called immediately before an outgoing IP packet leaves the computer

Synopsis

```
netfilter.ip.post_routing
```

Values

urg	TCP URG flag (if protocol is TCP; ipv4 only)
protocol	Packet protocol from driver (ipv4 only)
indev	Address of net_device representing input device, 0 if unknown
nf_accept	Constant used to signify an ’accept’ verdict
pf	Protocol family -- either “ipv4” or “ipv6”
ipproto_tcp	Constant used to signify that the packet protocol is TCP
rst	TCP RST flag (if protocol is TCP; ipv4 only)
dport	TCP or UDP destination port (ipv4 only)
nf_queue	Constant used to signify a ’queue’ verdict
nf_stolen	Constant used to signify a ’stolen’ verdict
ipproto_udp	Constant used to signify that the packet protocol is UDP
outdev_name	Name of network device packet will be routed to (if known)
ack	TCP ACK flag (if protocol is TCP; ipv4 only)
fin	TCP FIN flag (if protocol is TCP; ipv4 only)
nf_drop	Constant used to signify a ’drop’ verdict
psh	TCP PSH flag (if protocol is TCP; ipv4 only)
saddr	A string representing the source IP address
nf_stop	Constant used to signify a ’stop’ verdict
length	The length of the packet buffer contents, in bytes
daddr	A string representing the destination IP address
outdev	Address of net_device representing output device, 0 if unknown
syn	TCP SYN flag (if protocol is TCP; ipv4 only)
sport	TCP or UDP source port (ipv4 only)
nf_repeat	Constant used to signify a ’repeat’ verdict

indev_name	Name of network device packet was received on (if known)
family	IP address family
iphdr	Address of IP header

probe::netfilter.ip.pre_routing

probe::netfilter.ip.pre_routing — Called before an IP packet is routed

Synopsis

```
netfilter.ip.pre_routing
```

Values

urg	TCP URG flag (if protocol is TCP; ipv4 only)
protocol	Packet protocol from driver (ipv4 only)
indev	Address of net_device representing input device, 0 if unknown
nf_accept	Constant used to signify an ’accept’ verdict
pf	Protocol family - either ’ipv4’ or ’ipv6’
ipproto_tcp	Constant used to signify that the packet protocol is TCP
rst	TCP RST flag (if protocol is TCP; ipv4 only)
dport	TCP or UDP destination port (ipv4 only)
nf_queue	Constant used to signify a ’queue’ verdict
nf_stolen	Constant used to signify a ’stolen’ verdict
ipproto_udp	Constant used to signify that the packet protocol is UDP
outdev_name	Name of network device packet will be routed to (if known)
ack	TCP ACK flag (if protocol is TCP; ipv4 only)
fin	TCP FIN flag (if protocol is TCP; ipv4 only)
nf_drop	Constant used to signify a ’drop’ verdict
psh	TCP PSH flag (if protocol is TCP; ipv4 only)
saddr	A string representing the source IP address
nf_stop	Constant used to signify a ’stop’ verdict
length	The length of the packet buffer contents, in bytes
daddr	A string representing the destination IP address
outdev	Address of net_device representing output device, 0 if unknown
syn	TCP SYN flag (if protocol is TCP; ipv4 only)
sport	TCP or UDP source port (ipv4 only)
indev_name	Name of network device packet was received on (if known)
nf_repeat	Constant used to signify a ’repeat’ verdict
iphdr	Address of IP header
family	IP address family

probe::sunrpc.clnt.bind_new_program

probe::sunrpc.clnt.bind_new_program — Bind a new RPC program to an existing client

Synopsis

```
sunrpc.clnt.bind_new_program
```

Values

prog	the number of new RPC program
progname	the name of new RPC program
old_vers	the version of old RPC program
old_progname	the name of old RPC program
vers	the version of new RPC program
servername	the server machine name
old_prog	the number of old RPC program

probe::sunrpc.clnt.call_async

probe::sunrpc.clnt.call_async — Make an asynchronous RPC call

Synopsis

```
sunrpc.clnt.call_async
```

Values

prog	the RPC program number
progname	the RPC program name
procname	the procedure name in this RPC call
proc	the procedure number in this RPC call
dead	whether this client is abandoned
flags	flags
vers	the RPC program version number
port	the port number
prot	the IP protocol number
servername	the server machine name
xid	current transmission id

probe::sunrpc.clnt.call_sync

probe::sunrpc.clnt.call_sync — Make a synchronous RPC call

Synopsis

```
sunrpc.clnt.call_sync
```

Values

prog	the RPC program number
progname	the RPC program name
procname	the procedure name in this RPC call
proc	the procedure number in this RPC call
dead	whether this client is abandoned
flags	flags
vers	the RPC program version number
port	the port number
prot	the IP protocol number
servername	the server machine name
xid	current transmission id

probe::sunrpc.clnt.clone_client

probe::sunrpc.clnt.clone_client — Clone an RPC client structure

Synopsis

```
sunrpc.clnt.clone_client
```

Values

servername	the server machine name
vers	the RPC program version number
prog	the RPC program number
authflavor	the authentication flavor
progname	the RPC program name
port	the port number
prot	the IP protocol number

probe::sunrpc.clnt.create_client

probe::sunrpc.clnt.create_client — Create an RPC client

Synopsis

```
sunrpc.clnt.create_client
```

Values

servername	the server machine name
vers	the RPC program version number
prog	the RPC program number
authflavor	the authentication flavor
progname	the RPC program name
port	the port number
prot	the IP protocol number

probe::sunrpc.clnt.restart_call

probe::sunrpc.clnt.restart_call — Restart an asynchronous RPC call

Synopsis

```
sunrpc.clnt.restart_call
```

Values

tk_priority	the task priority
prog	the RPC program number
tk_pid	the debugging aid of task
tk_flags	the task flags
servername	the server machine name
tk_runstate	the task run status
xid	the transmission id

probe::sunrpc.clnt.shutdown_client

probe::sunrpc.clnt.shutdown_client — Shutdown an RPC client

Synopsis

```
sunrpc.clnt.shutdown_client
```

Values

om_ops	the count of operations
om_bytes_sent	the count of bytes out
prog	the RPC program number
authflavor	the authentication flavor
progname	the RPC program name
om_queue	the jiffies queued for xmit
om_rtt	the RPC RTT jiffies
om_bytes_recv	the count of bytes in
tasks	the number of references
netreconn	the count of reconnections
vers	the RPC program version number
om_execute	the RPC execution jiffies
prot	the IP protocol number
port	the port number
clones	the number of clones
servername	the server machine name
rpccnt	the count of RPC calls
om_ntrans	the count of RPC transmissions

probe::sunrpc.sched.delay

probe::sunrpc.sched.delay — Delay an RPC task

Synopsis

```
sunrpc.sched.delay
```

Values

prog	the program number in the RPC call
delay	the time delayed
tk_pid	the debugging id of the task
tk_flags	the flags of the task
vers	the program version in the RPC call
prot	the IP protocol in the RPC call
xid	the transmission id in the RPC call

probe::sunrpc.sched.execute

probe::sunrpc.sched.execute — Execute the RPC ‘scheduler’

Synopsis

```
sunrpc.sched.execute
```

Values

prog	the program number in the RPC call
tk_pid	the debugging id of the task
tk_flags	the flags of the task
vers	the program version in the RPC call
prot	the IP protocol in the RPC call
xid	the transmission id in the RPC call

probe::sunrpc.sched.new_task

probe::sunrpc.sched.new_task — Create new task for the specified client

Synopsis

```
sunrpc.sched.new_task
```

Values

prog	the program number in the RPC call
tk_flags	the flags of the task
vers	the program version in the RPC call
prot	the IP protocol in the RPC call
xid	the transmission id in the RPC call

probe::sunrpc.sched.release_task

probe::sunrpc.sched.release_task — Release all resources associated with a task

Synopsis

```
sunrpc.sched.release_task
```

Values

prog	the program number in the RPC call
tk_flags	the flags of the task
vers	the program version in the RPC call
prot	the IP protocol in the RPC call
xid	the transmission id in the RPC call

Description

`rpc_release_task` function might not be found for a particular kernel. So, if we can't find it, just return '-1' for everything.

probe::sunrpc.svc.create

probe::sunrpc.svc.create — Create an RPC service

Synopsis

```
sunrpc.svc.create
```

Values

prog	the number of the program
progname	the name of the program
pg_nvers	the number of supported versions
bufsize	the buffer size

probe::sunrpc.svc.destroy

probe::sunrpc.svc.destroy — Destroy an RPC service

Synopsis

```
sunrpc.svc.destroy
```

Values

sv_name	the service name
sv_prog	the number of the program
nettcpconn	the count of accepted TCP connections
netcnt	the count of received RPC requests
rpcbadauth	the count of requests drooped for authentication failure
sv_nrthreads	the number of concurrent threads
sv_progname	the name of the program
rpcbadfmt	the count of requests dropped for bad formats
rpccnt	the count of valid RPC requests

probe::sunrpc.svc.drop

probe::sunrpc.svc.drop — Drop RPC request

Synopsis

```
sunrpc.svc.drop
```

Values

rq_prot	the IP protocol of the reqeust
rq_proc	the procedure number in the request
rq_vers	the program version in the request
sv_name	the service name
rq_xid	the transmission id in the request
peer_ip	the peer address where the request is from
rq_prog	the program number in the request

probe::sunrpc.svc.process

probe::sunrpc.svc.process — Process an RPC request

Synopsis

```
sunrpc.svc.process
```

Values

rq_prot	the IP protocol of the reqeust
rq_proc	the procedure number in the request
rq_vers	the program version in the request
sv_name	the service name
sv_prog	the number of the program
sv_nrthreads	the number of concurrent threads
rq_xid	the transmission id in the request
peer_ip	the peer address where the request is from
rq_prog	the program number in the request

probe::sunrpc.svc.recv

probe::sunrpc.svc.recv — Listen for the next RPC request on any socket

Synopsis

```
sunrpc.svc.recv
```

Values

sv_name	the service name
sv_prog	the number of the program
timeout	the timeout of waiting for data
sv_nrthreads	the number of concurrent threads

probe::sunrpc.svc.register

probe::sunrpc.svc.register — Register an RPC service with the local portmapper

Synopsis

```
sunrpc.svc.register
```

Values

prog
: the number of the program

progname
: the name of the program

sv_name
: the service name

port
: the port number

prot
: the IP protocol number

Description

If *proto* and *port* are both 0, then unregister a service.

probe::sunrpc.svc.send

probe::sunrpc.svc.send — Return reply to RPC client

Synopsis

```
sunrpc.svc.send
```

Values

rq_prot	the IP protocol of the reqeust
rq_proc	the procedure number in the request
rq_vers	the program version in the request
sv_name	the service name
rq_xid	the transmission id in the request
peer_ip	the peer address where the request is from
rq_prog	the program number in the request

probe::tcp.disconnect

probe::tcp.disconnect — TCP socket disconnection

Synopsis

```
tcp.disconnect
```

Values

flags	TCP flags (e.g. FIN, etc)
name	Name of this probe
dport	TCP destination port
saddr	A string representing the source IP address
daddr	A string representing the destination IP address
sport	TCP source port
family	IP address family
sock	Network socket

Context

The process which disconnects tcp

probe::tcp.disconnect.return

probe::tcp.disconnect.return — TCP socket disconnection complete

Synopsis

```
tcp.disconnect.return
```

Values

ret Error code (0: no error)

name Name of this probe

Context

The process which disconnects tcp

probe::tcp.receive

probe::tcp.receive — Called when a TCP packet is received

Synopsis

```
tcp.receive
```

Values

urg	TCP URG flag
protocol	Packet protocol from driver
psh	TCP PSH flag
name	Name of the probe point
rst	TCP RST flag
dport	TCP destination port
saddr	A string representing the source IP address
daddr	A string representing the destination IP address
ack	TCP ACK flag
fin	TCP FIN flag
syn	TCP SYN flag
sport	TCP source port
family	IP address family
iphdr	IP header address

probe::tcp.recvmsg

probe::tcp.recvmsg — Receiving TCP message

Synopsis

```
tcp.recvmsg
```

Values

name	Name of this probe
dport	TCP destination port
size	Number of bytes to be received
saddr	A string representing the source IP address
daddr	A string representing the destination IP address
sport	TCP source port
sock	Network socket
family	IP address family

Context

The process which receives a tcp message

probe::tcp.recvmsg.return

probe::tcp.recvmsg.return — Receiving TCP message complete

Synopsis

```
tcp.recvmsg.return
```

Values

name	Name of this probe
dport	TCP destination port
size	Number of bytes received or error code if an error occurred.
saddr	A string representing the source IP address
daddr	A string representing the destination IP address
sport	TCP source port
family	IP address family

Context

The process which receives a tcp message

probe::tcp.sendmsg

probe::tcp.sendmsg — Sending a tcp message

Synopsis

```
tcp.sendmsg
```

Values

name
Name of this probe

size
Number of bytes to send

family
IP address family

sock
Network socket

Context

The process which sends a tcp message

probe::tcp.sendmsg.return

probe::tcp.sendmsg.return — Sending TCP message is done

Synopsis

```
tcp.sendmsg.return
```

Values

name Name of this probe

size Number of bytes sent or error code if an error occurred.

Context

The process which sends a tcp message

probe::tcp.setsockopt

probe::tcp.setsockopt — Call to `setsockopt`

Synopsis

```
tcp.setsockopt
```

Values

optlen
: Used to access values for `setsockopt`

name
: Name of this probe

optname
: TCP socket options (e.g. TCP_NODELAY, TCP_MAXSEG, etc)

optstr
: Resolves optname to a human-readable format

level
: The level at which the socket options will be manipulated

family
: IP address family

sock
: Network socket

Context

The process which calls setsockopt

probe::tcp.setsockopt.return

probe::tcp.setsockopt.return — Return from `setsockopt`

Synopsis

```
tcp.setsockopt.return
```

Values

ret
Error code (0: no error)

name
Name of this probe

Context

The process which calls setsockopt

probe::udp.disconnect

probe::udp.disconnect — Fires when a process requests for a UDP disconnection

Synopsis

```
udp.disconnect
```

Values

flags Flags (e.g. FIN, etc)

name The name of this probe

sock Network socket used by the process

Context

The process which requests a UDP disconnection

probe::udp.disconnect.return

probe::udp.disconnect.return — UDP has been disconnected successfully

Synopsis

```
udp.disconnect.return
```

Values

ret Error code (0: no error)

name The name of this probe

Context

The process which requested a UDP disconnection

probe::udp.recvmsg

probe::udp.recvmsg — Fires whenever a UDP message is received

Synopsis

```
udp.recvmsg
```

Values

name The name of this probe

size Number of bytes received by the process

sock Network socket used by the process

Context

The process which received a UDP message

probe::udp.recvmsg.return

probe::udp.recvmsg.return — Fires whenever an attempt to receive a UDP message received is completed

Synopsis

```
udp.recvmsg.return
```

Values

name The name of this probe

size Number of bytes received by the process

Context

The process which received a UDP message

probe::udp.sendmsg

probe::udp.sendmsg — Fires whenever a process sends a UDP message

Synopsis

```
udp.sendmsg
```

Values

name The name of this probe

size Number of bytes sent by the process

sock Network socket used by the process

Context

The process which sent a UDP message

probe::udp.sendmsg.return

probe::udp.sendmsg.return — Fires whenever an attempt to send a UDP message is completed

Synopsis

```
udp.sendmsg.return
```

Values

name The name of this probe

size Number of bytes sent by the process

Context

The process which sent a UDP message

Chapter 14. Socket Tapset

This family of probe points is used to probe socket activities. It contains the following probe points:

function::inet_get_ip_source

function::inet_get_ip_source — Provide IP source address string for a kernel socket

Synopsis

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

Arguments

sock pointer to the kernel socket

function::inet_get_local_port

function::inet_get_local_port — Provide local port number for a kernel socket

Synopsis

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

Arguments

sock pointer to the kernel socket

function::sock_fam_num2str

function::sock_fam_num2str — Given a protocol family number, return a string representation

Synopsis

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

Arguments

family The family number

function::sock_fam_str2num

function::sock_fam_str2num — Given a protocol family name (string), return the corresponding protocol family number

Synopsis

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

Arguments

family The family name

function::sock_prot_num2str

function::sock_prot_num2str — Given a protocol number, return a string representation

Synopsis

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

Arguments

proto The protocol number

function::sock_prot_str2num

function::sock_prot_str2num — Given a protocol name (string), return the corresponding protocol number

Synopsis

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

Arguments

proto The protocol name

function::sock_state_num2str

function::sock_state_num2str — Given a socket state number, return a string representation

Synopsis

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

Arguments

state The state number

function::sock_state_str2num

function::sock_state_str2num — Given a socket state string, return the corresponding state number

Synopsis

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

Arguments

state The state name

probe::socket.aio_read

probe::socket.aio_read — Receiving message via `sock_aio_read`

Synopsis

```
socket.aio_read
```

Values

protocol	Protocol value
flags	Socket flags value
name	Name of this probe
state	Socket state value
size	Message size in bytes
type	Socket type value
family	Protocol family value

Context

The message sender

Description

Fires at the beginning of receiving a message on a socket via the `sock_aio_read` function

probe::socket.aio_read.return

probe::socket.aio_read.return — Conclusion of message received via `sock_aio_read`

Synopsis

```
socket.aio_read.return
```

Values

success	Was receive successful? (1 = yes, 0 = no)
protocol	Protocol value
flags	Socket flags value
name	Name of this probe
state	Socket state value
size	Size of message received (in bytes) or error code if success = 0
type	Socket type value
family	Protocol family value

Context

The message receiver.

Description

Fires at the conclusion of receiving a message on a socket via the `sock_aio_read` function

probe::socket.aio_write

probe::socket.aio_write — Message send via `sock_aio_write`

Synopsis

```
socket.aio_write
```

Values

protocol	Protocol value
flags	Socket flags value
name	Name of this probe
state	Socket state value
size	Message size in bytes
type	Socket type value
family	Protocol family value

Context

The message sender

Description

Fires at the beginning of sending a message on a socket via the `sock_aio_write` function

probe::socket.aio_write.return

probe::socket.aio_write.return — Conclusion of message send via `sock_aio_write`

Synopsis

```
socket.aio_write.return
```

Values

success	Was receive successful? (1 = yes, 0 = no)
protocol	Protocol value
flags	Socket flags value
name	Name of this probe
state	Socket state value
size	Size of message received (in bytes) or error code if success = 0
type	Socket type value
family	Protocol family value

Context

The message receiver.

Description

Fires at the conclusion of sending a message on a socket via the `sock_aio_write` function

probe::socket.close

probe::socket.close — Close a socket

Synopsis

```
socket.close
```

Values

protocol	Protocol value
flags	Socket flags value
name	Name of this probe
state	Socket state value
type	Socket type value
family	Protocol family value

Context

The requester (user process or kernel)

Description

Fires at the beginning of closing a socket.

probe::socket.close.return

probe::socket.close.return — Return from closing a socket

Synopsis

```
socket.close.return
```

Values

name Name of this probe

Context

The requester (user process or kernel)

Description

Fires at the conclusion of closing a socket.

probe::socket.create

probe::socket.create — Creation of a socket

Synopsis

```
socket.create
```

Values

protocol	Protocol value
name	Name of this probe
requester	Requested by user process or the kernel (1 = kernel, 0 = user)
type	Socket type value
family	Protocol family value

Context

The requester (see requester variable)

Description

Fires at the beginning of creating a socket.

probe::socket.create.return

probe::socket.create.return — Return from Creation of a socket

Synopsis

```
socket.create.return
```

Values

success	Was socket creation successful? (1 = yes, 0 = no)
protocol	Protocol value
err	Error code if success == 0
name	Name of this probe
requester	Requested by user process or the kernel (1 = kernel, 0 = user)
type	Socket type value
family	Protocol family value

Context

The requester (user process or kernel)

Description

Fires at the conclusion of creating a socket.

probe::socket.readv

probe::socket.readv — Receiving a message via `sock_readv`

Synopsis

```
socket.readv
```

Values

protocol	Protocol value
flags	Socket flags value
name	Name of this probe
state	Socket state value
size	Message size in bytes
type	Socket type value
family	Protocol family value

Context

The message sender

Description

Fires at the beginning of receiving a message on a socket via the `sock_readv` function

probe::socket.readv.return

probe::socket.readv.return — Conclusion of receiving a message via `sock_readv`

Synopsis

```
socket.readv.return
```

Values

success	Was receive successful? (1 = yes, 0 = no)
protocol	Protocol value
flags	Socket flags value
name	Name of this probe
state	Socket state value
size	Size of message received (in bytes) or error code if success = 0
type	Socket type value
family	Protocol family value

Context

The message receiver.

Description

Fires at the conclusion of receiving a message on a socket via the `sock_readv` function

probe::socket.receive

probe::socket.receive — Message received on a socket.

Synopsis

```
socket.receive
```

Values

success	Was send successful? (1 = yes, 0 = no)
protocol	Protocol value
flags	Socket flags value
name	Name of this probe
state	Socket state value
size	Size of message received (in bytes) or error code if success = 0
type	Socket type value
family	Protocol family value

Context

The message receiver

probe::socket.recvmsg

probe::socket.recvmsg — Message being received on socket

Synopsis

```
socket.recvmsg
```

Values

protocol	Protocol value
flags	Socket flags value
name	Name of this probe
state	Socket state value
size	Message size in bytes
type	Socket type value
family	Protocol family value

Context

The message receiver.

Description

Fires at the beginning of receiving a message on a socket via the `sock_recvmsg` function

probe::socket.recvmsg.return

probe::socket.recvmsg.return — Return from Message being received on socket

Synopsis

```
socket.recvmsg.return
```

Values

success	Was receive successful? (1 = yes, 0 = no)
protocol	Protocol value
flags	Socket flags value
name	Name of this probe
state	Socket state value
size	Size of message received (in bytes) or error code if success = 0
type	Socket type value
family	Protocol family value

Context

The message receiver.

Description

Fires at the conclusion of receiving a message on a socket via the `sock_recvmsg` function.

probe::socket.send

probe::socket.send — Message sent on a socket.

Synopsis

```
socket.send
```

Values

success	Was send successful? (1 = yes, 0 = no)
protocol	Protocol value
flags	Socket flags value
name	Name of this probe
state	Socket state value
size	Size of message sent (in bytes) or error code if success = 0
type	Socket type value
family	Protocol family value

Context

The message sender

probe::socket.sendmsg

probe::socket.sendmsg — Message is currently being sent on a socket.

Synopsis

```
socket.sendmsg
```

Values

protocol	Protocol value
flags	Socket flags value
name	Name of this probe
state	Socket state value
size	Message size in bytes
type	Socket type value
family	Protocol family value

Context

The message sender

Description

Fires at the beginning of sending a message on a socket via the `sock_sendmsg` function

probe::socket.sendmsg.return

probe::socket.sendmsg.return — Return from socket.sendmsg.

Synopsis

```
socket.sendmsg.return
```

Values

success	Was send successful? (1 = yes, 0 = no)
protocol	Protocol value
flags	Socket flags value
name	Name of this probe
state	Socket state value
size	Size of message sent (in bytes) or error code if success = 0
type	Socket type value
family	Protocol family value

Context

The message sender.

Description

Fires at the conclusion of sending a message on a socket via the `sock_sendmsg` function

probe::socket.writev

probe::socket.writev — Message sent via `socket_writev`

Synopsis

```
socket.writev
```

Values

protocol	Protocol value
flags	Socket flags value
name	Name of this probe
state	Socket state value
size	Message size in bytes
type	Socket type value
family	Protocol family value

Context

The message sender

Description

Fires at the beginning of sending a message on a socket via the `sock_writev` function

probe::socket.writev.return

probe::socket.writev.return — Conclusion of message sent via `socket_writev`

Synopsis

```
socket.writev.return
```

Values

success	Was send successful? (1 = yes, 0 = no)
protocol	Protocol value
flags	Socket flags value
name	Name of this probe
state	Socket state value
size	Size of message sent (in bytes) or error code if success = 0
type	Socket type value
family	Protocol family value

Context

The message receiver.

Description

Fires at the conclusion of sending a message on a socket via the `sock_writev` function

Chapter 15. SNMP Information Tapset

This family of probe points is used to probe socket activities to provide SNMP type information. It contains the following functions and probe points:

function::ipmib_filter_key

function::ipmib_filter_key — Default filter function for ipmib.* probes

Synopsis

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

Arguments

skb
: pointer to the struct sk_buff

op
: value to be counted if *skb* passes the filter

SourceIsLocal
: 1 is local operation and 0 is non-local operation

Description

This function is a default filter function. The user can replace this function with their own. The user-supplied filter function returns an index key based on the values in *skb*. A return value of 0 means this particular *skb* should be not be counted.

function::ipmib_get_proto

function::ipmib_get_proto — Get the protocol value

Synopsis

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

Arguments

skb pointer to a struct sk_buff

Description

Returns the protocol value from *skb*.

function::ipmib_local_addr

function::ipmib_local_addr — Get the local ip address

Synopsis

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

Arguments

skb	pointer to a struct sk_buff
SourceIsLocal	flag to indicate whether local operation

Description

Returns the local ip address *skb*.

function::ipmib_remote_addr

function::ipmib_remote_addr — Get the remote ip address

Synopsis

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

Arguments

skb	pointer to a struct sk_buff
SourceIsLocal	flag to indicate whether local operation

Description

Returns the remote ip address from *skb*.

function::ipmib_tcp_local_port

function::ipmib_tcp_local_port — Get the local tcp port

Synopsis

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

Arguments

skb	pointer to a struct sk_buff
SourceIsLocal	flag to indicate whether local operation

Description

Returns the local tcp port from *skb*.

function::ipmib_tcp_remote_port

function::ipmib_tcp_remote_port — Get the remote tcp port

Synopsis

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

Arguments

skb pointer to a struct sk_buff

SourceIsLocal flag to indicate whether local operation

Description

Returns the remote tcp port from *skb*.

function::linuxmib_filter_key

function::linuxmib_filter_key — Default filter function for linuxmib.* probes

Synopsis

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

Arguments

sk pointer to the struct sock

op value to be counted if *sk* passes the filter

Description

This function is a default filter function. The user can replace this function with their own. The user-supplied filter function returns an index key based on the values in *sk*. A return value of 0 means this particular *sk* should be not be counted.

function::tcpmib_filter_key

function::tcpmib_filter_key — Default filter function for tcpmib.* probes

Synopsis

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

Arguments

sk pointer to the struct sock being acted on

op value to be counted if *sk* passes the filter

Description

This function is a default filter function. The user can replace this function with their own. The user-supplied filter function returns an index key based on the values in *sk*. A return value of 0 means this particular *sk* should be not be counted.

function::tcpmib_get_state

function::tcpmib_get_state — Get a socket's state

Synopsis

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

Arguments

sk pointer to a struct sock

Description

Returns the sk_state from a struct sock.

function::tcpmib_local_addr

function::tcpmib_local_addr — Get the source address

Synopsis

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

Arguments

sk pointer to a struct inet_sock

Description

Returns the saddr from a struct inet_sock in host order.

function::tcpmib_local_port

function::tcpmib_local_port — Get the local port

Synopsis

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

Arguments

sk pointer to a struct inet_sock

Description

Returns the sport from a struct inet_sock in host order.

function::tcpmib_remote_addr

function::tcpmib_remote_addr — Get the remote address

Synopsis

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

Arguments

sk pointer to a struct inet_sock

Description

Returns the daddr from a struct inet_sock in host order.

function::tcpmib_remote_port

function::tcpmib_remote_port — Get the remote port

Synopsis

```
tcpmib_remote_port:long(sk:long)
```

Arguments

sk pointer to a struct inet_sock

Description

Returns the dport from a struct inet_sock in host order.

probe::ipmib.ForwDatagrams

probe::ipmib.ForwDatagrams — Count forwarded packet

Synopsis

```
ipmib.ForwDatagrams
```

Values

skb pointer to the struct sk_buff being acted on

op value to be added to the counter (default value of 1)

Description

The packet pointed to by *skb* is filtered by the function `ipmib_filter_key`. If the packet passes the filter is is counted in the global *ForwDatagrams* (equivalent to SNMP's MIB IPSTATS_MIB_OUTFORWDATAGRAMS)

probe::ipmib.FragFails

probe::ipmib.FragFails — Count datagram fragmented unsuccessfully

Synopsis

```
ipmib.FragFails
```

Values

skb pointer to the struct sk_buff being acted on

op Value to be added to the counter (default value of 1)

Description

The packet pointed to by *skb* is filtered by the function `ipmib_filter_key`. If the packet passes the filter is is counted in the global *FragFails* (equivalent to SNMP's MIB IPSTATS_MIB_FRAGFAILS)

probe::ipmib.FragOKs

probe::ipmib.FragOKs — Count datagram fragmented successfully

Synopsis

```
ipmib.FragOKs
```

Values

skb pointer to the struct sk_buff being acted on

op value to be added to the counter (default value of 1)

Description

The packet pointed to by *skb* is filtered by the function `ipmib_filter_key`. If the packet passes the filter is is counted in the global *FragOKs* (equivalent to SNMP's MIB IPSTATS_MIB_FRAGOKS)

probe::ipmib.InAddrErrors

probe::ipmib.InAddrErrors — Count arriving packets with an incorrect address

Synopsis

```
ipmib.InAddrErrors
```

Values

skb pointer to the struct sk_buff being acted on

op value to be added to the counter (default value of 1)

Description

The packet pointed to by *skb* is filtered by the function `ipmib_filter_key`. If the packet passes the filter is is counted in the global *InAddrErrors* (equivalent to SNMP's MIB IPSTATS_MIB_INADDRERRORS)

probe::ipmib.InDiscards

probe::ipmib.InDiscards — Count discarded inbound packets

Synopsis

```
ipmib.InDiscards
```

Values

skb pointer to the struct sk_buff being acted on

op value to be added to the counter (default value of 1)

Description

The packet pointed to by *skb* is filtered by the function `ipmib_filter_key`. If the packet passes the filter is is counted in the global *InDiscards* (equivalent to SNMP's MIB STATS_MIB_INDISCARDS)

probe::ipmib.InNoRoutes

probe::ipmib.InNoRoutes — Count an arriving packet with no matching socket

Synopsis

```
ipmib.InNoRoutes
```

Values

skb pointer to the struct sk_buff being acted on

op value to be added to the counter (default value of 1)

Description

The packet pointed to by *skb* is filtered by the function `ipmib_filter_key`. If the packet passes the filter is is counted in the global *InNoRoutes* (equivalent to SNMP's MIB IPSTATS_MIB_INNOROUTES)

probe::ipmib.InReceives

probe::ipmib.InReceives — Count an arriving packet

Synopsis

```
ipmib.InReceives
```

Values

skb pointer to the struct sk_buff being acted on

op value to be added to the counter (default value of 1)

Description

The packet pointed to by *skb* is filtered by the function `ipmib_filter_key`. If the packet passes the filter is is counted in the global *InReceives* (equivalent to SNMP's MIB IPSTATS_MIB_INRECEIVES)

probe::ipmib.InUnknownProtos

probe::ipmib.InUnknownProtos — Count arriving packets with an unbound proto

Synopsis

```
ipmib.InUnknownProtos
```

Values

skb pointer to the struct sk_buff being acted on

op value to be added to the counter (default value of 1)

Description

The packet pointed to by *skb* is filtered by the function `ipmib_filter_key`. If the packet passes the filter is is counted in the global *InUnknownProtos* (equivalent to SNMP's MIB IPSTATS_MIB_INUNKNOWNPROTOS)

probe::ipmib.OutRequests

probe::ipmib.OutRequests — Count a request to send a packet

Synopsis

```
ipmib.OutRequests
```

Values

skb pointer to the struct sk_buff being acted on

op value to be added to the counter (default value of 1)

Description

The packet pointed to by *skb* is filtered by the function `ipmib_filter_key`. If the packet passes the filter is is counted in the global *OutRequests* (equivalent to SNMP's MIB IPSTATS_MIB_OUTREQUESTS)

probe::ipmib.ReasmReqds

probe::ipmib.ReasmReqds — Count number of packet fragments reassembly requests

Synopsis

```
ipmib.ReasmReqds
```

Values

skb pointer to the struct sk_buff being acted on

op value to be added to the counter (default value of 1)

Description

The packet pointed to by *skb* is filtered by the function `ipmib_filter_key`. If the packet passes the filter is is counted in the global *ReasmReqds* (equivalent to SNMP's MIB IPSTATS_MIB_REASMREQDS)

probe::ipmib.ReasmTimeout

probe::ipmib.ReasmTimeout — Count Reassembly Timeouts

Synopsis

```
ipmib.ReasmTimeout
```

Values

skb pointer to the struct sk_buff being acted on

op value to be added to the counter (default value of 1)

Description

The packet pointed to by *skb* is filtered by the function `ipmib_filter_key`. If the packet passes the filter is is counted in the global *ReasmTimeout* (equivalent to SNMP's MIB IPSTATS_MIB_REASMTIMEOUT)

probe::linuxmib.DelayedACKs

probe::linuxmib.DelayedACKs — Count of delayed acks

Synopsis

```
linuxmib.DelayedACKs
```

Values

sk Pointer to the struct sock being acted on

op Value to be added to the counter (default value of 1)

Description

The packet pointed to by *skb* is filtered by the function `linuxmib_filter_key`. If the packet passes the filter is is counted in the global *DelayedACKs* (equivalent to SNMP's MIB LINUX_MIB_DELAYEDACKS)

probe::linuxmib.ListenDrops

probe::linuxmib.ListenDrops — Count of times conn request that were dropped

Synopsis

```
linuxmib.ListenDrops
```

Values

sk Pointer to the struct sock being acted on

op Value to be added to the counter (default value of 1)

Description

The packet pointed to by *skb* is filtered by the function `linuxmib_filter_key`. If the packet passes the filter is is counted in the global *ListenDrops* (equivalent to SNMP's MIB LINUX_MIB_LISTENDROPS)

probe::linuxmib.ListenOverflows

probe::linuxmib.ListenOverflows — Count of times a listen queue overflowed

Synopsis

```
linuxmib.ListenOverflows
```

Values

sk Pointer to the struct sock being acted on

op Value to be added to the counter (default value of 1)

Description

The packet pointed to by *skb* is filtered by the function `linuxmib_filter_key`. If the packet passes the filter is is counted in the global *ListenOverflows* (equivalent to SNMP's MIB LINUX_MIB_LISTENOVERFLOWS)

probe::linuxmib.TCPMemoryPressures

probe::linuxmib.TCPMemoryPressures — Count of times memory pressure was used

Synopsis

```
linuxmib.TCPMemoryPressures
```

Values

sk Pointer to the struct sock being acted on

op Value to be added to the counter (default value of 1)

Description

The packet pointed to by *skb* is filtered by the function `linuxmib_filter_key`. If the packet passes the filter is is counted in the global *TCPMemoryPressures* (equivalent to SNMP's MIB LINUX_MIB_TCPMEMORYPRESSURES)

probe::tcpmib.ActiveOpens

probe::tcpmib.ActiveOpens — Count an active opening of a socket

Synopsis

```
tcpmib.ActiveOpens
```

Values

sk
: pointer to the struct sock being acted on

op
: value to be added to the counter (default value of 1)

Description

The packet pointed to by *skb* is filtered by the function `tcpmib_filter_key`. If the packet passes the filter is is counted in the global *ActiveOpens* (equivalent to SNMP's MIB TCP_MIB_ACTIVEOPENS)

probe::tcpmib.AttemptFails

probe::tcpmib.AttemptFails — Count a failed attempt to open a socket

Synopsis

```
tcpmib.AttemptFails
```

Values

sk pointer to the struct sock being acted on

op value to be added to the counter (default value of 1)

Description

The packet pointed to by *skb* is filtered by the function `tcpmib_filter_key`. If the packet passes the filter is is counted in the global *AttemptFails* (equivalent to SNMP's MIB TCP_MIB_ATTEMPTFAILS)

probe::tcpmib.CurrEstab

probe::tcpmib.CurrEstab — Update the count of open sockets

Synopsis

```
tcpmib.CurrEstab
```

Values

sk pointer to the struct sock being acted on

op value to be added to the counter (default value of 1)

Description

The packet pointed to by *skb* is filtered by the function `tcpmib_filter_key`. If the packet passes the filter is is counted in the global *CurrEstab* (equivalent to SNMP's MIB TCP_MIB_CURRESTAB)

probe::tcpmib.EstabResets

probe::tcpmib.EstabResets — Count the reset of a socket

Synopsis

```
tcpmib.EstabResets
```

Values

sk pointer to the struct sock being acted on

op value to be added to the counter (default value of 1)

Description

The packet pointed to by *skb* is filtered by the function `tcpmib_filter_key`. If the packet passes the filter is is counted in the global *EstabResets* (equivalent to SNMP's MIB TCP_MIB_ESTABRESETS)

probe::tcpmib.InSegs

probe::tcpmib.InSegs — Count an incoming tcp segment

Synopsis

```
tcpmib.InSegs
```

Values

sk pointer to the struct sock being acted on

op value to be added to the counter (default value of 1)

Description

The packet pointed to by *skb* is filtered by the function `tcpmib_filter_key` (or `ipmib_filter_key` for tcp v4). If the packet passes the filter is is counted in the global *InSegs* (equivalent to SNMP's MIB TCP_MIB_INSEGS)

probe::tcpmib.OutRsts

probe::tcpmib.OutRsts — Count the sending of a reset packet

Synopsis

```
tcpmib.OutRsts
```

Values

sk pointer to the struct sock being acted on

op value to be added to the counter (default value of 1)

Description

The packet pointed to by *skb* is filtered by the function `tcpmib_filter_key`. If the packet passes the filter is is counted in the global *OutRsts* (equivalent to SNMP's MIB TCP_MIB_OUTRSTS)

probe::tcpmib.OutSegs

probe::tcpmib.OutSegs — Count the sending of a TCP segment

Synopsis

```
tcpmib.OutSegs
```

Values

sk pointer to the struct sock being acted on

op value to be added to the counter (default value of 1)

Description

The packet pointed to by *skb* is filtered by the function `tcpmib_filter_key`. If the packet passes the filter is is counted in the global *OutSegs* (equivalent to SNMP's MIB TCP_MIB_OUTSEGS)

probe::tcpmib.PassiveOpens

probe::tcpmib.PassiveOpens — Count the passive creation of a socket

Synopsis

```
tcpmib.PassiveOpens
```

Values

sk pointer to the struct sock being acted on

op value to be added to the counter (default value of 1)

Description

The packet pointed to by *skb* is filtered by the function `tcpmib_filter_key`. If the packet passes the filter is is counted in the global *PassiveOpens* (equivalent to SNMP's MIB TCP_MIB_PASSIVEOPENS)

probe::tcpmib.RetransSegs

probe::tcpmib.RetransSegs — Count the retransmission of a TCP segment

Synopsis

```
tcpmib.RetransSegs
```

Values

sk pointer to the struct sock being acted on

op value to be added to the counter (default value of 1)

Description

The packet pointed to by *skb* is filtered by the function `tcpmib_filter_key`. If the packet passes the filter is is counted in the global *RetransSegs* (equivalent to SNMP's MIB TCP_MIB_RETRANSSEGS)

Chapter 16. Kernel Process Tapset

This family of probe points is used to probe process-related activities. It contains the following probe points:

function::target_set_pid

function::target_set_pid — Does pid descend from target process?

Synopsis

```
target_set_pid(pid:)
```

Arguments

pid The pid of the process to query

Description

This function returns whether the given process-id is within the “target set”, that is whether it is a descendant of the top-level `target` process.

function::target_set_report

function::target_set_report — Print a report about the target set

Synopsis

```
target_set_report()
```

Arguments

None

Description

This function prints a report about the processes in the target set, and their ancestry.

probe::kprocess.create

probe::kprocess.create — Fires whenever a new process or thread is successfully created

Synopsis

```
kprocess.create
```

Values

new_tid The TID of the newly created task

new_pid The PID of the newly created process

Context

Parent of the created process.

Description

Fires whenever a new process is successfully created, either as a result of fork (or one of its syscall variants), or a new kernel thread.

probe::kprocess.exec

probe::kprocess.exec — Attempt to exec to a new program

Synopsis

```
kprocess.exec
```

Values

filename	The path to the new executable

Context

The caller of exec.

Description

Fires whenever a process attempts to exec to a new program.

probe::kprocess.exec_complete

probe::kprocess.exec_complete — Return from exec to a new program

Synopsis

```
kprocess.exec_complete
```

Values

success	A boolean indicating whether the exec was successful
errno	The error number resulting from the exec

Context

On success, the context of the new executable. On failure, remains in the context of the caller.

Description

Fires at the completion of an exec call.

probe::kprocess.exit

probe::kprocess.exit — Exit from process

Synopsis

```
kprocess.exit
```

Values

code The exit code of the process

Context

The process which is terminating.

Description

Fires when a process terminates. This will always be followed by a kprocess.release, though the latter may be delayed if the process waits in a zombie state.

probe::kprocess.release

probe::kprocess.release — Process released

Synopsis

```
kprocess.release
```

Values

pid	Same as *released_pid* for compatibility (deprecated)
released_pid	PID of the process being released
released_tid	TID of the task being released
task	A task handle to the process being released

Context

The context of the parent, if it wanted notification of this process' termination, else the context of the process itself.

Description

Fires when a process is released from the kernel. This always follows a kprocess.exit, though it may be delayed somewhat if the process waits in a zombie state.

probe::kprocess.start

probe::kprocess.start — Starting new process

Synopsis

```
kprocess.start
```

Values

None

Context

Newly created process.

Description

Fires immediately before a new process begins execution.

Chapter 17. Signal Tapset

This family of probe points is used to probe signal activities. It contains the following probe points:

function::get_sa_flags

function::get_sa_flags — Returns the numeric value of sa_flags

Synopsis

```
get_sa_flags:long(act:long)
```

Arguments

act address of the sigaction to query.

function::get_sa_handler

function::get_sa_handler — Returns the numeric value of sa_handler

Synopsis

```
get_sa_handler:long(act:long)
```

Arguments

act address of the sigaction to query.

function::is_sig_blocked

function::is_sig_blocked — Returns 1 if the signal is currently blocked, or 0 if it is not

Synopsis

```
is_sig_blocked:long(task:long,sig:long)
```

Arguments

task address of the task_struct to query.

sig the signal number to test.

function::sa_flags_str

function::sa_flags_str — Returns the string representation of sa_flags

Synopsis

```
sa_flags_str:string(sa_flags:long)
```

Arguments

sa_flags the set of flags to convert to string.

function::sa_handler_str

function::sa_handler_str — Returns the string representation of an sa_handler

Synopsis

```
sa_handler_str(handler:)
```

Arguments

handler the sa_handler to convert to string.

Description

Returns the string representation of an sa_handler. If it is not SIG_DFL, SIG_IGN or SIG_ERR, it will return the address of the handler.

function::signal_str

function::signal_str — Returns the string representation of a signal number

Synopsis

```
signal_str(num:)
```

Arguments

num the signal number to convert to string.

function::sigset_mask_str

function::sigset_mask_str — Returns the string representation of a sigset

Synopsis

```
sigset_mask_str:string(mask:long)
```

Arguments

mask the sigset to convert to string.

probe::signal.check_ignored

probe::signal.check_ignored — Checking to see signal is ignored

Synopsis

```
signal.check_ignored
```

Values

sig_name	A string representation of the signal
sig	The number of the signal
pid_name	Name of the process receiving the signal
sig_pid	The PID of the process receiving the signal

probe::signal.check_ignored.return

probe::signal.check_ignored.return — Check to see signal is ignored completed

Synopsis

```
signal.check_ignored.return
```

Values

retstr
: Return value as a string

name
: Name of the probe point

probe::signal.checkperm

probe::signal.checkperm — Check being performed on a sent signal

Synopsis

```
signal.checkperm
```

Values

name	Name of the probe point
task	A task handle to the signal recipient
sinfo	The address of the siginfo structure
si_code	Indicates the signal type
sig_name	A string representation of the signal
sig	The number of the signal
pid_name	Name of the process receiving the signal
sig_pid	The PID of the process receiving the signal

probe::signal.checkperm.return

probe::signal.checkperm.return — Check performed on a sent signal completed

Synopsis

```
signal.checkperm.return
```

Values

retstr
: Return value as a string

name
: Name of the probe point

probe::signal.do_action

probe::signal.do_action — Examining or changing a signal action

Synopsis

```
signal.do_action
```

Values

sa_mask	The new mask of the signal
name	Name of the probe point
sig_name	A string representation of the signal
oldsigact_addr	The address of the old sigaction struct associated with the signal
sig	The signal to be examined/changed
sa_handler	The new handler of the signal
sigact_addr	The address of the new sigaction struct associated with the signal

probe::signal.do_action.return

probe::signal.do_action.return — Examining or changing a signal action completed

Synopsis

```
signal.do_action.return
```

Values

retstr Return value as a string

name Name of the probe point

probe::signal.flush

probe::signal.flush — Flushing all pending signals for a task

Synopsis

```
signal.flush
```

Values

name	Name of the probe point
task	The task handler of the process performing the flush
pid_name	The name of the process associated with the task performing the flush
sig_pid	The PID of the process associated with the task performing the flush

probe::signal.force_segv

probe::signal.force_segv — Forcing send of SIGSEGV

Synopsis

```
signal.force_segv
```

Values

name	Name of the probe point
sig_name	A string representation of the signal
sig	The number of the signal
pid_name	Name of the process receiving the signal
sig_pid	The PID of the process receiving the signal

probe::signal.force_segv.return

probe::signal.force_segv.return — Forcing send of SIGSEGV complete

Synopsis

```
signal.force_segv.return
```

Values

retstr Return value as a string

name Name of the probe point

probe::signal.handle

probe::signal.handle — Signal handler being invoked

Synopsis

```
signal.handle
```

Values

regs The address of the kernel-mode stack area (deprecated in SystemTap 2.1)

sig_code The si_code value of the siginfo signal

name Name of the probe point

sig_mode Indicates whether the signal was a user-mode or kernel-mode signal

sinfo The address of the siginfo table

sig_name A string representation of the signal

oldset_addr The address of the bitmask array of blocked signals (deprecated in SystemTap 2.1)

sig The signal number that invoked the signal handler

ka_addr The address of the k_sigaction table associated with the signal

probe::signal.handle.return

probe::signal.handle.return — Signal handler invocation completed

Synopsis

```
signal.handle.return
```

Values

retstr	Return value as a string
name	Name of the probe point

Description

(deprecated in SystemTap 2.1)

probe::signal.pending

probe::signal.pending — Examining pending signal

Synopsis

```
signal.pending
```

Values

name	Name of the probe point
sigset_size	The size of the user-space signal set
sigset_add	The address of the user-space signal set (sigset_t)

Description

This probe is used to examine a set of signals pending for delivery to a specific thread. This normally occurs when the do_sigpending kernel function is executed.

probe::signal.pending.return

probe::signal.pending.return — Examination of pending signal completed

Synopsis

```
signal.pending.return
```

Values

retstr
: Return value as a string

name
: Name of the probe point

probe::signal.procmask

probe::signal.procmask — Examining or changing blocked signals

Synopsis

```
signal.procmask
```

Values

how
: Indicates how to change the blocked signals; possible values are SIG_BLOCK=0 (for blocking signals), SIG_UNBLOCK=1 (for unblocking signals), and SIG_SETMASK=2 for setting the signal mask.

name
: Name of the probe point

oldsigset_addr
: The old address of the signal set (sigset_t)

sigset
: The actual value to be set for sigset_t (correct?)

sigset_addr
: The address of the signal set (sigset_t) to be implemented

probe::signal.procmask.return

probe::signal.procmask.return — Examining or changing blocked signals completed

Synopsis

```
signal.procmask.return
```

Values

retstr
: Return value as a string

name
: Name of the probe point

probe::signal.send

probe::signal.send — Signal being sent to a process

Synopsis

```
signal.send
```

Values

send2queue Indicates whether the signal is sent to an existing sigqueue (deprecated in SystemTap 2.1)

name The name of the function used to send out the signal

task A task handle to the signal recipient

sinfo The address of siginfo struct

si_code Indicates the signal type

sig_name A string representation of the signal

sig The number of the signal

shared Indicates whether the signal is shared by the thread group

sig_pid The PID of the process receiving the signal

pid_name The name of the signal recipient

Context

The signal's sender.

probe::signal.send.return

probe::signal.send.return — Signal being sent to a process completed (deprecated in SystemTap 2.1)

Synopsis

```
signal.send.return
```

Values

retstr The return value to either __group_send_sig_info, specific_send_sig_info, or send_sigqueue

send2queue Indicates whether the sent signal was sent to an existing sigqueue

name The name of the function used to send out the signal

shared Indicates whether the sent signal is shared by the thread group.

Context

The signal's sender. (correct?)

Description

Possible __group_send_sig_info and specific_send_sig_info return values are as follows;

0 -- The signal is successfully sent to a process, which means that, (1) the signal was ignored by the receiving process, (2) this is a non-RT signal and the system already has one queued, and (3) the signal was successfully added to the sigqueue of the receiving process.

-EAGAIN -- The sigqueue of the receiving process is overflowing, the signal was RT, and the signal was sent by a user using something other than `kill`.

Possible send_group_sigqueue and send_sigqueue return values are as follows;

0 -- The signal was either successfully added into the sigqueue of the receiving process, or a SI_TIMER entry is already queued (in which case, the overrun count will be simply incremented).

1 -- The signal was ignored by the receiving process.

-1 -- (send_sigqueue only) The task was marked exiting, allowing * posix_timer_event to redirect it to the group leader.

probe::signal.send_sig_queue

probe::signal.send_sig_queue — Queuing a signal to a process

Synopsis

```
signal.send_sig_queue
```

Values

sigqueue_addr	The address of the signal queue
name	Name of the probe point
sig_name	A string representation of the signal
sig	The queued signal
pid_name	Name of the process to which the signal is queued
sig_pid	The PID of the process to which the signal is queued

probe::signal.send_sig_queue.return

probe::signal.send_sig_queue.return — Queuing a signal to a process completed

Synopsis

```
signal.send_sig_queue.return
```

Values

retstr Return value as a string

name Name of the probe point

probe::signal.sys_tgkill

probe::signal.sys_tgkill — Sending kill signal to a thread group

Synopsis

```
signal.sys_tgkill
```

Values

name	Name of the probe point
sig_name	A string representation of the signal
sig	The specific kill signal sent to the process
tgid	The thread group ID of the thread receiving the kill signal
pid_name	The name of the signal recipient
sig_pid	The PID of the thread receiving the kill signal

Description

The tgkill call is similar to tkill, except that it also allows the caller to specify the thread group ID of the thread to be signalled. This protects against TID reuse.

probe::signal.sys_tgkill.return

probe::signal.sys_tgkill.return — Sending kill signal to a thread group completed

Synopsis

```
signal.sys_tgkill.return
```

Values

retstr
: The return value to either __group_send_sig_info,

name
: Name of the probe point

probe::signal.sys_tkill

probe::signal.sys_tkill — Sending a kill signal to a thread

Synopsis

```
signal.sys_tkill
```

Values

name	Name of the probe point
sig_name	A string representation of the signal
sig	The specific signal sent to the process
pid_name	The name of the signal recipient
sig_pid	The PID of the process receiving the kill signal

Description

The tkill call is analogous to kill(2), except that it also allows a process within a specific thread group to be targeted. Such processes are targeted through their unique thread IDs (TID).

probe::signal.syskill

probe::signal.syskill — Sending kill signal to a process

Synopsis

```
signal.syskill
```

Values

name	Name of the probe point
sig_name	A string representation of the signal
sig	The specific signal sent to the process
pid_name	The name of the signal recipient
sig_pid	The PID of the process receiving the signal

probe::signal.syskill.return

probe::signal.syskill.return — Sending kill signal completed

Synopsis

```
signal.syskill.return
```

Values

None

probe::signal.systkill.return

probe::signal.systkill.return — Sending kill signal to a thread completed

Synopsis

```
signal.systkill.return
```

Values

retstr The return value to either __group_send_sig_info,

name Name of the probe point

probe::signal.wakeup

probe::signal.wakeup — Sleeping process being wakened for signal

Synopsis

```
signal.wakeup
```

Values

resume
: Indicates whether to wake up a task in a STOPPED or TRACED state

state_mask
: A string representation indicating the mask of task states to wake. Possible values are TASK_INTERRUPTIBLE, TASK_STOPPED, TASK_TRACED, TASK_WAKEKILL, and TASK_INTERRUPTIBLE.

pid_name
: Name of the process to wake

sig_pid
: The PID of the process to wake

Chapter 18. Errno Tapset

This set of functions is used to handle errno number values. It contains the following functions:

function::errno_str

function::errno_str — Symbolic string associated with error code

Synopsis

```
errno_str:string(err:long)
```

Arguments

err The error number received

Description

This function returns the symbolic string associated with the giver error code, such as ENOENT for the number 2, or E#3333 for an out-of-range value such as 3333.

function::return_str

function::return_str — Formats the return value as a string

Synopsis

```
return_str:string(format:long,ret:long)
```

Arguments

format Variable to determine return type base value

ret Return value (typically `$return`)

Description

This function is used by the syscall tapset, and returns a string. Set format equal to 1 for a decimal, 2 for hex, 3 for octal.

Note that this function is preferred over `returnstr`.

function::returnstr

function::returnstr — Formats the return value as a string

Synopsis

```
returnstr:string(format:long)
```

Arguments

format Variable to determine return type base value

Description

This function is used by the nd_syscall tapset, and returns a string. Set format equal to 1 for a decimal, 2 for hex, 3 for octal.

Note that this function should only be used in dwarfless probes (i.e. 'kprobe.function("foo")'). Other probes should use `return_str`.

function::returnval

function::returnval — Possible return value of probed function

Synopsis

```
returnval:long()
```

Arguments

None

Description

Return the value of the register in which function values are typically returned. Can be used in probes where `$return` isn't available. This is only a guess of the actual return value and can be totally wrong. Normally only used in dwarfless probes.

Chapter 19. Device Tapset

This set of functions is used to handle kernel and userspace device numbers. It contains the following functions:

function::MAJOR

function::MAJOR — Extract major device number from a kernel device number (kdev_t)

Synopsis

```
MAJOR:long(dev:long)
```

Arguments

dev Kernel device number to query.

function::MINOR

function::MINOR — Extract minor device number from a kernel device number (kdev_t)

Synopsis

```
MINOR:long(dev:long)
```

Arguments

dev Kernel device number to query.

function::MKDEV

function::MKDEV — Creates a value that can be compared to a kernel device number (kdev_t)

Synopsis

```
MKDEV:long(major:long,minor:long)
```

Arguments

major Intended major device number.

minor Intended minor device number.

function::usrdev2kerndev

function::usrdev2kerndev — Converts a user-space device number into the format used in the kernel

Synopsis

```
usrdev2kerndev:long(dev:long)
```

Arguments

dev Device number in user-space format.

Chapter 20. Directory-entry (dentry) Tapset

This family of functions is used to map kernel VFS directory entry pointers to file or full path names.

function::d_name

function::d_name — get the dirent name

Synopsis

```
d_name:string(dentry:long)
```

Arguments

dentry Pointer to dentry.

Description

Returns the dirent name (path basename).

function::d_path

function::d_path — get the full nameidata path

Synopsis

```
d_path:string(nd:long)
```

Arguments

nd Pointer to nameidata.

Description

Returns the full dirent name (full path to the root), like the kernel d_path function.

function::inode_name

function::inode_name — get the inode name

Synopsis

```
inode_name:string(inode:long)
```

Arguments

inode Pointer to inode.

Description

Returns the first path basename associated with the given inode.

function::real_mount

function::real_mount — get the ’struct mount’ pointer

Synopsis

```
real_mount:long(vfsmnt:long)
```

Arguments

vfsmnt Pointer to ’struct vfsmount’

Description

Returns the ’struct mount’ pointer value for a ’struct vfsmount’ pointer.

function::reverse_path_walk

function::reverse_path_walk — get the full dirent path

Synopsis

```
reverse_path_walk:string(dentry:long)
```

Arguments

dentry Pointer to dentry.

Description

Returns the path name (partial path to mount point).

function::task_dentry_path

function::task_dentry_path — get the full dentry path

Synopsis

```
task_dentry_path:string(task:long,dentry:long,vfsmnt:long)
```

Arguments

task task_struct pointer.

dentry direntry pointer.

vfsmnt vfsmnt pointer.

Description

Returns the full dirent name (full path to the root), like the kernel d_path function.

Chapter 21. Logging Tapset

This family of functions is used to send simple message strings to various destinations.

function::error

function::error — Send an error message

Synopsis

```
error(msg:string)
```

Arguments

msg The formatted message string

Description

An implicit end-of-line is added. staprun prepends the string "ERROR:". Sending an error message aborts the currently running probe. Depending on the MAXERRORS parameter, it may trigger an `exit`.

function::exit

function::exit — Start shutting down probing script.

Synopsis

```
exit()
```

Arguments

None

Description

This only enqueues a request to start shutting down the script. New probes will not fire (except “end” probes), but all currently running ones may complete their work.

function::ftrace

function::ftrace — Send a message to the ftrace ring-buffer

Synopsis

```
ftrace(msg:string)
```

Arguments

msg The formatted message string

Description

If the ftrace ring-buffer is configured & available, see /debugfs/tracing/trace for the message. Otherwise, the message may be quietly dropped. An implicit end-of-line is added.

function::log

function::log — Send a line to the common trace buffer

Synopsis

```
log(msg:string)
```

Arguments

msg The formatted message string

Description

This function logs data. log sends the message immediately to staprun and to the bulk transport (relayfs) if it is being used. If the last character given is not a newline, then one is added. This function is not as effecient as printf and should be used only for urgent messages.

function::printk

function::printk — Send a message to the kernel trace buffer

Synopsis

```
printk(level:long,msg:string)
```

Arguments

level an integer for the severity level (0=KERN_EMERG ... 7=KERN_DEBUG)

msg The formatted message string

Description

Print a line of text to the kernel dmesg/console with the given severity. An implicit end-of-line is added. This function may not be safely called from all kernel probe contexts, so is restricted to guru mode only.

function::warn

function::warn — Send a line to the warning stream

Synopsis

```
warn(msg:string)
```

Arguments

msg The formatted message string

Description

This function sends a warning message immediately to staprun. It is also sent over the bulk transport (relayfs) if it is being used. If the last characater is not a newline, the one is added.

Chapter 22. Queue Statistics Tapset

This family of functions is used to track performance of queuing systems.

function::qs_done

function::qs_done — Function to record finishing request

Synopsis

```
qs_done(qname:string)
```

Arguments

qname the name of the service that finished

Description

This function records that a request originally from the given queue has completed being serviced.

function::qs_run

function::qs_run — Function to record being moved from wait queue to being serviced

Synopsis

```
qs_run(qname:string)
```

Arguments

qname the name of the service being moved and started

Description

This function records that the previous enqueued request was removed from the given wait queue and is now being serviced.

function::qs_wait

function::qs_wait — Function to record enqueue requests

Synopsis

```
qs_wait(qname:string)
```

Arguments

qname the name of the queue requesting enqueue

Description

This function records that a new request was enqueued for the given queue name.

function::qsq_blocked

function::qsq_blocked — Returns the time reqest was on the wait queue

Synopsis

```
qsq_blocked:long(qname:string,scale:long)
```

Arguments

qname queue name

scale scale variable to take account for interval fraction

Description

This function returns the fraction of elapsed time during which one or more requests were on the wait queue.

function::qsq_print

function::qsq_print — Prints a line of statistics for the given queue

Synopsis

```
qsq_print(qname:string)
```

Arguments

qname queue name

Description

This function prints a line containing the following

statistics for the given queue

the queue name, the average rate of requests per second, the average wait queue length, the average time on the wait queue, the average time to service a request, the percentage of time the wait queue was used, and the percentage of time request was being serviced.

function::qsq_service_time

function::qsq_service_time — Amount of time per request service

Synopsis

```
qsq_service_time:long(qname:string,scale:long)
```

Arguments

qname queue name

scale scale variable to take account for interval fraction

Description

This function returns the average time in microseconds required to service a request once it is removed from the wait queue.

function::qsq_start

function::qsq_start — Function to reset the stats for a queue

Synopsis

```
qsq_start(qname:string)
```

Arguments

qname the name of the service that finished

Description

This function resets the statistics counters for the given queue, and restarts tracking from the moment the function was called. This function is also used to create intialize a queue.

function::qsq_throughput

function::qsq_throughput — Number of requests served per unit time

Synopsis

```
qsq_throughput:long(qname:string,scale:long)
```

Arguments

qname queue name

scale scale variable to take account for interval fraction

Description

This function returns the average number or requests served per microsecond.

function::qsq_utilization

function::qsq_utilization — Fraction of time that any request was being serviced

Synopsis

```
qsq_utilization:long(qname:string,scale:long)
```

Arguments

qname queue name

scale scale variable to take account for interval fraction

Description

This function returns the average time in microseconds that at least one request was being serviced.

function::qsq_wait_queue_length

function::qsq_wait_queue_length — length of wait queue

Synopsis

```
qsq_wait_queue_length:long(qname:string,scale:long)
```

Arguments

qname queue name

scale scale variable to take account for interval fraction

Description

This function returns the average length of the wait queue

function::qsq_wait_time

function::qsq_wait_time — Amount of time in queue + service per request

Synopsis

```
qsq_wait_time:long(qname:string,scale:long)
```

Arguments

qname queue name

scale scale variable to take account for interval fraction

Description

This function returns the average time in microseconds that it took for a request to be serviced (`qs_wait` to `qa_done`).

Chapter 23. Random functions Tapset

These functions deal with random number generation.

function::randint

function::randint — Return a random number between [0,n)

Synopsis

```
randint:long(n:long)
```

Arguments

n Number past upper limit of range, not larger than 2**20.

Chapter 24. String and data retrieving functions Tapset

Functions to retrieve strings and other primitive types from the kernel or a user space programs based on addresses. All strings are of a maximum length given by MAXSTRINGLEN.

function::atomic_long_read

function::atomic_long_read — Retrieves an atomic long variable from kernel memory

Synopsis

```
atomic_long_read:long(addr:long)
```

Arguments

addr pointer to atomic long variable

Description

Safely perform the read of an atomic long variable. This will be a NOP on kernels that do not have ATOMIC_LONG_INIT set on the kernel config.

function::atomic_read

function::atomic_read — Retrieves an atomic variable from kernel memory

Synopsis

```
atomic_read:long(addr:long)
```

Arguments

addr pointer to atomic variable

Description

Safely perform the read of an atomic variable.

function::kernel_char

function::kernel_char — Retrieves a char value stored in kernel memory

Synopsis

```
kernel_char:long(addr:long)
```

Arguments

addr The kernel address to retrieve the char from

Description

Returns the char value from a given kernel memory address. Reports an error when reading from the given address fails.

function::kernel_int

function::kernel_int — Retrieves an int value stored in kernel memory

Synopsis

```
kernel_int:long(addr:long)
```

Arguments

addr The kernel address to retrieve the int from

Description

Returns the int value from a given kernel memory address. Reports an error when reading from the given address fails.

function::kernel_long

function::kernel_long — Retrieves a long value stored in kernel memory

Synopsis

```
kernel_long:long(addr:long)
```

Arguments

addr The kernel address to retrieve the long from

Description

Returns the long value from a given kernel memory address. Reports an error when reading from the given address fails.

function::kernel_pointer

function::kernel_pointer — Retrieves a pointer value stored in kernel memory

Synopsis

```
kernel_pointer:long(addr:long)
```

Arguments

addr The kernel address to retrieve the pointer from

Description

Returns the pointer value from a given kernel memory address. Reports an error when reading from the given address fails.

function::kernel_short

function::kernel_short — Retrieves a short value stored in kernel memory

Synopsis

```
kernel_short:long(addr:long)
```

Arguments

addr The kernel address to retrieve the short from

Description

Returns the short value from a given kernel memory address. Reports an error when reading from the given address fails.

function::kernel_string

function::kernel_string — Retrieves string from kernel memory

Synopsis

```
kernel_string:string(addr:long)
```

Arguments

addr The kernel address to retrieve the string from

Description

This function returns the null terminated C string from a given kernel memory address. Reports an error on string copy fault.

function::kernel_string2

function::kernel_string2 — Retrieves string from kernel memory with alternative error string

Synopsis

```
kernel_string2:string(addr:long,err_msg:string)
```

Arguments

addr The kernel address to retrieve the string from

err_msg The error message to return when data isn't available

Description

This function returns the null terminated C string from a given kernel memory address. Reports the given error message on string copy fault.

function::kernel_string2_utf16

function::kernel_string2_utf16 — Retrieves UTF-16 string from kernel memory with alternative error string

Synopsis

```
kernel_string2_utf16:string(addr:long,err_msg:string)
```

Arguments

addr The kernel address to retrieve the string from

err_msg The error message to return when data isn't available

Description

This function returns a null terminated UTF-8 string converted from the UTF-16 string at a given kernel memory address. Reports the given error message on string copy fault or conversion error.

function::kernel_string2_utf32

function::kernel_string2_utf32 — Retrieves UTF-32 string from kernel memory with alternative error string

Synopsis

```
kernel_string2_utf32:string(addr:long,err_msg:string)
```

Arguments

addr The kernel address to retrieve the string from

err_msg The error message to return when data isn't available

Description

This function returns a null terminated UTF-8 string converted from the UTF-32 string at a given kernel memory address. Reports the given error message on string copy fault or conversion error.

function::kernel_string_n

function::kernel_string_n — Retrieves string of given length from kernel memory

Synopsis

```
kernel_string_n:string(addr:long,n:long)
```

Arguments

addr The kernel address to retrieve the string from

n The maximum length of the string (if not null terminated)

Description

Returns the C string of a maximum given length from a given kernel memory address. Reports an error on string copy fault.

function::kernel_string_utf16

function::kernel_string_utf16 — Retrieves UTF-16 string from kernel memory

Synopsis

```
kernel_string_utf16:string(addr:long)
```

Arguments

addr The kernel address to retrieve the string from

Description

This function returns a null terminated UTF-8 string converted from the UTF-16 string at a given kernel memory address. Reports an error on string copy fault or conversion error.

function::kernel_string_utf32

function::kernel_string_utf32 — Retrieves UTF-32 string from kernel memory

Synopsis

```
kernel_string_utf32:string(addr:long)
```

Arguments

addr The kernel address to retrieve the string from

Description

This function returns a null terminated UTF-8 string converted from the UTF-32 string at a given kernel memory address. Reports an error on string copy fault or conversion error.

function::user_char

function::user_char — Retrieves a char value stored in user space

Synopsis

```
user_char:long(addr:long)
```

Arguments

addr the user space address to retrieve the char from

Description

Returns the char value from a given user space address. Returns zero when user space data is not accessible.

function::user_char_warn

function::user_char_warn — Retrieves a char value stored in user space

Synopsis

```
user_char_warn:long(addr:long)
```

Arguments

addr the user space address to retrieve the char from

Description

Returns the char value from a given user space address. Returns zero when user space and warns (but does not abort) about the failure.

function::user_int

function::user_int — Retrieves an int value stored in user space

Synopsis

```
user_int:long(addr:long)
```

Arguments

addr the user space address to retrieve the int from

Description

Returns the int value from a given user space address. Returns zero when user space data is not accessible.

function::user_int16

function::user_int16 — Retrieves a 16-bit integer value stored in user space

Synopsis

```
user_int16:long(addr:long)
```

Arguments

addr the user space address to retrieve the 16-bit integer from

Description

Returns the 16-bit integer value from a given user space address. Returns zero when user space data is not accessible.

function::user_int32

function::user_int32 — Retrieves a 32-bit integer value stored in user space

Synopsis

```
user_int32:long(addr:long)
```

Arguments

addr the user space address to retrieve the 32-bit integer from

Description

Returns the 32-bit integer value from a given user space address. Returns zero when user space data is not accessible.

function::user_int64

function::user_int64 — Retrieves a 64-bit integer value stored in user space

Synopsis

```
user_int64:long(addr:long)
```

Arguments

addr the user space address to retrieve the 64-bit integer from

Description

Returns the 64-bit integer value from a given user space address. Returns zero when user space data is not accessible.

function::user_int8

function::user_int8 — Retrieves a 8-bit integer value stored in user space

Synopsis

```
user_int8:long(addr:long)
```

Arguments

addr the user space address to retrieve the 8-bit integer from

Description

Returns the 8-bit integer value from a given user space address. Returns zero when user space data is not accessible.

function::user_int_warn

function::user_int_warn — Retrieves an int value stored in user space

Synopsis

```
user_int_warn:long(addr:long)
```

Arguments

addr the user space address to retrieve the int from

Description

Returns the int value from a given user space address. Returns zero when user space and warns (but does not abort) about the failure.

function::user_long

function::user_long — Retrieves a long value stored in user space

Synopsis

```
user_long:long(addr:long)
```

Arguments

addr the user space address to retrieve the long from

Description

Returns the long value from a given user space address. Returns zero when user space data is not accessible. Note that the size of the long depends on the architecture of the current user space task (for those architectures that support both 64/32 bit compat tasks).

function::user_long_warn

function::user_long_warn — Retrieves a long value stored in user space

Synopsis

```
user_long_warn:long(addr:long)
```

Arguments

addr the user space address to retrieve the long from

Description

Returns the long value from a given user space address. Returns zero when user space and warns (but does not abort) about the failure. Note that the size of the long depends on the architecture of the current user space task (for those architectures that support both 64/32 bit compat tasks).

function::user_short

function::user_short — Retrieves a short value stored in user space

Synopsis

```
user_short:long(addr:long)
```

Arguments

addr the user space address to retrieve the short from

Description

Returns the short value from a given user space address. Returns zero when user space data is not accessible.

function::user_short_warn

function::user_short_warn — Retrieves a short value stored in user space

Synopsis

```
user_short_warn:long(addr:long)
```

Arguments

addr the user space address to retrieve the short from

Description

Returns the short value from a given user space address. Returns zero when user space and warns (but does not abort) about the failure.

function::user_string

function::user_string — Retrieves string from user space

Synopsis

```
user_string:string(addr:long)
```

Arguments

addr the user space address to retrieve the string from

Description

Returns the null terminated C string from a given user space memory address. Reports “<unknown>” on the rare cases when userspace data is not accessible.

function::user_string2

function::user_string2 — Retrieves string from user space with alternative error string

Synopsis

```
user_string2:string(addr:long,err_msg:string)
```

Arguments

addr the user space address to retrieve the string from

err_msg the error message to return when data isn’t available

Description

Returns the null terminated C string from a given user space memory address. Reports the given error message on the rare cases when userspace data is not accessible.

function::user_string2_utf16

function::user_string2_utf16 — Retrieves UTF-16 string from user memory with alternative error string

Synopsis

```
user_string2_utf16:string(addr:long,err_msg:string)
```

Arguments

addr The user address to retrieve the string from

err_msg The error message to return when data isn't available

Description

This function returns a null terminated UTF-8 string converted from the UTF-16 string at a given user memory address. Reports the given error message on string copy fault or conversion error.

function::user_string2_utf32

function::user_string2_utf32 — Retrieves UTF-32 string from user memory with alternative error string

Synopsis

```
user_string2_utf32:string(addr:long,err_msg:string)
```

Arguments

addr The user address to retrieve the string from

err_msg The error message to return when data isn't available

Description

This function returns a null terminated UTF-8 string converted from the UTF-32 string at a given user memory address. Reports the given error message on string copy fault or conversion error.

function::user_string_n

function::user_string_n — Retrieves string of given length from user space

Synopsis

```
user_string_n:string(addr:long,n:long)
```

Arguments

addr the user space address to retrieve the string from

n the maximum length of the string (if not null terminated)

Description

Returns the C string of a maximum given length from a given user space address. Returns “<unknown>” on the rare cases when userspace data is not accessible at the given address.

function::user_string_n2

function::user_string_n2 — Retrieves string of given length from user space

Synopsis

```
user_string_n2:string(addr:long,n:long,err_msg:string)
```

Arguments

addr the user space address to retrieve the string from

n the maximum length of the string (if not null terminated)

err_msg the error message to return when data isn't available

Description

Returns the C string of a maximum given length from a given user space address. Returns the given error message string on the rare cases when userspace data is not accessible at the given address.

function::user_string_n_quoted

function::user_string_n_quoted — Retrieves and quotes string from user space

Synopsis

```
user_string_n_quoted:string(addr:long,n:long)
```

Arguments

addr the user space address to retrieve the string from

n the maximum length of the string (if not null terminated)

Description

Returns up to n characters of a C string from the given user space memory address where any ASCII characters that are not printable are replaced by the corresponding escape sequence in the returned string. Reports “NULL” for address zero. Returns “<unknown>” on the rare cases when userspace data is not accessible at the given address.

function::user_string_n_warn

function::user_string_n_warn — Retrieves string from user space

Synopsis

```
user_string_n_warn:string(addr:long,n:long)
```

Arguments

addr the user space address to retrieve the string from

n the maximum length of the string (if not null terminated)

Description

Returns up to n characters of a C string from a given user space memory address. Reports “<unknown>” on the rare cases when userspace data is not accessible and warns (but does not abort) about the failure.

function::user_string_quoted

function::user_string_quoted — Retrieves and quotes string from user space

Synopsis

```
user_string_quoted:string(addr:long)
```

Arguments

addr the user space address to retrieve the string from

Description

Returns the null terminated C string from a given user space memory address where any ASCII characters that are not printable are replaced by the corresponding escape sequence in the returned string. Reports “NULL” for address zero. Returns “<unknown>” on the rare cases when userspace data is not accessible at the given address.

function::user_string_utf16

function::user_string_utf16 — Retrieves UTF-16 string from user memory

Synopsis

```
user_string_utf16:string(addr:long)
```

Arguments

addr The user address to retrieve the string from

Description

This function returns a null terminated UTF-8 string converted from the UTF-16 string at a given user memory address. Reports an error on string copy fault or conversion error.

function::user_string_utf32

function::user_string_utf32 — Retrieves UTF-32 string from user memory

Synopsis

```
user_string_utf32:string(addr:long)
```

Arguments

addr The user address to retrieve the string from

Description

This function returns a null terminated UTF-8 string converted from the UTF-32 string at a given user memory address. Reports an error on string copy fault or conversion error.

function::user_string_warn

function::user_string_warn — Retrieves string from user space

Synopsis

```
user_string_warn:string(addr:long)
```

Arguments

addr the user space address to retrieve the string from

Description

Returns the null terminated C string from a given user space memory address. Reports “<unknown>” on the rare cases when userspace data is not accessible and warns (but does not abort) about the failure.

function::user_uint16

function::user_uint16 — Retrieves an unsigned 16-bit integer value stored in user space

Synopsis

```
user_uint16:long(addr:long)
```

Arguments

addr the user space address to retrieve the unsigned 16-bit integer from

Description

Returns the unsigned 16-bit integer value from a given user space address. Returns zero when user space data is not accessible.

function::user_uint32

function::user_uint32 — Retrieves an unsigned 32-bit integer value stored in user space

Synopsis

```
user_uint32:long(addr:long)
```

Arguments

addr the user space address to retrieve the unsigned 32-bit integer from

Description

Returns the unsigned 32-bit integer value from a given user space address. Returns zero when user space data is not accessible.

function::user_uint64

function::user_uint64 — Retrieves an unsigned 64-bit integer value stored in user space

Synopsis

```
user_uint64:long(addr:long)
```

Arguments

addr the user space address to retrieve the unsigned 64-bit integer from

Description

Returns the unsigned 64-bit integer value from a given user space address. Returns zero when user space data is not accessible.

function::user_uint8

function::user_uint8 — Retrieves an unsigned 8-bit integer value stored in user space

Synopsis

```
user_uint8:long(addr:long)
```

Arguments

addr the user space address to retrieve the unsigned 8-bit integer from

Description

Returns the unsigned 8-bit integer value from a given user space address. Returns zero when user space data is not accessible.

function::user_ushort

function::user_ushort — Retrieves an unsigned short value stored in user space

Synopsis

```
user_ushort:long(addr:long)
```

Arguments

addr the user space address to retrieve the unsigned short from

Description

Returns the unsigned short value from a given user space address. Returns zero when user space data is not accessible.

function::user_ushort_warn

function::user_ushort_warn — Retrieves an unsigned short value stored in user space

Synopsis

```
user_ushort_warn:long(addr:long)
```

Arguments

addr the user space address to retrieve the unsigned short from

Description

Returns the unsigned short value from a given user space address. Returns zero when user space and warns (but does not abort) about the failure.

Chapter 25. String and data writing functions Tapset

The SystemTap guru mode can be used to test error handling in kernel code by simulating faults. The functions in the this tapset provide standard methods of writing to primitive types in the kernel's memory. All the functions in this tapset require the use of guru mode (**-g**).

function::set_kernel_char

function::set_kernel_char — Writes a char value to kernel memory

Synopsis

```
set_kernel_char(addr:long,val:long)
```

Arguments

addr The kernel address to write the char to

val The char which is to be written

Description

Writes the char value to a given kernel memory address. Reports an error when writing to the given address fails. Requires the use of guru mode (-g).

function::set_kernel_int

function::set_kernel_int — Writes an int value to kernel memory

Synopsis

```
set_kernel_int(addr:long,val:long)
```

Arguments

addr The kernel address to write the int to

val The int which is to be written

Description

Writes the int value to a given kernel memory address. Reports an error when writing to the given address fails. Requires the use of guru mode (-g).

function::set_kernel_long

function::set_kernel_long — Writes a long value to kernel memory

Synopsis

```
set_kernel_long(addr:long,val:long)
```

Arguments

addr The kernel address to write the long to

val The long which is to be written

Description

Writes the long value to a given kernel memory address. Reports an error when writing to the given address fails. Requires the use of guru mode (-g).

function::set_kernel_pointer

function::set_kernel_pointer — Writes a pointer value to kernel memory.

Synopsis

```
set_kernel_pointer(addr:long,val:long)
```

Arguments

addr The kernel address to write the pointer to

val The pointer which is to be written

Description

Writes the pointer value to a given kernel memory address. Reports an error when writing to the given address fails. Requires the use of guru mode (-g).

function::set_kernel_short

function::set_kernel_short — Writes a short value to kernel memory

Synopsis

```
set_kernel_short(addr:long,val:long)
```

Arguments

addr The kernel address to write the short to

val The short which is to be written

Description

Writes the short value to a given kernel memory address. Reports an error when writing to the given address fails. Requires the use of guru mode (-g).

function::set_kernel_string

function::set_kernel_string — Writes a string to kernel memory

Synopsis

```
set_kernel_string(addr:long,val:string)
```

Arguments

addr The kernel address to write the string to

val The string which is to be written

Description

Writes the given string to a given kernel memory address. Reports an error on string copy fault. Requires the use of guru mode (-g).

function::set_kernel_string_n

function::set_kernel_string_n — Writes a string of given length to kernel memory

Synopsis

```
set_kernel_string_n(addr:long,n:long,val:string)
```

Arguments

addr The kernel address to write the string to

n The maximum length of the string

val The string which is to be written

Description

Writes the given string up to a maximum given length to a given kernel memory address. Reports an error on string copy fault. Requires the use of guru mode (-g).

Chapter 26. Guru tapsets

Functions to deliberately interfere with the system's behavior, in order to inject faults or improve observability. All the functions in this tapset require the use of guru mode (**-g**).

function::mdelay

function::mdelay — millisecond delay

Synopsis

```
mdelay(ms:long)
```

Arguments

ms Number of milliseconds to delay.

Description

This function inserts a multi-millisecond busy-delay into a probe handler. It requires guru mode.

function::panic

function::panic — trigger a panic

Synopsis

```
panic(msg:string)
```

Arguments

msg message to pass to kernel's `panic` function

Description

This function triggers an immediate panic of the running kernel with a user-specified panic message. It requires guru mode.

function::udelay

function::udelay — microsecond delay

Synopsis

```
udelay(us:long)
```

Arguments

us Number of microseconds to delay.

Description

This function inserts a multi-microsecond busy-delay into a probe handler. It requires guru mode.

Chapter 27. A collection of standard string functions

Functions to get the length, a substring, getting at individual characters, string seaching, escaping, tokenizing, and converting strings to longs.

function::isdigit

function::isdigit — Checks for a digit

Synopsis

```
isdigit:long(str:string)
```

Arguments

str string to check

Description

Checks for a digit (0 through 9) as the first character of a string. Returns non-zero if true, and a zero if false.

function::isinstr

function::isinstr — Returns whether a string is a substring of another string

Synopsis

```
isinstr:long(s1:string,s2:string)
```

Arguments

s1
string to search in

s2
substring to find

Description

This function returns 1 if string *s1* contains *s2*, otherwise zero.

function::str_replace

function::str_replace — str_replace Replaces all instances of a substring with another

Synopsis

```
str_replace:string(prnt_str:string,srch_str:string,rplc_str:string)
```

Arguments

prnt_str
: the string to search and replace in

srch_str
: the substring which is used to search in *prnt_str* string

rplc_str
: the substring which is used to replace *srch_str*

Description

This function returns the given string with substrings replaced.

function::stringat

function::stringat — Returns the char at a given position in the string

Synopsis

```
stringat:long(str:string,pos:long)
```

Arguments

str the string to fetch the character from

pos the position to get the character from (first character is 0)

Description

This function returns the character at a given position in the string or zero if thestring doesn't have as many characters.

function::strlen

function::strlen — Returns the length of a string

Synopsis

```
strlen:long(s:string)
```

Arguments

s the string

Description

This function returns the length of the string, which can be zero up to MAXSTRINGLEN.

function::strtol

function::strtol — strtol - Convert a string to a long

Synopsis

```
strtol:long(str:string,base:long)
```

Arguments

str string to convert

base the base to use

Description

This function converts the string representation of a number to an integer. The *base* parameter indicates the number base to assume for the string (eg. 16 for hex, 8 for octal, 2 for binary).

function::substr

function::substr — Returns a substring

Synopsis

```
substr:string(str:string,start:long,length:long)
```

Arguments

str	the string to take a substring from
start	starting position of the extracted string (first character is 0)
length	length of string to return

Description

Returns the substring of the given string at the given start position with the given length (or smaller if the length of the original string is less than start + length, or length is bigger than MAXSTRINGLEN).

function::text_str

function::text_str — Escape any non-printable chars in a string

Synopsis

```
text_str:string(input:string)
```

Arguments

input the string to escape

Description

This function accepts a string argument, and any ASCII characters that are not printable are replaced by the corresponding escape sequence in the returned string.

function::text_strn

function::text_strn — Escape any non-printable chars in a string

Synopsis

```
text_strn:string(input:string,len:long,quoted:long)
```

Arguments

input the string to escape

len maximum length of string to return (0 implies MAXSTRINGLEN)

quoted put double quotes around the string. If input string is truncated it will have "..." after the second quote

Description

This function accepts a string of designated length, and any ASCII characters that are not printable are replaced by the corresponding escape sequence in the returned string.

function::tokenize

function::tokenize — Return the next non-empty token in a string

Synopsis

```
tokenize:string(input:string,delim:string)
```

Arguments

input string to tokenize. If empty, returns the next non-empty token in the string passed in the previous call to `tokenize`.

delim set of characters that delimit the tokens

Description

This function returns the next non-empty token in the given input string, where the tokens are delimited by characters in the delim string. If the input string is non-empty, it returns the first token. If the input string is empty, it returns the next token in the string passed in the previous call to tokenize. If no delimiter is found, the entire remaining input string is returned. It returns empty when no more tokens are available.

Chapter 28. Utility functions for using ansi control chars in logs

Utility functions for logging using ansi control characters. This lets you manipulate the cursor position and character color output and attributes of log messages.

function::ansi_clear_screen

function::ansi_clear_screen — Move cursor to top left and clear screen.

Synopsis

```
ansi_clear_screen()
```

Arguments

None

Description

Sends ansi code for moving cursor to top left and then the ansi code for clearing the screen from the cursor position to the end.

function::ansi_cursor_hide

function::ansi_cursor_hide — Hides the cursor.

Synopsis

```
ansi_cursor_hide()
```

Arguments

None

Description

Sends ansi code for hiding the cursor.

function::ansi_cursor_move

function::ansi_cursor_move — Move cursor to new coordinates.

Synopsis

```
ansi_cursor_move(x:long,y:long)
```

Arguments

x Row to move the cursor to.

y Colomn to move the cursor to.

Description

Sends ansi code for positioning the cursor at row x and column y. Coordinates start at one, (1,1) is the top-left corner.

function::ansi_cursor_restore

function::ansi_cursor_restore — Restores a previously saved cursor position.

Synopsis

```
ansi_cursor_restore()
```

Arguments

None

Description

Sends ansi code for restoring the current cursor position previously saved with `ansi_cursor_save`.

function::ansi_cursor_save

function::ansi_cursor_save — Saves the cursor position.

Synopsis

```
ansi_cursor_save()
```

Arguments

None

Description

Sends ansi code for saving the current cursor position.

function::ansi_cursor_show

function::ansi_cursor_show — Shows the cursor.

Synopsis

```
ansi_cursor_show()
```

Arguments

None

Description

Sends ansi code for showing the cursor.

function::ansi_new_line

function::ansi_new_line — Move cursor to new line.

Synopsis

```
ansi_new_line()
```

Arguments

None

Description

Sends ansi code new line.

function::ansi_reset_color

function::ansi_reset_color — Resets Select Graphic Rendition mode.

Synopsis

```
ansi_reset_color()
```

Arguments

None

Description

Sends ansi code to reset foreground, background and color attribute to default values.

function::ansi_set_color

function::ansi_set_color — Set the ansi Select Graphic Rendition mode.

Synopsis

```
ansi_set_color(fg:long)
```

Arguments

fg Foreground color to set.

Description

Sends ansi code for Select Graphic Rendition mode for the given forground color. Black (30), Blue (34), Green (32), Cyan (36), Red (31), Purple (35), Brown (33), Light Gray (37).

function::ansi_set_color2

function::ansi_set_color2 — Set the ansi Select Graphic Rendition mode.

Synopsis

```
ansi_set_color2(fg:long,bg:long)
```

Arguments

fg Foreground color to set.

bg Background color to set.

Description

Sends ansi code for Select Graphic Rendition mode for the given forground color, Black (30), Blue (34), Green (32), Cyan (36), Red (31), Purple (35), Brown (33), Light Gray (37) and the given background color, Black (40), Red (41), Green (42), Yellow (43), Blue (44), Magenta (45), Cyan (46), White (47).

function::ansi_set_color3

function::ansi_set_color3 — Set the ansi Select Graphic Rendition mode.

Synopsis

```
ansi_set_color3(fg:long,bg:long,attr:long)
```

Arguments

fg Foreground color to set.

bg Background color to set.

attr Color attribute to set.

Description

Sends ansi code for Select Graphic Rendition mode for the given forground color, Black (30), Blue (34), Green (32), Cyan (36), Red (31), Purple (35), Brown (33), Light Gray (37), the given background color, Black (40), Red (41), Green (42), Yellow (43), Blue (44), Magenta (45), Cyan (46), White (47) and the color attribute All attributes off (0), Intensity Bold (1), Underline Single (4), Blink Slow (5), Blink Rapid (6), Image Negative (7).

function::indent

function::indent — returns an amount of space to indent

Synopsis

```
indent:string(delta:long)
```

Arguments

delta the amount of space added/removed for each call

Description

This function returns a string with appropriate indentation. Call it with a small positive or matching negative delta. Unlike the thread_indent function, the indent does not track individual indent values on a per thread basis.

function::thread_indent

function::thread_indent — returns an amount of space with the current task information

Synopsis

```
thread_indent:string(delta:long)
```

Arguments

delta the amount of space added/removed for each call

Description

This function returns a string with appropriate indentation for a thread. Call it with a small positive or matching negative delta. If this is the real outermost, initial level of indentation, then the function resets the relative timestamp base to zero. An example is shown at the end of this file.

Chapter 29. SystemTap Translator Tapset

This family of user-space probe points is used to probe the operation of the SystemTap translator (**stap**) and run command (**staprun**). The tapset includes probes to watch the various phases of SystemTap and SystemTap's management of instrumentation cache. It contains the following probe points:

probe::stap.cache_add_mod

probe::stap.cache_add_mod — Adding kernel instrumentation module to cache

Synopsis

```
stap.cache_add_mod
```

Values

dest_path the path the .ko file is going to (incl filename)

source_path the path the .ko file is coming from (incl filename)

Description

Fires just before the file is actually moved. Note: if moving fails, cache_add_src and cache_add_nss will not fire.

probe::stap.cache_add_nss

probe::stap.cache_add_nss — Add NSS (Network Security Services) information to cache

Synopsis

```
stap.cache_add_nss
```

Values

dest_path the path the .sgn file is coming from (incl filename)

source_path the path the .sgn file is coming from (incl filename)

Description

Fires just before the file is actually moved. Note: stap must compiled with NSS support; if moving the kernel module fails, this probe will not fire.

probe::stap.cache_add_src

probe::stap.cache_add_src — Adding C code translation to cache

Synopsis

```
stap.cache_add_src
```

Values

dest_path	the path the .c file is going to (incl filename)
source_path	the path the .c file is coming from (incl filename)

Description

Fires just before the file is actually moved. Note: if moving the kernel module fails, this probe will not fire.

probe::stap.cache_clean

probe::stap.cache_clean — Removing file from stap cache

Synopsis

```
stap.cache_clean
```

Values

path the path to the .ko/.c file being removed

Description

Fires just before the call to unlink the module/source file.

probe::stap.cache_get

probe::stap.cache_get — Found item in stap cache

Synopsis

```
stap.cache_get
```

Values

source_path	the path of the .c source file
module_path	the path of the .ko kernel module file

Description

Fires just before the return of get_from_cache, when the cache grab is successful.

probe::stap.pass0

probe::stap.pass0 — Starting stap pass0 (parsing command line arguments)

Synopsis

```
stap.pass0
```

Values

session the systemtap_session variable s

Description

pass0 fires after command line arguments have been parsed.

probe::stap.pass0.end

probe::stap.pass0.end — Finished stap pass0 (parsing command line arguments)

Synopsis

```
stap.pass0.end
```

Values

session the systemtap_session variable s

Description

pass0.end fires just before the `gettimeofday` call for pass1.

probe::stap.pass1.end

probe::stap.pass1.end — Finished stap pass1 (parsing scripts)

Synopsis

```
stap.pass1.end
```

Values

session the systemtap_session variable s

Description

pass1.end fires just before the jump to cleanup if s.last_pass = 1.

probe::stap.pass1a

probe::stap.pass1a — Starting stap pass1 (parsing user script)

Synopsis

```
stap.pass1a
```

Values

session the systemtap_session variable s

Description

pass1a fires just after the call to `gettimeofday`, before the user script is parsed.

probe::stap.pass1b

probe::stap.pass1b — Starting stap pass1 (parsing library scripts)

Synopsis

```
stap.pass1b
```

Values

session the systemtap_session variable s

Description

pass1b fires just before the library scripts are parsed.

probe::stap.pass2

probe::stap.pass2 — Starting stap pass2 (elaboration)

Synopsis

```
stap.pass2
```

Values

session the systemtap_session variable s

Description

pass2 fires just after the call to `gettimeofday`, just before the call to semantic_pass.

probe::stap.pass2.end

probe::stap.pass2.end — Finished stap pass2 (elaboration)

Synopsis

```
stap.pass2.end
```

Values

session the systemtap_session variable s

Description

pass2.end fires just before the jump to cleanup if s.last_pass = 2

probe::stap.pass3

probe::stap.pass3 — Starting stap pass3 (translation to C)

Synopsis

```
stap.pass3
```

Values

session the systemtap_session variable s

Description

pass3 fires just after the call to `gettimeofday`, just before the call to translate_pass.

probe::stap.pass3.end

probe::stap.pass3.end — Finished stap pass3 (translation to C)

Synopsis

```
stap.pass3.end
```

Values

session the systemtap_session variable s

Description

pass3.end fires just before the jump to cleanup if s.last_pass = 3

probe::stap.pass4

probe::stap.pass4 — Starting stap pass4 (compile C code into kernel module)

Synopsis

```
stap.pass4
```

Values

session the systemtap_session variable s

Description

pass4 fires just after the call to `gettimeofday`, just before the call to compile_pass.

probe::stap.pass4.end

probe::stap.pass4.end — Finished stap pass4 (compile C code into kernel module)

Synopsis

```
stap.pass4.end
```

Values

session the systemtap_session variable s

Description

pass4.end fires just before the jump to cleanup if s.last_pass = 4

probe::stap.pass5

probe::stap.pass5 — Starting stap pass5 (running the instrumentation)

Synopsis

```
stap.pass5
```

Values

session the systemtap_session variable s

Description

pass5 fires just after the call to `gettimeofday`, just before the call to run_pass.

probe::stap.pass5.end

probe::stap.pass5.end — Finished stap pass5 (running the instrumentation)

Synopsis

```
stap.pass5.end
```

Values

session the systemtap_session variable s

Description

pass5.end fires just before the cleanup label

probe::stap.pass6

probe::stap.pass6 — Starting stap pass6 (cleanup)

Synopsis

```
stap.pass6
```

Values

session the systemtap_session variable s

Description

pass6 fires just after the cleanup label, essentially the same spot as pass5.end

probe::stap.pass6.end

probe::stap.pass6.end — Finished stap pass6 (cleanup)

Synopsis

```
stap.pass6.end
```

Values

session the systemtap_session variable s

Description

pass6.end fires just before main's return.

probe::stap.system

probe::stap.system — Starting a command from stap

Synopsis

```
stap.system
```

Values

command the command string to be run by posix_spawn (as sh -c <str>)

Description

Fires at the entry of the stap_system command.

probe::stap.system.return

probe::stap.system.return — Finished a command from stap

Synopsis

```
stap.system.return
```

Values

ret a return code associated with running waitpid on the spawned process; a non-zero value indicates error

Description

Fires just before the return of the stap_system function, after waitpid.

probe::stap.system.spawn

probe::stap.system.spawn — stap spawned new process

Synopsis

```
stap.system.spawn
```

Values

ret the return value from posix_spawn

pid the pid of the spawned process

Description

Fires just after the call to posix_spawn.

probe::stapio.receive_control_message

probe::stapio.receive_control_message — Received a control message

Synopsis

```
stapio.receive_control_message
```

Values

len the length (in bytes) of the data blob

data a ptr to a binary blob of data sent as the control message

type type of message being send; defined in runtime/transport/transport_msgs.h

Description

Fires just after a message was receieved and before it's processed.

probe::staprun.insert_module

probe::staprun.insert_module — Inserting SystemTap instrumentation module

Synopsis

```
staprun.insert_module
```

Values

path the full path to the .ko kernel module about to be inserted

Description

Fires just before the call to insert the module.

probe::staprun.remove_module

probe::staprun.remove_module — Removing SystemTap instrumentation module

Synopsis

```
staprun.remove_module
```

Values

name the stap module name to be removed (without the .ko extension)

Description

Fires just before the call to remove the module.

probe::staprun.send_control_message

probe::staprun.send_control_message — Sending a control message

Synopsis

```
staprun.send_control_message
```

Values

len the length (in bytes) of the data blob

data a ptr to a binary blob of data sent as the control message

type type of message being send; defined in runtime/transport/transport_msgs.h

Description

Fires at the beginning of the send_request function.

Chapter 30. Network File Storage Tapsets

This family of probe points is used to probe network file storage functions and operations.

function::nfsderror

function::nfsderror — Convert nfsd error number into string

Synopsis

```
nfsderror:string(err:long)
```

Arguments

err   errnum

Description

This function returns a string for the error number passed into the function.

probe::nfs.aop.readpage

probe::nfs.aop.readpage — NFS client synchronously reading a page

Synopsis

```
nfs.aop.readpage
```

Values

i_size
: file length in bytes

dev
: device identifier

rsize
: read size (in bytes)

sb_flag
: super block flags

file
: file argument

page_index
: offset within mapping, can used a page identifier and position identifier in the page frame

__page
: the address of page

size
: number of pages to be read in this execution

i_flag
: file flags

ino
: inode number

Description

Read the page over, only fires when a previous async read operation failed

probe::nfs.aop.readpages

probe::nfs.aop.readpages — NFS client reading multiple pages

Synopsis

```
nfs.aop.readpages
```

Values

dev	device identifier
rsize	read size (in bytes)
file	filp argument
size	number of pages attempted to read in this execution
nr_pages	number of pages attempted to read in this execution
rpages	read size (in pages)
ino	inode number

Description

Fires when in readahead way, read several pages once

probe::nfs.aop.release_page

probe::nfs.aop.release_page — NFS client releasing page

Synopsis

```
nfs.aop.release_page
```

Values

dev device identifier

page_index offset within mapping, can used a page identifier and position identifier in the page frame

__page the address of page

size release pages

ino inode number

Description

Fires when do a release operation on NFS.

probe::nfs.aop.set_page_dirty

probe::nfs.aop.set_page_dirty — NFS client marking page as dirty

Synopsis

```
nfs.aop.set_page_dirty
```

Values

__page	the address of page
page_flag	page flags

Description

This probe attaches to the generic __set_page_dirty_nobuffers function. Thus, this probe is going to fire on many other file systems in addition to the NFS client.

probe::nfs.aop.write_begin

probe::nfs.aop.write_begin — NFS client begin to write data

Synopsis

```
nfs.aop.write_begin
```

Values

dev device identifier

page_index offset within mapping, can used a page identifier and position identifier in the page frame

__page the address of page

size write bytes

to end address of this write operation

ino inode number

offset start address of this write operation

Description

Occurs when write operation occurs on nfs. It prepare a page for writing, look for a request corresponding to the page. If there is one, and it belongs to another file, it flush it out before it tries to copy anything into the page. Also do the same if it finds a request from an existing dropped page

probe::nfs.aop.write_end

probe::nfs.aop.write_end — NFS client complete writing data

Synopsis

```
nfs.aop.write_end
```

Values

i_size
: file length in bytes

dev
: device identifier

sb_flag
: super block flags

page_index
: offset within mapping, can used a page identifier and position identifier in the page frame

__page
: the address of page

size
: write bytes

i_flag
: file flags

to
: end address of this write operation

ino
: inode number

offset
: start address of this write operation

Description

Fires when do a write operation on nfs, often after prepare_write

Update and possibly write a cached page of an NFS file.

probe::nfs.aop.writepage

probe::nfs.aop.writepage — NFS client writing a mapped page to the NFS server

Synopsis

```
nfs.aop.writepage
```

Values

for_reclaim a flag of writeback_control, indicates if it's invoked from the page allocator

i_size file length in bytes

dev device identifier

sb_flag super block flags

page_index offset within mapping, can used a page identifier and position identifier in the page frame

__page the address of page

size number of pages to be written in this execution

for_kupdate a flag of writeback_control, indicates if it's a kupdate writeback

wsize write size

i_flag file flags

i_state inode state flags

ino inode number

Description

The priority of wb is decided by the flags *for_reclaim* and *for_kupdate*.

probe::nfs.aop.writepages

probe::nfs.aop.writepages — NFS client writing several dirty pages to the NFS server

Synopsis

```
nfs.aop.writepages
```

Values

for_reclaim	a flag of writeback_control, indicates if it's invoked from the page allocator
dev	device identifier
wpages	write size (in pages)
size	number of pages attempted to be written in this execution
for_kupdate	a flag of writeback_control, indicates if it's a kupdate writeback
wsize	write size
nr_to_write	number of pages attempted to be written in this execution
ino	inode number

Description

The priority of wb is decided by the flags *for_reclaim* and *for_kupdate*.

probe::nfs.fop.aio_read

probe::nfs.fop.aio_read — NFS client aio_read file operation

Synopsis

```
nfs.fop.aio_read
```

Values

attrtimeo
: how long the cached information is assumed to be valid. We need to revalidate the cached attrs for this inode if jiffies - read_cache_jiffies > attrtimeo.

cache_valid
: cache related bit mask flag

count
: read bytes

parent_name
: parent dir name

dev
: device identifier

buf
: the address of buf in user space

cache_time
: when we started read-caching this inode

file_name
: file name

pos
: current position of file

ino
: inode number

probe::nfs.fop.aio_write

probe::nfs.fop.aio_write — NFS client aio_write file operation

Synopsis

```
nfs.fop.aio_write
```

Values

count	read bytes
parent_name	parent dir name
dev	device identifier
buf	the address of buf in user space
file_name	file name
pos	offset of the file
ino	inode number

probe::nfs.fop.check_flags

probe::nfs.fop.check_flags — NFS client checking flag operation

Synopsis

```
nfs.fop.check_flags
```

Values

flag file flag

probe::nfs.fop.flush

probe::nfs.fop.flush — NFS client flush file operation

Synopsis

```
nfs.fop.flush
```

Values

dev	device identifier
mode	file mode
ndirty	number of dirty page
ino	inode number

probe::nfs.fop.fsync

probe::nfs.fop.fsync — NFS client fsync operation

Synopsis

```
nfs.fop.fsync
```

Values

dev	device identifier
ndirty	number of dirty pages
ino	inode number

probe::nfs.fop.llseek

probe::nfs.fop.llseek — NFS client llseek operation

Synopsis

```
nfs.fop.llseek
```

Values

dev device identifier

origin the original position. The possible value could be: SEEK_SET (offset set to offset bytes), SEEK_CUR (offset set to its current location plus offset bytes), or SEEK_END (offset set to the size of the file plus offset bytes).

ino inode number

offset the offset of the file will be repositioned

probe::nfs.fop.lock

probe::nfs.fop.lock — NFS client file lock operation

Synopsis

```
nfs.fop.lock
```

Values

cmd	cmd arguments
dev	device identifier
fl_type	lock type
fl_end	ending offset of locked region
fl_flag	lock flags
i_mode	file type and access rights
fl_start	starting offset of locked region
ino	inode number

probe::nfs.fop.mmap

probe::nfs.fop.mmap — NFS client mmap operation

Synopsis

```
nfs.fop.mmap
```

Values

attrtimeo
: how long the cached information is assumed to be valid. We need to revalidate the cached attrs for this inode if jiffies - read_cache_jiffies > attrtimeo.

cache_valid
: cache related bit mask flag

parent_name
: parent dir name

vm_flag
: vm flags

vm_start
: start address within vm_mm

dev
: device identifier

buf
: the address of buf in user space

vm_end
: the first byte after end address within vm_mm

cache_time
: when we started read-caching this inode

file_name
: file name

ino
: inode number

probe::nfs.fop.open

probe::nfs.fop.open — NFS client file open operation

Synopsis

```
nfs.fop.open
```

Values

i_size	file length in bytes
dev	device identifier
flag	file flag
file_name	file name
ino	inode number

probe::nfs.fop.read

probe::nfs.fop.read — NFS client read operation

Synopsis

```
nfs.fop.read
```

Values

devname block device name

Description

SystemTap uses the vfs.do_sync_read probe to implement this probe and as a result will get operations other than the NFS client read operations.

probe::nfs.fop.release

probe::nfs.fop.release — NFS client release page operation

Synopsis

```
nfs.fop.release
```

Values

dev device identifier

mode file mode

ino inode number

probe::nfs.fop.sendfile

probe::nfs.fop.sendfile — NFS client send file operation

Synopsis

```
nfs.fop.sendfile
```

Values

attrtimeo how long the cached information is assumed to be valid. We need to revalidate the cached attrs for this inode if jiffies - read_cache_jiffies > attrtimeo.

cache_valid cache related bit mask flag

count read bytes

ppos current position of file

dev device identifier

cache_time when we started read-caching this inode

ino inode number

probe::nfs.fop.write

probe::nfs.fop.write — NFS client write operation

Synopsis

```
nfs.fop.write
```

Values

devname block device name

Description

SystemTap uses the vfs.do_sync_write probe to implement this probe and as a result will get operations other than the NFS client write operations.

probe::nfs.proc.commit

probe::nfs.proc.commit — NFS client committing data on server

Synopsis

```
nfs.proc.commit
```

Values

bitmask1	V4 bitmask representing the set of attributes supported on this filesystem
version	NFS version
bitmask0	V4 bitmask representing the set of attributes supported on this filesystem
prot	transfer protocol
size	read bytes in this execution
offset	the file offset
server_ip	IP address of server

Description

All the nfs.proc.commit kernel functions were removed in kernel commit 200baa in December 2006, so these probes do not exist on Linux 2.6.21 and newer kernels.

Fires when client writes the buffered data to disk. The buffered data is asynchronously written by client earlier. The commit function works in sync way. This probe point does not exist in NFSv2.

probe::nfs.proc.commit_done

probe::nfs.proc.commit_done — NFS client response to a commit RPC task

Synopsis

```
nfs.proc.commit_done
```

Values

count	number of bytes committed
status	result of last operation
version	NFS version
prot	transfer protocol
valid	fattr->valid ,indicates which fields are valid
timestamp	V4 timestamp, which is used for lease renewal
server_ip	IP address of server

Description

Fires when a reply to a commit RPC task is received or some commit operation error occur (timeout or socket shutdown).

probe::nfs.proc.commit_setup

probe::nfs.proc.commit_setup — NFS client setting up a commit RPC task

Synopsis

```
nfs.proc.commit_setup
```

Values

count	bytes in this commit
bitmask1	V4 bitmask representing the set of attributes supported on this filesystem
version	NFS version
bitmask0	V4 bitmask representing the set of attributes supported on this filesystem
prot	transfer protocol
size	bytes in this commit
offset	the file offset
server_ip	IP address of server

Description

The commit_setup function is used to setup a commit RPC task. Is is not doing the actual commit operation. It is does not exist in NFSv2.

probe::nfs.proc.create

probe::nfs.proc.create — NFS client creating file on server

Synopsis

```
nfs.proc.create
```

Values

version	NFS version (the function is used for all NFS version)
flag	indicates create mode (only for NFSv3 and NFSv4)
prot	transfer protocol
filelen	length of file name
filename	file name
fh	file handler of parent dir
server_ip	IP address of server

probe::nfs.proc.handle_exception

probe::nfs.proc.handle_exception — NFS client handling an NFSv4 exception

Synopsis

```
nfs.proc.handle_exception
```

Values

errorcode	indicates the type of error

Description

This is the error handling routine for processes for NFSv4.

probe::nfs.proc.lookup

probe::nfs.proc.lookup — NFS client opens/searches a file on server

Synopsis

```
nfs.proc.lookup
```

Values

name_len the length of file name

filename the name of file which client opens/searches on server

bitmask1 V4 bitmask representing the set of attributes supported on this filesystem

version NFS version

bitmask0 V4 bitmask representing the set of attributes supported on this filesystem

prot transfer protocol

server_ip IP address of server

probe::nfs.proc.open

probe::nfs.proc.open — NFS client allocates file read/write context information

Synopsis

```
nfs.proc.open
```

Values

mode	file mode
version	NFS version (the function is used for all NFS version)
flag	file flag
prot	transfer protocol
filename	file name
server_ip	IP address of server

Description

Allocate file read/write context information

probe::nfs.proc.read

probe::nfs.proc.read — NFS client synchronously reads file from server

Synopsis

```
nfs.proc.read
```

Values

count	read bytes in this execution
flags	used to set task->tk_flags in rpc_init_task function
version	NFS version
prot	transfer protocol
offset	the file offset
server_ip	IP address of server

Description

All the nfs.proc.read kernel functions were removed in kernel commit 8e0969 in December 2006, so these probes do not exist on Linux 2.6.21 and newer kernels.

probe::nfs.proc.read_done

probe::nfs.proc.read_done — NFS client response to a read RPC task

Synopsis

```
nfs.proc.read_done
```

Values

count	number of bytes read
status	result of last operation
version	NFS version
prot	transfer protocol
timestamp	V4 timestamp, which is used for lease renewal
server_ip	IP address of server

Description

Fires when a reply to a read RPC task is received or some read error occurs (timeout or socket shutdown).

probe::nfs.proc.read_setup

probe::nfs.proc.read_setup — NFS client setting up a read RPC task

Synopsis

```
nfs.proc.read_setup
```

Values

count	read bytes in this execution
version	NFS version
prot	transfer protocol
size	read bytes in this execution
offset	the file offset
server_ip	IP address of server

Description

The read_setup function is used to setup a read RPC task. It is not doing the actual read operation.

probe::nfs.proc.release

probe::nfs.proc.release — NFS client releases file read/write context information

Synopsis

```
nfs.proc.release
```

Values

mode	file mode
version	NFS version (the function is used for all NFS version)
flag	file flag
prot	transfer protocol
filename	file name
server_ip	IP address of server

Description

Release file read/write context information

probe::nfs.proc.remove

probe::nfs.proc.remove — NFS client removes a file on server

Synopsis

```
nfs.proc.remove
```

Values

version	NFS version (the function is used for all NFS version)
prot	transfer protocol
filelen	length of file name
filename	file name
fh	file handler of parent dir
server_ip	IP address of server

probe::nfs.proc.rename

probe::nfs.proc.rename — NFS client renames a file on server

Synopsis

```
nfs.proc.rename
```

Values

new_fh	file handler of new parent dir
old_filelen	length of old file name
version	NFS version (the function is used for all NFS version)
prot	transfer protocol
new_filelen	length of new file name
old_fh	file handler of old parent dir
new_name	new file name
old_name	old file name
server_ip	IP address of server

probe::nfs.proc.write

probe::nfs.proc.write — NFS client synchronously writes file to server

Synopsis

```
nfs.proc.write
```

Values

flags	used to set task->tk_flags in rpc_init_task function
bitmask1	V4 bitmask representing the set of attributes supported on this filesystem
version	NFS version
bitmask0	V4 bitmask representing the set of attributes supported on this filesystem
prot	transfer protocol
size	read bytes in this execution
server_ip	IP address of server
offset	the file offset

Description

All the nfs.proc.write kernel functions were removed in kernel commit 200baa in December 2006, so these probes do not exist on Linux 2.6.21 and newer kernels.

probe::nfs.proc.write_done

probe::nfs.proc.write_done — NFS client response to a write RPC task

Synopsis

```
nfs.proc.write_done
```

Values

count	number of bytes written
status	result of last operation
version	NFS version
prot	transfer protocol
valid	fattr->valid ,indicates which fields are valid
timestamp	V4 timestamp, which is used for lease renewal
server_ip	IP address of server

Description

Fires when a reply to a write RPC task is received or some write error occurs (timeout or socket shutdown).

probe::nfs.proc.write_setup

probe::nfs.proc.write_setup — NFS client setting up a write RPC task

Synopsis

```
nfs.proc.write_setup
```

Values

count bytes written in this execution

bitmask1 V4 bitmask representing the set of attributes supported on this filesystem

how used to set args.stable. The stable value could be: NFS_UNSTABLE,NFS_DATA_SYNC,NFS_FI
(in nfs.proc3.write_setup and nfs.proc4.write_setup)

version NFS version

bitmask0 V4 bitmask representing the set of attributes supported on this filesystem

prot transfer protocol

size bytes written in this execution

offset the file offset

server_ip IP address of server

Description

The write_setup function is used to setup a write RPC task. It is not doing the actual write operation.

probe::nfsd.close

probe::nfsd.close — NFS server closing a file for client

Synopsis

```
nfsd.close
```

Values

filename	file name

probe::nfsd.commit

probe::nfsd.commit — NFS server committing all pending writes to stable storage

Synopsis

```
nfsd.commit
```

Values

count	read bytes
flag	indicates whether this execution is a sync operation
size	read bytes
fh	file handle (the first part is the length of the file handle)
client_ip	the ip address of client
offset	the offset of file

probe::nfsd.create

probe::nfsd.create — NFS server creating a file(regular,dir,device,fifo) for client

Synopsis

```
nfsd.create
```

Values

iap_mode
: file access mode

iap_valid
: Attribute flags

filelen
: the length of file name

filename
: file name

fh
: file handle (the first part is the length of the file handle)

client_ip
: the ip address of client

type
: file type(regular,dir,device,fifo ...)

Description

Sometimes nfsd will call nfsd_create_v3 instead of this this probe point.

probe::nfsd.createv3

probe::nfsd.createv3 — NFS server creating a regular file or set file attributes for client

Synopsis

```
nfsd.createv3
```

Values

iap_mode
: file access mode

createmode
: create mode .The possible values could be: NFS3_CREATE_EXCLUSIVE, NFS3_CREATE_UNCHECKED, or NFS3_CREATE_GUARDED

verifier
: file attributes (atime,mtime,mode). It's used to reset file attributes for CREATE_EXCLUSIVE

iap_valid
: Attribute flags

truncp
: trunp arguments, indicates if the file shouldbe truncate

filelen
: the length of file name

filename
: file name

fh
: file handle (the first part is the length of the file handle)

client_ip
: the ip address of client

Description

This probepoints is only called by nfsd3_proc_create and nfsd4_open when op_claim_type is NFS4_OPEN_CLAIM_NULL.

probe::nfsd.dispatch

probe::nfsd.dispatch — NFS server receives an operation from client

Synopsis

```
nfsd.dispatch
```

Values

proto	transfer protocol
proc	procedure number
prog	program number
version	nfs version
client_ip	the ip address of client
xid	transmission id

probe::nfsd.lookup

probe::nfsd.lookup — NFS server opening or searching file for a file for client

Synopsis

```
nfsd.lookup
```

Values

filelen	the length of file name
filename	file name
fh	file handle of parent dir(the first part is the length of the file handle)
client_ip	the ip address of client

probe::nfsd.open

probe::nfsd.open — NFS server opening a file for client

Synopsis

```
nfsd.open
```

Values

access	indicates the type of open (read/write/commit/readdir...)
fh	file handle (the first part is the length of the file handle)
client_ip	the ip address of client
type	type of file (regular file or dir)

probe::nfsd.proc.commit

probe::nfsd.proc.commit — NFS server performing a commit operation for client

Synopsis

```
nfsd.proc.commit
```

Values

count	read bytes
proto	transfer protocol
version	nfs version
size	read bytes
fh	file handle (the first part is the length of the file handle)
client_ip	the ip address of client
offset	the offset of file

probe::nfsd.proc.create

probe::nfsd.proc.create — NFS server creating a file for client

Synopsis

```
nfsd.proc.create
```

Values

proto	transfer protocol
version	nfs version
filelen	length of file name
filename	file name
fh	file handle (the first part is the length of the file handle)
client_ip	the ip address of client

probe::nfsd.proc.lookup

probe::nfsd.proc.lookup — NFS server opening or searching for a file for client

Synopsis

```
nfsd.proc.lookup
```

Values

proto	transfer protocol
filelen	the length of file name
filename	file name
fh	file handle of parent dir (the first part is the length of the file handle)
version	nfs version
client_ip	the ip address of client

probe::nfsd.proc.read

probe::nfsd.proc.read — NFS server reading file for client

Synopsis

```
nfsd.proc.read
```

Values

vec	struct kvec, includes buf address in kernel address and length of each buffer
count	read bytes
proto	transfer protocol
fh	file handle (the first part is the length of the file handle)
version	nfs version
client_ip	the ip address of client
vlen	read blocks
offset	the offset of file
size	read bytes

probe::nfsd.proc.remove

probe::nfsd.proc.remove — NFS server removing a file for client

Synopsis

```
nfsd.proc.remove
```

Values

proto	transfer protocol
version	nfs version
filelen	length of file name
filename	file name
fh	file handle (the first part is the length of the file handle)
client_ip	the ip address of client

probe::nfsd.proc.rename

probe::nfsd.proc.rename — NFS Server renaming a file for client

Synopsis

```
nfsd.proc.rename
```

Values

tlen	length of new file name
flen	length of old file name
tfh	file handler of new path
filename	old file name
fh	file handler of old path
client_ip	the ip address of client
tname	new file name

probe::nfsd.proc.write

probe::nfsd.proc.write — NFS server writing data to file for client

Synopsis

```
nfsd.proc.write
```

Values

count	read bytes
proto	transfer protocol
version	nfs version
size	read bytes
vec	struct kvec, includes buf address in kernel address and length of each buffer
client_ip	the ip address of client
fh	file handle (the first part is the length of the file handle)
vlen	read blocks
offset	the offset of file
stable	argp->stable

probe::nfsd.read

probe::nfsd.read — NFS server reading data from a file for client

Synopsis

```
nfsd.read
```

Values

count	read bytes
file	argument file, indicates if the file has been opened.
size	read bytes
vec	struct kvec, includes buf address in kernel address and length of each buffer
fh	file handle (the first part is the length of the file handle)
client_ip	the ip address of client
vlen	read blocks
offset	the offset of file

probe::nfsd.rename

probe::nfsd.rename — NFS server renaming a file for client

Synopsis

```
nfsd.rename
```

Values

tlen	length of new file name
flen	length of old file name
tfh	file handler of new path
filename	old file name
fh	file handler of old path
client_ip	the ip address of client
tname	new file name

probe::nfsd.unlink

probe::nfsd.unlink — NFS server removing a file or a directory for client

Synopsis

```
nfsd.unlink
```

Values

filelen	the length of file name
filename	file name
fh	file handle (the first part is the length of the file handle)
client_ip	the ip address of client
type	file type (file or dir)

probe::nfsd.write

probe::nfsd.write — NFS server writing data to a file for client

Synopsis

```
nfsd.write
```

Values

count	read bytes
file	argument file, indicates if the file has been opened.
size	read bytes
vec	struct kvec, includes buf address in kernel address and length of each buffer
fh	file handle (the first part is the length of the file handle)
client_ip	the ip address of client
vlen	read blocks
offset	the offset of file

Chapter 31. Speculation

This family of functions provides the ability to speculative record information and then at a later point in the SystemTap script either commit the information or discard it.

function::commit

function::commit — Write out all output related to a speculation buffer

Synopsis

```
commit(id:long)
```

Arguments

id of the buffer to store the information in

Description

Output all the output for *id* in the order that it was entered into the speculative buffer by `speculative`.

function::discard

function::discard — Discard all output related to a speculation buffer

Synopsis

```
discard(id:long)
```

Arguments

`id` of the buffer to store the information in

function::speculate

function::speculate — Store a string for possible output later

Synopsis

```
speculate(id:long,output:string)
```

Arguments

id buffer id to store the information in

output string to write out when commit occurs

Description

Add a string to the speculaive buffer for id.

function::speculation

function::speculation — Allocate a new id for speculative output

Synopsis

```
speculation:long()
```

Arguments

None

Description

The `speculation` function is called when a new speculation buffer is needed. It returns an id for the speculative output. There can be multiple threads being speculated on concurrently. This id is used by other speculation functions to keep the threads separate.